A Configuration Management and Monitoring Toolkit for EOS and Kinetic Drive Cluster

Megha Arora
CERN Openlab

Report submitted in fulfillment of the requirements
for the CERN Openlab Summer Student Programme
on 12th August 2016

Dep-Group-Section: IT-ST-AD

Supervisors
Andreas-J. Peters
Paul Lensing

The European Organization for Nuclear Research (CERN)
Geneva, Switzerland

Student's Declaration

I hereby declare that the work presented in the report entitled **"A Configuration Management and Monitoring Toolkit for EOS and Kinetic Drive Cluster"** submitted by me for the fulfillment of the requirements for the *CERN Openlab Summer Student Programme* at CERN, Geneva, is an authentic record of our work carried out under guidance of **Andreas-J. Peters** and **Paul Lensing**. Due acknowledgements have been given in the report to all material used.

Megha Arora
Megha Arora

Place & Date: Geneva, 12th August 2016

Abstract

EOS is a simple storage software solution to manage multi PB storage at CERN. Seagate kinetic drives have been interfaced with the CERN EOS disk storage system abstracting multiple drives as a cluster. Drive cluster are managed by EOS like a single storage mount point. Each drive cluster, access keys and the cluster layout is described in a JSON file. Monitoring EOS and managing configuration files for thousands of disks was inefficient. This project aimed at the development of a web toolkit for configuration management and monitoring of thousands of kinetic drives and EOS via a simple web interface. The proposed application allows to define, modify and publish kinetic drive cluster configurations and to monitor key statistics and individual drive parameters on demand. In this report, I cover how I went about building this toolkit.

Keywords: EOS, Configuration Management, Monitor, Kinetic Drives, AngularJS, Drive cluster, CERN storage, Seagate, Performance Optimization

Acknowledgments

I'd like to thank *Andreas-J. Peters* for his support, advice and patience throughout the project. His inputs on various aspects of the toolkit have helped me come up with better solutions to the problems I faced. The meetings and impromptu sessions with him were pivotal in battling with everyday roadblocks.

I'm also very grateful to *Paul Lensing* for being a very supportive supervisor for this project. His help and valuable inputs have helped me to make much of this summer. His constant encouragement has a major part to play in my work.

I would like to thank *Alberto Di Meglio, Kristina Gunne, Maria Girone, Sotirios Pavlou,* and *Maria-Athanasia Pachou* for giving me an opportunity to work on this project and helping me with all the administrative issues.

I would like to thank other members of the *IT-ST-AD group* for their guidance, support and appreciation that kept me going.

I would also like to thank the *open source contributors* of Angular, Bootstrap, Bower, NPM, and Git. This toolkit could not have reached it's current stage without these amazing technologies and their extensive and user-friendly documentation.

Lastly, I'm grateful to all *summer students at CERN* from whom I have learnt so much in the past eight weeks.

Contents

Chapter 1

Introduction

1.1 EOS & Kinetic Drives

EOS is a simple storage software solution to manage multi PB storage. Core of the implementation is the XRootD framework providing feature-rich remote access protocol. Default mode of operation is to store files with two replicas. Files can be accessed via native XRootD protocol, a POSIX-like FUSE client or HTTP(S) & WebDav protocol.

Seagate kinetic drive is one of the worlds first Ethernet-connected HDD with an open source object API. It reduces TCO by simplifying cloud and data center storage hardware and software stacks, and improves performance by eliminating layers of antiquated file system software. Seagate kinetic drives have been interfaced with the CERN EOS disk storage system abstracting multiple drives as a cluster.

1.2 Motivation

Kinetic Drive clusters are managed by EOS like a single storage mount point. Each drive cluster, access keys and the cluster layout (redundancy configuration, striping) is described in a JSON file. Managing these configuration files for thousands of disks is inefficient with manual file editing and a better configuration management tool is required. EOS is used to store a vast amount of information and monitoring the system through its CLI, which consumes data in the fuse format, is cumbersome.

1.3 Project Goal

This project aims at the development of a web toolkit for configuration management and monitoring of thousands of kinetic drives and EOS via a simple web interface. The proposed application allows to define, modify and publish kinetic drive cluster configurations and to monitor key EOS statistics and individual drive parameters on demand.

