Backup Restoration Test System

Final Report (semester 2)

Rob Shelly

20068406

Supervisor: Dr. Rosanne Birney

BSc (Hons) in Applied Computing

Contents

1 Introduction

1.1 Problem Statement

In January 2017 GitLab suffered a data loss incident which was widely reported in the media. It began with spammers targeting GitLab.com and culminated in an engineer erroneously deleting 300GB of PostgreSQL data in a production environment. The lost data included merger requests, users and comments (GitLab, 2017a). The bigger story was to come later, however, when it was realised that GitLab's backup process had failed silently. The backups did not exist, thus resulting in a total loss of the data. As it transpired, conflicting major versions of *pg_dump* (a utility for backing up PostgreSQL databases) in use for the backup procedure and the PostgreSQL database resulted in an error and the procedure failing (GitLab, 2017b).

The incident was widely reported in the tech industry with the story being picked up by a number of outlets including TechCrunch (2017) and The Register (2017). For many the focal point of the story was the failed backups. The incident highlighted the need for regular verification of backups. A simple way of performing this verification is to regularly restore data. The method of verification is to perform a restore of the data, which can be a mundane and time consuming task. The aim of this project is to create a solution to the issue. A system which can notify administrators when backups have failed may have prevented the data loss in the GitLab ordeal.

1.2 Aims & Objectives

The overall objective is to create a system which tests that uncorrupted backups exist and contain valid, readable data. A system will be created that allows sysadmins to test backups and to schedule the regular testing of backups. This will be achieved by performing restoration on the backups.

The product owner of this project is Red Hat which has an interest in this area, having closely followed the incident detailed above. The main objectives of the system in accordance with the desires of Red Hat are as follows:

AO1 Eliminate the mundane and time consuming task of backup testing by automating regular backup restorations and recording the results;

AO2 Catch silent failures of the backup procedure by notifying sysadmins of failed backups;

AO3 Reduce the cost of backup restoration testing by automating the process of creating the necessary infrastructure (such as virtual machines on AWS), performing the restoration and destroying the infrastructure when results are obtained, thus minimising the uptime of infrastructure;

AO4 Perform the restoration check in a secure manner by managing encryption keys and the movement and decryption of data only when necessary in a safe environment.

The system will focus on backups of databases. For scope, design will focus on testing MongoDB data and MySQL data, thereby providing a sample of both relational and non-relational database management systems (DBMS). However, the system should be designed such that it can be easily modified to test data from others forms of database management systems. As part of the system, the following should be implemented:

- **Web app:** This will act as a front end for the sysadmins to run and schedule tests and to view results.
- **Automation Server:** This will be the backend of the system. It will take care of retrieving the backup data before performing some sorts of tests.
- **Container Platform:** This will be utilised by the backend to test the server. For example, when testing the data from a MongoDB database, the backend will spin up a container with MongoDB installed in order to verify the data.

2 Semester One Summary

Semester one consisted of the research phase of this project. As part of the research phase, the technical feasibility of the project was assessed by first defining the following three *Proof of Concepts*:

POC1 Validation: How will a restoration be validated? What criteria are needed for a backup to be deemed successful?

POC2 User Rules: How will a user use the system? Can the implementation of a backup restoration be abstracted from the user by means of a user friendly frontend?

POC3 Security: How will the system deal with encrypted backups? Can the Jenkins server safely handle credentials and decrypt backups in order to perform the test restoration?

2.1 POC1 Validation

This POC was tested by creating a Jenkins job which restores a backup file to the DBMS and performs a full read of the database.

2.1.1 Work Completed

In order to prove that validation can be achieved via Jenkins, the following infrastructure was deployed to AWS:

- **Backup Server:** An EC2 instance on which a backup file was saved.
- **Restoration Server:** An EC2 instance with the MongoDB DBMS installed and running.
- **Jenkins Server.**

A Jenkins job was then configured to execute the following tasks:

1. Copy backup file from *backup* server to *restoration* server (sample MongoDB data used (MongoDB, 2017));

2. Import the backup file into MongoDB on *restoration* server;
3. Run a *findAll* command on the MongoDB database to retrieve all entries;
4. Print the entries to Jenkins console;

2.1.2 Results

The POC showed backups can be successfully transported to and imported into a remote server. Once imported, a successful *findAll* command proves that the data is still readable. The POC also demonstrated that the SSH credentials for the various servers could be stored in Jenkins and used securely into order to connect to them.

2.2 POC2 User Rules

This POC was tested by creating a basic frontend with React which uses Jenkins API to trigger an arbitrary Jenkins Job.

2.2.1 Work Completed

The following tools were utilised to create a web interface for Jenkins.

- A basic web app was created with ReactJS, consisting of a simple form to input parameters;
- A parameterised Jenkins job was;
- The NodeJS wrapper for the Jenkins API, which allows Jenkins API calls to be made with NodeJS.

When the web form was completed and submitted, the web app triggered the parameterised job (including the parameters input to the form) via the API.

2.2.2 Results

The POC demonstrated that implementation of Jenkins jobs can be abstracted from the user therefore allowing the system to present the user with a simple form to complete in order to commence an automated restoration.

2.3 POC3 Security

This POC was tested using the same AWS infrastructure as POC1. In this instance, the backup file had been encrypted with a GPG.

2.3.1 Work Completed

In this instance the Jenkins job copied the encrypted backup to the *restoration* server. The backup was then decrypted using GPG and a private key on the *restoration* server and the contents printed in order to verify decryption.

2.3.2 Results

This POC demonstrated that the system, in its current architecture, can successfully copy an encrypted backup file to the *restoration* server where it is decrypted. It also demonstrated that the GPG private keys can be securely stored and used by Jenkins.

2.4 Prototype

As a final task, the three POCs were integrated for the purpose of creating a prototype. The prototype consisted of a single form on a web app which allowed the user to input the IP address or DNS and filename of a encrypted backup file. When the form was submitted, it triggered a Jenkins job which moved the file to a restoration server where it decrypted the file. It then imported the file into a DBMS and performed a full read of the database, thereby proving that the backup file was valid.

3 Technologies

3.1 Docker

Docker is a container platform which allows the building and managing applications. This project will implement the final product as Docker images that run on the platform. A container image is a modular piece of software. It encapsulates all the code and tools needed to run the software packaged in the image. The image can then be run in a container on any environment using a container platform or service. Thus, it runs independent of the hardware or operating system. The container also isolates the software from other images and software running within the environment (Docker, 2017).

The ability to build applications as OS agnostic images makes it appealing for this project. It allows the frontend application to be built as an image and run on any system running Docker, for example an AWS EC2 instance.

3.2 Amazon Web Services

The project will make extensive use of Amazon Web Services (AWS) with most or possibly all of the system's infrastructure deployed on AWS, particularly using EC2 and ECS.

EC2 is Amazon's compute service. It allows easy deployment and management of virtual compute resources within the cloud. The flexibility of operating systems, virtual machines (or instances as they are known in AWS) and size of volume of storage make it ideal for this project (Amazon, 2017a). It will allow the system to create instances with only the necessary resources required (i.e. memory and storage) to test the restoration of a given backup. This keeps the cost of testing to a minimum in keeping with Aim **AO3**.

ECS is Amazon's container management service. It allows Docker images to be easily deployed to and run on EC2 instances, taking care of Docker installation and management, for example managing port mappings between containers and the host. There is no added cost for using ECS. i.e. the customer only pays for the EC2 instances (Amazon, 2017b).

AWS also features a command line interface for building, modifying and destroying infrastructure across all of its services, providing a programmatic method of creating the resources. The ability to do so allows for the automation of the infrastructure creation and destruction. This makes AWS an ideal platform for this project as automation is an objective of the project set out in Aim **AO1**.

3.3 Jenkins

Jenkins is an automated build server used to implement continuous integration (CI) and continuous delivery (CD). Configuration and management of the server can be achieved using both a web interface and an API. Jenkins is also extensible through a library of plugins (Jenkins, 2017).

Jenkins was chosen as a backend service for this project as it, along with it's library of plugins, presents many useful features which will be beneficial to the implementation of the system:

- A built in email notification system which can be used to notify users of silently failed backups. This provided the functionality to implement a satisfactory solution to Aim **AO2**;
- A Credentials plugin which provides a means of storing various credentials in various forms (e.g. username/password pairs, SSH keys) along with a standard API for Jenkins and other plugins to access and use these credentials (Connolly, 2017). This provides a secure manner for using SSH keys for backup servers. This is a key objective of the project outlined in Aim **AO4**.
- The ability to schedule jobs to run at regular intervals will provide the functionality described in Aim **AO1**. This eliminates the need for the development of a scheduling system.
- The REST API can be used by a user-friendly web based frontend, allowing users unfamiliar with Jenkins or AWS to perform test restorations.

3.4 Node

The frontend of the system will be designed using Node (also known as Node.js). Node is a JavaScript runtime environment for building network applications. It is lightweight and efficient framework through its event driven, no blocking I/O implementation.

The default package manager of Node is *npm* (for Node Package manager). It is the worlds largest software registry (NPM, 2017). The vast registry of free and open source packages available make Node an attractive choice for this project. Of particular interest are the multiple Node clients for Jenkins. These are Node wrappers for the Jenkins REST API enabling easy integration of the frontend with the Jenkins backend.

Although any of a number of frameworks could have been used, for example Django, Node was chosen for this project due its lightweight design and extensibility through *npm*, including the aforementioned Jenkins API wrappers.

3.5 React

The UI element of the frontend will be built using React, a JavaScript library available through *npm* for building user interfaces. React is developed to work independently of other technologies, meaning it can be integrated easily with Node and other *npm* packages without the need for refactoring. React builds UIs as a set of components, each managing their own state and implementing their own render function. This allows fast and efficient rendering of data changes as only components that are updated will be re-rendered.

4 Design

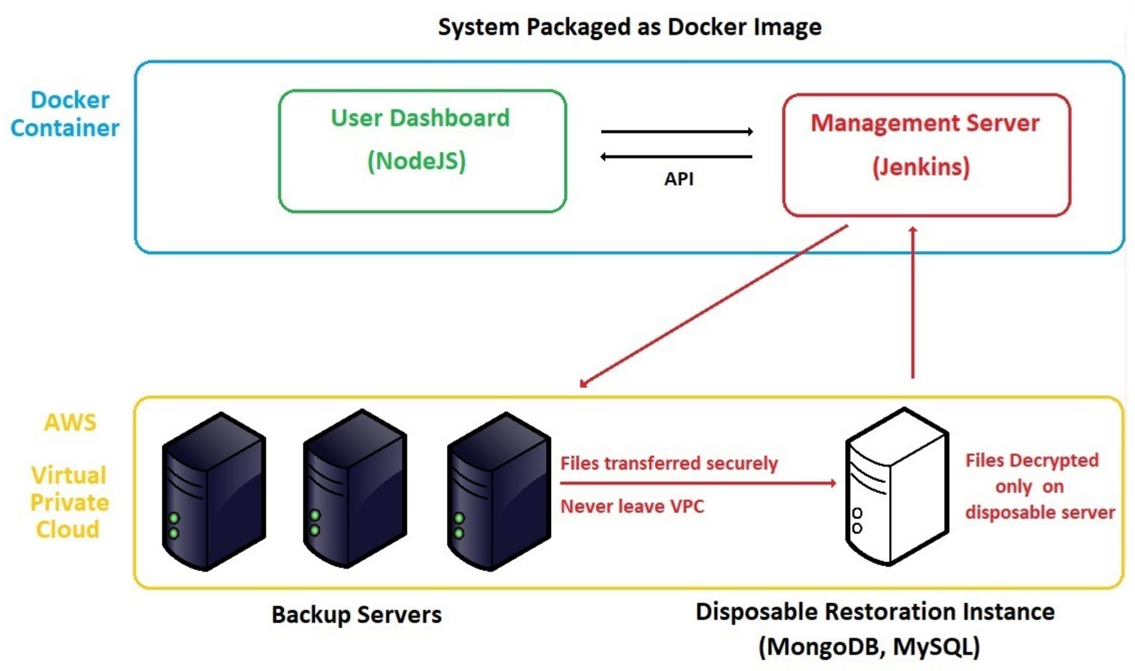
4.1 System Architecture Overview

The system will comprise of three main components:

- Management Server
- User Interface
- Disposable instances/containers

The system will also use existing ECS instances i.e where backups are stored. Depending on the user of the system there may be multiple backup servers in different location (such as AWS regions) or for different data types (relational and non-relational databases).

