THE UNIVERSITY OF
MELBOURNE
Melbourne School of Engineering

Final Report
2017

Student Name:	Verra Mukty
Student Number:	738024
Supervisor:	Professor Rajkumar Buyya
Credit:	25 Points
Type of Project:	Software Development Project
Subject Code:	COMP90019 – Distributed Computing Project
Project Title:	Sleep Apnea analysis, using Pulse Oximeter data, on Microsoft Azure.

Contents

1. Project Background

This project is an extension of an internship project that I have done on MNSI (Melbourne Networked Society Institute) University of Melbourne. The internship project itself was to develop an Android application that can read data from Pulse Oximeter device, and store the data on Nectar cloud server. The system architecture from the internship project can be seen on Image 1 below.

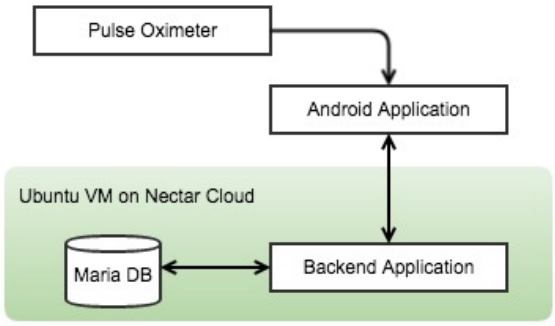

Image 1. Old System Architecture

Pulse oximetry is a technology used to measure the oxygen level in our blood and our heart rate. A finger pulse oximeter functions by shining light through your finger. The sensors detect how much oxygen is in our blood based on the way the light passes through our finger as illustrated on Image 2 below [1].

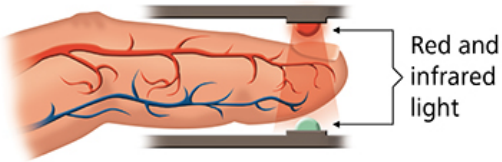

Image 2. Pulse Oximetry Technology [12]

The Pulse Oximeter device being used in the project is Nonin 3230 as can be seen on Image 3 below.

Image 3. Nonin 3230 Pulse Oximeter [13]

Pulse Oximeter is communicating with Android application through Bluetooth connection. The Android application is providing a listener method that will be called every time pulse oximeter data is changed as can be seen on Image 4 below. When this listener is called, Android application will show the updated data on application screen by calling setOximeterData method.

```
@Override
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic
characteristic) {
    setOximeterData(characteristic);
}
```

Image 4. Pulse Oximeter data listener on Android Application

Backend application is implemented using Java RESTful Web Services. Android application communicates with the Backend application by calling web services URL using POST method. Backend application respond the request using JSON message format. During user registration, user's sensitive data such as token key is protected using Asymmetric encryption (RSA). After the token key is sent securely, the next message exchanging between Android application and Backend application is encrypted using Symmetric encryption (AES) that incorporate a combination of token key and salt. The Backend application store user profile information and pulse oximeter data (oxygen level and heart rate) on a Database system Maria DB.

2. Project Goal

This distributed computing project is intended to utilise the extracted pulse oximeter data for health analysis. One of the health analysis that can be done is Sleep Apnea analysis. Another goal in this project is to create a scalable backend system that can handle growing number of request from users.

2.1. Sleep Apnea Analysis

Sleep Apnea is a disorder where someone have one or more pauses in breathing or shallow breaths while they sleep. The common symptoms are snoring, waking unrefreshed, daytime tiredness, and waking during the night choking or gasping for air. Untreated Sleep Apnea can increase the risk of high blood pressure, heart attack, stroke, obesity, and diabetes [2].

The current state of the art method to diagnose Sleep Apnea is by conducting sleep studies. The most common sleep study for diagnosing Sleep Apnea is Polysomnogram (PSG) as can be seen on Image 5. This study records brain activity, eye movements, heart rate, and blood pressure. It also records the amount of oxygen in patient's blood, air movement through nose while they breathe, snoring, and chest movements. The chest movements show whether patient making an effort to breathe [3].

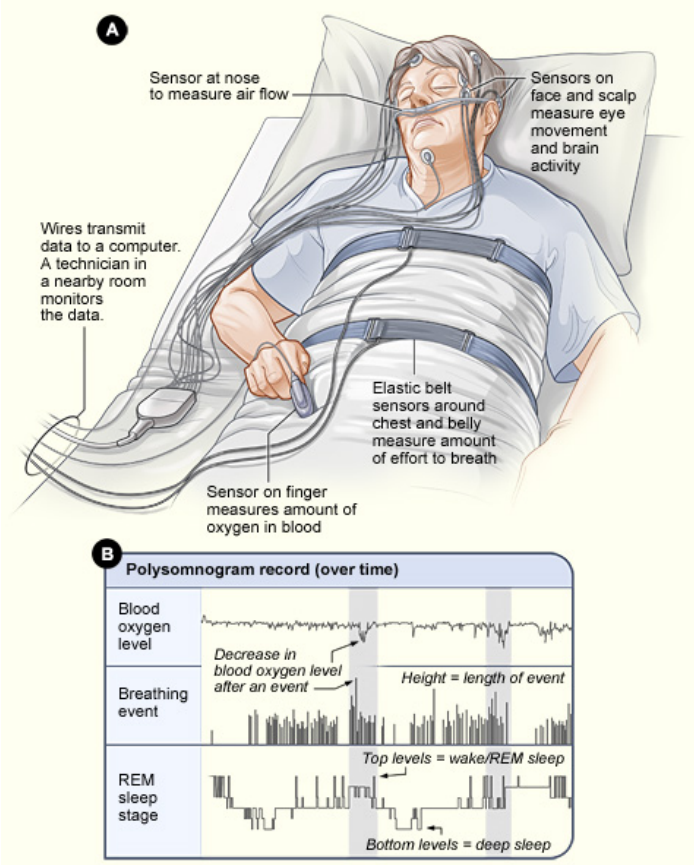

Image 5. Polysomnogram sleep study [14]

PSGs often are done at sleep centers or sleep labs. Patient will have sensors attached to patient's scalp, face, chest, limbs, and a finger. The staff at the sleep center will use the sensors to check on patient throughout the night. A sleep specialist will review the results of PSG to see whether patient have sleep apnea and how severe it is. Doctor will use the results to plan patient's treatment.

While PSG is the most widely accepted diagnostic test for the sleep apnea, the expensive and time-consuming nature of PSG prompts many sleep centres to perform an initial screening test in order to reduce the number of PSGs needed. One of the method that has been explored to do the initial test is by using Pulse Oximeter data.

