uberNow Documentation

Release 1.1.0

Anirban Roy Das

Sep 27, 2017

Contents

It is an app to notify users via email as to when to book an uber as per given time to reach a specified destination from a specified source.

Home Page : https://www.github.com/anirbanroydas/uberNow

CHAPTER 1

Details

Author Anirban Roy Das

Email anirban.nick@gmail.com

CHAPTER 2

Documentation:

Overview

uberNow is a small web app which takes four inputs from the user and notifies the user via email as to when to book an Uber.

All you(the user) have to do is give the 4 required inputs and submit the request. The app will take care of the rest.

The inputs are:

- 1. source
- 2. destination
- 3. email address
- 4. time to reach destination

It uses Tornado as the application server. The web app is created using the sockjs protocol. **SockJS** is implemented in many languages, primarily in Javascript to talk to the servers in real time, which tries to create a duplex bi-directional connection between the **Client(browser)** and the **Server**. Ther server should also implement the **sockjs** protocol. Thus using the sockjs-tornado library which exposes the **sockjs** protocol in Tornado server.

It first tries to create a Websocket connection, and if it fails then it fallbacks to other transport mechanisms, such as **Ajax**, **long polling**, etc. After the connection is established, the tornado server **(sockjs-tornado)** calls Uber Apis and Google Maps Apis to process the requests.

Problem Statement

Short Version

Given four inputs, (a). **Source** (b). **Destination** (c). **Email Address** (d). **Time of the day to reach the destination**, send an email to the given email address about booking its uber at that very moment so that you have ample time to reach your destination which takes into consideration the following conditions:

- Time the uber driver will take to reach your source.
- Time it will take to reach your destination in the current live traffic conditions.
- Whether you will be actually be able to book an uber at that very moment or not or will it take some time before you can actually book a cab.
- The actual time when you reach your destination should be as accurate or close to the pre defined time as possible.

Long Version with example

Given four inputs, (a). **Source** (b). **Destination** (c). **Email Address** (d). **Time of the day to reach the destination**. Assume that source and destination are provided as latitude/longitude and not as addresses(to keep things simple). The app needs to find the exact time a user needs to book an uber to be at that destination at that time. An email needs to be sent to the above email ID at this time saying "Time to book an uber!"

Example: I am in Koramangala(12.927880, 77.627600) and need to be in Hebbal(13.035542, 77.597100) at 8:00 PM for a meeting. The app will have to email me at 6:43 PM because the uberGO will take 9 mins to reach me at 6:43 PM(as per uber) and it takes 68 mins to drive from Koramangala to Hebbal(as per google maps)

Assumptions:

- To keep things simple, whatever the uber api and google maps api tell is true.
- Time to arrive at the destination is within today. The time entered is in IST.
- Not considering cases where it's already too late to book the uber. Assuming that there is still some time left.
- The maximum deviation of driving times at any time of the day is 60 mins. That means, if it takes 40 mins to drive from Koramangala to Hebbal now, assume that it's not more than 100 mins at any time of the day.
- The cab is UberGO.
- Accuracy is the most important factor.
- Second important factor is optimization of requests to the APIs. Pinging the APIs every minute to see if you should leave now is not the best solution.

Clarifications:

1. Lets say you want to reach destination B at 8:00 pm starting from source A at 7:30 pm, the travel time is 25 minutes but the next time you pool google api, it says now the travel time is 40 mins. In such case its already to late to book an uber.
2. Lets say, first time you pool google api and it says that it will take 1 hour from source A to destination B. But next time you check the google api, it says 1:45 mins (Huge Traffic Jam) and then next time it say 2:15 mins (It started raining :( ). You can assume that the maximum deviation is 1 hour at any time of the day i.e max it will take 2 hours to reach destination.

Features

Technical Specs

sockjs-client (optional) Advanced Websocket Javascript Client used in **webapp example**

Tornado Async Python Web Library + Web Server

sockjs-tornado SockJS websocket server implementation for Tornado

Uber Time-Estimation Apis HTTP Rest APIs to estimate time required to book an uber at given time

Google Maps Distance-Matrix Apis HTTP Rest Apis to calculate distance and duration to reach from source to destination

pytest Python testing library and test runner with awesome test discobery

pytest-flask Pytest plugin for flask apps, to test fask apps using pytest library.

Uber's Test-Double Test Double library for python, a good alternative to the mock library

Jenkins (Optional) A Self-hosted CI server

Travis-CI (Optional) A hosted CI server free for open-source projecs

Docker A containerization tool for better devops

Features Specs

- Web App
- Email Notification
- Uber Booking Reminder
- Microservice
- Testing using Docker and Docker Compose
- CI servers like Jenkins, Travis-CI

Installation

Prerequisite (Optional)

To safegurad secret and confidential data leakage via your git commits to public github repo, check `git-secrets`.

This git secrets project helps in preventing secrete leakage by mistake.

Dependencies

1. Docker
2. Make (Makefile)

See, there are so many technologies used mentioned in the tech specs and yet the dependencies are just two. This is the power of Docker.

Install

- **Step 1 - Install Docker**

  Follow my another github project, where everything related to DevOps and scripts are mentioned along with setting up a development environemt to use Docker is mentioned.

