Backup Restoration Test System

Status Report (semester 1)

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 to test 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 main objectives of the system are as follows:

AO1 Eliminate the mundane and time consuming task of backup testing by automating regular backup restorations and recording 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 once 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 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 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 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 test by creating a Jenkins job which restores a backup file to the DBMS and perform 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 a the SSH credentials for the various servers could be stored in Jenkins and used securely into order 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 trigger 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 to verify decryption.

2.3.2 Results

This POC demonstrated that the system, in it's 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 to create a prototype. The prototype consisted of a single form on a web app which allowed the user to inupt 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 import the file into a DBMS and preformed a full read of the database, thereby proving that the backup file was valid.

3 Technologies

3.1 Docker

Docker is a container platform for building and managing applications. This project is interested not in Dockers platform but rather in the 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 is 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 systems 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 it's 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 light-weight 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 it's light-weight 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

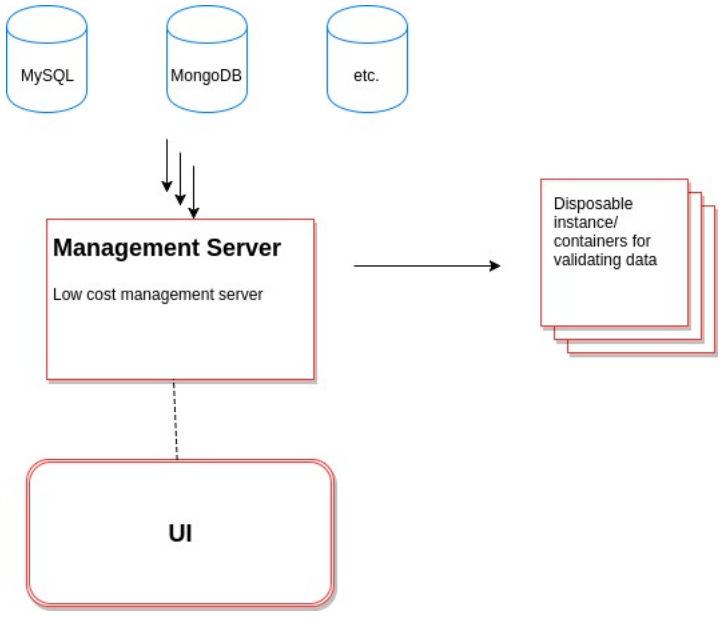
4.1 System Architecture Overview

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



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 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 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 verify 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 commands 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

