Dissertations

School of Computing

2015-05-10

Maturity of Cloud Application Interoperability Frameworks for Small to Medium Enterprises

John Warde
Technological University Dublin

Recommended Citation

Maturity of Cloud Application Interoperability Frameworks for Small to Medium Enterprises

John Warde

B.Sc. Hons, Computer Science, University of Limerick, 1994.

National Diploma, Software Engineering, Galway-Mayo Institute of Technology, 1992.

A dissertation submitted in partial fulfilment of the requirements of

Dublin Institute of Technology for the degree of

M.Sc. in Computing (Advanced Software Development)

March 2015

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Advanced Software Development), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research.

Signed: _____

Date: **06 March 2015**

ABSTRACT

Cloud computing has many benefits and organisations have bought into the cost effective and elastic solutions provided by major players in the market. However, cloud computing and Cloud Service Providers (CSP) are still evolving, hence there are differences in how customers connect with each provider to the orchestrate application lifecycle management. A lack of standards can create *vendor lock-in*. This work investigates current research and possible solutions to the vendor lock-in problem through the use of Cloud Interoperability or *multi-cloud* frameworks. Software developers and organisations can use these frameworks which abstract the differences between CSPs and mitigate vendor lock-in. A reference web application, with compute intensive operations, was developed and then adapted to each framework to evaluate the usability and stability of each multi-cloud framework, scaling up and down the underlying virtual infrastructure to meet varied demand. Cost conscious Small to Medium Enterprises can use these frameworks to stay competitive by having the ability to switch CSPs quickly for more favourable costs or better performance. Overall this will lead to increased competition and more innovation between CSPs benefiting the customer once more.

Keywords: *cloud computing, virtualisation, IaaS, multi-cloud, interoperability, scalability*

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1   INTRODUCTION

Organisations in all economies rely hugely on Information and Communications Technology (ICT) to achieve success. The development of complex, innovative software solutions is a key challenge. Computing professionals with advanced ICT skills are needed to design and build this software to help organisations develop solutions to compete in the global digital economy. ICT professionals are now working on large scale *cloud* computing projects which require a deep technical expertise in software design and development plus the ability to analyse problems, data, and information used in solving these large scale problem sets.

This research evaluates the maturity of multi-cloud frameworks for use in Small to Medium Enterprises (SMEs) to build scalable applications in the *cloud* (Third party internet connected data centres providing immediate access to temporary pay-per-use computing resources). Multi-Cloud frameworks are software used to enable Cloud Application Interoperability between heterogeneous Cloud Service Providers. They are built using advanced software development techniques, such as design patterns, to adapt and hide the differences between the run-time execution environments of different cloud service providers.

Existing research suggests that SMEs will benefit from the agility of being able to run business applications in multiple cloud providers over using cloud provider native solutions, thus avoiding the *vendor lock-in* scenario – where the costs of switching to another CSP are prohibitive due to incompatibility of vendor propriety services.

In this work, a number of multi-cloud frameworks are evaluated by building a reference application for each of the frameworks and assessing the technologies on factors such as development effort, performance and functionality.

1.1 Background

An Information Technology (IT) revolution is upon us, cloud computing is no longer a buzzword, it's here to stay. According to the "Trough of Disillusionment" in "Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business" [1], the levels of investment and spending ($566 billion dollars since 2010[2]) suggest it's too big to fail.

The IT revolution has been driven, in part, by demand from the recent growth of large scale social media and e-commerce web sites which in turn has been driven by us, the social media consumers, and our increased online spending. Companies such as Google and Facebook have had to invent their own paradigms and technologies such as MapReduce[3] and Cassandra[4] to cater for the increased demand for their services. These data hungry and mission critical applications are necessarily scalable and require the IT resources behind them to be scalable too.

This in turn has influenced other, sometimes more traditional, enterprises in their desire to achieve similar accelerated growth and flexibility to react to changes in the marketplace, or worse risk being left behind. Big Data and business intelligence (analytics of large amounts of market/user behavioural data) has helped companies to make better business decisions and reduce risk rather than rely solely on personal opinions[5].

A significant portion of the overall spend on cloud computing is not directly on the product (to the consumer) but rather on moving existing business support infrastructure and software into the cloud, such as Customer Relationship Management (CRM) and Human Resources (HR) systems.

---

[1] http://www.gartner.com/newsroom/id/2819918

[2] Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 2Q12 Update, https://www.gartner.com/doc/2126916/forecast-overview-public-cloud-services

[3] IBM - What is MapReduce, http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/

[4] What is Apache Cassandra?, http://planetcassandra.org/what-is-apache-cassandra/

[5] How Big Data Can Reduce Big Risk, http://www.cio.com/article/2371591/business-intelligence/how-big-data-can-reduce-big-risk.html

The effect is cutting across disciplines, it is influencing not only the infrastructure side of businesses but also the software development, project management, and business processes, and to some extent accounting where there is less capital expenditure spend with cloud computing (Marston et al., 2011).

Agile software development processes have been part of this industry for some years. It enables more flexibility by defining and building software iteratively in smaller components within small regular timeframes as opposed to the large monolithic phases of the waterfall software development methodologies. Agile has become an essential delivery method for many software development teams that need to respond to the ever changing demand of the market and the data analytics that indicate what the business should do next.

Cloud computing has also made IT infrastructure more flexible, no longer has the development team to wait weeks to months to get hardware provisioned by the IT department. Software engineers are skilling up to keep up with these changes – learning new Application Programmer Interfaces (APIs) and techniques needed to build robust and scalable applications that can span continents.

Global economies, and Ireland in particular, are currently engaged in a 'knowledge revolution' evolving from manufacturing to a service base fuelled by technological advances[6].

---

[6] Building Ireland's Knowledge Economy, Report to the Interdepartmental Committee on Science Technology and Innovation,
http://www.entemp.ie/publications/enterprise/2004/knowledgeeconomy.pdf

As part of its EU2020 strategy the European Union is currently progressing its Digital Agenda with the aim to find a fast road to economic recovery which puts Information and Communication Technologies (ICT) at the heart of this strategy[7]. The European Union is also funding much research and innovation, the Horizon 2020 is the largest EU programme ever with nearly €80 billion of funding available over 7 years[8], with cloud computing and ICT related research cutting across many diverse areas[9].

Utility services (such as electricity and telephone) as we know them today went through an evolution from innovation to agreed standards, to economies of scale. Innovation is still high in cloud computing, standards are only now being discussed/researched. Some global enterprise organisations are tentatively working with cloud technologies to evaluate the long term benefits; however they want to avoid the vendor lock-in of the past. Interoperability will thrust this fledging industry into commoditisation of services. It has been estimated that by 2025, the expected economic impact in the U.S. from cloud technologies is expected to be somewhere from 1.7 to 6.2 trillion dollars per year (Columbus, 2013).

1.2 Aims and Objectives

The aim of this research is to garner information and knowledge around the development and use of multi-cloud frameworks and to evaluate them with the view to applying these to a reference application design. The results of testing these frameworks will be used to analyse the maturity of each framework and then synthesise that knowledge to evaluate and give recommendations for use in SMEs and start-ups.

---

[7] A Digital Agenda for Europe, Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2010:0245:FIN:EN:PDF

[8] What is Horizon 2020? - The EU Framework Programme for Research and Innovation, http://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020

The Project Objectives are:

- Investigate the current state-of-the-art and research conducted to date on multi-cloud frameworks and Cloud Application Interoperability.

- Develop an experiment to evaluate the nominated frameworks based on technical criteria.

- Document and evaluate the findings from the experiment.

- Based on the evaluation, suggest a set of improvements that would increase the adoption of multi-cloud frameworks in start-ups and SMEs.

- Make recommendations for future research in this area.

1.3 Research Methodology

The research is carried out in the following phases:

- Perform a literature review of cloud computing and cloud application interoperability to assess the current status of interoperability research and inform the choice of frameworks for the experimentation part of this research.

- Design and develop a reference application software.

- Adapt the reference application to use the chosen frameworks.

- Assess the maturity of the frameworks by reviewing the success of the experimentation and the effort involved.

- Summarise the research, draw conclusions and recommend areas of future research.

The reference application is a scalable compute intensive application which consists of message queues for asynchronous communication between clients requesting the computation and the processing components. The processing components will read data from cloud based block storage or a database. This type of application would be used in many SaaS (Software as a Service) applications, for example, the photo sharing site flickr.com can apply effects to photos, or as a component in a web crawler which processes HTML documents to extract keywords from many web sites.

---

[9] Guide to ICT-related activities in Horizon 2020, http://ec.europa.eu//digital-agenda/en/news/guide-ict-

Results will explore if multi-cloud interoperability is on its way to achieving benefits for SMEs, and will examine if the frameworks reviewed are ready for use in production applications. While suitability is subjective, the main criterion for this research is scalability and interoperability (works on multiple Cloud Service Providers).

1.4 Research Scope and Limitations

This research will appraise the maturity of a number software frameworks/APIs that enable software developers to build applications in the cloud where that application can be moved from one cloud supplier to another seamlessly or at a very minimum of configuration change and cost. Some of these frameworks also allow individual components of the application to execute on two or more cloud service providers where those components can communicate across CSPs.

As the title suggests, this research is aimed or evaluated from the perspective of an SME. However, Small and Medium Enterprise is a broad term and multi-cloud framework software would require software development knowledge and would generally be aimed at companies where a significant part of their business is software development or a company with a business unit which produces software. Multi-cloud frameworks could also be very useful to Internet Start-ups that intend building an on-line application and want to keep their costs low by being able to switch providers to lower costs. However, enterprises without software developers could benefit from this research, they could come to a decision to build a software team that could use a multi-cloud framework to build an on-line application to innovate in their area of business without fear of vendor lock-in and could build on their competitive edge. An example in this case would be the Lotus Formula 1 racing team who have used cloud computing to reduce costs[10].

---

related-activities-horizon-2020

[10] The Real Cloud Computing Revolution, http://www.cio.com/article/2449646/cloud-infrastructure/the-real-cloud-computing-revolution.html

Experimentation in this research will include developing a reference application of a typical scalable web application and then adapting that application to execute in each of the *inter-cloud* frameworks. The reference application incorporates some compute intensive operation (such as applying special effects to photographs on photography website flickr.com) where the user interface must remain interactive. This is achieved by using message queues to manage the compute intensive operations as efficiently as possible. As the load on the web application increases the number of web servers and (compute intensive) processors can be increased to cater for the load and maintain the interactivity of the user interface. On the client side (browser/mobile), an AJAX (Asynchronous JavaScript and XML) web page will poll the server side scripts, which in turn will use the information in the response queue (when it eventually becomes available) to retrieve the results of the compute intensive operation (such as the new image with the applied special effect).

Chapters 2 and 3 reviews existing research in cloud computing and interoperability which includes automatic Service Level Agreements negotiations via automation. However, this functionality is not tested in the reference application by the experiment except to comment on its availability after reviewing the multi-cloud framework documentation.

1.5 Dissertation Roadmap

This research is organised as follows. Chapter 2 presents a brief history, background and definitions on cloud computing and recent research in the area. Building on that, Chapter 3 explores cloud application interoperability, the different types and approaches used by research teams in this area. Chapter 4 describes the design of the reference application software, the tools used to build it and the plan and process of adapting the reference application to each framework using the branching features of source code repositories. Consequently, Chapter 5 describes how the measurement effort is interpreted and how the experiment process is carried out. Chapter 6 documents the experiment and its results. Finally, Chapter 7 presents conclusions and possible future research.

2   CLOUD COMPUTING BACKGROUND

Cloud Computing involves the availability of temporary remote computing infrastructure (such as disk storage, compute time, routers etc.) and services (such as email, message queues) which are sold on a unit basis (hourly, monthly) with no long term contract. These can be provisioned (or de-provisioned) in a matter of minutes, rather than weeks or months for on premises systems. The idea of cloud computing has been around for some time. Computer scientist John McCarthy (son of an Irish immigrant) proposed the idea of computation being delivered as a public utility similar to the service bureaus which date back to the sixties (Arif Mohamed, 2009). In 1999, Salesforce.com launched a customer relationship management system which aligned with John McCarthy's idea and in 2002 Amazon Web Services offered infrastructure as a service, since then the interest, investment and innovation in cloud computing has sky rocketed. Cloud computing has had a transformative effect on IT infrastructure and services. It has become one of the most influential IT trends, registering in the Gartner Top 10 Strategic Technologies for the past 7 years[11].

2.1 What is Cloud Computing?

There are many different definitions of cloud computing, some research papers attempt to consolidate the different definitions (Vaquero et al., 2008). However, the National Institute of Standards and Technology (NIST) based in the USA gives a succinct definition: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models." (Mell and Grance, 2011).

---

[11]

https://www.google.ie/search?q=site:gartner.com+%22Top+10+Strategic+Technologies%22+(2009+OR+2010+OR+2011+OR+2012+OR+2013+OR+2014+OR+2015)

2.2 Characteristics of Cloud Computing

The NIST definition lists five characteristics of cloud computing, namely: On-demand self-service; Broad network access; Resource pooling, Rapid elasticity; Measured service.

2.2.1 On-demand self-service

The cloud service provider provides mechanisms to allow customers to provision computing resources such as disk storage, server machines, databases, routers, DNS and software applications. This self-service mechanism is usually provided through three methods, each method allows the consumer to request, interact and decommission computing resources:

- Application Programming Interfaces (APIs): When building software applications, software developers use APIs (defines rules how one software program talks to another) to automate the interaction with the CSP's computing resources. This is the most efficient of the interaction methods, requires a level of skill for usage and can have a steep learning curve. It allows full access to all the functionality that a cloud service provider exposes to consumers of their services. Figure 1 shows how to create a new virtual machine (VM) on Amazon Web Services (AWS), based on the AMI (Amazon Machine Image) called *ami-4b814f22* with an instance type of *m1.small* (defined by AWS, refers to the amount of memory on the VM). The security related key name *my-key-pair* and security group *my-security-group* are pre-defined by the customer. In basic terms, an AMI is a snapshot in time of a system hard disk with all the required applications installed and configured for the customer's purposes.

```
RunInstancesRequest runInstancesRequest =
        new RunInstancesRequest();

 runInstancesRequest.withImageId("ami-4b814f22")
                    .withInstanceType("m1.small")
                    .withMinCount(1)
                    .withMaxCount(1)
                    .withKeyName("my-key-pair")
                    .withSecurityGroups("my-security-group");
RunInstancesResult runInstancesResult =
        amazonEC2Client.runInstances(runInstancesRequest);
```

Figure 1 Java code to start an Amazon Web Services EC2 compute instance[12]

- Command Line Interface (CLI): Generally the domain of System Administrators and infrastructure engineers. This allows users to type in commands in a console like interface and is used to quickly commission/decommission computing resources and troubleshoot problems with provisioned resources. The commands issued in the console will in turn make calls to the APIs. Figure 2 essentially achieves the same result as Figure 1 except with different parameter values.

```
$ aws ec2 run-instances --image-id ami-xxxxxxxx --count 1 --instance-type t1.micro
                        --key-name MyKeyPair --security-groups MySecurityGroup
```

Figure 2 Starting an Amazon Web Services EC2 instance using Command Line Interface (Linux)[13]

- Web Based Graphical User Interface (GUI): The majority of users start here and generally after signing up on-line with a new cloud service provider you are presented with the GUI built in HTML 5 and using AJAX to allow the GUI to feel more like an installed application rather than a regular web page. The GUI in turn will make use of the API and/or CLI to carry out the instructions intimated by the user actions.

---

[12] Run an Amazon EC2 Instance, http://docs.aws.amazon.com/AWSSdkDocsJava/latest/DeveloperGuide/run-instance.html

[13] Launching an Instance, http://docs.aws.amazon.com/cli/latest/userguide/cli-ec2-launch.html

	Family	Type	vCPUs ⓘ	Memory (GiB)	Instance Storage (GB) ⓘ	Network Performance ⓘ
☐	General purpose	t2.micro Free tier eligible	1	1	EBS only	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	Low to Moderate
☑	General purpose	t2.medium	2	4	EBS only	Low to Moderate
☐	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	Moderate

Cancel   Previous   **Review and Launch**   Next: Configure Instance Details

Figure 3 Launching an Amazon Web Services Instance from Amazon Web Services' web interface

Cloud service providers are looking for efficiency at every level so that they can pass on the savings to the consumer and stay competitive. None of the above methods require human interaction with the service, the service providers build robust interfaces, detailed on-line documentation and advisor/expert type articles to reduce the need for support phone calls and face-to-face interactions. Generally when you are first exploring a new cloud service provider you start interacting with the above interaction methods in reverse as you gain knowledge and insight into the intricacies of the provider.

2.2.2 Broad network access

Cloud service consumers are able to access the CSP provided services anywhere any time – enabled by providing access via the internet. As outlined above, CSPs provide a Web 2.0 type interface and this allows access through workstations, laptop, tablets and smartphones; some CSPs also provide a native smartphone and/or tablet application to provide a better user experience on a mobile device and allow the consumer to react quickly to demand or troubleshooting. With broad network access come security concerns and these can be mitigated by security certificates, virtual private networks (VPNs) and other methods.

2.2.3 Resource pooling

A big enabler of cloud computing is the concept of virtualisation. Entire Operating Systems (OS) have been virtualised within physical machines; a number of virtual machines (VMs) can exist within a single physical machine. The VMs can have different operating systems installed within them i.e. a physical Linux machine could host a Linux VM, a Windows VM and a Mac VM within it simultaneously.

Usually, these VMs have moderate disk space configured on the same virtual disk as the OS and then the VMs connect to a Storage Area Network (SAN) where the majority of the CSP's customers store their data. The SAN will be a highly optimized set of hard disks or Solid State Drives (SSD) and this again is virtualised for customer disk quotas and for response times according to each customer's service level agreement (SLA). The SLA comes into play again when virtualising CPU cycles on the physical host servers so that VMs get the correct level of CPU time there.

Virtualisation enables the cloud service provider to pool its overall physical resources and get the most out of its hardware (Lakhani and Bheda, 2012). The CSP will keep a certain level of physical computing free, ready to respond to increased customer demand (Chapman et al., 2010) and then only order new physical resources when needed – to keep capital costs down.

However, for any one cloud server consumer, each of their VMs could exist on any one of thousands of physical machines within the CSPs data centres and their data could exist on various physical hard disks and in different countries – this is hidden/abstracted from the consumer and they can't tell which physical computer anything resides on. Consumers are given some level of control, to comply with laws (Cavoukian, 2008) such as EU Directive 95-46-EC[14] and to reduce latency to the consumer's users, by specifying geographical locations such as Europe West or United States of America East Coast, etc.

---

[14] http://www.dataprotection.ie/docs/EU-Directive-95-46-EC/89.htm

2.2.4 Rapid elasticity

For an on premises application where a company provisions their own hardware, they attempt to plan their capital expenditure (CAPEX) of computing resources around predicted demand for their service. This can lead to over provisioning and they may lose money or have under provisioned and can't meet current demand because their supplier is slow in delivering new hardware, thus reducing customer satisfaction due to poor response times. Figure 4 shows the relationship of actual demand to planned and actual provision of computing resources (e.g. server boxes), the orange area shows under provisioning and the grey shaded area shows overprovisioning.
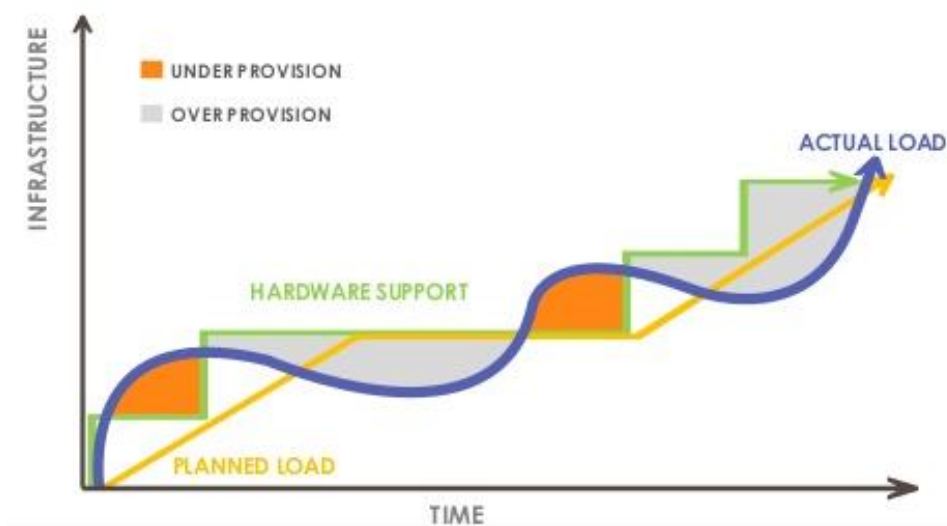
Figure 4 Under and over provisioning of I.T. resources in relation to user demand[15]

Through the use of APIs and automation, the cloud service providers allow their customers to automatically scale up or scale down. A customer can have different levels of control: full control directly through the APIs where the customer starts up (or tears down) another compute instance (virtual machine) or load balancer; or at an indirect level where the CSPs automation controls the numbers of active instances of nodes based on an algorithm supplied by the customer (or a default by CSP). This algorithm will use system performance numbers such as average CPU usage across all VMs, average Input/Output Operations Per Second (IOPS) etc. This elasticity allows spending on IT resources to better follow demand as depicted in Figure 5.

