Elfin - URL Shortener

Lukas Klingsbo

August 22, 2015

Elfin - URL Shortener

I was given this task by Uprise, as an exercise to show my capabilities. This URL shortener is built with Scala 2.11.7 and the Lift 2.6 Framework. The result can be seen on `http://elfin.se`

Disclaimer: This is the first Lift project that I have coded so please forgive any horrible breaking of convention that I might have unintentionally made.

If you have any questions,
feel free to contact me on:

Lukas Klingsbo
+46737-42 43 45
lukas.klingsbo@gmail.com

1 Background

The task was to make a scalable URL shortener that deterministically shortened the URL's fed to it.

2 Design decisions

The shortener core code was written with the help of MurmurHash3 which I use to convert the $URL \Rightarrow Int \Rightarrow String \Rightarrow URL$.

Another design choice that was considered was using an auto incremented value in a database that was to be converted to a string. This would have yielded shorter strings and avoided the problem with collisions that occur when using short hashes. The first design choice was chosen as the exercise stated that the same link preferably always should produce the same short URL. It also gives the advantage that the keys does not have to be synchronized on creation to the distributed KV-store, they still have to be synchronized for fetching though.

2.1 Database

As the task was to make a scalable URL shortener with high availability, the choice to use a distributed database (in production) was made. Two Key-Value stores were looked into, Riak and Redis. Redis was choosen simply because it had better support for Scala. When deploying Redis for production for this project I recommend using a single master + replicas and then use Sentinel for automatic failover. Which will make the system scalable and extremely fault tolerant as there is no sharding, all data will exist on all of the nodes and when the master goes down a new one is selected to become master by Sentinel.

As the algorithm for determining short URL's is deterministic as many web servers as needed can be used individually with a load balancer to scale the application.

2.2 Variable Key length

When using such short hashes that are used here, then collisions are unavoidable (See the pigeonhole principle). So consider how much entropy you need for the intended usage and set a fixed length of the key before deploying this application.

3 Setup

To try this application you can either use SBT directly or package a WAR file and deploy it on a web server and java servlet containers like Jetty or Tomcat. An instance of Redis is also needed to properly run the application.

4 Todo

In a future version the history of shortened URL's should be provided to the user by linking the URL's to the users session key.