데이터사이언스를위한 지식관리시스템

5주차: 실시간 데이터 수집하기 - 1 (RestAPI)
+ SSH Key 설정방법

SSH 로그인 방식 및 VSCode 설정

- 패스워드 로그인 방식: 우리가 지금까지 해왔던 방식 입니다

- 개인키 로그인 방식: VSCode(SSH Client) 에서는 개인키로 인증하고 Ubuntu(SSH Server) 에서는 공개키 목록으로부터 허용된 사용자인지 검증합니다
- 개인키, 공개키의 쌍을 만들고 개인키는 VSCode에 설정, 공개키는 Ubuntu 서버에 설정 합니다
- 이 과정을 진행하겠습니다

개인키, 공개키 쌍 만들기

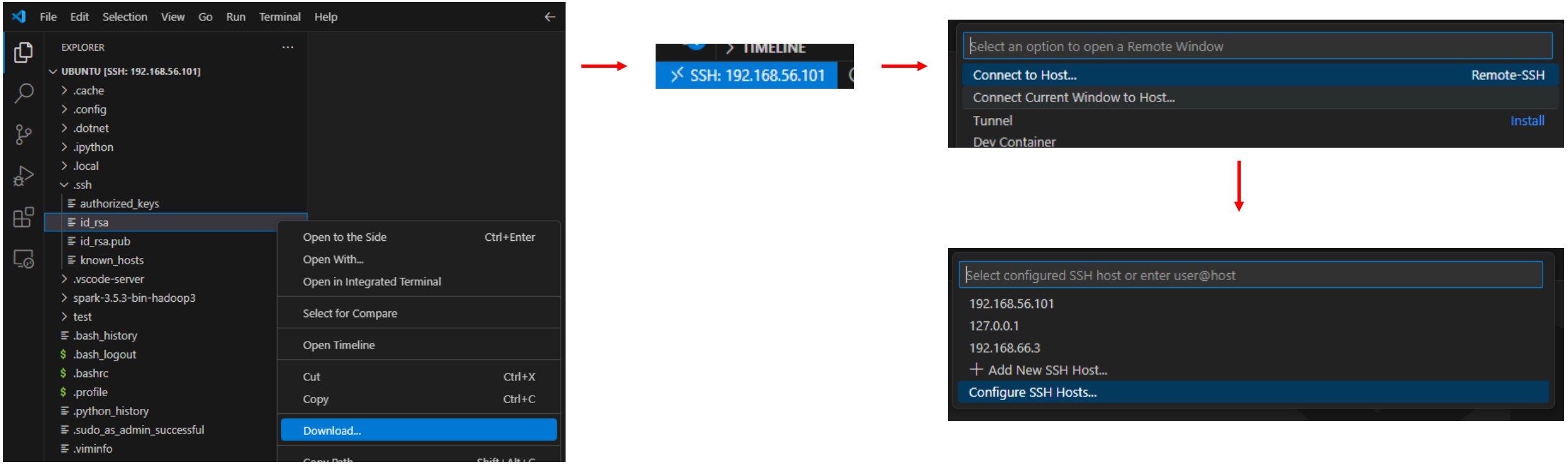
- ssh-keygen 이라는 커맨드를 입력합니다
- Host OS에서 만들어도 되고 Ubuntu 서버에서 만들어도 됩니다
- passphrase 는 개인키를 보호하기 위한 암호 입니다
- 이를 진행하고 나면 ~/.ssh/id_rsa(개인키) 와 ~/.ssh/id_rsa.pub 파일이 생깁니다
- 이 중에서 ~/.ssh/id_rsa.pub 는 ~/.ssh/authorized_keys에 복사하고 ~/.ssh/id_rsa는 VSCode로 가져올 것입니다

```
ubuntu@ubuntu-server:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:jCuMhyJw+oZEg5dZSUJxnjI8Z0Uz2lIl5WcFucnd3Ck ubuntu@ubuntu-server
The key's randomart image is:
+---[RSA 3072]----+
| .+oooBoo .o.    |
| . +o* =  ..     |
|. =+B . ..o+ o ..|
|.o+* . o o+ .Eo..|
|o.o   . S     .  |
|.+ +   .         |
|=.o + .          |
|oo.. .           |
| ..              |
+----[SHA256]-----+
ubuntu@ubuntu-server:~$
```

```
ubuntu@ubuntu-server:~$ ll .ssh/
total 20
drwx------  2 ubuntu ubuntu 4096 Oct 10 22:55 ./
drwxr-x--- 11 ubuntu ubuntu 4096 Oct 10 07:28 ../
-rw-------  1 ubuntu ubuntu    0 Sep 26 13:42 authorized_keys
-rw-------  1 ubuntu ubuntu 2610 Oct 10 22:55 id_rsa
-rw-r--r--  1 ubuntu ubuntu  574 Oct 10 22:55 id_rsa.pub
-rw-r--r--  1 ubuntu ubuntu  142 Oct  2 18:26 known_hosts
ubuntu@ubuntu-server:~$
```

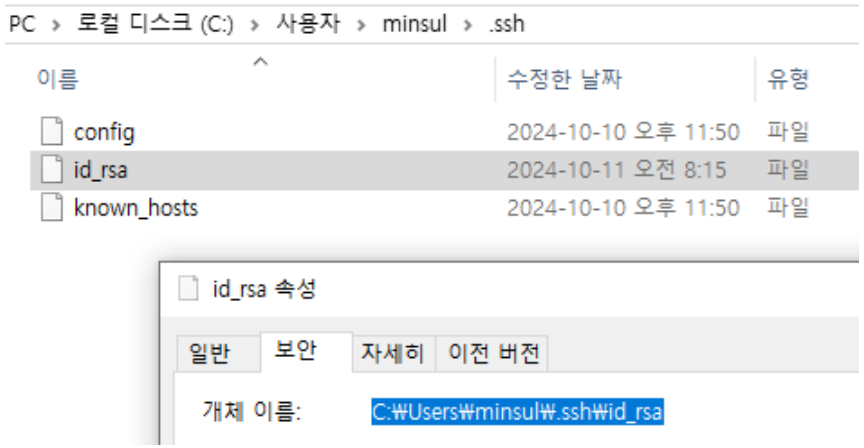
VSCode에 개인키 로그인 설정

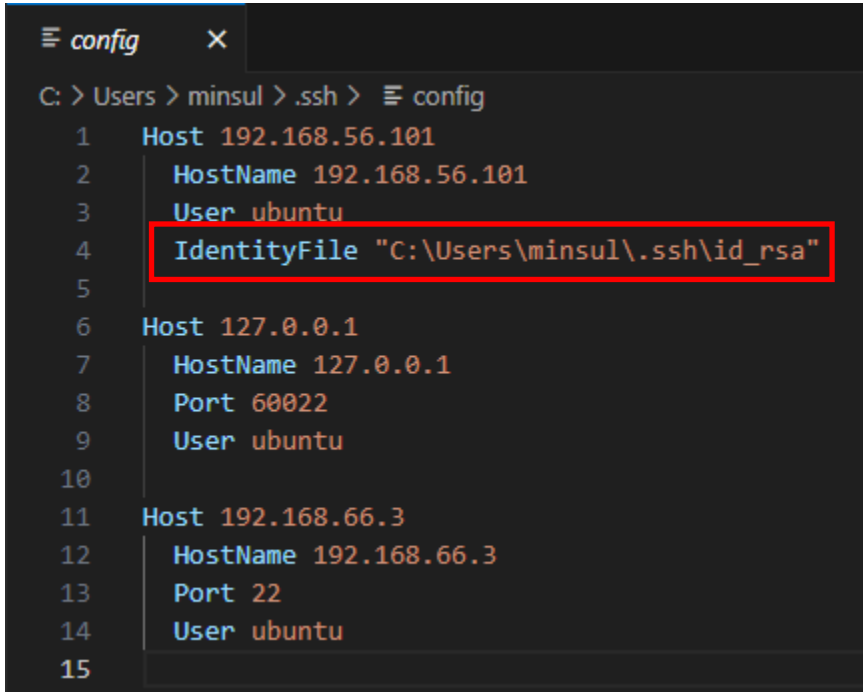
- ~/.ssh/id_rsa 파일을 다운로드 받습니다
- 다운로드된 위치를 잘 기억해둡니다
- 좌측 하단 파란색 버튼 → Connect to Host → Configure SSH Hosts 를 선택합니다



