Designing a Virtual Security Layer for Cloud Content

Lukas Klingsbo

November 24, 2015

Abstract

TODO: Abstract
Keywords:

Contents

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 folders and snapshots. A system was developed to make the underlying theory work in practice.

The research question that this report answers is whether it is practically feasible to use Copy-on-Write for a high-level system like the one that is implemented. The work also thoroughly compares different types of lazy snapshots to figure out which is the most

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 delta between the virtual DOMs [1]. React can be seen as the system for handling views in front-ends implementing a MVC (Model-View-Controller) architecture.

2.1.2 Reflux

Reflux [2] is an idea and a simple library of how to structure your application. It features a unidirectional dataflow (see Figure 1) which makes it more suitable, than for example Flux [3], when using a functional reactive programming style.

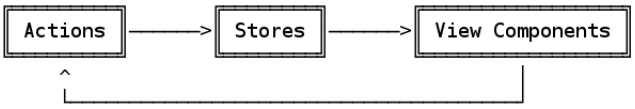

Figure 1: Reflux unidirectional dataflow

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 [4].

2.1.4 REST

REST stands for representational state transfer, it is an architectural idea for writing stateless services. These services usually use URIs to identify specific resources and HTTP to modify or query these resources [5].

In this work we use a RESTful API, which means that it is an API following the REST architecture.

2.1.5 MongoDB

MongoDB is a document-oriented database which means that it does not have the concept of rows as normal relational databases has. Instead each entity in the database is stored as a document which is not fixed to a predefined table structure [6]. MongoDB lacks support for joins too improve its possibility to scale, which can be a big down side to some applications containing the need for such logic.

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 [7].

Copy-on-Write is used in virtual memory management systems [8], snapshot functionality in both file systems [9] and logical volume management systems [10], and as an optimisation technique for objects and types in programming languages [11].

Its principle is that when processes or nodes 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 or copy 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 [12].

TODO: Polish and extend

4 Background

4.1 About Uprise

Uprise is a company based in Uppsala, Sweden. It is an EA studio focussing on creating great gaming experiences, that means that they are not focussed on the actual gameplay which other EA studios like DICE is.

At the moment they are for example very involved with producing the Star Wars Battlefront game by making menu systems and developing the companion app, Battlefront companion.

4.2 The current system

Today a system called battlebinary [13] 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 like this, which should solve the scalability problem and make it possible to implement wanted features like snapshots, cloning and concurrent modifications of content.

5 Model

5.1 Entities

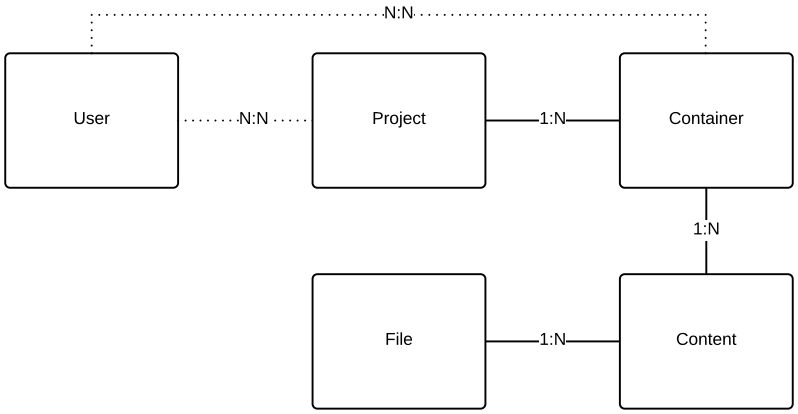

Figure 2: Entity Relationships

5.1.1 Content

Content is meta data about a file and is stored in a container, it is a form of virtual file. The content can refer to for example an image, video or binary blob.

5.1.2 Project

A project is what is created to contain all content and containers 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.

5.1.3 Container

A container is a virtual folder within a project which can contain content and other containers.