In this project, we utilise an existing open source code that created by researcher in this area. The research was titled "Combined index of heart rate variability and oximetry in screening for the sleep apnoea/hypopnoea syndrome" by Ben Raymond, R. M Cayton, M. J. Chappell (2003) [4]. In this research, they compared the accuracy of pulse oximeter data to diagnose Sleep Apnea with other method such as using electroencephalogram (EEG), and PSG.

2.2. Scalable Backend System

To implement a scalable backend system, we utilise both the Horizontal Scaling (Scale Out) and Vertical Scaling (Scale Up) method. To Scale Out means to add more nodes to a system, such as adding a new computer to a distributed software application. Meanwhile, to scale up means to add resources to a single node in a system, typically involving the addition of CPUs or memory to a single computer. [5]. In this project Scale Out method will be applied on Backend application server, and Scale Up will be applied on Database server.

3. System Design

To achieve the project goals, we construct a system architecture that can be seen on Image 6 below. The system consists of 5 components, those are:

1. Stress Test application
2. Android application
3. Backend application (Java REST Web Service)
4. Database system
5. Sleep Apnea application

The Load Balancer is part of Backend application VM scale set resource group. It is automatically created when we create the scale set. The Load Balancer will distribute request to VM scale set using Round-robin algorithm. VM scale set will be explained in more detail on section 3.1.3. Azure Scale Set.

Backend application, Database system, and Sleep Apnea application are deployed on Azure Cloud servers. The detail about Azure and each system components will be discussed in a subsequent subsection.

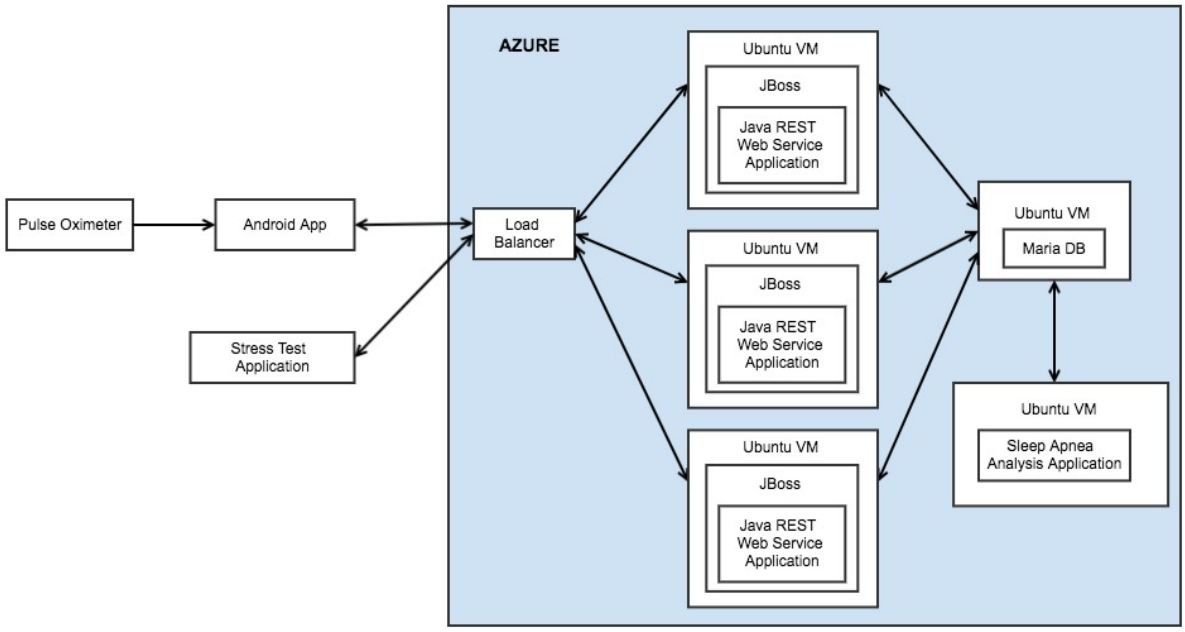

Image 6. System Architecture

3.1. Azure Cloud

Microsoft Azure is a cloud computing service created by Microsoft for building, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). It also supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems [6]. The feature offered by Azure for each cloud service model can be seen on Image 7 below.

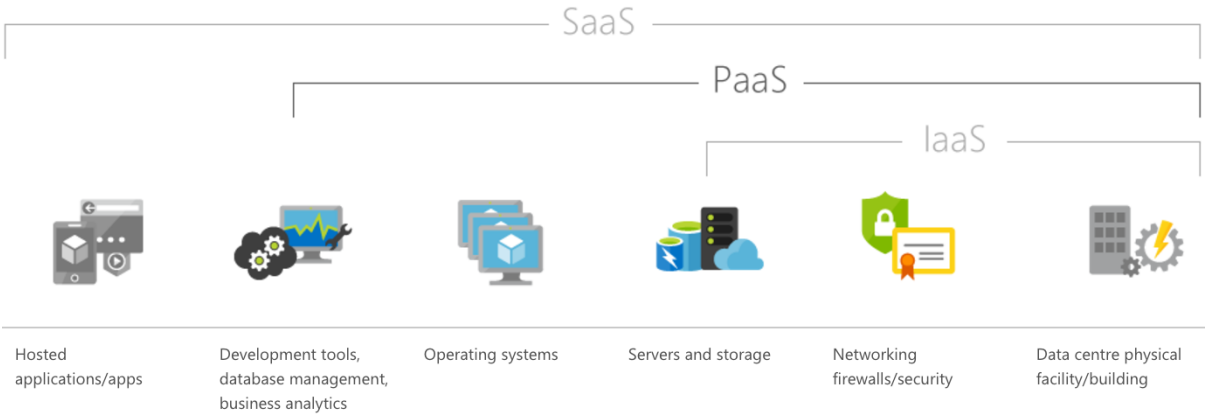

Image 7. Azure Cloud Service Model Features [15]

This project is using the PaaS service model. In Azure we could choose Ubuntu VM with various pre-installed software such JBoss, Apache, various DBMS, etc. The VM type being used for each system components can be seen on Table 1 below.

System Component	Total VM	VM Type
Backend application (Java REST Web Service)	1 can be scaled out to 3	OS: Ubuntu 14.04 Pre-installed software: • Java • JBoss wildfly 9.0.2
Database system	1	OS: Ubuntu 14.04 Pre-installed software: • Maria DB
Sleep Apnea Analysis application	1	OS: Ubuntu 14.04 Pre-installed software: • Java

Table 1. VM Types used on Project

3.1.1. PowerShell Scripting

Microsoft Azure provides an online portal to manage resources such as VM, Disk, Network Security Group, etc. The user interface itself is following the Windows operating system style as can be seen on Image 8 below.

