Backup Management and Orchestration System
Final Year Project

Rob Shelly
Applied Computing
20068406

November 29, 2017

Contents

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Aims & Objectives . . . 1

2 Technologies **2**
2.1 Docker . . . 2
2.2 Amazon Web Services . . . 2
2.3 Jenkins . . . 2
2.4 Node . . . 3
2.5 React . . . 3

3 Design **4**
3.1 System Architecture Overview . . . 4
3.2 Formal Modelling . . . 5
3.2.1 Sequence Diagrams . . . 5
3.2.2 User Stories . . . 7
3.3 Front End Design . . . 8
3.3.1 Wireframes . . . 8

4 Methodology **10**
4.1 Agile . . . 10
4.2 CI/CD with Jenkins . . . 10
4.3 Tool 2 . . . 10
4.4 Tool n . . . 10
4.5 Testing Approach . . . 10

5 Implementation of Prototype **11**
5.1 Sprint 1 . . . 11
5.2 Sprint 2 . . . 11
5.3 Sprint 3 . . . 11

6 Bibliography **12**

1 Introduction

1.1 Problem Statement

In January 2017 GitLab suffered a data loss incident which was widely reported in media. It began with spammers targeting GitLab.com and culminated in an engineer erroneously deleting 300GB of PostgreSQL data in a production. The lost data included merger requests, users and comments (GitLab, 2017a). The bigger story was to come later however when it was realised that GitLabs backup process had failed silently. The backups did not exist, 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 achived by performing restoration on the backups. The main objectives of the system are as follows:

- Eliminate the mundane and time consuming task of backup testing by automating regular backup restorations and recording results;
- Catch silent failures of the backup procedure by notifying sysadmins of failed backups;
- 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;
- Performs 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 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 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 Technologies

2.1 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 modularity of software makes Docker images appealing for this project. It will allow testing various data base types (e.g. MongoDB, MySQL) through it's software agnostic feature, by deploying an image with the corresponding DBMS software.

2.2 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. More specifically the project will make use of two specific services; Elastic Compute Cloud (EC2) and EC2 Containers Service (ECS).

EC2 is Amazons compute services. It allows easy deployment and management of virtual compute resource within the cloud. The flexibility of operating systems, virtual machine (or instances and they are known in AWS) size of volume of storage make it ideal for this project (Amazon, 2017a).

ECS is Amazons container management service. It allows Docker images to be easily deployed to and run on EC2 instances without the need to install Docker on the instances. ECS takes care of much of the container management issues that would arise when deploying a services if implemented though Docker alone. This includes managing port mappings between container ports and host ports, ensuring all containers are accessible if necessary. There is no added cost for using ECS. i.e. the customer only pays for the EC2 instances (Amazon, 2017b).

ECS is an appealing platform for running Docker images for the following reasons.

- Images can be deployed on EC2 instances, meaning there is no need to install Docker on the instance, without any extra charge.
- Containers are created within the customers own EC2 instances, meaning they are not exposed to other AWS customers. The are secured by the same infrastructure created by the customer for their instances, for example Virtual Private Clouds (VPCs) and Security Groups.

2.3 Jenkins

Jenkins is an automated build server. It can be 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 service is also extensible through a library of plugins (Jenkins, 2017). One plugin which will be of particular interest to this project is the Pipeline plugin. It allows the creation of pipeline as code.This means that for this project, pipelines can be used to create EC2 instances and deploy the necessary Docker images using ECS in order to test backup. Utilising the API, this can be achieved through a user-friendly web based frontend for user who are not familiar with Jenkins.

2.4 Node

The frontend of the system will be designed using Node (also known as Node.js). Node is a JavaScript runtime environment for build network applications. It is light-weight and efficient framework through it's 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 through 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.

2.5 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 integrated easily with Node and other *npm* packages without the needs for refactoring. React builds UI's as a set of components, each managing and their own state and implementing their own render function. This allows fast and efficient of rendering as data changes as only components that are updated will be re-rendered.

