# Dengue Prediction System

MIDS W205 Final Project – Summer '17

Felipe Campos, Frank Shannon, Josh Wilson and Matthew Holmes

# Architectural Documentation

## Introduction

The Dengue Prediction System is a full machine learning pipeline application that runs analysis on dengue historical data and uses a predictive model to indicate potential cases of dengue for the following week.

This document aims to explain in detail the architecture of the application, each component involved, pre-requisites and what each script or program does.

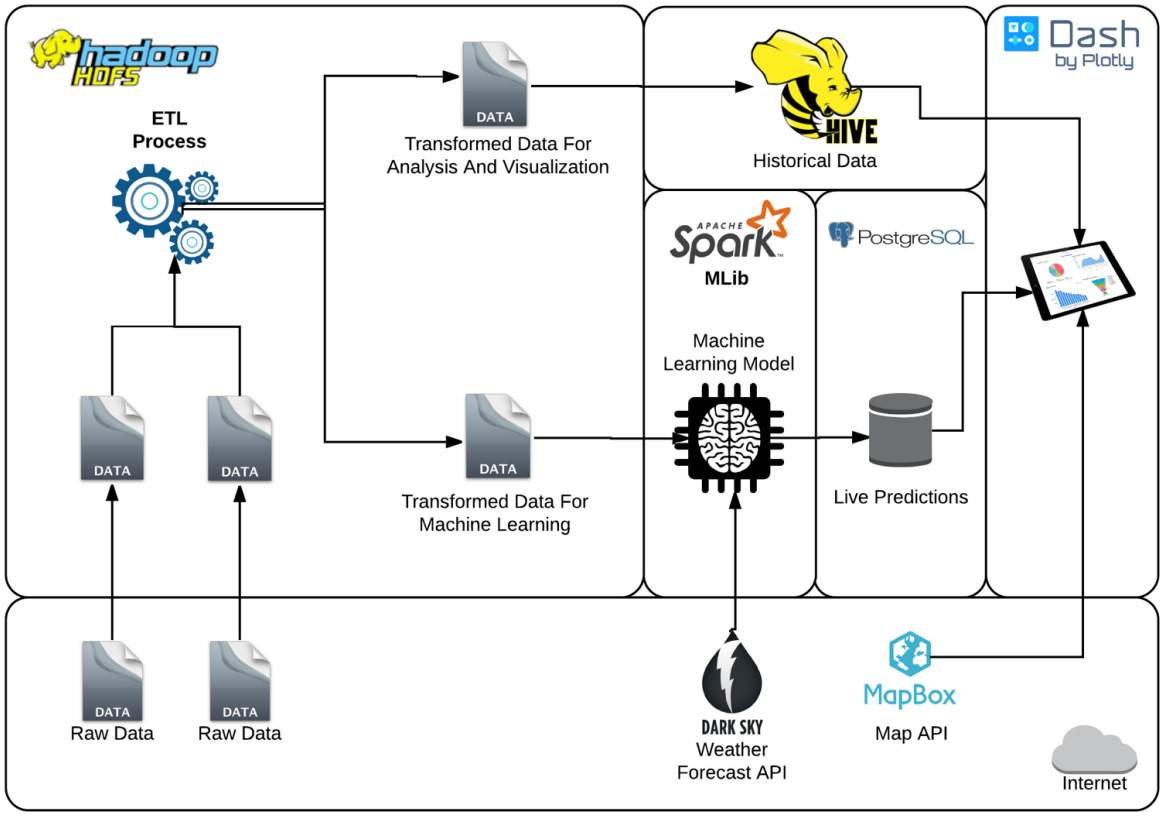
## About the Application

This software uses dengue historical data from different sources, merges them together into a single, unified dataset and uses it for training a machine learning model to predict future cases of dengue.

All the data can be visualized in a single place, via a Web browser, where the user has access to both the historical data presented in the model as well as a geo-located live map displaying potential occurrences for dengue for the next seven days.

## Application Architecture

The Dengue Prediction System was built using a variety of different technologies in order to produce a fully functional, easily updateable machine learning model. The architecture diagram displaying the technologies used, how they are interconnected as well as the sources of data and its flow inside the pipeline is depicted below.