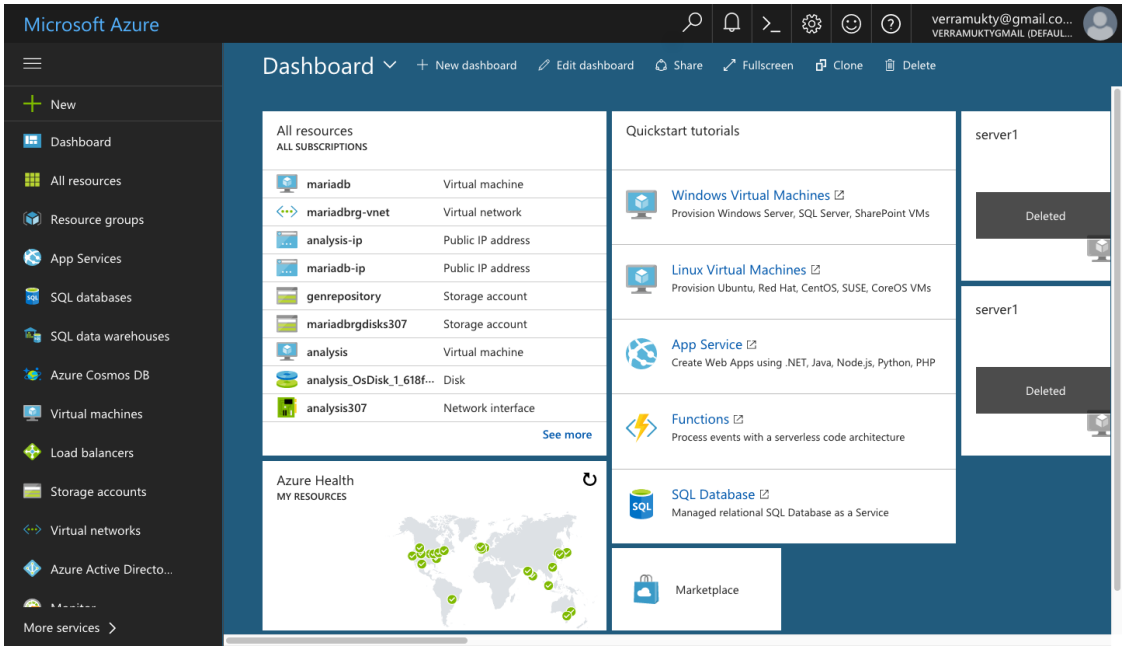

Image 8. Azure Portal

The Azure portal provides most of functionality that user needs to manage their cloud resources. However, during the development of this project, there are some functionality that still disabled on the portal such as creating inbound rule on scale set, or required manual Linux command such as creating VM image.

Another drawback of Azure portal is, user need to do quite a lot of steps to create resource. This is not the most effective way to manage resource because user have to manually do those steps. The solution for this problem is to use automation script. Currently the recommended automation script by Microsoft are:

- PowerShell
- CLI
- Visual Studio Code

In this project, we use PowerShell script. CLI is targeted for cross-platform command line interface so it can be used from anywhere such as Linux or Mac [7]. However, Microsoft only quite recently support it compared to PowerShell. CLI was announced on September 2016 [8], while PowerShell has been supported from the beginning of Azure. Thus, there is a possibility that there are some functionalities that's still not available on CLI while its available on PowerShell. As for the Visual Studio Code, this is a recommended tool that is cross platform, has great integration with source control, and fast. However, we only found about it after spend a lot of effort creating PowerShell script so we don't use it for this project.

Azure PowerShell list of instructions can be accessed on its online documentation that's available at URL:

https://docs.microsoft.com/en-us/powershell/module/azurerm.profile/add-azurermaccount?view=azurermps-4.0.0

The user interface itself is following the Javadoc model where we could see each method parameters, parameter description, output, and instruction usage example. The sample of PowerShell instruction documentation can be seen on Image 9 below.

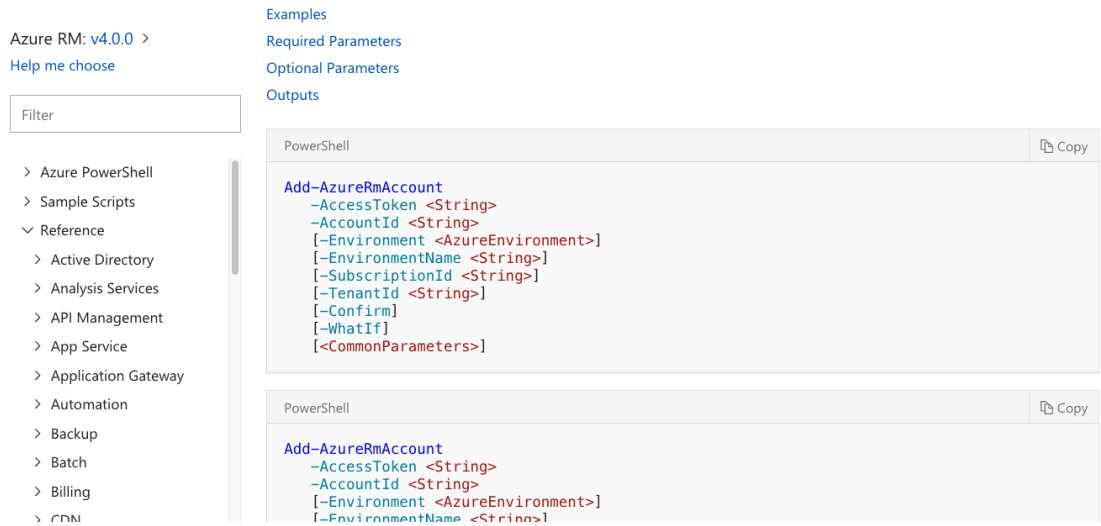

Image 9. PowerShell Online Documentation

The list of PowerShell scripts created to configure the Azure cloud can be seen on Table 2 below.