3 Design

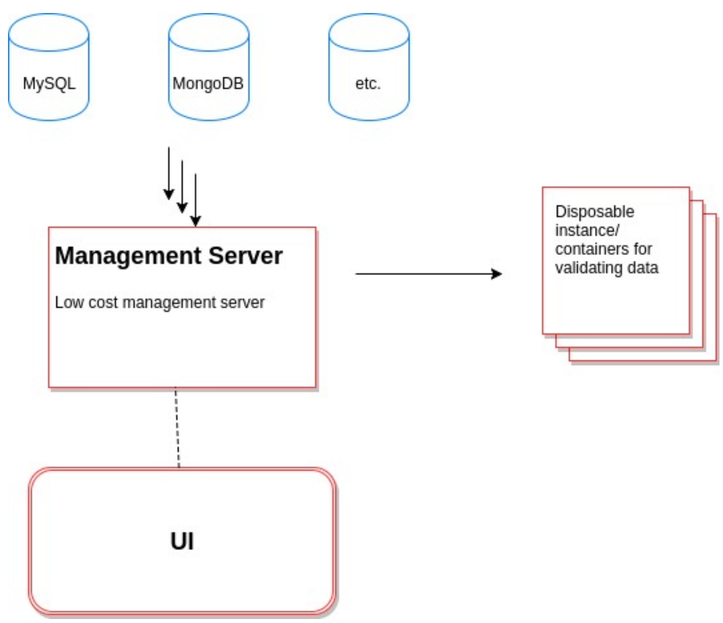
3.1 System Architecture Overview

The system will comprise of three main components:

- Management Server
- User Interface
- Disposable instances/containers

The system will also use existing infrastructure. This is where the backups are stored. Depending on the user of the system there may be multiple backup server in different location (such as AWS regions) or for different data types (relational and non-relational databases). Backup data may be stored in a variety of ways such as on EC2 instance or S3 buckets.

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 systems 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 it's location. The UI will utilise the Jenkins API to run execute the restoration with the parameters provided.

Disposable Instances or Containers: Disposable infrastructure will be used to perform the restoration. EC2 instances or containers can be used to quickly and easily deploy the necessary software to perform the restoration (i.e. the correct DB management system). They can also be destroyed afterwards, destroying the data and therefore maintaining confidentiality.

3.2 Formal Modelling

3.2.1 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. The trigger a Jenkins job automates the remaining steps. The backup is copied from the backup server to the test restoration server where it is imported to a Database Management System such as MongoDB. A read of the data is then performed to verify that the data is uncorrupted and readable. Finally it is detroyed from the restoration server.