5.1.4 Snapshot

A snapshot is a read-only container from the state which the container the was in when the snapshot was created. A snapshot can not be updated and can only be deleted from the root of the snapshot. Snapshots are by default stored as siblings

to the container which they were made from, but they can be contained by any container.

5.1.5 File

A file refers to an actual physical file. Files are stored in the database to make backup, deployment and migration easier.

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 needed 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. Furthermore 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 and determining the happened-before relation. In this work this problem is solved by implementing Copy-on-Write.

6.1 Comparison of Copy-on-Write system implementations

6.1.1 BTRFS

Btrfs is a B-tree file system for Linux which makes use of Copy-on-Write to make it able to do efficient writeable snapshots and clones. It also supports cloning of subtrees without having to actually copy the whole subtree, this is due to the Copy-on-Write effect. As several nodes in the tree can refer to the same node each node keeps track of how many parents it has by a reference counter so that the node can be deallocated once the node does not have any parents any more. The reference counter is not stored in the nodes themselves but rather in a separate data structure so that a nodes counter can be modified without modifying the node itself and therefore eludes the Copy-on-Write that would have to occur.

(a) Tree T_p (b) T_p cloned to T_q

Figure 3: Cloning mechanism of Btrfs [9]

6.1.2 Mach kernel

In the mid 80s when the development of the Mach kernel started, there was problems with that physically copying memory was too slow. Too minimise the copying of memory, Copy-on-Write was implemented. It was implemented so that virtual

copy operations could be done and so that tasks could share read-write memory [14].

TODO: Insert more systems here

7 Implementation

7.1 Perius

Perius is the implementation that was done to solve the problem at hand at Uprise. Perius features a backend written in Scala and a frontend written in Javascript.

7.2 Snapshots

This implementation is far from as efficient as the other Copy-on-Write systems described in Section 6 in most aspects, but more efficient in some. As the implementation is built upon MongoDB as persistent storage and not a pure tree structure, single nodes can be fetched in O(1) but when querying for subtrees they need to be built first, which takes O(log(n)), where n is the number of nodes in the subtree.

In Perius snapshots and clones are not taken in the fashion which Btrfs uses, which can be seen in Figure 3. As Perius does not have the tree structure pre-built and each node is instead stored in a flat storage space, such operation would be too computationally expensive as trees would have to be merged when collisions occur, due to the non-blocking nature of the application. Instead this implementation makes a full copy of the meta-data of the tree, but still refers to the same binary files until they are changed, which results in the creation of a new node.

7.3 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.

7.4 Resulting system

7.4.1 Persistent storage

7.4.1.1 MongoDB

MongoDB was chosen as the persistent storage because of its quick lookups and 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 an asynchronous and non-blocking driver for MongoDB and can therefore make the system scale even further [15].

7.4.1.2 API

REST was chosen as only basic CRUD operations needs to be performed and because the BSON format which is used in MongoDB is almost identical [16] to the standardised JSON format which is used by RESTful services [17].

7.4.2 REST Endpoints

For the front-end to communicate with the back-end, a RESTful service is implemented. 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

  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

  POST - create new snapshot in a specific project

- projects/{id}/containers

  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

  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

As can be seen several expected endpoints are missing, this is intentional as the operations missing can be performed in a more efficient way. Such endpoint is for example $GETprojects/\{id\}/containers$ as all containers exist in $GETprojects/\{id\}$ and the interface should present a file structure where both content and containers are shown.

7.5 Findings

7.5.1 Scalability

7.5.2 Security of the system

Authorization

Audit logs

8 Discussion

9 Summary

9.1 Conclusions

9.2 Future work

References

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

[2] "Github - refluxjs," Reflux, October 2015. `https://github.com/reflux/refluxjs`, accessed 2015-11-24.

[3] C. Gackenheimer, "Introducing flux: An application architecture for react," in *Introduction to React*, pp. 87–106, Springer, 2015.

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

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

[6] K. Chodorow, *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

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

[8] 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.

[9] 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.

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

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

[12] F. J. T. Fábrega, F. Javier, and J. D. Guttman, "Copy on write," 1995.

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

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

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

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

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