**Actions taken for improving reliability**

**Twitter limits (API, updates, and following)**

Limits alleviate some of the strain on the behind-the-scenes part of twitter and reduce downtime and error pages. For the sake of reliability, twitter have placed some limits on the account actions below.

Current twitter limits

The current technical limits for accounts are:

* Direct messages (daily): The limit is 1,000 messages sent per day.
* Tweets: 2,400 per day. The daily update limit is further broken down into smaller limits for semi-hourly intervals. Retweets are counted as Tweets.
* Changes to account email: Four per hour.
* Following (daily): The technical follow limit is 1,000 per day. Please note that this is a technical account limit only, and there are additional rules prohibiting aggressive following behaviour.
* Following (account-based): Once an account is following 5,000 other users, additional follow attempts are limited by account-specific ratios.

These limits include actions from all devices, including web, mobile, phone, API, etc. API requests from all third-party applications are tracked against the hourly API limit. People who use multiple third-party applications with their account will therefore reach the API limit more quickly.

These limits may be temporarily reduced during periods of heavy site usage. In such cases, twitter will post an update on the Twitter Status blog.

src: <https://support.twitter.com/articles/15364>

**Introducing Twitter Lite**

Twitter Lite, a Progressive Web App that is available at mobile.twitter.com. Twitter Lite is fast and responsive, uses less data, takes up less storage space, and supports push notifications and offline use in modern browsers. The web is becoming a platform for lightweight apps that can be accessed on-demand, installed without friction, and incrementally updated. Over the last year twitter has adopted new, open web APIs and significantly improved the reliability and user experience.

Hundreds of millions of people visit mobile.twitter.com every month. Twitter wants Twitter Lite to be the best way to use Twitter when users connectivity is slow, unreliable, limited, or expensive. They have been able to achieve speed and reliability through a series of incremental performance improvements known as the PRPL pattern and by using the new capabilities of modern browsers on Android (e.g., Google Chrome).

To reach every person on the planet, twitter needs to reach people on slow and unreliable networks. When available, twitter uses a Service Worker to enable temporary offline browsing and near-instant loading on repeat visits, regardless of the network conditions. The Service Worker caches the HTML application shell and static assets, along with a few popular emoji. And when scripts or data fail to load we provide “Retry” buttons to help users recover from the failure. All together, these changes improve reliability and contribute to significantly faster loading and startup times on repeat visits.

**Twitter Lite Architecture overview**

Twitter Lite is a client-side javascript application and a small, simple node.js server. The server handles user authentication, constructs the initial state of the app and renders the initial HTML application shell. Once loaded in the browser, the app requests data directly from the Twitter API. The simplicity of this basic architecture has helped us deliver exceptional service reliability and efficiency.

*src:* <https://blog.twitter.com/2017/how-we-built-twitter-lite>

**The infrastructure behind Twitter**

**Hardware efficiency**

Procuring and consuming hardware at Twitter’s scale comes with a unique set of challenges. In order to meet the demands of internal customers, twitter initially started a program to qualify and ensure the quality of purchased hardware. The team was primarily focused on performance and reliability testing ensuring that systems could meet the demands. Running systematic tests to validate the behaviour was predictable, and there were very few bugs introduced.

As twitter scaled their major workloads (Mesos, Hadoop, Manhattan, and MySQL) it became apparent the available market offerings didn’t quite meet the needs. Off-the-shelf servers come with enterprise features, like raid controllers and hot swap power supplies. These components improve reliability at small scale, but often decrease performance and increase cost; for example, some raid controllers interfered with the performance of SSDs and could be a third of the cost of the system. By stranding CPU cores and disk capacity to meet IOPS requirement, they were able to increase reliability and performance but it wasn’t a cost-effective solution.

**Major technology changes and adoption**

* 2012 - SSDs become the primary storage media for MySQL and key/value databases.
* 2013 - first custom solution for Hadoop workloads is developed, and becomes primary bulk storage solution.
* 2013 - custom solution is developed for Mesos, TFE, and cache workloads.
* 2014 - custom SSD key/value server completes development.
* 2015 - custom database solution is developed.
* 2016 - they developed GPU systems for inference and training of machine learning models.

*src:* <https://blog.twitter.com/2016/the-infrastructure-behind-twitter-efficiency-and-optimization>

**Evolution of Architecture**

At the beginning of Twitter the back-end service was only a MySQL database that was used to store all the messages that were being posted. Ruby on Rails was used to make the website and the interactions. Also, the front-end service was written in Ruby on Rails and it provided the communication between the back-end service. With this architecture, they could satisfy the users most important software quality aspects, reliability, availability, performance. At that time, nobody had foreseen the immense growth of twitter the following two years, so further improvement of the system was not a priority.



In 2007, Twitter really started to grow more rapidly and this exponential growth continued in 2008. With this growth, the front-end service alone couldn’t handle all the requests. The solution was a middle layer that could handle the enormous amount of Tweets posted per second. This middle layer was also written in Ruby on Rails and provided a queuing system in which the tweets could be saved before they were written to the back-end makes the system more robust. This improved the reliability quality aspect of the system.

**Reliability Testing**

Twitter strive to prepare for sustained traffic as well as spikes, some of which the can plan for, some of which comes at unexpected times or un unexpected ways. To prepare for those varied types of traffic, they continuously run tests against their infrastructure to ensure it remains a scalable and highly available.

They have created a framework to perform different types of load and stress tests in different stages of a service live cycle in different environments. These tests help them to be more confident that their products are highly available and responsive at all times and under any circumstance. As a part of this reliability testing, the generate distributed multi-datacenter load to analyse the impact and determine the bottlenecks.