

# Healthcare Data Lake

KENDAL, JOSEPH  
University of Bristol  
jk17246@bristol.ac.uk

SHERRED, JAGO  
University of Bristol  
j.sherred.2019@bristol.ac.uk

BENSON, LUKE  
University of Bristol  
wr19606@bristol.ac.uk

LIU, ANNA  
University of Bristol  
gf19916@bristol.ac.uk

CISMARU, ARMAND  
University of Bristol  
fz19792@bristol.ac.uk

November 19, 2020

## Abstract

Digital healthcare provided by the NHS in England typically operates in silos. GPs have electronic systems to manage patient care which are distinct from hospital systems which are distinct from the ambulance service, 111, mental health services etc. Each data owner has a wealth of data that, if combined, would generate a more valuable resource than it does in isolation. While there are solutions to integrate this data for direct care purposes, there is no centralised solution to use this data to inform future care or service provisioning. This project is designed to explore the benefits of cloud technologies to produce a prototype secure, scalable health data storage platform that can underpin local healthcare analytics.

## 1 Overview

### 1.1 Client

Dr. Philip D Harfield, Health Data Scientist (Informatics) at NIHR Bristol Biomedical Research Centre, University of Bristol.

### 1.2 Domain

The domain of our Healthcare Data Lake project is the Healthcare Analytics Environment team who will design, implement and test a set of candidate cloud-hosted analytics environments that provide sufficient functionality while also maintaining a secure data environment.

The overall domain of the three teams is NHS Healthier Together Sustainability Transformation Partnership Bristol, North Somerset, and South Gloucestershire (BNSSG) & Bristol Biomedical Research Centre, University of Bristol Medical School.

### 1.3 Project

The project entails combining a wealth of data from data owners(GPs Patient data, ambulance services, 111, mental health services .etc) into a data lake. This will be used to inform clinical

decisions making by providing more advanced insights into the longitudinal health of the patient on arrival and understand the merits of previous clinic decisions taken.

The project will explore the benefits of cloud technologies and produce a prototype secure, scalable health data storage platforms that can underpin local healthcare analytics.

The project is one of three designed an end-to-end proof of concept to the local NHS. We will be working alongside the "Healthcare Data Visualisation" and "Healthcare Analytics Environment" teams.

## **1.4 Vision**

The Healthcare Data Lake Project is envisioning a future integrated data storage solution, one that is scalable and portable, while using the latest cloud-based technologies. Starting with a prototype, the final scope is to create, alongside the Simulation and Analytics Projects, a system that is going to change how data is handled and used in the healthcare system. There is the real possibility that this three projects will represent the cornerstone of a future solution used and developed extensively by the NHS, which will bring immediate aid to the average medical worker and improve the quality of the service provided to the patients.

## 2 Stakeholder and Requirements Analysis

### 2.1 Primary Stakeholder and User Story

Philip Harfield at Bristol, North Somerset and South Gloucestershire CCG (BNSSG). Philip Harfield is our client, and this software is being developed for him at BNSSG. The user story for this stakeholder is that BNSSG require a piece of software that can be used to allow long-term healthcare data analytics from multiple data sources to inform clinical and strategic decisions. The reason for this is understanding the longitudinal health of a patient allows understanding of the merits of previous clinical decisions taken. In addition it can be used to inform strategic commissioning decisions using data on how effective services offered to patients have been.

#### 2.1.1 Flow steps

Considering the user story above, we can breakdown the story into a sequence of steps of user flow.

1. Local healthcare service authenticates their identity.
2. Local healthcare service provides information to the data lake.
3. Data is transformed and loaded into the data lake.
4. The data is catalogued in the data lake and stored in structured data marts as required.
5. The data lake allows access to a data analytics environment.
6. An analytics environment can run queries on the data lake and report results to analytics environment.
7. Clinical and strategic decisions can be taken based on analysis.

We can also identify alternative or exceptional flows.

#### Exceptional Flow:

1. Local healthcare service is authenticated.
2. Local healthcare services provide data to the data lake.
3. Data is not provided in an acceptable format to the API.
4. The healthcare service receives an error message to provide data in standard format.

#### Exceptional Flow:

1. Local healthcare service fails to be authenticated.
2. The service receives an error message to provide valid credentials.
3. Incoming data is not accepted.

