

|  |  |  |
| --- | --- | --- |
| **Project Title** | **:** | UI Airlines |
| **Module Code** | **:** | CT071-3-3-DDAC |
| **Intake Code** | **:** | UC3F1701SE |
| **Lecturer Name** | **:** | DR. KALAI ANAND A/L RATNAM |
| **Hand in Date** | **:** | 2 OCTOBER 2017 |

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| TP031955 | Ngui Tze Ken |

# Acknowledgements

Firstly, I would like to thank my parents for their continuous support in my studies I would also like to thank my lecturer Dr. Kalai for his guidance and expertise in the field of cloud computing. I would also like to thank my friends and fellow classmates for providing feedback to my ideas that would allow me to maintain the right path. My gratitude is also extended to Elon Musk for being an inspiration for me in continuing my journey into software engineering. I would also like to thank the Wiremod community for Garry’s mod for giving me an early understanding into the world of software and computing.

Table of Contents

[Acknowledgements 2](#_Toc494243669)

[1 Introduction 1](#_Toc494243670)

[2 Project Plan 2](#_Toc494243671)

[3 Design 3](#_Toc494243672)

[3.1 Design Considerations 3](#_Toc494243673)

[3.2 Modelling 3](#_Toc494243674)

[3.2.1 Use Cases 3](#_Toc494243675)

[3.2.2 Sequence Diagrams 4](#_Toc494243676)

[3.2.3 Data Modelling 6](#_Toc494243677)

[3.2.4 Sitemap 6](#_Toc494243678)

[3.3 Cloud Architecture 7](#_Toc494243679)

[3.4 Cloud Design Patterns 9](#_Toc494243680)

[4 Implementation 10](#_Toc494243681)

[4.1 Application Development 10](#_Toc494243682)

[4.2 Azure Publishing 15](#_Toc494243683)

[4.3 Application Scaling 16](#_Toc494243684)

[4.3.1 Reliability & Performance 19](#_Toc494243685)

[4.4 Testing 20](#_Toc494243686)

[4.4.1 Unit Testing 20](#_Toc494243687)

[4.4.2 Performance Testing 24](#_Toc494243688)

[4.4.3 Analysis 25](#_Toc494243689)

[4.5 Managed Databases 26](#_Toc494243690)

[5 Conclusion 30](#_Toc494243691)

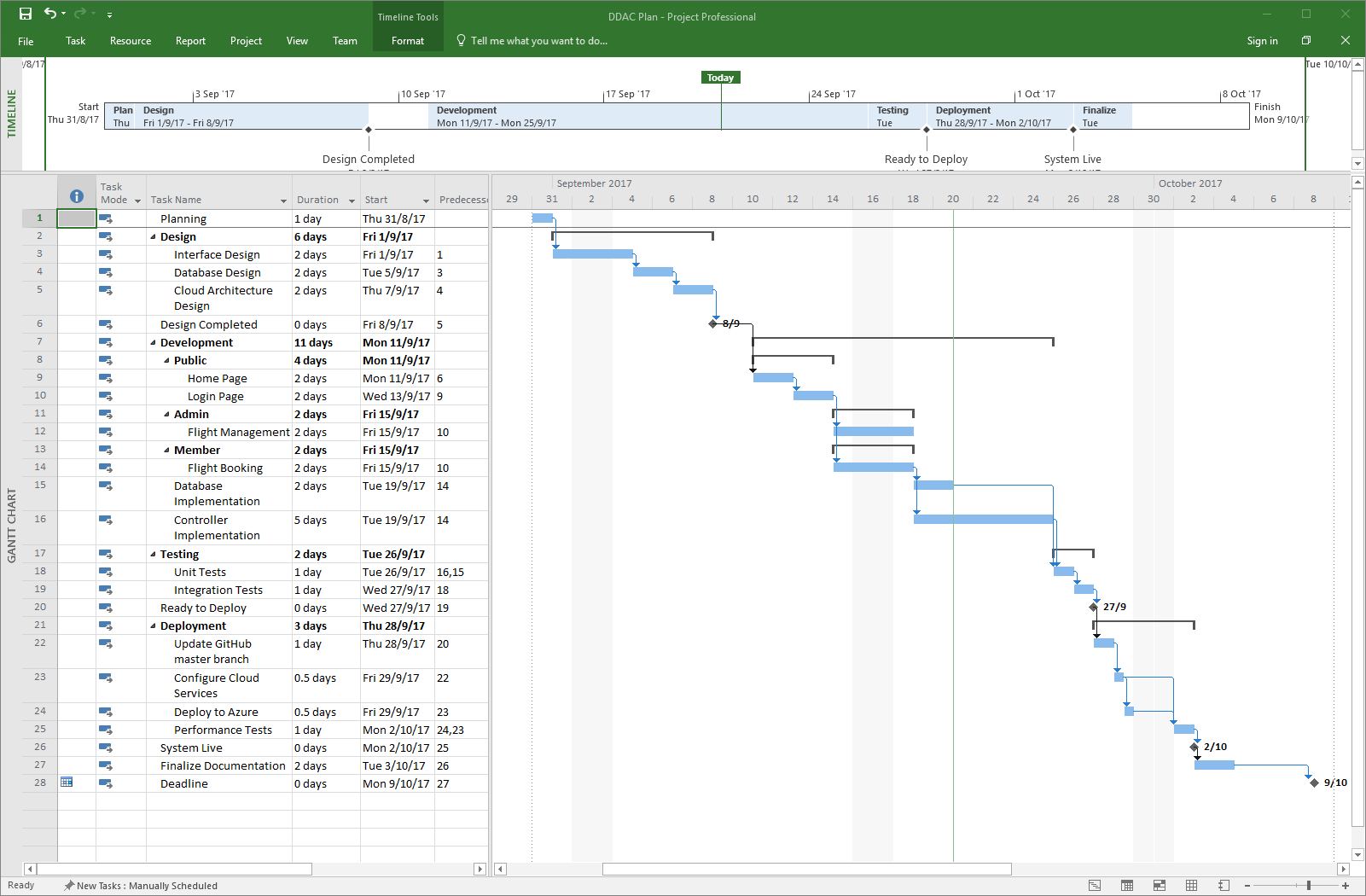
[6 References 31](#_Toc494243692)

# Introduction

Ukraine International Airlines (UIA), is looking at designing and developing an Online Flight Booking System. UIA looked at both Microsoft Azure and Amazon Web Services and chose Azure. Hence, the resulting UIA Online Flight Booking System will be developed and deployed onto the Azure cloud platform. With the advent of cloud services, it is no surprise that such a decision was made.

The application provides UIA’s clients the ability to create accounts and book flights online. Because UIA is looking to expand, the need for high availability and performance of said services has become crucial to the company’s success. This document details the development of the solution to UIA’s problems as well as how the resulting system is deployed onto the Azure cloud. It will also explain the decisions made in the designing and development of the system.

# Project Plan

The Gantt chart on the left details the timeline of the project. As with many projects, the timeline changes as issues are discovered during the development. This Gantt chart is the latest iteration of the project plan.

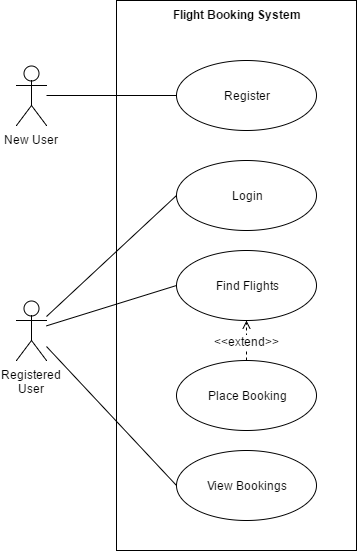
# Design

## Design Considerations

Before the design process begins, several assumptions and considerations have to be detailed first. One of which is UI Airlines’ intention with the system. Their goal is to expand into new markets, mainly into the US. They are also planning to test the system in the SEA region with a few routes. It should also be considered that for development and proof of concept, the team has been given RM 150 a month of Azure credit and is required to work around that.