Chapter 2

Getting Started

2.1 Understanding EOS & Kinetic Drive Clusters

Before the project kicked off, I was made accustomed to the CERN way of software development. I was given access to personal virtual machines and I spent some time on learning how to use them. I was given access to an EOS instance to play around with and understand how it works. Gradually, I began to appreciate the problems with the CLI. Unparalleled importance has been given to well documented code and an efficient work flow throughout the development of the application.

2.2 Requirement gathering & analysis

I began by determining expectations and the end goals of the application. Starting off with ambitious checkpoints, I decided on quantifiable, relevant and detailed expectations.

System architecture was of the utmost priority. Several brainstorming sessions went into deciding how the toolkit should be designed and which framework should be used. Most of the technologies I shortlisted for the toolkit were new to me. Despite the steep learning curve, I went ahead with these technologies to have a fruitful experience.

Selection of technologies and the reason behind it:

- ***GitHub***: It is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.
- ***AngularJS***[1]: It is a Google project and has been designed from ground up. MVC architecture, service providers and context aware communication were the primary reason for using this technology. MVC architecture made it convenient for us to have a data binding

[1] https://angularjs.org/

layer. The appropriate use of service providers allowed easy communication with the EOS API. AngularJS is single page web application framework which only loads those bits from server which have been updated and not the entire page. Due to its inherent architecture it increases the performance significantly.

It is popularly known to have a steep learning curve. With dependency injections, objects and MVC architecture introduced into Javascript, I had a challenging time beginning with AngularJS. But after getting a grip on Angular basics, I could appreciate its expressive, readable, and quick nature.

- ***Bootstrap***: Responsive, customisable and easy to use frontend components for faster development.
- ***Bower and NPM***: I required hundreds of libraries, assets and utilities for the web application for which I needed a package manager. Bower and NPM both have their own advantages. NPM being a javascript specific package manager had a larger library than Bower. Bower is optimised for the frontend with a tree dependency structure.
- ***Grunt***: To reduce repetitive tasks like minification, compilation, unit testing and linting, I configured a task runner to automate these processes. All the mundane work was done by Grunt including deploying the code using Githooks.
- ***HTML5 & LESS***: With cross browser support, accessibility and an easy to use local storage feature i chose these two technologies for the front end. LESS is a pre-processor for CSS which adds additional functionality to stylesheets and gets the same task done in lesser lines of code as compared to CSS.
- ***jQuery***: jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. It is the most popular JavaScript library in use today. Its syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications.
- ***Node.js***: It is an open-source, cross-platform runtime environment for developing server-side web applications. Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input-output operations, as well as for real-time web applications.

After setting up the development environment on my local and virtual machine, I finalised the project structure with my supervisors after a few iterations.

Chapter 3

Building the Toolkit

3.1 Modifying the API

The first step was to modify the response format for the EOS API. Earlier the response was just in fuse format (containing standard error and standard output), which is accessible through the CLI. For AngularJS, I needed a JSON callback object to be returned as a response. The update is shown in Figure 3.1.

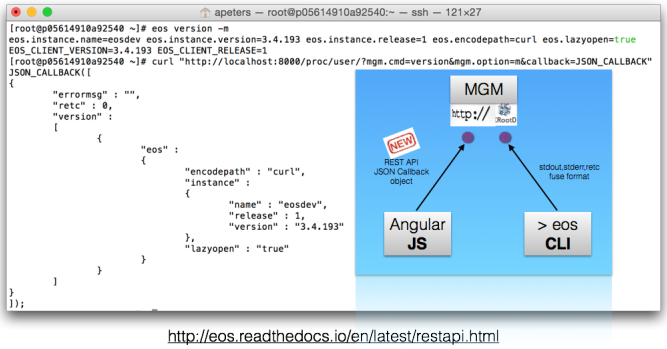

Figure 3.1: API Response - Before & After

3.2 System Design

The toolkit has been developed while following the *agile software development* methodology whereby the requirements and solutions evolve through collaboration. It promotes adaptive planning.

I decided on the basic features and requirements. The application is based on user centered design and has three major segments:

3.2.1 The Dashboard

Since it is a monitoring toolkit, a Dashboard was required for better and faster tracking. It cuts down the clutter to get to the essentials. It displays key parameters and the most frequently checked information. This was one of the primary motivations for the toolkit over the existing CLI.

