WF - Content Delivery Server

Contents

[Overview 2](#_Toc511815837)

[Key Questions 2](#_Toc511815838)

[Environments 3](#_Toc511815839)

[Development Workflow 3](#_Toc511815840)

[Technical Stack & Other Tools 4](#_Toc511815841)

[Backend 4](#_Toc511815842)

[Django (Python) 4](#_Toc511815843)

[Employee User Authentication 4](#_Toc511815844)

[Frontend 4](#_Toc511815845)

[JavaScript 4](#_Toc511815846)

[nvd3.js, d3.js 5](#_Toc511815847)

[Database 5](#_Toc511815848)

[SQLite 5](#_Toc511815849)

[Hosting 5](#_Toc511815850)

[Amazon Web Services 5](#_Toc511815851)

[Docker 5](#_Toc511815852)

[Git 6](#_Toc511815853)

[AJAX/JSON Mechanism 6](#_Toc511815854)

[TBD 6](#_Toc511815855)

[Node JS, Socket.io & Redis 6](#_Toc511815856)

[Other Tools 7](#_Toc511815857)

[Balsamiq 7](#_Toc511815858)

[Celery 7](#_Toc511815859)

[Redis 7](#_Toc511815860)

[Munin 8](#_Toc511815861)

[StatsD 8](#_Toc511815862)

[Amazon Web Services Details 8](#_Toc511815863)

[AWS Elastic Beanstalk 8](#_Toc511815864)

[Amazon Elastic Cloud Compute 9](#_Toc511815865)

[Amazon Simple Storage Service 9](#_Toc511815866)

[Amazon Virtual Private Cloud 10](#_Toc511815867)

[AWS Elastic Beanstalk + AWS Virtual Private Cloud 10](#_Toc511815868)

[AWS Relational Database Service 11](#_Toc511815869)

[AWS Elastic Beanstalk + AWS Relational Database Service 11](#_Toc511815870)

[Other Notes 11](#_Toc511815871)

# Overview

The content delivery system is a web application created and maintained by the WalterFedy operations team and hosted on Amazon’s cloud services. The content delivery system will be used by WalterFedy employees and WalterFedy clients.

Internally, the content delivery system will be used to manage key assets and streamline processes that are essential to day-to-day WalterFedy operations. This includes but is not limited to the management of resources such as keys, lockers, and projectors and the management of service requests. This system will centralize all the email and face-to-face requests made to operations, maintenance, and IT to enable better record keeping and lead to a more efficient and effective method of handling requests. The storage of data is also a key aspect of this project as WalterFedy looks to expand into the business of data.

Externally, key clients will have access to the content delivery system to view data-heavy and interactive visuals generated by WalterFedy staff.

# Key Questions

The following questions were asked by stakeholders prior to the start of the project.

Q: What are our current requirements for an internal web system?

A: Client authentication, SSH terminal access, and separate development and production environments.

Q: How are we going to manage our code?

A: Code will be managed through a Version Control System (Git). Details TBD.

Q: What is the difference between production and development environments?

A: AWS Elastic Beanstalk allows for the creation of multiple environments when multiple versions of an application need to be run. Multiple environments could be created for development and production.

Q: How do we host a sandbox?

A: Amazon EC2 instances will be used through AWS Elastic Beanstalk to run virtual machines in the cloud.

Q: What frameworks and packages are we currently using?

A: This is a basic overview of the core frameworks and packages being used.

1. **Git** will manage the code written.
2. **Django** will be used as the backend setup for user-logins and user management while **NodeJS** will add real-time functionality.
3. **SQLite** will be used as the database.
4. The web application will be packaged through **Docker** and **AWS Elastic Beanstalk** will be used for deployment.

This does not include all the technologies that will be used. Please see below for the full list and details.

# Environments

The development workflow should include development, staging, and production environments. A key feature to include when create the development process is the ability to rollback.

* Development – a locally stored or remote (separate AWS) environment where new features are created
* Staging – a copy of the production environment where new features are tested
* Production – the environment where tested features are deployed for public use

More research will have to be done to create a full picture of the workflow. Technologies to look at are Git, Docker, AWS Elastic Beanstalk, and other AWS services.

<http://tech.yipit.com/2011/11/02/django-settings-what-to-do-about-settings-py/>

<https://news.ycombinator.com/item?id=5251361>

# Development Workflow

* Work in close collaboration with end users; each iteration of product coincides with user feedback sessions
  + **Result:** Less likely to build something that people won’t use
* Create a story map/backlog of app requirements (Kanban)
  + **Result:** Create better understanding between user and developer and keep track of status of project
* Create wireframe with basic flow of app, layout, and mock functionality using Balsamiq prior to actual development of app
  + **Result:** Reduce probability of altering code already written and writing code that won’t be used
* Develop front end and back end simultaneously
  + **Result:** Immediately creates a “full picture” to help users decide if function will be used
* Minimum viable product is launched at the end of “sprint” period with minimum functionality implemented to be usable
  + **Result:** Easy to test basic functionality

# Technical Stack & Other Tools

## Backend

### Django (Python)

Django, a high-level Python web framework is used for the backend. Django provides the user-authentication system and the ability to connect via LDAP for the WalterFedy employee authentication.

