

Developing On AWS

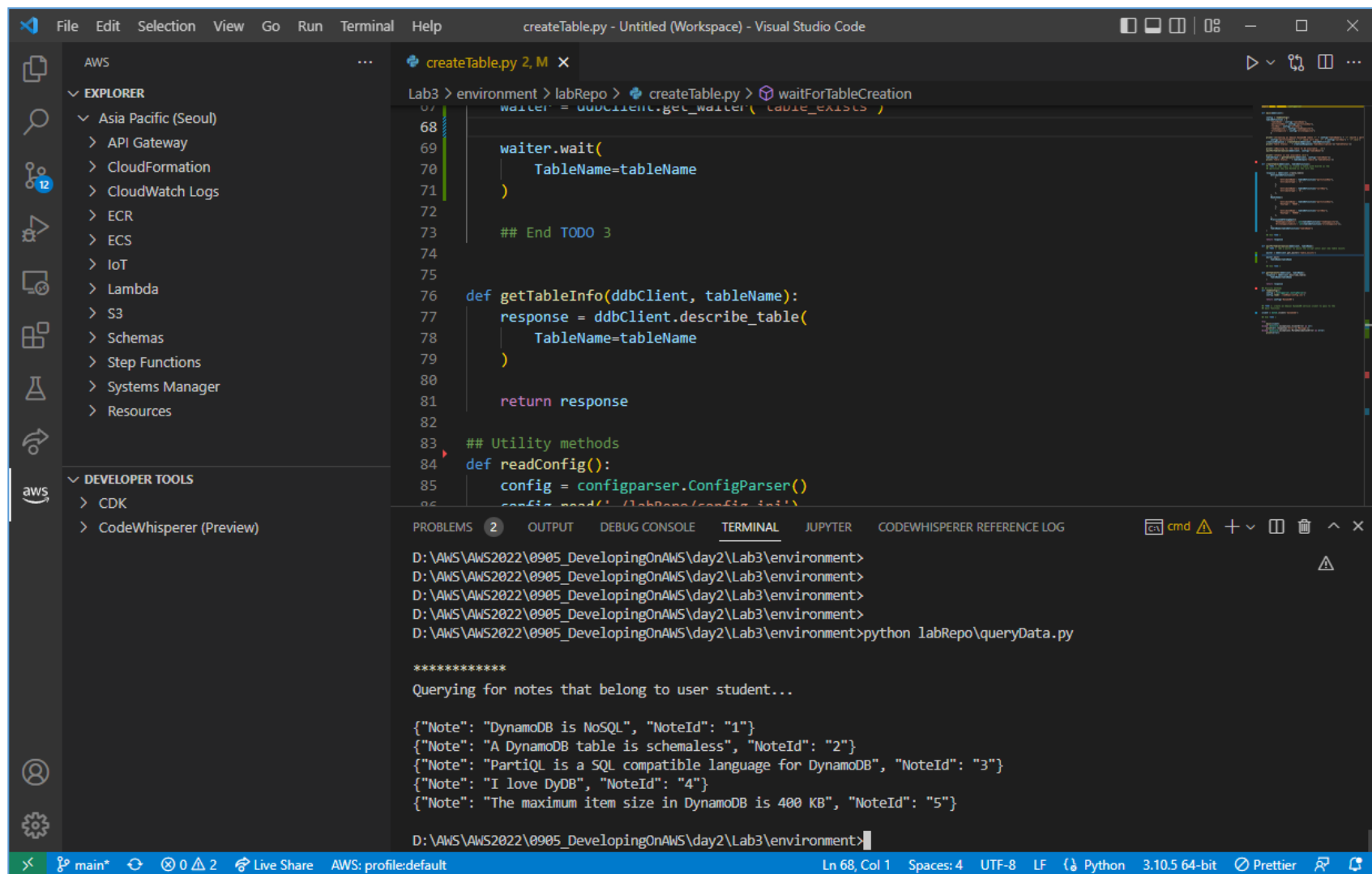


목 차

<u>1. AWS 개발환경</u>2
<u>2. 로컬 개발 환경</u>5
<u>3. AWS CLI 설치</u>11
<u>4. S3를 사용한 솔루션 개발</u>14
<u>5. DynamoDB를 사용한 솔루션 개발</u>20
<u>6. Lambda를 사용한 솔루션 개발</u>27
<u>7. API Gateway를 사용한 솔루션 개발</u>36
<u>8. 캡스톤 - 애플리케이션 구축 완료</u>40
<u>9. AWS X-Ray를 사용하여 애플리케이션 관찰</u>41

1. AWS 개발환경

로컬 환경 : IDE (VS Code)



The screenshot displays the Visual Studio Code IDE interface. The Explorer sidebar on the left shows a project structure for 'AWS' with folders like 'Asia Pacific (Seoul)' and 'API Gateway'. The main editor area shows a Python file named 'createTable.py' with the following code:

```
Lab3 > environment > labRepo > createTable.py > waitForTableCreation
waiter = boto3.client('dynamodb').get_waiter('table_exists')

68
69
70 waiter.wait(
71     TableName=tableName
72 )
73
74 ## End TODO 3
75
76 def getTableInfo(ddbClient, tableName):
77     response = ddbClient.describe_table(
78         TableName=tableName
79     )
80
81     return response
82
83 ## Utility methods
84 def readConfig():
85     config = configparser.ConfigParser()
86     config.read('labRepo\config.ini')
```

The TERMINAL panel at the bottom shows the execution of a Python script:

```
D:\AWS\AWS2022\0905_DevelopingOnAWS\day2\Lab3\environment>
D:\AWS\AWS2022\0905_DevelopingOnAWS\day2\Lab3\environment>
D:\AWS\AWS2022\0905_DevelopingOnAWS\day2\Lab3\environment>
D:\AWS\AWS2022\0905_DevelopingOnAWS\day2\Lab3\environment>
D:\AWS\AWS2022\0905_DevelopingOnAWS\day2\Lab3\environment>python labRepo\queryData.py

*****
Querying for notes that belong to user student...

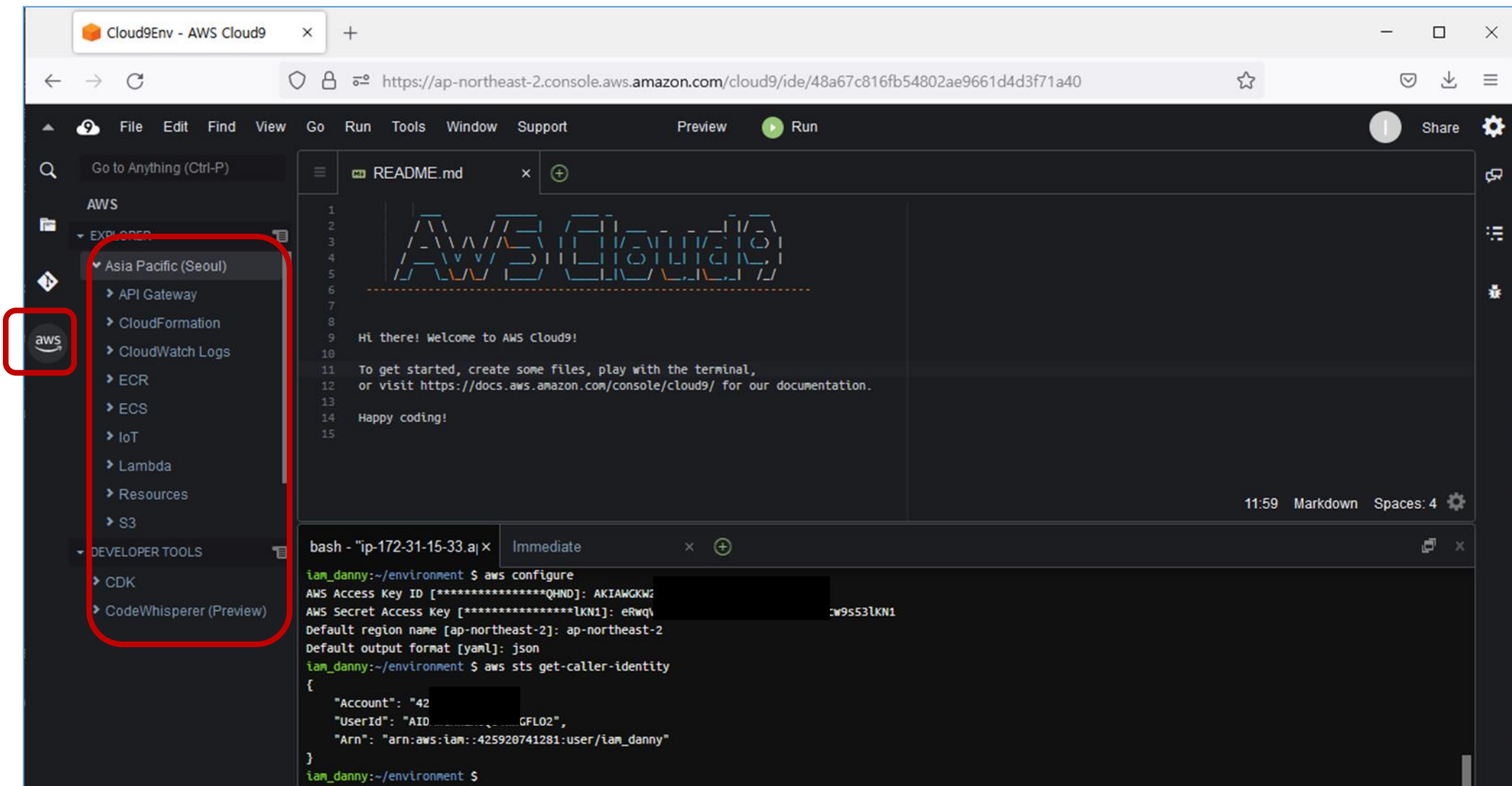
{"Note": "DynamoDB is NoSQL", "NoteId": "1"}
{"Note": "A DynamoDB table is schemaless", "NoteId": "2"}
{"Note": " PartiQL is a SQL compatible language for DynamoDB", "NoteId": "3"}
{"Note": "I love DyDB", "NoteId": "4"}
{"Note": "The maximum item size in DynamoDB is 400 KB", "NoteId": "5"}

D:\AWS\AWS2022\0905_DevelopingOnAWS\day2\Lab3\environment>
```