The dashboard not only displays critical information, but also allows the user to modify the status of the converter, quota, and balancers. It offers a live view, which updates every second, including time-related statistics like uptime and execution time latency. Apart from memory information, the dashboard also has four tachometers displaying percentage of active groups in default space, space used, network IO in, and network IO out. Figure 3.2 shows a snapshot of the dashboard screen.

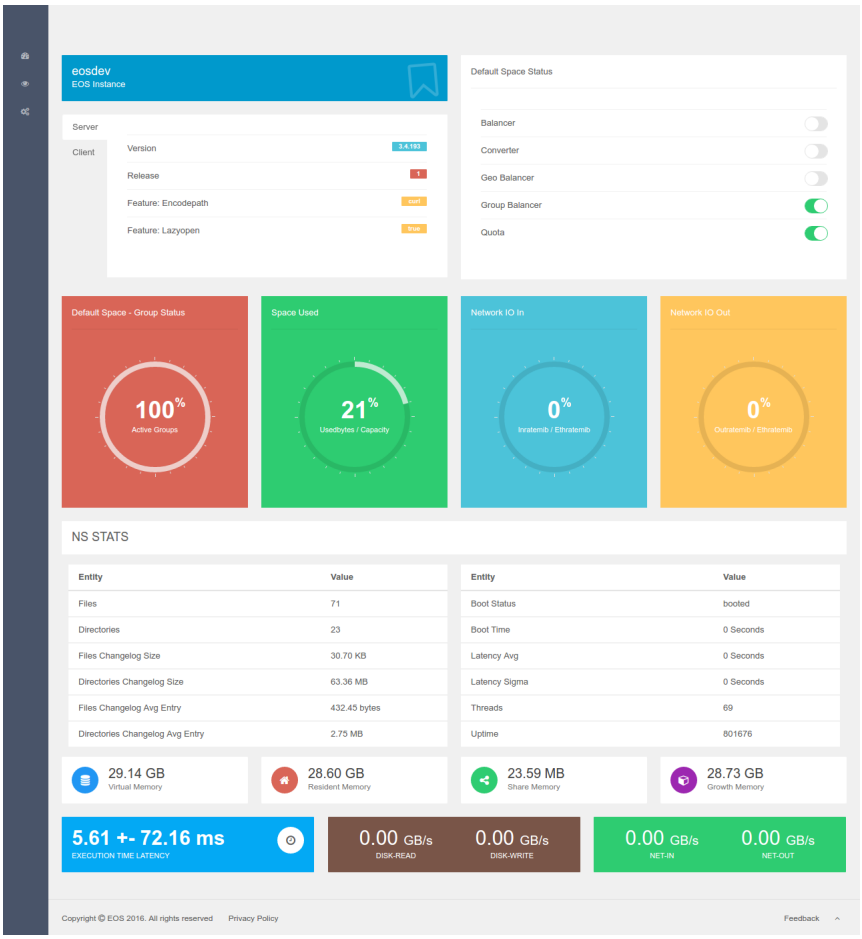

Figure 3.2: The Dashboard

3.2.2 Different Views

The second segment of the application was to display the four views which EOS offers, namely: group, space, file system, and node view. These are essentially well-formatted and accessible

outputs from group ls (Figure 3.3), space ls (Figure 3.4), fs ls, and node ls commands in the CLI.

The data tables also support a search functionality, which does string matching with all the attributes in the row. The tables can be sorted on the basis of any of the columns simply by clicking on the column heading. I have also paginated the views to accommodate large amount of rows, so that the view does not get cluttered as it happens in the CLI. These features also help in searching for particular entries the user might be interested in, for instance, finding inactive groups in the group view.

