
Masters

Science

5-2018

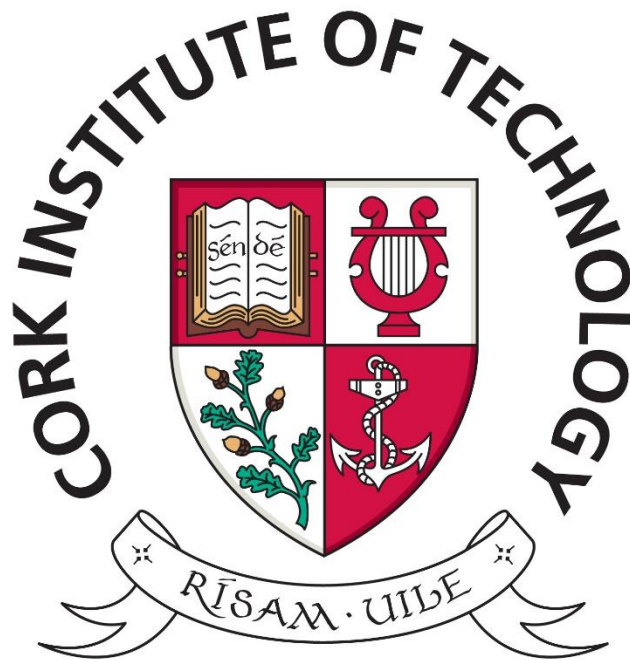
An Industry-Based Study on the Efficiency Benefits of Utilising Public Cloud Infrastructure and Infrastructure as Code Tools in the IT Environment Creation Process

Shane Callanan

Follow this and additional works at: <https://sword.cit.ie/scimas>



Part of the [OS and Networks Commons](#)



An industry-based study on the efficiency benefits of
utilising public cloud infrastructure and infrastructure as
code tools in the IT environment creation process

by

Shane Callanan

This thesis has been submitted in fulfilment for the degree of Masters of Science in

Cloud Computing

in the

Department of Computer Science

Cork Institute of Technology

May 2018

This thesis is dedicated to Orla Leahy

Declaration

I, Shane Callanan, declare that this thesis titled, ‘An industry-based study on the efficiency benefits of utilising public cloud infrastructure and infrastructure as code tools in the IT environment creation process’ and the work presented in it are my own.

I confirm that:

- *This work done wholly or mainly while in candidature for a postgraduate master’s degree at Cork Institute of Technology.*
- *Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.*
- *Where I have consulted the published work of others, this is always clearly attributed.*
- *Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.*
- *I have acknowledged all main sources of help.*
- *Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.*

Signed: _____

Date: _____

Abstract

The traditional approaches to IT infrastructure management typically involve the procuring, housing and running of company-owned and maintained physical servers. In recent years, alternative solutions to IT infrastructure management based on public cloud technologies have emerged. Infrastructure as a Service (IaaS), also known as public cloud infrastructure, allows for the on-demand provisioning of IT infrastructure resources via the Internet. Cloud Service Providers (CSP) such as Amazon Web Services (AWS) offer integration of their cloud-based infrastructure with Infrastructure as Code (IaC) tools. These tools allow for the entire configuration of public cloud based infrastructure to be scripted out and defined as code.

This thesis hypothesises that the correct utilization of IaaS and IaC can offer an organisation a more efficient type of IT infrastructure creation system than that of the organisations traditional method. To investigate this claim, an industry-based case study and survey questionnaire were carried out as part of this body of work. The case study involved the replacement of a manually managed IT infrastructure with that of the public cloud, the creation of which was automated via a framework consisting of IaC and related automation tools. The survey questionnaire was created with the intent to corroborate or refute the results obtained in the case study in the context of a wider audience of organisations.

The results show that the correct utilization of IaaS and IaC technologies can provide greater efficiency in the management of IT networks than the traditional approach.

Acknowledgements

Firstly, I would like to thank Eoin O'Regan and Dr. Donna O'Shea for their ceaseless assistance as my academic supervisors. I would like to acknowledge the time they have dedicated to review and evaluate my work over the course of the last two years. Secondly, I would like to thank Tomasz Serafinski for encouraging me to expand my horizons past the work place and into the world of academia. Thirdly, I would like to thank Declan Chambers, Mike Kissane, Tomasz Marciniszyn, Sean Barry, and all of my work colleagues for the advice and counsel they have given me during the course of this masters. Lastly, I would like to thank my employer, Aspen Grove Solutions, for the opportunity to conduct this research.

TABLE OF CONTENTS

Declaration.....	i
Abstract.....	ii
Acknowledgements.....	iii
List of Figures.....	vii
List of Tables.....	xi
Chapter 1. Introduction.....	1
1.1 Motivation.....	4
1.2 Research Aim.....	5
1.3 Research Objectives.....	5
1.4 Contribution.....	7
1.5 Methodology.....	8
1.6 Research Delimitation.....	11
Chapter 2. Background and Literature Review.....	12
2.1 Cloud Computing.....	12
2.1.1 Cloud Computing History.....	12
2.1.2 Modern Cloud Computing Definition.....	16
2.1.3 Infrastructure as a Service (IaaS) Benefits.....	19
2.1.4 Infrastructure as a Service Risks.....	22
2.2 Cloud Service Provider Comparison.....	27
2.2.1 Amazon EC2.....	27

2.2.2	Azure Virtual Machines.....	29
2.2.3	Google Compute Engine.....	31
2.2.4	Availability Comparison.....	33
2.2.5	Instance Price Comparison.....	34
2.2.6	Security.....	36
2.2.7	Results.....	37
2.3	Infrastructure as Code.....	37
2.3.1	Infrastructure as Code Benefits.....	43
2.3.2	Infrastructure as Code Risks.....	46
2.4	Infrastructure as Code and Infrastructure as a Service.....	47
2.5	State of the Art in Infrastructure as a Service Migration.....	48
2.6	Conclusion.....	53
Chapter 3. Design and Implementation.....		56
3.1	Case Study.....	56
3.1.1	Exploratory Phase.....	57
3.1.2	Project Planning Phase.....	59
3.1.3	Functional Requirements.....	63
3.1.4	Non-Functional Requirements.....	63
3.2	Framework Architecture.....	64
3.2.1	The Build Server.....	65
3.2.2	API Components.....	65
3.2.3	Framework Builds.....	67
3.2.4	Framework Prerequisites.....	71
3.2.5	Summary.....	72
3.3	Technologies Used.....	73
3.3.1	Cloud Service Provider - AWS.....	73

3.3.2	Build Server – TeamCity	74
3.3.3	Infrastructure-as-code - Terraform.....	74
3.3.4	Version Control System - Subversion	75
3.3.5	Configuration Management - Puppet.....	76
3.3.6	General Purpose Automation – PowerShell	77
3.3.7	Summary.....	78
3.4	Framework Use Case.....	78
3.4.1	User Interaction	79
3.4.2	Provisioning Build (PB)	81
3.4.3	Domain Build.....	82
3.4.4	Configuration Build	86
3.4.5	Deployment Build	87
Chapter 4.	Results.....	89
4.1	Creation/Recreation Experiment Context	89
4.1.1	Creation/Recreation Experiment Aims	89
4.1.2	Network Architecture	90
4.1.3	Environment Specification	96
4.1.4	Process Variables.....	99
4.1.5	Creation/Recreation Experiment Scope and Conduction	102
4.2	Creation/Recreation Experiment Results	103
4.2.1	Data Analysis.....	104
4.2.2	Sample Size	107
4.2.3	Comparison of Manual and Automated Datasets.....	108
4.2.4	Comparison of Creation and Recreation Data Datasets	115
4.2.5	Summary.....	117
4.3	Secondary Experiments	118

4.3.1	Instance Type Experiment Context	119
4.3.2	Instance Type Experiment Results	120
4.3.3	Storage Capacity Experiment Context	124
4.3.4	Storage Capacity Experiment Results	127
4.4	Survey	131
4.4.1	Sampling.....	131
4.4.2	Measurement Procedures	132
4.4.3	Data Collection	134
4.4.4	Respondent Category Comparison	134
4.4.5	Summary.....	137
4.5	Review of Results	137
4.5.1	Review of Creation/Recreation Experiment Results.....	137
4.5.2	Review of Secondary Experiment Results	146
4.5.3	Review of Survey Results.....	149
4.6	Limitations	152
Chapter 5.	Conclusions and Future Work	156
5.1	Discussion	156
5.2	Conclusions and Research Implications	157
5.4	Recommendations for Future Research	158
	Bibliography.....	161
	Appendices.....	171
	Appendix A. Interview with infrastructure member	171
	Appendix B. Interview with Release Management member	175
	Appendix C. Interview with Database Administration member	180

Appendix D. Survey Questionnaire 184

LIST OF FIGURES

Figure 1: Cloud Computing Overview.....	17
Figure 2: Cloud Computing Reported Benefits	22
Figure 3: Average Yearly Downtime.....	33
Figure 4: Average Yearly Outages.....	34
Figure 5: Average Yearly Availability.....	34
Figure 6: Instance Price Comparison	36
Figure 7: Reported benefits of IaC.....	45
Figure 8: AWS Shared Responsibility Model.....	60
Figure 9: Framework Architecture Overview	64
Figure 10: Overview of the Build Chain.....	68
Figure 11: Technologies Used	73
Figure 12: Sample Terraform Script.....	75
Figure 13: Sample Puppet Script	76
Figure 14: Terraform Script Transformation	81
Figure 15: Provisioning Build Sequence Diagram	82
Figure 16: Redirection of Traffic via DNS	84
Figure 17: Domain Build Sequence Diagram Part 1.....	85
Figure 18: Domain Build Sequence Diagram Part 2.....	85
Figure 19: Configuration Build Sequence Diagram	86
Figure 20: Sample PowerShell Function	88
Figure 21: On-site office to co-location datacentre diagram	92
Figure 22: On-site office to AWS diagram.....	95
Figure 23: Data Set Comparison: Automated vs. Manual	109
Figure 24: Task Comparison: Automated vs. Manual	110
Figure 25: Manual Timings: Breakdown of overall process.....	111

Figure 26: Automated Timings: Breakdown of overall process	112
Figure 27: Rate of Error in Automated Process	113
Figure 28: Median troubleshooting time per error occurrence	114
Figure 29: Data Set Comparison: Creation vs. Recreation	116
Figure 30: Task Comparison: Creation vs. Recreation	117
Figure 31: Instance Type Experiment Overall Comparison	122
Figure 32: Instance Type Experiment Task Comparison.....	123
Figure 33: Storage Capacity Experiment Overall Process Comparison	129
Figure 34: Storage Capacity Experiment Task Comparison.....	130
Figure 35: Respondent Category Timing Comparison	135
Figure 36: Staff involved in environment creation process	136

LIST OF TABLES

Table 1: Cloud Offering Downtime	33
Table 2: CSP Instance Comparison	35
Table 3: CSP Security Certificate Comparison.....	36
Table 4: Existing Migration Frameworks Phase Comparison	52
Table 5: Existing Migration Frameworks Features and Limitations Comparison	53
Table 6: Summary of technologies used in the framework.....	74
Table 7: System comparison of non-cloud and cloud environments	96
Table 8: Storage comparison of non-cloud and cloud environments.....	97
Table 9: Basic test environment software systems	98
Table 10: Organisation software on test environments.....	99
Table 11: Instance Types used	120
Table 12: Baseline AMI Storage	125
Table 13: Low Capacity AMI Storage	125
Table 14: High Capacity AMI Storage.....	126

Chapter 1. Introduction

The ubiquity of Internet access has reached an unprecedented rate, going from 400 million Internet users in 2000 to 3.7 billion users in 2017, an increase of 925% in just seventeen years (International Telecommunication Union, 2015) (Internet World Stats, 2017). This increase of Internet availability has vastly changed the world we live in, online services are quickly replacing their local equivalents in ways previously thought impossible. This is being done through the umbrella of technologies that cloud computing encapsulates.

At a very high level, the shift from local to online services can be simplified in the context of a delivery medium for movies: the old paradigm can be thought of as one where the customer travels to a DVD rental store, rents the physical DVD and brings the disk home for viewing on his local device. The cloud computing method uses an online video hosting service such as Netflix to handle payment for and deliver the movie to the client in an automated fashion through the Internet for portable viewing on a variety of devices. All the customer needs is a device that can access the service through the Internet. The simplicity and flexibility of this method compared to the traditional one are significant pull factors for the customer. Complimenting this are the resource savings from the provider's perspective, no longer needing to invest in buildings in a variety of locations and the fees that come with them, instead, they create and maintain an online service to handle certain aspects of their business.

The scope of cloud computing is broadening, accessing software online through a web browser instead of installing a local client is no longer thought of as an advanced technology as it has been globally adopted and used by millions of people for years: Facebook, Netflix, eBay and Gmail are all examples of this. The field of cloud computing has expanded so much that the paradigm known as X-as-a-Service (XaaS) has emerged. Duan et al. found that the XaaS term has become synonymous with cloud computing and is used describe the numerous services that can be delivered through the Internet, One such example of this is Infrastructure-as-a-Service (IaaS) (Duan, et al., 2015).

Industry-based surveys pertaining to trends in virtualization have shown that it is common place for organisations to utilize virtualisation technologies such as VMWare

in order to reduce their IT infrastructure costs and utilise their computing resources in a more efficient way (Davis & Lowe, 2015) (F5 Networks, 2009). Instead of buying, setting up and maintaining ten low specification physical machines to carry out business functions, one very powerful machine split into ten virtual machines can carry out the same functions for a fraction of the operating costs. This was a revolutionary system when it first came about and is still in use today, the cloud model of IaaS uses the same principles of virtualisation in a very different way. Instead of an organisation acquiring their own high specification machines for computing resources, a Cloud Service Provider (CSP) takes this, and associated responsibilities on in order to deliver virtualised networking components as a service to their clients (Mell, 2011). This method reduces costs considerably for both parties, instead of every business with IT infrastructure buying and maintaining their own physical servers, a CSP builds a very large data centre that houses hundreds, even thousands of physical machines and leases them out to cover the costs of the data centre construction, machine acquisition and physical server maintenance (Mell, 2011).

Existing research into the area of IaaS has shown that the reliability, scalability, interoperability and costing model associated with IaaS are motivating factors for organisations to migrate their existing non-cloud infrastructure to the public cloud, particularly Small to Medium sized Enterprises (SMEs) (Mateescu, et al., 2014) (Khajeh-Hosseini, et al., 2010) (RightScale, 2014) (RightScale, 2015). Both service provider and client have motivations for switching to this type of paradigm, and implementing cloud technologies is a high priority for many organisations. Evidence gathered in a survey of 196 technical professionals by the International Data Group Enterprise, a technology and research venture capital organisation, supports this claim. This survey pertains to each respondents organisation's expected investment in 2017 in various technologies, IaaS is the sixth highest area of technological investment with an overall spending increase of 27% compared to 2016 (IDG Enterprise, 2017). The above indicates that cloud computing is being actively pursued by businesses at an increasing rate.

However, there are challenges associated with the adoption of IaaS. Security issues pertaining to data hosting on third party infrastructure has been cited by Sadiku, et al. to be the greatest challenge when considering to adopt cloud computing platforms (Sadiku, et al., 2014). Various works outline the issue of organisations regulatory compliance in regards to the data they host being moved to a CSP's data centre which

may be located outside of an allowed geographical region for that data to exist in (Khan & Al-Yasiri, 2015) (Vu & Asal, 2012) (Manvi & Krishna Shyam, 2014). Hwang, et al., alongside Frey and Hasselbring state that the lack of automation in the process of migrating existing Virtual Machines (VMs) from an on-premise setting to a cloud based infrastructure is a major challenge for organisations (Hwang, et al., 2015) (Frey & Hasselbring, 2011). Mateescu et al. briefly describe the challenge of dealing with the complexity of migrating business processes from a non-cloud setting to the cloud platform (Mateescu, et al., 2014). As general challenges in adopting IaaS, Manvi and Krishna Shyam cite that, among others, the provisioning and management of large amount of VMs through standard system administration tasks requires a significant level of automation in order for IaaS to be a viable for organisations (Manvi & Krishna Shyam, 2014).

There have been several articles published in the field that present frameworks to tackle the aforementioned challenges pertaining to the migration of non-cloud infrastructure to the IaaS platform. Mateescu et al. propose The Migration Assessment Tool (MAT), which assesses existing systems designated for migration to cloud platforms and suggests service models alongside CSPs that could potentially host them (Mateescu, et al., 2014). Khan and Al-Yasiri present a step-by-step type framework for SMEs to decide what non-cloud systems they are running can be migrated to a particular cloud platform (Khan & Al-Yasiri, 2015). Bergmayr, et al. put forward the Advanced Software-based Service Provisioning and Migration of Legacy Software (ARTIST) framework, which provides a means for the reverse engineering and modernization of existing software systems in order to migrate it to the cloud platform, however, this framework focuses on the Software-as-a-Service (SaaS) platform as opposed to IaaS (Bergmayr, et al., 2013). Sabiri et al. outline a migration framework based on the Architecture Driven Modernization (ADM) paradigm, in which the legacy components of a system are analysed, reverse engineered and transformed in order to generate a model of the new system as it should act on a cloud platform (Sabiri, et al., 2015).

The above cited material deal primarily with the pre-migration phase of the migration process, assessing existing systems and suggesting potential cloud platforms that could host them. They do not address the complexity of migration challenge outlined by Mateescu et al. or the automation challenges described by Manvi and Krishna Shyam (Mateescu, et al., 2014) (Manvi & Krishna Shyam, 2014). They do not provide

any tangible automated process around which existing non-cloud infrastructure can be migrated to the IaaS platform in a standard, repeatable manner with as little human intervention as possible, this topic has been addressed by Hwang et al., who propose the Cloud Migration Orchestrator (CMO), a framework tested in laboratory scenarios which describes the entire migration procedure for non-cloud infrastructure to the IaaS platform from end-to-end and provides a semi-automated process for the live migration of existing VMs to IBM's Softlayer IaaS platform (Hwang, et al., 2015). However, to date, the CMO has only been tested under devised, laboratory conditions and has not been field tested, leaving a gap of knowledge in the area of industry-based studies on the migration of existing non-cloud infrastructure to the public cloud.

1.1 Motivation

The main motivation for this research is to address the lack of knowledge in the specific area of industry-based studies pertaining to the migration of non-cloud infrastructure to IaaS, which is outlined above. To further this point, the researcher performed a search through the digital library of the Institute of Electrical and Electronics Engineers (IEEE) which revealed that there is only one industry-based case study pertaining to the migration of in-house IT infrastructure to the IaaS platform available. This case study documents the organisational impact that a migration to AWS's IaaS platform would have and includes a cost comparison of the organisations existing in-house infrastructure versus the same infrastructure hosted in AWS (Khajeh-Hosseini, et al., 2010). But no migration was ever performed as part of the above case study due a number of reservations the organisation and its client had with the IaaS platform (Khajeh-Hosseini, et al., 2010). This means that the gap of knowledge in this area still exists as there has been no industry-based case study to date in academic literature that details the technical aspects of how a migration of existing enterprise level infrastructure to the public cloud can be performed. As there has been no industry-based study where existing infrastructure has been successfully migrated to the IaaS platform, there has also not been any study relating to the potential benefits or drawbacks of performing such a migration.

1.2 Research Aim

This research aims to develop and implement an automated framework for the migration of a SME's colocation-based IT infrastructure to AWS's IaaS platform. In doing so, it is aimed to gather metrics pertaining to the efficiency benefits of utilising such a framework in an industry-based setting when compared to the organisations traditional method of migration. It is also planned to prove the generalisability of these efficiency benefits in the context of the wider audience of SMEs. In order to gauge efficiency benefits in the context of this research, the following metrics are planned for inclusion:

1. Time

The time taken for the envisioned automated migration framework to perform actions in the context of creating an IT environment.

2. Effort

In the event where it is not possible to automate all actions involved in creating an IT environment via the framework then manual intervention will be required. This manual intervention will be measured by the effort metric.

3. Error occurrence

The recorded tendency of errors to be thrown on execution of the framework when performing actions to create an IT environment. This metric will tie in heavily with the effort metric as manual effort will be required to troubleshoot errors if they are to occur.

1.3 Research Objectives

The objectives of this research are as follows:

1. Analyse existing frameworks that allow for the migration of non-cloud infrastructure to the public cloud platform.

A large base of knowledge into the area of research should be obtained by the researcher before considering any industry-based development and implementation.

This base of knowledge will be created by carrying out a literature review of existing cloud migration frameworks. The analysis, documentation and categorisation of existing cloud migration frameworks should shed light on the existing technologies in the field along with the benefits and drawbacks of utilising each.

2. *Develop and implement an automated framework for the provisioning and management of public cloud infrastructure in a SME.*

Based upon the knowledge obtained in the preceding objective, the researcher should be placed in a SME that is planning on carrying out a migration of their non-cloud infrastructure to the public cloud. The researcher should carry out a case study within this SME. This case study should involve the observation, analysis and documentation of the organisations existing processes for the creation of their IT environments and how these processes are to be used in the context of migrating their existing infrastructure to the public cloud. The case study should also include the gathering of functional and non-functional requirements in regards to the design of the planned automated migration framework as it will be used in the SMEs infrastructure migration project. Based on the information outputted by the case study, the researcher should be in a position to begin developing and implementing the automated framework within the case study organisation.

3. *Utilise the automated framework to replace the SME's colocation-based IT environments with those on the public cloud for validation of the framework in an industry-based setting.*

Fulfilling this objective will require the researcher to participate heavily in the SMEs infrastructure migration project. By utilising the automated framework, it is planned that the researcher will be able to replace the SMEs existing infrastructure with that of the public cloud.

4. *Gather and analyse detailed statistics on the efficiency capabilities of the automated framework in the context of creating IT environments in the public cloud in comparison with the SMEs previous method.*

In carrying out the previous research objective, each execution of the framework will be scrutinized by the researcher, the data outputted by these executions will be analysed and categorised in order to form the metrics pertaining to the efficiency

benefits of utilising such a framework. These efficiency benefit metrics have been described in the Research Aim section.

5. Test the automated framework under a range of difference conditions.

It is planned that the framework is to be tested under various conditions in a series of secondary experiments in order to prove that variables in the execution of the framework can be controlled and modified in order to cause the results to differ. The ability to change and identify variables which ultimately effect the behaviour and data returned from the framework, it is expected that a better understanding of how to positively and/or negatively affect the framework can be found.

6. Survey the wider audience of SMEs in order to validate results from the automated framework in a generalizable fashion.

A survey questionnaire is planned to be created as part of this body of work. This survey will be distributed to active members of the software development world, preferably employees of companies with experience in cloud technologies. The aim of this survey and its distribution is to find a link between the use of IaaS and IaC tools and efficiency in the process of provisioning IT infrastructure. Thus proving the hypothesis of this thesis in the context of the wider audience of software development organisations.

1.4 Contribution

As contributions to the field, this thesis presents an automated framework consisting of automated IaC tools which has successfully rebuilt and configured an SME's co-location based VM environments on AWS's IaaS platform. This thesis also presents data pertaining to the efficiency benefits in the environment creation process the SME has gained by implementing the automated framework. To validate these efficiency benefits in the context of the audience of wider organisations, an industry-based survey was created and distributed, this survey draws correlations between efficiency in the environment creation process and each respondent's use of IaC tools and IaaS.

1.5 Methodology

This section outlines the research methodology followed in order to achieve the above research objectives. This section matches the methods chosen with the research objectives defined above.

1. Analyse existing frameworks that allow for the migration of non-cloud infrastructure to the public cloud platform.

The researcher performed a literature review of existing cloud migration frameworks via IEEE Xplore Digital Library and other academic sources, initially taking a broad view, inclusive of all service models of cloud computing, later focusing on just frameworks pertaining to the migration of non-cloud infrastructure to the IaaS platform. This was done in order to build a wide knowledge base pertaining to existing techniques, potential challenges, pitfalls and lack of innovation in current approaches in order to ultimately make the most informed decisions when developing and implementing such a system in an industry-based setting.

2. Develop and implement an automated framework for the provisioning and management of public cloud infrastructure in a SME.

Achieving this objective involved the placement of the researcher in a local SME, while there, the researcher carried out a case study. This case study was carried out in order to analyse and gain an in-depth understanding of how the SME's colocation-based IT environments were previously created and how it was planned to migrate these environments to AWS's IaaS platform.

Qualitative methods were adopted from the case study approach in order to engage with the members of the organisation the research was carried out in. This was done through semi-structured interviews, these interviews determined what specific metrics were to be measured and how to measure them, namely, each phase in the environment creation process, alongside the time and effort overheads incurred by carrying out each phase. Graham cites that properly performed semi-structured interviews are possibly the most important form of interviewing in case study research; the reasoning for this is that the semi-structured interview allows for more focus than that of an unstructured interview and allows for a less rigid and open communication experience than that of the structured interview (Graham, 2010). Semi-structured interviews were

deemed more appropriate than the unstructured interview type as the researcher already had a basic understanding of the subject area and had prepared a set of questions for each interview. Conversely, the structured interview was deemed too restrictive for this purpose as the researcher understood their own lack of advanced knowledge in the area and did not want the interviewees to rely on binary “yes” and “no” answers. Instead, it was aimed to guide the interviewees in the way the researcher had predefined while allowing them to be able to respond naturally to open ended questions in order to expose the researcher to parts of the environment creation process that the researcher may have overlooked or neglected to take into account properly.

This case study also included the gathering of client requirements as a means to design the migration project and define its scope and limitations. Lastly, this case study involved working with staff employed by the organisation in order to create the baseline networking components on AWS’s IaaS platform so that the framework would be capable of building environments that the organisation’s development and quality assurance departments would eventually be able to access and utilise the same way they accessed the existing colocation-based environments.

Once the above had been performed, the researcher had enough information to create the high-level architecture of the automated framework. Amaral et al. defines the “build” methodology as consisting of the design and creation of a novel software system to prove that it is possible (Amaral, 2011). Following the build method, the researcher developed a working prototype of the framework based on the high-level architecture, alongside all associated IaC scripts required for the framework to function.

3. Utilise the automated framework to replace the SME’s colocation-based IT environments with those on the public cloud for validation of the framework in an industry-based setting.

The researcher played an active role in the migration project within the SME in order to achieve this research objective. The researcher oversaw the execution, monitoring and troubleshooting of every run of the automated framework when it was used to rebuild the SME’s existing colocation-based environments, and create new environments on AWS’s IaaS platform. This method is defined by Amaral et al. as the experiment method, in which a system is evaluated under the scrutiny of the

researcher in order to answer a specific research question or achieve a specific research objective (Amaral, 2011). Throughout the course of this stage, data pertaining to the time and effort involved in the rebuilding of existing environments, and the creation of new environments in AWS were systematically retrieved and documented for analysis. The above metrics have been described in the Research Aim section.

4. Analyse and interpret detailed statistics on the efficiency capabilities of the automated framework in the context of creating IT environments in the public cloud in comparison with the SME's previous method.

The primary focus in achieving this research objective was the logical categorisation and statistical analysis of the quantitative data retrieved throughout the course of this project. The semi-structured interviews with staff members revealed the manual dataset of time and effort overheads involved in the organisation's previous environment creation process. This manual dataset was categorised and compared with the second dataset, the automated dataset, which was retrieved during the execution of the automated framework. Through this categorisation, statistical analysis and comparison, both datasets were interpreted in a meaningful way in order to display the efficiency benefits of implementing the automated framework on the IaaS platform when compared with the previous method utilised by the SME to create their environments.

5. Test the automated framework under as many different conditions as possible.

By performing a range of secondary experiments, it was possible to test the framework under all conditions possible given the context the framework was created in. By first identifying and then modifying controlled variables in the execution of the framework, it was possible to create separate categories of framework executions and compare their results with one another. In doing so, the researcher demonstrated how the modification of certain variables effect the data outputted by the framework and how optimisations may be applied to the framework to make it perform faster.

6. Survey the wider audience of SMEs in order to validate results from the automated framework in a generalizable fashion.

The survey questionnaire was created with the aim to achieve the above research objective by gaining an insight into the correlation between efficiency in the process

of provisioning IT infrastructure and the use of IaaS and IaC tools. This survey was distributed to a range of employees currently working in software development companies that are independent from the organisation the case study was carried out in. In doing so, two separate sets of data were gathered, one detailed set that is specific to the case study organisation, another less detailed set that aims to reflect a wider audience of software organisations. The full survey has been exported through a series of screenshots and can be found in Appendix D of this document.

1.6 Research Delimitation

This thesis is limited to the lower levels of IaaS, i.e. VM instances and the networks they reside on, other models of cloud technologies such as SaaS, PaaS, etc. will not be covered in great detail. This project is also limited to the set of technologies chosen by the case study organisation to best suit its needs. Therefore, technologies excluded from this set will not be covered in a comprehensive manner. Instead, justification of the usage of certain technologies over others will be discussed.

The company this work is being carried out on is a SME focusing on web-based application delivery. The company will be discussed as minimally as possible to avoid legal issues and to ensure the educational findings of this research will not be compromised in any way.

Chapter 2. Background and Literature Review

The topics of cloud computing and IaC technologies are relatively new, all being established in their modern forms the last decade or so. Overviews of both technologies comprise a significant section of this chapter, alongside the potential benefits and risks of adopting them. The topic of IaaS is the main focus in the cloud computing section as this is the specific technology this body of work deals with. A Cloud Service Provider (CSP) comparison is also detailed in this chapter, it contains a comprehensive evaluation of the three major CSPs in operation at the time of writing. This chapter progresses with a short section on the convergence and interconnectivity of IaaS and IaC technologies, detailing issues brought about through the scalability of IaaS which IaC tools can potentially offer a solution for. The chapter also contains a state of the art section on the current literature on frameworks for the migration of non-cloud infrastructure to the IaaS platform. The literature review concludes with an examination of the contents of the chapter alongside a section outlining the necessity of the case study and survey portions of this body of work.

2.1 Cloud Computing

2.1.1 Cloud Computing History

In the early 1950s, computers were large, expensive and monolithic; the execution of a single program meant solving a complex problem but it would take up the resources of the entire machine and usually needed an end-user with in-depth knowledge of the mainframe present to ensure it was running correctly. As time passed, interest in computers among the scientific community increased, as did investment into more advanced equipment. Because of this, execution times shortened to such a degree that the expenses associated with running an idle machine while the users interpret data and prepare the next job became more and more of a concern. This brought about the concept of queueing jobs for a machine to process using simple job monitoring systems, allowing all of the resources of the computer to be at use at any given time. This was an imperfect system though, the new model bolstered productivity but caused conflicts between users who constantly wanted more time on the mainframe

(Creasy, 1981) John McCarthy, a renowned professor at the Massachusetts Institute of Technology, realised this need and recalls formulating the idea for the progenitor to what we know as modern cloud computing as early as 1955 (McCarthy, 1992). McCarthy describes this concept as ‘time-sharing’ and defines it as ‘an operating system that permits each user of a computer to behave as though he were in sole control of a computer, not necessarily identical with the machine on which the operating system is running.’ (McCarthy, 1992)

Influenced by his lectures and writings on the subject, and frustrated by the expense and constraints of single-user mainframes, other MIT professors began taking an interest and implementing his ideas, among them was Fernando J. Corbató. As a result of this combined effort, the ‘Compatible Time-Sharing System’ (CTSS) was demonstrated on the university owned IBM 709 in November 1961. The CTSS supported four separate users operating Friden Flexowriter teleprinter terminals, each of which were directly connected to the input/output channel of the mainframe. Development continued on the CTSS until it was rolled out to users in MIT in the summer of 1963 (Walden, 2012). In this same year, the MIT science reporter John Fitch interviewed Corbató while he demonstrated the CTSS running on MIT’s IBM 7090, the episode was titled ‘A Solution to Computer Bottlenecks’, a fitting title, acting as a premonition to our contemporary application of computer time-sharing (Corbató, 1963). Needless to say, time-sharing systems boomed in this period, the cost efficiency alone was enough to surge the usage of computers in academic and business organisations into never before seen levels. The concept of sharing one mainframe for several users became so widespread and normalised that the single-user machine setup of the past quickly became redundant.

The next significant breakthrough was on 2nd August 1972 when IBM rolled out the world’s first computer capable of creating and running several virtual images of its own operating system (IBM, 2015). The Virtual Machine Facility/370 (VM/370) was the product of 13 years of research programs, all heavily influenced by MIT’s CTSS and, as such, it sought to overcome several of the constraints and issues present in CTSS. The VM/370 provided “multiple users with seemingly separate and independent IBM System 370 computing systems. These VMs are simulated using IBM System 370 hardware and have its same architecture” (Creasy, 1981). This revolutionary new system allowed organisations to scale their computing resources while eliminating several traditional overheads, namely the installation, housing and

running of new mainframes. As a new technology, the VM/370 was a huge success. In 1981, 9 years after its release, IBM estimates that 50,000 users were still running virtual machines provided by the VM/370 (Creasy, 1981). While the hardware and software of contemporary times have advanced far beyond the scope of the VM/370, many of the principles remain the same to this day, for example: The VM/370 allows for multiple users to access separate instances providing them with full operating system abstraction, meaning one VM cannot disrupt the operation of another VM (Creasy, 1981). It was the seminal system all of our modern VMs are based on and was an important catalyst to the evolution of cloud computing in the decades that followed.

The aforementioned technologies continued to develop over the next several years but failed to garner much of the attention they deserved. Over two decades after IBM's VM/370, several events occurred that led to an unanticipated and unprecedented spike in the demand for advancement in the distributed computing sector, these events led to what we know as the Internet Explosion. Defined by PC Magazine as:

“The period of tremendous growth of the Internet in the latter half of the 1990s. In the 1994-1996 time frame, it changed from a scientific and governmental research network to a commercial and consumer marketplace.” (PC Magazine, 2015)

This era saw the coining of the term ‘cloud’ when telecommunication companies began utilising Virtual Private Network (VPN) services with dynamic routing capabilities. VPNs allowed Internet providers to distribute bandwidth in a balanced and efficient manner across the network according to the needs of their clients, this allowed for on-demand scalability. Aptly so, VPNs became known as the ‘telecom cloud’. This type of network utilisation is an important milestone in the evolution of distributed computing as it is clearly a precursor to cloud computing (Kaufman, 2009).

In 1999, a massive change in the software industry occurred as Salesforce.com was created by Marc Benioff with the intention “to deliver business applications as a service over the Internet” (Benioff, 2009). Salesforce introduced the model where a client application could be hosted in a cost efficient manner and accessed on-demand through the Internet, it became the first provider of enterprise services over a distributed network of computers. Salesforce was extremely successful and influential

in the technology sector, its fundamentals and business model set the baseline that modern cloud providers still abide to.

In 2002, Amazon announced their first entrance into the cloud market by launching Amazon Web Services (AWS), although this service was aimed specifically at developers that were partnered with Amazon as it only allowed them to interface with features from Amazon.com through an external website (Amazon, 2002). The initial release of AWS may not have been an industry changing phenomenon but it did set the stage for Amazon's most popular services later in the decade.

Amazon Simple Storage Service (S3) was launched in March 2006 initially offering cloud-based storage infrastructure for any file up to 5 gigabytes (Arrington, 2006). The real innovation brought about by S3 was the pricing model, allowing for customers to pay nothing up front and pay only for the storage that they use, a model that quickly became a standard for cloud providers over the years. In June 2006, Google released Google Docs and Spreadsheets (eventually becoming just Google Docs), allowing users to collaboratively edit cloud-based documents in real-time (Google, 2006). In August 2006, Amazon Elastic Compute Cloud (EC2) was released, allowing users to access to a fully scalable virtual environment complete with a preloaded operating system of their choice. Similar to S3, the main advertised features was the pay-per-use pricing model and the potential for simple and cost effective scalability (Barr, 2006). Although it wasn't realized by many at the time, 2006 was a huge turning point in the IT industry as hardware and software were, for the first time, being leased out for personal use by trusted vendors and proven business giants through the medium of the Internet. Factor this with the ubiquity of high speed Internet availability and the rise in prominence of the smartphone in this era, it is unsurprising that the cloud changed from a buzzword that few outside the industry really understood to a household item in the years that followed.

Most modern cloud computing vendors offer clients much more than they did 10 years ago: full processing, storage and networking hardware as a service, all available through the pay-per-use pricing scheme initiated by Amazon S3. This standard has been set through years of consumer interest alongside massive investment and competition between industry giants such as Google, Amazon, Microsoft and IBM. 2014 was the first ever year to see public cloud IaaS workloads surpass that of on-premises infrastructure in terms of growth. Vice president of Gartner remarks that

“cloud IaaS is not merely a matter of hardware rental, but an entire data centre ecosystem as a service.” (Gartner, 2015)

2.1.2 Modern Cloud Computing Definition

Cloud is still an evolving paradigm, the most recent and widely accepted National Institute of Standards and Technology (NIST) definition describes the overall model as being comprised of five essential characteristics, three service models and four deployment models. This definition has been visually outlined in Figure 1 and explained in-depth in the section that follows.

There are five essential characteristics of cloud computing, they are listed and described below.

1. *Broad Network Access*

Cloud hosted resources can be accessed over a network from any location via a wide range of devices (smartphones, tablets, laptop, etc.).

2. *Rapid Elasticity*

Services from the cloud provider can be quickly (sometimes automatically) expanded to cater for fast scalability, be it with computing power or storage space of a single VM or expanding the capabilities of an entire network of servers. Abstracting the provisioning of resources to a seamless level from the end users perspective is the main aim of this characteristic.

3. *Measured Service*

Cloud providers automatically control and measure the services they provision via a metering system. Supplying provider and client with statistics of use, allowing full transparency on either side. This system is typically modelled after the pay-per-use paradigm, made famous with Amazon’s S3 service.



Figure 1: Cloud Computing Overview

(Cloud Security Alliance, 2011)

4. *On-Demand Self-Service*

Clients must be supplied with a means to manage their own cloud computing capabilities independently of the provider, there should be no need for discussion between customer and provider regarding immediate and regular up and downscaling of services.

5. *Resource Pooling*

The pooling of the cloud providers computing resources allow for them to support of a multi-tenant model of client use with little to no transparency of real physical machine usage to the end-user. This characteristic makes the providers resources seem limitless in the eyes of the client as virtual resources can be leased out as physical machines dynamically according to client demand.

The service models of cloud computing relate to the type of service being offered by the CSP. Three distinct service models exist, each with their defining characteristics and uses.

1. *Software-as-a-Service*

When a provider offers a cloud-based application to a client through a network, this software is typically accessible by various devices through a web browser. A key characteristic of this service model is that the client does not manage or control the underlying infrastructure the application is running on, regardless of the operating system, network configuration or storage type the application uses. This adds a huge layer of abstraction that is not visible to the end-user, as long as the client is accessing the software through a compatible medium, the actual user experience should be indistinguishable from that of a locally installed program. Examples of this type of service would be G-Mail, Facebook, Twitter, etc.

2. *Platform-as-a-Service*

When a provider offers a cloud-based environment for the client to deploy their own code to. The underlying infrastructure of the environment provided is not controlled or managed by the client, however, the client has full control over the deployed applications and configuration settings for the environment that is hosting the client code. Examples of this type of service would be Google App Engine and Microsoft Azure Web Sites.

3. *Infrastructure-as-a-Service*

When a provider offers computing infrastructure to a client that is accessible through a network; this infrastructure encompasses any kind of computing hardware, from processing power to storage to network components. Anything other than the pay-per-use pricing system for this model is uncommon. The client has full control over all software that runs on the leased device, which is typically a virtual machine running on a much more powerful physical machine, taking this into account, dynamic scaling of resources from the physical machine to a virtual instance can be achieved at an on-demand basis. Examples of this type of service would be Microsoft Azure, Amazon EC2, Amazon S3 and Google Compute Engine.

The deployment models of cloud computing represent the different types of cloud environments. All service models can be implemented on any deployment model of cloud computing, these deployment models are divided by ownership of the physical machinery offering cloud capabilities.

1. *Public cloud*

Resources are made available to the general public over a network. The CSP manages and maintains the data centres that are being leased, the physical machinery for this type of model resides on the premises of the CSP.

2. *Private cloud*

Resources are provisioned within the limits of a single organisation. This type of cloud can be owned and managed by the organisation they operate within, a third party or a combination of both. Private cloud data centres can be hosted on or off the premises of the organisation that uses it.

3. *Community cloud*

A multi-tenant model that provides resources to a specific collective of individuals or organisations with common computing concerns (geographical location, security requirements, company policy, etc.). This type of model may be owned and operated by a member of the community it caters to or a third party vendor, the machinery may exist on or off premises of the provider.

4. *Hybrid cloud*

Resources are delivered by a combination of two or more cloud deployment models. Each separate cloud remains its own entity in this model but can be linked to one another to allow data flow throughout the combined cloud network (Mell, 2011).

2.1.3 Infrastructure as a Service (IaaS) Benefits

Modern cloud computing technologies are being adopted on a global scale at a phenomenal rate. Global trends have shown that cloud computing is no longer a technology in its infancy and can potentially provide several benefits to the adopting organisation (RightScale, 2015).

Resource savings are widely cited as being a major positive effect of implementing cloud services, this is due to the lower maintenance cost of whatever service is being provided, for IaaS, the cost savings from not having to buy, house, power, cool and secure new servers are clear (Khajeh-Hosseini, et al., 2010)

However, other benefits related to resource savings may not be as apparent. The fact that when third party infrastructure is introduced, the technical staff in the adopting organisation that were in charge of physical server maintenance may find that their quality of work has improved dramatically as they can spend their time more productively. They are no longer required to spend laborious hours or days monitoring these servers and troubleshooting issues related to hardware, instead, these responsibilities belong to the CSP as it is part of the service they are offering (Mell, 2011, p. 3).

While it may not be directly related to savings, enhanced tracking of resource spending should certainly be mentioned in this section. It is simple to track the cost of buying the hardware in a traditional infrastructure setup, but with the cost of initial setup and maintenance, tracking the total cost of a single environment over a period of time is a difficult task and will undoubtedly require a large degree of estimation. AWS claims to solve this problem with cost allocation tags, these tags can be attached to every separate component in the AWS environment (e.g. EC2 instances, S3 buckets, etc.) in order to granulate the billing process in a transparent report that can be generated on demand to detail hourly, daily or monthly costs of each component (Amazon, 2015). Like many others, the company this project is being carried out in does not have an infrastructure cost calculation system as detailed as this, infrastructure costs of environments are manually derived and estimated from the overall cost of each physical server, travel expenses for engineers, hours taken to install the server along with the colocation provider fees. The ability to calculate with great precision the cost of a set of static environments that a specific department uses or the cost of temporary test environments that are used specifically for one project are examples of how this cost allocation tag can greatly benefit organisations.

Scalability is also a factor in the decision to migrate to the cloud, the ability to provision one or several new servers or increase the specifications of existing servers in a matter of minutes from a command line or web portal grants organisations unprecedented cost effective control over the scale of their IT infrastructure. Instagram is a prime example of, what started as a small business, grew at an unanticipated fast rate and handled their IT infrastructure with relative ease through Amazon's public cloud.

