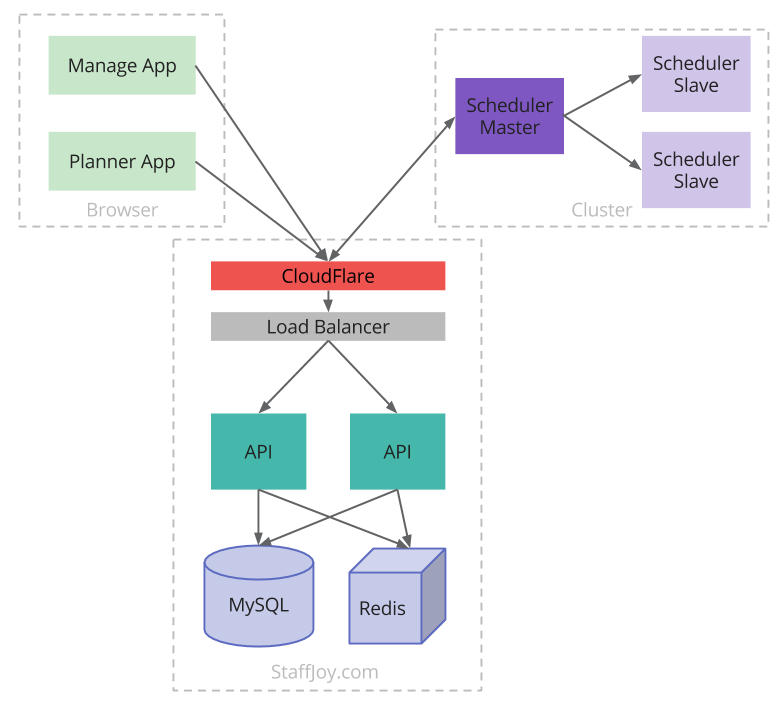
**Our Stack**

This post explains how [StaffJoy](https://www.staffjoy.com) designed a scalable, distributed, service-oriented architecture for scheduling workers.

**Architecture Diagram**



**Browser**

StaffJoy deploys three main single-page Javascript applications that run in users’ browsers. All three are built with [Backbone.JS](http://backbonejs.org/), which we chose because of familiarity from previous work (though it has [caused some hiccups](https://blog.staffjoy.com/2015/04/17/backbone-js-events/)). Scheduling managers use the **Manage App** to set forecasted demand, modify worker parameters, and view schedules. Workers use the **Planner App** to view current schedules and set availability for future weeks. The StaffJoy team uses the internal **Euler App** to onboard new users and conduct customer support. These three apps each have a different user experience and purpose, but communicate with a shared backend API using RESTful JSON.

**StaffJoy.com / API**

The core of StaffJoy is a [Flask](http://flask.pocoo.org/) application written using [Python](https://www.python.org/). It serves Javascript, compiles [LESS](http://lesscss.org/), sends emails, and communicates with the databases. The core StaffJoy API extends Twilio’s [Flask-RESTful](https://flask-restful.readthedocs.org/en/0.3.4/) package.

This app utilizes [Docker](https://www.docker.com/) on [Amazon Web Services](https://aws.amazon.com) for deployment. [Elastic Load Balancing](https://aws.amazon.com/elasticloadbalancing/) and [auto-scaling EC2](https://aws.amazon.com/autoscaling/) allow us to horizontally scale the app to accommodate demand. [Cloudflare](https://www.cloudflare.com/) sits in front of the load balancer to serve as a content delivery network and to provide additional security.

MySQL serves as our main datastore. Shared data such as sessions, large data queries, and rate limiting are cached in [Redis](http://redis.io/) for speed.

**Scheduler**

The Scheduler is written in Julia. This scientific programming language has made it straightforward to design [advanced scheduling algorithms](https://blog.staffjoy.com/2015/03/26/or-vs-ai/), but has raised some challenges with [testing](https://blog.staffjoy.com/2015/02/04/tdd-in-julia/) and interactions with REST APIs.

Functionally, the scheduler acts as a queue worker. However, because messages happen so infrequently, the scheduler queries the StaffJoy API directly. This reduces the need to monitor an additional component and provides a consistent datastore.

The Scheduler pulls tasks from the API, runs calculations using our algorithms, and returns schedules to the API. Calculations are highly parallel and support slaves for distributed calculations in order to decrease the time it takes to process a single schedule.

Each Scheduler master only runs one computation at a time due to high resource utilization, so we horizontally scale the number of clusters when load is high.

**Additional Tech**

We also rely on these technologies:

* [Github](https://github.com/StaffJoy)
* [Docker Hub](https://hub.docker.com/)
* [Jenkins CI](https://jenkins-ci.org/) for deployment, build promotion, and functional testing
* [Circle CI](https://circleci.com/) for unit testing and pull request building
* [Papertrail](https://papertrailapp.com/) for log monitoring and triggering alerts
* [Pagerduty](https://pagerduty.com/)
* [Mandrill](http://mandrill.com/)
* [Intercom](https://www.intercom.io/) for [user tracking](https://blog.staffjoy.com/2015/07/05/async-intercom-user-tracking-in-python/)
* [Ghost](https://ghost.org/) for the [StaffJoy Blog](https://blog.staffjoy.com)