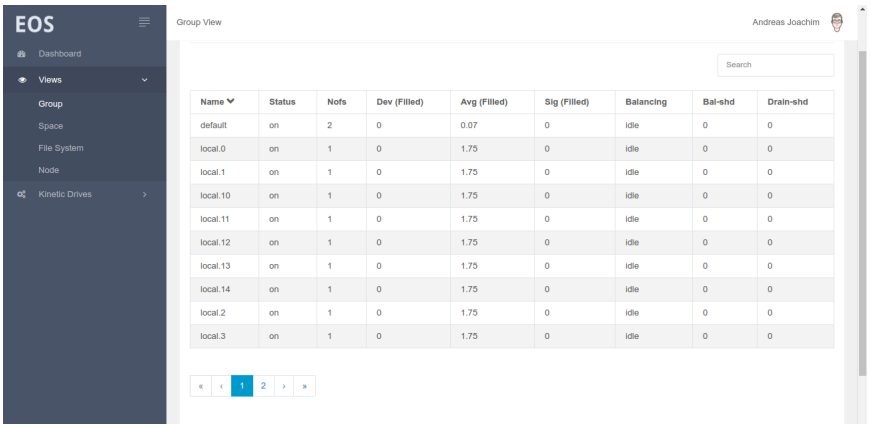

Figure 3.3: Group View

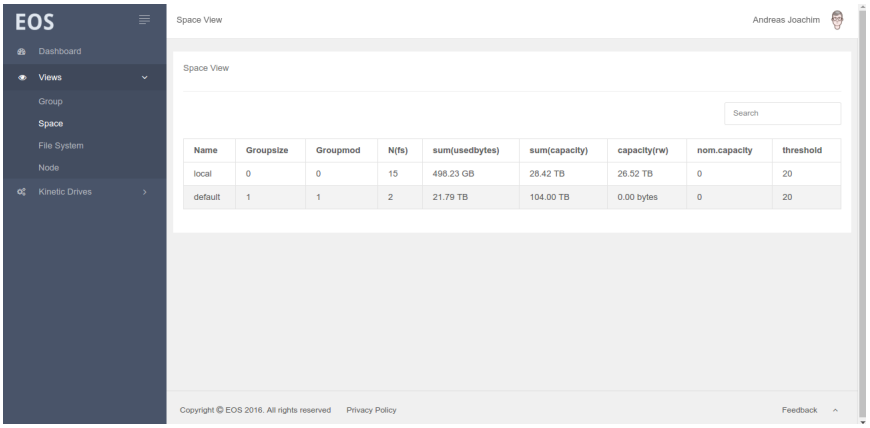

Figure 3.4: Space View

3.2.3 Kinetic Drive Clusters

This view was particularly tricky. Every cluster is associated with a cluster definition, a location definition and a security definition. These are stored in individual JSON files. I wanted the user to be able to add new clusters and drives, modify existing ones, update security settings individually and in bulk, and publish all these changes at the same time. On top of this, every cluster definition belongs to a space. I didn't want the user to select the space multiple times, nor did I want the space to be set in a separate settings option, increasing the number of clicks for

the user. Figure 3.5 shows the final interface for the cluster configuration. The flow is intuitive and supports all the functionalities possible. Subsequent figures show how these functionalities can be accessed through modals and other UI elements.

