Designing a Virtual Security Layer for Cloud Content

Lukas Klingsbo

October 15, 2015

Abstract

TODO: Abstract

Keywords:

Contents

1 Introduction 1

2 Related Terminology 2

2.1 Technologies 2

2.1.1 React 2

2.1.2 Flux 2

2.1.3 Scala 2

2.1.4 REST 2

2.1.5 MongoDB 2

2.2 Abbreviations 2

2.2.1 JPF 2

2.2.2 CDN 2

3 Related Work 3

3.1 Copy-on-Write 3

4 Background 3

4.1 About Uprise 3

4.2 The current system 3

4.3 Problem description 3

5 Model 5

5.1 Entities 5

5.1.1 Content 5

5.1.2 Project 5

5.1.3 Snapshot 5

5.1.4 Container 6

5.1.5 File 6

5.1.6 User 6

5.2 JPF 6

6 Copy-on-Write 8

7 Methods for determining implementation details 8

8 Security of the system 8

8.1 Authorization 8

8.2 Audit logs 8

9 Resulting system **8**
9.1 Persistent storage 8
9.1.1 MongoDB 8
9.2 API 8
9.2.1 REST Endpoints 9
9.3 Scalability 10

10 Discussion **10**

11 Summary **10**
11.1 Conclusions 10
11.2 Future work 10

1 Introduction

Developing larger projects containing static content usually involves using a Content Distribution Network to be able to scale to a large user base. The commercial Content Distribution Networks are usually fairly easy to use, the content that is to be used in a project is usually simply uploaded and then directly available to the public. For secret content this can be a problem and that is what this thesis is about. This work examines ways of enforcing access control on content and groups of content in the form of snapshots. A system was developed to make the underlying theory work in practice.

2 Related Terminology

2.1 Technologies

2.1.1 React

React is a JavaScript library for building user interfaces. React uses both its own virtual DOM and the browser's, this makes it able to efficiently update dynamic web pages after a change of state through comparing the old virtual DOM with the resulting virtual DOM after the state change and then only update the browser's DOM according to the difference between the virtual DOMs [1].

2.1.2 Flux

2.1.3 Scala

Scala is a multi-paradigm programming language. It most commonly runs on the JVM and compared to Java it supports most functional programming features at the same time as it supports object oriented programming [2].

2.1.4 REST

2.1.5 MongoDB

TODO: Write down related terminology, if any

2.2 Abbreviations

2.2.1 JPF

Java Path Finder - It was developed by NASA and in 2005 they released it under an open source licence, which made more people contribute to the project. JPF is usually used for doing model checking of concurrent programs to easily find for example race conditions and dead locks.

2.2.2 CDN

Content Distribution/Delivery Network - Replicates content to several servers, usually spread out geographically. Once a request is made, the network serves content from the server closest to the requester.

3 Related Work

3.1 Copy-on-Write

This work relies heavily on the Copy-on-Write principle which was founded and used in the Mach kernel [3].

Copy-on-Write is used in virtual memory management systems [4], snapshot functionality in both file systems [5] and logical volume management systems [6], and as an optimisation technique for objects and types in programming languages [7].

Its principle is that when processes share data in between each other, the data is not copied until one of the processes does changes to it. This is an optimisation as the processes does not have to send all of the related data that is in memory, rather they only have to send pointers to the data. After many Copy-on-Write's a complex structure can be built up, but it is possible to solve that structure [8].

4 Background

4.1 About Uprise

Uprise is a company based in Uppsala, Sweden.

4.2 The current system

Today a system called battlebinary [9] is used for managing and uploading files, mostly images, to content delivery networks. The current system does not make use out of the security features that the CDN's are offering, instead it uses a form of security by obscurity. When a file is uploaded to a CDN it is open for the public, but its filename is composed out of its original filename concatenated with a part of the MD5 hash of the content of the file, which makes it an extremely hard process to access the file on the CDN without access to the original file or a reference to the URI.

In the current system you can only upload a file once as there will be a collision in the upload otherwise, as the old and the new file will have the same MD5 hash.

4.3 Problem description

As the current system does not offer proper security measurements, is lacking a lot of features that is needed and does not scale very well, a new system should be developed. This work is about examining a way of implementing Copy-on-Write in a high level system, which should solve the scalability problem and makes it possible to implement the wanted features, like snapshots and concurrent modifications of content.