The status bar at the bottom indicates the current file is 'main*', the workspace is 'AWS: profile:default', and the Python version is '3.10.5 64-bit'.

클라우드 환경 : AWS Cloud9

AWS Explorer를 사용하여 다양한 AWS 서비스의 리소스를 확인하고, 생성하고, 업데이트하고, 삭제할 수 있습니다.

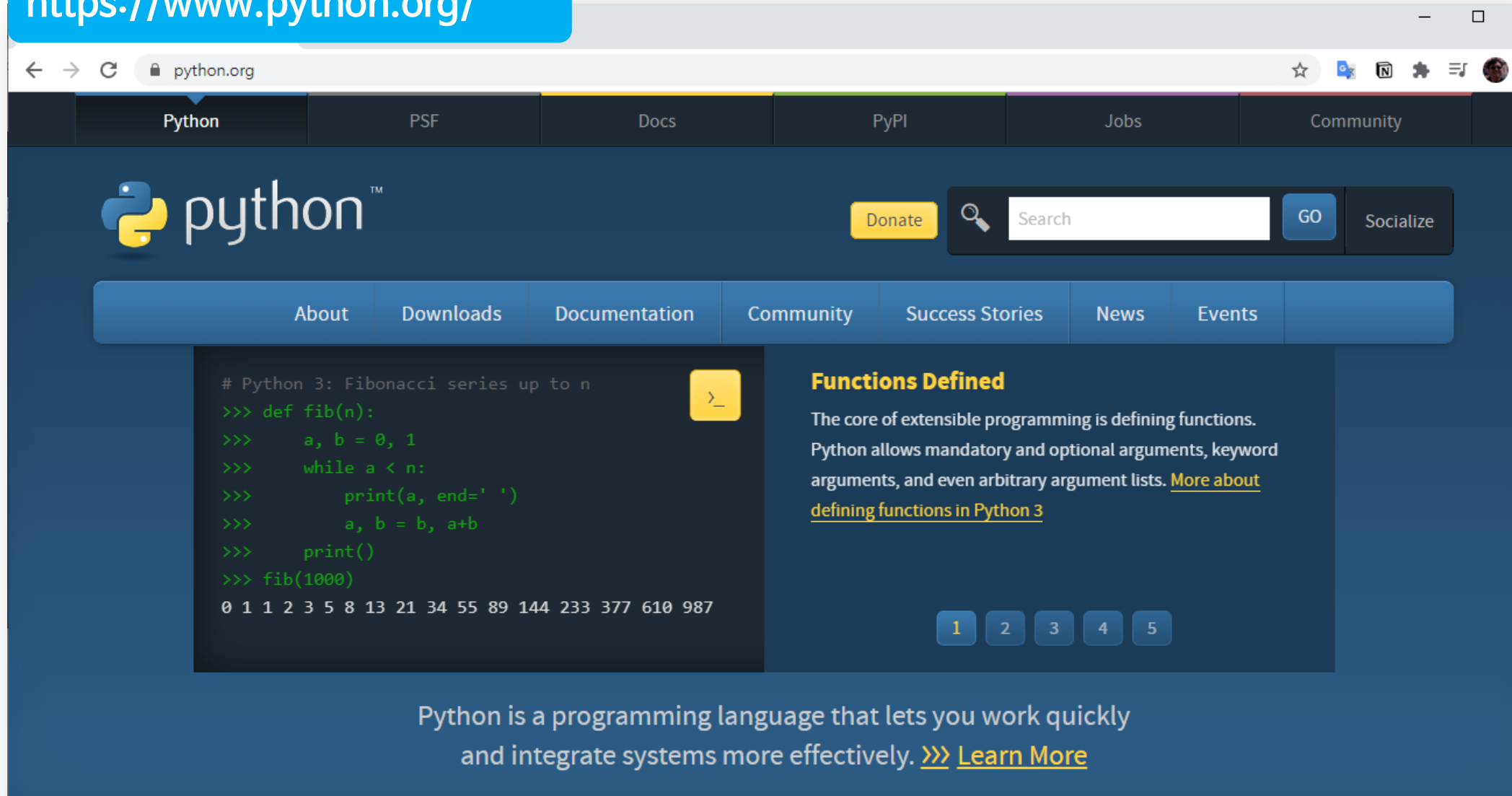


2. 로컬 개발 환경

Python 설치
VS Code 설치
Git 설치

Python 사이트

https://www.python.org/



The screenshot shows the Python.org homepage. At the top, there's a navigation bar with links: Python, PSF, Docs, PyPI, Jobs, and Community. Below this is a dark blue header with the Python logo, a 'Donate' button, a search bar with a 'GO' button, and a 'Socialize' button. A secondary navigation bar contains links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area is split into two columns. The left column features a code editor with a Python 3 Fibonacci function and its output. The right column has a section titled 'Functions Defined' with a paragraph about Python's extensibility and a link to 'More about defining functions in Python 3'. At the bottom, a blue banner contains the text: 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'.

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. [More about defining functions in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

Python 설치파일 다운로드

■ 파이썬 설치

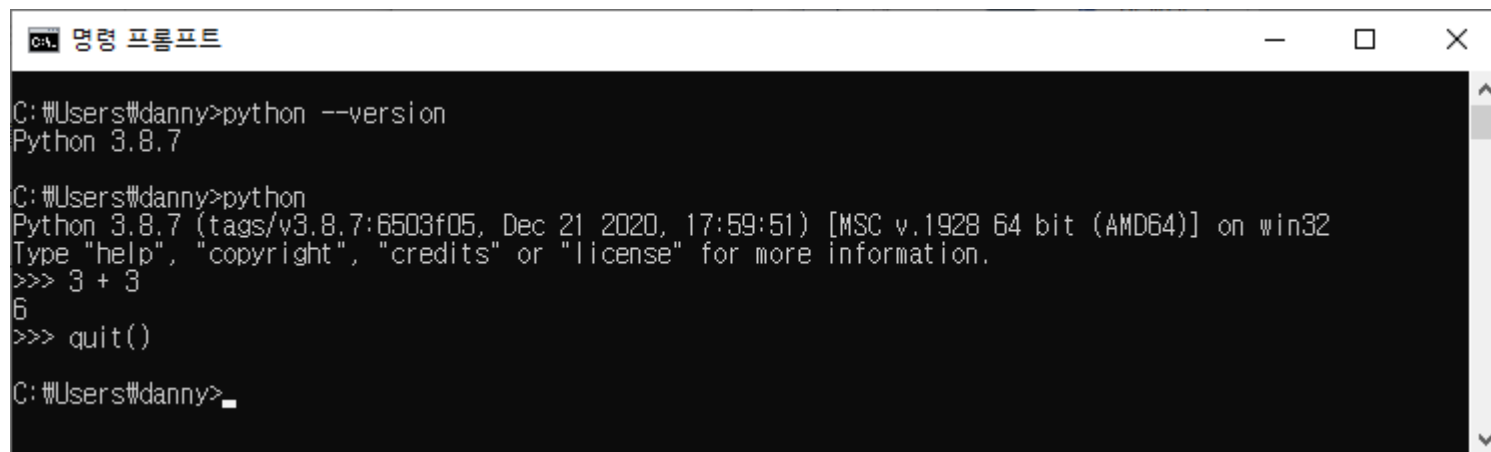
<https://www.python.org/downloads/>

Python 3.8.7

Release Date: Dec. 21, 2020

■ 파이썬 실행

- 버전 확인 : `python --version`
- 실행 : `python`
- 종료 : `quit()`



```
C:\Users\danny>python --version
Python 3.8.7

C:\Users\danny>python
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 3 + 3
6
>>> quit()

C:\Users\danny>
```