Figure 2: Run Restore

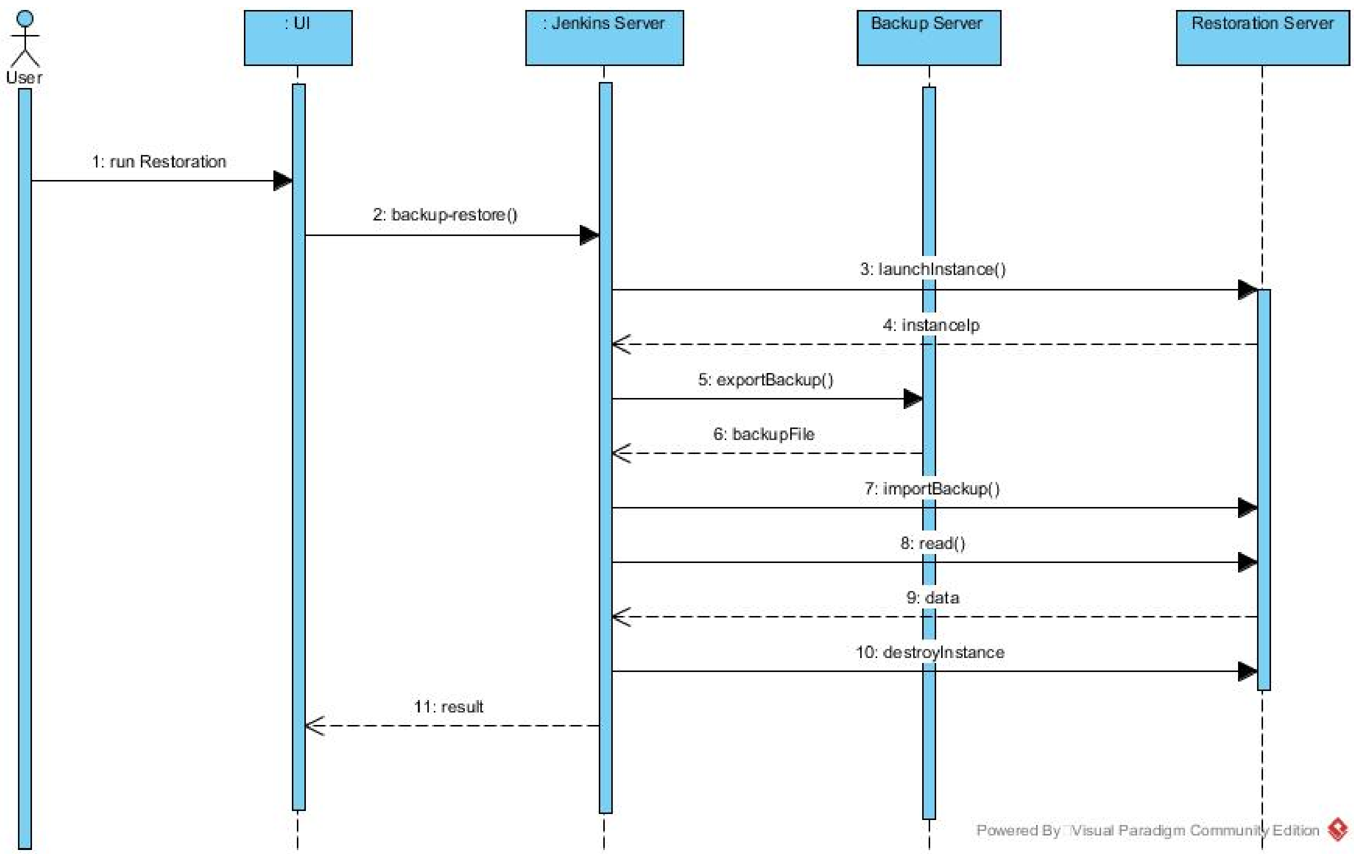


Figure 3 shows the process of a scheduling regular backup restoration tests. Again, this is triggered by a user from the web interface. The web interface will passes the JSON or xml configuration for a job to the Jenkins server. The server will then create and save the job. A status indicating whether the job was created successfully is returned to the user.

