SimpleTEX: Architectural Overview

The SimpleTEX architecture has six components:

- **Web** – Stateless servers, running (J)Ruby on Rails, for managing application UI. Also handles file upload/download.
- **Database** – Handles user information, document structure, and permissions.
- **LATEX Processors** – Stateless servers responsible for generating PDF documents from LATEX source. As demand increases more servers can be brought online.
- **EtherPad Servers** – Hosts EtherPad instances. As demand increases more can be provisioned.
- **Load Listener** – Monitors load of LATEX and EtherPad servers, and can provision additional servers if needed. Also able to remove servers as load decreases.
- **Message Bus** – Implemented via Simple Queue Service (SQS). All requests for LATEX processing, EtherPad instances, and other as-yet-undetermined resources from the web servers will be mediated by the message bus.

Figure 1 shows the relationships between these components.

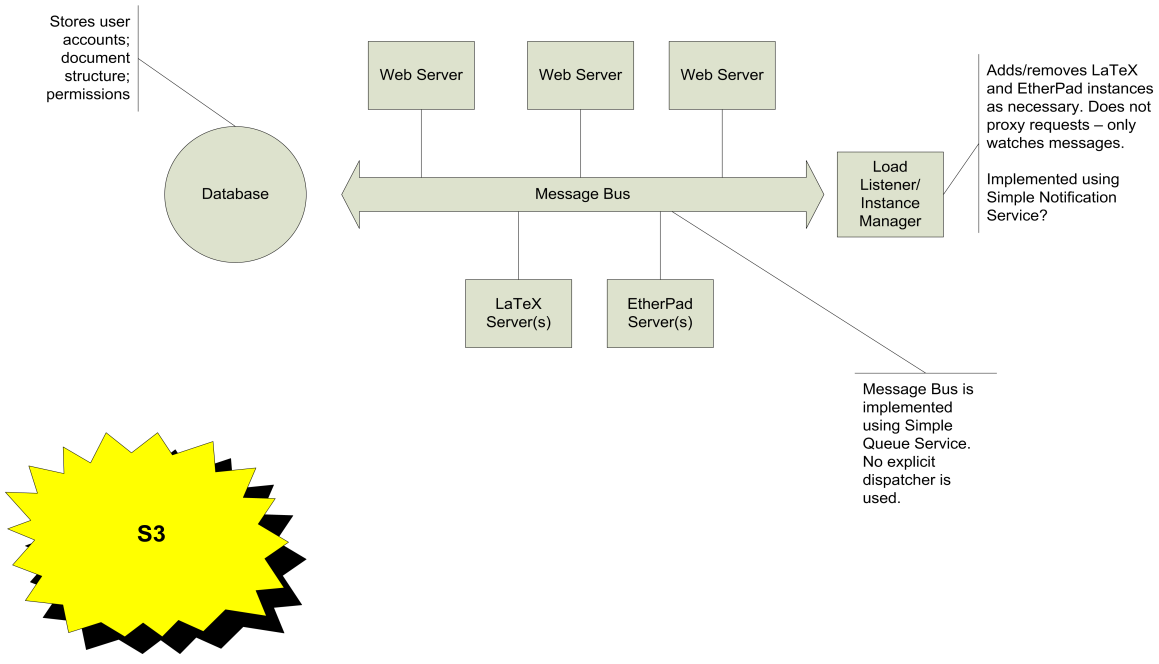

Figure 1: Initial sketch of the SimpleTEX architecture.

SimpleTEX is built around documents – managing and creating PDF files from LATEX source. Users own documents and invite others to collaborate, but the SimpleTEX does not manage its work based on users. It manages based on documents.

1 LATEX Processors

Each processor (Figure 1) watches the message bus for document processing requests. SQS ensures that only one server at a time will read a given request. The server will follow these steps once it receives a request:[1]

1. **Visibility** – Set the visibility timeout on the message to five minutes. This gives the server time to process the document.[2].
2. **Document files** – All files related to the given document are retrieved from storage and placed on the server. This includes custom packages and files.
3. **LATEX** – LATEX is run on the document and the output is placed back into storage.
4. **Notification** – The LATEX processor sends a notification that the document has finished processing and looks for another document to process.

Each LATEX processor is identical to the others. They have Linux installed, can run LATEX, have a default set of packages installed, and can interact with Amazon's services (e.g., SQS, S3, etc.). They do not have user accounts for each SimpleTEX user.

1.1 Document Security During Processing

Keeping users documents separate and isolated during processing requires special handling. In a normal UNIX environment, file system permissions keep users from reading each other's documents.

SimpleTEX cannot rely on file system permissions because the number of SimpleTEX users could grow very large. Additionally, keeping all LATEX processors synchronized with the latest set of user accounts would be difficult. SimpleTEX will instead use the following scheme:

- Each server will have 20 users (`nobody1` – `nobody20`) which cannot login interactively.
- A scratch directory will exist on each server. No one except root will be able to read or modify the directory.
- After receiving a request to create a document, the server will select a `nobody` to run the job. The `nobody` selected will not currently be running any jobs, and no jobs will be started while the `nobody` is "busy."

[1] Error handling and timeout issues are left out but also must be dealt with.

[2] We'll need to manage long-running documents. Five minutes probably indicates an error, but could be legitimate for long documents. Possibly a premium pricing thing?

- A randomly-named directory which only the selected `nobody` can access will be created under the scratch directory. An existing directory will *never* be re-used.
- The files necessary for the job will be copied from storage and placed in the new directory.
- LaTeX will be started in a new process, running as that user, configured to use the new directory.
- When finished, the final document will be retrieved and placed in storage. The directory will then be deleted.

This scheme ensures that no document can read files attributed to another document. The scratch directory can only be accessed by root, which ensures the `nobody` users cannot list its contents. Even knowing the name of another directory will not help, since different `nobody` user's own them. Finally, if the names chosen are truly random, it will be impossible to guess the name of another directory owned by the same `nobody`.

1.1.1 Limitations

This solution requires that the files for each document are copied from storage each time the document is processed. This will work fine if a document is only processed once a day, or even a few times an hour. It will not work very well when a document is processed repeatedly. Bandwidth usage and transfer time could be prohibitive.