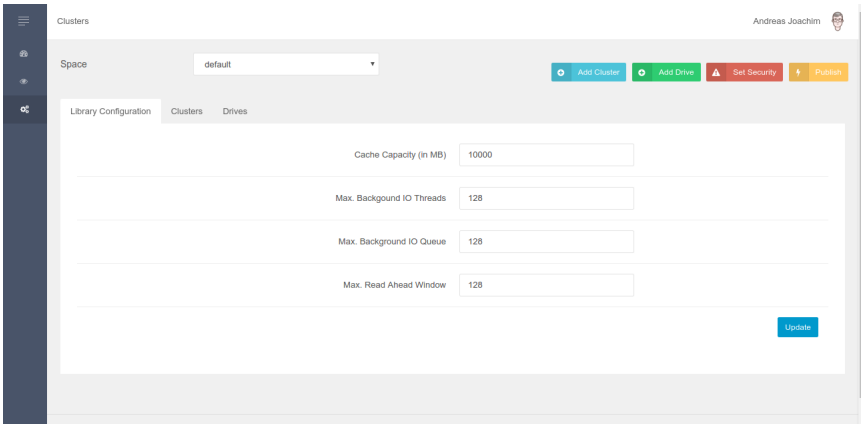

Figure 3.5: Single view for kinetic drive cluster management

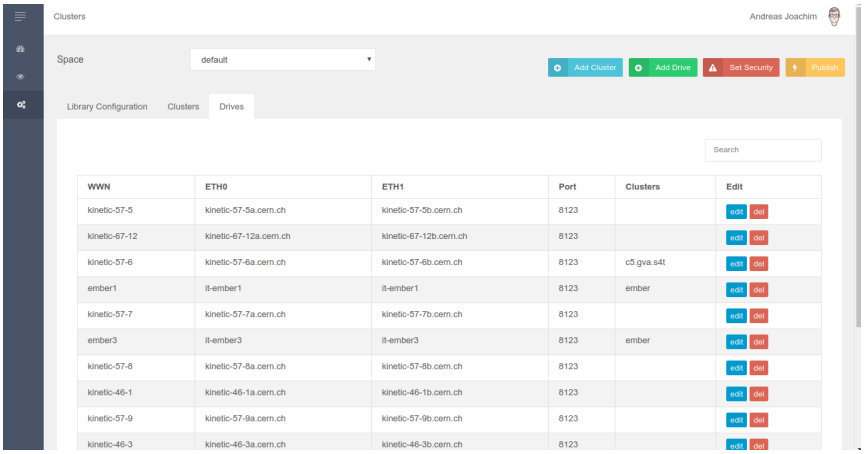

Figure 3.6: User can view all cluster and drive information

3.3 Navigation

The application has a simple and stylish vertical navigation bar on the left side of the screen. The navigation drawer can be closed to enlarge the view. No breadcrumbs have been used in the application as the sidebar shows where the user is currently. Only one sub-menu is open at a time to ensure that the user is not overwhelmed with options.

