UNIVERSITY OF
LINCOLN

Design, implementation, and evaluation of a virtual-machine based, high-availability (HA) MySQL Master-Master replication service

Oliver Martin Grooby

The purpose of this quantitative research project is to explore the scalability of Virtual Machine based MySQL instances in a Master-Master configuration and produce a disk image that can be used to redeploy a stack that combines the database and a web API functionality

School of Computer Science

University of Lincoln

2020

1. Acknowledgements

Dedicated to the memory of Dolly Dixon, who left us shortly before writing; a life well-lived. Also, to my family, without your nurture, I would not have been as ambitious as I am. To all my friends, thank you for being there. A big thank you to my fellow crew at 979 Ropewalk your constant support throughout the project helped me push through. And finally, to my Project Supervisor, Dr Derek Foster – your expertise and endless knowledge and resources in Cloud Computing have enabled the progress seen in this project.

2. Abstract

High availability (HA) storage services are now commonplace as part of cloud providers' PaaS offerings. Examples are Amazon's Aurora and Azure MSSQL/MySQL PaaS services where data storage is fully managed in terms of redundancy, scalability and load balancing. Though the benefits of a PaaS-based HA storage service are clear, there are scenarios where the need for a more portable and user-configurable solution is required where interoperability is key. A comprehensive review of cloud patterns/models including PaaS and IaaS, as well as HA services, will be undertaken including a review of internal and external cloud load balancing techniques. By building a bespoke and highly interoperable HA storage service using an IaaS approach, the service will have the capability to be deployed across a minimum of two Linux virtual machines with master-slave, master-master, and eventually load balancing configured if deployed to the cloud. The virtual machines will host the entire deployment stack, emulating the feature set of a PaaS HA storage service. The result is bespoke virtual machine images running the HA service which can be deployed on any cloud provider's platform. A basic REST service will be required to test the replication service's performance capabilities. This project is multi-faceted with some components that can be negotiated from the perspective of local and/or cloud development. Development in the first instance will take place on locally managed virtual machines using hosted virtualization, with a view to some cloud-based deployment for testing.

3. Introduction

Many enterprises and organisations are increasingly moving to cloud providers for their data and backend infrastructure needs. An IT asset of an organisation commonly migrated to a cloud service are databases; many organisational databases are mission-critical and rely on uptime, redundancy and performance – key factors that will be explored in this project. Cloud Providers often provide their Database platform as PaaS (Platform as a Service), and in many use cases, such platforms will suffice. However, such an approach lacks configuration options that could be found by creating custom infrastructure and platform by utilising Cloud IaaS (Infrastructure as a Service) to create bespoke Virtual Networks and Virtual Machines, running a customised application stack. The key aim of this project is to create such a bespoke application stack, in which 2 virtual machines are configured in such a way that a hosted MySQL database is replicated between 2 servers in a Master-Master configuration, as opposed to the more common Master-Slave configuration; this allows a more integrated use of Load Balancers which could improve Database throughput. This Master-Master configuration will have its performance reported and compared against the more common Master-Slave configuration as well as PaaS offerings.

4. Background & Literature Review

Cloud computing is a service that is being rapidly adopted by today's businesses, a study in 2013 discovered that in Small to Medium Enterprises (SMEs) there was a growth rate of over 20% per year of such enterprises utilising cloud computing (Arnold, et al., 2013). This paper provides an interesting insight into not only how SMEs are utilising Cloud technologies, but also large Logistical firms are also using Public Cloud to enhance their services. The paper also goes into detail on how an EU funded project, "Future Business Clouds" or FBC list about 60 different cloud computing Research and Development projects as part of the European Union's own Digital Future policies – this highlights the sheer importance of Cloud Technologies at not only an Enterprise level but also on the global stage.

Cloud Computing is generating this level interest and market penetration through many advantages that make a high density, high speed and high availability feasible for an increasing amount of Enterprises and Businesses that never would have had exposure to such facilities before due to constraints such as financial (Avram, 2014). Avram addresses some of the many advantages of cloud computing, such like the ability to immediately have access to hardware, without the high initial outlay that an Enterprise would be expected to pay, rather, the cloud often bills the user for access to hardware on an hourly basis, so the enterprise will only ever pay for the amount of time they are actively accessing the hardware. This is a fundamental aspect of the cloud computing business model and attracts such customers as it actively enables business start-ups to operate with smaller investments, and thus began an increase in the size of the online services sector. Avram also explores disadvantages to cloud computing, including reliability, a key topic of this study.

Cloud vendors offer a document known as an SLA (Service Level Agreement) which is a commitment from the vendor for quality of service, availability and responsibilities (Wieder, Butler, Theilmann and Yahyapour, 2011). SLAs can in some cases become legally binding contracts between the customer and the vendor which legally assure the stated uptime of the service, as well as quality assurance of support and the deployed service. Should the vendor drop any particular aspect of their SLA, for example, service uptime often drops below the threshold stated by the SLA, the customer can abandon the contract without penalty, if applicable, or in some cases be entitled to a monetary refund, such as in the case of Google Cloud Platform, where customers will receive credit if they are unable to provide the service they offer (Compute Engine Service Level Agreement (SLA), 2020).

There are many models of various cloud computing services, and vendors typically offer services with these various models in mind. Models are often named (X)aaS (as-a-Service), in this particular study, IaaS and PaaS – Infrastructure as a Service and Platform as a Service respectively – are investigated, other such models such as SaaS – Software as a Services – are offered, and these refer to such services like payroll management and office applications that are hosted online, easily accessible via a browser. This differs from IaaS and PaaS as these models are typically used to create infrastructure or are used to host Software. The PaaS model defines the delivery of a platform on which applications can be hosted from, without having to manage infrastructure (Carroll, van der Merwe and Kotze, 2011), common PaaS services include web services and database services as well as cloud storage for Content Delivery Networks (CDNs). IaaS offers a customer with a much finer control of their infrastructure, at the cost of increased complexity, this is because the customer will have to create their Virtual Machines (VMs) and Virtual Private Cloud (VPC) Networks, as well as having the responsibility of ensuring that the configuration is secured, and also being solely responsible for OS and Application updates. However, like PaaS hardware maintenance and upgrading is completed by the vendor without loss of uptime or interaction from the customer.

Cloud computing allows businesses to be being more agile, that is to be able to quickly adapt to demand, whether it is falling, or growing. Cloud computing offers a cost-effective means of being able to quickly scale computing resources as per the demand and workload experienced. There are 2 forms of scaling in cloud computing settings, “scale-out” and “scale-up” there is also the converse method of scaling “scale-down” which allows for cost savings when increased resources would be effectively wasting money. Scaling out is the act of using an increased number of machines, physical or virtual to manage demand. Scaling out is what most cloud vendors will perform if a customer enables autoscaling on their resources, as most platforms will use a persistent disk image and machine template to quickly start up additional virtual machines. Scaling up accomplishes scaling by increasing the resources available to a machine, again physical or virtual, useful if an application requires configuration on each node that is started. However, this comes with the drawback of being unable to be accomplished while the machine is powered on.

