CITY UNIVERSITY LONDON

MSc in Data Science

Project Report

2015/16

Using an Infrastructure as Code approach to a cloud-based Internet of Things Big Data Analysis platform

Alison Wells

Supervised by: Dr Tillman Weyde

September 2016

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: Alison Wells

Abstract

Sensor data from the Internet of Things provides an analysis problem due to its size and variety. Infrastructure asset management company Amey needed to understand how to build an architecture to analyse this data in a repeatable and efficient manner.

This project analysed research on architectural principles for big data analysis and Amey's requirements for a big data analytics platform and used these to design and develop CloudFormation templates that it showed can provision the infrastructure at least four times faster than manual provisioning. The infrastructure can analyse data streaming at up to 10MB/s and store PBs of data, sending out alerts when data values breach a threshold and allowing historical access to the data.

Keywords: *Infrastructure as code, Internet of Things, Big Data, Amazon Web Services, cloud*

Acknowledgements

I would like to thank Dr Pedro Baiz for suggesting this project and for believing that I could do it despite my lack of experience in this area. I would also like to thank Dr Tillman Weyde for his thoughtful comments and encouragement.

Table of Contents

1 Introduction and Objectives

Amey is a large infrastructure services company based in the UK, employing over 21,000 people and providing services in areas such as utilities, highways, waste management, rail, justice solutions, social housing and facilities management.

They have developed a system that analyses large amounts of data from asset sensors which allows asset owners to manage their assets more effectively. This system now needs to be scaled up and streamlined. This project will investigate how this scaling and streamlining can be done effectively and efficiently using an "Infrastructure as code" approach.

The research question for this project is: "Can an 'Infrastructure as code' approach be used to automatically produce a scalable Internet of Things big data analysis platform and what are the benefits in terms of speed?"

The project has the following objectives:

1) To produce a requirements analysis for a cloud-based big data analysis platform.
   This objective will be deemed to be met if a document is produced listing and justifying requirements.
2) To design a scalable infrastructure that can be implemented in Amazon Web Services (AWS) that can be used to analyse Internet of Things (IoT) sensor data.
   This objective will be deemed to be met if the infrastructure meets a defined set of the requirements collected during the requirements analysis.
3) To implement a selected part of the design through CloudFormation templates that can automatically provision the infrastructure.
   This objective will be deemed to be met if the templates can be run successfully and data can be processed from end to end in the system.
4) To evaluate the speed advantage of provisioning infrastructure via CloudFormation versus manual provisioning via AWS console.
   This objective will be deemed to be met if tests are carried out comparing the times taken to provision the same infrastructure manually and through templates, and the results analysed.

The beneficiaries of this work will be first of all, Amey, as the research on best practice for big data analytics architecture using AWS will allow them to decide whether they need to make changes to their existing system architecture and what changes should be made to make best use of AWS. Secondly the CloudFormation templates will allow them to implement the infrastructure immediately, and then build on it if they require. Outside of Amey, the research and templates could be used by anybody wanting to build a big data analysis platform using AWS as a way to understand the architecture options and get up and running as quickly as possible.

This report is divided into five main sections. After this introduction, the Context section reviews the main concepts around this project: Infrastructure as Code, cloud computing, Internet of Things and Big Data, and then looks at the architectural principles for each part of the analytics process. Next the Methods section covers the requirements gathering, design, development and evaluation plan, followed by an explanation of the resulting CloudFormation template and the evaluation results in the Results section. Finally, all this research is brought together and discussed in the Discussion, Evaluation, Reflections and Conclusions sections.

2 Context

2.1 Infrastructure as code[1]

2.1.1 What is "infrastructure as code"?

Sharma *et al.* [1] define "infrastructure as code" (IaC) as "the practice of specifying computing system configurations through code, and managing them through traditional software engineering methods." This approach is thought necessary in modern infrastructures by Morris [2], who describes a transition from the "Iron Age" where hardware is bought, installed and manually configured for each specific application to the "Cloud Age" where virtualisation is the norm and hardware ownership has transferred to Infrastructure as a Service (IaaS or more generally "cloud") providers such as Amazon or Google. Existing manual configuration practices have failed to keep up with the ease of spinning up large numbers of virtual machines which can also have many "containers" on them holding separate applications. Even when the individual configurations become more automated, the overall picture becomes increasingly complex and hard to manage with a lack of configuration management leading to many unique "snowflake" environments being created in an uncontrolled manner [3]. Alur *et al*. [4] specifically highlight automated configuration (i.e. Infrastructure as Code) as being an important approach to software development in the Internet of Things sphere because of the large number of Things that will be interacting in complex ways through hardware, software and networks.

IaC can improve this situation because by treating the infrastructure as code, it can be version controlled, documented and tested in the same way as application code. This means that administrators can quickly reproduce the existing environment on the same or new hardware or roll back to previous environments if necessary, and standards can be widely applied and automatically enforced [5][6]. Because adding new infrastructure now becomes scalable, with the same effort needed to set up 5 or 5000 servers, the ratio of admins to servers can be massively increased. Economou *et al*. [5] give figures comparing a recommended server to admin ratio of 10-20 to 1 as recently as 2002 with a statement from 2013 from Facebook that they were working with ratios of over 20,000 to 1.

2.1.2 Potential downsides of this approach

This approach requires a lot of cultural change and is generally associated with a move to a DevOps culture where developers, Quality Assurance and IT Operations staff not only work side by side, but actually merge roles. This is to try and break down the "Wall of Confusion" between developers, testers and IT Ops where code is "thrown over the wall" from one side to the other. It is also associated with the Agile development methodology where operational code is produced in very short (1-2 week) sprints and requirements are constantly being revised and reprioritised leading to requirements for frequent deployments [7]. To take advantage of these changes, systems are also often redesigned into a "microservices" architecture where a large monolithic application is broken down into many independent smaller services that communicate with each other. [8]

Testing methodologies also need to change once infrastructure can be changed as easily as code. Traditional methodologies use identical (in theory) development, test and operational environments and use end to end testing to slowly test new releases as they move from one environment to another. With a move to continuous integration / continuous deployment, testing needs to be carried out differently. As this is such a new area, there is no definitive answer but options include "blue / green

[1] This section is based on the Context section of the research proposal (Appendix 1)

releases" or "canary releases" [9] where the new release is spun up as an entirely separate system and the customers gently moved across from the old version, at which point the old version is destroyed. If there are any problems the customers can all be moved back to the old version immediately. Another idea is to focus on reducing the "Mean Time To Recover" (MTTR) rather than the "Mean Time Between Failures" (MTBF) and instil "antifragility" into the system. This means that things are expected to go wrong and monitoring is used to spot problems quickly and implement strategies to cope with them. "Chaos engineering" is the act of deliberately causing problems, for example by deleting servers randomly, so that code is written resiliently from the outset. [10]

2.1.3 Research gaps

This is a very new area and only a handful of research papers have investigated IaC. Scheuner *et al*. [11] used IaC to enable reproducible environments to compare different cloud services. Hummer *et al.* [12] looked at how IaC can ensure "idempotence" (identical results each time). Jiang and Adams [13] looked at the relationship between changes to infrastructure code and source code in an application and show that they need to be treated the same as they are equally prone to error. Sharma *et al.* [1] analyse large numbers of IaC scripts and identify code signatures that represent good or bad practice. All other publications on the benefits of IaC are anecdotal, and no published research can be found that attempts to quantify the benefits in terms of speed or cost over an existing system.

2.1.4 Technologies in this area

There are two main options for writing the templates to provision infrastructure in AWS, Terraform [14] and CloudFormation [15]. Terraform works with all cloud providers whereas CloudFormation is AWS-specific. Apart from this they are similar, except that there seems to be more documentation for CloudFormation and it is unclear how long it takes for new features added to CloudFormation to appear in Terraform. Tarry [16] recently compared the two approaches and decided that Terraform was easier to use, but was sometimes unstable and missing features from the various cloud providers.

Within the AWS services there are alternatives to CloudFormation for provisioning infrastructure requiring different levels of expertise which Padnick [17] describes as trading convenience for control. At one end, Elastic Beanstalk is easy to use and aimed at users wanting to deploy web apps, OpsWorks is in the middle, and CloudFormation offers the most flexibility but also is the most difficult to use as the templates have many options and can get unwieldy.

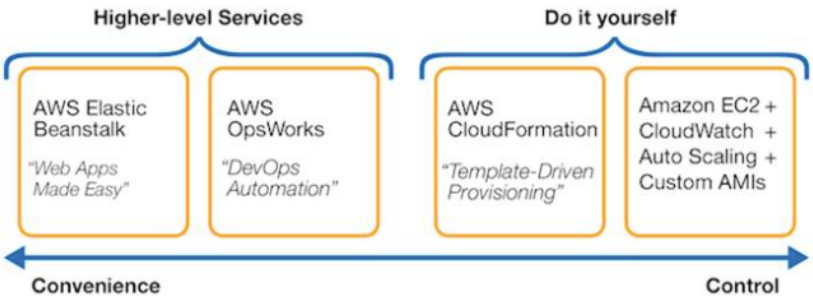


Figure 1: Trading convenience for control in AWS (from Padnick [17])

2.2 Cloud computing

2.2.1 What is Cloud Computing?

The US National Institute for Standards and Technology (NIST) has defined it as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” [18] Amazon defines it more colloquially as "the on-demand delivery of IT resources and applications via the Internet with pay-as-you-go pricing” [19].

These “clouds” can be either public, private or a mixture of the two (“hybrid”). Rather than providing users with specific hardware, cloud providers make use of “virtualization” which partitions real servers into one or more virtual servers which may be running completely different operating systems and software from each other and the underlying “real” server [20]. This allows for a much better use of resources as demands rise and fall.

NIST also define three service models for using cloud computing [18]:

- Software as a Service (SaaS) – the user can use applications that are not running locally, for example webmail
- Platform as a Service (PaaS) – the user can deploy software on to the service but has no control over the infrastructure
- Infrastructure as a Service (IaaS) – the user can configure the infrastructure to a certain extent, for example choosing operating systems and security settings

In each of these models, the key phrase is “as a Service”. The user does not own any of the hardware or software provided in this service, instead they pay a subscription of some sort.

The market for providing cloud services is large – Gartner [21] predicts that its value will grow 16.5% in 2016 to be US$204 billion worldwide although it is moving from the “peak of inflated expectations” to the “trough of disillusionment” in their 2015 hype cycle [22]. They put Amazon Web Services as the most capable provider, ahead of second place Microsoft in their August 2016 “Magic Quadrant” [23] which puts Google a long way behind in third place. AWS recorded sales of US$8 billion in 2015 compared with Google’s US$500 million although Google is trying to catch up aggressively, aiming to open 12 new facilities by 2017 [24]. According to Amazon, as of December 2015, AWS had over a million users in more than 190 countries [19]

2.2.2 Potential benefits of cloud computing

Using a cloud infrastructure and a “pay as you go” model can either lead to massive cost savings as computing power is only paid for as it used rather than sitting idle in a rack in an on-site data centre, or spiralling uncontrolled costs for a resource-hungry application. Tesco give an example [25] of a requirement for a webpage that was going to cost £3.5k in in-house infrastructure but cost 13p a month using cloud. News Corp are hoping to save “tens of millions of dollars” over 10 years by moving to the cloud and closing 40 of their 46 data centres [26]. However, marketing company Pubmatic abandoned their move to this model after their project ended up costing over 18 times the original budget with bills of hundreds of thousands of dollars a month [27].

Building resilience into a system can be hard to do for small and medium companies without access to dispersed locations for their data centres and this is where the major cloud providers can use their size to easily add this in. A secondary benefit of distributing content across the world is that users can

be served from their nearest copy reducing data transfer times. For example, AWS currently allows users to use 11 regions across the world containing 35 separate availability zones in total [28]. In theory, by replicating applications across these regions and zones, the chance of a system outage is significantly reduced. In the author's experience resilience tends to be an afterthought for projects where they have to deploy their own infrastructure and likely to be cut due to lack of money, but the likelihood of an outage should not be underestimated, Computer World [29] gives a list of significant data centre disasters which took users' applications offline with a range of causes including lightning, hurricanes, power surges, thieves, poor driving and squirrels.

As well as resilience, cloud computing can easily scale up and down as required, unlike on-premise data centres which are limited by the hardware the user can afford to have sitting on standby. At a talk attended by the author [30], digital media agency Parallax gave an example of how they used the cloud to deliver a web site that could quickly scale up to millions of users for an ad campaign based around the Euro football championships which would be no longer needed after a few weeks. He stated that the response time actually decreased with more simultaneous users and that overall infrastructure costs were only a few hundred UK pounds. Another advantage of the cloud is that it usually costs the same to run 10 servers for one hour as it does to run one server for 10 hours, so if you can design your process to run in parallel (not necessarily always possible) you can benefit from significantly reduced run times for the same cost.

Cloud computing is often described as being environmentally friendly. Data centres take a lot of power both to run the servers and to keep them cool. Cloud providers can run larger more efficient data centres – Amazon suggests that users can reduce power requirements by 84% after relocating an on-premises data centre to the cloud. Cloud providers are even locating them in cooler parts of the world such as Iceland, Norway and Sweden [31]. These countries also offer good sources of renewable energy as companies try to increase their green credentials by moving away from fossil fuels. Amazon has a "commitment" to 100% renewable energy through their own wind and solar farms [32] but a Greenpeace report [33] has lambasted the company for a lack of transparency on their achievements, putting them far behind Apple, Facebook and Google.

2.2.3 Potential risks of moving to the cloud

One of the major worries people have before moving to the cloud is security. While most of the threats to cloud infrastructure are the same as they would be for on-premises infrastructure, moving to a system where you are sharing hardware with other customers adds a level of additional threat. In general, using encryption, strong authentication and authorisation and well-designed "defence in depth" security [34] should guard against compromise and the major cloud providers supply tools for these tasks.

Another threat is that of vendor lock-in where a system is so entwined with proprietary services from a particular cloud provider that it is impossible to move it to a competitor without completely rewriting it. Recently "containerisation" (effectively a software configurable mini virtual server) has become a hot topic as these can be moved between environments without affecting the applications inside. Writing infrastructure templates using a vendor-neutral product such as Terraform could also help.

Also the cloud is not a panacea for everything. Weinman [35] gives a list of occasions **not** to use a cloud, for example when a service is in constant demand and doesn't need to use the powerful scaling ability of the cloud, or uses highly customised or legacy code.

2.3 Internet of Things

2.3.1 What is the Internet of Things?

There are a range of definitions of the "Internet of Things" (IoT). The World Economic Forum [36] defines it as "a network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment" that it believes can "fundamentally transform" human-machine interaction over the next 10 years. Zaslavsky *et al*. [37] define it as "[comprising] billions of devices that can sense, communicate, compute and potentially actuate". These objects or "Things" can be anything from a RFID tag to a lightbulb to a whole city [38]. In fact Akyildiz *et al*. [39] go even smaller to an Internet of "NanoThings" and "Bio-NanoThings" that would be biologically embedded in humans.

The term "Internet of Things" has been around since 1999 when it was coined by Kevin Ashton in a presentation on using RFID tags in a supply chain [40] but only recently has it started to become a reality due to a "tipping point" being reached because of the availability of cheap low-power sensors, almost ubiquitous connectivity, cheap, scalable processing and storage in the cloud and the move to IPv6 significantly increasing the number of individual things that can be identified (2^{128} addresses for IPv6 rather than 2^{32} for IPv4) [41]. Guinard [38] suggests the key moment for IoT was December 2013 when the phrase "Internet of Things" appeared in more news headlines than "Web 2.0" and Google acquired thermostat maker Nest for $3.3bn.

Gartner has IoT and big data analytics on the way to the top of the "Peak of Inflated Expectations" in its August 2015 hype cycle [22], but also sees it as "one of the most active areas for innovation" [42]. There are a range of predictions for the numbers of connected Things in 2020 with Gartner suggesting 26 billion [43] at one extreme and Morgan Stanley 75 billion at the other [44]. In a press release [45] IDC suggest that the market in IoT could be worth $7.1 trillion by 2020.

2.3.2 Potential applications for the IoT

For the consumer, the obvious uses of IoT are for so-called "Smart" homes where heating and lighting can be controlled via a smartphone app, and energy and water usage monitored in real time [4]; wearable devices that allow the tracking of such things as heart rate and steps taken leading to a "quantified self" [38]; as well as cars that are connected to the internet and able to automatically choose the best routes or find car parking spaces [46].

More interesting are the implications of IoT for industry (represented by the terms "Industry 4.0" or "Industrial Internet"). Cities will be able to monitor things like the structural health of buildings, air quality and noise pollution [47]. Farmers will be able to monitor the health of their crops with minute accuracy and ensure exactly the right amount of pesticides and nutrients are used [36]. Companies will be able to monitor all of their assets, no matter how remote, and know exactly when any problems occur, for example, Thyssenkrupp have developed a system using machine learning that monitors all their lifts to ensure that they can be maintained and fixed before they break down [48].

2.3.3 Risks of the IoT

Security is a key risk area, both ensuring devices aren't compromised (for example a cyberattack on a nuclear power station) and ensuring the security and privacy of the data collected from them. [49][50]. The infrastructure that the IoT relies on needs to be reliable and fast enough to run safety-critical applications – what is an acceptable interruption to a streaming video may not be for an air traffic

control system. Also Riggins and Wamba [44] highlight a lack of common standards used by IoT devices which needs to be addressed.

Relevant to this project are the frequently highlighted risks of not having the right tools and skills to make best use of the data collected [36] [44]. If the data is so large that it cannot be stored for long, or at all, then streaming algorithms need to be developed that can detect relevant changes in real time. Also approximate but fast data analyses may be preferred to slow and accurate ones [37].

2.4 Architectural principles for big data analysis

2.4.1 Big data

In 2012, Gartner [51] defined big data as "high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation." This has become the standard definition and known as the "3 V's" of big data. However, other "V"s have been added since such as "veracity" (uncertain reliability of data), "variability" (inconsistency in data) and "value" (the ability to extract value out of the data) [52][53] with Assunção *et al*. also noting 'viability' [54]. Zaslavsky *et al*. [37] describe it as consisting of "hidden gold (high-valued data) mixed with dirty (noise, erroneous and raw) data".

A 2014 report by technology market research company IDC [55] says the amount of data created annually is doubling every two years and by 2020 will be 44ZB (1 zettabyte is a trillion gigabytes) of which 40% will be stored in the cloud. It predicts that the data from the Internet of Things will be a large contributor to this, growing from 2% of this "digital universe" in 2013 to 10% by 2020, but states that only 15% of all this data will actually be able to be stored.

These large volumes of data mean that data analytics are an ever more important way to find and keep the "hidden gold". These analytics need to be able to cope with analysing high volumes of continuously streaming data at high velocity, without relying on querying data stored in a traditional relational database structure, and therefore the architectures they are built on have to be scalable, resilient and fast. As we have seen, cloud computing can provide all these features although Assunção *et al*. [54] believe that getting value out of big data is still time consuming and expensive.

2.4.2 Overall architectural principles

Meyers [56] suggests that for big data analysis, processing and storage should be separated, that the system should be "event driven" ("Event driven architecture" uses a publish – subscribe model for communicating between system components which means that components can be very loosely coupled [57]) and that the component parts of the system should be loosely coupled, allowing independent scaling up and down as required. He also says that it is important to get the right tool for each job ("data sympathy") with a focus on the data and the business need. He also proposes a general architecture which mixes real time stream processing with slower time batch analytics. Kambatla *et al*. [20] agree that each system will need a mix of approaches and state that a good architecture has distributed software and storage, can tolerate hardware failures and can scale automatically. Marz and Warren [58] believe that all data that is ingested should be immutable so that the data can't be corrupted by mistake, and that systems need to be robust, fault tolerant, scalable, generalizable, extensible and debuggable, needing minimum maintenance, with low latency access and the ability to run ad hoc queries.

In terms of scaling, Padnick [17] suggests that scaling out (adding more servers) is preferable to scaling up (using bigger more powerful servers) but that both options need to be considered. Scaling out can give increased resilience as if one server breaks, there are still others to carry the load, and if the system is designed to scale out, then it is easy to quickly bring a replacement server online. However, it can be difficult to design the system to scale in this way. For example, traditional relational databases usually need to be hosted on a single server [59].

Meyers [56] defines a framework for designing an architecture based around ***Collect > Store > Process > Consume***

2.4.3 Collect

Meyers's principles [56] for data collection architectures are:

- "Should be easy to install, manage and understand
- Meet requirements for data durability and recovery
- Ensure ability to meet latency and performance requirements
- Data handling should be sympathetic to the type of data"

Hashem *et al.* [53] group data sources into "Web & Social", "Machine", "Sensing", "Transactional" and "IoT", but these seem to overlap and they don't suggest that these sources should be treated differently from each other. Meyers [56] divides sources into "Applications", "Logging", "Transport" (in this case he means transporting data using an import device), "Messaging" and "IoT". The outputs of these groups are records, documents, files, messages and streams that can all be dealt with differently at the next stage.

Kambatla *et al*. [20] suggest that where data is collected from is also important, that data collected locally to the infrastructure may be treated differently than if it was far away. For example, Akhbar *et al*. [60] consider whether local processing in "micro-" or "nano-datacentres" should be carried out before the data is collected to reduce latency or data transport costs. This has also been called "edge" or "fog" computing [46] to distinguish it from centralised cloud computing.

2.4.4 Store

Meyers [56] lists the type of data stores that can be used as:

- "Cache – for heavily read data
- Databases – for consistently read, relational or document data
- Search – frequently read, optimised for searching
- File store – for files and objects
- Message queue – for messages intended for a single recipient with low latency
- Stream storage – for messages intended for many recipients with low latency"

Speed of access is one of the main differentiators of the different storage types. Meyers [56] and Kambatla *et al*. [20] both stress the importance of deciding whether data is "hot" (needing frequent, fast access) or "cold" (needing infrequent access where latency is not an issue). Data often starts off as "hot" and then "cools" as it gets older, although this may not always be the case. The main reason to make this decision is cost with "hot" data storage being considerably more expensive than "cold" as can be seen in Table 1.

	Hot	*Warm*	*Cold*
Volume	MB-GB	GB-TB	PB
Item Size	B-KB	KB-MB	KB-TB
Latency	Milliseconds	Milliseconds, seconds	Minutes, hours
Durability	Low-high	High	Very High
Request rate	Very high	High-medium	Low
Cost / GB	££-£	£-pp	p

Table 1: Cost of data storage based on the options available in Amazon Web Services (after Meyers [56])

In terms of database storage, the traditional choice has always been the relational database. This needs data to be structured in a known format and is hard to scale, however big data can be in a variety of formats and is often unstructured. New types of databases have been developed to deal with this which are generally put under the heading of NoSQL ("Not only SQL"). These are usually broken down into four types as described in Figure 2 [59]:

- Key / value stores
- Column – oriented stores
- Document stores
- Graph databases

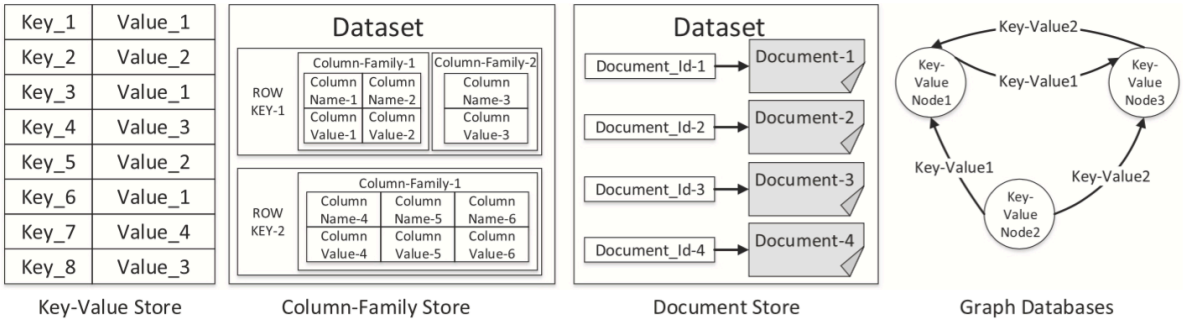


Figure 2: Different types of NoSQL data models (from Grolinger et al. [59])

In terms of performance, Borkar *et al*. [61] rate the different types as in Figure 3:

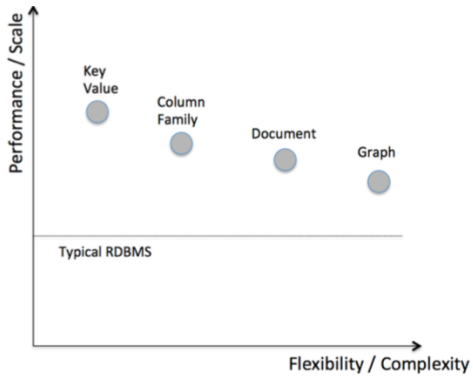


Figure 3: Performance of different types of NoSQL databases (from Borkar et al. [61])

Grolinger *et al*. [59] also describe a "NewSQL" type of database which is a hybrid of a NoSQL and relational database. Choosing the right database type is important as it affects the type of querying that can be done, for example, key-value stores can only be queried by using the keys, not the values.

Where NoSQL databases excel is in scalability. Relational databases can generally only be scaled "vertically" by putting them on more powerful servers, whereas NoSQL databases are designed to be scaled "horizontally" by adding more servers [59]. This scaling comes at a cost though. Brewer's CAP theorem [62] states that out of Consistency, Availability and Partition tolerance, a database can have two of these properties at once but not all three. Therefore, for a database to be highly available and partition tolerant to allow scaling, consistency must suffer. Relational database transactions should be ACID (Atomic, Consistent, Isolated, Durability) so to allow for reduced consistency, Pritchett [63] proposed a methodology called BASE (Basically Available, Soft state, Eventually consistent). Figure 4 shows the relationship between these concepts.

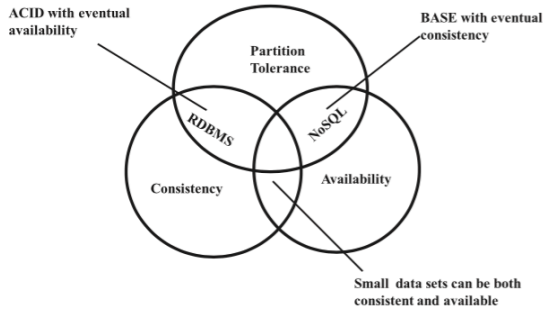


Figure 4: Relationship of CAP theorem to ACID and BASE methodologies (from Cooper and Mell [64])

Gessert *et al*. [65] rate a range of NoSQL databases depending on their strengths, and how they fit the CAP theorem, as well as the types of applications they would be best for in :

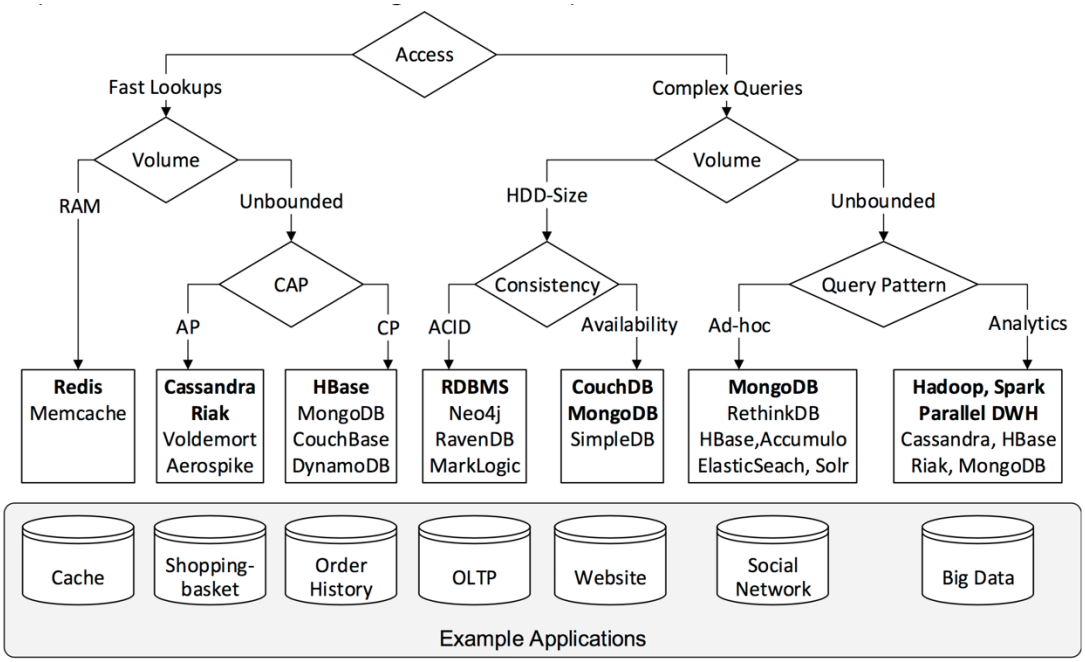


Figure 5: Strengths of different NoSQL databases (from Gessert et al. [65])

2.4.5 Process

Meyers [56] divides the different types of processing as follows:

Processing type	*Time to get result*	*"Temperature" of data (see Table 1)*
Batch	Minutes or hours	Cold
Interactive	Seconds	Warm / cold
Messaging	Milliseconds or seconds	Hot
Streaming (either micro-batch or real time)	Milliseconds or seconds	Hot

Table 2: Different types of processing (after Meyers [56])

It is also generally agreed that for big data it is important to have the processing done where the data is, as the costs of data transport become significant at scale [20] [54]. This is the basis of the MapReduce algorithm [66] which breaks down processing into parallel tasks that can be carried out locally on separate datasets and then the results combined together to get an answer.

Again, deciding on the right level of latency required has a bearing on cost, dealing with streaming data one item at a time can be very costly compared with batching it into even small batches [20] Marz and Warren [58] propose an architecture they call "lambda architecture" which attempts to balance this by doing most of the processing via batch jobs with a small amount of streaming processing to fill in the gap between the last data item that went into the batch job and the present moment. The decisions on architecture here will depend on how much context from previous data is needed to analyse incoming data, as well as how accurate the analysis is required to be. *Kambatla et al.* [20] give the example that healthcare data analysis needs to be more accurate (and probably faster) than analysis of climate change data where general trends are more important.

2.4.6 Consume

Meyers [56] has two main outputs from the processing stage. Firstly, results can be sent to applications and APIs and secondly, visualisation and visual analytics tools can be used to explore the data in more detail either by business users or data scientists. Assunção *et al*. [54] state that real-time interactive querying is difficult on big data, and that query complexity or accuracy need to be compromised on to get results at an acceptable speed. Conversely complex queries can be done if the user is happy to wait for an answer.

2.4.7 Security / Legal / Ethical issues

Questions that need to be asked here are: How sensitive is the data? Is it personal data? How long should it be retained for? How sensitive are the models? Do they need to be protected?