The pipeline begins in the Data Layer, by gathering raw data from the Internet and storing it. Currently, the data being used is a dataset from DengAI containing historical dengue records from San Juan and Iquitos, already formatted with variables of interest and dengue notification data from DATASUS, the Brazilian Ministry of Health database, which records dengue occurrences on a case-by-case basis and on neighborhood level. As the Brazilian DATASUS dataset contains only the notifications, we also need to pull historical weather data from Brazil to be merged into this dataset. A PySpark script is then used to perform the ETL process of merging the Brazilian DATASUS and weather datasets, then transforming it and merging the data with the DengAI data thus forming a single dataset with a common format. This resulting dataset is then written back into HDFS in two different formats: one for analysis and reporting, and the other with some transformations required for use in Machine Learning. Finally, a Hive/Spark-SQL schema is created on top of the analytical dataset so that it can be easily used in the visualization layer.

Analyzing the volume of data currently used, one may ask why the architectural decision of using HDFS instead of a much simpler architecture. First of all, HDFS provides reliability. The data and model can be stored in a distributed fashion, ensuring data redundancy and availability that other solutions can't offer. Second, scalability. Although the current volume of data is not so big to justify the usage of Hadoop, we are only currently predicting six cities and the system could eventually grow to support worldwide prediction. Doing so, however, would require keeping in store weather, socioeconomic, geographical and dengue case history data for every city in the world. The increase in volume of data to

expand the system in the future would then require the usage of a distributed, large scale storage system such as HDFS.

The second step in the pipeline is the Machine Learning layer. The ML layer uses Spark MLLib to load the transformed dataset and train a model to predict dengue cases according to the weather forecast. The transformed data is read and used as input to a machine learning algorithm (currently, Gradient Boosted Trees). After the model has been fitted, the model itself is written back to HDFS. This is done because according to the amount of data, machine learning algorithm used and also how large is the parameter space used for cross-validation and tuning the model, the model training step may take a long time to complete Writing the model back to HDFS allows the model to be trained only once and loaded when required, which is much faster than re-training the model every time. As the model is loaded when the Prediction system starts, it also allows the model to be re-trained using new data while the prediction system is running and therefore allows for much smaller downtime.

The third step consists of the Streaming and Prediction layer. The streaming layer runs as a background service and also leverages Spark MLLib, by loading the trained model from HDFS when it starts. It then uses the model to analyze real-time weather forecast data and perform a prediction on the number of cases of dengue for each city in the next seven days. This weather forecast data is gathered by the application in regularly timed intervals from the Dark Sky Weather Service through their provided API. The number of predicted cases of dengue for each week is then kept up-to-date in a PostgreSQL database.

The reason for using a PostgreSQL database in this case, and not HDFS, is that analyzing the weather forecast regularly will also require frequent atomic writes to a database, and atomic reads and writes are better performed by traditional relational databases than large-scale, distributed file systems such as HDFS.

The last layer in the architecture is the visualization layer. In order to provide users with the insights collected by the whole system, a web interface has been developed using Dash. Dash is a tool created by Plotly that allows serving graphs and building interfaces programmatically via a web server, while also allowing live updates of the graphs and dynamic user interaction such as filtering. The Dash interface connects to Hive when it starts, gathers the historical Dengue data stored through the Hive schema and caches it for fast user analysis and interaction. In regularly timed intervals, Dash accesses the PostgreSQL and then collects the current dengue prediction for the next seven days and plots it over a map for user visualization. Dash can, then, combine both sources of data into a single point for user access. The visualization interface also connects to the Mapbox service to retrieve the maps on which the predictions are plotted, allowing for high customization of the map.

Finally, according to user preference, both Hive and PostgreSQL data can be accessed externally through the use of other technologies for data visualization and analysis such as Tableau.

## Directory and File Structure

The following is an explanation of the purpose of each file and folder in the application directory.

- **data_layer:** folder containing the Data Layer files and scripts
    - **create_schema.sql:** Hive/SparkSQL schema creation script
    - **load_data_lake.sh:** Bash shell script that downloads the data and loads it into HDFS