Figure 1: Diagram of System Architecture


System Packaged as Docker Image
Docker Container
User Dashboard (NodeJS)
API
Management Server (Jenkins)
AWS
Virtual Private Cloud
Files transferred securely
Never leave VPC
Files Decrypted only on disposable server
Backup Servers
Disposable Restoration Instance (MongoDB, MySQL)

Management Server: This will be a small low cost AWS instance on which the Jenkins automation server will be installed. The majority of the system's functionality will be carried out and/or orchestrated by this server. Jenkins jobs will copy the backups from their location to a disposable instance and implement the necessary steps to validate them such as importing and reading.
User Interface: This will provide a simple user interface (UI) for the system, implemented as a simple web app, hosted on AWS. It will allow users with little knowledge of Jenkins and AWS to perform backup restoration checks by adding a layer of abstraction. Users will be able to run restorations by providing the parameters such as the backup file and its location. The UI will utilise the Jenkins API to execute the restoration with the parameters provided.
Disposable Instances: Disposable infrastructure will be used to perform the restoration. This will consist of EC2 instances running the necessary DBMS to perform the restoration. They can also be destroyed afterwards, destroying the decrypted backup, data and therefore maintaining confidentiality.

4.2 Formal Modelling

4.2.1 Sequence Diagrams

The main function of the systems have been demonstrated below in sequence diagrams. Figure 2 shows the process of running a single backup restore. This involves a user manually triggering a restoration using the web interface. This triggers a Jenkins job which automates the remaining steps:

1. Backup copied to *restoration* server;
2. Backup decrypted;
3. Backup import into DBMS;
4. Data read from DB;

Upon completion the *restoration* server is terminated.

Figure 3 shows the process of a scheduling regular backup restoration tests. Again, this is triggered by a user from the web interface. The web interface will pass the

JSON or XML configuration for a job to the Jenkins server. The server verifies the backup server exists before creating the job.

Figure 4 Show the process of deleting an existing scheduled job. The user triggers this process from the web interface. This sends a *delete* command to the Jenkins server via the API to remove the schedule job. The status of the command, indicating a successful or failed restore, is returned to the user.

Figure 2: Run Restore

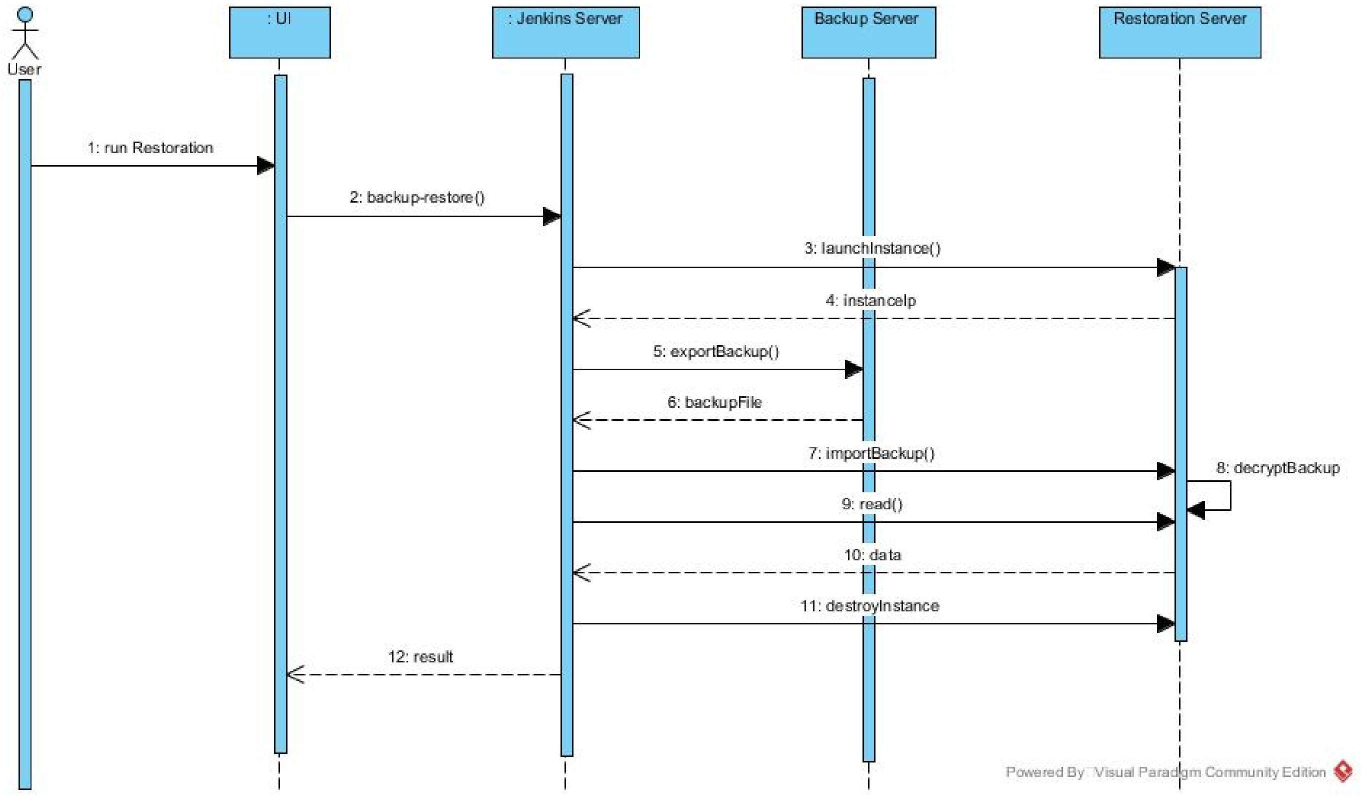


Figure 3: Schedule Regular Restore

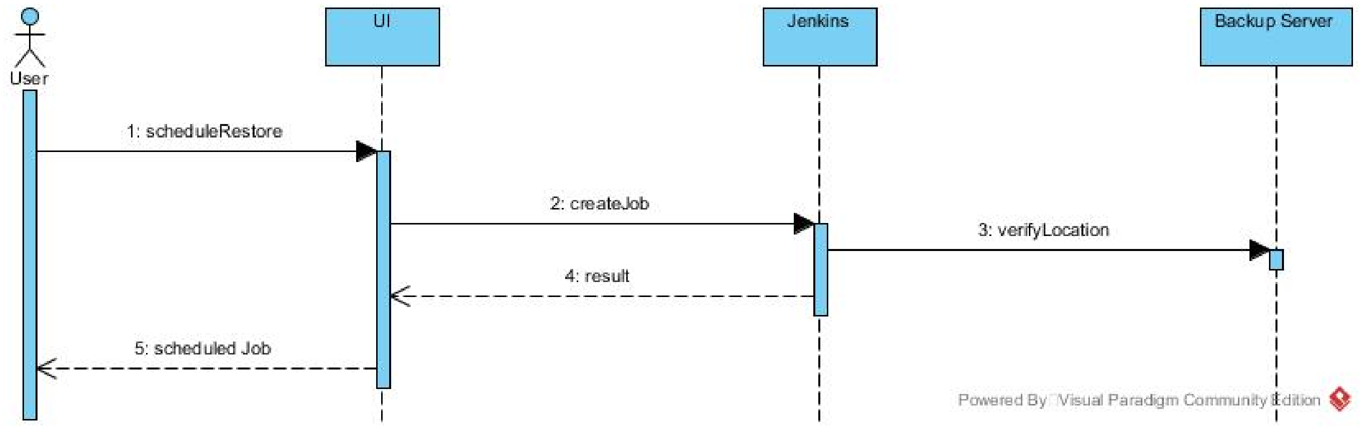
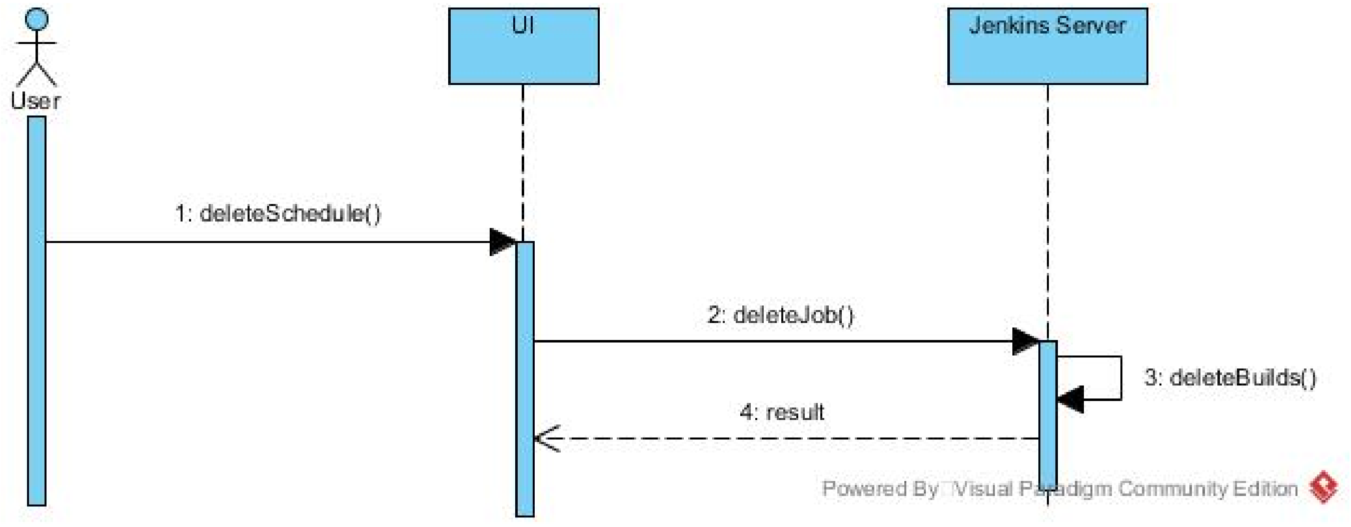


Figure 4: Delete Scheduled Restore



4.2.2 User Stories

User stories are provided in Table 1. Two types of system users and their privileges are described below;

Managers control security aspects of the system:

- Add and remove regular users;
- Manage SSH keys and decryption keys.