#### 2.1.2 Atomic Requirements

We can breakdown these steps into atomic requirements of the software.

1. An identity authentication process provides healthcare providers with credentials to supply data to the API.
2. An API takes in data from local healthcare providers as a HL7 FHIR message. We have chosen HL7 FHIR as it is the UK standard for transferring healthcare messages.
3. These data messages are stored in a cloud solution in a well structured data model.
4. Ingested data is catalogued.
5. An ETL tool is used to curate data marts.

6. These data marts can be queried by the analytics environment.

There are also additional requirements specified for the software:

1. Medical data is to be stored independently from pseudonymised patient identifiers.
2. Provide a user console to monitor automated ETL jobs.
3. Provide full audit trails.

## 2.2 Additional stakeholders and User Stories

This software will provide services to a number of local healthcare organisations such as NHS trusts and the Healthier Together STP and such all these additional users are secondary stakeholders to this project. These organisations will need to be able to provide healthcare information to the software which will need to be able to load and store the data for future analysis.

### 2.2.1 Flow Steps

Considering the user story above, we can breakdown the story into a sequence of steps of user flow.

1. Healthcare provider (e.g. Hospital Trust) provide data to the data lake by a HL7 FHIR message.
2. The data is stored in the data lake.

We can also consider the stakeholder of a developer and their user story of accessing a user control to monitor automated ETL jobs to identify live support issues.

**Flow Steps** This can be broken into the following user flow.

1. Developer authenticates identity.
2. Developer accesses user interface which logs all automated ETL jobs carried out on the data lake.
3. Developer can use information to identify issues and provide support.

## 3 Personal Data, Privacy, Security and Ethics Management

### 3.1 GDPR

The data lake solution developed by the “Healthcare Data Lake” project team is an integrating part of the larger prototype system, alongside “Healthcare Data Visualisation” and “Healthcare Analytics Environment” teams (*section 1.3*). The proposed system is going to be used in compliance with the *NHS Digital GDPR* compliance implementation and the liability for the personal data stored falls onto the respective primary stakeholder, Philip Harfield at Bristol, North Somerset and South Gloucestershire CCG (BNSSG) (*section 2.1*). The team developing the data lake infrastructure is responsible for creating a prototype secure, robust and scalable health data storage platform used for ingesting and interrogating patient’s data under the HL7 FHIR standard (*section 4.1*). As a client-specified requirement, medical data is going to be stored independently from the pseudonymised patient identifiers, thus protecting the identity and integrity of the patients whose data is stored in the data lake. The security of the data ingested as described in *section 3.2* is aligned with the *NHS Digital GDPR* compliance and practice.

## **3.2 Security**

### **3.2.1 VPC**

### **3.2.2 VPN**

### **3.2.3 Encryption**

### **3.2.4 IAM**

### **3.2.5 Logging**

### **3.2.6 Auditing**

## **3.3 Ethics**

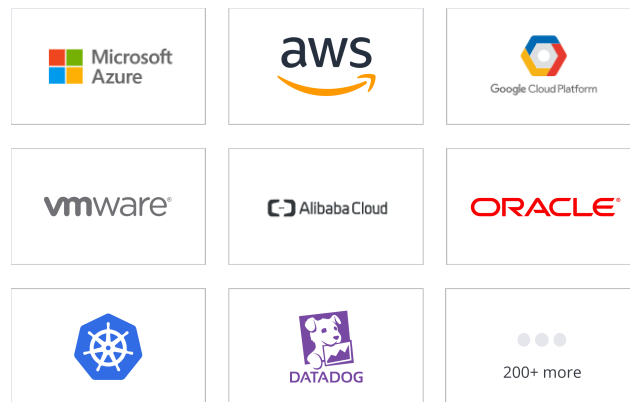
## 4 Architecture

### 4.1 Introduction

We propose a modular, (cloud)platform-independent solution that offers high scalability and performance at a low cost for maintenance, development and deployment. The key to achieving this is leveraging the practises of infrastructure-as-code (IaC), serverless architectures and open standards. Therefore, development expense is focused on providing the most value, security and flexibility to the user.



