## **Project Objective**

Design and Develop an end-to-end data processing pipeline for a real-world business scenario with a production dataset based on concepts learnt from UCB-W205 Storing and Retrieving Data course. More specifically:

* Use HDFS to host a large volume (750 MB) dataset for ingestion
* Use Apache Spark with PySpark and Spark-SQL to process the data in to business insights
* Use Postgres DB to store the processed information for lower latency
* Build a REST API based serving layer that can invoke either the Spark processing job or obtain data from a database
* Implement the data processing pipeline in a cloud-based server infrastructure

## **Business Use Case**

1. Business and leisure travelers are always looking for:
   1. “Which is the best restaurant (based on user reviews) to get some <Pancakes> around here”
   2. “So my team is going to this restaurant, wonder what dish is good out there (based on user reviews)”
2. If a restaurant is looking to expand or grow:
   1. “Our <food category> restaurant is taking off, which other cities should we consider to open another franchise”

## **Dataset**

The primary dataset is Yelp’s challenge dataset. You can download the data from <https://www.yelp.com/dataset_challenge>

1. Food Review Data - Yelp dataset (750 MB):

* 2.2M reviews and 591K tips by 552K users for 77K businesses
* Social network of 552K users for a total of 3.5M social edges.
* Aggregated check-ins over time for each of the 77K businesses

1. Business Data – Yelp dataset

* Data for 77K businesses
* 566K business attributes, e.g., hours, parking availability, ambience.

## **GitHub Repository**

<https://github.com/sriramrao1/project1/>

This is a public repository. So you should be able to make Pull requests.

I have provided ‘DwMcclary’ with access to contribute as well.

## **Project Repository File Structure:**

|  |  |
| --- | --- |
| **File Name** | **Description** |
| Dataset |  |
| Loaddata.sh | Script that loads data in HDFS files to be used in engine.py |
| Dbsetup.txt | Code to create the database tables |
| Server.py | Python code that creates a pyspark execution server where the app.py and engine.py applications can be executed  Copy this file to /data/project/ folder |
| App.py | Is a Flask web application with Python code for the REST API of the food ninja application |
| Engine.py | Python code that is invoked from app.py and performs the analytical processing |
| Postgres JDBC driver | The JDBC driver is required for copying the search results to the database |
|  |  |
| Readme.doc | This document with details on github repository, pre-requisites, data source parameters, and instructions to run the application |
| Architecture.pdf |  |
| Screenshots | Folder has the screenshots of execution of the application |
|  |  |

## **Pre-Requisites**

### Software:

1. Cherrypy
2. Flask
3. flask-restful
4. Paste
5. Postgres JDBC driver
6. UCB W205 Spring 2016 (ami-be0d5fd4)
7. Python – version 2.7 or higher

### Hardware:

1. Amazon EC2 Instance - m3.large
2. EBS Volume – 100 GB (to be attached as /data)
3. Ports to be opened: 8080, 5432, 22, 443

### Data:

* The primary dataset is Yelp’s challenge dataset. You can download the data from <https://www.yelp.com/dataset_challenge> The data is also made available in Amazon S3 -

### Others:

1. The AMI UCB W205 Spring 2016 (ami-be0d5fd4) along with its start-hadoop.sh scripts, start-postgres.sql scripts is the base configuration for the project
2. Postgres DB must be installed. The data source details are listed above
3. Most project-specific scripts are executed with w205 user