What to write about Battle Binary Security features of CDN's

What is needed * Security layers * snapshots * Virtual file structure * Versioning of content * Multi project support * Auth and audit logs * Users

5 Model

5.1 Entities

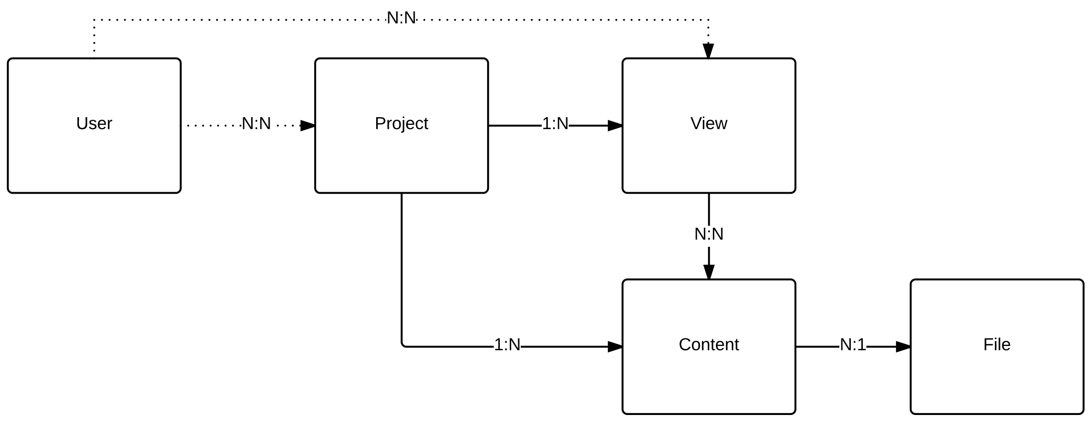

Figure 1: Entity Relationships

5.1.1 Content

Content is an asset contained in a project and/or snapshot, it is a form of virtual file. It could be described as a link between the snapshot or project and the real file. The content can be for example an image, video or binary blob, combined with meta-data.

5.1.2 Project

A project is what is created to contain all content related to a real project. Files can be changed within a project and the system can contain several projects and their virtual content are completely disjoint. snapshot

5.1.3 Snapshot

A snapshot is a subset of content, in a project, from the state which they were in when the snapshot was created. A project can contain many snapshots. A snapshot can have content which either is automatically dependant on its parent project and updated when the parent is updated or stay in the state which it was in when the snapshot was created. This results in that the content within a snapshot can not be updated independently from its project.

5.1.4 Container

A container is a virtual folder within the projects which contains content and other containers.

5.1.5 File

A file refers to an actual physical file in the file system.

5.1.6 User

A user is the structure that handles people who have been granted access to the system. Access to the system is handled by a separate service, like LDAP.

5.2 JPF

Java path finder was used to show that the model and plan of how to build the system was sound. The model was built in Java with the objective of being as reduced and simple as possible, without loosing any of the cases that needs to be covered by the model checker. As the users are mainly going to be handled by external systems they were not included in the model.

Each collection in the persistent storage was emulated by using the built-in ConcurrentHashMap type. Each client was represented by a thread and each action taken by the client was randomised. The id hashes which MongoDB is using for each entity was imported from the mongo-java-driver-2.13.3 and each object had its own id, generated in the same fashion as the real implementation is using, randomly generated by the ObjectId class to minimise collisions that is. Further more no locking or transactions were used and the threads were running fully concurrently, without any sleep statements.

ConcurrentHashMap had to be used in favour of the normal HashMap as the normal HashMaps can't be iterated over concurrently.

JPF checked each permutation of states that the threads can end up in, the result of the run can be seen in Listing 1.

Listing 1: Results of JPF run

```
elapsed time:        14:26:53
states:              new=160853259,
                     visited=451102505,
                     backtracked=611955764,
                     end=21640
search:              maxDepth=380,
                     constraints=0
```

```
choice generators:  thread=160853255
                    (signal=0,
                    lock=3603938,
                    sharedRef=146989208,
                    threadApi=3,
                    reschedule=10260106),
                    data=0

heap:               new=676056850,
                    released=435060996,
                    maxLive=655,
                    gcCycles=523950061

instructions:       11917045758
max memory:         6256MB
loaded code:        classes=111,
                    methods=2179
```

6 Copy-on-Write

As the persistent storage does not implement transactions or locks a lot of different problems can occur when several clients are working on the same data set at the same time. Such problems could be race conditions,

7 Methods for determining implementation details

This chapter introduces the different methods used to determine how the new system should be implemented, which DBMS it should use and how the estimation of long term scaling was done.