VSCode에 개인키 로그인 설정

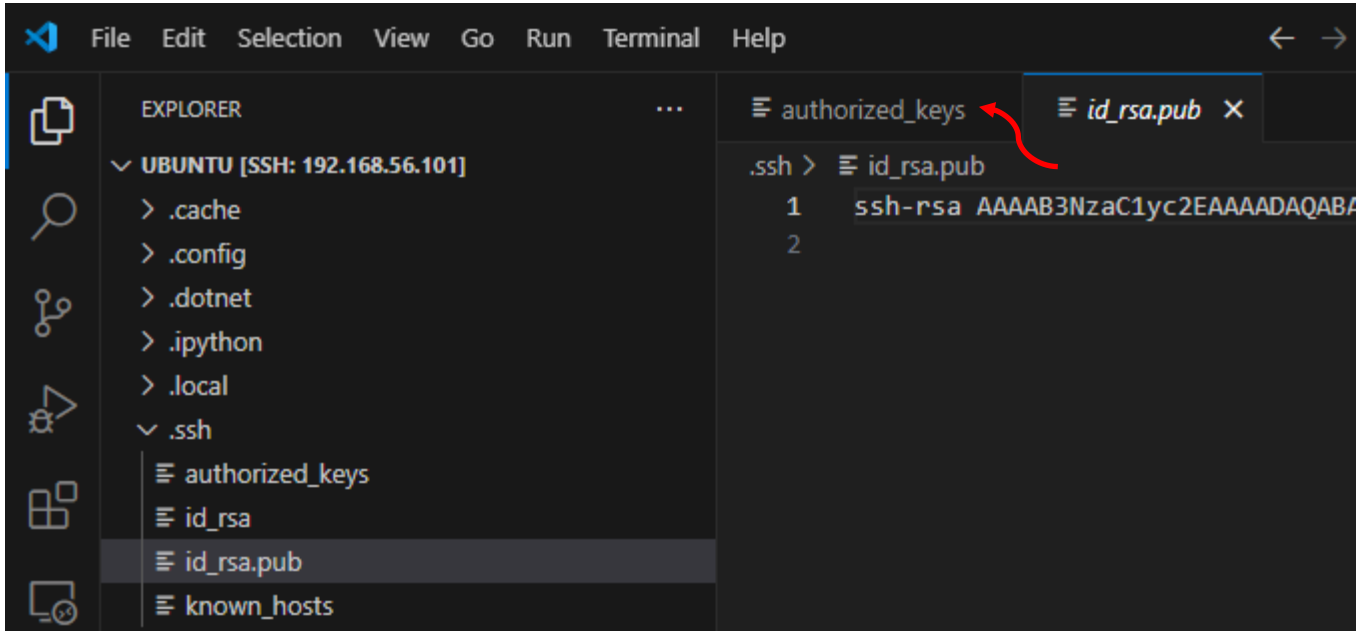
- 다운로드된 id_rsa 파일의 경로를 복사합니다
- ssh config 파일에 IdentityFile 설정을 합니다
- 이 때 디렉토리 구분자를 인식하지 못할 수 있으니 " " 기호로 감싸도록 합니다





Ubuntu 서버에 공개키 허용 목록 업데이트

- ~/.ssh/id_rsa.pub 파일을 VSCode에서 열고 이를 복사합니다
- ~/.ssh/authorized_keys 파일을 VSCode에서 열고 이를 붙여넣습니다
- 여기까지 완료 후 VSCode를 닫은 후 다시 열어서 SSH 연결이 자동으로 완료되는지 확인합니다



개인키 로그인 사용시 주의사항

- 개인키가 패스워드 대신 사용된다는 생각을 해보면 아래와 같은 주의사항이 필요합니다
- 공용으로 사용하는 PC 에서는 개인키 로그인 방식을 가급적 사용하지 않습니다
- 특별한 이유가 없다면 서버에는 개인키를 가급적 남기지 않습니다

5주차 목표

- 데이터 제공 방법으로 많이 사용되는 Rest API에 대해 알아봅니다
- 우리가 알고 있는 방법으로(커맨드라인) 테스트용 Rest API 의 CRUD를 진행해봅니다
- 실제 제공되는 공공데이터를 하나 지정해서 주기적인 수집을 진행해봅니다
- cron 이라고 하는 스케줄러를 사용하여 주기적인 Rest API 호출을 진행합니다
- 작업 스케줄을 정의할 때 고려해야 할 사항에 대해 알아봅니다

공공데이터 포털 살펴보기

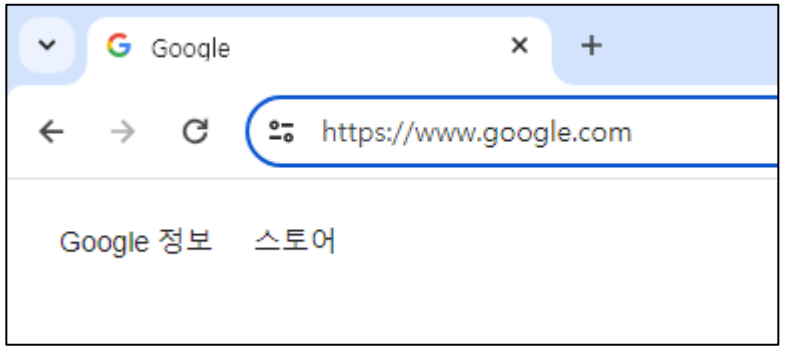
- https://www.data.go.kr
- https://www.data.go.kr > 데이터찾기 > 이슈 및 추천데이터 에서 임의의 카테고리를 선택합니다
- 데이터는 크게 보면 아래와 같이 2가지로 제공되는 것을 확인할 수 있습니다
- 파일형태로 제공, Open API 형태로 제공
- 이 중에서 Open API는 실시간 데이터에 많이 사용되는 방식입니다
- 따라서 이를 수집하는 클라이언트에서는 주기적으로 Open API를 호출하여 실시간 데이터를 받아옵니다

- Open API: 말 그대로 공개된 API(공공데이터 포털에서는 HTTP API를 주로 사용)
- Rest API: Restful 이라는 규격을 준수하여 만들어진 HTTP API

웹브라우저 열어보기

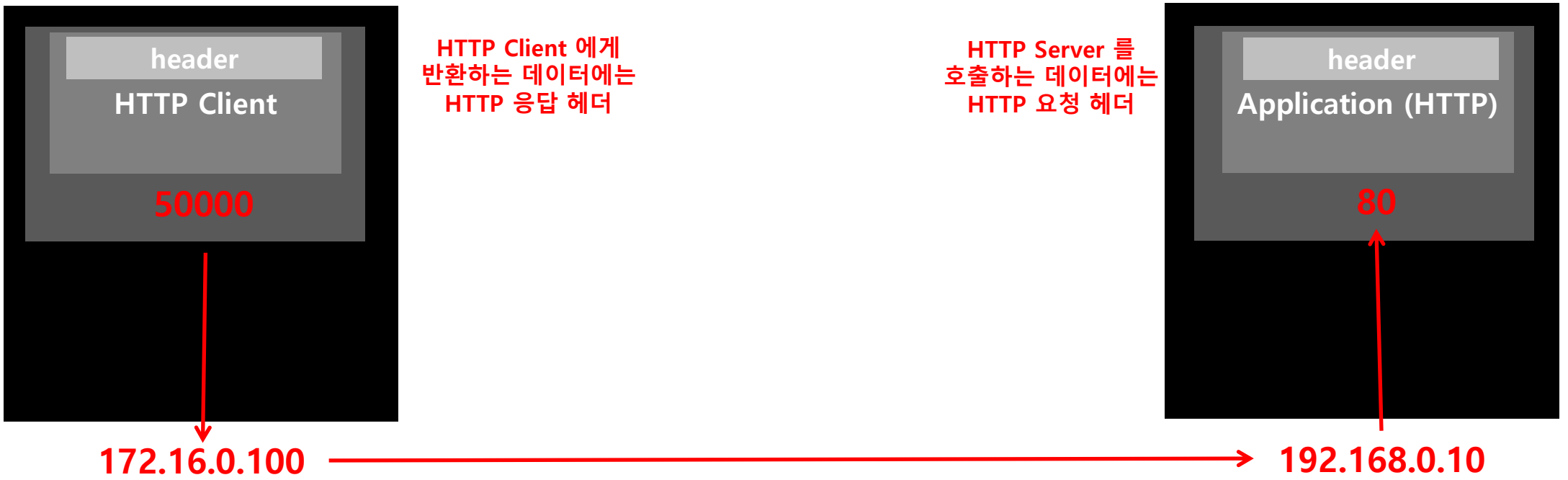
- 웹브라우저란 URL을 호출하고 HTML을 랜더링 하는 도구인 것입니다
- 아래에서 https://www.google.com 으로 접속을 했습니다
- www.google.com 이라고 하는 도메인에 매핑된 IP가 있습니다
- https로 프로토콜을 지정하면 포트를 별도 명시하지 않을 경우 기본 443 으로 접속을 시도합니다
- 즉 수신지 IP를 www.google.com 의 IP로 하고 수신지 포트를 443 으로 하는 연결을 하는 것입니다
- 여기까지는 IP:PORT의 전송레벨 프로토콜 입니다