Instagram is a photo-sharing based social networking platform, it was launched on October 6th 2010 and within 24 hours 25,000 users had registered to use the application. Just over 2 months later, 1 million people were actively using Instagram, this skyrocketed to 80 million in 2012. By the end of 2013, just 3 years after the initial launch, Instagram had 150 million active users (WeRSM, 2013). Instagram's IT infrastructure is hosted in Amazon's public cloud, using EC2 instances and S3 storage to cater for its extraordinary upsurge in consumer demand. In 2012 Instagram discussed their IT infrastructure on their official engineering blog, they stated that self-hosting their infrastructure was not an option they had explored due to their content with cloud services. Their principles for catering for rapid growth are to:

- *“Keep it very simple*
- *Don't re-invent the wheel*
- *Go with proven and solid technologies when you can trust” (Instagram, 2012)*

This is a very clear message that sums up the standard AWS deliver, along with this was the fact that Instagram only employed 3 networking engineers at that time even though they were running hundreds of servers, a scenario that was completely unheard of until IaaS became available. It is argued that, without the simple scalability and versatility of the cloud infrastructure, Instagram would never have been able to cater for the frantic growth it experienced without numerous outages due to lack of capacity and overheads associated with expanding and managing a traditional data centre.

In both 2014 and 2015, RightScale, a cloud portfolio management industry leader, conducted two surveys of 1068 and 930 technical professionals, respectively, regarding their organisations experience of adopting cloud computing. The results show that 93% of respondents report that they are in the process of, or have already adopted cloud technologies. Respondents of these surveys reported a wide range of benefits from switching to the cloud computing model. Those most pertinent to this body of work have been extracted from both surveys and plotted in Figure 2 to show to the reported benefits and increase in reported benefits from the 2014 results to the 2015 results:

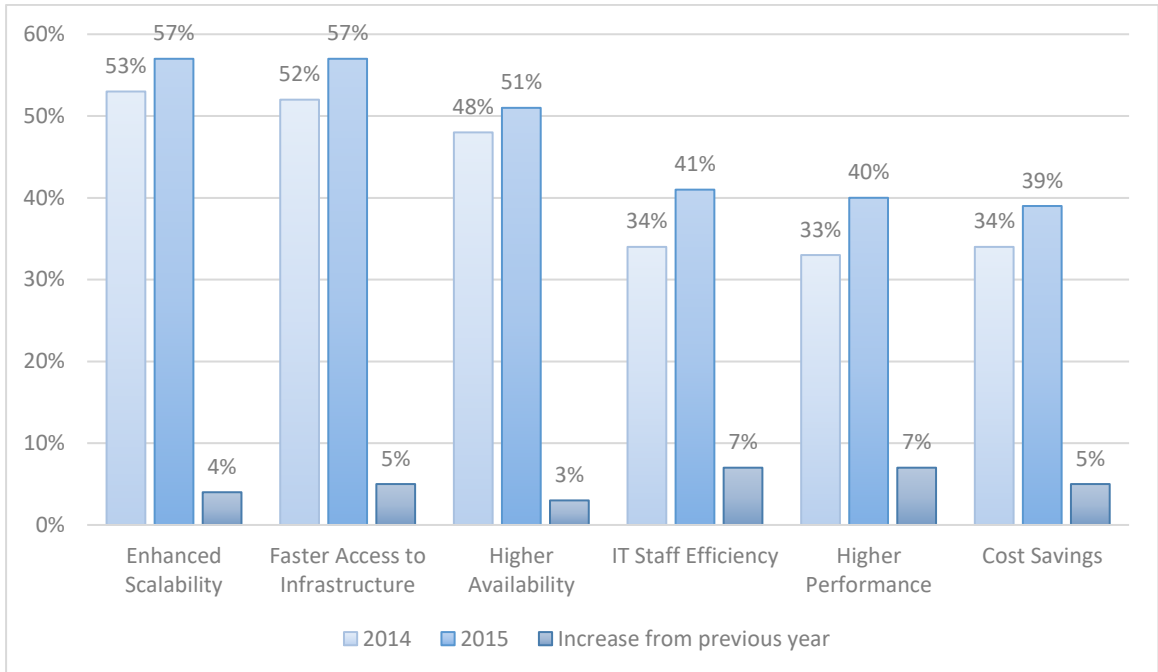


Figure 2: Cloud Computing Reported Benefits

On average, the amount of respondents reporting the above benefits increased by over 5% from the results of the 2014 survey to the 2015 survey. It should be noted that 29% of the SME respondents have reported to be out of the experimental phase and are using their cloud technologies heavily, seeking to optimize costs along with operations. However, of the large enterprise respondents, only 18% reported to be in this phase (RightScale, 2014) (RightScale, 2015). It is clear that implementing cloud technologies correctly can have huge positive impacts on a business, these benefits are increasing a dramatic rate as the cloud market is becoming larger, more mature and competitive.

2.1.4 Infrastructure as a Service Risks

While the benefits of adopting IaaS have been presented, the potential disadvantages have not yet been discussed; these include datacentre downtime along with company security and privacy concerns.

Datacentre downtime has been a widely discussed area since the advent of modern cloud computing and still is to this day. In a survey published in the International Journal of Cloud Computing and Services Science, researchers gathered a list of 78

major public cloud outages between 2007 and 2012; a large portion of which are accredited to power outages and hardware issues. Natural disasters, vehicle accidents causing damage to servers or dependant equipment, chain reactions caused by single pieces of power components failing are all documented sources of datacentre downtime, warning potential clients of the dangers of adopting cloud computing (Li, 2013). Downtime of leased servers can be catastrophic, each physical device may be split up into several virtual machines, meaning if one physical server goes down any virtual instances being hosted on that device are also inaccessible. This can lead to servers being out of sync with one another, developers working on the latest code release with no adequate environment to test against or prematurely interrupted client transactions if a production instance was effected. These are just a few symptoms of datacentre downtime, it is estimated that organisations can lose as much as €4,250 per minute of total downtime (Khan, 2014).

CSPs all have their own Service Level Agreements (SLAs), these SLAs describe the CSPs commitment to availability of their service, be it database hosting, virtual machine instances, storage, etc.. Typically, a CSP will propose a 99.95% availability, as is the case with Google, Azure and Amazon, meaning that the provider is guaranteeing that the virtual machines provisioned on their infrastructure will be unavailable to external connections for no longer than 4 hours and 23 minutes a year (Microsoft, 2015) (Amazon, 2013) (Google, 2015). This may not seem like much but if the previous estimate of total funds lost per minute of downtime is applied to 99.95% availability, the total amount lost per year is €1,117,750 (Khan, 2014). These outages are never pre-empted, therefore warnings cannot be given, a prime example of this is the AWS outage in 2011 where a bolt of lightning struck a generator in Amazon's Dublin based data centre which subsequently caused a fire that left the backup generators unusable, while some services were restored fully in a matter of hours, S3 storage was not at full working capacity for 2 days (Miller, 2011). This incident is of course an infamous one that has never recurred and has almost no chance of doing so, in reality the majority of outages do not occur in one single block, rather dozens of small incidents that may last very short periods of time for a variety of reasons. This 99.95% availability still means just over 5 minutes of downtime a week is allowed, even this could cause huge problems for any organisation that hosts client-facing sites capable of financial transactions. If, at any time, there are a number of users purchasing items or sending money, their transactions could be completely lost

due a 5 minute outage on the CSP's side. Due to the SLA, this scenario is legally allowed to occur every week of the year, the organisation that hosts their sites on these servers will have absolutely no grounds for compensation. For any organisation, careful consideration must be taken when examining what services should be migrated to cloud, as no vendor can offer 100% availability with complete certainty.

Security in cloud computing has been an area of controversy since its services became widely available for businesses, and rightly so: in 2014 it was reported that the UK based Institute and Faculty of Actuaries claims to have accessed 47 million NHS patient's medical records on the cloud for the purpose of determining insurance premiums (Khan, 2014). In a study by Khan and Al-Yasiri, 95% of SME interviewees stated that data security and privacy were a concern for them (Khan & Al-Yasiri, 2015). Gibson et al. outlines the high risk of security issues associated with the multi-tenancy involved in IaaS; CSPs host data for several other companies or organisations, sometimes this data is hosted in the same data centre or even on the same physical machine, which can lead to a higher risk of data leaks than self-hosting (Gibson, et al., 2012). Dawoud et al. purports that these types of activities are possible in multi-tenant environments on either the CSP side by exploiting the elevated privileges of the hypervisor in order to access the memory of a VM it is managing or on the client side by the use of malicious programs on interlinked VMs to spy on the data being passed to the hypervisor from other VMs it is managing (Dawoud, et al., 2010). This is another vital risk to consider when adopting the cloud's services, a company that hosts any sensitive information on the cloud is potentially leaving itself open to a litany of cyber-attacks that its previous infrastructure may have prevented. However, there are two sides to this debate, while in some cases traditional IT infrastructure will allow for more control over sensitive data and a perceived risk reduction of data loss, but, as cloud technologies are becoming more and more adopted and invested in, CSPs are increasing security to quell the concerns of potential and existing clients. Proper utilisation of CSP security groups has been outlined by Jin et al. as a highly effective method of restricting access to cloud-based VMs, they act as a network firewall on the CSP's side and are highly configurable to suit the client's needs (Jin, et al., 2016). Vaquero, et al. cites Amazon's Virtual Private Cloud (VPC) virtual networking component as an effective means of dealing with the security issues involved in the multi-tenant architecture of IaaS (Vaquero, et al., 2011).

Along with the above, there are a huge variety of standards and certificates available for cloud compliance, covering every one of these is beyond the scope of this thesis, instead a list of standards have been derived from the Cloud Standards Customer Council's whitepaper titled "Cloud Security Standards: What to Expect and What to Negotiate", this paper recommends compliance requirements an organisation should seek out when adopting cloud technologies (Cloud Standards Customer Council, 2016). This project pertains to IaaS and the business this project is being carried out for is a real estate and asset management organisation with a client base located primarily in the United States, these considerations were taken into account to further concentrate the scope of security requirements and compile a concise list of focused security standards that are appropriate to the business and the project being carried out, these standards will be described and compared on a CSP level later in this paper.

1. ISO 27018

First published in 2014, ISO 27018 is one of the most recent and possibly the most pertinent standard to public cloud computing security from the International Standards Organisation. It is related to the security requirements of public CSPs who store and transmit personally identifiable information of their clients. Organisations awarded this certificate have proved to uphold an internationally recognised set of security frameworks in relation to security around information that might identify an individual and their personal details, an essential requirement to this project considering the large amount of client related information the business stores (ISO/IEC, 2014).

2. PCI-DSS (Payment Card Industry – Data Security Standard)

A standard largely revolving around the secure handling of sensitive cardholder information. It is comprised of several requirements for organisations that electronically store, process or transmit any details of payment cards (PCI Security Standards Council, 2015).

3. ISO 27001

An internationally recognised standard which defines a framework of security requirements for information security management systems, its main aims are to protect the information of personnel and to systematically evaluate and manage information security risks. This standard is highly recommended to organisations

that deal with information regarding the financial sector (Certification Europe, 2015).

4. *SSAE 16 (Statement on Standards for Attestation Engagements)*

Developed by the America Institute of Certified Public Accountants, this auditing standard relates to the control objectives and activities of information in the target organisation. This standard pertains highly to organisations that host client data and offer IaaS (Frost, 2015).

5. *FIPS 140-2 (Federal Information Processing Standard)*

A United States federal standard created by NIST, it specifies the security requirements of storing, maintaining and implementing cryptographic modules to protect sensitive information. This standard is particularly pertinent to United States government organisations (National Institute of Standards and Technology, 2001).

6. *FedRAMP (Federal Risk and Authorisation Management Program)*

Administration for the United States government have collaborated with the National Institute of Standards and Technology (NIST), the Department of Defence (DOD), the Department of Homeland Security (DHS) and several other government and non-government organisations to develop FedRAMP. At a high level, the FedRAMP assessment process includes applying a set of state-of-the-art, transparent and reusable security standards to individual cloud technology offerings. Assessments are carried out by impartial third party assessment organisations on a CSPs demonstration environment, the result of which will be a full audit of a CSPs offering in order to determine if the system offered is secure enough for US federal government use. Effectively, FedRAMP offer CSPs the mark of approval for one of the highest standards of security possible. One of the main aims of FedRAMP is to bolster the current state of security in the cloud sector as a whole, while encouraging the global community to adopt cloud technologies (VanRoekel, 2011). At the time of writing, there are only 15 IaaS offerings compliant with FedRAMP (FedRAMP, 2015).

While it is not directly related to IaaS security concerns, the European Union's General Data Protection Regulation (GDPR) merits mention in this section; the GDPR was passed in April 2016 and is due to be enforced in May 2018, shortly after this

thesis has been finalised (Nadeau, 2018). In short, the GDPR sets out a rigorous regulatory framework pertaining to the processing, storing and deletion of data between consumers and organisations that handle their data; the GDPR encompasses the personally identifiable information of all European Union citizens (Blackmer, 2016). Failure to abide by the GDPR can lead to fines from the European Commission of up to 4% of the non-compliant organisation's income or €20 million, whichever is higher (EUGDPR.org, 2018). It is expected to have a huge impact on the field of cloud computing, especially in the area of IaaS where organisations utilising IaaS may not be aware if their client's data is being stored and processed in accordance with the GDPR framework (Webber, 2016). The ramifications of GDPR can only be speculated at the time of writing, as it has not yet come into effect, therefore, it will not be discussed any further in this thesis.

2.2 Cloud Service Provider Comparison

There are numerous IaaS providers active in the market today, for the implementation side of this body of work, the following three were examined as they are considered to be the prominent market leaders (Knorr, 2016) (Maguire, 2015):

1. *Amazon Web Services*
2. *Microsoft Azure*
3. *Google Cloud Engine*

What follows is a brief overview of each offering under similar headings along with an availability comparison of each CSPs virtual machine and storage offering.

2.2.1 Amazon EC2

Elastic Cloud Compute (EC2) is Amazon's IaaS offering, it allows users to create, modify and destroy scalable computing instances from a wide variety of operating systems through Amazon Machine Images (AMIs) in a matter of minutes via a web service interface or through an API (Amazon, 2015).

EC2 billing is specific to the instance type being used, as a whole, billing is based on a pay-per-use model. Users are billed on a monthly basis and are under no obligation to fulfil a minimum monthly amount of instance hours. Amazon offer a free tier account to new AWS customers for a year, this includes:

- *“750 hours of EC2 running Microsoft Windows Server, Linux, RHEL, or SLES t2.micro instance usage*
- *750 hours of Elastic Load Balancing plus 15 GB data processing*
- *30 GB of Amazon Elastic Block Storage in any combination of General Purpose (SSD) or Magnetic, plus 2 million I/Os (with Magnetic) and 1 GB of snapshot storage*
- *15 GB of bandwidth out aggregated across all AWS services*
- *1 GB of Regional Data Transfer” (Amazon, 2015)*

Clients are charged by hours their instances are on, partial instance hours are billed as full hours. Organisations will benefit greatly here by automating the stopping and starting of instances outside and inside of business hours respectively (Amazon, 2015). Amazon have a total of 38 predefined machine specifications that users can create instances from, ranging from 1GB RAM with 1 core to 244GB RAM with 36 cores (Amazon, 2015).

1. On-Demand Instances

Scalable computing resources are available on-demand and paid for by hour of use, recommended for systems with unpredictable workloads that may need additional capacity and need to be available within user specified times.

2. Reserved Instances

Allows users to purchase instances for a given length of time in one up-front payment, recommended for systems with a constant, predictable workload that require a set amount of capacity for a predefined set of time. There is a limit of 20 reserved instances per availability zone per user.

3. Spot Instances

Similar to on-demand instances in that computing resources are paid for hourly, spot instances are provisioned to the highest hourly bidder, prices obviously rise and fall given peak hours. It is a supply and demand type system where the user never pays more than they have agreed to pay but may lose their instances if they are outbid. Recommended for non-critical applications that can easily recover when interrupted (Amazon, 2015).

AMIs are pre-packaged environments, at their most basic, AMIs contain an out-of-box operating system. Clients have the option to choose pre-configured public AMI or create one of their own based on an existing operating system. Amazon boast a 1-Click launch function that swiftly deploys a preconfigured AMI with a single click. Advanced users will benefit from using custom AMIs as the image can contain applications, libraries, data and configuration settings. Take for example, if a user has specific custom applications that are configured in a certain way that is not default to the operating system's out of box settings then they can define all of this in an AMI their own, configured the way they want it and use this image to spawn as many as they need without additional application installs or configuration, AMIs can be set to private, so only the client that created it can view and use it, or public, so everyone using EC2 can use it. Elastic Block Storage (EBS) is Amazon's data persistence feature for EC2 instances, allowing clients to switch off their instances when they are not being used and turn them back on when they are being used (Amazon, 2015).

Amazon's Virtual Private Cloud (VPC) lets users create their own virtual networking environment that EC2 instances reside in. VPCs operate using an IP range specified by the user, combined with Security Groups and network ACLs, full control over instance and Internet communication is handed to the user. Existing IT infrastructure can be joined to VPCs via encrypted VPNs that come with AWS (Amazon, 2015).

2.2.2 Azure Virtual Machines

Azure virtual machines are Microsoft's IaaS platform, allowing customers to provision computing resources from multiple operating systems "nearly instantaneously" through a web portal with Azure Resource Manager (ARM) or Azure's own API (Microsoft, 2015). Users are billed monthly per minute of VM use. There are no upfront costs or termination fees. When the instance is shut down and

the cores the VM was using are no longer allocated to it, then billing is suspended. A month's free trial with \$200 prepaid into the users account is offered by Azure, all services are available in this trial. Azure do offer a 12 month prepay option that entitles users to a 5% discount on all Azure services, this discount is relative to the pay-per-use model. This option is only available with a minimum purchase of \$6000, Microsoft have a strict no refund policy for this service and users subscriptions are set to renew automatically with the same amount as purchased initially. All funds remaining in the users account at the end of the 12 month period will be absorbed by Microsoft (Microsoft, 2015) Azure has a total of 52 different predefined instance types for users to provision, ranging from 0.75GB RAM with 1 core to 448GB RAM with 32 cores (Microsoft, 2015).

ARM allows users to define and group their infrastructure as logically related resources from the web portal or API. This system of management works from an application level to a higher level infrastructure level and all components in between, examples of resources covered by ARM are virtual machines, data storage, virtual networks and 3rd party services. Customers can save these configurations as ARM templates in order to redeploy entire environments without additional setup, these templates encapsulate a greater level of infrastructure compared to AWS AMIs, which are specifically designed to define configuration on a virtual machine level. ARM templates are JSON files that let customers define their deployment and configuration of their systems in a declarative way, dependencies are dealt with automatically through ARM analysis prior to any execution of defined resources. This caters for repeatability and scalability through the simple updating of single or multiple components in any given network setup (Microsoft, 2015). Azure offer five types of storage systems.

1. Blob

Binary Large Object (BLOB) storage is recommended for large files that need to be stored on a long term basis, each BLOB can be up to 50GB and are replicated three times in the data centre they are stored in for redundancy and high availability purposes. This type of storage is external to instances, accessible from anywhere on the Internet, and persistent.

2. Table

Similar, but not full SQL tables, Azure offer storage of very large tables spanning millions of rows and columns. Like Blob storage, tables are replicated three times in the data centre they reside in. This type of storage is external to instances and persistent.

3. *Local disks*

Each Azure instance has at least one predefined local disk, it can be a hard disk drive or solid state drive. This type of storage is internal to instances and is not persistent.

4. *XDrives*

XDrives are virtual disk drives that reside outside of any instance, they can be mounted on any Azure instance and behave just like local disks. They are based on the Blob storage system, so they are persistent and not dependant on the instance they are mounted on (Microsoft, 2015).

5. *Queue*

Azure offers a persistent queue storage system for messages that can be accessed from any location, instances can connect to the queue to send messages to machines. The messages are limited to 64KB in size but the queue itself can store up to 100TB of messages (Cremers, 2012).

Azure Virtual Network provides a means for building virtual networking topologies capable of integrating with existing infrastructure through VPNs or Azure's own alternative: ExpressRoute. Users can define their own set of private IPs, subnets and traffic flows and run WAN optimizers, load balancers, and application firewalls in the Virtual Network. Azure fully supports hybrid applications that simultaneously work from both an external IT network and the Azure Virtual Network (Microsoft, 2015).

2.2.3 Google Compute Engine

Google's IaaS offering is called Google Compute Engine, it supports Ubuntu, Debian, Red Hat, SUSE, and Windows operating systems on Google's highly available

infrastructure. Like AWS and Azure, this service is managed through a web portal and/or API (Google, 2015).

Google Compute Engine's pricing model follows the standard pay-per-use model, users are charged for leased computing capacity on a 10 minute basis and 1 minute increments thereafter on a monthly basis. Instances running for more than 25% of a month qualify for a sustained use discount, this discount is automatically applied when instances run over a certain amount of time in any one billing period, it is limited to a net discount of 30% for instances that run continuously for an entire month (Google, 2015). Compute Engine has a library of 18 different machine specifications to create instances of, ranging from 0.6GB of RAM with 1 core to 208GB of RAM with 32 cores (Google, 2015).

Compute Engine offers two different types of storage:

1. *Persistent disks*

Users have the option of specifying a hard disk drive or a solid state drive for persistent storage. These disks are independent of instances and can be attached to any instance type. They are replicated in the region they reside for data redundancy for high availability and support snapshotting in order to attach a disk preloaded with data to any instance. Persistent disks can be used as boot devices for instances.

2. *Local SSDs*

These disks are attached to the instance when it is created, they are fully dependant on the instance they are attached to and will not persist when the instance is powered down or terminated. These disks are not replicated, do not support snapshots and cannot be used as boot devices for instances (Google, 2015).

Compute Engine provides its own networking hierarchy, allowing for multiple networks with multiple instances in each. Users define a gateway IP for each network and a network range for IPs of instances inside of that network. By default, all instances inside of a network can communicate with each other but all external incoming traffic is blocked by a configurable firewall. Routes allow for the handling of outgoing network traffic from an instance while VPNs can be set up to allow for existing external infrastructure to communicate with Compute Engine networks (Google, 2015).

Product	Outages	Total Downtime (minutes)	Availability (%)
Amazon S3	37	6.78	99.9987
Amazon EC2	14	6.92	99.9988
Google Cloud Storage	7	0.58	99.9998
Google Compute Engine	103	132	99.9749
Azure Object Storage	103	44.38	99.9916
Azure Virtual Machines	66	153.6	99.9738

Table 1: Cloud Offering Downtime

(CloudHarmony, 2015)

2.2.4 Availability Comparison

As previously mentioned, Google, Azure and Amazon all offer 99.95% availability for their IaaS offerings (Microsoft, 2015) (Amazon, 2013) (Google, 2015). However, analysis of the outages from October 25th 2014 to October 25th 2015 from Google, Amazon and Azure IaaS and storage offerings show that not one CSP had downtime extended past their SLA, as can be seen in Table 1. The data from Table 1 was used to make the following three graphs, which show a visual comparison of average yearly downtime, average yearly outages and average yearly availability of the aggregated compute and storage services offered by the three IaaS providers

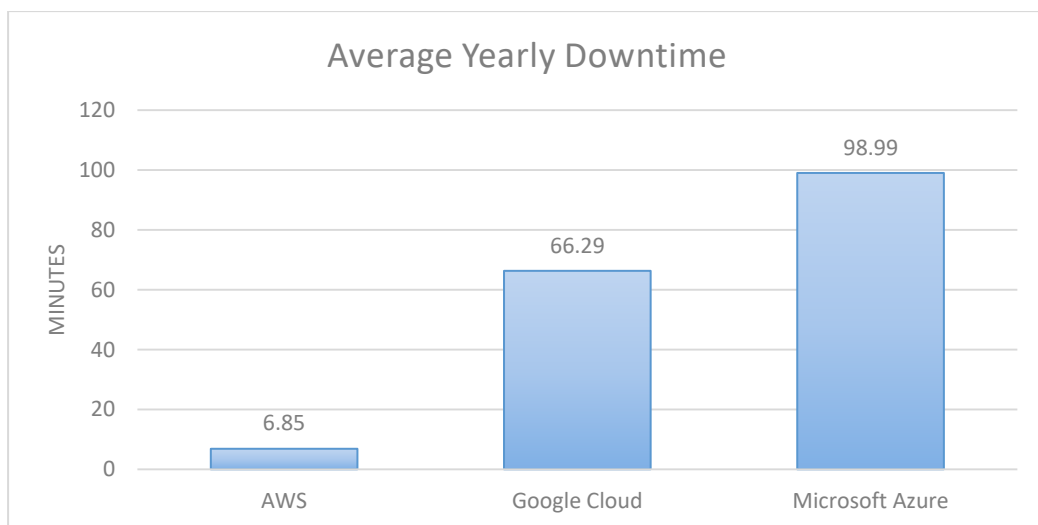


Figure 3: Average Yearly Downtime

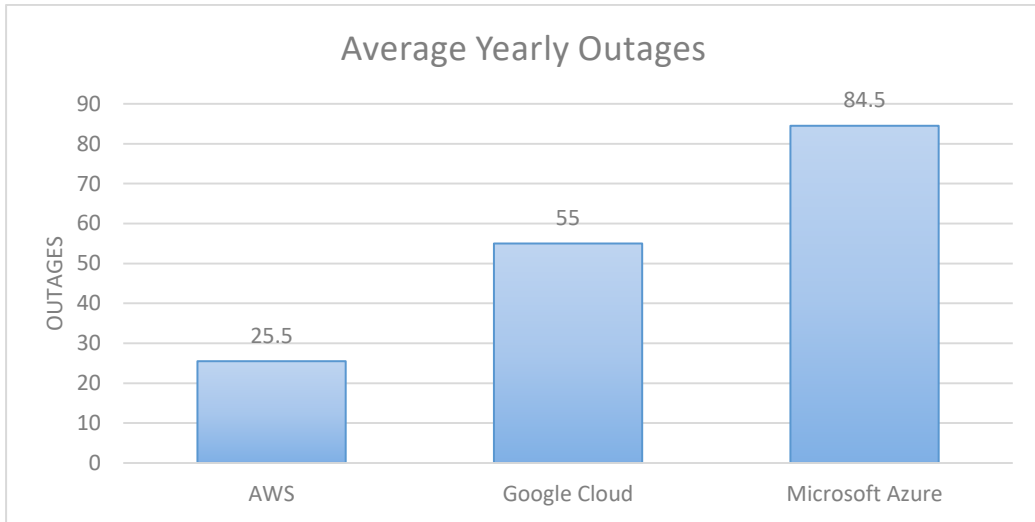


Figure 4: Average Yearly Outages

2.2.5 Instance Price Comparison

What follows is a price comparison of an instance from each CSP per hour, the instance types chosen for comparison are based on the environments recreated as part of this body work. These machines are have the following basic specifications: 8.00GB RAM, 2 cores, these metrics were taken into account when selecting the instances to compare.

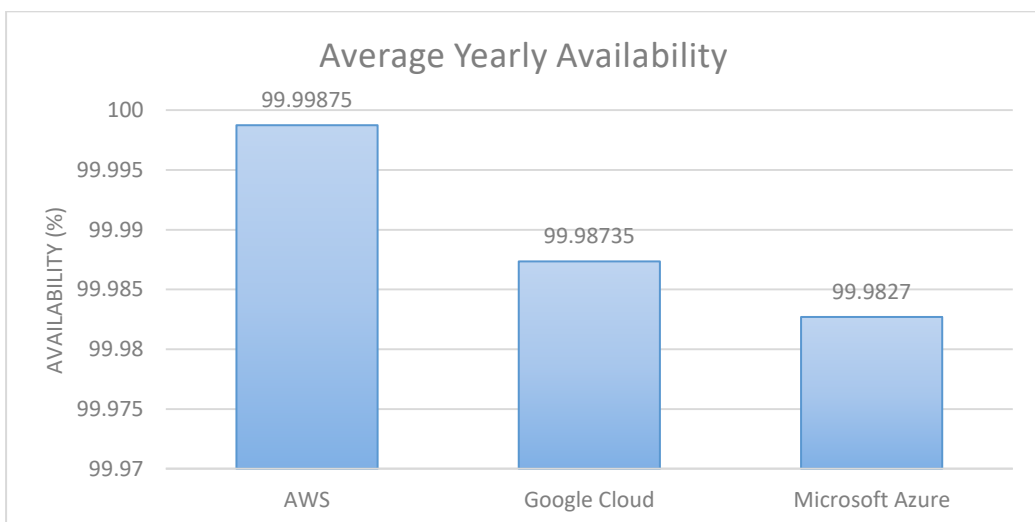


Figure 5: Average Yearly Availability

CSP	AWS	Google Cloud	Microsoft Azure
Instance Name	t2.large	n1-standard-2	A5
Cores	2	2	2
RAM (GB)	8	7.5	14
Storage (GB)	Elastic Block Storage	None	135
Storage Type	HDD	N/A	HDD
Cost \$/hr	0.134	0.19	0.32

Table 2: CSP Instance Comparison

(Amazon, 2015) (Azure, 2015) (Google, 2015)

AWS's t2.large instance type matches the machines to be replaced more adequately than Google and Azure, both of which did not offer identical compute power in their instances, therefore the compute power of the instances chosen for comparison had to be rounded down for Google and up for Azure.

Google's n1-standard-2 instance type had insufficient RAM and cost more than AWS's t2.large instance type, whereas Azure's A5 instance type had an excessive amount of RAM and cost far more than Google's n1-standard-2 and AWS's t2.large.

It should be noted here that the compared Azure instance includes 135GB storage space, but, as this disk is not persistent, it is of no interest to company this project is being carried out for. Figure 6 summarises the derived data in a simple chart form:



Figure 6: Instance Price Comparison

2.2.6 Security

At the time of writing, each of the CSPs considered for this project have achieved all of the previously discussed security requirements bar Google Cloud Compute which is only missing the FIPS 140-2, this data is outlined in Table 3.

Security Certificates	AWS	Google Cloud Compute	Microsoft Azure
ISO 27018	Yes	Yes	Yes
PCI-DSS	Yes	Yes	Yes
ISO 27001	Yes	Yes	Yes
SSAE 16	Yes	Yes	Yes
FIPS 140-2	Yes	No	Yes
FedRAMP	Yes	Yes	Yes

Table 3: CSP Security Certificate Comparison

2.2.7 Results

While all appear similar in regards to their interfaces (web portal alongside REST API), networking capabilities and security standards, AWS stood out above the rest in fulfilling the specific requirements for the organisation. The T2.Large EC2 instance type matches the organisation's needs more closely than Google and Azure, taking this specific instance type into account, the above charts shows that AWS offer cheaper instance runtime while maintaining the lowest average downtime, lowest average outages and highest average availability for their virtual machine and storage services, these were all important factors that were considered when determining the CSP for this project. Along with these, Amazon's maturity in the sector and their variety of instance types led to the conclusion that AWS should be chosen for the implementation side of this project.

2.3 Infrastructure as Code

Infrastructure as Code is a relatively new paradigm, allowing for all aspects of IT infrastructure and their configurations to be scripted out as code; the design of entire networks can be defined and source controlled as though it is application or database code which allows for granular change management, uniformity of servers and the potential for rapid scalability (Nelson-Smith, 2013).

It is not completely clear when the term 'Infrastructure as Code' was coined, sources indicate that the term came about after the release of AWS EC2 (Nelson-Smith, 2013). Configuration management code is discussed as a precursor to IaC in the following sections, though not explicitly stated by many sources, the seemingly interchangeable terms configuration management code and IaC are not one in the same.

Modern interpretations refer to both configuration management and IaC as one in the same; the definition of the term by Kief Morris, the author of book titled Infrastructure as Code is as follows:

"Infrastructure as Code is an approach to infrastructure automation based on practices from software development. It emphasizes consistent, repeatable

routines for provisioning and changing systems and their configuration.” (Morris, 2016)

However, it is the author’s opinion that the divide between configuration management code and IaC is clear: configuration management tools allow for components residing above the operating system layer (i.e. directories, application configuration, etc.) to be scripted out and executed as code while IaC tools allow for the lower level components such as virtual machines and the virtual networks they reside on to be scripted out and executed as code.

Information on the history of Infrastructure as Code is scarce, but, the first documented use of this type of technology dates as far back as 1993 when Mark Burgess, a post-doctoral researcher at Oslo University, created a small, open source command line tool that allowed him to automate the configuration management of workstations in the university in order to eliminate tedious tasks associated with manually setting these machines up, allowing him to get his work done in a more efficient manner. Burgess dubbed this tool “The Configuration Engine”, or CFEngine as it is more widely known (CFEngine, 2014).

Burgess describes CFEngine as:

“A very high level description language for UNIX machine-park configuration, intended to assist the administration of a local area network by defining the setup of all machines centrally from one file.” (Burgess, 1993)

This early form of automated configuration management through code allows for a single file to specify the configuration of several machines, once the CFEngine program is compiled on each machine, the configuration file is then passed to each machine in the network and each one executes the same file (Burgess, 1993). It is clear from reading the above that Burgess pioneered the idea of Infrastructure as Code; the same basic principles of self-describing code, portability and even similar execution methods can be seen in modern day configuration management and IaC tools.

Throughout the 90s, Unix systems evolved and became more complex, the initial release of CFEngine began to show its flaws and limitations when used against different Unix platforms, this, along with the fact that there was a lack of research and development in the configuration management area led to Burgess continuing his

work. In 1998, Burgess presented a paper called Computer Immunology at the Twelfth Systems Administration Conference, the landmark piece of work envisioned a type of self-healing computer system comparable to the human immune system (Burgess, 1998, p. 283).

In this paper, Burgess scornfully notes the massive amount of time system administrators need to spend diagnosing and fixing problems related to management of a network of computers and discusses the possibility of autonomous system maintenance, whereby faults in a system can be detected and fixed automatically without the need for human intervention. This is similar to way that most human immune systems can easily dispatch routine problems such as headaches, fatigue and small injuries without the need to be hospitalised for dedicated medical care by a health professional. Burgess furthers this analogy by writing that “it is as though all of our machines are permanently in hospital” (Burgess, 1998, p. 283). The system Burgess proposed to fix this prevalent problem can be summarised as a network of machines in which a “healthy” computer state is defined and automatically pushed to every machine on that network, this state data will then be enforced upon each machine to ensure every node in the network is in a healthy, uniform state (Burgess, 1998, pp. 283-288). As a direct result of the Computer Immunology paper, a major research effort in Oslo University took place with Burgess at the forefront, leading to the release of CFEngine 2 in March 2002, this new version featured machine learning and anomaly detection based on the ideals introduced in the Computer Immunology paper (CFEngine, 2014) (Burgess, 2002). Over 20 years later, Burgess’s ideals are clearly incorporated as the core principles that modern day automated configuration management tools adhere to. Tools created years after the initial CFEngine, like Puppet and Chef, are based on the idea that a computer’s state can be defined through code and pushed from a central location across multiple machines in an automated fashion in order to create a uniform network of computers (PuppetLabs, 2015) (Jacob, 2012).

For a whole 12 years, CFEngine ran unopposed in the automated configuration management field; finally, in 2005, a competitor emerged when Luke Kanies, an active user of CFEngine 2, created a Ruby-based, model-driven automation tool known as Puppet (PuppetLabs, 2015). Recalling the origins of Puppet in an interview with John Willis and Damon Edwards from DevOps Café in 2010, Kanies revealed that, as a system administrator years before creating Puppet, he was frustrated with

the fact that research and development in the area of configuration management automation was not being paid the attention it deserved (Kanies, 2010).

Kanies remembers speaking with several experts in the field about his dissatisfaction with the advances, or lack thereof, that CFEngine had made with its virtually unopposed reign in the sector. While many agreed with him, he found an unsettling prevalent theme among them: an acceptance of the fact that CFEngine had been, and was the only industry standard tool in that area, and that it did not appear to be relinquishing its monopoly at any time in the foreseeable future, as no other conceivable alternatives were available. Another motivating factor for Kanies to leave his job and create Puppet, was that he felt as though there was an unnecessary gap of knowledge between system administrators and developers in terms of configuration of servers through code. He believed this gap could be bridged by making configuration management code less intimidating to developers by creating modularised, granular libraries of self-describing code and treating these the same as database or application code libraries. Kanies hoped this would help encourage both departments to learn how to add their own configuration requirements to their servers through code, code that both, development and operations departments could easily understand (Kanies, 2010).

In 2009, Chef was released by a company called OpsCode, now Chef (Robbins, 2009). Like Puppet, Chef is a Ruby-based automated configuration management tool based around the core concepts of defining a machines desired state through code and centralised modelling of infrastructure (Chef, 2015). Adam Jacob, one of the original creators of Chef, recalls the reasoning behind making the tool in a presentation he made at Chef Conf 2012: Jacob was working as an IT infrastructure consultant, building networks for start-up companies. Much like Kanies with CFEngine, Jacob was an avid user of Puppet in his day-to-day work but was dissatisfied with the standard of configuration management tools on the market at the time. He began creating Chef to increase efficiency in his company while also abstracting complex networks through self-describing code to the point where they would translate well enough to be understandable to, and to be re-used for each individual client in his company's customer base (Jacob, 2012)

Each tool discussed above has more similar than unique aspects, all three were created by those tasked with system administration, who were attempting to create a faster

and more efficient way of automating the configuration of systems, and, in doing so, whether deliberately or inadvertently, contributed greatly to the DevOps field by creating a means of cross-functional collaboration between developers and operations, which is a defining feature in the DevOps culture (Dyck, 2015).

With the inception of Puppet in 2005 and Chef in 2009 into the configuration management sector, the monopoly once held by CFEngine was no more. The widespread, and continuing success of the three tools caused a previously absent competitive market to develop around them, this, coupled with the advent of cloud computing, prompted research and development in the area to progress at a rapid rate (Nelson-Smith, 2013). As with any emerging market, the configuration management software niche became flooded with new competitors, each offering different tools, examples of such include: Rudder, Ansible, SaltStack and Rex (Rudder, 2015) (Gerla, 2013) (SaltStack, 2015) (Rex, 2015).

Arguably, the value of these tools were not seen in their entirety until the advent of AWS's EC2 in 2006 (Dadgar, 2014). Maintaining server health and uniformity throughout an expanding and contracting network via automated methods allowed early cloud adopters to realise the benefits of tools such as CFEngine, Puppet and Chef by managing the configuration of their servers with unprecedented efficiency, and their popularity has grown alongside cloud technologies (Nelson-Smith, 2013). Puppet is a prime example of this: in an interview in 2009, Puppetlabs founder Luke Kanies stated that Puppet had 1,200 users (Matt Asay , 2009). Less than 5 years later, in 2014, TechCrunch reported that Puppet had over 18,000 users, a client base increase of 300% per annum (Lunden, 2014). Along with this, in November 2015, the standard library of resources for Puppet modules had over 4.85 million downloads (PuppetLabs, 2015).

Relatively speaking, the need for several brand new machines to be setup from scratch rarely arose until the advent of the disposable cloud instance (Morris, 2016). The introduction of IaaS meant that in-house operation costs went down and IT scalability possibilities sharply increased (Nelson-Smith, 2013). The ability to easily create large-scale increases to IT infrastructure at the rapid rate AWS was offering was revolutionary, but, anyone in a technical operations role could see daunting tasks ahead of them. Automated configuration management tools ensured that these tasks were not associated with the manual configuration of each individual server, while

scripting out the configuration of machines is a massive step forward in terms of efficiency and scalability, it did leave a large gap in automation where system administrators still needed to manually manage all aspects of the higher level infrastructure components, such as those associated with virtual machines and networks, including but not limited to:

- *Virtual private networks*
- *Subnets*
- *IP allocation for machines*
- *Storage assignment for machines*
- *Access control lists*

Along with all of this, a manual log of changes to networks and VMs would need to be kept, typically for disaster recovery and rollback reasons. If an adopting organisation is not satisfied with their experience with one CSP, they may choose to switch providers or revert back to their original infrastructure setup; to do this, they would have to spend a vast amount of time and resources documenting every aspect of their networks before they could migrate them to a different datacentre. This was the scenario until very recently when tools were created to manage these lower level infrastructure components, one such tool is Terraform. Terraform was released in 2014, it was written by Mitchell Hashimoto of Hashicorp with the intention solving the problems described above while granting a means of documenting and source controlling the configuration of entire networks through code (Hashimoto, 2015). Terraform aims to create a software-managed datacentre, that is, a virtualised network of computers, the components of which are abstracted into a libraries of executable code similar to any lower level configurations of which are defined through code (Dadgar, 2014).

The evolution of the configuration management tool since its inception has taken a steady path towards encompassing every aspect of IT infrastructure, from the most basic software configuration change to creating entire networks comprised of virtualised hardware. A recurring theme throughout this history has been the aim to improve the storing of infrastructure configuration through abstract libraries of self-describing code that both developers and operation engineers can understand, manage

and contribute to. The evolution of configuration management code to IaC has led to a convergence of the two terms, with IaC being the more popularly used term.

2.3.1 Infrastructure as Code Benefits

As discussed above, the benefits of implementing IaC are numerous. As of yet, in this thesis, none of these claims have been backed up by concrete statistics, this section aims to verify these claims. In January 2015, Microsoft commissioned Forrester, an independent research based consulting firm, to determine whether or not implementing IaC technologies and principles enhances the speed of software delivery from development to production without compromising their defined processes and security (Forrester, 2015).

Efficiency in the environment lifecycle, including creation, configuration and destroying of environments has been proposed as a benefit from implementing IaC, plainly because it removes the bulk of human error by providing a means of an automated and repeatable execution process for operations which were previously manual. To cover every single one of these operations is beyond the scope of this thesis, but, a short list from the authors experience as an environment manager follows:

- *Configuring server hardware specifications.*
- *Installing/configuring operating systems.*
- *Installing/configuring applications.*
- *Applying correct patches to installed applications.*
- *Adding and removing machines to and from the correct domain.*

Repeating all of the above operations on a day-to-day basis can be cumbersome for any system administrator, and delays can occur in the application lifecycle due to human error in the environment configuration process which may require a great deal of troubleshooting to identify. For example, a new environment has been created to test a new feature, during the development phase, this feature branch has been deployed to the new environment and is throwing errors in several places where it was previously working. Several software engineers are debugging through the code

they added in order to diagnose the problem. After many hours it is found that the operating system installed on the environment is missing several patches required by the new feature, or, that the server is running an outdated version of a database engine, or, that a disk drive is missing, or, that certain directories were not set up as they should have been. Regardless of the exact culprit, the cause here is due to one or many mistakes in manual configuration that an operations engineer will have to take time out of their day to fix. This is a purely hypothetical situation, but, instances of delays directly related to mistakes made in the manual configuration of environments have been widely reported. Forrester surveyed 300 IT professionals involved with the build and release of software and asked them:

“Where in the application release life cycle do you have the greatest friction?”

Friction, in this context, relates to errors, misconfigurations or conflicts which directly cause delays. The majority of respondents stated that the provisioning and configuration of infrastructure is the 2nd highest area of friction, followed closely by the provisioning and configuration of applications (Forrester, 2015). It should not be acceptable that the misconfiguration of environments results in second and third highest areas of delays, these are the bottlenecks that IaC was designed to eliminate.

Forrester surveyed a mix of 150 development and operations engineers from different companies that had already adopted IaC frameworks and asked them the following question:

“What benefits have you achieved from utilising infrastructure as code?”

It should be noted that respondents were allowed to choose one or many benefits in order to answer this question, results pertinent to this area have been plotted in Figure 7. It can be surmised that the correct implementation of IaC can potentially provide organisations with greater efficiency in the overall environment lifecycle.