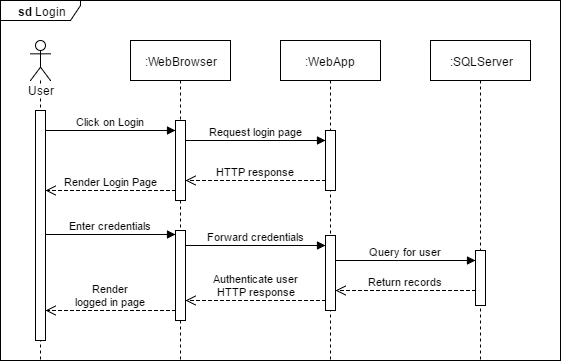
## Modelling

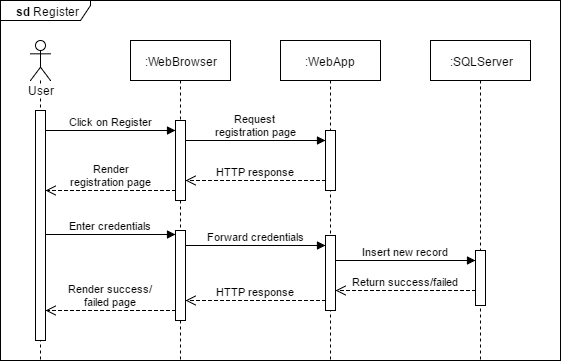
### Use Cases

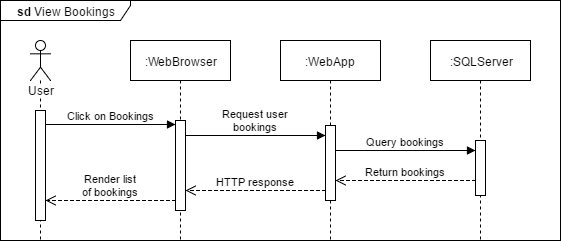
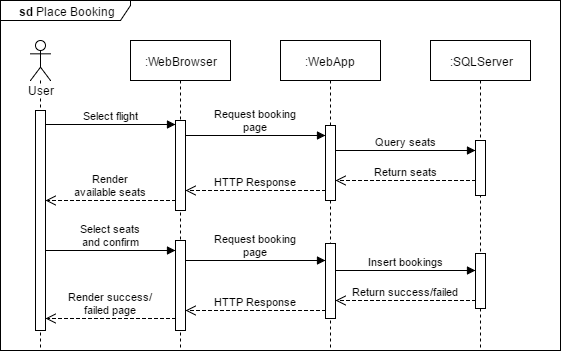
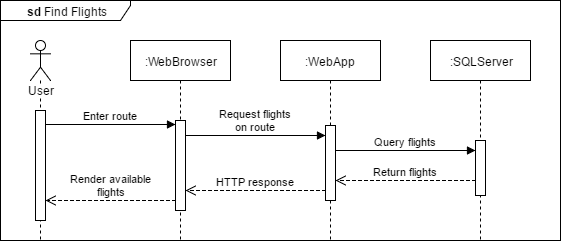
The use case diagram to the left identifies the functionalities available to each user. New users are able to register for an account whilst Registered users can login, find flights, place and view bookings.

### Sequence Diagrams

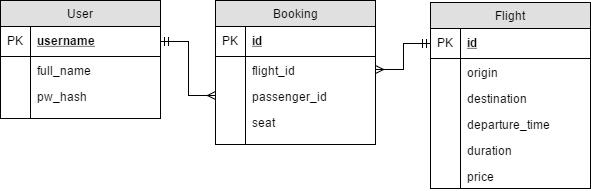
The sequence diagrams below correlate with each use case to detail how the application will execute the use cases.



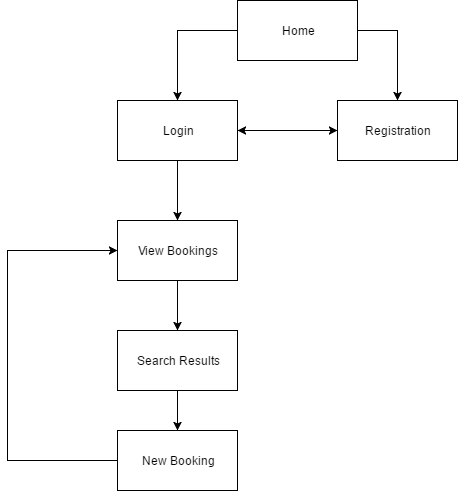




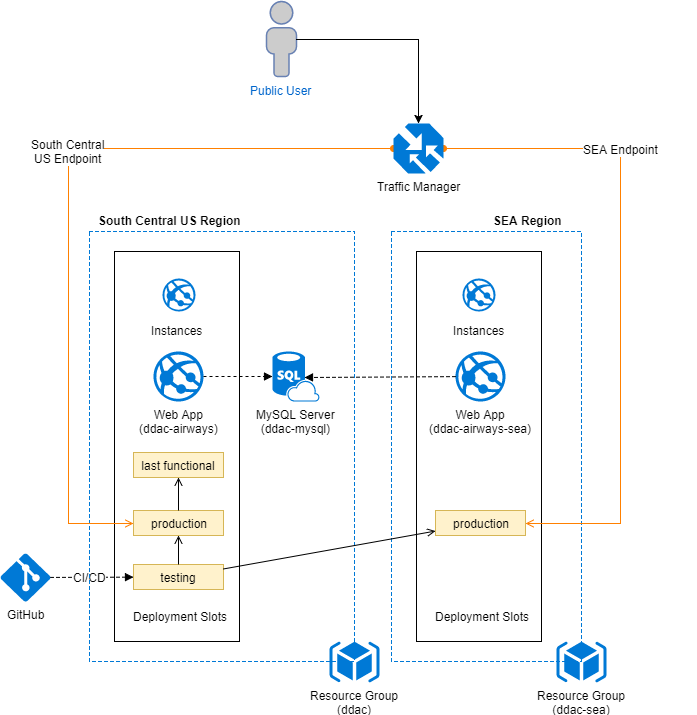
### Data Modelling

The ERD above shows the database model used for the application. The user table contains all user information. The flight table contains all available flights and the booking table contains the bookings made by the user with a link to the corresponding flight.

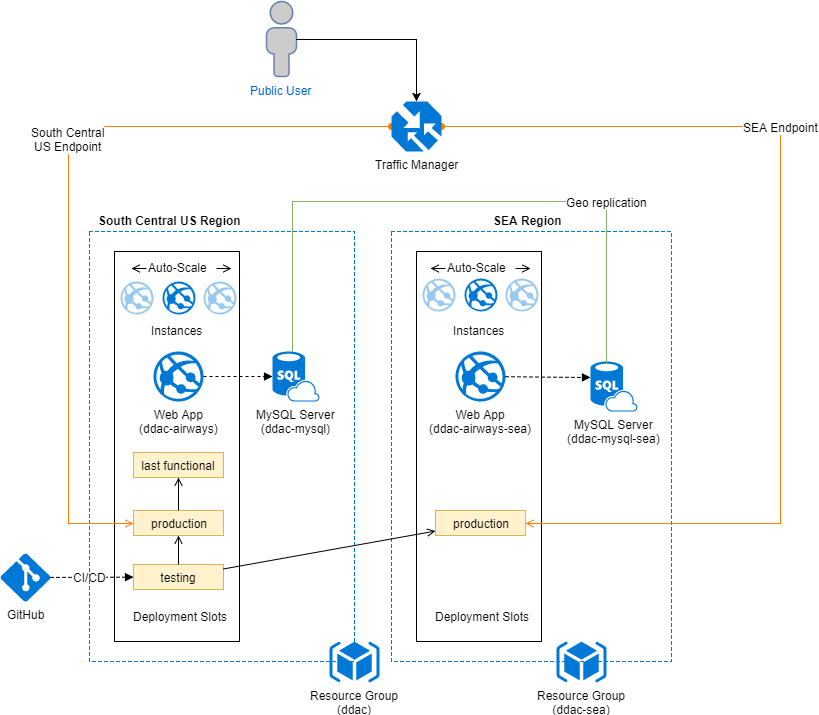
### Sitemap

The Sitemap on the left shows the flow of the application pages. All users begin at the Home page.

## Cloud Architecture



The diagram above is the cloud architecture used for deploying the application to the Azure cloud platform. As mentioned in the design considerations, UI Airlines is looking to expand primarily into the US region and slightly into the SEA region. Hence the decision to place the primary web app in the South Central US region and a secondary instance in the SEA region. Because the application was written using PHP and MySQL, a MySQL server was provisioned instead of the standard SQL server. This is also placed in the South Central US region and the database is shared with the SEA web app. The diagram above is what has been implemented based on the budget restrictions. The ideal architecture would be the one shown below.

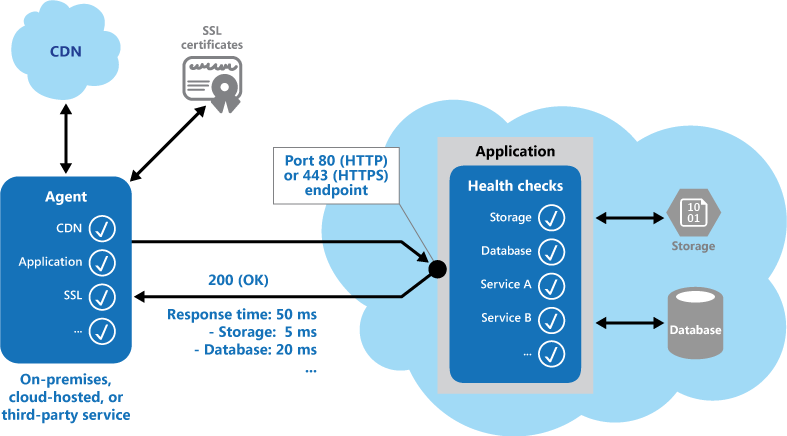


The total cost incurred by implementing the architecture above is shown in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Service type | Region | Description | Estimated Cost |
| App Service | South Central US | 3 instance(s) x 744 Hours, Size: S3, Standard tier | RM3,972.96 |
| App Service | Southeast Asia | 3 instance(s) x 744 Hours, Size: S1, Standard tier | RM993.24 |
| Traffic Manager | Central US | 5 million DNS queries/mo, 2 Azure endpoint(s) | RM15.22 |
| Azure Database for MySQL | South Central US | Standard: 400 Compute Units, 1 Servers, 125 GB Storage | RM1,289.89 |
| Azure Database for MySQL | Southeast Asia | Standard: 100 Compute Units, 1 Servers, 125 GB Storage | RM477.42 |
|  |  | **Monthly Total** | **RM6,748.72** |
|  |  | **Annual Total** | **RM80,984.69** |

## Cloud Design Patterns

One of the core cloud design patterns that were implemented is the Health Endpoint Monitoring Pattern. This pattern was introduced as a method for probing the health of a particular instance of an application deployed on the cloud. It allows for either the hypervisors or a traffic manager to check the status of an instance and if necessary, take automated action such as restarting said instance or redirecting traffic to other instances.