* Takes application from concept to launch with features such as user authentication, content administration, site maps, RSS feeds

### Employee User Authentication

Method of authentication has yet to be implemented. The connection will likely be made through LDAP.

<https://django-auth-ldap.readthedocs.io/en/latest/>

## Frontend

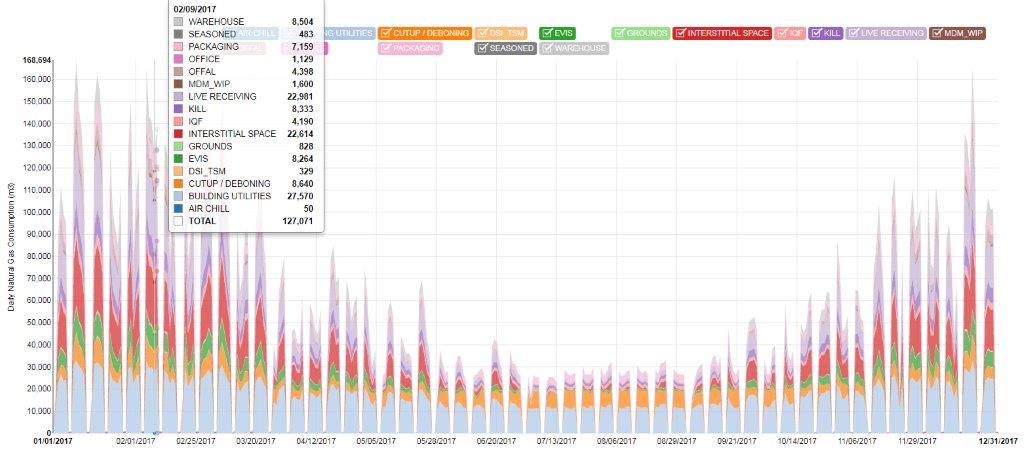
### JavaScript

Vanilla JavaScript is used for the frontend. Limited work on the front end has been completed so vanilla JavaScript has been sufficient thus far. A library or framework may be used in the future such as React, Vue, and jQuery (for AJAX).

### nvd3.js, d3.js

Nvd3.js is a library of re-usable charts from d3.js. The collection of charts is very customizable and easy to use. This tool will mainly be used on the client side of the portal with charts already made for EMS clients.

<http://nvd3.org/index.html>



* Django-nvd3 may be useful for using Django with nvd3.js

<https://github.com/areski/django-nvd3>

## Database

### SQLite

SQLite is used as the primary data store, once transitioned to Amazon Web Services, the database may be changed to PostgreSQL through the Amazon Relational Database Service.

## Hosting

### Amazon Web Services

The system will be hosted on Amazon Web Services. Amazon Elastic Beanstalk service is a platform-as-a-service (PaaS) which will help deploy and manage the web application. Amazon Elastic Beanstalk will run services such as Elastic Compute Cloud, Simple Storage Service, Virtual Private Cloud, and Relational Database to support the web application.

See more below.

### Docker

Docker is a container platform that allows organizations to solve multiple problems across a diverse set of requirements.

* Easy to package code for production with the kind of environment needed
* Service that monitors the state of containers

### Git

Git is a distribute open-source version control system that stores code, tracks revision history, merges code changes, and reverts to earlier code versions

* Stores source code and full development history locally in a repository
  + Create branch that can be worked on in parallel to the main version
  + Commit changes to save progress or merge your branch back into the main version
  + Every commit will be stored in a snapshot and compared to previous versions
* Allows for branching a merging which is ideal for testing and production environments
  + Could create a feature-based workflow with new branches for each new feature that can be switched on/off, deleted, or merged to the main branch that allows for disposable experimentation
* All operations performed locally allowing Git to be small and fast
* Distributed system with multiple backups available
  + No single point of failure unless only single copy of repository is available
  + Allows for different style of workflows (important when more than one developer is on the team!)
    - Subversion-style workflow, integrated manager workflow, dictator workflow
* Data model ensures cryptographic integrity of every file and commit
* Intermediate staging area where commits can be formatted and reviewed before completion