Elastic Scaling – commonly referred to or marketed as Autoscaling – is a means of cloud vendors offering a dynamic solution to evolving demand. Google Cloud Platform offers Autoscaling using Managed Instance Groups. If enabled the customer has the option to define a CPU threshold based on the total group CPU usage before starting up additional VMs created through Instance Templates and Images. Combining this with the use of an external load balancer additional VMs become rapidly available to deal with demand spikes. Inversely, Autoscaling will scale down resources if either the initial demand spike has passed, or, the deployed resources are no longer necessary. As stated previously cloud vendors charge for their resources per hour, as such, as demand increases, the billing costs increase as the service scales out, but as demand decreases, billing costs fall as additional resources are terminated and allocated elsewhere. (Compute Engine Documentation, 2020).

Elasticity can be seen as a natural progression of traditional cloud scaling, whereas previously scaling would be accomplished as part of strategic planning - part of a long term vision or goal – it can be done on the fly to match demand enhancing business agility and allowing for classes of services that might not have been possible in the past (Bilyk, n.d.). Previous studies indicated that in some cases, Elastic Scaling can be accounted for a 9.8% to 40% reduction in IT costs (Mao and Humphrey, 2011).

Scaling is key for market today due to constantly changing and evolving demand, and in ways, this flexibility and agility that cloud computing offers cannot be matched by traditional architectures such as onsite Data Centres. Such architectures would severely impact a business’ ability to grow and compete with competitors that utilise cloud architecture (Liu, Chan, Yang and Niu, 2018). Cloud computing is a fast-moving discipline, and, as highlighted by this paper, there has been a significant gap in research into the impact that cloud computing platforms have on business agility. It concludes that cloud computing provides a solution to the stereotypical notion that IT can impede operational agility if implemented correctly while also warning that increased spending on computing resources may have diminishing returns.

Another recent addition to IT architecture is the use of Hybrid Cloud, this is the processes of combining so-called Private Cloud (on-site Data Centres), with Public Cloud provided by vendors such as Google Cloud Platform (Goyal, 2014). Hybrid cloud is particularly useful in use cases where organisations have sensitive and confidential data and applications that need to be hosted on-site but have applications that would benefit from the scalability of a public cloud solution (ibid.). Once again, using load balancers, hybrid cloud architectures can leverage elastic scaling technologies on offer from various vendors. This allows for a private cloud to begin spilling over to public cloud hosting whenever required to meet demand, which in some cases may be a solution that makes better financial sense for some businesses. A typical hybrid cloud solution could be implemented by

hosting a web service on the public cloud that connects to databases stored in the private cloud, with both connected by a VPN. Some vendors offer the ability to maintain hardware hosted VPNs at an additional cost for customers that require such a facility.

This leads to a vital discussion of public cloud security. Public cloud spending by businesses is forecasted to reach approximately USD 249.8bn in 2020, with that figure expected to rise to around USD 331.2bn by 2022 (Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019, 2019). With this rapidly increasing expenditure, so too has the interest of people and organisations with malicious intent, and, as businesses continue to move increasing volumes of data and traffic to a publicly targetable platform (Almorsy, Grundy and Müller, 2016). Many cloud vendor SLAs omit any security guarantee, such breaches could fall under an exclusion within the SLA due to them being outside of the vendor's reasonable control. This report finds that elastic scaling engines can have significant security implications, as one tenant scales down after a spike in demand, another tenant may begin scaling into resources recently released by the previous tenant. This can cause confidentiality issues as there may be traces of confidential data still resident in the physical machine's RAM or persistent storage. While in most cases, the Hypervisor will clean-up resources before making them available to others – thus mitigating this vulnerability – one cannot discount the possibility that this process may fail, or, a more sophisticated attack could bypass this level of protection. However, this report also offers some useful insight into securing assets that are hosted on the private cloud. Vendors offer means of securing console access from unauthorised users, while also allowing roles to be created within an organisation which moderates access to resources. This system is known as IAM – Identity & Access Management – and it ensures only authenticated and authorised users can reconfigure services. Another aspect highlighted by the paper is the importance of ensuring application security, the use of internet protocols such as HTTP to transmit data must be considered, and thus ways of securing this transmission through the use of an encrypted protocol such as HTTPS must be used if sensitive data is being transferred from client to platforms hosted on the public cloud. VPC networks must also be secured using rules to close ports where necessary, by default many vendors will close all ports, and you must specify the ports you intend to use via creating rules. By default, may cloud platforms will block connections between VMs also, so again, firewall rules must be created to ensure proper, but secure, connectivity.

Databases are some of the most critical assets in a business or organisation. With many sectors being able to utilise databases to store vital data (Pivotal Role of Databases in Businesses, 2014). This article highlights the various advantages of a well-designed and easily accessible database will have on an organisation or enterprise. E-commerce settings will enjoy more refined and efficient inventory and order tracking capabilities, while healthcare organisations will be able to store a more concise patient history and be able to rapidly recall this history should a need ever arise, which can help aid in speeding up the treatment time. While having a well-designed database structure might seem trivial to start-ups, it is clear they can be the difference between a successful or failing venture.

An interesting use case for high performance, high availability and globally accessible databases came about in 2014, due to the Ebola outbreak in West Africa in 2014. The outbreak is described as highlighting the need for a centrally available database to help orchestrate an effective response to the virus (Durski et al., 2017). This database collected over 250,000 captured specimen tests from 47 laboratories located in Guinea, Liberia and Sierra Leone in near real-time, this allowed a multitude of tests to be performed thus helping contribute to the global relief effort. Before the implementation of this database, laboratory staff were using Excel spreadsheets to collect demographics from investigation forms. While developed in collaboration with the World Health Organisation – the spreadsheet was developed in Excel 2010 due to the lower computer literacy requirements

compared to a more bespoke system. Results were then emailed at least once daily to all facilities. The database was developed alongside a web platform to allow for rapid access and entry or retrieval of data and had appropriate security measures to ensure that only cleared users were allowed access. The system was designed so that it could be later used to help combat and document any infectious disease in the countries. This is a fantastic advocate for the use of well-developed High Availability databases. Without a fast and reliable way of storing and retrieving vital data the pandemic could have easily have lasted longer, as laboratories were initially using email to share data daily, with the database fully implemented, health authorities and laboratory workers could now share data in real-time which contributed to the end of the pandemic. The report also highlights the need for a data management system using such a database to be able to be rapidly adaptable. The public cloud enables such rapid adaptions due to the ability to quickly scale an application to meet demand, with little planning required at conception.

The above literature advocates an informed cloud strategy for businesses. Through careful planning and correctly determining each particular use case there are many businesses, through SMEs to large enterprises that can realise a significant benefit from utilising cloud technologies, whether through reduced operating costs, increased availability enabled through the using of scaling facilities offered by cloud vendors. This study will further investigate the use of cloud computing for hosting databases, specifically utilising custom IaaS infrastructure.

5. Methodology

5.1 Project Management

As a cloud project, the build process is somewhat different from what would be expected from a typical software development project. While scripting is employed to write out virtual networks and template out and deploy virtual machines, a significant amount of the work required can be completed via Web-Based UIs to accomplish a task. The result of this project will be a deployable virtual machine image that can be used with any cloud vendor or local hypervisor, rather than an executable program. Data is still collected as part of the process to demonstrate the performance of the built systems. There is still a small software development task as part of this project, which will be included as part of the system image.

As part of the project which requires an API server being created, as well as the fact that the templates and scripting can change as the project progresses, it is very important to use a form of SVC – Source Version Control – to effectively manage versions of the project as time progresses. Versions may change due to constraints being hit due to restrictions imposed by the cloud vendor, or networking changes may require a partial rewrite of the API or templates. Further research and testing may highlight a more efficient way of completing the project.

