

Fastcampus Sprint - Programming

Day 4. Advanced Web Scraping

Index

- Advanced Web Scraping with Selenium
- Web Scraping with cloud service

Requirements

- `$ pip install selenium`
- ChromeDriver: <https://chromedriver.storage.googleapis.com/index.html?path=78.0.3904.11/>

requests & BeautifulSoup

- 정적인 페이지를 수집할 때
- requests: HTTP 요청 -> HTML 응답
- BeautifulSoup: HTML 응답 -> 분석 후 요소 접근

But..

- BeautifulSoup은 AJAX나 javaScript로 그려지는(렌더링) 요소나 행동은 접근할 수 없음

Selenium!

- Web Application User test tool
- `$ pip install selenium`
- with webdriver

Pros & Cons

Pros

- 동적 페이지 제어 가능
- 사용자처럼 행동 가능
- iframe 제어 가능

Cons

- 느림
- BS4에 비해 신경써야 할 것이 많음

Route is important while using Selenium

- BeautifulSoup : 수집할 요소 선택 -> url 정보 수집 -> 스크래핑 수행
- Selenium: 수집할 요소 선택 -> 요소까지의 경로 선정 -> 스크래핑 수행

Web Scraping with Selenium

```
from selenium import webdriver  
ch_driver = webdriver.Chrome(<your webdriver path>)  
ch_driver.get('https://www.google.com/')
```

N사 포털 카페 서비스

| 특정 카페의 검색 결과물을 가져와봅시다.

Key script

```
query_input = ch_driver.find_element_by_id("<id of element>")  
query_input.send_keys(<query>)  
ch_driver.execute_script(<javascript>)  
ch_driver.switch_to_frame(<element>)
```

Twitter

더 많은 데이터 로딩을 위해 스크롤 후 데이터를 가져와봅시다.

```
ch_driver.execute_script("window.scrollTo(0, window.scrollY + 1000)")
```

Google Cloud

Cloud?

- 인터넷에 연결된 다른 컴퓨터로 연산을 하는 기술
- 접근성, 주문형 서비스 제공으로 경제적이고 효율적인 컴퓨팅 서비스 제공
- Amazon Web Service(Amazon), Google Cloud Platform(Google), Microsoft Azure(Microsoft), ..
- Virtual Machine, Cloud Storage, Database, Docker Engine 등 다양한 서비스 제공

Google Cloud Platform


- 2011년 Google이 출시한 클라우드 컴퓨팅 솔루션서비스
- 20개의 region과 61개의 zone 서비스 중
- 2020년 서울 region 오픈예정
- <https://cloud.google.com/>

Google Cloud Functions

- Pricing: <https://cloud.google.com/functions/pricing>

Google Cloud Functions

COMPUTE

 App Engine >

 Compute Engine  >

 Kubernetes Engine >

 Cloud Functions 

Google Cloud Functions

Google Cloud Functions

Google Cloud Functions is a lightweight, event-based, asynchronous compute solution that allows you to create small, single-purpose functions that respond to cloud events without the need to manage a server or a runtime environment

Create function

Google Cloud Functions



Cloud Functions



Create function

Name ?

get-nv-querys

Memory allocated

256 MB

Trigger

HTTP

URL

Google Cloud Functions

Source code

- ☒ Inline editor
- ☐ ZIP upload
- ☐ ZIP from Cloud Storage
- ☐ Cloud Source repository