## AJAX/JSON Mechanism

### TBD

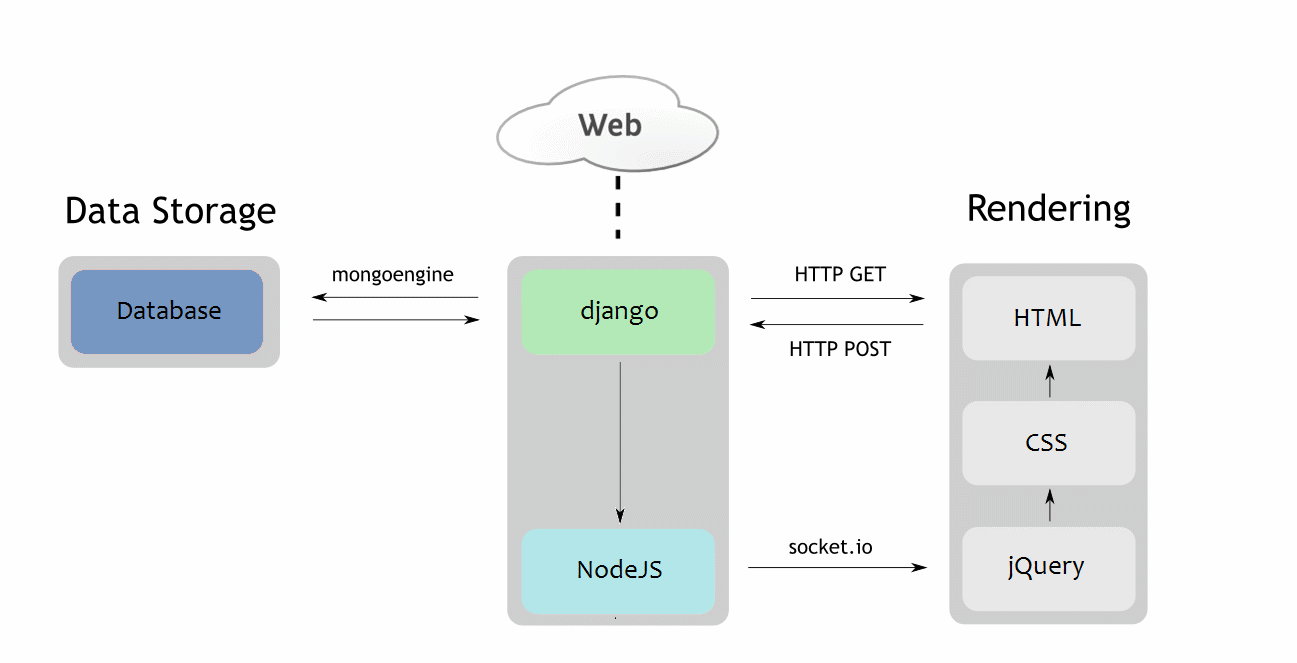
Django does not have a built in JSON HTTP response and the application is going to have to send data to the server using AJAX when data is used for the graphs on the client side. This service might be required earlier on the internal side for autofill on forms.

Decorator: <https://github.com/samuelclay/NewsBlur/blob/master/utils/json_functions.py>

Response: <https://github.com/clutchio/clutch/blob/master/django_ext/http.py>

### Node JS, Socket.io & Redis

* Node JS will be used to build the real time data retrieval
  + Server part of library runs on NodeJS which will provide a high-performance event-driven framework to manage message exchange with client
* Redis will be used as a message bus with architect socket-io server to send messages between Django and Node JS
  + Socket.io enables real-time bidirectional event-based communication
  + Redis will be used as an in-memory database structure store



* The above diagram is how the Django-NodeJS connection could be made (without Mongo engine connection as the database)

More details: <http://www.cuelogic.com/blog/how-to-use-both-django-nodejs-as-backend-for-your-application/>

## Other Tools

### Balsamiq

Balsamiq is a wire framing tool that allows you to create wireframes quickly to capture the look and feel of a web application or web page.

<https://balsamiq.com/>

### Celery

Celery is an asynchronous task queue based on distributed message passing. Allows for real-time operation and supports scheduling tasks. Celery can be used for a variety of purposes from sending emails to pulling data from the Facebook API. Redis is the recommended backend for Celery.

<http://www.celeryproject.org/>

### Redis

Redis is an in-memory data structure store, used as a database, cache and message broker. Redis can be used to queue Celery jobs, used to store sessions, used as a cache, used for auto completion among other functions.

Queue Celery jobs: <https://gist.github.com/mikeyk/910392>

Autocompletion: <https://github.com/RedisLabs/redis-completion>

Redis cache: <https://github.com/sebleier/django-redis-cache>

### Munin

Munin is a resource monitoring tool that can help analyze resource trends to determine causes of system failure or slow performance. More research will have to be done to determine whether or not this service will add value.

<http://munin-monitoring.org/>

### StatsD

StatsD aggregates and summarizes application metrics. This application will be useful for the content delivery server as it can easily create graphs and data for virtually anything. For example, StatsD can be used to measure the amount of login attempts or the average length of time a user will spend on the web application.