The diagram above is an example of a health check endpoint which monitors the storage, database and other services needed for an application to run. For the UI Airlines system, this is implemented using PHP in the ‘health.php’ file. An endpoint is created which will probe the web components as well as the database for a response. If any of them fail the health check, it responds with a 500 error and returns a JSON with the components’ statuses so the failed component can be identified. This is done as the response code indicates the status of the application and, optionally, any components or services it uses. The latency or response time check is performed by the monitoring tool or framework. The figure provides an overview of the pattern (Narumoto, et al., 2017).

# Implementation

## Application Development

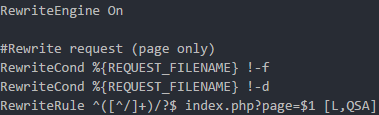
Development of the application was done using PHP with a MySQL database. The application uses an MVC structure which consists of the model, view and controller. The project is structured as shown in the image to the right. All requests to the application go through the index.php file. This file invokes the init.php which initializes the whole application by calling the classes required. The index.php page also provides a basic layout for the page, calling components of the page such as the header content, navigation bar and footer using the require\_once function as shown below.



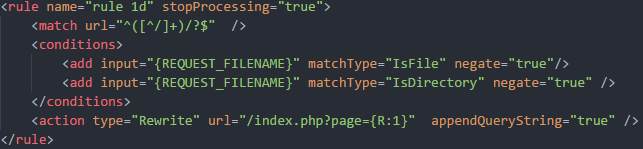
The pages are then requested based on a query string with the “page” variable. If there isn’t a page variable set, it will return to the homepage.



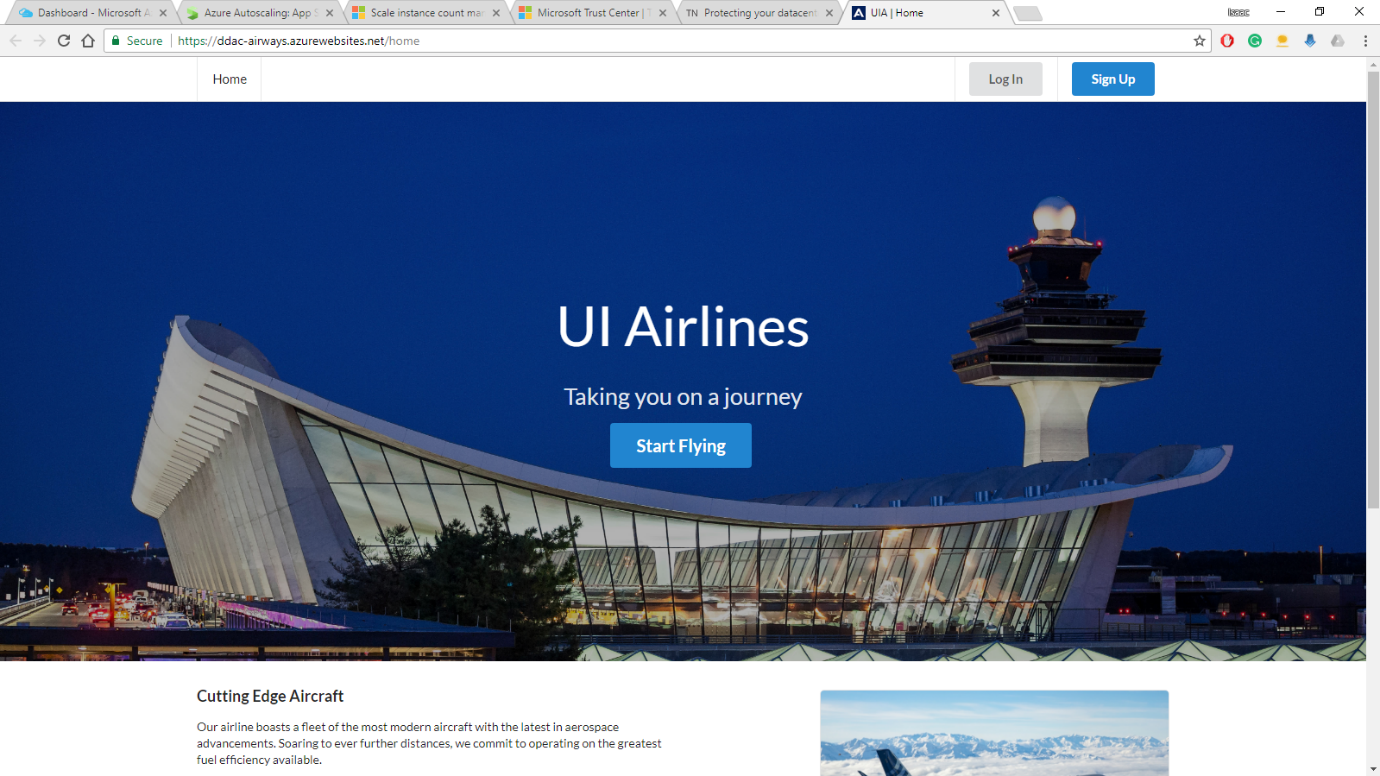
An example of a page request would be http://localhost/index.php?page=home. However, to avoid the users needing to type such a query, the .htaccess file is used with mod\_rewrite to rewrite URLs to the required format. For example, a request for http://localhost/home needs to be sent to the PHP server as the link earlier. The rules for the .htaccess file to perform such a rewrite is as follows:

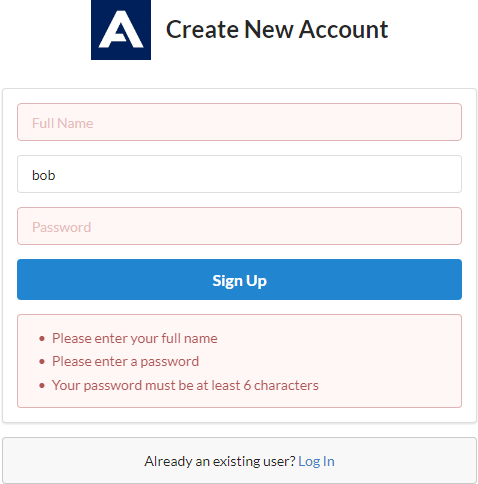


However, this rewrite rule only works with the apache server, when deploying to the Azure server, which runs IIS instead, the rule needs to be rewritten in web.config. For that purpose, the web.config rules are as follows:

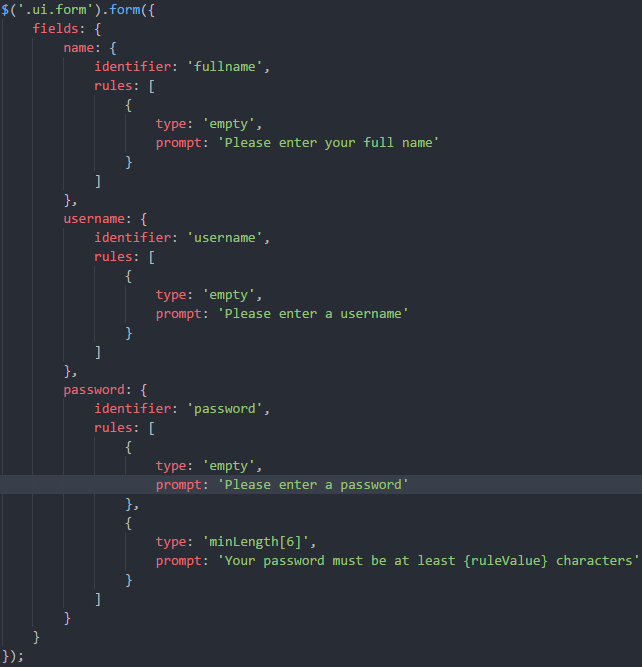


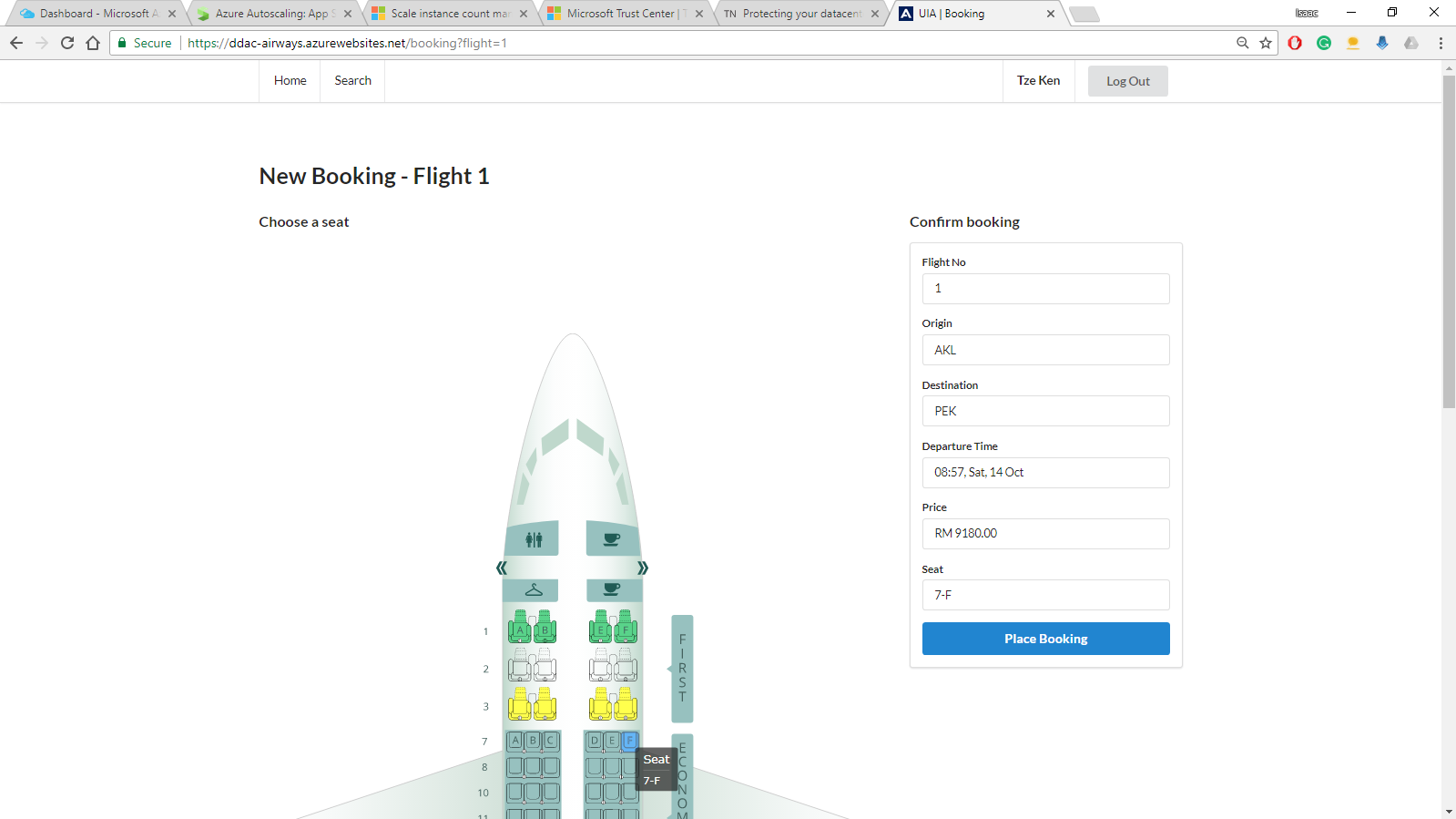
These rules will rewrite the URL to the format expected by the PHP server. The design of the web interface is done using a HTML5/Javascript framework known as Semantic UI (Semantic UI, 2017). Semantic UI is an open source frameworks that provides UI elements for use in a web page. The image below is the screenshot of the homepage as designed using the Semantic UI framework.

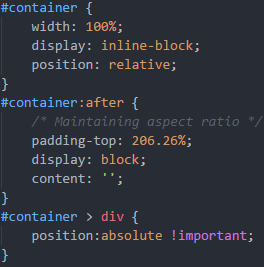




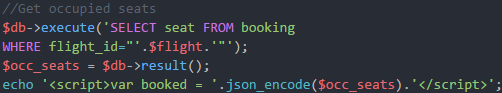
The framework provides features such as built in error checking using Javascript as shown in the image above. With Semantic UI the code to implement such a feature is much simpler, merely specifying the conditions and an error message in Javascript and a name in the HTML.



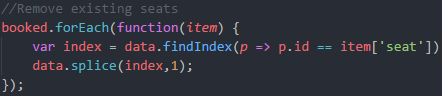
The most challenging aspect of the application development was implementing the seat picker. A third-party Javascript library was used called Anychart (AnyChart.com, 2017). The image below shows the result obtained:

The library uses SVG elements to display the seats, this caused several issues with the CSS, breaking the containers used by Semantic UI. A few CSS lines were able to force the seat picker to use a fixed aspect ratio and fill up the container correctly.

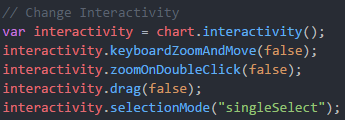
The next issue is that the data for which seats are available is stored as a multi-dimensional JSON array. To remove the seats available, the PHP first queries the database and creates an array of the unavailable seats.



Then, a Javascript function removes the unavailable seats from the selectable array by looping through it.



Finally, because the library is meant to be used with interactive charts, such as maps, it allows scrolling, zooming and other features which is not needed in this seat picker which is why they are disabled at the script initialization.

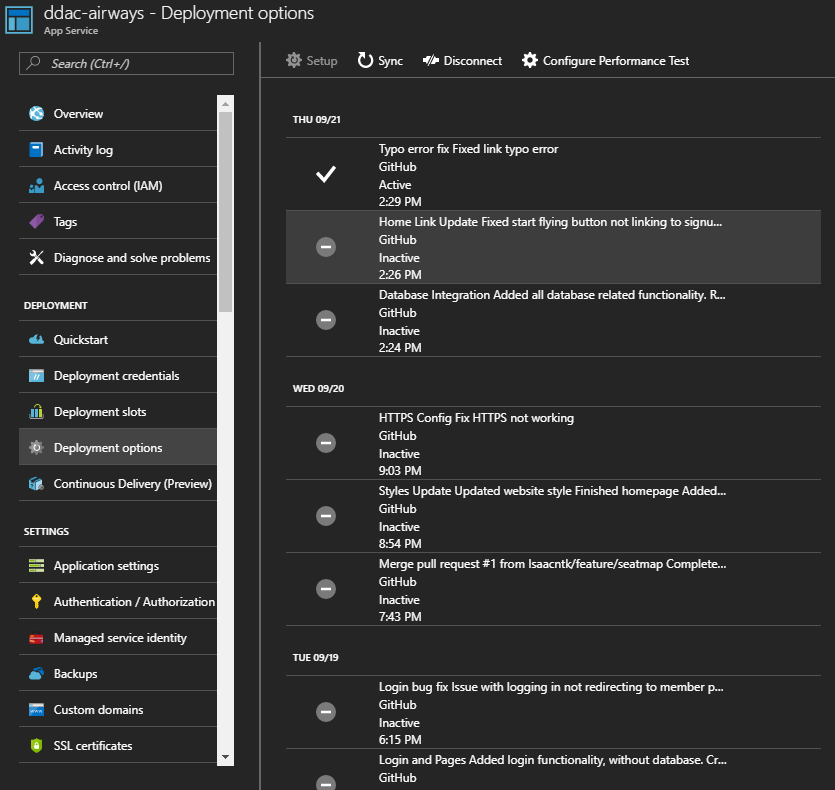


The reason these features were so difficult to implement is because the documentation provided for the library wasn’t comprehensive enough, making certain features very difficult to find.

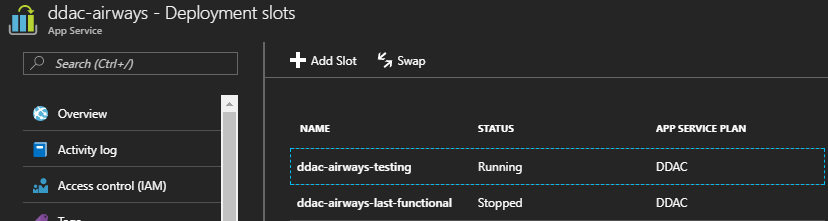
Finally, some flight data was needed for the application to return results and allow bookings. The data was generated using an open source tool known as generatedata.com (generatedata.com, 2017). The data generated was then inserted into the database which is already deployed onto Azure directly. A full demo of the functional system is included in the attached CD or online at either <https://web.microsoftstream.com/video/c24ac21a-1315-4979-a7da-083042f10d63> or <https://youtu.be/bn-rNQt0xkw>. The source code for the application is available at <https://github.com/Isaacntk/ddac-airline>

## Azure Publishing

The application was published using the Azure automatic deployment options through GitHub. The option allows for all changes pushed to a particular branch in GitHub to also be deployed directly to the Azure servers as shown below along with the branch history



This method was used in conjunction with the deployment slots to provide a continuous delivery environment. This practice is known in the industry as DevOps. There are two additional slots apart from the one in production named testing and last-functional. These are shown in the image below:



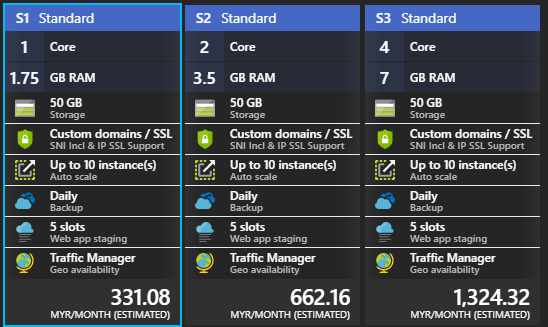
When the application was deployed for the first time, it was found that there was an issue with the URL routing. After some investigation, it was found that because the development was done on an apache server locally and uses a .htaccess and mod\_rewrite for URL rewriting, it didn’t work on the Azure servers. Since Azure runs an IIS server, the rules had to be converted to a web.config file instead (Sangapu, 2015).