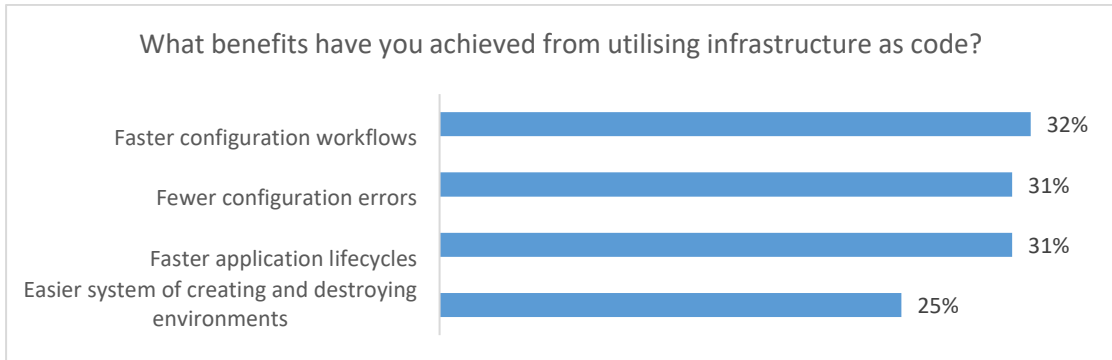


Figure 7: Reported benefits of IaC

Along with greater efficiency, comes repeatability, in the context of IaC, repeatability infers simple scalability. In the section above, the scenario where a single server needed to be manually setup was constructed. If, instead of a single server, multiple servers needed to be setup manually, then the situation changes greatly. The probability of human error causing delays in the initial setup increases relative to the number of servers to setup. People inevitably make mistakes when performing repetitive and mundane tasks, just like the calculator removes human error when performing mathematical calculations, IaC removes human error when provisioning and configuring servers. The ability to programmatically declare the desired state of a server once and apply it in an automated fashion to an array of servers allows organisations to scale rapidly. In a case study by Puppetlabs, Ben Hainline, a production operations engineer at Infusionsoft, was interviewed and queried on Infusionsoft’s experience with the configuration management tool. Hainline conveyed that the repeatable nature of Puppet allowed Infusionsoft to double the size of its infrastructure without hiring extra system administrators; Hainline is also quoted as saying “one person can manage 200 servers with Puppet” (PuppetLabs, 2015).

Another benefit of implementing IaC is the potential for resource saving. As previously mentioned, every aspect of environment creation that was once manual can now be scripted through higher level tools, such as Terraform, for virtual machine provisioning and network integration, while lower level configuration management tools, such as CFEngine, Puppet and Chef, handle the internal configuration of the virtual machine itself. Therefore, if a new environment needs to be setup, operations engineers need not spend hours or days carrying out manual tasks, they simply need to specify their requirements through code, execute said code, and carry on with their

other work. This type of workflow dramatically cuts costs associated with environment creation; when infrastructure provisioning and configuration tasks take less staff and fewer hours to complete, the organisation saves money. Mozilla's DevOps department uses Terraform to provision and maintain its IT infrastructure and claims that the use of IaC allows for an environment to be fully setup in a single working day, when they compared their pre-Terraform environment creation workflow to their current setup, Mozilla estimated that they save up to 500 operations staff hours per year (Hashicorp, 2015). It can be concluded that the benefits of adopting IaC have proven to be exhaustive (Hashicorp, 2015) (PuppetLabs, 2015) (Forrester, 2015).

2.3.2 Infrastructure as Code Risks

IaC is not without its risks and potential pitfalls, the benefits are difficult to overstate but can only be achieved when IaC is implemented correctly through changing how the adopting organisation treats IT infrastructure by educating and fostering close collaboration between operations and software engineers.

Organisations that have never utilised IaC and are planning to adopt it may encounter problems; nearly a third of organisations in this situation that were surveyed by Forrester stated that they feel their staff lack the expertise to implement IaC effectively (Forrester, 2015). Taking this into account, new staff may need to be hired or existing staff may need to undergo intensive training and possibly move to different roles in order to create and maintain IaC for organisations that have no previous history in the area. Questions pertaining to the skillset required and actual responsibilities of these new or retrained staff then arise. Will these new or retrained staff be operations based with development knowledge, vice-versa, or will an entire new team, dedicated to IaC need to be created? The bulk of cited problems with IaC relate to its adoption because it is not a traditional paradigm in the IT field, rather it is an interdepartmental technology that requires a great deal of effort to adopt and utilise to reap its benefits.

Monetary investment and staff training are naturally required when adopting any type of new technology, but adoption of IaC is not as simple as a new tool that one person or one department will use, it is of paramount importance that both development and

operations departments are equally involved in all aspects of IaC. This is because a wide variety of interdepartmental problems can stem from an incorrect adoption of IaC and associated principles. Forrester surveyed 150 IT professionals already utilising IaC and found that the conflict between development and operations department preference for specific tools and languages is the most difficult area when adopting IaC (Forrester, 2015). It is true that development and operations play two completely different roles in most organisations and forcing them to integrate and collaborate will inevitably cause conflicts of interest, especially if the benefits of IaC are not realised by everyone involved.

2.4 Infrastructure as Code and Infrastructure as a Service

The intersection of IaC and IaaS should be clear to any reader at this point, the two are complimenting technologies and have evolved to co-exist with one another. A point that should be considered here is that without the affordable scalability potential offered by IaaS that IaC would not have become as popular and as powerful of a tool as it is today, as discussed above, trends show that IaC usage and progress as a technology has increased significantly in the years after the release of EC2. It is cited that IaC is the natural path of progression for IT management to take in the cloud era, Morris is one such author, remarking that:

"The Infrastructure as Code approach is essential for managing cloud infrastructure of any real scale or complexity" (Morris, 2016).

Morris outlines challenges in managing the overwhelming amount of affordable infrastructure offered by the cloud, the most relevant to this paper are:

1. *Server Sprawl*

The ability to create new servers on-demand with little cost overhead can lead to IT teams being unable to manually manage them properly. Server sprawl can lead to configuration drift.

2. *Configuration Drift*

When new servers are created, the initial configuration may be consistent at the time, but over time, new systems and updates are rolled out, but the existing

servers are not updated. This leaves the old servers outdated in terms of software updates and essential configuration, and they are said to be in a state of configuration drift, which can lead to snowflake servers.

3. *Snowflake Servers*

When a server is different from all others but the difference cannot be replicated, a change has taken place on this server that causes it to either work for some unknown reason (Morris, 2016).

The challenges outlined above all stem from the element of human error, that is, the reliance on manually provisioning and configuring systems. Morris later concludes that the adoption of IaC technologies can be a solution to all of the above if implemented correctly through automated, standalone processes that require little to no human intervention (Morris, 2016).

The case study section of this body of work relies heavily on IaC tools in order to automate the migration of old, and creation of new environments in the cloud. The process to automate the above is based on the principles of effective use of IaC outlined by Burgess and later by Morris (Burgess, 1998, p. 283) (Morris, 2016).

2.5 Infrastructure as a Service Migration

Cloud migration has been defined as the deployment of an organisation's digital assets, services, IT resources or applications to the cloud (Pahl, et al., 2013). Security issues involved with the migration of sensitive data from non-cloud infrastructure to IaaS have been covered extensively in existing literature (Khan & Al-Yasiri, 2015) (Vu & Asal, 2012) (Manvi & Krishna Shyam, 2014). Another cited issue in the field is the process of the migration of non-cloud infrastructure to the IaaS platform, in particular, its technical aspects and lack of automation (Hwang, et al., 2015). This issue is particularly under researched in an industry-based setting. The migration process itself requires careful planning and typically involve custom ad-hoc execution plans based on client requirements, as the ultimate solution will inevitably vary from one client to another (Pahl, et al., 2013). A search of IEEE Digital Xplore online library revealed that there have been four generalizable frameworks proposed to

handle the migration of non-cloud infrastructure to IaaS, what follows is an overview of these frameworks, outlining the overall purpose and limitations of each.

The Migration Assessment Tool (MAT) presented by Mateescu, et al. is an online web application that provides organisations with a detailed assessment of their non-cloud infrastructure and determines what kind of IaaS solution would best suit their needs (Mateescu, et al., 2014). The MAT architecture consists of a presentation layer which handles user interaction, a business layer which creates and updates objects based on the users input and a data layer which contains objects in a database that MAT references and compares to the users input. All of the above components interoperate with one another in order to take an organisation's existing, non-cloud infrastructure as an input, map out the infrastructure within the MAT database and compute the best possible cloud-based solution and for the client. While this framework does pertain to the field of migration of existing non-cloud infrastructure to IaaS, it covers only pre-migration phase activities, it does not address the technical complexity aspects involved in carrying out such a migration or provide an automated, repeatable process for the migration itself.

Khan and Al-Yasiri have proposed a cloud migration framework for SMEs, this framework is based off the general cloud adoption challenges and solutions obtained from 72 interviews the researchers held as part of their study, interviewees range from representatives from SMEs, representatives from CSPs and developers who specialise in cloud technologies (Khan & Al-Yasiri, 2015). Khan and Al-Yasiri's framework aims to be generalizable to all service models of cloud computing and deals with all phases involved in the migration process, it's broad aim is to provide a stepwise guide for SMEs to follow for their cloud migration project (Khan & Al-Yasiri, 2015). This framework is broken down into the following three stages:

1. *Cloud Requirement Stage (CRS)*

This initial stage involves the assessment of client requirements regarding what services are to be migrated to what platform, knowledge applied in this stage is based on CSP advice and market studies.

2. *Cloud Preparation Stage (CPS)*

This middle stage is comprised of a comprehensive analysis of the adoption plan obtained in the CRS, this stage involves risk assessment regarding regulatory compliance, potential security issues and data classification.

3. *Cloud Migration Stage (CMS)*

This final stage outlines the migration and testing of live systems to the selected cloud platform

This framework is centred on industry-based, real-world requirements. It presents a guide for SMEs to decide what they can migrate, and the risks involved in doing so, there is only a small section covering the actual process of migration. As the framework encompasses all service models of cloud computing, and all service models are inherently different from one another, the migration process outlined in this paper does not cover any specific details on the process and technical details of how the migration of existing non-cloud infrastructure can be achieved.

Sabiri et al. present a framework based on the Architecture Driven Modernization (ADM) paradigm, the researchers describe a framework where legacy systems are modernized to best suit the cloud platform (Sabiri, et al., 2015). The architecture of this framework is comprised of a business layer which processes user requests and implements business logic and a data layer which stores all data for the application (Sabiri, et al., 2015). This framework involves the building of a Platform Specific Model (PSM) of the existing system to migrate and a Platform Independent Model (PIM) which is used to transform the PSM. The overall aim of this framework is to modify the existing system so that the architecture of the system fosters portability to a range of different platforms. This is achieved through a three step process:

1. *Reverse Engineering*

This first stage is comprised of the analysis of the source code of the legacy system in order to discover components, relationships and dependencies within the business logic, data layer and infrastructure layer of the system. From this analysis, a PSM representation of the system is derived, which is then transformed via the PIM transformation rules.

2. *Transformation Upgrade*

This second stage involves the optional addition of functionalities to the PIM outputted in the Reverse Engineering stage.

3. Forward Engineering

This final stage is comprised of the transformation of the PIM back to a PSM, the final output of this stage is the generation of the codebase for the new PSM (Sabiri, et al., 2015).

This framework proposes a model-based approach for the analysis and modernisation of a legacy system so that it can function on a cloud-based platform. This framework does not deal with the cloud migration process in any capacity, nor does it address the challenge of automation or implementation complexity involved the migration process.

At the time of writing, the CMO framework proposed by Hwang et al. is possibly the most pertinent piece in literature regarding the automated migration of non-cloud infrastructure to the IaaS platform (Hwang, et al., 2015). In their paper, Hwang, et al. describe the end-to-end process of cloud migration in its entirety, encompassing pre-migration, migration and post-migration phases; they also provide a semi-automated approach to the live migration of non-cloud infrastructure to IBM's Softlayer IaaS offering (Hwang, et al., 2015). The migration itself is performed by a three step process, all of which is orchestrated by IBM's Business Process Management (BPM) software:

1. The Provision Stage

This first stage is almost completely automated, it comprises the provisioning of the gateway, virtual network and VMs in Softlayer which match the non-cloud infrastructure chosen to migrate. After these resources are provisioned, a Java-based application configures them to behave in the same way their non-cloud equivalents do.

2. The Network Setup Stage

This stage involves the manual creation of a WAN connecting the non-cloud datacentre with the virtual cloud-based network created in The Provisioning Stage.

3. The Migration Stage

The final stage in the process entails the live migration of the VMs themselves, this is achieved by utilising third party migration tools such as VMWare Site Recovery Manager, vSphere Replication and VMWare Converter, all which CMO supports varying levels of automation for (Hwang, et al., 2015).

The CMO effectively tackles the issues of migration complexity and lack of automation in the migration process outlined by Mateescu et al. and Manvi and Krishna Shyam respectively, and it does so with great efficiency (Mateescu, et al., 2014) (Manvi & Krishna Shyam, 2014). In experimental results obtained from the CMO under laboratory settings, the time taken to migrate a small datacentre is 44 hours, whereas, the time taken to migrate a single VM with 200GB of disk attached is just over three hours (Hwang, et al., 2015). However, the CMO is specific to IBM's Softlayer as the target IaaS platform, and does not take into account other CSPs, therefore the issue of vendor lock-in is prevalent here (Hwang, et al., 2015). The live migration approach may be applicable for mission critical systems that require this type of migration with as little down-time incurred as possible, but live migration capability of CMO means that infrastructure is migrated to the cloud as-is. Using a live migration for legacy data centres containing a large amount of test environments where the issues of configuration drift, snowflake servers and server sprawl have already occurred will not solve this issues, rather, it will move the problems to a platform where the client is charged more for not solving them (Morris, 2016). The CMO has yet to be tested outside of a laboratory setting, therefore it lacks the validity of having been used in an industry-based setting (Hwang, et al., 2015).

The frameworks cited above all deal with various phases and activities involved in the migration of non-cloud infrastructure to the IaaS platform, for the purpose of clarity, the features of these frameworks have been summarised and plotted out in Table 4 and Table 5. Table 4 shows the specific phases each framework addresses; whereas, Table 5 shows the limitations and features of each framework.

Framework	Pre-Migration	Migration
MAT	Yes	No
Khan and Al-Yasiri	Yes	No
Sabiri et al.	Yes	No
CMO	Yes	Yes

Table 4: Existing Migration Frameworks Phase Comparison

Framework	Vendor Lock-in	Handles Migration Complexity	Automated Migration	Industry Tested
MAT	No	No	No	No
Khan and Al-Yasiri	No	No	No	No
Sabiri et al.	No	No	No	No
CMO	Yes	Yes	Yes	No

Table 5: Existing Migration Frameworks Features and Limitations Comparison

The MAT and the frameworks proposed by Sabiri et al. and Khan and Al-Yasiri all address the pre-migration phases of assessment and planning. They are all free from the issue of vendor lock-in as they are cloud agnostic in their methods. However, they offer no form of automated migration, they do not deal with the technical complexity of performing such a migration and they have never been tested in an industry setting. To the author’s knowledge, the CMO is the only available framework that handles an end-to-end migration scenario, encapsulating the assessment and planning activities in the pre-migration phase alongside the technical process of the migration of non-cloud infrastructure to the public cloud. The CMO offers a semi-automated approach to the migration process but it is specific to IBM’s Softlayer IaaS platform and has not been tested in an industry setting (Hwang, et al., 2015).

2.6 Conclusion

It is clear from reading the above that cloud computing is the most recent product of several decades of IT evolution from relatively simple beginnings in the 1950s. As a technology, the modern form of cloud computing is highly disruptive, and is rapidly changing the world of IT.

This is especially true for the IaaS model which recently outperformed its on-premises equivalent in terms of workloads, as mentioned above. The market is in a state of transition as organisations with IT infrastructure flock to major CSPs to take advantage of the many proposed benefits of adopting leased infrastructure.

The risks of adopting the IaaS approach are still widely controversial, with the ever emerging media reports of compromised cloud-based data and data centre outages causing havoc to organisations. It is the opinion of the author that human beings

mistrust change, and a change as dramatic as leasing out IT infrastructure through the Internet is bound to be met with scepticism, intense scrutiny and bias for several years after reaching mainstream popularity. Organisations wary of IaaS should be made aware that major CSPs aim to offer the most secure service possible, constantly striving to win the most stringent security awards available. The six mentioned in this chapter were the most recommended to have for those seeking secure 3rd party infrastructure, but they are six of numerous accreditations and awards that most major CSPs hold. IT security should be a high priority for any sized organisation with IT infrastructure, but most organisations security standards do not come close to matching that of industry giants such as Microsoft, Google or Amazon, each of which have years of experience in managing large scale data centres in a highly secure manner. Natural disasters occur, as does human error, as do power outages, the effects of each of these can materialise in any data centre, be it a small, on-premises server room with a single rack or a huge CSP data centre.

The benefits of adopting IaaS are numerous, among them are the elimination of cost overheads associated with procuring, housing and maintaining physical servers alongside the ability to scale at will to virtually unlimited capacity or rapidly downsize without incurring significant cost associated with decommissioning of physical machinery. Although the ability to scale at will with little restriction raises problems of its own, with configuration drift, non-uniformity of environments and undocumented changes to infrastructure and server configuration among the top offenders (Morris, 2016). It is argued by many that the solution to these problems come in the form of IaC (Dadgar, 2014) (Forrester, 2015) (Morris, 2016) (Nelson-Smith, 2013). The relatively new idea that entire networks, including the granular configuration of individual servers can be scripted out, source controlled and deployed in a repeatable manner to overcome the issues of maintaining the plethora of IT infrastructure available as a service through cloud computing.

New organisations have the choice to either create their entire IT systems native to the cloud or build their own data centre, however, prior to the launch of AWS's EC2 in 2006, the option to build cloud-native IT systems was not available and the de facto standard was to build a datacentre using physical servers (Barr, 2006). For organisations with IT infrastructure pre-dating 2006, the option of migrating the cloud is available, but the process of doing made extremely difficult by the fact that each organisation has its own specific migration requirements and the solution chosen for

migration is typically custom built for the each individual organisation (Pahl, et al., 2013). There are frameworks such as the MAT and the frameworks proposed by Sabiri et al. and Khan and Al-Yasiri which aide organisations in the planning and assessment phases of their cloud migration projects, but these frameworks do not handle the technical complexity of performing such a migration, nor do they offer any form of automated and repeatable process for the migration of large sets of testing environments (Mateescu, et al., 2014) (Sabiri, et al., 2015) (Khan & Al-Yasiri, 2015). CMO presented by Hwang, et al. does address the aforementioned issues of migration complexity and automation in the migration process (Hwang, et al., 2015). This framework does offer an automated and repeatable process, but it is locked to IBM's Softlayer IaaS platform, has not yet been tested outside of laboratory conditions and does not solve the issues of configuration drift, snowflake servers or server sprawl (Hwang, et al., 2015) (Morris, 2016). From analysing existing literature in the area, the conclusion can be drawn that there currently exists no automated framework that allows for the migration of non-cloud infrastructure to the IaaS platform that has been tested in an industry-based setting and deals with the issues outlined by Morris (Morris, 2016). In fact, at the time of writing, the only available industry-based paper in the IEEE Digital Xplore Library on the migration of existing, non-cloud infrastructure to the IaaS platform is Khajeh-Hosseini, et al., however, no migration was carried out as part of this study (Khajeh-Hosseini, et al., 2010).

Chapter 3. Design and Implementation

This chapter provides context regarding the architectural and design and implementation involved in this body of work. This chapter starts with a brief outline of the case study carried out in the target organisation. This is followed by detailed sections pertaining to the architecture and specific technologies used in the implementation of an automated framework of interlinked IaC and configuration management scripts. This is followed by a use case of the framework which provides a clear context to its preceding sections and a knowledge base of the sequence of technical processes involved in the running of automated framework. This chapter ends with a section on the experimental use of the framework which allowed the case study organisation to migrate their existing colocation based IT environment infrastructure to AWS's IaaS platform and create new IT environments on AWS's IaaS platform.

3.1 Case Study

The case study took place over the course of a 5 month period and involved the placement of the researcher within the target SME. The overall purpose of the case study was the gathering of functional and non-functional requirements for the automated framework in the context of the case study organisation. The case study also shaped the creation of a detailed project plan for the automated migration of the case study organisations non-cloud infrastructure to the AWS IaaS platform. The above was done through a phased process consisting of two distinct phases, both of which are outlined below, followed by a detailed description of each phase throughout the 5 month period:

1. *Exploratory Phase*
2. *Project Planning Phase*

3.1.1 Exploratory Phase

This phase began on the 1st of November 2015 and ended on the 22nd of January 2016. The purpose of this phase was to gather client requirements, which were then used to construct the architectural design of the framework. In order to achieve this, a detailed analysis of the organisation's traditional manual environment creation process was carried out, with a focus on the tasks performed, alongside the time and effort overheads imposed by carrying out each task. By engaging with staff belonging to the organisation, the researcher built a base of knowledge around the manual in-house environment creation process the organisation followed to create their environments and also identified three key participants in the organisations manual environment creation process, each working within an individual and unconnected technical department in the organisation.

The researcher conducted semi-structured interviews with these three staff members. These interviews revealed an in-depth set of tasks that each participant must carry out before handing the environment over to the next participant. From these interviews, the researcher grouped each task that takes place in chronological order during the entire manual environment creation process and abstracted them into the following six high-level groups:

1. Provisioning of the new infrastructure.

This task comprises the creation of a new virtual machine from an existing virtual machine. Included in this task are IP address, compute power and storage allocation. This task is largely manual and is performed by a member of the infrastructure department.

2. Documentation of the new infrastructure.

Documenting the specifications, location in the network and name of the new environment is done by amending a Visio diagram with the above information. This diagram is stored in a shared location that relevant employees within the organisation have access to. This task is completely manual and is performed by the infrastructure department.

3. Performing Active Directory domain operations.

This task involves two steps, the first is carrying out a Sysprep on the new machine. Sysprep is a Windows specific generalisation tool which is used when one Windows computer is cloned from another Windows computer, it remove all traces of a previous machine from the cloned machine (Microsoft, 2017). The second step in this task is to rename the new machine to a meaningful name that falls in line with the organisations server naming conventions. The third and final step is to add the machine to the correct organisational unit in the domain, which essentially allows the new server to become part of the organisations network of computers (Desmond, 2008).

4. *Creating the Domain Name System (DNS) entries for the environment.*

There are two separate kinds of DNS entries to be setup in this task. The first are simple Active Directory DNS entries which allow users connected to the organisations internal network to connect directly to the new server using the A and CNAME entries created in this task. The second type of DNS entries required for creation at this stage are the external DNS entries which allow users outside of the organisations network to connect to the sites on the new server via a web browser. These external DNS entries are not hosted within the organisation, rather, they are hosted by a third party DNS provider. This task is completely manual and is performed by the infrastructure department.

5. *Setting up the environment specific configuration on server.*

This task involves the modification of configuration files on the new server so that the old environment values are removed from them and the new environment values are inserted into them. Specific examples of these configuration files include system files such as the HOSTS file and machine.config file, along with application and website specific configuration files such as web.config and app.config files. Internet Information Services configuration files also need to be modified in this step. This step is completely manual and is performed by the release management department.

6. *Deploying the organisation's Application and Database (A&D) codebase to the new server.*

The final step in the process is the deployment of the latest release of the organisations A&D codebase to the new server. There is a large amount of code

from a range of different branches that is required to be deployed at this step, specifics on the size and number of branches that are deployed are discussed in section 4.1.3 of this thesis. This step is largely automated by existing deployment procedures, however, manual input is required in multiple places, and a significant amount of manual work is involved in monitoring the deployments and troubleshooting errors if they occur. This step is performed by the release management department.

These processes are heavily referenced in the sections that follow and play an important role in the architecture of the working system. The results of these interviews also formed the benchmark for the manual environment creation timings that became a key comparative variable in later sections of this document, the full transcripts of said interviews can be found in Appendices A, B and C. Once the researcher had a comprehensive understanding of the organisation's manual in-house environment creation process, this phase ended and was succeeded by the Project Planning Phase.

3.1.2 Project Planning Phase

This phase took place between the 25th of January 2016 and the 1st of April 2016. The scope of the migration project for the case study organisation was created in this phase. The initial project scope entailed a complete migration of the organisation's testing, staging and production environments to AWS's IaaS platform. As the project was being planned, the scope began to narrow due to two impediments, one major impediment and one less severe, both will be discussed in this section. The researcher believes these impediments and their consequence merit discussion in this section as both had a direct effect on the design and implementation of the framework and should give the reader an understanding of how industry requirements and academic research are not always aligned with one another.

The first impediment pertains to security which has been detailed by Sadiku, et al. as the greatest challenge when adopting public cloud infrastructure (Sadiku, et al., 2014). This security issue pertains to the compliance issues with data belonging to the clients of the case study organisation. One client in particular has a specific agreement with the case study organisation that they reserve the right to inspect the

physical machinery that their sensitive data resides on, inclusive in this clause is any data which relates to personally identifiable information. The implication was that, the servers that host the front-end applications that the clients interact with and enter data into, along with the servers that host the databases which contain the client interaction information and associated data must be geographically locatable and accessible if that client wishes to inspect it. In the case study organisation, this is typically done via the client sending out an IT engineer on their behalf to inspect the machine for physical faults and ensure it has not been tampered with in any way. The client has an agreement with the organisation that no specific reason needs to be given for this kind of inspection to be warranted.

This was an issue as it was found that Amazon follow a shared responsibility model, visualised in Figure 8, in which the client who is leasing infrastructure is responsible for all aspects of the data they host on that infrastructure, who can access it and how it's accessed, whereas AWS assumes the responsibility for securing the lower level layers, starting from the virtualisation layer of the physical machines all the way down to the security of the facilities in which the machines reside (Amazon, 2016).

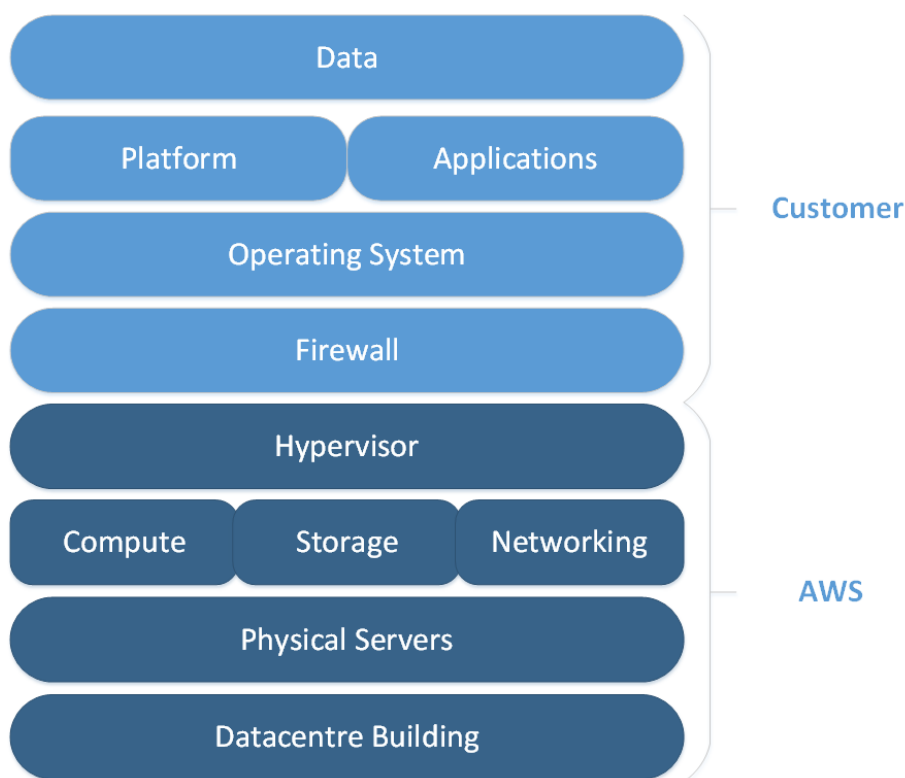


Figure 8: AWS Shared Responsibility Model

The responsibility of the security of physical machines is out of the control of AWS's clients, therefore, AWS do not allow any of their clients to physically inspect the computing machinery in their data centres, nor do they disclose the specific location of their machines or data centre buildings to their clients (Amazon, 2016).

As a result of this, the project scope had to be narrowed down to exclude all production and 3rd party testing environments, as these environments inherently contain sensitive client information. Only data necessary for functional testing of the organisation's systems that is not linked to any real person was allowed to be hosted on AWS infrastructure as part of this project. The project moved ahead regardless of this, encapsulating only internal test environments that contain dummy data required for development and testing.

At the project outset, eight existing internal testing environments needed to be migrated to the public cloud in a very small amount of time in order to minimize downtime for staff who would be actively using these environments. Another requirement that was agreed upon was the building of new testing environments native to the cloud. A system needed to be created that was versatile enough to handle both of these scenarios without differentiation.

It was planned to migrate the existing test environments directly to AWS, meaning they were going to be exported as machine images from the colocation centre and directly imported as AMIs across the Internet to the AWS data centre. AMIs are stored in S3, and there is no transfer cost involved in incoming data, therefore, this approach was seen as a straightforward and economically feasible one (Amazon, 2016). Following this approach, each individual environment would need to follow a relatively simple migration process, outlined below:

- 1. Take server off the organisation's domain.*
- 2. Sysprep and shutdown instance.*
- 3. Export server as a machine image.*
- 4. Import machine image to AWS as an AMI.*
- 5. Launch as an EC2 instance.*
- 6. Add instance to the organisation's domain.*

7. Modify all DNS entries that referenced the old machine to point to new instance in the cloud.

However, upon further scrutiny, problems with this approach quickly began to emerge. The cost overhead associated with duplicating each of individual environment's disk drives in AWS is one such factor. For instance, if eight environments with 250GB of disk space were migrated following this approach, then there would be eight imported AMIs taking up a combined total of over 2TB of disk space and eight instances with separate storage also taking up a combined total of over 2TB of disk space, essentially this would be doubling the amount of provisioned S3 storage. Along with this, applying patches, installing updates and new applications to each separate AMI and its associated instance also becomes a problem as maintenance of this type of system is potentially very inefficient and lacks scalability if more environments were to be migrated. This type of system also neglects new environments that are created native to the cloud, so an entirely different system would need to be designed for creating these new environments in AWS. Therefore, need for a single AMI that has the minimum amount configured on it (i.e. specific operating system, disk drives, etc.) arose. The environment specific configurations were to be defined through IaC in the form of Terraform and configuration management in the form of Puppet. Any updates that needed to be installed can be done through either configuration management code on the instances themselves or installed on the single AMI that the process would build instances from.

The concept of building new machines in the cloud brought about its own challenges, and led to the second impediment, which pertains to a section of the environment creation process that could not be automated through code, namely the setting up of the external DNS entries for the websites that are hosted on the environments. In the case study organisation, existing networking layer components such as internal IP addresses, internal and external DNS entries can all be reused for machines that were to be rebuilt in the cloud. For new machines being built in the cloud, all of these entries needed to be created. It was desired that any infrastructure, including networking, created in the cloud could be done through source controllable IaC. However, an issue was recognised early on in the case study that limited the scope of the automation. The DNS service provider that the case study organisation was subscribed to offered no Application Programming Interface (API) for the creation and modification of DNS entries, essentially meaning that the DNS provider did not

allow for IaC tools to create and manage DNS entries. These external DNS entries had to be created manually for any new environments being built in the cloud, which became a large gap in the process of automation.

3.1.3 Functional Requirements

Based on the above, the following functional requirements the framework were derived:

- 1. The framework should be capable of rebuilding the case study organisation's existing internal testing environments on an IaaS platform in an automated fashion.*
- 2. The framework should be capable of creating new testing environments belonging to the case study organisation on an IaaS platform in an automated fashion.*
- 3. The framework should be capable of outputting meaningful errors and terminating upon a non-zero exit code of any underlying script.*

3.1.4 Non-Functional Requirements

The non-functional requirements for the framework are as follows:

- 1. The framework should be capable of achieving the functional requirements with a single server image.*
- 2. The framework should only have a single human operator.*
- 3. The framework should abstract the underlying processes to the operator in a meaningful way for troubleshooting purposes.*
- 4. The framework should have a single entry point of execution.*
- 5. The framework should have a single point of monitoring.*

3.2 Framework Architecture

Based on above requirements derived from the case study, the high-level architecture of the automated framework was devised. The basic layout of this architecture is visualised in Figure 9, each component outlined in this diagram will be discussed in the sections that follow.

The Exploratory Phase revealed the key processes that the framework must automate and run in a very specific order to be successful. These processes range from system administration to database administration to developer tasks. It was not possible to automate the entirety of these processes through a single programming language or toolset, the researcher acknowledged that the framework would consist of a large number of different types of scripts that needed to be run in a particular order, the time spent running each of these scripts manually would be too much of an overhead to gain a significant efficiency benefit from. Therefore, the researcher first set out to determine a suitable engine capable of running different scripts in sequential order and in a variety of a languages while still handling errors and outputting log messages for informational and debugging purposes. It was planned that this engine would account for the foundation of the framework, the single suite that acts as an abstract front end for user input while also running the complex code the framework consists of.

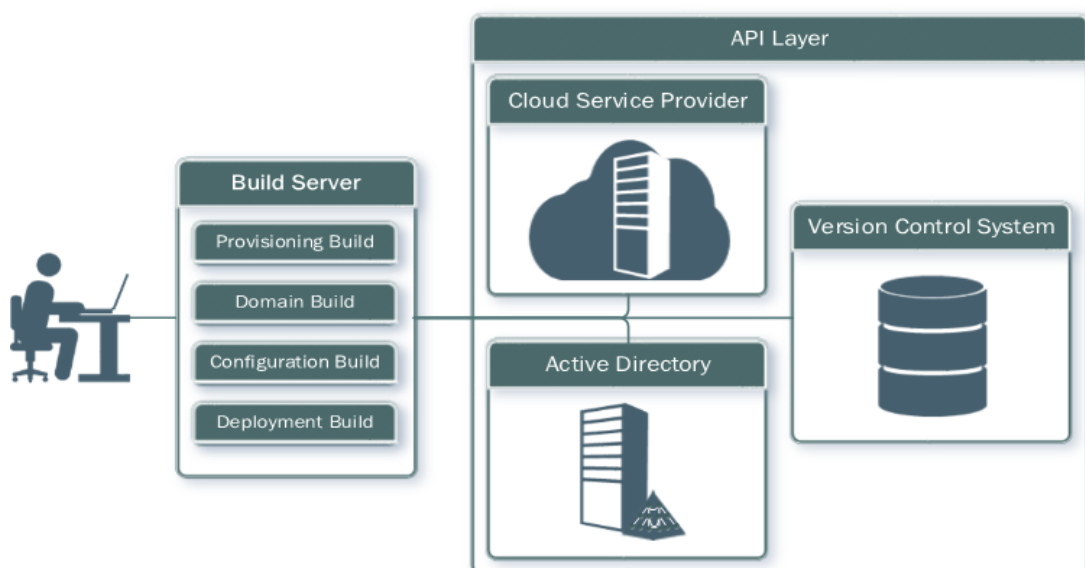


Figure 9: Framework Architecture Overview

3.2.1 The Build Server

A build server, commonly known as an automation or continuous integration server, is a software tool dedicated to compiling, executing and deploying source code through repeatable steps that the user explicitly defines (DevIQ, 2017). A build server typically has the capability of running several different types of scripts from a variety of languages. It allows users to define, modify and execute processes that can be comprised of a variety of different types of source code, essentially chaining a set of scripts together to act as a single process. As the scripts are all linked to one another by the build server, the user only needs to enter in a single set of parameters that all scripts in the process share. This allows for multiple scripts to be executed in sequence via a single user interaction while allowing each script to share a common set of variables defined by the user (Alexandrova, 2016). The terminology for a process or job that a user defines in a build server is called a build configuration, or simply, a build (Melymuka, 2012). Therefore, all subsequent references to processes or jobs that are handled by the build server will herein be referred to as “builds”.

The concept of build chaining is highly important in the design of the framework, a build chain is a series of linked builds which execute sequentially in order to achieve a desired result. The idea of sharing user parameters from one build to another expands the idea introduced above, whereby, a single set of parameters is shared throughout several scripts that make up a single build. With build chaining, this same single set of parameters can be shared throughout several individual builds in a build chain with only a single user interaction to start the chain of builds (Melymuka, 2012). This concept, and its importance in the framework is discussed more in later sections of this chapter.

3.2.2 API Components

An API is a tool designed to give programmers a method of accessing an external software system and integrating it into their own software system (Michel, 2013). Similarly, web service APIs are APIs that can be interacted with through the Internet, they are designed to give external software programmatic access to certain functions of a web application, for instance, the Google API allows for programmers to utilise

Google search functions directly from their own software (Gosnell, 2005). Web service APIs are vital architectural elements that make up the framework, the IaC scripts that execute as part of the framework query different web service APIs through HTTP to automate a variety of tasks. There are three API components that the framework interacts with:

1. *Cloud Service Provider (CSP)*
2. *Active Directory (AD)*
3. *Version Control System (VCS)*

These APIs, and their specific purposes, are discussed in this section.

The Cloud Service Provider API allows for new virtual machine instances to be created and configured in the public cloud. This API component is interacted with via an IaC tool which authenticates to the user's CSP account and sends a HTTP request to create a new virtual machine on the CSP's infrastructure. These operations are performed purely through code, with minimal human intervention.

The Active Directory API allows for the programmatic addition of servers to an organisational unit within an Active Directory domain. This API also grants the ability to add, remove and modify internal DNS entries through code.

Adding a server to an Active Directory organisational unit within a domain grants the server the same privileges as any other server in that organisational unit, this essentially makes the new server part of the organisation's network of computers (Desmond, 2008). In most cases, this will mean that users in the internal network will be able to access the machine, and all group policy rules defined for that organisational unit will be enforced on it (Solomon, 1998).

The DNS in Active Directory resolves hostnames to IP addresses; address records resolve hostnames to an IP address, while Canonical Name (CName) records map an alias to a hostname, provided the hostname already has an address record in place (Desmond, 2008). For example, the following address record resolves all requests to the *sample.com* hostname to the *10.40.69.216* IP address:

sample.com IN A 10.40.69.216

Changing the IP address of this record would direct requests to the *sample.com* hostname to the IP address of a different machine. Provided the above address record is in place, the following CName record resolves requests to the *web.sample.com* to the *sample.com* hostname, which in turn, resolves to the *10.40.69.216* IP address:

web.test.com IN CNAME example.com

The addition and modification of these DNS records play an integral part in the framework and is discussed in later sections of this thesis.

The Version Control System (VCS) API provides functionality to programmatically add and modify files in a VCS repository. In the framework, it is used to add, modify and track changes to IaC files in an automated fashion. In this way, the self-describing IaC scripts act as documentation of the public cloud infrastructure. All changes to the infrastructure is done through versioned code, which can be reverted back to previous states. This feature also allows for disaster recovery, if a virtual machine is ever unintentionally destroyed, corrupted or compromised in any way, it can be replaced or reverted back to a known working state via the IaC script that was added to the VCS.

3.2.3 Framework Builds

There are four builds in the framework, they are automated abstractions of the processes outlined in the Exploratory Phase and are listed in order of execution below:

1. *Provisioning Build*
2. *Domain Build*
3. *Configuration Build*
4. *Deployment Build*

Each build is composed of several different kinds of scripts. For architectural purposes, the scripts themselves will not be discussed in this section, rather, an overview of each build is covered here, and the scripts themselves are covered in more detail in the Technologies Used section.

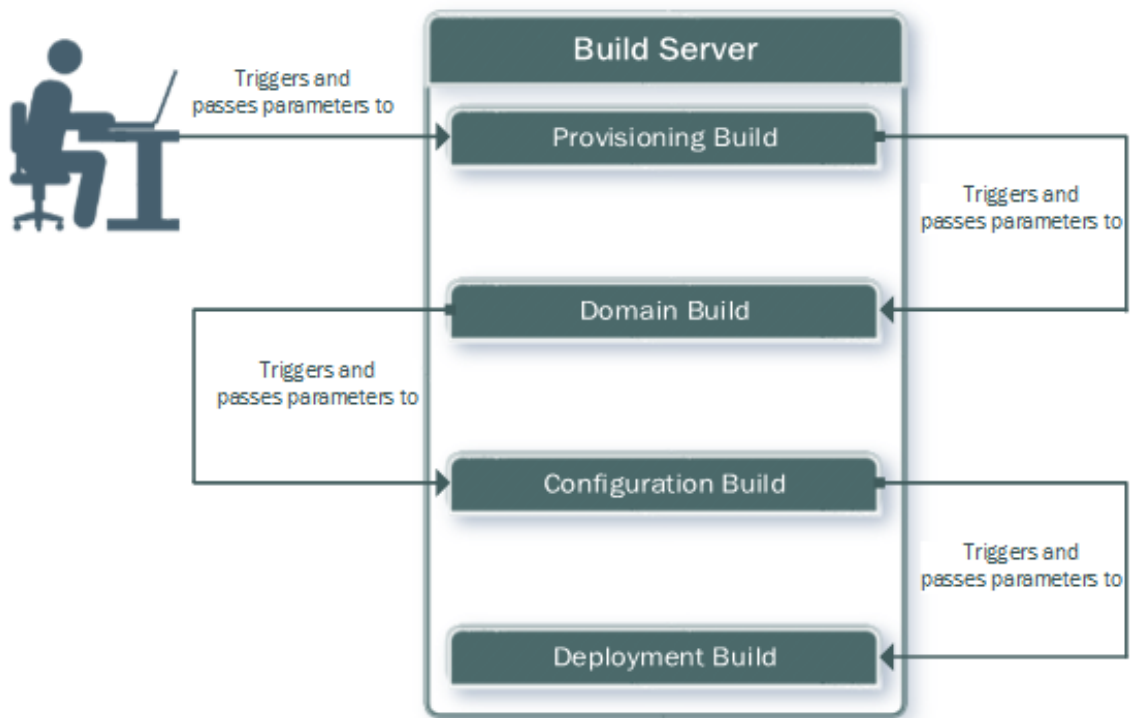


Figure 10: Overview of the Build Chain

The Provisioning Build is the first in the build chain, it automates the first two steps in the environment creation process, which are:

- *Provisioning of the new infrastructure.*
- *Documentation of the new infrastructure.*

Using an IaC tool, the Provisioning Build firstly interacts with the CSP API in order to authenticate to the organisation's CSP account, it then creates a new virtual machine in the cloud. The end results of this first operation are a new cloud-based virtual machine and a self-describing IaC script that essentially documents the new virtual machine in the cloud.

The final operation the Provisioning Build carries out is the addition of the new IaC script to the VCS, it does this by interacting with the VCS API. The automated addition of new scripts that describe environments is not functionally necessary for the framework to carry out an automated environment creation, but, incomplete infrastructure documentation and the inability to easily roll-back to a previously known working state both have the potential to become large issues in a disaster recovery scenario. By storing the resulting IaC scripts in the VCS, the new

infrastructure is documented in a central location and it allows the organisation to have version controlled executable scripts that provide a history of known working states, these are indispensable in a disaster recovery situation.

The Domain Build is the second build in the chain, it automates the third step in the environment creation process, which is:

- *Performing Active Directory domain operations.*

The Domain Build performs several operations that comprise what is described above as domain operations. Firstly, the new cloud-based virtual machine is queried to ensure it is available, following its creation and initial boot. Then, the new virtual machine is renamed from its default name. A command is issued to add the new virtual machine to the Active Directory domain. Lastly, the domain controllers are synched to ensure that the new machine entry is propagated through the Active Directory forest.

