Autumn Report

Thomas Flynn

16117743

Sean McGrath

M.Eng Information & Network Security

2017

Electronic & Computer Engineering

Docker containers deployed using Bluemix

1 Introduction and report outline

1.1 Project description

Utilizing Docker containers and a suitable cloud container deployment service, this project aims to provide a highly scalable back-end prototype for the university's campus wide mobile sensor application.

This will consist of developing micro-services running within Docker containers that will process Global Positioning System coordinates from the university students who use the mobile application. This information will be stored in a database for data analytic purposes.

Creating the back-end for such an application will allow me to grow both my development and operational skills in relation to designing highly scalable and secure cloud applications, which are skills that are in increasing demand in the world of software development.

1.2 Project aims and objectives

Primary objectives

1.) Deploy a micro-service running within a container on the cloud that will read mocked sensor data.

2.) Design and create a suitable database that can handle the load balancing factors of a highly scalable application.

3.) Create a sensor micro-service that will transmit GPS coordinates.

4.)

Secondary objectives

Personal goals

1.3 Report outline

1.4 Acronyms

MS - Microservice

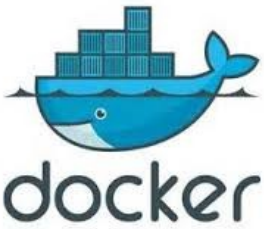
GPS - Global Positioning System

DB - Databases

2 Literature survey

2.1 Outline of research

2.2 Docker research



Docker containers

Docker containers wrap a piece of software in a complete file system that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment. **[7]**

Lightweight

Containers running on a single machine share the same operating system kernel; they start instantly and use less RAM. Images are constructed from layered filesystems and share common files, making disk usage and image downloads much more efficient. **[7]**

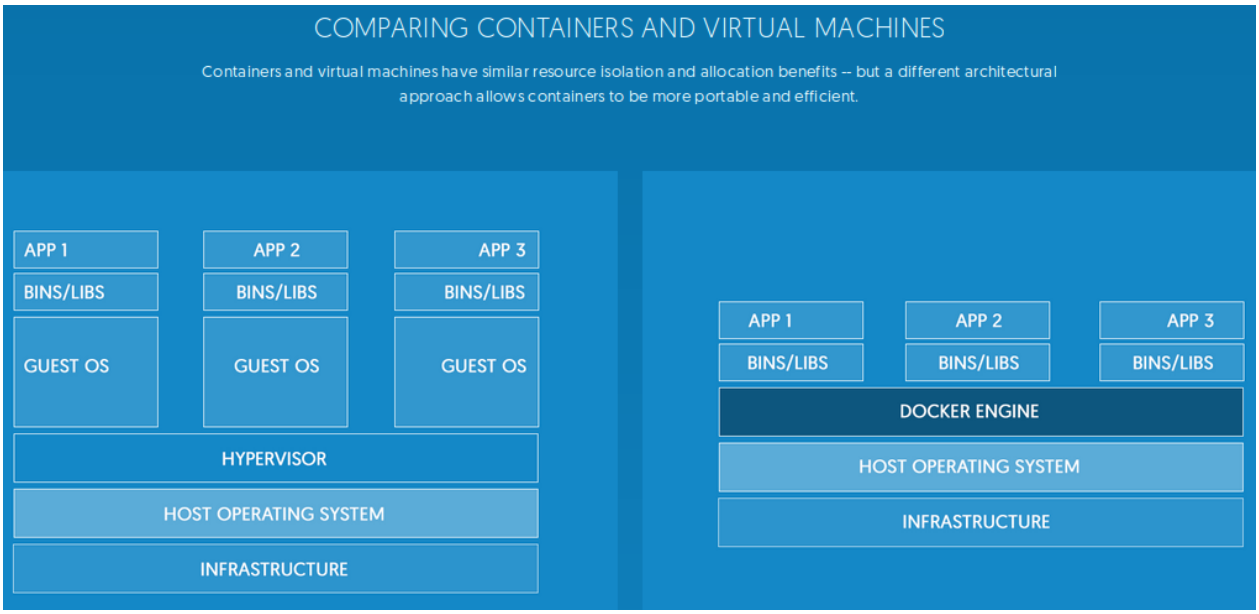
Open

Docker containers are based on open standards, enabling containers to run on all major Linux distributions and on Microsoft Windows -- and on top of any infrastructure. **[7]**

Secure by default

Containers isolate applications from one another and the underlying infrastructure, while providing an added layer of protection for the application. **[7]**

Comparing container and virtual machines



Virtual machines

Virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system -- all of which can amount to tens of Gbs. **[7]**

Containers

Containers include the application and all of its dependencies --but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud. **[7]**

Eliminate environment inconsistencies

Packaging an application in a container with its configurations and dependencies guarantees that the application will always work as designed in any environment: locally, on another machine, in test or production. No more worries about having to install the same configurations into different environments. **[7]**

