UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Spring Report

Project: Docker Containers Deployed Using Bluemix

Name: Thomas Flynn

ID: 16117743

Supervisor: Dr. Sean McGrath

Course: Information & Network Security MEng

Year: 2017

Department: Electronic & Computer Engineering

Table of Contents

1 Introduction and Report Outline....4

1.1 Project Description....4

1.2 Report Outline....5

2 Literature Survey....6

2.1 Section Outline....6

2.2 IoT architecture....7

2.3 Load Balancing....8

2.4 MQTT Brokers and Clients....9

2.5 Redis....10

2.6 Kafka....10

2.7 ELK Stack....11

2.8 IBM Bluemix Logging and Monitoring....12

2.10 Container Orchestration....13

2.11 Service Discovery....14

2.12 Amalgam8....15

2.13 DevOps CI/CD Pipeline....16

2.14 Similar Projects....17

3 Architecture....18

3.1 Load Balancing Diagram....18

3.2 Mosca Broker Container....19

3.3 Mosca Broker Container Group....20

3.4 HaProxy Configuration....21

3.5 Node-Red.....23
3.6 OpenWhisk.....24
4 Project Management.....25
4.1 Trello Labels.....25
4.2 Project Iteration Planning.....25
4.3 Iteration 1: May 1st – May 21st.....26
4.4 Iteration 2: May 22nd – June 11th.....26
4.5 Iteration 3: June 12th – July 2nd.....26
4.6 Iteration 4: July 3rd – July 23rd.....26
5 Progress to Date.....27
5.1 Christmas Break Summary.....27
5.2 Week 1 Log.....28
5.2.3 Week 1 Log Entries.....28
5.3 Week 2 Log.....30
5.4 Week 3 Log.....32
5.5 Week 4.....34
6 Bibliography.....36

1 Introduction and Report Outline

1.1 Project Description

Docker Containers remove unecessary resources out of the operating system. This results in more CPU, memory, and network resources for middleware and application execution. As a rule of thumb, Containers are 10x more efficient than virtual machines in terms of CPU, memory, network and other resource utilization.

IBM has formed a close partnership with Docker in order to make Containers Enterprise Ready. IBM Software products such as Websphere Application Server (WAS) and IBM BlueMix are already Container ready.

This project is based on engineering a scalable back-end solution for tracking in real-time the number of vehicles and students on the university campus. Due to their scalable capabilities, Containers were chosen in order to handle the variation of traffic on campus over a 24 hour period. During periods of peak traffic, the number of Containers handling the processing of sensor data can efficiently scale up to meet demand. Computing resources will not be wasted during periods of low traffic because the number of Containers will be automatically scaled down.

The clients (cars, students) will transmit GPS (Global Position System) data periodically to the server using a lightweight IoT protocol known as MQTT (MQ Telemetry Transport). This is a commonly used protocol for constrained devices. Both client and server will incorporate features to ensure that communication and caching of data is only performed when appropriate. For example the client will not attempt to transmit data if its coordinates are outside college bounds, similarly the server won’t cache data received that is outside college bounds.

IBM Cloudant is a managed NoSQL JSON database service built to ensure that the flow of data between an application and its database remains uninterrupted and highly performant. Cloudant has excellent features for indexing, quering and visualizing geo-spatial data. The prototype for this project will use Cloudant to store sensor data of the last 24 hours only, in order to reduce development cost. Cost evaluation of storing data for longer periods will be calculated at a later stage in the development life cycle.

Node-RED is a simple visual tool that makes it easy to wire together events and devices for the Internet of Things. To test the scalability of the application, multiple Node-Red flows will inject messages to different MQTT topics (e.g car-iotp and mobile-iotp) in order to determine the load balancing benchmarks of the project.

To help pave the way for future students to build applications on top of this project, various Development Operations aspects will be considered such as GitHub repositories, continuous integration/development pipelines, Container orchestration and microservices frameworks. This area of the project aims to reduce the operational complexity for future students who want to develop scalable Smart Campus Applications using microservices.

1.2 Report Outline

Overall Outline

The main objective of this report is to provide future students with sufficient documentation in order to help them with the initial learning curve when attempting to build on top of an existing project such as this one.

Progress on the project will be halted from March 13th until May1st. This time will be used to focus on studying for exams. The report will therefore also assist with refreshing memory when taking up work on the project after the hiatus.

2 Literature Survey

Discusses the initial research for the chosen technologies.

3 Architecture

Discusses the interaction of the various components in more detail.

4 Project Planning

Discusses the project management strategy.

5 Progress to Date

Discusses the progress made in each week of the Spring semester. It heavily references from the repository on GitHub that stores all of the theses documentation.**[1]**

6 Bibliography

Listing of references used.

2 Literature Survey

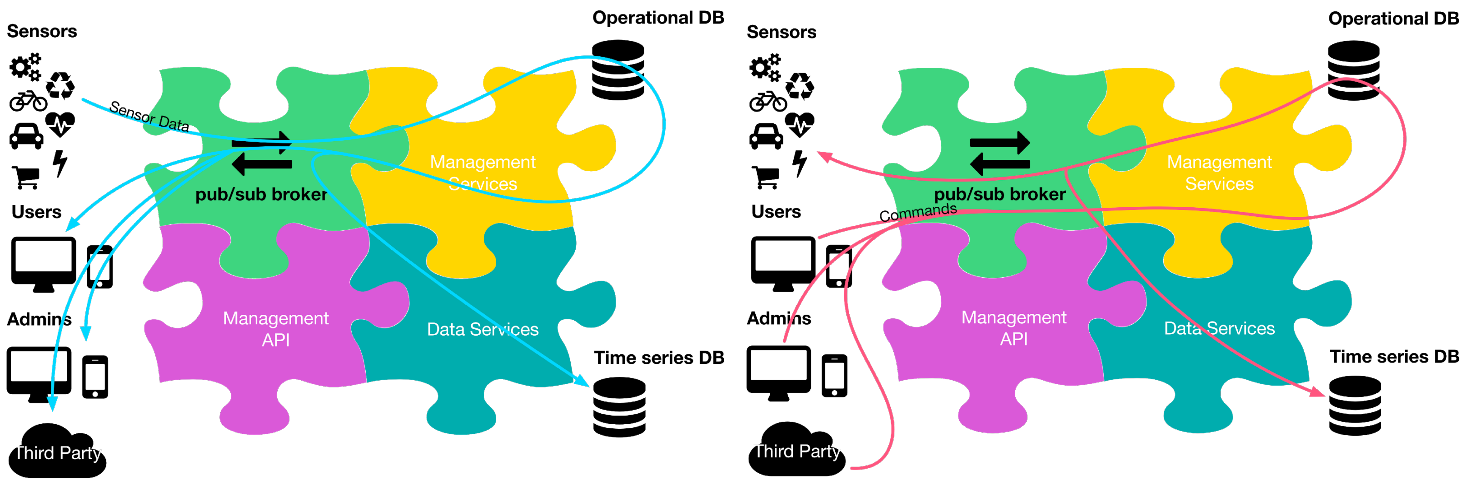
2.1 Section Outline

This section gives an overview of the content researched during the Spring semester. It starts with a high level discussion of IoT architecture. This will provide the reader with a perspective on both the design and scope of the project.

2.1.1 Section Layout

- Load balancing using HAProxy
- MQTT brokers
- Redis and Kafka
- **E**lastisearch, **L**ogstash, **K**ibana stack
- IBM's platform logging and monitoring solution
- A discussion on Container orchestration and the Docker compose tool
- A discussion on the problems associated with service discovery
- Amalgam8 microservices framework
- DevOps CI/CD pipeline
- A discussion on similar projects

2.2 IoT architecture



The main idea of the above architecture is to expose a pub/sub broker up front to the users and the devices. This design makes it is possible to direct the interaction between the users and the sensors, both for commands and data. This approach reduces latency, and allows for offline handling of time-series data and analytics.**[2]**

2.2.1 Sensor API

A sensor API, which is called by the sensors to deliver data readings and receive commands.

2.2.2 Public API

A public API, which is called by the sensors to retrieve real-time data, historical data, and to manage the devices.

2.2.2 Operations Services

A set of operational services, which are responsible for authentication and authorization, among other things; these services manage their own database.

2.2.3 Data Services

A set of data services, which are responsible for storing and analyzing the data, either in real time or offline.

2.3 Load Balancing

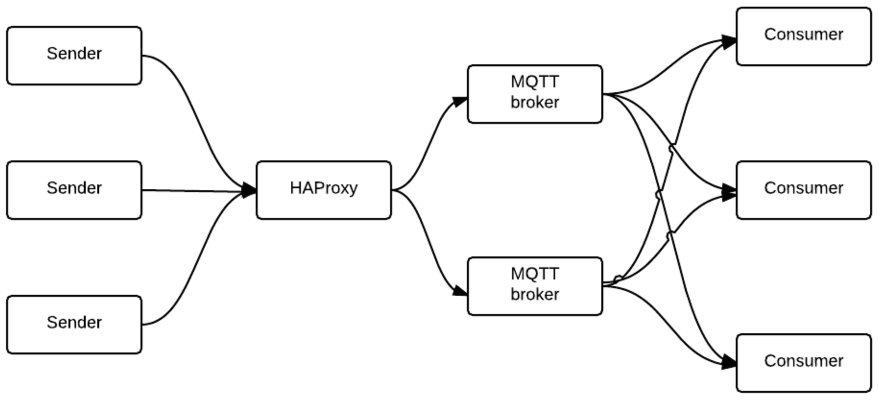
2.3.1 HAProxy

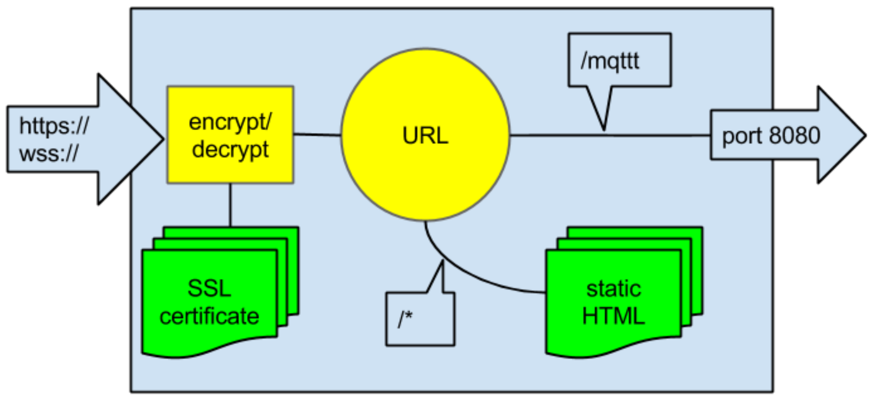
HAProxy**[3]** is an open source software load balancer for both layer four and layer seven. Written in C and runing on top of Linux, HAProxy uses an event-driven model to reduce memory usage and gain high performance. HAProxy also uses a single-buffering technique to reduce cost memory copy.

Although HAProxy is high performance, its design goal is to handle a large number of concurrent connections. It can only load balance traffic at the flow level and is unable to vertically divide a single elephant TCP flow into multiple mice flows**[4]**. HAProxy creates a session for each connection. A session consists of two TCP connections, one from the client to the load balancer and one from the load balancer to the server. The user needs to specify the load balancing policy in a config file before starting HAProxy.

Reasons for using MQTT over Websockets

- Web apps (those running in a browser - e.g. written in JavaScript)
- Any other applications that don't want to use the 1883/8883 port and want to go over HTTP/HTTPS instead - this could be so that there is less of a chance of being blocked by a firewall, as most firewalls will let HTTP traffic through





2.4 MQTT Brokers and Clients

2.4.1 Mosca

Mosca**[5]** sits between your system and the devices. It is written in Javascript so node.js is required to run it. Mosca is embeddable within any node application, allowing complete customization and the implementation of the pub-sub architecture.

Mosca Features

- MQTT 3.1 and 3.1.1 compliant.
- QoS 0 and QoS 1.
- Various storage options for QoS 1 offline packets, and subscriptions.
- Usable inside any other Node.js app.

https://github.com/mcollina/mosca

2.4.2 RabbitMQ

RabbitMQ**[6]** is a messaging broker - an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

RabbitMQ Features

- Persistence, delivery acknowledgements, publisher confirms, and high availability.
- Several RabbitMQ servers on a local network can be clustered together, forming a single logical broker.
- RabbitMQ supports messaging over a variety of messaging protocols.

2.4.3 MQTT.js Client Library

MQTT.js**[7]** is a client library for the MQTT protocol, written in JavaScript for node.js and the browser. MQTT.js is an OPEN Open Source Project. v2.0.0 removes support for node v0.8, v0.10 and v0.12, and it is 3x faster in sending packets. The new Client improves performance by a 30% factor, embeds Websocket support, and has better support for QoS 1 and 2.

2.5 Redis

Redis**[8]** is an open source, BSD licensed, advanced key-value cache and store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets, sorted sets, bitmaps and hyperlogs.

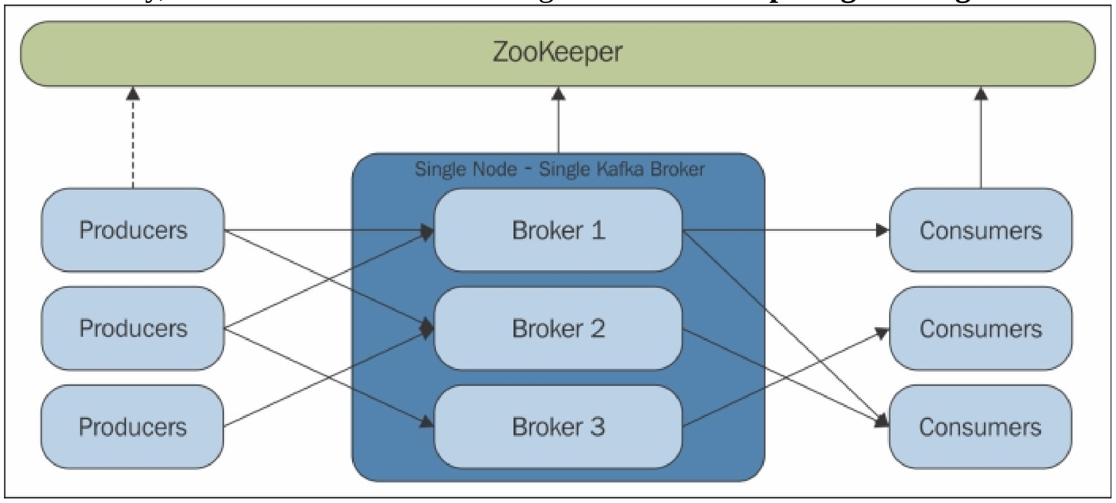
Redis is a bit different from Kafka in terms of its storage and various functionalities. At its core, Redis is an in-memory data store that can be used as a high-performance database, a cache, and a message broker. It is perfect for real-time data processing.

Redis also has various clients written in several languages which can be used to write custom programs for the insertion and retrieval of data.**[9]** This is an advantage over Kafka since Kafka only has a Java client. The main similarity between the two is that they both provide a messaging service. But for the purpose of log aggregation, the various data structures of Redis do it more efficiently.