---

[15] http://www.slideshare.net/Intelligrape/auto-scaling-26821141
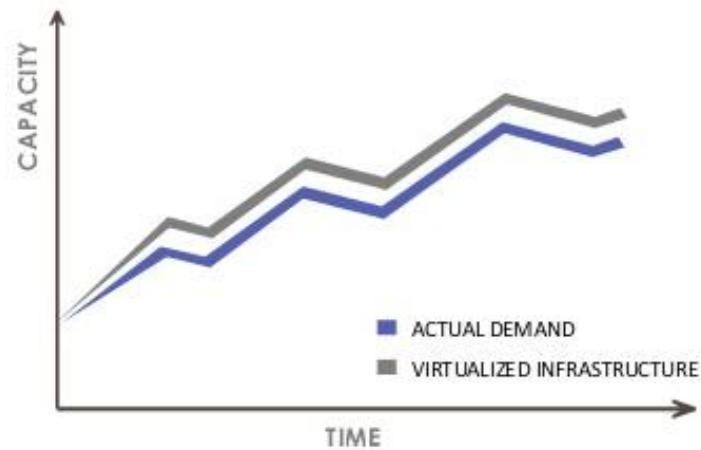
Figure 5 Auto Scaling with on demand compute resources can align closely with demand
Demystifying Auto Scaling[15]

2.2.5 Measured service

For acceptance by mainstream business, cloud computing needs to be transparent about the fees charged for their services. Everything is measured; disk space/quotas, number of input and output operations initiated, bandwidth travelling outside the cloud service providers' datacentres. However, unlike a utility where you might have one or two rates for one provider, cloud service providers have many different rates not only for the different type of computing resources but also for the different sizes and processing power of those computing resources, see Table 1. Determining the overall costs or verifying the costs is sometimes challenging when consumers first decide to adopt cloud computing and new businesses have started up around calculating this cost too.

Table 1 Sample Charges for General Purpose Compute Instance on AWS[16]

On-Demand Instance Prices

| Linux | RHEL | SLES | Windows | Windows with SQL Standard | Windows with SQL Web |

Region: EU (Ireland)

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.micro	1	Variable	1	EBS Only	0.01€ per Hour
t2.small	1	Variable	2	EBS Only	0.02€ per Hour
t2.medium	2	Variable	4	EBS Only	0.05€ per Hour
m3.medium	1	3	3.75	1 x 4 SSD	0.06€ per Hour
m3.large	2	6.5	7.5	1 x 32 SSD	0.12€ per Hour
m3.xlarge	4	13	15	2 x 40 SSD	0.25€ per Hour
m3.2xlarge	8	26	30	2 x 80 SSD	0.5€ per Hour

2.3 Virtualisation

Virtualisation is the cornerstone of cloud computing and has allowed it to flourish. It allows cloud service providers to efficiently use their hardware resources by utilising as much of the computing power of a single machine as much as possible. Cloud service providers are not the sole users of virtualisation, many companies across the globe have been creating VMs on their "on premises" hardware for many years. Virtualisation is a mature and robust technology.

Virtualisation allows multiple logical operating systems to operate on a single physical machine. This is done by creating a hardware abstraction layer; each virtual machine (VM) requests use of the hardware devices on the physical machine through this layer and the Virtual Machine Monitor (VMM), or more commonly known as a hypervisor.

---

[16] AWS Amazon EC2 Pricing, http://aws.amazon.com/ec2/pricing/

There are two types of hypervisors:

- Native or bare metal: the hypervisor sits directly on top of hardware and above this multiple Virtual Machines/Operating Systems and then the applications that run within those VMs, as depicted in Figure 6. The Hypervisor in this configuration talks directly to the hardware.
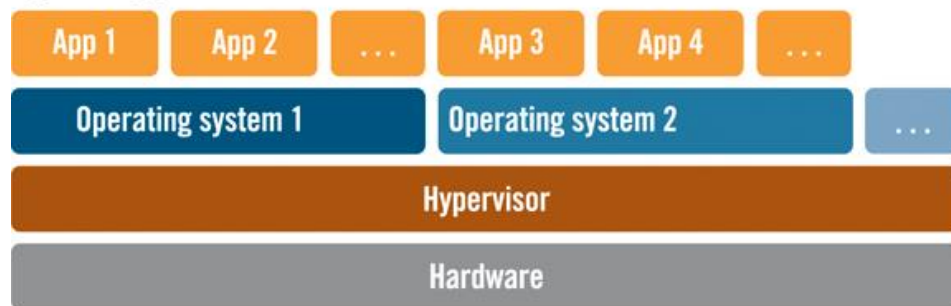
Figure 6 Component layers in Native or bare metal hypervisor usage[17]

- Hosted: this type of hypervisor sits above the operating system on the host physical machine. It is essentially just like any other application in the host operating system and provides the layer of abstraction to one or more VMs above this, see Figure 7. This type of hypervisor makes use of specialized CPU instructions to get CPU slices for the VMs it supports.
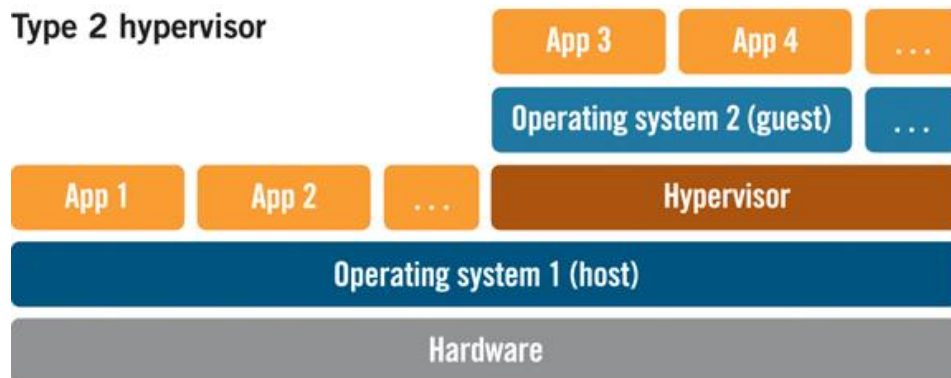
Figure 7 Component layers in a Hosted hypervisor[18]

---

[17] The Difference Between a 'Type 2' Hypervisor and a 'Type 1' Hypervisor,
http://www.virtzone.net/the-difference-between-a-type-2-hypervisor-and-a-type-1-hypervisor/

[18] The Difference between a 'Type 2' Hypervisor and a 'Type 1' Hypervisor http://www.virtzone.net/the-difference-between-a-type-2-hypervisor-and-a-type-1-hypervisor/

Bare metal hypervisors have a distinct advantage over hosted hypervisors in that they are closer to the hardware of the hosted server. Applications within VMs using Type 2 hypervisors need to go through two operating systems to get to the hardware. Bare metal hypervisors are favoured by cloud service providers because they require less CPU slices of the host hardware and therefore make more efficient use of the hardware overall (Chang et al., 2013).

Table 2 Hypervisor products categorised by hypervisor types
The Difference between a 'Type 2' Hypervisor and a 'Type 1' Hypervisor[18]

Type 1: Bare Metal Hypervisors	Type 2: Hosted Hypervisors
VMware ESXi	VMWare Player
Microsoft Hyper-V	Oracle VirtualBox
Xen	RedHat KVM

There are different implementations of hypervisors (for each type) as can be seen from Table 2 above and each hypervisor has their own virtual machine format. This leads to incompatibilities between cloud service providers when application owners want to move their application from one CSP to another leading to *vendor lock-in*. The Open Virtualisation Format (OVF)[19] by Distributed Management Task Force Inc. (DMTF)[20] attempts to overcome these differences by specifying resource definitions that are contained in a virtual machine so that the different hypervisors can interpret and use VMs coming from other hypervisors. (Galán et al., 2009) proposes extensions to the OVF format so that additional items can be specified such as Key Performance Indicators (KPIs) and Elasticity Rules among other. However, this format only describes what should be available to the VM at start up, KPIs and Elasticity rules can change dynamically over the lifetime of the VM.

Virtualisation has also enabled scalability and the ability to build robustness in applications. A user can build one VM with the desired OS, applications and configuration and then replicate (through a type of binary copy) that "golden master" VM to create more VMs to handle the application load.

---

[19] http://www.dmtf.org/standards/ovf

[20] http://www.dmtf.org/
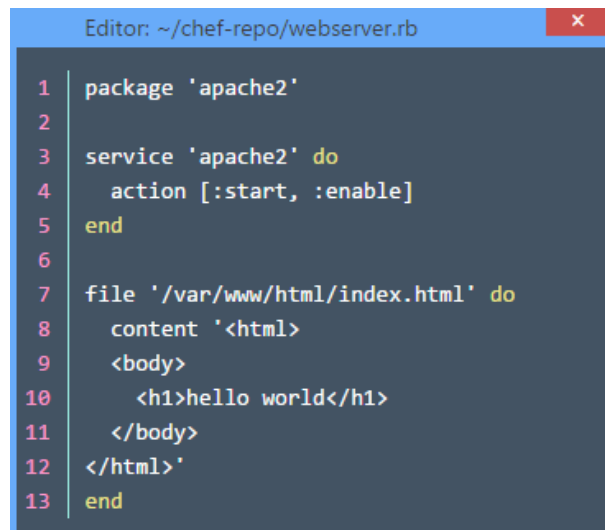
An application may have three or more types of server configuration which would mean three or more golden master VMs and any one of those can be replicated depending on the scenario or part of the application that needs to scale out, e.g. an application may have a processing server, a database server (slave) and load balancer golden master VMs to replicate.

Hardware always fails eventually and VMs fail too. However, by making copies of golden masters on the same hardware, on other hardware within a data centre, and on hardware at a different geographically located datacentre, then robustness can be built into an application as a whole. If one VM fails on a single hardware machine then other VMs (of same configuration) can take up the slack while a new VM is created within minutes. If a single hardware machine fails then copies of those VMs already exist on other hardware machines and these VMs can take the slack of the failed hardware machine while another hardware machine is configured with copies of the VMs. If an entire data centre fails e.g. a severed network connection or terrorist attack then another datacentre will have multiple hardware machines with copies of the same VMs and the application can continue to serve, maybe at a degraded level, but it continues until hardware/virtual machines have been setup/replicated again to restore the expected level of service. Cloud service providers build these tools to enable infrastructure automation and ensure that they work efficiently so that customers can have as many VMs, and other computing resources, to enable large applications to operate to the required Service Level Agreements (SLAs).

Recent innovations in the virtualisation space are trying to reduce the unit of virtualisation from an entire OS to containers that have just enough of the operating system features for the target application to operate - Linux containers (LXC) and the open source Docker project are gaining recognition in the industry with Microsoft recently announcing that they will support this technology (Foley, 2014). By reducing size, this allows the CSP to pack more virtualisation units onto a physical machine thus reducing costs and allowing them to reduce the unit cost computing power to the consumer. Amazon Web Services also recently announced AWS Lambda (Kar, 2014) which reduces the unit of virtualisation to functions – function instances are started within milliseconds on a customised event.

2.4 Desired State Configuration

The last few years has seen a move away from the use of golden master VMs to Desired State Configuration (DSC) tools. These allow infrastructure engineers to quickly install and configure an OS and applications onto an empty VM, through automation. When using *infrastructure automation* tools like Puppet or CHEF, you describe what state a server should be in. This is done through scripting code as shown in Figure 8 below. These tools allow you to list what applications should be present on a server and also how they should be configured. The server component of these automation tools then push these scripts to their client components on individual servers of an application. These individual infrastructure automation clients then run the same script which will bring all the servers in a group in-line with the state described, by initiating the appropriate operating systems commands. Different groups of servers can have a different script pushed to them, i.e. a script for application servers and a script for database servers.

```
Editor: ~/chef-repo/webserver.rb                    ✗

1   package 'apache2'
2
3   service 'apache2' do
4     action [:start, :enable]
5   end
6
7   file '/var/www/html/index.html' do
8     content '<html>
9     <body>
10      <h1>hello world</h1>
11    </body>
12  </html>'
13  end
```

Figure 8 Simple Web Server Setup Using Chef[21]

In Figure 8, when executed by the Chef Client component on a server (machine or virtual machine), this script will install the apache2 webserver software, start and enable the apache2 service and create a home page at /var/www/html/index.html with the basic content as shown above.

---

[21] Configure a package and service https://learn.chef.io/ubuntu/configure-a-package-and-service/

(Vanbrabant and Joosen, 2014) suggests that desired state configuration management tools can be used as a multi-cloud enabler but fail to address the major differences between some operating systems. For example, Microsoft Windows does have a standardised packing management system for installing applications and therefore installation different applications on that platform can be inconsistent. Applications in Linux are installed through a packing manager to ensure consistency – this is one of the reasons why Linux has flourished in the cloud computing world.

As with golden master VMs, you can create different configurations (or recipes in Chef Terminology) for different types of servers in your overall application.

Desired State Configuration offers more flexibility and robustness over golden master VMs:

- Infrastructure is described as code and becomes easier to understand whereas your golden master VM could differ from the documented steps accompanying the VM if infrastructure engineers are not fastidious.
- Infrastructure code can and is stored in source code repositories and therefore can be versioned and restored to previous versions if there is a problem with the latest version.
- DSC can offer self-healing features, where golden master VMs cannot. Chef clients on each server check that the configuration stays the same as described. If someone on the individual server changes a configuration item or application under DSC control then the chef client will determine how to bring that server back to the desired state.
- DSC can update configurations on every machine quickly. After making the necessary changes within a golden master VM, then all existing VMs that were created with the previous version need to be taken down one-by-one and replaced with a new VM styled from the updated golden master. Every Chef client communicates regularly with a Chef server to ensure that the local configuration (recipes) match what is on the server. If not, then they copy down the latest recipes (applicable to the type of server) and execute on local server to bring all servers in line with the updated desired configuration or policies.

- DSC offers some platform independence, in that the same "infrastructure code" can be run on the different flavours of Linux and on Linux Containers/Docker. Chef also supports Windows, however because the operating system and application/package names are quite different the infrastructure code will not transfer directly – but concepts and tools to run the Chef Server and Clients are very similar.

2.5 Chapter Summary

This chapter gave a broad overview of cloud computing and the early technologies that underpin and lead to the great growth of the industry. There are a number of dominant players in the cloud computing market which have lead the way in terms of innovation but that innovation has been at the expense of interoperability and those players have locked in many users of their technology and prevented those customers from moving to providers with better price points and/or better technology. The next chapter reviews research into the challenges and approaches of cloud interoperability.

3   CLOUD INTEROPERABILITY REVIEW

Cloud Application Interoperability is the ability to move an application that contains many virtual infrastructure elements, such as servers, routers, load balancers, storage and data from one cloud service provider to another cloud service provider with little or no changes to code or configuration. This gives the cloud computing consumer purchasing power and choice of operators and rates.

The novelty behind Cloud Computing is that distributed physical resources, such as storage, CPU and networking (Infrastructure), programming frameworks and libraries (Platform) and services (Software) can now be traded as economic goods, integrated and offered on-demand through the Internet in a "pay-as-you-go" model. Cloud computing interoperability problems arise when different cloud providers try to coordinate exchange of data, applications and virtual machines. These incompatibilities can be either technical, e.g. incompatible virtualisation implementations (VMware, Xen and KVM) or incompatible programming code (Java-based, PHP-based), or semantic. For instance, different cloud providers use different modelling and notation for exposing the same features (Loutas et al., 2011a)

The providers of multi-cloud frameworks range from standards bodies to universities to research arms of multi-national IT service companies. Many of the researchers in multi cloud attend the Future Internet Assembly[22] (FIA) conference held each year in various European countries, and is supported by Seventh Framework Programme (FP7)[23].   One of the tasks of this research is to evaluate the current situation of research and development into multi-clouds, this was helped by attending the "FIA Dublin Workshop: Multi-Cloud Scenarios for the Future Internet"[24] on May 7th, 2013. At this event a number of the multi-cloud framework providers gave an overview of their technologies, including what stage of development the frameworks are at.

---

[22] http://www.future-internet.eu/home/future-internet-assembly.html

[23] http://cordis.europa.eu/fp7/home_en.html

[24] http://www.cloud4soa.eu/fia-multicloud

3.1 Cloud Application Interoperability Research

In "Semantics Centric Solutions for Application and Data Portability in Cloud Computing" (Ranabahu and Sheth, 2010) proposes using Domain Specific Languages (DSL) and code generators to enable interoperability, generating artefacts (executables, configuration files, etc.) for deployment on multiple IaaS and PaaS solutions. The referenced proof (Atwood, 2008) may not go far enough to prove the stated optimised shortcomings of this approach and does not address the new paradigms that have emerged in scalable computing in recent years, such as Map/Reduce, column orientated data stores, etc. Also, monitoring is not addressed; monitoring is used extensively in scalable applications to identify bottlenecks or underperforming entities which could aid decisions about when to optimise the generated code. If your application is serving millions of users on thousands of servers then even small optimisations will achieve large economic benefit. Also this approach does not support mobility of components where one can move individual components to different service providers.

The "Service Orientated Cloud Computing Architecture" (Tsai et al., 2010) paper proposes a Service Orientated Cloud Computing Architecture (SOCCA), see Figure 9. It addresses the major concerns of enterprise organisations, such as security and multi-tenancy, automated negotiation of Service Level Agreements (SLA) and operating components of a distributed application on different cloud service providers. A service-orientated approach is engineered into the framework (top level in Figure 9); many enterprise organisations already use Service Orientated Architecture patterns to decouple components of a distributed application. Ontologies are used in the Cloud Ontology Mapping Layer to hide the differences in implementation at each CSP (Cloud, Cloud 2, … Cloud N in Figure 9). The framework is flexible enough to cater for new types of cloud computing resources that will inevitably materialise as innovation continues to thrive in this fledging industry.
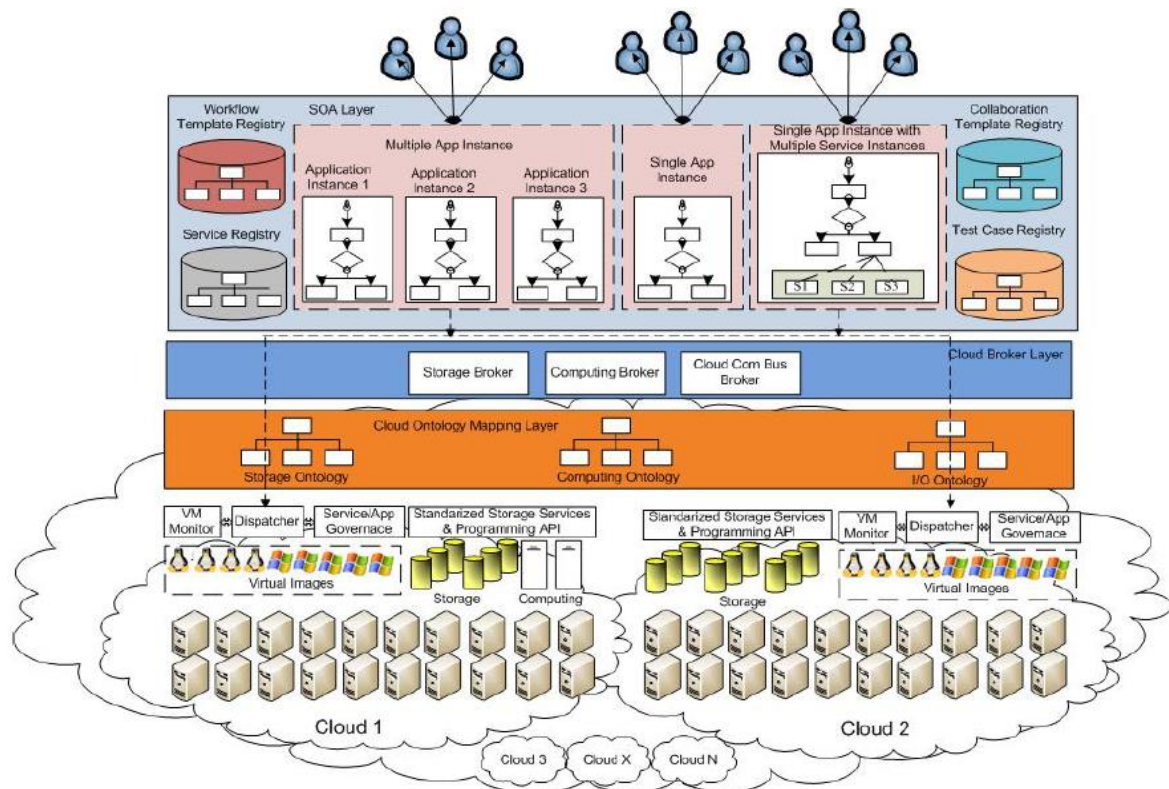
Figure 9 Service-Oriented Cloud Computing Architecture
(Tsai et al., 2010)

The Cloud Broker Layer communicates with the different CSPs through the Storage, Computing and Cloud Communication Bus Brokers; these use the ontologies to convert the messaging from the top layer to the messaging understood by the individual CSPs. Also, uniquely in this literature review, the paper proposes automated demand prediction model to enable negotiating/provisioning of resources on other cloud computing providers for expected demand in x number of days.

In "A Semantic Interoperability Framework for Cloud Platform as a Service" (Loutas et al., 2011b) attempts to eliminate the concept of IaaS as the majority of cloud computing components are now in the PaaS layer, i.e. more than one step away from the virtualised resource and this becomes the entry point for all SaaS application to computing resources. Again semantics are prevalent in this research paper.