5.2 Software Development

As the project deliverable can be separated into several different milestones, with each milestone delivering a functional component that can be used separately from the rest of the project; it was decided that an agile approach would be the preferred way of managing development.

The project has 3 distinct initial sprints, having the databases work locally, creating a REST API and finally moving the entire project to the cloud to create a deployable image. Each sprint is as important for project completion as the other sprints; however, each sprint can produce a completely independent, functional product. Each task also requires its own set of testing and different toolsets which again highlights a compartmentalised, multi-disciplined nature of this project.

The initial local development focuses more on software deployment, rather than development. Linux bash scripts will be developed, as well as an understanding of MySQL configuration which can be brought forward into other sprints in the project. A testing phase will ensure that the MySQL service is functional and is correctly replicating across servers.

The second sprint concerns writing a basic, functional API which can be used to interact and test the MySQL servers. An understanding of the REST principles of data transmission across networks will be developed, as well as the JavaScript language and the various frameworks used to create the REST API. Understanding of MySQL from the previous sprint will provide further insight into how to construct the API to effectively and efficiently communicate with the database. Finally, in this sprint a testing phase begins once again, ensuring the API service works as expected and provides the expected response from the database.

Finally, the last initial sprint begins where the basis of the cloud infrastructure is deployed. Being able to infer architectural designs from the local configuration aids in the development of the architecture as the Infrastructure as Code is written. Once deployed, the stack can be replicated onto the cloud-based VMs using previously written start-up scripts and some minor configuration changes using the Cloud Web UI. Once again, testing is performed not only to confirm operation but evaluate system performance.

Following an Agile principle, changes are easily accepted, and in the case of the project development moving forward, some modifications and updates may be required. Being able to break down the project into such sprints enables rapid and flexible development, which in cloud systems is key to exploit new performance benefits and reliability improvements.

5.3 Toolsets

Google Cloud Platform is a market-leading vendor of Cloud Services and is used to host this project and create the required images. This is partly due to the funding requirements fulfilled on my behalf, however, it offers extensive functionality in the forms of various PaaS, SaaS and IaaS services, such as Compute Engine and CloudSQL which feature heavily and facilitate in the development of this project. It also offers a plentiful amount of VPC Network Configuration options which allow for the usage of Load Balancing both internally and externally. The Web UI is particularly powerful as it allows for configuration and deployment of any service offered, however, also offers an in-browser console that can be used for automation, as well as a freely available SDK which can be downloaded onto a local machine – however, this was not used in this project.

The platform also allows the for use of scripting in YAML to create infrastructure, as well as templates that the scripts can use for rapid redeployment in Jinja. These 2 languages combine to create a powerful tool for the rapid deployment of resources that is easily repeatable and modified to meet changing requirements. While such tools might not be utilised to deploy a simple set of resources, such as a single Virtual Machine, they come into their element when tasked to create a more complex set of resources that include multiple networks, virtual machines and firewall requirements.

While other cloud vendors such as AWS and Azure exist, Google Cloud Platform offers a fantastic and rich set of documentation, with extensive community usage and support available. Google also offers many online classes which are readily available which allow for an increased understanding of the service and its intricacies as the project progress.

To create the server stack, it was decided to use MySQL to provide the database, NodeJS to build the REST API with Apache hosting. This departs from a typical LAMP stack as while Linux, Apache and MySQL are utilised, PHP is not, making way for NodeJS.

MySQL will be used instead of other relational database servers, such as PostgreSQL due to sheer compatibility. MySQL supports more replication topologies compared to PostgreSQL (Schumacher, 2010) as well as MySQL having more management interfaces written for it compared to PostgreSQL. Again, MySQL is used extensively, not only through enterprise but through the community using open source builds, providing an extensive library of documentation and community support that isn't found in other SQL servers.

The API will be built on NodeJS rather than the commonly used PHP. This is due to the simple fact of performance. NodeJS is more modern and scales across threads better than PHP will, due to NodeJS being inherently asynchronous. This allows NodeJS to deal with increased demand without degrading performance compared to PHP (Tripathee, 2019).

5.4 Research Methods

This project employs quantitative methods to provide a conclusion. Data collected, which in this case will be response times, will be used to the performance differences between different types of Database Architecture.

The data collected from the performance testing will be objective and ordinal. This is because the data generated from the tests is absolute, as times will be recorded, with performance categorised in an ordinal manner, as better performance is ordered from the fastest response time to the slowest, with the lowest time being the fastest and therefore being preferred over slower times. Lower response times will indicate better system performance thus defining a clear victor over the other architectures explored in the project. Response times are collected from 2 load testing suites that are described in section 6.4.

With only one metric being necessary for the study, a simple table will suffice for data presentation, a chart may also be included for the visual representation of the performance of the candidate architectures.

6. Design, Development & Evaluation

6.1 Requirements

The abstract dictates that result of the project will be a Virtual Machine image containing the necessary HA Database replication service that can be deployed to any vendor platform. A basic web REST service will also be included to evaluate system performance. To complete a comprehensive review and performance test against other cloud patterns, such as PaaS services and Master-Slave configurations, multiple Virtual Machines will be created and configured to compare system performance. The configuration of such Virtual Machines will be completed via YAML and Jinja templates for use in Deployment Manager – an integral part of the Google Cloud Platform's Compute Engine. The image once completely configured can be exported as a Persistent Disk Image, which can then be migrated or imported onto any cloud platform, such as Azure or AWS.

Also, a variety of Load Balancing techniques are to be explored. This includes External (HTTP/HTTPS) Load Balancing, as well as Internal (TCP/UDP) Load Balancing must be configured to explore the possible performance enhancement this would bring. The configuration of the Load Balancing will be completed via the Web Interface.

In cases where Load Balancing will be used, Autoscaling can also be configured using Managed Instance Groups, these will also be evaluated and configured once again through the Google Cloud Platform web interface.

6.2 Design

After determining the requirements for a cloud system, it is often very useful to begin prototyping the system via the use of diagrams. There are many services which offer facilities to be able to quickly sketch up a cloud architecture, one such service is Draw.IO. The service enables the usage of correct Google Cloud Platform icon set and colour usage – various colours in these diagrams denote Regions, Zones, VPC Networks and Subnetworks. While initially, it may seem trivial to start by drawing a diagram, it allows for visualisation of the architecture being planned, thus enabling rapid development and helps provide an initial framework for the infrastructure that will be coded. In this phase of design, one begins to determine the initial requirements for the VMs, typically, how many CPU cores and how much RAM they will be configured with, as well as what zones they will be placed and subnetworks they will be part of.