- 웹브라우저는 위의 경로로 요청을 보내고 응답으로 리턴된 HTML을 랜더링합니다
- 이제 전송완료 후 애플리케이션 레벨의 통신규격에 대해 알아보겠습니다



HTTP와 헤더규격

- 우리는 컴퓨터간 통신을 위해 수신지와 발신지의 포트와 IP 정보가 필요함을 확인했습니다
- 이를 기반으로 전달되는 메시지의 포맷은 애플리케이션에 따라 달라집니다
- 이것은 마치 모든 택배 겉면에는 수신지, 발신지 정보가 기록되고 내부 포장 규격은 제품마다 다른것과 같습니다
- HTTP 는 내부 포장 규격중 하나 입니다
- 이 때 포장 규격을 헤더라고 합니다



HTTP의 CRUD

- Create, Retrive, Update, Delete 동작에 따라 HTTP의 헤더값(method) 을 달리 합니다
- 각 기능별 method는 일반적으로 아래와 같습니다

- Create: POST (데이터를 생성하는 요청)
- Retrieve: GET (데이터를 조회하는 요청)
- Update: PUT (기존 데이터를 업데이트하는 요청)
- Delete: DELETE (기존 데이터를 삭제하는 요청)

- 이 중에서 웹브라우저가 주로 사용하는 method는 GET 입니다
- 웹 페이지 내용을 요청해서 받아오는 것이기 때문에 조회의 성격에 가깝습니다
- 따라서 웹브라우저에서 URL 접속 후 페이지를 받아오는 동작은 대개 GET method로 요청됩니다

웹브라우저를 열고 구글에 접속했을때의 내부 동작

- 개발자도구를 열고 Network 탭을 클릭합니다
- www.google.com 에 접속한 상태에서 새로고침을 합니다
- 최초 요청된 경로에 대해 Headers 를 확인합니다

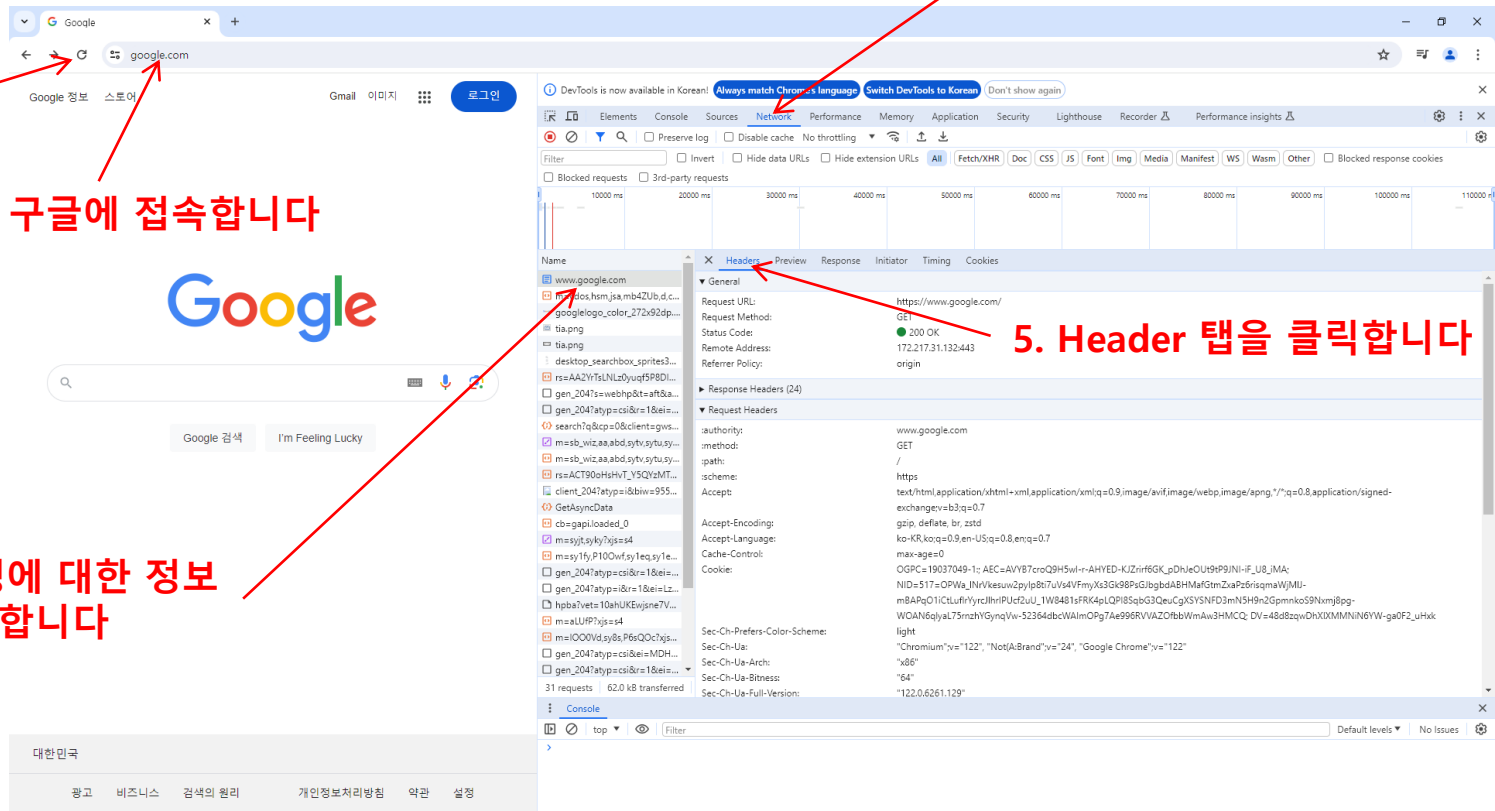
2. 개발자 도구를 열고
Network 탭을 클릭합니다
(MS Windows의 경우 F12)

3. 새로고침을 합니다
그러면 www.google.com
페이지를 요청하게 되겠습니다

1. 먼저 구글에 접속합니다

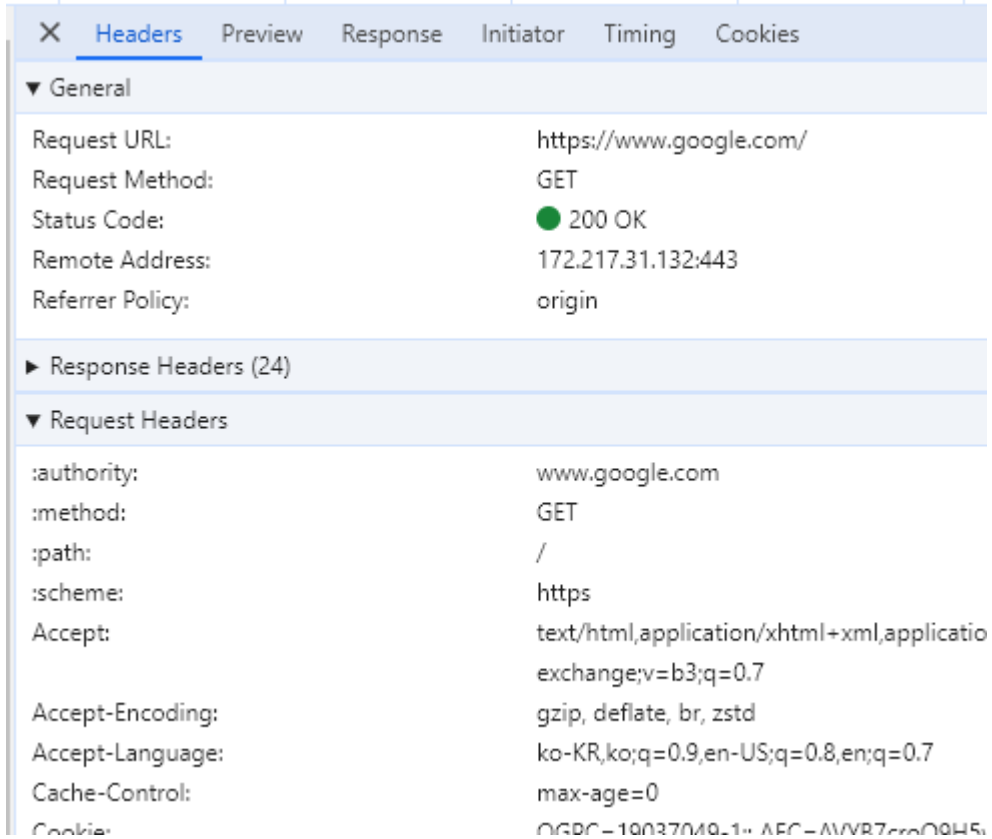
5. Header 탭을 클릭합니다

4. 페이지 요청에 대한 정보
를 확인합니다



HTTP GET method 헤더

- Headers 탭 확인시 Request Method에 GET이 확인됩니다
- 이 부분이 CRUD 기능에 따라 달라집니다
- 일반적으로 웹사이트의 페이지를 열때에는 GET 을 사용합니다