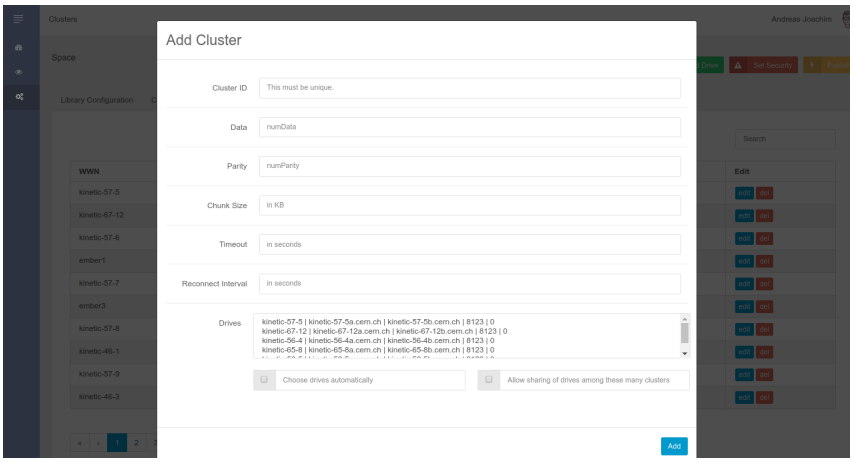

Figure 3.7: User can add new clusters and drives, and attach drives to clusters

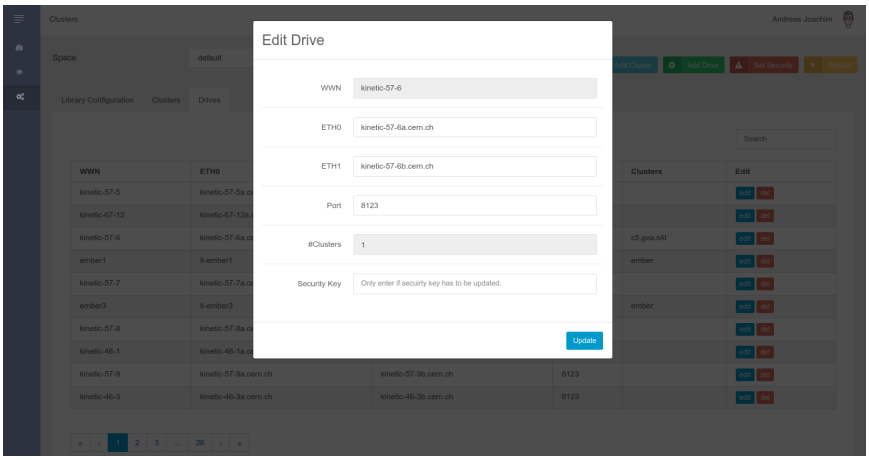

Figure 3.8: User can modify cluster and drive information

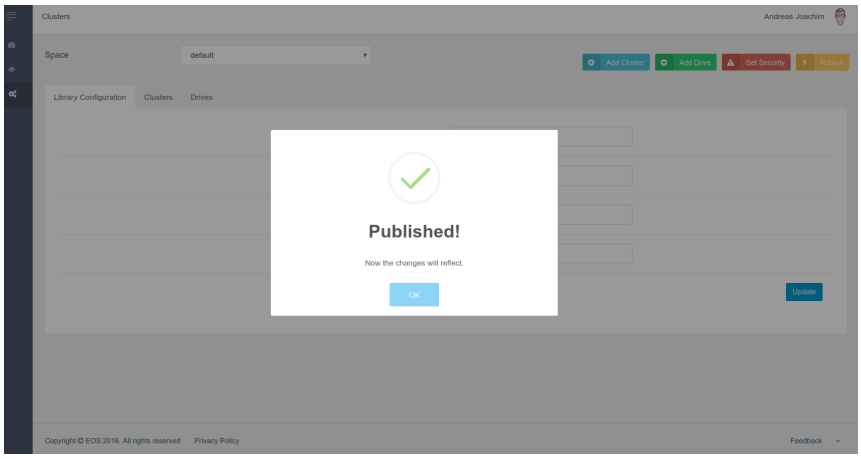

Figure 3.9: User is shown alerts when a change is made or successfully published

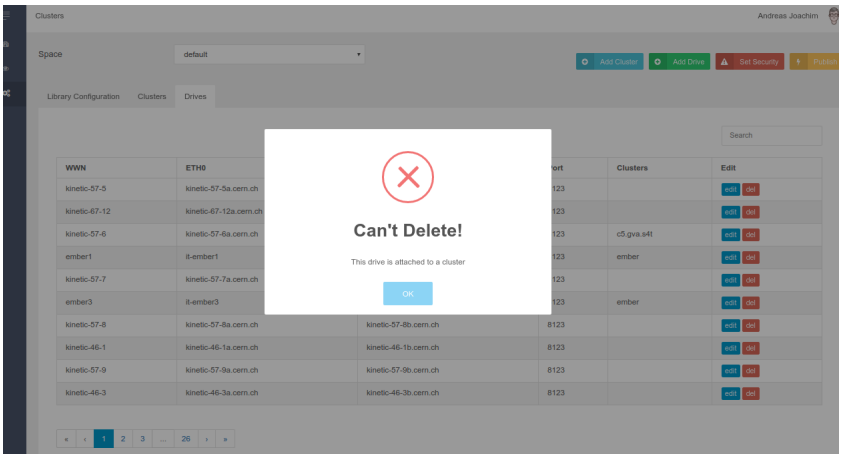

Figure 3.10: Check is performed before implementing security-critical tasks like deletion

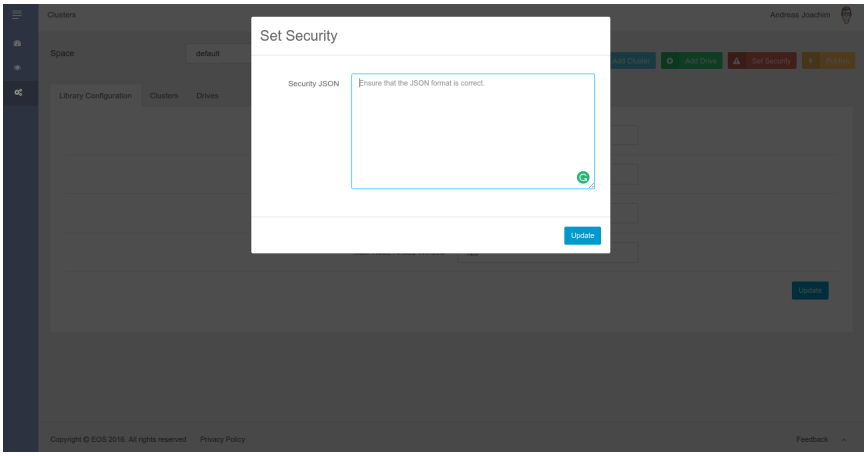

Figure 3.11: Updates can also be performed in bulk by uploading the updated JSONs