    - **transform_data.sh:** PySpark script that performs the transformation of the original data files for machine learning and analysis
- **docs:** folder containing project documentation
    - **data_exploration:** contains a notebook used for initial data exploration
        - **Dengue_Exploration.ipynb:** data exploration notebook
    - **presentations:** contains conducted project presentations
        - **Project Progress Presentation.pptx:** mid-project progress presentation
        - **Project Proposal Presentation.ppt:** initial project proposal presentation
    - **architecture.pdf:** this file
- **logs:** the script for a guided application configuration
    - **streaming.log:** log file for the streaming layer
    - **visualization.log:** log file for the visualization layer
- **ml_layer:** folder containing the Machine Learning Layer files and scripts
    - **delete_model.sh:** script for deleting the currently persisted model
    - **train_ml_model.py:** PySpark script that uses Spark MLLib to train and persist a new model
- **streaming_layer:** folder containing the Streaming Layer files and scripts
    - **create_postgres_db.py:** Python script that creates the PostgreSQL database and tables
    - **weather_updater.py:** PySpark script that continuously gathers weather forecast data, runs a prediction using the Spark MLLib model and writes results to the PostgreSQL database
- **visualization_layer:** folder containing the Visualization Layer files and scripts
    - **dashboard.py:** Python script that runs the web server serving the data visualization
- **install_prerequisites.sh:** Bash shell script that installs all requirements to run the application in an automated fashion
- **README.md:** application README file with a quick-start on application setup and use
- **setup_data_layer.sh:** Bash shell script that configures and initializes/updates the data layer
- **setup_ml_layer.sh:** Bash shell script that initializes/updates the ML layer by updating the trained predictive model
- **setup_streaming_layer.sh:** Bash shell script that initializes/updates the Streaming layer by creating the PostgreSQL database and tables and setting the required Dark Sky API Key
- **setup_visualization_layer.sh:** Bash shell script that sets the required Mapbox Access Token for the visualization layer
- **start_application.sh:** runs the prediction and web server application on the background
- **stop_application.sh:** stops the application running on the background

## Pre-Requisites

This application was built to be executed using the **UCB W205 Spring 2016 (ami-be0d5fd4)** AMI, available on the AWS Marketplace. The AMI should have some of the basic requirements already installed and configured, such as Apache Hadoop, Apache Spark and PostgreSQL Server. **Please ensure you have followed directions for setting up and configuring the AMI properly.** Those directions are in the form of labs available at https://github.com/UC-Berkeley-I-School/w205-summer-17-labs-exercises.

Make sure you have followed Lab 1 (instantiate EC2 instance), Lab 2 (set up Hadoop and PostgreSQL) and Step 3 of Lab 3 (set up Apache Spark).

There are other requirements for the application to work properly. There is an automated script in the root folder of the project that will install all of them. To use it, as **root**, run the command:

```
#./install_prerequisites.sh
```

Please note that the application uses two different versions of Python. That is because the AMI has two versions installed, and while some of the components in the architecture use Python 2.6, others require Python 2.7 to work properly. That is the reason the script uses both *pip2.7* and *pip*.

The full list of pre-requisites is, all installed by the script mentioned above, is:

**Python 2.7**

- python27-devel system package (installed using Yum)
- cyrus-sasl-devel system package (installed using Yum)
- Dash Python package (version 0.17.7, installed using pip2.7)
- Dash-renderer Python package (version 0.7.4, installed using pip2.7)
- Dash-html-components Python package (version 0.7.0, installed using pip2.7)
- Dash-core-components Python package (version 0.12.0, installed using pip2.7)
- Plotly Python package (version 2.0.13, installed using pip2.7)
- Numpy Python package (installed using pip2.7)
- Pandas Python package (installed using pip2.7)
- Psycopg2 Python package (installed using pip2.7)
- sasl Python package (installed using pip2.7)
- Thrift Python package (installed using pip2.7)
- Thrift-sasl Python package (installed using pip2.7)
- PyHive Python package (installed using pip2.7)
- Forecastio Python package (installed using pip2.7)

**Python 2.6**

- Numpy Python package (installed using pip)
- Pandas Python package (installed using pip)
- Psycopg2 Python package (installed using pip)
- Forecastio Python package (installed using pip)

In addition to the software packages mentioned above, you'll also need keys and tokens for two Web Services. They are:

**API Keys and Access Tokens**

- Dark Sky Weather API Key (freely obtained at https://darksky.net/dev)
- Mapbox Access Token (freely obtained by registering for a Starter account at https://www.mapbox.com)

## Using the Application

The README.md file provides a step-by-step, detailed configuration and usage guide that should be followed to get the application up and running. Please use the instructions in that file to configure and run the application.

Before starting the application, please make sure you have the following services up and running on your EC2 instance:

- PostgreSQL Server (as **root**, run */data/start_postgres.sh*)
- Hadoop (as **root**, run */root/start-hadoop.sh*)
- Hive Metastore (as **w205**, run */data/start_metastore.sh*)
- Hive Server 2 (as **w205**, run *hive –service hiveserver2 &*)

If you run into any problems or issues while configuring or running the application, please don't hesitate to get in touch with the developers or post an issue at our project GitHub page.