It is at this stage that the new virtual machine has become part of the organisations network of computers. The organisation the framework is implemented in use Active Directory internal DNS entries to for a variety of purposes (RDP, SQL sessions, inter-system communications, etc.), these internal DNS entries are essential for this specific organisation, but may not necessarily be required elsewhere. A requirement for the organisation was to have the creation and modification of these internal DNS entries automated, which is the last operation the Domain Build performs.

The Configuration Build is the third build in the chain, it automates the fifth step in the environment creation process, which is:

- *Setting up the environment specific configuration on server.*

The Configuration Build executes configuration management code in order to install applications and configure environment specific settings on the new virtual machine. Examples of operations performed in this build through configuration management code are:

1. *Windows service creation*
2. *Windows service configuration*
3. *Directory creation*

4. *Assignment of directory and file permissions*
5. *Internet Information Services configuration*
6. *Application installation*
7. *Application configuration*
8. *Modification of environment specific configuration files (HOSTS, machine.config, etc.)*

All of these configuration management operations prime the new virtual machine for use and ultimately facilitate the deployment and correct operation of the organisations Application and Database code.

The Deployment Build is the last build in the chain, it automates the sixth and final step in the environment creation process, which is:

- *Deploying the organisation's codebase to the new server.*

The Deployment Build calls existing builds which deploy all Application and Database code to the new virtual machine, populating all application directories created by the Configuration Build with compiled code that comprise the applications and deploys the databases that the applications use. The success of this build is the final requirement before a virtual machine can be considered a fully-fledged test environment that can be handed over to a development or testing department.

As discussed in the Project Planning Phase section, it was not possible to automate the fourth step (*Creating the external DNS entries for the environment*) in the environment creation process, this is a consequence of carrying out this research in a real-world industry setting as opposed to a hypothetical laboratory environment. This step remains a manual step in this implementation of the framework.

A chain of linked build configurations that contain calls to the underlying scripts were required to allow for the framework to have single user interface. This single user interface prompts the user for parameters, the values the user provides are passed to each build in the chain which ultimately determine the framework's behaviour and end result. For example, the user can provide the build chain with different instance types which determine the compute power of the instance that the IaC tool will create or different machine images on which to base the instance being provisioned. The

theory behind the build chain is simple, each build configuration is linked in sequential order, meaning that, when one build finishes, the next begins. The user interacts with the first build in the chain, executing it with a set of parameters that determine what kind of machine to create and where to create it, when this first build is successful, the next build in the chain is executed with the same set of parameters that the user initially supplied, and so on until the end of the build chain is reached, the basic overview of this system is shown in Figure 10.

3.2.4 Framework Prerequisites

The framework presented above essentially recreates existing non-cloud IT environments in the public cloud, therefore, it is assumed that a structured network infrastructure is already in place for the organisation that is utilising it and the necessary networking components on the CSP's side are also already in place. As prerequisites for the operation of the framework, the adopting organisation must already have the following components in place:

1. *An existing internal network infrastructure*

An Active Directory domain with at least one domain controller is required for the framework to operate. Active Directory objects such as computer accounts and internal Active Directory DNS are modified during the execution of the framework.

2. *A build server with pre-configured deployment processes*

The front-end for the framework is to be configured in a build server such as Jenkins or TeamCity. This build server must already contain a deployment process for the applications and databases that reside on the environment to be rebuilt in the public cloud. This build server must be capable of accessing all of the API components outlined in section 3.2.2.

3. *At least one existing fully configured IT environment in place*

This existing environment is used to fashion the configuration management scripts that will define that environment in the cloud. The existing IT environment must

be placed in an Active Directory domain, and have standard CName and A record Active Directory DNS associated with it.

4. *A pre-configured VCS*

The dynamically created infrastructure as code scripts and the pre-configured configuration management scripts must be source controlled in a VCS that the build server can access via an API. A file share that the build server can access would be adequate as a pre-requisite but a VCS is the ideal storage scenario for this code.

5. *A CSP account*

In order for the framework to create new infrastructure in the public cloud, it requires a CSP account to interact with. The API keys for this CSP account must be provided to the framework in order for it to interact with the specific account.

6. *A pre-configured virtual network on the CSP side*

Networking components for each of three major CSPs considered in this study are discussed in section 2.2 of this thesis, a common element from all three CSPs is the virtual network offering. This virtual network is required for the framework to operate as the framework creates instances within a user-defined subnet. The framework expects this subnet to already be in place, in order for a subnet to be created on the public cloud, it needs a virtual network to reside in.

3.2.5 Summary

This architecture presents a framework in which the user carries out a single interaction with a web based front-end in the form of a build server. Defined in this build server are four chained build configurations, which are comprised of a variety of scripts. These scripts interact with three API components in order to provision, document and fully configure a new virtual machine in the cloud so that it can be used as a testing environment.

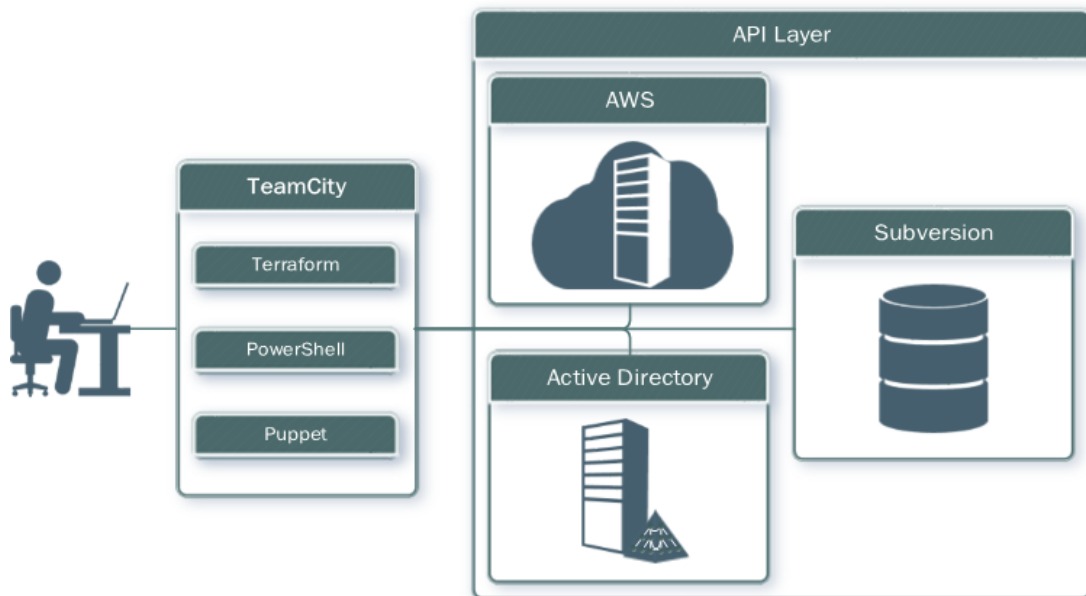


Figure 11: Technologies Used

3.3 Technologies Used

This section describes the technologies used in this implementation of the framework within the case study organisation. The scripts that comprise each of the four builds described in the Framework Architecture section above are discussed in this section. Included in this discussion is a description and justification of the choice of the software tools and programming technologies used in this particular instance of the framework.

3.3.1 Cloud Service Provider - AWS

AWS's IaaS platform was chosen as the CSP for this implementation of the framework, sections 2.2.4 and 2.2.5 of this thesis deal with the choice of this particular CSP over others that were considered for use in this study.

Technology Name	Software Name	Version
Build Server	TeamCity Enterprise	10.0.2
Infrastructure-as-code tool	Terraform	0.9.8
Version Control System	SVN	1.9.3
Configuration Management tool	Puppet	3.4.3
General Purpose Automation	PowerShell	4

Table 6: Summary of technologies used in the framework

3.3.2 Build Server – TeamCity

At the time of implementation, the case study organisation was already running a TeamCity build server to handle their application and database code deployments, it was decided that this build server would be the most appropriate engine for the framework.

The decision to use TeamCity in this implementation of the framework took into account the convenience of having the build server on-boarded and actively used in the case study organisation, along with unique benefits of using TeamCity over other build servers such as Jenkins or Team Foundation Server. TeamCity supports build chaining, parameter sharing across builds and a single user interface for build chains without any additional plugins or modification of code (Melymuka, 2012). Whereas, at the time of the design phase, Jenkins allowed for build chaining as standard but only allowed for parameter sharing across builds in a chain via a custom plugin (Whetstone, 2016). Under similar constraints, Team Foundation Server only allowed for build chaining through the use of custom code (Jacob, 2009).

3.3.3 Infrastructure-as-code - Terraform

Terraform was chosen for the IaC tool in the framework, it is used for creating virtualised instances in AWS, assigning storage to the new instances and placing the instances in the correct network. Terraform was chosen because it is an open-source and standalone command-line tool that uses a simple, declarative programming language to define the scripts it runs (Brikman, 2017).


```

provider "aws" {
  access_key = "xxxxxxxxxxxxxxxx"
  secret_key = "xxxxxxxxxxxxxxxx"
  region     = "us-east-1"
}

resource "aws_instance" "instance01" {
  instance_type = "t2.large"
  ami           = "ami-15ab4878"
}

```

Figure 12: Sample Terraform Script

Terraform is also cloud-agnostic, meaning that it can build infrastructure from code on a wide variety of infrastructure hosts, including AWS, Microsoft Azure and Google Cloud (Brikman, 2017) (Terraform, 2016). This is compared with tools such as AWS's alternative called CloudFormation which currently only works on AWS's infrastructure, and Azure's ARM template deployments which are specific to Microsoft's infrastructure (Somwanshi, 2015) (FitzMacken, 2016).

Terraform works by making API calls to a CSP in order to provision and configure infrastructure in the cloud, it abstracts the complexity of this process into simple Terraform scripts that determine what kind of API calls to make and what CSP to make the calls to (Brikman, 2017). A very simple Terraform script is outlined in Figure 12, this script authenticates to AWS using the access and secret keys defined in the provider section and builds a virtual instance in AWS's us-east-1 region.

3.3.4 Version Control System - Subversion

Subversion was chosen for the VCS component in the framework for a variety of reasons. It is open source and was already being used by the case study organisation as version control for their application and database code as it integrates with the TeamCity build server without the need to install any additional software (Revyakina, 2016). Subversion also has a simple command line interface that allows for the automation of all actions carried out by the framework, specifically, the checking-out of local copies of source, addition of new files to source and modification of existing files in source (Collins-Sussman, et al., 2011).

```
##### Required Directories
file { ['D:\websites\site1',]:
  | ensure => 'directory',
  }
##### Application Pool
iis_appool {'site1':
  ensure           => present,
  autostart        => 'true',
  }
##### IIS Site
iis_site {'site1':
  ensure           => present,
  bindings         => ["http://:80:site1-webqa.test.net",
  }

```

Figure 13: Sample Puppet Script

3.3.5 Configuration Management - Puppet

Puppet was chosen as the configuration management tool for the framework. Puppet was chosen over other tools in the area for its ease of use through its command line interface, its simple and self-documenting language along with its wide user base and versatile user-driven community of pre-configured modules (PuppetLabs, 2015) (Lunden, 2014). A simple puppet script is shown in Figure 13, when executed, this script will:

- Create a directory at 'D:\websites\site1'.
- Create an application pool for 'site1'.
- Create an IIS site for 'site1' with bindings for the site on the HTTP port 80 for all requests going to 'site1-webqa.test.net'. Meaning that, all requests to 'http://site1-webqa.test.net' will be forwarded to this site.

The complexity and volume of Puppet scripts to run as part of the framework is completely dependent on the amount of configuration to change from its default OS setting. For the case study organisation, a large amount of configuration is required for each instance to be fully configured. In total, 111 Puppet scripts were written for the case study organisations configuration management code repository. From the gathering of requirements to development and testing, these scripts took the

researcher approximately 4 working weeks to complete, after which, the configuration of environments was no longer manual.

At this stage in the development phase, a simple command line program was used to execute Terraform scripts to provision instances, and programmatically add the new IaC scripts to the Subversion VCS. Following this, system administration tasks such as creating internal and external DNS entries and adding the new instance to the organisations domain were performed manually. The instances were then configured with the Puppet scripts using a separate command line program which made multiple calls to the Puppet command line interface in order to run the large amount of Puppet scripts which configured the instance. Finally, the TeamCity builds that compile and deploy the organisations application and database code were manually kicked off. What was missing at this stage was a general purpose automation tool capable of performing the system administration tasks alongside combining each of the above steps to execute in order with only a single user interaction. Windows PowerShell was chosen as the automation tool to achieve this.

3.3.6 General Purpose Automation – PowerShell

Windows PowerShell is Windows native automation framework, it is an extension of the built-in command line tool (*cmd.exe*) that comes with every version of Windows (Stanek, 2014). It was chosen as the general purpose automation tool for the framework for a variety of reasons, mainly due to the fact that it is a Window's native tool designed for the automation of system administration tasks through a command line interface and the case study organisation is running a purely Windows-based infrastructure. It also required no installation on the case study organisation's machines as these machines are running the Windows Server 2012 R2 operating system, this operating system comes with PowerShell v4.0 built-in as standard (Ring, 2013). Upon further assessment, it was found that PowerShell has the capabilities of automating all of the manual system administration steps outlined in the previous section, as it allows for the programmatic modification of Active Directory objects such as machine and internal DNS entries (Talaat, 2013) (Microsoft, 2017). It is also supported as a standard build runner in TeamCity, meaning that the TeamCity build server integrates with PowerShell, giving it the ability to define build configurations that execute PowerShell scripts (Alexandrova, 2015).

In the framework, PowerShell scripts perform the following four functions:

1. *Automates the addition of the new AWS instance to the organisation's domain.*
2. *Automates the addition and modification of internal Active Directory DNS entries.*
3. *Executes the Terraform, SVN and Puppet command line tools in the appropriate order and validates the execution output appropriately.*
4. *Programmatically starts the deployment of the organisation's Application and Database code.*

While TeamCity is capable of simple execution of the Terraform, SVN and Puppet command line tools, PowerShell is needed for debugging and error handling purposes. For instance, the Terraform script that creates the instance in the cloud can be executed by TeamCity, but no error handling or output validation is performed on the running of this script. The TeamCity builds that deploy the organisations Application and Database code already existed prior to this research, PowerShell scripts were written to create simple web queries to call the function in TeamCity that executes these pre-existing deployment builds.

3.3.7 Summary

The above sections describe the technologies used in this implementation of the framework and also justify the use of each specific technologies. Figure 11 provides a visual guide as to how these technologies are used in the framework, while Table 6 summarises these technologies and the specific versions used in this implementation of the framework.

3.4 Framework Use Case

This section provides a use case for the framework to demonstrate how each process identified in the Exploratory Phase was automated. The individual steps in the process can be described under the following headings:

1. *User Interaction*

2. *Provisioning Build*
3. *Domain Build*
4. *Configuration Build*
5. *Deployment Build*

3.4.1 User Interaction

The process begins with the initial user interaction, this interaction consists of the user navigating to the TeamCity build server's web based front end and executing the Provisioning Build, which is the first build in the chain. In order to execute this build, the user must provide it with a set of parameters that will be passed throughout the chain and determine several factors about the environment to build. These parameters and their role in the framework are as follows:

1. *Amazon Machine Image ID*

As discussed in section 2.2.1 of this thesis, Amazon Machine Images (AMI) are Amazon specific machine image templates. Amazon Machine Images consist of pre-configured operating systems that new instances are created from (Amazon, 2015). As a parameter in the framework, the AMI ID links to the unique identifier of the AMI to base the new environment on.

2. *Environment Name*

The name of the environment to create will determine the environment's name in AWS in order to differentiate it from other environments. It will also determine what internal DNS names will be created or modified. For instance, if the *test.qa* environment name is chosen then the Terraform script that is generated for this environment will create an instance with an environment name tag populated with *test.qa* in AWS, similarly, the CNames added to the internal DNS will contain a reference called *test.qa* which will link to the instance name and private IP address parameters.

3. *Instance Name*

The name of new EC2 instance in AWS to create. This is used as the computer name on the machine itself and in Active Directory.

4. *Instance Type*

Amazon has a range of predefined machine specifications that instances can be based on (Amazon, 2015). This parameter determines the compute power of the instance to build.

5. *Organisational Unit Path*

Active Directory categorises domain resources such as machines into Organisational Unit (OU) paths, adding a machine to an Organisational Unit path will enforce all Group Policy configuration for that OU path on the machine (Microsoft, 2017). This parameter determines the OU path the new environment will be added to when it is added to the organisation's Active Directory domain.

6. *IP Address*

The IP address that will be assigned to the new instance.

7. *Security Group*

In AWS, security groups control traffic to and from instances in the cloud, they act as virtual firewalls for each instance (Amazon, 2017). This parameter determines what security group to add the new instance to.

8. *Subnet ID*

The unique identifier of the AWS subnet to add the new instance to.

9. *API Keys*

In order for the framework to connect to a specific AWS account, API keys must be provided to the framework.

10. *Region*

The region parameter determines the AWS region the new EC2 instance will be created in.

11. *Cost Code*

A custom tag for the new EC2 instance which allows for chargeback of AWS resources to a certain project or department, this is an optional parameter as it is not required to create a new instance.

3.4.2 Provisioning Build (PB)

The Provisioning Build is initiated by the build server on receipt of the user request containing the above parameters. Firstly, the Provisioning Build executes a PowerShell script which spawns a new Terraform script based off a pre-defined template, the values in the template Terraform script are transformed with the values the user provides to the build chain, this operation is portrayed with dummy values in Figure 14.

This new Terraform script is saved to the local disk and is then executed via a call to the Terraform command line interface. Upon execution, the script interacts with the AWS API in order to create a new instance based on the contents of the Terraform script, this interaction contains several calls to the AWS API, the complexity of these calls are abstracted by Terraform, which reveals only self-describing IaC files to the end user.

Following the creation of the new instance in the cloud, the PowerShell script then interacts with the Version Control System API in order to save the new IaC script to the VCS.

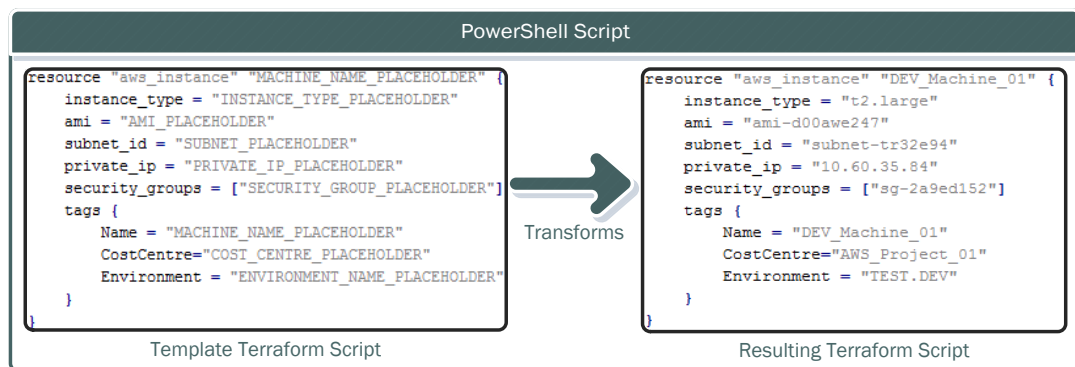


Figure 14: Terraform Script Transformation

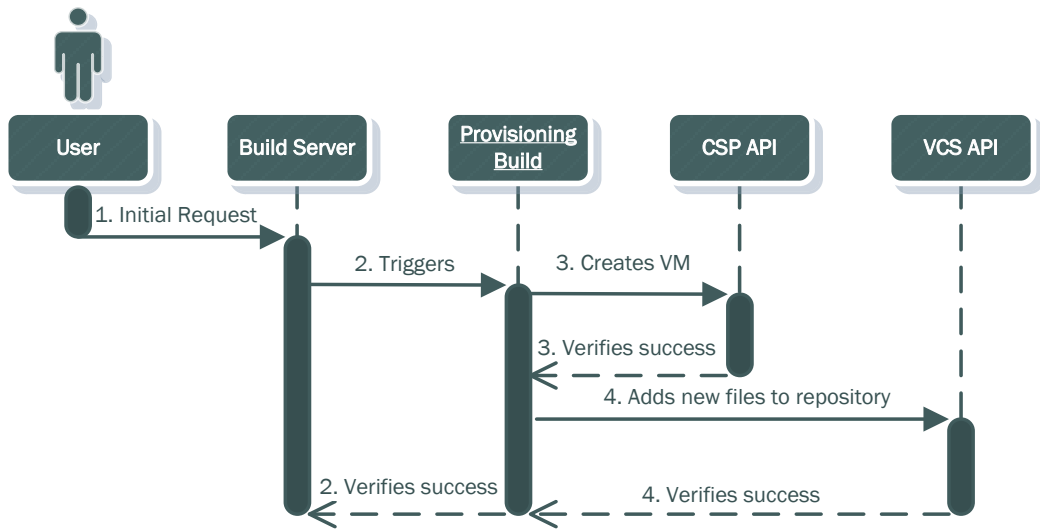


Figure 15: Provisioning Build Sequence Diagram

This script represents the executable documentation of the instance that Terraform just built in AWS, it contains all the information pertaining to the instance and allows for changes to the instance to be made through source controlled code, see Figure 14 for a reference to this final Terraform script. Storing this information in the VCS also caters for disaster recovery scenarios; given a situation where the instance has become corrupt and needs to be destroyed and recreated in the exact same way it was originally created then it can be done so via the Terraform script that resides in the VCS. The operations carried out in the Provisioning Build are visualised in sequence diagram form in Figure 15 for clarity, once these operations complete, and no errors have been thrown throughout the build, the build server triggers the next build in the chain.

3.4.3 Domain Build

At the time the Domain Build is triggered, a new instance exists in AWS but is not linked to the organisations network of computers in any way. The Active Directory manipulation capabilities of PowerShell are utilised in this build in order link the new instance to the organisation's network.

Based on the Organisational Unit Path parameter supplied by the user, a PowerShell script queries the Active Directory API in order to add the newly created AWS

instance to a specific OU within the organisation's domain. This effectively adds the new instance to the organisation's network of computers.

Once this operation completes, a PowerShell script queries the Active Directory API once again, this time it performs a search for DNS entries pertaining to the Environment Name the user provided as a parameter. There are two separate cases in this situation:

1. Creation case

If no records are found, the script will create them using a PowerShell cmdlet from the Active Directory library, the values of these entries are based on the IP Address, machine name and environment name parameters the user supplied to the build chain.

2. Modification case

If such records are found, the script changes them from their old IP address to the IP address of the new instance, this new IP address is based on the value specified by the user. This operation changes any internal DNS references to an existing environment to the new environment in the cloud. In the case where an environment does exist, this step is used to redirect all traffic to that environment to the new cloud-based instance, which is essential when rebuilding old, in-house environments and reusing these networking components. This operation is portrayed in Figure 16 for clarity, it shows the redirection of traffic based on Active Directory DNS modifications. The user attempting to access the environment before the Domain Build is run will have their request forwarded to the old in-house environment, and after the Domain Build is run, the user will have their requests forwarded to the new cloud-based environment.

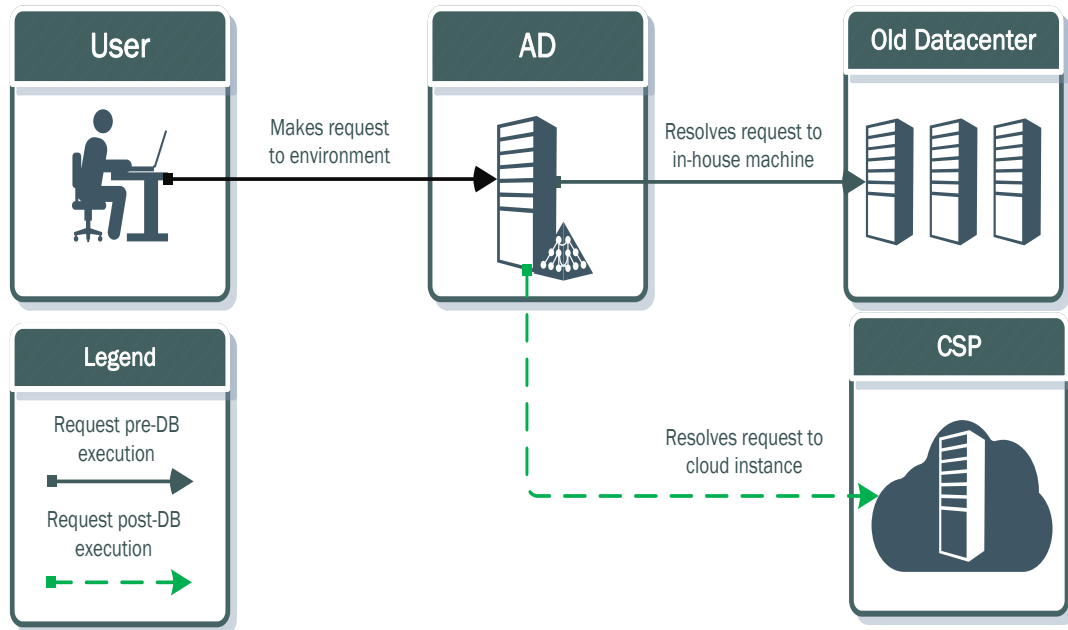


Figure 16: Redirection of Traffic via DNS

At this point, the new cloud-based instance has all the necessary networking components in place for it to be interacted with like any other test environment in the organisations fleet of environments. The sequence of events that occur in Domain Build are outlined in Figure 17 and Figure 18. Two diagrams were created as there are two separate scenarios which can occur depending on the presence of existing DNS entries for the environment in question. Figure 17 outlines the events that occur in the creation case outlined above; whereas, Figure 18 outlines the events that occur in the modification case outlined above. Once these operations are complete, the next build in the chain is triggered.

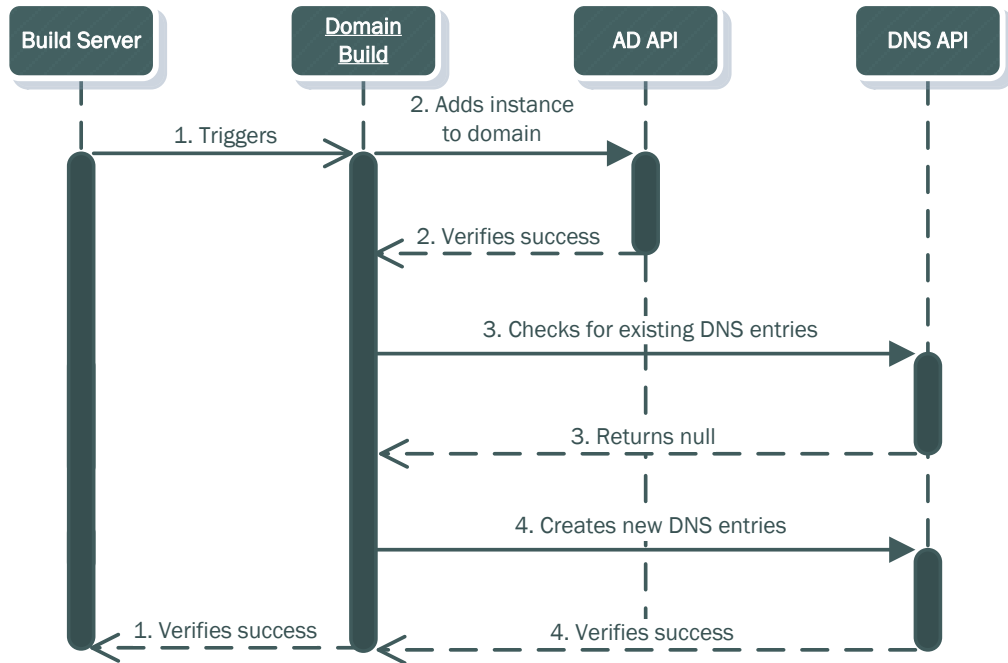


Figure 17: Domain Build Sequence Diagram Part 1

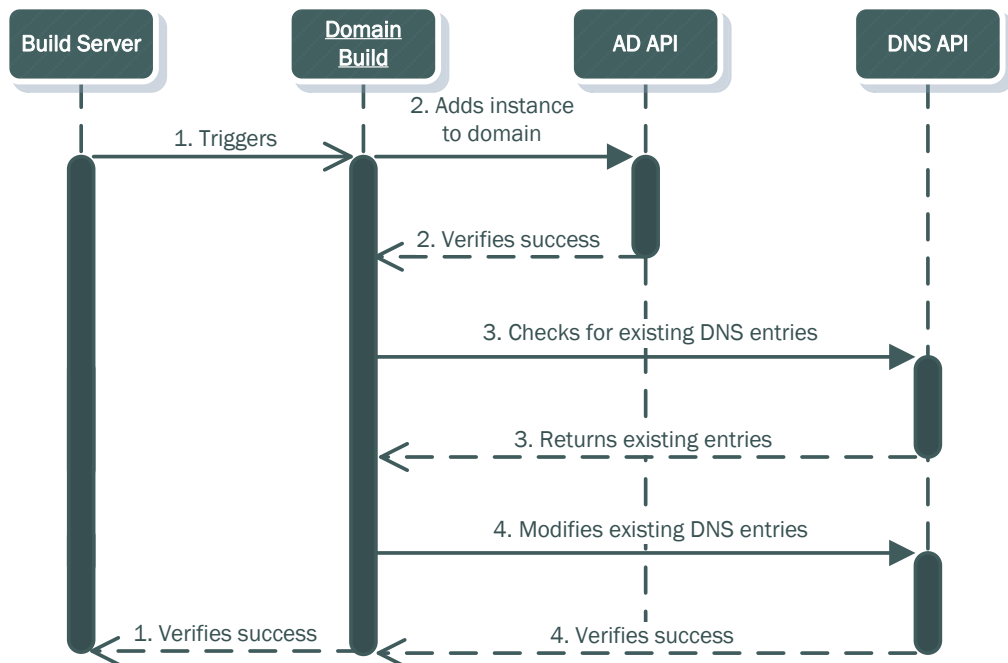


Figure 18: Domain Build Sequence Diagram Part 2

3.4.4 Configuration Build

Only after the new machine can be accessed via a UNC path from a remote machine in the same domain can the Configuration Build be executed. This build performs the following actions:

1. Copies all files required for Puppet to run on the new instance from the TeamCity server.
2. Executes the Puppet installer on the new instance.
3. Executes the Puppet configuration management code on the new instance.

The end result of the above three operations is a machine that has been configured to behave as though it is a functioning test environment. All applications required for the organisations code base to function on the instance have been installed, all environment specific configuration has been performed and any required files and directories have been created and put in place by the Puppet scripts. The actions carried out by the Configuration Build have been outlined in Figure 19:

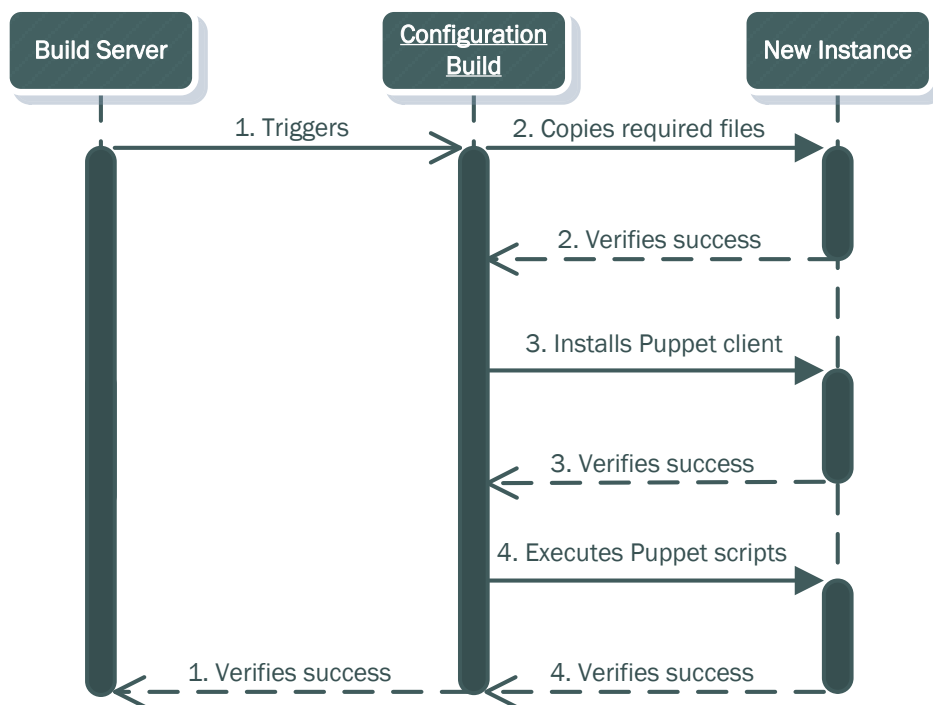


Figure 19: Configuration Build Sequence Diagram

Once all of these operations have been confirmed as having run successfully, the final build in the chain is triggered.

3.4.5 Deployment Build

The final build in the chain is the Deployment Build, this build utilises PowerShell automation in the form of simple web queries to programmatically trigger existing TeamCity build configurations that compile and deploy the organisation's Application and Database source code.

The Deployment Build is dependent on all other builds in the chain to have run before it, as it utilises internal DNS entries to locate the environment to deploy the code to. These internal DNS entries were either created or modified to point to the new instance in the cloud during the Domain Build. It is also dependant on the Configuration Build being run before it for a variety of reasons. The simplest to explain here is the directory structure and access control lists that the Configuration Build sets up, the Deployment Build writes the compiled source code to these directories on the target machine, therefore it is dependent on them being present with the necessary security assigned to them before being able to function correctly.

The Deployment Build is configured to iterate through a list of build configuration identifiers in order to execute the deployment of the organisations application and database code. For clarity, a simple version of this PowerShell function follows in Figure 20, this function uses a pre-authenticated WebClient object to execute a build configuration based on its build configuration identifier, a large list of these build configuration identifiers are hardcoded into the Deployment Build, it loops through each of these identifiers and runs this function against each one in order to execute it.

Once the deployments are triggered, another PowerShell script monitors the success of these deployments to the new environment. Once all deployments have been verified as having finished successfully, the cloud-based environment is in the exact same state as any other new, manually built, in-house environment. Provided the external DNS entries have been setup manually, following the success of the Deployment Build, the new environment can be handed over to a development department for use.

```

Function Execute-TeamCityBuild
{
    param([System.Net.WebClient]$webclient,
          [string]           $teamCityServer,
          [string]           $buildId,
          [string]           $environmentName)

    <#
        .SYNOPSIS
            Kicks off a build to a named environment
    #>

    # This string is the base TeamCity URL appended with the action to execute a build
    $teamCityBaseUrl = "http://$teamCityServer/httpAuth/action.html?add2Queue="

    #This string constructs the ID of the build to execute and environment to deploy to
    $queryString = "$buildId&name=targetEnvironment&value=$environmentName"

    # This string combines the above into a single query
    $finalQuery = "$teamCityBaseUrl$queryString"

    # Finally, the PowerShell webclient is used to query to the full URL
    # This query will execute the build
    $webclient.DownloadString($finalQuery)
}

```

Figure 20: Sample PowerShell Function

Chapter 4. Results

This chapter presents the results of the industry-based case study and experiments pertaining to the execution of the automated framework alongside the results of the survey questionnaire. The overall goal of the data collection, analysis and presentation is to provide the reader with a concise set of structured quantitative comparative metrics in order to determine if the adoption of automated public cloud and IaC technologies can provide greater efficiency to SMEs currently manually managing a traditional in-house, or colocation-based data centre. The findings outlined in this chapter make evident the efficiency benefits of employing public cloud and IaC technologies in the environment creation process.

4.1 Creation/Recreation Experiment Context

The experiment methodology is defined by Amaral et al. as one where a system is evaluated under the scrutiny of a researcher in order to answer a specific research question or achieve a specific research objective (Amaral, 2011). This experiment pertains to the controlled recreation of existing, and creation of new IT testing environments on AWS's IaaS platform via the automated framework. This experiment will be referred to as the Creation/Recreation experiment.

4.1.1 Creation/Recreation Experiment Aims

The aims of the Creation/Recreation experiment conducted as part of this thesis can be compartmentalised into two, high-level categories. The first is to test the functionality of the automated framework in order to assess whether or not it satisfies the functional and non-functional requirements outlined in sections 3.1.3 and 3.1.4 of this thesis. Inclusive of this test is the confirmation that framework has the capability to reliably rebuild existing, and create new environments on the public cloud, configure them according to specification and deploy the organisation's codebase to them in order to provide working environments for the target organisation in as much of an automated fashion as possible. The rate of error, manual troubleshooting time and time spent on each build in the overall process are encompassed in this test. The

second category in this experiment is the efficiency comparison between the case study organisation's previous co-location based environment creation process with the new environment creation process handled by the framework. This comparison is based on effort and timing metrics outputted by the build server when executing the framework in order to recreate existing, and create new environments on the public cloud. Data pertaining to the timing and effort metrics from the organisation's previous environment creation process was obtained through semi-structured interviews in the case study portion of this research, outlined in section 3.1.

4.1.2 Network Architecture

The organisation the framework was implemented in utilised a co-location infrastructure to host their testing environment before the framework rebuilt them in AWS's IaaS platform.

In the co-location infrastructure paradigm, the client purchases all of the infrastructure hardware, typically racks of physical servers and storage arrays, while the co-location service provider rents space in a secured networking facility for the client-owned hardware to operate in, along with the bandwidth to and from the data centre (Reichard, 1998). In most cases, the physical security, housing, powering and cooling of the physical machines are the responsibility of the co-location service provider, it is the client's responsibility to manage the software on the physical machines and the to ensure that the network they create in the data centre is secured by appropriate networking devices such as firewalls and switches. Once the physical devices are installed and configured, an internal network comprised of these devices is created within the co-location provider data centre, this network may then be connected to the client's existing network. One method of connecting an existing client network with the co-location data centre network is by creating a Virtual Private Network (VPN) that act acts a secure tunnel for traffic to be transmitted between two separate networks, devices called VPN gateways are deployed to both the client network and the co-location network in order for traffic to be sent to and received from the VPN (Gottlieb, 2012).

In this experiment, the organisation was leasing server space in a co-location data centre located in Sterling, Virginia in the United States, the architecture of this co-

location infrastructure was relatively simple, a VPN allowed users and servers from the on-site office to connect the off-site co-location data centre, this VPN utilised a 100Mbps link. A physical VPN gateway device on the on-site office side of the VPN processed requests to and from the co-location centre across the VPN. On the co-location centre side of the VPN, a virtual router acted as the gateway to the network. A virtualised WAN firewall filtered traffic from the VPN gateway device which was then sent to a WAN switch. The WAN switch then sent this filtered traffic to a LAN firewall before it entered the internal co-location LAN the test environment servers resided on. The structure of the internal co-location LAN was divided up into two VLANs, each with their own subnet:

1. *Development (Dev) VLAN for environments dedicated to development work.*
2. *Quality Assurance (QA) VLAN for environments dedicated to internal testing work.*

For the sake of clarity, this architecture has been plotted out in Figure 21.

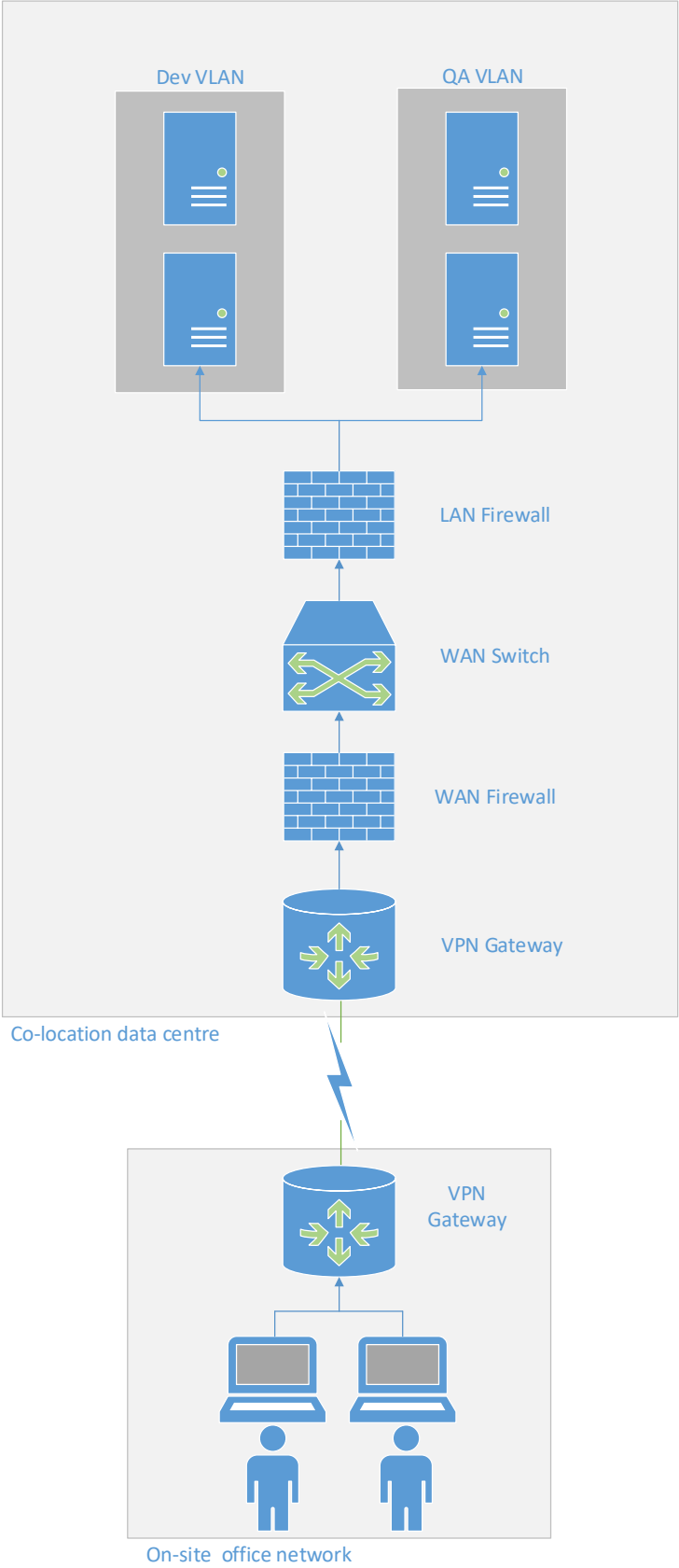


Figure 21: On-site office to co-location datacentre diagram

As discussed in section 3.2.4, in order for the framework to function, certain networking components were required to be built in AWS. The structure of this network is similar to that of its co-location counterpart. However, the naming of certain components are specific to AWS, these components are covered briefly in the section that follows.