2.6 Kafka

Apache Kafka**[10]** is a distributed publish-subscribe messaging system. It is designed to support the following Persistent messaging with O(1) disk structures that provide constant time performance even with many TB of stored messages.

High-throughput: even with very modest hardware Kafka can support hundreds of thousands of messages per second. Explicit support for partitioning messages over Kafka servers and distributing consumption over a cluster of consumer machines while maintaining per-partition ordering semantics. Kafka provides parallelism in processing. More than one consumer from a consumer group can retrieve data simultaneously, in the same order that messages are stored. **http://logz.io/blog/kafka-vs-redis/**



2.7 ELK Stack

Elastisearch Logstash Kibana is an open source stack of three applications that work together to provide an end-to-end search and visualisation platform for the investigation of log file sources in real time. Log file records can be searched from a variety of log sources, charts, and dashboards built to visualize the log records. The three applications are Elasticsearch, Logstash, and Kibana.

2.7.1 Elasticsearch

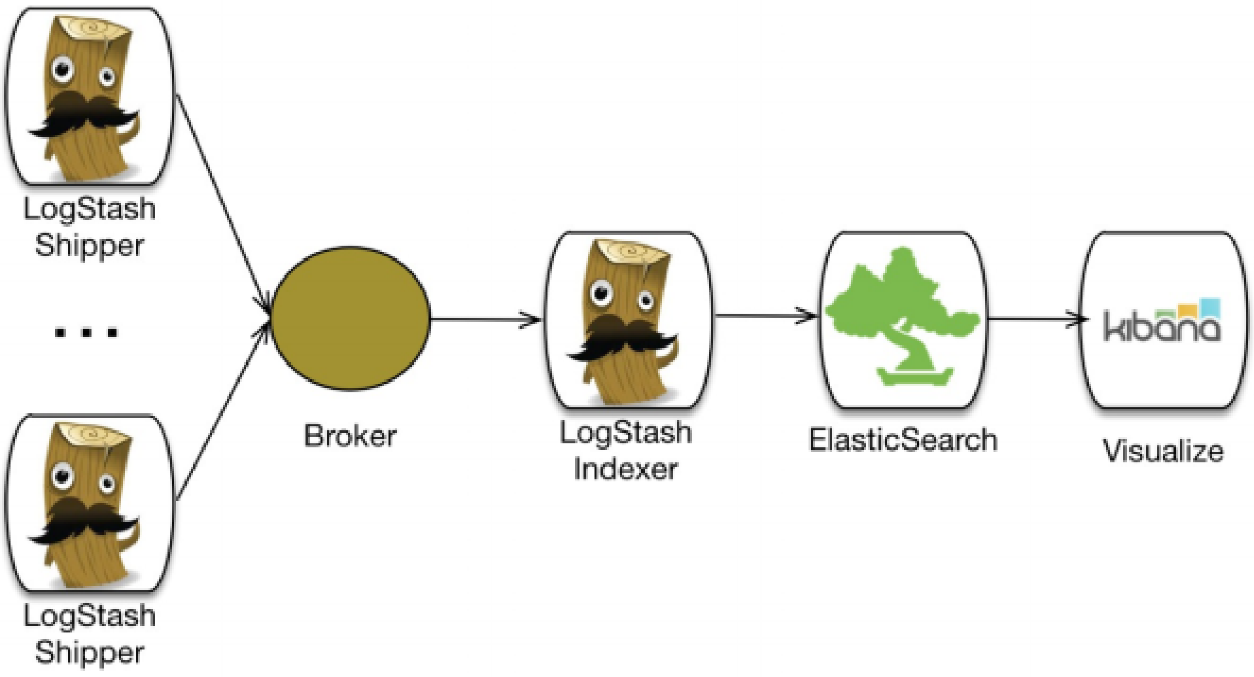
Elasticsearch**[11]** is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.

2.7.2 Logstash

Logstash**[12]** is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite “stash.”

2.7.3 Kibana

Kibana**[13]** lets you visualize your Elasticsearch data and navigate the Elastic Stack.



2.8 IBM Bluemix Logging and Monitoring

IBM's platform logging and monitoring solution runs on robust and popular open source offerings today. Logging is delivered from the Elasticsearch ELK stack. Monitoring is delivered from a Graphite and Grafana stack. Both are configured to use a common high-speed Kafka bus.

When data is collected from IBM Bluemix Container Service, users benefit from the use of a system crawler. The system crawler automatically collects select compute instance-related logs and metrics. Additionally, the crawler can be configured to collect more, all without agents or sidecar configurations.

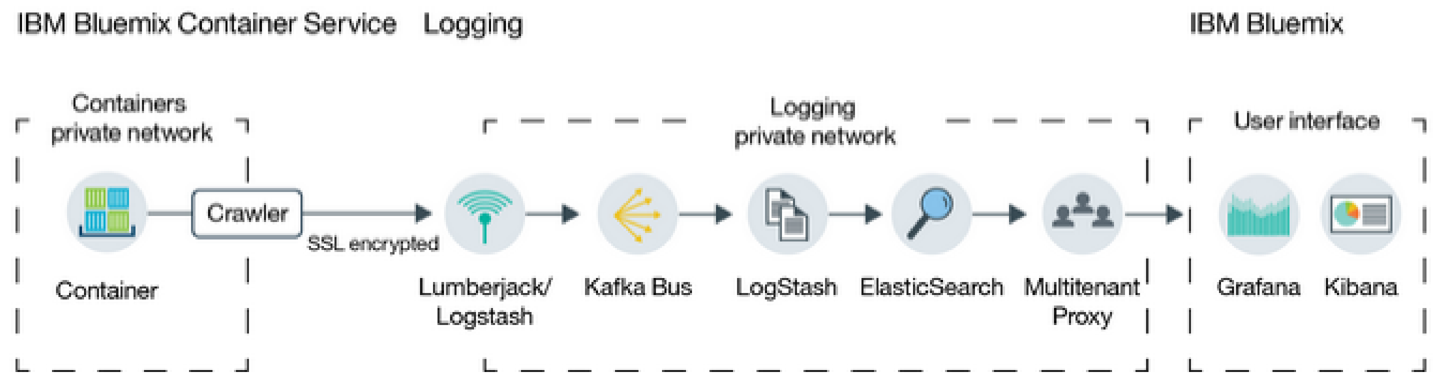
Data flows into the logging and monitoring service through an initial Logstash connection point. After the data is in the private network, the data is processed then stored in Elastic Search or Graphite. Users gain access to the data through the Multitenant Proxy.**[14]**

2.8.1 Monitoring Overview

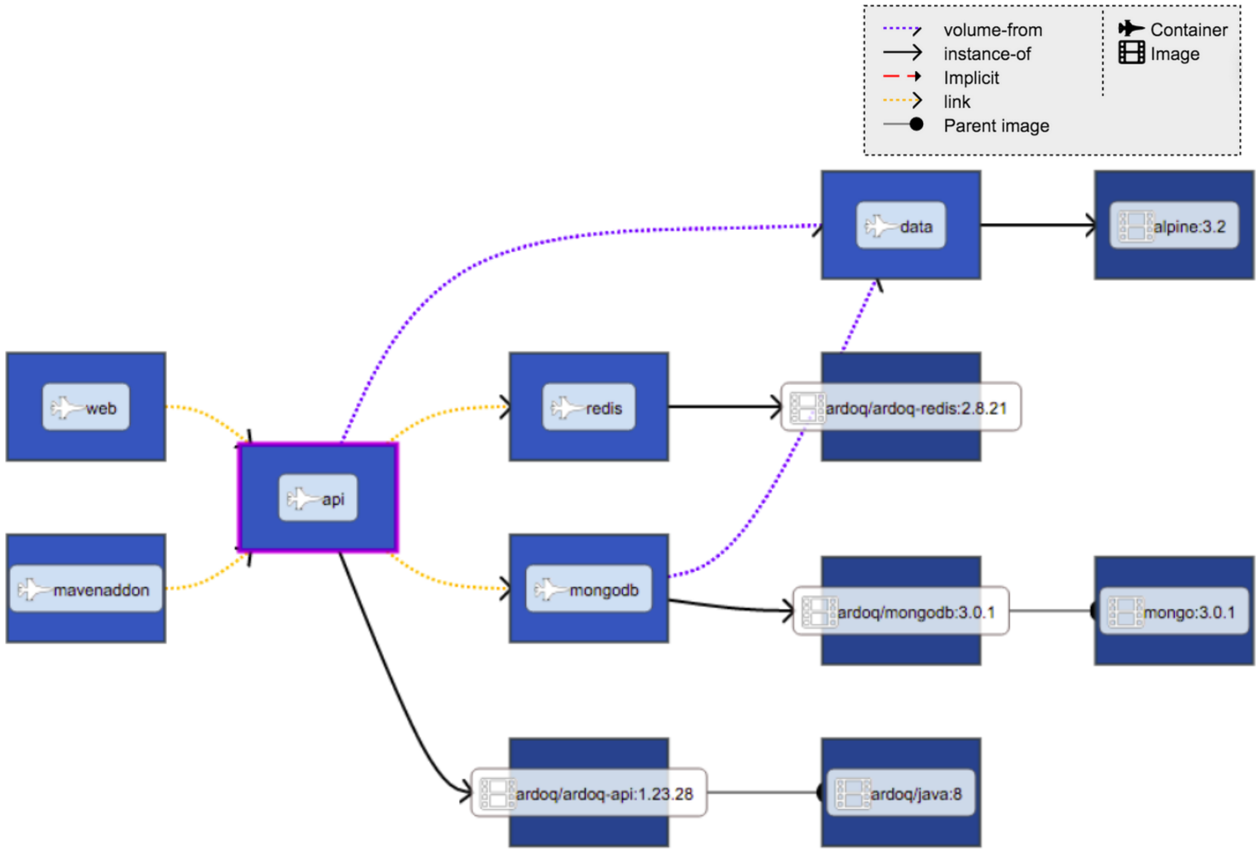
Container metrics are collected from outside of the container, without having to install and maintain agents inside of the container. In-container agents can have significant overheads and setup times for short-lived, lightweight cloud instances and auto-scaling groups, where containers can be rapidly created and destroyed. This out-of-band data collection approach eliminates these challenges and removes the burden of monitoring from the users.

2.8.2 Logging Overview

Similar to metrics, container logs are monitored and forwarded from outside of the container by using crawlers. The data is sent by the crawlers to a multi-tenant Elasticsearch in Bluemix, just like logs that are collected by other in-container agents, but without the hassle of having to install the agents inside the container.



2.9 Container Orchestration



Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.

Compose is great for development, testing, and staging environments, as well as CI workflows.

Using Compose is basically a three-step process.

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Lastly, run `docker-compose up` and Compose will start and run your entire app.

The real advantage of Compose is for applications that revolve around a single-purpose server that could easily scale out if architectural complexity is not a requirement. Development environments, which tend to use an all-in-one method of operation, fit well into this requirement.

The community's reception of Compose has been notably positive, but the practicality of its usage and the lack of ability to create a long-term vision around it tend to minimize the actual legitimacy of adopting it as a container orchestration technology.**[15]**

2.10 Service Discovery

Services typically need to call one another. In a monolithic application, services invoke one another through language-level method or procedure calls. In a traditional distributed system deployment, services run at fixed, well known locations (hosts and ports) and so can easily call one another using HTTP/REST or some RPC mechanism. However, a modern microservice based application typically runs in a virtualized or containerized environments where the number of instances of a service and their locations changes dynamically.**[16]**

Problem

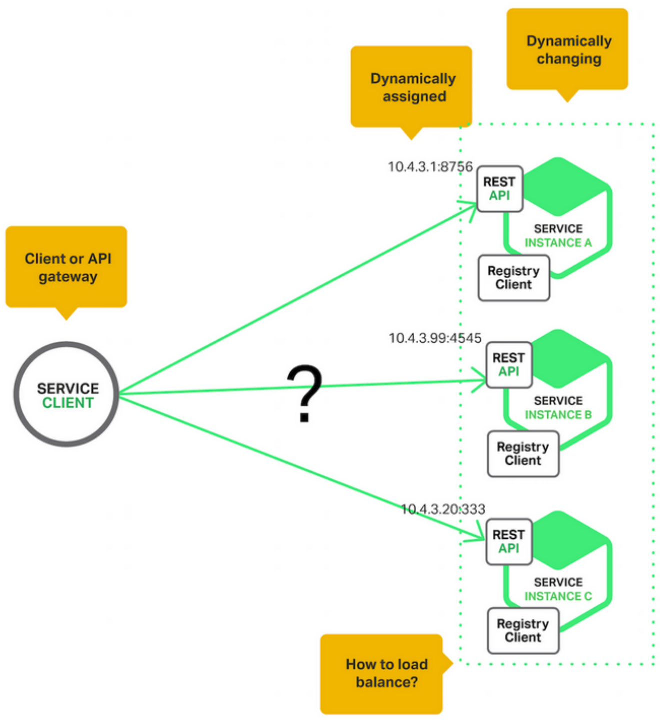
How does the client of a service - the API gateway or another service - discover the location of a service instance?

Forces

- Each instance of a service exposes a remote API such as HTTP/REST, or Thrift etc. at a particular location (host and port)
- The number of services instances and their locations changes dynamically.
- Virtual machines and containers are usually assigned dynamic IP addresses.
- The number of services instances might vary dynamically. For example, an EC2 Autoscaling Group adjusts the number of instances based on load.

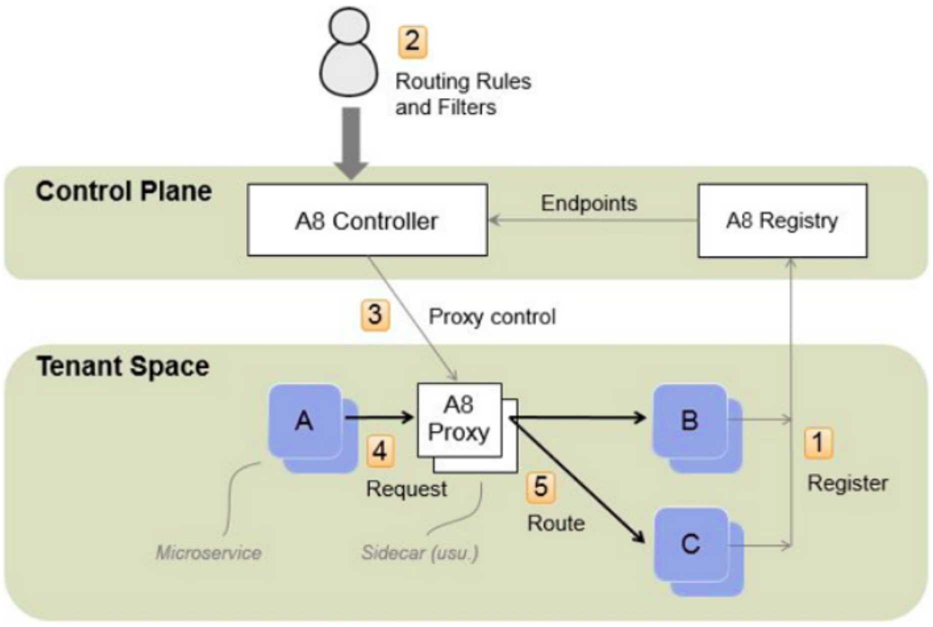
Solution

When making a request to a service, the client makes a request via a router that runs at a well known location. The router queries a service registry, which might be built into the router, and forwards the request to an available service instance.