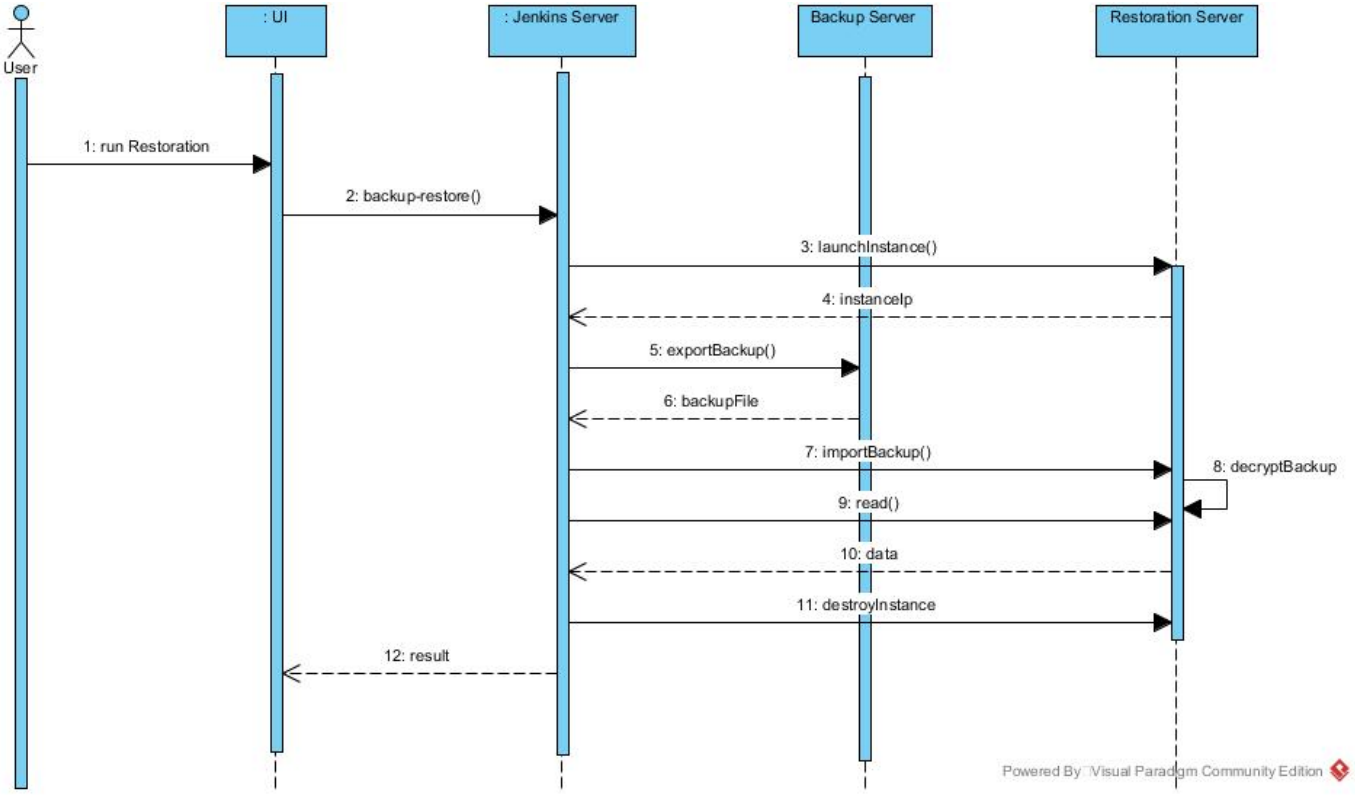


Figure 3: Schedule Regular Restore