Regular Users will perform the day-to-day tasks of the system:

- Perform restorations;
- Schedule restorations;
- View restoration results;

Table 1: User Stories

#	As a	I want to be able to	so that
US1	manager	implement a user system	I control who can run backup restores
US1.1	manager	add my team members to the system	they can run backup restores
US1.2	manager	remove users from the system	former team members no longer have access
US2	manager	add and control sensitive information within the system	I can implement a security policy
US2.2	manager	securely store credentials within the system	they don't need to be entered every time a restore is executed
US2.3	manager	add SSH keys for backup server	the system has secure access backup server
US2.4	manager	add decryption keys for backups	encrypted backups can be decrypted for testing
US2.5	manager	delete SSH keys	expired/outdated credentials are no longer stored
US2.6	manager	delete decryption keys	expired/outdated credentials are no longer stored
US3	manager	execute all same tasks as a regular user	I don't need a second set of credentials to run restores myself
US4	user	login	I can run restores
US5	user	logout	I avoid potential unauthorised access
US6	user	run a test restoration of a backup	I can verify that the backup exists, is a valid file, and is readable
US6.1	user	run a test by filling out a simple form with basic parameters (location, filename) of the backup to test	I can easily run a restore of a specific backup without needing to worry about the implementation
US6.2	user	view the current status of a running restoration	I can review the progress of long running restores
US6.3	user	check if a backup failed or succeeded	I can immediately investigate any failed backups
US7	user	create a schedule of automated restores for a given backup	I don't have to manually execute them myself on a regular basis
US7.1	user	choose the frequency of automated restores within a schedule, from daily through weekly to monthly	I control how often different backups are tested
US7.2	user	check if an automated restoration has started	I can verify my schedule is working correctly
US7.3	user	check the results of an automated restore	I can immediately investigate any failed backups
US7.4	user	view the all past results of an automated restore schedule	view the consistency of my backups success
US7.5	user	modify a scheduled restore	I can change the frequency of a scheduled restore
US7.6	user	the parameters of a schedule	any changes to the backups, such as location, will be reflect in the restoration schedule
US7.7	user	delete regularly scheduled restores	old backups/deleted backups are no longer tested
US8	user	view feedback of a failed restore	I might gain an insight into the fault in the backup
US9	user	notified when a restoration fails	silent, unnoticed fails are avoided

4.3 Front End Design

4.3.1 Wireframes

Wireframes for the frontend are shown below.

Figure 5 shows the homepage. It includes the following components:

- Form for running a restore;
- Form for creating a restore schedule;
- List of schedules (including links).

Figure 6 shows the details and past results of a scheduled restore.

Figure 5: Homepage

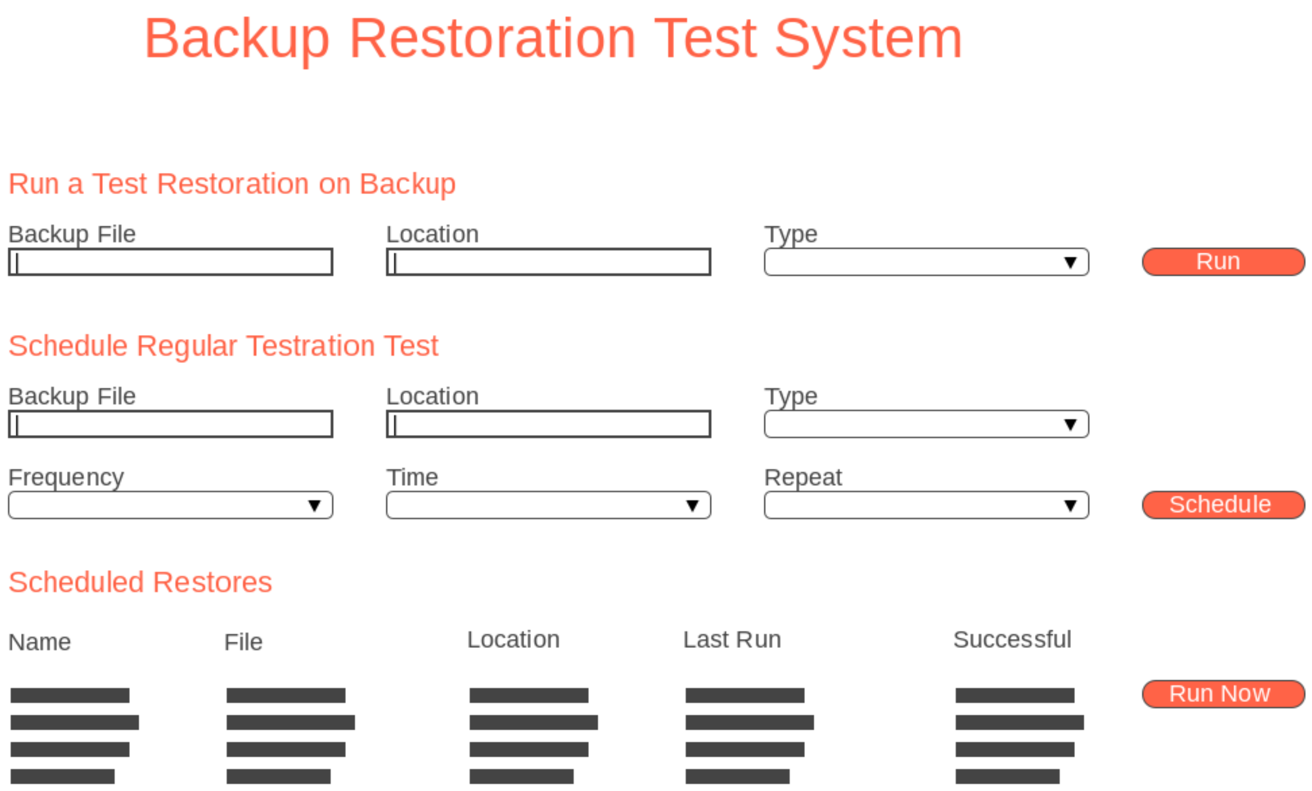
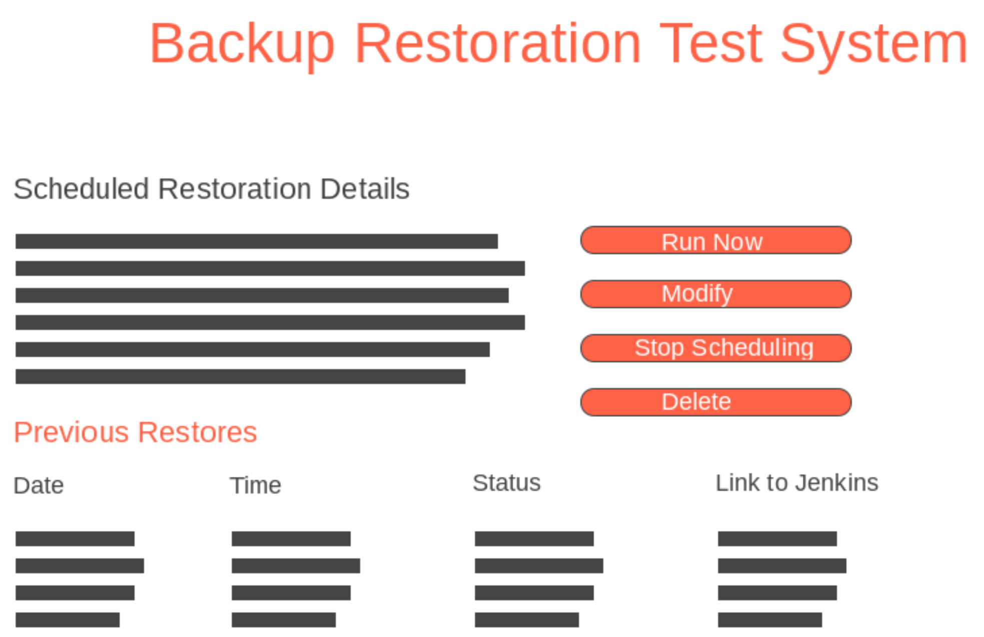


Figure 6: Scheduled Restore



4.4 Enterprise Level Consideration

As this project is being designed for the product owners to potentially deploy it to test production database backups, there are a number of key aspects which should be considered. This is to ensure that the product does not just achieved its main goals but does so in a secure and useable manner for the product owners.

4.4.1 Security

Security is a key aspects of this project. In fact , due to the importance of security it has been defined as one of the key aims of the project (AO4). The security of this system shall by determined by two main areas.

4.4.1.1 Storage of Private Keys

This pertains to the storage of the SSH private keys for accessing backup servers and the GPG private keys used to decrypt the backups. Safe storage of such keys is paramount to the project as exposing these keys would compromise production data. Accordingly, during the research phase of the project, POC3 was concerned with the investigation of this area. In this POC, Jenkins Credentials Plugin was used to store the credentials. A prototype was then used to demonstrate that the private keys could be used to implement a backup restoration while maintaining the confidentiality of the keys.

4.4.1.2 User Authentication

Users of the system will not be able to retrieve any sensitive data through its use, rather they will only have the ability to schedule and run backup restorations. Restorations do not affect the existing backup files or servers as the test is completed on a disposable server in AWS. Also restorations are idempotent, meaning running more than one test on a given backup won't affect the results.

Therefore, malicious users of the system could not affect data confidentiality. However, they could incur increased AWS costs by running and scheduling extra restorations. For that reason access to the UI should be restricted. Accordingly, a user authentication system has been implemented. This system currently consists of a single *admin* user, created during the Docker image build with credentials provided. For a minimum viable product it has been decided that this is sufficient. However, plans for future development included a full User system, allowing t*admin* user to add other uses to the systems. This would allow a team manager to provide each team member with their own set of credentials to use the system.

4.4.2 Deployment

During discussion with the product owners it was decided that the deployment on installation of the final product would be a simple task, requiring little or no configuration of the user. For this reason it was decided that the final product should be delivered as a Docker image.

However, the final system consists of two servers; the management server (Jenkins) and a web server (UI). This means two services are required to run within the Docker image. However, this is not the recommended practice for Docker images (Docker, 2018). A container should be run as a single process which if terminated will terminate the container. For this reason the final product should be package as two images. These images can still be easily run with any configuration required. They will run in separate containers but are preconfigured to be able to communicate.

This method of deploying systems as a network of containers is in fact a common practice in the industry, leading to the development of container management platforms such as Kubernetes (Kubernetes, 2018). Accordingly, OpenShift, Red Hat's own container management platform has been considered as the suitable deployment location for the final product.

4.4.3 Performance

The use of containers to run the final products play a key role in the performance of the system. Containers are similar to virtual machines allowing them to run on a variety of platforms. However, as containers virtualise software instead of hardware, they are much smaller and faster to run (Docker, 2017).

Packaging the final product as Docker images is therefore the suitable solution for implementing a high performance system. The images will contain, and therefore run, only the necessary software to run both the Jenkins server and the node app. Indeed, both the Jenkins server and app server can be easily run on a single device.

5 Methodology

5.1 Agile

The design methodology chosen for this project is Agile. Agile takes an iterative approach to designing and delivering products. It's a goal driven methodology that aims to build and deliver software incrementally from the beginning of the project, in contrast with traditional approaches such as Waterfall which deliver in one final stage. A notable aspect of Agile is user stories. The project is broken into small sections of functionality which can be independently developed and delivered upon completion (Rasmusson, 2017). A number of user stories have been described above, detailing the main requirements of this project.

5.1.1 Scrum

A particular Agile framework which will be used for this project is Scrum. Scrum defines terms used to organise development:

- **Product Backlog:** This is a prioritised list of the jobs which need to be completed. In its entirety, it represents the full development of the project, i.e. all the work required to deliver the final product.
- **Sprints:** Development is divided into a number of equal length periods (often two or three weeks) of work known as sprints. Each sprint has its own small goal to achieve, with some items from the head of the product backlog being developed. This project will be organized into six sprints of two weeks each.
- **Daily Scrum:** The daily scrum, also known as daily *standup*, is a daily meeting at which team members meet to discuss progress and address issues encountered.
- **Sprint Reviews:** At the end of each sprint a review of the work completed is carried out. The next sprint will then begin, developing the next group of items from the backlog being(ScrumAlliance, 2016).

Also defined are a number of roles:

- **Product Owner:** The product owner is responsible for the backlog. They are responsible for ensuring the development succeeds in its goals by implementing the work laid out in the backlog. It is the duty of the product owner to prioritise the backlog.

- **Scrum Master:** The scrum master is responsible for maintaining focus on the current batch of backlog items during each sprint AgileAlliance (2017).

Scrum is an ideal model for developing this project. The Product Owner will be Red Hat and the role of scrum master will be played by the project supervisor. Development will broken into six sprints of two weeks. However, as this is not a team project, daily stand-up meetings will not be held. Rather, meetings with the scrum master on weekly basis and meetings with the product owner on a similar schedule as needed. The user stories which have been used to describe the requirements of the project will be organised into the product backlog.