AWS's virtual network offering is called a Virtual Private Cloud (VPC), VPCs have previously been briefly covered in this thesis in section 2.2.1. The framework creates instances within a pre-defined subnet, in order for subnets to exist in AWS, a VPC must be present to house them. Similar to connectivity between an on-site network to a co-location centre network, a VPN is required to access instances that reside in a VPC as, by default, instances created within a VPC are not accessible directly from a private network, a VPN must be setup for this to be achieved (Amazon, 2017). AWS's Customer Gateway and Virtual Private Gateway components are required for a VPN connection to be made to a specific VPC (Amazon, 2017). A Customer Gateway is a hardware or software-based device that manages traffic to and from the client's private network and the specific AWS VPC they are connecting to, whereas a Virtual Private Gateway is virtualised VPN connector on AWS's side of the VPN (Amazon, 2017). The filtration of traffic within the VPC is done with AWS Security Groups, which act as virtual firewalls (Amazon, 2017).

Users residing in the on-site office network connect to the cloud-based environments through the VPN, the speed of this link matches the on-site office to the co-location data centre VPN link at 100Mbps. On the on-site office side of the VPN, a software-based AWS Customer Gateway is used to process inbound and outbound traffic to and from the VPC. On the AWS side of the VPN, a Virtual Private Gateway device connects the VPC back to the on-site office network. Traffic within the VPC is routed by a virtual router and filtered by AWS Security Groups before reaching the destination test environment server. Similar to the layout of the VLANs in the co-location datacentre network, environments are logically divided from each other via private subnets according to their type:

1. *Development (Dev) private subnet for environments dedicated to development work.*
2. *Quality Assurance (QA) private subnet for environments dedicated to internal testing work.*

The architecture described has been plotted out in Figure 22.

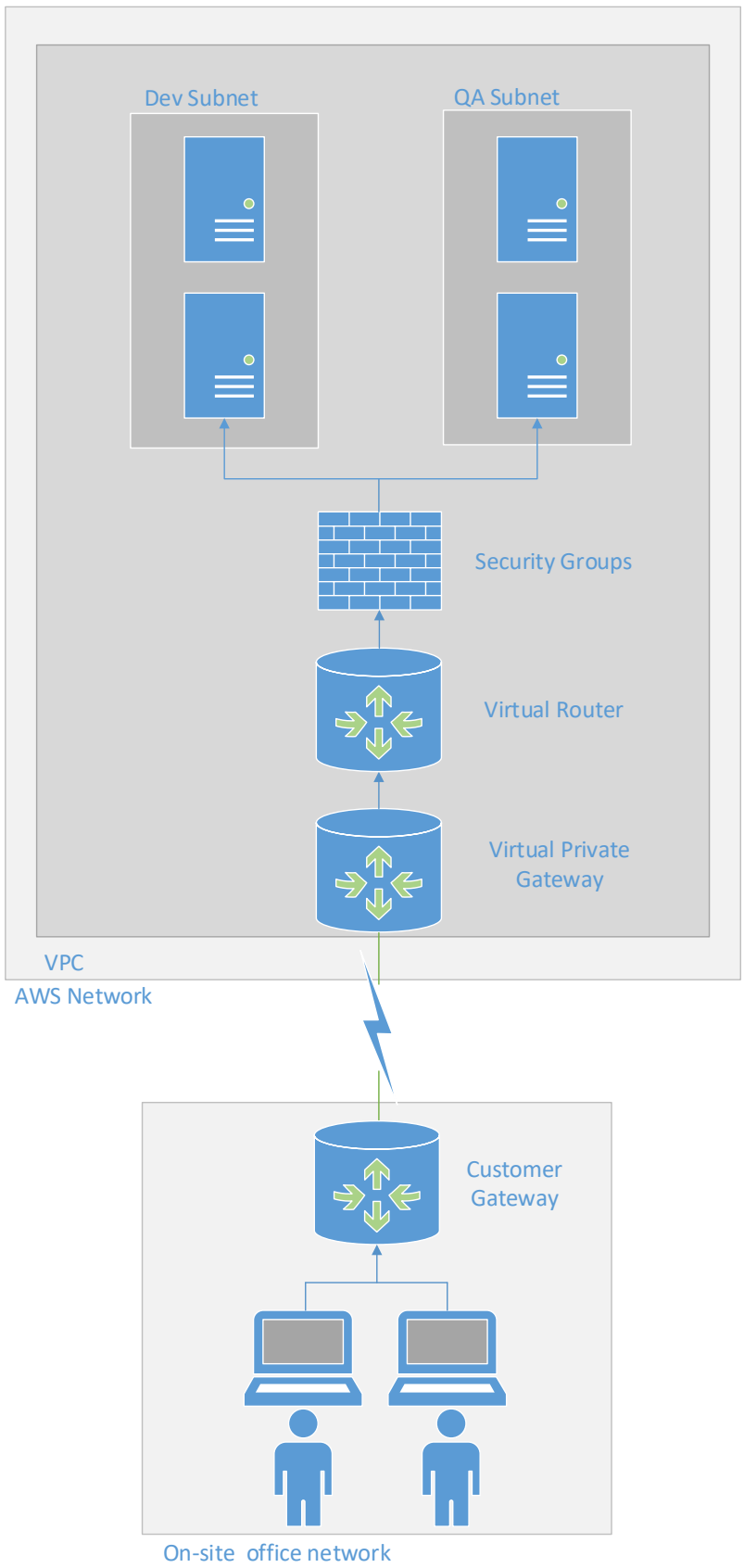


Figure 22: On-site office to AWS diagram

4.1.3 Environment Specification

This section discusses the specifications of the non-cloud environments that were recreated in AWS alongside the cloud-based instances that the framework built. The case study organisation's testing environments consist of a single VM each. This single VM runs Windows Server 2012 R2, it acts as a website, services, file share and database server for the organisations codebase to be developed and tested on. In terms of machine specification, it was desired by the case study organisation to recreate their existing testing environments utilising servers with specifications as similar to their existing equivalents as possible. The cost comparison and instance comparison outlined in sections 2.2.5 and 2.2.7 of this thesis show the comparison of different CSPs offerings of instance types as similar to the case study organisation's servers which have 8.00GB of RAM installed alongside two processors. It was aimed to match this type of compute power as closely as possible with the cloud-based environments by choosing AWS's t2.large instance type which has 8.00GB of RAM installed alongside two processors (Amazon, 2017).

System Summary	Co-location environments	AWS environments
OS Name	Microsoft Windows Server 2012 R2 Standard	Microsoft Windows Server 2012 R2 Standard
Version	6.3.9600 Build 9600	6.3.9600 Build 9600
System Manufacturer	VMWare, Inc.	Xen
System Model	VMWare Virtual Platform	HVM domU
System Type	x64-based PC	x64-based PC
Processor	Intel(R) Xeon(R) CPU E7540 @ 2.00GHz	Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
Processor Count	2	2
BIOS Version	Phoenix Technologies LTD 6.00	Xen 4.2.amazon
Installed Physical Memory (RAM)	8.00GB	8.00GB

Table 7: System comparison of non-cloud and cloud environments

The m4.large instance type has similar specifications to the co-location based environment servers, however, this instance type was not chosen as the t2.large instance type is far more cost effective and is recommended by AWS to be more suitable for testing environments that may not be under heavy load for a prolonged period of time (Amazon, 2017). A basic system information comparison between the existing co-location based environments and their cloud-based equivalents is displayed in Table 7. It shows that the cloud-based t2.large environments have a different processor with a slightly faster processor clock speed than the co-location based equivalents. While the compute power of the servers are similar, they are not identical, this is a recognised limitation as the three CSPs considered for this research offer only preconfigured server specifications and do not allow for custom specifications to be made by the user (Amazon, 2015) (Microsoft, 2015) (Google, 2015). The t2.large instance type chosen was the closest specification instance type available on AWS at the time the project was carried out.

In terms of storage, the environments being compared are almost identical, as AWS allows for the creation of custom AMIs which can have EBS volumes of user-defined sizes attached to them, these EBS volumes act as hard disk drives for all instances that are created from that AMI (Amazon, 2015). The only difference between the two is the storage manufacturer due to the change in hosting platform. A summary of the storage comparison is presented in Table 8.

Storage Summary	Co-location environments	AWS Environments
Name	VMWare Virtual Disk SCSI Disk Device	AWS PVDISK SCSI Disk Device
Media Type	Hard disk drive	Hard disk drive
Drive Count	4	4
Total Capacity (GB)	250	250

Table 8: Storage comparison of non-cloud and cloud environments

Platform Type	Name	Version
Database Management System	SQL Server 2014	12.0.5000
Web Server Software	Internet Information Services	8.5.9600.16384
Service Bus System	NServiceBus	6

Table 9: Basic test environment software systems

Due to the multi-functioning single server paradigm utilised by the case study organisation to host their testing environments, the platforms outlined in Table 9 are required to be installed and configured on the server in order for the organisation's websites, services and databases to run on.

Along with the above several default configurations must be modified to allow for the single server to act as a website host, file share, service handler and database server. These modifications are numerous and it is outside of the scope of this thesis to list them all, without digressing too far, some examples of some of these modifications follow.

- *The creation and configuration of Internet Information Services (IIS) websites and application pools.*
- *The creation of local users and groups.*
- *The enabling and disabling of Windows features.*
- *The enabling of specific SQL Server configurations (SSRS, SSIS, SSMS, etc.).*
- *The creation and configuration of file shares and access control lists on directories required by the organisation's websites and services.*
- *The installation of custom applications to support certain functionality of the websites and services (payment processing, document generation, geolocation, etc.)*

This single server environment architecture also requires all of the organisations codebase to be deployed and correctly installed on single server. At the time of writing, each of the organisation's test environments contain over 55GB of organisation specific website, service and database code from 174 individual codebases, the breakdown of this codebase is shown in the Table 10.

Software Type	Count	Combined Size (GB)
Websites	35	1.56
Services	79	1.63
Databases	60	52
Total	174	55.19

Table 10: Organisation software on test environments

4.1.4 Process Variables

In the context of experimental design, Antony defines a process as a transformation of inputs into outputs, process variables are the inputs to a process that has an effect on its output (Antony, 2003). The two different types of process variables are controlled and uncontrolled, both of which are present in this experiment, they are discussed in this section (Antony, 2003).

4.1.4.1 Controlled Variables

The majority of variables in this experiment are controlled, as the nature of automated computer processes allow for the exact same set of variables to be inputted into a process in order to obtain the same output. These controlled variables are listed below:

1. *The AMI to provision*

The AMI used throughout the course of this experiment did not change, it was an OVF export of a base Windows Server 2012 R2 machine from the co-location based infrastructure that was imported into AWS as an AMI. Therefore, an identical base instance was used in both the previous manual environment creation process and the framework environment creation process in this experiment.

2. *The build server*

The TeamCity build server that executes the framework scripts did not change throughout the course of this experiment and remained a static entity in regards to compute power and installed components.

3. *The IaC scripts*

The template IaC and associated wrapper scripts that provision new environments in AWS remained static from their first stable release and did not change throughout the course of this experiment.

4. *The applications residing on the environments*

No additional applications required to be installed or configured on the testing environments throughout the course of this experiment.

5. *The configuration management scripts*

As no additional applications or configurations required to be installed on test environments while the experiment was taking place, the configuration management scripts that configure the new environment in AWS remained the same throughout the course of this experiment.

6. *Instance type*

The t2.large instance type being provisioned did not change throughout the course of this experiment. As mentioned in section 4.1.3, the compute power of this instance type is not identical to the colocation based environment machines but it was chosen for use in this experiment as it matches the colocation based environments as closely as possible.

7. *Storage allocated to the instance*

The storage allocated to all instances being provisioned remained static throughout the experiment. As mentioned in section 4.1.3, the capacity of the drives attached to both the AWS based instances and the colocation based environments are identical, only the storage manufacturer changes due to the change in hosting platform.

4.1.4.2 Uncontrolled Variables

While the controlled variables are numerous due to the static nature of automation, there were uncontrolled variables in this experiment of varying severity. These variables are a symptom of carrying out industry-based research, as opposed to carrying out experiments in a laboratory scenario where the context has been devised specifically to execute the experiments. As such, variables such as those that follow

have been cited as an inevitable consequence of performing industry-based research (Costely & Armsby, 2007).

1. *Network latency across the VPN*

All users, including developers and testers, along with services and servers residing in the on-site office network shared the same VPN link to the AWS's based environments. While this link was more than adequate to maintain the functioning of business processes such as regular development and quality assurance testing on AWS environments, it did pose a significant risk to the functioning of the framework and the validity of the timing data obtained from it. This variable was remediated as much as possible by performing framework executions out of the regular working hours of developers and testers. Framework executions took place one at a time at night or on weekends. This remediation also satisfied a request from operations engineers from within the case study organisation, who did not want to risk network traffic saturation of the VPN link during business hours with the combined resource intensive execution of the framework and the regular development and testing processes.

2. *Shared tenancy of the TeamCity build server*

The TeamCity build server hosted deployment processes for all technical staff in the case study organisation, execution of the framework meant that a significant amount of resources from the TeamCity build server were dedicated to the framework and could not be used by others until it was released. Conversely, execution of other resource intensive processes from the TeamCity build server such as deployment of code to existing environments by developers or testers could have a negative effect on the performance of the framework itself. Similar to the previous uncontrolled variable described above, this variable was remediated as much possible by only executing the framework outside of regular working hours to ensure that the case study organisation's technical staff had the adequate resources available to them during their working hours and to ensure consistency in the execution of the framework and resulting data obtained from it.

3. *Changes to organisational code and deployment process*

This variable encompasses the changes the case study organisation development staff made to the organisation's code base which was deployed via the framework.

This variable was unavoidable and uncontrollable in the experiment, as the company the framework was implemented in is a software solutions provider with several active development teams. The changes to the deployed code base varies from release cycle to release cycle, over time, different functionalities, bug fixes and code optimisations were added by development staff on the request of the organisation's clients and project management department. Attempting to halt all development work across the 174 code bases that comprise the organisations websites, services and databases was beyond the remit of the researcher. A potential remediation to this uncontrolled variable is to take the revision numbers of each code base from the beginning of the experiment and only deploy every code base from that revision. This way, uniformity throughout this specific step in the experiment could be completely controlled. This method may work in a devised, laboratory setting, however, consider that one of the key claims made in this thesis is that the framework has the capabilities of creating working environments through code for an organisation in the software development industry, the validation of the framework's requirements lies in its ability to successfully create environments that can be used with as little time and manual effort overheads incurred as possible. If outdated code was deployed through the framework, the target environment would not be in a working state for developers or testers to work on. Therefore, if this variable was controlled by the method stated above then all results from the framework would be completely void and the inclusion of these results would be disingenuous as the development or testing team the environment was handed to after the framework execution would need to deploy updated versions of the codebase to the environment before they would be able to use it. This variable should be recognised as a limitation of this research when examining the results of this experiment.

4.1.5 Creation/Recreation Experiment Scope and Conduction

This section details the extent and conduct of the Creation/Recreation experiment in regards to the existing environments that were rebuilt and the new environments that were created on AWS's IaaS platform via the framework.

As previously mentioned in the Project Planning section of this thesis, eight in-house environments were scheduled for recreation in AWS's IaaS platform in this

experiment. Along with the initial eight environments for recreation, 35 new test environments were created native to the public cloud. All environments designated to reside in AWS were placed on a rigid migration schedule by the organisation's project management office, ranging from the 9th of September 2016 to the 30th of January 2017. As the process of rebuilding each environment incurred downtime of that environment for the duration of the framework execution, this schedule needed to be signed off by the managers of individual development departments. Once the scope of the migration project was agreed and the timelines were put in place, the researcher assumed full control over the environment and the experiment began.

The researcher firstly allocated a timeslot for the framework to execute, this was either during the evening if on a working weekday, or any time during a weekend or bank holiday. At the beginning of this timeslot, the researcher gathered the necessary values for the specific environment to be recreated and inputted these into the framework at the User Interaction stage, detailed in 3.4.1. The framework was left to execute and checked by the researcher at regular intervals for errors. If a terminating error was thrown and the build chain stopped, the researcher attempted to remediate the issue, if remediation was possible, the build chain was resumed immediately from its point of failure. In the cases where the researcher was not able to solve the issue, then technical support staff from within the organisation investigated on the next business day. Once the issue was resolved, the build chain was resumed the following evening. Following the success of the framework, the researcher gathered the raw timing data from the build server UI and, where applicable, retrieved timing estimates from supporting technical staff who dealt with error remediation. These timings and effort metrics were inputted into a spreadsheet. This process was repeated for each environment recreated by the environment and each environment created native to the public cloud.

4.2 Creation/Recreation Experiment Results

The results of the Creation/Recreation experiment are detailed in this section.

4.2.1 Data Analysis

This section describes the analysis and interpretation of the raw data obtained through the semi-structured interviews held in the case study portion of this body of work and the data obtained through execution of the automated framework in the context of the Creation/Recreation experiment.

4.2.1.1 Data Classification

There were two main data sets for analysis in the Creation/Recreation experiment, they are the Manual timings, and the Automated timings.

The Manual timings data set was derived from the semi-structured interviews held in the Exploratory Phase of the case study. The researcher analysed the transcripts of each interview and extracted the list of manual tasks the interviewees had performed during manual environment creation scenarios. These tasks were then mapped to their appropriate estimated timings provided by the interviewees. These tasks and their associated timings were then categorised into the appropriate groups for comparison with the automated timings data set. The transcripts for these interviews can be found in Appendix A, Appendix B and Appendix C of this thesis. All tasks in this set have an effort overhead of 100%, as each task is carried out manually and therefore requires the full attention of the person carrying it out, this is opposed to automated tasks where the user simply needs to issue a command to have the whole task executed programmatically.

The Automated timings data set is comprised of the raw data obtained from TeamCity execution logs of the framework, alongside estimates of any manual work that needed to be performed during each run of the framework. The majority of timing data that comprises this set was retrieved from the TeamCity front-end. TeamCity build chains allow for the individual execution details of each build in the chain to be accessed through a single web-based interface, these details include the ordering, status and execution times of each build in the chain (Alexandrova, 2016). By using the build chain interface, the researcher was able to create a central repository of execution timings. The build chain interface also exposes terminating errors that caused the build chain to fail, these were entered in along with the estimated troubleshooting

time the researcher or supporting technical staff spent to solve the errors and restart the build chain from its point of failure. The rate of failure of the framework should be of interest of any reader, as it shows how automated tasks can be more efficient in terms of time and effort, but also may be more unreliable than manually performing these tasks in practise. After each run of the framework, execution timings, rates of error and manual troubleshooting times for each build were extracted and placed in an Excel spreadsheet for later analysis. Along with the above, the researcher manually entered the category of the framework run into the spreadsheet. These categories are divided into two separate subsets, either Recreation or Creation, depending on the environment in question.

The Recreation timings were taken from execution runs of the framework where an existing, in-house machine is being recreated on public cloud infrastructure. The Creation timings were taken from execution runs of the framework where a new environment is created native to public cloud infrastructure. As previously mentioned in section 3.1.2, the creation of the external DNS entries required for each environment to function was not automatable due to the case study organisation's subscription to a DNS provider that did not provide an API for programmatic interaction. Therefore, the creation of these external DNS entries was performed manually for newly created environments that fall into this subset. This task is not applicable to environments in the Recreation subset as these entries were already in place for environments that had previously existed in-house. The timings for this task were derived from the manual estimates provided by staff in the semi-structured interviews and is the same as the timing for external DNS creation in the manual set.

The volume of data in the Automated set was far larger than that of the Manual set, but it was still manageable enough for the researcher to manually parse and enter this set into an Excel spreadsheet for analysis. The automated timings were gathered on a per-build basis, meaning that, each build in the chain provided its own raw timing data broken down into the following sections:

1. *Provisioning Build*
2. *Domain Build*
3. *Configuration Build*
4. *Deployment Build*

The Manual timings set had each individual task matched to an approximate time it took to complete with no form of categorisation of tasks. Whereas, the automated timing set was already categorised as outlined above. Therefore, the researcher classified the tasks in the manual timings set to match those from the automated timing set, factored in the troubleshooting work and external DNS entry creation and created the following data groups common to both for comparison:

1. *Provisioning Tasks*
2. *Domain Operations*
3. *Server Configuration*
4. *Deployment of Codebase*
5. *Troubleshooting*
6. *External DNS Creation*

4.2.1.2 Data Analysis

The timing data for both sets outlined above is relatively simple and does not contain excessive levels of complexity. The Manual set contains a single set of approximated timings for each task, these timings were retrieved from semi-structured interviews with staff belonging to the case study organisation who previous carried out these tasks on a regular basis. The Automated set contains a significantly larger volume of data as it was taken from repeated, real-world runs of the automated framework. Therefore, following classification, it was necessary to find the most appropriate calculation of the average timings for each data group in the automated data set.

The data for the automated timings did not contain a large amount of lower or higher extremes, due to the nature of automation itself, the execution times follow a regular pattern. Therefore, the researcher calculated the averages of the automated runtimes by calculating the mean average of each data group. There were some manual tasks that needed to be performed during the automated process for troubleshooting errors when the framework failed, to ensure uniformity in results comparison, the mean average was calculated for these timings to find the most appropriate midpoint of data. The above data groups from the Automated timings data set were compared to the

same data groups from the Manual timings data set. As the timings for each data group varied greatly, the researcher calculated the time in minutes per data group, as some tasks from the Automated data set take seconds, where other tasks take hours. Similarly, some tasks from the Manual data set have been approximated to take minutes to carry out, whereas other tasks from the same set have been cited to take several hours. This was necessary for comparative purposes as it is desirable to group all data together in a standard, uniform manner.

4.2.2 Sample Size

Varying sample sizes are present for each of the above sets, what follows is a discussion of the sample sizes from the Manual timings dataset and Automated timings dataset along with the Creation and Recreation subsets of the Automated timings dataset.

Data from the Manual timings dataset were extracted from transcripts of the semi-structured interviews with the case study organisation staff members, these interviews were held by the researcher in the Exploratory Phase of the case study. It can be said that the sample size for this set is limited to a single sample as the timings for a manual environment creation or migration are based on approximations provided by various staff members that were previously tasked with carrying out the manual steps in the environment creation process. Manual creations or migrations of environments theoretically could have taken place to provide a more ample sample size, but this was not possible to carry out due to resource constraints and the practical nature of performing industry-based research. In that, the researcher was not in a position to request that three technical engineers from within the organisation halt their work to manually create several test environments and detail the time and effort it took to perform each task to generate a larger sample size for this dataset. Due to this constraint, the sample size and quality of this sample is relatively poor when compared with the Automated dataset.

Data from the Automated dataset were obtained directly from the TeamCity build chain execution logs of the framework. These timings are far more precise and reliable than the manual timings set as these timings were captured automatically and in real-time by TeamCity. In total, 39 execution logs are included in the sample size for this

set. Similar to the limitation for the Manual timings set, the sample size for the Automated timings set was restricted by the case study organisations requirements. Each test environment that was generated via the automated framework was done so on-demand as a requirement for the case study organisation’s development or testing departments. Once the environment instance has been created at the beginning of the framework execution, it immediately incurs a direct cost overhead to the organisation. Therefore, the researcher could not execute the framework to generate test environments at will in order to increase the sample size for this set, as the cost effect of doing so would cause an unfeasible amount of strain on the resources that the organisation had dedicated to this project.

The Automated timings set is broken down into two subsets, Recreation and Creation. The Recreation subset consists of timing data retrieved from the execution of the framework where an existing, in-house machine is being recreated on public cloud infrastructure. A total of 8 execution logs are included in this subset. The Creation subset consists of timing data retrieved from the execution of the framework where a new environment is created native to public cloud infrastructure. A total of 31 execution logs are included in this subset.

4.2.3 Comparison of Manual and Automated Datasets

The Manual Timings and Automated Timings datasets are presented and compared in this section, by plotting the raw data from each dataset on the same charts, a clear comparison of timings from both processes can be seen. In this instance, the Automated Timings dataset is a general view of the dataset, it is comprised of the combined average means of the Creation and Recreation subsets.

4.2.3.1 Overall Process Comparison

The most important area of the results are presented here. The comparison of the overall environment creation process when performed manually and when performed through the automated framework is shown here. Figure 23 is by far the most simplistic, and possibly the most significant illustration of data in this entire document.

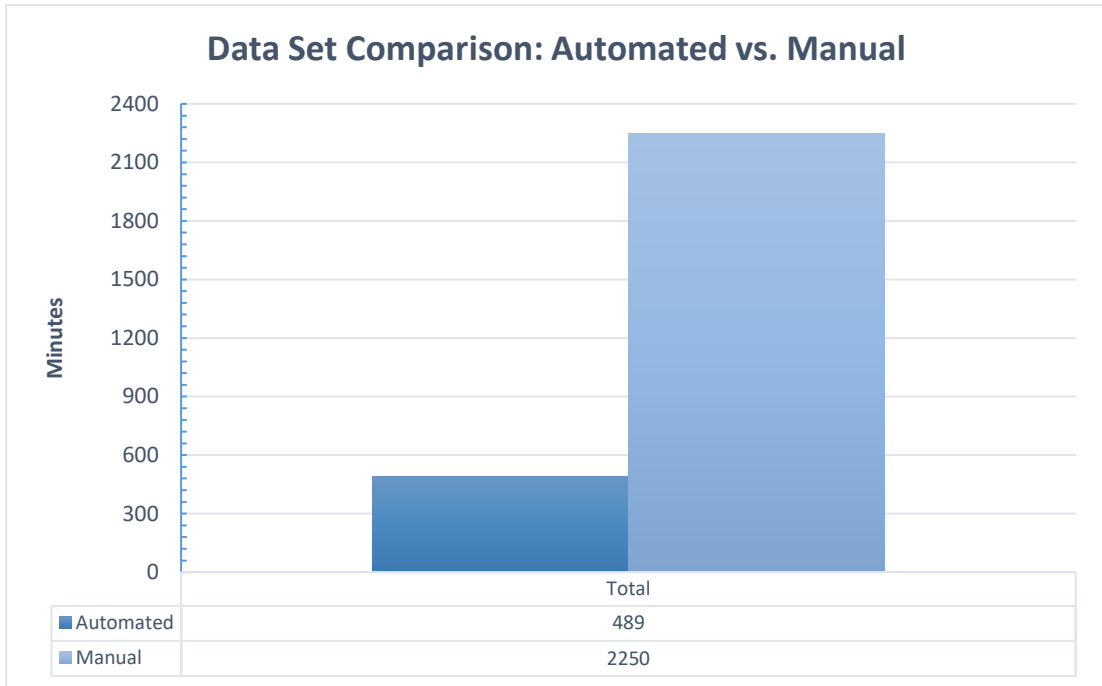


Figure 23: Data Set Comparison: Automated vs. Manual

It makes no distinction between tasks, rather, it combines all tasks in both processes to display an overall comparison of the automated and manual processes in terms of timings for both.

It can be seen in Figure 23 that the automated process as a whole is 360% faster than the manual process. The manual process itself takes 2,250 minutes or 37.5 hours in total, whereas, the automated process takes 489 minutes or 8.15 hours.

4.2.3.2 Individual Task Comparison

Figure 24 demonstrates a more detailed breakdown of this overall process comparison, it shows each task and their associated timings in a side-by-side comparison. Figure 24 provides a more detailed view of the efficacy of the automated framework than Figure 23. One can see from Figure 24 that, in the highest performing task the framework can handle the Provisioning Tasks 8,092% faster than the manual process, even in the lowest performing task, the framework can handle the Deployment of Code 140% faster than the manual process.

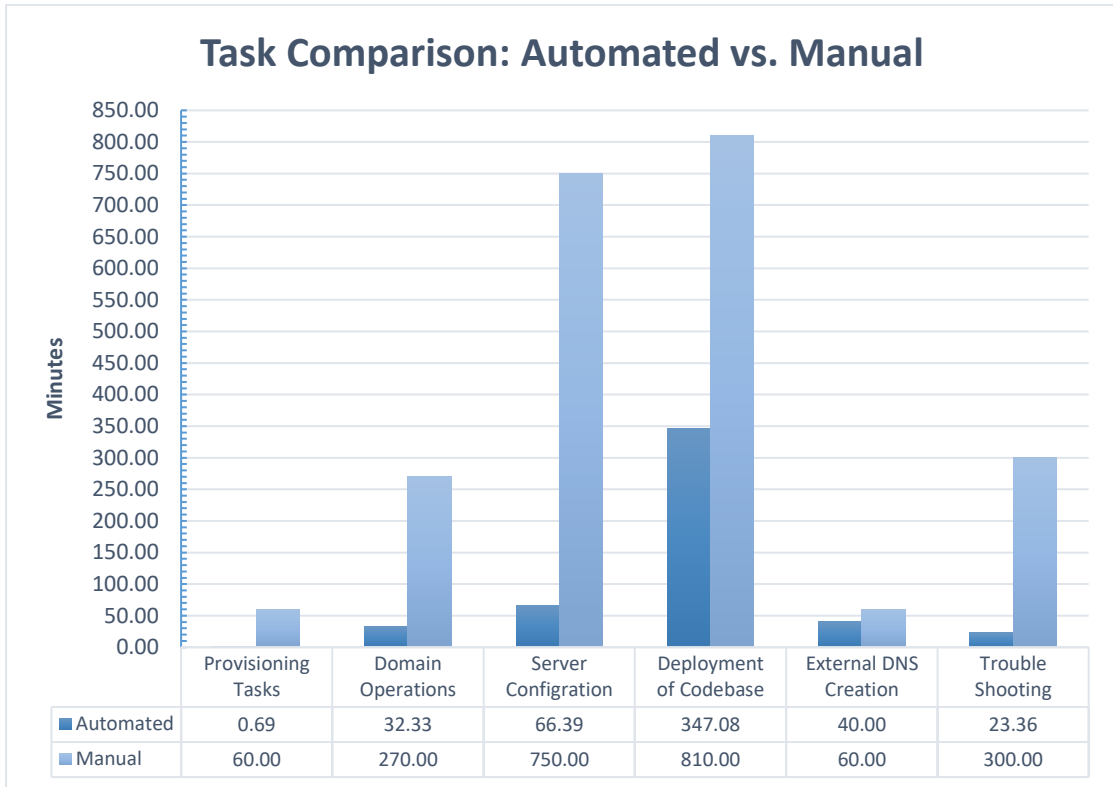


Figure 24: Task Comparison: Automated vs. Manual

4.2.3.3 Task Proportion Comparison

These data sets varied greatly not only in actual timings, but also in terms of proportion of time taken to complete various tasks in view of the overall process. Figure 25 shows the proportion of time each task took to complete in the manual process, whereas, Figure 26 shows the proportion of time each task took to complete in the automated process. It is evident from Figure 25 and Figure 26 that the operations in the Deployment of Codebase task take the most time to perform regardless of the use of public cloud and IaC technologies, in fact, the proportion of time this task took to complete in the automated process increased by over 91% when compared to the manual process. While it only accounts for a small portion of time in both processes, it is worth mentioning that the proportion of time to complete the External DNS Creation task increased by over 207% in environments created by the automated process. Speculation as to why these increases in proportional time occurred is covered in the Discussion chapter of this document.

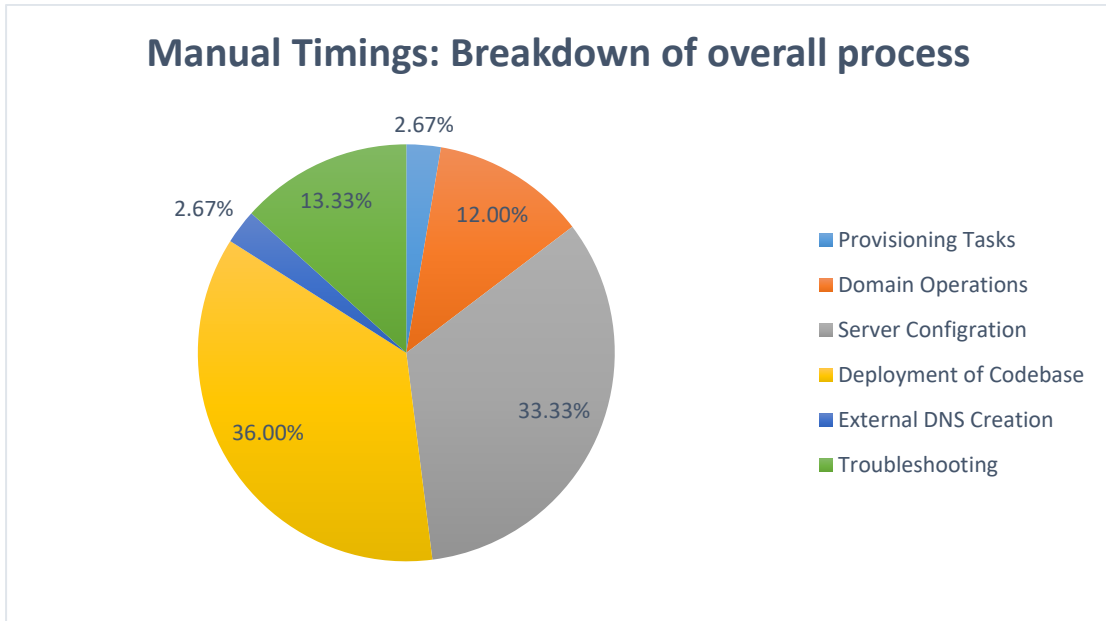


Figure 25: Manual Timings: Breakdown of overall process

One can also see from Figure 25 and Figure 26 that the proportion of time taken to perform the Provisioning Tasks, Domain Operations, Server Configuration and Troubleshooting tasks have all been reduced by a significant factor. Server Configuration is the most pertinent area to focus on here as these tasks are the second most time consuming to carry out in both the automated and the manual process. These tasks take up exactly one third of the total time in the manual process.

Whereas, in the automated process, Server Configuration tasks only account for 13.5%. Another interesting metric comparison to note from Figure 25 and Figure 26 is the discrepancy between the proportions of time spent on the Troubleshooting task. In the manual process, the Troubleshooting tasks account for 13.3% of the total process time, whereas, in the automated process only 2.4% of the total process time is spent on these tasks.

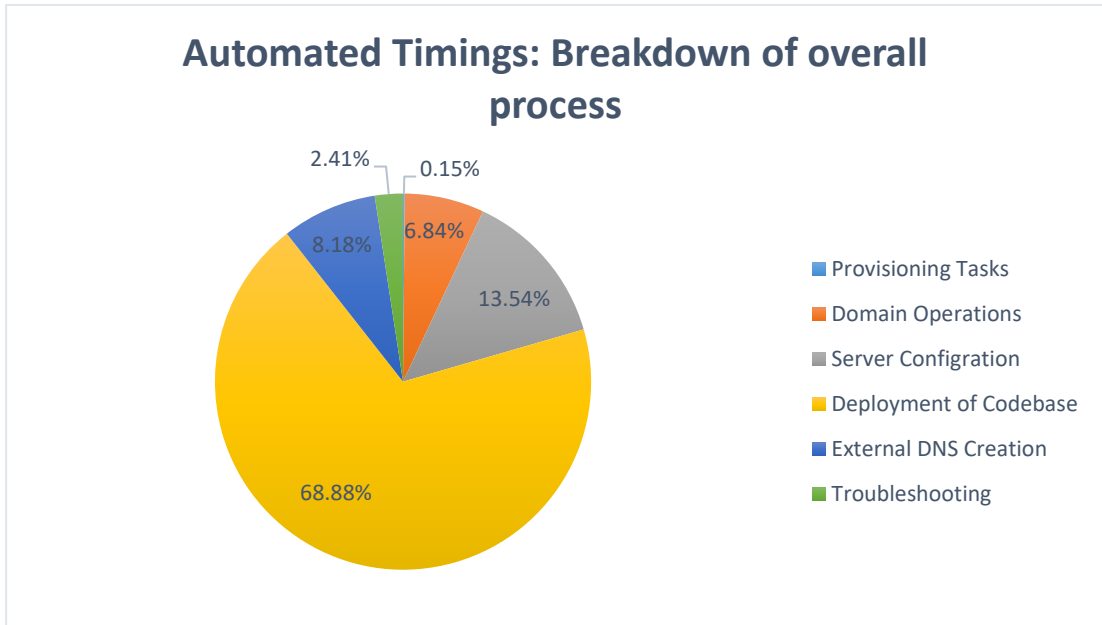


Figure 26: Automated Timings: Breakdown of overall process

4.2.3.4 Effort Comparison

Not only is the automated process several magnitudes faster than the manual equivalent, the effort overhead involved is also decreased significantly. The only tasks in the automated set that contain any effort overhead are the Troubleshooting and External DNS Creation tasks. While it was impossible to automate the External DNS Creation task, the framework does allow for a reduction in manual troubleshooting time of 4.8 hours per environment, which is a proportional decrease in time of 96%. The automated process contains only a fraction of manual work, calculated by adding the averages of the Troubleshooting and External DNS Creation tasks. The total manual work involved in the manual process is 2,250 minutes or 37.5 hours, whereas, the total average manual work involved across all runs of the automated process is a mere 52 minutes, this comparison reveals a difference between the two metrics. In total, the manual process requires over 43 times more manual effort than that of the automated process.

4.2.3.5 Error Tendency

The last metric presented in this section is the tendency for error in the environment creation process. In the manual dataset, error occurrence was not collected as a variable for comparison, as no accurate estimate could be given. The occurrence of errors was mentioned in every interview held with the case study organisation staff members, but no official error rate was declared. Instead, each interviewee allocated a relatively large amount of time for manual verification and troubleshooting purposes to pre-empt the time needed to deal with errors in the manual process.

Error occurrence was collected as a variable for the automated process, as these errors are exposed through the TeamCity front-end and are visible whenever they occur. The researcher encapsulates the time taken to resolve errors that occur during the run of the automated process in the Troubleshooting task, the timings of which were presented earlier in this section.

Across all 39 runs of the framework, the rate of error was relatively high. On average per framework run, at least one of the builds in the chain failed 51.8% of the time.

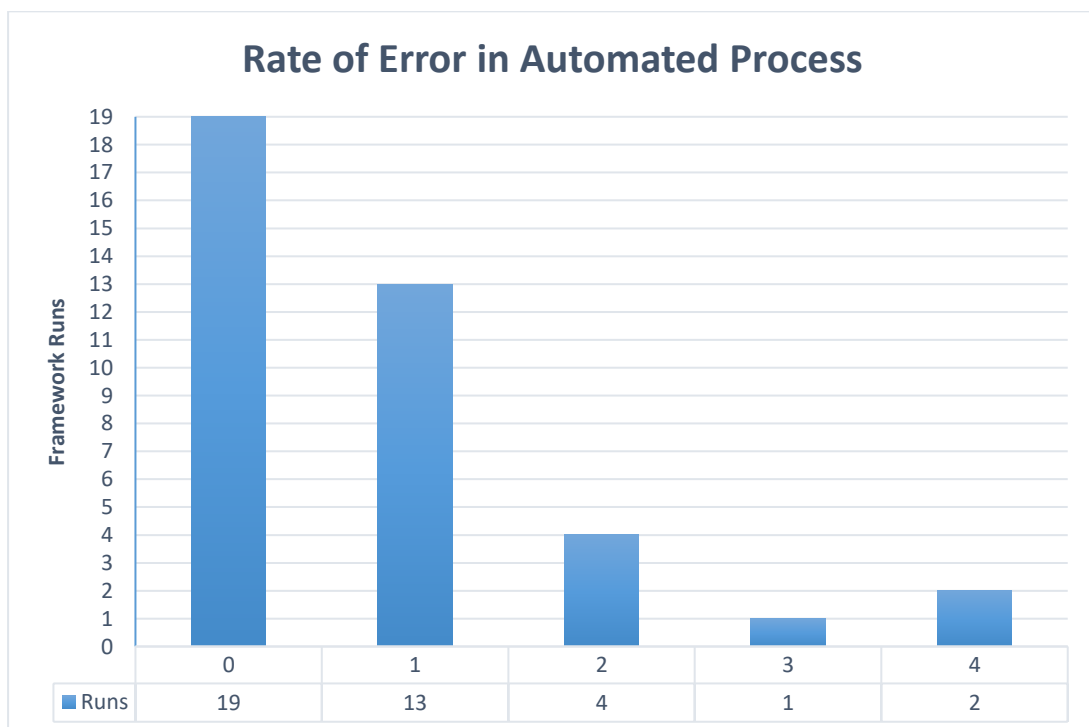


Figure 27: Rate of Error in Automated Process

Meaning that, in practise, the framework was more likely to fail with a terminating error than it was to succeed. This failure rate may seem unacceptably high, but the vast majority of these errors only occurred once during that specific framework run. In other words, if the framework failed once, it was more than likely not going to fail again after it was restarted. Figure 27 shows the breakdown in the rate of error across framework runs, 19 framework runs ran completely autonomously with no human intervention, whereas, 13 framework runs failed with a single error, the cause of which needed to be identified and resolved before the framework could be started again from its point of failure, after which the build chain continued without any subsequent errors. The remaining seven of framework runs failed more than once. As one can expect, the time spent troubleshooting framework runs with multiple errors increases with the amount of errors that occur. Figure 28 shows the median average time spent troubleshooting the automated environment creation process per error occurrence, note that the data visualised in Figure 27 and Figure 28 are comparing different metrics, but they are almost a mirror image of one another.

This infers that, the most frequent types of framework executions succeeded with fewer errors and fewer troubleshooting times, whereas, the least frequent types of framework executions failed with more errors and more troubleshooting times. The causes of, and remediation to these failures are covered in the Discussion chapter of this document as this current section is reserved for results only.

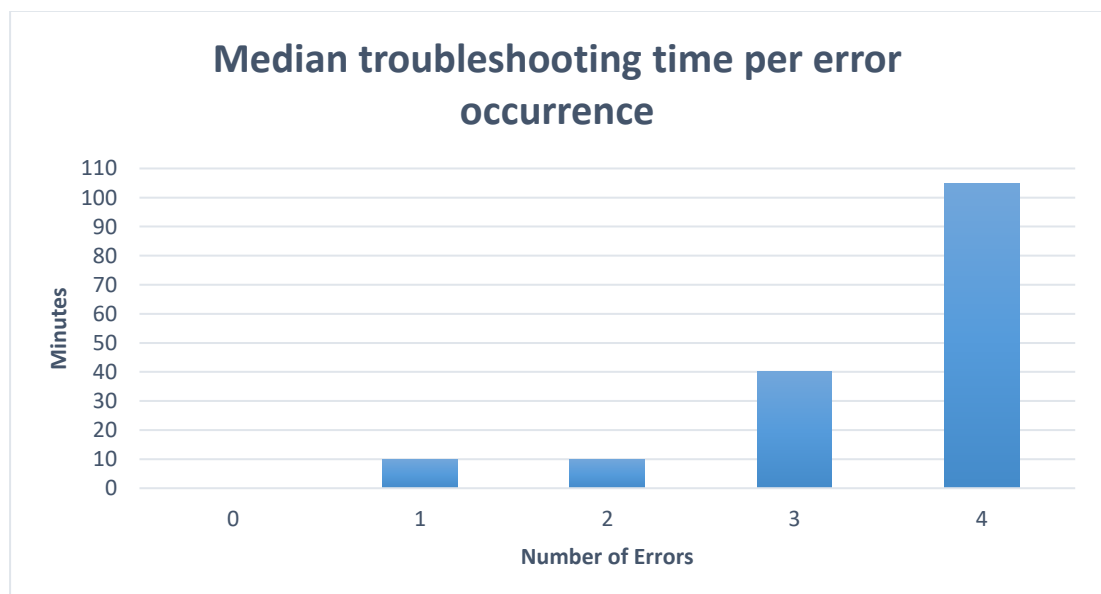


Figure 28: Median troubleshooting time per error occurrence

4.2.4 Comparison of Creation and Recreation Data Datasets

The Creation and Recreation subsets that comprise the Automated dataset are presented and compared in this section. The layout of this section follows the same format as the preceding section.