Quickly scale

Docker containers spin up and down in seconds, making it easy to scale application services to satisfy peak customer demand, and then reduce running containers when demand ebbs. **[7]**

2.1 IBM Bluemix

IBM and Docker integrated container solutions

IBM and Docker offer integrated container solutions that can meet the diverse needs of enterprises. Supporting the creation and deployment of multi-platform, multi-container workloads across hybrid infrastructures, IBM and Docker accelerate application delivery and enable application lifecycle management for Dockerized containers. **[1]**

Create scalable and portable apps and services

Having a hosted container service eliminates the need for users to install, operate and scale container infrastructure meaning they can create containerized applications quickly and easily. Since containers maintain all application configuration settings and dependencies, there is no need to change or reconfigure applications for different environments. Applications can move seamlessly between development, test and production environments. Operations teams can spend less time doing manual installations and troubleshooting issues. **[2]**

Easily package and deploy applications that will run in hybrid environments

IBM Containers on Bluemix makes it possible to more quickly and easily package and deploy applications that can run across hybrid environments. This technology also makes it possible to run many more applications on the same servers, providing improved resource utilization. **[2]**

Improve productivity and increase resource utilization

IBM Container users can spin up and tear down standardized environments across development and operations groups faster while increasing application density per server. **[2]**

Achieve maximum application portability

IBM Containerized applications are seamlessly portable across environments and computing platforms, letting users move them freely to, from and between Linux-based cloud environments. IBM Containers does not add any proprietary code to the Docker engine or registry. **[2]**

2.2 Microsoft Azure Container Service

Docker for Azure

An integrated, easy-to-deploy environment for building, assembling, and shipping applications on Microsoft Azure, Docker for Azure is a native Azure application optimized to take optimal advantage of the underlying Azure IaaS services while giving you a modern Docker platform that you can use to deploy portable apps. Docker for Azure installs a Swarm of Docker Engines secured end to end with TLS by default, and is integrated with Azure VM Scale Sets for autoscaling, Azure Load Balancer and Azure Storage. **[3]**

Create an optimized container hosting solution

Azure Container Service optimizes the configuration of popular open source tools and technologies specifically for Azure. You get an open solution that offers portability for both your containers and your application configuration. You select the size, the number of hosts, and choice of orchestrator tools, and Container Service handles everything else. **[4]**

Scale and orchestrate using DC/OS or Docker Swarm

Choose the tools and solution that best suits your needs for Docker container orchestration and scale operations. Use the Mesos-based DC/OS or use Docker Swarm and Compose for a pure Docker experience. **[4]**

2.3 Amazon Container Service

Docker For AWS

An integrated, easy-to-deploy environment for building, assembling, and shipping applications on AWS, Docker for AWS is a native AWS application optimized to take optimal advantage of the underlying AWS IaaS services while giving you a modern Docker platform that you can use to deploy portable apps. Docker for AWS does not require any software installed. **[5]**

Amazon EC2 Container Service

Amazon EC2 Container Service (ECS) is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.**[6]**

Flexible Container Placement

Amazon ECS is a shared state, optimistic concurrency system that supports multiple schedulers on the same cluster for each business or application-specific requirement.**[6]**

Performance at Scale

Amazon EC2 Container Service is built on technology developed from many years of experience running highly scalable services. You can launch tens or tens of thousands of Docker containers in seconds using Amazon ECS with no additional complexity.**[6]**

Secure

Amazon EC2 Container Service launches your containers on your own EC2 instances. No compute resources are shared with other customers. Your clusters run in a VPC allowing you to use your own VPC security groups and network ACLs. These features provide you a high level of isolation and help you use Amazon ECS to build highly secure and reliable applications.**[6]**

2.3 NoSQL Database

A NoSQL database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases.

2.3.1 NoSQL features

1. The ability to horizontally scale “simple operation” throughput over many servers,

2. The ability to replicate and to distribute (partition) data over many servers.

3. Simple call level interface or protocol (in contrast to SQL binding),

4. Efficient use of distributed indexes and RAM for data storage, and

5. The ability to dynamically add new attributes to data records.

2.3.2 Common types of NoSQL databases

Document store

Documents are addressed in the database via a unique *key* that represents that document. One of the other defining characteristics of a document-oriented database is that in addition to the key lookup performed by a key-value store, the database offers an API or query language that retrieves documents based on their contents.

Key value store

Data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection. The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented as an extension of it. The key-value model can be extended to a discretely ordered model that maintains keys in lexicographic order.

Graph

This kind of database is designed for data whose relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them. The type of data could be social relations, public transport links, road maps or network topologies.