curl 커맨드를 이용해 HTTP GET 요청

- 3주차에 데이터수집에 사용한 curl 커맨드를 다시 생각해보겠습니다
- 해당 커맨드 사용시 별도로 GET, POST, PUT, DELETE에 대한 헤더 설정은 하지 않았습니다
- 아래 커맨드로 curl 에서 HTTP method를 별도 설정하지 않으면 GET 으로 동작함을 확인합니다

```
man curl | grep “which defaults to GET”
```

테스트용 Rest API 체험해보기

- 지금까지 알아본 HTTP 방식의 API를 Rest API라고 합니다
- Rest API 테스트 사이트가 몇개 있는데 그 중에서 우리는 아래의 경로를 사용해봅니다
- Rest API test site 라는 내용으로 검색을 해보면 아래의 사이트가 보일 것입니다
- https://jsonplaceholder.typicode.com (JSONPlaceholder - Free Fake REST API)

GET

- https://jsonplaceholder.typicode.com (JSONPlaceholder - Free Fake REST API) 로부터 GET method를 테스트 합니다
- 전체 포스팅 내용을 조회합니다
- 총 100개의 포스팅이 있습니다

```
curl -X GET https://jsonplaceholder.typicode.com/posts
```

- 포스팅 ID 1에 대한 내용을 조회합니다

```
curl -X GET https://jsonplaceholder.typicode.com/posts/1
```

- 포스팅 ID 2에 대한 내용을 조회합니다

```
curl -X GET https://jsonplaceholder.typicode.com/posts/2
```

POST

- https://jsonplaceholder.typicode.com (JSONPlaceholder - Free Fake REST API) 로부터 POST method를 테스트 합니다
- posts 레코드를 추가하는 요청 입니다

```
curl -X POST https://jsonplaceholder.typicode.com/posts -H 'Content-Type:application/json' --data  '{"title" : "hello", "body":"hello world", "userId": 111 }'
```

- 기존에 100개가 있었으므로 추가되는 레코드는 101번째가 됩니다
- id는 서버에서 발행되어 응답되는 구조 입니다

```
ubuntu@ubuntu-server:~$ curl -X POST https://jsonplaceholder.typicode.com/posts -H 'Content-Type:application/json' --data  '{"title" : "hello"
, "body":"hello world", "userId": 111 }'
{
  "title": "hello",
  "body": "hello world",
  "userId": 111,
  "id": 101
}ubuntu@ubuntu-server:~$
```

PUT

- https://jsonplaceholder.typicode.com (JSONPlaceholder - Free Fake REST API) 로부터 POST method를 테스트 합니다
- 기존에 존재하는 posts 레코드를 수정하는 요청 입니다

```
curl -X PUT https://jsonplaceholder.typicode.com/posts/200 -H 'Content-Type:application/json' --data  '{"title" : "hello", "body":"hello world", "userId": 111 }'
```

- 레코드 ID 200은 존재하지 않기 때문에 아래와 같이 에러가 발생했습니다
- 레코드 ID를 100 이하로 변경하여 다시 테스트 해봅니다

```
ubuntu@ubuntu-server:~$ curl -X PUT https://jsonplaceholder.typicode.com/posts/200 -H 'Content-Type:application/json' --data  '{"title" : "hel
lo", "body":"hello world", "userId": 111 }'
TypeError: Cannot read properties of undefined (reading 'id')
    at update (/app/node_modules/json-server/lib/server/router/plural.js:262:24)
    at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
    at next (/app/node_modules/express/lib/router/route.js:137:13)
    at next (/app/node_modules/express/lib/router/route.js:131:14)
    at Route.dispatch (/app/node_modules/express/lib/router/route.js:112:3)
    at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
    at /app/node_modules/express/lib/router/index.js:281:22
    at param (/app/node_modules/express/lib/router/index.js:354:14)
    at param (/app/node_modules/express/lib/router/index.js:365:14)
    at Function.process_params (/app/node_modules/express/lib/router/index.js:410:3)ubuntu@ubuntu-server:~$
```

DELETE

- https://jsonplaceholder.typicode.com (JSONPlaceholder - Free Fake REST API) 로부터 GET method를 테스트 합니다
- 기존에 존재하는 레코드를 삭제합니다
- 레코드 ID 20 데이터를 삭제하는 요청입니다

```
curl -X  DELETE https://jsonplaceholder.typicode.com/posts/20
```

HTTP의 상태코드

- 정상: 200
- 리다이렉트: 300
- 요청측 에러: 400
- 서버측 에러: 500

- 지금까지 테스트했던 모든 요청들에 대해 -i 옵션을 추가하여 다시 진행해봅니다
- 존재하지 않는 ID에 대해 PUT(update) 요청했을 때 상태코드는 500이 확인됩니다
- 그 외에 나머지는 200이 확인 됩니다

```
ubuntu@ubuntu-server:~$ curl -i -X PUT https://jsonplaceholder.typicode.com/posts/200 -H 'Content-Type:application/json' --data  '{"title" : "
hello", "body":"hello world", "userId": 111 }'
HTTP/2 500
```

Path parameter, Query parameter, Request body

- 프로그램의 매개변수처럼 Rest API에 변수를 전달하는 방법 입니다

- Path parameter: URL 경로에 변수를 담아 전달
- Query parameter: URL 경로 뒤에 변수를 담아 전달
- Request body: URL과는 별도로 떨어진 필드(body)에 변수를 담아 전달

- 이전 슬라이드에서 테스트했던 GET, POST, PUT, DELETE에서는 어떠한 방법을 사용했는지 확인해봅시다
- GET: Path parameter
- POST: Request body
- PUT: Path parameter, Request body
- DELETE: Path parameter

- Query parameter는 날씨 데이터 수집할 때 사용했었습니다

Rest API 문서 읽기

- 다시 공공데이터 포털로 이동합니다
- https://www.data.go.kr > 데이터찾기 > 이슈 및 추천데이터에서 임의의 카테고리를 선택합니다
- 이후 OpenAPI 탭을 선택하고 실시간 제공 데이터를 선택합니다
- https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15095061 를 예시로 선택하겠습니다
- 인천국제공항공사 입국장현황 정보 서비스로 들어가서 "참고 문서" 를 다운로드 받아 이를 확인하겠습니다

API 규격문서에서 확인할 주요 사항

- 서비스 URL(endpoint) 는 어디인지
- Rest API 라면 GET, POST, PUT, DELETE등 메소드 중에서 어떤방식을 사용하는지
- 요청 변수는 무엇인고 자료형은 어떻게 되는지
- 필수 요청 변수는 무엇인지
- 선택 요청 변수의 경우 설정하지 않으면 어떻게 동작하는지
- 응답의 포맷은 어떻게 되는지
- 데이터 갱신주기는 어떻게 되는지 ->수집 주기와 밀접한 연관

- 이 과정에서 serviceKey 라는 것이 필요함을 확인하게 됩니다
- 데이터 활용신청시 개별 계정에 할당되는 키 입니다

인천국제공항공사 입국장현황 데이터 수집

- 우선 데이터 활용신청이 필요합니다
- 이후 획득된 serviceKey를 아래 serviceKey= 오른쪽에 붙여넣습니다
- numOfRows: 응답에 몇건의 데이터를 출력할지
- pageNo: 전체 데이터 건수 중에서 numOfRows를 기준으로 몇번째 페이지를 응답할지

```
curl -X GET "http://apis.data.go.kr/B551177/StatusOfArrivals/getArrivalsCongestion?type=json&serviceKey=&numOfRows=&pageNo="
```