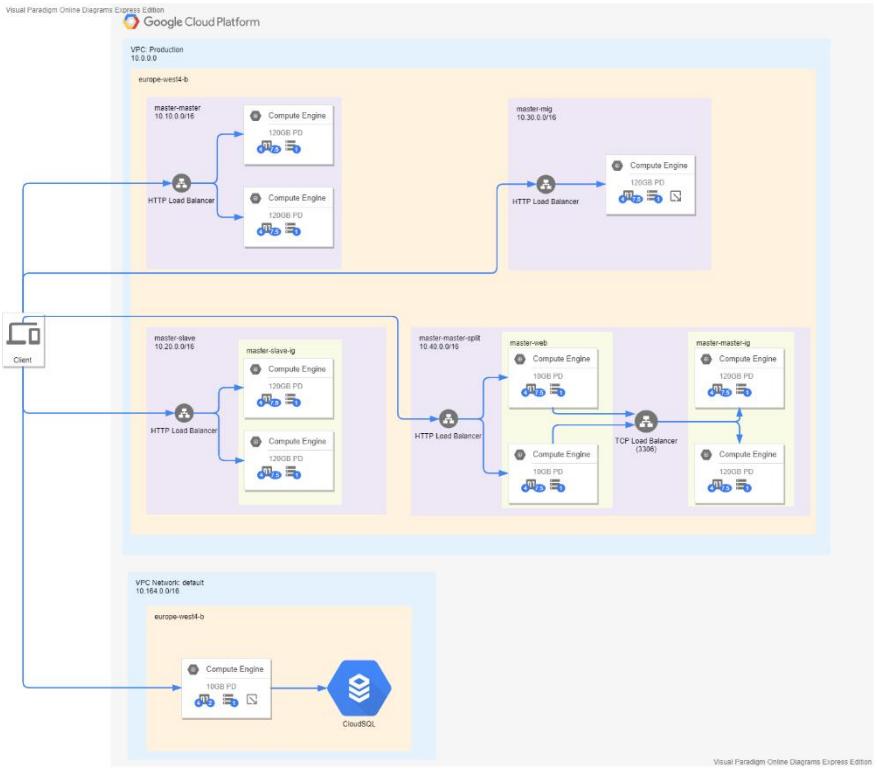

Figure 1- the finalised GCP Diagram

Once the virtual infrastructure has been planned, at least on paper, it can be used to help inform the configuration of the MySQL servers the project will be utilising. Knowing that there will be at least 2 machines per configuration being used allows for the planning of network connectivity on those instances, as well as a preliminary configuration of the MySQL daemons running on each server. MySQL has built-in replication services that allow for a Master-Master configuration, as well as Master-Slave and Group Replication as part of a Multi-Master topology, the specifics of these configurations will be discussed further on in this report. With the server having these services built-in, it was a clear choice in choosing how the replication will be set-up on the servers.

One other requirement for the project is a simple REST (Representational State Transfer) API that can be used to interact with the database servers. The API that will be deployed needs to be able to at least read from the Database, creating an endpoint that interacts with the Database by using a SELECT statement will trigger a JSON response, which will contain the result of the query, as well as response times. The response time will be a key performance metric as it will indicate in milliseconds the amount of time taken to process a request, reduced performance due to traffic would lead to higher response times.

6.3 Building

To begin the project, it was decided to have databases hosted locally before moving to cloud-hosted infrastructure. VMWare Workstation was used on a PC with 16 Threads and 16GB of RAM, such specification will allow 4 Virtual Machines to run at the same time with multiple threads and enough memory each to run their respective stacks. This allows for easier development as machines can be quickly cycled if required, code can be pushed faster to facilitate rapid development and reduces the overall cost of the project.

Ubuntu 18.04 (LTS) Server was selected as the Guest Operating System on each machine, with each machine having access to 4 threads each, and 3GB of memory allocated. The machines were connected to the network via a bridge, which allowed each machine to have a dedicated IP address that could be accessed by any machine on the network – this allows for external SSH access for easier administration. Each machine having their own address would also closely emulate a similar situation to what would be seen on the cloud, allowing for easier porting later in the process. Ubuntu was installed without any additional packages, bar SSH support, which allowed for more flexibility in choosing what packages would be part of the host stack.

To quickly replicate a similar software state across multiple virtual machines, as well as simplify the installation to the cloud VMs, a script was created which would install and configure several packages unattended. The first version of this script simply installed and configured both Apache2 – a web server – and MySQL server. The script would then proceed to create a simple Database Administrator user, which then allowed for the manual installation of PhpMyAdmin – a widely used web-based MySQL control panel which can be used for some configuration and monitoring of the MySQL instances. PhpMyAdmin was manually installed and configured in this case simply for ease and the fact that it is not a core component of the stack. PhpMyAdmin is simply being used as another means of monitoring, which can be accomplished using the MySQL CLI.

Once all machines ran the script, they would have an identical software configuration, which meant that replication could now be configured across the servers. With 4 separate VMs running, it allowed for the provision of 2 servers running in the Master-Slave configuration and 2 servers running a Master-Master configuration. The Master-Slave configuration was first completed. MySQL includes a replication service that can be used for a multitude of scenarios (MySQL:: MySQL 8.0 Reference Manual:: 17 Replication, n.d.), and will thus be used as the replication service for all of the replication schemes explored by this project.

To start, the MySQL configuration file had to be modified to enable the Replication service. The server must be bound to the IP of the VM, not localhost, this allows other servers to connect to it, which is accomplished by uncommenting the bind-address configuration line and changing the specified IP address which will usually be 127.0.01. The server ID must also be set for replication, as well as the log file and logging interval for synchronisation, then finally, the target database can be set, in this case, the database didn't exist yet, so the name needed to be noted for the configuration to work when a slave is connected. The MySQL service is then restarted for the new configuration to

take effect. Once the service has restarted, a new user should be created for the replication service to utilise, this will be called “replicator” in all instances, with the same password for ease. The database can now be created using the name specified in the configuration file, usually, at this point, the database would need to have a read lock applied to it, but seeing as the database is empty, and there are no active users, this step is skipped. Using the CLI again the query “SHOW MASTER STATUS;” is used to record the log file and position, this is used so that the synchronisation can take place.

To configure the slave server the MySQL CLI is used once again; initially only the database target for replication must be created, with the same name as on the configuration file and on the Master server. The server’s configuration file must be modified correspondingly to that of the Master server, however, the bind-address is left commented out, and the server-id must be unique. The log files will be set; however, a relay log will also be specified, and the database that will be replicated will be specified once again. As last time, the MySQL service must be restarted after editing the configuration file. Using the MySQL CLI again, the slave server must be edited using a query which specifies the master server’s IP address, as well as the user to use, including password, and the log file name and position recorded previously. Starting the Slave service on the Slave server and using the query “SHOW SLAVE STATUS\G;” displays the slave status, which shows both servers successfully connected and waiting for events.

The Master-Master configuration is essentially the same set of steps as creating a Slave in a Master-Slave configuration completed on both Masters. This may seem reverse intuitive as 2 Slaves are created in the process and not 2 Masters, however, as Slaves listen for events that happen on one server to replicate onto their copy of the database, having 2 connected means that they are both replicating from each other, rather than listening to a dedicated master.

As part of this project, a REST API was required. The API should be able to interact with the database to test functionality as well as provide a response time which can be used in benchmarking. The API utilises ExpressJS on a NodeJS backend to serve the pages and the API endpoints, and a MySQL library to interact with the database hosted on the server. ExpressJS includes many HTTP methods and takes care data serialisation which allowed for rapid development of this API.

A simple “/benchmark” endpoint is provided to allow for rapid benchmarking. The endpoint makes uses of a Haversine formula wrapped in an SQL statement and performs this on the database, the operation is designed to be both CPU and memory intensive which provides a challenging test for the instances and will allow for an adequate investigation into the performance of the servers. The API is made available on GitHub which allows for the API to be cloned locally and started on the server as part of a start-up script, which is particularly useful for Google Cloud Platform, as it allows for the unattended start-up of the web service.