5.2 Jira

In conjunction with the Scrum framework the Jira development tool will be used. Jira is a project management tool which allows the creation and tracking of tasks or issues. The product backlog, comprising of the project user stories, can be created within the management tool with tasks created for each of the stories on the backlog. Each task can then be prioritised in a manner chosen by the product owner (Atlassian, 2017).

A number of Jira's feature make it an ideal tool to aid in the organisation of this project:

- **Issue Tracking:** Progress on issues within the product backlog can be tracked through useful status tags such as In Progress, On Hold and Done. This provides a mechanism for recording progress on each of the user stories which need to be implemented for this project.

- **Scrum Boards:** Sprints can be represented using Scrum boards. Scrum boards provide a way of organising the issues in a visual manner, grouping them into logical categories (To Do, In Progress, Done) that focus on the progress of the current sprint ScrimInc (2017).

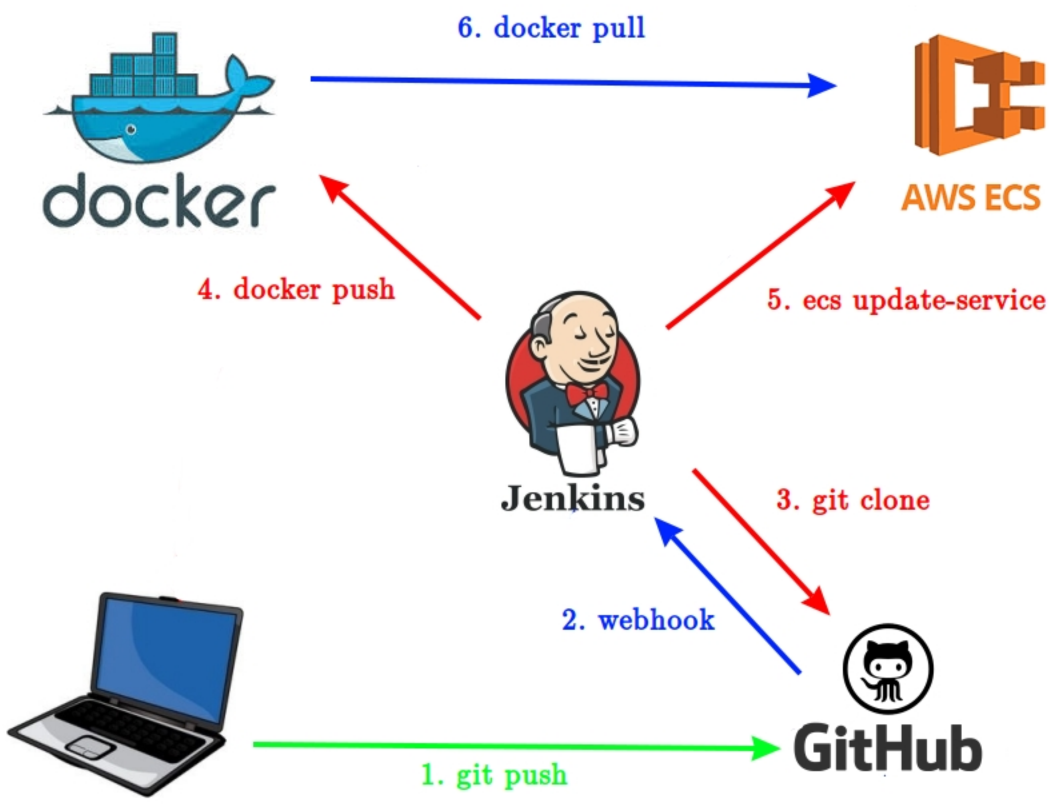
5.3 Continuous Integration/Deployment

Continuous Integration/Continuous Deployment (or continuous Delivery) is a development concept that focuses on the frequent and automated testing building and releasing of code. It aims to remove the large workload required when it is time to release a version or update of a product by performing the same process in a automated manner on every code commit (Pittet, 2017).

Continuous integration refers to preparing the code for release as often as code commits are performed. For example, running tests and building Docker images on each commit meaning code is prepared for release at each stage of development, instead of when it come to release time (Ramos, 2016). Continuous Deployment is a step beyond Continuous Integration. After the code is built it is deployed to a server. However, this may be a development server. Pushing the built code to production requires a manual trigger. Continuous Delivery automates this final manual trigger, meaning the entire process of moving code through testing, building and deployment to production is entirely automated (Ellingwood, 2017).

For this project, CI/CD (continuous development in this case) will be implemented using a Jenkins automated build server. Each time a commit of the frontend source code is pushed to GitHub, the app will be built as a Docker image and deployed to ECS by Jenkins on every code commit. This workflow is shown in Figure 7.

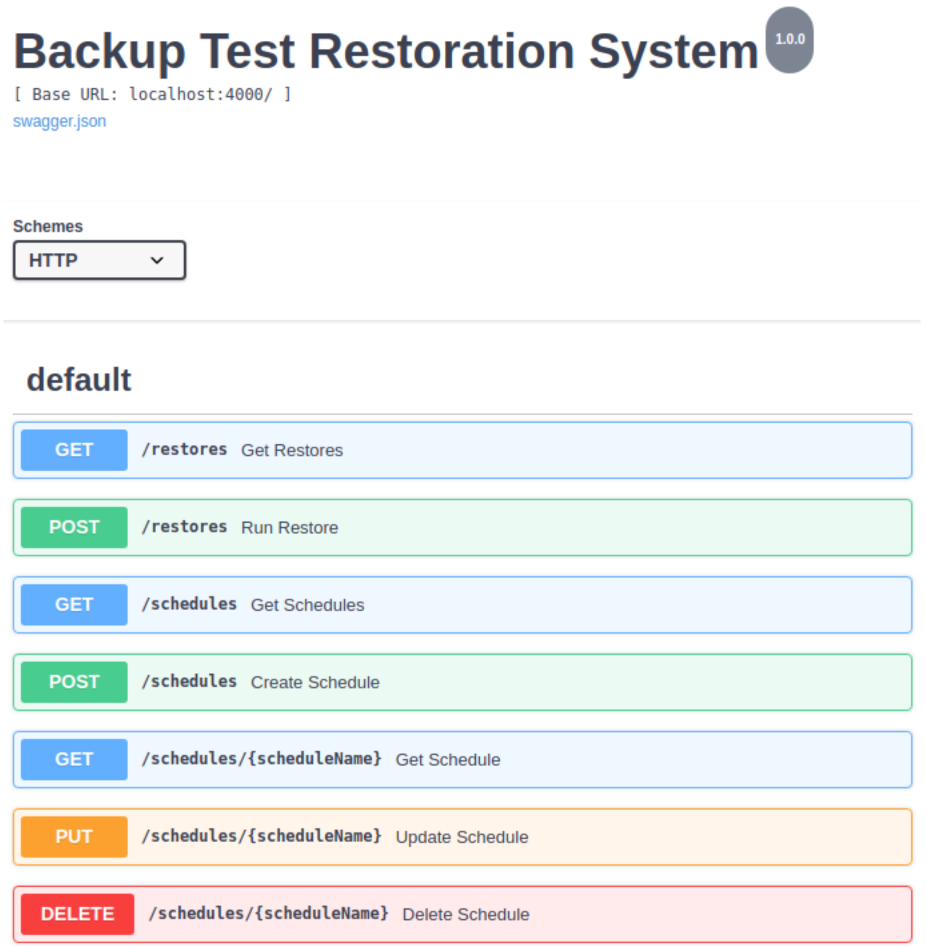
Figure 7: CI/CD of Wep App



5.4 Documentation & Testing

Documentation for the systems API is provided using Swagger. Swagger uses *yaml* or *json* files to document API endpoints, providing functions descriptions, parameters, possible responses and examples. Using the Swagger UI tool, the documented endpoints can then be visualized on the fronted of the web app (Swagger, 2017).

Figure 8: Some of the Systems API Endpoints


Backup Test Restoration System 1.0.0

```
[ Base URL: localhost:4000/ ]
```

swagger.json

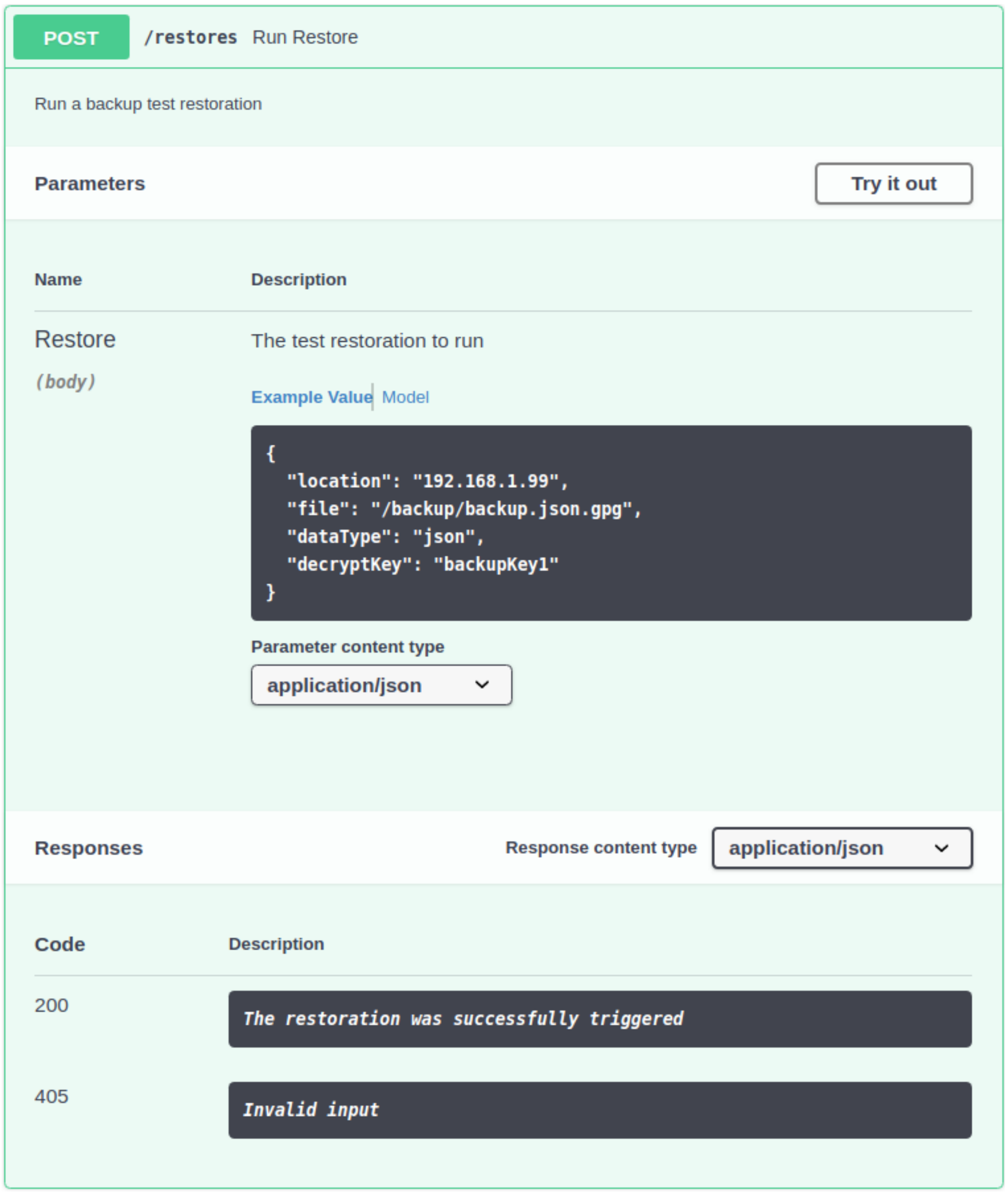
Schemes: HTTP

default

GET	/restores	Get Restores
POST	/restores	Run Restore
GET	/schedules	Get Schedules
POST	/schedules	Create Schedule
GET	/schedules/{scheduleName}	Get Schedule
PUT	/schedules/{scheduleName}	Update Schedule
DELETE	/schedules/{scheduleName}	Delete Schedule

By expanding any of the endpoints shown in Figure 8, the full documentation for that endpoint can be viewed. Along with providing the inbuilt documentation for the system, the Swagger UI tool also allows the user to test the endpoints, either using the sample data provided by the documentation or user input data. A sample of this can be seen in Figure 9. Thus, full testing the system's API can be completed using Swagger.

