System Architecture

1. Introduction

This document is a mock system architecture document concerning an instant messaging back-end, as part of the Test for Fullstack Developer at IDT Messaging.
As there are plenty of different solutions and combinations which would be suitable for such a task, the proposed architecture is an attempt to provide an efficient solution, keeping in mind assumed requirements such as scalability, availability, robustness and performance.

2. Structure

Database:

Given the possible huge amount of data being read and written to the DB, a scalable solution for this task is required. Even though SQL databases can handle the task, a NoSQL solution is chosen instead as a way to exploit their BASE transactions and allow to scale up much more and in a much easier way.
Redis is the chosen solution as it is a very popular solution, especially for instant messaging (they support Pub/Sub, which comes handy with IM).

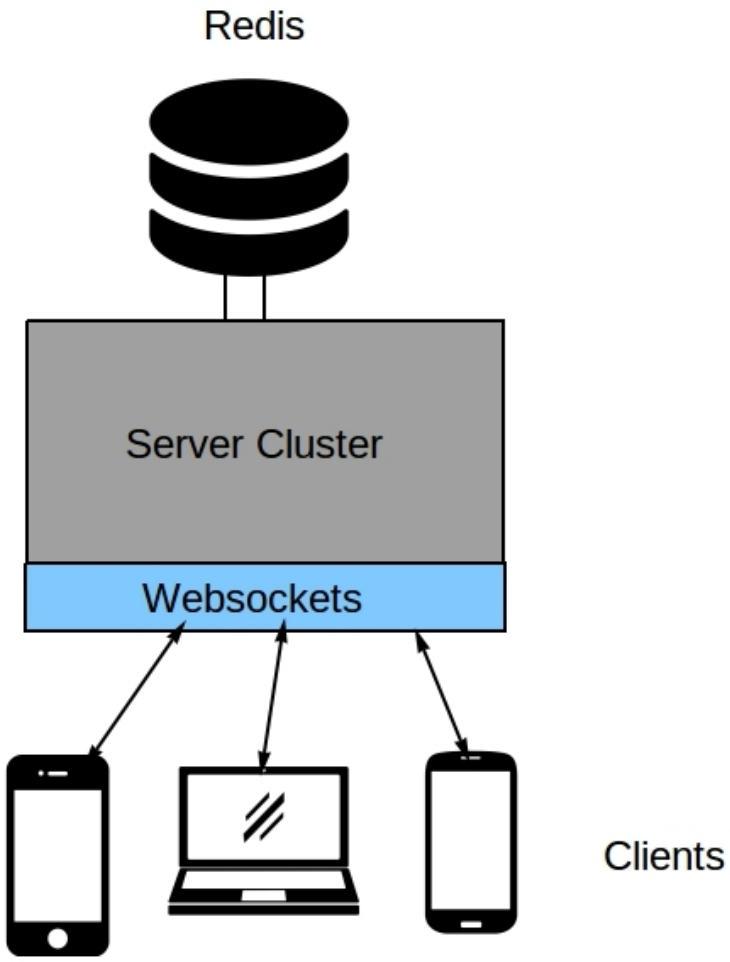
The database is used for storing messages (if offline messaging is required), user credentials and logs.

Server:

The server needs to be capable of managing sessions, user authentication, concurrency, provide proper security and be expandable into several clusters and integrate multiple SMP's.
An Erlang-based solution has proven to be a great choice for highly scalable and heavy load systems. Even though the performance comes with a price (not many Erlang developers), the final result still makes it the best solution for this task.

On the server, a way to handle clients for bi-directional and real-time communication is needed. Websockets and XMPP can do that very well. Choosing one over the other is currently just a matter of personal preference.

3. System Diagram



4. Possible Issues

Scalability and Performance:

Problems concerning scalability and performance are a priority for a messaging (or any other form of telecommunication) system with the goal of growing its user base.
The presented system tries to address such challenges by relying on components and practices that have been proven to work for huge amounts of users (over 900 million users, for WhatsApp as of Sept. 2015).

Scalability is obtained by using databases capable of dividing into multiple clusters with fast read/write times and having servers capable of serve million of users each (by using a lightweight server with a small footprint).

Performance is obtained by having servers capable of Symmetric multiprocessing (SMP) to increase the performance of each CPU core of a system.