**IaC** By building infrastructure through extensible configuration files, we make it easy to build, test, secure, update and rollback versions of architecture which can combine multiple cloud or on-premise services. Terraform is an IaC framework that supports all major cloud providers in addition to self-hosted options such as Kubernetes. This makes it a popular choice for a modern infrastructure team that seeks to avoid vendor lock-in and easily protect the security of it with robust tests and auditing.



**Serverless** Serverless computing is a cloud computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity [1]. The benefits of this approach are the huge reduction in infrastructure and development expense. Engineers can focus on shipping their microservices and have secure, scalable infrastructure taken care of for them.

The Serverless Framework is also an example of IaC and provides developers with the ability to develop and deploy their serverless application to different cloud providers or simply Kubernetes (using **Knative**) if they wanted to run it on-premise or avoid code exposure to a cloud native service such as AWS Lambda or Google Cloud Functions.

This also provides quicker developer on-boarding and flexibility as a broad array of popular languages are supported. Given that the Function as a Service (**FaaS**) model is centred around the

principles of loosely coupled microservices, this enables easy migration out of serverless architecture to a VM or container-based deployment. This may make sense in scenarios where there are cost savings to be made which typically applies to long-running, memory intensive tasks as opposed to short-lived and resource-light stateless services.

**Open standards** Open Standards allow people to share all kinds of data freely and with perfect fidelity. They prevent lock-in and other artificial barriers to interoperability, and promote choice between vendors and technology solutions. [2]

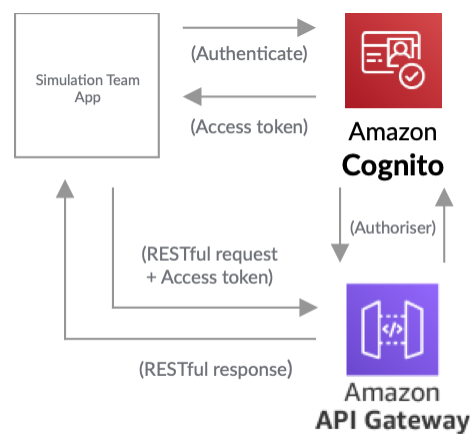
**OpenAPI** The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases. [3]

**HL7 FHIR** *Fast Healthcare Interoperability Resources* (FHIR, pronounced "fire") is a standard describing data formats and elements (known as "resources") and an [API] for exchanging electronic health records (EHR). The standard was created by the Health Level Seven International (HL7) health-care standards organization. One of its goals is to facilitate interoperation between legacy health care systems, to make it easy to provide health care information to health care providers and individuals on a wide variety of devices from computers to tablets to cell phones, and to allow third-party application developers to provide medical applications which can be easily integrated into existing systems. [4]

## 4.2 Data ingestion

### 4.2.1 API management

#### Authentication

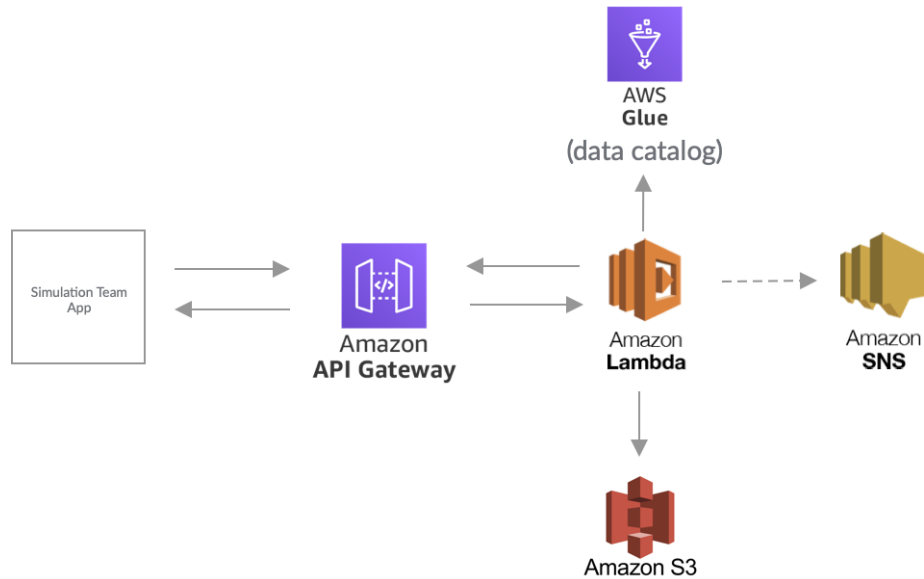


Example: AWS API Gateway

#### Gateway

#### 4.2.2 Microservices

The API Gateway will forward requests to a Lambda that stores the data in S3, may send a notification using SNS and write metadata to the Glue catalogue. The SNS notification may trigger another microservice that prepares an ETL job.