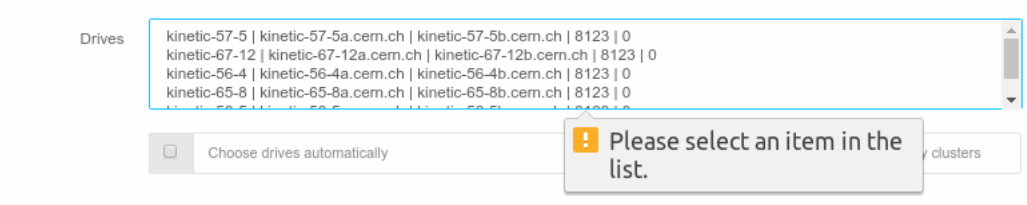

Figure 3.12: All form information is validated before submission

Chapter 4

Optimizations & Following Best Practices

4.1 Optimizing the Application

4.1.1 UI Router

Instead of the default routing mechanism that comes along with Angular core, the application uses UI Router. UI Router uses states instead of the site URL. This enabled me to configure all the routes in one place instead of having them spread out in the application. UI Router supports multiple views, which is a requirement for the toolkit. Multiple views is used to divide the screen into several smaller views, each view can have its own controller.

```
.config(['$stateProvider', '$urlRouterProvider',
  function ($stateProvider, $urlRouterProvider) {

    // For unmatched routes
    $urlRouterProvider.otherwise('/');

    // Application routes
    $stateProvider
        .state('app', {
          abstract: true,
          templateUrl: 'views/common/layout.html',
        })
```

Figure 4.1: UI Router configuration file

4.1.2 Organized MVC directory structure

I developed a skeleton MVC Directory to group the models, views and controllers together for faster navigation and easier development.

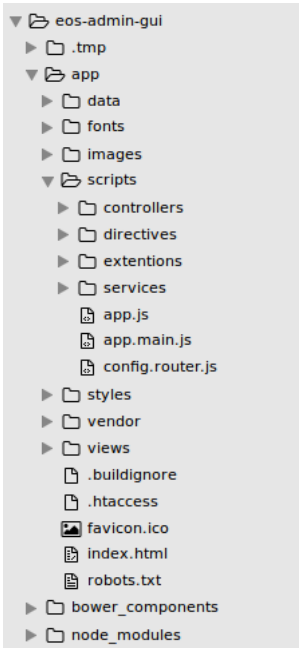

Figure 4.2: Directory structure of the application

4.1.3 Code Modularity

I divided the logic into different modules. Following the same concept of grouping features together allowed scalability and reusing modules across the application.

4.1.4 Minification Safe

I have used explicit dependency injections to make the entire application minification safe for production deployment. All the dependencies have been declared using array style notation. Despite the extra code that needs to be added for explicit dependency injection, it makes it possible to integrate the application into the build process using Grunt.

```
angular.module('app').controller('ModalDemoCtrl', ['$scope', '$state', '$modal', '$log
    ', 'eosService', ModalDemoCtrl])
```

So this way even if the minifier converts $scope to variable 'a' and $state to variable 'b', their identity is still preserved in the strings.

4.1.5 Responsive Design

Responsive design is designing web applications in such a way that it provides optimal viewing experience across all platforms, irrespective of their screen sizes. All the design components have been developed or picked in such a way that they are responsive and can be viewed on a screen of any size.

4.1.6 Global Dependencies

Global dependencies have been declared in a single file and are made available across the app using Grunt.

4.1.7 Avoiding controller bloating

The application uses providers (services and factories) extensively to avoid controller-bloating. Controllers are only bartering between view and data model, and are lean.

4.1.8 Local storage

Instead of cookies, I have used browser's local storage, which has better performance. Even if the cookies are disabled in the user's browser, the application data is unaffected. Local storage can store data in MBs whereas cookies can only be used to store KBs.