Figure 3: Schedule Regular Restore

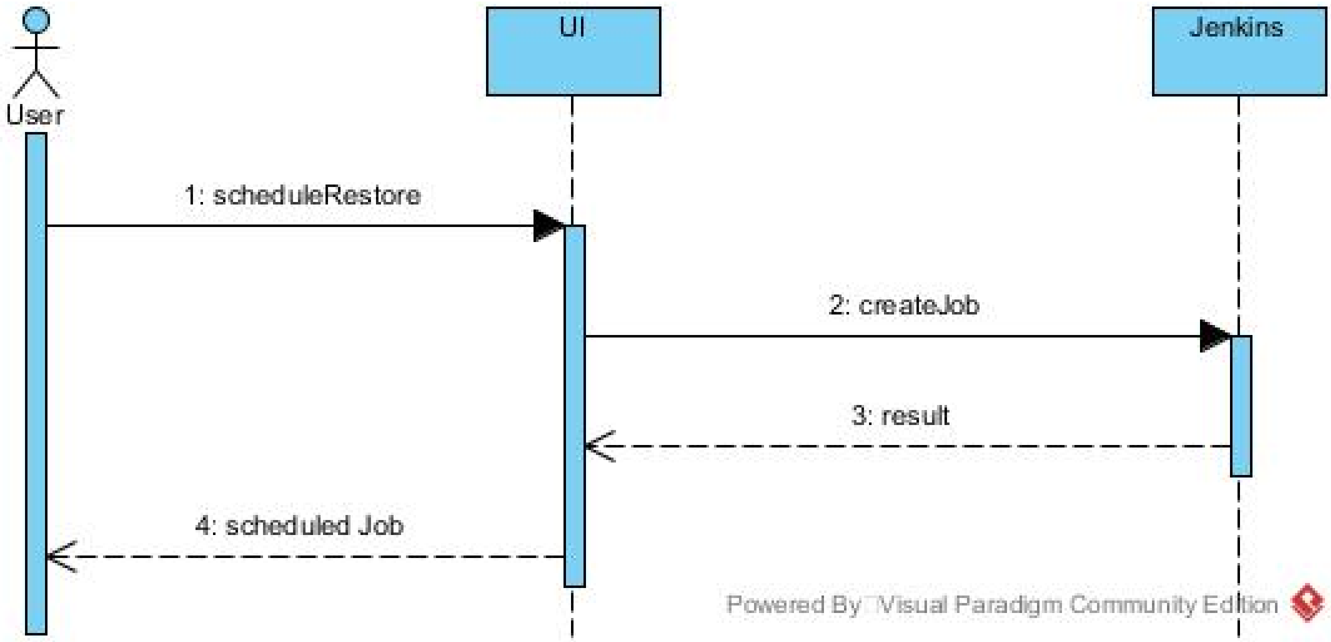
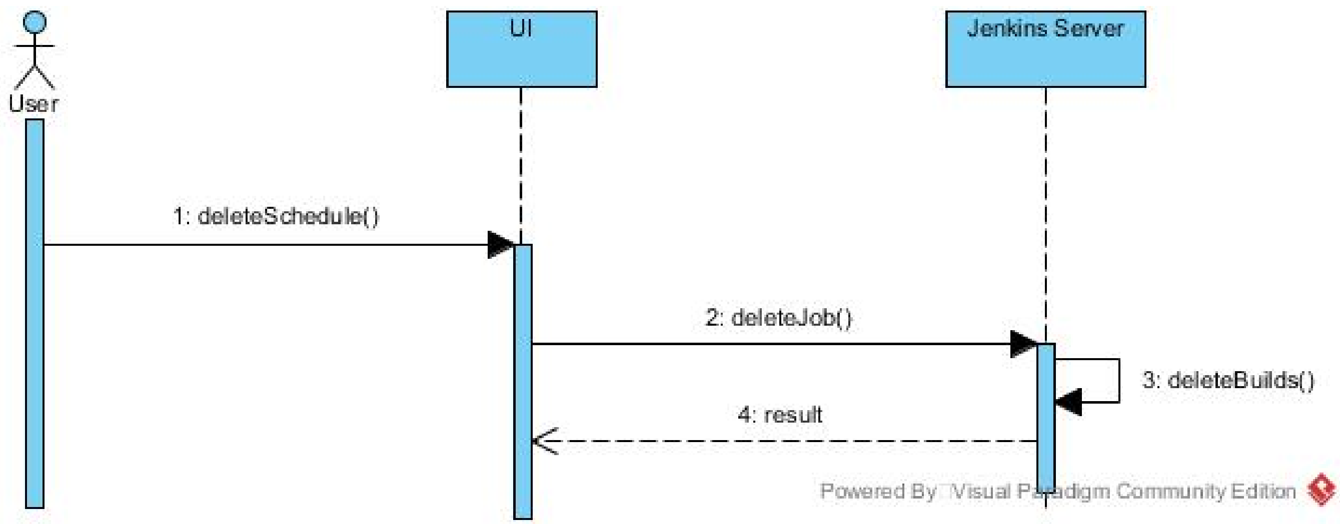


Figure 4 Show the process of deleting an existing scheduled job. This is required if a user not longer want to run scheduled restoration of a particular backup (for example if that backup is non longer needed and deleted). The user must delete the job on the Jenkins server in order to prevent further attempted restorations running. 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 4: Delete Scheduled Restore



3.2.2 User Stories

User stories are provided in Table 1. These the stories for general user such as running and scheduling restores. Also included are administration user stories. As system is create and destroys infrastructure on AWS, it would be necessary to limit access to the system.

