TweetDesk API: reducing the cost

Andrew R. Patterson

February 28, 2014

Abstract

During development of the TweetDesk application it became clear that the pull-based architecture used unneccessary resources. This was most evident in the polling of the database every five seconds using AJAX. The obvious alternative was to implement a push-based architecture using a different technology in the hope of reducing the number of requests.

1 Introduction

The previous API was implemented as a Django application which was queried for event data through the use of simple AJAX polling from the client side. This of course proved costly when it was scaled up. These issues resulted in a new API being developed.

2 TweetDesk API

2.1 Websockets for pushing

Further analysis of the problem showed the limitations of the HTTP protocol for real-time services. Consequently, a new technology that favoured push-notifications was sought and two technologies were identified as being suitable: Websockets and Server-Sent Events. Both technologies were feasible in their own right however Websockets was chosen as the preferred option. This was mostly down to its popularity, its ever-increasing support, and that the client believed it would be a good choice for a "current real-time HTML5 web ap"".

2.2 Tornado Web Server

The previous API ran as a Django application on a Django server. This setup did keep the development environment binded to the Django framework, which proved useful during prototyping. Nonetheless, further research suggested that the Django framework was not best suited to a push-based architecture and that for good results an alternative server and framework must be sought.

Research was carried out and Facebook's Tornado webserver and framework was chosen. The benefits of this server over alternatives included its high-performance, its asynchronous ability and its proved ability to scale easily. Being developed in Python also helped us keep our development environments similar across the TweetDesk system.

2.3 The API application

The application was of course written using Tornado's application framework. The application queries the database every 5 seconds through a single database connection. The result of this query is then pushed to each and every client connected over Websockets.

A simple client-side implementation would be to store the JSON message from the websocket and then compare with previous JSON results. If both results are different then the page should be refreshed.

3 Evaluation

The next steps for the API would be to make it a true push-style application; that is the database would push to the API instead of the API polling the database. The current database is implemented using PostGreSQL and it does provide features - such as notify and listen - that are capable of satisfying this condition. However due to deadlines this was not implemented.

The Tornado server also provides documentation on how to scale the application up over multiple servers using Nginx and could be a good improvement if the application proves to be resource intensive in future releases.

Finally, it was noted in the previous API that the Django modules for accesing and querying the database provided a useful abstraction of the database model as objects. In the future, integration of these modules may be an efficient and clean way of accessing the database and thereby reduce developers' work.