Python 가상환경 설치

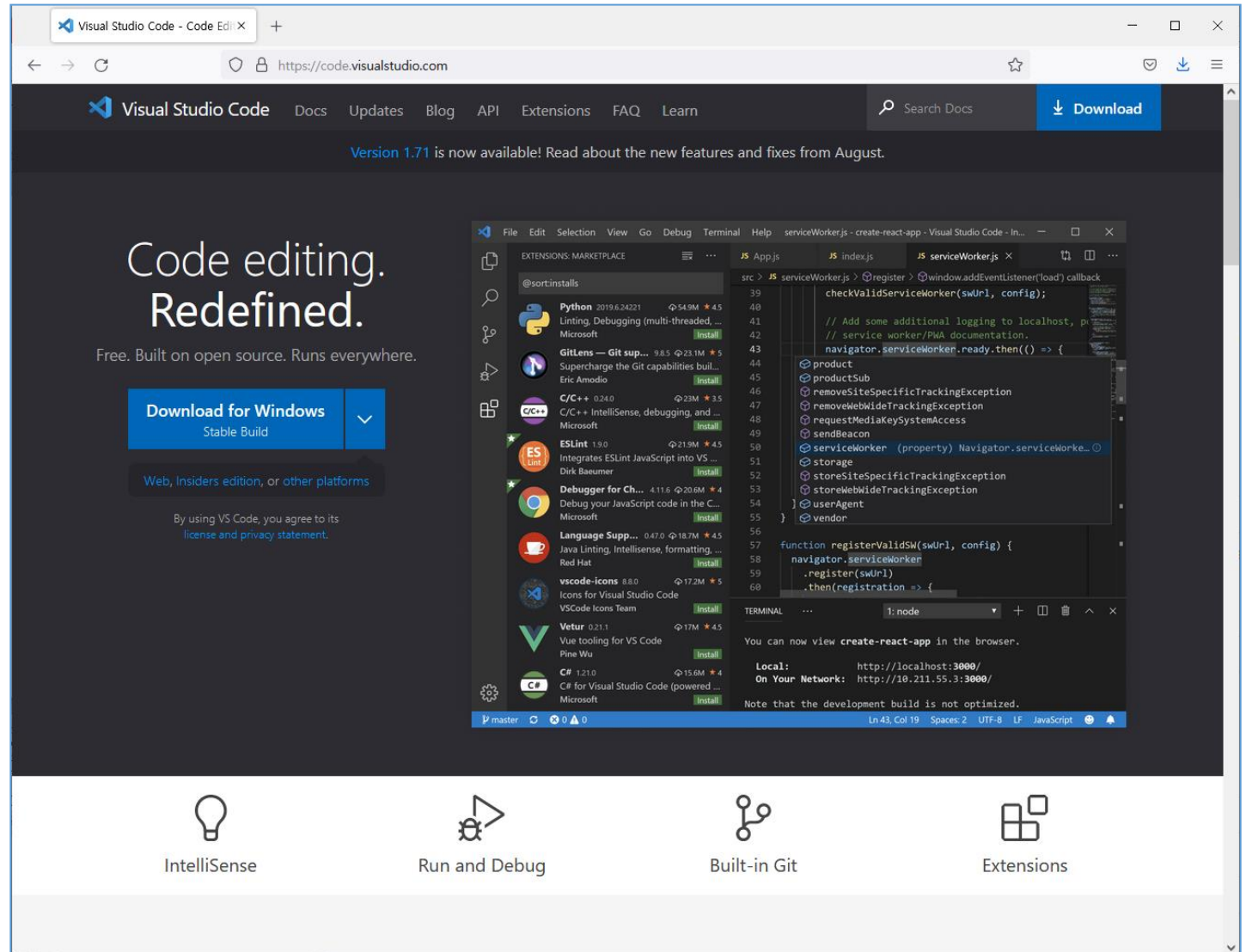
가상 환경(virtual environment)으로 프로젝트별로 독립된 파이썬 실행 환경을 사용할 수 있습니다.

- 가상환경 생성 : `python -m venv venv`
- 가상환경 실행
windows : `venv\Scripts\activate.bat`
Linux, macOS : `source venv/bin/activate`
- JupyterLab/ Jupyter Notebook 설치
`pip install jupyterlab`
`pip install notebook`
- JupyterLab/ Jupyter Notebook 실행
`jupyter-lab` (또는 `jupyter lab`)
`jupyter-notebook` (또는 `jupyter notebook`)
- 패키지 목록 관리
`pip freeze > requirements.txt`
`pip install -r requirements.txt`

VS Code 설치

■ VS Code 설치

<https://code.visualstudio.com/>

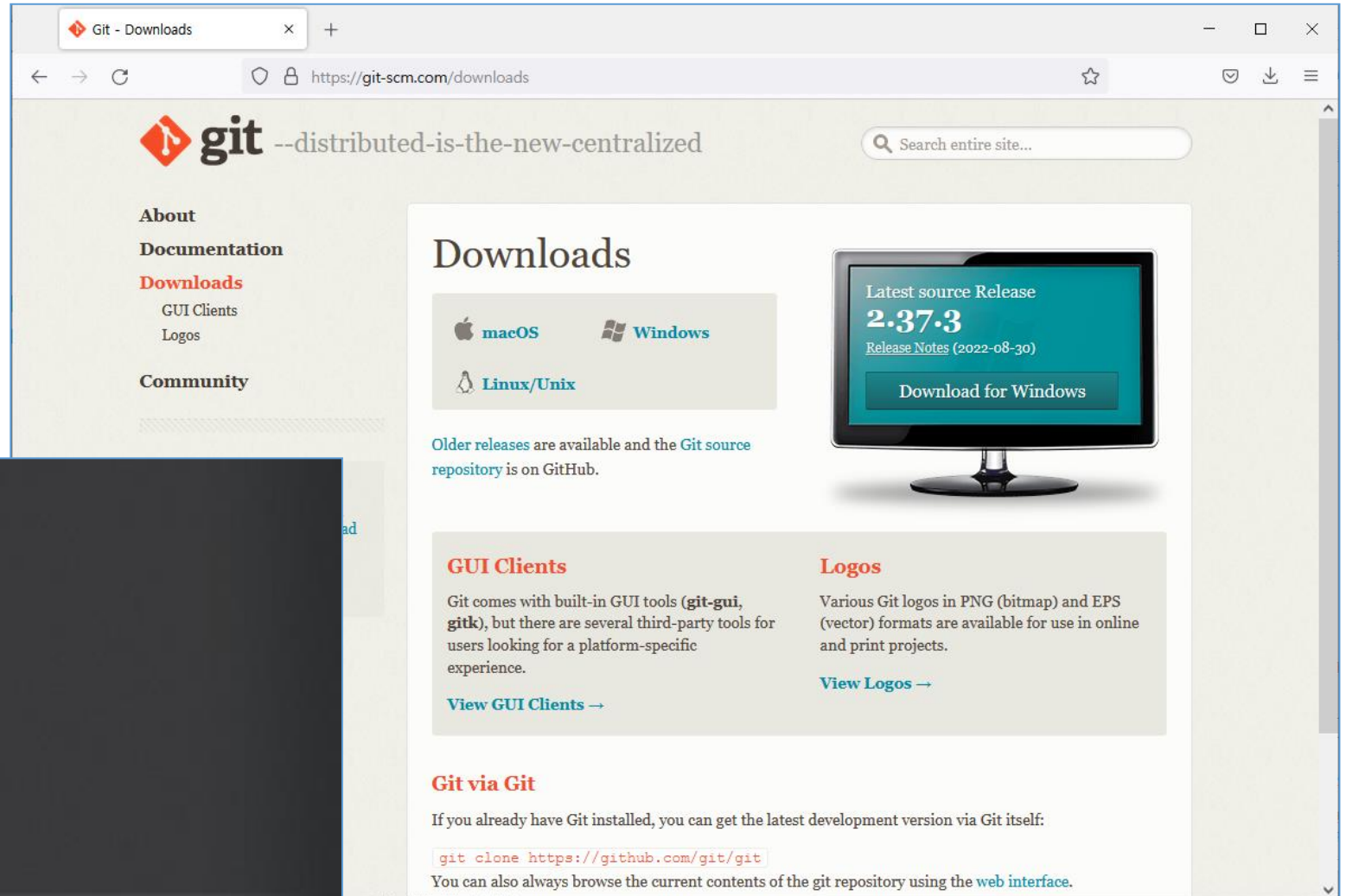
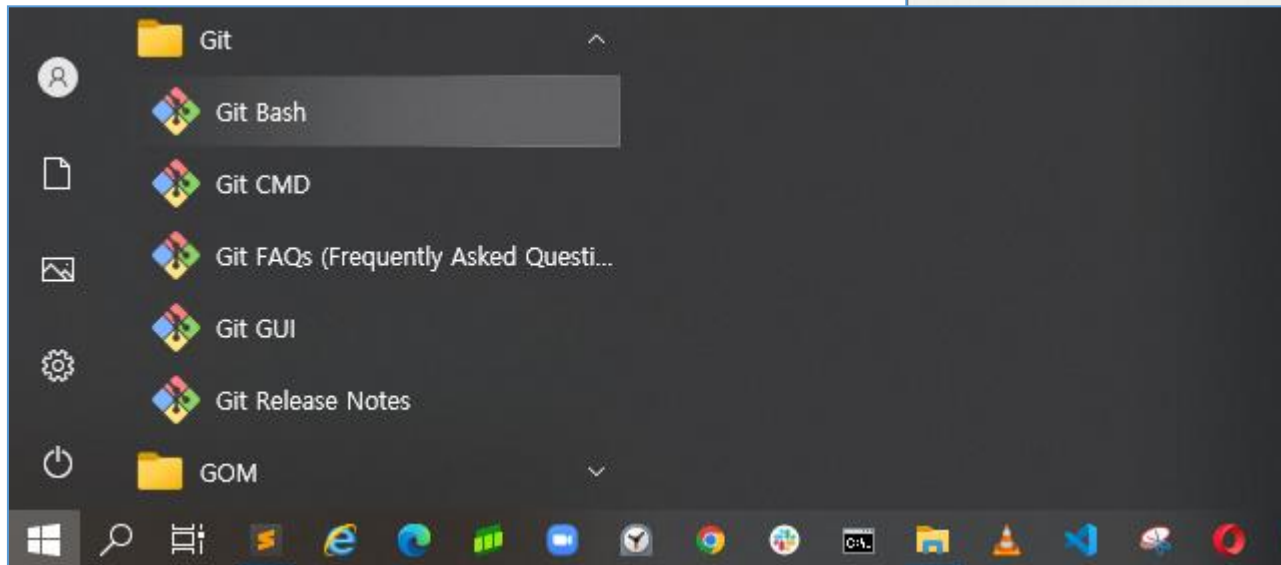