The paper titled "Intercloud Directory and Exchange Protocol Detail Using XMPP and RDF" (Bernstein and Vij, 2010), proposes the creation of Intercloud Directories and Exchanges to facilitate interoperability as shown in Figure 10. Intercloud Directories would be analogous to the Universal Description Discovery and Integration (UDDI) directories which have not enjoyed stellar success in the SOA space. Intercloud Exchanges would translate messaging to/from distributed SaaS applications – it is another possible point of failure that application architects would need to take into account to provide continuous service to their customers. This approach may prove too political to implement – who manages and defines the intercloud messaging? However, ontologies are used extensively again to facilitate the matching of components between cloud service providers.
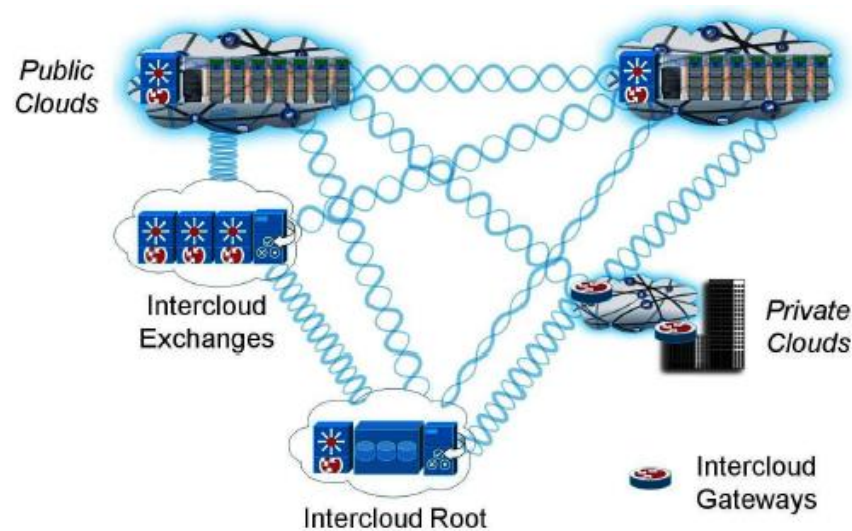
Figure 10 Reference Intercloud Topology
(Bernstein and Vij, 2010)

A review of underlying architectures contained in "Cloud computing interoperability: the state of play" (Loutas et al., 2011a) reveals common approaches in multi-cloud architectures, see Table 3; however it does not evaluate the maturity of those frameworks in its findings.

Table 3 Summary of multi-cloud framework types
(Loutas et al., 2011a)

TABLE I.	STANDARDIZATION ACTIVITIES IN CLOUD COMPUTING	
Standardized API	**Cloud Model**	**Broker**
OGF/OCCI, CDMI/SNIA	DMTF/Open Cloud Standards Incubator, DMTF/CMWG, DMTF/OVF, OASIS, Open Group Cloud Work Group, IEEE P2301, IEEE P2302, CCIF, OCC, Data Center Alliance, OGF/OCCI	TM Forum/Cloud Services Initiative, CCIF, CIF

C.   *Cloud Computing Interoperability Frameworks*

3.2 Chapter Summary

This chapter gave an overview of the state of play for research in cloud application interoperability and inter-cloud frameworks, identifying the complex nature of the problems to enable computing resources to execute at multiple host cloud service providers. This research seeks to go one stage further and evaluate a number of frameworks and ascertain their suitability for use in production by Small to Medium Enterprises.

Cloud computing has levelled the playing field for companies in the technology sector, allowing smaller or start-up companies to compete and disrupt in the marketplace by reducing the capital expenditure needed up-front to compete against bigger players. Inter-cloud frameworks will level the playing field further, by allowing companies to switch providers to stay competitive on price and/or performance, if needed.

The next chapter covers the design of the reference application used in the experiment and the design of the experiment itself.

4   DESIGN

This chapter first gives an overview and design of the distributed reference application used in the experiment and then goes through the design of the experiment.

4.1 Reference Application Design

The reference application follows a typical web application where a compute intensive operation (such as applying special effects to photographs) happens on the server side while the user interface on the client side must still remain interactive to the user.

This is achieved by using message queues to manage the compute intensive operations as efficiently as possible. Message queues are software that allows components to atomically and asynchronously send messages to other components or applications.

The diagram in Figure 11 shows the high-level design of the reference application. Starting from the left, client browsers on workstations and laptops or mobile apps connect to a website (i.e. www.interop.com). DNS (Domain Name System) settings point the domain name interop.com to the load balancer networking device which evenly distributes HTTP (Hyper Text Communication Protocol) messages to one or more application servers. When an application server is asked, by the user, to apply an imaging effect it will send a message to a request queue for processing. This message will have a unique identifier, details of the image and the name of the effect to be applied to the image. One or more Processor components are waiting to consume and process those messages.
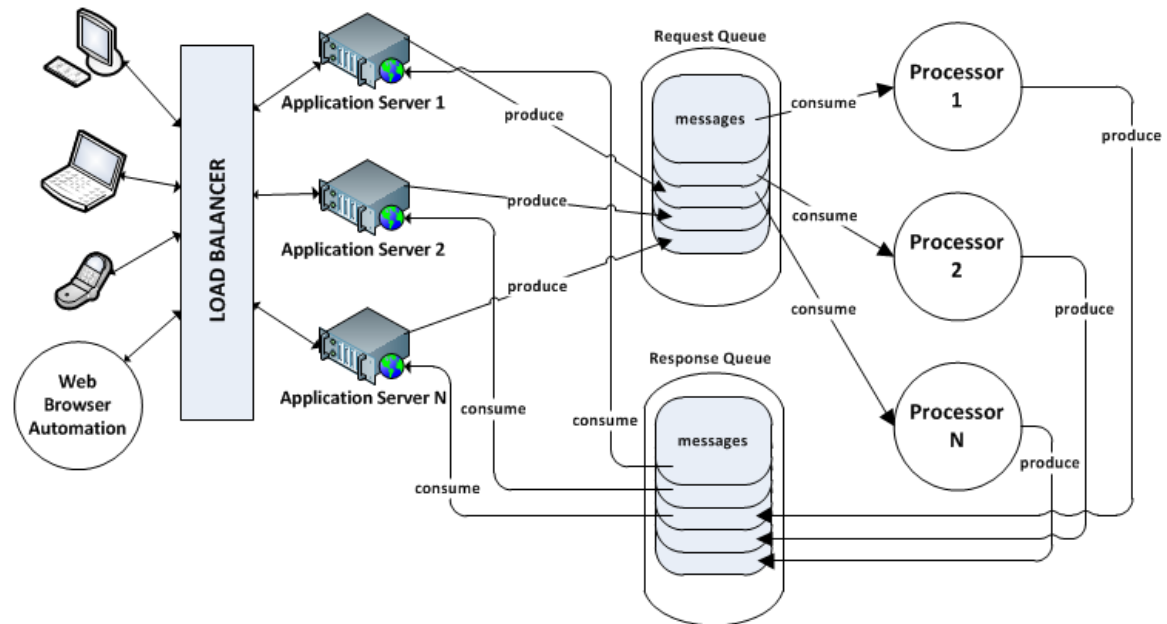
Figure 11 Design for a scalable web application

When an individual processor receives a message from the request queue, it uses the details in the message to retrieve the image file and apply the specified effect to the image. Once this operation is complete, the Processor sends a message to a Response queue and acknowledges, to the Request queue, that the original message has been successfully processed and it can be removed from that queue (by the queueing software).

Meanwhile the application server(s), on behalf of the client, will continually poll the Response queue until a message that corresponds to the unique identifier (sent with the original request) is retrieved. When this happens, the application server uses the details in the message to send a notification back to the client to re-point the image it is showing on the web browser window to the new image created by the processor which effectively shows the same image with the effect applied. Although not depicted in this diagram, all the images are stored on a Storage Area Network (SAN) of disks so that the individual components can access the images uniformly.

As more and more users log onto the web application and are applying effects to images, the number of requests towards the processors increases. If the processors cannot keep up with the requests then the Request queue fills up. Infrastructure operators or automation, monitoring the queue will then start up more Processor server VMs to handle the extra load. In the same vein, if the application servers are slowing down, due to the number of HTTP requests coming in through the load balancer, then automation or operators will increase the number of available application server VMs, thereby increasing the throughput of the overall application. Also when the rate of requests from clients reduces it allows the number Processor server VMs and/or application server VMs to be reduced. Thereby, allowing the customer of a CSP to keep costs in-line with demand by increasing or reducing the number of units (VMs) in use as depicted previously in Figure 5.

The design of the web application other than what is described above and in Figure 11, is kept simple as the main aim of this research is to test the inter-cloud frameworks for interoperability and scalability. In general, the application has a user/password login functionality to simulate multiple users. The credential details are stored in an in-memory database which is populated at run-time from configuration items. The images are stored on block storage in user named folders; each user folder is flat and only contains image files.

When logged in, the user will see the images from their associated user folder along with the filename of the image file without the preceding path. The user can then click on a single image which instructs the web server to display that image on a page of its own with a drop-down selection box containing the names of some image processing effects and a save button which will save the image with the applied effect back to the user's folder, see Figure 12.
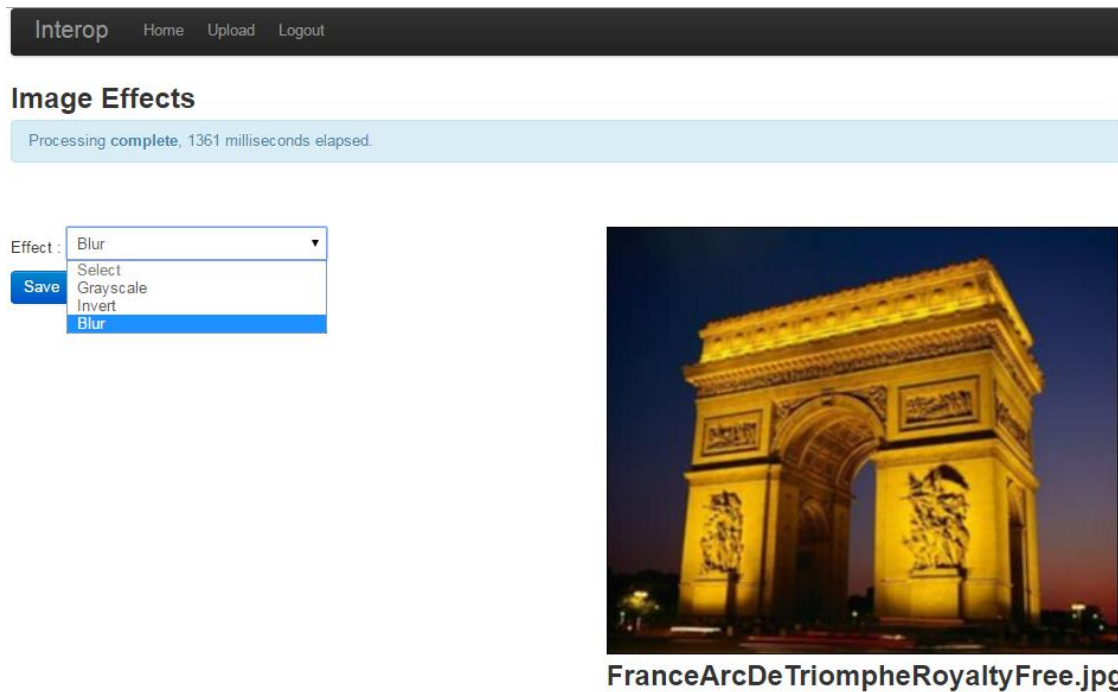
Figure 12 Browser screenshot for reference application, image processing page

When the user selects a new effect from the drop-down box, the asynchronous messaging comes into effect. This event causes the client to send an asynchronous message to the application server to apply the chosen effect to the image file. When the application server receives this message, it creates a corresponding message with a request identifier, sends it to the message queue and finally sends the request identifier back to the client browser.

The client browser periodically asks the web server if the image processing request has completed by sending asynchronous messages, all the while it keeps the client browser interactive by displaying an animated wait icon over the image. For the purposes of this experiment, the client browser will continuously display the elapsed time since the initial request. When asked by the client, the application server polls the response queue with the request identifier. If there is a message that matches the request identifier then that message will be returned to the application server along with other message details.

When the application server eventually gets a message from the response queue then the web server will repackage the request and send it back in the same asynchronous connection. That message contains the URL (Uniform Resource Location) to the newly created image. The client side will then show the new image to the user by manipulating the HTML page to show it. The user can then click the 'Save' button to save the new file to their user folder, this action will return the user to the main page with all the users images.

One or more processor components will continuously wait until the queuing software serves up one of the image processing requests. On receipt of a message, which contains the name of the effect and the full path filename to the user's image, the processor reads the image into memory, applies the effect algorithm and saves the file out to block storage. Also on the processor side, the image file storage is kept simple, the generated images are stored in a single folder. The filename contains the processing request identifier, which is a Universally Unique Identifier (UUID), and the original file's extension (i.e. ".png") to aid the image processing APIs identify the image type. By using a UUID, it guarantees that there won't be filename clashes in this folder. This new filename is packaged into a message along with other details from the request and sent into the response queue.

During the experiment, the artefacts that will be uploaded to the target cloud service provider will be the WebApp component, the Processor component and configuration files associated with these executables. If a target CSP does not have compatible queueing software then the queueing software used in building the reference application will be used and uploaded. The CSP's own queueing software is preferred in the experiment as it will test more of the inter-cloud framework's functionality and should have a positive effect on the overall application performance – as their queueing software will have been optimised for their infrastructure.

4.2 Experiment Design

The aim of the experiment is to evaluate the usability of a number of inter-cloud or cloud application interoperability frameworks. In the context of this research, usability means the following:

- *Documentation*: is the documentation a reference or a guide that starts small, builds concepts to enable the programmer to understand how the framework should be used to achieve the best from the software.

- *Application Programmers Interface*: is the API flexible enough so that a target application does not have to be substantially altered or restructured to use the framework?

- *Portability*: which cloud service providers does the framework support?

- *Stability*: Does it operate correctly on the stated supported Cloud Service Providers?

- *Auto Scaling*: If demand increased (or decreased) to the application, are the infrastructure resources automatically scaled up/scaled down by the framework and/or the Cloud Service Provider's automation.

For each inter-cloud framework, the process depicted in Figure 13 was followed, each framework was evaluated separately to keep views on a single framework independent of previously reviewed frameworks. After reviewing all nominated frameworks, a compare-and-contrast appraisal was conducted.

This process uses a qualitative approach to the documentation review which will include analysing the architecture to comment on the technologies and techniques used (such as design patterns); the programming languages the framework supports externally (i.e. C#, PHP, SQL Server etc.); requirements; limitations; accessibility (which cloud service providers are supported); legal issues identified/missing and the usability/readability of the documentation.
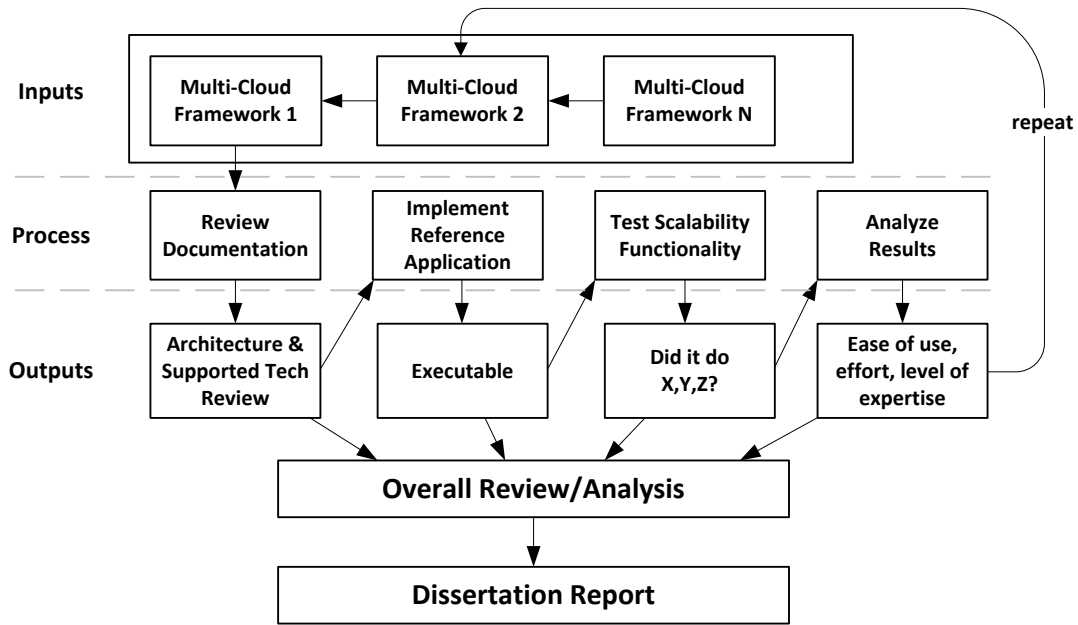
Figure 13 Experiment Process

The building and testing of the reference application brings a quantitative approach to this research. Each of the reference applications built (per framework) will be tested on a number of cloud service providers for interoperability and scalability.

A second qualitative approach is used after the experiment stage to give an overall review of the frameworks to appraise on ease of use, expertise level required, technical requirements, reportage functionality, infrastructure items supported, service items supported and cloud service providers supported.

4.3 *Source Code and Infrastructure Script Diversion Measurement*

Version Control Management (VCM) tools are used to measure the amount of change that occurred in the source code when adapting the reference application to use one of the frameworks. The branching functionality in VCMs are used in this regard. For the first and subsequent inter-cloud frameworks a new branch is created from the initial master branch which contained the completed reference application.

The master branch will be compared to each of the framework branches, the VCM tools will report how many files and how many lines have changed in each source code file or configuration file. A visual differencing tool is also used in the appraisal of the structural changes that were needed to adapt the original source code to use each framework.

4.4 Telemetry Measurements

Centralised log file management are basic features that CSPs provide to customers to allow them to monitor the performance and consumption of the virtual infrastructure resources that they create. These features are also used by the CSPs to determine what to charge their customers. These facilities are used in the experimentation phase to determine if the reference application scales up and back down in line with the demand on the application.

4.5 Documentation and Support Review

For the experiment, a review of the documentation for each inter-cloud framework is carried out. The documentation will be assessed on the following factors:

- Does documentation contain a guide section that brings the developer from basic concepts through to reasonable real world solutions?
- General readability.
- Does it contain a reference section? Is it logically organised and complete?
- Is the framework backed by support? Can questions be posed and answers received whether through an on-line forum, email or phone.

These factors will be appraised and rated on a factor of 0 to 10, zero is awarded when that item is omitted i.e. no support provided.

4.6 Chapter Summary

This chapter presented a high level overview of the design of the reference application and also experiment design. The next chapter goes into more detail, presenting how the different components of the experiment will be implemented.

5 IMPLEMENTATION

This chapter will catalogue and describe the software used in building the reference application and the software used for measuring the outcomes of the experiment.

5.1 Building the Reference Application

The Java programming language was chosen to build the reference application as a preliminary review of inter-cloud frameworks suggested they themselves are built using Java, some but not all of the frameworks have one or more other source code language APIs.

The Spring Framework[25] was also chosen to help build the reference application which is gaining popularity in many software development companies. Traditionally, the Spring Framework has a steep learning curve, however the Spring Boot Project[26] was enlisted to partially provide basic functionality of a typical application server such as user login security and enablement of the Thymeleaf template engine[27]. Thymeleaf allows developers to quickly build web pages using the Model View Controller (MVC) design pattern. The MVC pattern allows developers to separate business logic code from presentation code. This allows web page designers to work on HTML and Javascript code independently of back-end code which Java developers work on. With an MVC implementation in place, different template engines can be enlisted to provide different presentation code for different devices, i.e. native smart phone applications.

---

[25] http://projects.spring.io/spring-framework

[26] http://spring.io/spring-projects/spring-boot

[27] Thymeleaf: java XML/XHTML/HTML5 template engine, http://www.thymeleaf.org/

Spring also provides integration with the Maven[28] build and automated testing frameworks. The Eclipse[29] based Spring Tool Suite ™ (STS)[30] Integrated Development Environment (IDE) was also used in the construction and modification (for the inter-cloud frameworks) of the reference application.

The reference application comes in two compilation units or components: WebApp and Processor and both make use of open source queuing software to provide the asynchronous communication between these two executables. These are described in more detail in the next sections.

5.2 WebApp Component

The WebApp (application server) serves up web pages to users, or to be more specific, serves up web pages to the client browser that the user is operating. The web pages sent by the application server are constructed using Hyper Text Mark-up Language (HTML) and JavaScript client-side programming language to provide structure and interactivity to the web application.

The client web browser renders this content graphically in the browser window. The web page uses the jQuery[31] JavaScript library. This provides a vast array of functionality and allows the JavaScript programmer to easily select and manipulate any part of the Document Object Model (DOM). The DOM is the object model representation of the HTML document in the client browser's memory and the browser uses this to render the HTML document graphically. When jQuery and/or JavaScript manipulates the DOM, for example changing the colour of a table row, the browser instantly reflects/renders this change graphically to the user without having to reload the entire web page.

---

[28] http://maven.apache.org/

[29] Eclipse - The Eclipse Foundation open source community website, http://www.eclipse.org/

[30] https://spring.io/tools/sts

[31] http://jquery.com/

The image page uses AJAX technologies to communicate with the web server asynchronously. A jQuery call is made which uses the browser's XMLHttpRequest (XHR) API, to communicate with the remote application server using the HTTP protocol and waits for a response from the application server allowing for a timeout – should the server not respond.

Despite the naming (AJAX and XHR), there is no requirement for the application server to respond with XML (eXtensible Mark-up Language) content and can instead respond with JSON (JavaScript Object Notation) or just plain text. The client web page makes the request to apply an imaging effect to using the REST (Representational State Transfer) architectural style for distributed systems (Fielding, 2000) and expects a response from the server in the JSON format, see Table 4.