Kune *et al.* [67] acknowledge that security is very important in big data projects or they could turn into a "security and compliance nightmare". They propose an "onion model" of security at different layers – infrastructure, data, analytics and users – with a mix of authentication, authorisation and encryption to keep data secure and private. Hashem *et al.* [53] consider whether encryption is possible at all stages of processing and whether encrypting intermediate datasets is cost-effective. Obviously the type of data is also a key consideration with, for example, personal healthcare data requiring a much higher level of protection than climate data [20].

When using cloud infrastructure, the location of data is usually irrelevant to the user. However, different countries have different data privacy laws and so this is where the location is important. Also anyone considering setting up a big data analysis platform should also set up a governance framework that specifies policies that reduce the risk of data breaches, such as retention periods [53].

2.4.8 The path to Serverless

As has already been covered, with the advent of the cloud, users are no longer concerned with physical hardware, instead using virtual servers. The next step from this is "containerisation" where each application is given its own container – a lightweight way of combining an application with its runtime environment that allows it to be moved from server to server [68]. This technology has been around for a while but has become popular in the last couple of years as the release of software like Docker [69] has made it a much more user-friendly process. Jlassi and Martineau [70] showed that running an application in separate containers rather than separate virtual servers on a single real server was more efficient.

Virtualisation and containerisation still need the user to ensure that they are being scaled up and down appropriately, even if this is done through automatic means. One of the downsides of the pay as you go model is that you can't pay for a percentage of a server and in the case of AWS you can't pay for less than an hour's use [71]. Also while scaling up the number of containers can be faster than scaling up the number of servers, this may not be quick enough to cope with a sudden surge in traffic. This means that if usage of the system is irregular, there may be delays when traffic peaks, but also costly over-capacity if traffic drops to zero.

The "Serverless" model moves from packaging the application as a whole to packaging each function separately. While each function still runs on a server somewhere, the user no longer has any control over this, instead paying for each millisecond of compute time. Each function is run in an "event-driven" manner with a copy of each function started for each new event (for example, web page request or data item received). Roberts [71] lists the advantages of this model as including having automatic scalability built in, cost-savings for non-uniform or low traffic, easier operational management as no servers to manage and potential environmental benefits due to a reduction in idle servers which still need a minimum level of power and space. On the other hand, the technology is still immature – the leader is AWS Lambda which has only been around since 2014, multitenancy is still an issue – containers are less protected than virtual servers, and functions less so again, vendor lock-in also becomes an issue as functions are written specifically for the cloud provider's model and not all systems are suitable for breaking down into simple functions as these functions have no "memory" and are stateless.

Fundamentally decisions about using a containerised or serverless model are driven by whether the system can be designed to make the most of the advantages of each model, and how much vendor lock-in is an issue.

2.5 Summary

Modern technological advances, such as the Internet of Things, have brought new opportunities to gather data and get new insights to make significant improvements to the world, however they have also brought challenges in terms of the amount of data that is now available to be processed and stored. Cloud computing can address some of these challenges by providing a scalable infrastructure with a low barrier to entry. Using infrastructure as code can also reduce costs and risk as infrastructure details can be version controlled, easily updated and shared. Amazon Web Services is the market leader in this area with a large array of pay as you go services.

3 Methods

This section looks at the methods used to gather the requirements and design, develop and evaluate the system.

3.1 Requirements analysis

Robertson and Robertson state that the main function of requirements analysis is "understanding a business problem and providing a solution to it" [72]. The requirements analysis process involves understanding the business problem a new system is trying to solve, identifying the key stakeholders, the scope of the system and the functional and non-functional requirements that need to be fulfilled in order to solve the identified business problem.

The literature reviewed and the author's own knowledge were also used to design a list of things that need to be considered in terms of designing a big data analytics infrastructure, structured around the 5 Vs:

Volume

- How much data will be sent to the system every day?
- How long does it need to be stored for?

Variety

- What format(s) will the data be in?
- Will the data be highly structured / semi-structured / unstructured?
- What type of media will the data be? E.g. text / images / video?

Velocity

- How fast will the data arrive?
- How quickly does it need to be processed and analysed?

Veracity

- How complete or accurate will the data be?
- How much data cleaning will need to be done?
- How accurate does the analysis need to be?

Value

- Does the analysis need to be simple or complex?
- Will the analysis use all the data or just the latest data?
- What type of analytics will be used? E.g. Descriptive (model past behaviour), Predictive (forecast based on available data) or Prescriptive (assess actions, assist decision making) [54]?
- How will the users of the analysis interact with it? E.g. Through visualisations / web apps / email alerts?
- Will the results of the analysis be used to directly affect something?

These questions were used in a semi-structured interview with Dr Pedro Baiz to determine the requirements of the prototype platform. Other methods of gathering requirements were looking at the requirements for Amey's current system and looking at the literature for suitable requirements for big data analysis in general. These requirements are a simplified form of the more complex

requirements that the Amey system needs to fulfil, so that the development is achievable in the research timescales.

After the requirements specification (see 4.1) was written it was presented back to the stakeholders and agreed with them.

3.2 Design

3.2.1 Overview of Amazon Web Services

AWS currently provides 66 different services [19] across a range of areas, for example, computing, storage, networking, analytics and security (see Appendix 2 for a list). In fact, there are so many services and they are changing so often (in December 2014 there were only 37 services [17]), it can be difficult to know where to start. For this project each of the relevant services were evaluated for their suitability using the documentation and Meyers's Collect > Store > Process > Consume framework [56] to design each part of the system.

3.2.2 Overall high level design

First of all, the high level design was developed based on Meyers's architectural design [56] which combined real time streaming analytics and slower time batch processing. Figure 6 shows the data taking two pathways, one for real time comparison with a threshold that then triggers an email alert if the thresholds are breached (to fulfil requirement FR 4), and a second into a data store which is then accessed by an analytics web app (to fulfil requirement FR 5). For more details on all the requirements see 4.1) The size of the arrows represents the quantity of data being moved at each stage – initially every piece of data is processed, but then this is filtered for the following stages.

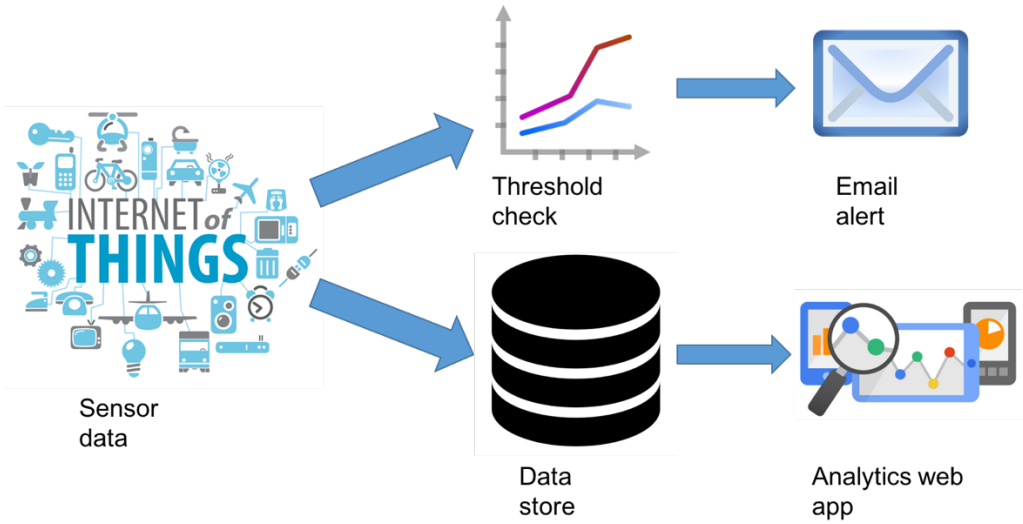


Figure 6: Overall high level design for the system

Using the options analysis for each stage of the Collect > Store > Process > Consume framework, the overall system design (Figure 20) was developed with the choices made for each AWS service that was to be used. These choices aimed to fulfil the rest of the functional and non-functional requirements, by choosing services that could transfer the volumes of data required by requirement NFR 1, store the volumes required by NFR 2, process the types of data required by FRs 1, 2 and 3 and scale the web app as required by NFR 3.

3.3 Development

3.3.1 An anatomy of a CloudFormation template

The project was developed using CloudFormation templates. These were chosen as they were AWS specific and therefore should be up to date with all the available AWS resources, and as the project is focussing on AWS as the current market leader, it wasn't considered a requirement to be vendor-neutral (Constraint 2 from 4.1.3).

CloudFormation is a method of defining a "stack" of infrastructure using a template written in JSON. This makes it easy for the user to set up multiple resources at once and ensure that the stack is always set up in the same way. By version controlling the templates, the stack can be updated with a complete record of changes made. If any problems arise the stack can be easily rolled back to the previous stable configuration. This has been the norm for software development for a long time, as have scripts used in hardware configuration, but it is only in the last couple of years that the whole stack can be defined in this way.

The CloudFormation template is made up of six parts [100]. The template is "declarative" which means that it describes the end state of the infrastructure rather than the steps needed to get there.

1. Template version / description / metadata (optional) – describes the template and what it is for
2. Parameters (optional) – allows templates to be customised when they are run, for example to specify the size of the EC2 instance
3. Mappings (optional) – a key-value section that allows lookups, for example to get the right AMI to use for the region specified
4. Conditions (optional) – allows the template to change behaviour based on some condition, for example, create different resources based on a parameter entered
5. Resources (required) – the main section of the template that specifies the details of the infrastructure
6. Outputs (optional) – allows for details of the stack to be output in the right format for further use, for example, outputting a URL for access to the web application

Within the template there are "intrinsic functions" which allow for cross-referencing of the various parts of the template. For example, the "Ref" function looks up the actual physical id of a resource which was given when it was created which wouldn't have been known in advance. Alternatively the Fn::Join function joins text together to form e.g. a URL from the Public IP address given to an EC2 instance when it was set up.

Apart from logically separating the template, there is no intrinsic order to the way resources are created. The CloudFormation system analyses the template and looks for explicit dependencies to work out the order in which resources should be set up. These explicit dependencies can either be through the use of intrinsic functions to add details of other resources, or through an explicit attribute "DependsOn" which names the resources that need to be set up first. Although the majority of the dependencies are worked out automatically, care still needs to be given to checking that they are all explicitly defined as a stack may be created perfectly several times and then inexplicably fail from the same template because resources were set up in a different order.

3.3.2 Developing in stages

An incremental model as described by Dawson [101] was then used to break down the system design into smaller fully functioning sections that could be built on to form the final system. This ensured a working system was developed as early as possible.

For each stage, the system level design was further broken down into a component level design which showed exactly which AWS components were required and how they depended on each other using arrows to show dependency such that the component that needs to be built first is at the start of the arrow, and the component that depends on it is at the end. Any loops in the model would mean that it was impossible to build. UML models were considered and rejected as these component dependencies are very important to get right so that the infrastructure is created in the correct order. These component level diagrams also show how security roles would be used to fulfil security requirement SR2.

3.4 Evaluation

As the project was developed using an incremental development method, it is suitable to be evaluated using a traditional "V-model" [102]

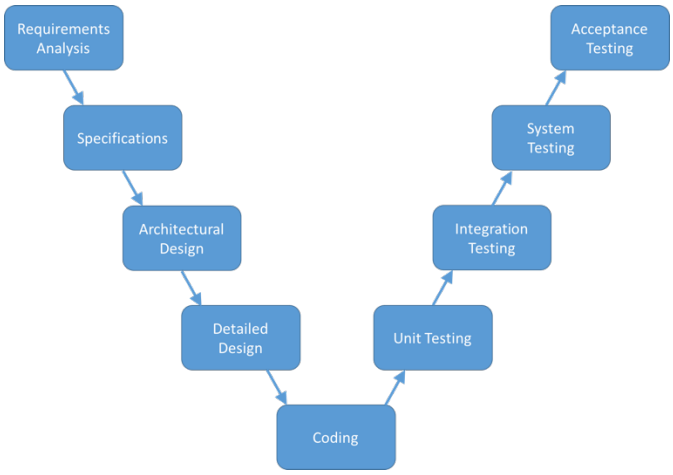


Figure 7: V-model for software development (after Mathur & Malik [102])

3.4.1 Unit testing

This consisted of uploading each individual component of the CloudFormation template separately to ensure that it was built successfully. The CloudFormation management console gives feedback on the success or failure of each build together with error messages. This was a slightly different approach to traditional software unit testing as it was more practical to use the CloudFormation console to test that the infrastructure was correctly built although this meant that an assumption had to be made that the infrastructure was working correctly if there were no CloudFormation error messages rather than by explicitly checking this. This was felt to be acceptable as this was covered by later testing.

3.4.2 Integration testing (part 1)

This consisted of creating CloudFormation templates that contained all the components required for each stage of the design and uploading them to AWS. Again, the CloudFormation management console gives feedback on the success or failure of each build together with error messages.

3.4.3 Integration testing (part 2)

Because the system was developed in an incremental way, once each stage of the design had been developed separately, they were then joined together into one large CloudFormation template and this again tested to check that the infrastructure was built successfully.

3.4.4 System testing

Once the infrastructure was built correctly, the functional, non-functional and security requirements were tested using synthetic test data and theoretical calculations as follows:

Objective	*Test*
FR 1: *The system must be able to process time series data (i.e. a value and a timestamp)* ***FR 2:*** *The system must be able to process data formatted in JSON* ***FR 3:*** *The system must be able to receive data via HTTP REST interface* ***FR 4:*** *The system should output an email alert in real time if an input signal goes above or below a certain threshold* ***FR 5:*** *The system should allow limited interactive visualisation of historical data* ***SR 3:*** *Systems sending data should not need to have access to AWS private keys to protect these keys.*	This will be tested through use of a Python script to send JSON formatted randomly generated timeseries data via HTTP PUT request. The script will request data from the DynamoDB database and the web app to compare against the original data and check for anomalies.
NFR 1: *The system must be able to scale to process data at speeds of up to 10MB/s*	This will be tested theoretically as it would be too expensive to test. Using assumptions of 10 shards at $0.015 per shard hour, with each record 1KB in size, Kinesis would cost $2.06 per hour to process the data. However, using assumptions of 128 bytes per data item and 1000ms processing time per data item for both Lambda functions (taken from actual processing time in CloudWatch logs), the Lambda functions would cost $613 per hour to run, far above the £100 limit (Constraint 5)
NFR 2: *The system must be able to scale to store data volumes of 10s of PBs*	This will be tested theoretically as it would be too expensive to test. Using assumptions of $0.25 per GB month for Dynamo storage, 10PB of data would cost $2.5m per month to store. Even using the cheapest storage, Glacier at $0.007 per GB month, 10 PB would cost $70,000 per month to store.

Objective	*Test*
NFR 3: *The visualisation app should automatically scale depending on the amount of people using it*	This will be tested by using Apache Bench to simulate a large enough number of concurrent users to set off the scaling alarms. The logs for ECS, EC2 and CloudWatch will show the actions that were automatically taken to increase the number of containers and EC2 instances.
SR 1: *The system should allow SSH access only to servers with a specified key*	This will be evaluated by attempting to SSH to an EC2 instance using the correct key, an alternative key and no key. The first should work and the latter two fail.
SR 2: *The system should enforce access controls internally to ensure access is limited only to individual components that need to have access.*	This will be evaluated through a network diagram showing how access is controlled through the system at each point where systems and users connect

Table 3: Evaluation tests

3.4.5 Acceptance testing

This set of testing is to determine whether the system actually solves the original business problem, which in this case is to show that using templates is faster than provisioning the infrastructure manually. This was tested by comparing the time taken to provision the system manually against the time taken by CloudFormation as given by the logs.

The whole system will also be presented back to the stakeholders at Amey and feedback sought.

3.5 Summary

This project will carry out a requirements gathering exercise followed by production of a requirements specification. An options analysis of AWS services will then be carried out to see which of these will best meet the requirements. Those services will be decomposed into the exact components required and the dependencies between the individual components analysed so that CloudFormation templates can be made specifying the exact infrastructure to be delivered. This infrastructure will then be tested in a variety of ways to ensure that it meets the initial requirements.

4 Results

4.1 Requirements specification

The requirements are presented using the Volere Requirements Specification Template as proposed by Robertson and Robertson [72]. A streamlined version has been used as it was felt too time-consuming to the use the full version.

4.1.1 The purpose of the project

Amey want to use templates to provision the infrastructure for a big data analytics platform so that they can quickly and repeatably update the current platform or make new copies which can be kept synchronised. The time taken to provision the infrastructure through templates should be less than that taken to provision the infrastructure manually.

4.1.2 Stakeholders

Dr. Pedro Baiz (Amey)
Dr. Richard Mathie (Amey)

4.1.3 Constraints

Constraint 1: The big data platform shall be constructed from services available from Amazon Web Services to reduce complexity and dependencies on third party services.
Constraint 2: The templates shall be written in JSON format in the CloudFormation style to reduce complexity and dependencies on third party services.
Constraint 3: Docker shall be used as a container for any front end application to replicate the current Amey system.
Constraint 4: The whole project needs to be completed in three months due to the MSc dissertation deadline.
Constraint 5: Testing and development of the system should be completed using a separate AWS account and within the $35 free credit available to student users of AWS and the free tier with up to £100 available from Amey if required.
Constraint 6: Testing and development needs to be carried out on synthetic data rather than real customer data to protect the customer data.
Constraint 7: Developer has limited technical skills in this area, all will have to be learnt from scratch
Constraint 8: Any programming will be done in Python as the developer has experience of this

4.1.4 Assumptions

Assumption 1: Amey will not use the big data analytics platform directly but will use it as a starting point to develop further to suit their specific needs
Assumption 2: As Constraints 1 and 2 require the use of AWS services, the resulting templates do not need to be vendor neutral.

4.1.5 Scope

Scope 1: Advanced network security (e.g. Virtual Private Clouds) beyond basic access controls is out of scope due to time constraints (Constraint 4)

Scope 2: Data cleansing and validation is out of scope due to the need to use synthetic rather than real customer data and time constraints (Constraints 4 and 6)
Scope 3: The customer-facing web app only needs to be a proof of concept rather than a full featured application due to time constraints (Constraint 4)
Scope 4: While Amey has wider requirements for data types, the system only needs to be able to process time series data formatted in JSON and sent via HTTP REST interface due to time constraints (Constraint 4)
Scope 5: Sending messages back to IoT sensors is out of scope due to the need to use synthetic rather than real customer data and time constraints (Constraints 4 and 6)

4.1.6 Functional requirements

FR 1: The system must be able to process time series data (i.e. a value and a timestamp)
FR 2: The system must be able to process data formatted in JSON
FR 3: The system must be able to receive data via HTTP REST interface
FR 4: The system should output an email alert in real time if an input signal goes above or below a certain threshold
FR 5: The system should allow limited visualisation of historical data

4.1.7 Non-Functional requirements

NFR 1: The system must be able to scale to process data at speeds of up to 10MB/s based on projected forecasts
NFR 2: The system must be able to scale to store data volumes of 10s of PBs based on projected forecasts
NFR 3: The visualisation app should automatically scale depending on the amount of people using it

4.1.8 Security requirements

SR 1: The system should allow SSH access only to servers with a specified key
SR 2: The system should enforce access controls internally to ensure access is limited only to individual components that need to have access.
SR 3: Systems sending data should not need to have access to AWS private keys to protect these keys.

4.2 Options analysis of AWS components

4.2.1 Collect

Data Ingest
When dealing with a potentially large number of sensors all sending data, there needs to be a central collection point for this data before it is sent to be processed.

Option 1: API Gateway
This allows the setup of a REST API that can be sent data over HTTPS and either provide a response or simply map the data to a different data standard and send to another AWS service such as Lambda or Kinesis [73].

Option 2: IoT
The IoT service allows sensors to communicate with AWS via HTTPS, WebSockets or MQTT. It also allows messages to be sent back to the sensors to change their state through the Device Shadows

service and stores the details of the sensors in the Registry. Once the messages have been received they can be sent to other AWS services such as Lambda or Kinesis using the Rules Engine [74]

Options	*Pros*	*Cons*
API Gateway	• Flexible • Don't need to know what Things you have in advance • Cheaper than IoT ($3.50 per million messages vs $5 for IoT)	• Can't send messages back to Things • Can only use HTTPS
IoT	• Able to communicate with Things in both directions • Can keep track of Things via the Registry • Can use lightweight messaging protocols such as MQTT	• Not available in CloudFormation at beginning of project [75]

Table 4: Options analysis for Data Ingest

Although the IoT service would be a superior long-term answer, the API Gateway route was chosen due to the fact that it fulfils requirements FR 1, FR 2, FR 3 and SR 3, was accessible via CloudFormation at the time the choice was made (Constraint 2), and the main focus of the project is on analysis of IoT data rather than control of the sensors, so it was deemed not important to be able to send messages back to the sensors (Scope 5).

Data transport

Once the data has been ingested into the system, it needs to be sent to the right places for storage and processing. Using streaming / queuing systems smooths the load on the follow on processing and allows for scaling and resilience. The requirements are that data needs to be processed in real time so that a threshold alert email can be generated and that velocities of up to 10MB/s can be supported. There are four main options for data transport in AWS:

Option 1: Data Pipeline

This allows scheduled data movements and processing across the system as a managed service [76]

Option 2: Kinesis Firehose

This is a version of Kinesis which is designed simply to move large quantities of data very fast into data stores, primarily S3, Redshift or ElasticSearch. Data is batched from the stream by either time interval or size (minimum 60 seconds or 1MB) and then sent directly to the relevant store. Each record can be up to 1000KB in size and each stream can handle up to 5MB/s. Data is kept in the stream for 24 hours in case of failures.

Option 3: Kinesis Streams

Kinesis Streams is designed to allow processing of streaming data in real time with data available to read within a second of it being written to the stream. The stream can be processed by multiple processors as the messages are not deleted when they are read. Each stream is divided into "shards" which can have up to 1MB/s written to them and 2MB/s read from them with up to 5 reads per second. To scale up the streams, it is just a case of adding more shards. Data is kept in the stream for 24 hours in case of failures [77].

Option 4: Simple Queue Service – SQS

SQS is a queue service that stores messages while they are waiting to be processed [78]. The messages can be consumed by multiple processors but they can only be processed once before being deleted. The system does not enforce a strict "first in, first out" order that the messages are processed in as it

is a distributed service. Messages can be up to 256KB in size and are retained on the queue for up to 4 days in case of problems [79].

Option 5: Third party messaging application on EC2, e.g. Kafka
Apache Kafka is an open source distributed publish / subscribe messaging system that is the main competitor to Kinesis that can handle "hundreds of megabytes of reads and writes per second" [80]. While this can't be used as a managed service, it could be installed on a EC2 virtual server and used that way. Kafka also has a stream processing library called Kafka Streams. Comparisons of Kafka with Kinesis by Schreiter in January 2016 [81] and Ramachandra in May 2016 [82] agreed that Kafka performed better technically but was harder to set up than Kinesis. Ramachandra calculated that Kinesis was over twice as expensive as running Kafka on a reserved instance EC2, however this relies on there being a constant data ingestion rate as the EC2 instance would have to be paid for whether data was being processed or not, whereas Kinesis is paid for in direct proportion to the data processed.

Options	*Pros*	*Cons*
Data Pipeline		• Not event-driven, needs to be scheduled
Kinesis Firehose	• Managed service • Easy to scale and make resilient over multiple availability zones	• Can't process data directly from stream, has to be indirectly accessed from S3 / Redshift / ElasticSearch • Data is batched, not real time
Kinesis Streams	• Managed service • Easy to scale and make resilient over multiple availability zones • Can process data multiple times • Data can be processed in real time	• Potentially not as fast as Kafka
SQS	• Managed service • Easy to scale and make resilient over multiple availability zones	• Can't process data multiple times
Third party messaging application on EC2, e.g. Kafka	• Potentially better performance than Kinesis	• Not a managed service, needs technical skill to set up, manage and monitor

Table 5: Options analysis for Data transport

It was decided to use the Kinesis Streams component to move the data into the system once it had been collected. This is because it meets requirement NFR 1 for processing up to 10MB/s (with 10 shards) and the stream is available to be processed by many customers, unlike with SQS, so there can be a real time threshold analysis on the data (FR 4) as well as processing of the data into a data store (FR 5). I decided against Kinesis Firehose as it didn't allow for the real time analysis, and against Kafka on EC2 as it needed too much technical expertise to set up (Constraints 4 and 7).

4.2.2 Store

The data needs to be stored so that analysis of historical data can be carried out. The requirement for Volume is that the amount of data that can be stored and queried should be in the range of PBs.

Option 1: DynamoDB
DynamoDB is a NoSQL database that is designed to be fast and available using SSD storage. Each table requires only a primary key to identify each item, but there is no predefined schema so each item can

have different attributes. In NoSQL terms it can act as a document store or a key-value store. There are no limits on the number of attributes each item can have up to a maximum item size of 400KB. There is also no limit on the size of the table and a nominal limit of 10,000 reads/s and 10,000 writes/s that can be increased if necessary [83].

Option 2: ElastiCache

Elasticache manages in-memory data storage allowing very fast data access [84]. This is useful if data is being accessed and used almost continually, for example today's data might be heavily used, whereas yesterday's data is much less of interest.

Option 3: Relational Database Service (RDS)

RDS is a managed service which provides a traditional relational database such as MySQL, Oracle or Amazon's own relational database, Aurora [85].

Option 4: Redshift

Redshift is a structured columnar database which has been optimised for large scale data analysis, which can be done via SQL. It can be specified as either "Dense Storage" which uses normal hard drives and is cheaper for very large volumes of data but not as fast as "Dense Compute" which is optimised for fast data analyses using SSD drives [86].

Option 5: Simple Storage Service (S3)

S3 is a document store that is good for storing large amounts of unstructured data, for example Amazon use it themselves to store the Amazon.com web site. Each item can be up to 5TB in size. There is also a version called "Glacier" which can be used for cheap long term data storage in return for data access times measured in hours [87].

Option 6: Third party database on EC2 (e.g. Cassandra)

DB-Engines ranks databases based on how much interest there is in them on the web, and puts Cassandra [88] as the top NoSQL columnar database [89], above DynamoDB. Gessert *et al*. [65] (see Figure 5) recommend that Cassandra and DynamoDB are good for simple queries and fast lookups, but that Cassandra is better for high availability whereas DynamoDB is better for consistency.

Options	*Pros*	*Cons*
DynamoDB	• Managed service • Automatically resilient across multiple Availability Zones and potential to run across Regions • Highly scalable with no downtime • Fine-grained item level access control	• May take up to 1 second for the database to be consistent • Does not support complex querying • Can't be deployed inside a VPC
Elasticache	• Very fast lookup	• Very expensive
RDS	• Managed service • Can be deployed inside a VPC • Can be deployed across multiple availability zones	• Needs a defined schema • Not easily scalable
Redshift	• Managed service • Can store PBs of data • Supports complex queries • Managed service including backup and replication	• Needs a defined schema • Can only be run in a single Availability Zone • Scaling requires a short outage

Options	*Pros*	*Cons*
	• Data automatically encrypted at rest and in transit, can be part of a VPC	
S3	• Managed service • Relatively cheap • Automatically resilient across multiple Availability Zones and potential to run across Regions • Data can be encrypted and access control granted at bucket level • Can set up lifecycle management of data	• Not designed for data analysis
Third party database on EC2 (e.g. Cassandra)	• Can choose any database	• Needs technical skills to set up and manage the database

Table 6: Options analysis for data storage

It was decided to use a DynamoDB database for storage because of its flexibility and scalability (NFR 2), however there are arguments for using Redshift instead if a defined schema could be used as it would allow more complex queries. It was decided that the speed of Elasticache was not necessary to fulfil FR 5, S3 was rejected because it was not designed for data analysis, and RDS because it was not easily scalable (NFR 2). A third party database on EC2 was also rejected because of the extra technical skills it would require (Constraints 4 and 7).

4.2.3 Process

There are several places where processing needs to be carried out, for example, inputting the data received into a data store, checking each reading against a threshold in order to send out an email alert, and performing historical analysis on data.

Option 1: Amazon Machine Learning
This is a managed machine learning service where you can create simple classification or regression machine learning models and use them on incoming data to make predictions [90].

Option 2: Data Pipeline
As previously mentioned in 4.2.1, this allows scheduled data movements and processing across the system as a managed service [76]

Option 3: Elastic Map Reduce (EMR)
EMR is a managed Hadoop cluster based on EC2 that can be used for transforming large quantities of unstructured or semistructured data through parallel processing using MapReduce or Spark. It can also use Hive (a SQL-like querying language) to analyse data stored in DynamoDB [91].

Option 4: Lambda
Lambda is a way of running code without having to provision a server for it to run on. It is designed for light-weight event driven code and supports Node.js, Python and Java. The code is automatically scaled as a new copy of the code is run every time it is triggered by an event, but because of this it needs to be run in a stateless fashion (i.e. variables can't be persisted within the code between events), however stateful objects can be stored in data stores such as S3 or DynamoDB and accessed when the code is run. There is a time limit of 5 minutes for each execution of the code [92].

Option 5: Third party analysis application on EC2 (e.g. Storm)
Apache Storm [93] is a fast scalable streaming processing system which is an alternative to EMR and Spark Streaming.

Options	*Pros*	*Cons*
Amazon Machine Learning	• Simple to use	• Can only read data from S3, RedShift or RDS • Can't do streaming analytics or more complex models
Data Pipeline	• Could be useful for running regular queries on historical data	• Scheduled, not event driven
EMR	• Managed service • Good for unstructured or semistructured data • Can read data from DynamoDB or S3	• Costs extra to the number of EC2 instances needed to run it
Lambda	• Good for code that is triggered in response to events • Highly scalable • Code is encrypted at rest • Can use IAM roles to determine exactly what the Lambda code has access to • Don't need to worry about setting up servers • Pay only when the code is triggered	• Code has to be stateless
Third party analysis application on EC2 (e.g. Storm)	• Storm does genuinely real time analytics whereas Spark streaming uses microbatches of data	• Needs technical skills in Storm • Need to scale EC2 instances separately

Table 7: Options analysis for data processing

Lambda functions were chosen for processing data into DynamoDB and for checking for the threshold as they were very simple functions which could be programmed quickly in Python (Constraints 4, 7 and 8). Storm on EC2 was decided against because it is more complicated to scale as it is based on EC2, and EMR as it was too complex (Constraints 4 and 7). However, these are both relevant options as the analysis platform becomes more complex, as is potentially use of the Data Pipeline to run regular batch queries on historical data.

4.2.4 Consume

The analysis results are consumed in two places, firstly by an email alert when a threshold is breached, and secondly when investigating historical data.

Email alerts
Option 1: Simple Notification Service (SNS)
SNS sends messages to topics to which users can subscribe to email alerts.

SNS was chosen for the email alerts as it was the only sensible option (FR 4).

Display of historical data

Option 1: ElasticSearch Service

This is a managed service for ElasticSearch which is the top-rated enterprise search engine according to DB-Engines [94].