- 아래와 같은 응답을 확인할 수 있습니다


```
{"response":{"header":{"resultCode":"00","resultMsg":"NORMAL SERVICE."},"body":{"numOfRows":10,"pageNo":1,"totalCount":41,"items":[{"terno":"T
1","entrygate":"D","korean":"0.0","foreigner":"0.0","scheduletime":"202409171850","estimatedtime":"202409171922","airport":"GUM","gatenumber":
"119","flightid":"TW304"},{"terno":"T1","entrygate":"B","korean":"0.0","foreigner":"0.0","scheduletime":"202409171905","estimatedtime":"202409
171946","airport":"SYD","gatenumber":"12","flightid":"OZ602"},{"terno":"T1","entrygate":"E","korean":"0.0","foreigner":"0.0","scheduletime":"2
02409171915","estimatedtime":"202409171911","airport":"HKG","gatenumber":"50","flightid":"CX418"},{"terno":"T1","entrygate":"B","korean":"0.0"
,"foreigner":"1.0","scheduletime":"202409171920","estimatedtime":"202409172001","airport":"PVG","gatenumber":"7","flightid":"OZ366"},{"terno":
"T1","entrygate":"B","korean":"0.0","foreigner":"0.0","scheduletime":"202409171925","estimatedtime":"202409171926","airport":"SGN","gatenumber
":"15","flightid":"OZ732"},{"terno":"T1","entrygate":"C","korean":"1.0","foreigner":"0.0","scheduletime":"202409171925","estimatedtime":"20240
9171947","airport":"SGN","gatenumber":"113","flightid":"VJ860"},{"terno":"T1","entrygate":"C","korean":"0.0","foreigner":"0.0","scheduletime":
"202409171935","estimatedtime":"202409171936","airport":"MFM","gatenumber":"114","flightid":"7C2002"},{"terno":"T1","entrygate":"C","korean":"
0.0","foreigner":"0.0","scheduletime":"202409171945","estimatedtime":"202409172009","airport":"SPN","gatenumber":"115","flightid":"7C3475"},{"
terno":"T1","entrygate":"E","korean":"0.0","foreigner":"1.0","scheduletime":"202409171945","estimatedtime":"202409171941","airport":"KHH","gat
enumber":"31","flightid":"BR172"},{"terno":"T1","entrygate":"B","korean":"0.0","foreigner":"0.0","scheduletime":"202409171950","estimatedtime"
:"202409171949","airport":"KIX","gatenumber":"17","flightid":"OZ113"}]}}}ubuntu@ubuntu-server:~$
```

주기적 수집 적용

- 데이터 갱신주기인 1분에 맞추어서 1분 주기 수집을 적용해봅니다
- 2주차에 확인했던 반복 수집(while true; do ~ & sleep 60; done) 을 적용해봅니다
- 마찬가지로 파일명에는 수집시각을 기록합니다
- sleep 60 으로 1분 주기 수집합니다
- 수집에 소요되는 시간이 sleep을 blocking 하지 않도록 수집 커맨드와 sleep 사이에 & 처리하여 non-blocking 수집 합니다

```
while true; do curl -o $(TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_ICN.json') -X GET "http://apis.data.go.kr/B551177/StatusOfArrivals/getArrivalsCongestion?type=json&serviceKey=&numOfRows=&pageNo=" & sleep 60; done
```

HTTP GET의 bash → python 번역

- 지금부터 3주차에 사용했던 curl을 이용한 CRUD를 python으로 각각 번역해봅니다
- 아래와 같은 날씨 API를 사용했었습니다

```
curl -X GET "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m";
```

같은 호출을 다르게 표현한 것입니다

```python
# GET
import requests

url = "https://api.open-meteo.com/v1/forecast"
params = {
    "latitude": 52.52,
    "longitude": 13.41,
    "current": "temperature_2m,wind_speed_10m",
    "hourly": "temperature_2m,relative_humidity_2m,wind_speed_10m"
}

response = requests.get(url, params=params)

print(response.text)
```

```python
import requests

url = "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"

response = requests.get(url)

print(response.text)
```

HTTP POST의 bash → python 번역

- requests.post 함수를 사용합니다
- request body를 보내는 방법으로 data와 json 이라는 매개변수 중의 하나를 사용할 수 있습니다
- 코드는 온라인강의실에 첨부된 실습파일을 사용합니다

```
curl -X POST https://jsonplaceholder.typicode.com/posts -H 'Content-Type:application/json' --data  '{"title" : "hello", "body":"hello world", "userId": 111 }'
```



```python
# POST
import requests
# import json

url = "https://jsonplaceholder.typicode.com/posts"
headers = {
    "Content-Type": "application/json"
}
body = {
    "title": "hello",
    "body": "hello world",
    "userId": 111
}

response = requests.post(url, headers=headers, json=body)
# response = requests.post(url, headers=headers, data=json.dumps(body))

print(response.text)
```

body를 json 매개변수로 넘기려면 dict 자료형을 그대로 사용합니다
body를 data 매개변수로 넘기려면 dict 자료형을 문자열로 바꾸어 사용합니다

HTTP PUT의 bash → python 번역

- POST와 다르게 requests.put 함수를 사용합니다
- POST와 동일하게 request body를 보내는 방법으로 data와 json 이라는 매개변수 중의 하나를 사용할 수 있습니다
- 코드는 온라인강의실에 첨부된 실습파일을 사용합니다

```
curl -X PUT https://jsonplaceholder.typicode.com/posts/100 -H 'Content-Type:application/json' --data  '{"title" : "hello", "body":"hello world", "userId": 111 }'
```



```python
# PUT
import requests
# import json

url = "https://jsonplaceholder.typicode.com/posts/100"
headers = {
    "Content-Type": "application/json"
}
body = {
    "title": "hello",
    "body": "hello world",
    "userId": 111
}

response = requests.put(url, headers=headers, json=body)
# response = requests.post(url, headers=headers, data=json.dumps(body))


print(response.text)
```

body를 json 매개변수로 넘기려면 dict 자료형을 그대로 사용합니다
body를 data 매개변수로 넘기려면 dict 자료형을 문자열로 바꾸어 사용합니다

HTTP DELETE의 bash → python 번역

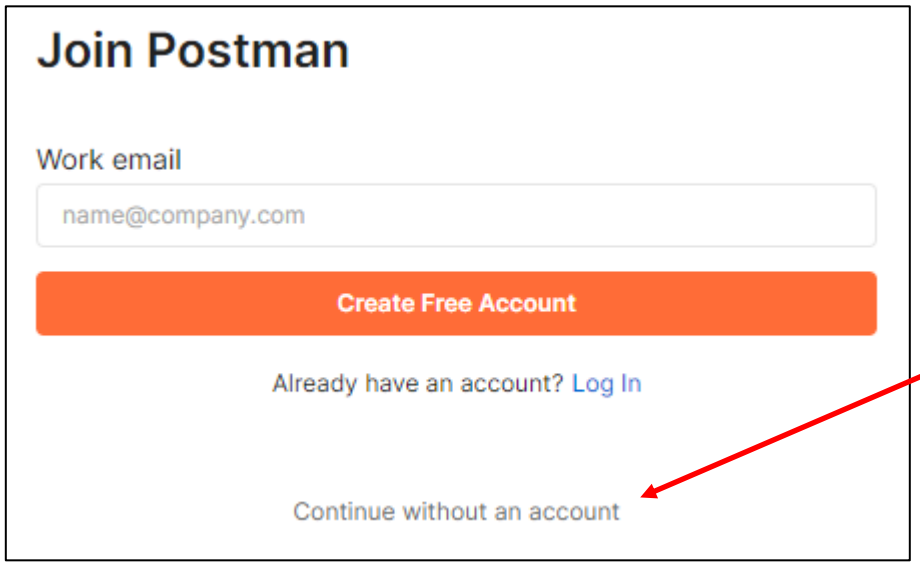
- 아래 테스트 API의 경우 별도의 query string을 사용하지 않습니다
- 그러나 DELETE method 역시 GET method와 마찬가지로 query string을 매개변수로 사용함을 인지하기 위해 params를 명시적으로 사용해보았습니다

```
curl -X  DELETE https://jsonplaceholder.typicode.com/posts/20
```

```python
import requests

url = "https://jsonplaceholder.typicode.com/posts/20"

params = {}

response = requests.delete(url, params=params)

print(response.text)
```

postman 도구

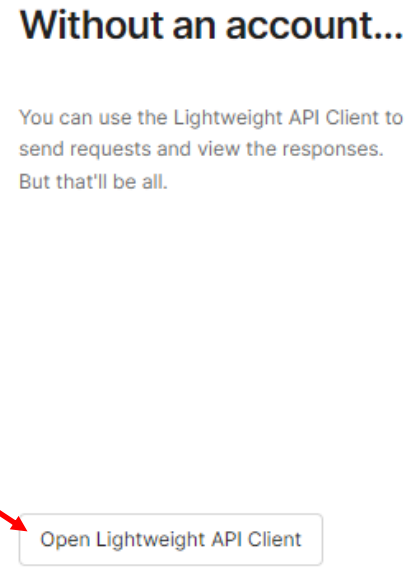
- 이상의 번역 작업을 도와줄 수 있는 postman 이라는 도구를 사용해봅니다
- https://www.postman.com/downloads/ 에서 다운로드 받습니다
- 무료버전의 라이선스에 대해서는 https://community.postman.com/t/can-i-use-the-free-version-for-business-purposes-company-larger-than-3-persons/39527 를 참고합니다