  – Project: https://github.com/anirbanroydas/DevOps

  – Go to setup directory and follow the setup instructions for your own platform, linux/macos

- **Step 2 - Install Make**

  ```
  # (Mac Os)
  $ brew install automake

  # (Ubuntu)
  $ sudo apt-get update
  $ sudo apt-get install make
  ```

- **Step 3 - Install Dependencies**

  Install the following dependencies on your local development machine which will be used in various scripts.

  1. openssl
  2. ssh-keygen
  3. openssh

CI Setup

If you are using the project in a CI setup (like travis, jenkins), then, on every push to github, you can set up your travis build or jenkins pipeline. Travis will use the `.travis.yml` file and Jenknis will use the `Jenkinsfile` to do their jobs. Now, in case you are using Travis, then run the Travis specific setup commands and for Jenkins run the Jenkins specific setup commands first. You can also use both to compare between there performance.

The setup keys read the values from a `.env` file which has all the environment variables exported. But you will notice an example `env` file and not a `.env` file. Make sure to copy the `env` file to `.env` and **change/modify** the actual variables with your real values.

The `.env` files are not commited to git since they are mentioned in the `.gitignore` file to prevent any leakage of confidential data.

After you run the setup commands, you will be presented with a number of secure keys. Copy those to your config files before proceeding.

NOTE: This is a one time setup. **NOTE:** Check the setup scripts inside the `scripts/` directory to understand what are the environment variables whose encrypted keys are provided. **NOTE:** Don't forget to **Copy** the secure keys to your `.travis.yml` or `Jenkinsfile`

NOTE: If you don't want to do the copy of `env` to `.env` file and change the variable values in `.env` with your real values then you can just edit the `travis-setup.sh` or `jenknis-setup.sh` script and update the values their directly. The scripts are in the `scripts/` project level directory.

IMPORTANT: You have to run the `travis-setup.sh` script or the `jenkins-setup.sh` script in your local machine before deploying to remote server.

Travis Setup

These steps will encrypt your environment variables to secure your confidential data like api keys, docker based keys, deploy specific keys.

```
$ make travis-setup
```

Jenkins Setup

These steps will encrypt your environment variables to secure your confidential data like api keys, docker based keys, deploy specific keys.

```
$ make jenkins-setup
```

Usage

After having installed the above dependencies, and ran the **Optional** (If not using any CI Server) or **Required** (If using any CI Server) **CI Setup** Step, then just run the following commands to use it:

You can run and test the app in your local development machine or you can run and test directly in a remote machine. You can also run and test in a production environment.

Run

The below commands will start everythin in development environment. To start in a production environment, suffix `-prod` to every **make** command.

For example, if the normal command is `make start`, then for production environment, use `make start-prod`. Do this modification to each command you want to run in production environment.

Exceptions: You cannot use the above method for test commands, test commands are same for every environment. Also the `make system-prune` command is standalone with no production specific variation (Remains same in all environments).

- **Start Applcation**

  ```
  $ make clean
  $ make build
  $ make start

  # OR

  $ docker-compose up -d
  ```

- **Stop Application**

  ```
  $ make stop

  # OR

  $ docker-compose stop
  ```

- **Remove and Clean Application**

  ```
  $ make clean

  # OR

  $ docker-compose rm --force -v
  $ echo "y" | docker system prune
  ```

- **Clean System**

```
$ make system-prune

# OR

$ echo "y" | docker system prune
```

Logging

- To check the whole application Logs

```
$ make check-logs

# OR

$ docker-compose logs --follow --tail=10
```

- To check just the python app's logs

```
$ make check-logs-app

# OR

$ docker-compose logs --follow --tail=10 identidock
```

API

This contains all the modules and classes used to make the app. You can go through each of them for better understanding of the project.

Main View

IndexHandler

BookingHandler

Api Calls Module

Api

Testing

Now, testing is the main deal of the project. You can test in many ways, namely, using `make` commands as mentioned in the below commands, which automates everything and you don't have to know anything else, like what test library or framework is being used, how the tests are happening, either directly or via `docker` containers, or may be different virtual environments using `tox`. Nothing is required to be known.

On the other hand if you want fine control over the tests, then you can run them directly, either by using `pytest` commands, or via `tox` commands to run them in different python environments or by using `docker-compose` commands to run differetn tests.

But running the make commands is lawasy the go to strategy and reccomended approach for this project.

NOTE: Tox can be used directly, where `docker` containers will not be used. Although we can try to run `tox` inside our test contianers that we are using for running the tests using the `make` commands, but then we would have to change the `Dockerfile` and install all the `python` dependencies like `python2.7`, `python3.x` and then run `tox` commands from inside the `docker` containers which then run the `pytest` commands which we run now to perform our tests inside the current test containers.

CAVEAT: The only caveat of using the make commands directly and not using `tox` is we are only testing the project in a single `python` environment, nameley `python 3.6`.

- To Test everything

  ```
  $ make test
  ```

  Any Other method without using make will involve writing a lot of commands. So use the make command preferrably

- To perform Unit Tests

  ```
  $ make test-unit
  ```

- To perform Component Tests

  ```
  $ make test-component
  ```

- To perform Contract Tests

  ```
  $ make test-contract
  ```

- To perform Integration Tests

  ```
  $ make test-integration
  ```

- To perform End To End (e2e) or System or UI Acceptance or Functional Tests

  ```
  $ make test-e2e

  # OR

  $ make test-system

  # OR

  $ make test-ui-acceptance

  # OR

  $ make test-functional
  ```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search