Table 4 Requests & responses to apply effect and return URL result

Request (REST)	GET http://localhost:8080/**effectrequest**/Blur/holiday.png
Response (JSON)	{"status":"**submitted**","requestId":"**a8e75e0e-ba32-43ab-bce8-b0f34588cd24**","created":1424719658469}
Request (REST)	http://localhost:8080/**effectfetch**/**a8e75e0e-ba32-43ab-bce8-b0f34588cd24/1424719658469**
Response (JSON)	{"status":"**notready**","requestId":"**a8e75e0e-ba32-43ab-bce8-b0f34588cd24**","url":"","elapsedTime":3107}
Request (REST)	http://localhost:8080/**effectfetch**/**a8e75e0e-ba32-43ab-bce8-b0f34588cd24/1424719658469**
Response (JSON)	{"status":"**completed**","requestId":"**a8e75e0e-ba32-43ab-bce8-b0f34588cd24**","url":"**/resources/processedfiles/a8e75e0e-ba32-43ab-bce8-b0f34588cd24.png**","elapsedTime":8227}

On the server side, the Java Servlet uses the Spring Framework MVC implementation. Figure 14 contains the controller code that responds to the user clicking on an image (A) on the main page. It packages the necessary parameters of the request (B), the security model properties (C) and the image file details on block storage (D) into the model object (E). Then sends it to the "image" view (F) which maps to the image.html file in Figure 15 (partial code shown).

```java
A @RequestMapping(value = "/image", method = RequestMethod.GET)
  public String image(Map<String, Object> model,
        B @RequestParam("name") String imageName) {
    C String user = getLoggedInUser();
      UserImageFileRepository store =
    D        new UserImageFileRepository(user, config.getImageFilesRoot());
      model.put("hostname", config.getHostName());
      model.put("user", user);
      model.put("imagesWebPath", imagesWebPath);
    E model.put("imagename", imageName);
      model.put("imageref", store.getWebPath(imageName));
      model.put("effects", new String[]{"Grayscale", "Invert", "Blur"});
      model.put("message", "");
    F return "image";
  }
```

Figure 14 Controller Mapping for viewing an image

The 'image' view is then sent back to the client browser after the Thymeleaf template engine replaces the 'th' namespace attributes and '${}' placeholders with the appropriate values from the model sent by the controller, see Figure 15.

```html
<div class="pull-right">
    <div id="imagecontainer">
        <img id="imagetag" th:alt-title="${imagename}" th:src="${imagesWebPath + '/' + imageref}" />
    </div>
    <div>
        <h2 th:text="${imagename}">image name does here</h2>
    </div>
</div>
<div class="pull-left">
    <form name="form" th:action="@{/image}" action="/image" method="POST">
        <fieldset>
            <input type="hidden" id="imagename" name="imagename" th:value="${imagename}" value="" />
            <input type="hidden" id="imagenew" name="imagenew" value="" />
            <input type="hidden" id="requestidentider" name="requestid" value="" />
            <input type="hidden" id="requestcreated" name="requestcreated" value="" />
            Effect :
            <select id="effect" name="effect">
                <option selected="selected" disabled="disabled" >Select</option>
                <option th:each="e : ${effects}" th:text="${e}"></option>
            </select>
        </fieldset>
        <input type="submit" id="save" value="Save" class="btn btn-primary" />
    </form>
</div>
```

Figure 15 View code using Thymeleaf templating to render HTML and JavaScript

When the server receives an AJAX asynchronous request from the client browser, i.e. to apply a certain effect to the current image, it takes the request information in Table 4 and repackages it into another message that the image processor will use to apply the special effect to the chosen image.

The main structure difference between the messages sent from the client browser and the messages sent to the processor is user information and the user file path. On the client side, only the image file name was specified (i.e. relative to the logged in user), the full server path to the image is specified in the message to the processor so that the processor can process images from any user. The messages to and from the processor (via the queue) are in the JSON format.

The reference application was built with the API to the RabbitMQ[32] open source software. RabbitMQ uses the Advanced Messaging Queuing Protocol (AMQP)[33] a recently approved international standard in messaging (Carol Geyer, 2014).

In particular, the reference application uses the Remote Procedure Call (RPC) pattern provided by RabbitMQ[34]. This automatically and dynamically creates the response queue. A correlation identifier is generated by the Java servlet and is sent along with the request details to the request queue. The correlation identifier is a UUID and becomes the request identifier on the client side of the application. In Figure 16, the Client is the WebApp component and the Server is the Processor component, the "rpc_queue" is the request queue and the "reply_to" is the response queue.

The Processor component is kept simple, it continually waits for a message from the request queue, processes the information contained in the message and returns to waiting for the next message from any user. When a message is received it unpacks the JSON formatted message and uses the properties of the message to load the image file into memory and apply the special effect algorithm to it.

---

[32] RabbitMQ - Messaging that just works, http://www.rabbitmq.com/

[33] http://www.amqp.org/
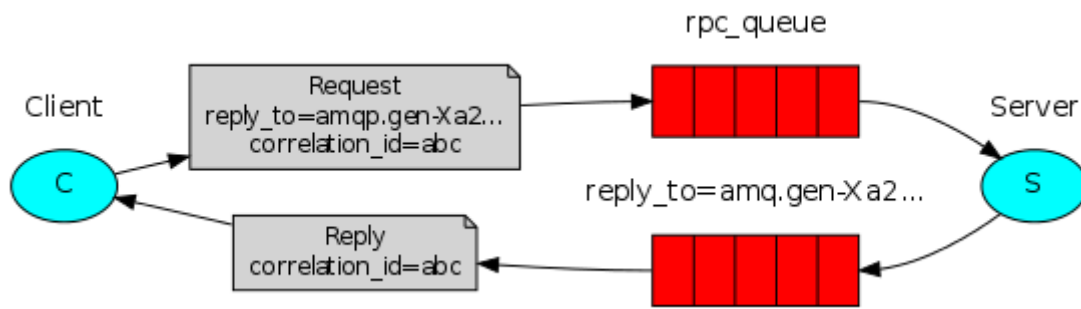
Figure 16 Remote Procedure Call messaging using RabbitMQ/AMQP
RabbitMQ - RabbitMQ tutorial - Remote procedure call (RPC)[34]

When complete, the image in memory is saved out to a processed folder on block storage (when in the cloud, to a Storage Area Network) and the filename contains the correlation identifier. The full filename was supplied by the client side (WebApp) in the message from the request queue. When the processor component completes the save to disk it sends the response message to the response queue and finally sends an acknowledgement to the request queue to indicate that the request message has been successfully processed and can be removed from that queue. The Processor returns to waiting for the next message from the request queue.

The asynchronous messaging between the browser and the servlet, the servlet and the request queue, the request queue and the Processor and then the return loop, enables the interactive nature of the reference application and also enables efficient use of computing resources.

5.3 Queue Component

The open source queuing software RabbitMQ was used in the implementation phase of this project and is used in the experimentation phase where an appropriate (uses the AMQP protocol) equivalent is not available through the inter-cloud frameworks. As described in the previous section and in Figure 16 the RPC pattern is used in this project. The RabbitMQ server also provides a web interface at URL http://localhost:15672/#/ which contains a useful telemetry tool showing the number of queued messages and the message rates per minute, as shown in Figure 17.

---

[34] http://www.rabbitmq.com/tutorials/tutorial-six-java.html

41

Figure 17 Message graphs in RabbitMQ web interface

5.4 Processor Component

The Processor component simply waits on the Request queue to supply a request sent to it by the WebApp component on behalf of the user using the client side browser. When one of these JSON formatted messages is received, its contents are read and acted upon and then the processor returns to waiting on the Request queue for the next imaging effect request.

The incoming message contains the full path to the source image file and the full path to the destination file which is used to save the in-memory image to block storage after the effect algorithm is applied. The message also contains one of the valid effects names. Table 5 contains the effect names and where the effect algorithms were sourced from. An EffectsApplicator object reads the image file into memory and applies the appropriate algorithm and then writes out the BufferedImage object to block storage where it is ready to be referenced by the client side browser.

Table 5 Effect algorithms used in Processor component

Effect Name	URL
Grayscale	`http://www.tutorialspoint.com/java_dip/grayscale_conversion.htm`
Blur	`http://www.javaworld.com/article/2076764/java-se/image-processing-with-java-2d.html`
Invert	`http://stackoverflow.com/questions/8662349/convert-negative-image-to-positive`

5.5 Choice of SCM

Git[35] Version Control Management and in particular the GitHub[36] service is used by many open source projects to manage and distribute projects. Many cloud service providers provide integration with GitHub which enables users or automation to directly pull the latest source code and infrastructure scripts direct from the repository into their service infrastructure. Plus many also provide "build in the cloud" services, for example DEV@cloud[37]. For these reasons, Git is used in this project to manage source code and configuration files. More importantly, it is also used to measure the diversion of the reference application source code for each inter-cloud framework adapted.

5.6 Test Automation

In the execution of the experiment, a test harness web page within the WebApp component of the reference application was used to generate many client side effect requests to the application server. The requests were generated using AJAX in the web browser and reusing some of the same mechanisms that were already built into the WebApp component. Figure 18 shows the test harness web page in the web browser.

---

[35] http://git-scm.com/

[36] GitHub - Build software better, together, https://github.com

[37] https://www.cloudbees.com/products/dev

Figure 18 Web page for the test harness functionality

The form fields on the left are for entering parameters to the test run. The buttons below control the initiation and cessation of the test. These buttons do not submit anything directly to the application server – event listeners are registered on these buttons with jQuery based functions. When the 'Start' button is clicked, the registered function reads the form values (at the same time disables them) and will initiate a number of effect requests based on the test parameters. The 'Shutdown' button will stop submitting any more requests to the application server and gracefully wait until all outstanding effect requests (maintained in an array/queue) have completed before re-enabling the form. The 'Abort' button also does not submit any more effect requests and it will ignore any outstanding requests (by clearing out the queue) and is used only when the browser starts to become unresponsive.

While the browser page is single threaded by design, many requests can still be queued up quickly. If running one test harness page is not generating enough requests to the application server simultaneously then one or more other browser tabs (or windows) are opened up with other instances of the test harness web page. These live in separate processes – this is at least true for the Google Chrome web browser.

The Selenium browser automation suite is used to demonstrate that the web application in the browser remains responsive when under load from the test harness.

The Selenium IDE[38], which is a plugin to the Firefox browser, is used to generate the initial browser automation scripts. Selenium IDE has its own domain specific language (DSL) to simulate the interaction a user would have with a browser. It uses the HTML code behind the browser page to specify actions on graphical items on the rendered browser page i.e. clicking a button.

Selenium IDE allows the tester to define a series of steps with each step performing an action on a target with a value. For example, (select, id=effect, label=Grayscale) performs a 'select' action on the DOM object that has a HTML id attribute of 'effect' and chooses the list item called 'Grayscale'.

Figure 19 shows a Selenium script to simulate a user logging in with a specific username and password, clicking the Login button, then waiting for the home page to render, then chooses an individual image and finally choosing the three different effects in succession.

Selenium IDE allows the tester to export the DSL script to various programming languages including Java. However, this option was not followed because on initial testing the resulting code did not execute all the steps in the script.

---

[38] http://www.seleniumhq.org/
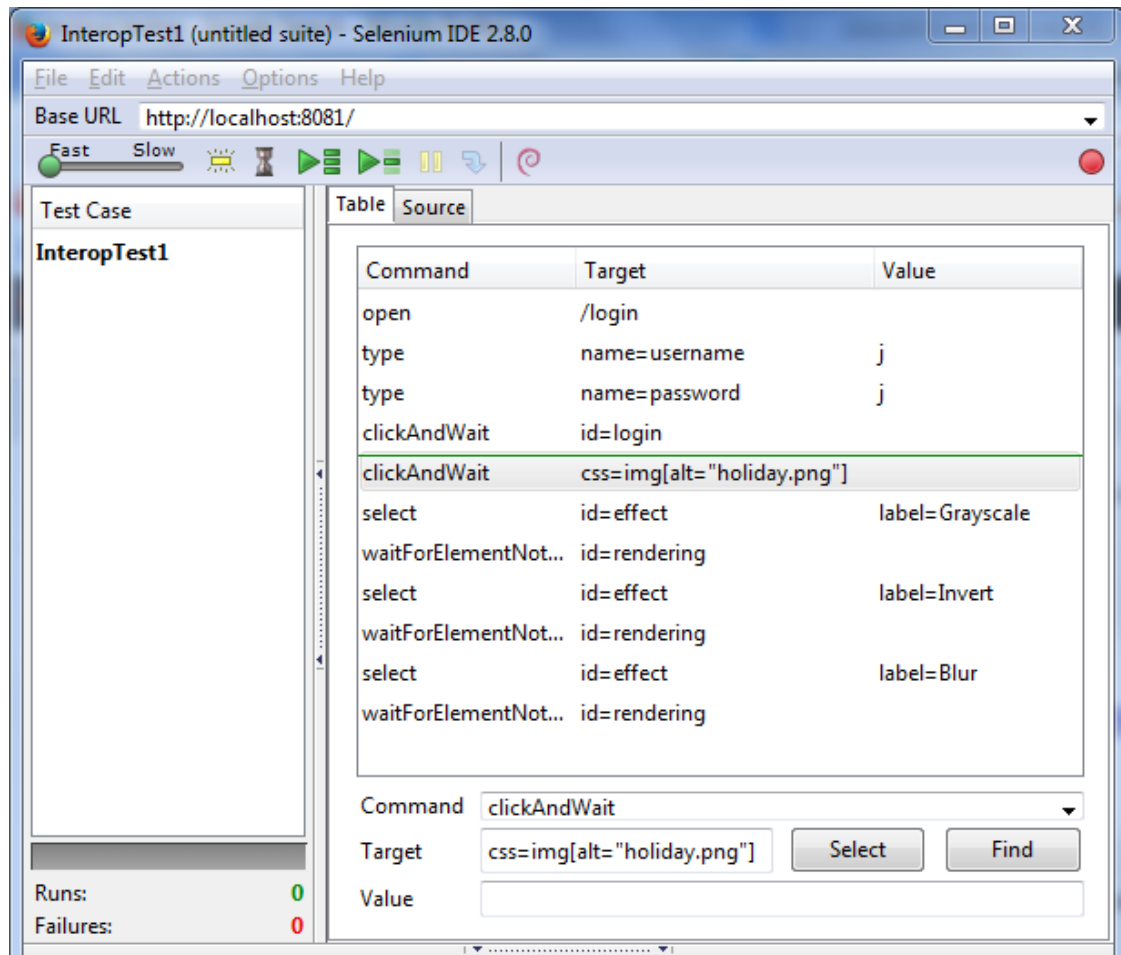
Figure 19 Selenium script to simulate user login, image select and selecting different special effects on the image.

5.7 Telemetry Implementation

Both the WebApp and Processor components write out standard Java logging information to log files. In particular the WebApp component writes to the log when it initially sends the special effect request to the request queue and when it receives the notification message that the image processing is complete (and the output image is ready for use). This information is used to determine how long an individual request took, then by looking at all request durations the rate of processing over time can be determined and graphed.

Figure 20 shows a filtered (UUID length reduced and some columns removed for presentation) sample output from the WebApp component. Each log entry has a timestamp, a logging level, PID (Process Identifier), the operation, outcome of the operation, the username, the request identifier and finally, if applicable the duration of the request on the server side.

```
2015-02-23 19:15:27.133 INFO effectrequest  success http://localhost:8080   j    2f7bf2f2-b1ce-5a61b4ee146c
2015-02-23 19:15:27.941 INFO effectfetch     completed  http://localhost:8080   j    2f7bf2f2-b1ce-5a61b4ee146c  1654
2015-02-23 19:15:29.456 INFO effectrequest  success http://localhost:8080   j    d9a03b60-e7e2-31bef07ef829
2015-02-23 19:15:30.601 INFO effectfetch     completed  http://localhost:8080   j    d9a03b60-e7e2-31bef07ef829  1147
2015-02-23 19:15:32.192 INFO effectrequest  success http://localhost:8080   j    72aacdcc-5085-c6ce1d30e0c2
2015-02-23 19:15:32.708 INFO effectfetch     completed  http://localhost:8080   j    72aacdcc-5085-c6ce1d30e0c2  518
```

Figure 20 Log file output from WebApp component

The number of effects completed per minute will be monitored in the experiment. Using the test automation described in Section 5.6, the number of effect requests will be increased and a drop in the number of effects completed will be observed. When this rate reaches a certain threshold, then the framework or CSP automation should step in to increase the number of computing resources (WebApp or Processor components) and the effects per minute should rise again.

When the browser automation is dialled down to reduce the number of incoming requests and the effects per minute should increase again and when it reaches a certain threshold the inter-cloud framework or CSP automation should shut down underutilised components and the effects per minute rate should drop again. Observing the effects per minute rise and fall as computing resources are added / withdrawn will reliably determine if scalability has been enabled by the inter-cloud framework.

5.8 Pre-Experiment Testing

The development environment used was Windows, so some initial testing on a Linux Server VM on Amazon Web Services was conducted before experiment start. This is to ensure that system incompatibilities did not become part of the experiment and have a negative reflection on the first multi-cloud framework adopted by the reference application.

After some initial testing, as outlined in Section 5.6, suitable applications were not found to render graph data directly from the log files. In the end, the *sed* (stream editor) Linux command was used to filter and convert the log entries to a tab separated file suitable for processing by the LiveGraph[39] java application.

5.9 Chapter Summary

This chapter went into detail about the implementation of the reference application and how the asynchronous messaging within the application works and provides efficiencies. Also covered was how the reference application is measured at the cloud service provider, how scalability is measured and the criteria for assessing the documentation associated with each inter-cloud framework. The next chapter covers the experiment execution and evaluation of the results.

---

[39] http://www.live-graph.org/

6   EXPERIMENTATION & EVALUATION

The purpose of this experiment is to evaluate a number of cloud interoperability or inter-cloud frameworks for use by Small to Medium Enterprises. Software developers within SMEs could use inter-cloud frameworks to build cloud based applications that can operate on multiple cloud service providers, thereby reducing or preventing *vendor lock-in*.

As outlined in Section 4.2, this experiment evaluates a number of aspects of multi-cloud frameworks to ascertain the level of effort it takes to successfully use them. For each framework the accompanying documentation is reviewed. The level of user support is evaluated. Stability and portability that the framework provides is established by porting the reference application to use each framework API and testing the resulting executables at a number of the CSPs supported by each framework.

The experiment evaluates the effort involved in adopting a framework quantitatively by comparing the source code of the reference application with the source code after the multi-cloud framework has been adopted into the codebase. This is done using the branching feature of the GIT version control system and the file differencing application Beyond Compare V4[40].

6.1 Experimentation

This section is segregated into each of the evaluated multi-cloud frameworks, each with a sub-heading for the evaluation criteria.

---

[40] http://www.scootersoftware.com/features.php

6.1.1 jclouds®

"Apache jclouds is an open source multi-cloud toolkit for the Java platform that gives you the freedom to create applications that are portable across clouds while giving you full control to use cloud-specific features." [41]

jclouds is an Apache Software Foundation (ASF)[42] top-level project since 2013[43]. ASF is a non-profit corporation in the United States and is a decentralized community of software developers. It hosts and supports many well-known open source software projects, most notably Apache HTTP Server[44] which runs many websites across the globe and Apache Hadoop[45] which is a popular Big Data tool for efficiently analysing large amounts of data across a cluster of machines.

Documentation

From the website menu there are two sets of documentation, "Getting Started" and "Documentation". The "Getting Started" section includes sub-headings: "What is Apache jclouds?"; "Installation Guide"; "Core Concepts"; "ComputeService"; "BlobStore"; "Examples". This section gives a brief introduction to the framework without much detail and appears designed to get you interested with the simplicity presented.

Upfront they do state that they only abstract or support three virtual infrastructure entities, namely *ComputeService*, *BlobStore* and *LoadBalancer*. However the documentation uses the analogy of using the Java Database Connection (JDBC) to connect and abstract from many database implementations but does not provide abstraction to connect to databases at various CSPs.

---

[41] https://github.com/jclouds/jclouds

[42] http://apache.org/

[43] http://incubator.apache.org/projects/#graduated

[44] http://httpd.apache.org/

[45] http://hadoop.apache.org/

Also, the framework allows access to the cloud provider specific APIs, if required to access functionality not yet abstracted. They do state that taking this approach will reduce portability, however it will be portable between CSPs that support the same native APIs, they give the example of Amazon Web Services and CSPs that support OpenStack[46] (for example RackSpace[47]).

The "Amazon Web Services: Getting Started Guide" [48] example page presents some steps and code but gives no context about how or what environment to run Java executable in. It may be difficult for a developer who has not used any CSP before to fully understand these pages.

On the website's home page and the "Providers" [49] page there are many supported CSPs listed. However by analysing the Providers page further, as laid out in Table 6, it is clear that the support for all virtual infrastructure items is not consistent across all CSPs.

A typical web application, similar to the reference application would require support for all three of these infrastructure items yet the framework only provides support under two CSP providers (Amazon Web Services and Rackspace). Finally there are several asterisks against some of the providers but these are not explained anywhere on this page.

There appears to be no readily available published books on jclouds after searching on Amazon[50] and the Book Depository[51].

---

[46] http://www.openstack.org/

[47] http://www.rackspace.com/

[48] http://jclouds.apache.org/guides/aws/

[49] http://jclouds.apache.org/reference/providers/

[50] http://www.amazon.com/ , http://www.amazon.co.uk/

[51] http://www.bookdepository.com/

Support

There are no structured forums present on the main website however they have an active Internet Relay Chat (IRC)[52] available which is used equally by jclouds developers and jclouds users. Signing up to the user@jclouds.apache.org mailing list allows you to receive updates from that channel.