계정 없이 실행 가능합니다

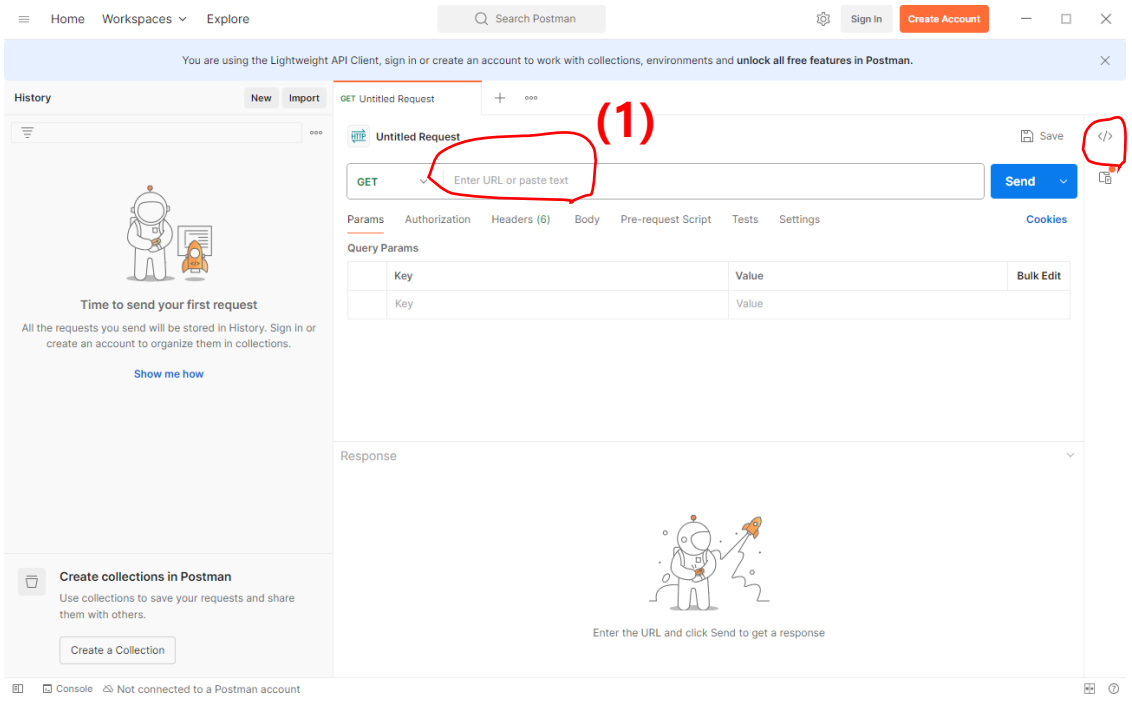
(1)

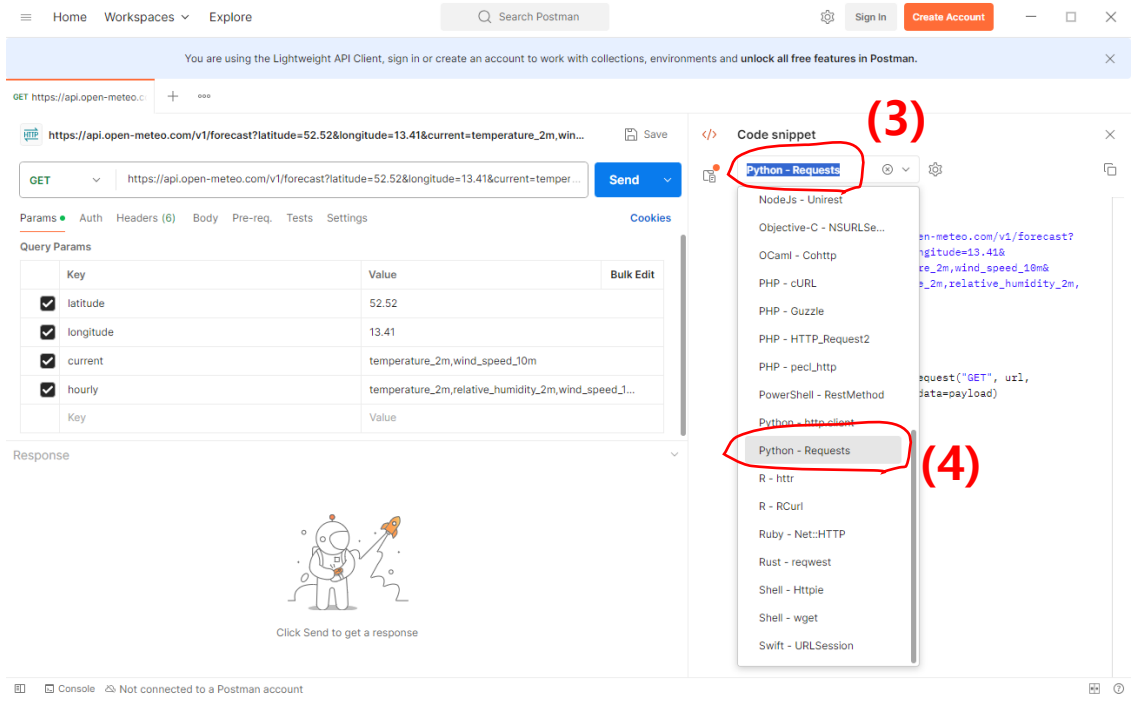
(2)



postman 도구

- 이후 아래 화면에 진입하게 되는데 "Enter URL or paste text" 에 아래 URL을 그대로 붙여넣어 봅니다
- https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m
- Query Params에 파라미터들이 들어가는 것을 확인할 수 있을 것입니다
- 이어서 우측의 </> 버튼을 눌러서 나오는 Code snippet 에서 Python - requests 를 선택해봅니다