2.11 Amalgam8

IBM has made available an open source project named Amalgam8**[17]** which takes the tedium out of microservice management, enabling faster development, more control, and better resiliency of microservices without impacting existing implementation code. It provides advanced DevOps capabilities such as systematic resiliency testing, red/black deployment, and canary testing necessary for rapid experimentations and insight.



At the heart of Amalgam8 are two multi-tenanted services:

- ***Registry* –** A high-performance service registry that provides a centralized view of all the microservices in an application, regardless of where they are actually running.
- ***Controller* –** A tool that monitors the Registry and provides a REST API for registering routing and other microservice control-rules, which it uses to generate and send control information to proxy servers running within the application

Applications run as tenants of these two servers. They register their services in the Registry and use the Controller to manage proxies, usually running as sidecars of the microservices.

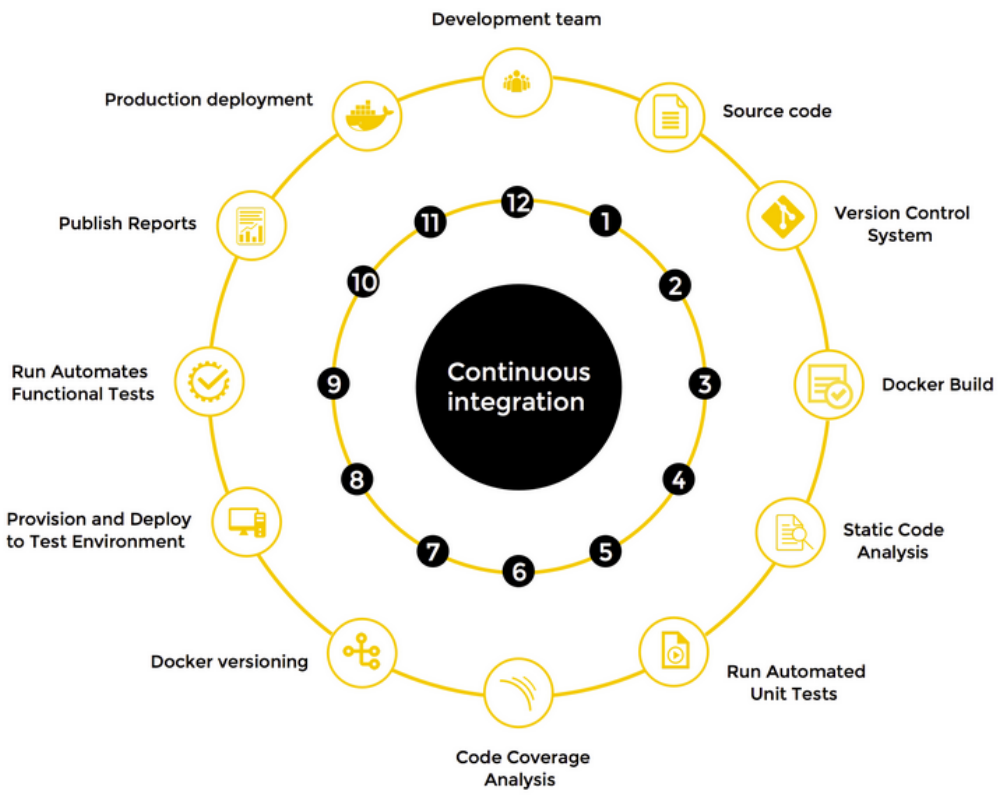
- Microservice instances are registered in the *A8 Registry*.
- An *Administrator* specifies routing rules and filters (such as version rules and test delays) to control traffic flow between microservices.
- An *A8 Controller* monitors the A8 Registry and administrator input, and then generates control information that is sent to the A8 Proxies.
- Requests to microservices are made through an *A8 Proxy* (usually a client-side sidecar of another microservice).
- The A8 Proxy forwards requests to an appropriate microservice, depending on the request path and headers and the configuration specified by the controller.

2.12 DevOps CI/CD Pipeline

2.12.1 Continuous Integration

Continuous Integration *(CI)* is a strategy for how a developer can integrate code to the mainline continuously - as opposed to frequently.

CI was created for agile development. It organizes development into functional user stories. These user stories are put into smaller groups of work, sprints. The idea of continuous integration is to find issues quickly, giving each developer feedback on their work and Test Driven Development (TDD) evaluates that work quickly. With TDD, you build the test and then develop functionality until the code passes the test. Each time, when you make new addition to the code, its test can be added to the suite of tests that are run when you build the integrated work. This ensures that new additions don't break the functioning work that came before them, and developers whose code does in fact "break the build" can be notified quickly.**[18]**



2.12.2 Continuous Delivery

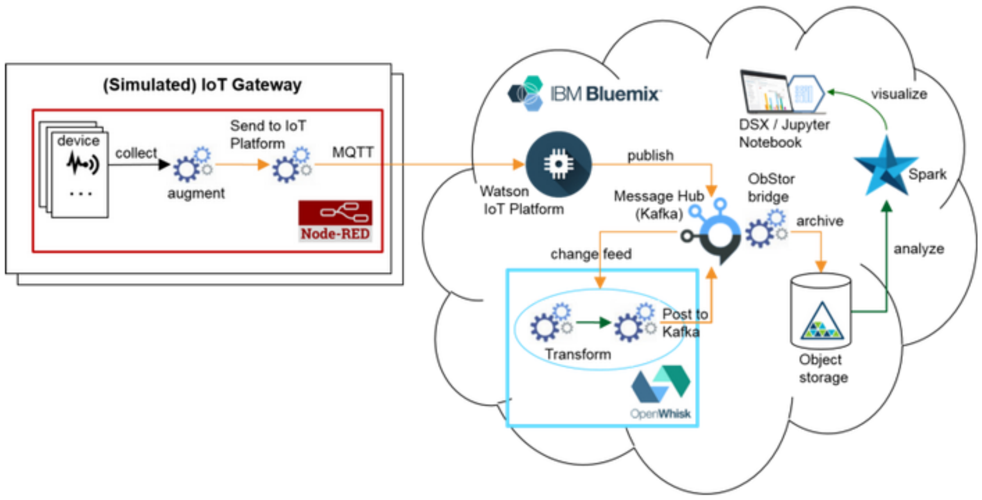
Continuous Delivery (CD) is a core technique that stems from Continuous Integration (CI), where as per business needs, quality software is deployed frequently and predictably to Production in an automated fashion. This improves feedback and reduces shelf-time of new ideas, and thereby improves the sustainability of businesses. Continuous Delivery includes Continuous Deployment and Continuous Testing.

2.13 Similar Projects

2.13.1 TRANSIT: Flexible pipeline for IoT data with Bluemix and OpenWhisk

This Blog post**[19]** was the first tutorial found that helped explain how to interface Node-Red with OpenWhisk. The proposed architecture uses the Watson IoT Platform and Message Hub in order to handle MQTT client messages. These messages are published to a Kafka server which in turns uses OpenWhisk to decode the base64 encoded data. The data is stored using IBM Object Storage which is then sent to Apache Spark for analytics.

Although this article helped with understanding IoT simulation using Node-Red, the architecture of this theses however is aimed more towards using Containers to handle the load balancing of MQTT clients.



2.13.2 How to Build a High Availability MQTT Cluster for the Internet of Things

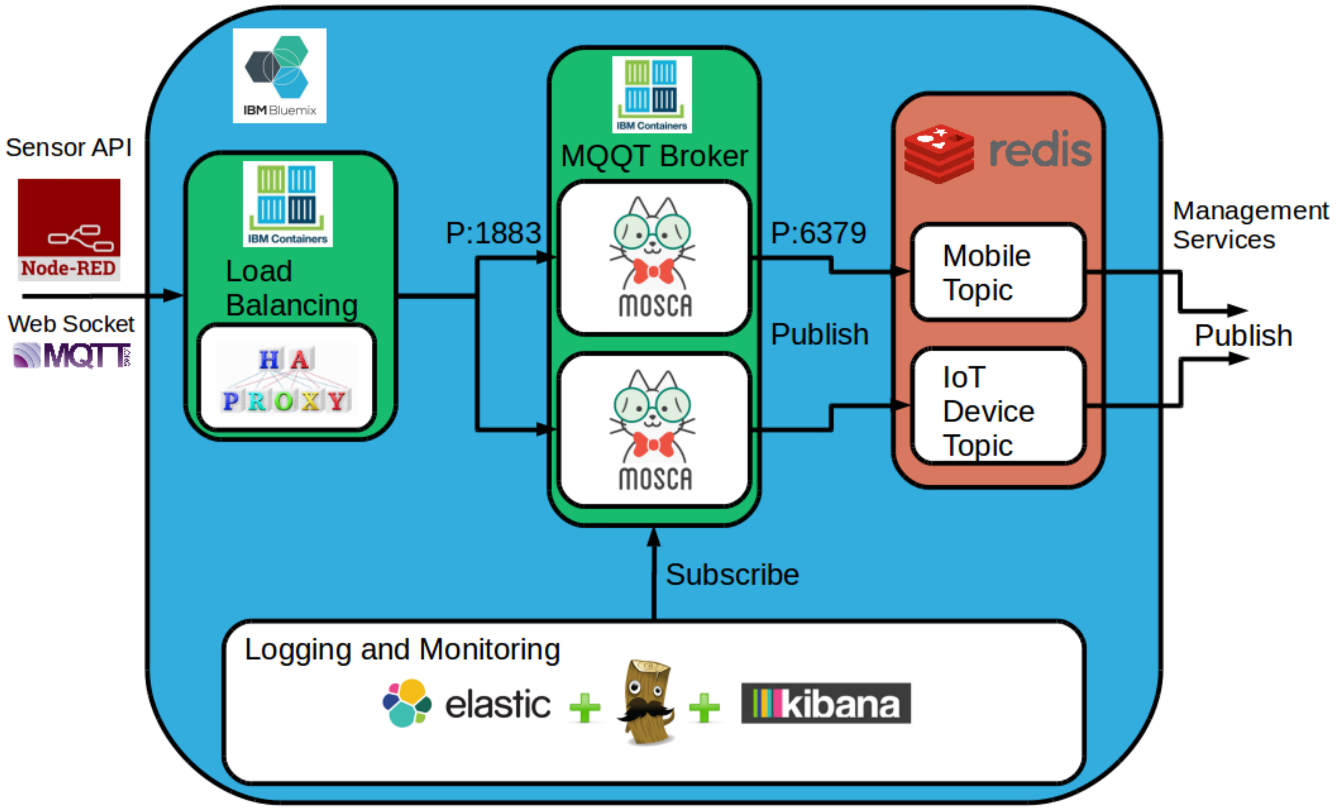
A Significant amount of the proposed architecture for this project is borrowed from this blog post.**[20]** It details the following.

- Setting up a Mosca Broker with Redis
- Dockerizing the MQTT server
- Adding HAProxy as a load balancer
- Access Control List configuration
- Making MQTT secure with SSL

A more detailed summary of the problems encountered with this implementation will be discussed in section 3.

3 Architecture

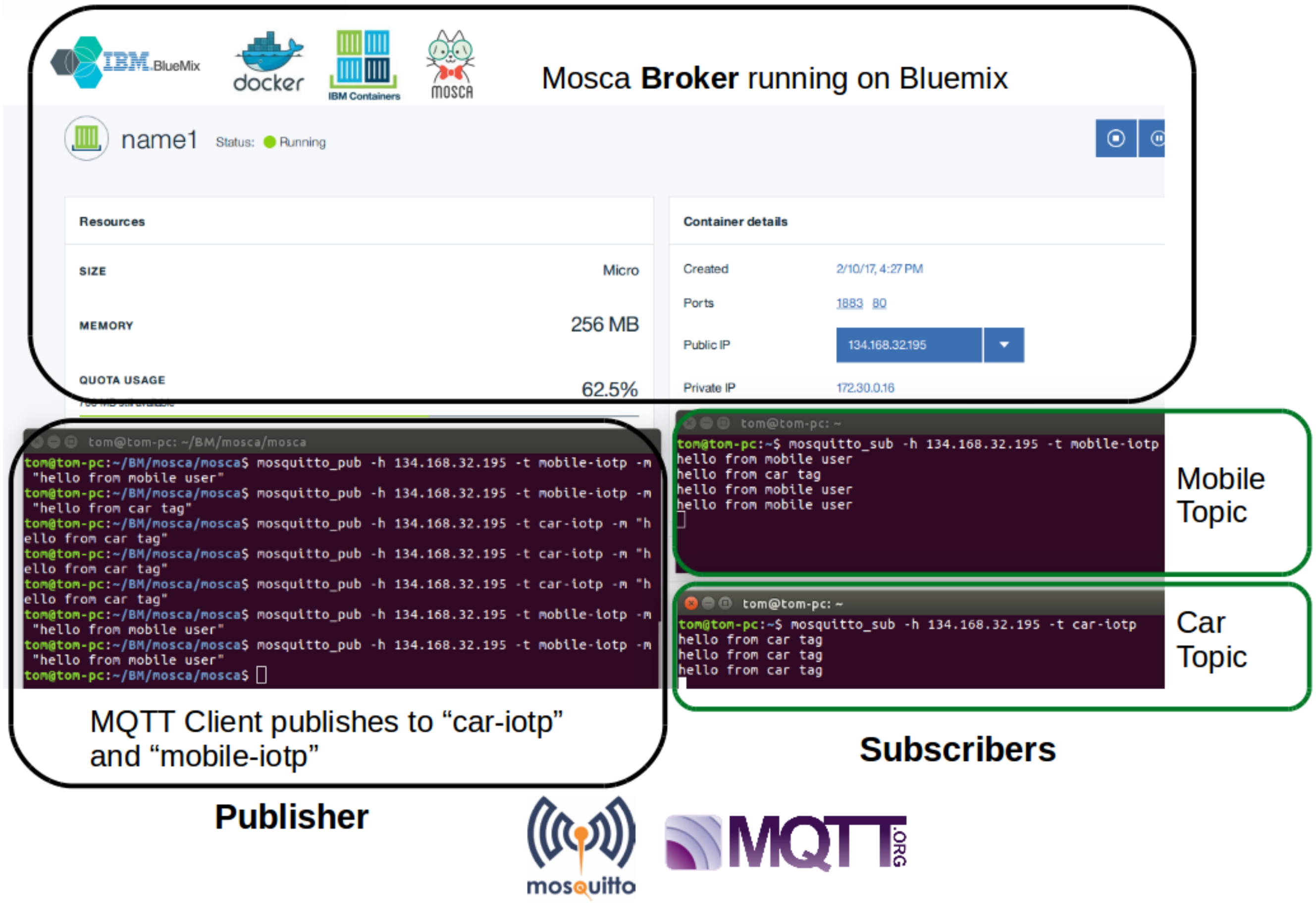
3.1 Load Balancing Diagram



The above figure shows the main architecture of the project. It describes the flow of data from left to right and the various components it interacts with. To give a brief summary, Node-RED simulates MQTT clients and injects messages to the HAProxy Container which then in turn load balances these messages between two Mosca brokers that publish to Redis for persistence and Logstash for monitoring. A more detailed description of this architecture follows after this subsection.