Table 6 jclouds CSP by Infrastructure support matrix

	ComputeService	Blob Storage	LoadBalancer
AWS	Yes	Yes	Yes
CloudSigma	Yes		
DigitalOcean	Yes		
Docker	Yes		
ElasticHosts	Yes		
Go2Cloud	Yes		
GoGrid	Yes		
Google Compute Engine	Yes		
HP Helion	Yes	Yes	
Microsoft Azure		Yes	
Open Hosting	Yes		
Rackspace	Yes	Yes	Yes
ServerLove	Yes		
SkaliCloud	Yes		
SoftLayer	Yes		

Better structured examples were found on sites external to the jclouds at Rackspace Support[53] and StackOverflow[54]. No jclouds related content could be found at Amazon Web Services.

There is no offline telephone support either paid or unpaid. There are meet up events[55] however they are almost exclusively based in the United States.

---

[52] https://botbot.me/freenode/jclouds/

[53] https://developer.rackspace.com/blog/categories/jclouds/

[54] http://stackoverflow.com/questions/tagged/jclouds

Implementation

A standard approach when using a new API for the first time is to start with small examples as presented in the "Amazon Web Services: Getting Started Guide"[56]   and build on those before implementing in production code.

After successfully following the Maven project set-up, described on the "Installation Guide"[57] page, which executed a Maven build to download the libraries and other dependencies of jclouds. A simple project was created and the sample code was placed in a basic Java file and the provider specific API presented in the latter half of the example was omitted.

However the compilation produced an error (see Figure 21) stating that the newBlob() method does not exist. This was confirmed using the intelli-sense feature and referring to the API documentation for the BlobStore class[58].

A decision was made to attempt to use the blobBuilder() method instead. Based on the information given in the method documentation[59], an extra line was added which returns a Blob object – which the subsequent code expects, see Figure 22. This allowed for successful compilation and executing the code did upload a file to the *testbucket.net* S3 bucket in the account specified by *accessid* and *secretkey*.

However, downloading this file from the S3 service and opening up the image file produced an "invalid image" message; also the file sizes between the source image file and the uploaded file were different. Investigating one step further, by supplying a non-existing file name to the BlobBuilder object, the operation did not produce an error and a file was still uploaded to the S3 bucket and it could not be determined where the contents of this file came from.

---

[55] http://www.meetup.com/jclouds/

[56] http://jclouds.apache.org/guides/aws/

[57] http://jclouds.apache.org/start/install/

[58] http://jclouds.apache.org/reference/javadoc/1.8.x/org/jclouds/blobstore/BlobStore.html
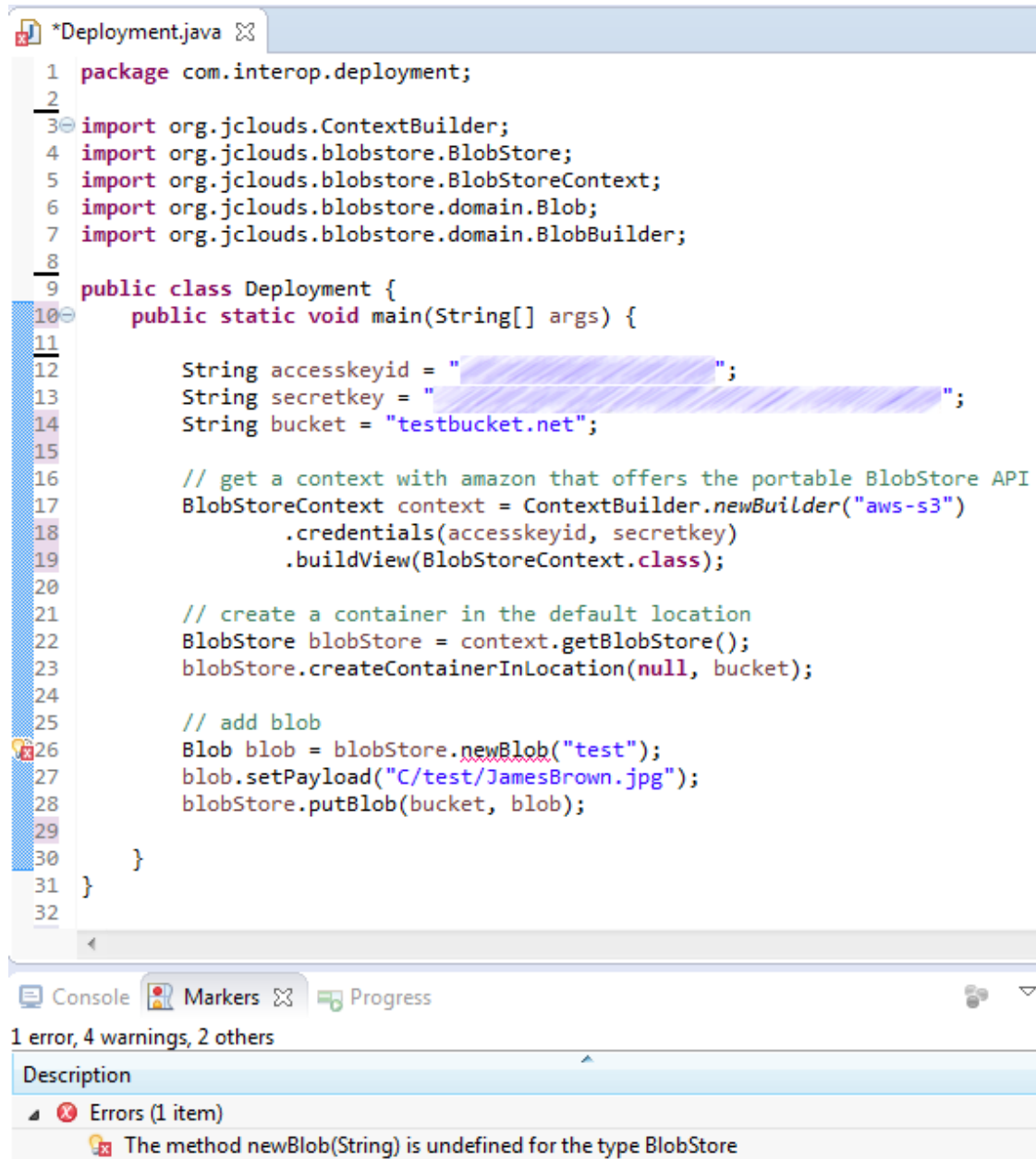
[59]

http://jclouds.apache.org/reference/javadoc/1.8.x/org/jclouds/blobstore/BlobStore.html#blobBuilder(java.lang.String)

A project management decision was made to discontinue the effort in adopting the reference application to use the jclouds framework as a result of these early unexpected failures outlined above.

```java
*Deployment.java ⊠
 1  package com.interop.deployment;
 2
 3⊖ import org.jclouds.ContextBuilder;
 4  import org.jclouds.blobstore.BlobStore;
 5  import org.jclouds.blobstore.BlobStoreContext;
 6  import org.jclouds.blobstore.domain.Blob;
 7  import org.jclouds.blobstore.domain.BlobBuilder;
 8
 9  public class Deployment {
10⊖     public static void main(String[] args) {
11
12          String accesskeyid = "                    ";
13          String secretkey = "                          ";
14          String bucket = "testbucket.net";
15
16          // get a context with amazon that offers the portable BlobStore API
17          BlobStoreContext context = ContextBuilder.newBuilder("aws-s3")
18                  .credentials(accesskeyid, secretkey)
19                  .buildView(BlobStoreContext.class);
20
21          // create a container in the default location
22          BlobStore blobStore = context.getBlobStore();
23          blobStore.createContainerInLocation(null, bucket);
24
25          // add blob
26          Blob blob = blobStore.newBlob("test");
27          blob.setPayload("C/test/JamesBrown.jpg");
28          blobStore.putBlob(bucket, blob);
29
30      }
31  }
32
```

```
Console   Markers ⊠   Progress
1 error, 4 warnings, 2 others
Description
⊿  ⊗ Errors (1 item)
       ⊠ The method newBlob(String) is undefined for the type BlobStore
```

Figure 21 Compilation error for jclouds S3 usage example

```java
// add blob
// Blob blob = blobStore.newBlob("jcloudstest.jpg");
BlobBuilder blobBuilder = blobStore.blobBuilder("jcloudstest.jpg");
Blob blob = blobBuilder.build();
blob.setPayload("C/test/JamesBrown.jpg");
blobStore.putBlob(bucket, blob);
```

Figure 22 Modified code to resolve error in Figure 21

54

6.1.2 brooklyn®

"Brooklyn is a framework for modelling, monitoring, and managing applications through autonomic blueprints."[60]

Brooklyn[61] became an Apache Software Foundation (ASF)[62] incubator project on May 1st, 2014 but has been in development prior to this – this was evident after attending the "FIA Dublin Workshop: Multi-Cloud Scenarios for the Future Internet"[63] as referenced in Chapter 3.

It is both a web application tool and an API. A blueprint is supplied to the web application UI which then starts and monitors your application at any of the supported locations which can be a cloud service provider, your localhost/laptop or your own internal servers or a mixture of these – allowing for "Hybrid Clouds"[64]. It also provides auto-scaling to meet demand and this can be defined by the user.

The blueprint allows you to describe the virtual infrastructure of your application in a hierarchy using the YAML (YAML Ain't Markup Language[65]) format. The sample shown in Figure 23 creates a cluster of server virtual machines with a load balancer and a database. In the *location* tag, you define where you want your application infrastructure to reside. In the example below, (A) defines the location to be *aws-ec2*, which is Amazon Web Service's Compute Cloud[66] and the associated identity and credential tags allow the Brooklyn web application access to the user's account on AWS.

---

[60] https://brooklyn.incubator.apache.org/learnmore/theory.html

[61] http://incubator.apache.org/projects/brooklyn.html

[62] http://apache.org/

[63] http://www.cloud4soa.eu/fia-multicloud

[64] http://searchcloudcomputing.techtarget.com/definition/hybrid-cloud

[65] http://yaml.org/

[66] http://aws.amazon.com/ec2/

```
name: My Web Cluster

location:
  jclouds:aws-ec2:
    identity: ████████████████
    credential: ████████████████████████████████
                                                    A

services:

- type: brooklyn.entity.webapp.ControlledDynamicWebAppCluster
  name: My Web                                                    B
  brooklyn.config:
    wars.root:
    http://search.maven.org/remotecontent?filepath=io/brooklyn/example/brooklyn-example
    -hello-world-sql-webapp/0.6.0/brooklyn-example-hello-world-sql-webapp-0.6.0.war
    java.sysprops:                                                                    C
      brooklyn.example.db.url: >
        $brooklyn:formatString("jdbc:%s%s?user=%s\\&password=%s",
        component("db").attributeWhenReady("datastore.url"),
        "visitors", "brooklyn", "████████")
                                                                    D
- type: brooklyn.entity.database.mysql.MySqlNode
  id: db                                                           E
  name: My DB
  brooklyn.config:
    creationScriptUrl: https://bit.ly/brooklyn-visitors-creation-script
                                                                    F
```

Figure 23 Sample Brooklyn Blueprint YAML script[67]

In the *services* tag, the different types of entities in the user's application are listed – in this definition there are two types. The first *type* tag (B) points to the Java class file *brooklyn.entity.webapp.ControlledDynamicWebAppCluster*, the second type tag (E) points to the *brooklyn.entity.database.mysql.MySqlNode* Java class file. These are Java Blueprints created by the user as reusable components that define the virtual infrastructure items required to run the components of the user's application. The tags below these *type* tags are parameters passed to the Java classes.

(C) points to the user's application WAR (**W**eb application **AR**chive) file[68] that will be deployed into the cluster, (D) defines how to connect to the associated database "My DB". (F) points to a SQL (Structured Query Language) file which is executed by the *brooklyn.entity.database.mysql.MySqlNode* Java class file to populate the database with table definitions and initial data.

---

[67] https://brooklyn.incubator.apache.org/v/latest/start/blueprints.html

[68] https://docs.oracle.com/cd/E19316-01/820-3748/aduvz/index.html

```
1   @Catalog(name="Controlled Dynamic Web-app Cluster", description="A cluster of
    load-balanced web-apps, which can be dynamically re-sized")
2   @ImplementedBy(ControlledDynamicWebAppClusterImpl.class)
3   public interface ControlledDynamicWebAppCluster extends DynamicGroup, Entity,
    Startable, Resizable, MemberReplaceable,
4           Group, ElasticJavaWebAppService, JavaWebAppService.CanDeployAndUndeploy,
            JavaWebAppService.CanRedeployAll {
5
6       @SetFromFlag("initialSize")
7       public static ConfigKey<Integer> INITIAL_SIZE = ConfigKeys.newConfigKeyWithDefault
        (Cluster.INITIAL_SIZE, 1);
8
9       @SetFromFlag("controller")
10      public static BasicAttributeSensorAndConfigKey<LoadBalancer> CONTROLLER = new
        BasicAttributeSensorAndConfigKey<LoadBalancer>(
11          LoadBalancer.class, "controlleddynamicwebappcluster.controller", "Controller
            for the cluster; if null a default will created (using controllerSpec)");
12
13      @SuppressWarnings({ "unchecked", "rawtypes" })
14      @SetFromFlag("controllerSpec")
15      public static BasicAttributeSensorAndConfigKey<EntitySpec<? extends LoadBalancer>>
         CONTROLLER_SPEC = new BasicAttributeSensorAndConfigKey(
16          EntitySpec.class, "controlleddynamicwebappcluster.controllerSpec", "Spec
            for creating the controller (if one not supplied explicitly); if null an
            NGINX instance will be created");
```

Figure 24 Excerpt from ControlledDynamicWebAppCluster.java example[69]

The two class files in this example use the Brooklyn API to create the infrastructure
items of the user application. An excerpt from
brooklyn.entity.webapp.ControlledDynamicWebAppCluster type is shown in Figure
24. Lines 6 and 7 define a default initial cluster size of 1, an **initialSize** tag could have
been specified in the YAML blueprint to override this, so an "initialSize: 3" would
create three server application VMs for the user's application. The remaining lines
define a load balancer for the cluster of VMs to distribute data traffic between them.
The classes inherited by *ControlledDynamicWebAppCluster* on line 1 helps to build
and abstract the infrastructure entities between different CSPs.

Similarly, Figure 25 shows an excerpt from *MySqlNode* class definition which clearly
shows that the MySQL server installation is completed by downloading from the URL
specified on lines 10 and 11. Lines 14 and 17 define how to configure the port that the
MySql sever listens to and where the database server saves its data files, respectively.

---

[69] https://github.com/apache/incubator-
brooklyn/blob/master/software/webapp/src/main/java/brooklyn/entity/webapp/ControlledDynamicWeb
AppCluster.java

57

```
1   @Catalog(name="MySql Node", description="MySql is an open source relational database
    management system (RDBMS)", iconUrl="classpath:///mysql-logo-110x57.png")
2   @ImplementedBy(MySqlNodeImpl.class)
3   public interface MySqlNode extends SoftwareProcess, HasShortName, DatastoreCommon,
    DatabaseNode {
4
5       // NOTE MySQL changes the minor version number of their GA release frequently, check
        for latest version if install fails
6       @SetFromFlag("version")
7       public static final ConfigKey<String> SUGGESTED_VERSION = ConfigKeys.
        newConfigKeyWithDefault(SoftwareProcess.SUGGESTED_VERSION, "5.5.37");
8
        //http://dev.mysql.com/get/Downloads/MySQL-5.5/mysql-5.5.21-linux2.6-i686.tar.gz/from/ht
        tp://gd.tuwien.ac.at/db/mysql/
9       @SetFromFlag("downloadUrl")
10      public static final BasicAttributeSensorAndConfigKey<String> DOWNLOAD_URL = new
        StringAttributeSensorAndConfigKey(
11              Attributes.DOWNLOAD_URL,
                "http://dev.mysql.com/get/Downloads/MySQL-5.5/mysql-${version}-${driver.osTag}.t
                ar.gz/from/${driver.mirrorUrl}");
12
13      @SetFromFlag("port")
14      public static final PortAttributeSensorAndConfigKey MYSQL_PORT = new
        PortAttributeSensorAndConfigKey("mysql.port", "MySQL port", PortRanges.fromString(
        "3306, 13306+"));
15
16      @SetFromFlag("dataDir")
17      public static final ConfigKey<String> DATA_DIR = ConfigKeys.newStringConfigKey(
18              "mysql.datadir", "Directory for writing data files", null);
```

Figure 25 Excerpt from MySqlNode example[70]

Documentation

The "THE THEORY BEHIND BROOKLYN" [71] page describes very clearly what
Brooklyn does and what it wants to achieve for the future and gives a degree of
confidence if an SME were to put effort into adopting this application tool and
framework.

---

[70] https://github.com/apache/incubator-
brooklyn/blob/master/software/database/src/main/java/brooklyn/entity/database/mysql/MySqlNode.java
[71] https://brooklyn.incubator.apache.org/learnmore/theory.html

58

The documentation is good, giving concise examples on how to get some basic infrastructure up and running quickly[72] to give confidence to the prospective user that this is a worthwhile investment. The User Guide[73] pages has many structured pages to guide the user though the process of defining your own Java Blueprints – to build a virtual infrastructure item portable between CSPs. The JavaDoc[74] documentation is also good and contains many written explanations over the basic class, method and parameter naming.

Documentation also exists externally with examples from Amazon Web Services[75]. There appears to no published books in relation to the Brooklyn multi-cloud framework after searching on Amazon and Book Repository.

After reviewing the documentation for Brooklyn, it was noted that Brooklyn uses the jcloud framework for parts if it functionality. Brooklyn however adds much more functionality. It is however, difficult to determine if *jclouds* limits the support for Brooklyn as analysed in Table 6.

Support

There are no structured forums on the Apache Brooklyn website. They have listed information for an IRC channel[76] but as of writing, this does not appear to be working and effectiveness could not be evaluated. External to the Apache Brooklyn website, help can be found on StackOverflow[77] and information about events/meet ups can be found on the Cloudsoft corporation website[78].

---

[72] https://brooklyn.incubator.apache.org/v/latest/start/running.html

[73] https://brooklyn.incubator.apache.org/v/latest/index.html

[74] http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html

[75] https://s3-eu-west-1.amazonaws.com/cloudsoft-amp/AMPGettingStarted+2.0.0-M1.pdf

[76] https://brooklyn.incubator.apache.org/community/irc.html

[77] http://stackoverflow.com/search?q=jclouds
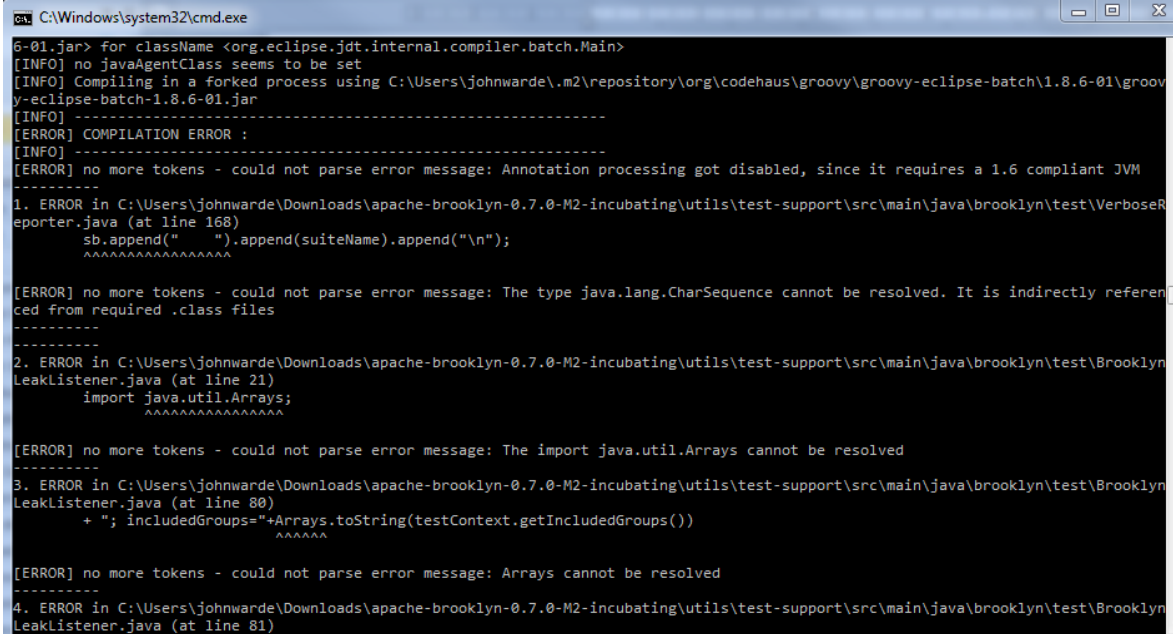
Implementation

The steps on the "RUNNING BROOKLYN"[79] page instruct the user how to build and execute the Brooklyn application. This was successful on an Ubuntu 14.04 Linux VM, however the same steps were not successful in a Windows environment producing compilation errors during the Maven build, see Figure 26.

Figure 26 Compilation errors when building Brooklyn on Windows

Again, by creating a small example first, it ensures that the basic application and configuration is working before attempting integration with the reference application. The steps in the "DEPLOYING BLUEPRINTS"[80] page suggests a basic blueprint (essentially the same as Figure 23 above, substituted with valid AWS *identity* and *credential* values) to get an initial cluster of VMs, a database server and a load balancer up and running on AWS. After supplying the blueprint to the Brooklyn web application you can see the application realised in the Brooklyn Web UI, see Figure 27.