Instead of an SQL server, a MySQL server with Azure integration was used instead as the development was done with PHP and MySQL. This provides all the features that come with an Azure SQL database such as scaling and monitoring but with MySQL instead (Microsoft, 2017). Since the database operates independently of the application and cannot be stopped, it was used directly during the development to avoid the need to change configuration files upon pushing changes to the production stage.

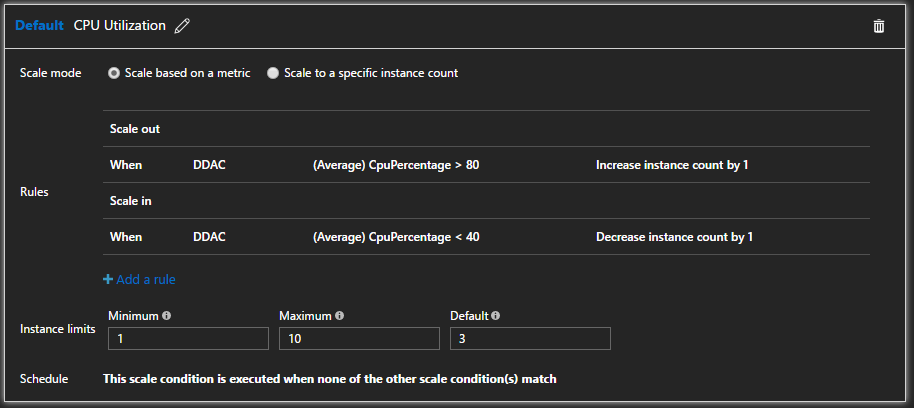
## Application Scaling

The web application deployed to both the South Central US and SEA region will be using the S1 tier App Service. This is mainly due to the budget restrictions of RM150. When the application is approved and ready to be released for public use, the application deployed in South Central US should be upgraded to the S3 plan. As a 4 core 7GB RAM server will be enough for the expected traffic. The SEA application will remain on an S1 plan until UI Airlines decides to make their presence there permanent.

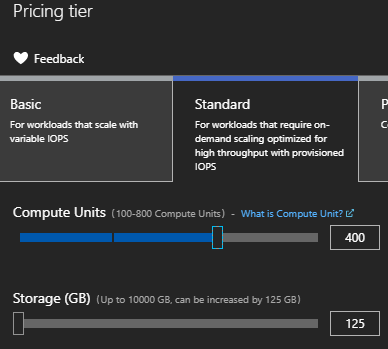


All the applications will use at least a Standard plan or above because it enables essential features to the company such as custom domains. It allows for up to 10 instances to be created and auto scaled (out or in) based on metrics such as server load, memory usage, or even on a set schedule (Luijbregts, 2017). On the standard plan, a daily backup of the configuration and server data is done, which is important to ensure no data is lost. Also, only on the Standard plan or above can deployment slots be provisioned. For the CI/CD workflow employed, this is needed. The deployment slots also allow the application to be rolled back to the last-functional slot in event that a security breach or bug is discovered in a build already rolled out to production.

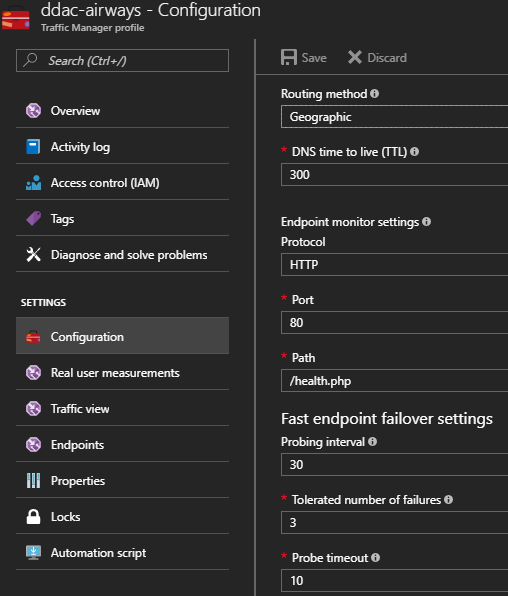
Furthermore, the standard plan is the minimum plan on which the Traffic Manager is supported. For UI airlines’ customers to access the site from the same domain whilst still accessing the nearest server to their region, this is critical. For scaling out, the SEA region will used a fixed number of 3 application instances. This was decided on to save costs as more instances require higher monthly payments. The South Central US region which is the UI Airlines’ main target will use a default of 3 application instances with auto scaling enabled based on the server load.



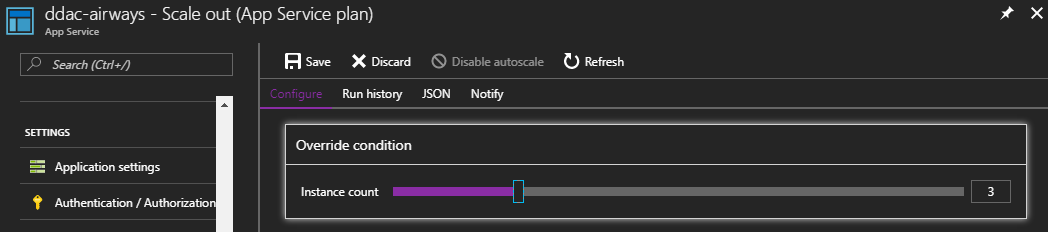
An additional instance will be added whenever the server CPU utilization increases above 80% and an instance will be removed when it drops below 40% In the future when business needs expands, the application service can be scaled to use the premium or isolated plans instead which provides even more features such as higher backup frequencies and better hardware such as SSDs (Lin, 2016).

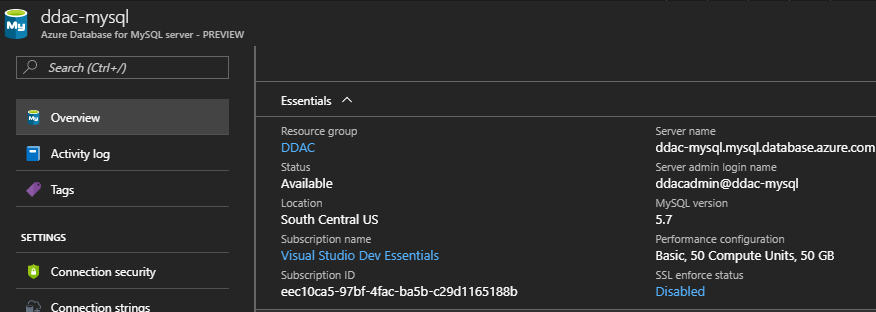
The MySQL server will be using a standard level as well. It will use 400 compute units as 100 compute units equate to one core (Hao, et al., 2017). 4 cores should be sufficient for serving two regions of traffic. It will also have 125GB of storage as the database designed uses a very minimal amount of storage. When requirements do change and the number of users increase, this can be increased.

### Reliability & Performance

To ensure reliability of the web application, several measures were taken. Firstly, the application contains a health endpoint as described in section 3.4. This health endpoint allows the traffic manager to monitor the health of each web application instance. If and when an instance fails (doesn’t respond with a HTTP code 200), the traffic manager will redirect traffic to other instances instead.

There will also be two web applications deployed in separate geographic locations – South central US and South East Asia. This allows traffic in those regions to be directed to their nearest location instead of requesting service from a server in another region. This ensures low response times and also means there is a backup server available if any one region’s servers are down, for example, if the SEA server is down, the traffic manager redirects traffic to the South Central US server instead.

Furthermore, each region’s web application will run 3 instances for higher reliability. At S1, each application is given up to 10 instances. Each web app service contains an integrated load balancer which distributes traffic among all the instances (Microsoft, 2017). This is to increase both maximum throughput as well as increase reliability. Azure also allows the option to scale out these instances automatically based on user set metrics such as on a schedule or by response time (Cavale, et al., 2017). The UI Airlines Application will use a default of 3 instances for each region and auto scale based on the current load. On a standard plan, the server configurations and applications are also backed up daily to ensure no data is lost.