As NodeJS is not a system service, some extra considerations must be made to ensure the web service is started at system start-up, as well as managed, so that any application exceptions are automatically restarted to ensure uptime. To do this, PM2 is used. PM2 allows for NodeJS applications to be started and managed under it as a system service, which allows for automatic-start when the server comes up, it will also restart the application should there be an error of some kind that would usually take the API down with it. PM2 also includes a web monitoring function, so that the Applications can be monitored remotely, extensive logging is also available for troubleshooting more complex issues.

With testing completed as highlighted in section 6.4, the move to a cloud platform can begin. Google Cloud Platform, and other Cloud Vendors, include the ability to using scripting to create infrastructure and VMs; this is often known as Infrastructure as Code due to the virtual parallels of hardware networking equipment. GCP utilises the scripting language YAML to be able to write out the deployment. However, templates can also be written in a language like YAML called Jinja, which also has similarities to Python. Jinja allows templates to be written which can then be interacted with using a YAML script. This enables the rapid redeployment of similar machines and services without having to rewrite an entire YAML deployment script each time a new deployment is required. This automation is another key benefit of Cloud Computing.

To leverage Jinja correctly, it is best to separate the components of the deployment into different templates. For example, in this project, separate templates are used to deploy the VPC Network, Subnetwork, Firewall Rules and VM instances. The templates can then be imported and interacted with in a YAML deployment script where particulars are defined such as machine names, the machine types, network types and connections are automated within the Jinja template and do not need to be rewritten in the deployment script, start-up scripts can also be imported and used to automate deployment further.

The machines are deployed using a GCP standard machine type of “n1-standard-4” which offers 4 vCPUs and 15GB of memory, however, custom types could be defined if required. 4 machines are deployed initially across 2 subnetworks each with their IP range, thus each subnetwork has 2 machines each, the amount required for the MySQL duplication. Each machine is deployed with the start-up script earlier which configures the OS for usage.

The machines are accessible after around 5 minutes of the deployment being completed, this delay is due to the OS being copied onto the virtual disks and having the script run. Once each machine deployed is confirmed as accessible, they can be configured using duplicate copies of the MySQL configuration files from the locally hosted servers, with a few minor changes to reflect the new IPs of the cloud VMs.

Now that the machines are configured, 2 in a Master-Master and 2 in a Master-Slave configuration, load balancing can be introduced into the service. Load Balancing is applied on top of an instance group, so the 2 sets of machines are put into 2 different Unmanaged Instance Groups which will be performed using the Web Interface. Unmanaged Instance Groups do not provide any Scaling abilities and machines must be configured manually before they are introduced into the groups. Load balancing can be configured either for external (incoming) traffic via HTTP or internal TCP/UDP traffic. Initially, an external load balancer will be used to send requests to alternating services to balance the load evenly, a health check will be created to check the API by waiting for a response and waiting for a set time, any failures will automatically alert the administrator. A separate outward-facing IP address is created, and the web service can be accessed by this address usually within a few minutes of configuration. All tests will be pointed through these load balanced IP addresses – again testing will be outlined in section 6.4.

As multiple masters can be configured rapidly, they are a prime candidate for autoscaling. Autoscaling requires 3 components, an image, a machine template and a Managed Instance Group – referred to as a MIG. To create the Disk Image, one can simply take an existing Virtual Machine and clone its image to create the new Disk Image. By using a previously configured Master Virtual Machine, it is possible to create an image that is readily configured with the entire stack, that can be deployed onto any Machine. Now that the image is ready, a machine template can be created, utilising that image to create a new disk. The same machine type of “n1-standard-4” is used as well

as the same zone; “europe-west-4b”. Using this template, a managed instance group can be created, using the same zone as the template created. Autoscaling in the MIG can be toggled between off, on and up only. “Up only” prevents GCP from scaling the group back down again, which may be useful for maintaining a set number of instances that will be always able to manage the same amount of traffic that caused the amount of scaling without waiting for the scaling to catch up, but of course, this will increase billing costs substantially more so than autoscaling up and down. Other configurable variables in creating a MIG also include specifying a target CPU utilisation that the number of instances will average, as well as a threshold in which the autoscaling will start a new instance from the template. These provide the logic for scaling down and up. A period can be specified in which Autoscaling will wait before increasing or decreasing the number of instances, and a maximum and minimum can be specified. With the MIG created and the first instance up and running, a load balancer can be applied in the same steps as before, and as traffic increases, the number of resources made available will also increase.

A final configuration utilises 2 separate web servers where the API is configured to connect to the previously deployed Master-Master configuration. This allows for the use of 2 separate instance groups and therefore, internal (TCP) and external Load Balancers to be deployed. For this configuration to exist, testing must be performed on the previous configuration before deployment, as the Master-Master group must have its external Load Balancer removed for the internal Load Balancer to be implemented.

The same image used for the MIG can be exported to bucket storage for use with other cloud vendors with the vendor-specific configuration required, thus satisfying the requirement for a disk image that can be used across platforms.

6.4 Testing

The first phase of testing was completed as part of building out the service locally. This phase of development was an opportunity to ensure the basic functionality of MySQL and the REST API were correct, as well as ensure the start-up shell script was written correctly. Once the MySQL instances had been installed and the first configuration (Master-Slave) had been completed, it was vital to ensure that the replication service was indeed running as intended. Initially, there were issues in getting the Slave server to connect to the Master server; this was caused by misconfiguration of the bind addresses used on the Slave server; thus, it would not accept events from the master. Once this was diagnosed; the issue was soon resolved, and replication was working correctly. A test dataset was inserted into the database and using PhpMyAdmin, the replication could be confirmed, as intended the dataset had indeed been pushed from the Master to the Slave in real-time, ensuring the 2 databases were completely synchronised. This confirmed that the time required to perform the duplication would have a very marginal impact on performance gains.

The Master-Master configuration worked perfectly on the first connection attempt, with lessons learned from the previous configuration. Both masters connected and immediately listened for events. The same dataset used to check the Master-Slave was used to also test these instances. Imported on one Master, again, the other Master picked up the dataset immediately. To ensure that either Master could read and write to the database and duplicate the changes; insert statements were used to check this functionality and as expected, the changes were replicated instantly.

The REST API was also tested before being moved to the cloud. The NodeJS application was started, initially without the PM2 process manager on the machine running the VMs, not the VMs themselves. The API was successfully able to perform calls on the Database on the remote machine, thus also ensuring the API could run on a separate machine if required. The API was packaged then

cloned on to the VMs. Again, without using the PM2 manager the REST API was able to respond to requests, even though the Proxy running on the VM. However, when the application was being managed through the PM2 process manager, an issue made itself apparent. It transpired that the entry point one would usually use to start the program meant that through PM2, the server was not allocated a port, and therefore, wouldn't receive any connections. This was easily solved by starting the server through a different script within the application. Having this confirmed working meant that it would now be possible to rapidly deploy this stack to the cloud.

Once deployed to the cloud, a similar set of tests to ensure that all was working correctly was utilised. While the Apache webserver was working as expected, as well as PhpMyAdmin access and standalone databases, there were initially some minor issues with the replication service. This was being caused by a different version of MySQL being used on the VMs hosted in the cloud that required the use of securely hashed passwords. As a workaround, the servers could be forced into using native MySQL passwords by altering the replication users and by also forcing the servers to accept native passwords. After a service restart on all the servers, replication was restored.