Option 2: In-house web application based on EC2

This option allows for a tailor made web application to be developed. This application can also be packaged into a container using e.g. Docker to make it easier to deploy. These containers can be managed and scaled by using the EC2 Container Service (ECS) or using Docker's inbuilt swarm mode which has been released in the latest beta version (1.12)

Option 3: Quicksight

Amazon has developed its own Business Intelligence (BI) tool, Quicksight, which is currently still only available in a "preview" form in one Region [95]. It provides simple visualisations and is aimed at corporate users.

Option 4: Serverless web application

Serverless web applications use Lambda functions and the API Gateway to create web pages on the fly without having to set up EC2 instances (see 2.4.8 for more information). This means that they are automatically and infinitely scalable and don't cost anything to run if nobody accesses them. Setting all these functions up can be fiddly so a Serverless Framework [96] has been developed to help with building these types of applications.

Option 5: Third party visualisation application on EC2 e.g. Tableau

As with all the other parts of the chain, it is possible to use third party software on EC2. Tableau [97] is a major player in the business analytics and visualisation software market and can be configured to run scalably on EC2 [98].

Options	*Pros*	*Cons*
ElasticSearch	• Good at search	• Search not really relevant for time series data
In-house web app on EC2	• Application can be built to specific requirements • Application can be aimed at any level of user • Application can be moved to another service	• Need technical skills to build and maintain web app • Need technical skills to manage Docker containers and EC2 scaling • Docker 1.12 still in beta
Quicksight	• Managed service – easy to set up	• Aimed at professional users • Cost of licencing
Serverless web app	• Very scalable	• Need technical skills to build apps in serverless manner • Application is locked in to AWS
Third party application on EC2 e.g. Tableau	• No skills needed to build application • Lots of in built functionality	• Aimed at professional users • Cost of licencing • Technical skills required to maintain software and scale EC2 instances

Table 8: Options analysis for data consumption

It was decided to write a very simple web application (using the Flask framework [99] for Python (Constraint 8)) and run it on Docker and EC2 because it is a very flexible option and it gives a good demonstration of using Docker and EC2 via CloudFormation (Constraints 2 and 3). It was also decided to use ECS to manage the containers rather than the new Docker Swarm mode as this was still in beta and access wasn't available at the start of the project (Constraints 4 and 7). Quicksight was rejected because of the costs (Constraint 5) and the fact that it is still a preview version (Constraints 4 and 7). Tableau was also rejected because of the costs (Constraint 5) and the technical skills required (Constraints 4 and 7). The serverless web app could be a worthwhile project in its own right, but it was also rejected because of the technical skills required (Constraints 4 and 7).

4.3 System design

4.3.1 Stage 1: Putting data into a data store

Stage 1 focussed on just importing data into DynamoDB. For simplicity, the test data was sent direct to the Kinesis stream rather than using the API gateway. A Lambda function was used to import data into DynamoDB so that the input data could be reformatted. If any data cleaning functions needed to be added in future they could be done here too.

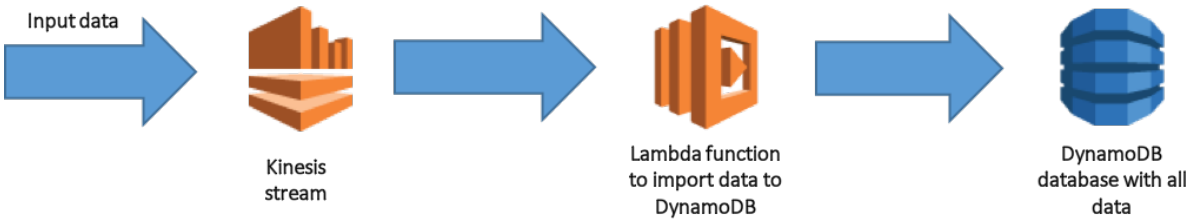


Figure 8: System level design for Stage 1 (a subset of Figure 20)

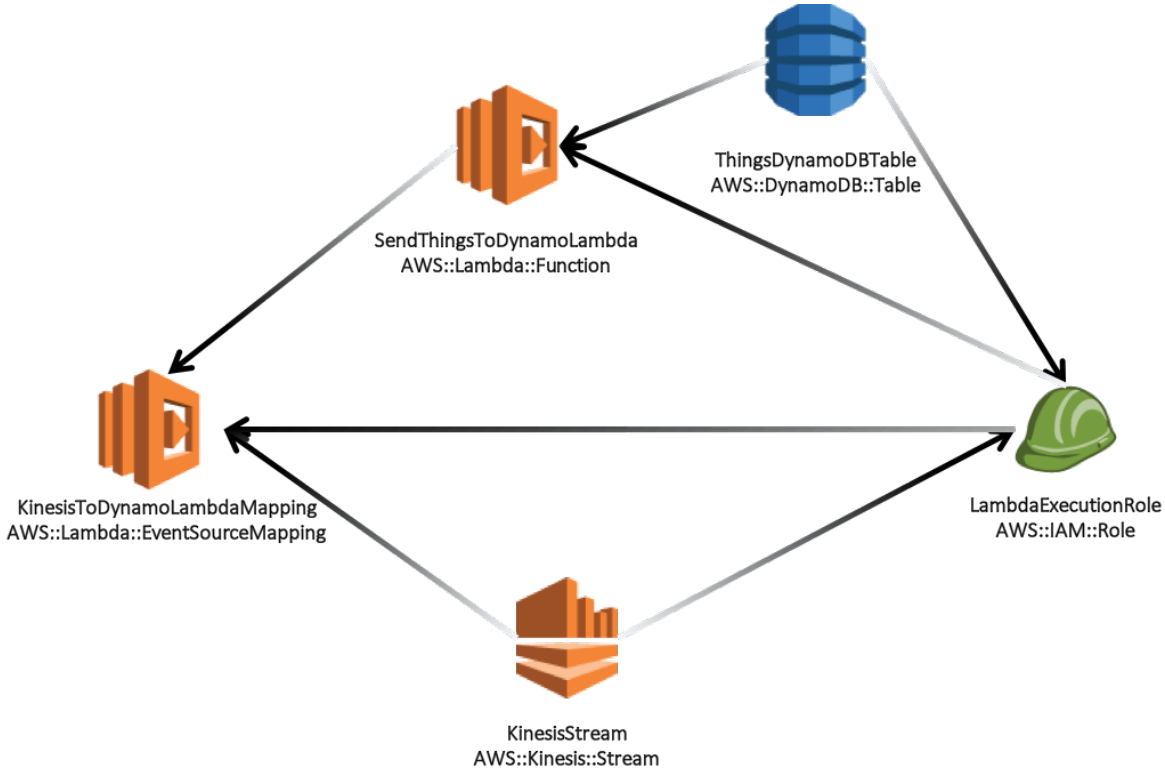


Figure 9: Component level design for Stage 1 with dependencies

Component	*Function*
KinesisStream	Stream that moves the data around
KinesisToDynamoLambdaMapping	Links the Kinesis stream to the Lambda function
LambdaExecutionRole	IAM role that allows the Lambda function to access the Kinesis stream and the DynamoDB table
SendThingsToDynamoLambda	Function that takes each data item from the stream and puts it into DynamoDB
ThingsDynamoDBTable	Table to contain the data

Table 9: Description of components in Stage 1

4.3.2 Stage 2: Dockerised web app

Stage 2 focussed on building a simple "hello world" web app that just displayed the phrase "Hello World!" in a browser. For simplicity, scaling of either the EC2 instances or the containers was ignored, the web app did not depend on access to DynamoDB and an existing Docker hub repository was used rather than building an ECR Repository.

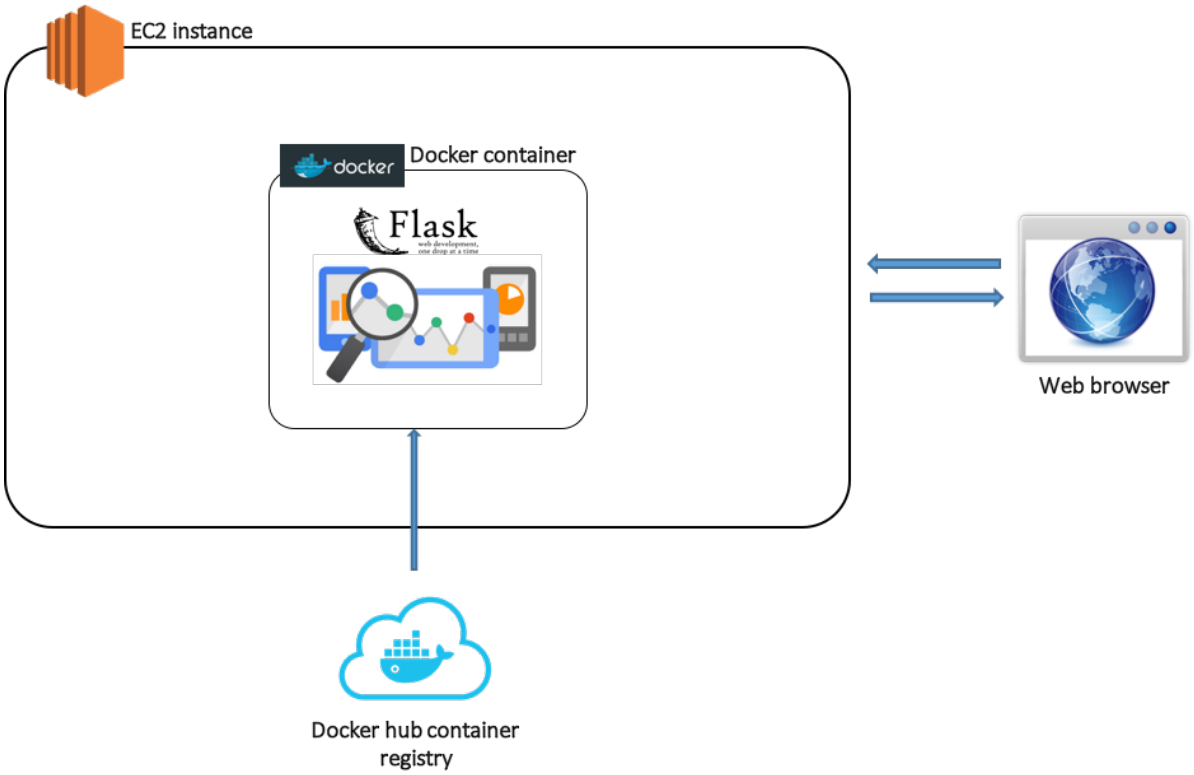


Figure 10: System level design for Stage 2

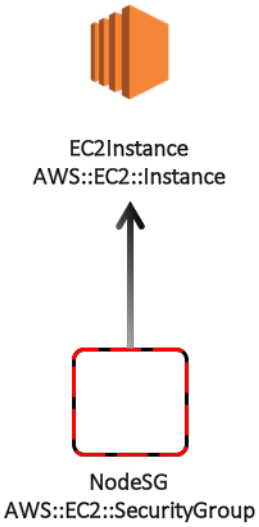


Figure 11: Component level design for Stage 2 with dependencies

Component	*Function*
EC2Instance	EC2 server that installs Docker and pulls an image from the Docker hub
NodeSG	Security group that allows SSH and web access to the server

Table 10: Description of components for Stage 2

4.3.3 Stage 3: Combining stages 1 and 2

Stage 3 focussed on combining stages 2 and 3. This involved writing a new Flask web app that displayed the data in the DynamoDB table, and adding a connection between the EC2 instance the web app was on and the DynamoDB table (see Appendix 3 for the web app code).

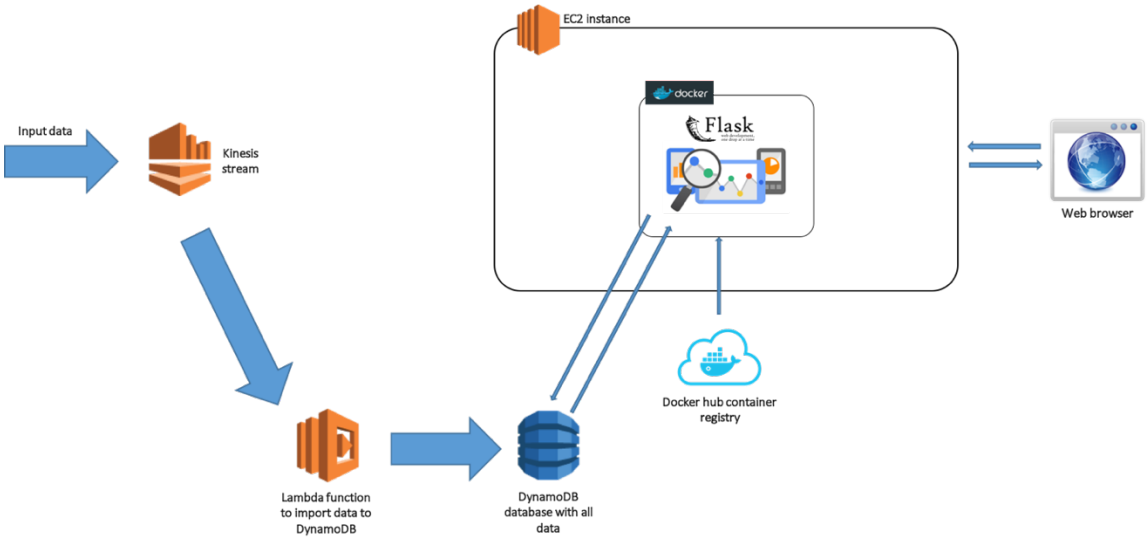


Figure 12: System level design for Stage 3

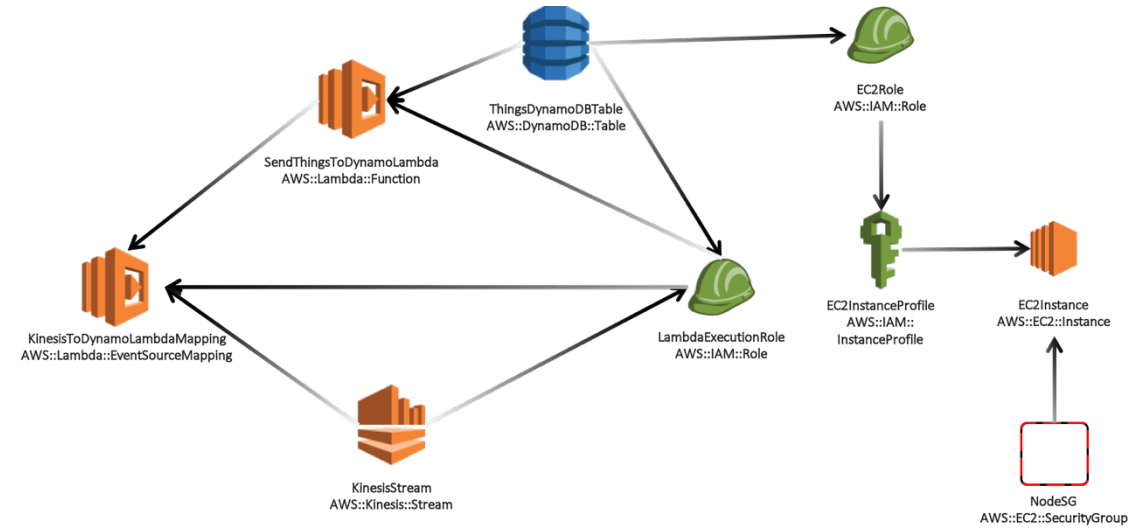


Figure 13: Component level design for Stage 3 with dependencies

Component	*Function*
EC2Instance	EC2 server that installs Docker and pulls an image from the Docker hub
EC2InstanceProfile	Link between the EC2 instance and the EC2 role
EC2Role	Role that allows the EC2 instance to get access to the DynamoDB table
KinesisStream	Stream that moves the data around
KinesisToDynamoLambdaMapping	Links the Kinesis stream to the Lambda function
LambdaExecutionRole	IAM role that allows the Lambda function to access the Kinesis stream and the DynamoDB table
NodeSG	Security group that allows SSH and web access to the server
SendThingsToDynamoLambda	Function that takes each data item from the stream and puts it into DynamoDB
ThingsDynamoDBTable	Table to contain the data

Table 11: Description of components for Stage 3

4.3.4 Stage 4: Adding real time threshold analysis and email alerts

Stage 4 focussed on processing data from the Kinesis stream, testing it against thresholds stored in a DynamoDB table, and then the data item is higher than the maximum threshold or lower than the minimum threshold it then sends a message to an SNS topic which is then emailed to any subscribers.

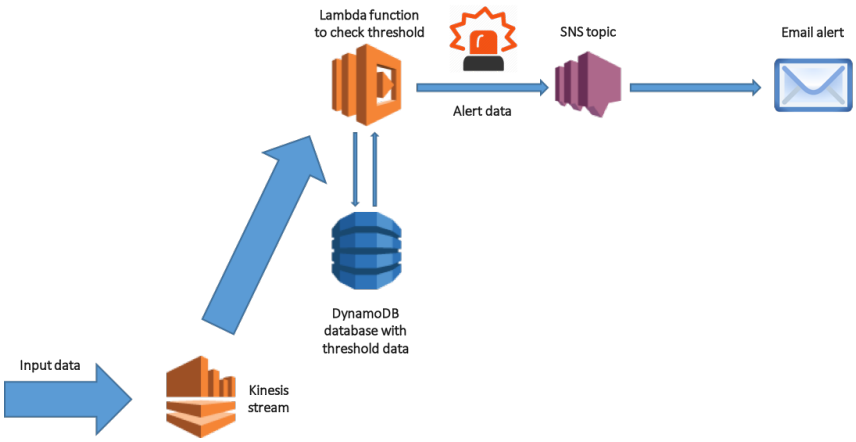


Figure 14: System level design for Stage 4

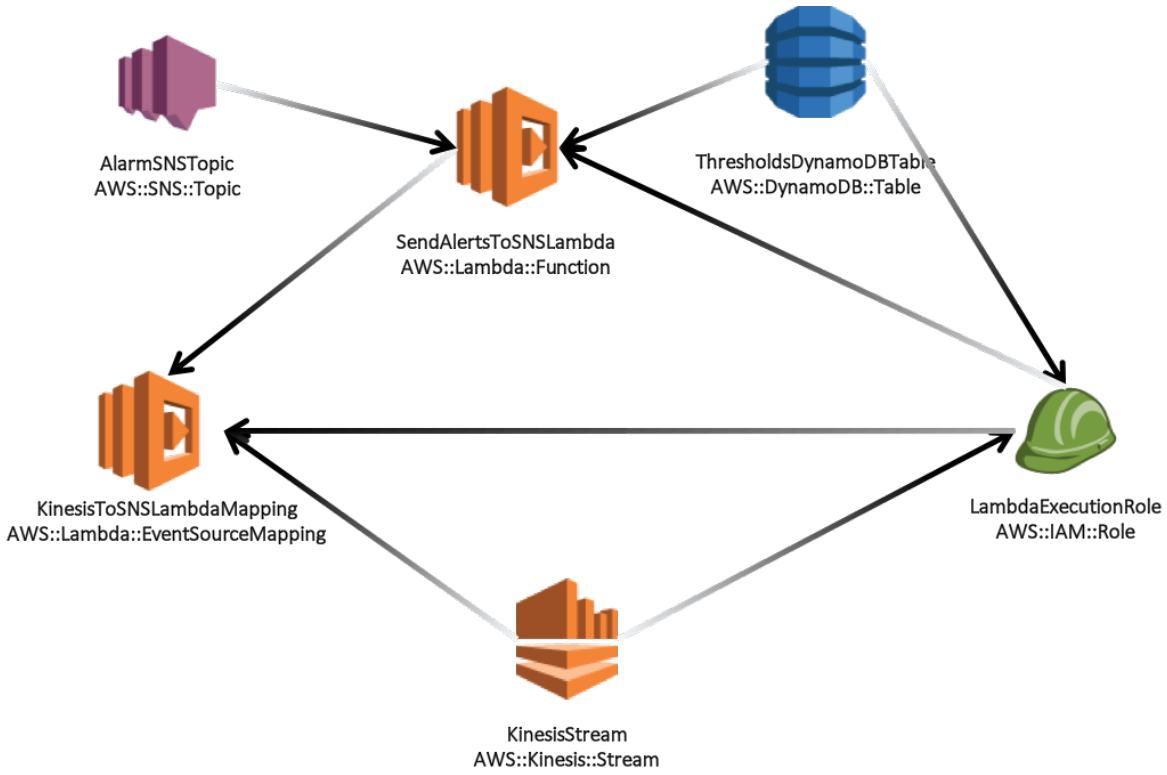


Figure 15: Component level design for Stage 4 with dependencies

Component	*Function*
KinesisStream	Stream that moves the data around
KinesisToSNSLambdaMapping	Links the Kinesis stream to the Lambda function
LambdaExecutionRole	IAM role that allows the Lambda function to access the Kinesis stream and SNS
SendAlertsToSNSLambda	Function that takes each data item from the stream, checks it against the threshold stored in the DynamoDB table and if it breaches it, sends a message to the SNS topic
ThresholdsDynamoDBTable	Table to contain the threshold data

Table 12: Description of components in Stage 4

4.3.5 Stage 5: Scaling of web app

This stage focussed on taking the "hello world" web app used in Stage 2 and using ECS to scale the EC2 instances and the Docker containers. It also replaced the Docker Hub repository with an ECR one. For simplicity it did not try to use the web app used in Stage 3 or connect the web app to DynamoDB.

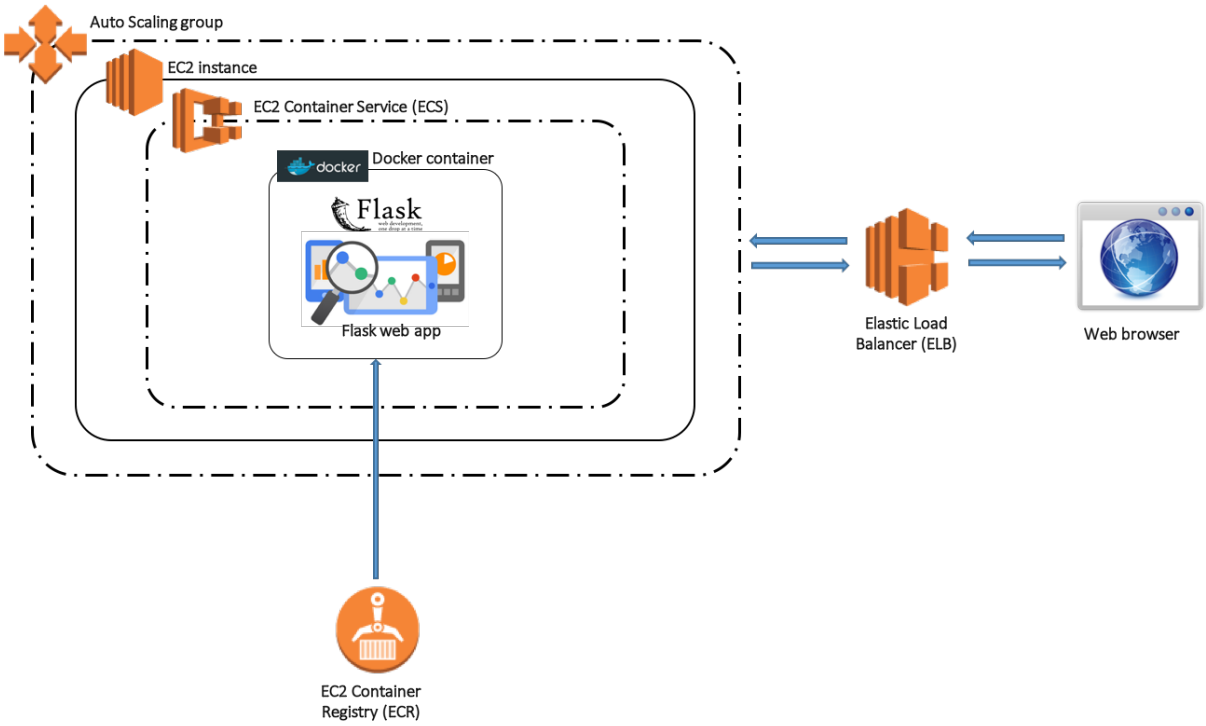


Figure 16: System level design for Stage 5

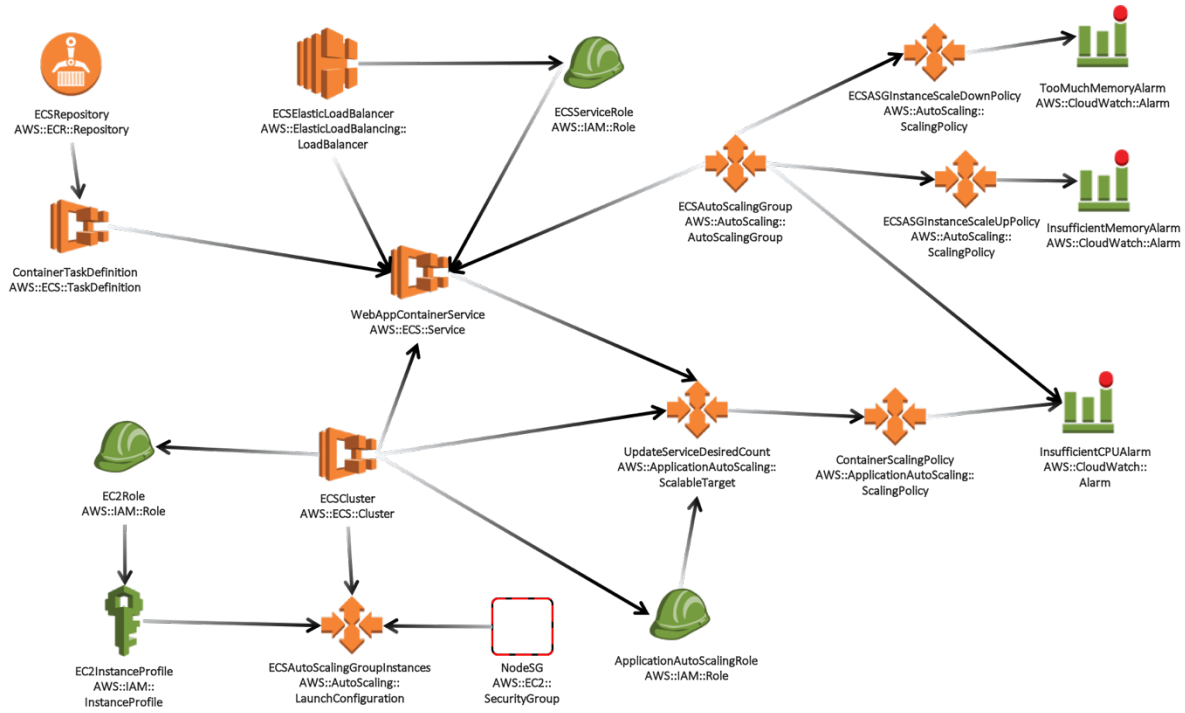


Figure 17: Component level design for Stage 5 with dependencies

Component	*Function*
ApplicationAutoScalingRole	Role that allows the Application AutoScaling service to add and remove Docker containers from the ECS Cluster
ContainerScalingPolicy	Policy that links the InsufficientCPUAlarm to a policy that adds an extra container when the alarm goes off
ContainerTaskDefinition	Details of Docker container to be run
EC2InstanceProfile	Link between the EC2 instances and the EC2 role
EC2Role	Role that allows the EC2 instances to get access to the container registry and manage containers
ECSASGInstanceScaleDownPolicy	Policy that links the TooMuchMemoryAlarm to a policy that removes an EC2 instance when the alarm goes off
ECSASGScaleUpPolicy	Policy that links the InsufficientMemoryAlarm to a policy that adds an EC2 instance when the alarm goes off
ECSAutoScalingGroup	Group of EC2 instances that can be scaled up or down to a given maximum and minimum
ECSAutoScalingGroupInstances	Description of how to launch the EC2 instances and what to install on them
ECSCluster	Cluster of EC2 instances to run Docker containers on
ECSElasticLoadBalancer	Load Balancer to provide an entry point to the Auto Scaling Group of EC2 instances
ECSRepository	Repository for Docker images
ECSServiceRole	IAM Role to allow the ECS Cluster to add and remove EC2 instances from the Load Balancer
InsufficientCPUAlarm	Alarm that goes off if CPU Utilisation goes above a certain value – showing that more copies of the webapp are needed
InsufficientMemoryAlarm	Alarm that goes off when amount of reserved memory across the cluster goes above a certain value – showing that there aren't enough EC2 instances to add an extra container if required
NodeSG	Security group that allows SSH and web access to the EC2 instances
TooMuchMemoryAlarm	Alarm that goes off when amount of reserved memory across the cluster goes below a certain value – showing that there are wasted EC2 resources
UpdateServiceDesiredCount	Describes what needs to be scaled – in this case the number of running containers
WebAppContainerService	Links to containers that need to be run on the ECS Cluster

Table 13: Description of components used in Stage 5

4.3.6 Stage 6: HTTP REST interface

This stage focussed on getting data into the Kinesis stream via the API gateway.

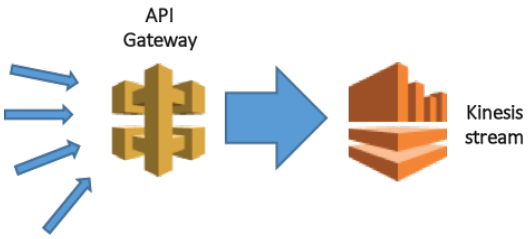


Figure 18: System level design for Stage 6

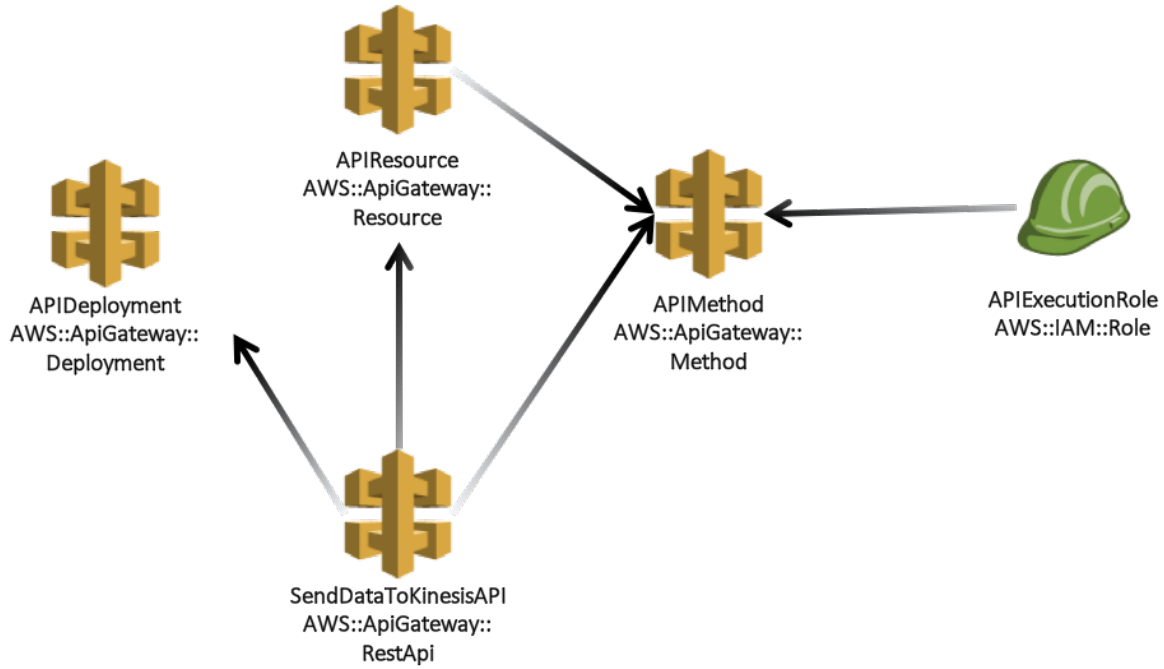


Figure 19: Component level design for Stage 6 with dependencies

Component	*Function*
APIDeployment	Publishes API
APIExecutionRole	Role that allows the API to send data to Kinesis
APIMethod	Translates JSON data to correct format for Kinesis stream
APIResource	Creates a subsection of the API to receive data
SendDataToKinesisAPI	Overall API details

Table 14: Description of components used in Stage 6

4.3.7 Stage 7: Combining all stages

This stage focussed on bringing all the previous stages together to form the overall system design in Figure 20. This involved reinstating the web app that displayed data from the DynamoDB table and the link between the EC2 instances and the DynamoDB table.