3.2 Mosca Broker Container

The Mosca image was pushed to the Bluemix image registry after testing the functionality of the Mosca Broker on the local Ubuntu machine. The test consisted of using the Mosquitto service installed on the local machine. Mosquitto verified the Mosca Broker's functionality by publishing and subscribing in different terminals.



```
tom@tom-pc: ~/BM/mosca/mosca
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t mobile-iotp -m
 "hello from mobile user"
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t mobile-iotp -m
 "hello from car tag"
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t car-iotp -m "h
ello from car tag"
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t car-iotp -m "h
ello from car tag"
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t car-iotp -m "h
ello from car tag"
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t mobile-iotp -m
 "hello from mobile user"
tom@tom-pc:~/BM/mosca/mosca$ mosquitto_pub -h 134.168.32.195 -t mobile-iotp -m
 "hello from mobile user"
tom@tom-pc:~/BM/mosca/mosca$
```

```
tom@tom-pc: ~
tom@tom-pc:~$ mosquitto_sub -h 134.168.32.195 -t mobile-iotp
hello from mobile user
hello from car tag
hello from mobile user
hello from mobile user
```

```
tom@tom-pc: ~
tom@tom-pc:~$ mosquitto_sub -h 134.168.32.195 -t car-iotp
hello from car tag
hello from car tag
hello from car tag
```

3.2.1 Broker

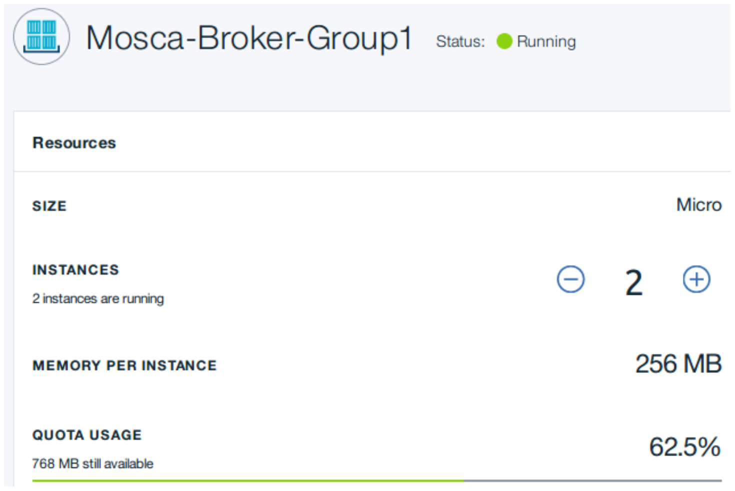
The top half of the above figure shows the Bluemix Dashboard. It displays the Mosca Broker running in a Container called "name1".

3.2.2 Mosquitto

The terminal on the left is using Mosquito to publish to the topics "mobile-iotp" and "car-iotp". These messages are being sent to the Mosca Broker Container running on Bluemix.

The terminals on the right are using Mosquito to subscribe to the mobile-iotp and car-iotp topics.

3.3 Mosca Broker Container Group


Mosca-Broker-Group1 Status: Running

Resources

SIZE Micro

INSTANCES 2
2 instances are running

MEMORY PER INSTANCE 256 MB

QUOTA USAGE 62.5%
768 MB still available

An IBM Container Scalable Group consists of multiple containers that all share the same image. The original architecture for this project heavily depended on using Container Groups for scaling out MQQT brokers. It turns out that IBM Containers Scalable Groups do not support non-HTTP traffic for the exposed ports. In a Scalable Group, an external URL is bound to the Go router serving the platform. HTTPS requests made to port 80 on the external URL are sent to the port specified during configuration on the internal container hosts. Direct external access to the ports on the external container is not allowed.

Further research into a solution to this problem is required. If Scalable Groups cannot be implemented for the Mosca Brokers then a manual Container Orchestration solution will have to be drawn up. This would be configured using the docker-compose CLI.

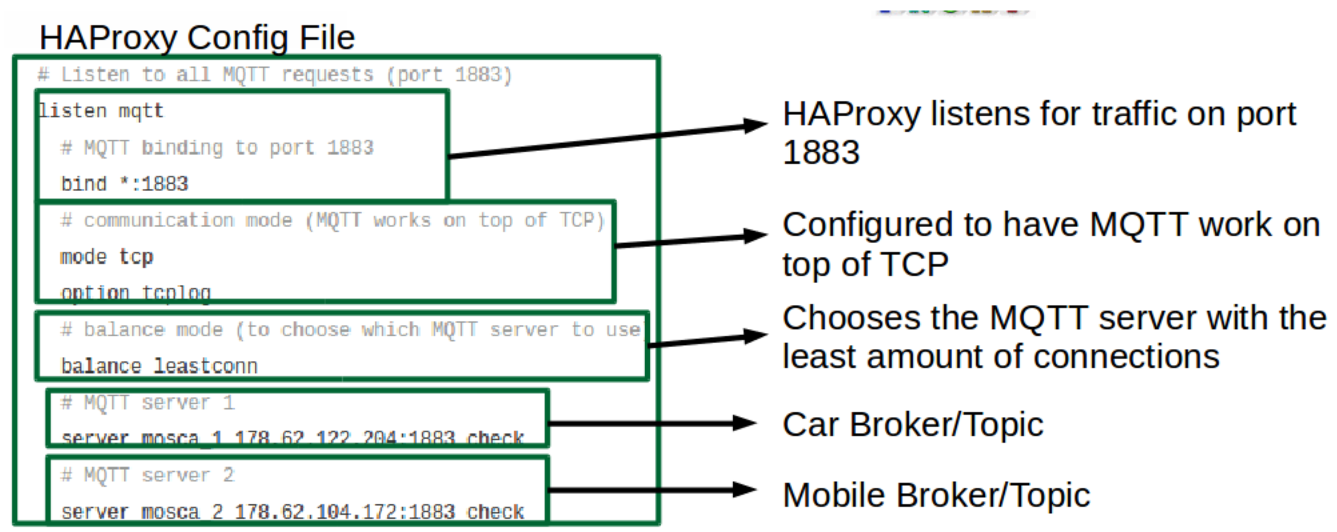
```
docker-compose scale broker=3
```

The problems that need to be solved with this approach.

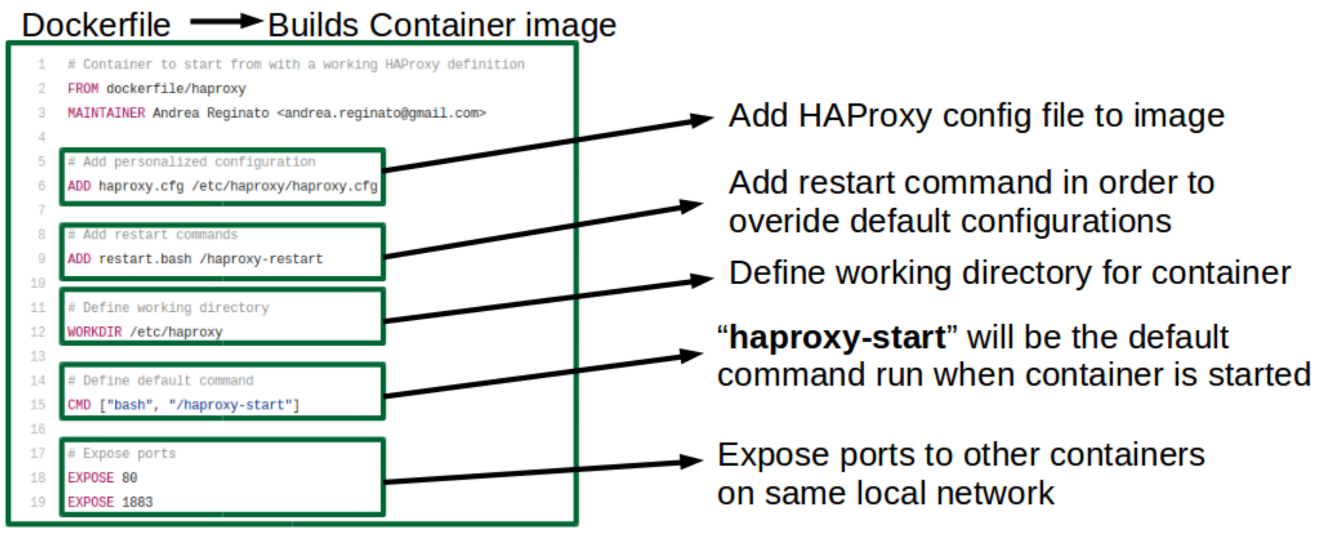
- Configuration of detecting “scale out” “scale in” triggers
- Configuring Docker Remote API
- Each Mosca Broker will need to be configured with the required ssl certs to make a call using the Docker Remote API in order to scale out or in.
- Figuring out how the Docker Remote API is integrated with Bluemix

3.4 HaProxy Configuration

Below is a HAProxy config file for load balancing between 2 Mosca Brokers. The HAProxy listens for all requests coming to port 1883 and forwards them to the MQTT servers (mosca_1 and mosca_2) using the *leastconn* balance mode (selects the server with the least number of connections).


HAProxy Config File
HAProxy listens for traffic on port 1883
Configured to have MQTT work on top of TCP
Chooses the MQTT server with the least amount of connections
Car Broker/Topic
Mobile Broker/Topic

```
# Listen to all MQTT requests (port 1883)
listen mqtt
  # MQTT binding to port 1883
  bind *:1883
  # communication mode (MQTT works on top of TCP)
  mode tcp
  option tcplog
  # balance mode (to choose which MQTT server to use
  balance leastconn
  # MQTT server 1
  server mosca_1 178.62.122.204:1883 check
  # MQTT server 2
  server mosca_2 178.62.104.172:1883 check
```


Dockerfile
Builds Container image
Add HAProxy config file to image
Add restart command in order to overide default configurations
Define working directory for container
"**haproxy-start**" will be the default command run when container is started
Expose ports to other containers on same local network

```
# Container to start from with a working HAProxy definition
FROM dockerfile/haproxy
MAINTAINER Andrea Reginato <andrea.reginato@gmail.com>

# Add personalized configuration
ADD haproxy.cfg /etc/haproxy/haproxy.cfg

# Add restart commands
ADD restart.bash /haproxy-restart

# Define working directory
WORKDIR /etc/haproxy

# Define default command
CMD ["bash", "/haproxy-start"]

# Expose ports
EXPOSE 80
EXPOSE 1883
```

The above figure shows the configuration for the dockerfile in order to build the haproxy image.

3.4.1 Problems Encountered

A common problem found with tutorials in relation to Docker is that the examples are quickly outdated. For example

FROM dockerfile/haproxy

needs to be changed to

FROM library/haproxy

After building a HAProxy container with the above Dockerfile. The command "**docker logs <haproxy container id>**" is run in order to determine if the build was successful. An error was reported in relation to the command

CMD ["bash", "/haproxy-start"]

The error states that the "haproxy-start" cannot be found. A quick fix to remove this error was to change the line to

CMD ["bash", "/haproxy-restart"]

This was done merely in a problem solving effort to remove the error to get the container to build.

Another problem encountered was attaching the "overide-config" file as a volume to the container. The process is as follows.

Build and run the container using the original haproxy image with the Dockerfile defined above.

docker pull library/haproxy

docker run -d -p 80:80 library/haproxy

Then issue the following command to the container once it is running.

docker run -d -p 1883:1883 -v <override-dir>:/haproxy-override dockerfile/haproxy

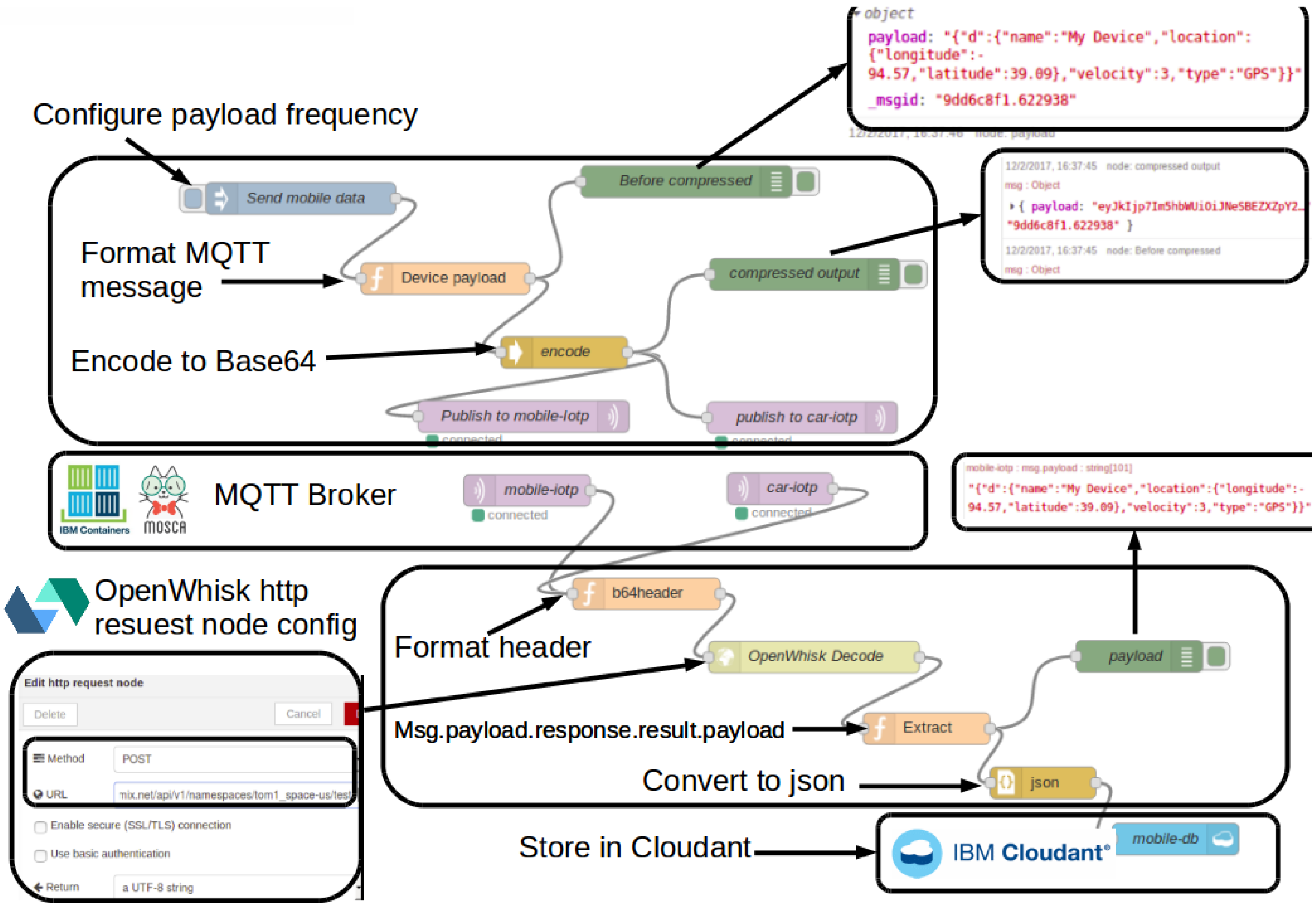
The HAProxy container accepts a configuration file as data volume option (-v), where *<override-dir>* is an absolute path of a directory that contains *haproxy.cfg*.