Figure 9: The Documentation for the *Run Restore* Endpoints



POST `/restores` Run Restore

Run a backup test restoration

Parameters

Name	Description
Restore *(body)*	The test restoration to run

```
{
  "location": "192.168.1.99",
  "file": "/backup/backup.json.gpg",
  "dataType": "json",
  "decryptKey": "backupKey1"
}
```

Parameter content type: application/json

Responses

Response content type: application/json

Code	Description
200	*The restoration was successfully triggered*
405	*Invalid input*

5.5 Completion and Handover

Due to the goal driven and iterative aspects of Agile methodology, a clear idea of a completed project is required. This will provide the final goal to work towards and a clear understanding of when this goal is reached. Accordingly, a *Definition of Done* is a a key element of Scrum (Panchal, 2008). The design methodology for this project will have the following stipulations:
The *Definition of Done* for each sprint will be as follows:

- Each issue (user story) has been tracked within Jira.
- Each issue is implemented;
- Each code for each issue has been documented.
- A sprint review meeting has been held;

The *Definition of Done* for the project will be as follows:

- User stories have been categorised as *Functional Goals* and *Stretch Goals* in an initial backlog refinement;
- All *functional* user stories have been implemented with considerations made for *stretch goals*;
- All code committed to GitHub repository;
- All code documented;
- Latest image of web app deployed to ECS through CI/CD;
- Latest image available on Docker Hub registry;
- A working deployment of the entire system is running and available for demonstration.

The following deliverables will be required for completion of the project.

- Presentation of the project;

- Project report;
- Descriptive poster of project;
- Demonstrative video of project feature.

6 Implementation

6.1 Sprint 0

6.1.1 Sprint Planning

As the initial one of the project, this sprint was concerned with setting up and configuring CI/CD tools for the project. The main tool of the CI/CD process considered was a Jenkins build server to continuously build and deploy the code. Given the agile development methodology it was important to implement these steps early in the project and therefore they were undertaken during the initial sprint.

6.1.2 Sprint Review

The following goals were achieved:

- A Jenkins build server was deployed to AWS;
- An AWS ECS instance to run the web app was deployed;
- A skeleton React app was created with a defined Dockerfile to build image of the app;
- A Jenkins pipeline was created to build the image of the app and deploy it to AWS on every commit to GitHub;

6.1.3 Sprint Retrospective

What did we do well?

- CI/CD process has been set-up and implemented.

What could have been done better?

- Story Points have not been allocated to tickets;
- Breakdown of backlog could be improved, better utilising Epics, Issues and Sub-tasks.

Actions

- Rob Shelly to further refine the product backlog;
- Rob Shelly to add story points to tickets on backlog.

6.1.4 Personal Reflection

This sprint highlighted the need for story points as the lack of such has meant there is not burndown chart for the sprint. Adding story points along with a well refined and organised backlog will allow for better sprint planning in the future.

6.2 Sprint 1

6.2.1 Sprint Planning

The planning for this sprint was based on the existing Jenkins Jobs created during the prototyping process. These jobs need to exist on the Jenkins server when the system is initially deployed/installed by a user. i.e, the user does not create these jobs. Therefore, these jobs need to be created during the installation process. This sprint focused on creating *yaml* files for these jobs which can be used by *Jenkins Job Builder* (JJB) to create the jobs during set-up. This also provided the ability to quickly recreate the jobs should the Jenkins Server crash during development.

6.2.2 Sprint Review

The *yaml* files were created and tested for the following Jenkins jobs:

- Deploy: Spins up a restoration server in AWS (i.e. a server with the correct DBMS installed)
- Decrypt: Moves a backup file to a restoration server and decrypts it.
- Restore: Imports a backup file into the DBMS and performs a read of the data.
- Destroy: Destroys the restoration server after a successful backup restoration.
- Backup Restoration Pipeline: Performs a full backup test restoration (using the above Jenkins jobs).

The sprint delivered on its main goal of creating the *yaml* files. However, further tickets which would have built upon the prototype (displaying the result of the restoration) were not completed.

6.2.3 Sprint Retrospective

What did we do well?

- Technically mastered the complexity of the project, have a vision and a direction to move towards;
- Backlog is very mature, the Minimum Viable Product (MVP) is almost visible;
- JJB was utilised and I now understand how Jenkins jobs work;
- Story Points were introduced and helped to guide the work.

What could have been done better?

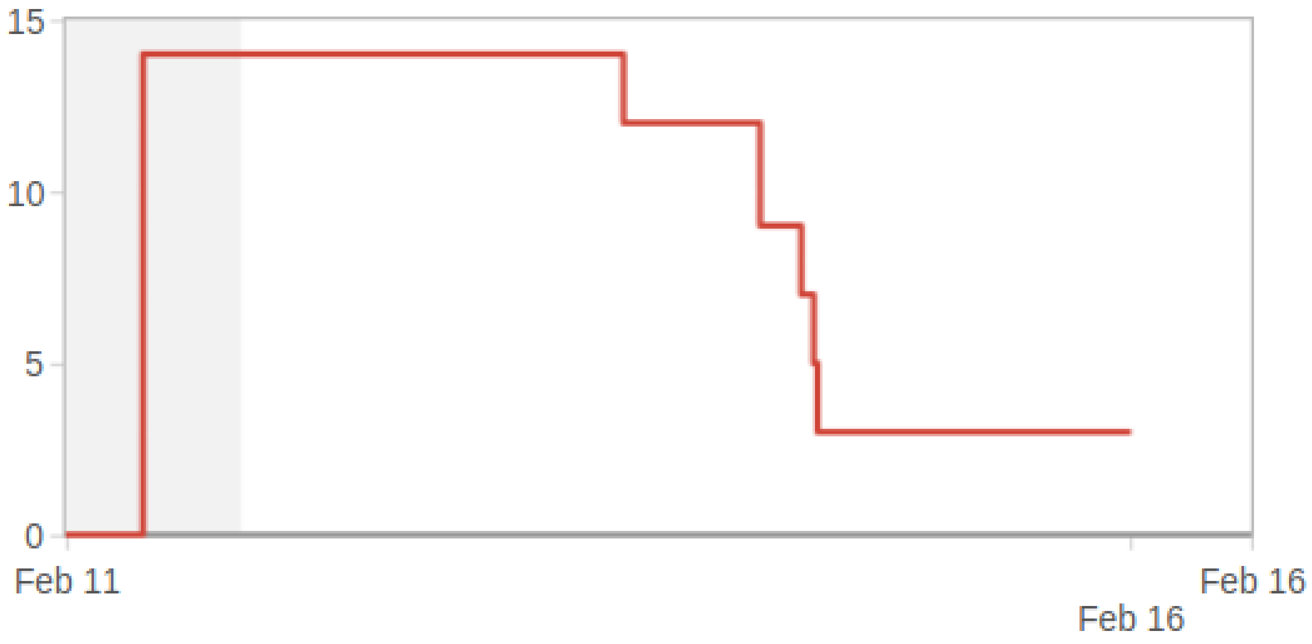
- Burndown was inconsistent as most of the work was completed within a short period of time during the sprint;
- Domain knowledge in certain areas (JJB) was not as strong as I would have liked it to be, which slowed me down and was not reflective of the complexity I had awarded those tickets;
- Outward communication to the stakeholders (Leigh & Paul) was not as good as it could have been.

Actions

- Rob Shelly to get a draft of wireframes to Leigh by next Monday;
- Rob Shelly to complete MVP plan for the backend system and include an API definition and CLI guide;
- Rob Shelly to send regular updates on progress to stakeholders.

6.2.4 Sprint Burndown

Figure 10: Sprint 1 Burndown Chart



6.2.5 Personal Reflection

I was happy with the progress of this sprint as I felt creating the *yaml* files for the Jenkins jobs was a vital task, not only to enable an MVP to be packaged as a deployable system, but also to provide an easy method of recreating the Jenkins jobs should there be a loss of AWS servers during development.

This sprint also highlighted the importance of planning the sprint with respect to other college work to be completed in the given time frame. Although, this sprint was planned for two weeks, the entirety of the work was completed within a number of days during the later of two weeks. In future, if other college work will take precedence for a number of days, I may plan to delay beginning the sprint, allowing the final burndown chart to better reflect the progress of work over the sprint.

6.3 Sprint 2

6.3.1 Sprint Planning

Planning for this sprint focused on completing the API for my systems. Through conversations with the stakeholders it was decided that the priority for this sprint should be completing the API and its documentation. Creating the web app frontend for the API could be completed later. However, given that there was still quite a larger number of API related tickets on the product backlog, not all were brought into this sprint.

6.3.2 Sprint Review

The following API function were implemented:

- Get results of all recent test restoration;
- Get a list of all scheduled test restorations;
- Create a new test restoration schedule;
- Update test restoration schedule;
- Delete a test restorations schedule;
- Get results of all test restorations for a given schedule;
- Add an SSH key to the system;
- Add a GPG key to the system.

Documentation for each of the API functions was also completed using Swagger Docs. Thus, the sprint delivered on it's goal of completing and documenting the bulk of the API functions.

6.3.3 Sprint Retrospective

What did we do well?

- Working on a concentrated area (API) was very beneficial;
- Product owners were satisfied with the increase in progress updates;
- A lot of knowledge was gained around the documentation using Swagger, helping to develop the backend without the need for a frontend to test it;
- Time management was better than the previous sprint, results on all tickets being completed.

What could have been done better?

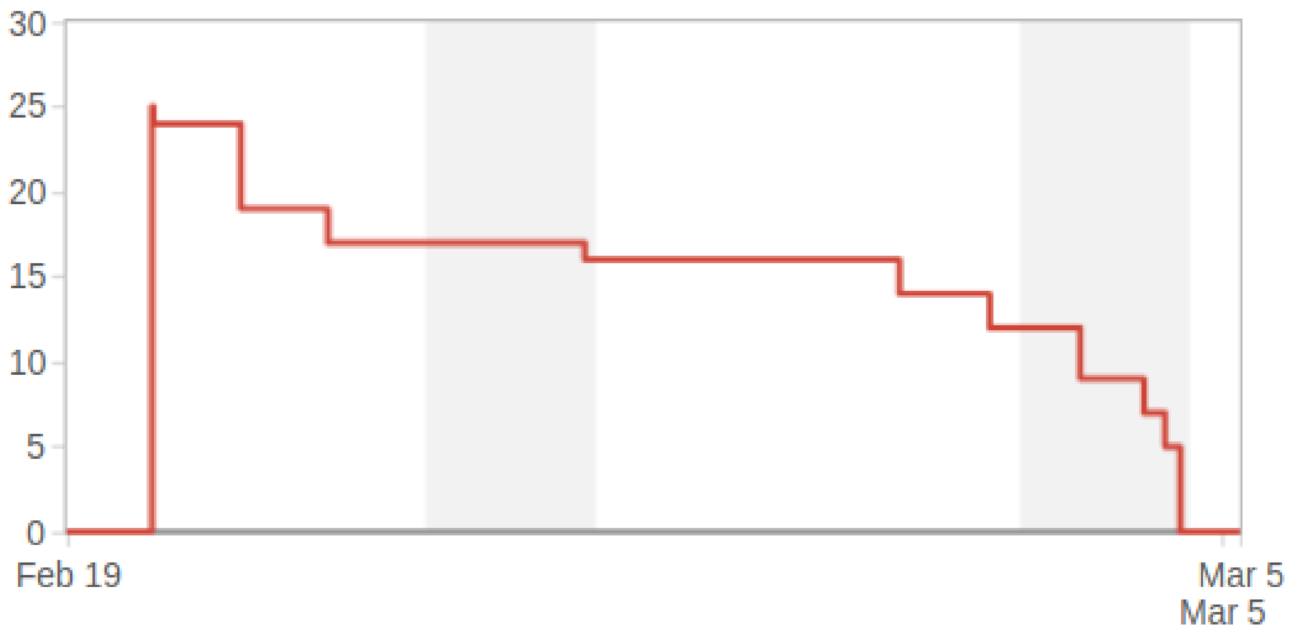
- Burndown chart, although better than the previous sprint is still somewhat inconsistent. However, with all tickets completed, the overall result is still good.