Once the stack service functionality was confirmed to be working, the load balancing also needed to be confirmed. It typically takes a short amount of time for a newly deployed load balancer to be publicly accessible via the given IP address. One this short time had passed, it was accessible as had been expecting, and load balancing could be confirmed by opening SSH access to included servers and checking server logs. By trying to access a page that didn't exist on each server, this would generate a clear error in the logging. As expected, the error would appear in the log on one server, and after multiple refreshes, it could be confirmed that the error would appear in both servers, confirming load balancing across the servers.

Now the stack was fully deployed onto the cloud, it became possible to begin load testing on both database architectures. To complete this, a couple of load testing utilities were utilised, Loader.io and K6 proved to be the most feature-rich options available for a project of this scope. While Loader.io can provide a heavy saturation of users over a short period, which is good for simulating a rapid burst of sudden demand, K6 offers a more sustained level of testing with fewer connections, but over a much longer time. Both approaches are very common situations that web and storage stacks will encounter, thus, having performance metrics from both scenarios offer valuable insight into the performance of this deployed stack. To compare performance against a PaaS offering from the same vendor, an identical database was also deployed using CloudSQL, with the same API service deployed on the other servers connecting to the hosted database.

6.5 Operation

Anyone can maintain this image with minimal configuration. If hosted on a service other than GCP some initial reconfiguration will be required. However, after that, databases can be easily added and replicated across servers either via CLI access, or the inbuilt PhpMyAdmin access. This will allow for rapid deployment of any service of any scale, depending entirely on the resources allocated to the machine. As is the image also supports being used in an Autoscaler, and of course, can be modified quickly and easily for any scaling service by changing bound IP address when required.

The image is accessible at https://storage.googleapis.com/16605155/master-stack-v3.tar.gz, where it can be downloaded. To use the image, follow the guidance of the cloud vendor that is intended to be used. The web service is started by running 'pm2 start /mysql-rest/bin/www –watch'. Follow PM2's guidance on daemonizing the process for the service to be started at system startup.

The REST API client can also be very quickly modified and repurposed due to its MVC architecture and NodeJS backend. As is, a backend is provided to show the performance of the database via the use of the Haversine formula on a complex dataset, but future endpoints can be added by the router functionality provided by the Express framework.

It is also worth mentioning that the image can also be used in a variety of locally hosted hypervisors, with the same level of configuration as expected when moving the image between cloud vendors this allows for hybrid cloud architecture in the future if that is what is required by anyone deciding to use the image.

7. Conclusion

	10K Requests/Min	50 Requests/Min for 12 Minutes
PaaS DB	96	95.45
Master-Slave with External Load Balancing	95	95.13
Master-Master with External Load Balancing	94	94.72
Master-Master with Full Load Balancing	93	95.11
Master-Master with Load Balancing & Autoscaling	95	119.24

Table 1 - Average Stack Response Time in milliseconds (10K Column rounded due to service restrictions)

In total, 5 distinct architectures were tested per the method outlined in section 6.4; with response times recorded as can be seen in Table 1.

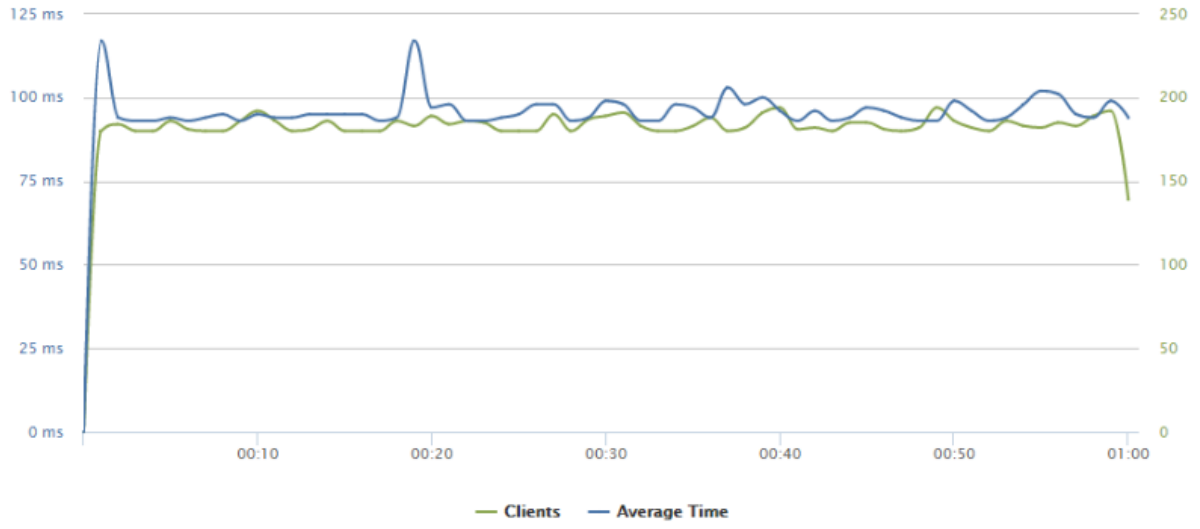

Figure 2 - Graph showing response times of PaaS Database

The first test completed was the benchmarking of combining the web service with a pre-existing PaaS Database solution, CloudSQL. Running a mild sustained load for 12 minutes resulted in an average response time of 95.45ms and under a large demand spike for 1 minute returned a 96ms response time from the API. The benchmarking tests appeared to create good response times, but with an increased amount of variation compared to the other configurations deployed and tested.

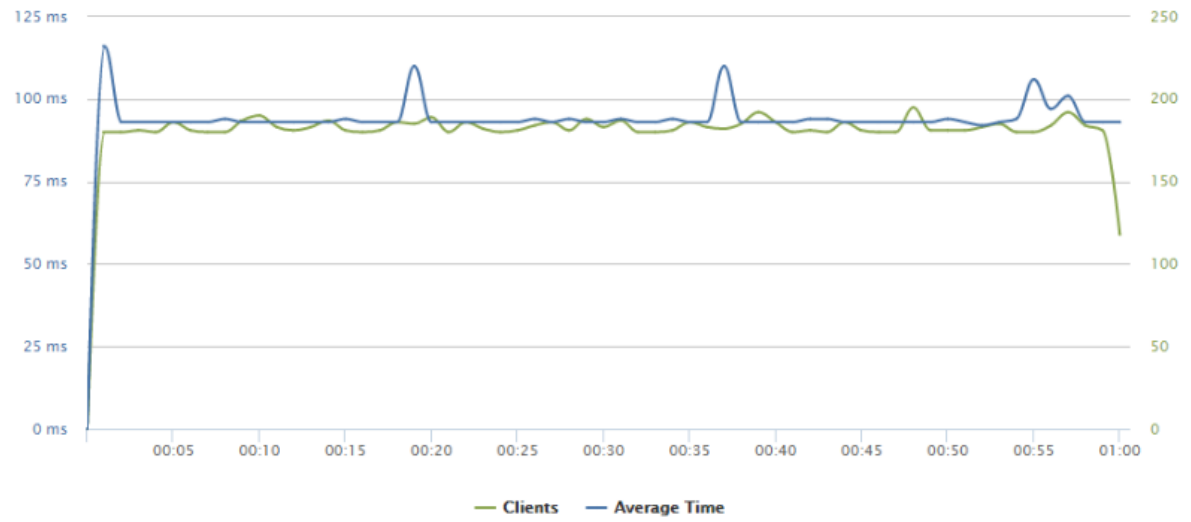

Figure 3 - Graph showing response times of the Master-Slave configuration w/ External Load Balancing

The Master-Slave benchmarking produced response times of 95ms for the large demand spike test, and 95.13ms for the mild sustained load. Compared to the PaaS configuration, the response times are improved but only slightly, with only nanoseconds separating the configurations. However, note the much more stable response times over the benchmark, it should be noted that spikes of +10ms are displayed. This is most likely due to query queuing by the SQL service running on the server, as the requests are routed to the Master only.