## **Data Source Parameters**

Database: foodninjadb

Tables: toprestaurants, topfoods, topcities

DB Username used in the code: w205

DB Password used in the code: postgres

## **Instructions to run the application:**

### **Application set up:**

|  |  |  |
| --- | --- | --- |
| **#** | **Instructions** | **Reference Script or Documentation** |
|  | Start a EC2 instance with UCB W205 Spring 2016 (ami-be0d5fd4) – AMI | Please refer to the pre-requisites for the ports to be opened |
|  | Have a 100 GB volume mounted to the EC2 instance as /data | mount -t ext4 /dev/xvdf /data |
|  | Start Hadoop | /root/start-hadoop.sh |
|  | Start Postgres database | /data/start\_postgres.sh |
|  | Switch to w205 user | su – w205 |
|  | Navigate to /data folder | ‘cd /data’ |
|  |  |  |
|  | Clone the scripts from my github repository and store in a EC2 instance folder | git clone <https://github.com/sriramrao1/project1.git> |
|  | Grant access to all users | ‘chmod a+rwx /data/project1’  ‘cd /project1’ |
|  | Copy the postgres JDBC driver to the /tmp/jars folder | ‘mkdir /tmp/jars’  mv postgresql-9.4.1208.jre6.jar /tmp/jars/postgresql-9.4.1208.jre6.jar |
|  | As root user Create the application database ‘foodninjadb’ and the 3 tables with owner permissions for the user name and password defined in the ‘Data Source Parameters’ section  (optionally in another terminal window) | Follow the steps listed in DBsetup.txt |
|  | As root user install cherrypy | ‘pip install cherrypy’ |
|  | As root user install flask | ‘pip install flask’ |
|  | As root user install flask-restful | ‘pip install flask-restful’ |
|  | As root user install paste | ‘pip install paste’ |
|  | Download the Dataset from S3 and copy it to  /data/project1 | Place instructions here |
|  | Copy the data to HDFS | hdfs dfs -mkdir /user/w205/project1  hdfs dfs -mkdir /user/w205/project1/yelpbusinessdata  hdfs dfs -mkdir /user/w205/project1/yelpreviewdata  hdfs dfs -put /data/project1/yelp\_academic\_dataset\_business.json  /user/w205/project1/yelpbusinessdata  hdfs dfs -put /data/project1/yelp\_academic\_dataset\_review.json /user/w205/project1/yelpreviewdata |

### **Instructions:**

|  |  |  |
| --- | --- | --- |
| **#** | **Instructions** | **Expected Result** |
|  | Validate that the /tmp/jars folder has the postgresql-9.4.1208.jre6.jar file |  |
|  | Navigate to /data/project1/ folder |  |
|  | Start the Server  $SPARK\_HOME/bin/spark-submit --jars /tmp/jars/postgresql-9.4.1208.jre6.jar --driver-class-path /tmp/jars/postgresql-9.4.1208.jre6.jar server.py |  |
|  | From a browser window:  Execute Restaurant Search from the browser (OR invoke the /restsearch/<string:city>/<string:food> REST API)  For example:  <http://ec2-54-89-102-75.compute-1.amazonaws.com:8080/restsearch/Las> Vegas/pizza | The first time a restaurant search request is made with a set of city, food parameters, the search takes about 3 min to complete. With this search the result is stored in the Postgres database  The second time the search is run with the same parameters, the results are returned in less than a second |
|  | Execute Food Search from the browser (OR invoke the /foodsearch/<string:city>/<string:restaurant> REST API):  For example:  http:// ec2-54-89-102-75.compute-1.amazonaws.com:8080/foodsearch/Phoenix/Vovomeena | I have not implemented the Data storage for this REST API  Search results will take about 3 min |
|  | Execute City Search from the browser (OR invoke the /citysearch/<string:category> REST API):  For example:  http://ec2-54-89-102-75.compute-1.amazonaws.com:8080/citysearch/Italian | The first time a city search request is made with a food category, the search takes about 1 min to complete. With this search the result is stored in the Postgres database  The second time the search is run with the same parameters, the results are returned in less than a second |
|  | You can also execute the above searches from the command line using  ‘curl’ in front of the API call. For example  curl 'http:// ec2-54-89-102-75.compute-1.amazonaws.com:8080/restsearch/Las Vegas/pizza' |  |

## **Production Architecture**

Below is schematic of the **proposed** Production Architecture of the end-to-end data processing pipeline application with a clearly defined data ingest layer, processing layer and data persistence layer.