2.5 Neo4j

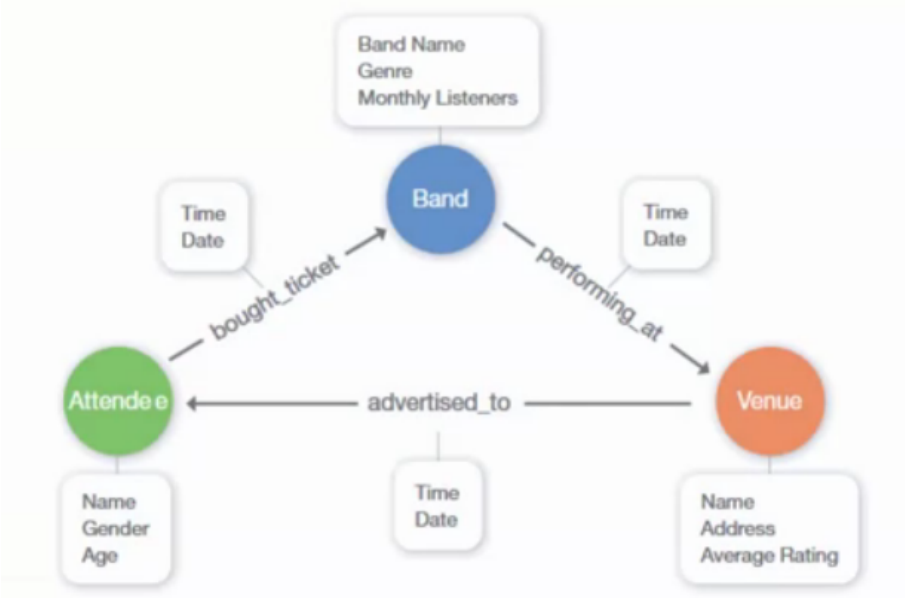
Graphs are the most efficient and intuitive way of working with data, mimicking the interconnectedness of ideas in the human mind. Neo4j is built from the ground up to harness the power of graphs for real-time, bottom-line insights.

Graph databases

Graph databases are based on graph theory. Graph databases employ nodes, edges and properties.

- **Nodes** represent entities such as people, businesses, accounts, or any other item you might want to keep track of. They are roughly the equivalent of the *record*, *relation* or *row* in a relational database.
- **Edges**, also known as *graphs* or *relationships*, are the lines that connect nodes to other nodes; they represent the relationship between them.
- **Properties** are pertinent information that relate to nodes.

Neo4j graph example



Neo4j vs Relational Databases

Category	Relational Database	Neo4j, Native Graph Database
Data Storage	Storage in fixed, pre-defined tables with rows and columns with connected data often disjointed between tables, crippling query efficiency.	Graph storage structure with index-free adjacency results in faster transactions and processing for data relationships.
Data Modeling	Database model must be developed with modelers and translated from a logical model to a physical one. Since data types and sources must be known ahead of time, any changes require weeks of downtime for implementation.	Flexible, "whiteboard-friendly" data model with no mismatch between logical and physical model. Data types and sources can be added or changed at any time, leading to dramatically shorter development times and true agile iteration.
Query Performance	Data processing performance suffers with the number and depth of JOINs (or relationships queried).	Graph processing ensures zero latency and real-time performance, regardless of the number or depth of relationships.
Query Language	**SQL**: A query language that increases in complexity with the number of JOINs needed for connected data queries.	**Cypher**: A native graph query language that provides the most efficient and expressive way to describe relationship queries.
Transaction Support	ACID transaction support required by enterprise applications for consistent and reliable data.	Retains ACID transactions for fully consistent and reliable data around the clock – perfect for always-on global enterprise applications.
Processing at Scale	Scales out through replication and scale up architecture is possible but costly. Complex data relationships are not harvested at scale.	Graph model inherently scales for pattern-based queries. Scale out architecture maintains data integrity via replication. Massive scale up possibilities with IBM POWER8 and CAPI Flash systems.
Data Center Efficiency	Server consolidation is possible but costly for scale up architecture. Scale out architecture is expensive in terms of purchase, energy use and management time.	Data and relationships are stored natively together with performance improving as complexity and scale grow. This leads to server consolidation and incredibly efficient use of hardware.