Actions

- Rob Shelly to complete the last few API related tickets from the backlog in order to allows work to proceed to focus solely on the UI.

6.3.4 Sprint Burndown

Figure 11: Sprint 2 Burndown Chart



6.3.5 Personal Reflection

I was very happy with the work completed during this sprint. Prior to Sprint planning and backlog refinement I had been planning on completing tickets by grouping API functions with their corresponding UI elements. However, on the advice of the product owners I decided to focus solely on completing the API first before moving on to the UI.

Having completed the sprint I now feel that separating the API and UI was a much more productive approach. Swagger, which was used to document the API also contains a feature to test the the documented API calls. Therefore, by focusing only on writing and documenting the API, I was able to stay in the mindset of the API functions without having to constantly transition between developing API calls and UI elements, while still being able to test the functions using Swagger.

6.4 Sprint 3

6.4.1 Sprint Planning

Planning for this sprint focused on two main areas the first of the which was to finalise the backend API. The plan set out with the product owners prior to the previous sprint dictated that the backend API should be completed before focusing on the UI. Therefore, in accordance with the actions dictated during the previous retrospective, finishing the last remaining tickets concerning the API would be the initial focus of this sprint.

Given that there was only a small number of API related tickets remaining on the backlog, the second aspect of this sprint was beginning work on the UI. Accordingly. a number of user

6.4.2 Sprint Review

The sprint achieved it's goal by completing the final API related tickets and implementing the first features of the UI. The following API functions were completed:

- API function to list all SSH keys;
- API function to list all GPG keys;
- API function to delete an SSH key;
- API function to delete a GPG key.

Additionally, the following UI elements were created:

- A web form to run a single backup restoration;
- A page displaying the results of recent backup restorations;
- A web form to schedule regular backup restorations.

6.4.3 Sprint Retrospective

What did we do well?

- Having completed all the backend API calls, the MVP was clearly defined;
- Time was better managed than in previous sprints, having chosen to delay the start of the sprint while working on other projects;
- Burndown chart was consistent and reflective of better time management.

What could have been done better?

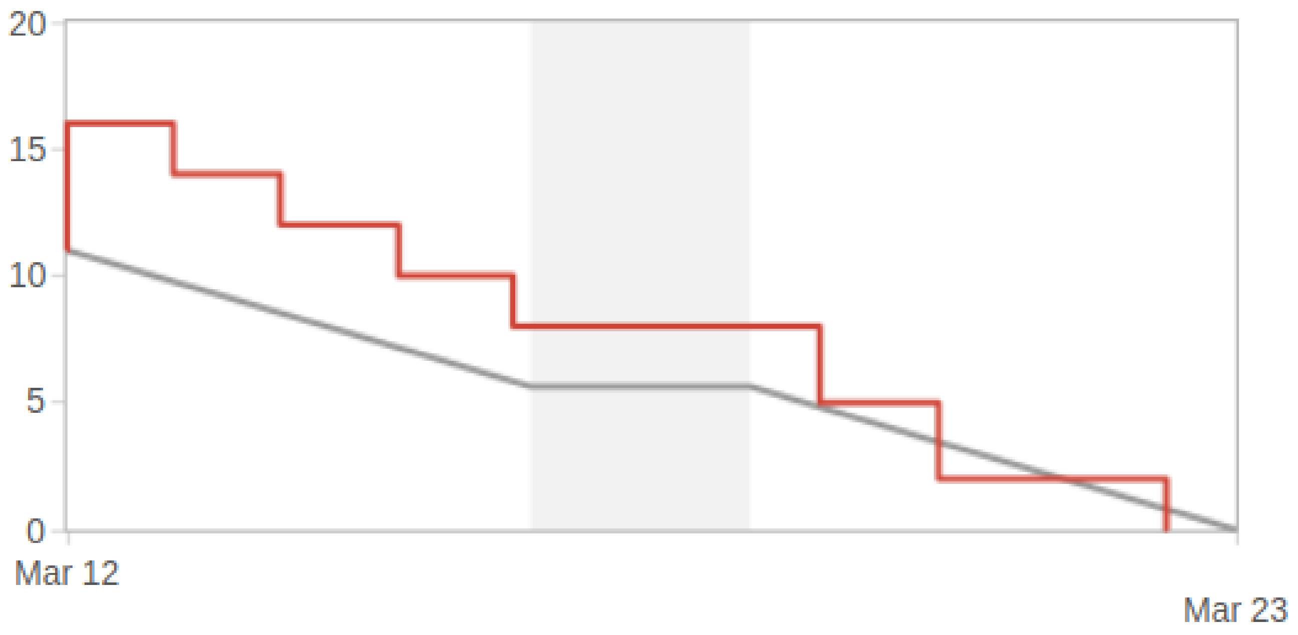
- Communication with the product owners was bad due to time spent on other projects;
- UI design highlighted an issue with the API leading to some refactoring.

Actions

- Rob Shelly to decide if a Command Line Interface (CLI) should be implemented.

6.4.4 Sprint Burndown

Figure 12: Sprint 3 Burndown Chart



6.4.5 Personal Reflection

I was happy to complete the backend during this sprint, allowing time to begin working on the UI without need to worry about the underlying API calls. This reinforced benefit of product owners recommendation to complete the API before beginning the UI. However, beginning work on the UI did highlight an issue with the backend. During UI work I realised that I had not created one of the necessary function calls. Although it was a simple solution to create the API call when needed, the time needed to do so was a setback on UI work.

I felt my time management was much better during this sprint. Previous sprints featured periods of inactivity due to time spent on other projects. This was indicated by inconsistent burndown charts. Therefore I chose to delay the start of this sprint while I worked on other projects. This allowed me to focus on this project for the duration of the sprint once it had commenced. This was reflected in a much more consistent burndown chart.

During this sprints retrospective the product owners mentioned the possibility of creating a command line interface (CLI) for the system. A CLI would allow more advance users to use the system without the need to visit the UI. This is something the product owners have indicated is a common desire amongst more advanced users who are more comfortable using the command line. I was disappointed that this thought had not occurred to me during project planning. At this stage of development, undertaking such a task might not be feasible, or may detract from my ability to implement a satisfactory UI. I felt that my two options for an MVP were to either focus entirely on the UI or reduce some UI features and also implement a CLI. I decided with the product owners that I should make the decisions prior to commencing my next sprint.

6.5 Sprint 4

6.5.1 Sprint Planning

Having completed work on the API during the previous sprint, this sprint focused heavily on creating the UI. During the planning phase of this sprint a decision was made to focus entirely on the UI for the purpose of creating an MVP and that a CLI would be implemented in a future version. Therefore, it was decided that the remaining UI related tickets should all be completed during this sprint.

6.5.2 Sprint Review

This sprint achieved its goal of finishing the remaining UI related tickets. The following UI elements were created:

- A web form to add SSH keys;
- A web form to remove SSH keys;
- A web form to add GPG keys;

- A web form to remove GPG keys;
- A web form to modify a scheduled restoration;
- A web page to display all schedules;
- A web page to display all results of a given schedule.

This sprint added to email notification function to the system.

6.5.3 Sprint Retrospective

What did we do well?

- Time management was good, resulting in a consistent burndown chart;
- A decision was made and justified regarding the implementation of a CLI;
- Communication with the product owners was good, including keeping them up to date on the decision to focus on the UI.

What could have been done better?

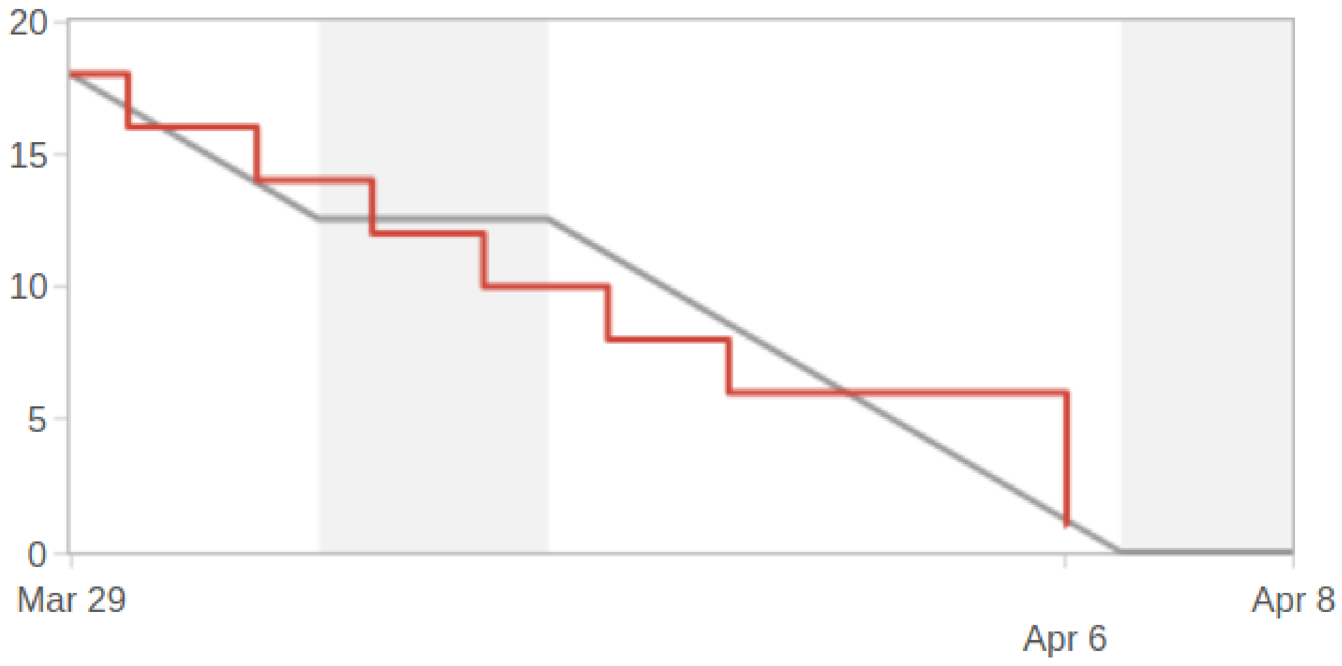
- Email notification system should have been implemented earlier;
- Implementation of the email system highlighted issues with system design.

Actions

- Rob Shelly to consider how to package final product.

6.5.4 Sprint Burndown

Figure 13: Sprint 4 Burndown Chart



6.5.5 Personal Reflection

I was very pleased with the result of this sprint having completed the UI work. At this point an MVP was almost complete. I was very happy that I had taken the product owner's advice to separate work on the backend and frontend. The benefits of this decision really showed during this sprint when I was able to focus almost exclusively on the UI, except for some minor refactoring.