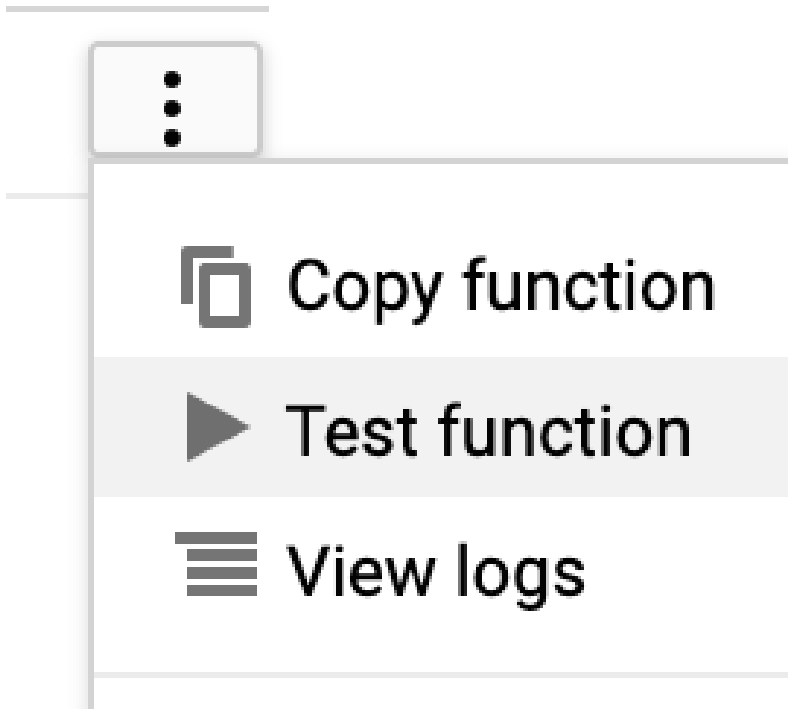
Runtime

Python 3.7

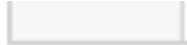
[main.py](#) requirements.txt

```
1 def hello_world(request):
2     """Responds to any HTTP request.
3     Args:
4         request (flask.Request): HTTP request object.
5     Returns:
6         The response text or any set of values that can be turned into a
7         Response object using
8         `make_response` <http://flask.pocoo.org/docs/1.0/api/#flask.make_response>
9     """
10    request_json = request.get_json()
11    if request.args and 'message' in request.args:
12        return request.args.get('message')
13    elif request_json and 'message' in request_json:
14        return request_json['message']
15    else:
16        return f'Hello World!'
17
```

Google Cloud Functions



Google Cloud Functions



Test the function

Output

```
Hello World!
```

Nv_query with gcloud

```
import requests
import lxml
from bs4 import BeautifulSoup
from time import ctime
import json

def get_nv_query(request):
    base_uri = "https://www.naver.com/"
    exec_time = ctime()
    response = requests.get(base_uri)
    html_text = response.text
    soup = BeautifulSoup(html_text, 'lxml')
    ul_tag = soup.find("ul", attrs={"class": "ah_l"})
    querys = []
    for li in ul_tag.find_all("li"):
        query = li.find("span", attrs={"class": "ah_k"}).text
        querys.append(query)
    result = {
        "time": exec_time,
        "items": querys,
    }
    print(result)
    return json.dumps(result, ensure_ascii=False)
```

Google Cloud Functions

[main.py](#) requirements.txt

```
1 import requests
2 import lxml
3 from bs4 import BeautifulSoup
4 from time import ctime
5 import json
6
7
8 def get_nv_query(request):
9     base_uri = "https://www.naver.com/"
10    exec_time = ctime()
11    response = requests.get(base_uri)
12    html_text = response.text
13    soup = BeautifulSoup(html_text, 'lxml')
14    ul_tag = soup.find("ul", attrs={"class": "ah_l"})
15    queries = []
16    for li in ul_tag.find_all("li"):
17        rank = li.find("span", attrs={"class": "ah_r"}).text
18        query = li.find("span", attrs={"class": "ah_k"}).text
19        queries.append({rank: query})
20
21    result = {
22        "time": exec_time,
23        "items": queries,
24    }
25    print(result)
26    return json.dumps(result, ensure_ascii=False)
```


Google Cloud Functions

main.py [requirements.txt](#)

```
1 # Function dependencies, for example:
2 # package>=version
3 requests==2.20.0
4 beautifulsoup4==4.6.3
5 lxml==4.4.1
```

Google Cloud Functions

```
us-central1-dev-sprint-4week.cloudfunctions.net/nv-queries
{
  time: "Tue Oct 22 06:47:37 2019",
  items: [
    - {
      1: "무신사 니트 핫세일"
    },
    - {
      2: "닥터포헤어 현빈"
    },
    - {
      3: "청년곡창 방탄크릴오일"
    },
    - {
      4: "닥터포헤어"
    },
    - {
      5: "신세경 치랭스 1&1"
    },
    - {
      6: "미쓰백"
    },
    - {
      7: "티몬 명륜진사갈비"
    },
    - {
      8: "닥터포헤어 폴리젠"
    },
  ],
}
```

Store data with MongoDB atlas

- <https://www.mongodb.com/cloud/atlas>

```

import requests
import lxml
from bs4 import BeautifulSoup
from time import ctime
#import json
from pymongo import MongoClient

def get_nv_query(event, context):
    # ...
    result = {
        "time": exec_time,
        "items": querys,
    }
    # ...
    # insert into mlab
    mongo_uri = "mongodb://betteradmin:1q2w3e4r@ds255403.mlab.com:55403/allquerys?retryWrites=false"
    try:
        client = MongoClient(mongo_uri)
        db = client.allquerys
        db_nvquerys = db.nvquerys
        db_nvquerys.insert_one(result)
    except:
        return "failed on {}".format(exec_time)
    return "success on {}".format(exec_time)

```

Google Cloud Functions

main.py

[requirements.txt](#)

```
1 # Function dependencies, for example:
2 # package>=version
3 requests==2.20.0
4 beautifulsoup4==4.6.3
5 lxml==4.4.1
6 pymongo==3.9.0
```