Script Name	Description
/mariadb/deploy.ps1	Deploy database system VM
/mariadb/configure.ps1	Configure database system: - Create inbound rule to enable access port 3306 from remote server - Create myhealthrecord DB and its tables - Create user to access the myhealthrecord DB from remote server
/wildfly/deploy.ps1	Deploy JBoss wildfly VM
/wildfly/configure.ps1	Configure Backend application: - Create inbound rule to enable access port 8080 from public address - Upload database driver - Upload backend application war file - Configure database connection

/scaleset/prepare_repository.ps1	Create repository resource group to store required files to create scale set such as: - Custom VM image - vmss.json to accommodate scale set deployment using custom VM image - lb.json to accommodate creating inbound rule to enable access port 8080 from public
/scaleset/create_wildfly_image.ps1	Automate the process of capturing image of wildfly VM such as: - Deprovsion VM - Stop and generalized VM - Capture VM image - Remove old wildfly resource group Notes: The purpose of VM deprovision is to remove sensitive information, so the VM image is suitable for redistribution. It will remove information such as provisioned user account on the VM. The VM generalization process is to prepare system before image capturing. It will clean the VM Operating System from machine and user's specific settings such as machine name, SID, administrator password, etc. This process is irreversible, which means once it is done, the VM cannot be used or functioning again.
/scaleset/deploy.ps1	Deploy VM scale set using wildfly custom image, and define the scale out rules. The rules are: - If CPU load > 75% for 5 minutes, add 1 VM instance. - If CPU load < 25% for 5 minutes, remove 1 VM instance. - The maximum total VM instance is 3
/other/upgrade_machine.ps1	Scale up DB VM from using 1 core CPU and 0.75 GB memory into 2 cores CPU and 14 GB memory.
/analysis/deploy.ps1	Deploy analysis application VM
/analysis/configure.ps1	Configure analysis application VM: - Upload MATLAB runtime compiler to server - Unzip the installer the uploaded installer

	- Install MATLAB runtime compiler on server - Open access for log folder so the Apnea Analysis application can write its log file - Upload Apnea Analysis application jar and its library to /usr/local - Upload apnea_analysis.sh to home dir

Table 2. PowerShell Script List

3.1.2. Azure Resource Group

To deploy VM, currently Azure provides 2 different deployment model, the classical deployment model, and resource manager. Azure originally provided only the classic deployment model. In this model, each resource existed independently and we cannot group related resources together. We have to manually track which resources made up a solution or application, and remember to manage them in a coordinated approach. To deploy a solution, we had to either create each resource individually through the classic portal or create a script that deployed all the resources in the correct order. To delete a solution, we had to delete each resource individually [9].

In 2014, Azure introduced Resource Manager, which added the concept of a resource group. Currently Azure portal only support the resource group deployment model. In resource group, we can deploy, manage, and monitor all the services in a solution as a group, rather than handling these services individually. Some of the benefits of this resource group model are:

- We can apply access control to all resources in a resource group, and those policies are automatically applied when new resources are added to the resource group.
- We can repeatedly deploy a solution throughout its lifecycle and have confidence that resources are deployed in a consistent state.
- We can use JSON to define the infrastructure solution. The JSON file is known as a Resource Manager template.
- We can define the dependencies between resources so they are deployed in the correct order.

There are 4 resource groups created in this project. The resource group name and its description can be seen on Table 3 below.

Resource Group Name	Description
mariadbrg	To deploy Database System
scalesetrg	To deploy Backend application (Java REST Web Service)
analysisrg	To deploy Sleep Apnea Analysis application
repositoryrg	To store Backend application VM image and scaleset vmss & load balancer configuration.

Table 3. Resource Groups

The decision to create those resource groups is for the convenience during development process. For example, during the research and trial of scale set deployment, we could freely drop, deploy, and configure the scale set resource group. At the same time, other resource group that host different part of system such as database and analysis application will remain unaffected and guaranteed to be functioning well.

3.1.3. Azure Scale Set

To create a scalable backend application, we need to deploy it in a VM scale set. VM scale set is an Azure resource that can be used to deploy and manage a set of identical VMs. With all VMs configured the same, scale sets are designed to support autoscale, and no pre-provisioning of VMs is required [10].

To implement an autoscale system, we need to define the scaling rules. For example, when system is boot initially, there's only one VM instance run. When CPU load is higher than 75 percent for 5 minutes, then add one VM instance to share the workload. When CPU load is less than 25 percent for 5 minutes, then remove one VM instance because the system workload has been decreasing. We could also configure the number of VM that should be added and removed every time system is scaled out or scaled in. For example, we could choose to add 2 instances rather than 1 to quickly scale out the system.

When a new VM is being added to the scale out, we need to make sure that it has all the components to function properly such as application war is deployed on JBoss deployment folder, database driver and connection setting are properly set up, etc. Azure provide 2 different ways to implement it, those are:

- Use off the shelf VM that available on the Azure Marketplace and use VM extension script to configure the VM. VM extension script is automatically executed when a new VM is created during scale out.

- Use custom image that has been configured and ready to be used as a backend application server.

In this project, we use the second option. The implementation script for this process is explained on Table 2 row "create_wildfly_image.ps1". This solution is chosen because we consider that configuring VM will take some time. If every time a new VM added to scale set needs to be configured first, it will make the VM creation process takes longer times, and slow down the scale out process compared to if use the pre-configured custom image.

3.1.4. Azure Scale Up

Azure scale up is applied to Database system. If we are using traditional DB server (not deployed in Cloud), to migrate DB into bigger machine requires us to do these steps:

1. Back up the DB
2. Stop the entire system from the outer layer (Backend application), propagate to the inner layer (DB system).
3. Run query to get the DB modification between the time DB backup is started and the entire system is shut down.

4. Store the DB backup on new machine
5. Store DB Modification on new machine
6. Start the system from the inner layer to the outer layer.

When we are using scale up feature on Azure, we could simplify those steps into:

1. Stop the entire system from the outer layer (Backend application), propagate to the inner layer (DB system).
2. Run scale up script
3. Start the system from the inner layer to the outer layer.

The scale up feature makes the system migration a lot simpler compared to doing the migration manually.

3.2. Stress Test Application

To simulate the situation of high load system, we need to generate a lot of requests to backend application. The stress test application being used in this project is JMeter. The Apache JMeter™ application is a java open source software that can be used as a load testing tool for analysing and measuring the performance of a variety of services, with a focus on web applications [11].

JMeter can be run in 2 modes, GUI mode and console mode. GUI mode is usually used to create and validate test plan. Once the test plan is confirmed to be run correctly, we use console mode the execute load testing. GUI mode is not recommended to be used for load testing as it can cause resource exhaustion on the computer that executing it (PC is hang). The screenshot of JMeter GUI interface can be seen on Image 10 below.