Example: AWS Lambda

#### 4.2.3 Metadata cataloguing

Glue

#### 4.2.4 Schema-less object store

S3

### 4.3 Data warehousing

#### 4.3.1 Federated querying

#### 4.3.2 Common models

### 4.4 ETL & data marts

#### 4.4.1 Scheduling

#### 4.4.2 Developers

#### 4.4.3 Console

### 4.5 Secure access

#### 4.5.1 Encryption

When data is received, it will require a key to encrypt the data. This key is typically generated by a key manager. When data is being retrieved or queried it will then use the same key to decrypt



the data to be used.

A key manager is used to generate, exchange, use, replace and delete cryptographic keys. It's able to deal with asymmetric and symmetric keys. Some key managers encrypt keys after generation to add another layer of security.

Client-side encryption is an alternative option to encrypt the data locally in the application before it gets to the lake. This means the lake is not involved in the cryptographic process of the data, as the application will use its own encryption key which the lake cannot access. This is beneficial for the users as it provides them peace of mind.

**Solutions** Cloud computing services *Azure*, *GCP* and *AWS* all provide options for client and server-side encryption.

We will be looking at AWS Simple Storage Service (S3) [5], AWS Redshift [6] and AWS Key Management Service (KMS) for an example of the implementation of encryption.

**AWS S3** Server-side encryption uses three different modes of server-side encryption: SSE-S3, SSE-C, or SSE-KMS. The following will be an example of SSE-KMS.

SSE-KMS enables the choice of a customer-managed Customer Master Key (CMK) or the AWS managed CMK for Amazon S3 in the account. AWS KMS and Amazon S3 perform the following actions on encryption:

- Amazon S3 requests a plaintext data key and a copy of the key encrypted under the specified CMK.
- AWS KMS generates a data key, encrypts it under the CMK and sends both the plaintext data key and the encrypted data key to Amazon S3.
- Amazon S3 encrypts the data using the data key and removes the plaintext key from memory as soon as possible after use.
- Amazon S3 stores the encrypted data key as metadata with the encrypted data.

Amazon S3 and AWS KMS perform the following actions when data is requested to be decrypted:

- Amazon S3 sends the encrypted data key to AWS KMS.
- AWS KMS decrypts the key by using the same CMK and returns the plaintext data key to Amazon S3.
- Amazon S3 decrypts the ciphertext and removes the plaintext data key from memory as soon as possible.

Client-Side Encryption with AWS can use Amazon S3 Encryption Client in the AWS SDK of an application to encrypt objects and upload them to Amazon S3. This method ensures its security as it passes to the Amazon S3 service. The Amazon S3 service receives the encrypted data; it does not play a role in encrypting or decrypting it.

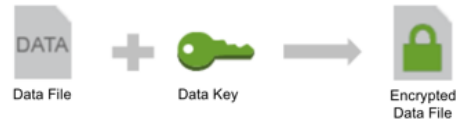
**AWS Redshift** Amazon Redshift uses a four-tier, key-based architecture for encryption. The architecture consists of data encryption keys, a database key, a cluster key and a master key.

1. Data encryption keys encrypt data blocks in the cluster. Each data block is assigned a randomly generated 256-bit AES key. These keys are encrypted by using the database key for the cluster.
2. The database key encrypts data encryption keys in the cluster. The database key is a randomly generated AES-256 key. It is stored on disk in a separate network from the Redshift cluster and passed to the cluster across a secure channel.
3. The cluster key encrypts the database key for the Redshift cluster. KMS, AWS CloudHSM, or an external hardware security module (HSM) can be used to manage the cluster key.
4. The master key encrypts the cluster key. KMS CMK can be used as the master key for Redshift.