8 Security of the system

8.1 Authorization

8.2 Audit logs

9 Resulting system

9.1 Persistent storage

9.1.1 MongoDB

MongoDB was chosen as the persistent storage because of its internal storage format called BSON, which is very similar to JSON which the API is using. As the formats are similar, the process of marshalling and unmarshalling becomes quite easy between the core code, MongoDB instance and REST interface. The second reason was that if the system needs to scale in the future it is very easy to distribute MongoDB and if needed the system can easily be migrated to Reactive Mongo, which is asynchronous and non-blocking and can therefore scale even further [10].

9.2 API

REST was chosen as the BSON format which is used in MongoDB is almost identical [11] to the standardised JSON format which is used by RESTful services [12].

9.2.1 REST Endpoints

For the frontend to communicate with the backend REST is used, the following endpoints were configured:

- projects

  GET - list all projects

  POST - create new project

- projects/{id}

  GET - get specific project

  PUT - update existing project

  DELETE - delete existing project

- projects/{id}/content

  GET - list all content in a specific project

  POST - create new content in a specific project

- projects/{id}/content/{id}

  GET - get specific content in a specific project

  PUT - update existing content in a specific project

  DELETE - delete existing content in a specific project

- projects/{id}/snapshots

  GET - list all snapshots in a specific project

  POST - create new snapshot in a specific project

- projects/{id}/snapshots/{id}

  GET - get specific snapshot in a specific project

  PUT - update existing snapshot

  DELETE - delete existing snapshot

- projects/{id}/snapshots/{id}/content

  GET - list all content in a specific snapshot

  POST - create new content in a specific snapshot

- projects/{id}/snapshots/{id}/content/{id}

  GET - get specific content in a specific snapshot

  PUT - update existing content in a specific snapshot

  DELETE - delete existing content in a specific snapshot
- projects/{id}/containers

  GET - list all containers in a specific project

  POST - create new container in a specific project
- projects/{id}/containers/{id}

  GET - get specific container in a specific project

  PUT - update existing container

  DELETE - delete existing container
- projects/{id}/containers/{id}/content

  GET - list all content in a specific container

  POST - create new content in a specific container
- projects/{id}/containers/{id}/content/{id}

  GET - get specific content in a specific container

  PUT - update existing content in a specific container

  DELETE - delete existing content in a specific container

9.3 Scalability

10 Discussion

11 Summary

11.1 Conclusions

11.2 Future work

Stuff to write about: Modular design, every piece should be interchangable LDAP - why it was used as standard AUTH

References

[1] A. React, “Javascript library for building user interfaces,” 2014.

[2] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger, “The scala language specification,” 2004.

[3] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, “Mach: A new kernel foundation for unix development,” 1986.

[4] J. M. Smith and G. Q. Maguire Jr, “Effects of copy-on-write memory management on the response time of unix fork operations,” *Computing Systems*, vol. 1, no. 3, pp. 255–278, 1988.

[5] O. Rodeh, J. Bacik, and C. Mason, “Btrfs: The linux b-tree filesystem,” *ACM Transactions on Storage (TOS)*, vol. 9, no. 3, p. 9, 2013.

[6] D. Teigland and H. Mauelshagen, “Volume managers in linux.,” in *USENIX Annual Technical Conference, FREENIX Track*, pp. 185–197, 2001.

[7] R. G. White, “Copy-on-write objects for c++,” *The C Users Journal*, 1991.

[8] F. J. T. Fábrega, F. Javier, and J. D. Guttman, “Copy on write,” 1995.

[9] “Github - battlelog/battlebinary,” Uprise, September 2015. `http://curl.haxx.se/docs/faq.html#What_is_cURL`, accessed 2015-10-15.

[10] R. Haddock, “Intelligent internet system with adaptive user interface providing one-step access to knowledge,” Mar. 14 2014. US Patent App. 14/212,654.

[11] D. Tomaszuk, “Document-oriented triple store based on rdf/json,” *Studies in Logic, Grammar and Rhetoric,(22 (35))*, p. 130, 2010.

[12] L. Richardson and S. Ruby, *RESTful web services.* " O’Reilly Media, Inc.", 2008.