---

[78] http://www.cloudsoftcorp.com/blog/category/events/

[79] https://brooklyn.incubator.apache.org/v/latest/start/running.html

[80] https://brooklyn.incubator.apache.org/v/latest/start/blueprints.html

Within minutes the corresponding virtual machines are visible via the Amazon Console (Web UI). The Brooklyn application logs into the created VM using Secure Shell (SSH) access to install and configure applications. While initial issues with SSH were overcome with help from Brooklyn developers on the mailing list, the set-up of applications within the VMs was prevented by AWS instance limits on the AWS account. Limit increases were requested to AWS Support, however three days of in-action prevented progress.
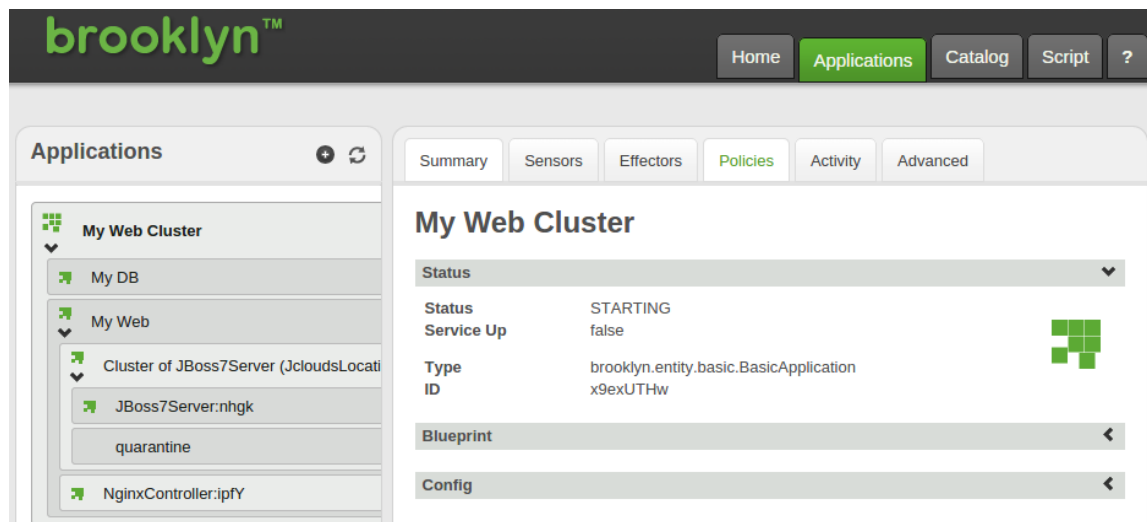
Figure 27 Test Blueprint application starting up on Brooklyn

Brooklyn allows the user to essentially use the same blueprint to create the same virtual infrastructure on another provider by supplying different location and credential details, Figure 28 shows a set-up for a Rackspace account.

Figure 28 Brooklyn blueprint location change

Unfortunately, details on how to obtain the credential value for the Rackspace account (named *secretkey* in Rackspace) was not found on Rackspace or Brooklyn support pages and this prevented the VMs from being created within the Rackspace account, see Figure 29.
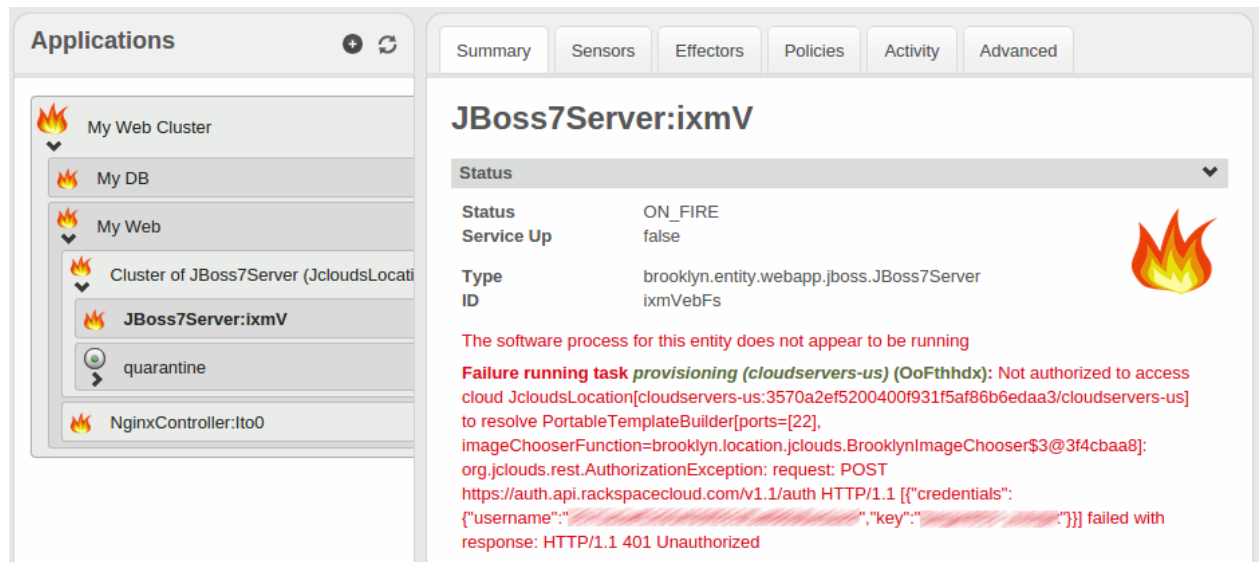
Figure 29 Failed Rackspace blueprint test

However, the simple location change in the YAML blueprint clearly shows the approach taken by Brooklyn to abstract across CSPs.

6.2 Evaluation

The selection criteria used for the reviewed inter-cloud frameworks are as follows:

- After the various presentations at the "FIA Dublin Workshop: Multi-Cloud Scenarios for the Future Internet"[81] the presentation by the Cloudsoft Corporation[82] (created and contributed Apache Brooklyn to ASF[83]) stood out as the most mature of the frameworks and is currently being used by global companies[84] such as BetFair[85].

- During the literature review jclouds was discovered. Due to the stated number of companies using it, the supported CSPs listed[86] and a post 1.0 version number, it was selected for evaluation.

---

[81] http://www.cloud4soa.eu/fia-multicloud

[82] http://www.cloudsoftcorp.com

[83] http://www.cloudsoftcorp.com/blog/2015/02/feb-22-26-ibm-interconnect-2015/

[84] http://www.cloudsoftcorp.com/wp-content/uploads/2013/11/Betfair-Appcloud-Platform-Cloudsoft-Case-Study-v1.0.pdf

[85] http://corporate.betfair.com/

[86] http://jclouds.apache.org/

As covered in Section 6.1.2 the Brooklyn framework is linked to the jclouds framework, therefore investing time in jclouds would not be wasted if an SME decided to wait until a post 1.0 version of Apache Brooklyn is released. However is not clear if the same virtual infrastructure and CSP support matrix for jclouds (in Table 6) applies to Apache Brooklyn.

Table 7 summarises the evaluation, as far as possible, under the headings outlined in Section 4.2.   Although experiments were not completed as planned, from reviewing the documentation and source code for Brooklyn, it is the recommended multi-cloud framework to provide abstraction and application lifecycle orchestration across multiple CSPs.

Table 7 Multi-cloud Evaluation Matrix

	jClouds®	**Brooklyn®**
License Type	Open Source Apache License, Version 2.0	Open Source Apache License v2.0.
Software Version	1.81	0.7.0-M2-incubating
On-line documentation	Minimal starter guides, JavaDoc with little usage explanation.	Good starter guides. JavaDoc has more than interface definitions in many places
On-line support	IRC, StackOverflow, Rackspace Support	IRC, StackOverflow
Offline support	None	Offered through Cloudsoft Corporation
Community Support	IRC, USA based events	IRC, UK based events
API Language Support	Java	Java
Supported Virtual Infrastructure	Compute Instances, Block Storage, Load Balancer	Numerous, see 'Entities' and 'Policies' tabs on the Catalog[87] page.
Coding Effort	Not confirmed	Not confirmed
Operational Stability	Incorrect documentation prevents confirming this.	Incorrect documentation prevents confirming this.
Portability	Not confirmed	Not confirmed
Scalability	Not confirmed	Not confirmed

---

[87] https://brooklyn.incubator.apache.org/learnmore/catalog/index.html

6.2.1 Factors in Choosing an Interoperability Framework

The following are factors that should be considered then reviewing a multi-cloud framework.

- What organisations are using the framework?

- Is it in use by other organisations for production tasks?

- How mature is it? A post 1.0 version number is not always a good indicator however the answers to the above questions would give a better indication of maturity.

- When dealing with a company providing support for a multi-cloud framework, ask questions. For a given application already using the multi-cloud framework and a specific CSP, what are the exact changes required to enable the set-up to work at another CSP. Whatever about the up-front effort to adopt an application to a multi-cloud framework, the effort to switch providers should be minimal. If only configuration changes are required then this is very positive.

- Use the processes described in Chapter 6 and the branching feature of source code management described in section 4.3. For example, i.e. run a pilot test on a small application or component, when completed a useful estimation of effort can be derived and applied to the remaining components.

6.2.2 Software Engineer/Architect Competence

When building any scalable in-house application, a senior engineer/architect would be required, even before considering a migration to the cloud. A junior software engineer with experience of design patterns could be capable of building the software that interfaces with the APIs of multi-cloud frameworks with guidance from the senior engineer. Both architect and engineer should have knowledge of infrastructure set-up and maintenance, or have access to a systems engineer that has, to aid the coding of deployment components in the chosen *multi-cloud* framework API, where required.

6.3 Chapter Summary

This chapter first summarised the aim of the experiment and then reviewed each of the target multi-cloud frameworks under several headings.   These headings include a general introduction to the multi-cloud framework and what it achieves; a review of the documentation available to help the user; the levels of user support and finally the steps involved in modifying the reference application to use the framework to enable it to work in multiple CSPs.

After covering each of the multi-cloud frameworks an evaluation and analysis was carried out to ascertain the overall maturity of the frameworks and what aspects should be considered when evaluating the frameworks in a real-world SME environment. The final chapter will summarise and conclude this dissertation.

7   CONCLUSIONS AND FUTURE WORK

There are many companies either using, evaluating or considering cloud computing to help them stay competitive in the marketplace as ICT technologies continue to push into new markets. One major concern is vendor lock in. Multi-cloud frameworks seek to address this issue and this research assesses the maturity of some of those frameworks.

7.1 Research Overview

This research carried out an investigation into Cloud Interoperability (or *multi-cloud*) frameworks to assess the usability targeted at Small to Medium Enterprises. By adopting a multi-cloud framework, SMEs can gain an economic advantage over competitors or bigger players in the market when starting out. They also future-proof their technology as multi-cloud frameworks would be better placed to adopt new innovations as they continue to arise – because these frameworks are designed to be flexible.

The aim of this work was to garner knowledge around the research and development of multi-cloud frameworks and adopt the multi-cloud frameworks into a reference application. The results of which allowed a quantitative assessment of the maturity of the frameworks.

7.2 Contributions to the Body of Knowledge

The inter-cloud frameworks reviewed are not in common use in the commercial software sector as can be gleaned by reviewing popular tags[88] on the StackOverflow website – the jclouds and brooklyn tags numbers are low compared to other technologies referenced on this website. Currently, there does not appear to be any commercial project work around these technologies in Ireland[89]. By contributing this research work alongside other published work, this dissertation strives to raise awareness of the availability of multi-cloud frameworks.

Chapter 5 describes the open source RabbitMQ asynchronous messaging application using the standardised AMQP protocol. Only released in May 2014, it provides a robust and scalable RPC implementation and provides a cost effective method of scaling compute intensive applications in many programming languages[90].

7.3 Experimentation, Evaluation and Limitations

The experimentation phase of this work was not completed to the level anticipated at the outset of this project – this was due to a number of factors which added time and delayed milestones.

As the majority of research based source code is in Java and executed in Linux based operating systems, an initial attempt was made to develop the reference application in a Linux environment (due to resource constraints this was done through the hosted hypervisor VirtualBox and an Ubuntu 14.04 Linux guest virtual machine).

---

[88] http://stackoverflow.com/tags

[89] https://www.google.ie/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#safe=off&q=jclouds+%2Bireland+-aws, http://jobsearch.monster.ie/jobs/?q=jclouds&cy=ie, http://jobsearch.monster.ie/jobs/?q=brooklyn&cy=ie

[90] http://www.rabbitmq.com/devtools.html

However, many problems were experienced in getting the STS/Eclipse IDE tool to work in Ubuntu. An attempt was also made to set up a development environment on a Macbook but this too ran into problems and upon learning that MacOS X operating system is based on BSD Unix and not a Linux derivative, this was abandoned. The reference application was developed in a Windows environment and pre-experiment testing showed that all executables (JAR files) executed successfully in Windows and Linux based operating systems.

The effort involved in developing the reference application was underestimated. This was partially due to the steep learning curve of the Java Spring Framework, however once getting past a point, the Spring Framework does have advantages in reducing the amount of code required to add new features.

During the experiment phase, while evaluating the Brooklyn multi-cloud framework, the Maven build was unsuccessful on the Windows platform (see Figure 26) but was successfully built on the Linux VM environment. However, the VM used experienced problems and on a number of occasions, failing to boot up – adding to the delays late in the experiment stage. After some effort these were resolved each time.

A limitation of this research work is the number of multi-cloud frameworks that were assessed in the experiment stage. A minimum of three frameworks was the aim for this work however five or more would be ideal to give a more balanced view of the maturity of the frameworks in the multi-cloud arena.

Multi-cloud frameworks are complex because of what they are trying to achieve. On one level they are providing device (virtual hardware) independence, on another level operating system independence and finally independence from the different APIs used by CSPs. The effort in adapting and configuring the reference applications to the multi-cloud frameworks was underestimated.

Building the reference application was one way to evaluate these frameworks but it does not exercise the full capabilities, nor did the experiment fully evaluate the stability of each framework in its entirety. Finally every application developed has different requirements and can accept different trade-offs when it comes to implementation, for example social networking web sites can tolerate missing a few updates from users whereas a banking application cannot.

7.4 Future Work & Research

The following are suggestions which can add to this research to gain further knowledge and promote adoption of multi-cloud frameworks as a viable technology to reduce vendor-lock-in in the cloud computing market and continue to encourage competitiveness and innovation within that marketspace:

- Applying the same approach used in this research to more multi-cloud frameworks would better inform users of the most appropriate framework to use for their application.
- Security is another major concern for companies and oftentimes a barrier to entry, an evaluation of the security of the frameworks reviewed in this work plus others would be of enormous benefit.
- A number of the frameworks highlighted in literature review talked about auto-negotiation of contracts based on Service Level Agreements. This functionality is comparable to the automatic buying and selling of currencies and company stock by major brokers. This would be an interesting area of research.
- Location of personal identifiable information is important in some jurisdictions and especially in the European Union. Being able to define which data can reside where is useful in this respect.

BIBLIOGRAPHY

Arif Mohamed, 2009. A history of cloud computing [WWW Document]. URL http://www.computerweekly.com/feature/A-history-of-cloud-computing

Atwood, J., 2008. Hardware is Cheap, Programmers are Expensive. Coding Horror.

Bernstein, D., Vij, D., 2010. Intercloud Directory and Exchange Protocol Detail Using XMPP and RDF, in: 2010 6th World Congress on Services (SERVICES-1). Presented at the 2010 6th World Congress on Services (SERVICES-1), pp. 431 –438. doi:10.1109/SERVICES.2010.131

Carol Geyer, 2014. ISO and IEC Approve OASIS AMQP Advanced Message Queuing Protocol [WWW Document]. URL https://www.oasis-open.org/news/pr/iso-and-iec-approve-oasis-amqp-advanced-message-queuing-protocol

Cavoukian, A., 2008. Privacy in the clouds. Identity Inf. Soc. 1, 89–108. doi:10.1007/s12394-008-0005-z

Chang, B.R., Tsai, H.-F., Chen, C.-M., 2013. Evaluation of virtual machine performance and virtualized consolidation ratio in cloud computing system. J. Inf. Hiding Multimed. Signal Process. 4, 192–200.

Chapman, C., Emmerich, W., Marquez, F.G., Clayman, S., Galis, A., 2010. Elastic service definition in computational clouds, in: Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP. Presented at the Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP, pp. 327–334. doi:10.1109/NOMSW.2010.5486555

Columbus, L., 2013. Roundup of Cloud Computing Forecasts Update, 2013 [WWW Document]. Forbes. URL http://www.forbes.com/sites/louiscolumbus/2013/11/16/roundup-of-cloud-computing-forecasts-update-2013/ (accessed 12.2.13).

Fielding, R.T., 2000. Architectural styles and the design of network-based software architectures. University of California, Irvine.

Foley, M.J., 2014. Docker container support coming to Microsoft's next Windows Server release [WWW Document]. ZDNet. URL http://www.zdnet.com/article/docker-container-support-coming-to-microsofts-next-windows-server-release/ (accessed 12.22.14).

Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., Vaquero, L.M., 2009. Service specification in cloud environments based on extensions to open standards, in: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, COMSWARE '09. ACM, New York, NY, USA, pp. 19:1–19:12. doi:10.1145/1621890.1621915

Kar, S., 2014. Amazon Redefining Cloud Services With AWS Lambda | CloudTimes.

Lakhani, J., Bheda, H., 2012. Scheduling technique of data intensive application workflows in Cloud computing, in: 2012 Nirma University International Conference on Engineering (NUiCONE). Presented at the 2012 Nirma University International Conference on Engineering (NUiCONE), pp. 1–5. doi:10.1109/NUICONE.2012.6493191

Loutas, N., Kamateri, E., Bosi, F., Tarabanis, K., 2011a. Cloud Computing Interoperability: The State of Play, in: 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom). Presented at the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 752 –757. doi:10.1109/CloudCom.2011.116

Loutas, N., Kamateri, E., Tarabanis, K., 2011b. Cloud4SOA Cloud Semantic Interoperability Framework.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., Ghalsasi, A., 2011. Cloud computing — The business perspective. Decis. Support Syst. 51, 176–189. doi:10.1016/j.dss.2010.12.006

Mell, P., Grance, T., 2011. The NIST definition of cloud computing.

Ranabahu, A., Sheth, A., 2010. Semantics Centric Solutions for Application and Data Portability in Cloud Computing, in: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom). Presented at the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), pp. 234 –241. doi:10.1109/CloudCom.2010.48

Tsai, W.-T., Sun, X., Balasooriya, J., 2010. Service-Oriented Cloud Computing Architecture, in: 2010 Seventh International Conference on Information Technology: New Generations (ITNG). Presented at the 2010 Seventh International Conference on Information Technology: New Generations (ITNG), pp. 684 –689. doi:10.1109/ITNG.2010.214

Vanbrabant, B., Joosen, W., 2014. Configuration Management As a Multi-cloud Enabler, in: Proceedings of the 2Nd International Workshop on CrossCloud Systems, CCB '14. ACM, New York, NY, USA, pp. 1:1–1:3. doi:10.1145/2676662.2676672

Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M., 2008. A Break in the Clouds: Towards a Cloud Definition. SIGCOMM Comput Commun Rev 39, 50–55. doi:10.1145/1496091.1496100

Appendix A: Software/Technologies/Tools Used

Name	Category	Home page
Java JDK 1.8 (Windows)	Programming Language	http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Spring Framework	API	http://sping.io
Thymeleaf	Java Template Engine	http://thymeleaf.org
Firefox Browser	Internet Browser	https://www.mozilla.org/en-US/firefox/new/
Selemium IDE	Firefox Extension	http://www.seleniumhq.org/
AWS CLI	API	http://aws.amazon.com/cli/
Git	SCM Client	http://git-scm.com/
GitHub	SCM Client	http://github.com/
Beyond Compare 4	File/Folder Comparison	http://www.scootersoftware.com/
Windows 7	OS	http://microsoft.com
VirtualBox 4.3	Virtualisation	http://virtualbox.org/
Ubuntu 14.04 Linux Desktop VM (on above)	OS	http://ubuntu.com

Appendix B: REFERENCE APPLICATION SOURCE CODE

The following is source from the Reference Application, as covered in Sections 5.2 and 5.4, before it was modified to adopt the multi-cloud frameworks. Standard/boilerplate code files is omitted, these are mainly JavaBeans files which just define access to properties of objects.