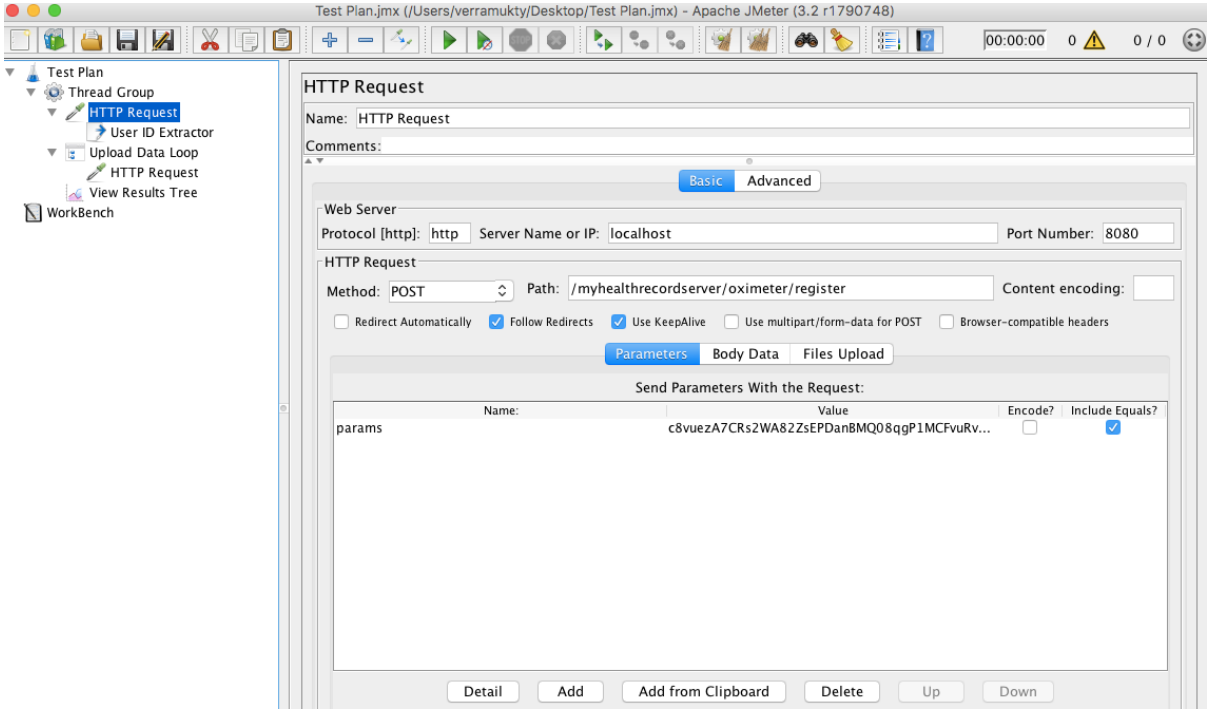

Image 10. JMeter GUI interface

As we can see on the above image, the test plan for this project consists of 1 thread group that consist 2 HTTP requests. The HTTP requests is executed based on its order on the test plan. The first request is to generate user's registration process, and the second request is to generate user's pulse oximeter data upload. The second HTTP request is put inside a loop with setting loop 100.

The second HTTP request depends on the result of the first HTTP request because to upload pulse oximeter data, we need userId information which is given after registration run successfully. To pass the userId information from registration to upload pulse oximeter data, we create userId data extractor. This data extractor could read the response message from registration, and get the userId information using regex matching. The extracted userId can be accessed using variable like "${user_id}".

For each HTTP request, we need to define Web Server address (IP and Port), and the request parameters. The screenshot of HTTP request configuration can be seen on Image 11 below.

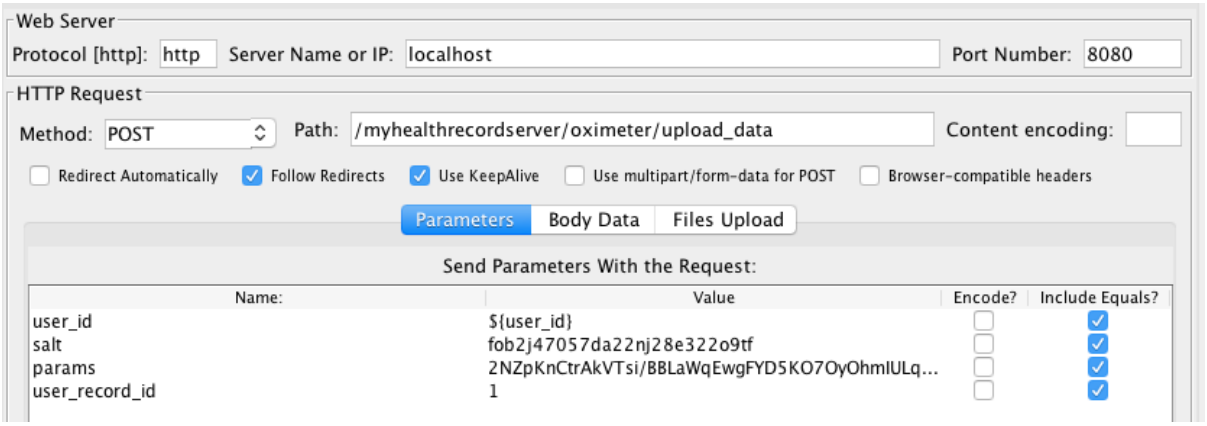

Image 11. JMeter HTTP request parameters setting

The purpose of this test plan design is to create many parallel user registration process, and after the user is registered, generate pulse oximeter data upload repeatedly. To evaluate the backend application performance, we use the response time that recorded from JMeter. The response time is recorded on a target csv file that we define during load testing command execution.

3.3. Android Application

The Android application in this project is developed based on the Android application from internship project. The features enhancement of new Android application can be seen on Table 4 below.

Existing Android App Features	Enhancement Android App Features
Registration	Use as is
Login	Use as is
Logout	Use as is
Scan Oximeter page	Use as is
View Oximeter Data page	Changed
* not available on existing Android app	View Pulse Oximeter Record page
* not available on existing Android app	View Analysis Result page

Table 4. Android Application Feature Enhancement

3.3.1. View Oximeter Data Page

The original view oximeter data page and its new page comparison can be seen on Image 12 below. This page is the landing page shown to user after successfully login. On the original page, user could upload pulse oximeter data by clicking "Upload" button. The data itself is only a single SPO2 and Pulse data that being read when user click the button. This approach is obviously not the best way to record a pulse oximeter data for health analysis because it cannot monitor the fluctuation of pulse oximeter data for certain period. However, this lack of functional quality was done on purpose, because the internship project was only intended to get data transported from Android application to server.