Native Graph Storage

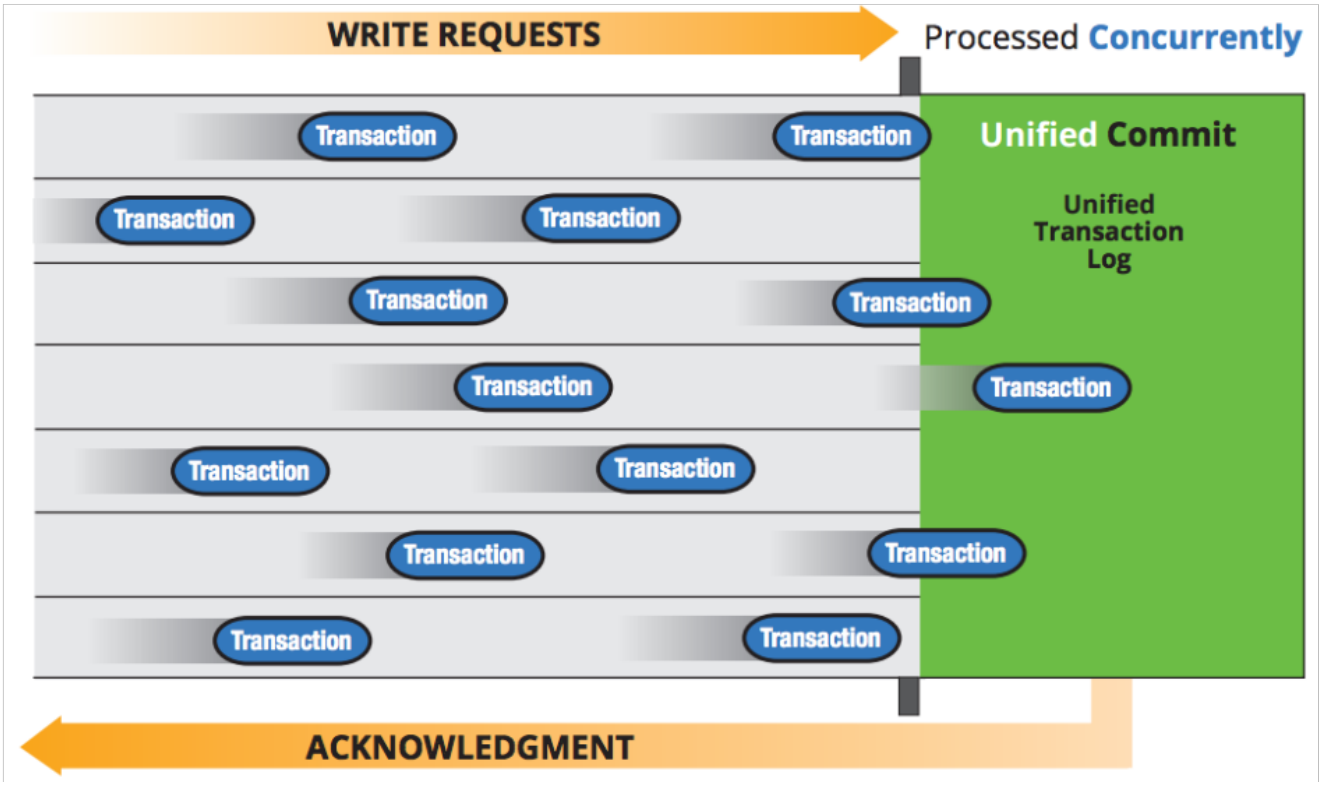
Neo4j uses native graph storage that is specifically designed to store and manage interconnected data. Each piece of data in Neo4j has an explicit connection to every related entity, meaning database queries can ignore anything that's not connected rather than crawl the entire dataset. The result is unparalleled speed and scale.

Non-native graph databases use relational or object-oriented databases for data storage instead, which becomes much more latent as data volume and query complexity grow.

Native Graph Processing

Neo4j's native graph processing (also known as “index-free adjacency”) is the most efficient means of processing graph data because connected nodes physically “point” to each other in the database. This means Neo4j can evaluate results at a rate of millions of hops per second delivering constant-time performance regardless of the size of the dataset.

Non-native graph processing often defaults to expensive index lookups which result in reduced performance.



In Neo4j 2.2, concurrent writes are bundled together, optimizing throughput by minimizing the number of disk operations and amortizing transaction cost. The result is a huge improvement in concurrent transactions per second.

We have also eliminated the two-phase commit that occurred with every write operation, by unifying the transaction logs for graph data and related indexes, recognizing them as a single, mechanically sympathetic operational event. This immediately benefits all transactions.For data import we went one step further, allowing bulk imports to be performed as one highly mechanically efficient operation. The new `neo4j-import` tool writes the graph directly to disk, bypassing the transactional database writer for offline databases. This results in sustained write throughput into the million of records per second, for graphs of all sizes, even into the tens of billions of nodes and relationships.Neo4j 2.2's write performance delivers exceptional performance:

- Up to 100x higher transactional throughput with concurrent load
- Vastly improved core scale-up, to more fully utilize modern hardware
- Bulk data import at over 1M records/second, loading all of DBpedia (4.58M nodes and 20M+ relationships) in under 100 seconds

2.6 Neo4j on IBM POWER8

Neo4j on IBM POWER8 is the result of a joint effort between Neo4j and IBM engineering to provide the world's most scalable graph database platform capable of storing and processing graphs of extremely large size all in-memory – shattering all previous real-time scalability limits.

IBM Power Systems Built with POWER8

IBM Power Systems are specifically designed to capture and manage data from a variety of sources and put that data to work in your enterprise – including real-time graph processing.