Table 1: User Stories

As a	**I want to**	**so that**
manager	be able to add/remove my team members to the system	they can test backups
user	perform a restoration of a database backup	I can verify the backup restore process works
user	view the results of a backup restore	I can verify the backup contains valid, readable data
user	schedule a regular automated restoration of a particular backup	I don't have to manually do it myself
user	view the results of an automated restoration check	I can verify the backup contains valid, readable data
user	view past results of all automated checks	I can keep track of successful and unsuccessful restorations
user	be easily notified when a restoration fails	promptly address the issue a possible backup failures

3.3 Front End Design

3.3.1 Wireframes

Figure 5: Homepage

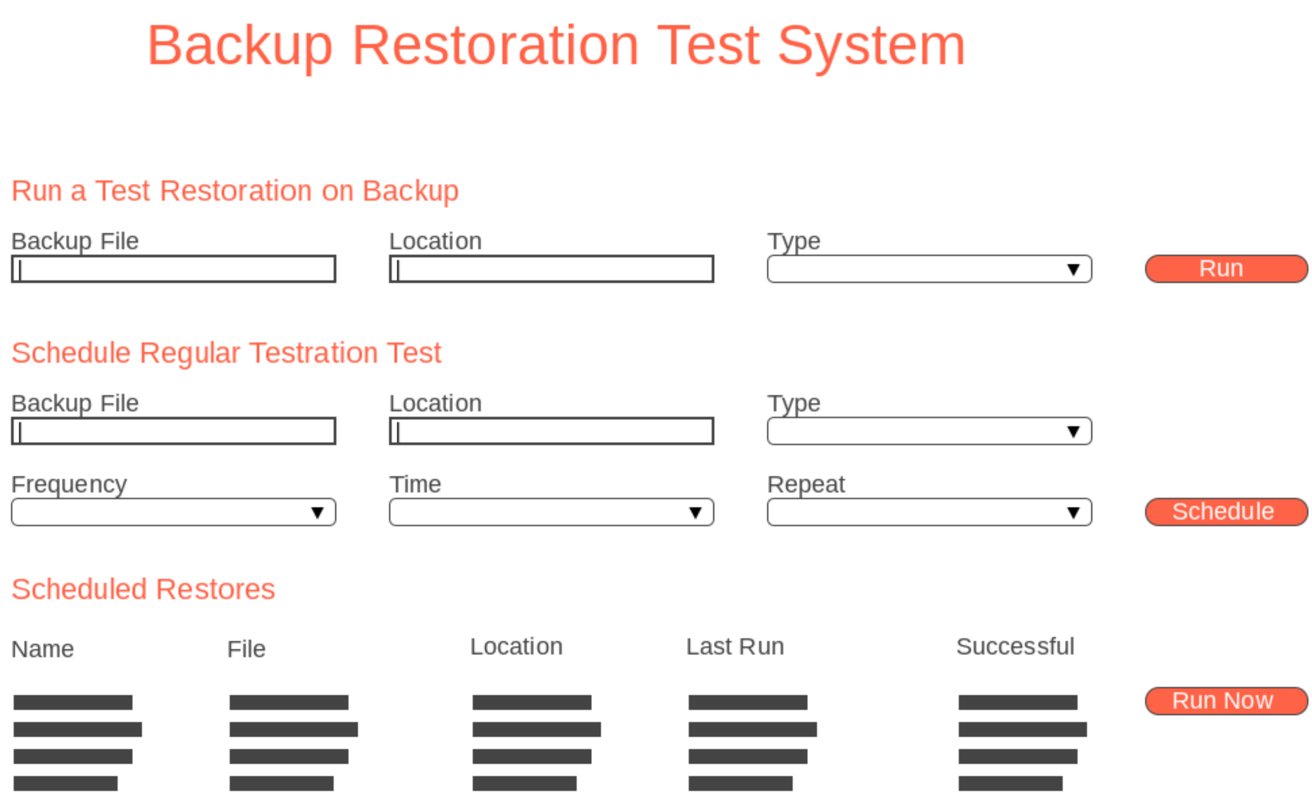
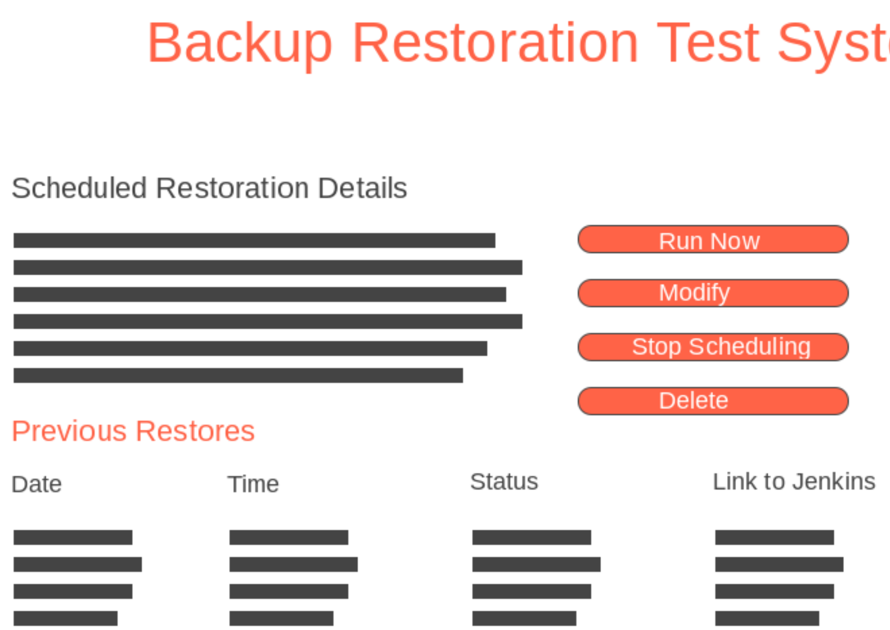