WebApp Component

webapp/src/main/java/com.interop.webapp/WebApp.java

```java
/*
 * Copyright 2012-2014 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.interop.webapp;

import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static java.nio.file.StandardCopyOption.*;

import javax.annotation.PostConstruct;
import javax.sql.DataSource;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.rabbitmq.client.*;
import com.rabbitmq.client.AMQP.BasicProperties;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.security.SecurityProperties;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.core.annotation.Order;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.context.SecurityContextHolder;
```

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.config.annotation.ContentNegotiationConfigurer;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@EnableAutoConfiguration
@ComponentScan
@Controller
public class WebApp extends WebMvcConfigurerAdapter {
    @Autowired
    private WebAppConfig config;

    static String imagesWebPath = "resources";
    static String processedFilesWebPath = "processedfiles";
    static String imagesWebPathMask = "/" + imagesWebPath + "/**";

    static final Logger log = LoggerFactory.getLogger(WebApp.class);

    private Connection connection;
    private Channel channel;
    private String replyQueueName;
    private QueueingConsumer consumer;

    @Override
    public void configureContentNegotiation(
            ContentNegotiationConfigurer configurer) {
        configurer.favorPathExtension(false);
    }

    @PostConstruct
    public void setUpQueue() throws Exception {

        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(config.getQueueHostName());
        connection = factory.newConnection();
        channel = connection.createChannel();

        replyQueueName = channel.queueDeclare().getQueue();
        consumer = new QueueingConsumer(channel);
        channel.basicConsume(replyQueueName, true, consumer);
        // Create the folder that will hold the processed files
        String foldername = config.getImageFilesRoot() + '/' + processedFilesWebPath;
        File folder = new File(URI.create(foldername));
        if (folder.exists()) {
                return;
        }
        log.info("creating directory: " + foldername);
        try {
            folder.mkdir();
        }
        catch (SecurityException se) {
            log.error(String.format("creating directory: %s (%s)", foldername,
se.getMessage()));
        }
        catch (Exception e) {
            log.error(String.format("creating directory: %s (%s)", foldername,
e.getMessage()));
        }
    }


    /**
     * @return String - logged in user
     */
    private String getLoggedInUser() {
            return SecurityContextHolder.getContext().getAuthentication().getName();
    }
```

```java
@RequestMapping("/")
public String home(Map<String, Object> model) {
    String user = getLoggedInUser();
        UserImageFileRepository store =
                        new UserImageFileRepository(user, config.getImageFilesRoot());
        List<ImageDetailBean> collection = store.getWebPaths();
        model.put("user", user);
        model.put("imagesWebPath", imagesWebPath);
        model.put("imagerefs", collection);
        model.put("count", collection.size());
        return "home";
}


@RequestMapping(value = "/image", method = RequestMethod.GET)
public String image(Map<String, Object> model,
                @RequestParam("name") String imageName) {
    String user = getLoggedInUser();
        UserImageFileRepository store =
                        new UserImageFileRepository(user, config.getImageFilesRoot());
        model.put("hostname", config.getHostName());
        model.put("user", user);
        model.put("imagesWebPath", imagesWebPath);
        model.put("imagename", imageName);
        model.put("imageref", store.getWebPath(imageName));
        model.put("effects", new String[]{"Grayscale", "Invert", "Blur"});
        model.put("message", "");
        return "image";
}


@RequestMapping(value = "/image", method = RequestMethod.POST)
public String imageSave(Map<String, Object> model,
                @RequestParam(value="imagename", defaultValue="") String imageName,
                @RequestParam(value="imagenew", defaultValue="") String imageNew) {
        String user = getLoggedInUser();
        String filesRoot = config.getImageFilesRoot();
        if (!imageNew.equals("")) {
                // We have a new image from processing, need to replace the original.
                UserImageFileRepository store =
                                new UserImageFileRepository(user, filesRoot);
                String destFilename = store.getPath(imageName);
                String srcFilename = filesRoot + '/' + processedFilesWebPath +
                                '/' + imageNew.substring(imageNew.lastIndexOf('/') +
1);
                try {
                        Files.move(Paths.get(URI.create(srcFilename)),
                                        Paths.get(URI.create(destFilename)),
                                        REPLACE_EXISTING );
                } catch (Exception e) {
                        log.error(String.format("Failed to copy [%s] to [%s] (%s)",
                                        srcFilename, destFilename, e.getMessage()));
                }
        }
        return "redirect:/";
}


@RequestMapping(value = "/upload", method = RequestMethod.GET)
public String upload(Map<String, Object> model) {
        return "upload";
}

@RequestMapping(value = "/upload", method = RequestMethod.POST)
public String upload(Map<String, Object> model,
                @RequestParam(value="imagename", defaultValue="") String imagename,
                @RequestParam("uploadfile") MultipartFile uploadedfile) {
        String user = getLoggedInUser();
        UserImageFileRepository store =
                        new UserImageFileRepository(user, config.getImageFilesRoot());
        Boolean success = store.newUpload(imagename, uploadedfile);
        model.put("success", success);
```

```java
                model.put("imagename", imagename);
                return "upload";
        }


        @RequestMapping("/testharness")
        public String testHarness(Map<String, Object> model) {
            String user = getLoggedInUser();
                model.put("user", user);
                model.put("hostname", config.getHostName());
                model.put("processingtimeout", config.getProcessingTimeout());
                return "testharness";
        }



        @RequestMapping("/logout")
        public String logout(Map<String, Object> model) {
                SecurityContextHolder.getContext()
                        .getAuthentication().setAuthenticated(false);
                return "redirect:/";
        }



/*
 * REST methods for AJAX calls from client
 */

    /*
     * Because we are sending in a filename, we needed to add the
     * regular expression to the end of the RequestMapping because Spring
     * defaults to a regular expression of [^.]* (everything but a period)
     * However, because we a sending names of image/binary files the Spring
     * component MappingJackson2HttpMessageConverter determines that the JSON
     * content that we are sending back does not match the content requested
     * and sends back a HTTP error code 406 back to the client.  To mitigate
     * this we need to define the configureContentNegotiation() method above.
     */
    @RequestMapping(value="/effectrequest/{name}/{imagename:[a-zA-Z0-9%\\.]*}",
                headers="Accept=*/*", method=RequestMethod.GET,
                produces = "application/json")
    public @ResponseBody EffectRequest effectRequest(
                @PathVariable("name") String name,
                @PathVariable("imagename") String imageName)
    {
            String user = getLoggedInUser();
                String hostName = this.config.getHostName();
                UserImageFileRepository store =
                            new UserImageFileRepository(user, config.getImageFilesRoot());

                String status = "submitted";
        String correlationId = java.util.UUID.randomUUID().toString();


                Map<String, Object> mapObject = new HashMap<String, Object>();
                String imageFullPath = store.getPath(imageName);
                String ext = imageFullPath.substring(imageFullPath.lastIndexOf('.') + 1);
                String processedFileName = String.format("%s/%s/%s.%s",
                            config.getImageFilesRoot(),
                            processedFilesWebPath,
                            correlationId, ext);

                long created = System.currentTimeMillis();
                mapObject.put("correlationId", correlationId);
                mapObject.put("user", user);
                mapObject.put("inputPath", imageFullPath);
                mapObject.put("ouputPath", processedFileName);
                mapObject.put("effectName", name);
                mapObject.put("requestHost", hostName);
                mapObject.put("requestCreated", created);

                JsonWrapper objJson = new JsonWrapper();
                String message = objJson.toJson(mapObject);
```

```java
        BasicProperties props = new BasicProperties
                                .Builder()
                                .correlationId(correlationId)
                                .replyTo(replyQueueName)
                                .build();
        try {
                channel.basicPublish("", config.getQueueName(), props,
message.getBytes());
                        log.info(String.format("effectrequest\tsuccess\t%s\t%s\t%s",
                                        hostName, user, correlationId));
                } catch (IOException e) {
                        log.error(String.format("effectrequest\tfail\t%s\t%s\t%s\t%s",
                                        hostName, user, correlationId, e.getMessage()));
                        status = "failed";
                }
        return new EffectRequest(status, correlationId, created);
    }


    @RequestMapping(value="/effectfetch/{requestid}/{requestcreated}",
                headers="Accept=*/*", method=RequestMethod.GET,
                produces = "application/json")
    public @ResponseBody EffectFetch effectFetch(
                @PathVariable("requestid") String correlationId,
                @PathVariable("requestcreated") long requestCreated)
    {
            String user = getLoggedInUser();
                String hostName = this.config.getHostName();
        String status = "notready";
        String url = "";
        long millisecondsElapsed = 0;

        String response = null;
        QueueingConsumer.Delivery delivery = null;
        try {
                    delivery = consumer.nextDelivery(333);
                if (delivery != null &&
                delivery.getProperties().getCorrelationId().equals(correlationId)) {
                    response = new String(delivery.getBody());
                        JsonWrapper objJson = new JsonWrapper();
                        Map<String, Object> mapObject = objJson.fromJson(response);
                    status = (String) mapObject.get("status");
                    String newFileName = (String) mapObject.get("ouputPath");
                    newFileName =
                                newFileName.substring(newFileName.lastIndexOf('/') + 1);
                        url = "/" + imagesWebPath + "/" +
                                processedFilesWebPath + "/" + newFileName;

                    long curr = System.currentTimeMillis();
                    millisecondsElapsed = curr - requestCreated;    //Time difference in
milliseconds

                        log.info(String.format("effectfetch\t%s\t%s\t%s\t%s\t%d",
                                            status, hostName, user, correlationId,
millisecondsElapsed));
                } else {
                millisecondsElapsed = System.currentTimeMillis() - requestCreated;
                if (millisecondsElapsed > config.getProcessingTimeout()) {
                            status = "failed";
                            log.error(String.format("effectrequest\tfail\t%s\t%s\t%s\t%s",
                                            hostName, user, correlationId, "Max time
exceeded"));
                }
                }
                } catch (ShutdownSignalException e) {
                        status = "failed";
                        log.error(String.format("effectrequest\tqueuefail\t%s\t%s\t%s\t%s",
                                        hostName, user, correlationId, e.getMessage()));

                } catch (ConsumerCancelledException e) {
                        status = "failed";
                        log.error(String.format("effectrequest\tqueuefail\t%s\t%s\t%s\t%s",
```

```java
                              hostName, user, correlationId, e.getMessage()));

                } catch (InterruptedException e) {
                        status = "failed";
                        log.error(String.format("effectrequest\tqueuefail\t%s\t%s\t%s\t%s",
                                        hostName, user, correlationId, e.getMessage()));

                }
        return new EffectFetch(status, correlationId, url, millisecondsElapsed);
}


        public static void main(String[] args) throws Exception {
                new SpringApplicationBuilder(WebApp.class,
                                "classpath:/META-INF/application-context.xml").run(args);
        }

        @Override
        public void addViewControllers(ViewControllerRegistry registry) {
                registry.addViewController("/login").setViewName("login");
        }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler(imagesWebPathMask)
        .addResourceLocations(config.getImageFilesRoot() + '/');
    }

        @Bean
        public ApplicationSecurity applicationSecurity() {
                return new ApplicationSecurity();
        }

        @Order(SecurityProperties.ACCESS_OVERRIDE_ORDER)
        protected static class ApplicationSecurity extends WebSecurityConfigurerAdapter {

                @Autowired
                private SecurityProperties security;

                @Autowired
                private DataSource dataSource;

                @Override
                protected void configure(HttpSecurity http) throws Exception {
                        http
                                .authorizeRequests()
                                .antMatchers("/css/**",
imagesWebPathMask).permitAll().anyRequest()

        .fullyAuthenticated().and().formLogin().loginPage("/login")
                                        .failureUrl("/login?error").permitAll();
                }

                @Override
                public void configure(AuthenticationManagerBuilder auth) throws Exception {
                        auth.jdbcAuthentication().dataSource(this.dataSource);
                }

        }
```

webapp/src/main/java/com.interop.webapp/UserImageRepository.java

```java
package com.interop.webapp;

import java.util.List;

import org.springframework.web.multipart.MultipartFile;


public abstract class UserImageRepository {
        protected String owner;
        public UserImageRepository(String owner) {
                this.owner = owner;
        }

        abstract public List<ImageDetailBean> getWebPaths();
        abstract public String getWebPath(String imagename);
        abstract public String getPath(String imagename);
        abstract public Boolean newUpload(String imagename, MultipartFile file);
        abstract public void removeImage(String imagename);
}
```

webapp/src/main/java/com.interop.webapp/UserImageFileRepository.java

```java
package com.interop.webapp;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import java.util.List;

import org.apache.log4j.Logger;
import org.springframework.web.multipart.MultipartFile;


public final class UserImageFileRepository extends UserImageRepository {
        // Root of the file repository.
        private String root;

        static Logger log = Logger.getLogger(WebApp.class.getName());

        /**
         * @param owner
         */
        public UserImageFileRepository(String owner, String repositoryRoot) {
                super(owner);
                this.root = repositoryRoot;
        }

        /* (non-Javadoc)
         * @see com.interop.webapp.UserImageRepository#getWebPaths()
         */
        @Override
        public List<ImageDetailBean> getWebPaths() {
                ensureOwnerFolder();
                String absolutePath;
                List<ImageDetailBean> collection = new ArrayList<ImageDetailBean>();
                File folder = new File(URI.create(getUserFolder()));
                File[] listOfFiles = folder.listFiles();
                for (int i = 0; i < listOfFiles.length; i++)
                {
                    if (listOfFiles[i].isFile())
                    {
                            ImageDetailBean img = new ImageDetailBean();
                            img.setImageName(listOfFiles[i].getName());
                            absolutePath = (String)
listOfFiles[i].toURI().toString().replace("C:/", "");
```

```java
                    img.setRelativePath(absolutePath.substring(root.length()));
                    img.setAbsolutePath(absolutePath);
                    collection.add(img);
                }
            }
            return collection;
        }

        /* (non-Javadoc)
         * @see com.interop.webapp.UserImageRepository#getWebPath(java.lang.String)
         */
        @Override
        public String getWebPath(String imagename) {
            return owner + '/' + imagename;
        }

        /* (non-Javadoc)
         * @see com.interop.webapp.UserImageRepository#getPath(java.lang.String)
         */
        @Override
        public String getPath(String imagename) {
            return getUserFolder() + '/' + imagename;
        }

        /* (non-Javadoc)
         * @see com.interop.webapp.UserImageRepository#newUpload(java.lang.String,
org.springframework.web.multipart.MultipartFile)
         */
        @Override
        public Boolean newUpload(String imagename, MultipartFile uploadedfile) {
            ensureOwnerFolder();
            if (imagename.trim().equals("")) {
                imagename = uploadedfile.getOriginalFilename();
            }
            String filename = getPath(imagename);
            File imageFile = new File(URI.create(filename));
            if (imageFile.exists()) {
                return false;
            }
        System.out.println("moving uploaded image file to : " + filename);
        log.info("moving uploaded image file to : " + filename);
    try {
                FileOutputStream fos = new
FileOutputStream(URI.create(filename).getRawPath());
                fos.write(uploadedfile.getBytes());
                fos.close();
            } catch (IllegalStateException e) {
                System.out.println(String.format("ERROR: moving uploaded image file: %s
(%s)", filename, e.getMessage()));
            } catch (IOException e) {
                System.out.println(String.format("ERROR: moving uploaded image file: %s
(%s)", filename, e.getMessage()));
                log.error(String.format("moving uploaded image file: %s (%s)", filename,
e.getMessage()));
            }
            return true;
        }

        /* (non-Javadoc)
         * @see com.interop.webapp.UserImageRepository#removeImage(java.lang.String)
         */
        @Override
        public void removeImage(String imagename) {
            String filename = getPath(imagename);
            File imageFile = new File(URI.create(filename));
            if (!imageFile.exists()) {
                return;
            }

        System.out.println("deleting image file: " + filename);
        try {
            imageFile.delete();
        } catch (SecurityException se) {
```

```java
                    System.out.println(String.format("ERROR: deleting image file: %s (%s)",
filename, se.getMessage()));
            }
        }

        private String getUserFolder() {
                return root + '/' + owner;
        }

        private void ensureOwnerFolder() {
                String userFolder = getUserFolder();
                File ownerFolder = new File(URI.create(userFolder));
                if (ownerFolder.exists()) {
                        return;
                }

            System.out.println("creating directory: " + userFolder);
            try {
                ownerFolder.mkdir();
            }
            catch (SecurityException se) {
                    System.out.println(String.format("ERROR: creating directory: %s (%s)",
userFolder, se.getMessage()));
            }
            catch (Exception e) {
                    System.out.println(String.format("ERROR: creating directory: %s (%s)",
userFolder, e.getMessage()));
            }
        }
}
```

webapp/src/main/java/com.interop.webapp/JsonWrapper.java

```java
package com.interop.webapp;

import java.io.IOException;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.Map;

import org.apache.log4j.Logger;

import com.fasterxml.jackson.core.JsonGenerationException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonWrapper {
        static Logger log = Logger.getLogger(WebApp.class.getName());
        ObjectMapper objectMapper = new ObjectMapper();

        public String toJson(Map<String, Object> mapObject) {
                StringWriter json = new StringWriter();
                try {
                        objectMapper.writeValue(json, mapObject);
                } catch (JsonGenerationException e) {
                        e.printStackTrace();
                } catch (JsonMappingException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return json.toString();
        }

        public Map<String, Object> fromJson(String jsonMessage) {

                StringReader jsonString = new StringReader(jsonMessage);
                Map<String, Object> mapObject = null;
                try {
                        mapObject = objectMapper.readValue(jsonString,
                                        new TypeReference<Map<String, Object>>() {
                                        });
                } catch (JsonGenerationException e) {
                        e.printStackTrace();
                } catch (JsonMappingException e) {
                        // Badly formatted JSON message, ignore
                        log.error(String.format("Badly formatted JSON message, ignoring (%s)",
e.getMessage()));
                        mapObject = null;
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return mapObject;
        }
}
```

webapp/src/main/resources/templates/home.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
        xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Home</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}"
        href="../../css/bootstrap.min.css" />
</head>
<body>
        <div class="container">
                <div th:replace="fragments/header :: header">...</div>
                <h1 th:inline="text">Image files for [[${user}]]</h1>
                <br/>
                <div th:if="${count == 0}" class="alert alert-info">No images, use
upload.</div>

                <div th:each="i : ${imagerefs}">
                        <br/>
                        <br/>
                        <div>
                                <a th:href="${'/image?name=' + i.getImageName()}">
                                        <img th:alt-title="${i.getImageName()}"
th:src="${imagesWebPath + '/' + i.getRelativePath()}" />
                                </a>
                        </div>
                        <div>
                                <h2>
                                        <a th:href="${'/image?name=' + i.getImageName()}"
th:text="${i.getImageName()}">image name does here</a>
                                </h2>
                        </div>
                </div>
                <br/>
                <br/>
                <div th:if="${count == 1}" th:inline="text" class="alert alert-
info">[[${count}]] image</div>
                <div th:if="${count > 1}" th:inline="text" class="alert alert-
info">[[${count}]] images</div>
        </div>
</body>
</html>
```

webapp/src/main/resources/templates/upload.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
        xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Upload</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}"
        href="../../css/bootstrap.min.css" />
</head>
<body onload="document.form.imagename.focus();">
        <div class="container">
                <div th:replace="fragments/header :: header">...</div>
                <div class="content">
                        <h2>Upload Image</h2>
                        <p th:if="${success == true}" class="alert alert-info"
th:inline="text">[[${imagename}]] has been uploaded successfully.</p>
                        <p th:if="${success == false}" class="alert alert-error"
th:inline="text">Upload of [[${imagename}]] has failed, check name.</p>
                        <form name="form" th:action="@{/upload}" action="/upload"
method="POST" enctype="multipart/form-data">
                                <fieldset>
                                        <input type="text" name="imagename" value=""
placeholder="Image Name (no spaces)" />
                                        <input type="file" name="uploadfile"  />
                                </fieldset>
                                <input type="submit" id="upload" value="Upload" class="btn
btn-primary" />
                        </form>
                </div>
        </div>
</body>
</html>
```

webapp/src/main/resources/templates/image.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
          xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Image</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}"
        href="../../css/bootstrap.min.css" />
</head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

<script th:inline="javascript">

var timeOut = 500;
var requestTimeOut = 30 * 1000; // secs multiplied to get milleseconds
var timeStart;

$(document).ready(function() {
        $("#effect").change(function(){
                //Need to encode the perod in filename otherwise we get a HTTP 406 error.
                var effectName = $(this).val();
        var imageName = $("#imagename").val();
                var requestUri = '/effectrequest/' + effectName + '/' + imageName;
            $.ajax({
                url: /*[+   [[${hostname}]] + requestUri  +]*/
            }).then(function(data) {
                    switch(data.status) {
                       case 'submitted':
                                        timeStart = new Date();
                                        $("#requestidentider").val(data.requestId);
                        $("#requestcreated").val(data.created);
                                        var timeNow = new Date();
                                        var milliseconds = timeNow.getTime() -
timeStart.getTime();

                                        var msg = 'Request submitted, ' + milliseconds + '
milliseconds elapsed.';

                                        $("#msgbox").text(msg);
                                        $( "#save" ).prop( "disabled", true );
                                        $( "#effect" ).prop( "disabled", true );
                                        $("<div />").css({
                                         position: "absolute",
                                         width: "100%",
                                         height: "100%",
                                         left: 0,
                                         top: 0,
                                         zIndex: 1000000,  // to be on the safe side
                                         background: "url(http://j1edit.com/images/icon-
loading-animated.gif) no-repeat 50% 50%"

}).attr("id","rendering").appendTo($("#imagecontainer").css("position", "relative"));
                                        setTimeout("checkRequest();", timeOut);
                                        break;
                        case 'failed':
                                var msg = 'Request failed, please try again.';
                                $("#msgbox").text(msg);
                            break;
                      }
               });
        });
});

function resetWaitingUi() {
    $( "#save" ).prop( "disabled", false );
    $( "#effect" ).prop( "disabled", false );
    $( "#rendering" ).remove();
}

function checkRequest(){
        var requestId = $("#requestidentider").val();
    var requestCreated = $("#requestcreated").val();
        var requestUri = '/effectfetch/' + requestId + '/' + requestCreated;
```

```
    $.ajax({
        url: /*[+    [[${hostname}]] + requestUri  +]*/
    }).then(function(data) {
        switch(data.status) {
                case 'completed':
                        var timeNow = new Date();
                        var milliseconds = timeNow.getTime() - timeStart.getTime();
                        var msg = 'Processing <strong>complete</strong>, ' + milliseconds + '
milliseconds elapsed.';
                        $("#msgbox").html(msg);
                    $("#imagetag").attr("src", data.url);
                $("#imagenew").val(data.url);
                resetWaitingUi();
                    break;
                case 'notready':
                        var timeNow = new Date();
                        var milliseconds = timeNow.getTime() - timeStart.getTime();
                        var msg = "";
                        if (milliseconds > requestTimeOut) {
                            msg = 'Failed to get a reply for request after ' + milliseconds + '
milliseconds.';
                            resetWaitingUi();
                        } else {
                            msg = 'Processing continuing, ' + milliseconds + ' milliseconds
elapsed.';
                        setTimeout("checkRequest();", timeOut);
                        }
                        $("#msgbox").text(msg);
                    break;
            case 'failed':
                var msg = 'Server side processing failed, please try again.';
                $("#msgbox").text(msg);
                resetWaitingUi();
                break;
                default:
                    alert('Unknown error : ' + data.status);
                    resetWaitingUi();
            }
    });
}


</script>
<body onload="document.form.imagename.focus();">
        <div class="container">
                <div th:replace="fragments/header :: header">...</div>
                <div class="content">
                        <h2>Image Effects</h2>
                        <p id="msgbox" class="alert alert-info">Select an effect to apply to
the image</p>
                        <br/>
                        <br/>
                        <div class="pull-right">
                                <div id="imagecontainer">
                                        <img id="imagetag" th:alt-title="${imagename}"
th:src="${imagesWebPath + '/' + imageref}" />
                                </div>
                                <div>
                                        <h2 th:text="${imagename}">image name does here</h2>

                                </div>
                        </div>
                        <div class="pull-left">
                                <form name="form" th:action="@{/image}" action="/image"
method="POST">
                                        <fieldset>
                                                <input type="hidden" id="imagename"
name="imagename" th:value="${imagename}" value="" />
                                                <input type="hidden" id="imagenew"
name="imagenew" value="" />
                                                <input type="hidden" id="requestidentider"
name="requestid" value="" />
                                        <input type="hidden" id="requestcreated" name="requestcreated" value=""
/>
```

```
                                                            Effect :
                                                            <select id="effect" name="effect">
                                                                      <option selected="selected"
disabled="disabled" >Select</option>
                                                                      <option th:each="e : ${effects}"
th:text="${e}"></option>
                                                            </select>
                                                  </fieldset>
                                                  <input type="submit" id="save" value="Save" class="btn
btn-primary" />
                                        </form>
                              </div>
                    </div>
          </div>
</body>
</html>
```

webapp/src/main/resources/templates/testharness.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Test Harness</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" href="../../css/bootstrap.min.css"
/>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