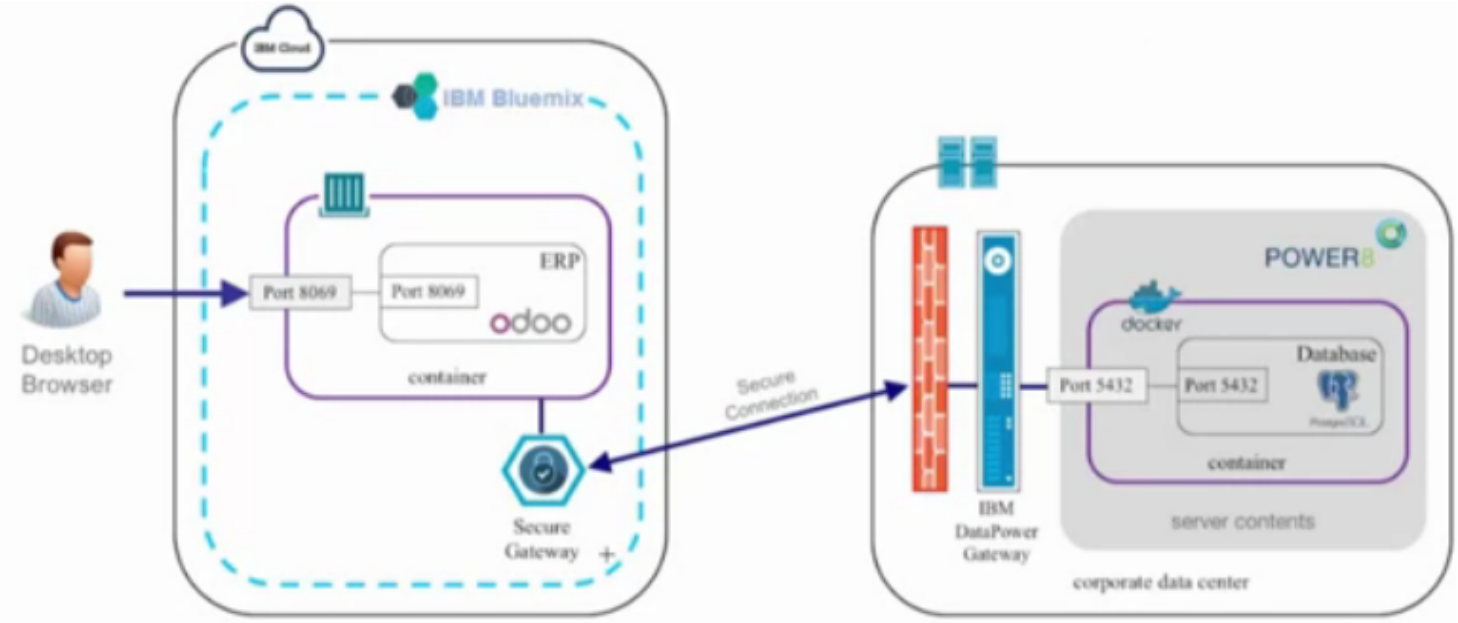
Highlights of the POWER8 advantage:

- **Processors:**
  - 4x more hardware threads per core (versus an x86 core)
  - 96 threads on a 12-core chip
  - Up to 1536 threads per system
  - Better Simultaneous Multi-threading (SMT) performance than Hyper-threading (HT)
- **Memory:**
  - 4x more memory bandwidth (versus an x86 core)
  - Up to 16 TB of DRAM
  - 192 GBs of sustained memory bandwidth per scale-out socket
  - Up to 230 GBs per second of memory bandwidth for enterprise-class server
- **I/O Bandwidth:**
  - 96 GBs per second of peak bandwidth
  - 4x more bandwidth than previous generation (POWER7)

3 Theory

3.1 Analytic and theoretical aspects of the project

IBM Bluemix and POWER8 configuration example



The above figure demonstrates an example of an IBM container running within IBM Bluemix (left hand side) connecting to a database running within a Docker container that is deployed with IBM's POWER8 (right hand side). The IBM container is linked with the secure gateway service which helps provide a secure connection to the database.

To connect to the database you have to go through 2 layers. The first is a firewall. The second is the IBM DataPower gateway which will make a connection with the secure gateway on the Bluemix side. Behind these 2 layers is the POWER8 system which is hosting the Docker environment.