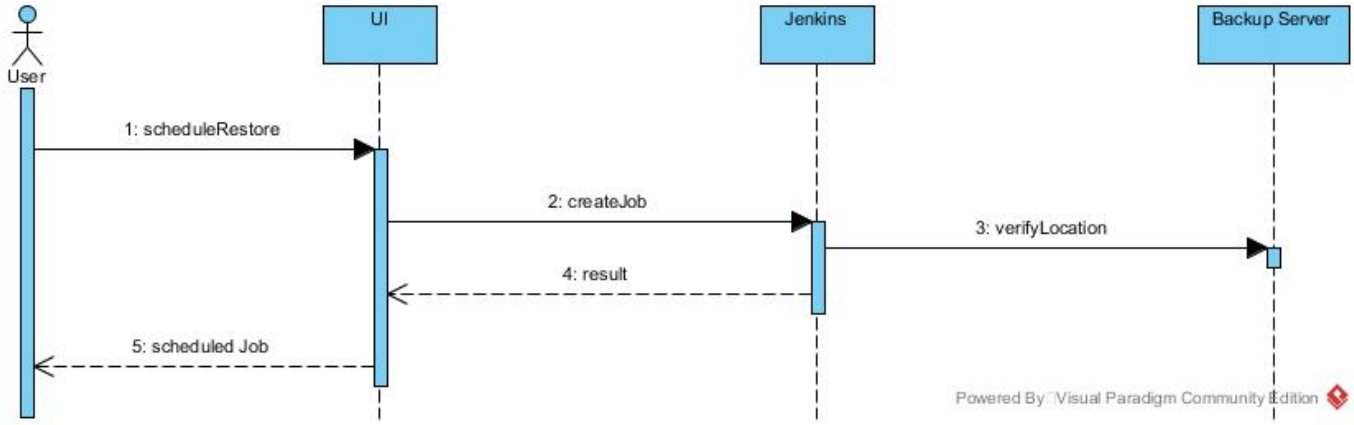
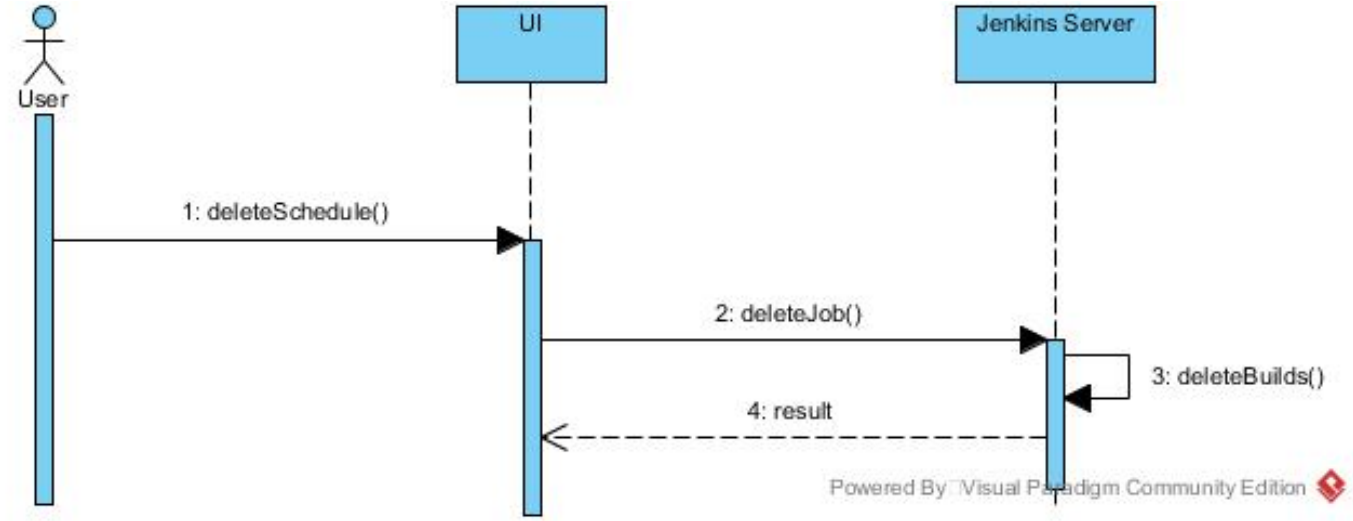