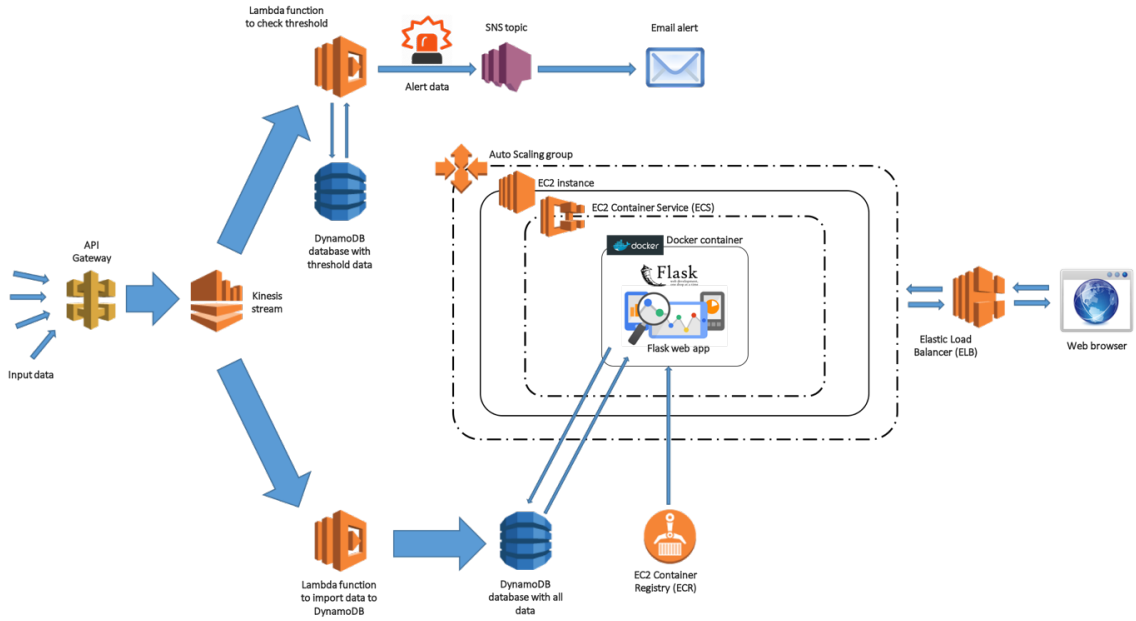


Figure 20: Overall system level design for the system

4.4 Implementation of CloudFormation templates

All the components needed to implement the design in Figure 20 and their parameters were then defined in JSON as explained in 3.3.1 to make a CloudFormation template. It was not possible to produce one large template that covered everything and so three smaller templates were produced (see Appendix 10 for the full templates). The first covers the setting up of the Repository for the Docker image. This needed to be set up separately as the Docker image needs to be uploaded to it before the next part of the template is run. The second is the "Main" template which covers nearly all of the infrastructure. The third has been called the "Service Scaling" template. This couldn't be part of the Main template as it is not possible to use the internal CloudFormation functions to get the correct format for the ResourceID field in the ScalableTarget Resource. This is almost certainly because it is very new functionality in CloudFormation. To work around this problem this resource and everything depending on it had to be put into a separate template and parameters used to input the relevant details from the Main stack (this process is described in the User Documentation in Appendix 4).

Overall there are 35 separate Resources, 5 Parameters and 3 Outputs across the just over 1000 lines of code in the three templates. One of the main problems is managing this information in an understandable way. To mitigate this, within each template the various resources were listed with meaningful names in alphabetical order of their "Types". This meant that similar resources were grouped together. The metadata section was used to provide comments where possible as JSON doesn't allow the use of comments as would be traditionally used in code.

The CloudFormation template language allows two types of interaction with the infrastructure. For most of the time simple predefined parameters are used. The simplest example of this is the ECS Cluster which sets up a group of EC2 instances in order to manage the containers on them. This resource just assigns a name to an instance of the ECS Cluster type.

```
"ECSCluster": {
  "Type": "AWS::ECS::Cluster"
            },
```

Figure 21: Excerpt of CloudFormation template for ECSCluster

However, the most complicated Resources contain embedded code such as shell scripts or Python. The way this is embedded can make the JSON code look messy and difficult to read as the scripts are generally made up using a Fn::Join command to join lines of text together. The best examples of inserting whole scripts are the Lambda functions SendAlertsToSNSLambda and SendDataToDynamoLambda on lines 717-792 of Main.json (Appendix 10). Here the "Code" element contains the whole code in a Fn::Join statement. The extra punctuation to enable this text joining (i.e. quotes, commas and \n for new line) make it difficult to read.

```
        "SendThingsToDynamoLambda": {
            "Type": "AWS::Lambda::Function",
            "Properties": {
                "Code": {
                    "ZipFile": {
                        "Fn::Join": ["", [
                            "from __future__ import print_function\n",
                            "import base64\n",
                            "import boto3\n",
                            "import json\n",
                            "print('Loading function')\n",
                            "def lambda_handler(event, context):\n",
                            "    dynamodb = boto3.resource('dynamodb')\n",
                            "    table = dynamodb.Table('ThingsDynamoDBTable')\n",
                            "    for record in event['Records']:\n",
                            "      payload =
    json.loads(base64.b64decode(record['kinesis']['data']))\n",
                            "      thingid = payload['id'] + '/temperature'\n",
                            "      table.put_item(Item={'ThingID':thingid,
    'Timestamp':payload['timestamp'], 'Value':payload['temperature']})\n",
                            "    return 'Successfully processed {}
    records.'.format(len(event['Records']))\n"
                        ]]
                    }
                },
                "Runtime": "python2.7"
            }
        },
```

Figure 22: Excerpt of CloudFormation template for SendThingsToDynamoLambda

Another complicated Resource is the ECSAutoScalingGroupInstances which includes bash shell scripts to run configuration tasks on the EC2 instances as they are launched.

```
        "ECSAutoScalingGroupInstances": {
            "Type": "AWS::AutoScaling::LaunchConfiguration",
            "Metadata": {
                "AWS::CloudFormation::Init": {
                    "config": {

                        "commands": {
                            "01_add_instance_to_cluster": {
                                "command": {
                                    "Fn::Join": ["", ["#!/bin/bash\n", "echo ECS_CLUSTER=", {
                                        "Ref": "ECSCluster"
                                    }, " >> /etc/ecs/ecs.config"]]
                                }
                            }
                        },
```

Figure 23: Excerpt of CloudFormation template for ECSAutoScalingGroupInstances

In some of the situations, it is possible to put long code sections into the AWS file database S3 and reference them from the template. This is a good thing to do if code is long and doesn't rely on knowing the exact name of an infrastructure component. Some components (but not all) can be given friendly names that can be used instead of the AWS assigned names (AWS calls these "Logical IDs"). These can then be hard coded into scripts. The downside of this is that If the template is changed and these infrastructure components renamed, the scripts need to be edited too which will almost certainly be overlooked as there is no direct link. In summary, there is a difficult decision to be made between having complicated and potentially messy code in the template which can take advantage of being automatically updated if something changes in the infrastructure, or tucking it neatly away in S3 but having to remember to update it manually if there are changes that affect it. For this project it was decided to have all the code embedded directly in the templates as each code snippet was relatively short and it was useful to have the automatic links to the infrastructure.

For more detailed descriptions of the function of each piece of each template and their relationship to the requirements, see Appendix 9.

4.5 Evaluation test results

4.5.1 Unit tests and Integration test (Part 1) results

These tests were run continuously during development. For reasons of space the results have been left out of this report as they are effectively covered by the results for Integration tests (part 2)

4.5.2 Integration test (Part 2) results

This test involved uploading the CloudFormation templates as described in Appendix 4. The infrastructure stack was built successfully as shown by the screenshots of the CloudFormation Management Console in Appendix 5.

4.5.3 System testing results

This table relates to the table of proposed tests in section 3.4.4 and gives the details of the testing process. Detailed test results are available in Appendices 6,7 and 8 as described. Overall, all the requirements were successfully fulfilled apart from NFR 3 which was partially fulfilled with the automatic app scaling working correctly but the web app itself needing extra software to process a large number of concurrent users.

Objective	*Test*
FR 1: *The system must be able to process time series data (i.e. a value and a timestamp)* ***FR 2:*** *The system must be able to process data formatted in JSON* ***FR 3:*** *The system must be able to receive data via HTTP REST interface* ***FR 4:*** *The system should output an email alert in real time if an input signal goes above or below a certain threshold* ***FR 5:*** *The system should allow limited visualisation of historical data* ***SR 3:*** *Systems sending data should not need to have access to AWS private keys to protect these keys.*	The Python test script to test these requirements can be found in Appendix 6. The script has access to the AWS keys in order to access DynamoDB directly to verify the data has been stored successfully, but does not use these to send the data. The results of the testing are in Appendix 7. These show that 10 data items were sent to the system and successfully retrieved by querying the DynamoDB database directly and by accessing the web app. The synthetic data provided was time series data, formatted in JSON, thus satisfying FR 1 and FR 2. The data was then sent using a HTTP PUT command satisfying FR 3 and SR 3. The test script flags how many synthetic data items should trigger the threshold alarm, and this was successfully matched to the number of alert emails received thus satisfying FR 4. An example of the alert email is in Appendix 7 The visualisation of historical data was limited to a table display due to time constraints, this is ingested and processed by the test script and successfully matched to the input data thus satisfying FR 5. A screenshot of the web app displaying the output data table is in Appendix 7.

Objective	*Test*
NFR 1: *The system must be able to scale to process data at speeds of up to 10MB/s*	The requirement is met by the assertion by Amazon that a Kinesis stream is divided into shards with each shard able to take a maximum of 1MB/s and 1000 PUT records. Each account can have a maximum of 10 shards (although this can be increased on request) [103] which would give 10MB/s without an increase so long as input data was batched so that each record was 1KB in size. The Lambda functions also have no limits to how they scale as each one can run in parallel on each data item [92] There is also no limit to the speed data can be written to DynamoDB as the number of "Write capacity units" can be increased up to 40,000 per table in the US East region (enough for 10MB/s assuming each record is around 0.25KB) [104].
NFR 2: *The system must be able to scale to store data volumes of 10s of PBs*	This requirement is met by Amazon's declaration that for DynamoDB "There is no practical limit on a table's size" [104] In reality there is a practical limit, which is the total size of Amazon's storage. Using assumptions used by Morgan [105] in estimating the number of servers owned by AWS in 2014 of 65000 servers per datacentre on average and approximately three datacentres per Availability Zone, along with the fact there are currently 35 Availability Zones [28] gives an estimate of 6.83m servers. If each of those servers had a terabyte of storage that gives around 6700 PBs of storage. 10 PBs would be a significant chunk of this but not impossible.
NFR 3: *The visualisation app should automatically scale depending on the amount of people using it*	While testing, it transpired that Apache Bench could not be used for testing as the web app was unable to serve more than one connection per container instance due to a lack of supporting infrastructure. Instead the scaling systems were tested by using ssh to connect to the first EC2 instance in the cluster and using the stress package [106] to load the CPU to 100% on that instance. The CloudWatch alarm metrics and service histories were then used to prove that the numbers of EC2 instances, containers and running web apps had changed in response to the CPU load. The CPU load was then terminated and again the metrics and histories used to check that the numbers of EC2 instances, containers and running web apps had decreased in response. See Appendix 8 for the testing details.
SR 1: *The system should allow SSH access only to servers with a specified key*	This was successfully tested by attempting to SSH using an alternative key and no key and the attempts failing, while the attempt with the correct key was successful.
SR 2: *The system should enforce access controls internally to ensure access is limited only to individual components that need to have access.*	This was tested theoretically by drawing a network diagram showing each role and the access it gave to specific components of the stack (see Appendix 11). This showed that only certain components had access to certain other components. Where sensible, this was locked down to specific named components. The time constraints

Objective	*Test*
	meant that this was all that could be done to test this, but much more work could be done here to analyse exactly which functions were needed and whether it would be possible to specifically name other components that currently aren't named as this would further improve security.

4.5.4 Acceptance testing results

From the screenshots in Appendix 5 it can be seen that each part of the stack took the following amount of time to provision:

Stack	*Time*
Repository	8 seconds
Main stack	6 minutes 19 seconds
Service scaling	1 minute 33 seconds
Total:	***8 minutes***

To provision the infrastructure manually it was found that setting up a resource manually for a subsample of the resources generally took about a minute each. As there are 35 separate resources to be created, that would take at least 35 minutes in total assuming that all the details had been worked out in advance together with the order in which they needed to be set up, as well as that no mistakes were made during the process.

This whole project was also presented to Amey as one of their "Wissen Talks". Feedback was positive, particularly on new approaches that they hadn't considered, such as the use of the API Gateway to accept data, and called the project "the first building stone of future cloud platforms for Amey Strategic Consulting" (see Appendix 12 for the full feedback).

5 Discussion

This section of the report looks at the results and how well the objectives have been met with recommendations and suggestions for further work.

5.1 Objective 1: To produce a requirements analysis for a cloud-based big data analysis platform.

The requirements specification (section 4.1) was produced and approved by the stakeholders in the project – Dr Pedro Baiz and Dr Richard Mathie from Amey, so this objective has been met. The requirements were kept quite generic for this project so that it could be completed in the time constraints and to keep options open for further work. In future, each part of the "Collect > Store > Process > Consume" pipeline could be analysed for more detailed requirements reflecting the actual data that Amey need to process, for example more tailored data structures, input mechanisms and visualisation requirements.

5.2 Objective 2: To design a scalable infrastructure that can be implemented in Amazon Web Services (AWS) that can be used to analyse Internet of Things (IoT) sensor data.

The final design (section 4.3.7) which was implemented, easily met all the functional requirements specified in section 4.1 as shown by the system testing in section 4.5.3. For FR 4 an email alert was sent every time the threshold was breached which could lead to information overload if a threshold is continually breached for a long period of time, in future work could be done to look at the last time an alert was sent for a particular Thing or group of Things and only send new alerts after a certain time threshold had passed or the data values had returned to normal again. For FR 5, a simple table was implemented just to prove that the data could be accessed through the infrastructure, however in future the web app could be significantly improved, using best practice in data visualisation to present the data and allowing interactive querying of the historical data to show trends and comparisons. Overall due to time constraints, the system isn't robust to poorly formed data and has limited error handling. This could be significantly improved in future work.

Unfortunately testing the non-functional requirements around scale proved to be more difficult than testing the functional requirements due to time and budget constraints. Testing big data applications requires a different approach to desk-based systems due to the large quantities of data being expensive to process both in terms of money and environmental costs due to the amount of power required to both run and cool the infrastructure. A theoretical approach was taken to NFR 1 and NFR 2 but these could be tested if required using the test scripts provided running in parallel with a substantially increased number of data items.

NFR 3 was only partially met as the web app itself proved to be a bottleneck in the scaling process. Future work to improve the web app using a web server such as nginx and a web server gateway interface such as uWGSI is necessary for better performance. Due to time constraints it was not possible to implement these in the project. Instead of being able to use Apache Bench to test the web app scaling directly, the testing had to be done in a more ad hoc way to show that the alarms could be triggered and the scaling activated manually. If the web app was to be improved so that Apache Bench could be used for load testing it would also make it easier to tune the parameters for CPU utilisation and memory reservation to get the most effective scaling performance.

NFR 3 was delivered using Docker and ECS as the new Docker orchestration functionality in version 1.12 wasn't available until mid-way through the project [107] and it was considered too high risk to change during the project. Future work here could compare the ease of use of ECS and the inbuilt Docker orchestration for scaling Docker containers.

The security requirements were met through the use of roles and security groups as described in the templates in section 4.1 and tested in section 4.5.3. Protecting customer data is very important and with more time much more advanced security can be added through virtual private cloud (VPC) networks and limited internet gateways to counter external threats, while also guarding against internal threats by using access controls on the console to ensure that developers only have access to the specific data they need to get their job done.

5.3 Objective 3: To implement a selected part of the design through CloudFormation templates that can automatically provision the infrastructure.

This objective has been met by the CloudFormation templates described in section 4.4 which were shown to successfully build the required infrastructure through the integration testing in section 4.5.2. Although the templates were written directly in JSON as required by Constraint 2, these JSON files can quickly get large, unmanageable and hard to read. Further work could be done to evaluate the pros and cons of programmatically producing these templates using, for example, the python troposphere library [108].

Also rather than aiming to produce one large template there may be management advantages to separating the templates into smaller chunks of related functionality and using nested templates instead. The issue with not being able to get the name of the AnalysisWebApp in the right format to be used for the ScalableTarget service for the container service needs to be reported to AWS and the documentation checked in future to see if this has been fixed which would make the templates significantly easier to manage.

Other useful additions would be the use of the AWS Route53 service to provide DNS to allow the use of user-defined memorable URLs for the ELB and the API which would make them easier to use.

5.4 Objective 4: To evaluate the speed advantage of provisioning infrastructure via CloudFormation versus manual provisioning via AWS console.

This objective was met through the acceptance testing in section 4.5.4. Fundamentally the main reason to use CloudFormation templates is that while they can be time-consuming to set up initially, once they have been written, the infrastructure can be provisioned and updated quickly, repeatably and without introducing random human errors (bugs) which are difficult to track down. This project has shown that for this particular infrastructure of medium complexity, the templates are at least four times faster than manual provisioning.

6 Evaluation, Reflections and Conclusions

The project has been successful in meeting all its objectives and I believe that this work will be useful to Amey as the initial research and design options analyses provide a thorough grounding for all the decisions that have been made. Amey can now use this research to see whether their more specific requirements would cause them to make different decisions at the design stage and use the templates I have provided as a basis for their infrastructure. In terms of their requirements to store PBs of data, I think that they should try and re-evaluate this requirement due to the large cost that it could incur as calculated in section 3.4.4. Anybody else can also use these templates to get started with a big data analysis platform which will save them significant amounts of time on research as there are so few examples out there on the internet.

In the wider context, this 'Infrastructure as Code' technique is going to become more and more important as system infrastructures become larger and more complex to deal with big data. The initial barrier to this is the time needed to learn the definition language, in this case CloudFormation, but I have shown that this can be achieved in a short time, after which the benefits in terms of speed and reduction of errors will be huge. However there is much scope to make these languages more user-friendly and I believe that this will happen through open source projects such as those mentioned.

Also based on the current growth of cloud services, it seems inevitable that within a few years, cloud computing will be the norm, particularly for new projects, with in-house data centres increased used only for legacy systems. AWS is the clear market leader at the moment but Google and Microsoft will almost certainly catch up, at which point keeping a system vendor-neutral to be able to move between offerings at a moment's notice may become much more important, while the cloud providers will want systems to be much closer tied to their proprietary services.

This project involved a steep learning curve as previously I had no background in Computer Science or software development and no real knowledge of AWS, CloudFormation, Docker or writing web apps. Each of these I have had to learn from scratch in order to produce each part of the project. The project was made even more challenging by the fact that a lot of the technology was very new and constantly evolving with little documentation or examples and announcements of new functionality made throughout the time I spent on the project which I had to decide whether to use or leave out. The project also covered a large range of tasks from data input, processing, storage and visualisation. In the end I overcame these challenges and have delivered a project that I found rewarding to carry out,due to the amount I learnt along the way, and am proud of, that delivers an end to end analytics platform with lots of scope for additions and improvements.

Initially I identified the biggest risk to the project being my lack of technical expertise which I attempted to rectify early by concentrating on learning how to use these new technologies. However, it soon became clear that the biggest risk was actually unclear requirements (a common project problem) and these took longer than I would have liked to get right. My lack of expertise in the technologies meant that I didn't have enough time to make the system as robust or the web app as beautiful as I would have liked to, but I am still proud that it is a good solid base that works end to end which can be improved upon. I could have focussed on a smaller section but I believe that the most interesting part of the project is how data can be processed end to end rather than solely focussing on data import and cleaning or data visualisation.

If I were to do the project again I would try and get the requirements pinned down much earlier in the process before researching all the technologies, although in the end the research also helped inform what good practice looked like. I was particularly inspired by the presentation given by Meyers at the AWS Summit [56] which really helped give me a framework to my designs and Robertson &

Robertson's book on the requirements process [72] which I will definitely refer back to in any future requirements exercise. Other than that I feel that I used my time well and by dividing the development work up into incremental stages I generally kept to the project plan. The best thing that I did was to start writing up the dissertation in the middle of July which kept me focussed on what I needed to do to fulfil each stage of the research process. I also found that booking regular face to face meetings with my supervisor every three or so weeks gave me much useful feedback that wasn't possible over email, as well as deadlines to aim for, and I wish I had started these much earlier.

Glossary

API: Application Programming Interface – defines a language different parts of an application can use to communicate with each other
AWS: Amazon Web Services – the provider of the cloud service used in this project
CloudFormation: the AWS service used to convert JSON formatted text describing an infrastructure stack into the actual infrastructure
EC2 instance: AWS Elastic Compute Cloud – a virtual server provided by AWS
ELB: Elastic Load Balancer – AWS service that load balances between different copies of the same application on different servers
FR: Functional Requirement – a requirement that specifies what the system should do
IaC: Infrastructure as Code – using scripts to define infrastructure. See section 2.1.1
IoT: Internet of Things – connecting many sensors together. See section 2.3.1
NFR: Non-Functional Requirement – a requirement that specifies how the system should behave
NoSQL database: Not Only SQL – databases that behave differently from traditional fixed schema relational databases. See section 2.4.4
SR: Security Requirement – a requirement that specifies how the system should be secured

References

[1] T. Sharma, M. Fragkoulis, and D. Spinellis, 'Does Your Configuration Code Smell?', in *Proceedings of the 13th International Conference on Mining Software Repositories*, New York, NY, USA, 2016, pp. 189–200.

[2] K. Morris, 'Infrastructure as Code: From the Iron Age to the Cloud Age', *ThoughtWorks*, 08-Jan-2016. [Online]. Available: https://www.thoughtworks.com/insights/blog/infrastructure-code-iron-age-cloud-age. [Accessed: 16-Jun-2016].

[3] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, 'DevOps', *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

[4] R. Alur, E. Berger, A. W. Drobnis, L. Fix, K. Fu, G. D. Hager, D. Lopresti, K. Nahrstedt, E. Mynatt, S. Patel, and others, 'Systems Computing Challenges in the Internet of Things', *ArXiv Prepr. ArXiv160402980*, 2016.

[5] F. Economou, J. C. Hoblitt, and P. Norris, 'Your data is your dogfood: DevOps in the astronomical observatory', *ArXiv14076463 Astro-Ph*, Jul. 2014.

[6] M. Virmani, 'Understanding DevOps bridging the gap from continuous integration to continuous delivery', in *2015 Fifth International Conference on Innovative Computing Technology (INTECH)*, 2015, pp. 78–82.

[7] D. Spinellis, 'Being a DevOps Developer', *IEEE Softw.*, vol. 33, no. 3, pp. 4–5, May 2016.

[8] L. Zhu, L. Bass, and G. Champlin-Scharff, 'DevOps and Its Practices', *IEEE Softw.*, vol. 33, no. 3, pp. 32–34, May 2016.

[9] 'bliki: CanaryRelease', *martinfowler.com*. [Online]. Available: http://martinfowler.com/bliki/CanaryRelease.html. [Accessed: 17-Jun-2016].

[10] S. Smith, 'End-To-End Testing Considered Harmful', *Always Agile Consulting*, 12-Dec-2015. .

[11] J. Scheuner, P. Leitner, J. Cito, and H. Gall, 'Cloud Work Bench – Infrastructure-as-Code Based Cloud Benchmarking', in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014, pp. 246–253.

[12] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, 'Testing Idempotence for Infrastructure as Code', in *Middleware 2013*, D. Eyers and K. Schwan, Eds. Springer Berlin Heidelberg, 2013, pp. 368–388.

[13] Y. Jiang and B. Adams, 'Co-evolution of Infrastructure and Source Code - An Empirical Study', in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 45–55.

[14] 'Terraform by HashiCorp'. [Online]. Available: https://www.terraform.io/. [Accessed: 16-Aug-2016].

[15] 'Amazon CloudFormation FAQs', *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/cloudformation/faqs/. [Accessed: 03-Aug-2016].

[16] A. Tarry, 'Thoughts on Terraform', 18-Mar-2016. .

[17] J. Padnick, 'A Comprehensive Guide to Building a Scalable Web App on Amazon Web Services - Part 1'. [Online]. Available: https://www.airpair.com/aws/posts/building-a-scalable-web-app-on-amazon-web-services-p1. [Accessed: 27-Jul-2016].

[18] P. Mell and T. Grance, 'The NIST definition of cloud computing', *Recomm. Natl. Inst. Stand. Technol.*, no. Special Publication 800–145, 2011.

[19] 'Overview of Amazon Web Services', Amazon Web Services, Inc., Dec. 2015.

[20] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, 'Trends in big data analytics', *J. Parallel Distrib. Comput.*, vol. 74, no. 7, pp. 2561–2573, Jul. 2014.

[21] Gartner Inc, 'Gartner Says Worldwide Public Cloud Services Market Is Forecast to Reach $204 Billion in 2016'. [Online]. Available: http://www.gartner.com/newsroom/id/3188817. [Accessed: 13-Jul-2016].

[22] Gartner Inc, 'Gartner's Hype Cycles for 2015: Five Megatrends Shift the Computing Landscape'. [Online]. Available: https://www.gartner.com/doc/3111522/gartners-hype-cycles-megatrends-shift. [Accessed: 15-Jul-2016].

[23] Gartner Inc, 'Magic Quadrant for Cloud Infrastructure as a Service, Worldwide', Aug. 2016.
[24] M. Finnegan, 'Google takes aim at AWS - now it must convince CIOs it is right cloud for the enterprise', *ComputerworldUK*. [Online]. Available: http://www.computerworlduk.com/cloud-computing/google-takes-aim-at-aws-now-it-must-proove-it-is-right-cloud-for-enterprise-3637270/. [Accessed: 13-Jul-2016].
[25] M. Finnegan, 'How Tesco Bank has adopted AWS cloud as "business as usual" in eight months', *ComputerworldUK*. [Online]. Available: http://www.computerworlduk.com/cloud-computing/how-tesco-bank-has-adopted-aws-cloud-as-business-as-usual-in-eight-months-3629767/. [Accessed: 13-Jul-2016].
[26] M. Finnegan, 'News Corp: Moving SAP ERP to Amazon cloud will save "tens of millions of dollars"', *ComputerworldUK*. [Online]. Available: http://www.computerworlduk.com/news/it-vendors/news-corp-moving-sap-erp-amazon-cloud-will-save-tens-of-millions-of-dollars-3514102/. [Accessed: 13-Jul-2016].
[27] T. Magee, 'Pubmatic swaps AWS for OpenStack after public cloud hidden-cost horror show #Openstacksummit', *ComputerworldUK*. [Online]. Available: http://www.computerworlduk.com/cloud-computing/pubmatic-cloud-chief-on-his-six-figure-aws-hidden-cost-horror-show-3639325/. [Accessed: 20-Jul-2016].
[28] 'Global Infrastructure', *Amazon Web Services, Inc.* [Online]. Available: http://aws.amazon.com/about-aws/global-infrastructure/. [Accessed: 20-Jul-2016].
[29] M. Murphy, 'Top 10 data centre disasters', *ComputerworldUK*. [Online]. Available: http://www.computerworlduk.com/galleries/infrastructure/ten-datacentre-disasters-that-brought-firms-offline-3593580/. [Accessed: 13-Jul-2016].
[30] D. Bryen, 'Serverless Applications on AWS', presented at the AWS Summit London, 07-Aug-2016.
[31] 'Nordic Countries Increasingly Attractive as Sites for Data Centers - The New York Times'. [Online]. Available: http://www.nytimes.com/2012/04/30/business/global/nordic-countries-increasingly-attractive-as-sites-for-data-centers.html?_r=0. [Accessed: 16-Aug-2016].
[32] 'AWS & Sustainability', *Amazon Web Services, Inc.* [Online]. Available: http://aws.amazon.com/about-aws/sustainability/. [Accessed: 20-Jul-2016].
[33] 'Clicking Clean: A Guide to Building the Green Internet', Greenpeace, May 2015.
[34] F. Y. Rashid, 'The dirty dozen: 12 cloud security threats', *InfoWorld*, 11-Mar-2016. [Online]. Available: http://www.infoworld.com/article/3041078/security/the-dirty-dozen-12-cloud-security-threats.html. [Accessed: 21-Jul-2016].
[35] J. Weinman, Ed., 'When—and When Not—to Use the Cloud', in *Cloudonomics*, John Wiley & Sons, Inc., 2012, pp. 91–105.
[36] 'Industrial Internet of Things: Unleashing the Potential of Connected Products and Services', World Economic Forum, Jan. 2015.
[37] A. Zaslavsky, C. Perera, and D. Georgakopoulos, 'Sensing as a Service and Big Data', *ArXiv13010159 Cs*, Jan. 2013.
[38] D. Guinard and V. Trifa, *Building the Web of Things*. Manning Publications, 2016.
[39] I. F. Akyildiz, M. Pierobon, S. Balasubramaniam, and Y. Koucheryavy, 'The internet of bio-nano things', *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 32–40, 2015.
[40] K. Ashton, 'That "internet of things" thing', *RFiD J.*, vol. 22, no. 7, pp. 97–114, 2009.
[41] C. McLellan, 'The internet of things and big data: Unlocking the power', *ZDNet*. [Online]. Available: http://www.zdnet.com/article/the-internet-of-things-and-big-data-unlocking-the-power/. [Accessed: 15-Jul-2016].
[42] Gartner Inc, 'Focus on Startups and Small Vendors as Drivers for IoT Innovation'. [Online]. Available: https://www.gartner.com/doc/3083422/focus-startups-small-vendors-drivers. [Accessed: 15-Jul-2016].