As previously discussed, the framework is built to handle two distinct scenarios, either the creation of a new environment, native to public cloud infrastructure, or the recreation of an existing, in-house environment on public cloud infrastructure. These subsets of data were compared in order to determine if the same metrics presented in section 4.2.3 vary in any form when the framework is building a new environment or recreating an old environment.

4.2.4.1 Overall Process Comparison

As a whole, the overall process timings between the Creation and Recreation subsets are far closer to one another in value than the Automated and Manual sets are to one another. This comparison is illustrated in Figure 29.

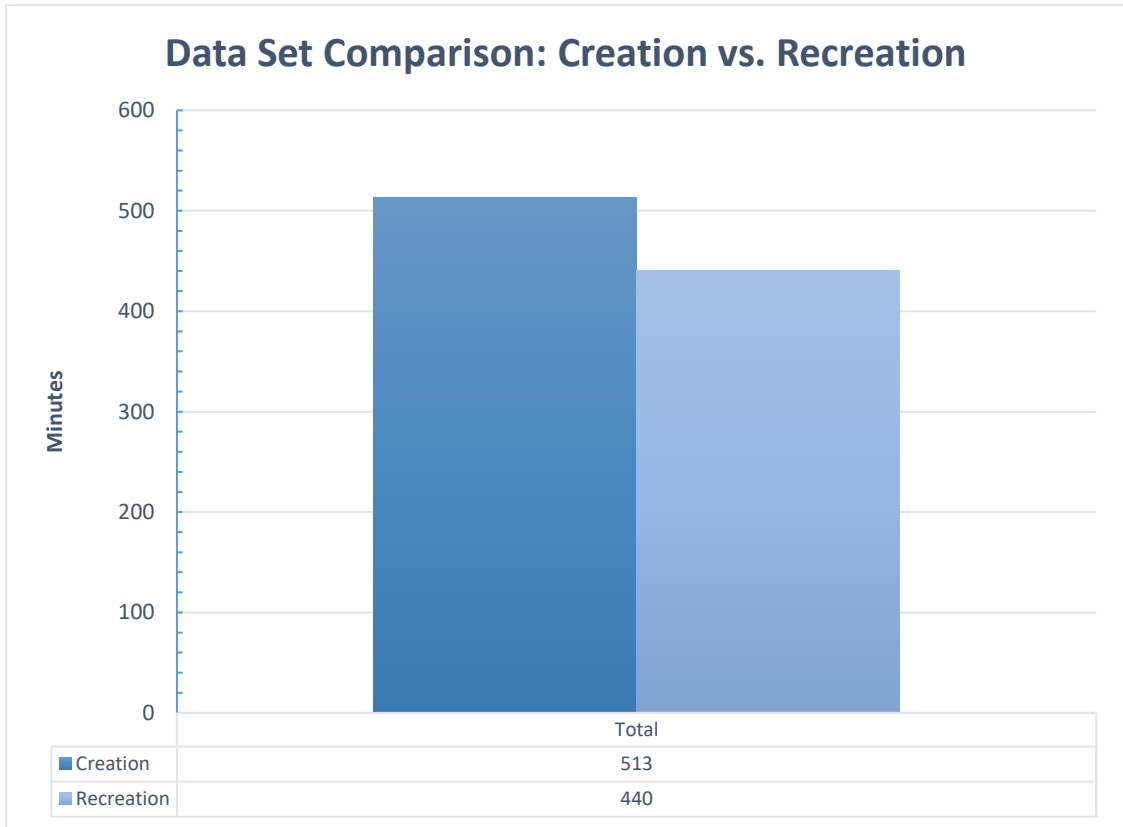


Figure 29: Data Set Comparison: Creation vs. Recreation

While the comparison of the two subsets are not as dramatically juxtaposed as previous comparisons, there is a large discrepancy between the two that should be addressed. 73 minutes or 16.59% of a difference was recorded in average process timings between the two subsets.

4.2.4.2 Individual Task Comparison

A view of the individual task comparison between these two subsets reveal that most tasks are relatively similar in timings, which is what was expected when comparing two sets of automated process timings that are largely performing the same actions. Figure 30 presents the average time taken to perform each individual task from both subsets of data.

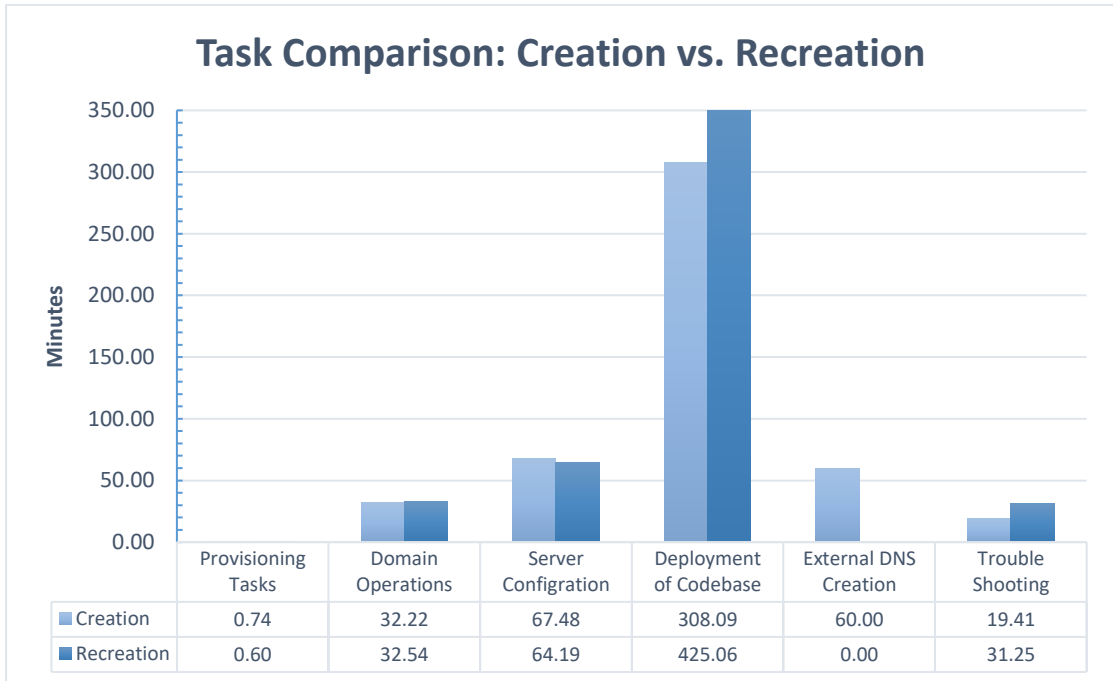


Figure 30: Task Comparison: Creation vs. Recreation

The disparity in timings seen in Figure 29 is easily accounted for by examining Figure 30, the External DNS Creation task is the main culprit for the difference in the overall process timings, adding an extra 60 minutes to the Creation timings. This is because, in the Recreation process, these entries were already in place and, therefore, did not need to be created, so this task was only applicable to the Recreation process. Were this task removed from the Creation timings, then the Recreation timings would be 13 minutes faster than the Creation timing sets.

In terms of manual effort overhead, task proportion and error rates between the two subsets, the data are relatively identical, bar the External DNS Creation task, therefore, further detailed comparisons of these results are not warranted. If it was possible to automate the External DNS Creation task, the two subsets would be so similar that a comparison of timings would be completely redundant.

4.2.5 Summary

The overall trend in data goes to show that the automated process is far more efficient and more autonomous than the manual process. The framework itself is far from perfect, but even with its tendency for failure, it is by far a more efficient system for

creating new, and rebuilding old environments in the public cloud. In terms of process comparison between the two framework execution scenarios, creation and recreation, the only significant difference is the time taken to perform the single remaining static manual task. The results show that, if it were possible to automate the External DNS Creation task, the process of creation and recreation would be practically identical in terms of timings.

4.3 Secondary Experiments

This section details the secondary experiments carried out as part of this thesis. The purpose of these experiments is to test the framework in scenarios where controlled variables are intentionally modified by the researcher in order to demonstrate the efficacy of the framework when executing under different conditions. By doing so, the research objective where the framework is tested under as many conditions as possible is achieved.

The environments created via the framework for the secondary experiments were not used by the organisation the framework was implemented in, as these environments were built for the purpose of extensive testing of the framework. Along with this, the controlled modification of process variables inferred that the environments were being created via a non-standard method to all other environments recreated and created by the framework in the Recreation/Creation experiment, it was not desirable for the organisation's development or testing departments to work on non-standard environments, regardless of how insignificant the modification to the environment creation process was. As it was planned that the environments created as part of these secondary experiment were not to be used by the organisation the framework was implemented in, the External DNS Creation task did not take place for these environments as this was not automated as part of the framework and carrying out this task was a redundant step. Therefore, this task is not represented in the timing data presented in the sections that follow.

The time spent troubleshooting these environments were relatively low compared to the results presented in the Recreation/Creation experiment. As the secondary experiments were conducted in a short span of time, changes to the 174 code bases

deployed to the environments were minimal, which resulted in a generally low rate of error and predictable patterns for error correction.

Two secondary experiments were carried out by the research to achieve the research objective, both follow the same methodology as the Recreation/Creation experiment outlined by Amaral (Amaral, 2011). These secondary experiments took place under the same context as the Recreation/Creation experiment, in that, they were conducted via the same implementation of the framework, in the same organisation and were used to create the same environments. The only difference in these experiments is the controlled modification of process variables. The first experiment consists of the modification of the instance type process variable, this experiment will henceforth be referred to as the Instance Type experiment. This experiment involves the carrying out of full framework executions in order to create environments of varying compute power. The second consists of the modification of the storage allocated to the instance and the AMI to provision process variables, this experiment will henceforth be referred to as the Storage Capacity experiment. This experiment involves the carrying out of full framework executions in order to create environments with varying amounts of storage allocated to them.

4.3.1 Instance Type Experiment Context

The purpose of Instance Type experiment is to determine whether or not the performance of the automated framework is affected by the compute power of the instance it is targeting. By changing the controlled variable of the instance type being provisioned, this experiment demonstrates how the framework behaves when instructed to build environments of varying compute power.

As mentioned above, the Instance Type experiment modifies the instance type process variable, this variable is outlined in 4.1.4. In the Recreation/Creation experiment, the t2.large instance type was provisioned for all environments built via the framework, therefore, the t2.large is the baseline instance type for comparison. This experiment encompasses the automated framework execution when instance types being provisioned are set to the t2.medium, the baseline t2.large and the t2.xlarge instance types. The compute power of all three of the aforementioned instance types have been detailed in Table 11.

Instance Type	RAM (GB)	vCPUs
t2.medium	4	2
t2.large	8	2
t2.xlarge	16	4

Table 11: Instance Types used

Taking the already established baseline of the t2.large instance type into account, the above instance types were chosen by the amount of RAM and vCPUs allocated to them relative to the t2.large instance type. The t2.xlarge instance type has twice the amount of RAM and vCPUs allocated to it than the t2.large. Whereas, the t2.medium instance type has half the amount of RAM allocated to it than the t2.large, but has the same amount of vCPUs as this is the minimum amount of vCPUs available for instance types in the t2 instance family with 4GB of RAM (Amazon, 2017).

4.3.1.1 Instance Type Experiment Scope and Conduction

The scope of the Instance Type experiment encompasses the automated creation of new environments via the framework when the instance type variable is modified in a controlled manner. For the purpose of eliminating errors in measurements and calculating an accurate average for the results, the framework built three environments from each instance type. The framework built three environments of the t2.medium instance type, then built 3 environments of the t2.large instance type and finally built three environments of the t2.xlarge instance type.

The conduction of this experiment was similar to the conduction of the Recreation/Creation experiment, where the researcher allocated a timeslot for the framework to run on out of business hours if the framework execution took place on a weekday or anytime during a weekend. Following the success of the framework execution, the researcher manually entered the timing data into a spreadsheet for later analysis. To ensure that there was as little interference as possible in the results, only a single framework execution took place at any given time throughout the course of this experiment.

4.3.2 Instance Type Experiment Results

The results of the Instance Type experiment are detailed in this section.

4.3.2.1 Data Analysis

This section describes the analysis and interpretation of the raw data obtained through execution of the automated framework in the context of the Instance Type experiments.

Similar to the Recreation/Creation experiment, the raw data was gathered via the TeamCity build chain interface, alongside estimates of all manual troubleshooting tasks if they needed to be performed. The following categories are included in the Instance Type experiment dataset:

1. *Provisioning Tasks*
2. *Domain Operations*
3. *Server Configuration*
4. *Deployment of Codebase*
5. *Troubleshooting*

The mean average for each of the above categories of timing data obtained from the Instance Type experiment was calculated. Following this, the time taken for each task in minutes was calculated as some builds take hours, while others take seconds and it was desirable to group all data in a single format in order for it to be readable.

4.3.2.2 Sample Size

A total of nine environments were created through the framework in this experiment, making a total sample size of three datasets for each instance type.

4.3.2.3 Comparison of Instance Type Datasets

The Instance Type datasets are presented and compared in this section, the raw data from each dataset is plotted out on the same charts in order to provide a clear comparison of timings from all three datasets.

4.3.2.4 Overall Process Comparison

The timing data from the framework executions involved in the Instance Type experiment as a whole process are presented and compared in this section. Figure 31 shows the overall difference in timings between framework executions when the instance type variable is modified.

Figure 31 shows that the type of instance being built by the framework has an effect on the overall execution times of the process as a whole. The t2.medium instance type has the lowest amount of compute power allocated to it and, on average, takes the longest to provision, configure and deploy to. The t2.large instance type is as close to the mid-point in compute power as was possible in this experiment, environments set to this instance type are created 23 minutes or 3.6% faster than environments of the t2.medium instance type.

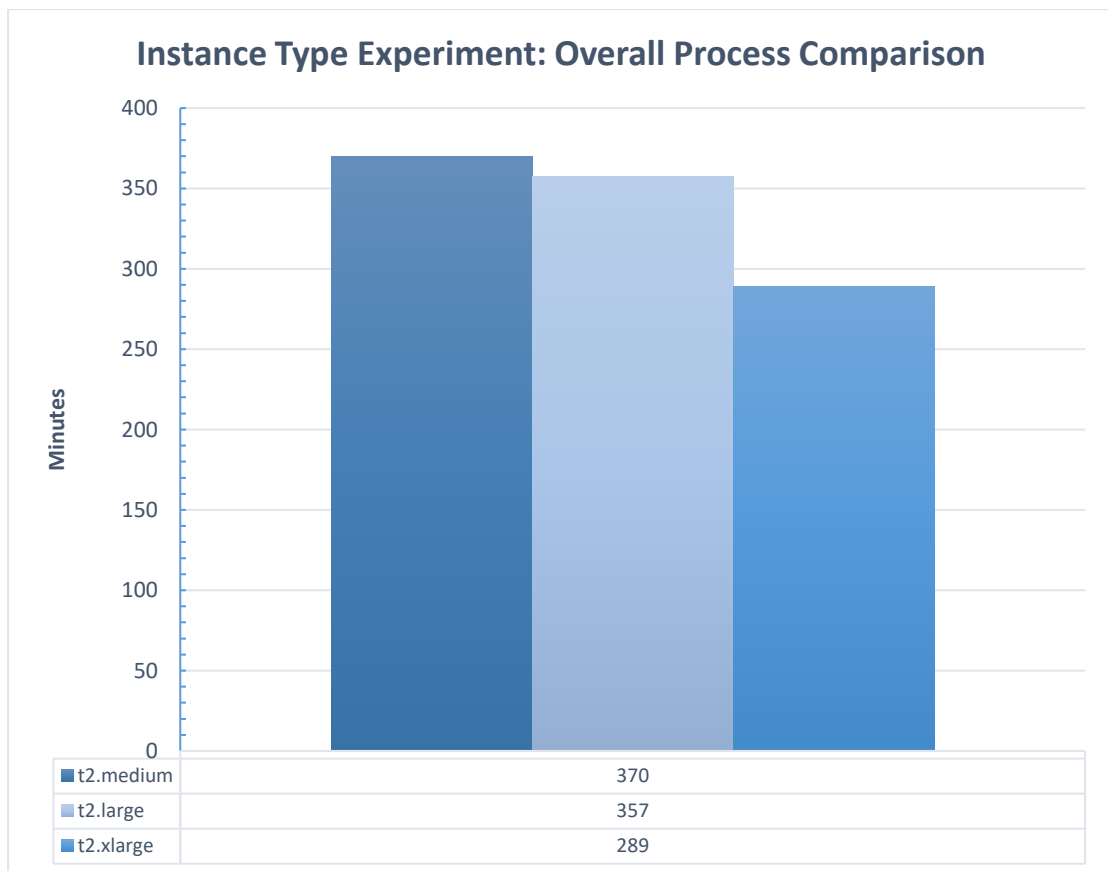


Figure 31: Instance Type Experiment Overall Comparison

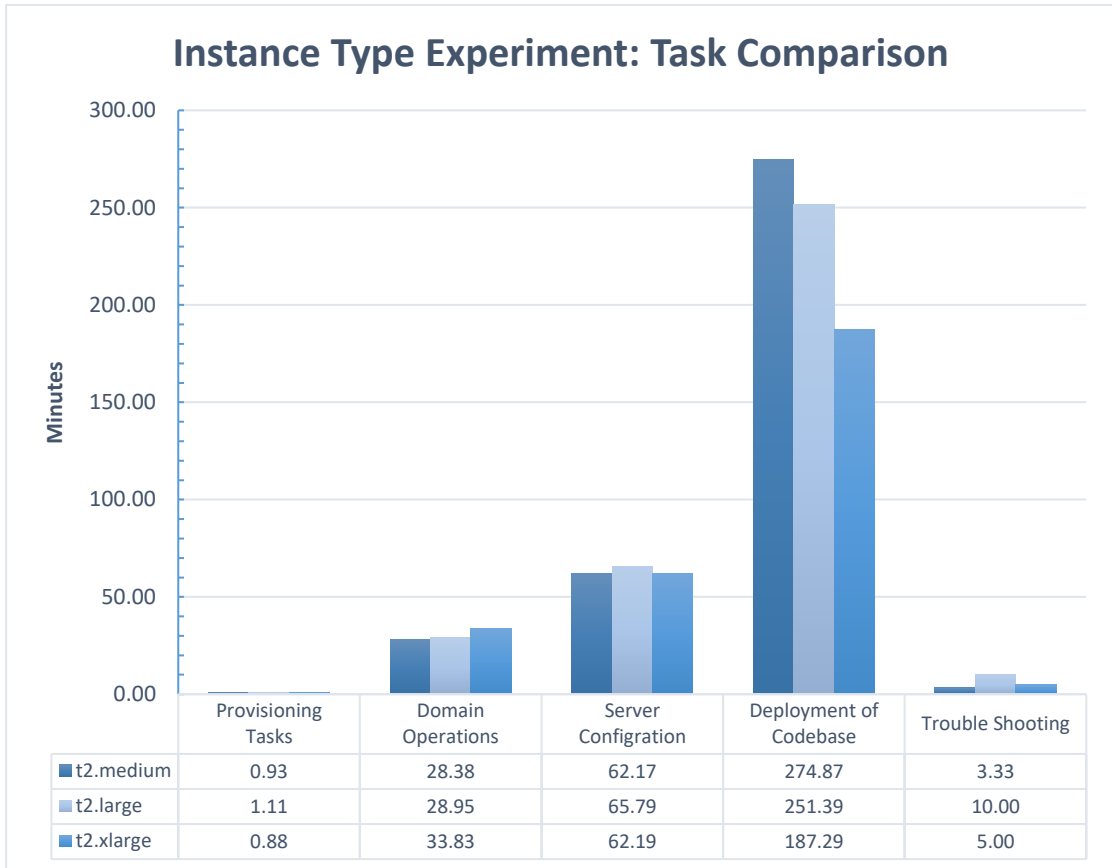


Figure 32: Instance Type Experiment Task Comparison

This difference in framework execution times is not a significant one, considering that the compute power of the environment being created was effectively halved. However, this difference in timings is far greater with the t2.xlarge instance type. Environments set to the t2.xlarge instance type were created 68 minutes or 23.5% faster than their t2.large equivalents and 81 minutes or 28% faster than their t2.medium equivalents.

4.3.2.5 Individual Task Comparison

Figure 32 shows a breakdown of the overall process, displaying each task and the associated average time taken for each task. Figure 32 shows that the compute power of the environment being created significantly affects the Deployment of Codebase task, while all other tasks are relatively the same throughout each instance type if uncontrolled variables such as network latency across and the VPN and the shared tenancy of the TeamCity build server are taken into account. Whereas, the Domain

Operations task took the least amount of time for the t2.medium, followed by the t2.large instance type.

The t2.xlarge instance type took the longest to complete the Domain Operations step. No significant differences can be seen in the timing data for the Server Configuration tasks, this can be attributed to the identical configuration management scripts that were run on the each instance in this experiment. The main gains in efficiency based on compute power can be seen in the Deployment of Codebase task which largely resulted in the t2.xlarge instance type handling this task 46.7% faster than the t2.medium and 34.2% faster than the standard t2.large instance type.

4.3.3 Storage Capacity Experiment Context

The purpose of the Storage Capacity experiment is to determine whether or not the performance of the framework is affected by the amount of storage allocated the environment being provisioned. This test will demonstrate how the framework behaves in circumstances where the allocated storage space of the environment being provisioning is modified. It is expected that the framework will be capable of creating environments with less storage space allocated to them faster than that of environments with more space allocated to them.

The Storage Capacity experiment modifies the storage allocated to the instance variable, in order to do this, the AMI to provision variable also needed to be modified. These variables are outlined in section 4.1.4. Environments covered in the Recreation/Creation experiment were allocated four drives, with a combined total of 250GB of disk space. These four drives are required for the framework to execute as the configuration and deployment of the target organisation's application and database code are dependent on these four drives being present and for these four drives to have adequate space for the code base to be deployed to and operate in. This experiment encompasses the automated framework execution when the storage allocated to the instance being provisioned is set to varying amounts. The storage capacity allocated to environments in the Recreation/Creation experiment acts a baseline capacity in this experiment, a lower capacity storage amount relative to the baseline and a higher capacity storage amount relative to the baseline are also used in this experiment.

Baseline AMI	Capacity (GB)	Used (GB)	Free (GB)
C	70	44	26
D	100	65	35
E	40	27	13
F	40	1	39
Total	250	137	113

Table 12: Baseline AMI Storage

As the four drives are required for the framework to execute in the context of the target organisations environments, the following storage options will be used in this test. Table 12 outlines the storage allocated to environments already created by the framework. Table 13, the Low Capacity AMI and Table 14. The High Capacity AMI outline a difference of overall provisioned storage space of 20% from the baseline. The Low Capacity totalling at 20% lower than the Baseline, and the High Capacity totalling at 20% higher than the Baseline. This specific metric was chosen as 20% lower storage space than the Baseline is close to the minimum amount required for a working test environment to function. An increase of more than 20% storage space was chosen as it corresponds with the amount reduced in the Low Capacity test. Comparing the framework execution time against an increase of 90% would be an interesting test, but would offer no real value without an accompanying test where the storage was reduced by 90%, which, as outlined above, was impossible to perform. The capacity on the C drive could have been reduced further, but it is not currently possible to reduce a Windows EBS system volume (AWS, 2010). The instance type chosen for the instances in this test was the t2.large, this is to reflect the instance type used in the Recreation/Creation experiment tests.

Low Capacity AMI	Capacity (GB)	Used (GB)	Free (GB)
C	70	46	24
D	75	65	10
E	40	27	13
F	15	1	14
Total	200	139	61

Table 13: Low Capacity AMI Storage

High Capacity AMI	Capacity (GB)	Used (GB)	Free (GB)
C	70	46	24
D	140	65	75
E	50	27	23
F	40	1	39
Total	300	139	161

Table 14: High Capacity AMI Storage

Drives attached to environments built from the High Capacity AMI were extended drives based on the Baseline AMI, these extended drives were not filled with any data. The new capacity on these extended drives was comprised of empty space. The discrepancy of free space across the drives from the Low Capacity, Baseline and High Capacity tests is recognized but not addressed in the experiment as doing so would void the experimental methodology followed. It was desired to keep the contents and integrity of the disk drives standard throughout the experiment. For instance, adding an unnecessary 65GB file to the D drive in the High Capacity AMI to reflect the amount of free space on that drive in the Low Capacity tests would introduce a new variable to the test case that was not apparent in the original Recreation/Creation experiment. All drives begin with the same type and volume of data throughout each of these tests.

4.3.3.1 Storage Capacity Experiment Scope and Conduction

The scope of the Storage Capacity experiment encompasses the automated creation of new environments via the framework when the storage allocated to the instance and the AMI to provision variables are modified in a controlled manner. For the purpose of eliminating errors in measurements and calculating an accurate average for the results, the framework built three environments from each AMI. In order to achieve this, the researcher created two separate AMIs for each storage type that was not the baseline as the baseline AMI was the same used in the Recreation/Creation experiment and Instance Type experiment. One AMI was created for the Low Capacity test and one AMI was created for the High Capacity test. The framework built three environments from the low capacity AMI, then built 3 environments from the baseline AMI and finally built three environments from the high capacity AMI.

The conduction of this experiment was similar to the conduction of the Recreation/Creation experiment, and the Instance Type experiment where the researcher allocated a timeslot for the framework to run on out of business hours if the framework execution took place on a weekday or anytime during a weekend. Following the success of the framework execution, the researcher manually entered the timing data into a spreadsheet for later analysis. To ensure there was as little interference as possible in the results, only a single framework execution took place at any given time throughout the course of this experiment.

4.3.4 Storage Capacity Experiment Results

The results of the Storage Capacity experiment are detailed in this section.

4.3.4.1 Data Analysis

This section describes the analysis and interpretation of the raw data obtained through execution of the automated framework in the context of the Storage Capacity experiments. Similar to the Recreation/Creation experiment and the Instance Type experiment, the raw data was gathered via the TeamCity build chain interface, alongside estimates of all manual troubleshooting tasks if they needed to be performed. The following categories are included in the Storage Capacity experiment dataset:

1. *Provisioning Tasks*
2. *Domain Operations*
3. *Server Configuration*
4. *Deployment of Codebase*
5. *Troubleshooting*

The mean average for each of the above categories of timing data obtained from the Storage Capacity experiment was calculated. Following this, the time taken for each task in minutes was calculated as some builds take hours, while others take seconds and it was desirable to group all data in a single format in order for it to be readable.

4.3.4.2 Sample Size

A total of nine environments were created through the framework in this experiment, making a total sample size of three datasets for each instance type.

4.3.4.3 Comparison of Storage Capacity Datasets

The Storage Capacity datasets are presented and compared in this section, the raw data from each dataset is plotted out on the same charts in order to provide a clear comparison of timings from all three datasets.

4.3.4.4 Overall Process Comparison

The timing data from the framework executions involved in the Storage Capacity experiment as a whole process are presented and compared in this section. Figure 33 shows the overall difference in timings between framework executions when the storage allocated to the instance the framework is targeting is modified.

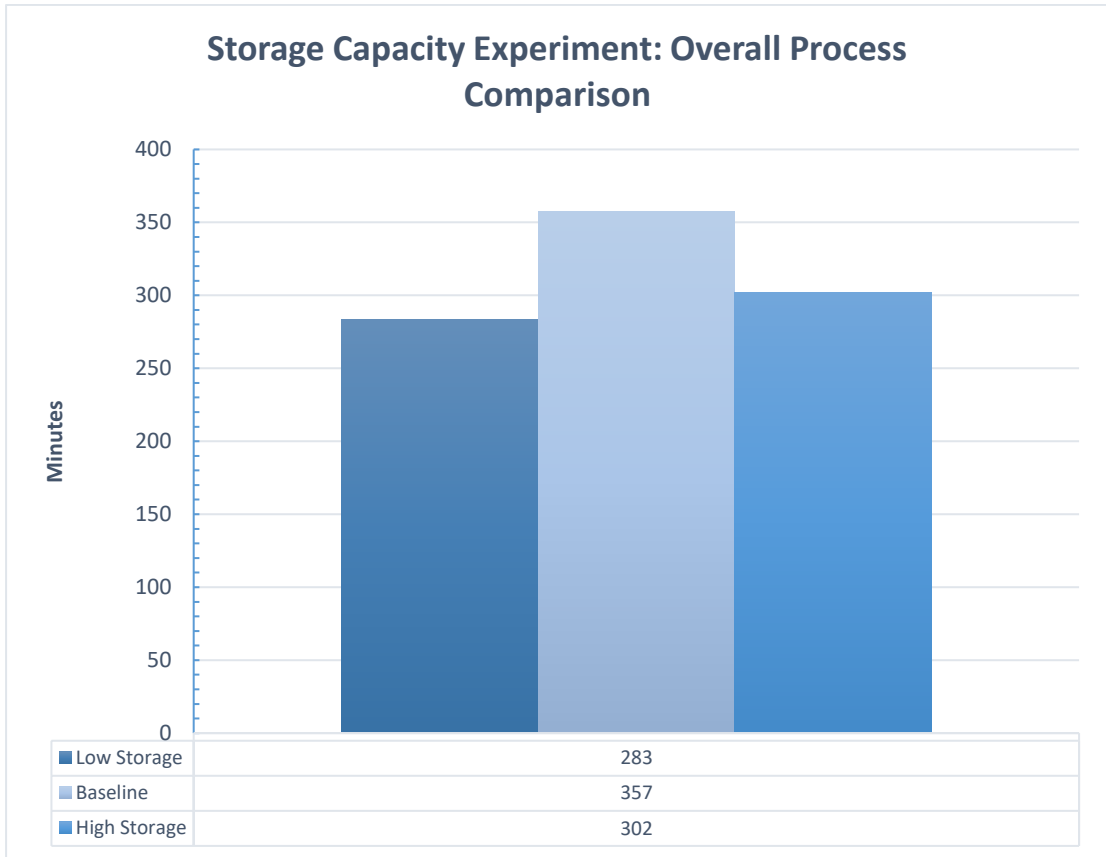


Figure 33: Storage Capacity Experiment Overall Process Comparison

Figure 33 shows that the storage allocated to the instance being built by the framework has an effect on the overall execution times of the process as a whole. The expected result was that instances allocated less storage will be provisioned, configured and deployed to faster than instances with more allocated storage, this expectation was proven by this experiment, yet the results do not follow a linear pattern. The time taken for the framework to build environments from the Low Storage AMI was by far the lowest. On average, the Low Storage environments were built 74 minutes, or 26.1% faster than environments built from the Baseline AMI and 19 minutes, or 6.7% faster than environments built from the High Storage AMI.

4.3.4.5 Individual Task Comparison

Figure 34 shows a breakdown of the overall process, displaying each task and the associated average time taken for each task.

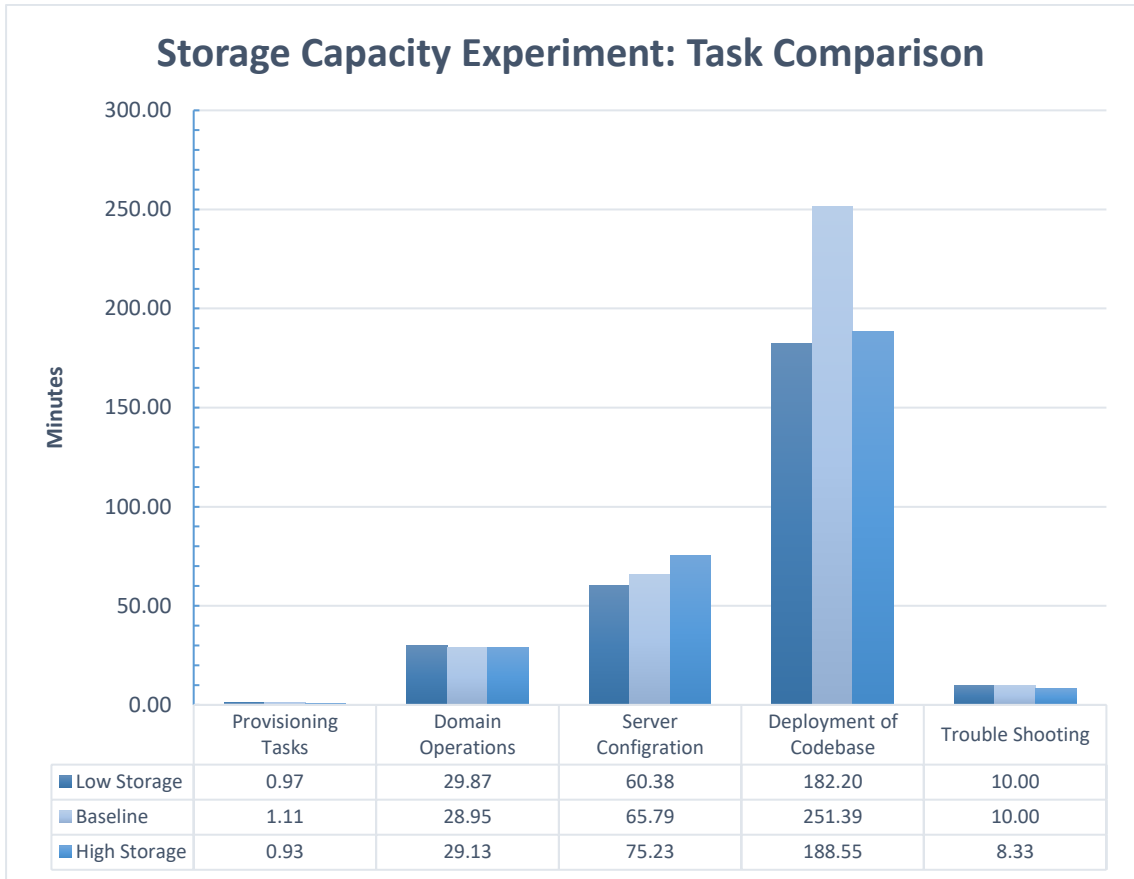


Figure 34: Storage Capacity Experiment Task Comparison

Figure 34 shows that the storage allocated to the environment being created by the framework can have a dramatic and non-linear effect on the speed of various tasks. The main tasks responsible for timing discrepancy in this experiment are the Server Configuration and Deployment of Codebase tasks, while all other tasks take relatively the same amount of time. An unexpected result is the increase in time require for the Server Configuration task to complete as the storage allocated to the environment is increased. The Low Storage AMI handles this task faster than the Baseline AMI, similarly the Baseline AMI handles this task faster than the High Storage AMI. The Deployment of Codebase task is the main culprit for the unpredicted skew in timing data, taking far longer for the Baseline AMI than the Low Storage or High Storage AMI. Another unanticipated difference to note here is the slight difference in timing between the Low Storage and High Storage AMIs. The Low Storage AMI handles this task slightly faster than the High Storage AMI.

4.4 Survey

The results of the survey questionnaire are presented in this section. The overall aim of the survey questionnaire results are to corroborate the correlation between the use of IaC tools and public cloud infrastructure and efficiency in the environment creation process found via the experimental results outlined in the previous sections. This corroboration was achieved by surveying and obtaining data from a wider audience of participants that represent other organisations independent from the organisation the researcher carried out the case study, framework implementation and subsequent experiments in.

The survey research method was employed in order to achieve the research objective pertaining to the surveying of the wider audience of SMEs in order to validate the results obtained from the automated framework discussed above in a generalizable fashion. The survey method allows for inferences to be made about a population based on information obtained from a smaller subset of that population (Schonlau, 2002). The survey was created with the aim to investigate if a correlation exists between efficiency in the process of provisioning IT infrastructure and the use of IaaS and IaC tools in the context of the wider audience of software engineering organisations. The survey respondents were comprised of technical employees currently working in software engineering companies that are independent from the organisation the case study and experiments were carried out in.

4.4.1 Sampling

Etikan et al. defines exponential non-discriminative snowball sampling as a non-probability based sampling technique in which the initial research participant, or participants, recruits at least one more participant, those participants then recruit at least one more participant each, and so on until sampling has ended (Etikan, 2016). This sampling technique was adopted for retrieving respondents to the survey. The sample set was initially seeded by the connections of the researcher known to be working in the field. This initial pool of potential respondents was limited to 8 people. Alongside this, a staff member employed within the case study organisation was aware of the survey and its distribution model, they voluntarily provided contact details for

5 of their previous work colleagues. This was key to extending the reach of the survey beyond potential participants that were known personally to the researcher. This led to the final seed pool of potential participants, which totalled at 13.

All survey respondents were asked to forward the survey onto their connections, based on the criteria that their connection is working for an SME and has the necessary knowledge to fill out the questions in the survey. This snowball process continued until it was apparent that no more survey respondents were being received. By the time the survey was closed, 19 respondents, each a member of a different SME operating inside of the Republic of Ireland, attempted the survey. Of those, 13 finished the survey completely. 11 of the completed responses qualified for inclusion into the final results set. The 2 excluded respondents failed to qualify for the final results set because they showed a lack of understanding in their organisation's environment creation process that may have compromised the validity of the results if their answers were included. These 2 respondents stated that they did not know whether or not IaC tools are used in their organisation's environment creation process. The use of these tools are key comparative variable in the results.

Due to the sampling method chosen, it is impossible for the researcher to calculate response and refusal rate, as it is not known exactly how many respondents recruited other respondents. According to the Irish Times, there are 139 technology SMEs operating in the Republic of Ireland (Irish Times, 2017). If this is taken as the total population size, then the final set of 11 respondents can be said to represent 7.91% of that population.

4.4.2 Measurement Procedures

A web-based survey questionnaire was chosen as the method of delivery for this portion of the research. This method was deemed most appropriate as interviewing each potential subject was not practical in terms of the time and resource overhead involved.

The questionnaire itself consists of a mixed of open and closed-ended questions, with the bulk of data being derived from a matrix question consisting of dropdown menus. The initial draft was created with the intention of encompassing as much as possible but was deemed far too long when the survey was tested amongst the researcher's

peers, the final draft consists of 13 questions. Each question in the final draft was designed with the research question and hypothesis in mind. The introduction page of the survey consists of a brief message thanking the respondent for taking the survey and explaining what the results will be used for and why they are needed. The survey begins with classification questions in order to eliminate respondents deemed outside of the scope of the research, for instance: those not aware of how environments in their organisations are provisioned or how many people are involved in the environment creation process. After the classification questions, the central matrix question is posed. This page begins with an introduction, explaining the terms used in the question and specifying the importance of the participants understanding of the terms. The respondent is then prompted to translate the following tasks into their own organisations environment creation process and provide timings for each of the following processes:

- *Provisioning of new environment.*
- *Domain operations.*
- *Configuration of server/servers.*
- *Deployment of codebase.*

For each of the above, the respondent is given a choice of the following timings:

- *1 - 10 minutes*
- *10 - 30 minutes*
- *30 - 60 minutes*
- *1 - 2 hours*
- *2 - 5 hours*
- *5 - 8 hours*
- *8 - 16 hours*
- *16 - 24 hours*
- *1 - 2 days*
- *2 - 3 days*

Appended to the end of this question is a free text box for the participant to input any steps, and associated timings, that the researcher did not account for. Once the main matrix question is complete, the respondents have entered the bulk of the data that the final results set will consist of. The remaining questions are close-ended and were used to categorise each of the respondents qualified for inclusion into the final results set. These classification questions are largely optional but were included to give to researcher a more expansive view of the respondent's organisations environment infrastructure and automation tool usage. The full survey has been exported through a series of screenshots and can be found in Appendix D in this document.

4.4.3 Data Collection

The survey went live and the researcher began looking for adequate respondents on 26th of August 2016. The feedback from the survey was received between 28th of August 2016 to 26th of September 2016, the survey was subsequently closed on 3rd of October 2016 after a week of respondent inactivity. After the survey was closed, the data itself was exported into Microsoft Excel for the researcher to analyse and interpret.

4.4.4 Respondent Category Comparison

Due to the volume and variation of raw data retrieved through the survey questionnaire regarding the timings of each respondent's environment creation process, the results have been simplified in order to display them as part of this thesis. Firstly, each respondent was categorised based on their answers to the classification questions at the beginning of the survey, this allows for a meaningful comparison and differentiation between respondents.

- *Category A - Three respondents using IaC tools on in-house infrastructure*
- *Category B - Three respondents using IaC tools on public cloud infrastructure*
- *Category C - Three respondents not using IaC tools on in-house infrastructure*
- *Category D - Two respondents not using IaC tools on public cloud infrastructure*

Secondly, the midpoint of the timings provided by each respondent for each step in their organisation’s environment creation process was extracted, i.e. if a respondent stated that the provisioning step in their process took 10 - 30 minutes, then the value extracted is 15 minutes, similarly, if a respondent stated that the deployment of their application and database code base step took 4 - 8 hours then the value extracted is 6 hours or 21,600 minutes. Because of the difference in the amount of respondents in each of the above categories, the results had to be simplified again by finding the median average of each timing in each category. The results plotted out in Figure 35 show a clear comparison between categories which suggest that Category B, those utilising infrastructure as code tools alongside public cloud, alongside Category A, those utilising infrastructure as code tools in-house both have a dramatically lower environment creation time compared to the other respondent categories.

Category A respondents are over 13 times more efficient than Category C respondents and over 29 times more efficient than Category D respondents. While Category B respondents are over twice as efficient as respondents in Category A and several magnitudes more efficient than respondents from Category C and Category D. The timespan an environment is actively used for before being destroyed is an important metric derived from the survey results.

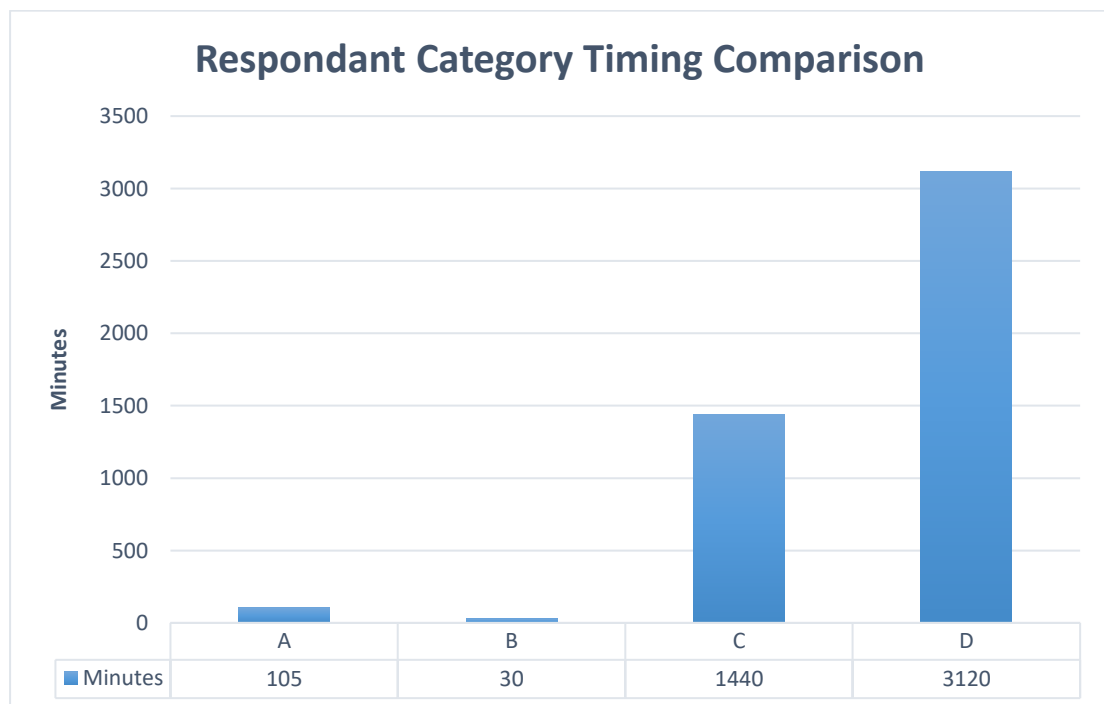


Figure 35: Respondent Category Timing Comparison

The majority of Category A and Category B respondents stated that their environments are used for a maximum of one year before being destroyed, whereas, the majority of Category C and Category D respondents stated that their environments are active for at least a year before being destroyed.