Other problems such as "cannot find user id for 'haproxy' " had to be solved as well in order to remove errors when debugging with the "**docker logs <haproxy container id>**" command.

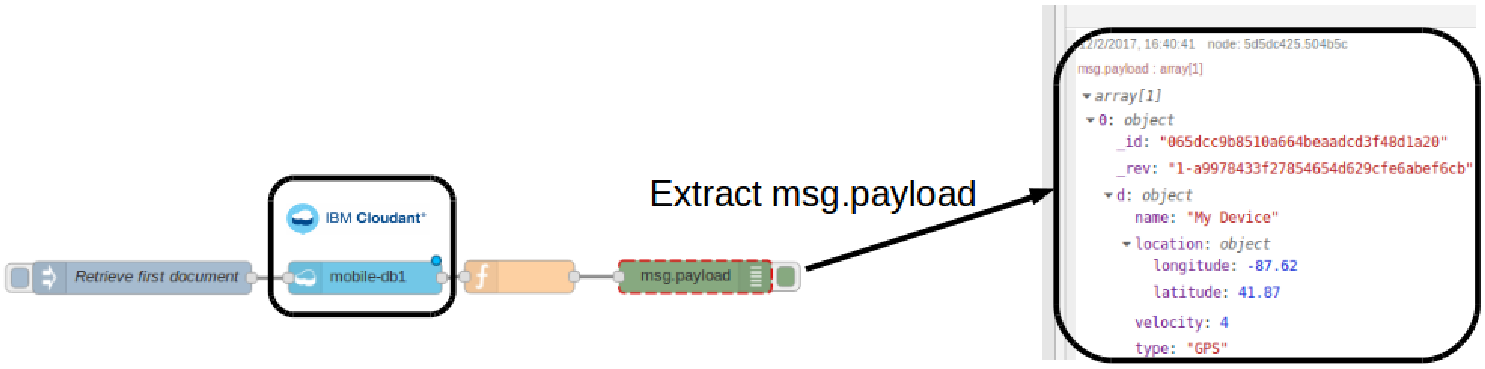
3.5 Node-Red

3.5.1 Primary Flow

This flow represents the the core functionality of the theses. Data is flowing from the simulated IoT gateway to the Mosca broker hosted on Bluemix. The data is then decoded using OpenWhisk and stored as JSON in a Cloudant database.

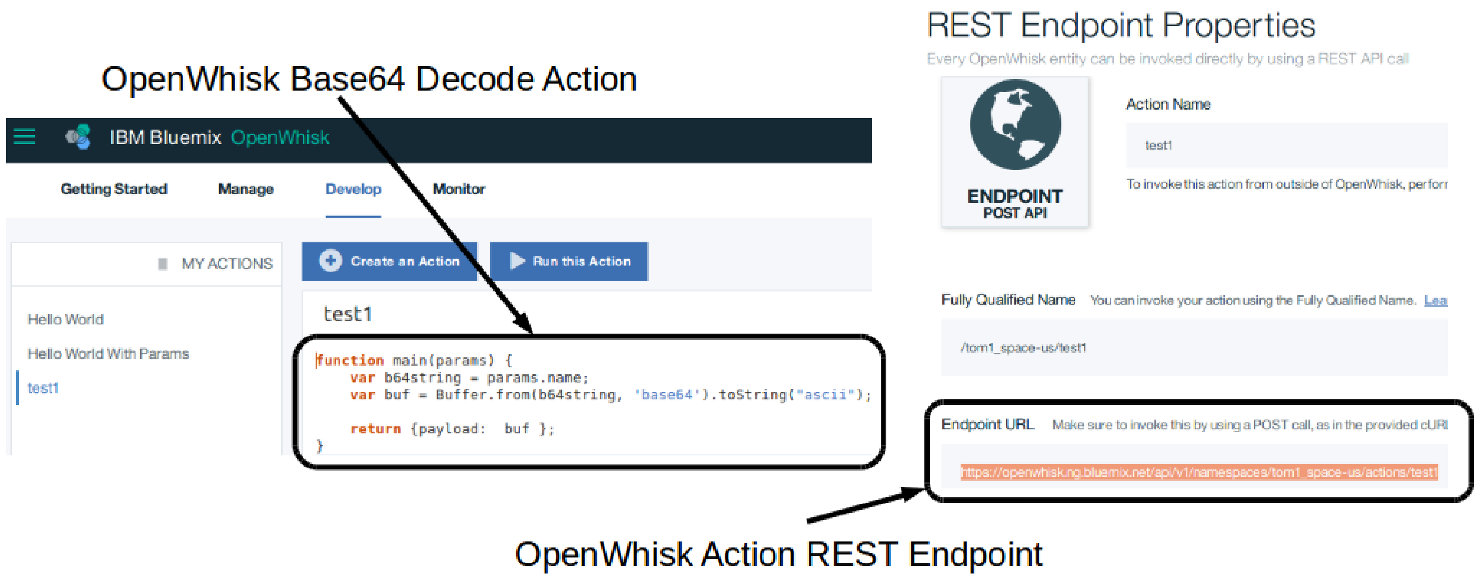

Configure payload frequency
Send mobile data
Format MQTT message
Device payload
Encode to Base64
encode
Before compressed
compressed output
Publish to mobile-Iotp
publish to car-iotp
connected
object
payload: "{"d":{"name":"My Device","location":{"longitude":-94.57,"latitude":39.09},"velocity":3,"type":"GPS"}}"
_msgid: "9dd6c8f1.622938"
12/2/2017, 16:37:45 node: compressed output
msg : Object
{ payload: "eyJkIjp7ImShbWUiOiJNeSBEZXZpY2…
"9dd6c8f1.622938" }
12/2/2017, 16:37:45 node: Before compressed
msg : Object
IBM Containers
MOSCA
MQTT Broker
mobile-iotp
car-iotp
connected
mobile-iotp : msg.payload : string[101]
"{"d":{"name":"My Device","location":{"longitude":-94.57,"latitude":39.09},"velocity":3,"type":"GPS"}}"
OpenWhisk http resuest node config
Edit http request node
Delete
Cancel
Method
POST
URL
nix.net/api/v1/namespaces/tom1_space-us/test
Enable secure (SSL/TLS) connection
Use basic authentication
Return
a UTF-8 string
Format header
b64header
OpenWhisk Decode
payload
Msg.payload.response.result.payload
Extract
Convert to json
json
Store in Cloudant
IBM Cloudant
mobile-db

3.5.2 Test Cloudant Flow



3.6 OpenWhisk

3.6.1 Base64 Decode Action Configuration



```
function main(params) {
    var b64string = params.name;
    var buf = Buffer.from(b64string, 'base64').toString("ascii");

    return {payload:  buf };
}
```

Action Name: test1

To invoke this action from outside of OpenWhisk, perforr

Fully Qualified Name — You can invoke your action using the Fully Qualified Name. Lea

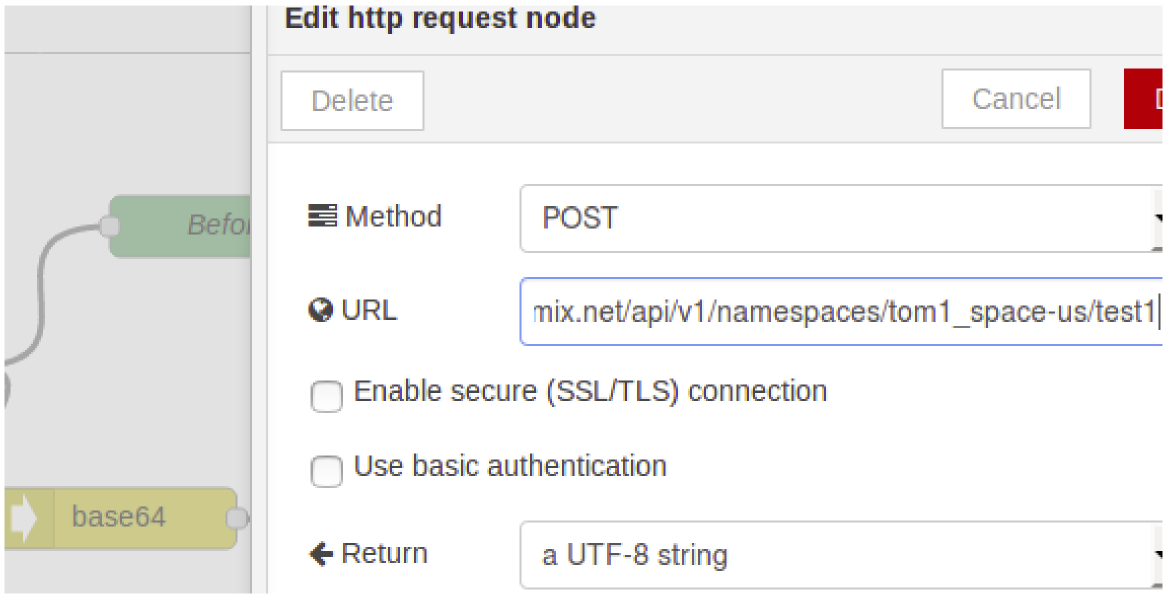
/tom1_space-us/test1

Endpoint URL — Make sure to invoke this by using a POST call, as in the provided cURl

https://openwhisk.ng.bluemix.net/api/v1/namespaces/tom1_space-us/actions/test1

3.6.2 Node-RED http post configuration

Node-RED has an OpenWhisk node available that makes it straightforward to call actions. Invoking the base64 decode action through the http request node was done as a quick fix for the problems encountered using the OpenWhisk node.



Method: POST

URL: mix.net/api/v1/namespaces/tom1_space-us/test1

Enable secure (SSL/TLS) connection

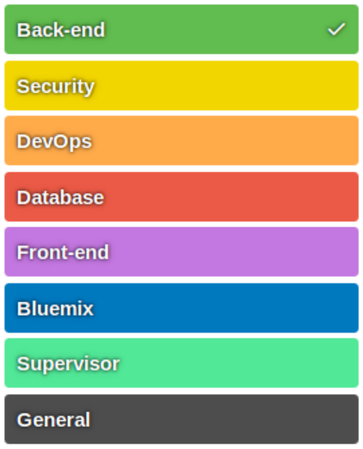
Use basic authentication

Return: a UTF-8 string

4 Project Management

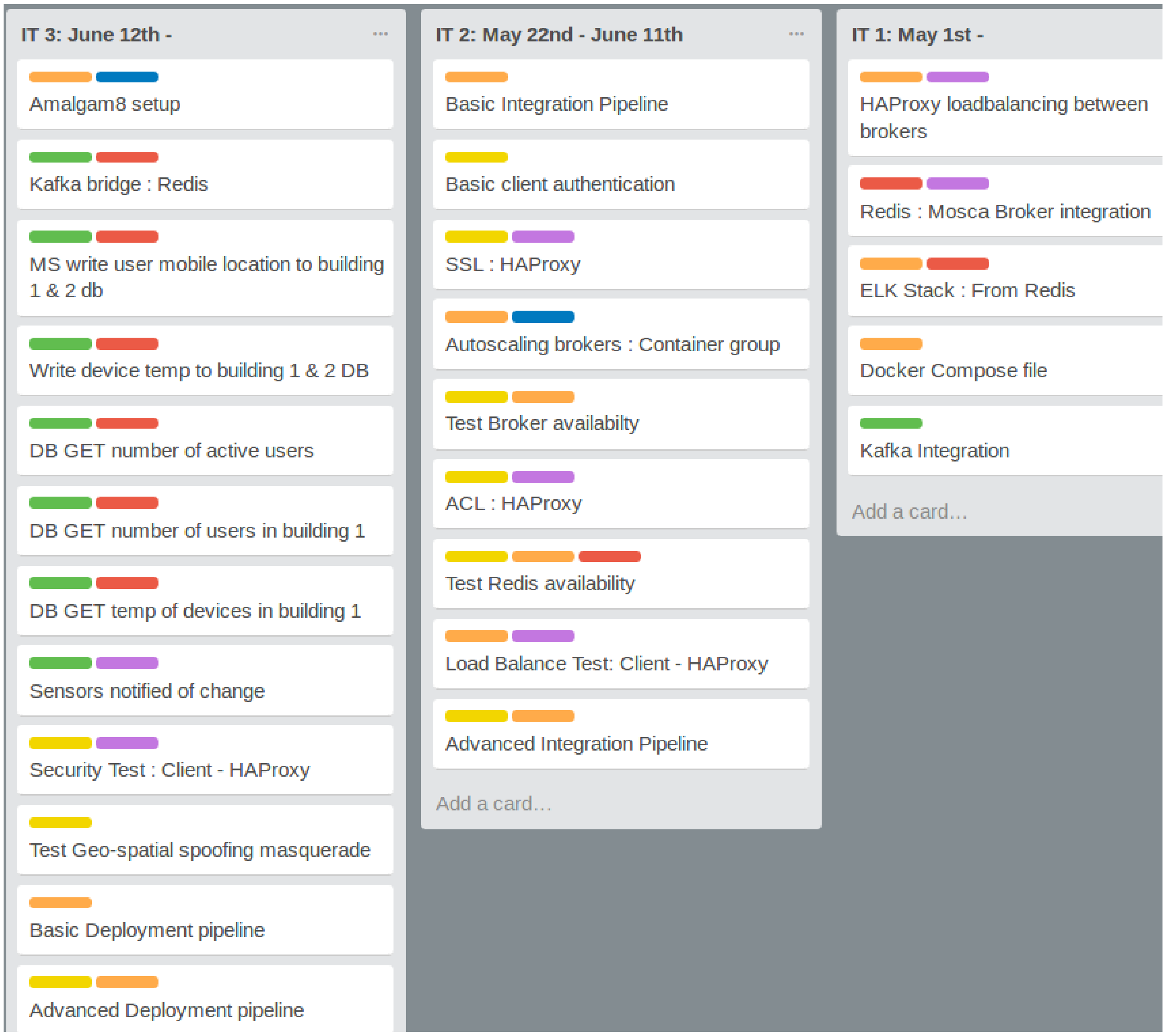
4.1 Trello Labels

Labels are used in order to identify the context of each task.