[43] Gartner Inc, 'Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020'. [Online]. Available: http://www.gartner.com/newsroom/id/2636073. [Accessed: 21-Jul-2016].
[44] F. J. Riggins and S. F. Wamba, 'Research directions on the adoption, usage, and impact of the internet of things through the use of big data analytics', in *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, 2015, pp. 1531–1540.
[45] 'The Internet of Things Moves Beyond the Buzz: Worldwide Market Forecast to Exceed $7 Trillion by 2020, IDC Says | Business Wire', 03-Jun-2014. [Online]. Available: http://www.businesswire.com/news/home/20140603005446/en/Internet-Moves-Buzz-Worldwide-Market-Forecast-Exceed. [Accessed: 21-Jul-2016].
[46] F. Bonomi, 'The Smart and Connected Vehicle and the Internet of Things', presented at the Workshop on Synchronization and Timing Systems, 2013.
[47] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, 'Internet of Things for Smart Cities', *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.
[48] ThyssenKrupp, 'ThyssenKrupp launches MAX: Maximizing urban efficiency with leading Microsoft Azure IoT-enabled technology'. [Online]. Available: http://www.thyssenkrupp-elevator.com/en/ba-et/press/press-releases-1346.html. [Accessed: 15-Jul-2016].
[49] T. T. Mulani and S. V. Pingle, 'Internet of Things', *Int. Res. J. Multidiscip. Stud.*, vol. 2, no. 3, 2016.
[50] Y. Agarwal and A. K. Dey, 'Toward Building a Safe, Secure, and Easy-to-Use Internet of Things Infrastructure', *Computer*, vol. 49, no. 4, pp. 88–91, 2016.
[51] Gartner Inc, 'What Is Big Data? - Gartner IT Glossary - Big Data', *Gartner IT Glossary*, 25-May-2012. [Online]. Available: http://www.gartner.com/it-glossary/big-data/. [Accessed: 20-Jul-2016].
[52] 'Big data', *Wikipedia, the free encyclopedia*. 18-Jul-2016.
[53] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, 'The rise of "big data" on cloud computing: Review and open research issues', *Inf. Syst.*, vol. 47, pp. 98–115, Jan. 2015.
[54] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. S. Netto, and R. Buyya, 'Big Data computing and clouds: Trends and future directions', *J. Parallel Distrib. Comput.*, vol. 79–80, pp. 3–15, May 2015.
[55] 'The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things', EMC / IDC, Apr. 2014.
[56] I. Meyers, 'Big Data Architectural Patterns and Best Practices on AWS', presented at the AWS Summit London, 07-Jul-2016.
[57] B. Michelson, 'Event-Driven Architecture Overview : 5th Anniversary Edition'. Feb-2011.
[58] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, 2015.
[59] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, 'Data management in cloud environments: NoSQL and NewSQL data stores', *J. Cloud Comput. Adv. Syst. Appl.*, vol. 2, no. 1, p. 1, 2013.
[60] F. Akhbar, V. Chang, Y. Yao, and V. Méndez Muñoz, 'Outlook on moving of computing services towards the data sources', *Int. J. Inf. Manag.*, vol. 36, no. 4, pp. 645–652, Aug. 2016.
[61] D. Borkar, R. Mayuram, G. Sangudi, and M. Carey, 'Have Your Data and Query It Too: From Key-Value Caching to Big Data Management', in *Proceedings of the 2016 International Conference on Management of Data*, New York, NY, USA, 2016, pp. 239–251.
[62] E. A. Brewer, 'Towards robust distributed systems', in *PODC*, 2000, vol. 7.
[63] D. Pritchett, 'BASE: An Acid Alternative', *Queue*, vol. 6, no. 3, pp. 48–55, May 2008.
[64] M. Cooper and P. Mell, 'Tackling Big Data', presented at the Federal Computer Security Program Managers' Forum, Jun-2012.

[65] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, 'NoSQL Database Systems: A Survey and Decision Guidance'.
[66] J. Dean and S. Ghemawat, 'MapReduce: simplified data processing on large clusters', *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[67] R. Kune, P. K. Konugurthi, A. Agarwal, R. R. Chillarige, and R. Buyya, 'The anatomy of big data computing', *Softw. Pract. Exp.*, vol. 46, no. 1, pp. 79–105, Jan. 2016.
[68] P. Rubens, 'What are containers and why do you need them?', *CIO*, 20-May-2015. [Online]. Available: http://www.cio.com/article/2924995/enterprise-software/what-are-containers-and-why-do-you-need-them.html. [Accessed: 22-Jul-2016].
[69] 'It's Here: Docker 1.0', *Docker Blog*, 09-Jun-2014. [Online]. Available: https://blog.docker.com/2014/06/its-here-docker-1-0/. [Accessed: 22-Jul-2016].
[70] A. Jlassi and P. Martineau, 'Virtualization Technologies for the Big Data Environment', in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, New York, NY, USA, 2016, pp. 542–545.
[71] M. Roberts, 'Serverless Architectures', *martinfowler.com*. [Online]. Available: http://martinfowler.com/articles/serverless.html. [Accessed: 22-Jul-2016].
[72] S. Robertson and J. Robertson, *Mastering the requirements process: getting requirements right*, 3rd ed. Addison-Wesley, 2013.
[73] 'Amazon API Gateway FAQs', *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/api-gateway/faqs/. [Accessed: 15-Aug-2016].
[74] 'AWS IoT FAQs', *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/iot/faqs/. [Accessed: 15-Aug-2016].
[75] 'AWS CloudFormation Adds Support for AWS IoT and Additional Updates'. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2016/07/aws-cloudformation-adds-support-for-aws-iot-and-additional-updates/. [Accessed: 15-Aug-2016].
[76] 'AWS Data Pipeline FAQs', *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/datapipeline/faqs/. [Accessed: 15-Aug-2016].
[77] 'What Is Amazon Kinesis Streams?', *Amazon Web Services*. [Online]. Available: http://docs.aws.amazon.com/streams/latest/dev/introduction.html. [Accessed: 02-Aug-2016].
[78] 'Introduction to Amazon SQS', *Amazon Web Services*. [Online]. Available: http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/Introduction.html. [Accessed: 02-Aug-2016].
[79] 'Limits in Amazon SQS - Amazon Simple Queue Service'. [Online]. Available: http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-limits.html. [Accessed: 02-Aug-2016].
[80] 'Apache Kafka'. [Online]. Available: http://kafka.apache.org/. [Accessed: 02-Aug-2016].
[81] T. Schreiter, 'Ingestion Comparison: Kafka vs Kinesis', 15-Jan-2016. [Online]. Available: http://insightdataengineering.com/blog/ingestion-comparison/. [Accessed: 02-Aug-2016].
[82] S. Ramachandra, 'Evaluating Message Brokers: Kafka vs. Kinesis vs. SQS', *OpsClarity*, 10-May-2016. .
[83] 'Amazon DynamoDB FAQs – Amazon Web Services (AWS)', *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/dynamodb/faqs/. [Accessed: 02-Aug-2016].
[84] 'Amazon ElastiCache FAQs', *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/elasticache/faqs/. [Accessed: 15-Aug-2016].
[85] 'Amazon RDS FAQs', *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/rds/faqs/. [Accessed: 02-Aug-2016].
[86] 'Amazon Redshift FAQs', *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/redshift/faqs/. [Accessed: 02-Aug-2016].
[87] 'Amazon Simple Storage Service (S3) FAQs', *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/s3/faqs/. [Accessed: 02-Aug-2016].

[88] ‘Apache Cassandra’. [Online]. Available: http://cassandra.apache.org/. [Accessed: 15-Aug-2016].
[89] ‘DB-Engines Ranking - popularity ranking of database management systems’. [Online]. Available: http://db-engines.com/en/ranking. [Accessed: 15-Aug-2016].
[90] ‘Amazon Machine Learning’, *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/machine-learning/details/. [Accessed: 15-Aug-2016].
[91] ‘Amazon Elastic MapReduce (EMR) FAQs’, *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/elasticmapreduce/faqs/. [Accessed: 02-Aug-2016].
[92] ‘AWS Lambda FAQs’, *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/lambda/faqs/. [Accessed: 02-Aug-2016].
[93] ‘Apache Storm’. [Online]. Available: http://storm.apache.org/. [Accessed: 02-Aug-2016].
[94] ‘DB-Engines Ranking - popularity ranking of search engines’. [Online]. Available: http://db-engines.com/en/ranking/search+engine. [Accessed: 15-Aug-2016].
[95] ‘Amazon QuickSight FAQs’, *Amazon Web Services*. [Online]. Available: http://aws.amazon.com/quicksight/faqs/. [Accessed: 03-Aug-2016].
[96] ‘Backstory · Serverless’, *Serverless*. [Online]. Available: http://docs.serverless.com/docs/backstory. [Accessed: 03-Aug-2016].
[97] ‘Tableau Software’, *Tableau Software*. [Online]. Available: https://www.tableau.com/. [Accessed: 03-Aug-2016].
[98] ‘Running Tableau Server on Amazon AWS’, *Tableau Software*. [Online]. Available: http://kb.tableau.com/articles/knowledgebase/running-tableau-server-on-amazon-aws. [Accessed: 03-Aug-2016].
[99] ‘Welcome | Flask (A Python Microframework)’. [Online]. Available: http://flask.pocoo.org/. [Accessed: 03-Aug-2016].
[100] ‘Template Anatomy - AWS CloudFormation’. [Online]. Available: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html. [Accessed: 03-Aug-2016].
[101] C. W. Dawson, *Projects in computing and information systems: a student’s guide*, Third. Harlow, England: Pearson, 2015.
[102] S. Mathur and S. Malik, ‘Advancements in the V-Model’, *Int. J. Comput. Appl.*, vol. 1, no. 12, 2010.
[103] ‘Amazon Kinesis Streams FAQs’, *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/kinesis/streams/faqs/. [Accessed: 11-Aug-2016].
[104] ‘Limits in DynamoDB - Amazon DynamoDB’. [Online]. Available: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Limits.html. [Accessed: 11-Aug-2016].
[105] T. P. Morgan, ‘A Rare Peek Into The Massive Scale of AWS’, *EnterpriseTech*, 14-Nov-2014. [Online]. Available: http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/. [Accessed: 04-Sep-2016].
[106] ‘stress’. [Online]. Available: https://people.seas.harvard.edu/~apw/stress/. [Accessed: 10-Sep-2016].
[107] ‘Docker Built-in Orchestration Ready for Production: Docker 1.12 Goes GA’, *Docker Blog*. .
[108] ‘cloudtools/troposphere: troposphere - Python library to create AWS CloudFormation descriptions’. [Online]. Available: https://github.com/cloudtools/troposphere. [Accessed: 16-Aug-2016].

Appendix 1: Proposal

Research Project Proposal:

"Using an Infrastructure as Code approach to a cloud-based Internet of Things Big Data Analysis platform"

Alison Wells
MSc Data Science

Supervisor: Dr Tillman Weyde

July 2016

Proposal: "Using an Infrastructure as Code approach to a cloud-based Internet of Things Big Data Analysis platform"

Introduction

Amey are a large infrastructure services company based in the UK, employing over 21,000 people and providing services in areas such as utilities, highways, waste management, rail, justice solutions, social housing and facilities management.

They have developed a system that analyses large amounts of data from asset sensors which allows asset owners to manage their assets more effectively. This system now needs to be scaled up and streamlined. This project will investigate how this scaling and streamlining can be done effectively and efficiently using an "Infrastructure as code" approach.

The research question for this project is: "Can an "Infrastructure as code" approach be used to automatically produce a scalable Internet of Things big data analysis platform and what are the benefits in terms of speed?"

The project has the following objectives:

1) To produce a requirements analysis for a cloud-based big data analysis platform.
   This objective will be deemed to be met if a document is produced listing and justifying requirements.
2) To design a scalable infrastructure that can be implemented in Amazon Web Services that can be used to analyse Internet of Things sensor data.
   This objective will be deemed to be met if the infrastructure meets a defined set of the requirements collected during the requirements analysis.
3) To implement a selected part of the design through CloudFormation templates (see Appendix 1 for an example) that can automatically provision the infrastructure.
   This objective will be deemed to be met if the templates can be run successfully and data can be processed from end to end in the system.
4) To evaluate the speed advantage of provisioning infrastructure via CloudFormation versus manual provisioning via AWS console.
   This objective will be deemed to be met if tests are carried out comparing the times taken to provision the same infrastructure manually and through templates, and the results analysed.

The beneficiaries of this work will be first of all, Amey, as the research on best practice for big data analytics architecture using Amazon Web Services will allow them to decide whether they need to make changes to their existing system architecture and what changes should be made to make best use of AWS. Secondly the CloudFormation templates will allow them to implement the infrastructure immediately, and then build on it if they require. Outside of Amey, the research and templates could be used by anybody wanting to build a big data analysis platform using AWS as a way to understand the architecture options and get up and running as quickly as possible.

Context and literature review

What is "infrastructure as code"?
Sharma et al [1] define "infrastructure as code" (IaC) as "the practice of specifying computing system configurations through code, and managing them through traditional software engineering methods.".

Why is this approach useful?
Kief Morris [2] describes a transition from the "Iron Age" where hardware is bought, installed and manually configured for each specific application to the "Cloud Age" where virtualisation is the norm and hardware ownership has transferred to Infrastructure as a Service (IaaS or more generally "cloud") providers such as Amazon or Google. Existing manual configuration practices have failed to keep up with the ease of spinning up large numbers of virtual machines. These virtual machines can also have many "containers" on them holding separate applications. Even when the individual configurations become more automated, the overall picture becomes increasingly complex and hard to manage with a lack of configuration management leading to many unique "snowflake" environments being created in an uncontrolled manner. [3]

IaC can improve this situation because by treating the infrastructure as code, it can be version controlled, documented and tested in the same way as application code. This means that administrators can quickly reproduce the existing environment on the same or new hardware or roll back to previous environments if necessary, and standards can be widely applied and automatically enforced [4][5]. Because adding new infrastructure now becomes scalable, with the same effort needed to set up 5 or 5000 servers, the ratio of admins to servers can be massively increased. Economou et al [4] give figures comparing a recommended server to admin ratio of 10-20 to 1 as recently as 2002 with a statement from 2013 from Facebook that they were working with ratios of over 20,000 to 1.

Potential downsides of this approach
This approach requires a lot of cultural change and is generally associated with a move to a DevOps culture where developers, Quality Assurance and IT Operations staff not only work side by side, but actually merge roles. This is to try and break down the "Wall of Confusion" between developers, testers and IT Ops where code is "thrown over the wall" from one side to the other. It is also associated with the Agile development methodology where operational code is produced in very short (1-2 week) sprints and requirements are constantly being revised and reprioritised leading to requirements for frequent deployments [6]. To take advantage of these changes, systems are also often redesigned into a "microservices" architecture where a large monolithic application is broken down into many independent smaller services that communicate with each other. [7]

Testing methodologies also need to change once infrastructure can be changed as easily as code. Traditional methodologies use identical (in theory) development, test and operational environments and use end to end testing to slowly test new releases as they move from one environment to another. With a move to continuous integration / continuous deployment, testing needs to be carried out differently. As this is such a new area, there is no definitive answer but options include "blue / green releases" or "canary releases" [8] where the new release is spun up as an entirely separate system and the customers gently moved across from the old version, at which point the old version is destroyed. If there are any problems the customers can all be moved back to the old version immediately. Another idea is to focus on reducing the "Mean Time To Recover" rather than the "Mean Time Between Failures" and instil "antifragility" into the system. This means that things are expected to go wrong and monitoring is used to spot problems quickly and implement strategies

to cope with them. “Chaos engineering” is the act of deliberately causing problems, for example by deleting servers randomly, so that code is written resiliently from the outset. [9]

Research gaps

This is a very new area and only a handful of research papers have investigated IaC. Scheuner et al [10] used IaC to enable reproducible environments to compare different cloud services. Hummer et al [11] looked at how IaC can ensure “idempotence” (identical results each time). Jiang and Adams [12] looked at the relationship between changes to infrastructure code and source code in an application and show that they need to be treated the same as they are equally prone to error. Sharma et al [1] analyse large numbers of IaC scripts and identify code signatures that represent good or bad practice. All other publications on the benefits of IaC are anecdotal, and no published research can be found that attempts to quantify the benefits in terms of speed or cost over an existing system.

Approach

This project will be a “Design and Creation” type project as described by Oates [13]. The research aspect is focussed on how the new technique of Infrastructure as Code can be applied to an existing system design and the evaluation of the benefits that it brings.

The following artefacts will be delivered:

Literature review

The literature review will be expanded from the initial review in this proposal to cover any concepts uncovered in the analysis and system design phases. It will also include a full review of all components in the Amazon Web Services system and an analysis of their relevance to the system design.

Requirements analysis

The requirements analysis will be a document listing and justifying each requirement of the system based on literature research.

Design of new system architecture including options analysis

Figure 1: High level system design of the current system

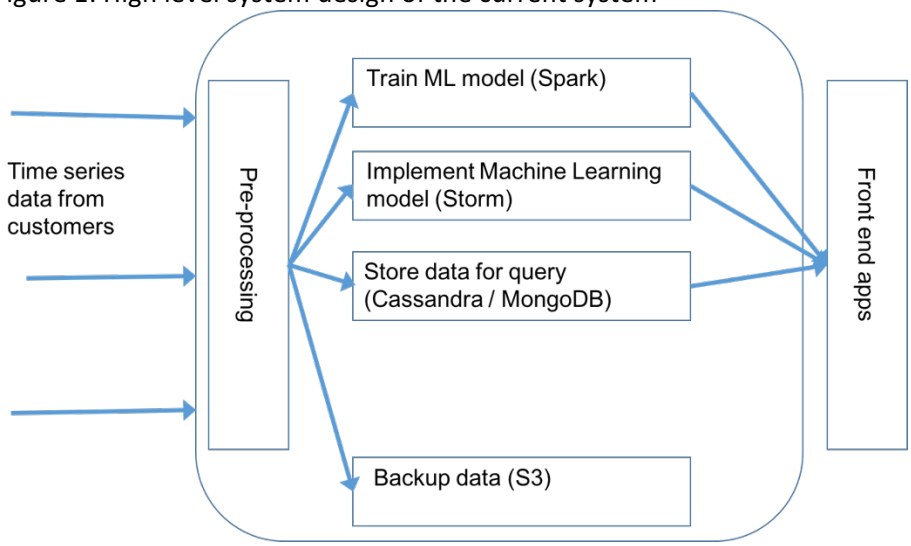
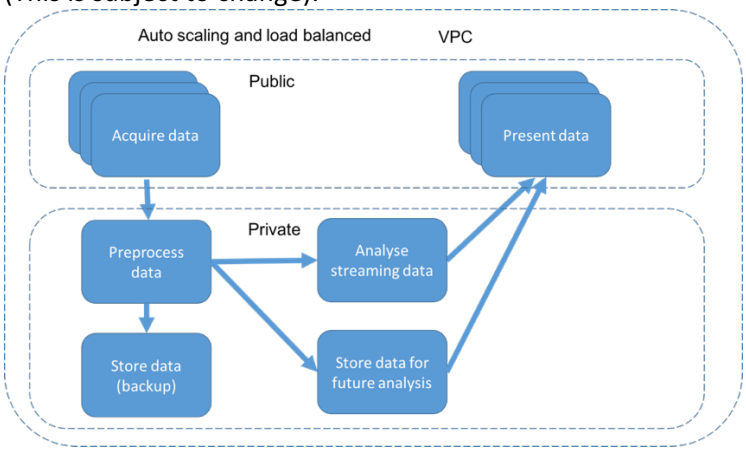


Figure 2: An initial idea of the high level architecture for the system to be developed in this project. (This is subject to change):



This will be implemented using Amazon Web Services components (such as Kinesis, Lambda, DynamoDB, S3, EC2, ELB etc). More than one option may be available for the whole design or for specific parts, so an options analysis will be presented and more than one option may be chosen to be developed depending on time.

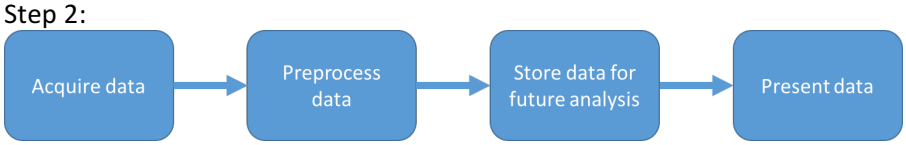
Development plan

The high level architecture has been broken down into 5 steps which build on each other to develop the final architecture. This will mean that even if there is not enough time to complete the whole design, some value will be gained. This plan will be revisited weekly throughout the development process.

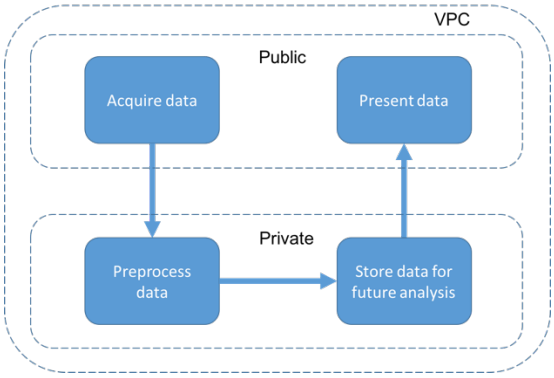
The following figures show a potential design implementation of the project based on breaking down the system into small chunks of functionality that can be built up. These are subject to change.

Step 1:

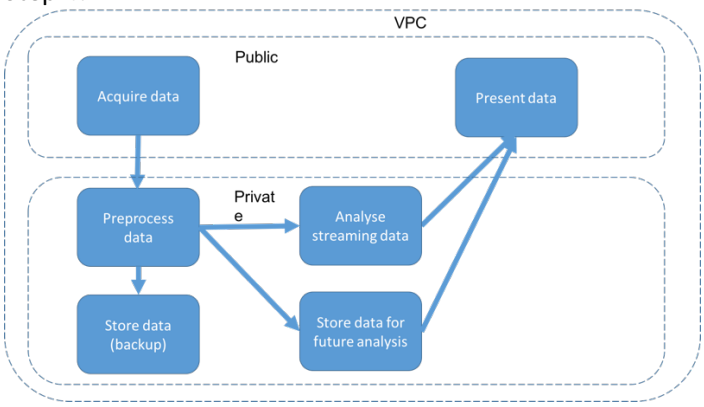
Step 2:



Step 3:



Step 4:



Step 5 is the final architecture referred to in Figure 2

Implementation of new system architecture

This will consist of the scripts and documentation necessary to implement the new architecture.

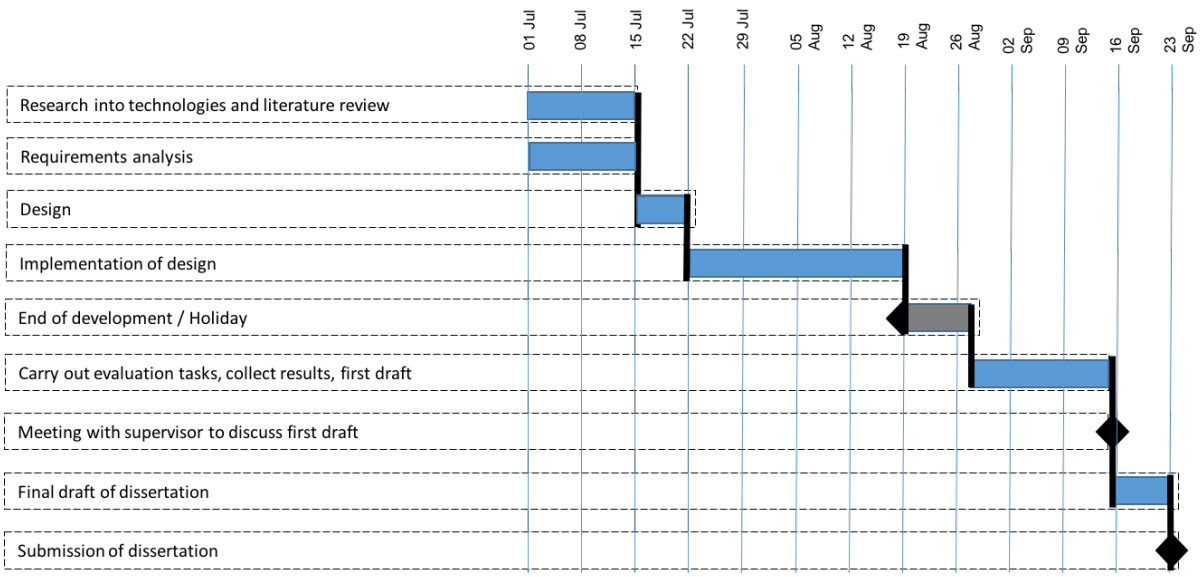
Evaluation

For each step, the scripts will be tested to ensure that they create the correct resources and the time taken captured. Each step will also be tested to see how well they scale with increasing amounts of data / web page requests. The results will be compared and analysed to determine the benefits (if any) of the new IaC approach

Ethics

This project does not involve people or any personally identifiable data. The performance tests will be carried out on synthetic data rather than live data, and thus avoid any issues with confidentiality.

Work plan



Risks

Risk	Likelihood	Impact	Mitigation
Not enough time to implement whole new system architecture	H	H	• Use Agile methodology to divide work into separate pieces of functionality. Prioritise functionality that can be implemented in isolation or that demonstrate significant benefit in terms of cost or time saving
Not enough time to implement new design as too many new technologies to learn and integrate	H	H	• Keep number of systems to integrate to a minimum by using components native to Amazon Web Services where possible.
Not able to complete project due to lack of technical expertise in AWS	M	H	• Ask for help from AWS architects and other members of Amey team with experience. Attend AWS events
Dissertation lost due to computer failure	L	H	• Back up regularly to alternative storage locations such as iCloud

References

[1] T. Sharma, M. Fragkoulis, and D. Spinellis, 'Does Your Configuration Code Smell?', in Proceedings of the 13th International Conference on Mining Software Repositories, New York, NY, USA, 2016, pp. 189–200.

[2] 'Infrastructure as Code: From the Iron Age to the Cloud Age', ThoughtWorks, 08-Dec-2015. [Online]. Available: https://www.thoughtworks.com/insights/blog/infrastructure-code-iron-age-cloud-age. [Accessed: 16-Jun-2016].

[3] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, 'DevOps', IEEE Softw., vol. 33, no. 3, pp. 94–100, May 2016.

[4] F. Economou, J. C. Hoblitt, and P. Norris, 'Your data is your dogfood: DevOps in the astronomical observatory', ArXiv14076463 Astro-Ph, Jul. 2014.

[5] M. Virmani, 'Understanding DevOps bridging the gap from continuous integration to continuous delivery', in 2015 Fifth International Conference on Innovative Computing Technology (INTECH), 2015, pp. 78–82.

[6] D. Spinellis, 'Being a DevOps Developer', IEEE Softw., vol. 33, no. 3, pp. 4–5, May 2016.

[7] L. Zhu, L. Bass, and G. Champlin-Scharff, 'DevOps and Its Practices', IEEE Softw., vol. 33, no. 3, pp. 32–34, May 2016.

[8] 'bliki: CanaryRelease', martinfowler.com. [Online]. Available: http://martinfowler.com/bliki/CanaryRelease.html. [Accessed: 17-Jun-2016].

[9] 'Always Agile Consulting • End-To-End Testing Considered Harmful'. .

[10] J. Scheuner, P. Leitner, J. Cito, and H. Gall, 'Cloud Work Bench – Infrastructure-as-Code Based Cloud Benchmarking', in 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom), 2014, pp. 246–253.

[11] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, 'Testing Idempotence for Infrastructure as Code', in Middleware 2013, D. Eyers and K. Schwan, Eds. Springer Berlin Heidelberg, 2013, pp. 368–388.

[12] Y. Jiang and B. Adams, 'Co-evolution of Infrastructure and Source Code - An Empirical Study', in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 45–55.

[13] B. J. Oates, Researching Information Systems and Computing. Sage Publications Ltd., 2006.