Git 설치

■ Git 설치

<https://git-scm.com/downloads>

■ Git Bash 사용 : Linux Shell



3. AWS CLI 설치

AWS Console : 액세스 키 만들기

The screenshot shows the AWS IAM Management Console interface. The left sidebar contains navigation links for Identity and Access Management (IAM), including Dashboard, Access Management, Users, Roles, Policies, Credential Providers, Reports & Access, Access Analyzer, Archiving, Analysis, Settings, Credential Reports, Activity, and SCP (Service Control Policies). The main content area is titled '내 보안 자격 증명' (My Security Credentials) and shows account details for 'iam_danny'.

Account Details:

- 계정 ID: 4259-2074-1281
- IAM 사용자: iam_danny
- 사용자 이름: iam_danny (에 생성됨)
- 사용자 ARN: arn:aws:iam::425920741281:user/iam_danny
- AWS 계정 ID: 425920741281
- 계정 정규 사용자 ID: d68d60c90cc4e201dc14a2f4dc112d28c624880702a4f6aff3ad64ed87906049

Security Credentials Section:

- 콘솔 액세스를 위한 암호: IAM 사용자로서 AWS Management Console에 액세스하는 데 암호가 필요합니다. 암호를 정기적으로 변경하는 것이 좋습니다.
- CLI, SDK 및 API 액세스를 위한 액세스 키: 액세스 키를 사용하여 AWS CLI, PowerShell용 도구, AWS SDK 또는 직접 AWS API 호출을 통해 AWS를 프로그래밍 방식으로 호출합니다. 한 번에 최대 두 개의 액세스 키 (활성 또는 비활성)를 가질 수 있습니다.

Access Key Table:

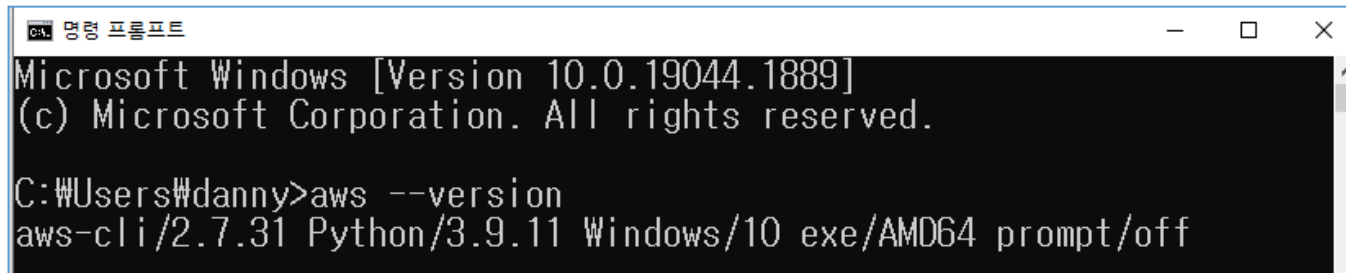
액세스 키 ID	상태	생성 완료	마지막 사용	작업
AKIAWGKWX2X6Q4Q4NQHND	(활성)	2022-09-08 00:00 UTC+0900	2022-09-09 00:17 UTC+0900	비활성화 삭제

Buttons: '액세스 키 만들기' (Create Access Key), '암호 변경' (Change Password), '역할 전환' (Switch Role), '로그아웃' (Logout).

AWS CLI 설치

■ AWS CLI 설치

- https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/getting-started-install.html
- `aws --version`

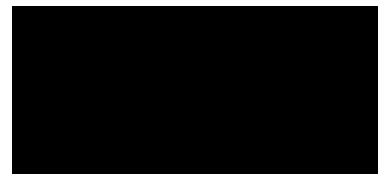


```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

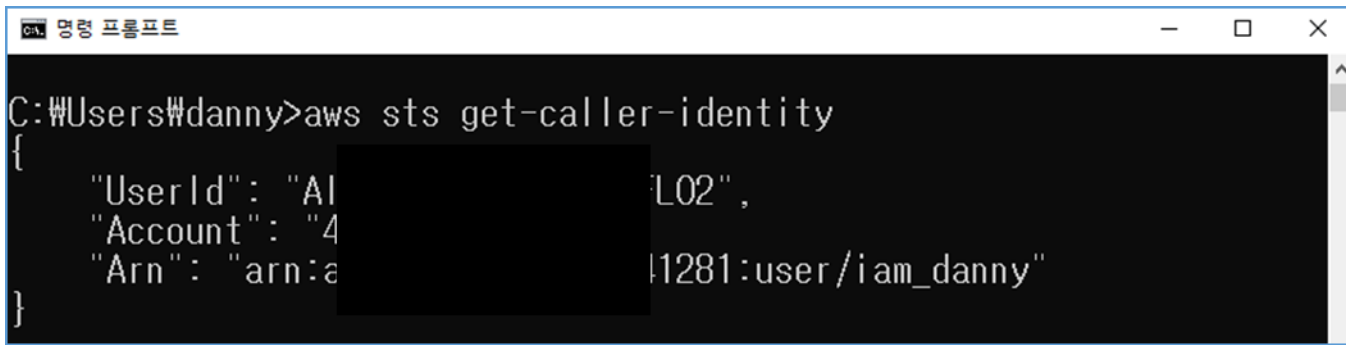
C:\Users\danny>aws --version
aws-cli/2.7.31 Python/3.9.11 Windows/10 exe/AMD64 prompt/off
```

■ aws configure

- AWS Access Key ID [비워 둌]: ENTER 키를 누릅니다.
- AWS Secret Access Key [비워 둌]: ENTER 키를 누릅니다.
- Default region name [적절한 리전으로 업데이트]: REGION
- Default output format [yaml으로 업데이트]: yaml



■ aws sts get-caller-identity



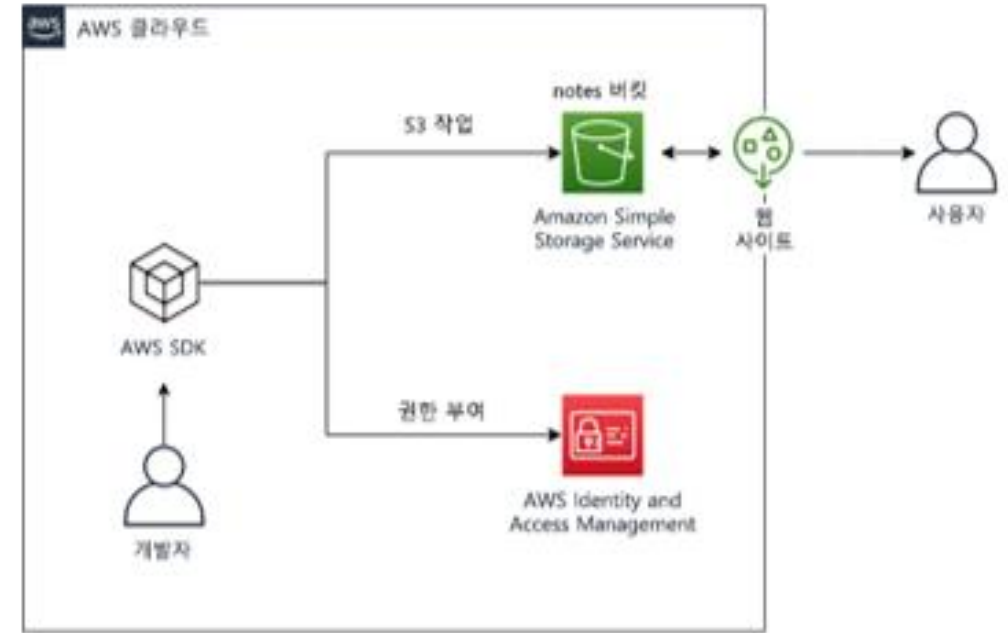
```
C:\Users\danny>aws sts get-caller-identity
{
  "UserId": "A1XXXXXXXXXXXXL02",
  "Account": "411111111111",
  "Arn": "arn:aws:iam::411111111111:user/iam_danny"
}
```