Turn into pubsub and schedule function

Google Cloud Functions

Memory allocated

256 MB

Trigger

HTTP

Cloud Pub/Sub

Google Cloud Functions

Trigger

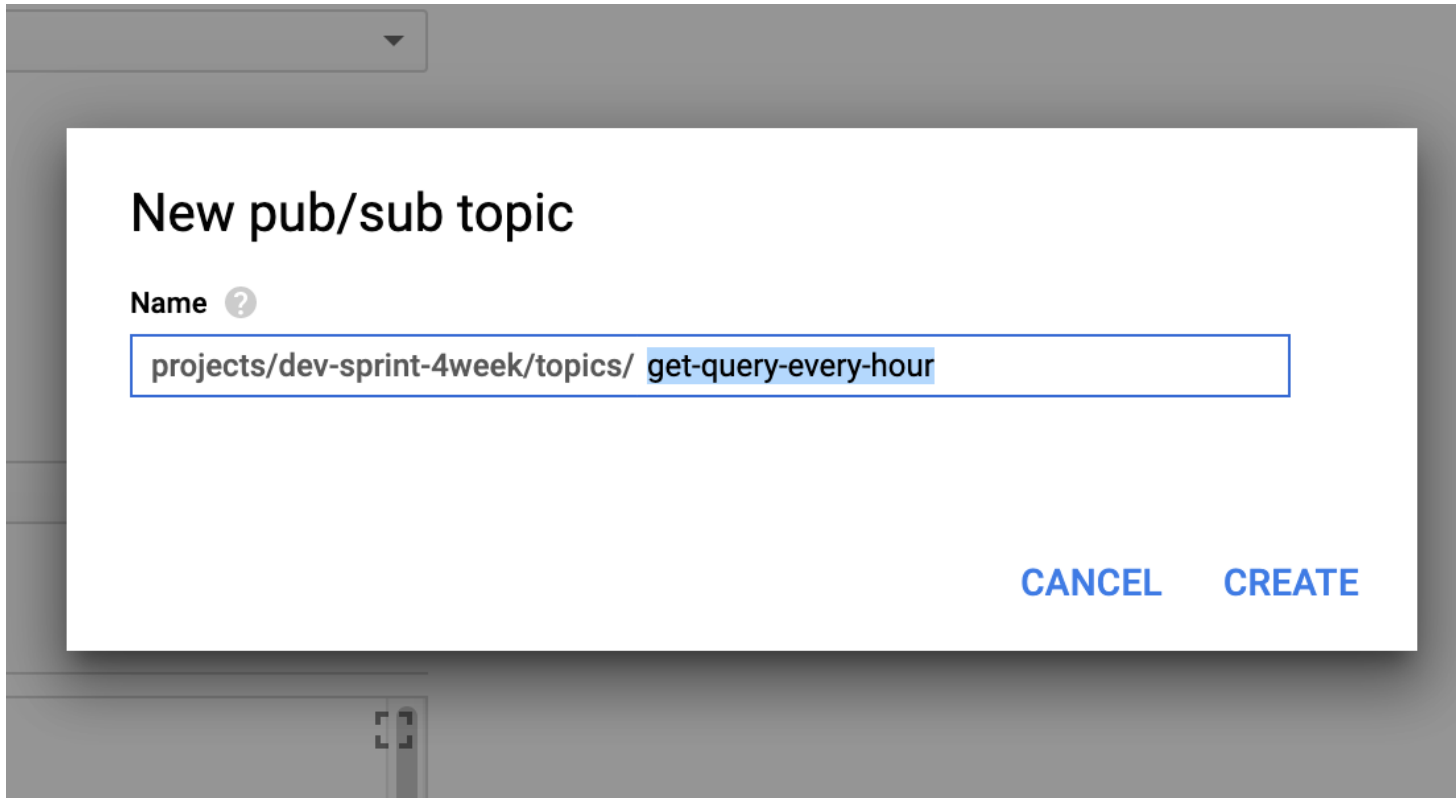
Cloud Pub/Sub

Topic

Create new topic...

Source code

Google Cloud Functions



A screenshot of the Google Cloud console interface showing a modal dialog for creating a new pub/sub topic. The dialog has a title 'New pub/sub topic' and a label 'Name' with a help icon. A text input field contains the path 'projects/dev-sprint-4week/topics/' followed by 'get-query-every-hour', which is highlighted in blue. At the bottom right of the dialog are two buttons: 'CANCEL' and 'CREATE'.

New pub/sub topic

Name ?

projects/dev-sprint-4week/topics/ get-query-every-hour

CANCEL CREATE

Google Cloud Functions


Google Cloud Scheduler

Google Cloud Scheduler is a fully managed cron job scheduling service. Use trigger jobs on App Engine, send Pub/Sub messages, or hit an arbitrary HTTP endpoint on a recurring schedule.

[CREATE JOB](#)

[BROWSE DOCS](#)

Google Cloud Functions

 Cloud Scheduler

[←](#) get-query-every-hour

Description

Frequency *

0 * * * *

Schedules are specified using unix-cron format. E.g. every minute: "* * * * *", every 3 hours: "0 */3 * * * *", every monday at 9:00: "0 9 * * 1". [Learn more](#)

Timezone *

Central Daylight Time (CDT) ▼

Target *

Pub/Sub ▼

Topic *

get-query-every-hour

Payload *

jobstart