Appendix 1: Example CloudFormation Template

Provided by AWS at https://s3-us-west-2.amazonaws.com/cloudformation-templates-us-west-2/DynamoDB_Table.template

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",
 "Description" : "AWS CloudFormation Sample Template DynamoDB_Table: This template demonstrates the creation of a DynamoDB table.  **WARNING** This template creates an Amazon DynamoDB table. You will be billed for the AWS resources used if you create a stack from this template.",

 "Parameters" : {
  "HashKeyElementName" : {
   "Description" : "HashType PrimaryKey Name",
   "Type" : "String",
   "AllowedPattern" : "[a-zA-Z0-9]*",
   "MinLength": "1",
   "MaxLength": "2048",
   "ConstraintDescription" : "must contain only alphanumberic characters"
  },
  "HashKeyElementType" : {
   "Description" : "HashType PrimaryKey Type",
   "Type" : "String",
   "Default" : "S",
   "AllowedPattern" : "[S|N]",
   "MinLength": "1",
   "MaxLength": "1",
   "ConstraintDescription" : "must be either S or N"
  },
  "ReadCapacityUnits" : {
   "Description" : "Provisioned read throughput",
   "Type" : "Number",
   "Default" : "5",
   "MinValue": "5",
   "MaxValue": "10000",
   "ConstraintDescription" : "must be between 5 and 10000"
  },
  "WriteCapacityUnits" : {
   "Description" : "Provisioned write throughput",
   "Type" : "Number",
   "Default" : "10",
   "MinValue": "5",
   "MaxValue": "10000",
   "ConstraintDescription" : "must be between 5 and 10000"
  }
 },
 "Resources" : {
  "myDynamoDBTable" : {
   "Type" : "AWS::DynamoDB::Table",
   "Properties" : {
    "AttributeDefinitions": [ {
     "AttributeName" : {"Ref" : "HashKeyElementName"},
     "AttributeType" : {"Ref" : "HashKeyElementType"}
    } ],
    "KeySchema": [
     { "AttributeName": {"Ref" : "HashKeyElementName"}, "KeyType": "HASH" }
    ],
    "ProvisionedThroughput" : {
     "ReadCapacityUnits" : {"Ref" : "ReadCapacityUnits"},
     "WriteCapacityUnits" : {"Ref" : "WriteCapacityUnits"}
    }
   }
  }
 },

 "Outputs" : {
  "TableName" : {
   "Value" : {"Ref" : "myDynamoDBTable"},
   "Description" : "Table name of the newly created DynamoDB table"
  }
 }
}
```

Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

Part A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

Part B: Ethics Proportionate Review Form. Students who have answered "no" to questions 1 – 18 and "yes" to question 19 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in this case. The approval may be provisional: the student may need to seek additional approval from the supervisor as the project progresses.

A.1 If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval.		*Delete as appropriate*
1.	Does your project require approval from the National Research Ethics Service (NRES)? For example, because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/.	**No**
2.	Does your project involve participants who are covered by the Mental Capacity Act? If so, you will need approval from an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/.	**No**
3.	Does your project involve participants who are currently under the auspices of the Criminal Justice System? For example, but not limited to, people on remand, prisoners and those on probation? If so, you will need approval from the ethics approval system of the National Offender Management Service.	**No**

A.2 If your answer to any of the following questions (4 – 11) is YES, you must apply to the City University Senate Research Ethics Committee (SREC) for approval (unless you are applying to an external ethics committee).		*Delete as appropriate*
4.	Does your project involve participants who are unable to give informed consent? For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	**No**
5.	Is there a risk that your project might lead to disclosures from participants concerning their involvement in illegal activities?	**No**
6.	Is there a risk that obscene and or illegal material may need to be accessed for your project (including online content and other material)?	**No**

7.	Does your project involve participants disclosing information about sensitive subjects? For example, but not limited to, health status, sexual behaviour, political behaviour, domestic violence.	**No**
8.	Does your project involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (See http://www.fco.gov.uk/en/)	**No**
9.	Does your project involve physically invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	**No**
10.	Does your project involve animals?	**No**
11.	Does your project involve the administration of drugs, placebos or other substances to study participants?	**No**

A.3 If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be referred to the Senate Research Ethics Committee.		*Delete as appropriate*
12.	Does your project involve participants who are under the age of 18?	**No**
13.	Does your project involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	**No**
14.	Does your project involve participants who are recruited because they are staff or students of City University London? For example, students studying on a specific course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	**No**
15.	Does your project involve intentional deception of participants?	**No**
16.	Does your project involve participants taking part without their informed consent?	**No**
17.	Does your project pose a risk to participants or other individuals greater than that in normal working life?	**No**
18.	Does your project pose a risk to you, the researcher, greater than that in normal working life?	**No**

A.4 If your answer to the following question (19) is YES and your answer to all questions 1 – 18 is NO, you must complete part B of this form.		
19.	Does your project involve human participants or their identifiable personal data? For example, as interviewees, respondents to a survey or participants in testing.	**No**

Appendix 2: Overview of Amazon Web Services

AWS is currently made up of the following components. For more detail on each see the Amazon White Paper "Overview of Amazon Web Services" [19]

Compute
Amazon EC2 - Virtual Servers in the Cloud
Amazon EC2 Container Registry - Store and Retrieve Docker Images
Amazon EC2 Container Service - Run and Manage Docker Containers
AWS Elastic Beanstalk - Run and Manage Web Apps
AWS Lambda - Run Your Code in Response to Events
Auto Scaling - Automatic Elasticity
Elastic Load Balancing - High Scale Load Balancing

Storage & Content Delivery
Amazon S3 - Scalable Storage in the Cloud
Amazon CloudFront - Global Content Delivery Network
Amazon EBS - EC2 Block Storage Volumes
Amazon Elastic File System - Fully Managed File System for EC2
Amazon Glacier - Low-Cost Archive Storage in the Cloud
AWS Import/Export Snowball - Large Scale Data Transport
AWS Storage Gateway - Hybrid Storage Integration

Database
Amazon RDS - Managed Relational Database Service
AWS Database Migration Service - Migrate Databases with Minimal Downtime
Amazon DynamoDB - Managed NoSQL Database
Amazon ElastiCache - In-Memory Caching Service

Networking
Amazon VPC – Virtual Private Cloud, Isolated Cloud Resources
AWS Direct Connect - Dedicated Network Connection to AWS
Elastic Load Balancing - High Scale Load Balancing
Amazon Route 53 - Scalable Domain Name System

Analytics
Amazon QuickSight - Fast Business Intelligence Service
Amazon Redshift - Fast, Simple, Cost-Effective Data Warehousing
Amazon Machine Learning - Machine Learning for Developers
Amazon Kinesis - Work with Real-Time Streaming Data
Amazon Elasticsearch Service - Run and Scale Elasticsearch Clusters
Amazon EMR - Hosted Hadoop Framework
AWS Data Pipeline - Orchestration Service for Periodic, Data-Driven Workflows

Enterprise Applications
Amazon WorkSpaces - Desktop Computing Service
Amazon WorkMail - Secure and Managed Business Email and Calendaring
Amazon WorkDocs - Enterprise Storage and Sharing Service

Mobile Services
AWS Mobile Hub - Build, Test, and Monitor Mobile Apps
Amazon API Gateway - Build, Deploy, and Manage APIs
Amazon Cognito - User Identity and App Data Synchronization
AWS Device Farm - Test Android, FireOS, and iOS Apps on Real Devices in the Cloud
Amazon Mobile Analytics - Collect, View, and Export App Analytics

AWS Mobile SDK - Mobile Software Development Kit
Amazon SNS - Push Notification Service

Internet of Things
AWS IoT - Connect Devices to the Cloud

- *Device SDK* - Connect, Authenticate and Exchange Messages
- *Registry* - Unique Identity to Each Device
- *Device Shadows* - Persistent Device State
- *Rules Engine* - Transform Device Messages Based on Rules

Developer Tools
AWS CodeCommit - Source Code Management, Store Code in Private Git Repositories
AWS CodeDeploy - Automate Code Deployment
AWS CodePipeline - Release Software using Continuous Delivery

Management Tools
Amazon CloudWatch - Monitoring & Logs, Monitor Resources and Applications
AWS CloudFormation - Create and Manage Resources with Templates
AWS CloudTrail - Track User Activity and API Usage
AWS Config - Track Resource Inventory and Changes
AWS OpsWorks - Automate Operations with Chef
AWS Service Catalog - Create and Use Standardized Products
Trusted Advisor - Optimize Performance and Security

Security & Identity
AWS Identity and Access Management (IAM) - Access Control, Manage User Access and Encryption Keys
AWS Certificate Manager - Provision, Manage, and Deploy SSL/TLS Certificates
AWS CloudHSM - Hardware-Based Key Storage for Regulatory Compliance
AWS Key Management Service - Managed Creation and Control of Encryption Keys
AWS Directory Service - Host and Manage Active Directory
Amazon Inspector - Analyze Application Security
AWS WAF - Web Application Firewall, Filter Malicious Web Traffic

Application Services
Amazon API Gateway - Build, Deploy, and Manage APIs
Amazon AppStream - Low-Latency Application Streaming
Amazon CloudSearch - Managed Search Service
Amazon Elastic Transcoder - Easy-to-Use Scalable Media Transcoding
Amazon SES - Email Sending and Receiving Service
Amazon SNS - Push Notification Service
Amazon SQS - Message Queue Service
Amazon SWF - Workflow Service for Coordinating Application Components

Appendix 3: Web app code

The code for the Web app has the following directory structure

- webapp
  - Dockerfile
  - webapp
    - app.py
    - requirements.txt

Dockerfile
This contains the details for the image which is based on Ubuntu linux, with python installed. The python libraries required are in requirements.txt and the app is run by the CMD line on port 5000.

```
FROM ubuntu:14.04
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q python-all
python-pip
ADD ./webapp/requirements.txt /tmp/requirements.txt
RUN pip install -qr /tmp/requirements.txt
ADD ./webapp /opt/webapp/
WORKDIR /opt/webapp
EXPOSE 5000
CMD ["python", "app.py"]
```

requirements.txt
These are the python libraries that are needed.

```
flask
boto3
```

app.py
This is the (very basic) flask web app which displays the data held in the ThingsDynamoDB table

```
import os
from flask import Flask
import boto3

app = Flask(__name__)

@app.route("/")
def table():
    # Connect to DynamoDB table. No access keys are needed here because the access is controlled by the EC2Role
    client = boto3.client('dynamodb', 'us-east-1')

    # Get all data items from the table and construct a HTML table
    response = client.scan(TableName='ThingsDynamoDBTable')
    htmltext = "<table>"
    for i in response["Items"]:
        htmltext = htmltext + "<tr><td>" + i['ThingID']['S'] + "</td><td>" + i['Value']['S'] +  "</td><td>" + i['Timestamp']['S'] + "</td></tr>"
    htmltext = htmltext + "</table>"
    return htmltext

if __name__ == '__main__':
    # Bind to PORT if defined, otherwise default to 5000.
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port)
```

Appendix 4: User documentation

To run all the code used in this project you need to have the following:

1) An AWS account (Sign up at http://aws.amazon.com/)
2) AWS Command Line Interface (CLI) installed and configured (See http://docs.aws.amazon.com/cli/latest/userguide/installing.html)
3) An AWS key pair (see http://docs.aws.amazon.com/gettingstarted/latest/wah/getting-started-prereq.html)
4) Docker installed (See https://docs.docker.com/engine/installation/)

At the command line, navigate into the folder of CloudFormation templates and set up the Repository for the Docker image using the Repository.json template using the following command:

```
aws cloudformation create-stack --stack-name Repository --template-body file://Repository.json --capabilities CAPABILITY_IAM
```

Next navigate into the webapp folder which contains the DockerFile and create the Docker image as follows (note the full stop at the end):

```
docker build -t analysiswebapp .
```

Next retrieve the appropriate command to authenticate the Docker client to the Repository with the following command:

```
aws ecr get-login --region us-east-1
```

This returns a long docker command which you should cut and paste into the command line and run exactly as is. This will give you a valid connection for 12 hours.

Next tag the image:

```
docker tag analysiswebapp:latest [your 12 digit aws account no].dkr.ecr.us-east-1.amazonaws.com/analysiswebapp:latest

e.g. docker tag analysiswebapp:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/analysiswebapp:latest
```

Finally push the image to the AWS Repository:

```
docker push [your 12 digit aws account no].dkr.ecr.us-east-1.amazonaws.com/analysiswebapp:latest

e.g. docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/analysiswebapp:latest
```

Navigate back to the folder with the CloudFormation templates and you are now ready to create the rest of the infrastructure stack using the following command:

```
aws cloudformation create-stack --stack-name MainStack --template-body file://Main.json --parameters ParameterKey=KeyName,ParameterValue=[your key name]
```

```
ParameterKey=AlarmEmailAddress,ParameterValue=[your email address] --capabilities CAPABILITY_IAM

e.g. aws cloudformation create-stack --stack-name MainStack --template-body file://Main.json --parameters ParameterKey=KeyName,ParameterValue=MyKey ParameterKey=AlarmEmailAddress,ParameterValue=me@gmail.com --capabilities CAPABILITY_IAM
```

During the provisioning of this stack you will receive an email asking you to click a link to subscribe to the alert service – click on this link to activate it.

To see when the stack is complete (allow about 6 minutes), use:

```
aws cloudformation describe-stacks --stack-name MainStack
```

(or use the CloudFormation Management Console)

When it is complete you will get a message like this which contains the stack details and the outputs:

```
STACKS  2016-09-04T07:28:33.923Z  False
    arn:aws:cloudformation:us-east-1:503984416348:stack/ali0820/2fba53b0-7271-11e6-b5f4-50a686e4bbe6
    ali0820CREATE_COMPLETE
CAPABILITIES CAPABILITY_IAM
OUTPUTS  ARN of AnalysisWebAppAnalysisWebAppName   arn:aws:ecs:us-east-1:503984416348:service/MainStack-WebAppContainerService-JEV2LOZ4YGAZ
OUTPUTS  Name of the ECS Cluster  ECSClusterNameMainStack-ECSCluster-18VMCL5W1UKBO
OUTPUTS  Name of the ECS Cluster  AutoScalingGroupName MainStack-ECSAutoScalingGroup-K2TDH8I0Y87L
PARAMETERS  KeyName  MyKey
PARAMETERS AlarmEmailAddress    me@gmail.com
```

You need to use these outputs as inputs to the ServiceScalingStack. The ECS Cluster Name and The AutoScaling Group Names can be cut and pasted exactly, but the AnalysisWebApp Name needs to be just the part after ":service/" as follows:

```
aws cloudformation create-stack --stack-name ServiceScalingStack --template-body file://ServiceScaling.json --parameters ParameterKey=AnalysisWebAppName,ParameterValue=[your Analysis Web App Name] ParameterKey=ECSClusterName,ParameterValue=[your ECS Cluster Name] ParameterKey=AutoScalingGroupName,ParameterValue=[your AutoScaling Group Name] --capabilities CAPABILITY_IAM
```

for example:

```
aws cloudformation create-stack --stack-name ServiceScalingStack --template-body file://ServiceScaling.json --parameters ParameterKey=AnalysisWebAppName,ParameterValue=MainStack-WebAppContainerService-H3LL02H6H7JJ ParameterKey=ECSClusterName,ParameterValue=MainStack-ECSCluster-JAS0NL5W1UKBO ParameterKey=AutoScalingGroupName,ParameterValue=MainStack-ECSAutoScalingGroup-1SAACSI0Y87L --capabilities CAPABILITY_IAM
```

Again, to see when the stack has completed, use:

```
aws cloudformation describe-stacks --stack-name ServiceScalingStack
```

To test the stack, first you need to have subscribed to the alert service emails by clicking on the link sent in the email. Navigate to the folder containing the StackTest.py script and edit it to add your AWS public and private key (line 22), and the correct URLs for the API (line 46) and the ELB (line 80) which can be found using the Management Console. Then run the script using the following command:

```
python StackTest.py
```

This will output the number of alarm emails expected. This can be compared with the number of emails actually received to test that the threshold alert is working.

It will also output a message saying whether the database test has been a success or failure. This means that the data in the database matches that which was sent.

It will also output a message saying whether the app test has been a success or failure. This means that the data output by the web app matches that which was sent.

If you want to run the test again, you need to go into the AWS DynamoDB console and manually delete all the data from the ThingsDynamoDB table.

Appendix 5: Integration test results

Repository stack

▼ Resources

Logical ID	Physical ID	Type	Status	Status Reason
ECSRepository	analysiswebapp	AWS::ECR::Repository	CREATE_COMPLETE	

▼ Events

2016-08-13	Status	Type	Logical ID	Status reason
▸ 10:19:04 UTC+0100	CREATE_COMPLETE	AWS::CloudFormation::Stack	Repository	
▸ 10:19:01 UTC+0100	CREATE_COMPLETE	AWS::ECR::Repository	ECSRepository	
▸ 10:19:01 UTC+0100	CREATE_IN_PROGRESS	AWS::ECR::Repository	ECSRepository	Resource creation Initiated
10:18:59 UTC+0100	CREATE_IN_PROGRESS	AWS::ECR::Repository	ECSRepository	
▸ 10:18:56 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	Repository	User Initiated

Main stack

▼ Resources

Logical ID	Physical ID	Type	Status
APIDeployment	dd7vl5	AWS::ApiGateway::Deployment	CREATE_COMPLETE
APIExecutionRole	ali0820-APIExecutionRole-ASEXEJ9RNY7P	AWS::IAM::Role	CREATE_COMPLETE
APIMethod	ali08-APIMe-OQZKNZTBEOER	AWS::ApiGateway::Method	CREATE_COMPLETE
APIResource	6rgxad	AWS::ApiGateway::Resource	CREATE_COMPLETE
AlarmSNSTopic	arn:aws:sns:us-east-1:503984416348:ThresholdBreachAlarm	AWS::SNS::Topic	CREATE_COMPLETE
ContainerTaskDefinition	arn:aws:ecs:us-east-1:503984416348:task-definition/ali0820-ContainerTaskDefinition-TFVRYUVTSZM0:1	AWS::ECS::TaskDefinition	CREATE_COMPLETE
EC2InstanceProfile	ali0820-EC2InstanceProfile-IIXQOF2NQH8T	AWS::IAM::InstanceProfile	CREATE_COMPLETE
EC2Role	ali0820-EC2Role-H77ZFZ1ADFR3	AWS::IAM::Role	CREATE_COMPLETE
ECSASGInstanceScaleDownPolicy	arn:aws:autoscaling:us-east-1:503984416348:scalingPolicy:cee16af3-cf15-4ab0-a522-e4d83be6f948:autoScalingGroupName/ali0820-ECSAutoScalingGroup-K2TDH8I0Y87L:policyName/ali0820-ECSASGInstanceScaleDownPolicy-TPJWU9JWSPO4	AWS::AutoScaling::ScalingPolicy	CREATE_COMPLETE
ECSASGInstanceScaleUpPolicy	arn:aws:autoscaling:us-east-1:503984416348:scalingPolicy:71673873-cd2b-4975-957a-c63ab7ecd097:autoScalingGroupName/ali0820-ECSAutoScalingGroup-K2TDH8I0Y87L:policyName/ali0820-ECSASGInstanceScaleUpPolicy-1QRWI2IF52LRT	AWS::AutoScaling::ScalingPolicy	CREATE_COMPLETE
ECSAutoScalingGroup	ali0820-ECSAutoScalingGroup-K2TDH8I0Y87L	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE
ECSAutoScalingGroupInstances	ali0820-ECSAutoScalingGroupInstances-RBGNVC80TI8C	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE
ECSCluster	ali0820-ECSCluster-18VMCL5W1UKBO	AWS::ECS::Cluster	CREATE_COMPLETE
ECSElasticLoadBalancer	ali0820-ECSElastic-8KIAYM9AAJ0Q	AWS::ElasticLoadBalancing::LoadBalancer	CREATE_COMPLETE
ECSServiceRole	ali0820-ECSServiceRole-6DX2B02RKRWE	AWS::IAM::Role	CREATE_COMPLETE
InsufficientMemoryAlarm	ali0820-InsufficientMemoryAlarm-1D7CGCDQUVFXH	AWS::CloudWatch::Alarm	CREATE_COMPLETE
KinesisStream	ThingsStream	AWS::Kinesis::Stream	CREATE_COMPLETE
KinesisToDynamoLambdaMapping	6a563f35-a5b5-46f7-9f72-3fecdfefe489	AWS::Lambda::EventSourceMapping	CREATE_COMPLETE
KinesisToSNSLambdaMapping	7cf884c7-2661-496c-be9c-6300f70c9038	AWS::Lambda::EventSourceMapping	CREATE_COMPLETE
LambdaExecutionRole	ali0820-LambdaExecutionRole-9BF8PA4LP2YE	AWS::IAM::Role	CREATE_COMPLETE
NodeSG	ali0820-NodeSG-1MPKVU78RPCL7	AWS::EC2::SecurityGroup	CREATE_COMPLETE
SendAlertsToSNSLambda	ali0820-SendAlertsToSNSLambda-QZBE86KVFA44	AWS::Lambda::Function	CREATE_COMPLETE
SendDataToKinesisAPI	nkguts5gfl	AWS::ApiGateway::RestApi	CREATE_COMPLETE
SendThingsToDynamoLambda	ali0820-SendThingsToDynamoLambda-WL49E4BPHSJC	AWS::Lambda::Function	CREATE_COMPLETE
ThingsDynamoDBTable	ThingsDynamoDBTable	AWS::DynamoDB::Table	CREATE_COMPLETE
ThresholdsDynamoDBTable	ThresholdsDynamoDBTable	AWS::DynamoDB::Table	CREATE_COMPLETE
TooMuchMemoryAlarm	ali0820-TooMuchMemoryAlarm-1KNC6U0NCCV3P	AWS::CloudWatch::Alarm	CREATE_COMPLETE
WebAppContainerService	arn:aws:ecs:us-east-1:503984416348:service/ali0820-WebAppContainerService-JEV2LOZ4YGAZ	AWS::ECS::Service	CREATE_COMPLETE

▾ Events

2016-09-04	Status	Type	Logical ID	Status reason
08:34:52 UTC+0100	CREATE_COMPLETE	AWS::CloudFormation::Stack	ali0820	
08:34:49 UTC+0100	CREATE_COMPLETE	AWS::ECS::Service	WebAppContainerService	
08:34:25 UTC+0100	CREATE_COMPLETE	AWS::CloudWatch::Alarm	TooMuchMemoryAlarm	
08:34:25 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	TooMuchMemoryAlarm	Resource creation Initiated
08:34:24 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	TooMuchMemoryAlarm	
08:34:24 UTC+0100	CREATE_COMPLETE	AWS::CloudWatch::Alarm	InsufficientMemoryAlarm	
08:34:24 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	InsufficientMemoryAlarm	Resource creation Initiated
08:34:23 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	InsufficientMemoryAlarm	
08:34:19 UTC+0100	CREATE_COMPLETE	AWS::AutoScaling::ScalingPolicy	ECSASGInstanceScaleUpPolicy	
08:34:19 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::ScalingPolicy	ECSASGInstanceScaleUpPolicy	Resource creation Initiated
08:34:19 UTC+0100	CREATE_IN_PROGRESS	AWS::ECS::Service	WebAppContainerService	Resource creation Initiated
08:34:18 UTC+0100	CREATE_COMPLETE	AWS::AutoScaling::ScalingPolicy	ECSASGInstanceScaleDownPolicy	
08:34:18 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::ScalingPolicy	ECSASGInstanceScaleDownPolicy	Resource creation Initiated
08:34:18 UTC+0100	CREATE_IN_PROGRESS	AWS::ECS::Service	WebAppContainerService	
08:34:18 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::ScalingPolicy	ECSASGInstanceScaleUpPolicy	
08:34:18 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::ScalingPolicy	ECSASGInstanceScaleDownPolicy	
08:34:13 UTC+0100	CREATE_COMPLETE	AWS::AutoScaling::AutoScalingGroup	ECSAutoScalingGroup	
08:34:12 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::AutoScalingGroup	ECSAutoScalingGroup	Received SUCCESS signal with UniqueId i-0126d7773f119d30b
08:32:46 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::AutoScalingGroup	ECSAutoScalingGroup	Resource creation Initiated
08:32:45 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::AutoScalingGroup	ECSAutoScalingGroup	
08:32:41 UTC+0100	CREATE_COMPLETE	AWS::AutoScaling::LaunchConfiguration	ECSAutoScalingGroupInstances	
08:32:41 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::LaunchConfiguration	ECSAutoScalingGroupInstances	Resource creation Initiated
08:32:40 UTC+0100	CREATE_IN_PROGRESS	AWS::AutoScaling::LaunchConfiguration	ECSAutoScalingGroupInstances	
08:32:34 UTC+0100	CREATE_COMPLETE	AWS::IAM::InstanceProfile	EC2InstanceProfile	
08:31:15 UTC+0100	CREATE_COMPLETE	AWS::Lambda::EventSourceMapping	KinesisToSNSLambdaMapping	
08:31:15 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::EventSourceMapping	KinesisToSNSLambdaMapping	Resource creation Initiated
08:31:14 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::EventSourceMapping	KinesisToSNSLambdaMapping	
08:31:11 UTC+0100	CREATE_COMPLETE	AWS::ApiGateway::Deployment	APIDeployment	
08:31:11 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::Deployment	APIDeployment	Resource creation Initiated
08:31:11 UTC+0100	CREATE_COMPLETE	AWS::Lambda::EventSourceMapping	KinesisToDynamoLambdaMapping	
08:31:10 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::EventSourceMapping	KinesisToDynamoLambdaMapping	Resource creation Initiated
08:31:10 UTC+0100	CREATE_COMPLETE	AWS::Lambda::Function	SendAlertsToSNSLambda	
08:31:10 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::Deployment	APIDeployment	
08:31:09 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::EventSourceMapping	KinesisToDynamoLambdaMapping	
08:31:09 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::Function	SendAlertsToSNSLambda	Resource creation Initiated
08:31:08 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::Function	SendAlertsToSNSLambda	
08:31:05 UTC+0100	CREATE_COMPLETE	AWS::ApiGateway::Method	APIMethod	
08:31:04 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::Method	APIMethod	Resource creation Initiated
08:31:04 UTC+0100	CREATE_COMPLETE	AWS::Lambda::Function	SendThingsToDynamoLambda	
08:31:04 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::Function	SendThingsToDynamoLambda	Resource creation Initiated
08:31:04 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::Method	APIMethod	
08:31:03 UTC+0100	CREATE_IN_PROGRESS	AWS::Lambda::Function	SendThingsToDynamoLambda	
08:30:59 UTC+0100	CREATE_COMPLETE	AWS::IAM::Role	LambdaExecutionRole	
08:30:59 UTC+0100	CREATE_COMPLETE	AWS::IAM::Role	APIExecutionRole	
08:30:33 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::InstanceProfile	EC2InstanceProfile	Resource creation Initiated
08:30:33 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::InstanceProfile	EC2InstanceProfile	
08:30:27 UTC+0100	CREATE_COMPLETE	AWS::IAM::Role	EC2Role	
08:29:52 UTC+0100	CREATE_COMPLETE	AWS::IAM::Role	ECSServiceRole	
08:29:48 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	LambdaExecutionRole	Resource creation Initiated
08:29:47 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	APIExecutionRole	Resource creation Initiated
08:29:47 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	LambdaExecutionRole	
08:29:47 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	APIExecutionRole	
08:29:43 UTC+0100	CREATE_COMPLETE	AWS::Kinesis::Stream	KinesisStream	
08:29:16 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	EC2Role	Resource creation Initiated
08:29:15 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	EC2Role	
08:29:12 UTC+0100	CREATE_COMPLETE	AWS::DynamoDB::Table	ThresholdsDynamoDBTable	
08:29:11 UTC+0100	CREATE_COMPLETE	AWS::DynamoDB::Table	ThingsDynamoDBTable	
08:28:57 UTC+0100	CREATE_COMPLETE	AWS::EC2::SecurityGroup	NodeSG	
08:28:56 UTC+0100	CREATE_IN_PROGRESS	AWS::EC2::SecurityGroup	NodeSG	Resource creation Initiated
08:28:53 UTC+0100	CREATE_COMPLETE	AWS::SNS::Topic	AlarmSNSTopic	
08:28:46 UTC+0100	CREATE_COMPLETE	AWS::ECS::TaskDefinition	ContainerTaskDefinition	
08:28:46 UTC+0100	CREATE_COMPLETE	AWS::ApiGateway::Resource	APIResource	
08:28:46 UTC+0100	CREATE_IN_PROGRESS	AWS::ECS::TaskDefinition	ContainerTaskDefinition	Resource creation Initiated
08:28:46 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::Resource	APIResource	Resource creation Initiated
08:28:45 UTC+0100	CREATE_IN_PROGRESS	AWS::ECS::TaskDefinition	ContainerTaskDefinition	
08:28:45 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::Resource	APIResource	
08:28:42 UTC+0100	CREATE_IN_PROGRESS	AWS::SNS::Topic	AlarmSNSTopic	Resource creation Initiated
08:28:42 UTC+0100	CREATE_COMPLETE	AWS::ElasticLoadBalancing::LoadBalancer	ECSElasticLoadBalancer	
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::ElasticLoadBalancing::LoadBalancer	ECSElasticLoadBalancer	Resource creation Initiated
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::Kinesis::Stream	KinesisStream	Resource creation Initiated
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	ThresholdsDynamoDBTable	Resource creation Initiated
08:28:41 UTC+0100	CREATE_COMPLETE	AWS::ApiGateway::RestApi	SendDataToKinesisAPI	
08:28:41 UTC+0100	CREATE_COMPLETE	AWS::ECS::Cluster	ECSCluster	
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::SNS::Topic	AlarmSNSTopic	
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::RestApi	SendDataToKinesisAPI	Resource creation Initiated
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	ECSServiceRole	Resource creation Initiated
08:28:41 UTC+0100	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	ThingsDynamoDBTable	Resource creation Initiated
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::ECS::Cluster	ECSCluster	Resource creation Initiated
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::Kinesis::Stream	KinesisStream	
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	ECSServiceRole	
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	ThresholdsDynamoDBTable	

08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::ElasticLoadBalancing::LoadBalancer	ECSElasticLoadBalancer	
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::EC2::SecurityGroup	NodeSG	
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::ApiGateway::RestApi	SendDataToKinesisAPI	
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	ThingsDynamoDBTable	
08:28:40 UTC+0100	CREATE_IN_PROGRESS	AWS::ECS::Cluster	ECSCluster	
08:28:33 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	ali0820	User Initiated

▾ Outputs

Key	Value	Description
AnalysisWebAppName	arn:aws:ecs:us-east-1:503984416348:service/ali0820-WebAppContainerService-JEV2LOZ4YGAZ	ARN of AnalysisWebApp
ECSClusterName	ali0820-ECSCluster-18VMCL5W1UKBO	Name of the ECS Cluster
AutoScalingGroupName	ali0820-ECSAutoScalingGroup-K2TDH8I0Y87L	Name of the ECS Cluster

Service scaling stack

▾ Resources

TooMuchCPUAlarm

Logical ID	Physical ID	Type	Status
ApplicationAutoScalingRole	ali0820scaling-ApplicationAutoScalingRole-1ESBLLH6CSF0C	AWS::IAM::Role	CREATE_COMPLETE
ContainerScalingDownPolicy	arn:aws:autoscaling:us-east-1:503984416348:scalingPolicy:007ae85c-c030-4470-a5c6-377777b6f745:resource/ecs/service/ali0820-ECSCluster-18VMCL5W1UKBO/ali0820-WebAppContainerService-JEV2LOZ4YGAZ:policyName/StepDownPolicy	AWS::ApplicationAutoScaling::Scalin…	CREATE_COMPLETE
ContainerScalingUpPolicy	arn:aws:autoscaling:us-east-1:503984416348:scalingPolicy:007ae85c-c030-4470-a5c6-377777b6f745:resource/ecs/service/ali0820-ECSCluster-18VMCL5W1UKBO/ali0820-WebAppContainerService-JEV2LOZ4YGAZ:policyName/StepUpPolicy	AWS::ApplicationAutoScaling::Scalin…	CREATE_COMPLETE
InsufficientCPUAlarm	ali0820scaling-InsufficientCPUAlarm-1DD26QYMTIF45	AWS::CloudWatch::Alarm	CREATE_COMPLETE
TooMuchCPUAlarm	ali0820scaling-TooMuchCPUAlarm-1TLV7M97PK5C7	AWS::CloudWatch::Alarm	CREATE_COMPLETE
UpdateServiceDesiredCount	service/ali0820-ECSCluster-18VMCL5W1UKBO/ali0820-WebAppContainerService-JEV2LOZ4YGAZ|ecs:service:DesiredCount|ecs	AWS::ApplicationAutoScaling::Scala…	CREATE_COMPLETE

▾ Events

2016-09-04	Status	Type	Logical ID	Status reason
08:40:56 UTC+0100	CREATE_COMPLETE	AWS::CloudFormation::Stack	ali0820scaling	
08:40:53 UTC+0100	CREATE_COMPLETE	AWS::CloudWatch::Alarm	InsufficientCPUAlarm	
08:40:53 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	InsufficientCPUAlarm	Resource creation Initiated
08:40:53 UTC+0100	CREATE_COMPLETE	AWS::CloudWatch::Alarm	TooMuchCPUAlarm	
08:40:53 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	TooMuchCPUAlarm	Resource creation Initiated
08:40:53 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	InsufficientCPUAlarm	
08:40:52 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudWatch::Alarm	TooMuchCPUAlarm	
08:40:49 UTC+0100	CREATE_COMPLETE	AWS::ApplicationAutoScaling::ScalingPolicy	ContainerScalingUpPolicy	
08:40:49 UTC+0100	CREATE_COMPLETE	AWS::ApplicationAutoScaling::ScalingPolicy	ContainerScalingDownPolicy	
08:40:49 UTC+0100	CREATE_IN_PROGRESS	AWS::ApplicationAutoScaling::ScalingPolicy	ContainerScalingUpPolicy	Resource creation Initiated
08:40:49 UTC+0100	CREATE_IN_PROGRESS	AWS::ApplicationAutoScaling::ScalingPolicy	ContainerScalingDownPolicy	Resource creation Initiated
08:40:48 UTC+0100	CREATE_IN_PROGRESS	AWS::ApplicationAutoScaling::ScalingPolicy	ContainerScalingUpPolicy	
08:40:48 UTC+0100	CREATE_IN_PROGRESS	AWS::ApplicationAutoScaling::ScalingPolicy	ContainerScalingDownPolicy	
08:40:44 UTC+0100	CREATE_COMPLETE	AWS::ApplicationAutoScaling::ScalableTarget	UpdateServiceDesiredCount	
08:40:44 UTC+0100	CREATE_IN_PROGRESS	AWS::ApplicationAutoScaling::ScalableTarget	UpdateServiceDesiredCount	Resource creation Initiated
08:40:42 UTC+0100	CREATE_IN_PROGRESS	AWS::ApplicationAutoScaling::ScalableTarget	UpdateServiceDesiredCount	
08:40:39 UTC+0100	CREATE_COMPLETE	AWS::IAM::Role	ApplicationAutoScalingRole	
08:39:28 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	ApplicationAutoScalingRole	Resource creation Initiated
08:39:27 UTC+0100	CREATE_IN_PROGRESS	AWS::IAM::Role	ApplicationAutoScalingRole	
08:39:23 UTC+0100	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	ali0820scaling	User Initiated

Appendix 6: System testing Python script