4 High level design

4.1 High level design

4.2 Software

4.3 Process related block 1

4.4 Process related block 2

4.5 Process related block 3

4.6 Process related block 4

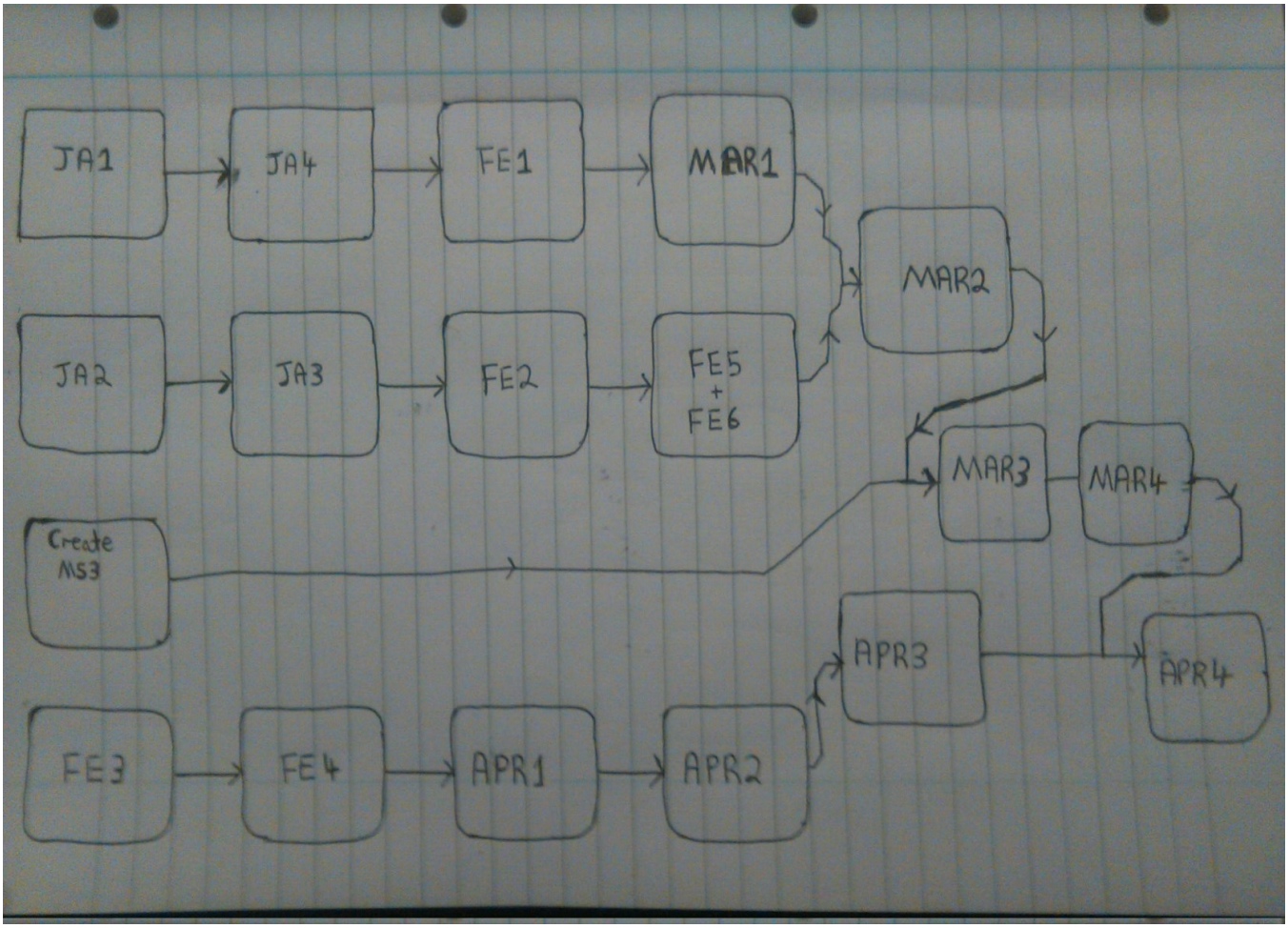
5 Detailed action plan

5.1.1 Spring action plan

Table of tasks for Spring semester

Task Description	Task Identity	Duration (days)	Preceding Task
Research similar microservices	JA1	5	
Research Neo4J and Cypher	JA2	5	
Research POWER8 graph database examples	JA3	5	JA2
Deploy and run MS1 that creates and transmits pseudo sensor data	JA4	5	JA1
Deploy and run MS2 that handles incoming data	FE1	5	JA4
Deploy a basic Neo4j graph database on POWER8 with suitable pseudo data	FE2	5	JA3
Research Bluemix website deployment examples	FE3	3	
Deploy a basic website on Bluemix	FE4	3	FE3
Create permanent graph DB on POWER8	FE5	5	FE4
Create temporary DB on POWER8	FE6	5	FE5
Develop code for MS2 to store incoming data for a short period of time	MAR1	5	FE1
MS2 writes to temporary database every 15 minutes	MAR2	5	MAR1 & FE6
MS3 reads data from temporary database	MAR3	5	
MS3 writes to permanent POWER8 database	MAR4	5	MAR3
Develop map user interface for website	APR1	5	FE4
Identify important UL GPS coordinates for map user interface	APR2	5	APR1
Develop code for website to submit a simple database request	APR3	5	APR2
MS4 handles a simple website client database request	APR4	5	APR3 & MAR4