<https://codeascraft.com/2011/02/15/measure-anything-measure-everything/>

# Amazon Web Services Details

## AWS Elastic Beanstalk

AWS Elastic Beanstalk is a service for deploying and scaling web applications and services. Deployment will be automatically handled with capacity provisioning, load balancing, auto-scaling, and application health monitoring included.

* Application can be deployed through AWS management console, Git repository, or IDE (AWS Toolkit for Eclipse)
* Developers retain full control over AWS resources powering application
  + Select OS requirements, available database, availability zone
* Supports multiple running environments that is independently configured and runs on their own AWS resources
  + Potential use in integration testing, pre-production, and production
  + New environment created for each Git branch
* Database and storage
  + Amazon S3 bucket will be created and all files copied from local client to S3


        Elastic Beanstalk Architecture Diagram
      

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html?icmpid=docs_elasticbeanstalk_console>

### 

## Amazon Elastic Cloud Compute

Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud.

* Launch instances with a variety of operating systems, load with custom application environment, manage access permission and run with as many (or few) systems as needed
* Instances are virtual machines created and run on the cloud
  + Specify software and specifications of the instances
  + Choose instance type (CPU, memory, storage, and network capacity)
  + Connect to instance using Remote Desktop Protocol client or through SSH

## Amazon Simple Storage Service

Amazon S3 is an object storage built to store and retrieve any amount of data from anywhere.

* Designed to deliver 99.99% durability and provide comprehensive security and compliance capabilities that meet zone regulatory requirements
  + Distributes data across three physical facilities (minimum) with data replicated to any region for easy of transfer and security
  + Supports 3 types of encryption
* Build applications that make use of Internet storage (websites and web apps)

## Amazon Virtual Private Cloud

Amazon VPC lets you provision a logically isolated section of the AWS cloud to launch AWS resources in a custom defined virtual network.

* Grants complete control over virtual networking environment
  + Selection of own IP address ranges, creation of subnets, and configuration of route tables and network gateways
  + Could have hardware VPN connection between current datacenter and VPC
    - Leverage AWS cloud as an extension of datacenter
  + Define own network space and control how network and Amazon EC2 resources are exposed to the internet

### AWS Elastic Beanstalk + AWS Virtual Private Cloud

* Define virtual network in isolated section of the AWS cloud
* New class of web applications can be deployed on Elastic Beanstalk
  + Internal web applications (ex. recruiting application)
  + Web applications connecting to an on premise database (using VPN connection)
  + Private web service back ends
* Instances must have access to internet directly or through NAT device
* The following resembles what our server web application could be (consult Sergiu)
  + Deploying EB application with Amazon RDS in a VPC using a NAT gateway


      Elastic Beanstalk and VPC Topology with Amazon RDS
    

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/vpc.html?icmpid=docs_elasticbeanstalk_console>

## AWS Relational Database Service

Amazon RDS is a managed relational database service that uses familiar database engines and handles routine database tasks such as provisioning, patching, backup, recovery, failure detection, and repair.

### AWS Elastic Beanstalk + AWS Relational Database Service

* Allows user to add database instance to Elastic Beanstalk environment
  + PostgreSQL can be added during or after environment is created
* Database instance can be launched outside environment for production environment
  + Connect to database from multiple environments and tear down environment without affecting the database instance
* Can be used with a variety of different databases running on EC2

# Other Notes

* When creating Django models for new functions (e.g. projectors), store as much relevant data as possible even if there is no immediate use for the data
  + This allows for better record keeping and will help when changes need to be made or features need to be added
* Use named URLs and the URL template tag to prevent breaks when URLs are changed
* Create a new application for the client portal outside of the “catalog” application which is used for the internal portal
* This tutorial is a good starting point to understand the components of the current application as the project was started here
  + <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Tutorial_local_library_website>
* Look for applications already available for Django before creating anything new
  + <https://django-model-utils.readthedocs.io/en/latest/>
  + <http://django-extensions.readthedocs.io/en/latest/>
* Unsplash has a great library for free photos
  + <https://unsplash.com/>
  + This is the formatting I used for the homepage photos ?dpr=2&fit=crop&fm=jpg&h=825&q=50&w=1450
* Dashboard inspiration:
  + <https://medium.muz.li/30-handpicked-excellent-dashboards-347e2407a057>
  + <https://medium.muz.li/dashboard-analytics-page-inspiration-65a5d8f20683>
* Form inspiration: <https://medium.muz.li/form-design-inspiration-6bb9a350f2d8>
* Helpful links for deploying applications to AWS
  + <https://realpython.com/deploying-a-django-app-and-postgresql-to-aws-elastic-beanstalk/>
  + <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html>
  + <https://jamesonricks.com/tutorial-deploying-python-3-django-postgresql-to-aws-elastic-beanstalk/>