Prior to starting this sprint I made the decision to focus on implementing a comprehensive UI for the MVP, choosing to push the CLI to the backlog and implement it for a later version. Due to the limited time to complete this project I felt that I could not satisfactorily implement both a UI and CLI. The only option would have been to complete only basic user features for the UI and advanced user features for the CLI. However, I believe in order to deliver a useable and demonstrable system that at least one of the interfaces should be completed in its entirety. Thus, I made the decision to focus on the UI, which graphically emphasises the results of restorations.

During this week I also completed work on the email notification system. In retrospect I should implemented this function earlier as notification of failed backups is a main objective of the system. Also, during the development of the function a flaw in the systems logic was noticed which may cause the system to fail to notify the users when a backup process has not succeeded. Although a fix was easily implemented, it would have been preferable to notice and overcome this issue earlier in development.

In accordance with the actions listed during the previous retrospective, prior to commencing this sprint some consideration was given to packaging the final product. Previously the plan had been to package the UI as a Docker image, while the Jenkins backend would run on an AWS instance. This however would require some setup of the systems. Therefore, it was decided that if possible the system should be packaged entirely within a single Docker image.

6.6 Sprint 5

6.6.1 Sprint Planning

Planning for this spring was concerned with delivering the MVP as a packaged product. During the previous sprint retrospective with the product owners, they suggested that the final product be packaged as a Docker image. However, after some research into this it was decided that the better solution would be to deliver two Docker images, one for the Jenkins management server and one for the UI web server. This in fact is a better practice as it is not advisable to run more than one service inside a Docker container. This will still allow the final product to be easily run as both containers will be able to communicate with one another.

Prior to this sprint it was decided that although a fully implemented user system would be pushed to the backlog for development for a later version, it would be preferable to have at least a single *admin* account to authenticate at the frontend.

This would provide an extra layer of security allowing the web app to be deployed to a public domain without the system being exposed.

6.6.2 Sprint Review

This sprint achieved its goal of delivering a packaged MVP to the product owners. The following tickets were completed in order to package the system:

- Created a Dockerfile which defines the image of the UI;
- Created a Dockerfile which defines the image of the Jenkins server;
- Created XML definitions of all necessary Jenkins jobs to allow their creation during the image build process.
- Create Groovy script to configure Jenkins during the image build process including configuring jobs, security and the SMTP server for email.

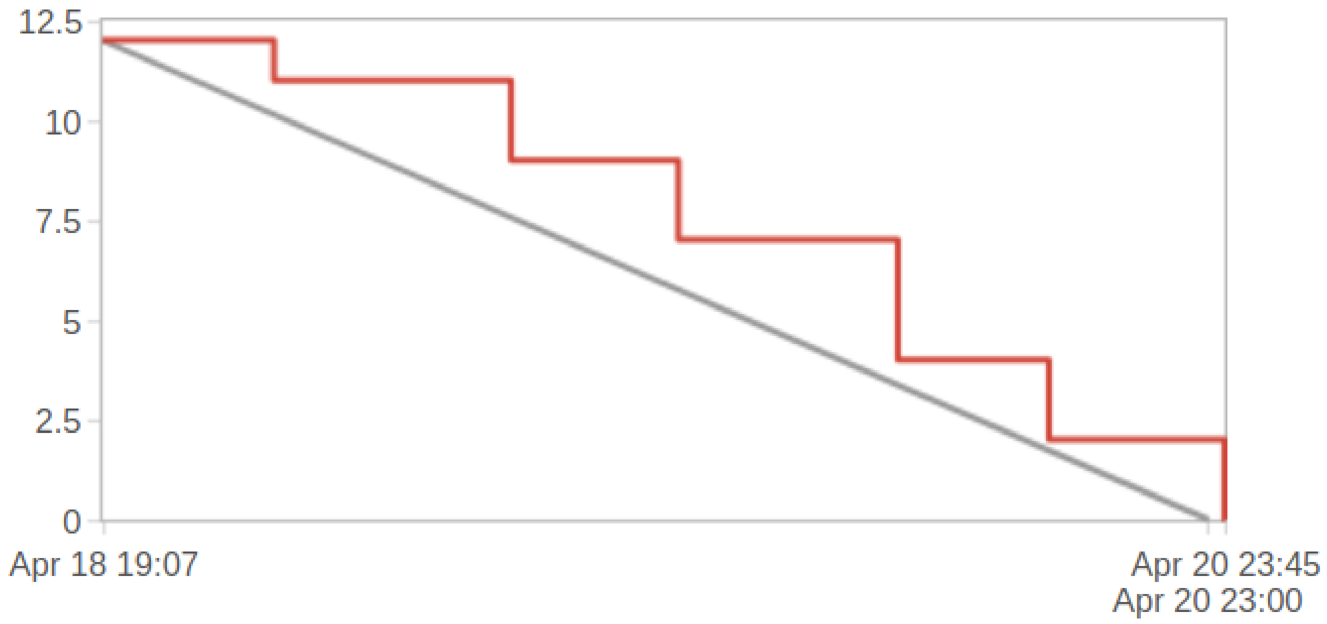
6.6.3 Sprint Retrospective

What did we do well?

- System was packed as easily runnable Docker images;
- An MVP was delivered to the product owners on schedule;
- Consideration was made for the security of the system and basic user authentication implemented as a result;
- Time management was good, reflected by a consistent burndown chart;
- Communication with products owners was good.

6.6.4 Sprint Burndown

Figure 14: Sprint 5 Burndown Chart



6.6.5 Personal Reflection

I felt this was a very successful sprint as it not only delivered an MVP to the product owners but did so on schedule. I was very happy with the final product packaged as two Docker containers. During sprint planning I experimented with running two services inside a single container and found that although it was possible, it was quite unstable and often crashed. After reading Docker's supporting documentation I feel not only was the choice to implement two images a practical decision but also in keeping with best practices. The final MVP is also very stable in this configuration.

I was also very pleased to have been able to implement basic user authentication during this sprint. At previous sprint planning sessions and retrospectives it had been suggested that due to the current sprint velocities, authentication may need to be pushed to the backlog for development in a version succeeding the MVP. Although I felt this was a safe decision to make for an MVP as the system would quite possibly by to deployed as an internal service (i.e. within private networks), the addition of authentication certainly strengthens the security of the product.

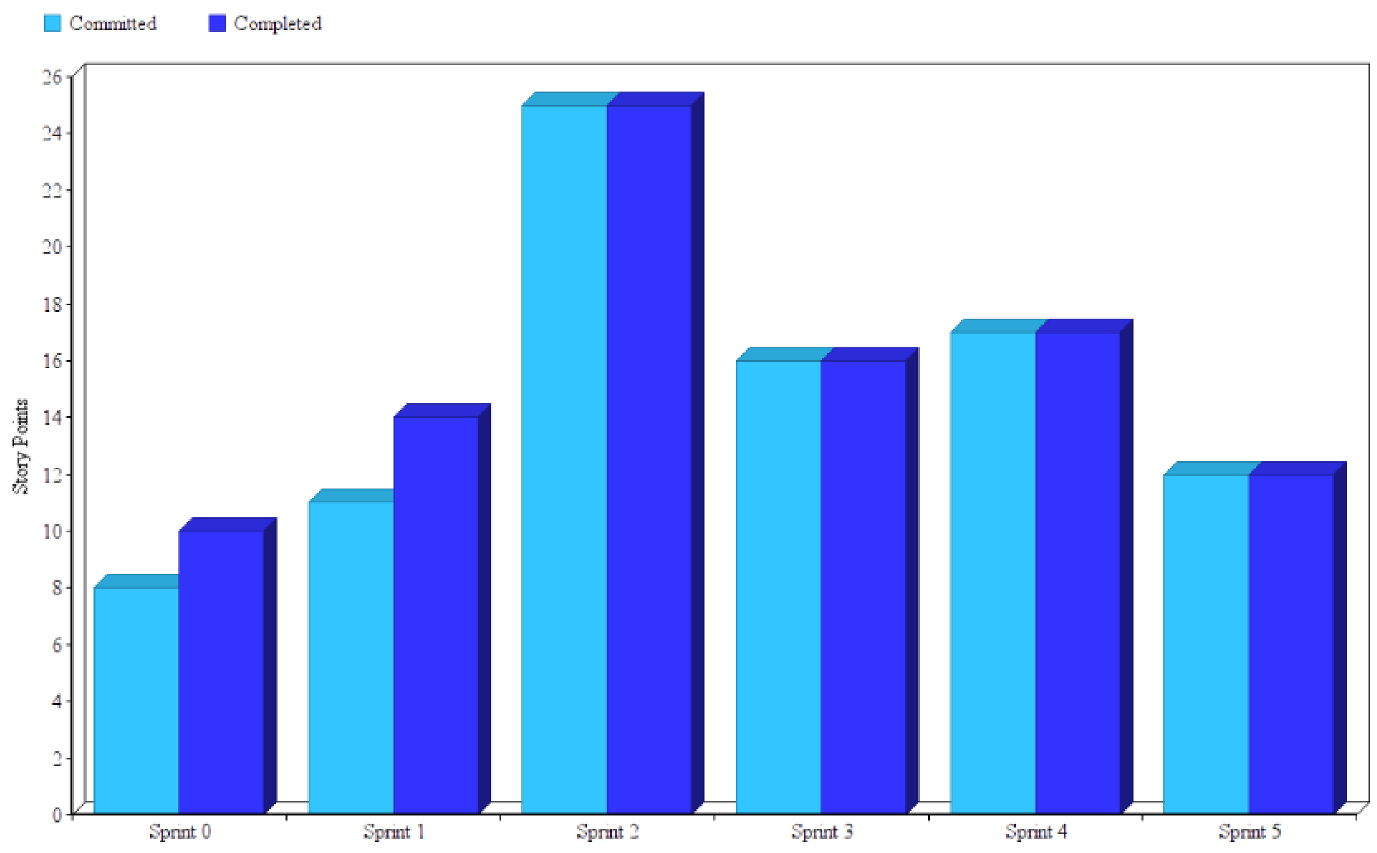
Overall I was quite happy with this sprint as I feel it was quite reflective of the lessons learned over previous sprints. Strong sprint planning meant that no refactoring or backtracking was needed as the right decision was made with regards to building one or two Docker images. This also helped time management and contributed to an ideal burndown of story points.

6.7 Sprint Metrics

6.8 Sprint Velocities

A Velocity Chart is shown in Figure 15. This shows the work committed to and completed in each of the sprints in terms of story points. The chart shows that initially the development time available during the sprints was being overestimated. Thus, in the first two sprints all tickets committed were not completed. However, by the third sprint this had been rectified, with better estimations and time management leading to more consistent results.

Figure 15: Velocity Chart



6.9 Sprint Burndowns

Sprint burndown charts have been provided for completed sprints. The burndown charts provided valuable feedback on the development of each sprint. This is especially evident from earlier sprints where the charts highlighted that much of the development work was begin carried out during a relatively short period of the two-week-long sprints. This was due to time also spent on other projects. This feedback helped to better manage time in subsequent sprints in which they may have been delayed or shortened to reflected only the time allocated to this project.