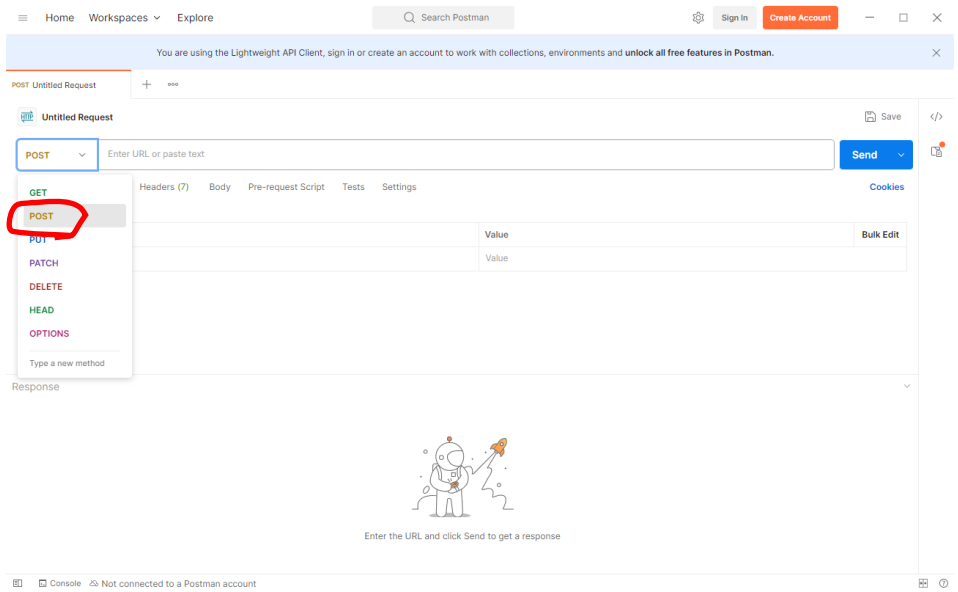


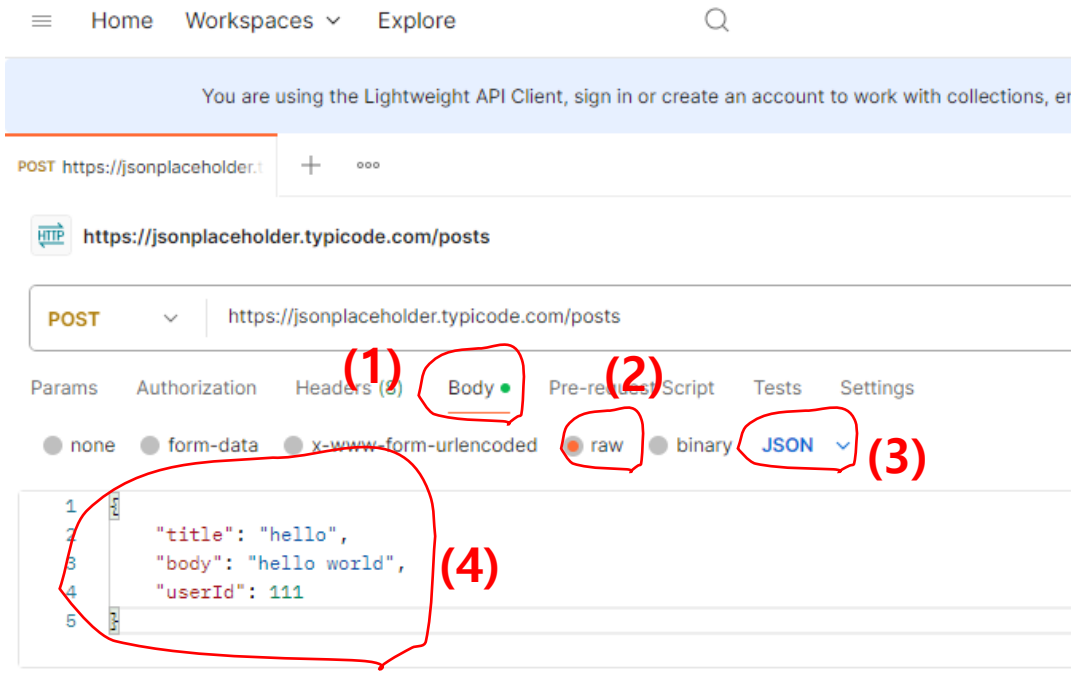


postman 도구

- POST method도 동일한 방법으로 진행해봅니다
- GET method 탭을 눌러서 POST로 변경하고 URL을 https://jsonplaceholder.typicode.com/posts 으로 붙여넣습니다
- Params 대신 Body → raw → JSON 을 선택합니다
- Body 편집창에 json body를 붙여넣습니다
- 이이서 code snippet 을 확인해봅니다

```
{
  "title": "hello",
  "body": "hello world",
  "userId": 111
}
```





스케줄러와 무한루프 방식의 차이

- 이제 우리는 HTTP CRUD를 규격에 맞게 호출해보고 이를 python코드로 변환할 수 있습니다
- 지금부터는 스케줄러에 의한 수집 트리거를 적용할 것입니다
- 그 전에 기존의 무한루프+sleep에 의한 주기적 수집 방식과 스케줄러에 의한 주기적 수집 방식의 차이에 대해 알아봅니다
- 아래와 같은 이유에 의해 스케줄러를 사용합니다

	무한루프+sleep	스케줄러
프로세스의 수명	지속됨	생성, 소멸을 반복
스케줄링 원리	sleep에 의한 스케줄링	스케줄링 서비스에 의한 스케줄링
프로세스 비정상 종료시 영향도	재실행 시켜주지 않으면 수집되지 않음	스케줄링 서비스에 의해 다음 스케줄을 받아서 정상 실행 가능

cron 스케줄러

- 가장 간단하게 사용해볼 수 있는 스케줄러 입니다
- Ubuntu 서버 에서는 이미 cron 스케줄러가 동작하고 있습니다
- 다만 정의된 스케줄이 없습니다

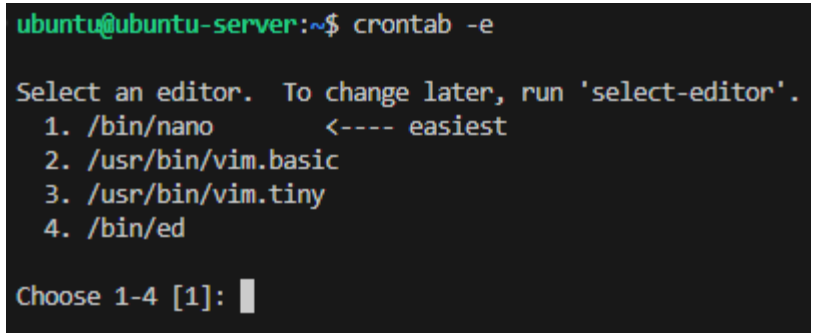
```
ubuntu@ubuntu-server:~$ crontab -l
ubuntu@ubuntu-server:~$
```

- 아래와 같이 나오면 정의된 스케줄이 있는 것입니다
- 매 1분마다 “Scheduled at YYYYMMdd-HHMMSS” 문자열을 /home/ubuntu/schedule.txt 에 누적하는 스케줄 입니다
- 이를 어떻게 해석하고 설정하는지 알아보겠습니다

```
ubuntu@ubuntu-server:~$ crontab -l
*/1 * * * *  date +'Scheduled at %Y%m%d-%H%M%S' >> /home/ubuntu/schedule.txt
ubuntu@ubuntu-server:~$
```

cron 에디터

- crontab -e 를 입력하면 스케줄러의 편집 모드로 들어갑니다
- 이 때 TUI 기반의 에디터
- 우리는 이미 GUI 기반의 에디터를 이용해 원격 접속을 했습니다
- 따라서 TUI 기반의 에디터 보다는 GUI 기반의 에디터를 사용해봅니다
- 아래 화면에서 ctrl + z 를 눌러서 편집을 취소합니다



VSCode로 cron 스케줄러 편집

- 파일 브라우저를 통해 crontab 이라는 파일을 만듭니다
- 파일명은 자유롭게 지정하셔도 됩니다
- <minute> <hour> <day-of-month> <month> <day-of-week> <command> 의 포맷으로 하나의 스케줄이 정의 됩니다
- day-of-week 에는 0~6의 숫자 또는 3-letter 약자가 허용됩니다(0: SUN, 1: MON, 2: TUE, 3: WED...)
- * 으로 기록된 것은 해당 필드의 모든 숫자를 의미합니다
- 예를 들어 minute 필드가 * 이면 1분부터 59분까지 매 분마다의 스케줄을 의미합니다

- 아래와 같이 crontab 파일을 편집하고 저장해 봅니다

```
*/1 * * * *  "date +'Scheduled at %Y%m%d-%H%M%S' >> /home/ubuntu/schedule.txt"
```

- 아래 명령을 입력하여 스케줄을 적용하고 다시 현재 스케줄링 정책을 표시해봅니다

```
ubuntu@ubuntu-server:~$ crontab -u ubuntu crontab
ubuntu@ubuntu-server:~$ crontab -l
*/1 * * * *  "date +'Scheduled at %Y%m%d-%H%M%S' >> /home/ubuntu/schedule.txt"
ubuntu@ubuntu-server:~$
```

- 이제 1분마다 schedule.txt 파일이 어떻게 업데이트 되는지 확인해봅니다

crontab 스케줄러의 로깅

- 아래와 같은 명령으로 스케줄링 히스토리를 확인할 수 있습니다
- /var/log/syslog 에는 시스템 로그가 적재 됩니다
- tail 커맨드에 f 옵션을 사용하면 follow 형태로 파일을 출력합니다
- 이를 확인해보면 date 커맨드가 제대로 들어가지 않았습니다

```
ubuntu@ubuntu-server:~$ tail -f /var/log/syslog | grep CRON
Oct 11 02:40:01 ubuntu-server CRON[5565]: (ubuntu) CMD ("date +'Scheduled at )
Oct 11 02:40:01 ubuntu-server CRON[5564]: (CRON) info (No MTA installed, discarding output)
Oct 11 02:41:02 ubuntu-server CRON[6115]: (ubuntu) CMD ("date +'Scheduled at )
Oct 11 02:41:02 ubuntu-server CRON[6114]: (CRON) info (No MTA installed, discarding output)
Oct 11 02:42:01 ubuntu-server CRON[6263]: (ubuntu) CMD (date +"Scheduled at )
```