4. S3를 사용한 솔루션 개발

S3

■ 목표

- AWS SDK 및 AWS CLI를 사용하여 프로그래밍 방식으로 Amazon S3와 상호 작용
- waiter를 사용하여 버킷을 생성하고 서비스 예외 코드를 확인
- 메타데이터가 첨부된 Amazon S3 객체를 업로드하는 데 필요한 요청을 빌드
- 버킷에서 객체를 다운로드하는 요청을 빌드하고, 데이터를 처리하고, 객체를 버킷에 다시 업로드
- AWS CLI를 사용하여 웹 사이트를 호스트하고 소스 파일을 동기화하도록 버킷을 구성



■ config.ini

```
[S3]
bucket_name = notes-bucket-danny-1214 (변경 필요)
object_name = ./notes
key_name = notes
source_file_extension = .csv
source_content_type = text/csv
processed_file_extension = .json
processed_content_type = application/json
metaData_key = myVal2
metaData_value = lab2-testing-upload
```


1. S3 버킷 생성 : create-bucket.py

- JupyterLab / Jupyter Notebook 설치
pip install jupyterlab

행
lab)
notebook)

```
In [2]: def createBucket(s3Client, name):
        session = boto3.session.Session()

        # Obtain the region from the boto3 session
        current_region = session.region_name
        print('\nCreating ' + name + ' in ' + current_region)

        # Start TODO 3: Create a new bucket in the users current region
        # and return the response in a response variable.
        if current_region == 'us-east-1':
            response = s3Client.create_bucket(Bucket=name)
```

```
In [10]: s3Client = boto3.client('s3')

        config = readConfig()
        bucket_name = config['bucket_name']

        ##### Verify that the bucket exists.
        verifyBucketName(s3Client, bucket_name)
        print(bucket_name)

        ##### Create the notes-bucket-
        createBucket(s3Client, bucket_name)

        ##Pause until the the bucket is in the account
        verifyBucket(s3Client, bucket_name)
```

2. Amazon S3에 객체 업로드 : create-object.py

```
In [ ]: import boto3, botocore, configparser
```

```
In [15]: def uploadObject(s3Client, bucket, name, key, contentType, metadata={}):
    ## create a object by transferring the file to the S3 bucket,
    ## set the contentType of the file and add any metadata passed to this function.
    response = s3Client.upload_file(
        Bucket=bucket,
        Key=key,
        Filename=name,
        ExtraArgs={
            'ContentType': contentType,
            'Metadata': metadata
        }
    )

    return "Finished creating object\n"
```

```
In [16]: s3Client = boto3.client('s3')

config = readConfig()

bucket_name = config['bucket_name']
source_file_name = config["object_name"] + config['source_file_extension']
key_name = config['key_name'] + config['source_file_extension']
contentType = config['source_content_type']
metaData_key = config['metaData_key']
metaData_value = config['metaData_value']

#### Create object in the s3 bucket
print(uploadObject(s3Client, bucket_name, source_file_name, key_name, contentType, {metaData_key: metaData_value}))
```

3. Amazon S3에 저장된 객체의 데이터 처리 : convert-csv-to-json.py

```
In [20]: def convertToJSON(input):
        jsonList = []
        keys = []

        csvReader = csv.reader(input.split('\n'), delimiter=",")

        for i, row in enumerate(csvReader):
            if i == 0:
                keys = row
            else:
                obj = {}
                for v, val in enumerate(keys):
                    obj[val] = row[v]

        return jsonList

In [25]: client = boto3.client('s3')

        config = readConfig()

        bucket_name = config['bucket_name']
        source_file_name = config['object_name'] + config['source_file_extension']
        key_name = config['key_name'] + config['source_file_extension']
        processed_file_name = config['key_name'] + config['processed_file_extension']
        contentType = config['processed_content_type']
        metaData_key = config['metaData_key']
        metaData_value = config['metaData_value']

        ##### Get the object from S3
        csvStr = getCSVFile(s3Client, bucket_name, key_name)

        ## Convert the object to the new format
        jsonStr = convertToJSON(csvStr)

        ## Uploaded the converted object to S3
        createObject(s3Client, bucket_name, processed_file_name, jsonStr, contentType, {metaData_key: metaData_value})
```

4. AWS CLI를 사용하여 S3 버킷에서 정적 웹 사이트 호스팅 구성

- 버킷 이름이 포함된 변수를 생성

```
mybucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `notes-bucket`) == `true`].Name')
```

- html 폴더의 파일을 버킷과 동기화

```
aws s3 sync ~/html/. s3://$mybucket/
```

- Amazon S3 웹 사이트 호스팅을 활성화

```
aws s3api put-bucket-website --bucket $mybucket --website-configuration file:///~/website.json
```

- 버킷 이름을 교체

```
sed -i "s/#[BUCKET#]/$mybucket/g" ~/policy.json
```

```
cat ~/policy.json
```

- 버킷 정책 적용

```
aws s3api put-bucket-policy --bucket $mybucket --policy file:///~/policy.json
```

- 리전 값 변수 설정

```
region=ap-northeast-2
```

[http://\\$mybucket.s3-website-\\$region.amazonaws.com](http://$mybucket.s3-website-$region.amazonaws.com) 접속

<http://notes-bucket-danny-1214.s3-website.ap-northeast-2.amazonaws.com> 접속

5. DynamoDB를 사용한 솔루션 개발

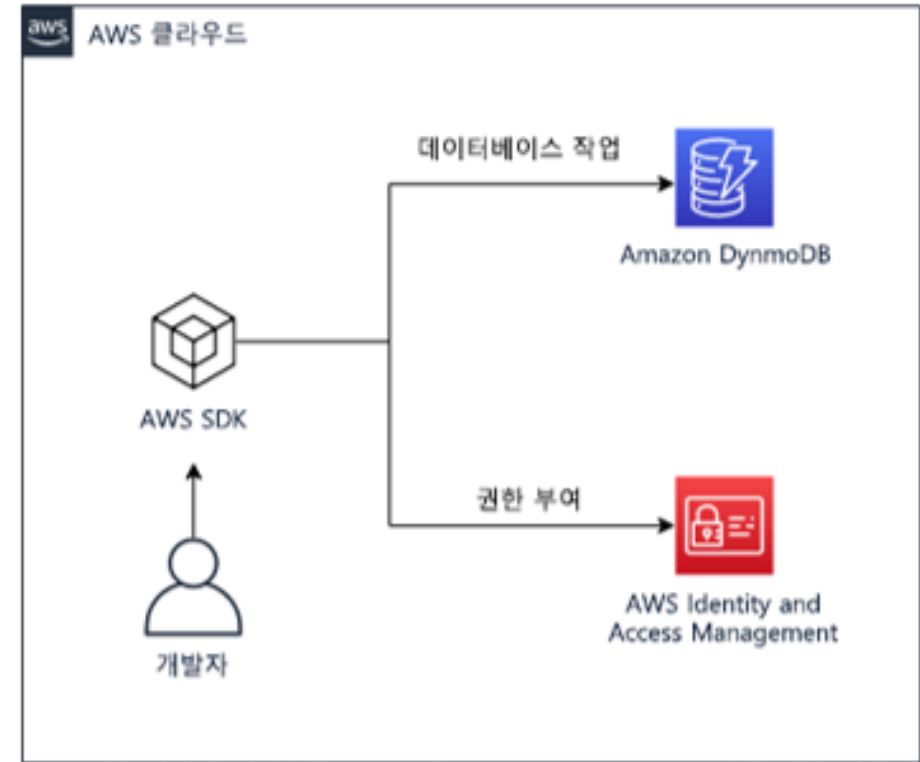
DynamoDB

■ 목표

- 프로그램의 하위 수준, 문서 및 상위 수준 API를 사용하여 프로그래밍 방식으로 DynamoDB와 상호 작용
- 파티션 키, 정렬 키 및 적합한 프로비저닝된 처리량을 포함한 Waiter를 사용하여 테이블 생성
- 파일에서 JSON 객체를 읽고 테이블을 로드
- 키 속성, 필터, 표현식 및 페이지 배열을 사용하여 테이블에서 항목을 검색
- 새 속성을 추가하고 조건부로 데이터를 변경하여 항목을 업데이트
- PartiQL을 사용하여 DynamoDB 데이터에 액세스

■ config.ini

```
[DynamoDB]
tableName = Notes
partitionKey = UserId
sortKey = NotedId
readCapacity = 1
writeCapacity = 1
sourcenotes = ./notes.json
queryUserId = student
pageSize = 3
queryNotedId = 5
notePrefix = The maximum item size in DynamoDB is
```