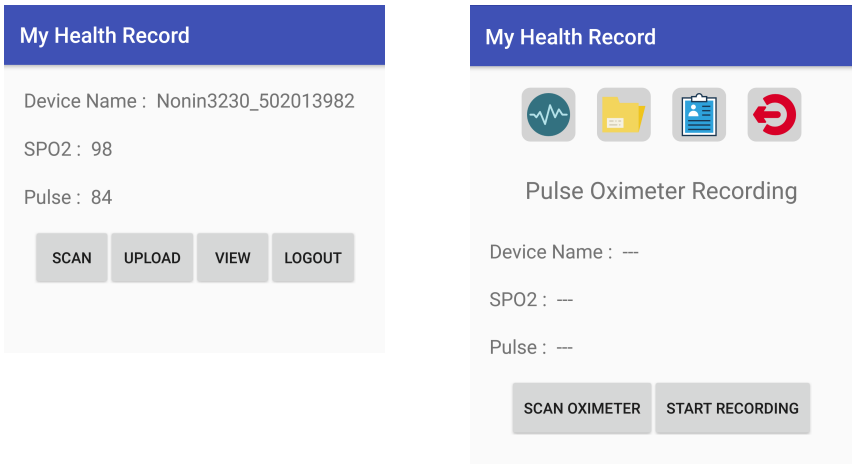

Image 12. Left: Original View Oximeter Data Page, Right: New Page

The new View Oximeter Data page replace the “Upload” button with “Start Recording” button. When user click “Start Recording”, the button will be changed into “Stop Recording”. During this time, application is recording the pulse oximeter data for every 1 second, and store the data on Android SQLite DB until user click the “Stop Recording” button. The period between user click the “Start Recording” and click the “Stop Recording” button is considered as 1 Pulse Oximeter record.

The idea behind this design is, when user is going to sleep, they click the “Start Recording” button, and when they wake up, they click the “Stop Recording” button. As a result, they will 1 pulse oximeter record for 1 sleep period that ready to be uploaded to server and being analysed.

The 1 second recording interval is required because the Apnea Analysis application required the SPO2 data to be recorded on that interval. The 1 second recording interval is achieved by adding Thread.sleep(1000) inside a while loop.

Another modification that being done on this page is adding 4 image buttons on top of the page that act as navigation button. The functionality of those buttons can be seen on Table 5 below.

Buttons	Function
	Navigate to View Oximeter Data page
	Navigate to View Pulse Oximeter Record page
	Navigate to View Analysis Result page
	Logout

Table 5. Navigation Image Buttons

Those image buttons are implemented in one Android Fragment that can be included on many pages. By using Android Fragment, we don’t need to duplicate the codes each time we need to put those buttons on the screen.

3.3.2. View Pulse Oximeter Record Page

All Pulse Oximeter record that has been created by user can be accessed from this page. In this page, user can select which records that they wanted to be uploaded to server by ticking the Record ID checkboxes and click “Upload” button. After record is successfully uploaded, it will be deleted from Android SQLite DB to reduce application storage usage. The record will also disappear from this screen. The screenshot of this page can be seen on Image 13.

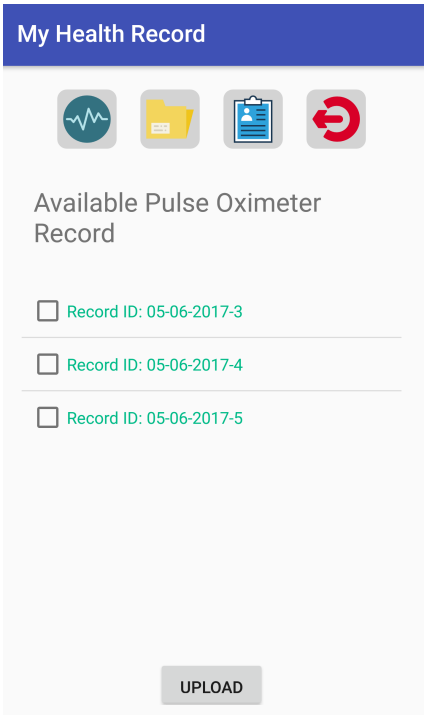

Image 13. View Pulse Oximeter Record page

3.3.3. View Analysis Result Page

Pulse oximeter record that has been analysed on the server can be accessed from this page. This page retrieve the analysis result data by calling Web Servicce API and show the result. It doesn't keep the analysis data on Android SQLite DB. The result analysis is the 2% dip index value. The interpretation of this value will be discussed on section 3.6. Sleep Apnea Analysis Appliation. The screenshot of this page can be seen on Image 14 below.

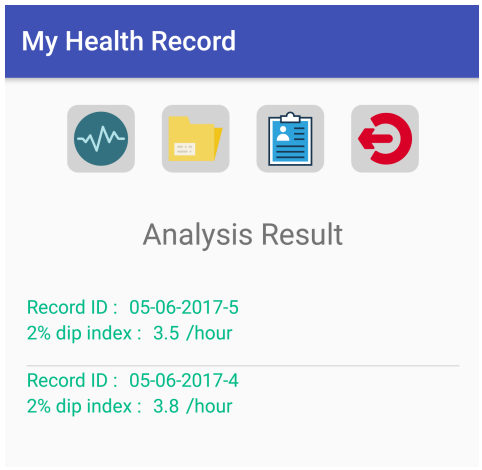

Image 14. View Analysis Result page

3.4. Backend Application

The Backend application in this project is developed based on the Backend application from internship project. The features enhancement of new Backend application can be seen on Table 6 below.

Existing Backend App Features	Enhancement Backend App Features
Registration	Use as is
Upload Pulse Oximeter Record	Changed
* not available on existing Android app	View Analysis Result Data

Table 6. Backend Application Feature Enhancement

3.4.1. Upload Pulse Oximeter Record

The modification being done in this feature is by implementing multiple pulse oximeter data upload. The existing system doesn't have a concept of Record, it's just put the uploaded Pulse Oximeter data as unrelated records. To implement the Record model, the Application flow is implemented as:

- Create Record data on database, and get the created Record ID.
- Iterate the sent pulse oximeter data, and save it to database.
- All those pulse oximeter data are grouped by one Record ID

3.4.2. View Analysis Result Data

This feature search for user's Record that has been Analysed by Sleep Apnea Analysis application. The record that has been analysed are marked done on the Database.

3.5. Database System

The database design in this project can be seen on Image 15 below. The user and token and table are similar with existing DB system from internship project. Table user is to store user information, while table token is to store user's token data. The current design support user to have multiple tokens. This design could accommodate if application would like to utilise more than one token to secure message communication between Android app and Backend application.