🟩 Back-end ✓
🟨 Security
🟧 DevOps
🟥 Database
🟪 Front-end
🟦 Bluemix
🟩 Supervisor
⬛ General

4.2 Project Iteration Planning



IT 3: June 12th - ···

🟧🟦 Amalgam8 setup
🟩🟥 Kafka bridge : Redis
🟩🟥 MS write user mobile location to building 1 & 2 db
🟩🟥 Write device temp to building 1 & 2 DB
🟩🟥 DB GET number of active users
🟩🟥 DB GET number of users in building 1
🟩🟥 DB GET temp of devices in building 1
🟩🟪 Sensors notified of change
🟨🟪 Security Test : Client - HAProxy
🟨 Test Geo-spatial spoofing masquerade
🟧 Basic Deployment pipeline
🟨🟧 Advanced Deployment pipeline

IT 2: May 22nd - June 11th ···

🟧 Basic Integration Pipeline
🟨 Basic client authentication
🟨🟪 SSL : HAProxy
🟧🟦 Autoscaling brokers : Container group
🟨🟧 Test Broker availabilty
🟨🟪 ACL : HAProxy
🟨🟧🟥 Test Redis availability
🟧🟪 Load Balance Test: Client - HAProxy
🟨🟧 Advanced Integration Pipeline
Add a card…

IT 1: May 1st -

🟧🟪 HAProxy loadbalancing between brokers
🟥🟪 Redis : Mosca Broker integration
🟧🟥 ELK Stack : From Redis
🟧 Docker Compose file
🟩 Kafka Integration
Add a card…

4.3 Iteration 1: May 1st – May 21st

The deliverable is focused towards load balancing between brokers. The lower priorities being the Redis and Logstash caching of data. It will be necessary to write a Docker Compose file that will orchestrate the various containers together.

4.4 Iteration 2: May 22nd – June 11th

With the core functionality implemented, work will be directed towards setting up a pipeline that will continuously test the code any time a change is made. Basic tests will then be written to test the authentication of devices.

With load balancing working successfully between two Mosca brokers, attention will be directed towards the problem of the Container Scalable Group. Specifically the problem of the group not being able to expose port 1883.

Security features such as SSL and ACL will be implemented for the HAProxy Container.

Load balance testing will be carried out on the HAProxy, Mosca and Redis Containers. At this point the continuous integration pipeline will need to be reviewed.

4.5 Iteration 3: June 12th – July 2nd

With a solid foundation setup for IoT and testing, focus will be directed towards Amalgam8 which provides capabilities such as systematic resiliency testing, red/black deployment, and canary testing necessary for rapid experimentations and insight.

Microservices for storing and retrieving data based on location will be implemented using Amalgam8. The goal here is to provide future students with both an API and a DevOps tool that they can work with in order to create smart campus applications for the university.

A brief investigation into Geospatial location spoofing.

Finally an investigation into how continuous deployment can be achieved.

4.6 Iteration 4: July 3rd – July 23rd

Deliverable will be a demonstration of canary testing and red/black deployment using Amalgam8.

5 Progress to Date

5.1 Christmas Break Summary

The work completed during the Christmas break proved to be somewhat wasteful. More time should have been spent on topics such as load balancing, MQQT client simulation/authentication, brokers, pub/sub protocols.

A basic front-end application based on Angular.js was specified in the Autumn report. The application was added to the scope of the project in order to make the final product more demonstrable. The original application was specified to have a basic web page that would display a map of the UL campus with the location of the active users of the mobile application. This lead to the discovery of the Ionic framework which provides a foundation for developing a mobile application using Angular.js. After returning from the Christmas break, the project supervisor advised against taking this new direction as it dramatically increased the scope of the project far beyond the time allotted.

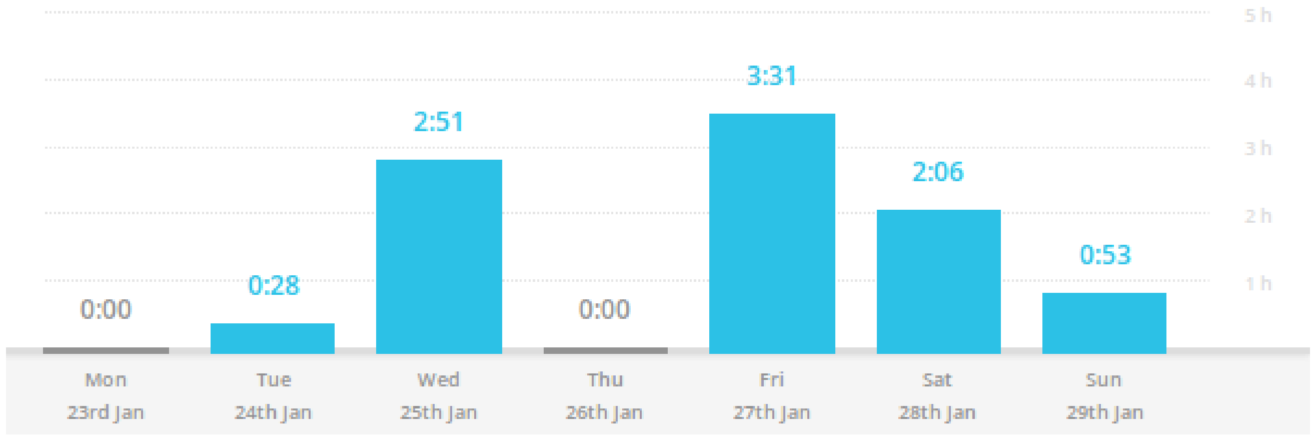
The other half of the work completed revolved around the area of DevOps. This consisted of learning the basics of the Docker CLI as well as networking/orchestrating Containers using the Docker Compose tool.

The Jenkins build tool was studied for continuous integration, it is a DevOps tool for automating repetitive tasks. A tutorial was completed on how to implement a “web hook” to a GitHub code repository. Any changes to the code in the repository triggered a “Jenkins job” which simply ran a Bash script that echoed “hello world”. It turns out Bluemix provides everything a basic DevOps Engineer would need for a CI/CD pipeline, in fact the the IBM DevOps continuous delivery pipeline is using Jenkins behind the scenes. This removes the pains of installing and provisioning a Jenkins server.

In summary, the work done over the Christmas break provided the necessary skills and background knowledge to go about developing and designing the proposed architecture as specified in this Spring report.

5.2 Week 1 Log

5.2.1 Week 1 Time Bar Chart



5.2.2 Tasks Completed

- **Meet with supervisor [21]**
- **Report Template [21]**
- **Research MQTT [21]**
- **Research Kafka [21]**
- **Find similar project [21]**
- **Research databases [21]**

5.2.3 Week 1 Log Entries

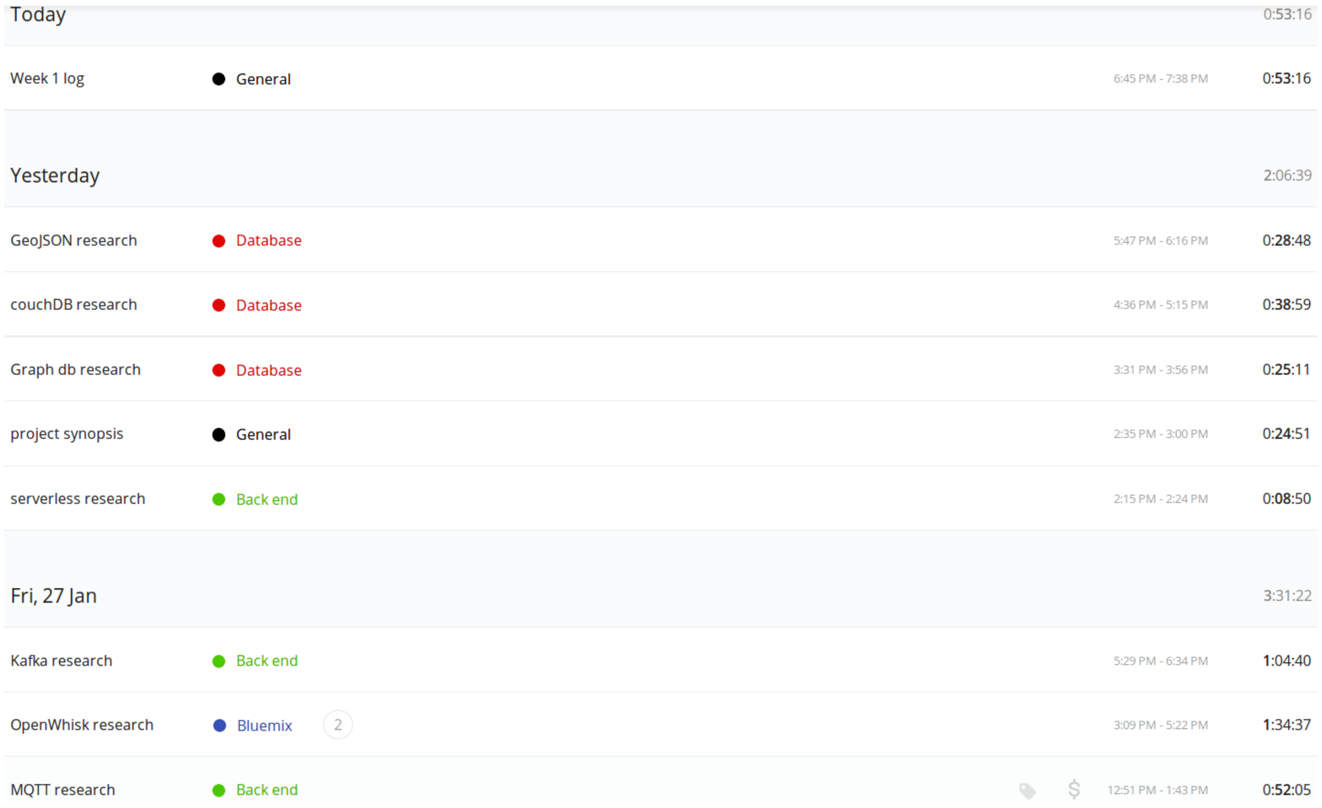
Entry 24/01/17:

“Today I met with my supervisor. We discussed various aspects of the project. My supervisor advised me to focus on the back-end architecture for now as the scope of the project is too large.”**[21]**

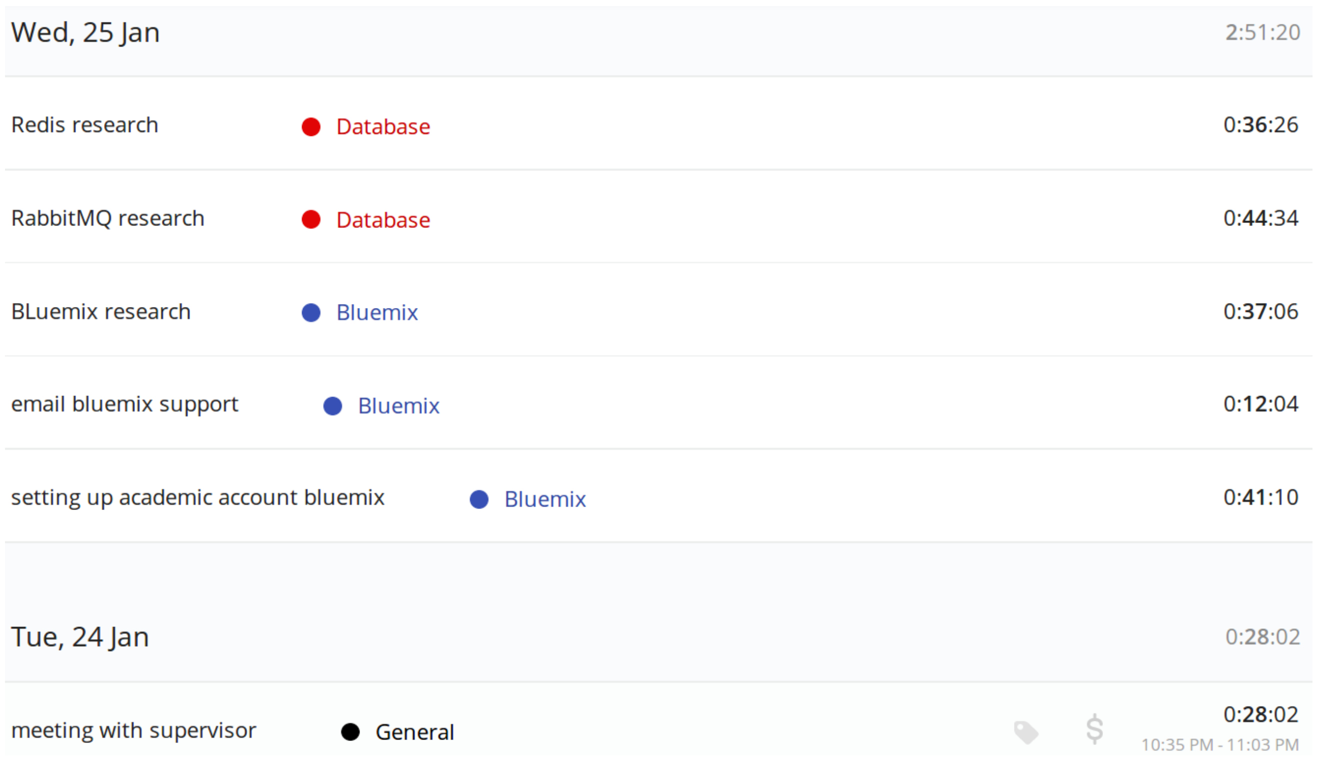
Entry 25/01/17:

“Today I ran into some problems setting up the academic free trial. I had to email the Bluemix support team in order to resolve the issue.”**[21]**

5.2.4 Week 1 Time Log



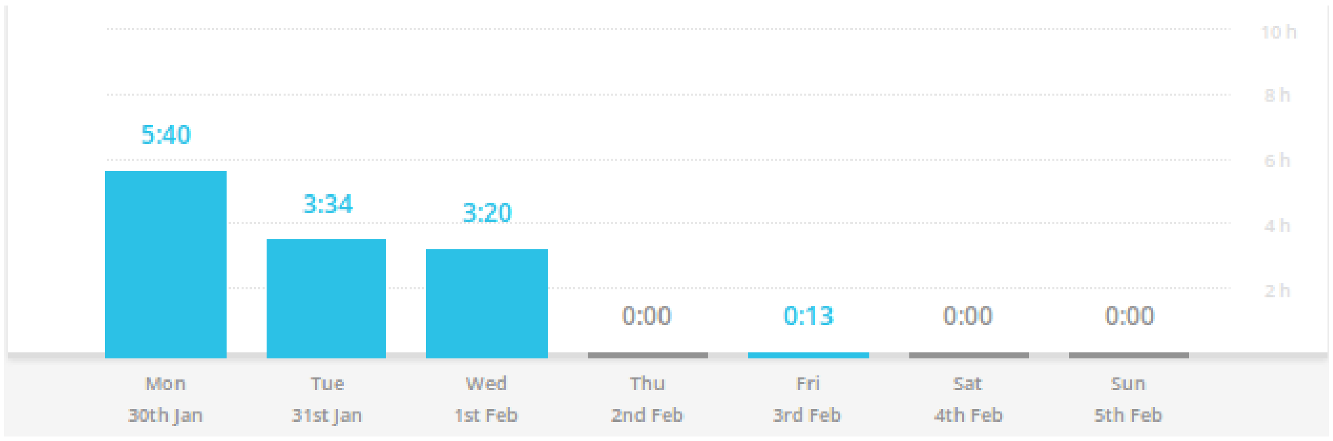
Today			0:53:16
Week 1 log	● General	6:45 PM - 7:38 PM	0:53:16
Yesterday			2:06:39
GeoJSON research	● Database	5:47 PM - 6:16 PM	0:28:48
couchDB research	● Database	4:36 PM - 5:15 PM	0:38:59
Graph db research	● Database	3:31 PM - 3:56 PM	0:25:11
project synopsis	● General	2:35 PM - 3:00 PM	0:24:51
serverless research	● Back end	2:15 PM - 2:24 PM	0:08:50
Fri, 27 Jan			3:31:22
Kafka research	● Back end	5:29 PM - 6:34 PM	1:04:40
OpenWhisk research	● Bluemix 2	3:09 PM - 5:22 PM	1:34:37
MQTT research	● Back end	12:51 PM - 1:43 PM	0:52:05