DynamoDB

2. 테이블에 데이터 로드 : loadData.py

```
In [21]: import boto3, botocore, configparser, json
```

```
In [22]: def putNote(table, note):  
    print("loading note " + str(note))  
    table.put_item(  
        Item={  
            'UserId': note["UserId"],  
            'NoteId': int(note["NoteId"]),  
            'Note': note["Note"]  
        }  
    )
```

```
In [23]: ddbResource = boto3.resource('dynamodb')
```

```
In [24]: tableName = config['tableName']  
    jsonFileName = config['sourcenotes']  
  
    # Opening JSON file  
    f = open(jsonFileName)  
  
    print(f"Loading {tableName} table with data from file {jsonFileName}")  
    # Load json object from file  
    notes = json.load(f)  
  
    # Create dynamodb table resource  
    table = ddbResource.Table(tableName)  
  
    # Iterating through the notes and putting them in the table  
    for n in notes:  
        putNote(table, n)  
  
    # Closing the JSON file  
    f.close()  
    print("Finished loading notes from the JSON file")
```

DynamoDB

3. 파티션 키 및 프로젝션을 사용하여 데이터 쿼리 : loadData.py

```
In [25]: import boto3, botocore, json, decimal, configparser
from boto3.dynamodb.conditions import Key, Attr
from boto3.dynamodb.types import TypeDeserializer
```

```
In [26]: config = readConfig()
tableName = config['tableName']
UserId = config['queryUserId']
```

```
In [27]: def queryNotesByPartitionKey(ddbClient, tableName, qUserId):
    response = ddbClient.query(
        TableName=tableName,
        KeyConditionExpression='UserId = :userId',
        ExpressionAttributeValues={
            ':userId': {"S": qUserId}
        },
        ProjectionExpression="NoteId, Note"
    )
    return response["Items"]
```

```
In [28]: ## Utility methods
def printNotes(notes):
    if isinstance(notes, list):
        for note in notes:
            print(
                json.dumps(
                    {key: TypeDeserializer().deserialize(value) for key, value in note.items()},
                    cls=DecimalEncoder
                )
            )
```


DynamoDB

4: Paginator를 사용하여 테이블 스캔 : paginateData.py

```
In [31]: import boto3, botocore, json, decimal, configparser
from boto3.dynamodb.conditions import Key, Attr
from boto3.dynamodb.types import TypeDeserializer
```

```
In [32]: def queryAllNotesPaginator(ddbClient, tableName, pageSize):
    # Create a reusable Paginator
    paginator = ddbClient.get_paginator('scan')

    # Create a PageIterator from the Paginator
    page_iterator = paginator.paginate(
        TableName=tableName,
        PaginationConfig={
            'PageSize': pageSize
        })

    pageNumber = 0
    for page in page_iterator:
        if page["Count"] > 0:
            pageNumber += 1
            print("Starting page " + str(pageNumber))
            printNotes(page['Items'])
            print("End of page " + str(pageNumber) + "\n")
```

```
In [33]: config = readConfig()
tableName = config['tableName']
pageSize = config['pageSize']

ddbClient = boto3.client('dynamodb')

print("\n*****\nScanning with pagination...\n")
queryAllNotesPaginator(ddbClient, tableName, pageSize)
```

5. 테이블의 항목 업데이트 : updateItem.py

```
In [34]: import boto3, botocore, configparser
```

```
In [35]: def updateNewItem(ddbClient, tableName, qUserId, qNoteId):
    ## TODO : Add code to set an 'Is_Incomplete' flag to 'Yes' for the note that matches the
    ## provided function parameters
    response = ddbClient.update_item(
        TableName=tableName,
        Key={
            'UserId': {'S': qUserId},
            'NoteId': {'N': str(qNoteId)}
        },
        ReturnValues='ALL_NEW',
        UpdateExpression='SET Is_Incomplete = :incomplete',
        ExpressionAttributeValues={
            ':incomplete': {'S': 'Yes'}
        }
    )
    return response['Attributes']
```

```
In [36]: def updateExistingAttributeConditionally(ddbClient, tableName, qUserId, qNoteId, notePrefix):
    try:
        ## TODO Add code to update the Notes attribute for the note that matches
        # the passed function parameters only if the 'Is_Incomplete' attribute is 'Yes'

        notePrefix += ' 400 KB'
        response = ddbClient.update_item(
            TableName=tableName,
            Key={
                'UserId': {'S': qUserId},
                'NoteId': {'N': str(qNoteId)}
            },
```

DynamoDB

6. DynamoDB용 PartiQL(SQL 호환 쿼리 언어) 사용 : partiQL.py

```
In [39]: import boto3, botocore, json, decimal, configparser
        from boto3.dynamodb.conditions import Key, Attr
        from boto3.dynamodb.types import TypeDeserializer
```

```
In [40]: def querySpecificNote(ddbClient, tableName, qUserId, qNoteld):
        response = ddbClient.execute_statement(
            Statement="SELECT * FROM " + tableName + " WHERE UserId = ? AND Noteld = ?",
            Parameters=[
                {"S": qUserId},
                {"N": str(qNoteld)}
            ]
        )
        return response["Items"]
```

```
In [41]: config = readConfig()
        tableName = config['tableName']
        UserId = config['queryUserId']
        Noteld = config['queryNoteld']

        ddbClient = boto3.client('dynamodb')

        print(f"\n*****\nQuerying for note {Noteld} that belongs to user {UserId}...\n")
        printNotes(querySpecificNote(ddbClient, tableName, UserId, Noteld))
```

Querying for note 5 that belongs to user student...

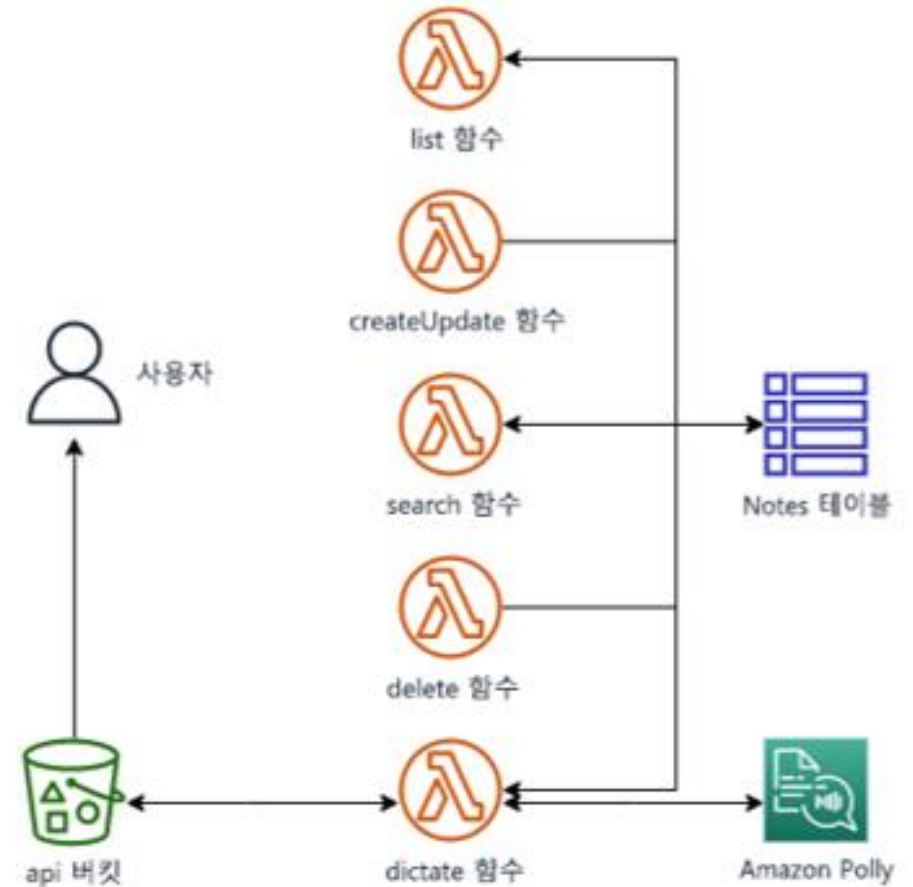
```
{"Note": "The maximum item size in DynamoDB is 400 KB", "UserId": "student", "Noteld": "5", "Is_Incomplete": "No"}
```

6. Lambda를 사용한 솔루션 개발

Lambda

■ 목표

- AWS Lambda 함수를 생성하고 AWS SDK 및 AWS CLI를 사용하여 프로그래밍 방식으로 상호 작용
- Lambda 함수를 구성하여 환경 변수를 사용하고 다른 서비스와 통합
- AWS SDK를 사용하여 Amazon S3 미리 서명된 URL을 생성하고 버킷 객체에 대한 액세스를 확인
- .zip 파일 아카이브를 사용하여 Lambda 함수를 배포하고 필요한 경우 테스트
- AWS Console 및 AWS CLI를 사용하여 AWS Lambda 함수를 호출



Role 생성 : lambdaPollyRole

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "polly:SynthesizeSpeech",
        "s3:ListBucket",
        "s3:PutObject",
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:ConditionCheckItem",
        "dynamodb:PutItem",
        "dynamodb:DescribeTable",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:Scan",
        "dynamodb:Query",
        "dynamodb:UpdateItem"
      ],
      "Resource": "*"
    }
  ]
}
```

Lambda 함수 : app.py

<https://github.com/kgpark88/cloud-native/blob/main/Lambda/app.py>