The MySQL server instance is also optimized internally using proper indexing as suggested by database tuning advisors. For now, there is only one instance of the server, kept in the South Central US region. In the future, geo-replication can be added to further increase data integrity and decrease the chance of massive data loss. These steps allow for the application to continue operation and for data to be kept secure even in an event of disaster.

## Testing

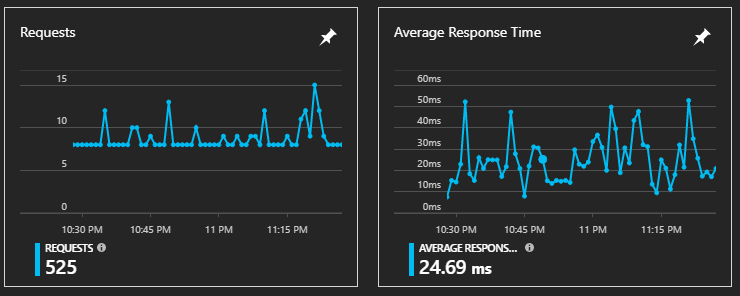
### Unit Testing

The table below contains the test plan created and the results for the unit tests. These tests are meant to test the individual components in the application for functionality. As with all development projects, tests will identify bugs in the system that need fixing. Since this is the latest version with all the fixes applied already, the results of all the tests here are passed. Note that all the tests described here are done locally unless otherwise stated and can be performed on the deployed Azure version by changing instances of “localhost” to the hosted domain name.

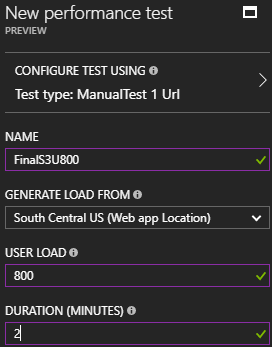
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Title | Description | Expected Result | Actual Result |
| **General Site Tests** | | | | |
| GS-1 | 404 Not Found | Test 404 page functionality   1. Request the URL “localhost/spacex” | Redirected to 404 Page | As Expected |
| GS-2 | 403 Forbidden | Test 403 page functionality   1. Do not login 2. Request the URL “localhost/member” | Redirected to 403 Page | As Expected |
| GS-3 | Homepage redirect | Redirect logged in members to their homepage instead of public homepage   1. Login as member 2. Request the URL “localhost/home” | Redirected to Member page | As Expected |
| GS-4 | web.config conversion | Check that web.config rules are translated correctly   1. Deploy application on Azure with converted web.config 2. Request the URL “localhost/login?username=asd&password=123” | Login page loads with invalid username or password error | As Expected |
| **Components Tests** | | | | |
| CT-1 | Navbar Public | Test loaded navbar is public   1. Do not login 2. Request the URL “localhost” | Navbar contains Login button and Signup button | As Expected |
| CT-2 | Navbar Home | Test home link on navbar   1. Do not login 2. Request the URL “localhost/login” 3. Click on “Home” in navbar | Home page loads | As Expected |
| CT-3 | Navbar Login | Test login link on navbar   1. Do not login 2. Request the URL “localhost/home” 3. Click on “Log In” in navbar | Login page loads | As Expected |
| CT-4 | Navbar Signup | Test signup link on navbar   1. Do not login 2. Request the URL “localhost/home” 3. Click on “Sign Up” in navbar | Signup page loads | As Expected |
| CT-5 | Navbar Member | Test loaded navbar is member   1. Login as member | Navbar contains Home, Search, Full name and Logout button. | As Expected |
| CT-6 | Navbar  Member Home | Test member home link on navbar   1. Login as member 2. Request the URL “localhost/search” 3. Click on “Home” in navbar | Member homepage loads | As Expected |
| CT-7 | Navbar  Member Search | Test member search link on navbar   1. Login as member 2. Click on “Search” in navbar | Member search page loads | As Expected |
| CT-8 | Navbar  Logout | Test logout link on navbar   1. Login as member 2. Click on “Log Out” in navbar | Public Homepage loads | As Expected |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Title | Description | Expected Result | Actual Result |
| **Public Homepage Tests** | | | | |
| HP-1 | Start Flying Button | Test start flying button links to signup page   1. Request the URL “localhost” 2. Click on “Start Flying” | Signup Page loads | As Expected |
| **Signup Page Tests** | | | | |
| SU-1 | Input Validation | Test Input Validation   1. Request the URL “localhost/signup” 2. Click on “Sign Up” | All fields highlighted in red, 4 error messages shown | As Expected |
| SU-2 | Signup | Test Signing up   1. Request the URL “localhost/signup” 2. Create user with username “bob” | Page redirected to member page with fullname displayed | As Expected |
| SU-3 | Existing user account validation | Test Input Validation   1. Request the URL “localhost/signup” 2. Create user with username “bob” 3. Logout 4. Create user with username “bob” again | Error message shown informing username already taken | As Expected |
| **Login Page Tests** | | | | |
| LI-1 | Input Validation | Test Input Validation   1. Request the URL “localhost/login” 2. Click on “Log In” | All fields highlighted in red, 2 error messages shown | As Expected |
| LI-2 | Invalid credentials | Test Invalid Credentials   1. Request the URL “localhost/login” 2. Enter incorrect credentials 3. Click on “Log In” | Error message shown invalid username or password | As Expected |
| LI-3 | Login | Test login   1. Request the URL “localhost/login” 2. Enter correct credentials 3. Click on “Log In” | Page redirected to Member home page | As Expected |
| **Member Home Page Tests** | | | | |
| MH-1 | View No Bookings | Test no bookings view   1. Login as member with no bookings placed | Upcoming flights show no bookings message | As Expected |
| MH-2 | View Bookings | Test bookings view   1. Login as member with 2 bookings placed | Upcoming flights show 2 rows of booking details | As Expected |
| MH-3 | Search Flights | Test flight search works   1. Login as member 2. Select Origin as “AKL” 3. Select Destination as “HND” 4. Click “Search” | 2 flight results displayed | As Expected |
| **Search Page Tests** | | | | |
| SF-1 | Search Flights | Test flight search works   1. Login as member 2. Request the URL “localhost/search” 3. Select Origin as “AKL” 4. Select Destination as “HND” 5. Click “Search” | 2 flight results displayed | As Expected |
| SF-2 | Empty Search Results | Test empty results   1. Login as member 2. Request the URL “localhost/search?origin=ASD” | Results table shown with no flights message | As Expected |
| SF-3 | Book Flights | Test flight booking redirect works   1. Login as member 2. Request the URL “localhost/search” 3. Select Origin as “AKL” 4. Click “Search” 5. Click on the first flight | Redirected to booking screen for Flight 1 | As Expected |
| **Booking Page** | | | | |
| B-1 | Data Loading | Test flight booking loads correct data   1. Login as member 2. Request the URL “localhost/booking?flight=1” | Confirm booking panel shows correct info for flight ID 1 | As Expected |
| B-2 | Sticky Panel | Verify confirm booking panel follows page scroll   1. Login as member 2. Request the URL “localhost/booking?flight=1” 3. Scroll page to bottom | Confirm booking panel follows scroll down to bottom | As Expected |
| B-3 | Seat Picker | Test occupied seats not selectable  1. Login as member  2. Request the URL “localhost/booking?flight=1”  3. Place booking for Seat 1-A | Redirected to member home page, new booking shown | As Expected |
| B-4 | Seat Picker Occupied | Test occupied seats not selectable   1. Login as member 2. Request the URL “localhost/booking?flight=1” 3. Place booking for Seat 1-A 4. Request the URL “localhost/booking?flight=1” 5. Click on Seat 1-A | Seat 1-A not selectable on Seat Picker | As Expected |
| B-5 | Invalid Flight Redirect | Test occupied seats not selectable  1. Login as member  2. Request the URL “localhost/booking?flight=1337” | Redirected back to member home page | As Expected |

### Performance Testing



The graph above shows the response times for the average usage of the site. The average response time of the site after 525 requests is 24.69ms. Which is well within the requirements of 100ms.

The performance tests will be conducted with the help of the functionality provided as part of an Azure web app. The tests will be conducted on the main web app resource and will test a user load of 200 to 800 for two minutes with each test increasing in user load by 200 user increments. The results that will be gathered include response time and failed requests. These tests will be conducted on three app service plans – S1, S2 and S3. The response times recorded for the tests are the time taken for the entire web page to load, and will be recorded in seconds. The table below shows the data gathered from the tests described.