**AWS KMS** AWS Key Management Service (KMS) allows you to create and manage cryptographic keys and control their use across a wide range of AWS services and in the applications. AWS KMS uses hardware security modules that have been validated under FIPS 140-2, or are in the process of being validated, to protect keys. AWS KMS is integrated with AWS CloudTrail to provide logs of all key usage to help meet applicable regulatory and compliance needs. [7]

What is going on when S3 handles the process of server-side encryption [8]

- When an object is uploaded into S3 and specified SSE, a unique 256-bit key (a data key) is generated and the data is encrypted using that key with AES-256.



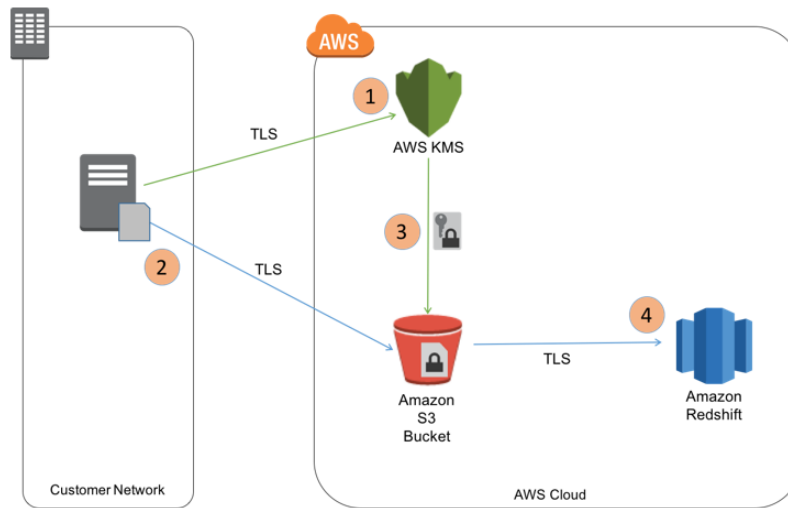
- The data key is then encrypted using a master key, which S3 retrieves from KMS using the master key ID that was supplied in the `s3 cp` command.



- The encrypted data key is then stored with the encrypted data in the S3 bucket.



**Overview** The major steps in this process are



1. You create a master key in KMS
2. You load the data into S3.
3. S3 retrieves the key from KMS and encrypts the data.
4. You run the COPY (`cp`) command in Redshift to load the data from S3.

#### 4.5.2 IAM

**Concept** Identity and Access management (IAM) is used to make sure authorised users have the right to services while preventing access to non-authorised users (guaranteeing secure access). It does this by defining and managing the roles, the access privileges of individual users and the circumstance users are granted those privileges.

IAM allows access management over computing, storage, database and application services. It gives one identity per individual. Once that identity has been introduced, it must be maintained, modified and monitored throughout each user's access lifecycle. [9]

**Solutions** Each cloud service provider offers IAM. Independent IAM includes Okta Identity Cloud, JumpCloud and Auth0 which are all compatible with a range of cloud service providers.

**Examples** We will be examining AWS IAM [10]. IAM is a feature of an AWS account offered at no additional charge.

The use cases of AWS IAM include finite-grained access control to AWS resources, multi-factor authentication for high privilege users, analysis of access across the AWS environment and ability to integrate with the corporate directly (integration of current IAM system).

**How it works** AWS IAM allows us to:

- Manage IAM users and their access – Create users in IAM, assign them individual security credentials (access keys, passwords and multi-factor authentication devices), or request temporary security credentials to provide users access to AWS services and resources. Manage permissions in order to control which operations a user can perform.
- Manage IAM roles and their permissions – Create roles in IAM and manage permissions to control which operations can be performed by the entity, or AWS service, that assumes the role. Define which entity can assume the role. In addition, you can use service-linked roles to delegate permissions to AWS services that create and manage AWS resources automatically.
- Manage federated users and their permissions – Enable identity federation to allow existing identities (users, groups and roles) in the enterprise to access the AWS Management Console, call AWS APIs and access resources, without the need to create an IAM user for each identity. Enable the use of any identity management solution that supports SAML 2.0.

