Abstract

The term Container was inspired by Containers which allow shipping of goods across nations on large shipping hassles. In other words, if you can fit your cargo in that given specific space, you can transport it. The same principle applies to IT application workloads, if one can develop an application which can fit in a defined Container then this application can run and be transported seamlessly across any type of compute (hardware, OS, network) environment. Containers strip unwanted resources out of the OS. This frees up CPU, memory, network and other resources are freed up for middleware and application execution. As a rule of thumb, Containers are 10x more efficient than virtual machine in terms of CPU, memory, network and other resource utilization. IBM has forged a very close partnership with Docker in order to make them Enterprise Ready. IBM Software products such as Websphere Application Server (WAS) and IBM BlueMix are already Container ready. [1]

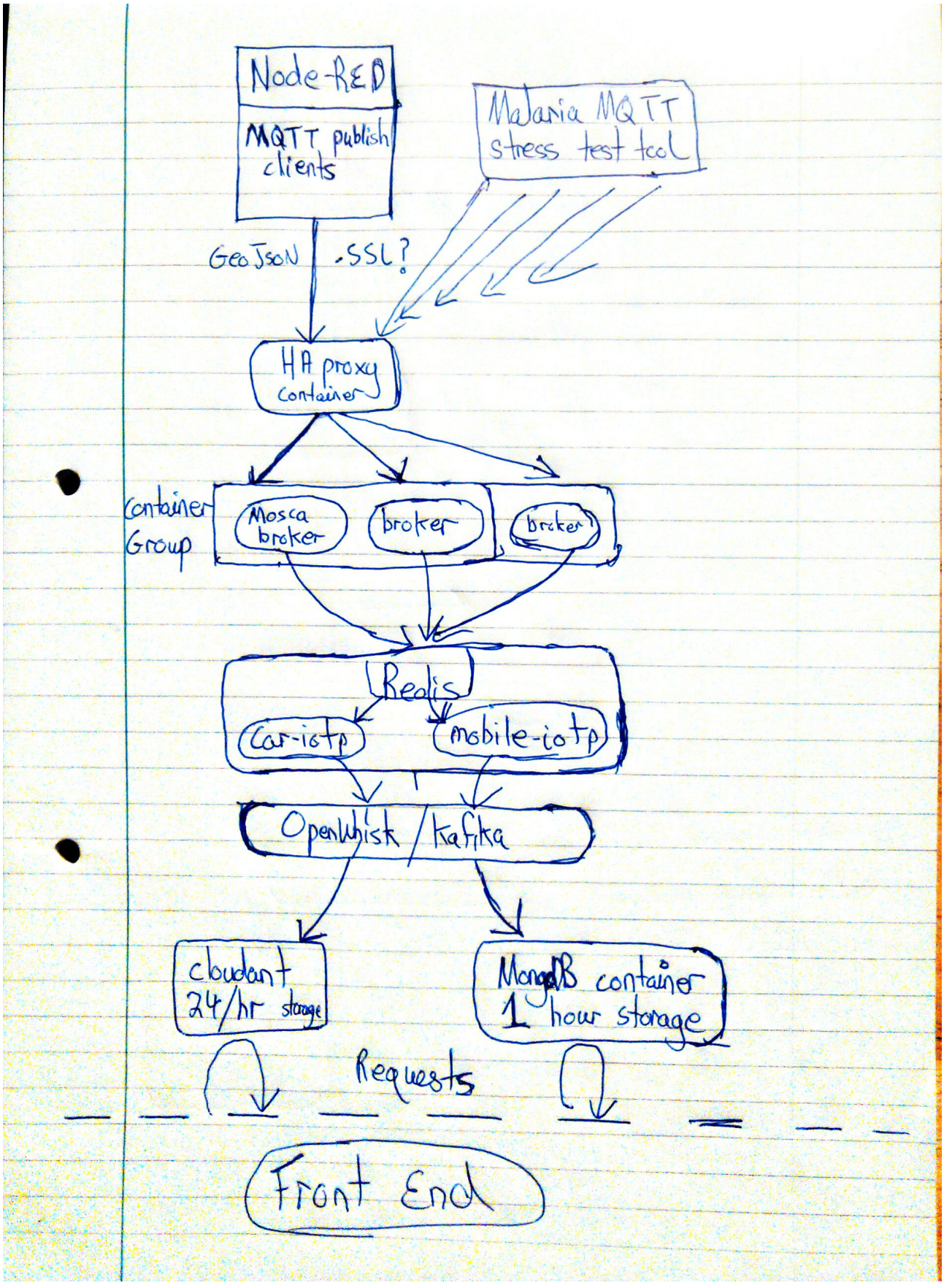
This project aims to engineer a scalable back-end solution for tracking in real-time the number of vehicles and students on the university campus. Due to their scalable capabilities, Containers were chosen in order to handle the variation of traffic on campus over a 24 hour period. During periods of 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 Containers will be significantly scaled down.