The oximeter table is changed, initially it didn't have record_id column. The major changed on Database system is the addition of new table record. This table keeps the record analysis result on dip2_score column. In the status column, when record is just created, the status is set into1 (new). When Sleep Apnea analysis application start processing the record, it will set the status into 2 (on process). After analysis application finished the process, it will set the status into 3 (done). If analysis process is finished successfully, the status result will be set into 1 (succesful). Meanwhile, if its failed the status result will be set into 2 (failed). This status and status_result classification is important for system monitoring so we could find out how many analysis process is successful/failed.

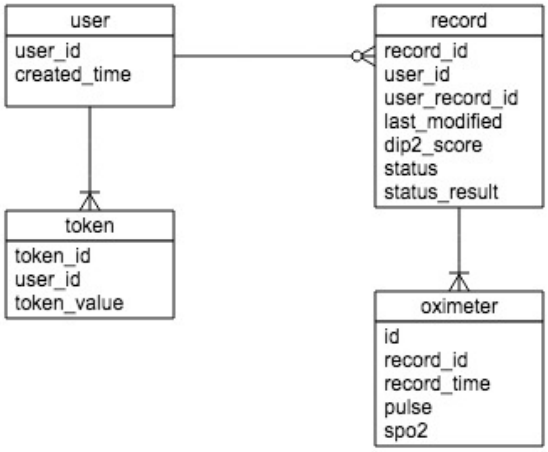

Image 15. Database Design

3.6. Sleep Apnea Analysis Application

The Sleep Apnea Analysis application is created using an open source software that published by Ben Raymond, R. M Cayton, M. J. Chappell to support their research that titled "Combined index of heart rate variability and oximetry in screening for the sleep apnoea/hypopnoea syndrome" [4].

The original code is created using Matlab. Due to technology expertise consideration, we decided to repackage the original code into java jar library to make it easier to integrate with the existing Database system, because currently we already have knowledge on how to integrate Maria DB with java during the Backend application development. The code repackaging itself is done using Matlab Library Compiler.

The research itself analyse the accuracy of Sleep Apnea analysis using electroencephalogram (EEG), and Polysomnogram (PSG). The PSG itself is the benchmark method, as it's the most accurate method available right now.

The original code combines data from EEG and Pulse Oximeter to do analysis using the method that called CODI score. However, in this project we only have access to Pulse Oximeter data, so we cannot directly use the original code. Beside implementing CODI score method, the researcher provides alternative analysis method by calculating the number of oxygen level dip per hour. In this project, we use this level dip per hour method to analyse the Pulse Oximeter data.

3.6.1. Analysis Result Interpretation

Apnea Hypopnea Index (AHI) is the number of Apnea event recorded per hour of sleep. The Apnea event itself is series of physical reaction that started with a person's airway repeatedly becomes blocked despite efforts to breathe. When sufficient air doesn't get into a person's lungs, the level of oxygen in the blood falls and the level of carbon dioxide (a waste product of metabolism) rises. After a few minutes of not breathing, a person may die.

Fortunately, with OSA, after a period of not breathing, the brain wakes up, and breathing resumes. This period of time can range from a few seconds to over a minute. When breathing resumes, the size of the airway remains reduced in size. The tissues surrounding this narrow airway vibrate—what we call snoring. In other words, snoring is a sign of an obstructed airway, but it does mean that a person is breathing; silence might indicate that the airway is completely blocked. Because waking up is necessary to end an episode of apnea or hypopnea, a person with OSA wakes up again and again throughout sleep. This reduces the duration and quality of sleep [16].

Based on the AHI, the severity of Sleep Apnea is classified as follows:

Severity	**AHI**
None/Minimal	AHI < 5 per hour
Mild	AHI ≥ 5, but < 15 per hour
Moderate	AHI ≥ 15, but < 30 per hour
Severe	AHI ≥ 30 per hour

Table 7. AHI Score Interpretation

Reductions in blood oxygen levels (desaturation) can be used as an indicator of an Apnea event. On the research paper, the oxygen level desaturation is referred as oxygen level dip. Based on the research result, the index of 2% dips (ODI-2) is the best predictor of the AHI compared to 3% dips (ODI-3) and 4% dips (ODI-4). So, in this project we use the ODI-2 value, which is called the "2% dip index" on the Android Analysis Result page. The ODI-2 value is used to classify the severity of Sleep Apnea using the same threshold classification as in Table 7.

3.6.2. Apanea Analysis Application Management

The Apnea Analysis application is created using Java programming. It is called Matlab function that has been repackaged into jar to calculate the ODI-2 value. The application itself run as Linux background process. Below are some of the linux command that can be used to start, monitor, and stop the process.

Task	**Linux Command**
Start Process	./apnea_analysis.sh Notes: This command start the application as Linux background process.
Monitor Proces	jps | grep "AnalysisController" Notes: If the application is running, this command will show the application Process ID (PID) and the Process name
Stop Process	kill -9 PID Notes: To get the PID use the Monitor Process command as mentioned above.
Monitor the log	/var/log/apnea_analysis.log Notes: This is the location of application log file.

Table 8. Apnea Analysis Application Management Task

4. System Performance Evaluation and Analysis

To test the system performance, we generate a test load that consist of 500 concurrent users (generated through Registration API request), and each user will generate 50 sequential requests (generated through Upload data APi request). In total, there will be 25000 requests to the Backend application.

We tried several VM configurations, and evaluate the average response time for each of them. The detail of VM configuration and its average response time can be seen on Table 9 below.

No	Backend Application VM Configuration	Average Response Time (in milliseconds)
1	Scale Set, 3 VM (A0 Machine Type) 1 core CPU 0.75 GB Memory Scale out rule: • CPU usage > 75% for 5 minutes, add 1 VM to scale set • CPU usage < 25% for 5 minutes, remove 1 VM from scale set	11198
2	Scale Set, 3 VM (A0 Machine Type) 1 core CPU 0.75 GB Memory Scale out rule: • Average number of request > 100 for 5 minutes, add 1 VM to scale set • Average number of request < 10 for 5 minutes, remove 1 VM to scale set	9838
3	No Scale Set, 1 VM (A2 Machine Type) 2 cores CPU 3.5 GB Memory	10071
4	No Scale Set, 1 VM (A3 Machine Type) 4 cores CPU 7 GB Memory	7574

Table 9. VM Configuration Performance

From the observation result on Table 9, below are the conclusions that can be made:

- Configuration 1 doesn't perform as initially expected. The scale out rule which use CPU percentage turns out never executed. The CPU usage never reach 75%. The highest CPU usage observed on Azure monitoring tools is only 28%. This configuration is deemed to be unsuitable for this project, because the Backend application turns out doesn't generate CPU intensive tasks.
- Configuration 3 have slightly bigger memory compared to Configuration 2, but the number its CPU cores is smaller than Configuration 2. From this mixed system advantages, turns out Configuration 2 performs better than Configuration 3.
- Configuration 4 have the best performance among others. However, considering that its machine configuration is superior even compared to the combination of 3 VMs on scale set, average response time might be not the fairest factor to judge the best configuration.