#### 4.5.3 Logging

**Concept** A Log is data that can describe application information, system performance and user activity, for example, might include Timestamp, Application, User, Session ID etc.

Using logs to monitor the use of the system provides security to check for violations and unpredicted behaviour from users and the system. Analysing, searching and filtering logs discovers and alerts these incidents. It enables access to historic information for debugging and forensic investigations. [11]

**Examples** The example which will show the use of logs is AWS CloudWatch. CloudWatch can be used to monitor, store and access the log files from AWS Services.

CloudWatch enables the centralization of the logs from all over the systems, applications and AWS services that are used, in a single, highly scalable service. You can then easily view them, search them for specific error codes or patterns, filter them based on specific fields, or archive them securely for future analysis. [12]

#### Features

- Querying – CloudWatch Insights can be used to interactively search and analyse log data. CloudWatch provides sample queries, command descriptions, query autocompletion and log field discovery to help get started.
- Monitor AWS CloudTrail Events – You can create alarms in CloudWatch and receive notifications of particular API activity as captured by CloudTrail and use the notification to perform troubleshooting.
- Log Retention – By default, logs are kept indefinitely and never expire. You can adjust the retention policy for each log group, keeping the indefinite retention, or choosing a retention period between 10 years and one day.
- Archive Log Data – CloudWatch Logs can be used to store log data in highly durable storage. The CloudWatch Logs agent makes it easy to quickly send both rotated and non-rotated log data off of a host and into the log service. You can then access the raw log data when needed.

## 5 Development Testing

### 5.1 Local environment

#### 5.1.1 API

**Serverless** The Serverless Framework enables developers to run an emulated environment locally instead of needing to deploy that serverless app for testing. This comes with over 1,000 plugins that can emulate services such as AWS Lambda and API Gateway.

Therefore, developers are expected to build and test their code locally with identical behaviour to that of the deployment environment.

**Postman** Postman is a very popular tool used by developers that want to conveniently test and build APIs in a collaborative way.

As we are implementing the OpenAPI specification, Postman enables the import of Swagger files to automatically generate a Postman environment. This makes developer on-boarding much easier as versioned mock endpoints and documentation are hosted for us as part of the DevOps deployment.

#### 5.1.2 Infrastructure

The shared infrastructure differs from that of the application infrastructure (in this case referring to the serverless API). The shared infrastructure can be built, updated, reverted and destroyed easily using the Terraform command-line tools (`cli`).

**Docker** To achieve consistency across developers' local machines, a Docker container will be used to run any Terraform commands.

**Terraform** Terraform workspaces are utilised to isolate development, staging and production environments. It also gives developers the option to work on their own accounts without breaking global state. However, a global state backend will be configured with S3 and a state lock with DynamoDB to enable a team to work on the same infrastructure.