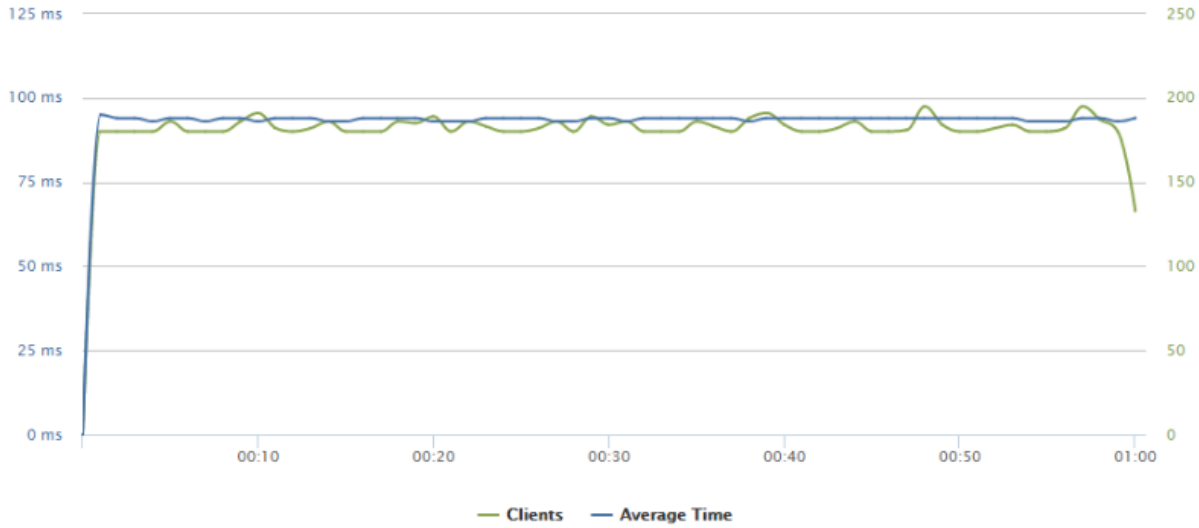

Figure 4 - Graph showing response times of the Master-Master configuration w/ External Load Balancing

The Master-Master configuration with External (HTTP) Load Balancing responded to the benchmark endpoint requests in 94ms for the spike test, and 94.72ms for the sustained test. Again, only a small improvement compared to previous configurations, however, response time stability is greatly improved, with only very marginal differences recorded. This is likely due to database queries now able to take place on multiple servers, rather than just one master, removing the queueing bottleneck from the configuration.

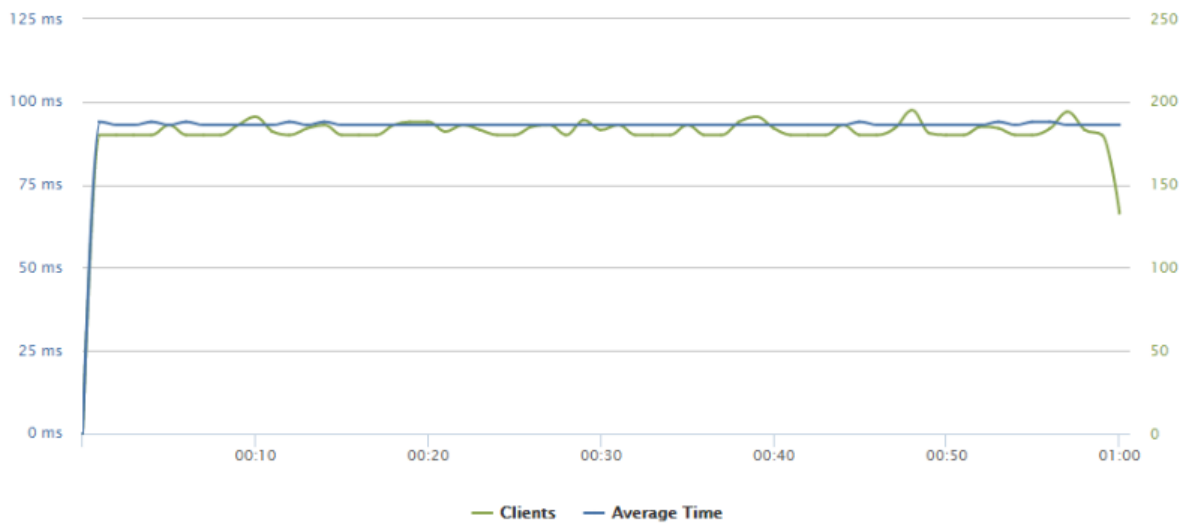

Figure 5 - Graph showing response times of the Master-Master configuration w/ Full Load Balancing

The same Master-Master configuration, however in this run with a separated web server and Internal (TCP) and External (HTTP) load balancing was able to produce average response times of 93ms for the load spike test, and 95.11ms for the constant demand test. A further small improvement compared to just External Load Balancing during the spike test, but the longer duration mild demand resulted in a higher response time that just External Load Balancing. However, response times are at the point mostly flattened, more so that previously, this is likely due to SQL requests making their way to whichever server is experiencing the least load due to the TCP load balancing working on the MySQL listening port. This result also demonstrates how little of an impact the web service has on overall stack performance, as decoupling it from the SQL service only created marginal gains over hosting the web service and database on the same server.

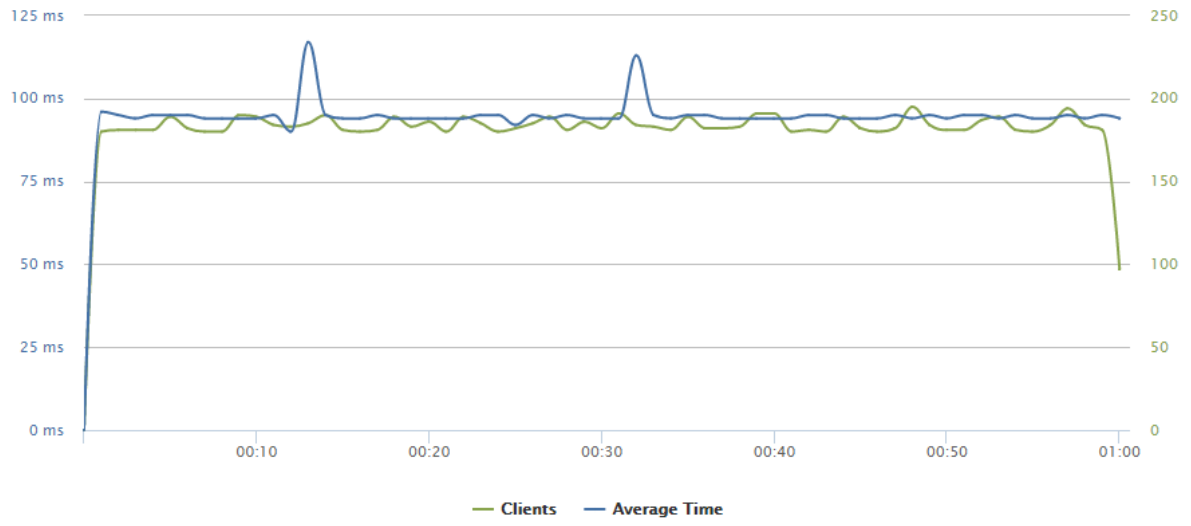

Figure 6 - Graph showing response times of the Master-Master configuration deployed with Autoscaling and Load Balancing

The final test run was on a similar Master-Master configuration as before, however, configured in a Managed Instance Group with Autoscaling enabled. This set of results was particularly interesting as it displayed increased response times of 95ms and 119.24ms for the spike and constant demand test

respectively. Also, for the first time on these sets of benchmarks, an error rate was noted, .8% on the longer duration test, and, 1.8% on the spike test. This is potentially due to constant load balancer updates as machines are allocated, however, the 1-minute spike test falls within the Autoscaler cooldown period, and therefore, resources are not allocated in time.