The number of servers that comprise a test environment in the respondent's organisation is another important metric, it should be mentioned that, like the case study organisation, all respondents in Category B declared that their test environments consist of a single server. However, at least one respondent from all other categories declared that over ten servers comprise a single test environment in their organisation, which may explain why the average environment creation time is several magnitudes higher than those in Category B. While the above is a major variable to take into consideration when assessing the viability of Figure 35, an undisputable factor relating to efficiency and cost savings in the environment creation process was revealed by the survey results. This factor is the amount of staff members involved in the respondents environment creation process. Figure 36 outlines the average number of staff members required to perform tasks in the environment creation process across each category of respondents.

Category B respondents require only a single person to perform tasks which are largely automated by IaC tools and public cloud infrastructure.

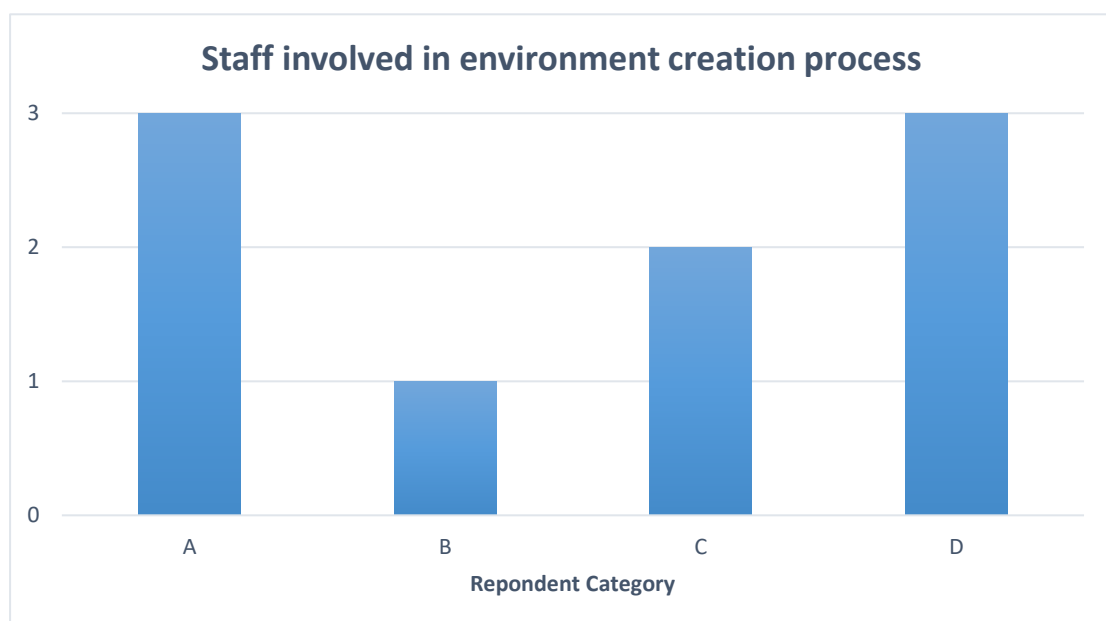


Figure 36: Staff involved in environment creation process

One can see that all other respondents involve no less than an average of two staff members in their environment creation process.

4.4.5 Summary

The survey results main use in this body of work is to support or refute the Creation/Recreation experiment results. From reading the above, one can see that the survey results corroborate the results obtained from the Creation/Recreation experiment. These results show how the proper use of IaC tools and public cloud infrastructure can lead to a highly efficient environment creation process when compared with those not using IaC and/or public cloud infrastructure. For the intents and purposes of this body of work, the survey results are an adequate compliment to the Creation/Recreation experiment results.

4.5 Review of Results

This section provides an in-depth review and discussion of the results presented in the above sections in order to provide a meaningful interpretation of the data and its impact in the field of study. Throughout this section, context for certain phenomena outlined but not explained in the above results sections is provided. To ensure as much of a logical flow as possible, this section reviews and discusses the results in the same order that they were presented in above.

4.5.1 Review of Creation/Recreation Experiment Results

The results of the Creation/Recreation experiment are reviewed and discussed in this section, covering each section that merits discussion in the order they were presented.

4.5.1.1 Review of Overall Process Comparison Results

The overall comparison between the manual and automated processes are presented in section 4.2.3.1 of this thesis. The data outlined in this section demonstrates the

efficiency capabilities that these technologies can provide to an organisation when they are implemented correctly. Figure 23 is the best method of visualizing the difference in timings between the two processes. It shows that, when the processes are compared, the automated process is 360% faster than the manual process. The data this figure illustrates is an adequate means for answering the research question inside of the context of the organisation the framework was implemented in.

A valid question arises when examining Figure 23 one pertaining to the generalizability of the manual process timings in the context of the wider audience. It could be argued that the case study organisation had an extremely inefficient environment creation process and the automated timing comparison is intentionally providing a false equivalency to bolster the efficacy of the framework. However, the survey results presented in section 4.4.4 show that organisations not utilizing IaC tools take, on average, relatively the same time to create environments through their manual processes. The results of the survey questionnaire carried out as part of this study also show that organisations that have implemented IaC tools have a far more streamlined environment creation process than that achieved in the case study via the automated framework. To further this point, external industry-based surveys detailed in sections 2.1.3 and 2.3.1 of the Background and Literature Review chapter also present data suggesting that faster access to infrastructure and faster configuration workflows are two of the most cited benefits of implementing IaaS and IaC tools (RightScale, 2014) (RightScale, 2015) (Forrester, 2015). Therefore, from the above, one can conclude that the comparison between the overall manual and automated processes is indeed valid, and that similar results could be obtained if the framework was to be implemented in an organisation that was not currently utilizing IaC tools or IaaS.

4.5.1.2 Review of Individual Task Comparison Results

The individual task comparison between the manual and automated processes is presented in section 4.2.4.2. Figure 24 visualizes varying discrepancies between the timings in the automated and manual processes, all tasks are faster when run under the automated system, but some outrank others by several magnitudes in terms of speed. Figure 25 presents the proportion of time taken for each task in the manual

process and Figure 26 presents the proportion of time taken for each task in the automated process.

From examining the data presented in this section, one can see that the Provisioning Tasks and Server Configuration tasks perform the best under the automated system. This is because these tasks are comprised almost entirely of the automated modification of simple IaC and configuration management scripts and the execution of them in order to create and configure the new resources in AWS. When these tasks are performed manually, it becomes drastically more time-consuming not because of the complexity involved, or level of skill required to perform these tasks, but simply because a human cannot carry these tasks out as fast as the scripts can. One would need to login to the AWS web portal, navigate to the appropriate sections of the UI, then create and tag each of these numerous resources by hand and verify that they have been created and tagged properly. Whereas, Terraform does all of this in the span of seconds, as opposed to minutes. Similarly, it would be impossible for a human to login to a cloud-based instance via RDP and perform all the server configuration by hand faster than a Puppet script. Puppet reduces the time taken for this server configuration by several magnitudes. By utilizing the full power of the AWS API through Terraform and the configuration management potential of Puppet, all actions performed in these tasks under the automated system are run instantaneously.

This is not quite the case for the tasks performed in the Domain Operations section, which is why these tasks do not perform as well under the automated system when compared to the Provisioning Tasks or the Server Configuration tasks. The Domain Operations task involves wait periods where the framework itself is not performing any operation other than polling the new instance for connectivity. The new instance firstly needs to perform its initial boot after it has been created; once reachable, the new server is renamed and added into the organisations domain. These operations require the new server to go through an initial boot of the operating system, followed by two subsequent reboots, all of which take a relatively large amount of time to perform. The time taken for the new server to boot for the first time and to reboot following the rename and addition to the domain takes up the largest portion of time in the Domain Operations task under the automated system. Once these operations are complete, a synchronization of the domain controllers within the Active Directory forest is performed, this is performed in order for the rename and addition of the new instance to register across the organisation's network. This synchronization only takes

a few seconds to perform, but verifying this synchronization takes about three minutes due the amount of domain controllers in the network, increasing the overall time spent waiting in this task. All of the above means that, the bulk of the time allocated to performing the Domain Operations task in the automated system is actually spent waiting for the instance to become available when booting from a shutdown state and waiting for the domain controllers to be synchronized. The reason why the automated system can handle this task more efficiently than the manual process is because the scripts poll the new instance for connectivity at short, regular intervals and the automation continues to run the moment the instance is available.

Outlier instances that were quicker to boot than the average brought down the average time taken to perform the Domain Operations task in the automated system, these outlier instances were polled regularly by the automated system and were configured immediately after they became available. The researcher argues that, these outliers saved minutes on each environment as the human actor who knows that it takes an average of thirty minutes for an instance to be available after it is created and another five minutes to become available in subsequent reboots will not want to waste their time trying to connect to an instance that may not currently be available, and will more than likely wait over the average thirty minutes for an instance to finish its initial boot and wait over the average five minutes for subsequent reboots to take place before attempting to carry on with their tasks.

The time spans in the boot and reboot times of these instances can be attributed to the type of operating system running on them and the amount and size of disk drives attached to them. The operating system used in the case study organisation for test environments was Windows Server 2012 R2 with four disk drives attached, which reached a combined total of 250GB of disk space. In a study performed by Mao and Humphrey on the performance of various virtual machines boot times on different IaaS platforms, Linux servers performed far faster than Windows servers of the same specifications when boot times were compared against AWS infrastructure (Mao & Humphrey, 2012). This study also outlined the increase in initial boot times when more disk space has been allocated for the instance being provisioned (Mao & Humphrey, 2012). Therefore, one could argue that, if the case study organisation had opted to use Linux-based operating systems with a single, small disk drive attached for their test environments, then the time spent waiting for the server to become available following creation and subsequent reboots could be brought to a minimum.

It is of the opinion of the researcher that, the time spent performing the Domain Operations tasks in the automated system could theoretically be brought to the same performance standard as those in the Provisioning Tasks or Server Configuration tasks if more efficient AWS resources were used in place of Windows servers with large disk drives attached. This is a hypothesis in this paper and this current body of work has no metrics to back this claim up in an industry-based setting.

The lowest performing task in the automated system when compared with the manual process is the Deployment of Codebase task. This can be attributed to the fact that the scripts that run in this section are calling existing deployment processes that are configured on the TeamCity build server. Meaning that, the limitation on efficiency in this task is benchmarked by pre-existing deployment processes, if these deployment processes take an hour when executed manually, then they will take an hour when executed through the automated process. The reason the automated framework handles this task significantly faster than the manual process is because of the volume of deployment processes that require to be executed. Another significant factor here is the specific order in which the codebase requires to be deployed. At the time of writing, the Deployment of Codebase task consists of the execution of 174 existing deployment processes. Similar to the time savings in the Provisioning Tasks and Server Configuration tasks, the Deployment of Codebase task is made faster by the automated system because a human is unable login to the TeamCity build server web portal, locate each of these deployment processes and execute them with the correct parameters as fast as the scripts in the framework do. Similar to the discussion on the Domain Operations task in the above section, in the manual process, the human user would need to execute each deployment process in a specific order and wait for it to complete before kicking off the next deployment process. The automated process handles all of this through code which polls for the completion of each deployment process before executing its dependent deployment processes. This ordering is specific to the case study organisation, as their legacy systems are tightly coupled. Their requirement was that certain database code needs to be deployed prior to the code of applications and services which utilize those databases. Taking the above in account, the time savings outlined in this comparison may not be generalizable to the wider audience of software engineering organisations, those with loosely coupled systems may not be bound by this specific constraint as, theoretically, they could deploy all of their systems codebase at one time, in no specific order. If this was the

scenario for the case study organisation, then it is the opinion of the researcher that the results would be different for this specific task and only a slight efficiency benefit would be found when performing this task through the automated framework.

Proportionally, the time taken to execute the Deployment of Codebase task in the automated process almost doubled. This is because the bulk of the time allocated to this task in the both the manual and the automated processes is spent waiting for the completion of existing deployment processes in the TeamCity build server. Therefore, the real time spent performing this task in the automated system did not rise, but the time taken for other tasks in the overall automated process dropped by several magnitudes and the time spent on the Deployment of Codebase task did not, this caused the overall proportion of time spent on this task in the automated process to increase dramatically. The same effect can be seen for the External DNS Creation task, which remained to be the only static manual task in the process. This task had to be completed manually for each new environment to be created native to AWS infrastructure, therefore, the real time spent on this task for environments built via the framework did not increase or decrease, but, because of the overall savings in time across the process as a whole, the proportion of time spent on this task in the automated timings data set increased considerably.

4.5.1.3 Review of Effort Comparison Results

The effort overhead involved in both the manual and automated processes are an invaluable metric presented in this study. It was mentioned in section 4.2.3.4 that all tasks in the manual process can be said to have an effort overhead of 100% as they are carried out in a completely manual fashion, requiring the full attention of the human performing them. The name of the automated data set would imply that all tasks were automated and the effort overhead is null, however, it was impossible to automate the External DNS Creation task via IaC tools as no API was made available by the DNS provider the case study organisation was subscribed to at the time. Therefore, this element of static manual work remained for all new environments created native to public cloud infrastructure throughout the course of the case study. The effort overhead of performing the manual External DNS Creation is added to by the Troubleshooting task. Due to the fact that the framework is error prone, a manual

troubleshooting time needed to be allotted to the majority of environments built via the automated process.

While there is still a proportional decrease of 96% in effort overhead when using the automated framework, this still accounts for almost an hour of work, on average per environment. Taking into account that 39 environments were built via the framework, and the average work week is 40 hours, one can arrive at the conclusion that almost a full working week of manual effort for a single staff member was put into the building of all of these environments in AWS via the automated framework. Opinions may vary on how acceptable this level of overhead is in terms of effort, especially when factored with the error prone nature of the framework. For instance, the case study organisation no longer creates environments through the manual method, and, at the time of writing, it is actively using the framework as it is several magnitudes faster than their previous manual method. However, in the survey results, respondents from Category B, those utilizing IaC tools and public cloud infrastructure, stated that they can provision, configure and deploy to a new test environment, on average, in the space of 30 minutes. Similarly, respondents from Category A, those utilizing IaC tools on in-house infrastructure, stated that they can perform the same operations, on average, in the span of 105 minutes.

One can see that the use of IaC tools and IaaS has allowed for an impressive proportional decrease in time and effort overheads for the case study organisation, however, cross referencing the data from the case study with the data from the survey, it is apparent that the survey respondents from Category A and Category B have a far more streamlined and efficient environment creation processes by their own implementation of IaC tools when compared with automated framework implemented in the case study organisation.

4.5.1.4 Review of Error Tendency Results

The causes of, and remediation to the errors in the framework execution were recorded by the researcher but were not presented in the Results chapter as they were encountered sporadically and followed no discernible pattern on which preventative code could have put into place to stop them from recurring. What is meant by this is that the causes of failures were either networking related or caused by the errors in

the case study organisations codebase which failed to deploy correctly to the new environment.

Networking issues encapsulates all communication or authentication problems between any of the components in the framework. These components are detailed in the Framework Architecture section at 3.2. If any one of the numerous transactions between the components in the framework failed then the framework itself would throw an error and stop. Therefore, if any internal or external server, web service or API could not be contacted at any time during the framework execution then the framework would fail, and a networking issue is said to have caused this failure. What follows are examples of these networking issues which did occur at least once on framework execution and subsequently caused the framework to throw an error.

- 1. A timeout in establishing a connection from the TeamCity server to the VCS server.*
- 2. A dropped connection from the TeamCity server to the VCS server.*
- 3. A malfunction in the Active Directory domain controller that the TeamCity server utilizes which caused an issue authenticating the TeamCity service user to the VCS server.*
- 4. The TeamCity server service stopped unexpectedly.*
- 5. A system reachability test failed in AWS when building a new instance.*
- 6. Adding the new instance to the Active Directory domain failed as there was already a server with the same name in the domain.*

The first four of above the issues caused problems to other departments within the organisation and had to be investigated and resolved by the infrastructure department which deals with all networking issues for the organisation. Once these issues were resolved, the framework was simply executed from its initial point of failure. The last two issues outlined above were remediated by destroying the instance being built, and recreating it with different parameters. This covers the networking issues, which account for only a small proportion of errors encountered in runs of framework, the main cause of errors pertained to issues with the organisations Application and Database codebase and is discussed in the following section.

When the Deployment Build is executed, all of the organisation's Application and Database code is compiled and deployed to the new instance. This step in the process is the most likely portion of the framework to throw an error, this is because the Deployment Build is dependent on 174 individual codebases to be in a stable state, and to be compatible with a new environment at the time the build is executed. The compilation of these codebases failed occasionally and a fix needed to be put in place by a developer for the framework to deploy it correctly. But application and service compilation failures do not represent the majority of deployment errors, rather, the deployment of the database code caused the majority of errors.

The researcher noted that, the bulk of changes made to the database code which caused errors to be thrown on deployment to new AWS environments were actively being deployed elsewhere. Meaning that, the changes put into the database codebase were deployed successfully to existing environments that developers were already using, but not deployed successfully to new environments that contained no previous data. The cause being that, developers were making changes to database code which contained references to existing entries in the database that their code required to be present if a deployment was to be successful. In most cases, it was found that these entries were typically manually entered into existing test environments that were actively being used. As these entries were not present on new environments, the deployment of the database code failed and the error was communicated to the developer who made the breaking change so they could resolve the issue. The fix was usually a block of code at the beginning of the failing script that checks for the data that is being referenced, if it is not found, the script creates it in order to continue without error. Once the fix was made, the framework was executed from its previous point of failure. The above description of the database deployment errors may not be clear to those with little experience in the software engineering or database administration field, but to those with this experience, the above should read as a standard software development issue which is regularly encountered in the field, especially when multiple and parallel streams of development are taking place across large applications and databases.

In summary, the overall cause of errors encountered by the framework were not to do with the code the framework is made from or issues with the framework itself. The errors were mainly pertaining to the infrastructure the framework is built on and communicates with, alongside the sheer volume of external code that it compiles and

deploys to new environments. Whether or not these results are generalizable to other organisations depends entirely on the volume and stability of the codebase for their specific systems.

4.5.2 Review of Secondary Experiment Results

The results of the secondary experiments carried out as part of this thesis are presented in sections 4.3.2 and 4.3.4. The results of these experiments demonstrate the difference in execution times and possible efficiency benefits and drawbacks when parameters supplied to the framework are modified in a controlled manner.

4.5.2.1 Review of Instance Type Experiment Results

The execution timings of the framework vary according to the compute power of the environment it is instructed to create. The expected result from this experiment was that the instances that are allocated more compute power will be provisioned, configured and deployed to faster than instances with less allocated compute power, this expectation was proven by this experiment. Figure 31 illustrates this comparison in the context of the process as a whole, it shows that environments of the t2.medium instance type takes by far the longest to create via the framework. Environments of the baseline instance type, the t2.large, follow closely behind environments of the t2.medium instance type, being created 3.6% faster. An interesting point to note here is that environments of the t2.xlarge instance type are built 23.5% faster than their t2.large equivalents, a great deal faster than the difference between the t2.medium and t2.large instance types. While this proves that environments of higher compute power are built faster by the framework, this large gap in execution times between instance types merits review and discussion. Figure 32 shows the execution timings for each build in the framework by instance type. The main gains in efficiency stem from the Deployment of Codebase task. From examining the build logs for this task across each framework execution in this experiment, the researcher observed that the 79 services that are deployed and installed as part of this task installed and were able to start far faster on environments set to the t2.xlarge instance type. The 60 databases that were built against the t2.xlarge environments also built significantly faster than the t2.large and t2.medium equivalents. This non-linear discrepancy in timing data may be

attributed to the type of resource the framework consumes when it is executing and t2.xlarge instance type having more of that resource than the other two instance types in this experiment, this is covered in the section that follows.

The instance types chosen for this experiment all belong to the t2 instance type family as the baseline instance type for the Creation/Recreation experiments was the t2.large instance type, it was desired to choose instances from the same family of lower and higher compute power for this specific experiment in order to ensure uniformity in the results throughout each experiment. Take into account that the compute power of instances in AWS are offered at pre-defined specifications, AWS offer no service whereby instances of client defined compute power can be created (Amazon, 2017). Referring back to Table 11, one can see that the differences in compute power between these instance types do not follow a completely linear pattern, the RAM assigned to each instance type increases in regular increments, beginning at 4GB for the t2.medium, the t2.large instance type has twice the amount of RAM installed on it, totalling at 8GB of RAM. This figure is doubled again for the t2.xlarge instance type which has 16GB for the t2.xlarge. However, the amount of vCPUs allocated the instance types chosen for this experiment do not increment in the same fashion. The t2.medium and t2.large instance types both have two vCPUs allocated to them, whereas the t2.xlarge instance type has four vCPUs allocated to it. This implies that RAM may not be as important as a factor in framework efficiency as processing power brought about through vCPU allocation. The results are skewed between instance type comparison as a result of this as the t2.medium and t2.large instance type have the same amount of vCPUs allocated to them. The reason the t2.medium instance type was included in this experiment is because it is the only instance type in the t2 family with 4GB of RAM allocated to it. Which is half the RAM as the baseline t2.large instance type. The t2.medium instance type was the only available instance type which was closest to half the compute power of the t2.large instance type at the time (Amazon, 2017). The results of this experiment indicate that the increased amount of vCPUs allocated to the t2.xlarge instance type are responsible for the discrepancy in framework execution timing between the instance types in this experiment.

The Instance Type experiment demonstrates how the framework behaves when the instance type variable is modified. The results demonstrate how environments of higher compute power can be provisioned, configured and deployed to faster than environments of lower power. These results also demonstrate the importance of the

vCPU specification when building environments via the framework, allowing environments to be created far faster relative to the amount of vCPUs dedicated to the instance.

4.5.2.2 Review of Storage Capacity Experiment Results

The execution timings of the framework differ when the storage allocated to the environment being provisioned is modified. It was expected from this experiment that the framework would be capable of provisioning, configuring and deploying to environments with less storage capacity allocated to them than environments with higher storage capacity allocated to them. This result was expected in part from results obtained from Mao and Humphrey's study which indicate that AWS EC2 instances with less storage attached boot faster than instances with more storage attached (Mao & Humphrey, 2012). The results plotted in Figure 33 do show that instances built from the Low Capacity AMI are completed far faster than AMIs with higher storage capacity associated with them. However, they also show a non-linear pattern whereby instances built from the Baseline AMI appear to take the most amount of time while instances built from the High Capacity AMI take range in the middle of the Low Capacity AMI and High Capacity AMI.

From examining the results plotted in Figure 34, one can see that the bulk of the Provisioning Tasks, Domain Operations and Troubleshooting times for all three AMIs in this experiment are relatively the same when variables such as the shared tenancy of the TeamCity server and network latency are taken into account. The Server Configuration tasks timings increment according to how much storage is allocated to the instance, starting at 60.38 minutes for the Low Capacity AMI, incrementing by 8.95% to reach 65.78 minutes for the Baseline AMI, then incrementing a further 14.34% to 75.23 minutes for the High Capacity AMI. The difference in timings for this specific task pales in comparison to the difference in the Deployment of Codebase task. The Deployment of Codebase task took the shortest amount of time for the Low Capacity AMI, totalling at 182.20 minutes, this increased by 37.97% to reach 251.39 minutes for the Baseline AMI, from there, it decreased by 24.99% at 188.55 minutes for the High Storage AMI. Taking into account the results of the Instance Type experiment, where the timings for the environment being built by the framework decreased in time depending on how much compute power was allocated to the

instance it was building, one would expect to either see a similar pattern in the this experiment or no difference at all considering each instance was of equal compute power at the t2.large specification.

The results of this experiment were unexpected as the researcher had full control over storage allocated to each AMI. This experiment is unlike the Instance Type experiment where explainable discrepancies in timing data arose from the non-customisable nature of instance compute power specification in AWS. The storage type used in each of the framework executions in the Storage Capacity experiment was identical, all storage devices were AWS PVDISK SCSI hard disk drives. The storage allocated to each AMI was modified in a controlled manner as described in section 4.3.3. The causes of the non-linear distribution of average timings in this experiment are not known, specific sets of data for each of the framework executions in this experiment were analysed yet no concrete cause could be found.

4.5.3 Review of Survey Results

The results of the survey questionnaire have been cross referenced in the above sections to corroborate the Creation/Recreation experiment results, but have not yet been discussed individually. This section details a review and discussion of the survey results in order to provide a meaningful interpretation of them. For the sake of clarity, a brief description of the respondent categories follows as these categories are referenced several times in the section that follows:

- *Category A - Respondents using IaC tools on in-house infrastructure*
- *Category B - Respondents using IaC tools on public cloud infrastructure*
- *Category C - Respondents not using IaC tools on in-house infrastructure*
- *Category D - Respondents not using IaC tools on public cloud infrastructure*

The survey results are presented in section 4.4, the main finding outlined in this section is that Category B respondents have, on average, the most efficient environment creation process when compared with all other categories of respondents. One can see this data visualized in Figure 35. This chart also shows that Category A respondents also have a relatively short environment creation process, but is still over

three times as long as those from Category B. However, these results were expected, and they do not refute any preconceived notions about the efficiency capabilities of IaC running on in-house infrastructure and IaC coupled with IaaS. A result that was not expected was discrepancy in timing data retrieved from Category C and Category D respondents. From reading the Background and Literature Review chapter and reviewing the results from the Recreation/Creation experiment, one would expect that Category D respondents would have a far more efficient environment creation process than Category C respondents. As using in-house server infrastructure with no automated IaC tools would indicate that these respondents should have the longest environment creation process of all respondents. Whereas, the survey results reveal that Category C respondents have the longest environment creation process times than all other respondent categories. The raw data obtained from the survey itself does not provide any discernible pattern of variables that could be used to explain this unexpected result.

As previously stated in section 4.2.4.2, each respondent entered the approximate timespan their test environments are actively used for before being destroyed. The majority of Category A and Category B respondents stated that their environments are used for a maximum of one year before being destroyed, whereas, the majority of Category C and Category D respondents stated that their environments are active for at least a year before being destroyed. This metric merits reiteration and discussion as it pertains to an issue previously described in section 2.4 in the Background and Literature Review chapter. The longer an environment is left active, the more it changes and diverges from how it was originally built. An issue known as configuration drift; the ability to confidently destroy and efficiently create environments from IaC scripts ensures that environments are always in a uniform and reproducible state and can stop the effects of configuration drift (Morris, 2016). A suggestion from this data is that the use of IaC tools allows for the adopting organisation to prevent the issue of configuration drift from growing to unmanageable levels by regularly destroying their existing environments and creating new environments in their place. While the respondents were never asked about configuration drift or its symptoms in their test environments, the results imply that Category A and Category B respondents destroy their environments so regularly that any significant configuration drift is not allowed to occur. By cross referencing the timings provided by each respondent category with the average lifespan of their

environments, it can be implied that the process of creating new environments from a known state is a relatively common and quick task to complete for respondents using IaC tools, and more so for those using a combination of IaC tools and IaaS. While the opposite can be suggested for respondents not using IaC tools, who have a significantly more time consuming environment creation process. These efficiency comparisons have mainly been drawn between timing data thus far, and have neglected to mention the amount of staff members involved.

The amount of staff members involved in the environment creation process directly impacts the cost of the entire process and the involved staff members work. This data for each respondent category is presented in Figure 36. From this chart, one can see that respondents utilising a combination of IaC tools and IaaS involve, on average, a single staff member in their environment creation process, while all other categories involve at least two, and at most three. The following hypothetical scenarios aim to provide the reader with a proper grasp on how the use of IaC tools and the amount of staff members involved in the process impacts the overall cost of the process:

- *Scenario A: A single staff member executes interlinked IaC scripts that run for four hours in order to provision, configure and deploy to an environment.*
- *Scenario B: Two staff members manually carry out tasks simultaneously that take four hours to provision, configure and deploy to an environment.*

In Scenario A, the effort overhead and cost overhead in terms of work hours is minimal, as the staff member is just executing an automated process and carrying on with their other work while the job is executing. The only real work the single staff member needs to perform is monitoring for, and possibly troubleshooting, errors that may occur during the execution of the IaC scripts.

However, in Scenario B, the absence of IaC automation causes the effort and cost overhead for the entire process to become several times that of Scenario A, as the two staff members involved are required to dedicate four hours of their time to creating the environment while ignoring all other work. Essentially, Scenario B is taking a full working day in terms of work hours, as two staff members are required to put in four hours of work each and the time taken in work hours is twice what it would be if a single staff member was performing these tasks. While Scenario A can take as little as a few minutes of real work hours to perform, assuming no errors occur in the

execution of the IaC scripts. These are purely hypothetical scenarios, and offer only anecdotal evidence based on the researcher's own experience, this current body of work has no metrics to prove how these specific types of situations occur in practice.

However, the categorized and aggregated data obtained from the survey does reveal clear patterns. One conclusion that can be derived from the results is that the combined use of IaC tools and IaaS is associated with a highly efficiency environment creation process. One can also see from the survey results that the use of IaC tools on in-house infrastructure is also associated with a slightly less efficient environment creation process than those using a combination of IaC and IaaS. The last inference from the survey data is that, regardless of the use of IaaS or in-house infrastructure, those who do not use IaC tend to have the most time consuming environment creation process.

4.6 Limitations

This thesis is focused on the relationship between the use of public cloud computing and associated automation technologies, namely IaC and configuration management tools. Technologies outside of the remit outlined above are out of the scope of any conclusions to be derived from this research. Results and conclusions that arise from the undertaking of industry-based research are inherently only truly applicable to the specific context in which that research takes place (Costely & Armsby, 2007). Taking the above into account, one of the most important limitations of the results from the case study, implementation of the automated framework within the case study organisation and subsequent experiments are their potential lack of external validity. Meaning that, the case study, implementation and experiments all took place within a specific organisation, any conclusions derived from these results are specific to that organisation and may only have limited viability to the wider community, the researcher recommends that others should not flippantly use these results and conclusions to generalise the wider community as a whole.

The case study, framework implementation and experiments all took place within a small to medium sized software solutions enterprise based in the Republic of Ireland, this organisation had been active for twenty years at the time of writing. Although the physical location of the organisation may be of little importance in this section, it

does merit inclusion here if the results of this research are put under severe scrutiny. The size and age of the organisation should be important limiting factors when examining the conclusions from this research as both the size and age of an organisation are indicators of that organisations likelihood to support their own legacy systems. These legacy systems may be built upon monolithic architecture and be comprised of deprecated technologies, these types of systems have been documented as obstacles when moving to the cloud (Menychtas, et al., 2013). This is opposed to a recently founded organisation with modern and versatile systems that may have been created with modern platforms such as the cloud in mind, therefore migration of these newer systems may be a very easy task.

Across all experiments conducted as part of this thesis, the uncontrolled variables described in section 4.1.4.2 should also be considered as being limitations. Variables such as network latency and shared tenancy of system hardware and software were uncontrollable in this thesis as the research itself took place in an industry setting as opposed to a hypothetical laboratory scenario. These variables were unmeasurable and must be taken into account when interpreting the results of the experiments. While these variables may have had an impact on the final results set for each experiment carried out as part of this thesis, it should also be mentioned that the value of industry based research and the results obtained from it can show a valuable, real-world results that a would be impossible to simulate in devised scenarios (Costely & Armsby, 2007).

A limitation specific to the Creation/Recreation experiment is the manual timings dataset which was derived from the semi-structured interviews with staff members. The timings retrieved from the interviewees were taken during each interview, requesting this level of detail for each specific task from each interviewee at a single time introduces the possibility of error on the interviewee's part. It should be mentioned here that each timing is an estimate from a single source who was put in an interview situation with very little prior knowledge of the questions that were to be asked. That being said, all interviewees were given an open invitation to make further contact with the researcher if they felt that they had any corrections to the answers they gave in their respective interviews, no follow up interactions between interviewees and the researcher ever occurred, so one must assume that the information derived from the interviews is correct.

There are also limitations specific to the secondary experiments carried out as part of this thesis. In the Instance Type experiment, it was desired to test the framework under conditions where it built environments under the following specifications.

1. *The same compute power used in the Creation/Recreation experiment which acts as a baseline.*
2. *Half the compute power of the baseline*
3. *Double the compute power of the baseline*

It was planned that the above would provide a linearly decreasing scale of timing data starting. The highest being obtained from environments built with half the compute power of the baseline, the mid-point being the baseline compute power specification and the lowest timing data from environments built from the double the compute power of the baseline. However, for reasons explained in sections 4.3.1 and 4.5.2.1, it was not possible to select compute power specifications for environments that entirely satisfied the above requirements and the researcher was forced to use instance types which increased non-linearly in allocated vCPUs. In turn, this choice caused the results to be skewed in favour of the instance type with the most vCPUs allocated to it. While this should be included as a limitation, this experiment did prove that the expected result would be found, in that, the more compute power allocated to an environment, the faster the framework can build it, it also provided an interesting explanation for the results obtained from carrying out this experiment. This result may provide a useful base of information to those wishing to carry out performance testing across instance types in AWS.

A limitation specific to the Storage Capacity Experiment resides in its results and the failure on the researchers part to explain the phenomena that occurred which skewed the results sets presented in section 4.3.4. As with the above limitation for the Instance Type experiment, the results from the Storage Capacity experiment did prove that the expected result would be found, in that, the framework can build environments with less storage allocated to them faster than environments with more storage allocated to them. However, the non-linear fluctuations in timing data obtained from the framework building the different AMIs used in this experiment are its main limitation. Environments from both the Low Capacity AMIs and High Capacity AMIs were built far faster than the Baseline AMI. This result was not expected and due to time and

resource constraints, the results could not be investigated to any real scientific degree, leaving room for future research in this area.

Chapter 5. Conclusions and Future Work

This final chapter aims to provide the reader with a discussion of the thesis, recommendations for future research and ends with general conclusions reached by this thesis.

5.1 Discussion

The aims of this research were to develop and implement an automated framework that allowed for a SME to migrate their colocation-based IT infrastructure to AWS's IaaS platform and gather metrics pertaining to the efficiency benefits of implementing such a framework in an industry-based setting. It was also planned to prove the generalisability of these efficiency benefits in the context of the wider audience of SMEs.

These aims have been achieved in this thesis. When discussing the design and of the framework itself, the state of the art in cloud migration frameworks in the Background and Literature Review chapter should be mentioned as the basis for what has already been created in the field and what the gaps of knowledge were present at the time. The final deciding factors for the design of the framework were dependant on the results from the industry-based case study, as the framework was not only required to satisfy theoretical baselines of the academic world, but also provide real-world functional value to an enterprise. The aim to design and develop the framework has been realised by the above.

The case study also allowed for the gathering of timings pertaining to the organisations previous colocation-based environment creation process. The Creation/Recreation experiment consisted of the organisations internal testing environments being recreated on AWS's IaaS by the framework, this experiment allowed for the gathering of timing data relating to the frameworks execution time. These two sets of timing data were compared to reveal that the overall process of building IT environments on the public cloud via the framework is 360% faster than using the SME's previous environment creation process. This satisfies the aim to

gather metrics pertaining to the efficiency benefits of implementing such an automated framework.

In order to test the boundaries of the framework as much as possible, the secondary experiments were carried out. These experiments test the framework under different operating modes in order to demonstrate how the resources made available through IaaS can affect the environment creation process under the framework. These resources were compute power and storage capacity, the results of these experiments show how an increase in compute power can have a clear effect on how quickly an environment can be created and how a decrease in compute power can have a negative effect, slowing down the environment creation process time. The storage capacity test demonstrates how a lower amount of storage allocated to an environment causes the environment to be built faster via the framework, whereas increasing the storage allocated to the environment has the opposite effect, causing longer creation times.

One of the main limitations from the results of the case study and subsequent experiments are their lack of external validity. The aim to prove the generalisability of the above was achieved by performing the industry-based survey questionnaire. The results of which corroborate the results obtained from the case study and experiment in the context of the wider audience of software engineering SMEs based in the Republic of Ireland. By comparing environment creation times between respondents utilising public cloud or infrastructure as code technologies to respondents not utilising either, the results show that public cloud and infrastructure as code tools have staggering efficiency benefits to those who use them.

5.2 Conclusions and Research Implications

From a high level examination of the results as a whole, one can arrive at the conclusion that the utilization of automated IaC tools, coupled with IaaS allow for a dramatically more efficient IT environment creation process than that of a manual, in-house equivalent. The case study and experimental results answer the research question adequately in the context of the specific organisation the case study and experiments were carried out in. While the survey results corroborate these findings by demonstrating how other organisations using automated IaC tools and IaaS have a

highly efficient environment creation process. The survey results can be used to answer the research question outside of the context of the case study organisation.

The results presented in this thesis are novel as there has never been such a specific type of study performed in the field to date that prove the real-world efficiency capabilities of implementing an automated system comprising of IaC tools and public cloud infrastructure. The researcher argues that the previously cited studies by Jamshidi et al., Hay et al., Mateescu et al. and Khajeh-Hosseini et al. can be used to highlight the lack of research in this particular area and the need for an industry-based study outlining the design and proving the benefits of implementing public cloud and infrastructure as code technologies in practise (Jamshidi, et al., 2013) (Hay, 2011) (Mateescu, et al., 2014) (Khajeh-Hosseini, et al., 2010).

Throughout the Literature Review chapter of this document, several surveys and articles are cited that make claim to the benefits of IaC and cloud computing, efficiency in the area of provisioning new IT infrastructure being key among them, the results presented in this study support the results of other studies cited in this thesis, at least in the area of efficiency (RightScale, 2014) (RightScale, 2015) (Forrester, 2015) (PuppetLabs, 2015) (Hashicorp, 2015) (Morris, 2016).

5.3 Recommendations for Future Research

Research carried out as part of this thesis included the development and implementation of an automated framework comprised of IaC and automation software which recreated existing, and created new IT environments on Amazon's public cloud infrastructure. However, this source code was created specifically for the case study organisation's adoption and continued use of Amazon's public infrastructure, generalising this code so it will function for other organisations, and building environments on other CSP's public cloud infrastructure was beyond the scope of this project. Also, at the time of writing, this organisation is actively using this framework and retains the rights to its source code. Therefore, the researcher's main recommendation for future research is the development, successful implementation and open-source distribution of a unified, cloud-agnostic framework which has the capabilities to migrate existing, and create new environments on any CSP's public cloud infrastructure. This primary recommendation is similar to that of

Jamshidi et al., who state that there is a requirement for an established framework based around the migration of in-house infrastructure to the IaaS platform, and that more research into this specific area is required (Jamshidi, et al., 2013). The architecture of the automated framework which was created and utilized in the experimental portion of this body of work is outlined in section 3.2 and the specific technologies used in this implementation is outlined in section 0. It is hoped that this architecture and specific technologies may be used by future researchers as a blueprint for the development of the aforementioned unified framework.

The main focus of the case study and primary experiment was an efficiency comparison in the environment creation process between a purely manual process, performed on in-house infrastructure and an automated process comprised of IaC tools performed on public cloud infrastructure. This study was the first of its kind, but was highly focussed on the area of efficiency. Taking the above into account, several questions pertaining to the potential benefits or drawbacks of implementing public cloud infrastructure still remain, there is ample room for future research in the area of comparative studies between the use of public cloud infrastructure and in-house infrastructure. For example, future research in this area may include industry-based studies targeting quantitative metrics such as the scalability potential or the monetary effects associated with the utilization of IaaS over in-house infrastructure in a real-world setting. In terms of qualitative future research, studies encompassing the effect of adopting public cloud infrastructure on staff belonging to the organisation, and clients of the organisation could potentially be carried out.

The survey questionnaire created and distributed as part of this thesis is of questionable scientific value when examined in isolation from the case study and primary experiment. The main limitations are its sample size and sample method, which may be used to dispute the generalisability of the results obtained, these were limiting factors due to time and resource constraints in this project. For future surveys, the researcher recommends that a dedicated study is performed, one which targets a specific and relatively small population. If this dedicated study were to take place, the researcher carrying it out should consider the resources required to distribute the survey to a representative base of willing respondents while utilising a more adequate sampling method than the snowball sampling method chosen in this body of work. Nevertheless, in future surveys pertaining to the efficiency capabilities

of public cloud infrastructure coupled with IaC tools, the results presented in this thesis can act as a benchmark for expected results.

It is the opinion of the researcher that, all of the above recommendations for future research are viable studies, which would potentially influence the collective understanding of the relatively under researched field.

BIBLIOGRAPHY

Alexandrova, J., 2015. *PowerShell*. [Online]

Available at: <https://confluence.jetbrains.com/display/TCD9/PowerShell>

Alexandrova, J., 2016. *Build Chain*. [Online]

Available at: <https://confluence.jetbrains.com/display/TCD10/Build+Chain>

Alexandrova, J., 2016. *Defining and Using Build Parameters in Build Configuration*. [Online]

Available at:

<https://confluence.jetbrains.com/display/TCD9/Defining+and+Using+Build+Parameters+in+Build+Configuration>

Amaral, J. N., 2011. *About Computing Science Research Methodology*, Alberta: University of Alberta.

Amazon, 2002. *Amazon.com Launches Web Services*. [Online]

Available at: <http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle&ID=503034&highlight=>

[Accessed 30 July 2015].

Amazon, 2013. *Amazon EC2 Service Level Agreement*. [Online]

Available at: <https://aws.amazon.com/ec2/sla/>

Amazon, 2015. *Amazon*. [Online]

Available at: <https://aws.amazon.com/ec2/pricing/>

[Accessed 1 December 2015].

Amazon, 2015. *Amazon EBS Product Details*. [Online]

Available at: <https://aws.amazon.com/ebs/details/>

[Accessed 2 December 2015].