### 5.2 DevOps

#### 5.2.1 GitHub

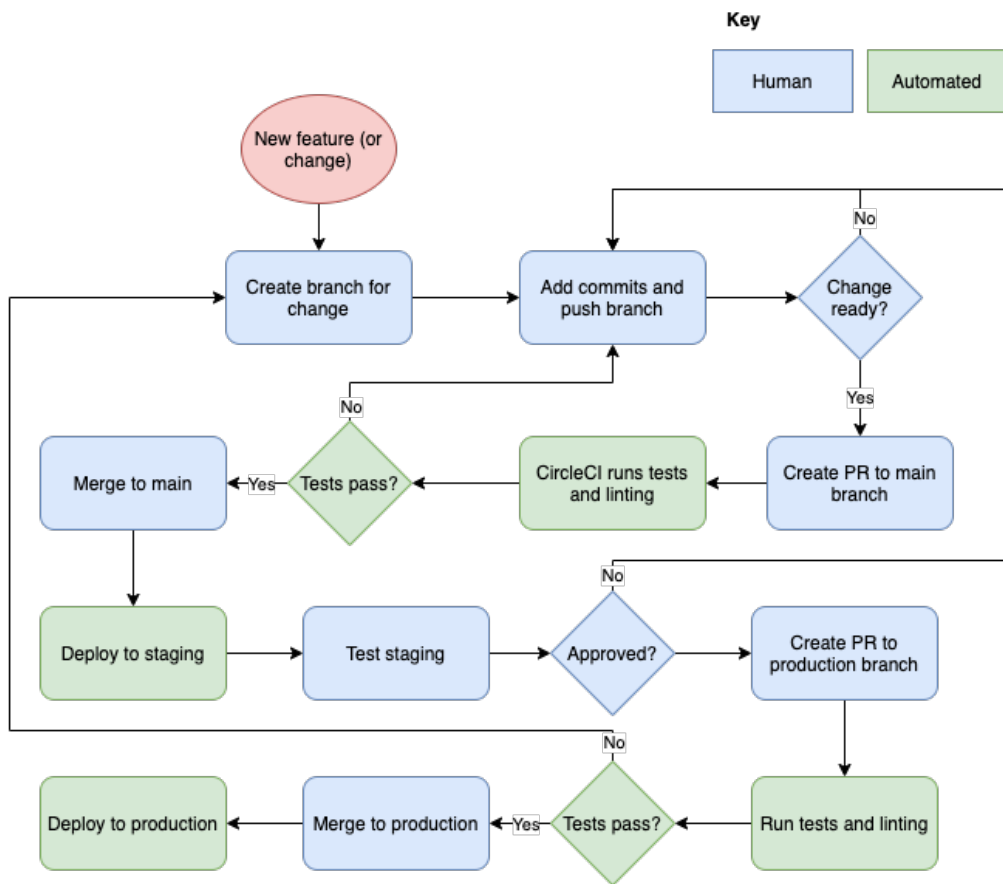
The git repository is hosted on GitHub in the custody of the organisation. However, the repository will be MIT licensed.

#### 5.2.2 CircleCI

The Continuous Integration & Continuous Delivery are run by CircleCI servers. The workflows of jobs are defined in a YAML configuration file.

#### 5.2.3 Terraform

Terraform handles the deployment by using the relevant APIs - in this case it will use the AWS API but may use any other providers as required.



## 6 Release Testing

### 6.1 Staging

### 6.2 Production

## References

- [1] Ron Miller. *AWS Lambda Makes Serverless Applications A Reality*. TechCrunch. 24. URL: <https://techcrunch.com/2015/11/24/aws-lambda-makes-serverless-applications-a-reality/>.
- [2] *Overview of Open Standards*. Free Software Foundation Europe. URL: <https://fsfe.org/freesoftware/standards/def.en.html>.
- [3] *OpenAPI Specification*. Version 3.0.3. SmartBear Software. URL: <https://swagger.io/specification/>.
- [4] *Fast Healthcare Interoperability Resources*. Wikipedia Foundation. Nov. 2016. URL: [https://en.wikipedia.org/wiki/Fast\\_Healthcare\\_Interoperability\\_Resources](https://en.wikipedia.org/wiki/Fast_Healthcare_Interoperability_Resources).
- [5] *How Amazon Simple Storage Service (Amazon S3) uses AWS KMS*. Amazon Web Services. URL: <https://docs.aws.amazon.com/kms/latest/developerguide/services-s3.html#s3-encryption-context>.
- [6] *How Amazon Redshift uses AWS KMS*. Amazon Web Services. URL: <https://docs.aws.amazon.com/kms/latest/developerguide/services-redshift.html>.
- [7] *AWS Key Management Service (KMS)*. Amazon Web Services. URL: <https://aws.amazon.com/kms/>.
- [8] Russell Nash. *Encrypt Your Amazon Redshift Loads with Amazon S3 and AWS KMS*. Amazon Web Services. Apr. 2016. URL: <https://aws.amazon.com/blogs/big-data/encrypt-your-amazon-redshift-loads-with-amazon-s3-and-aws-kms/>.
- [9] James A. Martin and John K. Waters. *What is IAM? Identity and access management explained*. IDG Communications. Oct. 2018. URL: <https://www.csoonline.com/article/2120384/what-is-iam-identity-and-access-management-explained.html>.
- [10] *AWS Identity and Access Management (IAM)*. Amazon Web Services. URL: <https://aws.amazon.com/iam/>.
- [11] Raffael Marty. *Cloud Application Logging for Forensics*. URL: <https://pixlcloud.com/applicationlogging.pdf>.
- [12] *What Is Amazon CloudWatch Logs?* Amazon Web Services. URL: <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html>.