Critical path map



Spring Gantt chart

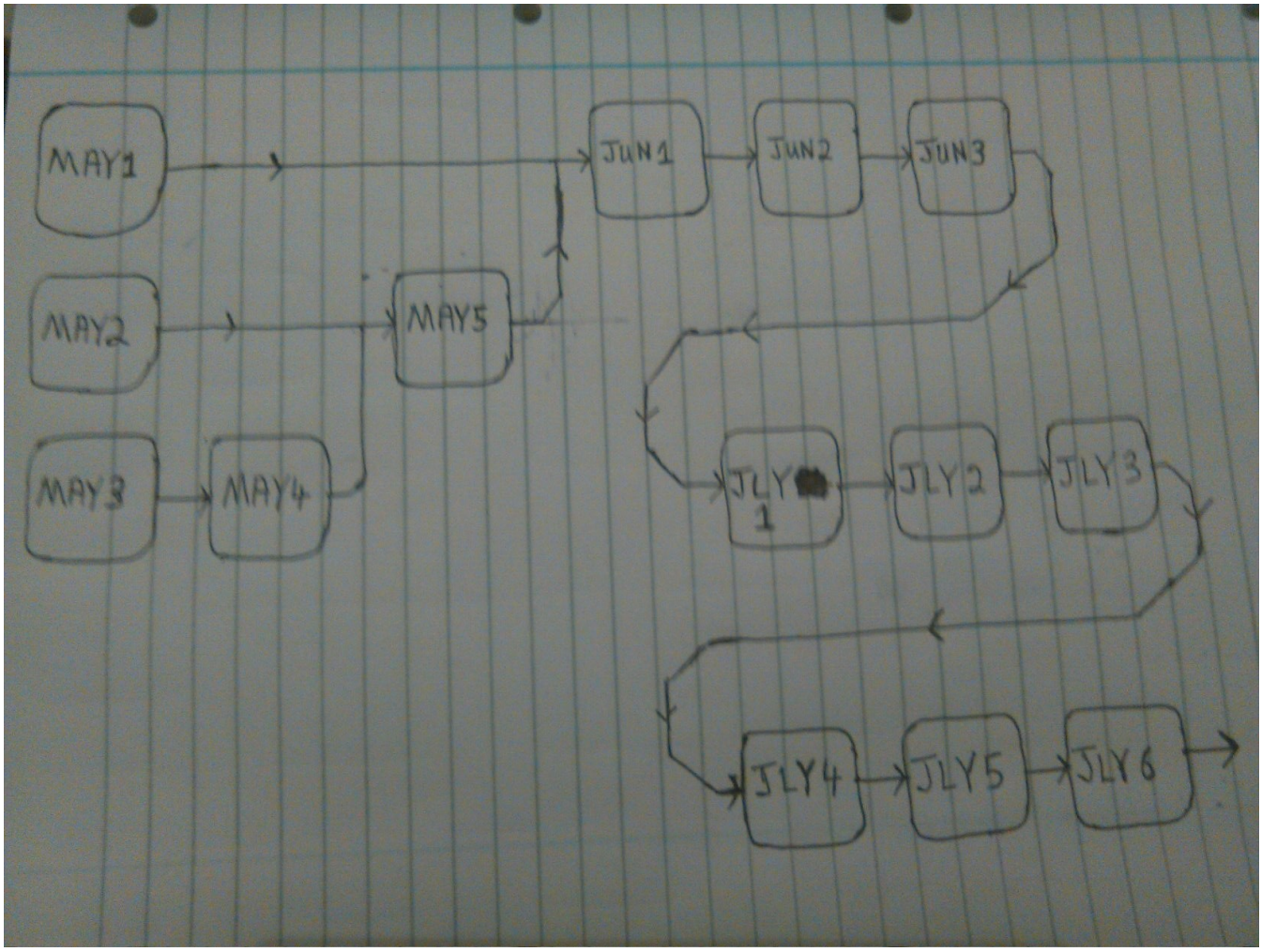
Task Identity	Start	Days	End
JA1	02/01/17	5	08/01/17
JA2	09/01/17	5	15/01/17
JA3	16/01/17	5	22/01/17
JA4	23/01/17	5	29/01/17
FE1	30/01/17	5	05/02/17
FE2	06/02/17	5	12/02/17
FE3	13/02/17	3	15/02/17
FE4	16/02/17	3	20/02/17
FE5	20/02/17	5	26/02/17
FE6	27/02/17	5	05/03/17
MAR1	06/03/17	5	12/03/17
MAR2	13/03/17	5	19/03/17
MAR3	20/03/17	5	26/03/17
MAR4	27/03/17	5	02/04/17
APR1	03/04/17	5	09/04/17
APR2	10/04/17	5	16/04/17
APR3	17/04/17	5	23/04/17
APR4	24/04/17	5	30/04/17

5.4.2 Summer action planning

Table of tasks for Summer semester

Task Description	Task Identity	Duration	Preceding Activity
Website displays DB readings on website map user interface	MAY1	5	APR4
Load balancing and scalability research	MAY2	5	
Increase funding	MAY3	5	
Increase MS1 instances	MAY4	5	MAY3
MS2 Automatically handles increase/decrease of incoming sensor data from MS1	MAY5	5	MAY2 & MAY 4
Test scalability of MS2 writing to temporary database	JUN1	9	MAY1 & MAY 5
Test scalability of MS3 reading from temporary database and writing to permanent database	JUN2	8	JUN1
End to end testing of website and permanent database under increased scale	JUN3	3	JUN2
End to end testing of the whole system	JLY1	4	JUN3
Collect results	JLY2	2	JLY1
Analyze results	JLY3	3	JLY2
Discuss results with supervisor	JLY4	3	JLY3
Write up initial report and submit to supervisor for review	JLY5	3	JLY4
Edit initial report	JLY6	3	JLY5