Figure 6: Scheduled Restore



4 Methodology

4.1 Agile

The design methodology chosen for this project is Agile. Agile takes an iterative approach toe designing and delivery products. It is a goal driven methodology that aims to build and deliver software in a iteratively and incrementally from the beginning of the project. This is in contrast with more traditional approaches such as Waterfall which deliver in one final stage. A notable aspect of Agile is user stories. The project is broken down is small sections of functionality which can be independently developed and delivers upon completion (Rasmusson, 2017).

A particular Agile framework which will be used for this project is Scrum. SCrum organises development in to cycles of *sprints*. A sprint consists short time-limited periods of development each with it's own development goals based on work within the backlog. Each sprint will consists of regular update and a final review/retrospective before beginning the next sprint. The Scrum master keeps the sprint focused on its development goal (ScrumAlliance, 2016).

Scrum is an ideal model for developing this project. The project supervisor plays a role in line with the concept of a Scrum master. Also, the use of user stories means sprints can be aligned with the implementation of stories.

4.2 CI/CD with Jenkins

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 tp 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 and often as code commits are performed. For example, running tests and building Docker images on each commit. IT means that code is prepared for release at each stage of development, instead of when it come to release time. This may occur often as many times a day (Ramos, 2016). Continuous Deployment is a step beyond Continuous Integration. After code is prepared for release, the built code is deployed to a server. However, this may be a development server. Pushing the built code to production required 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 Jenins automated build server. The fronted web app will be built aa s Docker image and deployed to ECS by Jenkins on every code commit.

4.3 JIRA

// TODO

4.4 Tool n

4.5 Testing Approach

//TODO

5 Implementation of Prototype

5.1 Sprint 1

5.2 Sprint 2

5.3 Sprint 3

6 Bibliography

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

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

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/.

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

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.

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/.