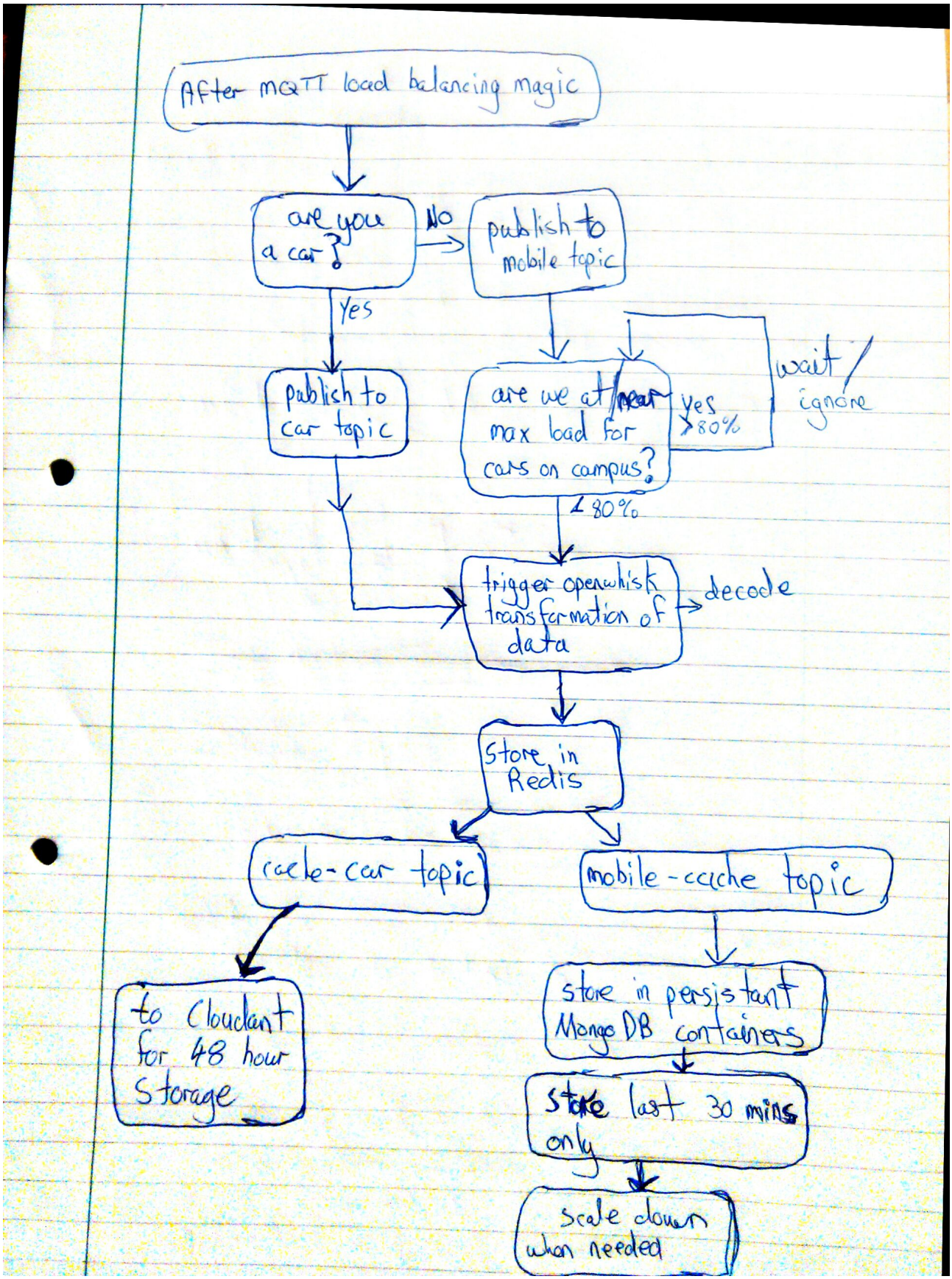
The clients (cars, students) will transmit GPS (Global Position System) data periodically to the server using a lightweight IoT protocol know as MQTT (MQ Telemetry Transport). This is a commonly used protocol for constrained devices. Both client and server will incorporate features to ensure that communication and caching of data is only performed when appropriate. 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 instances will subscribe to different topics (e,g car-iot and mobile-iot) in order to help split up the incoming messages between clients.

A basic prototype of the mobile application will be developed using the Ionic framework which is based on Angular.js. The application will be able to both transmit GPS data using MQTT as well as display the number of students and cars currently on campus. A map UI feature will be implemented if all higher priority tasks are completed on time.


Node-RED
MQTT publish clients
Malaria MQTT stress test tool
GeoJson
SSL?
HA proxy container
Container Group
Mosca broker
broker
broker
Redis
Car-iotp
mobile-iotp
OpenWhisk / kafka
cloudant 24/hr storage
MongoDB container 1 hour storage
Requests
Front End


After MQTT load balancing magic
are you a car?
No
publish to mobile topic
Yes
publish to car topic
are we at/near max load for cars on campus?
yes >80%
wait / ignore
< 80%
trigger openwhisk transformation of data
decode
store in Redis
cache-car topic
mobile-cache topic
to Cloudant for 48 hour storage
store in persistant Mongo DB containers
store last 30 mins only
scale down when needed

Back-end

Below is copied and pasted from

http://microservices.io/articles/scalecube.html

I need to read more on this and describe exactly where in my own project I will use x,y,z scaling.