Summer critical path map



Summer Gantt chart

Task Identity	Start	Days	End
MAY1	01/05/17	5	07/05/17
MAY2	08/05/17	5	14/05/17
MAY3	15/05/17	5	21/05/17
MAY4	23/05/17	5	29/05/17
MAY5	29/05/17	5	04/06/17
JUN1	05/06/17	9	15/06/17
JUN2	16/06/17	8	27/06/17
JUN3	28/06/17	3	02/07/17
JLY1	03/07/17	4	06/07/17
JLY2	07/07/17	2	10/07/17
JLY3	11/07/17	3	13/07/17
JLY4	14/07/17	3	18/07/17
JLY5	19/07/17	3	23/07/17
JLY6	24/07/17	3	26/07/17

6 Discussion

6.1 Progress to date

6.2 Challenges and risks

7 Requirements of facilities and materials

8 References

[1]"IBM", *Docker*, 2015. [Online]. Available: https://www.docker.com/IBM. [Accessed: 14- Oct- 2016].

[2]"IBM Bluemix - Containers", *Ibm.com*, 2016. [Online]. Available: http://www.ibm.com/cloud-computing/bluemix/containers/. [Accessed: 14- Oct- 2016].

[3]"Microsoft", *Docker*, 2015. [Online]. Available: https://www.docker.com/microsoft. [Accessed: 14- Oct- 2016].

[4]"Azure Container Service | Microsoft Azure", *Azure.microsoft.com*, 2016. [Online]. Available: https://azure.microsoft.com/en-us/services/container-service/. [Accessed: 14- Oct- 2016].

[5]"Get Started with Docker and AWS", *Docker*, 2015. [Online]. Available: http://www.docker.com/aws. [Accessed: 14- Oct- 2016].

[6]"Amazon EC2 Container Service – Docker Management – AWS", *Amazon Web Services, Inc.*, 2016. [Online]. Available: https://aws.amazon.com/ecs/. [Accessed: 14- Oct- 2016].

[7]"What is Docker?", *Docker*, 2015. [Online]. Available: https://www.docker.com/what-docker. [Accessed: 14- Oct- 2016].

[8]"IBM Bluemix Docs", *Console.ng.bluemix.net*, 2016. [Online]. Available: https://console.ng.bluemix.net/docs/services/graphdb/index.html. [Accessed: 15- Oct- 2016].

[9]"IBM Graph - API", *Ibm-graph-docs.ng.bluemix.net*, 2016. [Online]. Available: https://ibm-graph-docs.ng.bluemix.net/api.html. [Accessed: 15- Oct- 2016].

[10]"GremlinDocs", *Gremlindocs.spmallette.documentup.com*, 2016. [Online]. Available: http://gremlindocs.spmallette.documentup.com/. [Accessed: 15- Oct- 2016].

[11]A. TinkerPop, "Apache TinkerPop", *Tinkerpop.apache.org*, 2016. [Online]. Available: https://tinkerpop.apache.org/. [Accessed: 15- Oct- 2016].

[12]"Neo4j on IBM POWER8® - Neo4j Graph Database", *Neo4j Graph Database*, 2016. [Online]. Available: https://neo4j.com/neo4j-on-ibm-power8/. [Accessed: 15- Oct- 2016].

[13]"Neo4j Graph Database: Unlock the Value of Data Relationships", *Neo4j Graph Database*, 2016. [Online]. Available: https://neo4j.com/product/. [Accessed: 15- Oct- 2016].

[14]"Implementing a Containers-based Hybrid Cloud ERP environment using IBM Bluemix", *YouTube*, 2016. [Online]. Available: https://www.youtube.com/watch?v=9XbzUqCvl3I. [Accessed: 15- Oct- 2016].

[15]"Neo4j Spatial: Finding Things Close to Other Things - Neo4j Graph Database", *Neo4j Graph Database*, 2011. [Online]. Available: https://neo4j.com/blog/neo4j-spatial-part1-finding-things-close-to-other-things/. [Accessed: 15- Oct- 2016].

[16]R. Catell, "Scalable SQL and NoSQL Data Stores", 2016. [Online]. Available: http://www.cattell.net/datastores/Datastores.pdf. [Accessed: 16- Oct- 2016].

[17]N. Staff, "Neo4j 2.2.0 – Massive Write & Read Scalability, Faster Cypher Query Performance, Improved Developer Productivity - Neo4j Graph Database", *Neo4j Graph Database*, 2015. [Online]. Available: https://neo4j.com/blog/neo4j-2-2-0-scalability-performance/. [Accessed: 16- Oct- 2016].