<style>
input[type="text"]
{
    width:50px;
}

section {
    width: 100%;
    height: 300px;
    margin: auto;
    padding: 0px;
}
div#one {
    width: 40%;
    height: 300px;
    float: left;
}
div#two {
    margin-left: 60%;
    height: 300px;
}
</style>

<script th:inline="javascript">
//<![CDATA[

var processQueueCount = 0;
var requestTimeOut = /*[+   [[${processingtimeout}]]  +]*/; // milleseconds
var timeStart = 0;

// processingtimeout

var status = '';
var timeOut = 0;
var maxLogLines = 0;
var maxOpenRequests = 0;
var batchSize = 0;

var pointer = 0;

var reqqueue = [];
var testImages  = ['FranceArcDeTriompheRoyaltyFree.jpg', 'holiday.png', 'Pierce.jpg'];
```

```
var testEffects = ['Grayscale', 'Blur', 'Invert'];

function getRandomSelection(sourceArray) {
    var index = Math.floor((Math.random() * sourceArray.length));
    return sourceArray[index];
}

function logMessage(msg){
    if ($("#logentries").children().length > maxLogLines) {
        $("#logentries div:last-child").remove();
    }
    $("#logentries").prepend("<div>" + msg + "</div>");
}

function logReset(){
    $("#logentries").empty();
    $("#logentries").prepend("<div>Log messages</div>");
}


function removeRequest(requestid){
    for (var i = 0, len = reqqueue.length; i < len; i++) {
        if (reqqueue[i].requestid == requestid) {
            // Remove from queue
            reqqueue.splice(i,1);
            return true;
        }
    }
    //logMessage('Did not find request ID in queue: ' + requestid);
    return false;
}

function setRequestResetCreated(requestid){
    for (var i = 0, len = reqqueue.length; i < len; i++) {
        if (reqqueue[i].requestid == requestid) {
            reqqueue[i].fetchactive = 0;
        }
    }
}


function getRequest(requestid){
    for (var i = 0, len = reqqueue.length; i < len; i++) {
        if (reqqueue[i].requestid == requestid) {
            return reqqueue[i];
        }
    }
    return false;
}

function clearRequests(){
    while (reqqueue.length > 0) {
        reqqueue.pop();
    }
}



function processQueue(){
    if (status == 'abort') {
        return;
    }
    var thisBatchCount = 0;
    var requestUri = '';
    if (status == 'processqueue' && reqqueue.length < maxOpenRequests) {
        // Submit some requests
        thisBatchCount = batchSize;
        while (thisBatchCount > 0) {
            var effectName = getRandomSelection(testEffects);
            var imageName = getRandomSelection(testImages);
            requestUri = '/effectrequest/' + effectName + '/' + imageName;
            $.ajax({
                url: /*[+    [[${hostname}]] + requestUri  +]*/
```

```
            }).then(function(data) {
                switch(data.status) {
                    case 'submitted':
                        // Add to the end of the queue
                        reqqueue.push({requestid:data.requestId, created:data.created,
fetchactive:0});
                        logMessage('Logged request ' + data.requestId);
                        break;
                    case 'failed':
                        logMessage('Failed to get respone for Request');
                        break;
                }
            });
            thisBatchCount--;
        }
    }
    thisBatchCount = batchSize;
    if (processQueueCount > 0) {  // First time round don't make any effectfetch requests
        for (var i = 0, len = reqqueue.length; i < len; i++) {
            if (reqqueue[i].fetchactive == 0) {
                requestUri = '/effectfetch/' + reqqueue[i].requestid + '/' +
reqqueue[i].created;
                reqqueue[i].fetchactive = (new Date()).getTime();
                $.ajax({
                    url: /*[+   [[${hostname}]] + requestUri  +]*/
                }).then(function(data) {
                    switch(data.status) {
                        case 'completed':
                            // Reference the image so that the browser attempts to
download the image content - load on server
                            $("#imagetag").attr("src", (data.url + '?' + Math.random()));
                        var request = getRequest(data.requestId);
                        if (request == false) {
                            logMessage('Request ID ' + data.requestId + ' not longer exists
in the queue, ignoring.');
                        } else {
                            var msg = 'Processing <strong>completed</strong> for ' +
data.requestId + ', ' + data.elapsedTime + ' milliseconds elapsed.';
                                removeRequest(data.requestId);
                                logMessage(msg);
                        }
                            break;
                        case 'notready':
                            var request = getRequest(data.requestId);
                            if (request == false) {
                                logMessage('Request ID ' + data.requestId + ' not longer
exists in the queue, ignoring.');
                            } else {
                                var msg = "";
                                if (data.elapsedTime > requestTimeOut) {
                                    msg = 'Failed to get a reply for request ' +
data.requestId + ' after ' + data.elapsedTime + ' milliseconds.';
                                    removeRequest(data.requestId);
                                } else {
                                    msg = 'Processing continuing for request ' +
data.requestId + ', ' + data.elapsedTime + ' milliseconds elapsed.';
                                        if (!setRequestResetCreated(data.requestId)) {
                                            removeRequest(data.requestId);
                                        }
                                }
                                logMessage(msg);
                            }
                            break;
                        case 'failed':
                            var msg = 'Server side processing failed for ' +
data.requestId;
                            removeRequest(data.requestId);
                            logMessage(msg);
                            break;
                        default:
                            alert('Unknown error : ' + data.status);
                    }
                });
            }
```

```
                thisBatchCount--;
            }
        }
        processQueueCount++;
        if (status == 'shutdown' && reqqueue.length == 0) {
            $( "#abort" ).trigger( "click" );
        }
        $("#noofopenreqs").text(reqqueue.length);
        setTimeout("processQueue();", timeOut);
}




$(document).ready(function() {

    $("#start").click(function(){
        maxOpenRequests = parseInt($( "#maxopenreq" ).val());
        maxLogLines     = parseInt($( "#maxloglines" ).val());
        batchSize       = parseInt($( "#batchsize" ).val());
        timeOut         = parseInt($( "#timeout" ).val());
        var msg = [];
        if (batchSize > maxOpenRequests) {
        msg.push('Batch size cannot be greater then max open requests');
        }
        if (msg.length > 0) {
            $( "#msgbox" ).text(msg.join(';'));
        } else {
                $( "#msgbox" ).text('Running ...');
                $( "#start" ).prop( "disabled", true );
                $( "#maxopenreq" ).prop( "disabled", true );
                $( "#timeout" ).prop( "disabled", true );
                $( "#maxloglines" ).prop( "disabled", true );
                $( "#batchsize" ).prop( "disabled", true );
                processQueueCount = 0;
                $( "#shutdown" ).prop( "disabled", false );
                $( "#abort" ).prop( "disabled", false );
                status = 'processqueue';
                setTimeout("processQueue();", timeOut);
        }
        logReset();
        return false;
    });

    $("#shutdown").click(function(){
        status = 'shutdown';
        $( "#shutdown" ).prop( "disabled", true );
        $( "#msgbox" ).text('Stopping ...');
        return false;
    });


    $("#abort").click(function(){
        status = 'abort';
        clearRequests();
        $("#noofopenreqs").text('0');
        $( "#start" ).prop( "disabled", false );
        $( "#maxopenreq" ).prop( "disabled", false );
        $( "#timeout" ).prop( "disabled", false );
        $( "#maxloglines" ).prop( "disabled", false );
        $( "#batchsize" ).prop( "disabled", false );
        $( "#shutdown" ).prop( "disabled", true );
        $( "#abort" ).prop( "disabled", true );
        $( "#msgbox" ).text('Enter test parameters');
        return false;
    });

});


//]]>

</script>
```

```html
    </head>

    <body>
        <div class="container">
            <div th:replace="fragments/header :: header">...</div>
            <h1 th:inline="text">Test Harness for [[${user}]]</h1>
            <br/>
            <p id="msgbox" class="alert alert-info">Enter test parameters</p>
                <section>
                    <div id="one">
                    <form name="form" th:action="@{/testharness}" action="/testharness"
method="POST">
                        <fieldset>
                            <p><input type="text" id="maxopenreq" value="100" /> Target
throttle for requests</p>
                            <p><input type="text" id="batchsize" value="25" /> Batch size</p>
                            <p><input type="text" id="timeout" value="1000" /> Time between batches
(milliseconds)</p>
                            <p><input type="text" id="maxloglines" value="50" /> Maximum log
lines (below)</p>
                        </fieldset>
                        <input type="submit" id="start" value="Start" class="btn btn-primary"
/>
                        <input type="submit" id="shutdown" value="Shutdown" class="btn btn-
primary" disabled="disabled" />
                        <input type="submit" id="abort" value="Abort" class="btn btn-primary"
disabled="disabled" />
                    </form>
                    </div>
                    <div id="two">
                    <div id="imagecontainer">
                        <img id="imagetag" alt="Rendered Image" src="../../images/interop.png"
/>
                    </div>
                    </div>
                </section>
            <div>Number of open requests: <span id="noofopenreqs">0</span> </div>
            <br />
            <div id="logentries">
                <div>Log entries appear here</div>
            </div>
            <br />
            <br />
            <br />
            <br />
        </div>
    </body>
</html>
```

webapp/src/main/resources/templates/fragments/header.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
        xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title th:text="${title}">Title</title>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}"
        href="../../css/bootstrap.min.css" />
</head>
<body>
        <div class="navbar" th:fragment="header">
                <div class="navbar-inner">
                        <a class="brand" href="https://github.com/johnwarde/mscdissertation">
Interop </a>
                        <ul class="nav">
                                <li><a th:href="@{/}" href="home.html"> Home </a></li>
                                <li th:if="${#authentication}?
${#authorization.expression('isAuthenticated()')}">
                                        <a th:href="@{/upload}" href="upload"> Upload </a>
                                </li>
                <li th:if="${#authentication}?
${#authorization.expression('isAuthenticated()')}">
                        <a th:href="@{/testharness}" href="testharness"> Test Harness </a>
                </li>
                                <li th:if="${#authentication}?
${#authorization.expression('isAuthenticated()')}">
                                        <a th:href="@{/logout}" href="logout"> Logout </a>
                                </li>
                        </ul>
                </div>
        </div>
</body>
</html>
```

webapp/src/main/resources/application.properties

```
debug: true
spring.thymeleaf.cache: false
security.basic.enabled: false
logging.level.org.springframework.security: INFO
# Added by johnwarde
server.port = 8080
webapp.queuehostname = localhost
webapp.queuename = processor_rpc_queue
webapp.hostname = localhost
webapp.imageFilesRoot = file:/Users/johnwarde/Downloads/webappresources
webapp.processingtimeout = 60000
logging.file = /Users/johnwarde/Downloads/webappresources/webapp.log
```

92

webapp/pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <!-- PARENT -->
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>1.2.2.BUILD-SNAPSHOT</version>
        </parent>

        <!-- PROJECT INFO -->
        <groupId>com.interop.webapp</groupId>
        <artifactId>webapp</artifactId>
        <name>Interop Web App</name>
        <description>Interop Web Application based from Spring Boot Web Secure JDBC
Sample</description>
        <url>https://github.com/johnwarde/mscdissertation</url>
        <organization>
                <name>John Warde</name>
                <url>https://www.linkedin.com/in/johnwarde</url>
        </organization>

    <!-- DEPENDENCIES -->

    <dependencies>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-security</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.thymeleaf.extras</groupId>
                        <artifactId>thymeleaf-extras-springsecurity3</artifactId>
                        <!-- <version>${thymeleaf-extras-springsecurity3.version}</version> --
>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-thymeleaf</artifactId>
                </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-jdbc</artifactId>
                </dependency>

                <dependency>
                        <groupId>com.h2database</groupId>
                        <artifactId>h2</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.apache.httpcomponents</groupId>
                        <artifactId>httpclient</artifactId>
                        <scope>test</scope>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
```

```xml
                    <scope>test</scope>
                </dependency>

                <dependency>
                    <groupId>ch.qos.logback</groupId>
                    <artifactId>logback-classic</artifactId>
                    <version>1.0.13</version>
                </dependency>

                <dependency>
                    <groupId>org.apache.jclouds</groupId>
                    <artifactId>jclouds-all</artifactId>
                    <version>1.8.1</version>
                </dependency>

        </dependencies>

        <!-- REPOSITORIES -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </pluginRepository>
    </pluginRepositories>

        <!-- BUILD PLUGINS -->
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

        <issueManagement>
                <system>GitHub Issues</system>
                <url>https://github.com/johnwarde/mscdissertation/issues</url>
        </issueManagement>
</project>
```

Processor Component

processor/src/main/java/com.interop.webapp/ProcessorApp.java

```java
package com.interop.processor;

import java.util.Map;

import javax.annotation.PostConstruct;

import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.*;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.web.bind.annotation.*;

import com.rabbitmq.client.*;
import com.rabbitmq.client.AMQP.BasicProperties;


@RestController
@EnableAutoConfiguration
public class ProcessorApp {
        @Autowired
        private ProcessorConfig config;

        static Logger log = Logger.getLogger(ProcessorApp.class.getName());

    @RequestMapping("/")
    String home() {
        return "Hello World!";
    }


        @PostConstruct
    public void setUpProcessor() throws Exception {

                JsonWrapper objJson = new JsonWrapper();
                EffectsApplicator effects = new EffectsApplicator();

        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(config.getQueueHostName());
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare(config.getQueueName(), false, false, false, null);
        channel.basicQos(1);
        QueueingConsumer consumer = new QueueingConsumer(channel);
        channel.basicConsume(config.getQueueName(), false, consumer);

                log.info(String.format("processorstart\tsuccess\t%s\t%s",
config.getQueueHostName(), config.getQueueName()));
        while (true) {
                String status = "";
                String response = "";
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            BasicProperties props = delivery.getProperties();
            BasicProperties replyProps = new BasicProperties
                                .Builder()
                                .correlationId(props.getCorrelationId())
                                .build();

            String message = new String(delivery.getBody());
            Map<String, Object> map = objJson.fromJson(message);
            if (map != null) {
                                log.info(String.format("processrequest\tsuccess\t%s\t%s\t%s",
                                        config.getQueueHostName(), map.get("user"),
props.getCorrelationId()));

                                status = effects.apply((String) map.get("effectName"),
```

```java
                                        (String) map.get("inputPath"),
                                        (String) map.get("ouputPath"));

                            map.put("status", status);
                            map.put("requestCompleted", "");
                            response = objJson.toJson(map);

                    log.info(String.format("processfinish\tsuccess\t%s\t%s\t%s\t%s",
                                    config.getQueueHostName(), map.get("user"),
props.getCorrelationId(),
                                    map.get("ouputPath")));
                } else {
                    response = "";
                }
                channel.basicPublish("", props.getReplyTo(), replyProps, response.getBytes());
                channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
                    log.info(String.format("processcompleted\t%s\t%s\t%s\t%s", status,
                                    config.getQueueHostName(), map.get("user"),
props.getCorrelationId()));
            }

        }

        public static void main(String[] args) throws Exception {
                    new SpringApplicationBuilder(ProcessorApp.class,
                                    "classpath:/META-INF/application-context.xml").run(args);
        }

}
```

processor/src/main/java/com.interop.webapp/EffectsApplicator.java

```java
package com.interop.processor;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.awt.image.BufferedImageOp;
import java.awt.image.ConvolveOp;
import java.awt.image.Kernel;
import java.io.File;
import java.net.URI;

import javax.imageio.ImageIO;

import org.apache.log4j.Logger;

public class EffectsApplicator {
    private enum Effect {Grayscale, Invert, Blur}
    static Logger log = Logger.getLogger(ProcessorApp.class.getName());

    public String apply(String effectName, String inputPath,
                    String ouputPath) {
        BufferedImage image = null;
    Boolean success = true;
        try {
            File input = new File(URI.create(inputPath));
            image = ImageIO.read(input);
            }
        catch (Exception e) {
            log.error(String.format("Failed to read input image file (%s) - %s",
                        inputPath, e.getMessage()));
            success = false;
        }
        if (!success) {
                return "failed";
        }

        try {

                    switch (Effect.valueOf(effectName)) {
                    case Grayscale:
                            success = grayscale(image);
                            break;
                    case Invert:
                            success = invert(image);
                            break;
                    case Blur:
                            image = blur(image);
                            break;
                    default:
                        return "failed";
                    }
            } catch (Exception e) {
            log.error(String.format("Invalid effect name (%s), ignoring", effectName));
            success = false;
            }

        try {
            File ouptut = new File(URI.create(ouputPath));
            String ext = ouputPath.substring(ouputPath.lastIndexOf('.') + 1)
                            .toLowerCase();
            ImageIO.write(image, ext, ouptut);
                    }
        catch (Exception e) {
            log.error(String.format("Failed to write to image file (%s) - %s",
                        ouputPath, e.getMessage()));
            success = false;
        }
        if (success) {
                    return "completed";
        }
```

```java
            return "failed";
    }

    // Algorithm sourced from:
    //    http://www.tutorialspoint.com/java_dip/grayscale_conversion.htm
    private Boolean grayscale(BufferedImage image) {
            int width;
            int height;
            width = image.getWidth();
            height = image.getHeight();
            for(int i=0; i<height; i++){
                    for(int j=0; j<width; j++){
                        Color c = new Color(image.getRGB(j, i));
                        int red = (int)(c.getRed() * 0.299);
                        int green = (int)(c.getGreen() * 0.587);
                        int blue = (int)(c.getBlue() *0.114);
                        int rgbAdded = red+green+blue;
                        Color newColor = new Color(rgbAdded, rgbAdded,rgbAdded);
                        image.setRGB(j,i,newColor.getRGB());
                    }
            }
            return true;
    }

    // Algorithm sourced from:
    //    http://stackoverflow.com/questions/8662349/convert-negative-image-to-positive
    private Boolean invert(BufferedImage image) {
    for (int x = 0; x < image.getWidth(); x++) {
        for (int y = 0; y < image.getHeight(); y++) {
            int rgba = image.getRGB(x, y);
            Color col = new Color(rgba, true);
            col = new Color(255 - col.getRed(),
                            255 - col.getGreen(),
                            255 - col.getBlue());
            image.setRGB(x, y, col.getRGB());
        }
    }
            return true;
    }

    // Algorithm sourced from:
    //    http://www.javaworld.com/article/2076764/java-se/image-processing-with-java-
2d.html
    private BufferedImage blur(BufferedImage image) {
            float ninth = 1.0f / 9.0f;
            float[] blurKernel = {
                ninth, ninth, ninth,
                ninth, ninth, ninth,
                ninth, ninth, ninth
            };
            BufferedImageOp blur = new ConvolveOp(new Kernel(3, 3, blurKernel));
            BufferedImage alteredImage = blur.filter(image, null);
            return alteredImage;
    }

}
```

processor/src/main/resources/application.properties

```
# Added by johnwarde
server.port = 8081
processor.queuehostname = localhost
processor.queuename = processor_rpc_queue
logging.file: /Users/johnwarde/Downloads/webappresources/processor.log
```

processor/pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <!-- PARENT -->
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>1.2.2.BUILD-SNAPSHOT</version>
        </parent>

        <!-- PROJECT INFO -->
        <groupId>com.interop.processor</groupId>
        <artifactId>processor</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>Interop Processor App</name>
        <description>Interop Processor Application will process files from the WebApp, based on
the Spring Boot Starter AMPQ project</description>
        <url>https://github.com/johnwarde/mscdissertation</url>
        <organization>
                <name>John Warde</name>
                <url>https://www.linkedin.com/in/johnwarde</url>
        </organization>

    <!-- DEPENDENCIES -->

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>

    </dependencies>

        <!-- REPOSITORIES -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </pluginRepository>
    </pluginRepositories>

        <!-- BUILD PLUGINS -->
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
```

```xml
                            <artifactId>spring-boot-maven-plugin</artifactId>
                    </plugin>
            </plugins>
        </build>

        <issueManagement>
                <system>GitHub Issues</system>
                <url>https://github.com/johnwarde/mscdissertation/issues</url>
        </issueManagement>
</project>
```