```
import boto3
import os
from contextlib import closing

dynamoDBResource = boto3.resource('dynamodb')
pollyClient = boto3.client('polly')
s3Client = boto3.client('s3')

def lambda_handler(event, context):
    # Extract the user parameters from the event and environment
    UserId = event["UserId"]
    NoteId = event["NoteId"]
    VoiceId = event['VoiceId']
    mp3Bucket = os.environ['MP3_BUCKET_NAME']
    ddbTable = os.environ['TABLE_NAME']

    # Get the note text from the database
    text = getNote(dynamoDBResource, ddbTable, UserId, NoteId)

    # Save a MP3 file locally with the output from polly
    filePath = createMP3File(pollyClient, text, VoiceId, NoteId)

    # Host the file on S3 that is accessed by a pre-signed url
    signedURL = hostFileOnS3(s3Client, filePath, mp3Bucket, UserId, NoteId)

    return signedURL
```

Lambda 함수 생성

■ Lambda 함수 생성(AWS Console 에서 실행)

- Function name : dictate-function
- Runtime : Python 3.9
- Change default execution role을 확장하고 Use an existing role을 선택
- Existing role에서 lambdaPollyRole을 선택

■ 환경 변수 추가(git bash 에서 실행)

- `apiBucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `notes-bucket`)] == `true`' | [0].Name)`
- `notesTable='Notes'`
- `aws lambda update-function-configuration --function-name dictate-function --environment Variables="{MP3_BUCKET_NAME=$apiBucket, TABLE_NAME=$notesTable}"`

■ Lambda 함수 게시(git bash 에서 실행)

- o **app.py** 파일을 zip 파일로 압축

`zip dictate-function.zip app.py`

- o Lambda 함수에 새 코드를 업로드

`aws lambda update-function-code --function-name dictate-function --zip-file fileb://dictate-function.zip`

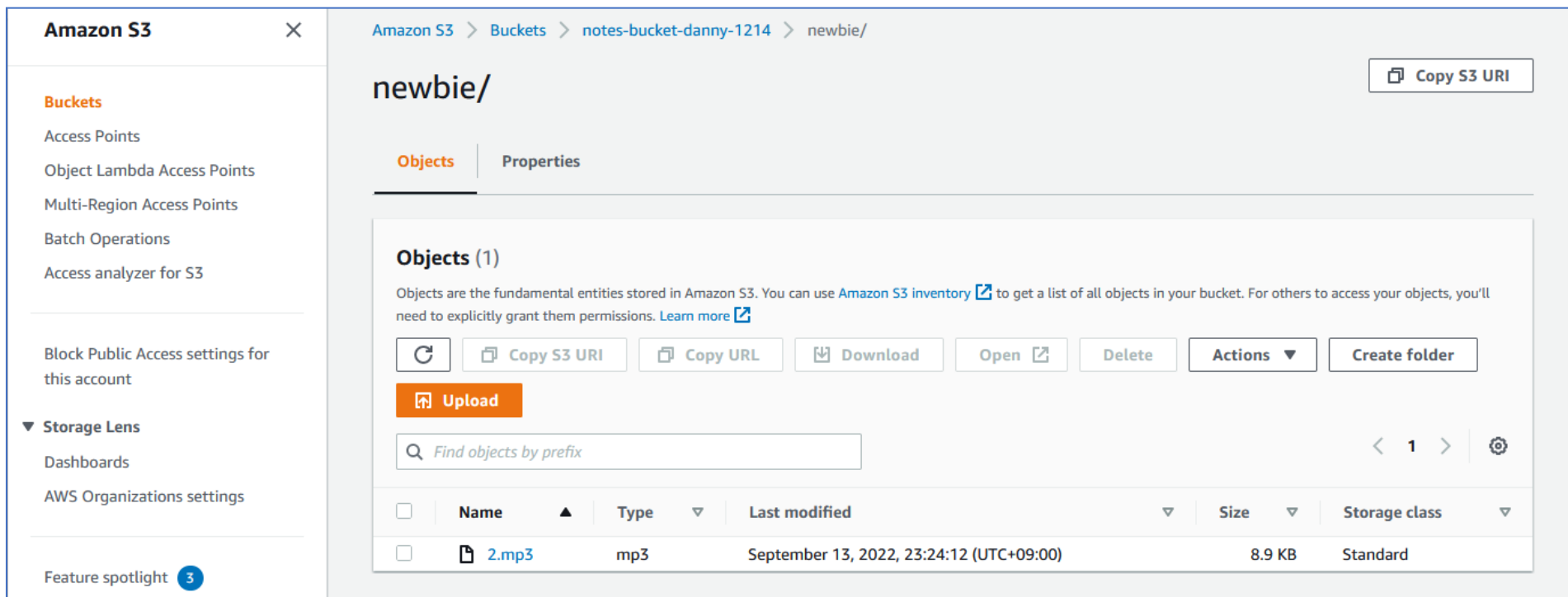
- o 함수 핸들러 업데이트

`aws lambda update-function-configuration --function-name dictate-function --handler app.lambda_handler`

Lambda 함수 호출(git bash 에서 실행)

- dictate-function 폴더에 오른쪽 마우스를 클릭 하고 New File 를 선택하여 event.json 파일 생성
- event.json 편집

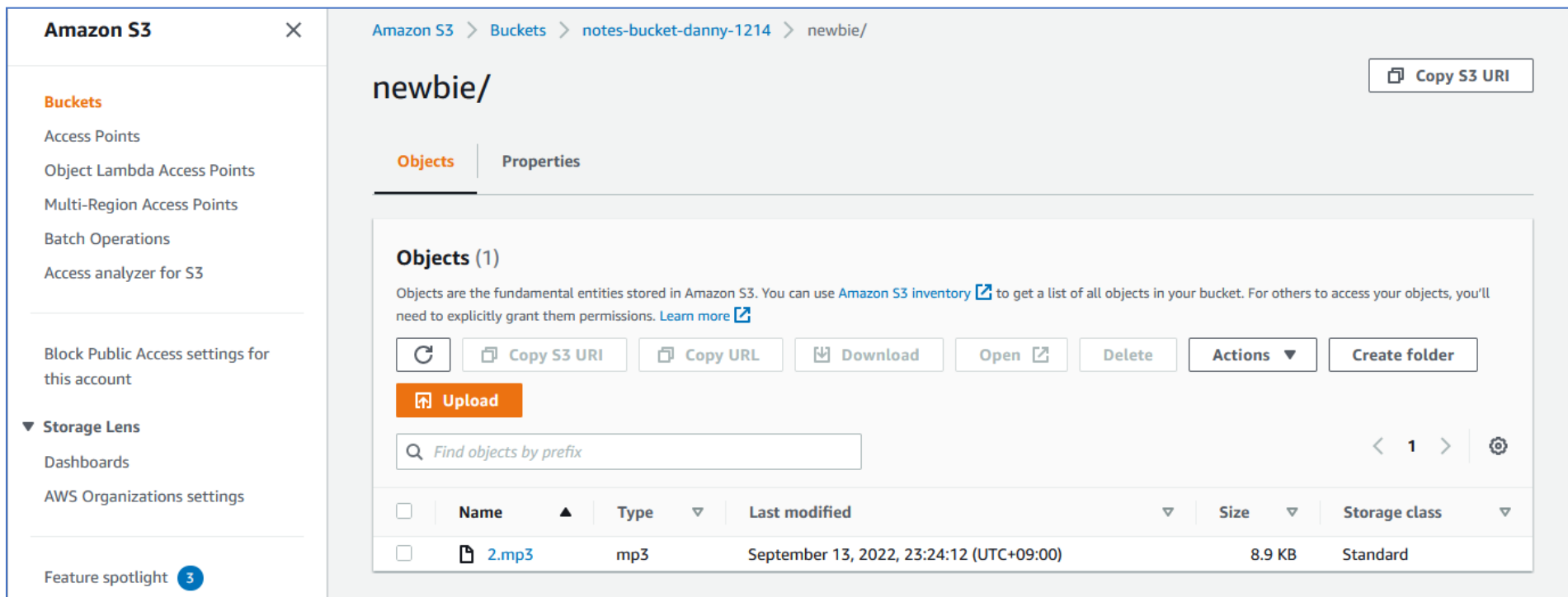
```
{ "UserId": "newbie", "NotelId": "2", "VoicelId": "Joey" }
```
- Lambda 함수 호출
`aws lambda invoke --function-name dictate-function --payload fileb://event.json response.txt`
- 결과 확인



Lambda 함수 호출(git bash 에서 실행)

- dictate-function 폴더에 오른쪽 마우스를 클릭 하고 New File 를 선택하여 event.json 파일 생성
- event.json 편집

