ASIAN BANKING SCHOOL

Let's Talk Digital Series #14

Run Your Code without Managing Servers

RUN YOUR CODE *WITHOUT MANAGING SERVERS*

Most data professionals run their Python or R scripts on notebooks hosted on their local machines, or on the cloud like AWS' SageMaker, GCP's AI Platform notebook, or even Google Colab. After writing code, it's likely your code will be used:

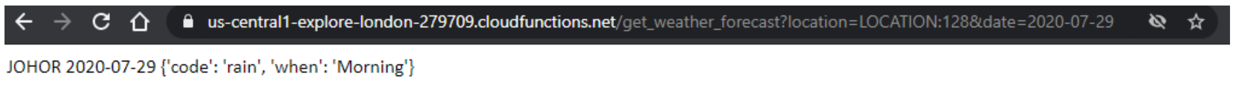
1. To send notifications to users in an app
2. To perform data cleaning and maintenance
3. To run tasks in the cloud instead of on the app
4. To integrate your code with third-party services or APIs or etc.

Traditionally, most of these applications would require your code to be hosted in an on-site server for it to run in a production environment. This obviously comes at a cost from server maintenance and other hardware costs. There are now cheaper options where you can write code, and deploy into the cloud for a production environment, without the hassle of managing servers or instances. One example of this is using Google Cloud Functions . This example is applicable to AWS' Lambda as well.

Google's Cloud Functions is a serverless environment for developers to build and connect cloud services. All developers need to do is to write simple functions which are automatically triggered when an event being watched happens. Here is a simple example: notice the URL below has several parameters: `location` and `date`. This is a cloud function I built to extract weather forecast data using APIs offered by the Malaysian Meteorological Department for Pagoh for 29 July 2020.

https://us-central1-explore-london-279709.cloudfunctions.net/get_weather_forecast?location=LOCATION:128&date=2020-07-29

Using a browser to head to the URL gives you a snapshot below.

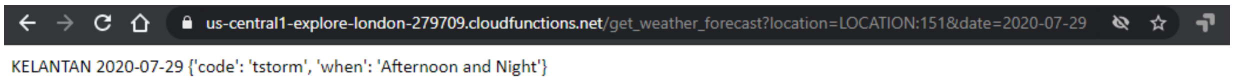


```
JOHOR 2020-07-29 {'code': 'rain', 'when': 'Morning'}
```

The URL returns the weather forecast for a location in Johor on a specified date.

Varying the URL to a location in Kelantan gives

https://us-central1-explore-london-279709.cloudfunctions.net/get_weather_forecast?location=LOCATION:151&date=2020-07-29



```
KELANTAN 2020-07-29 {'code': 'tstorm', 'when': 'Afternoon and Night'}
```

https://cloud.google.com/functions
https://aws.amazon.com/lambda/
https://api.met.gov.my/

The URL now returns the weather forecast for a location in Kelantan on a specified date.

At the time of writing, the MET API Version 1 is working but due to be replaced with API Version 2 to be released at an unspecified date.

Feel free to try varying the location and date parameters. A sample of 10 location codes are listed as below:

ID	NAME
LOCATION:135	Segamat
LOCATION:123	Batu Pahat
LOCATION:157	Pasir Mas
LOCATION: 426	Serdang
LOCATION:122	Ayer Hitam

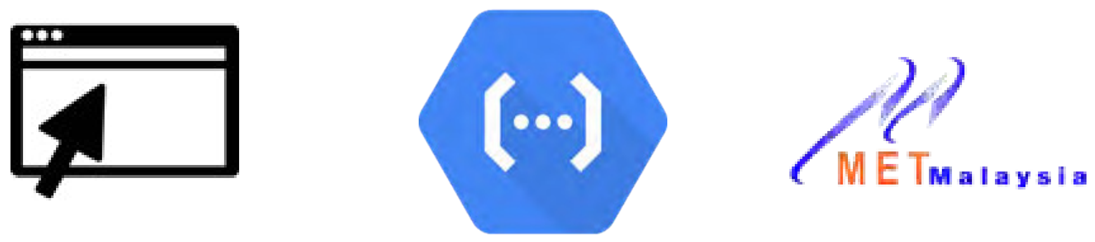
ID	NAME
LOCATION:154	Kuala Krai
LOCATION:128	Pagoh
LOCATION:410	Kubang Pasu
LOCATION:156	Machang
LOCATION:153	Kota Bahru

```python
import requests

def get_location_date(request):
  location = request.args.get('location', 'LOCATION:237')
  date = request.args.get('date', '2020-07-27')
  location_date = get_weather_forecast(location, date)
  return location_date

def get_weather_forecast(location, date):
  headers = {'Authorization': 'METToken __INSERT TOKEN HERE__'}
  params = (
      ('datasetid', 'FORECAST'),
      ('datacategoryid', 'GENERAL'),
      ('locationid', location),
      ('start_date', date),
      ('end_date', date),
  )
  response = requests.get('https://api.met.gov.my/v2/data', headers=headers, params=params)
  sample = response.json()
  return(str(sample["results"][5]['locationrootname'])+" "+str(date)+" "+str(sample["results"][5]
['attributes']))
```

get_location_date(request) accepts the location and date parameters from the URL. The default parameters have been set for 'LOCATION:237' and '2020-07-27' for demonstration purposes.

get_weather_forecast(location, date) takes the location and date parameters, feeds it into the MET API, and processes the response for display on a website.

The architecture diagram for this script is a simple one:



1. Using an internet browser, the user heads to the URL which has the query parameters.
2. Google's Cloud Function will process the parameters and query MET API to get the relevant data.
3. The resulting data is then sent back to the user.

This cloud function has been programmed to respond to a HTTP request, which means the parameters the users are looking for are available within the URL. The response can be displayed on a website or be sent to an app for user consumption purposes.

Such functions can be configured to run at a certain frequency. One example would be a data cleaning function to run at the end of every day. The function can also be triggered in the event of new information being made available, i.e. a financial transaction to update a customer database etc. A common application would be to integrate a function with third-party services and APIs.

The advantage of using Functions-as-a-Service (FaaS) like Cloud Functions or Lambda, is the ease of deploying functions to a production environment, and the low cost involved (since no hardware costs are involved).

If you would like to do some hands on work with Google Cloud Functions, there are tutorials available on Tutorials | Cloud Functions Documents . A few example architectures can also be found on What can I do with Cloud Functions?

https://cloud.google.com/functions/docs/tutorials
https://firebase.google.com/docs/functions/use-cases