- % 기호가 crontab 에서는 특수문자로 사용 됩니다

```
ubuntu@ubuntu-server:~$ man 5 crontab | grep "%"
       The ``sixth'' field (the rest of the line) specifies the command to be run.  The entire command portion of the line, up to a newline or
       the SHELL variable of the crontab file.  Percent-signs (%) in the command, unless escaped with backslash (\), will be changed into newli
       0 22 * * 1-5    mail -s "It's 10pm" joe%Joe,%%Where are your kids?%
       0 4 8-14 * *    test $(date +\%u) -eq 6 && echo "2nd Saturday"
       0 4 * * Sat   [ "$(date +\%e)" = "$(LANG=C ncal | sed -n 's/^Sa .* \([0-9]\+\) *$/\1/p')" ] && echo "Last Saturday" && program_to_run
ubuntu@ubuntu-server:~$
```

- 가이드를 따라 아래와 같이 ₩ 기호를 사용하여 다시 crontab을 적용하면 정상 동작합니다
- crontab 파일을 아래와 같이 수정하고 crontab -u ubuntu crontab 명령으로 적용합니다

```
*/1 * * * *  date +"Scheduled at ₩%Y₩%m₩%d-₩%H₩%M₩%S" >> /home/ubuntu/schedule.txt
```

스케줄 반복 주기내에 작업을 끝내지 못했을 때에 발생하는 일

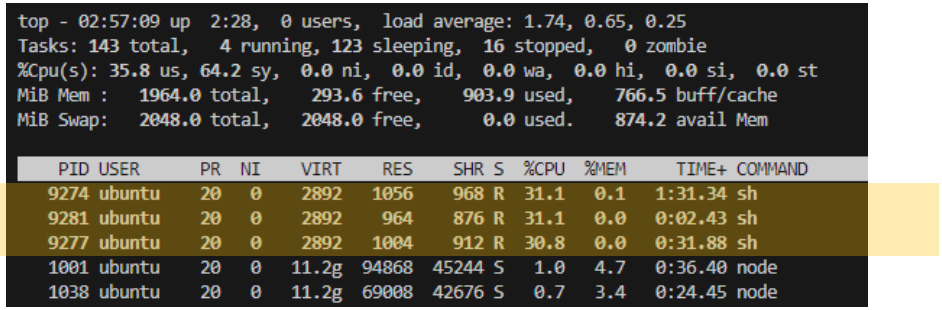
- 작업은 끝내지 못했지만 스케줄 트리거는 계속 반복됩니다
- 프로세스의 수가 점점 늘어나고 이를 해결하지 못하면 처리하지 못하는 프로세스의 양은 급격히 증가합니다
- 결국 정상 처리할 수 있는 작업까지 처리하지 못하는 상황에 처하게 됩니다

- crontab 파일을 아래와 같이 수정하고 crontab -u ubuntu crontab 명령으로 적용합니다
- 1분 마다 스케줄이 반복되지만 작업은 하루가 지나야 종료됩니다

```
*/1 * * * *  sleep 86400; date +"Scheduled at ₩%Y₩%m₩%d-₩%H₩%M₩%S" >> /home/ubuntu/schedule.txt
```

- 만약 스케줄의 내용이 아래와 같다면 어떻게 될까요?

```
*/1 * * * *  while true;do echo "hello"; done date; +"Scheduled at ₩%Y₩%m₩%d-₩%H₩%M₩%S" >> /home/ubuntu/schedule.txt
```



종료되지 못하는 스케줄이 계속 누적중인 상황

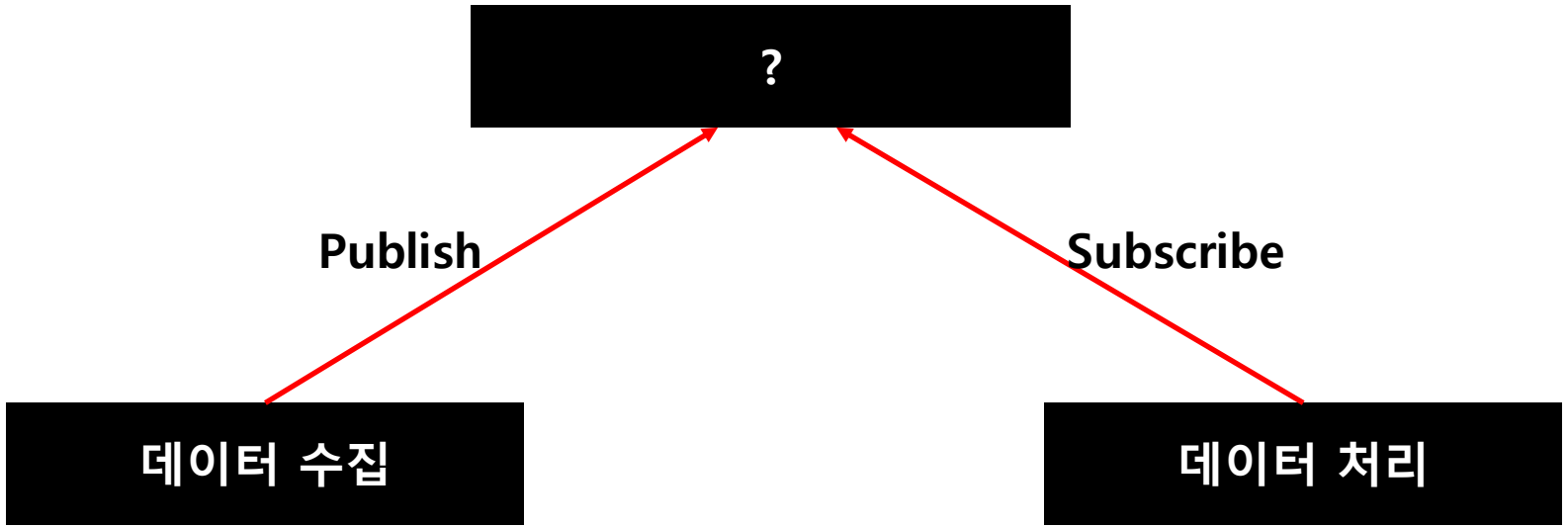
스케줄 반복 주기내로 작업 시간을 제한하기

- 작업을 python으로 전환하여 아래 코드를 생각해봅니다
- 실습을 위한 코드는 온라인강의실에서 실습파일을 다운로드 받아서 사용합니다
- 마찬가지로 hello를 무한히 출력하는 코드입니다
- 그러나 5초의 제한이 있습니다
- 5초가 지나면 SIGALRM이 발생하면서 timeout_handler가 실행됩니다
- timeout_handler는 TimeoutError를 발생시키면서 무한루프로부터 탈출하도록 합니다

```
import signal

def timeout_handler(signum, frame):
    raise TimeoutError("The operation timed out")

signal.signal(signal.SIGALRM, timeout_handler)
signal.alarm(5)

try:
    while True:
        print('hello')
except TimeoutError as e:
    print(e)
```

수집작업과 처리작업을 분리하기

- 수집 프로세스는 수집에만 집중하고 Publish 합니다
- 처리 프로세스는 Subscribe하면서 처리에만 집중합니다
- 이를 위해서 중간에 데이터를 관리해주는 무언가가 필요합니다



정리

- Rest API의 규격을 읽는 방법, Rest API에 매개변수를 전달하는 여러 방법에 대해 알아보았습니다
- 요청 매개변수를 변화시키면서 응답이 변화하는 것을 확인하였습니다
- HTTP CRUD의 python 코드 변환을 위해 보조 도구(postman)을 이용해서 code snippet 추출을 해보았습니다
- cron 스케줄러를 이용해 수집 스케줄을 적용하고 이 때 고려해야 할 유의사항에 대해 알아보았습니다
- 이 중에서 스케줄 내에 데이터 처리가 끝나지 않을 경우를 해결하기 위해 수집 프로세스와 처리 프로세스의 분리를 생각해보았습니다
- 다음 시간에는 Request-Response 구조가 아니라 Publish-Subscribe에 대해 알아보겠습니다
- Apache Kafka를 이용해 Publish-Subscribe 형태로 데이터를 수집하고 처리해봅니다