To make a better judgement, we created a formula to score each configuration. The formula is: Score = VM price * Number of VM * Average Response Time.

The formula is created to judge which configuration has the best performance for each of its additional resource. Configuration that has the lowest score means, the price is low (lower capacity machine), and the average response time is also low (system has fast response). In conclusion, system with the lowest score is considered as the best configuration. The machine cost can be seen on Image 16 below.

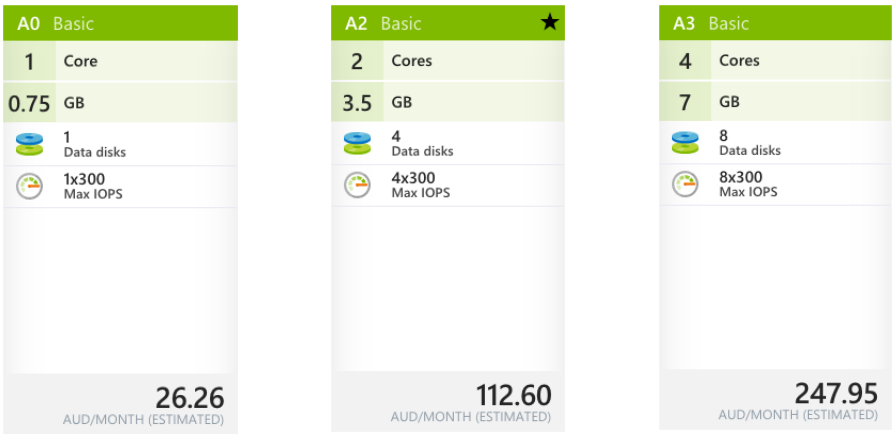

Image 16. Azure Machine Cost

Based on the score formula and observation data, below is the score for each configuration.

- Configuration 2: 26,26 * 3 * 9838 = 775.038
- Configuration 3: 112.6 * 1 * 10071 = 1.113.995
- Configuration 4: 247.95 * 1 * 7574 = 1.877.973

From the above calculation, Configuration 2 come out as configuration with the best performance. One of the possible reason why system with superior capacity such as Configuration 4 cannot deliver best performance for each of its additional resource, is probably because we send a lot of request to 1 node. The request queue on that node will pile up. As a result, the response got slower. Meanwhile, if we spread the request into multiple nodes like in Configuration 2, the request queue on each node is not a big as queue on Configuration 4. From

this result, to achieve a better performance, it might be better to use a lot of small machines rather than using 1 big machine.

5. Future Works

For future works, there are some possibility for improvement, those are:

- Use JBoss batch framework JBatch rather than regular java application.
  JBatch offer some good features that can be helpful to monitor batch application such as web interface batch process monitoring. However, due time constraint we cannot utilise this framework in this project.

- Combined Sleep Apnea analysis with data from another health device such as EEG.
  Combining data from other health device such as EEG will be able to improve the accuracy of Sleep Apnea analysis, because we are not just considering one symptom of Apnea event.

6. References

[1] Nonin. (2017). What is Pulse Oximetry.
Retrieved From: http://www.nonin.com/What-is-Pulse-Oximetry

[2] National Heart, Lung, and Blood Institute. (2017). What is Sleep Apnea.
Retrieved From: https://www.nhlbi.nih.gov/health/health-topics/topics/sleepapnea

[3] National Heart, Lung, and Blood Institute. (2017). Sleep Apnea Diagnosis.
Retrieved From: https://www.nhlbi.nih.gov/health/health-topics/topics/sleepapnea/diagnosis

[4] Ben Raymond, R. M Cayton, M. J. Chappell. (2003, February). Combined index of heart rate variability and oximetry in screening for the sleep apnoea/hypopnoea syndrome. In Journal of Sleep Research Volume 12, Issue 1. European Sleep Research Society.

[5] Wikipedia. (2017). Scalability.
Retrieved From: https://en.wikipedia.org/wiki/Scalability

[6] Wikipedia. (2017). Microsoft Azure.
Retrieved From: https://en.wikipedia.org/wiki/Microsoft_Azure

[7] Chris Walden. (2017). Developer and administrator tools for Microsoft Azure.
Retrieved From: https://blogs.technet.microsoft.com/uktechnet/2017/05/24/developer-and-administrator-tools-for-microsoft-azure/

[8] Microsoft. (2017). CLI 2 Announcement.
Retrieved From: https://azure.microsoft.com/en-gb/blog/announcing-azure-cli-2-preview/

[9] Microsoft. (2017). Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources.
Retrieved From: https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-deployment-model

[10] Microsoft. (2017). What are virtual machine scale sets in Azure?.
Retrieved From: https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-overview

[11] Wikipedia. (2017). Apache JMeter.
Retrieved From: https://en.wikipedia.org/wiki/Apache_JMeter

[12] Nonin (2017). Pulse Oximeter Illustration [digital image].
Retrieved From:
http://www.nonin.com/Go2Nonin/images/learnmore/PulseOxFingerIllustration.jpg

[13] Proshop (2017). Nonin Pulse Oximeter [digital image].
Retrieved From:
http://www.shop.proactmedical.co.uk/photos/1.445065nonin_3230_op_hand.jpg

[14] ORA® Oral Surgery, Sleep Disorder & Implant (2017). Polysomnography [digital image].

Retrieved From:
http://nympw3kj6tl1io7m52utiz4g.wpengine.netdna-cdn.com/wp-content/uploads/2012/01/Polysomnography-a-sleep-study.jpg

[15] Microsoft (2017). PaaS [digital image].
Retrieved From:
https://azurecomcdn.azureedge.net/cvt-9d9c17002381b078199fcf784d4ed60d6ce505353dc459585b7a0c5afb13ef12/images/page/overview/what-is-paas/what-is-paas.png

[16] Harvard Medical School. (2017). Understanding AHI and Oxygen Desaturation.
Retrieved From:
http://healthysleep.med.harvard.edu/sleep-apnea/diagnosing-osa/understanding-results