Amazon, 2015. *Amazon EC2*. [Online]

Available at: <https://aws.amazon.com/ec2/>

Amazon, 2015. *Amazon EC2 Instance Purchasing Options*. [Online]

Available at: <http://aws.amazon.com/ec2/purchasing-options/>

Amazon, 2015. *Amazon EC2 Instances*. [Online]

Available at: <https://aws.amazon.com/ec2/instance-types/>

Amazon, 2015. *Amazon EC2 Product Details*. [Online]

Available at: <https://aws.amazon.com/ec2/details/>

Amazon, 2015. *Using Cost Allocation Tags*. [Online]

Available at: <http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html>

Amazon, 2016. *Amazon S3 Pricing*. [Online]

Available at: <https://aws.amazon.com/s3/pricing/>

- Amazon, 2016. *Overview of Security Processes*, s.l.: Amazon Web Services Ltd..
- Amazon, 2017. *Adding a Hardware Virtual Private Gateway to Your VPC*. [Online]
Available at: http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_VPN.html
[Accessed 11 June 2017].
- Amazon, 2017. *Amazon EC2 Instance Types*. [Online]
Available at: <https://aws.amazon.com/ec2/instance-types/>
[Accessed 12 June 2017].
- Amazon, 2017. *Security Groups for Your VPC*. [Online]
Available at:
http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html
[Accessed 14 February 2017].
- Antony, J., 2003. Fundamentals of Design of Experiments. In: *Design of Experiments for Engineers and Scientists*. Edinburgh: Elsevier, pp. 8-16.
- Arrington, M., 2006. *Amazon: Grid Storage Web Service Launches*. [Online]
Available at: <http://techcrunch.com/2006/03/14/amazon-grid-storage-web-service-launches/>
[Accessed 21 February 2016].
- AWS, 2010. *Shrink an EBS - How ??*. [Online]
Available at: <https://forums.aws.amazon.com/message.jspa?messageID=177749>
[Accessed 12 March 2016].
- Azure, 2015. *Azure Instance Pricing Calculator*. [Online]
Available at: <https://azure.microsoft.com/en-us/pricing/calculator/>
- Barr, J., 2006. *Amazon EC2 Beta*. [Online]
Available at: https://aws.amazon.com/blogs/aws/amazon_ec2_beta/
[Accessed 2 August 2015].
- Benioff, M., 2009. Introduction. In: *Behind the Cloud*. San Francisco(California): Josset-Bass, p. 17.
- Bergmayr, A., Brunelière, H., Izquierdo, J. L. C. & J., G., 2013. *Migrating Legacy Software to the Cloud with ARTIST*. 2013 17th European Conference on Software Maintenance and Reengineering, Genova, IEEE, pp. 465-468.
- Brikman, Y., 2017. Chapter 1. Why Terraform. In: *Terraform: Up and Running*. s.l.:O'Reilly Media, Inc., pp. 5-20.
- Burgess, M., 1993. *Configuration Engine V2.0*, Oslo: cfengine.com.
- Burgess, M., 1998. *Computer Immunology*. Twelfth Systems Administration Conference (LISA '98), Boston, USENIX.
- Burgess, M., 2002. *cfengine 2.0.0*. [Online]
Available at: <https://lists.gnu.org/archive/html/info-gnu/2002-03/msg00010.html>
[Accessed 12 November 2016].

- Certification Europe, 2015. *What is ISO 27001*. [Online]
Available at: <http://certificationeurope.com/what-is-iso-27001/>
- CFEngine, 2014. *The History of CFEngine*. [Online]
Available at: <https://auth.cfengine.com/the-history-of-cfengine>
[Accessed 7 November 2015].
- Chef, 2015. *An Overview of Chef*. [Online]
Available at: https://docs.chef.io/chef_overview.html
- Cloud Security Alliance, 2011. *Security Guidance for Critical Areas of Focus in Cloud Computing V3.0*, s.l.: Cloud Security Alliance.
- Cloud Standards Customer Council, 2016. *Cloud Security Standards: What to Expect & What to Negotiate*, s.l.: Cloud Standards Customer Council.
- CloudHarmony, 2015. *CloudSquare: Service Status*. [Online]
Available at: <https://cloudharmony.com/status-1year-of-compute-and-storage-and-dns-group-by-regions-and-provider>
- Collins-Sussman, B., Fitzpatrick, B. W. & Pilato, C. M., 2011. *Version Control with Subversion*. 2nd ed. Chicago: Subversion.
- Corbató, F. J., 1963. *A Solution to Computer Bottlenecks*. [Online]
Available at: <https://www.youtube.com/watch?v=Q07PhW5sCEk&feature=related>
[Accessed 4 July 2015].
- Costely, C. & Armsby, P., 2007. Methodologies for undergraduates doing practitioner investigations at work. *Journal of Workplace Learning*, 19(3), pp. 131-145.
- Creasy, R. J., 1981. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, September.25(5).
- Cremers, R., 2012. *Everything you need to know about Windows Azure queue storage to build disconnected and reliable systems*. [Online]
Available at: <http://robbincremers.me/2012/03/07/everything-you-need-to-know-about-windows-azure-queue-storage-to-build-disconnected-and-reliable-systems-2/>
- Dadgar, A., 2014. *Automating Infrastructure Management with Terraform*. [Online]
Available at: <https://www.youtube.com/watch?v=WdV4eYZO5Ao>
[Accessed 18 October 2016].
- Davis, D. & Lowe, S. D., 2015. *2015 State of Virtualization and Storage Management - Survey Results*, s.l.: ActualTech Media.
- Dawoud, W., Takouna, I. & Meinel, C., 2010. *Infrastructure as a service security: Challenges and solutions*. Informatics and Systems (INFOS), 2010 The 7th International Conference on, Cairo, IEEE.
- Demeyer, S., 2011. *Research Methods in Computer Science*, Antwerp: University of Antwerp.
- Desmond, B., 2008. *Active Directory*. 4th ed. s.l.:O'Reilly Media.

- DevIQ, 2017. *Build Server*. [Online]
Available at: <http://deviq.com/build-server/>
- Duan, Y., Cao, Y. & Sun, X., 2015. *Various "aaS" of everything as a service*. 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Takamatsu, IEEE, pp. 1-6.
- Dyck, A., 2015. *Towards Definitions for Release Engineering and DevOps*. Florence, IEEE/ACM, p. 3.
- Etikan, I., 2016. Comparison of Snowball Sampling and Sequential Sampling Technique. *Biometrics & Biostatistics International Journal*, 3(1), pp. 1,2.
- F5 Networks, 2009. *Trends in Enterprise Virtualization Technologies*, s.l.: F5 Networks.
- FedRAMP, 2015. *FedRAMP Compliant Systems*. [Online]
Available at: <https://www.fedramp.gov/marketplace/compliant-systems/>
- FitzMacken, T., 2016. *Azure Resource Manager overview*. [Online]
Available at: <https://azure.microsoft.com/en-us/documentation/articles/resource-group-overview/>
- Forrester, 2015. *Infrastructure As Code: Fueling The Fire For Faster Application Delivery*, s.l.: Forrester.
- Frey, S. & Hasselbring, W., 2011. *Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach*. Lisbon, Portugal, Proceedings of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 155-158.
- Frost, 2015. *SSAE 16 Overview*. [Online]
Available at: <http://www.frostssae16.com/overview/>
[Accessed 23 November 2015].
- Gartner, 2015. *Gartner Says Worldwide Cloud Infrastructure-as-a-Service Spending to Grow 32.8 Percent in 2015*. [Online]
Available at: <http://www.gartner.com/newsroom/id/3055225>
- Gerla, T., 2013. *Hello Ansible!*. [Online]
Available at: <http://www.ansible.com/blog/2013/03/04/hello-ansible>
- Gibson, J., Rondeau, R., Eveleigh, D. & Tan, Q., 2012. *Benefits and challenges of three cloud computing service models*. Sao Carlos, IEEE, pp. 198 - 205.
- Google, 2006. *Google Announces Google Docs & Spreadsheets*. [Online]
Available at: http://googlepress.blogspot.ie/2006/10/google-announces-google-docs_11.html
- Google, 2015. *Adding Local SSDs*. [Online]
Available at: <https://cloud.google.com/compute/docs/disks/local-ssd>
[Accessed 12 December 2015].

- Google, 2015. *Google Cloud Platform Instance Calculator*. [Online]
Available at: <https://cloud.google.com/products/calculator/>
- Google, 2015. *Google Compute Engine Pricing*. [Online]
Available at: <https://cloud.google.com/compute/pricing>
- Google, 2015. *Google Compute Engine Service Level Agreement (SLA)*. [Online]
Available at: <https://cloud.google.com/compute/sla?hl=en>
- Google, 2015. *Machine Types*. [Online]
Available at: <https://cloud.google.com/compute/docs/machine-types>
- Google, 2015. *Networking and Firewalls*. [Online]
Available at: <https://cloud.google.com/compute/docs/networks-and-firewalls>
- Google, 2015. *What is Google Compute Engine?*. [Online]
Available at: <https://cloud.google.com/compute/docs/>
- Gosnell, D. M., 2005. Anatomy of a Web API. In: *Professional Web APIs : Google, eBay, Amazon.com, MapPoint, FedEx*. Hoboken(New Jersey): Wiley.
- Gottlieb, A., 2012. *The benefits of Colocation facilities for the Next-generation Enterprise WAN*. [Online]
Available at: <http://www.networkworld.com/article/2223058/cisco-subnet/the-benefits-of-colocation-facilities-for-the-next-generation-enterprise-wan.html>
[Accessed 1 July 2017].
- Graham, B., 2010. *Continuum Research Methods : Case Study Research Methods*. 1st ed. London: Continuum.
- Hashicorp, 2015. *Mozilla uses Terraform and Atlas by Hashicorp to embrace infrastructure as code*, s.l.: Hashicorp.
- Hashimoto, M., 2015. *Mitchell Hashimoto on Consul, Terraform, Atlas, Go as a Language for Tools* [Interview] (3 June 2015).
- Hay, B., 2011. *Storm Clouds Rising: Security Challenges for IaaS Cloud Computing*. Kauai, HI, IEEE, pp. 1 - 7.
- Hwang, J., Huang, Y. W., Vukovic, M. & Anerousis, N., 2015. *Enterprise-scale cloud migration orchestrator*. Ottawa, IEEE, pp. 1002-1007.
- IBM, 2015. *VM History and Heritage*. [Online]
Available at: <http://www.vm.ibm.com/history/>
- IDG Enterprise, 2017. *Tech Forecast Study 2017*, s.l.: IDG Enterprise.
- Instagram, 2012. *What Powers Instagram: Hundreds of Instances, Dozens of Technologies*. [Online]
Available at: <http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances>

International Telecommunication Union, 2015. *ICT Facts and Figures – The world in 2015*, Geneva: International Telecommunication Union.

Internet World Stats, 2017. *World Internet Users and 2017 Population Stats*. [Online]
Available at: <http://www.internetworldstats.com/stats.htm>
[Accessed 7 May 2017].

Irish Times, 2017. *Top 1000 - Industry: Technology*. [Online]
Available at: <http://www.top1000.ie/industries/technology>

ISO/IEC, 2014. *ISO/IEC 27018:2014*. [Online]
Available at: <https://www.iso.org/obp/ui/#iso:std:61498:en>
[Accessed 8 October 2015].

Jacob, A., 2012. *Opscode Chef State of the Union Part 1: Chef, Past and Present*. [Online]
Available at: <https://www.youtube.com/watch?v=bAWjqE5FCxl>

Jacob, S., 2009. *How to Chain TFS Builds?*. [Online]
Available at: [https://blogarchive.claritycon.com/blog/2009/08/how-to-chain-tfs-builds/SajoJacob \(Alumni\)](https://blogarchive.claritycon.com/blog/2009/08/how-to-chain-tfs-builds/SajoJacob%20Alumni)
[Accessed 24 January 2017].

Jamshidi, P., Ahmad, A. & Pahl, C., 2013. Cloud Migration Research: A Systematic Review. *Cloud Computing, IEEE Transactions on*, 1(2), pp. 142 - 157.

Jin, C., Srivastava, A. & Zhang, Z. L., 2016. *Understanding security group usage in a public IaaS cloud*. San Francisco, IEEE, pp. 1-9.

Kanies, L., 2010. *Episode 17: Luke Kanies on DevOps Cafe: History of Puppet & DevOps*. [Online]
Available at:
http://hwcdn.libsyn.com/p/2/c/8/2c844d2632f63e9b/Episode_17.mp3?c_id=2916961&expiration=1447368145&hwt=c0e68f369da6f7ee70129f59aff91ee5
[Accessed 12 November 2015].

Kaufman, L., 2009. Data security in the world of cloud computing. *Security & Privacy, IEEE*, 7(4), p. 61.

Khajeh-Hosseini, A., Greenwood, D. & Sommerville, I., 2010. *Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS*. Miami, FL, IEEE, pp. 450 - 457.

Khan, N. & Al-Yasiri, A., 2015. Framework for Cloud Adoption: A Roadmap for SMEs to Cloud Migration. *International Journal on Cloud Computing: Services and Architecture*, 5(5/6), pp. 01-15.

Khan, S. U., 2014. Elements of Cloud Adoption. *Cloud Computing, IEEE*, May, 1(1), pp. 71 - 73.

Kitchenham, B. et al., 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8), pp. 721-734.

- Knorr, E., 2016. *2016: The year we see the real cloud leaders emerge*. [Online]
Available at: <http://www.infoworld.com/article/3018046/cloud-computing/2016-the-year-we-see-the-real-cloud-leaders-emerge.html>
[Accessed 1 June 2017].
- Li, Z., 2013. The Cloud's Cloudy Moment: A Systematic Survey of Public Cloud Service Outage. *International Journal of Cloud Computing and Services Science*, 2(5), p. 321 – 331.
- Lunden, I., 2014. *Puppet Labs Raises \$40M More To Take Its IT Automation Business Global*. [Online]
Available at: <http://techcrunch.com/2014/06/19/puppet-labs-raises-40m-to-take-its-it-automation-business-global/>
[Accessed 17 August 2017].
- Maguire, J., 2015. *Cloud Computing Market Leaders*. [Online]
Available at: <http://www.webopedia.com/Blog/cloud-computing-market-leaders-2015.html>
[Accessed 18 June 2016].
- Manvi, S. S. & Krishna Shyam, G., 2014. Resource Management for Infrastructure as a Service (IaaS) in Cloud Computing: A Survey. *Journal of Network and Computer Applications*, Volume 41, pp. 425-438.
- Mao, M. & Humphrey, M., 2012. *A Performance Study on the VM Startup Time in the Cloud*. Honolulu, IEEE.
- Mateescu, G., Vlădescu, M. & Sgârciu, V., 2014. *Auditing cloud computing migration*. Timisoara, IEEE, pp. 263 - 268.
- Matt Asay , 2009. *Reductive Labs nails \$2 million in funding--Q&A*. [Online]
Available at: <http://www.cnet.com/news/reductive-labs-nails-2-million-in-funding-q-a/>
- McCarthy, J., 1992. Reminiscences on the History of Time Sharing. *IEEE Annals of the History of Computing*, 14(1), pp. 19-24.
- Mell, P., 2011. *The NIST Definition of Cloud Computing*, Gaithersburg: National Institute of Standards and Technology.
- Melymuka, V., 2012. Enhanced Techniques. In: *TeamCity 7 Continuous Integration Essentials*. Birmingham: Packt Publishing Ltd, pp. 88 - 92.
- Melymuka, V., 2012. Getting Started with TeamCity. In: *TeamCity 7 Continuous Integration Essentials*. Birmingham: Packt Publishing Ltd, pp. 10-13.
- Menychtas, A., Santzaridou, C. & Kousiouris, G., 2013. *ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud*. Timisoara, IEEE, pp. 424-431.
- Michel, J. P., 2013. *Web Service APIs and Libraries*. Chicago: ALA Editions.
- Microsoft, 2015. *12-Month Prepay Offer*. [Online]
Available at: <https://azure.microsoft.com/en-us/offers/ms-azr-0026p/>
[Accessed 12 December 2015].

- Microsoft, 2015. *Azure Resource Manager overview*. [Online]
Available at: <https://azure.microsoft.com/en-us/documentation/articles/resource-group-overview/>
- Microsoft, 2015. *FAQ About Azure for Research*. [Online]
Available at: <http://research.microsoft.com/en-us/projects/azure/faq.aspx>
- Microsoft, 2015. *SLA for Virtual Machines*. [Online]
Available at: https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_0/
[Accessed 12 November 2015].
- Microsoft, 2015. *Virtual Machines*. [Online]
Available at: <https://azure.microsoft.com/en-us/services/virtual-machines/>
- Microsoft, 2015. *Virtual Machines Pricing*. [Online]
Available at: <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/>
- Microsoft, 2015. *Virtual Network*. [Online]
Available at: <https://azure.microsoft.com/en-us/services/virtual-network/>
- Microsoft, 2017. *Active Directory Cmdlets in Windows PowerShell*. [Online]
Available at: <https://technet.microsoft.com/en-us/library/ee617195.aspx>
[Accessed 12 January 2017].
- Microsoft, 2017. *Organizational Units*. [Online]
Available at: <https://technet.microsoft.com/en-us/library/cc978003.aspx>
[Accessed 14 February 2017].
- Microsoft, 2017. *Sysprep (System Preparation) Overview*. [Online]
Available at: <https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/sysprep--system-preparation--overview>
[Accessed 31 December 2017].
- Miller, R., 2011. *Outage in Dublin Knocks Amazon, Microsoft Data Centers Offline*. [Online]
Available at: <http://www.datacenterknowledge.com/archives/2011/08/07/lightning-in-dublin-knocks-amazon-microsoft-data-centers-offline/>
[Accessed 27 September 2015].
- Morris, K., 2016. *Infrastructure as Code*. Sebastopol: O'Reilly Media, Inc..
- National Institute of Standards and Technology, 2001. *SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES*, Gaithersburg: National Institute of Standards and Technology.
- Nelson-Smith, S., 2013. Chapter 1. The Philosophy of Test-Driven Infrastructure. In: *Test-driven Infrastructure with Chef (2nd Edition)*. Beijing: O'Reilly Media, Inc..
- Pahl, C., Xiong, H. & Walshe, R., 2013. *A comparison of on-premise to cloud migration approaches*. Malaga, European Conference on Service-Oriented and Cloud Computing, pp. 11-13.
- PC Magazine, 2015. *Definition of: Internet explosion*. [Online]
Available at: <http://www.pcmag.com/encyclopedia/term/45221/internet-explosion>

PCI Security Standards Council, 2015. *PCI Security Standards: Getting Started*. [Online]
Available at: https://www.pcisecuritystandards.org/security_standards/getting_started.php

PuppetLabs, 2015. *Continuous Integration at Infusionsoft with Puppet Enterprise*. [Online]
Available at: <https://puppetlabs.com/case-studies/continuous-integration-at-infusionsoft-with-puppet-enterprise>

PuppetLabs, 2015. *Frequently Asked Questions*. [Online]
Available at: <http://docs.puppetlabs.com/guides/faq.html#whats-special-about-puppets-model-driven-design>

PuppetLabs, 2015. *Frequently Asked Questions*. [Online]
Available at: <http://docs.puppetlabs.com/guides/faq.html#what-is-puppet>

PuppetLabs, 2015. *stdlib*. [Online]
Available at: <https://forge.puppetlabs.com/puppetlabs/stdlib>

Reichard, K., 1998. *Web Hosting and Colocation: A Helping Hand For Business*, New York: Mixed Media.

Revyakina, E., 2016. *Subversion*. [Online]
Available at: <https://confluence.jetbrains.com/display/TCD9/Subversion>
[Accessed 9 April 2016].

Rex, 2015. *(R)ex Deployment & Configuration Management*. [Online]
Available at: <http://www.rexify.org/>
[Accessed 24 December 2015].

RightScale, 2014. *State of the Cloud Report*, s.l.: RightScale.

RightScale, 2015. *State of the Cloud Report*, s.l.: RightScale.

Ring, J. E., 2013. *How to Install Windows PowerShell 4.0*. [Online]
Available at: <https://social.technet.microsoft.com/wiki/contents/articles/21016-how-to-install-windows-powershell-4-0.aspx>
[Accessed 5 August 2017].

Robbins, J., 2009. *Announcing Chef*. [Online]
Available at: <https://www.chef.io/blog/2009/01/15/announcing-chef/>
[Accessed 18 September 2017].

Rudder, 2015. *FAQ*. [Online]
Available at: http://www.rudder-project.org/site/documentation/faq/#about_rudder

Sabiri, K., Benabbou, F., Moutachaouik, H. & Hain, M., 2015. *Towards a cloud migration framework*. Marrakech, IEEE, pp. 1-6.

Sadiku, M. N. O., Musa, S. M. & Momoh, O. D., 2014. Cloud Computing: Opportunities and Challenges. *IEEE Potentials*, 33(1), pp. 34-36.

SaltStack, 2015. [Online]
Available at: <http://saltstack.com/enterprise/>

Schonlau, M., 2002. *Conducting Research Surveys via E-mail and the Web*. Santa Monica: US: RAND Corporation.

Schonlau, M., 2002. *Conducting Research Surveys via E-mail and the Web*. Santa Monica(California): RAND Corporation.

Solomon, D. A., 1998. The Windows NT kernel architecture. *Computer*, 31(10), pp. 40 - 47.

Somwanshi, S., 2015. *Choosing the Right Tool to Provision AWS Infrastructure*. [Online] Available at: <https://www.thoughtworks.com/insights/blog/choosing-right-tool-provision-aws-infrastructure> [Accessed 20 June 2016].

Stanek, W., 2014. Chapter 1. Windows PowerShell Essentials. In: *Windows PowerShell : The Personal Trainer for Windows PowerShell 3.0 and Windows PowerShell 4.0*. Seattle: Stanek & Associates, pp. 23-30.

Talaat, S., 2013. PowerShell 3.0 Advanced Administration Handbook. In: *Managing Active Directory with PowerShell*. Olton: GB: Packt Publishing, pp. 173-188.

Terraform, 2016. *Providers*. [Online] Available at: <https://www.terraform.io/docs/providers/>

VanRoekel, S., 2011. *Security Authorization of Information Systems in Cloud Computing*, Washington: Chief Information Officers Council.

Vaquero, L., Rodero-Merino, L. & Morán, D., 2011. Locking the sky: a survey on IaaS cloud security. *Computing*, 91(1), pp. 93-118.

Vu, Q. H. & Asal, R., 2012. *Legacy Application Migration to the Cloud: Practicability and Methodology*. Honolulu, 2012 IEEE Eighth World Congress on Services, pp. 270-277.

Walden, D., 2012. *The Compatible Time Sharing System (1961–1973) Fiftieth Anniversary*, Washington DC: IEEE Computer Society.

WeRSM, 2013. *The Complete History of Instagram*. [Online] Available at: <http://wersm.com/the-complete-history-of-instagram/>

Whetstone, K., 2016. *Parameterized Trigger Plugin*. [Online] Available at: <https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Trigger+Plugin> [Accessed 12 December 2016].

Zainal, Z., 2007. Case study as a research method. *Jurnal Kemanusiaan*, pp. 1-6.

Appendices

Appendix A. Interview with infrastructure member

- *This interview took place on the 13/11/15, from 11:05 to 11:18*

Researcher: Good morning, [name omitted], thanks for taking the time out of your day for this interview. I've booked this room for 30 minutes but I know you are busy with work so I only plan on it lasting about 10 minutes or so.

Infrastructure member: No problem, I'm happy to help.

Researcher: Good to hear, so I'm hoping to cover the manual steps involved in the test environment creation process from an infrastructure perspective in this interview. You are already aware that I'm conducting this research as part of the master's degree I am studying for so I'm going to try to get as much information about the process from you as possible. I'm aware of a few of the steps involved in your part of the process, I am going to list them out and I want you give me a rough estimate on how long it has taken you to perform them in the past. Be sure to stop me at any stage if you think that I've missed anything or if I've made a mistake in the steps you take.

Infrastructure member: Alright.

Researcher: So the first step is to the create the actual virtual machine in VMWare, so I would assume that you would need to choose how much compute power the machine will have along with storage and so on.

Infrastructure member: Well, we start out finding the next free IP address in the subnet the machine should reside in then we just take an existing test environment that is up to date and signed off on by QA and clone that. So we would get the same CPU and RAM but would have to assign storage manually for the drives attached to the new machine. All of these are small tasks.

Researcher: Do you have an estimate in minutes or hours for each of these tasks?

Infrastructure member: Let's say all of those tasks take a maximum of 30 minutes. There's no point in splitting them because some of them could take a few seconds.

Researcher: So at this stage the machine is up and running, so you would need to document it.

Infrastructure member: We do documentation, but usually not until later on in the process. Once we are done with all other tasks and hand over the machine to the release management team then we would add the environment document to a Visio diagram along with all details like machine name, spec and storage. We also add it to list on SharePoint of the machine name and IP and hostname of the box that we would have to update so anyone can RDP to the box and access sites externally without needing to ask us for the details.

Researcher: Is there any other kind of documentation done for these new machines?

Infrastructure member: Not really, we always provide the details of the machine to whoever requested it because the diagram is internal to infrastructure and they might not have a link to the SharePoint page. It's just so they know what the machine name is and have links to the sites on the machine once they are setup.

Researcher: How long do you think that whole documentation process takes you?

Infrastructure member: Not a whole pile of time really, all the information is already there, it's just putting it down on paper outside of VMWare. I would say it could take another 30 minutes.

Researcher: Alright, so that's provisioning and documentation out of the way, I'm moving onto the domain and DNS operations now as long as you're confident that we haven't missed any task.

Infrastructure member: I am.

Researcher: Good, so next you'd need to add the new machine to the domain and create the DNS entries, do you have an estimate on how long this takes?

Infrastructure member: Sure, but you have missed a few steps, we first need to Sysprep the machine and rename so we can add it to active directory because it's a clone and its name was copied with it, so there is already an entry in active directory from the machine it was cloned from. We'd also need to find the correct OU [organisational unit] in the domain to add the new machine to. Then we need to install all Windows updates on the new machine and finally reinstall SCCM [System Centre Configuration Manager] as a GUID [Globally Unique Identifier] in the local registry

needs to update to let the network know that it's a new machine as opposed to the machine it was cloned from. After this I reboot the machine for the last time and make sure all services are up and running.

Researcher: OK, I'll mark that down in my notes. Let's just break up the domain operations from the other configuration you described. How long do the domain specific tasks you've described take?

Infrastructure member: It depends on the size of the drives and how powerful the machine is, the server needs to restart a few times during this so I would say it took me half a day for everything the last time I did it for [name omitted]'s test environment.

Researcher: OK, so would be about 3 hours for the domain operations and another 2 for the configuration of the Windows updates, reinstalling SCCM and verifying the services?

Infrastructure member: Yes, I would say that's accurate.

Researcher: Alright, so DNS and any other networking would come next, can you break these down into steps and tell me how long each one would take?

Infrastructure member: Yeah, we have a standard set of internal and external DNS to add for each of these environments. Internal takes about a half an hour, then we set up each of the external DNS entries manually and it's a tedious process. Last time, [name omitted]'s environment took me two hours of copying and pasting into a web form to get it all setup.

Researcher: Alright, so internal is half an hour and external takes another two hours?

Infrastructure member: Yeah, that sounds about right, sometimes it can take longer if a production issue occurs or I get called into meetings.

Researcher: So is there anything else you can think of that we haven't covered here today? I know you've mentioned that you need to make sure the services are up and running earlier, but is there any other form of manual verification of the changes you've made?

Infrastructure member: I do ping tests against the machine hostname and IP and make sure everything is OK and nothing has slipped through the cracks. I RDP to the

machine and make sure it's hooked up the correct DC [domain controller] and make sure there are no networking related issues in the event viewer. The only other work I would do for this is if someone came back to me about connectivity or incorrect storage.

Researcher: OK, I would classify that as manual verification in my list, do you know how long you usually spend verifying all of the changes and updates you make to these new environments?

Infrastructure member: Everything I've just mentioned can take up to 2 hours, that's only if something strange has happened to that box and I need to troubleshoot. If everything's done correctly then it might take 30 minutes.

Researcher: So are you OK with me finding a middle ground there and documenting that there is usually an hour of manual verification in this process?

Infrastructure member: Sure

Researcher: OK, one last question before we finish up. Do all of these tasks require your full attention? I mean, are they tasks that you can carry out while doing some other more important form of work?

Infrastructure member: I would need to be there performing each of these tasks manually, so they would take up all my focus while I'm doing them. If more important work came up then I would have to stop making the environment and put it on hold until I had capacity to do it or [manager's name omitted] would prioritise it for someone else so they could do it.

Researcher: OK, that's fair enough. I think we're about done here so. Thanks so much for taking part today and I'll speak to you later if I have any questions.

Appendix B. Interview with Release Management member

- *This interview took place on the 27/11/15, from 14:30 to 14:38*

Researcher: Good afternoon, [name omitted], thanks for coming to the interview today. I've booked this room for 30 minutes in case there is a lot to discuss, but the last interview with [name omitted] took less than 15 minutes.

Release Management member: You're grand, we can take our time and go through everything.

Researcher: Good to hear, so you already know what I plan on covering here: the manual steps involved in the test environment creation process from release management perspective. It should go without saying that I'm conducting this research as part of my master's degree. I'm confident that I'm aware of all of the steps involved in your part of the process, but I'm not aware of how long each task actually takes. I am going to list them out and I want you give me a rough estimate on how long it takes you to perform them. Be sure to stop me at any stage if you think that I've missed anything or if I've made a mistake in the steps you take.

Release Management member: Will do.

Researcher: So once infrastructure have finished their work on provisioning, documenting and creating all the networking for the new environment, the first step you take is to clear down all environment specific data from the new machine. Can you tell me how long that usually takes?

Release Management member: Data? Not long, we know what folders are holding data from the old environment so clearing them down is simple enough and it's not exactly mandatory unless there's no disk space available. Let's say in total this could take 30 minutes.

Researcher: Alright, you said it's not mandatory, but is it a regular task that you would perform?

Release Management member: It would be, yeah, just in case the machine is going to be used for testing large uploads from the front end.

Researcher: OK, and do you think this is a symptom of taking clones for environments that are currently active, as opposed to creating new environments from a base operating system image?

Release Management member: Of course, if we took a brand new vanilla Windows box then we wouldn't have to do this, but we'd need to spend hours or days configuring the machines by hand, a load of applications and directories and configuration are taken over into new clones so we only have to change few things on them to get them running as opposed to installing and configuring everything from scratch.

Researcher: That's interesting, I'll mark that down in my notes. Is there any other step you need to take to clean down anything else that is environment specific? Do you need to reinstall applications or Windows features that are effected by the cloning process to avoid duplication of IDs in the network?

Release Management member: Ah, if you're talking about services and features then it's a long enough process. We would need to uninstall some of our own custom internal services, replace all environment specific content in their configuration files, then install them again and make sure they're working. Along with this there are some applications and Windows features that are hooked up to the network like MSMQs as they won't function unless a new GUID [Globally Unique Identifier] needs to be put in the registry and that's how the network tells the new machine from the machine is was cloned from.

Researcher: Alright, and how long would all that take?

Release Management member: Making sure there's been nothing added since the last environment was created and actually carrying it out usually takes a few hours.

Researcher: Do you remember how many hours the last it took on the last environment

Release Management member: I suppose for [name omitted]'s environment, it took about 2 to 3 hours, you need to restart the machine at least once for the updated registry keys to propagate when you're done, the old entries are stored in memory and there's no sure way to get them out everywhere without a reboot.

Researcher: I'll mark it down as 2.5 hours if you're OK with that

Release Management member: Sure

Researcher: OK, so the next part would be updating all configuration files for the environment, so I have listed here: HOSTS file, web.configs, machine.config and app.configs

Release Management member: There are few more in that list but they'd fall under the same umbrella as the web and app.config files. The HOSTS and machine.config files are easy, there's only one of each to update. The others are in multiple locations across different drives on the machine, it would take a full day to do this manually but we'd normally do this through a search and replace program from a server with access to the new clone and look for the name of the machine the new one was cloned from and replace it with the new value, same goes for any web links, anything environment specific really we would need to find and replace, and there could be over 50 files to make several different replacements in.

Researcher: Do you have an estimate on this whole process? Take enough time to think and try to be as verbose as possible because it sounds like there's a lot going on in this step.

Release Management member: With connecting to the server, modifying the HOSTS and machine.config then connecting to another server and searching the whole machine remotely, I supposed this step could take the best part of a day.

Researcher: Was that how long this step took for [name omitted]'s environment?

Release Management member: I would say so, 4 and 6 hours sounds about right.

Researcher: I'll put that down as 5 then.

Release Management member: OK.

Researcher: Is there any other replacement of environment variables you need to perform? Are the IIS [Internet Information Services] sites and application pools in a stable state at this point?

Release Management member: Ah, the IIS site bindings do need to be updated too. So I suppose this could take another 2 hours as well.

Researcher: Alright, I'll put down 7 hours for the whole process of updating these configuration files then. So we've covered clearing down the residual data from the

previous environment, reinstalling applications and services and replacing configuration files. I'm going to move on to the code base deployment section as long as you're satisfied we haven't missed anything so far.

Release Management member: Yeah, I think we've got everything in those steps alright.

Researcher: So is the only remaining step deploying out certain applications from a certain feature or release branch?

Release Management member: We always deploy out the latest release of everything to these machines so we know that the codebase is a reflection of the versions deployed to production.

Researcher: So all services, web sites, databases and everything else goes out in this step? TeamCity handles most of this so surely it's a case of kicking off the builds and waiting till they are done.

Release Management member: There are a few configuration variables that need to be setup in TeamCity for this to work, without getting into details, there are about 4 different variables that need to be defined before we kick any of the builds off.

Researcher: Ok, and how long does the preparation stage take?

Release Management member: I would give this an hour considering you need to connect to the machine and extract these variables from the box itself and add them to TeamCity.

Researcher: And the deployment of the code base?

Release Management member: There are over 170 different builds that need to be kicked off, nearly all of them require additional user input to specify what branch to build, this, along with monitoring the success of the deploys. I wouldn't commit to having everything deployed from scratch in anything under a full day, just to take deployment failures and troubleshooting into account.

Researcher: That's a long time for manual verification, is it an error-prone process?

Release Management member: Deployments could fail if something was done incorrectly before this point as we are relying on a lot manual work to have been done correctly up to this point. and the environment that the new environments was cloned

from may not be able to support the latest release of code, there has been tables or schemas missing from databases or site folders missing from the machine that new code depends on. All of this needs to be taken into account.

Researcher: OK, that's very good information, I'll take that down in my notes for later. So if we were to break the full deployment process down: deployment preparation is an hour, then the deploy of the code base itself takes about 3 hours if nothing goes wrong. Then you're saying that you leave doing manual verification of the builds and doing any additional troubleshooting can take another 4 hours, bringing the total up to a day's work.

Release Management member: Conservatively, yes.

Researcher: Alright, I was going to ask you about manual verification of the whole process, but I think we've covered it already in the last answer.

Release Management member: I think so, the verification is mostly in the monitoring of the builds, one of them is sure to fail if something that precedes it was done incorrectly.

Researcher: That's fair enough, so I have one last question if you're confident that we have covered all and any tasks in your part in the process.

Release Management member: I am, we've definitely talked about all the tasks we perform when a new environment comes in.

Researcher: Alright then, do all of these tasks require your full attention? Can these tasks be done while you're carrying out some other form of work?

Release Management member: There are too many manual tasks here, so I need to pay full attention, if I don't the deployments could fail and I could spend hours chasing my tail on a configuration file I missed. Monitoring the deployments is the only part where I can take a short break to look at something else, but that's about a 10 minute window, enough to read an email, and make a quick response to it. I would have to be there to catch failures fast in case a deployment does fail.

Researcher: That's perfect, we are all done so. Thanks so much for taking part today and I'll speak to you later if I have any questions.

Appendix C. Interview with Database Administration member

- *This interview took place on the 20/01/16, from 17:05 to 17:18*

Researcher: Good evening, [name omitted], thanks for meeting me today. I've booked the room for 30 minutes but the other interviews were over and done with within 15 minutes.

Database Administrator: That's fine, I'll be heading away after this anyway.

Researcher: Alright, we'll wrap this up quick enough so. You know that I'm conducting research as part of the master's degree here in [company name omitted], and it is directly related to the process I've built around creating new test environments in the cloud from code. I finished a prototype of it recently, but a new requirement has come in from management to include the latest scrubbed production databases in these test environments. At the moment, it's a manual process that you perform, but eventually, this process will be integrated into my framework so it will be completely automated. I plan on covering all the manual steps involved in this process alongside estimates from yourself on how long each step takes. Are you OK with providing this to me?

Database Administrator: Sure, the scrubbing process itself takes a few days, and it's just replacing real data linked to clients with dummy data, but it's only done every few months so I'm not sure if you want details on this.

Researcher: I wouldn't say so, the way I see it working is that these databases will be purged of real data and replaced with dummy data then they will be placed in a central location that my process will be able to pull them from and restore them to the new server

Database Administrator: Alright, so what tasks are you looking for estimates on then?

Researcher: Everything bar the scrubbing process, as it falls out of scope of what I'm doing. So if you were to start with a completely blank machine from infrastructure I assume there's some preparation work you need to do before you copy the scrubbed databases over to the machine to restore them.

Database Administrator: There's a few things to do alright.

Researcher: OK, let's start with the preparation work then.

Database Administrator: Alright, so first we would RDP [remote desktop protocol] to the machine and ensure the SQL services are running.

Researcher: and how long would that normally take?

Database Administrator: About 10 minutes or so

Researcher: OK, and are there connections you need to setup to make sure the SQL services are operational?

Database Administrator: We would need to setup the SSRS [SQL Server Reporting Services] connection on the new machine, this can take up to an hour.

Researcher: Alright, what is the purpose of setting this up?

Database Administrator: Without the SSRS connection setup, the reports that the databases call will not be accessible, the front-end will break if certain pages that call the reports are accessed

Researcher: OK, so this would be classified as a prerequisite to restoring the databases?

Database Administrator: Yes.

Researcher: Alright, is there any other preparation task involved here?

Database Administrator: We need to run a set of scripts to allow for the databases to be restored, in these scripts, server level logins are created along with the setup of linked servers and a few other small things that need to be setup.

Researcher: Alright, and how long would these scripts take to run?

Database Administrator: I would say another 10 minutes or so

Researcher: OK, so we're at the stage now where the databases can be copied over to the new server and restored, correct?

Database Administrator: Yes, we usually start the copy of the databases beforehand as there is over 27GB of databases in 30 different files that need to be copied over, it can take about 2 hours to copy over if they are not on the same network.

Researcher: and AWS machines are not on the same network as where the scrubbed databases reside?

Database Administrator: No, they are on the [network name omitted] network.

Researcher: Alright so once this is done, it's just a case of running the database restore statements?

Database Administrator: Yeah, we have scripts saved to do this so I suppose you'll be taking these and automating them into your process down the line.

Researcher: That's the plan anyway. How long does the restore actually take?

Database Administrator: In total, the restores can take 5 hours, but a lot of that is just waiting around for large databases to restore.

Researcher: Alright, and once the restores are complete then your part in the process is over?

Database Administrator: No, I need to run another set of scripts after the restore to make sure there's no orphaned users on the server and take the databases out of read only mode so release management can deploy to them.

Researcher: Alright, is there any other function to the scripts you run at this stage?

Database Administrator: There are some users set up so QA can test the functionality before handing off to dev, that's about it really.

Researcher: Alright, and how long does it take to run these scripts on average?

Database Administrator: It can take up to 45 minutes or so.

Researcher: OK, it sounds like we're about done here unless you can think of anything we've missed so far.

Database Administrator: There are the TDE [Transparent Data Encryption] certificates that need to be imported for encrypted data to be accessed.

Researcher: Alright, and are these already scripted out?

Database Administrator: Yes, but the cert files need to exist on the machine before we run the scripts, the scripts just import the files into the SQL server

Researcher: OK, and how long exactly would copying the certificate files over and running the scripts take?

Database Administrator: Assuming you've already created a SQL session to the box, then it would take 10 minutes.

Researcher: and if you didn't have a SQL session created?

Database Administrator: 15 max.

Researcher: Alright, one last question, it sounds like you don't need to be paying full attention to a lot of the steps as copying takes an hour and the restores take an hour. How much effort would you say in hours is involved in this process?

Database Administrator: The prep is the only real section that requires my full attention, but the copies, scripts and restores can fail for lots of reasons so you need to be monitoring them. There could be networking issues, drives not mapped correctly or scripts run in the wrong order, stuff like that happens all the time. So it's not a case of "I'm going to run these scripts and come back in 5 hours". You'd really need to be paying a bit of attention to the whole process to make sure everything is copied to the right location and the databases are being restored to the right place. Plus, you'd need to make sure everything is working after the scripts do run. All the steps need to be done in a specific order so I can't just write one script that will do everything and kick it off.

Researcher: Alright, I think we can wrap it now. Thanks so much for coming in and participating today, [name omitted], your contribution will play an important part in my final result.

Appendix D. Survey Questionnaire

Test Environment Creation Process Survey

1. Test Environment Creation Information

Thank you for taking the time to complete this survey. The 13 questions in this survey are mostly multiple choice and are aimed at technical professionals who have knowledge of their organisation's IT Infrastructure and test environment creation process.

The results of this survey are imperative to an industry-based research project currently being carried out in conjunction with Aspen Grove Solutions and Cork Institute of Technology. By taking part in this survey you are directly contributing to the academic world.

All of the information you input will be held in the strictest confidence and will only be used to advance academic knowledge. Your IP address will be logged specifically for geolocation data.

1. Is your company's test environment creation process:

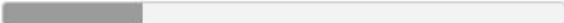
- Completely automated
- Semi-automated
- Completely manual
- Don't know

2. How many people are involved in your environment creation process?

- 1
- 2
- 3
- 3+
- Don't know

3. Does your company utilise an infrastructure-as-code tool in it's environment creation process

- Yes
- No
- Don't know

1 / 4  25%

Next

Test Environment Creation Process Survey

2. Test Environment Creation Timings

What follows is a breakdown of the steps involved in a generic test environment creation process. It is important to understand each of these steps and how they translate to your organisation's own process.

Provisioning of new environment:

Creating new machines, assigning storage, tagging of new resources, documentation of new machines, verifying connectivity, etc.

Domain operations:

Adding new machines to the domain, inserting machines to the correct organisational unit, adding DNS entries for new machines, etc.

Configuration of Servers:

Configuring OS and environment settings on new machines, modifying system files, installing required applications, configuring services, etc.

Deployment of codebase:

Compilation and deployment of organisations code to new machines. This includes, but is not limited to: websites, databases, services, APIs, etc.

* 4. Please indicate which of these steps apply to your environment creation process.

	Time to perform
Provisioning of new environment	<input type="text"/>
Domain operations	<input type="text"/>
Configuration of server/servers	<input type="text"/>
Deployment of codebase	<input type="text"/>

5. If other steps are involved in your organisations environment creation process, please specify them along with the amount of time each step takes

2 / 4  50%

Test Environment Creation Process Survey

3. Test Environment Classification Questions

6. How many test environments does your company currently run?

- 1-10
- 10-50
- 50-100
- 100-500
- 500-1000
- 1000+
- Don't know

7. How many servers comprise a single test environment in your organisation?

- 1
- 2-5
- 5-10
- 10+
- Don't know

8. What is the typical lifespan of a test environment your organisation

- Less than 1 week
- At least 1 week but less than 1 month
- At least 1 month but less than 1 year
- 1 year or more
- Don't know

9. What operating systems do your company's test environments run on?

- Windows
- OS X
- Linux
- Other (please specify)

10. Is your company's test environment infrastructure:

- Public cloud based
- In-house virtualised
- In-house physical
- Hybrid
- Don't know
- Other (please specify)

11. What platform or platforms is your test environment infrastructure based on:

- AWS
- Microsoft Azure
- Google Compute Engine
- VMWare vCloud
- Don't know

Other (please specify)

3 / 4  75%

Prev

Next

Test Environment Creation Process Survey

4. Respondent Classification Questions

12. What is your job role?

- Department Manager / Team Lead
- Software Engineer
- Operations Engineer
- Database Administrator
- QA Tester

Other (please specify)

13. About how many years have you been in your current position?

- Less than 1 year
- At least 1 year but less than 3 years
- At least 3 years but less than 5 years
- At least 5 years but less than 10 years
- 10 years or more

4 / 4



100%

Prev

Done