```
import requests
import json
import datetime
import random
import boto3
import time
import re


######################
# Set data thresholds #
######################

# Connect to database (use relevant keys)
dynamodb = boto3.client('dynamodb','us-east-1',    aws_access_key_id='XXX',aws_secret_access_key='XXX') # Need to
replace these with correct keys

# Set thresholds for alarms
MaxThreshold = 25
MinThreshold = 18
response = dynamodb.put_item(TableName ='ThresholdsDynamoDBTable',
Item={'ThingID':{"S":"things/harrow/001/temperature"}, 'MaxThreshold':{"S":str(MaxThreshold)},
'MinThreshold':{"S":str(MinThreshold)}})

###################
# Send data to API #
###################

# Initialise variables
headers = {'content-type': 'application/json'} # tells the API that it is being sent JSON
messagelist =[] # Initialise empty list for messages
numTests=10 # Number of data points to be sent
alarms=0

# Send randomly generated data points to API
for i in range (0,numTests):
    temperature = round(random.gauss(20,3),2)
    if temperature > MaxThreshold or temperature < MinThreshold:
        alarms+=1
    messagetext = json.dumps({"Data":{"id":"things/harrow/001", "temperature":str(temperature) , "timestamp":
str(datetime.datetime.utcnow())}})
    messagelist.append(messagetext) # Save messages for checking later
    response = requests.put("https://asgxbj31ue.execute-api.us-east-1.amazonaws.com/prod/thingsstream",
data=messagetext, headers=headers) # Need to replace this with correct API URL
    if response.status_code != 200:
        print("Failed data upload. Response code = " + str(response.status_code)) # Check they are successfully
sent - i.e. get a 200 response

# Show how many expected alarm emails
print("Number of alarm emails expected: " + str(alarms))

# Allow time for data to reach database before checking them
time.sleep(3)

#########################
# Get data from database #
#########################

# Get all data from table
response = dynamodb.scan(TableName='ThingsDynamoDBTable')

# Create a list of data points formatted in the same way as the input data
outputlist =[]
for i in response["Items"]:
    outputtext = json.dumps({"Data":{"id":i['ThingID']['S'].replace('/temperature',''), "temperature":
i['Value']['S'], "timestamp": i['Timestamp']['S']}})
    outputlist.append(outputtext)

# Check that input and output lists are the same
if len(set(outputlist).intersection(messagelist))==numTests:
    print('Successful database test')
else:
    print('Failed database test')

########################
## Get data from webapp #
########################

# Connect to web app via load balancer
response = requests.get("http://ali0820-ECSElastic-1R97EVD0VGYV2-838464487.us-east-1.elb.amazonaws.com") # Need to
replace this with correct ELB URL
if response.status_code != 200:
    print("Failed connection to web app. Response code = " + str(response.status_code)) # Check the web app is
working - i.e. get a 200 response

# Get web app output and scrape data
apptext = response.content.decode("utf-8")
p = re.compile('<td>(.*?)</td>')
appdata = re.findall(p, apptext)
```

```
    applist=[]
    for i in range (0,3*numTests, 3):
        apptext = json.dumps({"Data":{"id":appdata[i].replace('/temperature',''), "temperature": appdata[i+1],
    "timestamp": appdata[i+2]}})
        applist.append(apptext)

    # Check that input and output lists are the same
    if len(set(applist).intersection(messagelist))==numTests:
        print('Successful app test')
    else:
        print('Failed app test')

    print('Data sent' + str(messagelist))

```

Appendix 7: System testing results

Results from running testing script

```
Alisons-iMac:com~apple~CloudDocs Ali$ python StackTest.py
Number of alarm emails expected: 1
Successful database test
Successful app test
Data sent['{"Data": {"temperature": "19.57", "id": "things/harrow/001", "timestamp": "2016-09-09 13:33:07.381019"}}'
, '{"Data": {"temperature": "11.39", "id": "things/harrow/001", "timestamp": "2016-09-09 13:33:08.211298"}}', '{"Dat
a": {"temperature": "24.43", "id": "things/harrow/001", "timestamp": "2016-09-09 13:33:08.791000"}}', '{"Data": {"te
mperature": "18.37", "id": "things/harrow/001", "timestamp": "2016-09-09 13:33:09.367586"}}', '{"Data": {"temperatur
e": "22.86", "id": "things/harrow/001", "timestamp": "2016-09-09 13:33:09.927509"}}', '{"Data": {"temperature": "20.
07", "id": "things/harrow/001", "timestamp": "2016-09-09 13:33:10.439630"}}', '{"Data": {"temperature": "21.0", "id"
: "things/harrow/001", "timestamp": "2016-09-09 13:33:10.932964"}}', '{"Data": {"temperature": "19.28", "id": "thing
s/harrow/001", "timestamp": "2016-09-09 13:33:11.481556"}}', '{"Data": {"temperature": "18.73", "id": "things/harrow
/001", "timestamp": "2016-09-09 13:33:12.059937"}}', '{"Data": {"temperature": "18.66", "id": "things/harrow/001", "
timestamp": "2016-09-09 13:33:12.588159"}}']
```

Alert email received

AWS Notifications Today at 14:31
To: alisonwells123@gmail.com
AWS Notification Message

AN

Temperature alarm for things/harrow/001: Temperature = 11.39

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:503984416348:ThresholdBreachAlarm:6d1d092a-0471-46a1-9840-29c7732207a2&Endpoint=alisonwells123@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Screenshot of web app showing table of data

things/harrow/001/temperature	19.57	2016-09-09 13:33:07.381019
things/harrow/001/temperature	11.39	2016-09-09 13:33:08.211298
things/harrow/001/temperature	24.43	2016-09-09 13:33:08.791000
things/harrow/001/temperature	18.37	2016-09-09 13:33:09.367586
things/harrow/001/temperature	22.86	2016-09-09 13:33:09.927509
things/harrow/001/temperature	20.07	2016-09-09 13:33:10.439630
things/harrow/001/temperature	21.0	2016-09-09 13:33:10.932964
things/harrow/001/temperature	19.28	2016-09-09 13:33:11.481556
things/harrow/001/temperature	18.73	2016-09-09 13:33:12.059937
things/harrow/001/temperature	18.66	2016-09-09 13:33:12.588159

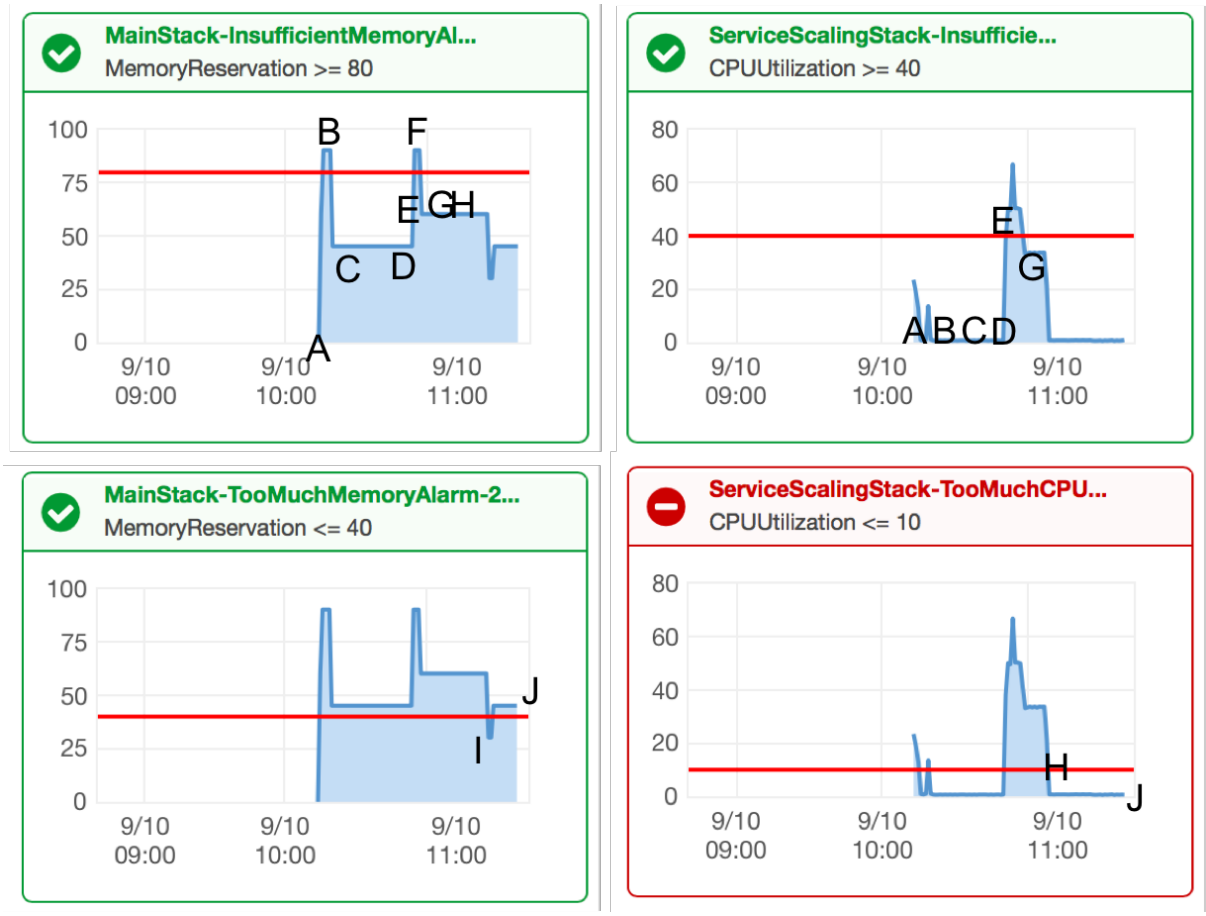
HTML from page source of web app

```
<table>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>19.57</td>
    <td>2016-09-09 13:33:07.381019</td>
```

```
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>11.39</td>
    <td>2016-09-09 13:33:08.211298</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>24.43</td>
    <td>2016-09-09 13:33:08.791000</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>18.37</td>
    <td>2016-09-09 13:33:09.367586</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>22.86</td>
    <td>2016-09-09 13:33:09.927509</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>20.07</td>
    <td>2016-09-09 13:33:10.439630</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>21.0</td>
    <td>2016-09-09 13:33:10.932964</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>19.28</td>
    <td>2016-09-09 13:33:11.481556</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>18.73</td>
    <td>2016-09-09 13:33:12.059937</td>
  </tr>
  <tr>
    <td>things/harrow/001/temperature</td>
    <td>18.66</td>
    <td>2016-09-09 13:33:12.588159</td>
  </tr>
</table>
```

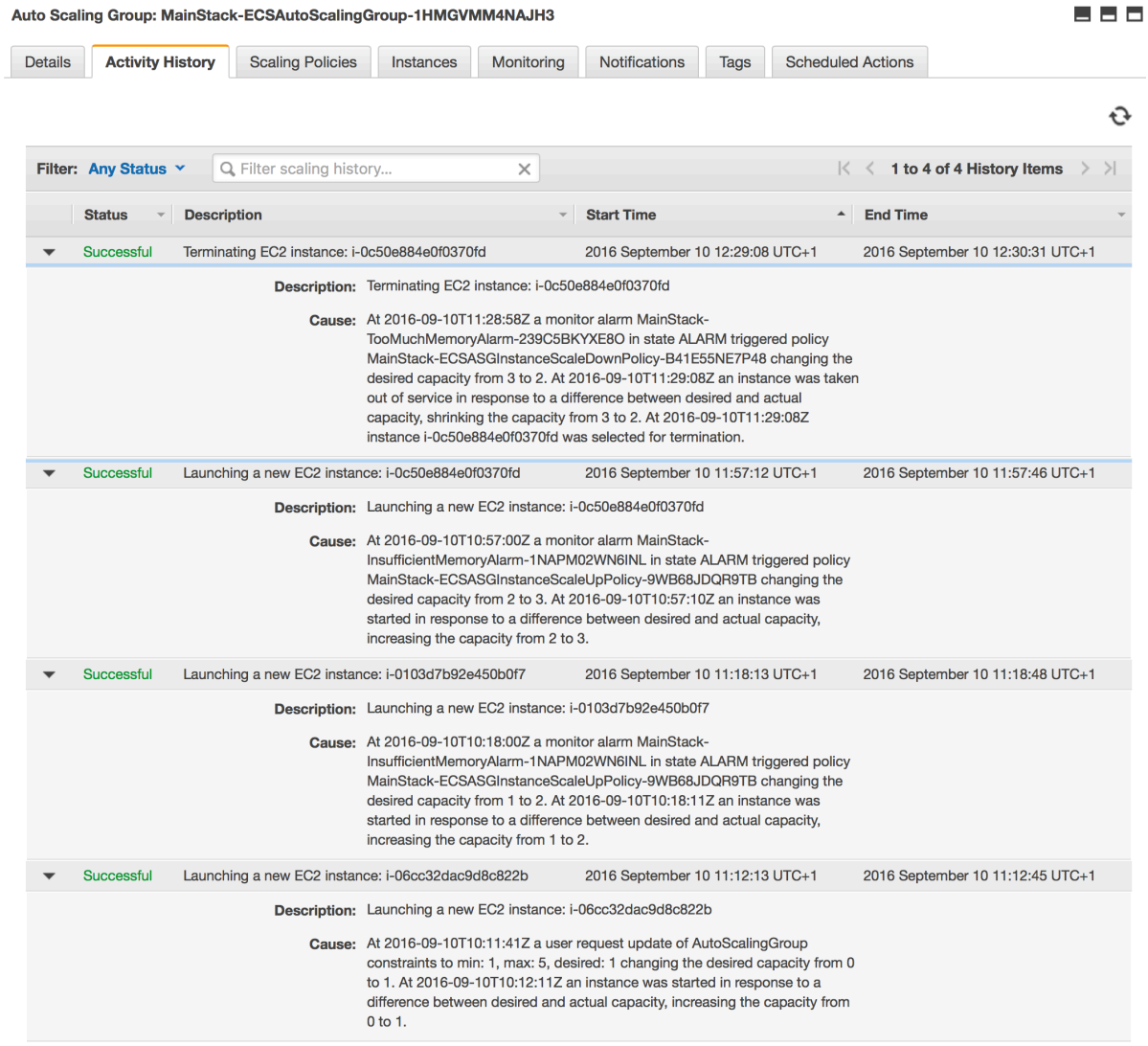
Appendix 8: Scaling testing results

CloudWatch Alarms showing changes in scaling

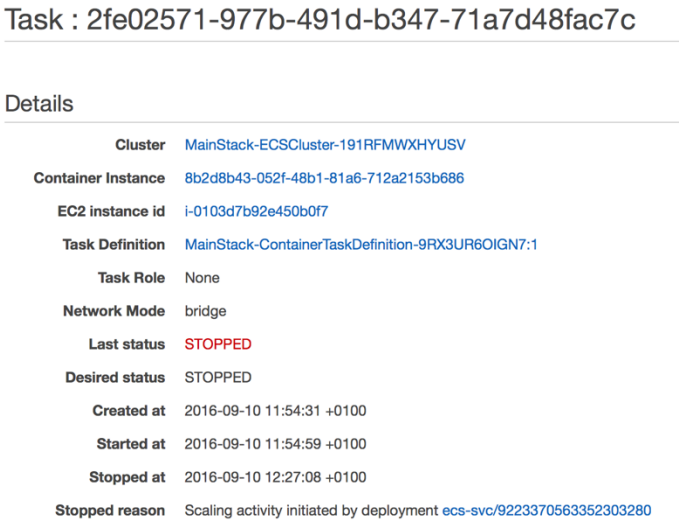


A – Initially 1 EC2 instance with 1 web app task running.
B – Insufficient memory alarm is set off as >80% of memory is reserved. This leads to a new EC2 instance being instantiated. Too much CPU alarm is also set off but it cannot reduce the number of web app tasks below the minimum of 1.
C – There are now 2 EC2 instances with 1 web app task running. Insufficient memory alarm is turned off due to >80% but >40% of memory reserved. System is stable.
D – stress load used to increase CPU Utilisation
E – Insufficient CPU Alarm is set off as CPU Utilisation now >50%. This leads to a new web app task being instantiated on the 2nd EC2 instance.
F – There are now 2 EC2 instances with 2 web app tasks running. The Insufficient memory alarm is set off as >80% of memory is reserved. This leads to a new EC2 instance being instantiated.
G – There are now 3 EC2 instances with 2 web app tasks running. Insufficient memory alarm is turned off due to >80% but >40% of memory reserved. Insufficient CPU alarm is turned off as CPU Utilisation is now <50%. System is stable.
H – Stress load turned off to reduce CPU Utilisation. Too much CPU alarm is turned on as CPU Utilisation goes below 10%. This stops one of the web app tasks.
I – There are now 3 EC2 instances with 1 web app task running. The Too much memory alarm is set off as <40% of the memory is reserved. This terminates one of the EC2 instances.
J – There are now 2 EC2 instances with 1 web app task running. The Too much CPU alarm is still turned on but it cannot reduce the number of web app tasks below the minimum of 1. System is stable.

History from AutoScalingGroup showing changes in EC2 instances



Task definition from ECS Container Service showing web app task being stopped due to scaling policy



Appendix 9: Detailed description of CloudFormation templates

Repository : Lines 1-28 : Resources : ECSRepository

This section initialises the repository for the analysiswebapp Docker image (FR 5) and allows any service to access it in the listed ways.

Main : Lines 1-13 : Parameters : KeyName & AlarmEmailAddress

This section allows for the input of parameters which will be unique to the person or organisation setting up the infrastructure – the key pair used to connect to the EC2 instances (SR 1) and the email address that will receive the alarm notifications (FR 4). More email addresses can be added through subscribing to the AlarmSNS topic via the Management console.

Main : Lines 14-28 : Resources : APIDeployment

This section makes a live deployment of the SendDataToKinesisAPI API (FR 3, SR 3) and gives it a path name of "prod" (short for production).

Main : Lines 29-67 : Resources : APIMethod

This section defines the method that translates the JSON data to the correct format for the Kinesis stream and adds it to the API (FR 2, FR 3, SR 3). A new method would be needed for each different JSON data format. Currently there is no extra authorisation required here, but it could be added to ensure that only the right people are uploading data to the system. The "RequestTemplates" section takes the data input and formats it correctly.

Main : Lines 68-77 : Resources : SendDataToKinesisAPI

This section initialises the API (FR 3, SR 3).

Main : Lines 78-93 : Resources : APIResource

This section creates a subsection of the API to receive the data (FR 3, SR 3).

Main : Lines 94-122 : Resources : ECSAutoScalingGroup

This section sets up the AutoScalingGroup of EC2 instances (NFR 3). Initially it is set to 1 instance and can scale up to 5 instances. In this example the EC2 instances are only in one Availability Zone but more could be added.

Main : Lines 123-137 : Resources : ECSAutoScalingGroupInstances

This resource contains a lot of information about how to set up each EC2 instance. The first part contains the config command that joins each instance to the ECSCluster of EC2 instances when the instance is created (NFR 3). Next, there are config commands to initialise the EC2 instances and connect them to the AutoScaling Group (NFR 3). The ImageID gives the AMI (Amazon Machine Image) ID for an image that has Linux on it and is optimised to use the ECS Container Service. The InstanceType is a t2.micro – this is the smallest type of server and was used because it is available for free and

therefore meets Constraint 5. Other more powerful server types could be used here. The SecurityGroups section links to the NodeSG Security Group that allow SSH and HTTP access to the servers (SR 1). The IAMInstanceProfile links to the role that has been set up for the EC2 servers to access the other services in the stack (SR 2). The KeyName links to the key that was given as one of the parameters that will be used to secure SSH access to the server (SR 1). The UserData section again links the EC2 instances to the AutoScaling Group (NFR 3).

Main : Lines 228-257 : Resources : ECSASGInstanceScaleDownPolicy & ECSASGInstanceScaleUpPolicy

This section initialises the policies for adding or removing instances from the AutoScaling Group (NFR 3). This basically means that the ECS Cluster is full of containers and if another were to be needed, then a new EC2 instance would need to be added to the cluster first. As this takes a couple of minutes, it would be advantageous to have a spare EC2 ready although of course this also costs money. Similarly the TooMuchMemoryAlarm will go off if there is less than 40% of the memory is allocated on the Cluster to allow an EC2 instance to be turned off to save money.

Main : Lines 258-308 : Resources : InsufficientMemoryAlarm & TooMuchMemoryAlarm

This section specifies the alarms that will set off the scaling policies for the EC2 instances in the AutoScalingGroup. InsufficientMemoryAlarm will add an extra EC2 instance if more than 80% of the memory is reserved by web app tasks, whereas TooMuchMemoryAlarm will remove an EC2 instance if less than 40% of the memory is reserved, to save money.

Main : Lines 309-336 : Resources : ThingsDynamoDBTable

This section sets up a DynamoDB table to store the time series data for later analysis (NFR 2, FR 5). Each data item is identified uniquely by the ID of the Thing it refers to and the timestamp (FR 1). This unique identifier is the only thing that needs to be set up initially as it is a NoSQL database. The ReadCapacityUnits and WriteCapacityUnits can be increased to 40,000 each if necessary to store large quantities of data at speed (NFR1, NFR 2) but should only be increased if this is likely to happen as they are charged for whether or not they are used.

Main : Lines 337-358 : Resources : ThresholdsDynamoDBTable

This section sets up the DynamoDB Table to hold the maximum and minimum thresholds that can be used to set off the alarm emails (FR 4). Each item is uniquely identified by the ID of the Thing it relates to. Again, the ReadCapacityUnits and WriteCapacityUnits can be increased to 40,000 each if necessary to store large quantities of data at speed (NFR1, NFR 2) but should only be increased if this is likely to happen as they are charged for whether or not they are used.

Main : Lines 359-379 : Resources : NodeSG

This section sets up a security group that allows SSH to the EC2 instances on port 22 and HTTP on port 80 from anywhere (represented by IP 0.0.0.0/0) (SR 1). To add extra security, the SSH could be locked down to a specific IP range.

Main : Lines 380-385 : Resources : ECSCluster

This section initialises the ECS Cluster which will have EC2 instances added to it, and then containers loaded on to them (NFR 3).

Main : Lines 386-412 : Resources : WebAppContainerService

This section sets up a group of containers and links them to the ECS Cluster, the task definition that holds the exact details of the container, the access control role that the containers can have and the load balancer that shares the load across the EC2 instances (NFR 3).

Main : Lines 413-441 : Resources : ContainerTaskDefinition

This section defines the individual container details that are grouped together by the WebAppContainerService (NFR 3). It specifies the Repository location to get the Docker image from, the port mappings to allow the web app to be accessed via HTTP and the maximum amount of memory and minimum share of CPU that the container is allowed to use. The container has been allocated here 900MiB of the 1GiB of memory available on a t2.micro server as there is only one container on each server because there is only one type of container (and therefore wouldn't make sense to have two identical containers competing for resources). If more different containers were used or larger servers, then the amount of memory or CPU could be changed.

Main : Lines 442-463 : Resources : ECSElasticLoadBalancer

This section sets up the load balancer which is specifically balancing load on HTTP traffic on port 80 (NFR 3). Currently it is only set to be in 1 Availability Zone but more could be added.

Main : Lines 464-476 : Resources : EC2InstanceProfile

This section links each EC2 instance to the EC2Role security role (SR 2).

Main : Lines 477-508 : Resources : APIExecutionRole

This security role allows the API gateway to send data to the specific Kinesis stream set up by the template (SR 2, SR 3). It allows the API to carry out any Kinesis related action on the stream, for improved security this could be locked down to the minimum actions necessary to send data.

Main : Lines 509-563 : Resources : EC2Role

This security role allows the EC2 instances to access the ECR Repository, run specific tasks on the ECS Container Service and send logs to CloudWatch (SR 2). It also allows them access to the specific ThingsDynamoDBTable (SR 2).

Main : Lines 564-601 : Resources : ECSServiceRole

This security role allows the ECS service to perform specific tasks on the load balancer and EC2 instances to manage the pool of EC2 instances in the cluster (SR 2).

Main : Lines 602-666 : Resources : LambdaExecutionRole

This security role is used for both the Lambda functions and allows access only to the specific Kinesis stream and DynamoDB tables used in the stack (SR 2). It also allows access to SNS for the alarm emails and to send logs to CloudWatch (SR 2).

M**ain : Lines 667-677 : Resources : KinesisStream**

This section initialises the Kinesis stream with 2 shards (which can process up to 1MB/s). This can be increased to 10 if necessary to process up to 10MB/s but this should only be done when this volume of data is expected as each shard is charged for separately (NFR 1).

Main : Lines 678-693 : Resources : KinesisToDynamoLambdaMapping

This section links the Kinesis stream to the Lambda function that imports the data into DynamoDB (FR 5).

Main : Lines 694-710 : Resources : KinesisToSNSLambdaMapping

This section links the Kinesis stream to the Lambda function that sends email alerts if a threshold is breached (FR 4).

Main : Lines 711-750 : Resources : SendAlertsToSNSLambda

This section sets up the Lambda function that checks whether a data item has breached a threshold and sends a message to a SNS topic (FR 4). The function is written in Python using the boto3 library to access the AWS services.

Main : Lines 751-783 : Resources : SendThingsToDynamoLambda

This section sets up the Lambda function that imports each data item to DynamoDB so that it can be analysed later (FR 5). Again it is written in Python using the boto3 library to access the AWS services.

Main : Lines 784-800 : Resources : AlarmSNSTopic

This section sets up the SNS topic that will be sent messages when a data item breaches a threshold alarm, and subscribes the email address that was given in the parameters to it (FR 4).

Main : Lines 801-822 : Outputs : AnalysisWebName, ECSClusterName & AutoScalingGroupName

These outputs are used to get the information that is needed to be input as parameters to the ServiceScaling stack. This is to get around the problem that the ScalableTarget service needs to have the name of the webapp in a certain format that is currently impossible to provide using the CloudFormation tools. If this was changed by AWS then these outputs could be removed and the ServiceScaling stack template merged into the Main template.

ServiceScaling : Lines 1-16 : Parameters : AnalysisWebAppName, ECSClusterName & AutoScalingGroupName

This section defines the parameters of the resources set up by the Main stack that are depended on by ServiceScaling stack. This is necessary because of the current inability to get the name of the AnalysisWebApp resource programmatically as described in the introduction.

ServiceScaling : Lines 17-55 : Resources : ContainerScalingUpPolicy & ContainerScalingDownPolicy

This section defines the policies for adding and removing containers. Simply they just add or remove one container and wait 60 seconds before acting again if required (NFR 3).

ServiceScaling : Lines 56-74 : Resources : UpdateServiceDesiredCount

This section defines which container should be scaled if required, and the maximum and minimum number of containers in service at once (NFR 3).

ServiceScaling : Lines 75-107 : Resources : ApplicationAutoScalingRole

This security role allows the application-autoscaling service (which scales containers not EC2 instances) to access CloudWatch alarms and the ECS container service to add and remove containers (NFR 3).

ServiceScaling : Lines 108-153 : Resources : InsufficientCPUAlarm & TooMuchCPUAlarm

This section defines CloudWatch alarms that go off if the CPU Utilisation in the AutoScaling Group of EC2 instances goes above 40% so that a new container can be added, or below 10% in which case one is removed. These figures can be tuned in practice depending on the volatility of the traffic to get the best compromise between response times and cost (NFR 3).

Appendix 10: CloudFormation templates

Repository.json

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSRepository": {
            "Type": "AWS::ECR::Repository",
            "Properties": {
                "RepositoryName": "analysiswebapp",
                "RepositoryPolicyText": {
                    "Version": "2008-10-17",
                    "Statement": [{
                        "Sid": "AllowPushPull",
                        "Effect": "Allow",
                        "Principal": "*",
                        "Action": [
                            "ecr:GetDownloadUrlForLayer",
                            "ecr:BatchGetImage",
                            "ecr:BatchCheckLayerAvailability",
                            "ecr:PutImage",
                            "ecr:InitiateLayerUpload",
                            "ecr:UploadLayerPart",
                            "ecr:CompleteLayerUpload"
                        ]
                    }]
                }
            }
        }
    }

```

Main.json

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Parameters": {
        "KeyName": {
            "Description": "Key pair to be used to connect to EC2 instances",
            "Type": "AWS::EC2::KeyPair::KeyName"
        },
        "AlarmEmailAddress": {
            "Description": "Email address to get alarm notifications",
            "Type": "String"
        }
    },
    "Resources": {
        "APIDeployment": {
            "Metadata": {
                "Description": "Live deployment of API to send data to Kinesis"
            },
            "DependsOn": "APIMethod",
            "Type": "AWS::ApiGateway::Deployment",
            "Properties": {

                "RestApiId": {
                    "Ref": "SendDataToKinesisAPI"
                },
                "StageName": "prod"
            }
        },
        "APIMethod": {
            "Metadata": {
                "Description": "Translates JSON data to correct format for Kinesis stream"
            },
            "Type": "AWS::ApiGateway::Method",
            "Properties": {