While the fully Load Balanced Master-Master configuration performs slightly better than other configurations in the 1-minute demand spike benchmark, the Externally Load Balanced Master-Master performs better in the more reasonably expected mild constant demand benchmark. One might also consider the slightly more simplified deployment of the single Load Balancer to be preferable scenarios while in some scenarios, having the decoupled web and SQL servers using load balancing throughout provide higher throughput in high demand situations may be a requirement.

In either of these scenarios, the bespoke deployments offer a higher level of control, stability and performance than what could be realised by using a PaaS solution. While PaaS is particularly useful for cases where ease of setup and management is required, designing a bespoke solution for any production environment will offer better performance over the long term.

8. Reflective Analysis

This project felt like it was a success for me; managing to produce the deliverable was fantastic, but it also satisfied my curiosity into a subject I deeply enjoy studying. Performing constant modifications and testing using GCP felt brilliant and I enjoyed working on the project.

I must admit, I am somewhat surprised with the tight spread of results. Only 3ms separates all configurations on the so-called spike benchmark, although, a large margin was observed with the Autoscaling instances on the spread demand benchmark, which again initially surprised me. However, breaking down the result, looking at the graph and having a firmer understanding of the field that previously, I feel as if I was able to come up with an explanation to the anomaly observed.

Looking back on the project, I feel as if some extra benchmarks could be performed if performed again. GCP offers the user the ability to use either HDD or SSD persistent storage. In this project, HDD was used throughout, and it would've been interesting to see if there was an observable difference to response time, I suspect there would be due to the increased throughput of an SSD, as well as faster deployment time on the Autoscaler. It would also be interesting to increase the resources available to the VMs (scaling out) and seeing if there is a noticeable advantage, and if so, where this drops off.

While the load testing facilities are excellent, I was only ever able to use the free tiers of the platforms, which restricted the amount of testing possible. This was due to the inhibitive cost of these facilities, as they are often targeting the enterprise market, not university students. The higher tiers of these facilities allow for longer test periods with a larger number of users performing requests on the servers.

Finally, I will note that toward the end of this project, particularly during the testing phase, the world was enduring through the COVID-19 global pandemic. This did, unfortunately, limit access to resources at the university, however, it did well to help ensure students could still study effectively. An interesting thing to consider is the impact this pandemic has had on cloud services. Due to the lockdowns enforced across the global, video calling and other work from home services have been used extensively, as well as an increase in video streaming and online multiplayer. Azure – Microsoft's cloud platform – runs not only customer applications but their online services, such as Office 365, Skype, Teams and Xbox Live and has seen in some cases a 7.5x increase in resource

utilisation. One can assume such increases are being seen elsewhere, such as GCP. It would be fascinating to see that if due to this increased demand has led to higher API response times and therefore been a factor in the results of this project.

9. References

Almorsy, M., Grundy, J. and Müller, I., 2016. An Analysis Of The Cloud Computing Security Problem. [online] arXiv.org. Available at: <https://arxiv.org/abs/1609.01107v1>.

Arnold, U., 2013. Advancements In Cloud Computing For Logistics - IEEE Conference Publication. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/abstract/document/6644146>.

Avram, M., 2014. Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective. Procedia Technology, 12, pp.529-534.

Bilyk, V., n.d. Elasticity Vs Scalability: Main Differences In Cloud Computing. [online] Theappsolutions.com. Available at: <https://theappsolutions.com/blog/cloud/cloud-computing-elasticity-vs-scalability/>.

Carroll, M., van der Merwe, A. and Kotze, P., 2011. Secure cloud computing: Benefits, risks and controls. 2011 Information Security for South Africa, [online] Available at: <https://ieeexplore.ieee.org/abstract/document/6027519>.

Durski, K., Singaravelu, S., Teo, J., Naidoo, D., Bawo, L., Jambai, A., Keita, S., Yahaya, A., Muraguri, B., Ahounou, B., Katawera, V., Kuti-George, F., Nebie, Y., Kohar, T., Hardy, P., Djingarey, M., Kargbo, D., Mahmoud, N., Assefa, Y., Condell, O., N'Faly, M., Van Gurp, L., Lamanu, M., Ryan, J., Diallo, B., Daffae, F., Jackson, D., Malik, F., Raftery, P. and Formenty, P., 2017. Development, Use, and Impact of a Global Laboratory Database During the 2014 Ebola Outbreak in West Africa. The Journal of Infectious Diseases, [online] 215(12), pp.1799-1806. Available at: <https://doi.org/10.1093/infdis/jix236>.

Gartner. 2019. Gartner Forecasts Worldwide Public Cloud Revenue To Grow 17.5 Percent In 2019. [online] Available at: <https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g>.

Google Cloud Platform. 2020. Compute Engine Service Level Agreement (SLA). [online] Available at: <https://cloud.google.com/compute/sla>.

Google Cloud Platform Documentation. 2020. Autoscaling Groups Of Instances | Compute Engine Documentation. [online] Available at: <https://cloud.google.com/compute/docs/autoscaler>.

Goyal, S., 2014. Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review. International Journal of Computer Network and Information Security, 6(3), pp.20-29.

Liu, S., Chan, F., Yang, J. and Niu, B., 2018. Understanding the effect of cloud computing on organizational agility: An empirical examination. International Journal of Information Management, [online] 43, pp.98-111. Available at: <https://www.sciencedirect.com/science/article/pii/S0268401217307090>.

Mao, M. and Humphrey, M., 2011. Auto-Scaling To Minimize Cost And Meet Application Deadlines In Cloud Workflows - IEEE Conference Publication. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/abstract/document/6114435>.

Svitla.com. 2014. Pivotal Role Of Databases In Businesses. [online] Available at: <https://svitla.com/blog/the-role-of-databases-for-business>.

Schumacher, R., 2010. Comparing Mysql And Postgres 9.0 Replication. [online] TheServerSide.com. Available at: <https://www.theserverside.com/feature/Comparing-MySQL-and-Postgres-90-Replication>.

Tripathee, Y., 2019. NODE.JS Vs PHP Comparison - Get The Job Done Purpose. [online] Medium. Available at: <https://medium.com/@yuiltripathee/node-js-vs-php-comparison-get-the-job-done-purpose-d3d63351ea8a>.

Wieder, P., Butler, J., Theilmann, W. and Yahyapour, R., 2011. Service Level Agreements For Cloud Computing. New York, NY: Springer New York, pp.43-67.