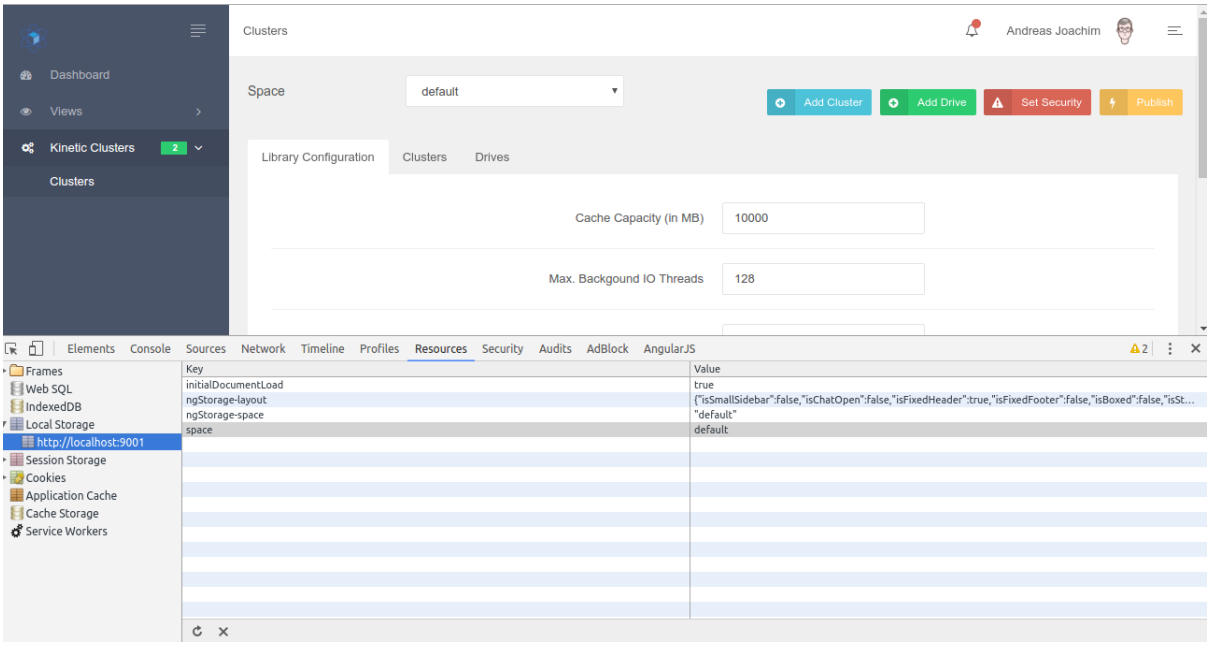

Figure 4.3: Using browser local storage to save the selected space

4.1.9 Assets Optimization

- ***Concatenation***: Using Grunt I defined a task to concatenate all the CSS, LESS and JS files into a single file for optimization. This reduced the page load time significantly and improved the performance on the client side. Using Grunt I have built the application in such a way that all the assets (JS files, CSS, images, icons, font etc) got placed in single directory for easy deployment and better performance.

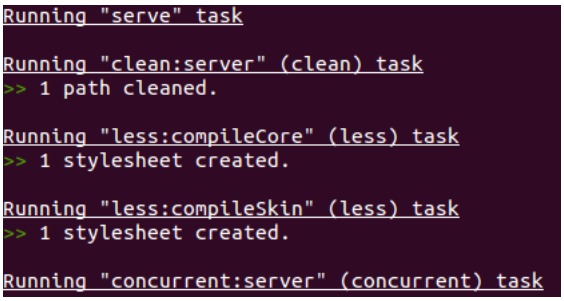

Figure 4.4: Grunt task running to compile all Javascript and CSS files into single files

- ***Compression & Minification***: All of the assets were minified using grunt to improve performance.

Chapter 5

Conclusion

EOS and Seagate kinetic drives are integral to the storage systems at CERN. In this project, I started with building an essential platform for monitoring and configuring them. I was successfully able to complete the project, which will benefit the EOS team members who work with kinetic drive clusters and monitor EOS operations. This summer has not only been challenging but has also taught me important lessons in professionalism and managing responsibility. The time spent at CERN has definitely contributed to my goal of becoming an accomplished computer scientist. Ironically my most productive days at work have been the ones where I have removed hundreds of lines of code. Never more have I appreciated the importance of writing lesser code and building human-centered applications. I would like to summarize my entire learning at CERN by quoting David Kadavy:

'The user experience design of a product essentially lies between the intentions of the product and the characteristics of your user.'