```
{ "UserId": "newbie", "Noteld": "2", "Voiceld": "Joey" }
```
- Lambda 함수 호출
`aws lambda invoke --function-name dictate-function --payload fileb://event.json response.txt`
- 결과 확인



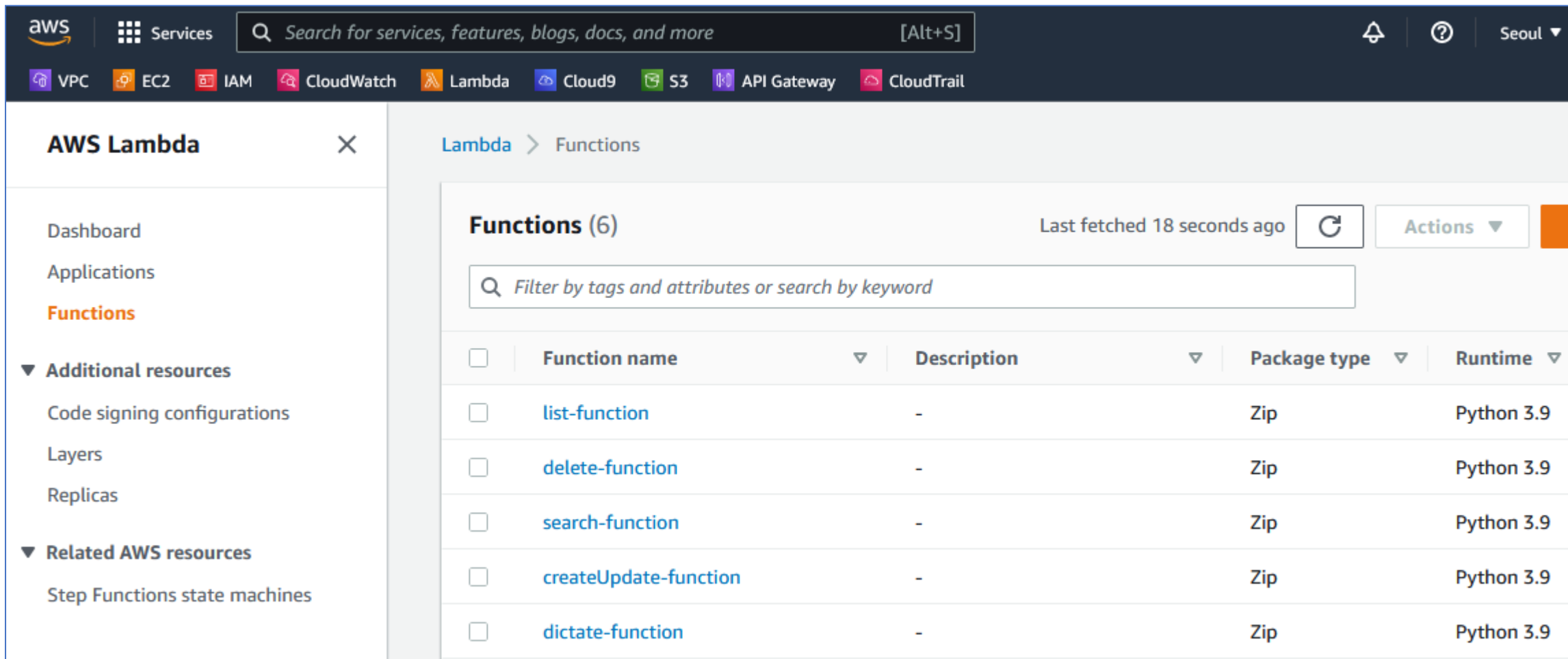
추가 Lambda 함수 생성

- 추가 함수명 변수 생성
folderName=createUpdate-function
- lambdaPollyRole의 ARN에 대한 변수 생성
roleArn=\$(aws iam list-roles --output text --query 'Roles[?contains(RoleName, `lambdaPollyRole`) == `true`].Arn')


■ 함수 추가 단계

- cd ~/\$folderName
- zip \$folderName.zip app.py
- functionName=\$folderName
- aws lambda create-function --function-name \$functionName --handler app.lambda_handler --runtime python3.9 --role \$roleArn --environment Variables={TABLE_NAME=\$notesTable} --zip-file fileb://\$folderName.zip
- 위 함수 추가 단계를 사용하여 나머지 함수를 생성
folderName=search-function
folderName=delete-function
folderName=list-function

추가 Lambda 함수 생성



The screenshot shows the AWS Lambda console interface. The top navigation bar includes the AWS logo, a search bar, and a list of services: VPC, EC2, IAM, CloudWatch, Lambda, Cloud9, S3, API Gateway, and CloudTrail. The left sidebar is titled 'AWS Lambda' and contains links to Dashboard, Applications, Functions (highlighted), and Additional resources (Code signing configurations, Layers, Replicas). Below these are Related AWS resources (Step Functions state machines). The main content area is titled 'Lambda > Functions' and shows a list of 6 functions. The list has a search bar and a refresh button. The functions listed are: list-function, delete-function, search-function, createUpdate-function, and dictate-function. Each function has a checkbox, a name, a description (all are '-'), a package type (all are 'Zip'), and a runtime (all are 'Python 3.9').

Functions (6) Last fetched 18 seconds ago  **Actions** ▼

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Function name ▼	Description ▼	Package type ▼	Runtime ▼
<input type="checkbox"/>	list-function	-	Zip	Python 3.9
<input type="checkbox"/>	delete-function	-	Zip	Python 3.9
<input type="checkbox"/>	search-function	-	Zip	Python 3.9
<input type="checkbox"/>	createUpdate-function	-	Zip	Python 3.9
<input type="checkbox"/>	dictate-function	-	Zip	Python 3.9

7. API Gateway를 사용한 솔루션 개발

API Gateway 및 리소스 생성

- AWS Management Console에서 API Gateway 서비스를 선택
- Create API를 선택
- Choose an API type의 REST API 카드에서 Build를 선택
 - Choose a protocol: REST Create new API: New API API name: PollyNotesAPI
- Create API를 선택
- Actions ▼를 선택하고 Create Resource를 선택한 후 다음 값을 설정
 - Resource Name: notes
 - Resource path*: notes
 - Enable API Gateway CORS: ☒ (확인란 선택)
- Create Resource를 선택

GET 메서드 구성

- . /notes 리소스를 선택하고 Actions ▼를 선택한 다음, Create Method를 선택
 - /notes 리소스 다음에 표시되는 상자에서 GET을 선택하고 (확인 표시)를 선택
 - Lambda Function에서 list-function을 입력하거나 선택
- 나머지 모든 값을 기본값으로 유지하고 Save를 선택
- Add Permission to Lambda Function 모달에서 OK를 선택
- . /notes - GET - Method Execution 패널에서 TEST를 선택
 - Test를 선택
- Lambda 함수가 성공하고 200의 Status 값을 반환
Response Body가 DynamoDB 테이블의 모든 항목이 포함된 JSON 어레이를 반환

GET 메서드 테스트 결과

The screenshot displays the AWS Management Console interface for the Amazon API Gateway service. The top navigation bar includes the AWS logo, a search bar, and the user's profile (iam_danny @ 4259-2074-1281). The left sidebar shows the navigation menu with categories like APIs, Custom Domain Names, VPC Links, and API: PollyNotesAPI. The main content area is titled "PollyNotesAPI (6890l8t38b) > Resources > /notes (h1yev0) > GET". The "Resources" tab is active, showing a tree view with "/notes" expanded and "GET" selected. The "Method Execution" tab is also visible, showing the test results for the GET method. The test results include the Path, Query Strings, Headers, Request, Status, Latency, and Response Body. The Response Body is a JSON array of note objects.

APIs

- Custom Domain Names
- VPC Links
- API: **PollyNotesAPI**
 - Resources**
 - Stages
 - Authorizers
 - Gateway Responses
 - Models
 - Resource Policy
 - Documentation
 - Dashboard

Resources

- /notes
 - GET
 - OPTIONS

Method Execution /notes - GET - Method Test

Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Path

No path parameters exist for this resource. You can define path parameters by using the syntax `{myPathParam}` in a resource path.

Query Strings

`{notes}`

param1=value1¶m2=value2

Headers

`{notes}`

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.

Request: /notes

Status: 200

Latency: 750 ms

Response Body

```
[{"Note": "hello", "UserId": "testuser", "NoteId": 1}, {"Note": "this is my first note", "UserId": "testuser", "NoteId": 2}, {"Note": "DynamoDB is NoSQL", "UserId": "student", "NoteId": 1}, {"Note": "A DynamoDB table is schemaless", "UserId": "student", "NoteId": 2}, {"Note": " PartiQL is a SQL compatible language for DynamoDB", "UserId": "student", "NoteId": 3}, {"Note": "I love DyDB", "UserId": "student", "NoteId": 4}, {"Note": "The maximum item size in DynamoDB is 400 KB", "UserId": "student", "NoteId": 5}, {"Note": "Is_Incomplete": "No", "UserId": "student", "NoteId": 5}, {"Note": "Free swag code: 1234", "UserId": "newbie", "NoteId": 1}, {"Note": "I love DynamoDB", "UserId": "newbie", "NoteId": 2}]
```


8. 캡스톤 애플리케이션 구축 완료

9. AWS X-Ray를 사용하여 애플리케이션 관찰

Thank you