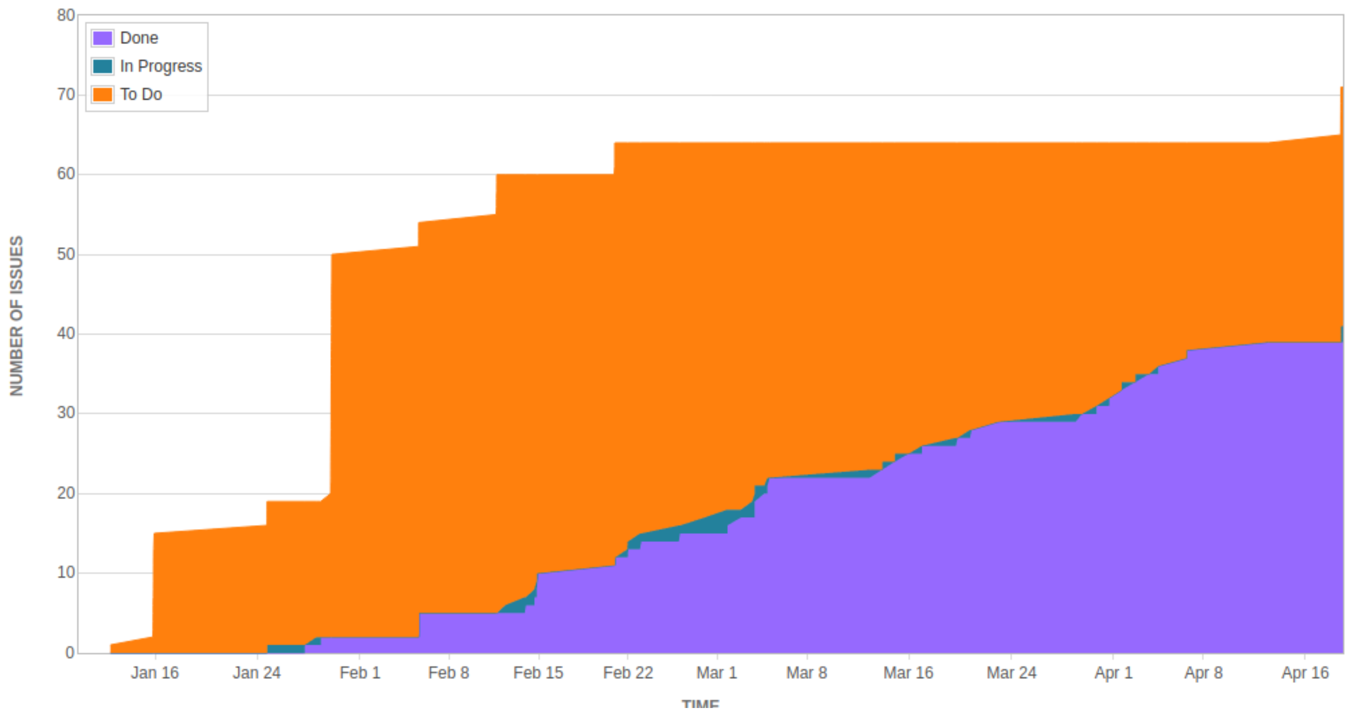
7 Summary

7.1 Project Direction

The project at this point has succeeded in delivering an MVP to the product owners. However, development is not finished and is still ongoing.

Current projections estimate that the project should complete its backlog within two/three more sprints, at which point a fully implemented version of the product as set out by the project's aims and objectives will be completed. This projection can be extrapolated from the cumulative flow chart shown in Figure 16 which shows the progression of tickets under the three categories; *To Do*, *In Progress* and *Done*.

Figure 16: Cumulative Flow Chart



This chart shows not only completion of work but also the discovery of new tasks and issues during the development process, leading to an increase in the workload in the *To Do* category. Of particular note in this chart is the significant increase in the workload that occurred in early February. This is reflective of the discussion and actions after an early retrospective with the product owners to separate the concerns of the backend API from the fronted UI. This resulted in intensive backlog refinement which included the creation of some new tickets.

7.1.1 Future Development

This project has been designed with future expansion of product features in mind. Indeed a noted aspect of this project is that it can currently test no relational data, but can be easily expanded to test relational data. This is facilitated by the modularity of the management servers tasks (created as Jenkins jobs) which can be interchanged as new tasks are added. For example:

- The task for restoring and reading data could be swapped with one which restores MySQL data instead of JSON.
- The decryption task (which use asymmetric cryptography) could be swapped with one which uses symmetric cryptography.

Neither of these operations would affect other tasks and would be invisible to the end user. The user could simply enter the correct setting at the dashboard and the necessary tasks would be *chained* together using Jenkins Pipelines.

7.2 Review

7.2.1 Core Goals

This project delivered on its core goals set out by the stakeholders. In relation to its initial aims and objectives, the project has delivered the following aspects:

- AO1: An automated system for running a scheduling test restorations has been implemented:
- AO2: A notification system has been implemented to notify users of failed backups;

- AO3: The creation and destruction of the necessary infrastructure has been automated in order to reduce uptime and therefore also costs;
- AO4: Credentials have been stored and utilised in a secure manner and sensitive backup data has not been exposed;

The delivered product implemented three main components:

- A management server for the configuration and orchestration of the backup restores, implemented as a Jenkins server;
- A comprehensive ExpressJS API for interacting with the management server, abstracting the full configuration of Jenkins away from the user.
- A comprehensive frontend UI for users to use the API.

7.2.2 Stretch Goals

The following stretch goals have been defined and remain on the product backlog for further development:

- **CLI:** Create a CLI for more advanced users who are more comfortable or proficient in command lines to use the system without visiting the user dashboard.
- **User Authentication:** Implement a more flexible user system which allows the addition of more users.

7.3 Learning Outcomes

This project has produced a number of learning outcomes.

7.3.1 Technical Learning Outcomes

The following technical skills were gained or developed over the course of this project:

- Realised the benefits of the separation of frontend and backend through the development of a backend API and a frontend UI, emphasized through the product owners recommendation to complete one before the other as oppose to concurrently;

- Gained valuable insight into the containerisation of applications through researching the option of deploying multiple services to one or more containers;
- Learned about the CI/CD process through the implementation of a Jenkins CI/CD pipelines to build and deploy Docker images, a common software development practice in the industry when developing as part of a large team;
- Gained experience in developing a MEAN-stack application using ExpressJS, NodeJS and ReactJS having previously had little experience developing *'full-stack'* applications;
- Reinforced the importance of API documentation and testing though the use of SwaggerUI;

7.3.2 Non-Technical Learning Outcomes

There were also a number of non-technical skills gained or developed over the course of this project.

- Learned about the benefits of project planning using Jira to define entire project workload early in the development stage.
- Experience was gained when dealing with a product owner. This included not only listening to their requirements and recommendation but also making decisions during development and updating the product owners accordingly.
- Learned how to correctly track workload and its progress with Jira. The benefits of this were emphasised later in development when enough work had been completed to compile graphs which highlighted problem areas. This helped keep the project focused on it's goal and on schedule.
- Communication skills improved over the course of the project as a result of regular meetings with the scrum master and product owners.

7.4 Personal Reflection

As the learning outcomes above show, I feel I have gained or improved many skills while working on this project. Many of the skills I believe will be very beneficial to me in my future career. I found great satisfaction in completing and delivering the MVP as this is the largest and most comprehensive project I have undertaken.

Personally I feel one of the most important learning outcomes of this project was the benefit of project planning. After the initial meeting with the product owners, at which point the main details of the project were outlined, there was a quite time-consuming and tedious job of splitting the entire project into small tasks. These tasks were then added to Jira as tickets. The granularity at which this had to be done was something I had never completed before. In previous projects I would set out *To Do* lists at a much higher level, with only a few items on the list.

However, the amount of time and effort put into creating these tickets and organising them into *Epics* and *Issues* proved to be hugely beneficial later on in the development phase. Having the tickets created that clearly defined all the steps required to implement each user story meant I never lost focus of what I was doing. Also on completion of any given task, there was no confusion or decisions needed as to what to do next. This was always easily determined due to clearly defined tickets and a well refined backlog.

I also found the prototyping stage to be of huge benefit. Initially I was unsure as to whether the project would be technically feasible. In the past I would have determined this only though beginning development and hoping to overcome issues as they arose. However, the product owners' recommendation and help in identifying the key areas of uncertainty and defining *proof of concepts* accordingly was a valuable lesson. This is a lesson I will certainly remember for future projects i.e. identify problem areas before development begins and investigate their feasibility with a *proof on concept* prototype.

Overall I found the project to be greatly beneficial to my growth as a developer. Accordingly, I would deem the project to be a personal success with many valuable lessons learned and experiences gained.

8 Bibliography

AgileAlliance. Agile glossary, 2017. URL https://www.agilealliance.org/agile101/agile-glossary/.

Amazon. Amazon EC2, 2017a. URL https://aws.amazon.com/ec2/.

Amazon. Amazon EC2 Conatiner Service, 2017b. URL https://aws.amazon.com/ecs/.

Atlassian. Jira software, 2017. URL https://www.atlassian.com/software/jira.

Stephen Connolly. Credentials plugin, 2017. URL https://wiki.jenkins.io/display/JENKINS/Credentials+Plugin.

Docker. What is a container, 2017. URL https://www.docker.com/what-container.

Docker. Run multiple services in a container, 2018. URL https://docs.docker.com/config/containers/multi-service_container/.

Justin Ellingwood. An introduction to continuous integration, delivery, and deployment, May 2017. URL https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment.

GitLab. Gitlab.com database incident, February 2017a. URL https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/.

GitLab. Postmortem of database outage of january 31, February 2017b. URL https://about.gitlab.com/2017/02/10/postmortem-of-database-outage-of-january-31/.

Jenkins. Jenkins, 2017. URL https://jenkins.io/.

Kubernetes. Production-Grade Container Orchestration, 2018. URL https://kubernetes.io/.

Natasha Lomas. Gitlab suffers major backup failure after data deletion incident, February 2017. URL https://techcrunch.com/2017/02/01/gitlab-suffers-major-backup-failure-after-data-deletion-incident/.

MongoDB. Import example dataset, 2017. URL https://docs.mongodb.com/getting-started/shell/import-data/.

NPM. Npm, 2017. URL https://www.npmjs.com/.

Dhaval Panchal. What is definition of done DoD?, 09 2008. URL https://www.scrumalliance.org/why-scrum.

Sten Pittet. Continuous integration vs. continuous delivery vs. continuous deployment, 2017. URL https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd.

Marcia Ramos. Continuous integration, delivery, and deployment with gitlab, 08 2016. URL https://about.gitlab.com/2016/08/05/continuous-integration-delivery-and-deployment-with-gitlab/.

Jonathan Rasmusson. Agile in a Nutshell:what is agile, 2017. URL http://www.agilenutshell.com/how_does_it_work.

ScrimInc. Scrum board, 2017. URL https://www.scruminc.com/scrum-board/.

ScrumAlliance. Learn about Scrum, 2016. URL https://www.scrumalliance.org/why-scrum.

Simon Sharwood. Gitlab.com melts down after wrong directory deleted, backups fail, February 2017. URL https://www.theregister.co.uk/2017/02/01/gitlab_data_loss/.

Swagger. Swagger: The worlds most popular api tooling, 2017. URL https://swagger.io/.

Appendix A Code Repositories

Source code for the Jenkins management server can be found at the following link:
https://github.com/robshelly/fyp-jenkins-server

Source code for the UI frontend server can be found at the following link:
https://github.com/robshelly/fyp-frontend

During development the following code was used to quickly create jobs on a Jenkins server (remote or local) but does not feature in the final product:
https://github.com/robshelly/fyp-jenkins-jobs

Appendix B Docker Images

In order to run the Docker images they should be first built by the user, allowing the user to configure their own authentication credentials on the Jenkins server and their own email address for Jenkins' SMTP server.

Code Sample 1: Build Jenkins Management Server

```
git clone https://github.com/robshelly/fyp-jenkins-server.git

docker build \
--build-arg jenkins_username='admin' \
--build-arg jenkins_password='fyp-project' \
--build-arg email_address='EMAIL' \
--build-arg email_password='PASSWORD' \
-t jenkins-backup-test-centre .

docker run -d --name jenkins -p 8080:8080 jenkins-backup-test-centre
```

Code Sample 2: Build Fronted UI Server

```
git clone https://github.com/robshelly/fyp-frontend.git

docker build -t dashboard-backup-test-centre .

docker run --net="host" -d --name dashboard -p 3000:3000 -p
    4000:4000 dashboard-backup-test-centre
```