And in particular, scaling of mqtt brokers. Also I need to break up the components of the back-end and discuss each one.

X-axis scaling

X-axis scaling consists of running multiple copies of an application behind a load balancer. If there are N copies then each copy handles 1/N of the load. This is a simple, commonly used approach of scaling an application.

One drawback of this approach is that because each copy potentially accesses all of the data, caches require more memory to be effective. Another problem with this approach is that it does not tackle the problems of increasing development and application complexity.

Y-axis scaling

Unlike X-axis and Z-axis, which consist of running multiple, identical copies of the application, Y-axis axis scaling splits the application into multiple, different services. Each service is responsible for one or more closely related functions. There are a couple of different ways of decomposing the application into services. One approach is to use verb-based decomposition and define services that implement a single use case such as checkout. The other option is to decompose the application by noun and create services responsible for all operations related to a particular entity such as customer management. An application might use a combination of verb-based and noun-based decomposition.

Z-axis scaling

When using Z-axis scaling each server runs an identical copy of the code. In this respect, it’s similar to X-axis scaling. The big difference is that each server is responsible for only a subset of the data. Some component of the system is responsible for routing each request to the appropriate server. One commonly used routing criteria is an attribute of the request such as the primary key of the entity being accessed. Another common routing criteria is the customer type. For example, an application might provide paying customers with a higher SLA than free customers by routing their requests to a different set of servers with more capacity.

Front-end

I need to discuss in more detail what is ionic. What can I realistically get completed between now and July. For now I'm not going to focus on this, just have it as part of "blue sky" development plan.

If I can simply demonstrate pressing a button and sending a GeoJSON object to the database. I will be happy.

Happier still if I can display a simple counter for the number of cars and mobile users currently active.

Delighted If I can display a map with dots of users and cars.

Over the moon If can separate those dots by colour.

Red -> eng students

Blue -> science

green -> arts

yellow -> cars

That's as far as I would go with the app.

How will development of an ionic application integrate with Bluemix/DevOps services?

What's the learning curve for such a framework?

Interfacing

Node-RED flow to broker

How exactly will devices be simulated, how can horizontal scaling be achieved using Node-Red?

What limitations are there to using Node-RED for testing scalability?

Discussion on mqtt topics and brokers

How is load balancing implemented?

How do you handle failure?

Is there a single point of failure?

Discussion on broker to Redis

How will you separate car devices from mobile devices?

How can the broker facilitate scaling?

Are all MQTT clients notified/subscribed to the same topic?

What caching issues/weaknesses are there with Redis?

How many topics will there be?

Discussion on consumer services of Redis

Kafka subscribers/ and or Openwhisk triggers to transform/decompress the data.

Where do the consumer services sit in with this architecture? Is it serverless?

Will you notify mobile clients from here of change in number of cars/mobile devices on campus?

Discussion on storing in Cloudant

Will there be separate databases for the cars and mobile devices?

How long will they be stored for?

How much will it cost for storage? Under increased load?

Will you notify mobile clients every time there is a change in the database?

Ionic (angular) to broker

How does the mobile application differ from the a car IoT device?

Is all communication (map features included) taken place using MQTT?

Where does REST/http fit in with this?

Have you considered using web sockets? How would this affect your solution?

Testing

Load balance testing

How will you use the load impact service (on Bluemix) for testing the MQTT brokers?

How will you scale horizontally the number of Node-RED flow instances?

How exactly will you use the malaria mqtt stress test tool?

How will Node-RED sit in with this method of stress testing?

Does Node-RED simply inject mqtt messages or will it handle a significant amount of logic?

How will you generate random GPS coordinates within the defined range?

Will you have to configure each device for authentication?

What will be the limit before an increase in funding be required?

Security

How will authenticate a mobile user over REST?

How will you prevent spoofed GPS coordinates? DOS attacks?

How will data in transit be secured? How much overhead will this add?

How will the mobile application be secure?

What attacks is this system weak against?

What does failure look like during an DOS attack?

How exactly is your gateway configured for incoming traffic?

Other testing

How will you test your code as you integrate it?

How will you test deployments?

What other aspects of testing are you not considering?

What testing can be omitted in order to keep on track for delivering a half working prototype?

Development Operations

How exactly will your continuous integration and development pipeline be integrated?

Will you have separate repositories for front and backend? What problems would you face with this?

How would you go about rolling out an update for A/B testing for a given feature?

Where will you implement security testing in the pipeline?

What additional parameters should be considered when configuring load-balancing?

How many different container groups will you use? Are they long running or short?

How would roll back during a deployment failure?

How will you debug microservice failure? -> ELK stack for logging

How will you deal with developing stateless microservices if state becomes an issue?

How will you break up the microservices? How many are there? What do they do?

What aspects of container orchestration will you need to learn in order to complete this project?

Will you be dealing with single and group containers? Or will cluster of containers be incorporated also? What issues arise with clustering?

Whats your time-line for this? Will you be using Agile or Kan-ban? How would using either one affect your pipeline?