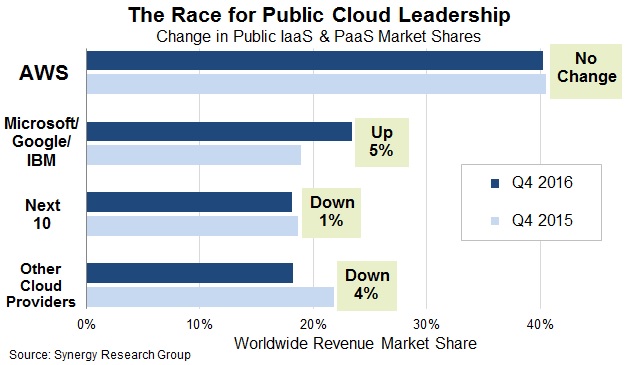
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Concurrent Users  App  Service Plan | 200 | 400 | 600 | 800 |
| S1 | 0.59 Sec  0 Failed | 1.74 Sec  0 Failed | 6.75 Sec  1 Failed | 10.83 Sec  309 Failed |
| S2 | 0.25 Sec  0 Failed | 1.42 Sec  0 Failed | 3.63 Sec  2 Failed | 8.09 Sec  251 Failed |
| S3 | 0.27 Sec  0 Failed | 10.6 Sec  0 Failed | 3.93 Sec  0 Failed | 7.18 Sec  215 Failed |

### Analysis

From the data gained in the tests above, it can be deduced that, as expected, the better service plan results in better handling of higher concurrent users. From the results gathered, it can be seen that S2 shows a significant increase in performance over S1. Whilst S3 doesn’t show as significant an increase compared to S2 and even performed worse on one of the tests (600 concurrent users). The maximum acceptable threshold for page load times on this application is 4 seconds. Which means S1 can reliably handle 400 concurrent users whilst S2 and S3 both can handle 600 concurrent users. Considering the costs of increasing to a higher tier, which is roughly double the price every tier increase, it can be concluded that S2 is the best valued tier for use in this project at an estimated RM662.16 per month.

## Managed Databases

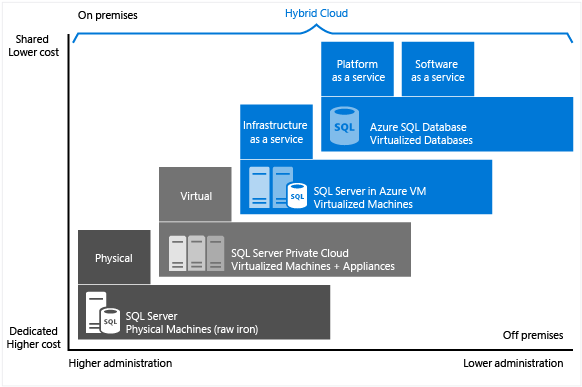
With the increasing popularity of cloud and managed services available online, more and more services previously only available through a locally provisioned server are becoming available at the click of a button. The concept of platform as a service (PaaS) is becoming a valuable tool to a market that is constantly trying to shave the time between development and deploying off.



As larger companies such as Amazon, Microsoft, Google and IBM (Synergy Research Group, 2017) are leading the market share in cloud computing, the services offered are becoming more and more accessible with a wide variety of features otherwise difficult or expensive to obtain on a locally provisioned server.

There are many advantages to using a PaaS instead of a local server. Firstly, the speed of provisioning allows for much more agility in innovation (Engine Yard, 2017). A PaaS allows a developer to create 10 servers using a single script, then remove it after 5 minutes of testing. It also allows for costs of operation to be reduced. With features such as auto-scaling (Cavale, et al., 2017) available for Microsoft Azure services, costs can be kept down by ensuring only the required resources are provisioned when needed.

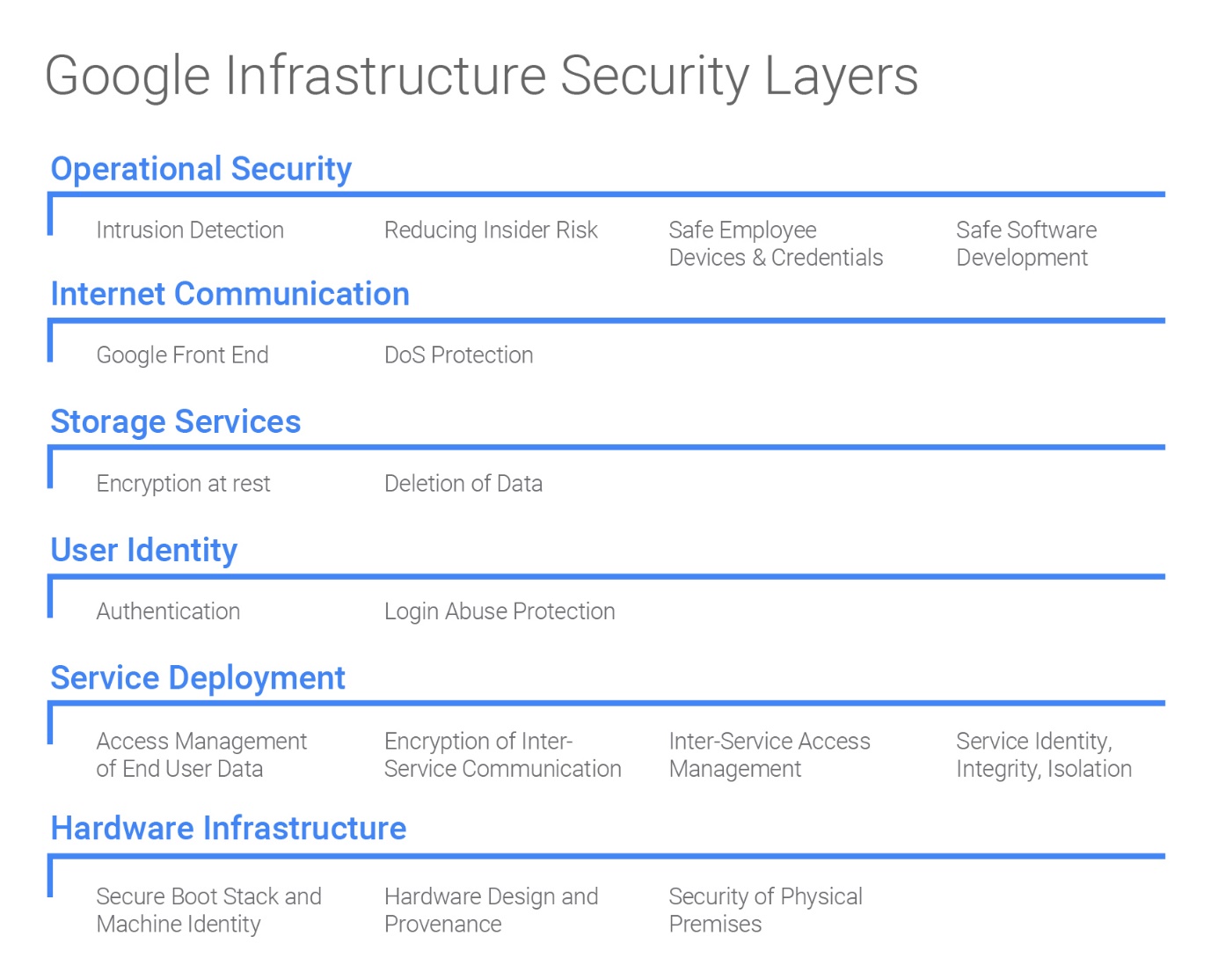
Furthermore, if the user decides to go with one of the top 4 companies in cloud computing, they are likely to have near perfect uptime as they all offer at least 99% uptime in their service level agreements (Amazon Web Services, 2013). That level of availability is further increased if one provisions more services as failover services. Such high availability is hard to achieve in a self-designed and deployed server without the necessary expertise.