Wed, 25 Jan			2:51:20
Redis research	● Database		0:36:26
RabbitMQ research	● Database		0:44:34
BLuemix research	● Bluemix		0:37:06
email bluemix support	● Bluemix		0:12:04
setting up academic account bluemix	● Bluemix		0:41:10
Tue, 24 Jan			0:28:02
meeting with supervisor	● General	10:35 PM - 11:03 PM	0:28:02

5.3 Week 2 Log

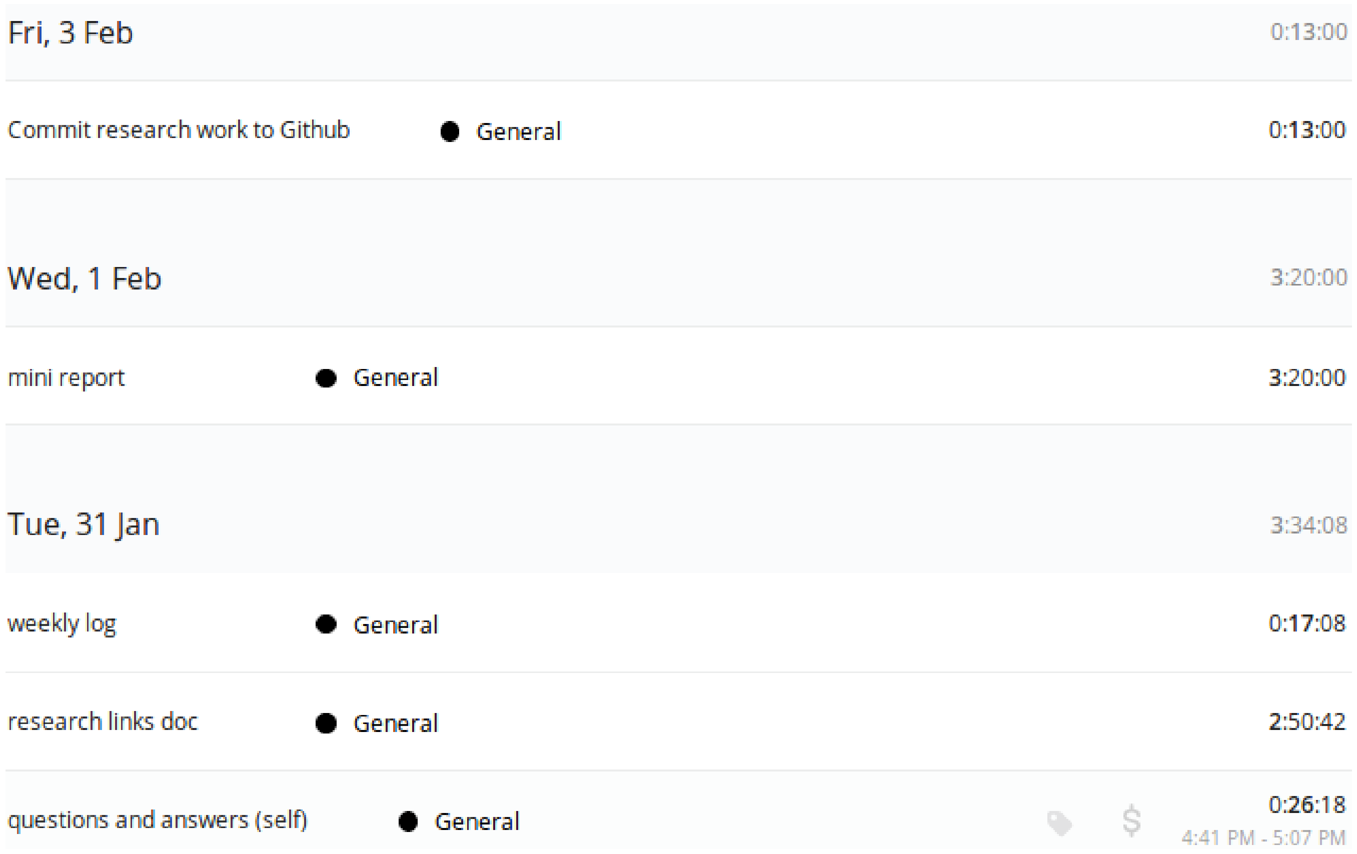
5.3.1 Week 2 Time Bar Chart



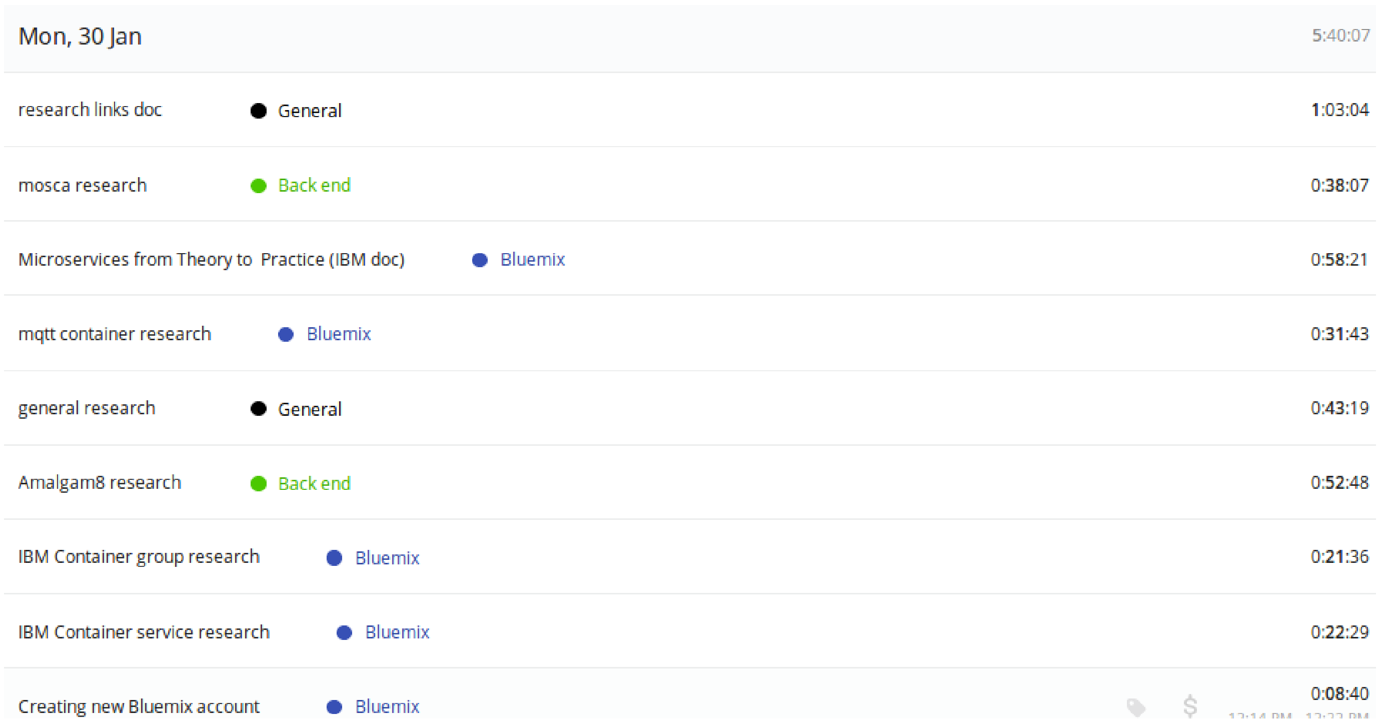
5.3.2 Tasks Completed

- **Creating new Bluemix account for academic free trial** [22]
- **IBM Container Service research** [22]
- **IBM Container group research** [22]
- **Amalgam8 research** [22]
- **General research (jumping between mqtt brokers and bluemix links)** [22]
- **MQTT container research** [22]
- **Microservices from Theory to Practice (IBM document)** [22]
- **Mosca research** [22]
- **Questions & answers (self evaluation)** [22]
- **Research links document for supervisor** [22]
- **(rough) architectural diagram** [22]

5.3.3 Week 2 Time Log



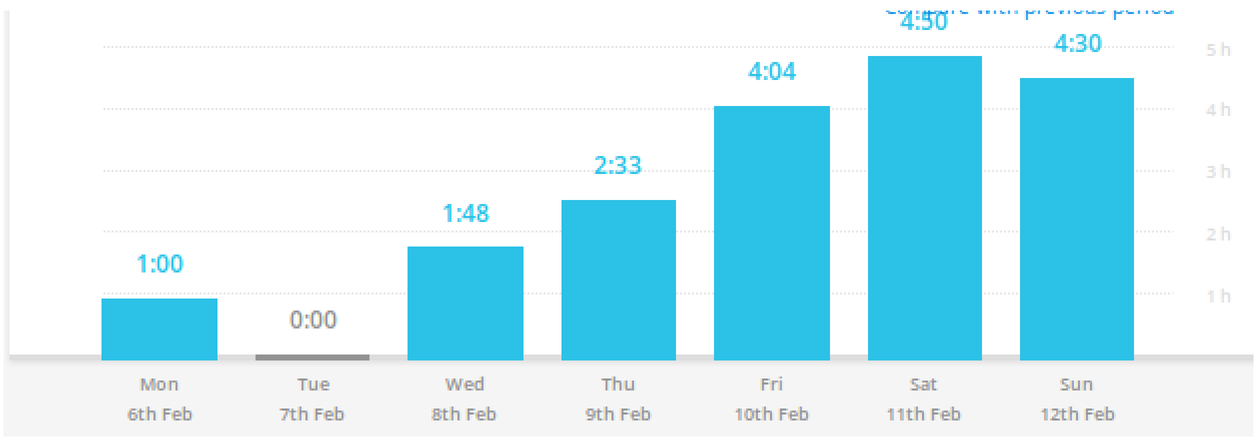
Fri, 3 Feb		0:13:00
Commit research work to Github	● General	0:13:00
Wed, 1 Feb		3:20:00
mini report	● General	3:20:00
Tue, 31 Jan		3:34:08
weekly log	● General	0:17:08
research links doc	● General	2:50:42
questions and answers (self)	● General	0:26:18 4:41 PM - 5:07 PM



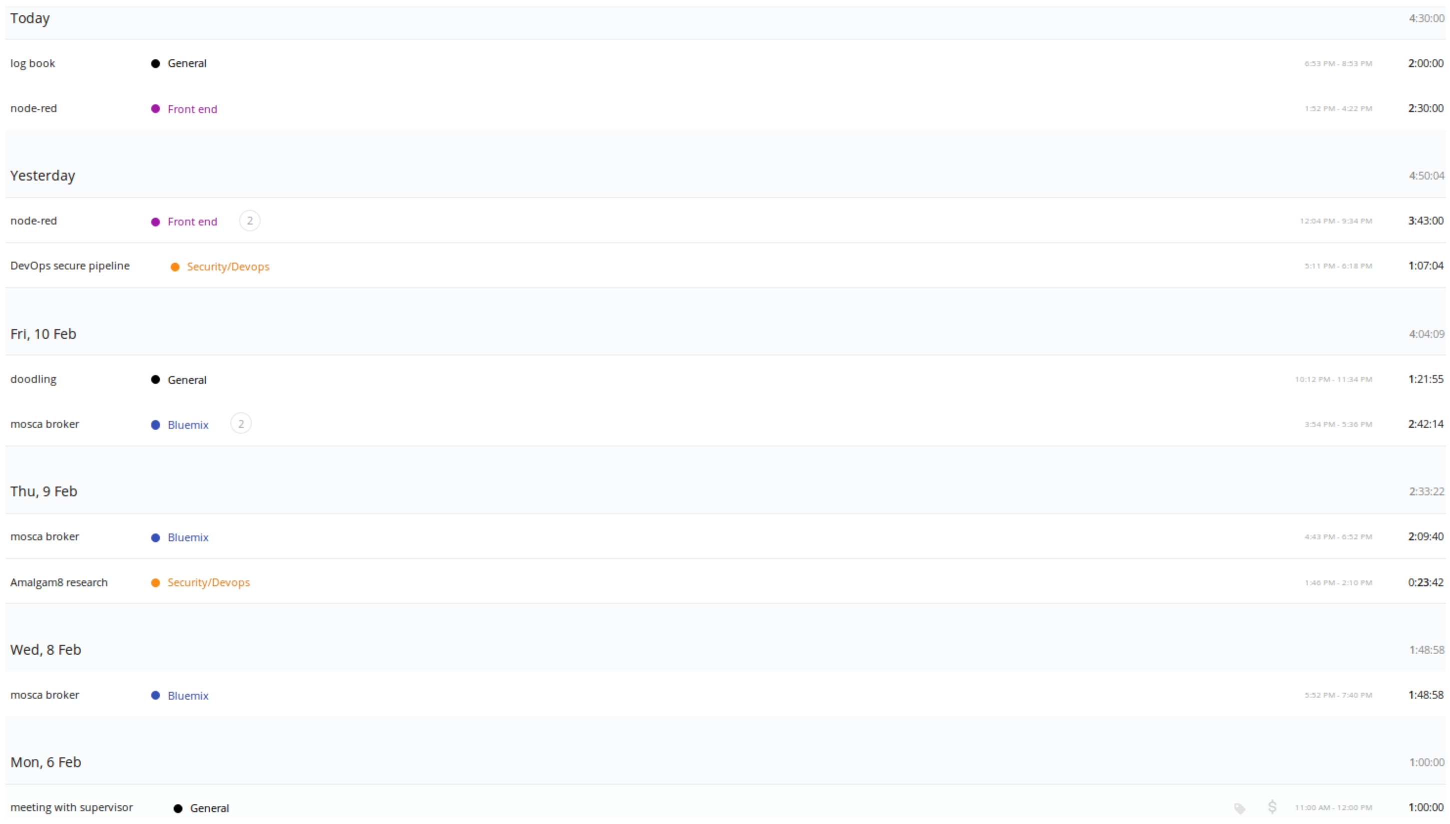
Mon, 30 Jan		5:40:07
research links doc	● General	1:03:04
mosca research	● Back end	0:38:07
Microservices from Theory to Practice (IBM doc)	● Bluemix	0:58:21
mqtt container research	● Bluemix	0:31:43
general research	● General	0:43:19
Amalgam8 research	● Back end	0:52:48
IBM Container group research	● Bluemix	0:21:36
IBM Container service research	● Bluemix	0:22:29
Creating new Bluemix account	● Bluemix	0:08:40

5.4 Week 3 Log

5.4.1 Week 3 Time Bar Chart



5.4.2 Week 3 Time Log



Today				4:30:00
log book	● General		6:53 PM - 8:53 PM	2:00:00
node-red	● Front end		1:52 PM - 4:22 PM	2:30:00
Yesterday				4:50:04
node-red	● Front end	2	12:04 PM - 9:34 PM	3:43:00
DevOps secure pipeline	● Security/Devops		5:11 PM - 6:18 PM	1:07:04
Fri, 10 Feb				4:04:09
doodling	● General		10:12 PM - 11:34 PM	1:21:55
mosca broker	● Bluemix	2	3:54 PM - 5:36 PM	2:42:14
Thu, 9 Feb				2:33:22
mosca broker	● Bluemix		4:43 PM - 6:52 PM	2:09:40
Amalgam8 research	● Security/Devops		1:46 PM - 2:10 PM	0:23:42
Wed, 8 Feb				1:48:58
mosca broker	● Bluemix		5:52 PM - 7:40 PM	1:48:58
Mon, 6 Feb				1:00:00
meeting with supervisor	● General	$	11:00 AM - 12:00 PM	1:00:00

5.4.3 Tasks completed

- **Mosca Broker working locally [23]**
- **Mosca Broker on Bluemix [23]**
- **Node-RED injects msgs [23]**
- **Node-RED sends to broker on bluemix [23]**
- **Node-RED encodes and decodes base 64 [23]**
- **Node-RED stores data in cloudant [23]**
- **Security DevOps pipeline tutorial [23]**

5.4.3 Week 3 Log Entries

Entry 08/01/17:

“While still recovering from a flu, I managed to familiarize myself with the cloud foundry Bluemix CLI. I made my goal to simply get a custom container image onto Bluemix. I ran into various problems such as logging into the american “ng” domain and not the “eu-gb” europe domain. Half of the delay was due to poor memory of how I did this last time. The other half being my state of mind (illness).”