Figure 4: Delete Scheduled Restore



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

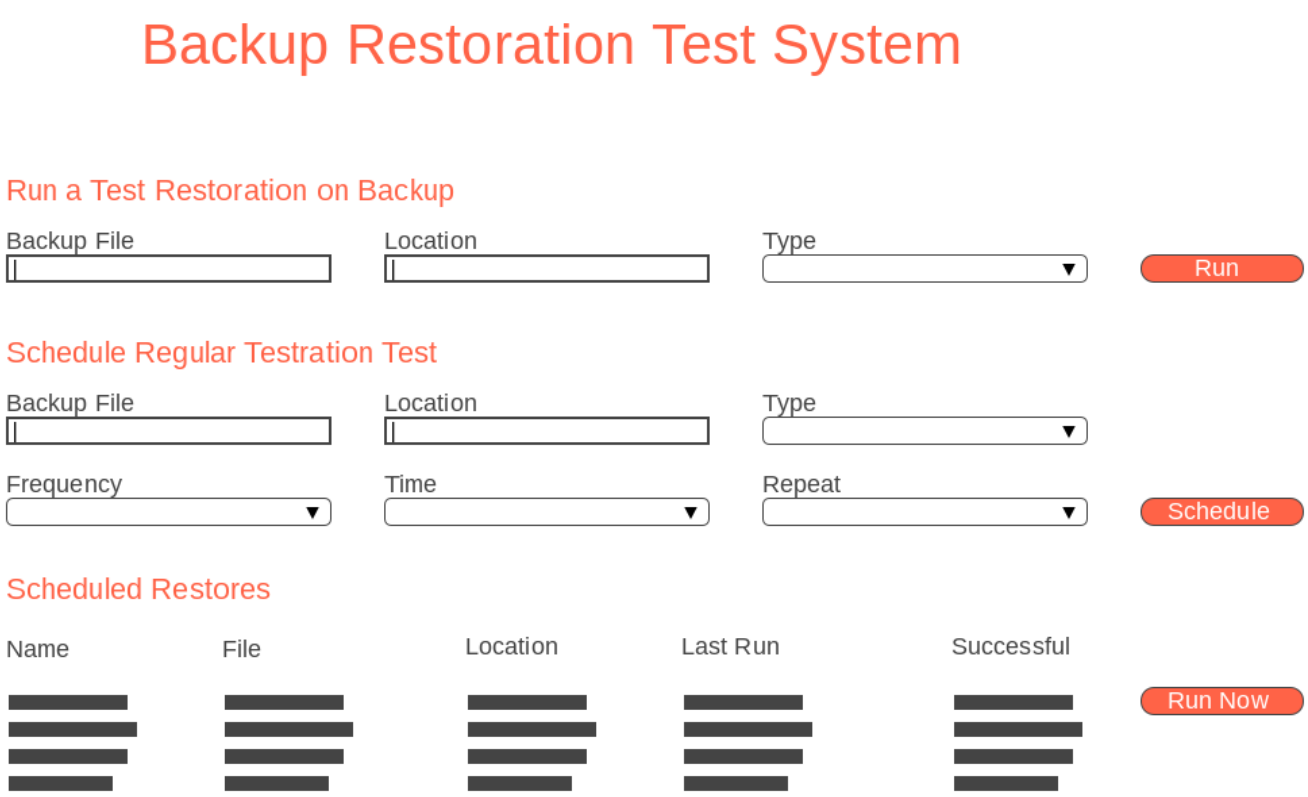


Figure 6: Scheduled Restore

4.4 Enterprise Level Consideration

TODO

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 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 it's 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 as the in the 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 though 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 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) focus on the progress of the current sprint ScrimInc (2017).

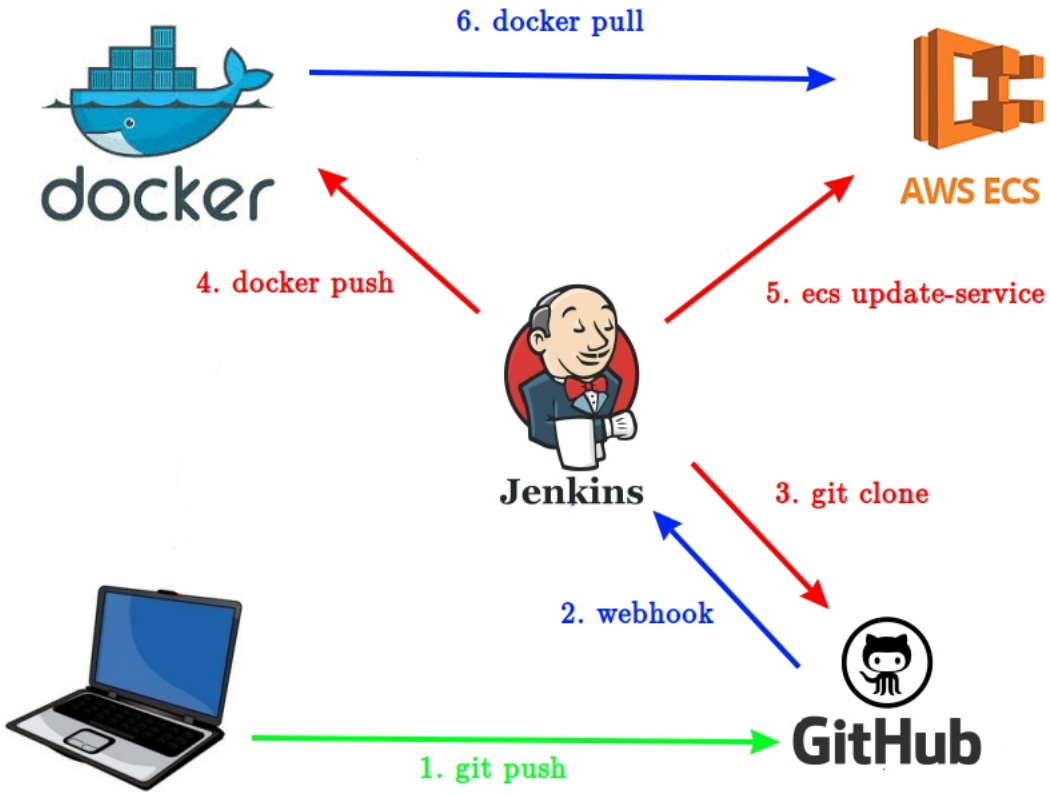
5.3 Continous Integration/Deployment

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 push 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 Code Quaity/Coverage

TODO

5.5 Documentation

TODO

5.6 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 sprint 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 were a Jenkins build server to continuously build and deploy the code and SonarQube to test code coverage and quality. Given the agile developments methodology 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;
- On account on SonarCloud was created, a cloud based service of SonarQube.

6.2 Sprint n

6.3 Sprint Metrics

TODO

7 Summary

7.1 Review

TODO

7.2 Learning Outcomes

TODO

7.3 Project Directions

TODO

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.

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/.

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/.