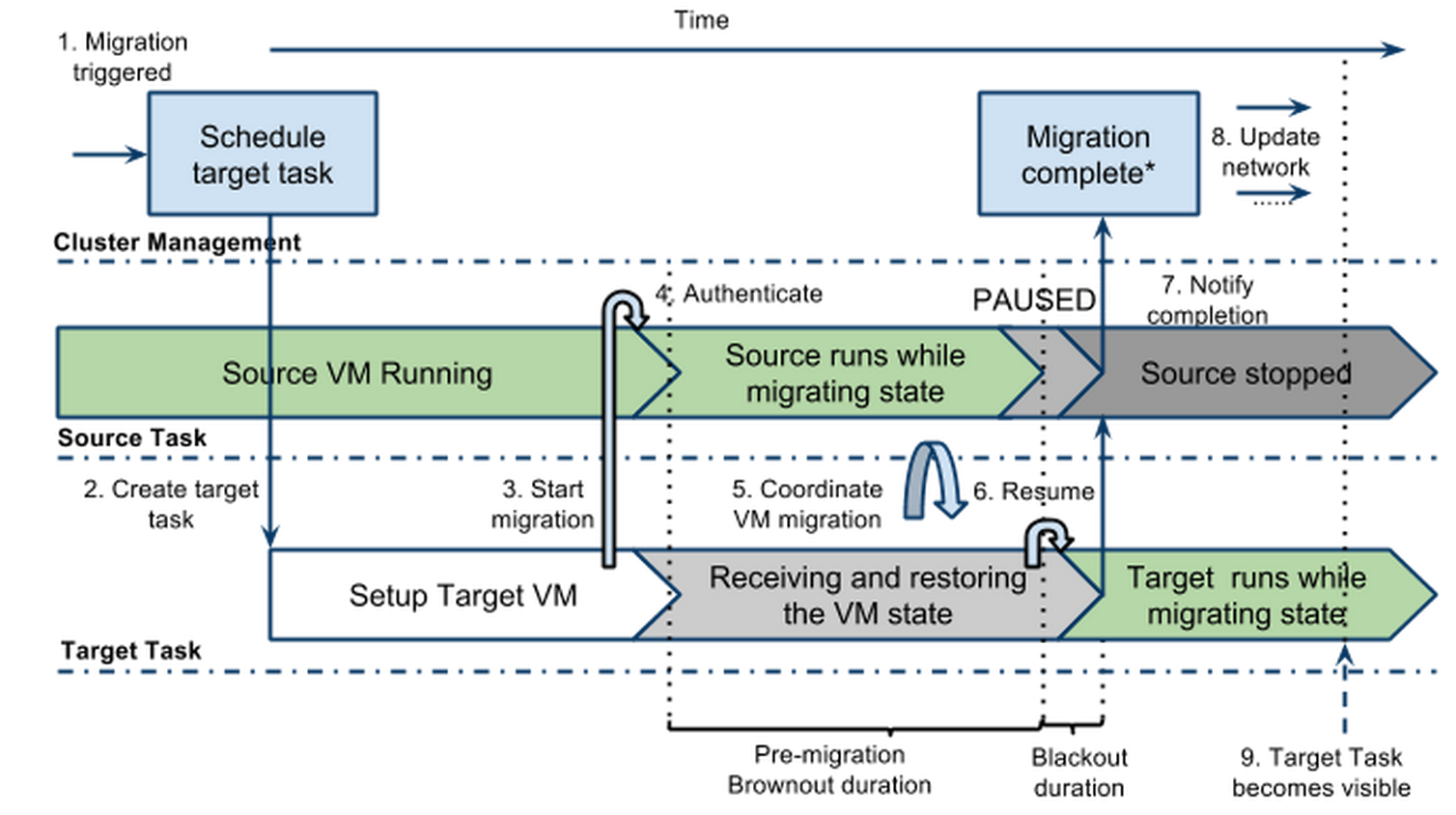
One of the most interesting PaaS services provided by many cloud providers is the managed database service. Amazon has it in the form of the Amazon RDS whilst Microsoft offers their Azure SQL Database service. The PaaS model that these services are built on allows for lower costs and lower administration yet still enables advanced features (Rabeler, et al., 2017). The one trade-off is that the data is stored on a third-party’s servers instead of your own and the lack of access to the raw machine.

As everything on a PaaS lies on many layers of virtualization, the underlying hardware can be swapped without affecting the top layer services, ensuring no downtime between hardware updates or failures. This allows the infrastructure to be continuously updated. Also, since the service is delivered through a web application, new feature releases don’t require a server to be shut down, then manually updated to the latest version like most server software would.

Some of the features the Azure managed SQL database provides straight out of the box that a local SQL server would need setup for are encryption, firewalls, monitoring and automatic backups. Even advanced features such as geo-replication (Nosov, 2016), automatic tuning and threat detection are available right out of the box. The Azure SQL service even allows for alerts to be set based on user set conditions such as a spike in response time. Amazon’s offering supports 6 types of databases including MySQL, Oracle and SQL Server right out of the box. They even offer the functionality of database snapshots, allowing for rollbacks of data if any critical errors occur in the database (Amazon Web Services, 2017).



The Google Cloud Platform also has its own managed database services named Cloud SQL which supports MySQL and PostgreSQL. The service offers 6 layers of infrastructure security to ensure user data is protected with the highest possible security (Google, 2017). It also employs live migration technology so that infrastructure can be serviced without any application downtime – Transparent Maintenance.



The diagram above shows the Transparent Migration process in action. VMs or PaaS services are notified that they will be evicted from the host machine. The first stage is pre-migration which sends the state of the VM to the target. Then the next stage is a very brief moment of blackout when the VM is not running anywhere but its state is paused and sent to the target infrastructure. After that, the post-migration phase is where the VM is executed on the target, continuing from whatever state the VM was in previously (Google, 2015). This process allows for seamless migration without any downtime.

# Conclusion

In conclusion, the development of a cloud based application for Ukraine International Airlines has proven to be a challenging albeit informative one. The process of not just building but also deploying an application fit for the cloud and one that utilizes all the cloud has to offer has provided new, relevant skills which will be invaluable to the modern informational technology industry. Concepts such as cloud design patterns and cloud resource provisioning as well as the considerations needed to design a cloud architecture have become much clearer. A deeper understanding of the underlying technology of cloud computing through Azure has also been obtained. Hence, this project has been a very productive one and I look forward to more like it in the future.

# References

Amazon Web Services, 2013. *Amazon EC2 Service Level Agreement.* [Online]   
Available at: https://aws.amazon.com/ec2/sla/  
[Accessed 22 September 2017].

Amazon Web Services, 2017. *Amazon RDS Product Details.* [Online]   
Available at: https://aws.amazon.com/rds/details/  
[Accessed 22 September 2017].

AnyChart.com, 2017. *AnyChart.* [Online]   
Available at: https://www.anychart.com/  
[Accessed 21 September 2017].

Apache Friends, 2017. *XAMPP Apache + MariaDB + PHP + Perl.* [Online]   
Available at: https://www.apachefriends.org/  
[Accessed 22 September 2017].

Cavale, A., Boucher, R. J. & Kemnetz, J., 2017. *Scale instance count manually or automatically.* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/insights-how-to-scale  
[Accessed 22 September 2017].

Engine Yard, 2017. *Top 10 Advantages of Platform as a Service.* [Online]   
Available at: http://www.engineyard.com/whitepapers/top-10-advantages-of-platform-as-a-service  
[Accessed 22 September 2017].

generatedata.com, 2017. *generatedata.com.* [Online]   
Available at: https://www.generatedata.com/  
[Accessed 16 September 2017].

GitHub Inc., 2017. *GitHub.* [Online]   
Available at: https://github.com/  
[Accessed 21 September 2017].

Google, 2015. *Google Compute Engine uses Live Migration technology to service infrastructure without application downtime.* [Online]   
Available at: https://cloudplatform.googleblog.com/2015/03/Google-Compute-Engine-uses-Live-Migration-technology-to-service-infrastructure-without-application-downtime.html  
[Accessed 22 September 2017].

Google, 2017. *Google Infrastructure Security Design Overview.* [Online]   
Available at: https://cloud.google.com/security/security-design/  
[Accessed 22 September 2017].

Hao, L., H, J. & Hubbard, J., 2017. *Explaining Compute Units in Azure Database for MySQL.* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/mysql/concepts-compute-unit-and-storage  
[Accessed 22 September 2017].

Lin, C., 2016. *Scale up an app in Azure.* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/app-service/web-sites-scale  
[Accessed 22 September 2017].

Luijbregts, B., 2017. *How to Autoscale Azure App Services & Cloud Services.* [Online]   
Available at: https://stackify.com/autoscale-azure-app-services-cloud-services/  
[Accessed 22 September 2017].

Microsoft, 2017. *App Service plans.* [Online]   
Available at: https://azure.microsoft.com/en-us/pricing/details/app-service/plans/  
[Accessed 22 September 2017].

Microsoft, 2017. *Azure Database for MySQL.* [Online]   
Available at: https://azure.microsoft.com/en-us/services/mysql/  
[Accessed 22 September 2017].

Microsoft, 2017. *Microsoft Azure Documentation.* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/  
[Accessed 21 September 2017].

Narumoto, M., Bennage, C. & Wasson, M., 2017. *Health Endpoint Monitoring Pattern.* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/architecture/patterns/health-endpoint-monitoring  
[Accessed 21 September 2017].

Nosov, A. (., 2016. *Azure SQL Database now supports powerful geo-replication features for all service tiers.* [Online]   
Available at: https://azure.microsoft.com/en-us/blog/azure-sql-database-now-supports-powerful-geo-replication-features-on-all-service-tiers/  
[Accessed 22 September 2017].

Rabeler, C., Squillace, R. & Kunz, A., 2017. *Choose a cloud SQL Server option: Azure SQL (PaaS) Database or SQL Server on Azure VMs (IaaS).* [Online]   
Available at: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-paas-vs-sql-server-iaas  
[Accessed 22 September 2017].

Sangapu, M., 2015. *Converting Apache .htaccess rules to IIS web.config using IIS Manager for Azure Websites.* [Online]   
Available at: https://blogs.msdn.microsoft.com/azureossds/2015/04/23/converting-apache-htaccess-rules-to-iis-web-config-using-iis-manager-for-azure-websites/  
[Accessed 22 September 2017].

Semantic UI, 2017. *Semantic UI.* [Online]   
Available at: https://semantic-ui.com/  
[Accessed 21 September 2017].

Synergy Research Group, 2017. *Microsoft, Google and IBM Public Cloud Surge is at Expense of Smaller Providers.* [Online]   
Available at: https://www.srgresearch.com/articles/microsoft-google-and-ibm-charge-public-cloud-expense-smaller-providers  
[Accessed 22 September 2017].

The PHP Group, 2017. *PHP Manual.* [Online]   
Available at: http://php.net/docs.php  
[Accessed 21 September 2017].