Entry 09/01/17:

“Today I managed to get a mosca broker running on my local Docker host. Using Mosquitto service to pub and sub.”**[23]**

Entry 10/01/17:

“Today I deployed a mosca broker on IBM containers. I carried out the same test I did locally.”**[23]**

Entry 11/01/17:

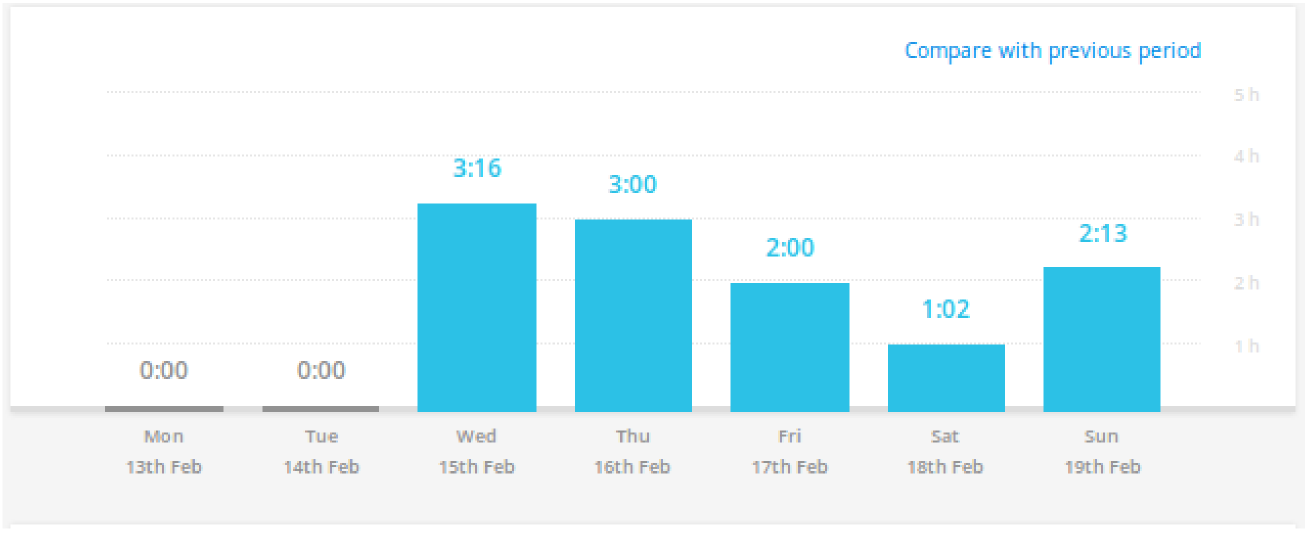
“Today I used Node-RED for the first time properly. I managed to get data flowing from an inject node to my broker running on IBM Container on Bluemix.”**[23]**

Entry 12/01/17:

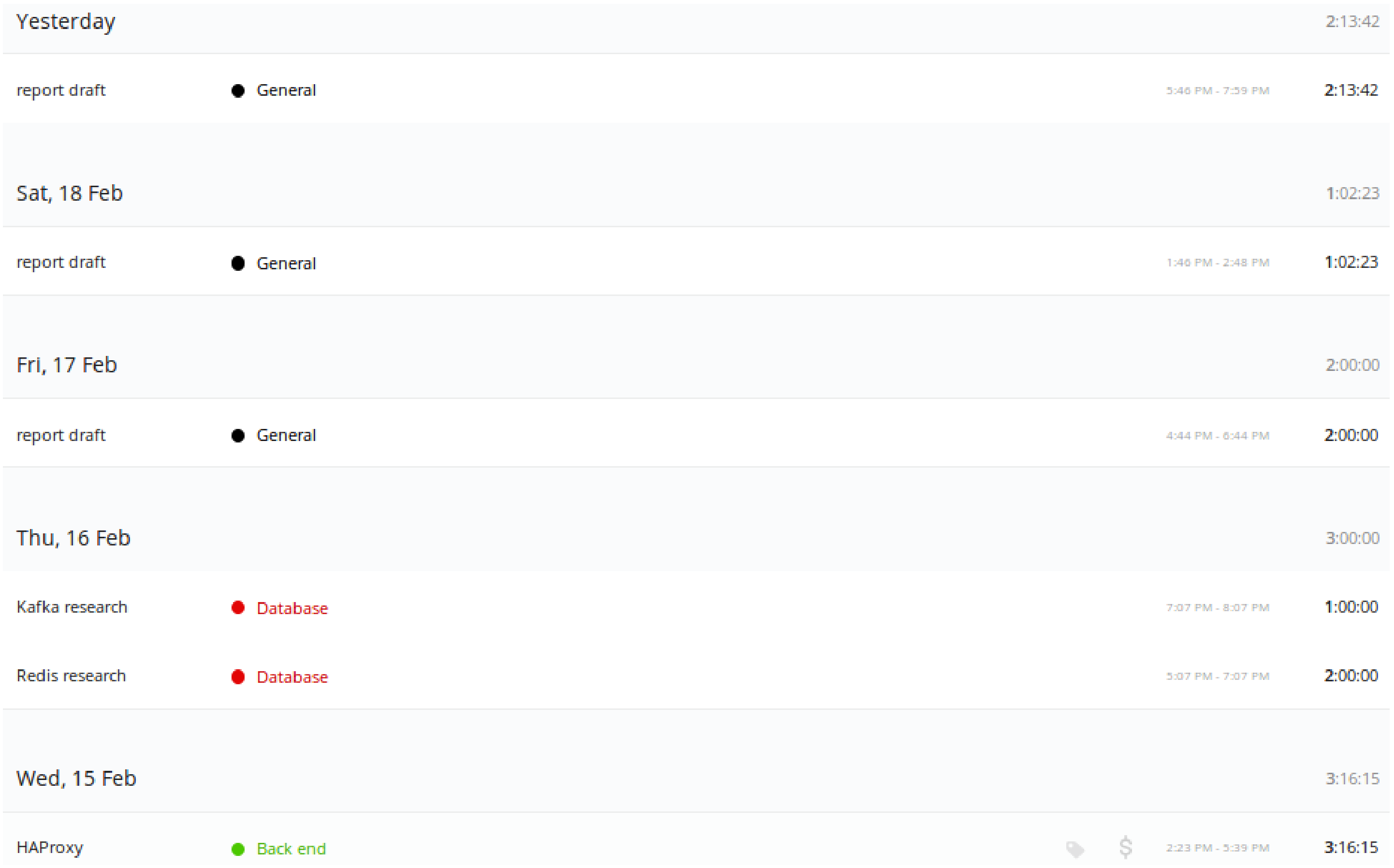
“Today I managed to encode/decode base 64 using Node-RED. I also managed to get the decoded data into a cloudant database. I ran into many problems this week, but time invested carried me through.”

5.5 Week 4

5.5.1 Week 4 Time Bar Chart



5.5.2 Week 4 Time Log



Yesterday			2:13:42
report draft	● General	5:46 PM - 7:59 PM	2:13:42
Sat, 18 Feb			1:02:23
report draft	● General	1:46 PM - 2:48 PM	1:02:23
Fri, 17 Feb			2:00:00
report draft	● General	4:44 PM - 6:44 PM	2:00:00
Thu, 16 Feb			3:00:00
Kafka research	● Database	7:07 PM - 8:07 PM	1:00:00
Redis research	● Database	5:07 PM - 7:07 PM	2:00:00
Wed, 15 Feb			3:16:15
HAProxy	● Back end	2:23 PM - 5:39 PM	3:16:15

5.5.3 Tasks Completed

- **Attempted to implement HAProxy container [24]**
- **Redis research [24]**
- **Kafka research [24]**
- **ELK Stack research [24]**
- **report draft [24]**
- **Literature survey spring report [24]**

5.5.4 Week 4 Log Entries

Entry 15/02/17:

"Today I struggled to get the HAProxy container working. I spent the first hour and a half just trying to attach an "overide config" as a volume to the container. Once "**docker logs <haproxy container id>**" stopped reporting errors, I knew at least the container was building successfully. The next step is to export HAProxy logs in order to find out more information as to what the problem might be."**[24]**

Entry 16/02/17:

"Today I researched both Kafka and Redis, trying to understand the difference between them and why/where you would use one over the other."**[24]**

Entry 17/02/17:

"Today I started my Spring report, this consisted of laying out the template and putting in section headings. I also research Amalgam8 microservices framework."**[24]**

Entry 18/02/17:

"Today I worked on "section 2 Literature survey" of my spring report. I also researched container monitoring and logging using the ELK stack."**[24]**

Entry 19/02/17:

"Today I continued work on my Spring report, finishing off the literature survey section."**[24]**

6 Bibliography

[1]T. Flynn, "GitHub - 16117743/INS-Thesis-Documentation: Documentation of Information & Network Security MEng Thesis", *Github.com*, 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation. [Accessed: 23- Feb- 2017].

[2]M. Collina, "An Internet of Things System - How To Build It Faster", *nearForm*, 2017. [Online]. Available: http://www.nearform.com/nodecrunch/internet-of-things-how-to-build-it-faster/. [Accessed: 23- Feb- 2017].

[3]"HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer", *Haproxy.org*, 2017. [Online]. Available: http://www.haproxy.org/. [Accessed: 23- Feb- 2017].

[4]J. Wu, "A Network Function Virtualization based Load Balancer for TCP", Master of Science, ARIZONA STATE UNIVERSITY, 2017.

[5]M. Collina, "GitHub - mcollina/mosca: MQTT broker as a module", *Github.com*, 2017. [Online]. Available: https://github.com/mcollina/mosca. [Accessed: 23- Feb- 2017].

[6]"RabbitMQ - Messaging that just works", *Rabbitmq.com*, 2017. [Online]. Available: https://www.rabbitmq.com/. [Accessed: 23- Feb- 2017].

[7]"mqtt", *npm*, 2017. [Online]. Available: https://www.npmjs.com/package/mqtt. [Accessed: 23- Feb- 2017].

[8]"Redis", *Redis.io*, 2017. [Online]. Available: https://redis.io/. [Accessed: 23- Feb- 2017].

[9]"Kafka vs. Redis: Log Aggregation Capabilities and Performance", *Logz.io*, 2017. [Online]. Available: http://logz.io/blog/kafka-vs-redis/. [Accessed: 23- Feb- 2017].

[10]"Kafka", 2017. [Online]. Available: https://kafka.apache.org/. [Accessed: 23- Feb- 2017].

[11]"Elasticsearch: RESTful, Distributed Search & Analytics | Elastic", *Elastic.co*, 2017. [Online]. Available: https://www.elastic.co/products/elasticsearch. [Accessed: 23- Feb- 2017].

[12]"Logstash: Collect, Parse, Transform Logs | Elastic", *Elastic.co*, 2017. [Online]. Available: https://www.elastic.co/products/logstash. [Accessed: 23- Feb- 2017].

[13]"Kibana: Explore, Visualize, Discover Data | Elastic", *Elastic.co*, 2017. [Online]. Available: https://www.elastic.co/products/kibana. [Accessed: 23- Feb- 2017].

[14]"Bluemix Container monitoring and logging", *Console.ng.bluemix.net*, 2017. [Online]. Available: https://console.ng.bluemix.net/docs/containers/monitoringandlogging/container_ml_ov.html. [Accessed: 23- Feb- 2017].

[15]A. AS, "Visualizing Docker Compose with Ardoq - Ardoq Blog", *Ardoq*, 2017. [Online]. Available: https://ardoq.com/visualizing-docker-compose/. [Accessed: 23- Feb- 2017].

[16]"Server-side service discovery pattern", *Microservices.io*, 2017. [Online]. Available: http://microservices.io/patterns/server-side-discovery.html. [Accessed: 23- Feb- 2017].

[17]E. Norman and E. Norman, "Amalgam8: Framework for composition and orchestration of microservices - Bluemix Blog", *Bluemix Blog*, 2017. [Online]. Available: https://www.ibm.com/blogs/bluemix/2016/07/amalgam8-framework-for-microservices-orchestration/?S_TACT=M16103KW. [Accessed: 23- Feb- 2017].

[18]"practice continuous integration", *Ibm.com*, 2017. [Online]. Available: https://www.ibm.com/devops/method/content/code/practice_continuous_integration/. [Accessed: 23- Feb- 2017].

[19]"TRANSIT: Flexible pipeline for IoT data with Bluemix and OpenWhisk – OpenWhisk", *Medium*, 2017. [Online]. Available: https://medium.com/openwhisk/transit-flexible-pipeline-for-iot-data-with-bluemix-and-openwhisk-4824cf20f1e0#.kv9gk4j2g. [Accessed: 23- Feb- 2017].

[20]"How to Build an High Availability MQTT Cluster for the Internet of Things", *Medium*, 2017. [Online]. Available: https://medium.com/@lelylan/how-to-build-an-high-availability-mqtt-cluster-for-the-internet-of-things-8011a06bd000#.iku8iazv6. [Accessed: 23- Feb- 2017].

[21]T. Flynn, "Spring Week 1 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk1.pdf. [Accessed: 23- Feb- 2017].

[22]T. Flynn, "Spring Week 1 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk1.pdf. [Accessed: 23- Feb- 2017].

[23]T. Flynn, "Spring Week 1 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk1.pdf. [Accessed: 23- Feb- 2017].

[24]T. Flynn, "Spring Week 1 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk1.pdf. [Accessed: 23- Feb- 2017].