                "RestApiId": {
                    "Ref": "SendDataToKinesisAPI"
                },
                "ResourceId": {
                    "Ref": "APIResource"
                },
                "HttpMethod": "PUT",
                "AuthorizationType": "NONE",
                "MethodResponses": [{
                    "StatusCode": "200"
                }],
                "Integration": {
                    "Credentials": {
                        "Fn::GetAtt": ["APIExecutionRole", "Arn"]
                    },
                    "IntegrationHttpMethod": "POST",
                    "IntegrationResponses": [{
                        "StatusCode": "200"
                    }],
                    "PassthroughBehavior": "NEVER",
                    "RequestParameters": {
                        "integration.request.header.Content-Type": "'application/x-amz-json-1.1'"
                    },
                    "RequestTemplates": {
                        "application/json": "{\"StreamName\": \"ThingsStream\",\"Data\":
\"$util.base64Encode($input.json('$.Data'))\",\"PartitionKey\": \"pk4\"}"
                    },
                    "Type": "AWS",
                    "Uri": "arn:aws:apigateway:us-east-1:kinesis:action/PutRecord"
                }

            }
        },
        "SendDataToKinesisAPI": {
            "Metadata": {
                "Description": "API that enables data to be sent to Kinesis"
            },
            "Type": "AWS::ApiGateway::RestApi",
            "Properties": {
                "Name": "SendDataToKinesis"
            }

        },
        "APIResource": {
            "Metadata": {
                "Description": "Creates a subsection of the API to receive data"
            },
            "Type": "AWS::ApiGateway::Resource",
            "Properties": {

                "RestApiId": {
                    "Ref": "SendDataToKinesisAPI"
                },
                "ParentId": {
                    "Fn::GetAtt": ["SendDataToKinesisAPI", "RootResourceId"]
                },
                "PathPart": "thingsstream"
```

```
            }
        },
        "ECSAutoScalingGroup": {
            "Metadata": {
                "Description": "Sets up the Auto Scaling Group for the ECS Container cluster"
            },
            "Type": "AWS::AutoScaling::AutoScalingGroup",
            "Properties": {

                "AvailabilityZones": ["us-east-1b"],
                "LaunchConfigurationName": {
                    "Ref": "ECSAutoScalingGroupInstances"
                },
                "MinSize": "1",
                "MaxSize": "5",
                "DesiredCapacity": "1"
            },
            "CreationPolicy": {
                "ResourceSignal": {
                    "Timeout": "PT15M"
                }
            },
            "UpdatePolicy": {
                "AutoScalingRollingUpdate": {
                    "MinInstancesInService": "1",
                    "MaxBatchSize": "1",
                    "PauseTime": "PT15M",
                    "WaitOnResourceSignals": "true"
                }
            }
        },
        "ECSAutoScalingGroupInstances": {
            "Type": "AWS::AutoScaling::LaunchConfiguration",
            "Metadata": {
                "AWS::CloudFormation::Init": {
                    "config": {

                        "commands": {
                            "01_add_instance_to_cluster": {
                                "command": {
                                    "Fn::Join": ["", ["#!/bin/bash\n", "echo ECS_CLUSTER=", {
                                        "Ref": "ECSCluster"
                                    }, " >> /etc/ecs/ecs.config"]]
                                }
                            }
                        },

                        "files": {
                            "/etc/cfn/cfn-hup.conf": {
                                "content": {
                                    "Fn::Join": ["", [
                                        "[main]\n",
                                        "stack=", {
                                            "Ref": "AWS::StackId"
                                        }, "\n",
                                        "region=", {
                                            "Ref": "AWS::Region"
                                        }, "\n"
                                    ]]
                                },
                                "mode": "000400",
                                "owner": "root",
                                "group": "root"
                            },
                            "/etc/cfn/hooks.d/cfn-auto-reloader.conf": {
                                "content": {
                                    "Fn::Join": ["", [
                                        "[cfn-auto-reloader-hook]\n",
                                        "triggers=post.update\n",

"path=Resources.ECSAutoScalingGroupInstances.Metadata.AWS::CloudFormation::Init\n",
                                        "action=/opt/aws/bin/cfn-init -v ",
                                        "         --stack ", {
                                            "Ref": "AWS::StackName"
                                        },
                                        "         --resource ECSAutoScalingGroupInstances ",
                                        "         --region ", {
                                            "Ref": "AWS::Region"
                                        }, "\n",
                                        "runas=root\n"
                                    ]]
                                }
                            }
                        },

                        "services": {
                            "sysvinit": {
                                "cfn-hup": {
                                    "enabled": "true",
                                    "ensureRunning": "true",
                                    "files": ["/etc/cfn/cfn-hup.conf", "/etc/cfn/hooks.d/cfn-auto-reloader.conf"]
                                }
                            }
                        }
```

```
                    }
                }
            },
            "Properties": {
                "ImageId": "ami-55870742",
                "InstanceType": "t2.micro",
                "SecurityGroups": [{
                    "Ref": "NodeSG"
                }],
                "IamInstanceProfile": {
                    "Ref": "EC2InstanceProfile"
                },
                "KeyName": {
                    "Ref": "KeyName"
                },
                "UserData": {
                    "Fn::Base64": {
                        "Fn::Join": ["", [
                            "#!/bin/bash -xe\n",
                            "yum install -y aws-cfn-bootstrap\n",

                            "/opt/aws/bin/cfn-init -v ",
                            "         --stack ", {
                                "Ref": "AWS::StackName"
                            },
                            "         --resource ECSAutoScalingGroupInstances ",
                            "         --region ", {
                                "Ref": "AWS::Region"
                            }, "\n",

                            "/opt/aws/bin/cfn-signal -e $? ",
                            "         --stack ", {
                                "Ref": "AWS::StackName"
                            },
                            "         --resource ECSAutoScalingGroup ",
                            "         --region ", {
                                "Ref": "AWS::Region"
                            }, "\n"
                        ]]
                    }
                }
            }
        },
        "ECSASGInstanceScaleDownPolicy": {
            "Metadata": {
                "Description": "Remove an instance from the Auto Scaling Group"
            },
            "Type": "AWS::AutoScaling::ScalingPolicy",
            "Properties": {

                "AdjustmentType": "ChangeInCapacity",
                "AutoScalingGroupName": {
                    "Ref": "ECSAutoScalingGroup"
                },
                "Cooldown": "300",
                "ScalingAdjustment": "-1"
            }
        },
        "ECSASGInstanceScaleUpPolicy": {
            "Metadata": {
                "Description": "Add an instance from the Auto Scaling Group"
            },
            "Type": "AWS::AutoScaling::ScalingPolicy",
            "Properties": {

                "AdjustmentType": "ChangeInCapacity",
                "AutoScalingGroupName": {
                    "Ref": "ECSAutoScalingGroup"
                },
                "Cooldown": "300",
                "ScalingAdjustment": "1"
            }
        },
        "InsufficientMemoryAlarm": {
            "Metadata": {
                "Description": "Alarm if too much memory reserved by containers"
            },
            "Type": "AWS::CloudWatch::Alarm",
            "Properties": {

                "ActionsEnabled": "True",
                "AlarmActions": [{
                    "Ref": "ECSASGInstanceScaleUpPolicy"
                }],
                "ComparisonOperator": "GreaterThanOrEqualToThreshold",
                "Dimensions": [{
                    "Name": "ClusterName",
                    "Value": {
                        "Ref": "ECSCluster"
                    }
                }],
                "EvaluationPeriods": "1",
                "MetricName": "MemoryReservation",
                "Namespace": "AWS/ECS",
```

```
                "Period": "60",
                "Statistic": "Average",
                "Threshold": "80"
            }
        },
        "TooMuchMemoryAlarm": {
            "Metadata": {
                "Description": "Alarm if too much spare memory"
            },
            "Type": "AWS::CloudWatch::Alarm",
            "Properties": {
                "ActionsEnabled": "True",
                "AlarmActions": [{
                    "Ref": "ECSASGInstanceScaleDownPolicy"
                }],
                "ComparisonOperator": "LessThanOrEqualToThreshold",
                "Dimensions": [{
                    "Name": "ClusterName",
                    "Value": {
                        "Ref": "ECSCluster"
                    }
                }],
                "EvaluationPeriods": "1",
                "MetricName": "MemoryReservation",
                "Namespace": "AWS/ECS",
                "Period": "60",
                "Statistic": "Average",
                "Threshold": "40"
            }
        },
        "ThingsDynamoDBTable": {
            "Metadata": {
                "Description": "Sets up DynamoDB table to store data"
            },
            "Type": "AWS::DynamoDB::Table",
            "Properties": {

                "AttributeDefinitions": [{
                    "AttributeName": "ThingID",
                    "AttributeType": "S"
                }, {
                    "AttributeName": "Timestamp",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "ThingID",
                    "KeyType": "HASH"
                }, {
                    "AttributeName": "Timestamp",
                    "KeyType": "RANGE"
                }],
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": "10",
                    "WriteCapacityUnits": "5"
                },
                "TableName": "ThingsDynamoDBTable"
            }
        },
        "ThresholdsDynamoDBTable": {
            "Metadata": {
                "Description": "Sets up DynamoDB table to store thresholds"
            },
            "Type": "AWS::DynamoDB::Table",
            "Properties": {

                "AttributeDefinitions": [{
                    "AttributeName": "ThingID",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "ThingID",
                    "KeyType": "HASH"
                }],
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": "10",
                    "WriteCapacityUnits": "5"
                },
                "TableName": "ThresholdsDynamoDBTable"
            }
        },
        "NodeSG": {
            "Metadata": {
                "Description": "Sets up Security Group to allow SSH access to EC2 instances"
            },
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {

                "GroupDescription": "Node SecurityGroup",
                "SecurityGroupIngress": [{
                    "IpProtocol": "tcp",
                    "FromPort": "22",
                    "ToPort": "22",
                    "CidrIp": "0.0.0.0/0"
                }, {
```

```
                    "IpProtocol": "tcp",
                    "FromPort": "80",
                    "ToPort": "80",
                    "CidrIp": "0.0.0.0/0"
                }]
            }
        },
        "ECSCluster": {
            "Metadata": {
                "Description": "Cluster of EC2 Instances with containers"
            },
            "Type": "AWS::ECS::Cluster"
        },
        "WebAppContainerService": {
            "Metadata": {
                "Description": "Sets up group of containers"
            },
            "Type": "AWS::ECS::Service",
            "DependsOn": ["ECSAutoScalingGroup", "ECSCluster"],
            "Properties": {

                "Cluster": {
                    "Ref": "ECSCluster"
                },
                "DesiredCount": "1",
                "TaskDefinition": {
                    "Ref": "ContainerTaskDefinition"
                },
                "Role": {
                    "Ref": "ECSServiceRole"
                },
                "LoadBalancers": [{
                    "ContainerName": "analysiswebapp",
                    "ContainerPort": "5000",
                    "LoadBalancerName": {
                        "Ref": "ECSElasticLoadBalancer"
                    }
                }]
            }
        },
        "ContainerTaskDefinition": {
            "Metadata": {
                "Description": "Sets up individual container"
            },
            "Type": "AWS::ECS::TaskDefinition",
            "DependsOn": ["ECSCluster"],
            "Properties": {

                "ContainerDefinitions": [{
                    "Name": "analysiswebapp",
                    "Image": {
                        "Fn::Join": [
                            "", [{
                                    "Ref": "AWS::AccountId"
                                },
                                ".dkr.ecr.us-east-1.amazonaws.com/analysiswebapp:latest"
                            ]
                        ]
                    },
                    "Cpu": "10",
                    "PortMappings": [{
                        "ContainerPort": "5000",
                        "HostPort": "80"
                    }],
                    "Memory": "900",
                    "Essential": "true"
                }]
            }
        },
        "ECSElasticLoadBalancer": {
            "Metadata": {
                "Description": "Sets up load balancer"
            },
            "Type": "AWS::ElasticLoadBalancing::LoadBalancer",
            "Properties": {

                "AvailabilityZones": ["us-east-1b"],
                "Listeners": [{
                    "LoadBalancerPort": "80",
                    "InstancePort": "80",
                    "Protocol": "HTTP"
                }],
                "HealthCheck": {
                    "Target": "HTTP:80/",
                    "HealthyThreshold": "2",
                    "UnhealthyThreshold": "10",
                    "Interval": "30",
                    "Timeout": "5"
                }
            }
        },
        "EC2InstanceProfile": {
            "Metadata": {
                "Description": "Assigns an IAM role to each EC2 instance"
```

```
            },
            "Type": "AWS::IAM::InstanceProfile",
            "Properties": {

                "Path": "/",
                "Roles": [{
                    "Ref": "EC2Role"
                }]
            }
        },
        "APIExecutionRole": {
            "Metadata": {
                "Description": "Role for API to send data to Kinesis"
            },
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Principal": {
                            "Service": ["apigateway.amazonaws.com"]
                        },
                        "Action": ["sts:AssumeRole"]
                    }]
                },
                "Path": "/",
                "Policies": [{
                    "PolicyName": "APIPolicy",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [{
                            "Effect": "Allow",
                            "Action": "kinesis:*",
                            "Resource": {
                                "Fn::GetAtt": ["KinesisStream", "Arn"]
                            }
                        }]
                    }
                }]
            }
        },
        "EC2Role": {
            "Metadata": {
                "Description": "Role for EC2 instance to get access to the container registry and manage
containers"
            },
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [{
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "ec2.amazonaws.com"
                            ]
                        },
                        "Action": [
                            "sts:AssumeRole"
                        ]
                    }]
                },
                "Path": "/",
                "Policies": [{
                    "PolicyName": "ec2-ecs",
                    "PolicyDocument": {
                        "Statement": [{
                            "Effect": "Allow",
                            "Action": [
                                "ecs:CreateCluster",
                                "ecs:DeregisterContainerInstance",
                                "ecs:DiscoverPollEndpoint",
                                "ecs:Poll",
                                "ecs:RegisterContainerInstance",
                                "ecs:StartTelemetrySession",
                                "ecs:Submit*",
                                "ecr:*",
                                "logs:CreateLogStream",
                                "logs:PutLogEvents"
                            ],
                            "Resource": "*"
                        }, {
                            "Effect": "Allow",
                            "Action": "dynamodb:*",
                            "Resource": {
                                "Fn::Join": ["", ["arn:aws:dynamodb:", {
                                    "Ref": "AWS::Region"
                                }, ":", {
                                    "Ref": "AWS::AccountId"
                                }, ":table/", {
                                    "Ref": "ThingsDynamoDBTable"
                                }]]
                            }
                        }]
```

```
                    }
                }]
            }
        },
        "ECSServiceRole": {
            "Metadata": {
                "Description": "Role for ECS cluster to manage EC2 instances"
            },
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [{
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "ecs.amazonaws.com"
                            ]
                        },
                        "Action": [
                            "sts:AssumeRole"
                        ]
                    }]
                },
                "Path": "/",
                "Policies": [{
                    "PolicyName": "ecs-service",
                    "PolicyDocument": {
                        "Statement": [{
                            "Effect": "Allow",
                            "Action": [
                                "elasticloadbalancing:Describe*",
                                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                                "ec2:Describe*",
                                "ec2:AuthorizeSecurityGroupIngress"
                            ],
                            "Resource": "*"
                        }]
                    }
                }]
            }
        },
        "LambdaExecutionRole": {
            "Metadata": {
                "Description": "Role for Lambda functions to get access to the DynamoDB tables, Kinesis stream and
SNS"
            },
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Principal": {
                            "Service": ["lambda.amazonaws.com"]
                        },
                        "Action": ["sts:AssumeRole"]
                    }]
                },
                "Path": "/",
                "Policies": [{
                    "PolicyName": "KinesisDynamoPolicy",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [{
                            "Effect": "Allow",
                            "Action": "kinesis:*",
                            "Resource": {
                                "Fn::GetAtt": ["KinesisStream", "Arn"]
                            }
                        }, {
                            "Effect": "Allow",
                            "Action": "dynamodb:*",
                            "Resource": {
                                "Fn::Join": ["", ["arn:aws:dynamodb:", {
                                    "Ref": "AWS::Region"
                                }, ":", {
                                    "Ref": "AWS::AccountId"
                                }, ":table/", {
                                    "Ref": "ThingsDynamoDBTable"
                                }]]
                            }
                        }, {
                            "Effect": "Allow",
                            "Action": "dynamodb:*",
                            "Resource": {
                                "Fn::Join": ["", ["arn:aws:dynamodb:", {
                                    "Ref": "AWS::Region"
                                }, ":", {
                                    "Ref": "AWS::AccountId"
                                }, ":table/", {
                                    "Ref": "ThresholdsDynamoDBTable"
                                }]]
                            }
```

```
                        }, {
                            "Effect": "Allow",
                            "Action": "logs:*",
                            "Resource": "*"
                        }, {
                            "Effect": "Allow",
                            "Action": "sns:*",
                            "Resource": "*"
                        }]

                    }
                }]
            }
        },
        "KinesisStream": {
            "Metadata": {
                "Description": "Sets up Kinesis stream"
            },
            "Type": "AWS::Kinesis::Stream",
            "Properties": {

                "Name": "ThingsStream",
                "ShardCount": "2"
            }
        },
        "KinesisToDynamoLambdaMapping": {
            "Metadata": {
                "Description": "Links the Kinesis stream to the Lambda function that will move the data to
DynamoDB"
            },
            "Type": "AWS::Lambda::EventSourceMapping",
            "Properties": {

                "EventSourceArn": {
                    "Fn::GetAtt": ["KinesisStream", "Arn"]
                },
                "FunctionName": {
                    "Fn::GetAtt": ["SendThingsToDynamoLambda", "Arn"]
                },
                "StartingPosition": "TRIM_HORIZON"
            }
        },
        "KinesisToSNSLambdaMapping": {
            "Metadata": {
                "Description": "Links the Kinesis stream to the Lambda function that will check the thresholds and
send alerts"
            },
            "Type": "AWS::Lambda::EventSourceMapping",
            "Properties": {

                "EventSourceArn": {
                    "Fn::GetAtt": ["KinesisStream", "Arn"]

                },
                "FunctionName": {
                    "Fn::GetAtt": ["SendAlertsToSNSLambda", "Arn"]
                },
                "StartingPosition": "TRIM_HORIZON"
            }
        },
        "SendAlertsToSNSLambda": {
            "Metadata": {
                "Description": "Function that checks thresholds and sends a message to SNS if they are breached"
            },
            "Type": "AWS::Lambda::Function",
            "Properties": {

                "Handler": "index.lambda_handler",
                "Role": {
                    "Fn::GetAtt": ["LambdaExecutionRole", "Arn"]
                },
                "Code": {
                    "ZipFile": {
                        "Fn::Join": ["", [
                            "from __future__ import print_function\n",
                            "import base64\n",
                            "import boto3\n",
                            "import json\n",
                            "print('Loading function')\n",
                            "def lambda_handler(event, context):\n",
                            "   client = boto3.client('sns')\n",
                            "   dynamodb = boto3.resource('dynamodb')\n",
                            "   table = dynamodb.Table('ThresholdsDynamoDBTable')\n",
                            "   for record in event['Records']:\n",
                            "     payload = json.loads(base64.b64decode(record['kinesis']['data']))\n",
                            "     thingid=payload['id']+'/temperature'\n",
                            "     thresholds = table.get_item(Key={'ThingID':thingid})\n",
                            "     if float(payload['temperature']) > float(thresholds['Item']['MaxThreshold']) or
float(payload['temperature']) < float(thresholds['Item']['MinThreshold']):\n",
                            "       messagetext = 'Temperature alarm for ' + payload['id'] + ': Temperature =
'+payload['temperature']\n",
                            "       response=client.publish(TopicArn ='", {
                            "Ref": "AlarmSNSTopic"
                        },
```

```
                                "', Message = messagetext)\n",
                                "       print(messagetext)"
                            ]]
                        }
                    },
                    "Runtime": "python2.7"
                }
            },
            "SendThingsToDynamoLambda": {
                "Metadata": {
                    "Description": "Function that imports data to DynamoDB"
                },
                "Type": "AWS::Lambda::Function",
                "Properties": {

                    "Handler": "index.lambda_handler",
                    "Role": {
                        "Fn::GetAtt": ["LambdaExecutionRole", "Arn"]
                    },
                    "Code": {
                        "ZipFile": {
                            "Fn::Join": ["", [
                                "from __future__ import print_function\n",
                                "import base64\n",
                                "import boto3\n",
                                "import json\n",
                                "print('Loading function')\n",
                                "def lambda_handler(event, context):\n",
                                "   dynamodb = boto3.resource('dynamodb')\n",
                                "   table = dynamodb.Table('ThingsDynamoDBTable')\n",
                                "   for record in event['Records']:\n",
                                "     payload = json.loads(base64.b64decode(record['kinesis']['data']))\n",
                                "     thingid = payload['id'] + '/temperature'\n",
                                "     table.put_item(Item={'ThingID':thingid, 'Timestamp':payload['timestamp'],
    'Value':payload['temperature']})\n",
                                "   return 'Successfully processed {} records.'.format(len(event['Records']))\n"
                            ]]
                        }
                    },
                    "Runtime": "python2.7"
                }
            },
            "AlarmSNSTopic": {
                "Metadata": {
                    "Description": "SNS Topic that can be subscribed to for email alerts of alarms"
                },
                "Type": "AWS::SNS::Topic",
                "Properties": {

                    "Subscription": [{
                        "Endpoint": {
                            "Ref": "AlarmEmailAddress"
                        },
                        "Protocol": "email"
                    }],
                    "TopicName": "ThresholdBreachAlarm"
                }
            }
        },
        "Outputs": {
            "AnalysisWebAppName": {
                "Description": "ARN of AnalysisWebApp",
                "Value": {
                    "Ref": "WebAppContainerService"
                }
            },
            "ECSClusterName": {
                "Description": "Name of the ECS Cluster",
                "Value": {
                    "Ref": "ECSCluster"
                }
            },
            "AutoScalingGroupName": {
                "Description": "Name of the ECS Cluster",
                "Value": {
                    "Ref": "ECSAutoScalingGroup"
                }
            }

        }
    }
```

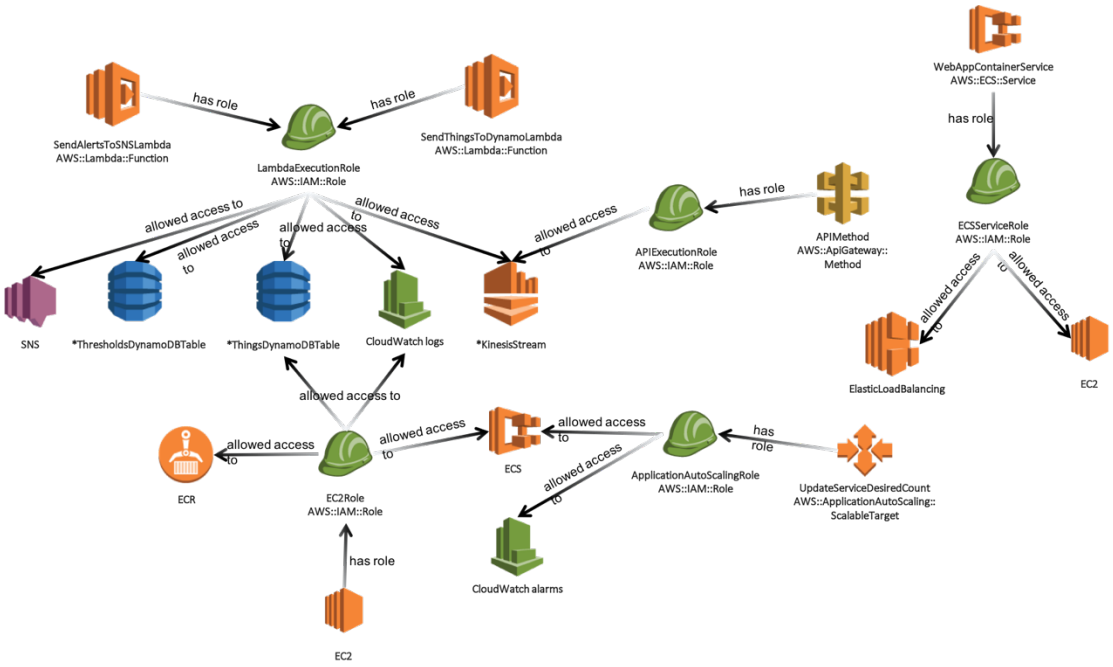
ServiceScaling.json

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Parameters": {
        "AnalysisWebAppName": {
            "Description": "Name of AnalysisWebApp",
            "Type": "String"
        },
        "ECSClusterName": {
            "Description": "Name of the ECS Cluster",
            "Type": "String"
        },
        "AutoScalingGroupName": {
            "Description": "Name of the ECS Cluster",
            "Type": "String"
        }
    },
    "Resources": {
        "ContainerScalingUpPolicy": {
            "Type": "AWS::ApplicationAutoScaling::ScalingPolicy",
            "Properties": {
                "PolicyName": "StepUpPolicy",
                "PolicyType": "StepScaling",
                "ScalingTargetId": {
                    "Ref": "UpdateServiceDesiredCount"
                },
                "StepScalingPolicyConfiguration": {
                    "AdjustmentType": "ChangeInCapacity",
                    "Cooldown": "300",
                    "MetricAggregationType": "Average",
                    "StepAdjustments": [{
                        "MetricIntervalLowerBound": "0",
                        "ScalingAdjustment": "1"
                    }]
                }
            }
        },
        "ContainerScalingDownPolicy": {
            "Type": "AWS::ApplicationAutoScaling::ScalingPolicy",
            "Properties": {
                "PolicyName": "StepDownPolicy",
                "PolicyType": "StepScaling",
                "ScalingTargetId": {
                    "Ref": "UpdateServiceDesiredCount"
                },
                "StepScalingPolicyConfiguration": {
                    "AdjustmentType": "ChangeInCapacity",
                    "Cooldown": "300",
                    "MetricAggregationType": "Average",
                    "StepAdjustments": [{
                        "MetricIntervalUpperBound": "0",
                        "ScalingAdjustment": "-1"
                    }]
                }
            }
        },
        "UpdateServiceDesiredCount": {
            "Type": "AWS::ApplicationAutoScaling::ScalableTarget",
            "Properties": {
                "MaxCapacity": "5",
                "MinCapacity": "1",
                "ResourceId": {
                    "Fn::Join": ["", ["service/", {
                        "Ref": "ECSClusterName"
                    }, "/", {
                        "Ref": "AnalysisWebAppName"
                    }]]
                },
                "RoleARN": {
                    "Fn::GetAtt": ["ApplicationAutoScalingRole", "Arn"]
                },
                "ScalableDimension": "ecs:service:DesiredCount",
                "ServiceNamespace": "ecs"
            }
        },
        "ApplicationAutoScalingRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [{
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "application-autoscaling.amazonaws.com"
                            ]
                        },
                        "Action": [
                            "sts:AssumeRole"
                        ]
                    }]
                },
                "Path": "/",
```

```
                    "Policies": [{
                        "PolicyName": "ecs-service-autoscaling",
                        "PolicyDocument": {
                            "Statement": [{
                                "Effect": "Allow",
                                "Action": [
                                    "ecs:DescribeServices",
                                    "ecs:UpdateService",
                                    "cloudwatch:DescribeAlarms"
                                ],
                                "Resource": "*"
                            }]
                        }
                    }]
                }
            },
            "InsufficientCPUAlarm": {
                "Type": "AWS::CloudWatch::Alarm",
                "DependsOn": "ContainerScalingUpPolicy",
                "Properties": {
                    "ActionsEnabled": "True",
                    "AlarmActions": [{
                        "Ref": "ContainerScalingUpPolicy"
                    }],
                    "ComparisonOperator": "GreaterThanOrEqualToThreshold",
                    "Dimensions": [{
                        "Name": "AutoScalingGroupName",
                        "Value": {
                            "Ref": "AutoScalingGroupName"
                        }
                    }],
                    "EvaluationPeriods": "1",
                    "MetricName": "CPUUtilization",
                    "Namespace": "AWS/EC2",
                    "Period": "60",
                    "Statistic": "Average",
                    "Threshold": "40"
                }
            },
            "TooMuchCPUAlarm": {
                "Type": "AWS::CloudWatch::Alarm",
                "Properties": {
                    "ActionsEnabled": "True",
                    "AlarmActions": [{
                        "Ref": "ContainerScalingDownPolicy"
                    }],
                    "ComparisonOperator": "LessThanOrEqualToThreshold",
                    "Dimensions": [{
                        "Name": "AutoScalingGroupName",
                        "Value": {
                            "Ref": "AutoScalingGroupName"
                        }
                    }],
                    "EvaluationPeriods": "1",
                    "MetricName": "CPUUtilization",
                    "Namespace": "AWS/EC2",
                    "Period": "60",
                    "Statistic": "Average",
                    "Threshold": "10"
                }
            }
        }
```

Appendix 11: Access Control network diagram

This diagram shows how the roles allow access from certain components to certain other components. Those components labelled with an asterisk (*) are specifically named, otherwise access is to the whole service, e.g. access to SNS gives access to any SNS topic owned by the account.



Appendix 12: Feedback from Amey

From: "Baiz, Pedro" <Pedro.Baiz@amey.co.uk>
Subject: Feedback Re: Alison PG-Wells
Date: 19 September 2016 at 09:59:35 BST
To: "PG-Wells, Alison" <Alison.Wells.2@city.ac.uk>
Cc: "Weyde, Tillman" <T.E.Weyde@city.ac.uk>, "Bedeman, Tom" <Tom.Bedeman@amey.co.uk>

Dear Alison,

I am writing to you to express my sincere gratitude for your work at Amey Strategic Consulting (SC) during your MSc Project on “Using an Infrastructure as Code approach to a cloud-based Internet of Things Big Data Analysis platform”.

I would like to particularly highlight that:

· Your work is one of the first of its kind. As your work revealed, the relatively new concept of “Infrastructure as Code” combined with very new AWS services (some of which were released during your project, summer 2016) are making possible to develop scalable and complex cloud data solutions with a fraction of the effort that was required just 1 year ago (e.g. Internet of Things). This new concept is taking by surprise even top leading universities which now need to consider adjusting program syllabus in order to train professional for this new industry standard.

· Your independence and proactivity. Although the work was proposed by Amey SC and you had limited experience on the topics covered, you were able to understand its goal and objectives in order to take complete ownership of the project, demonstrating not just independent thinking but also proactivity in order to deliver the first building stone of future cloud platforms for Amey Strategic Consulting.

· Your team work attitude. Amey is an organisation that values people and aims to build work environments in which each of its members can fulfil its full potential. You were able to demonstrate during your stay how to work as a team member in order to deliver good work within a tight deadline, always engaging all stake holders in order to completely fulfil their individual requirements: namely Amey SC and City University.

Thank you very much and all the best in all your future endeavours,

Best Regards,

Pedro Baiz, PhD
Senior Consultant | Strategic Consulting
Amey

m: +44 7916253021 e: pedro.baiz@amey.co.uk
3rd Floor Chancery Exchange | 10 Furnival Street | London | England | EC4A 1AB