# Design of Google Docs

Let's understand how we can design the collaborative document editing service using various components.

**We'll cover the following**

- Design
  - Components
  - Workflow

## Design

We’ll complete our design in two steps. In the first step, we’ll explain different components and building blocks and the reason for their choice in our design. The second step will describe how we fulfill various functional requirements by depicting a workflow.

### Components

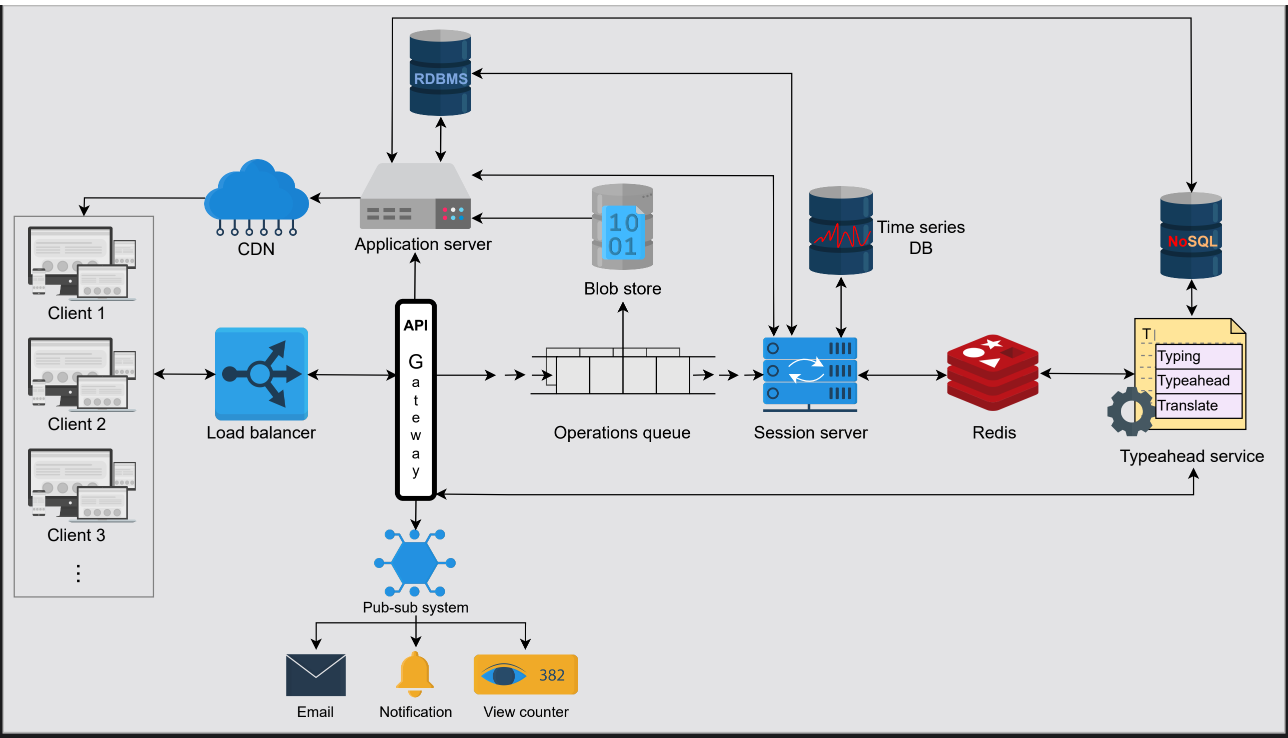
We’ve utilized the following set of components to complete our design:

- **API gateway**: Different client requests will get intercepted through the API gateway. Depending on the request, it’s possible to forward a single request to multiple components, reject a request, or reply instantly using an already cached response, all through the API gateway. Edit requests, comments on a document, notifications, authentication, and data storing requests will all go through the API gateway.
- **Application servers**: The application servers will run business logic and tasks that generally require computational power. For instance, some documents may be converted from one file type to another (for example,

from a PDF to a Word document) or support features like import and export. It’s also central to the attribute collection for the recommendation engine.

- **Data stores**: Various data stores will be used to fulfill our requirements. We’ll employ a relational database for saving users’ information and document-related information for imposing privilege restrictions. We can use NoSQL for storing user comments for quicker access. To save the edit history of documents, we can use a time series database. We’ll use blob storage to store videos and images within a document. Finally, we can use distributed cache like Redis and a CDN to provide end users good performance. We use Redis specifically to store different data structures, including user sessions, features for the typeahead service, and frequently accessed documents. The CDN stores frequently accessed documents and heavy objects, like images and videos.
- **Processing queue**: Since document editing requires frequently sending small-sized data (usually characters) to the server, it’s a good idea to queue this data for periodic batch processing. We’ll add characters, images, videos, and comments to the processing queue. Using an HTTP call for sending every minor character is inefficient. Therefore, we’ll use WebSockets to reduce overhead and observe live changes to the document by different users.
- **Other components**: Other components include the session servers that maintain the user’s session information. We’ll manage document access privileges through the session servers. Essentially, there will also be configuration, monitoring, pub-sub, and logging services that will handle tasks like monitoring and electing leaders in case of server failures, queueing tasks like user notifications, and logging debugging information.

The illustration below provides a depiction of how different components and building blocks coordinate to provide the service.



A detailed design of the collaborative document editing service

Quiz

**Question 1**

Why should we use WebSockets instead of HTTP methods? Why are WebSockets optimal for this kind of communication?

Show Answer

1 of 3

## Workflow

In the following steps, we'll explain how different requests will get entertained after they reach the API gateway:

- **Collaborative editing and conflict resolution**: Each request gets forwarded to the operations queue. This is where conflicts get resolved between different collaborators of the same document. If there are no conflicts, the data is batched and stored in the time series database via session servers. Data like videos and images get compressed for storage optimization, while characters are processed right away.
- **History**: It's possible to recover different versions of the document with the help of a time series database. Different versions can be compared using DIFF operations that compare the versions and identify the differences to recover older versions of the same document.
- **Asynchronous operations**: Notifications, emails, view counts, and comments are asynchronous operations that can be queued through a pub-sub component like Kafka. The API gateway generates these requests and forwards them to the pub-sub module. Users sharing documents can generate notifications through this process.
- **Suggestions**: Suggestions are in the form of the typeahead service that offers autocomplete suggestions for typically used words and phrases. The typeahead service can also extract attributes and keywords from within the document and provide suggestions to the user. Since the number of words can be high, we'll use a NoSQL database for this purpose. In addition, most frequently used words and phrases will be stored in a caching system like Redis.
- **Import and export documents**: The application servers perform a number of important tasks, including importing and exporting documents. Application servers also convert documents from one format to another. For example, a `.doc` or `.docx` document can be converted in to `.pdf` or vice versa. Application servers are also responsible for feature extraction for the typeahead service.

> **Note:** Our use of WebSockets speeds up the overall performance and enables us to facilitate chatting between users who are collaborating on the same document. If we combine WebSockets with a Redis-like cache, it's possible to develop an effective chatting feature.

Quiz

**Question 1**

We’re implementing view counters through an asynchronous method, which means that the number of views of a document may be outdated. Could we use sharded counters or Redis counters to get effective results?

Show Answer

1 of 2

Back

Requirements of Google Docs’ Design

✓ Completed

Next

Concurrency in Collaborative Editing