* Data Ingest Layer: In the production environment, it is expected that the business data and the review data will be refreshed on an hourly or at the least daily schedule (VELOCITY). The data VOLUME is also expected to be large. The test dataset used in this Proof of Concept has data for about 300 medium and small cities. The dataset size for the 300 cities is 750 MB. This is expected to be in the several hundred GB range in Production. The data ingest layer therefore has to be robust. We propose a HDFS storage layer for the data that will be ingested by the data pipeline.
* Data Processing layer: The data pipeline is being configured to handle hundreds of requests that can processed by multiple workers in Production. Apache Spark with its versatile feature set that allows request jobs to be run with python, java and SQL is the most appropriate solution. The processing layer must also be run within a clustered environment.
* Data Persistence Layer: For fast (sub 2 seconds) responses, all known combinations of data inputs will processed in advance and materialized in a database. Based on our initial analysis, the application is expected handle about 50 concurrent requests a second. To support this concurrency with fast response times, the Spark responses will be cached in an Oracle database.
* Serving Layer: The requests for the food ninja food recommendation engine is expected from several third-party applications. REST API with its powerful yet simple interfaces provides the fastest option to serve the end users. The REST API will be architected with CherryPy and flask.

**PROCESSING LAYER**

**PERSISTENCE LAYER**

**SERVING LAYER**

Apache Spark

Python

Pyspark, SPARK SQL

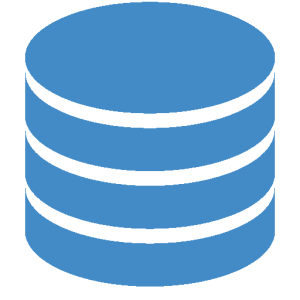
Postgres DB (OR)

Parquet Table

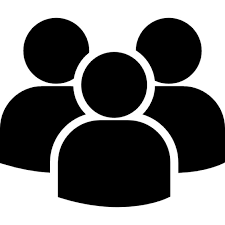
SQL, REST API

Flask

CherryPy



app.py



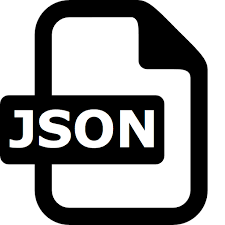
engine.py

server.py

Hadoop HDFS

Shell Scripts

**DATA INGEST LAYER**



Restaurant Info

Food Info

loaddata.sh

Review Data

Business Data

***Spark CLUSTER***

***ORACLE***

***WEb FRONT-END***

## **Proof of Concept Architecture**

This proof of concept is being developed to quickly demonstrate the feasibility to provide a food recommendation engine with Yelp data. Therefore we have made a few changes to the Production Architecture described above. For the most part the application will follow the layered architecture described above. However the following changes have been made:

* The data pipeline is executed in a single EC2 instance
* The data is persisted in a Postgres database
* When a request is received for the first time from a REST API call, it triggers a pyspark/Spark-sql batch job. The results of that job will be stored in the database for subsequent requests. Therefore initial requests are expected to be slow. Similar requests made later will be served from the database and are therefore expected to be quicker.
* The proof of concept solution has been developed for the following three REST API calls:
  + Restaurant Search with City and Food as input parameters: /restsearch/<string:city>/<string:food>
  + Food Search with City and Restaurant as input parameters: /foodsearch/<string:city>/<string:restaurant>
  + City Search with Food category as input parameter: /citysearch/<string:category>

**PROCESSING LAYER**

**PERSISTENCE LAYER**

**SERVING LAYER**

Apache Spark

Python

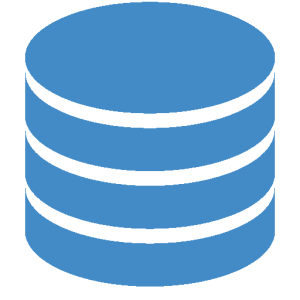
Pyspark, SPARK SQL

Postgres DB

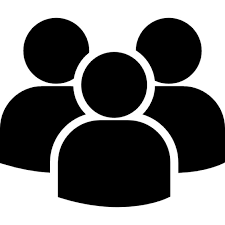
SQL, REST API

Flask

CherryPy



app.py



***AMAZON EC2 Instance***

**DATA INGEST LAYER**

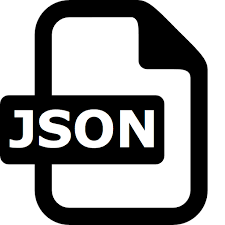
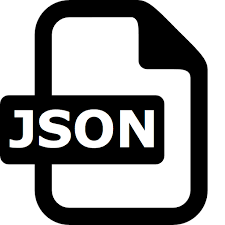


engine.py

server.py

Hadoop HDFS

Shell Scripts



Restaurant Info

Food Info

loaddata.sh

Review Data

Business Data

## **Data Model**