CentraleSupélec

CENTRALESUPELEC

MSDI - METZ

Tweetoscope

Groupe 02

Anas LAAROUSSI
Charaf BOUCHTIOUI
Chen SIYUAN

1er décembre 2020

Table des matières

0.1 Introduction

This document describes the main stages of the development of the Tweetoscope application, the main implemented features, the deployment and architecture of the application.

0.2 Development and integration

The CI/CD pipeline is divided to three stages :

- `Build` : To test the compilation with cmake of all c++ code component as well as the doxygen generated documentation. The documentation is pushed to gitlab pages whenever the master branch is modified or a new tag is added (stable version of the software).

- `Unit_tests` : Some unit_tests are performed on the collector to evaluate its behaviour regarding sending partial cascades and terminating cascade since it is a crucial step of the application.

- `Create Docker images` : Docker images are automatically created and pushed to the dockerhub.

0.3 Main features of Tweetoscope software

`Generator` : The generator is in charge of injecting tweets and retweets into Kafka streams.

`Collector` : Collects tweets and retweets and sends partial cascades (a cascade is a series of a tweet and its retweets) according to a set of observation windows. It also manages cascade termination to send the true length of the cascade (Tweet popularity).

Estimator : Uses Marked Hawkes process with exponential kernels to estimate the tweet popularity based on the history of retweets.

`Predictor` : The predictor receives random forest models trained offline on Hawkes parameters to adjust predictions. It also receives the real length of a cascade when it considered terminated, and sends statistics (ARE) of the predictions alongside with samples to train the RF models.

`Learner` : Retrieves samples sent by the predictor from Kafka streams, when a mini batch of samples for a specific observation window is received, it triggers the training of the models (each observation window is associated with a model). When a batch of data is received, it triggers the hypertunning of the random forests. The trained models are sent to the predictor to improve the predictions.

`Monitor` : Acts as a Prometheus client client, retrieves the ARE metric from Kafka streams and sent it the a Prometheus push gateway.

`Dashboard` : Acts as a Prometheus client, retrieves predictions and sends alerts to a Prometheus push gateway.

The popularity is received and displayed by the Prometheus Alert Manager which could be configured to send emails or notifications in case of predicting that a tweet would be very popular. The ARE is also received by Prometheus server, which could be plugged in as a Data Source for Grafana to keep monitoring the accuracy of the predictions.

0.4 Architecture and Deployment

Our vision for deployment is to have a scalable and maintainable application. To do so, we first made the application services as modular as possible. Then we created each topic in our broker with four partitions and mapped the messages with same keys in the same topic partitions even if they are produced by 2 different nodes (the case for cascade properties), this allow us for example to launch 2 instances of the learner in case we have two observation windows. We also launch the maximum number of Estimator (4) since it is the critical node in our application and it needs to forward the estimations to the predictor as soon as possible to make predictions in real time.

In our strategy for deploying the application on kubernetes, we first deploy the middleware (kafka and zookeper) alongside with the prometheus server, prometheus push gateway and the prometheus alert manager as services and then we expose the prometheus server via nodeport 30187 and the alert manager via 30189. this enables us to launch third parties tools for visualization and monitoring outside of our cluster (like grafana).

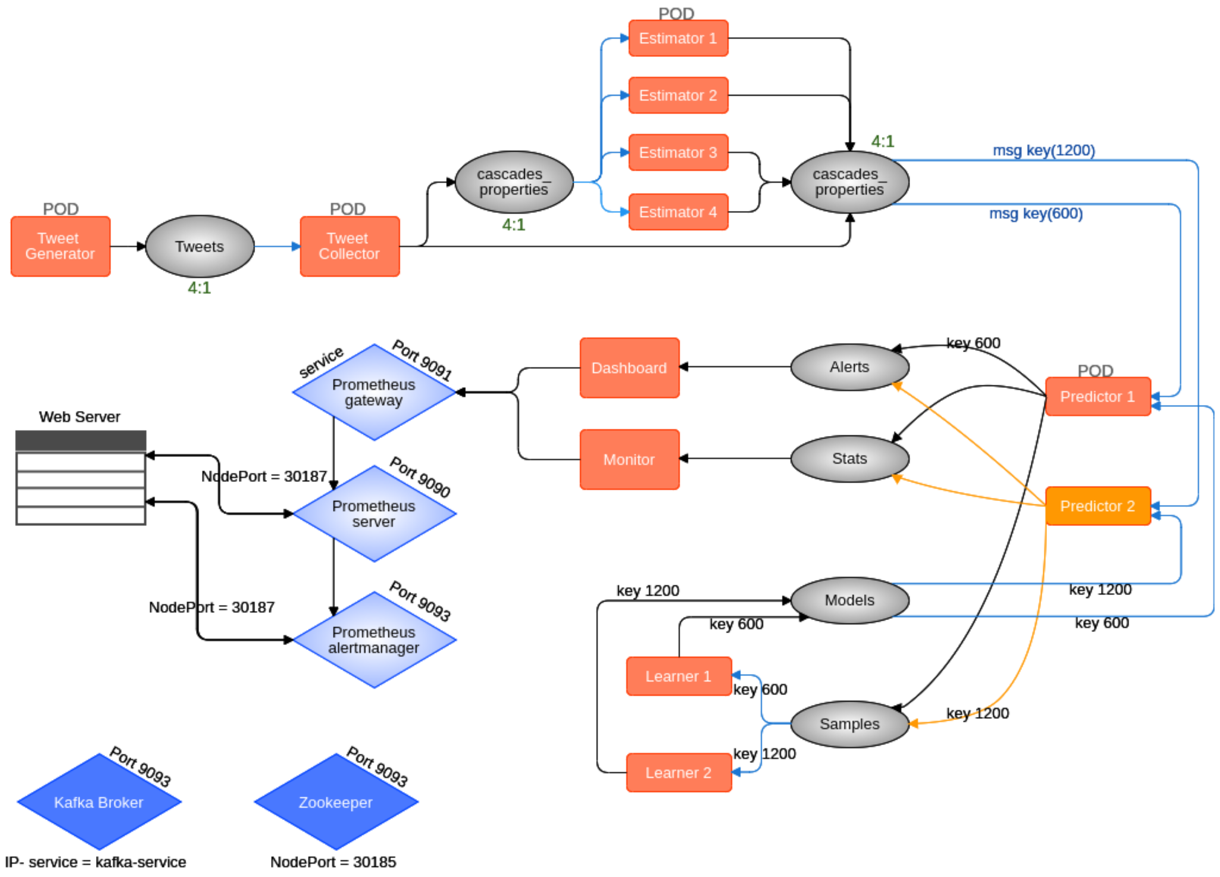

FIGURE 1 – Tweetoscope Architecture