Congress in the Cloud

Michael Coughlin and Michael Skirpan

The Congressional Dataset which includes all of the Congressional Record (i.e., words spoken on the floor), Sub-Committee Hearings, and Votes is a massive and continually-growing set of words that is difficult to access for the common voter. Currently the only place on the web where Congressional data is stored in an indexable and searchable format is on the Sunlight Foundation's Web API. While Sunlight's project is a massive step forward, it is still mostly accessible for those computer savvy enough to work with REST APIs. Our project was aimed at building a cloud infrastructure that would establish the foundation of what is needed to support a web-based search engine allowing end-users the ability to search through congressional records.

Project Goals

The larger objective of this project will take a lot of front-end design and algorithmic design to support optimized search functionality and natural language processing. However, for this class our goal was to learn about building distributed systems in the cloud in order to get the working parts of such a project going. Aspects we focused on were:

- Automated configuration of nodes in the cloud
- Distributed data stores
- Basic publisher-subscriber systems
- Crawling and data ingestion
- Using Remote Procedure Calls (RPC) to develop scalable response patterns

Our goal was to write an automated procedure for building out worker, data, and web server nodes. Application setup is always one of the most time consuming tasks in any distributed system. Thus, we wanted to learn how to simplify and streamline this process as much as possible.

Next, we wanted to iterate over different options for a distributed data store.

Specifically, we wanted a NoSQL database and were hoping to decide between Cassandra, Riak, and Redis. Once we had a database configured, we planned to write a crawler to pull down our data, parse it, and fill the databases.

Finally we hoped to use a publisher-subscriber system to handle communication between nodes. Given that our final aim was to be able to open up this resource to end-users for the purpose of applying more complicated algorithms, we planned to design our pub-sub system so as to learn about RPC.

Description of Process

Initially, there was some difficulty determining the network topology and basic functionality of our system. In particular, after deciding on the basic infrastructure, we had three primary questions on which to decide:

1. Do we put pub-sub exchanges between both the data and worker and the web server and worker?
2. Should we allow the web server's node to also host the pub-sub exchange?
3. Do we ingest our data with a script on the worker node or handle novel data ingestion from the web server?

After some consideration, we decided that (1) and (3) were questions that needed to be answered coextensively. We only really needed the extra pub-sub exchange in (1) if we were going to handle both ingestion and query tasks at the level of the web server. In this case, we would've needed the extra exchange in order to allow certain data nodes to be subscribed to new information being pushed while mirrors of those databases could have taken care of query responses.

Given that the Congressional Record has a single, centralized outpost, we deemed it unnecessary to allow users to provide us with data since we can easily keep up to date with any new data posted by the federal government. As for (2), we chose to keep the pub-sub exchange on the same node as the web server in order to avoid the problem of our exchange going down but the Web Server still taking requests.

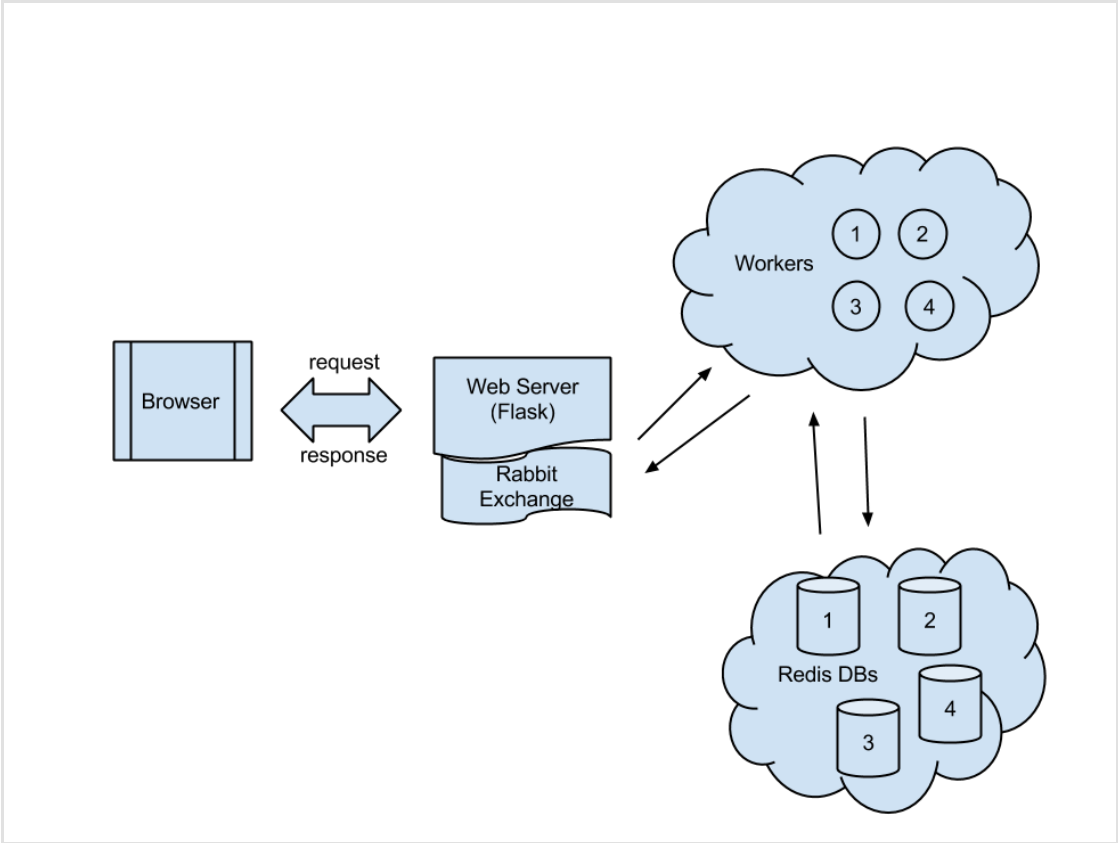
[Michael can describe the process of using Puppet here]

In choosing a database, we started by looking into Cassandra, but quickly realized this was much too large of a system for our needs. We didn't really need JSON-like blobs and instead just needed a way to look up documents. Then we moved on to reading into Riak, which is supposed to be built for large, distributed systems. However, we found that the configuration of Riak, was more than we wanted to commit to this portion of project. Thus, in the end we chose Redis due to the simplicity of configuring the server and setting up mirroring as well as the well supported Python API.

One of the biggest challenges in this process was getting the crawler, parsing, and indexing working correctly. At first, we wrote a simple web scraper using Tornado. This worked fine, but then we got bogged down trying to get the parsing right. In the end, we found that Sunlight Foundation had already written a parser that converted Congressional Documents to marked-up XML. Since their tools are open source, we decided to attach their parser to my crawler.

Technology Used

The final topography, including technology used can be seen in the figure below:



Findings and Lessons Learned

Future Implementations

Conclusion