

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)', 'API Gateway', 'CloudFormation', 'CloudWatch Logs', 'ECR', 'ECS', 'IoT', 'Lambda', 'S3', 'Schemas', 'Step Functions', 'Systems Manager', and 'Resources'. The 'DEVELOPER TOOLS' section includes 'CDK' and 'CodeWhisperer (Preview)'. The main editor window shows a Python file named 'createTable.py' with the following code:

```
Lab3 > environment > labRepo > createTable.py > waitForTableCreation
waiter = ddbClient.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 command prompt output:

```
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...

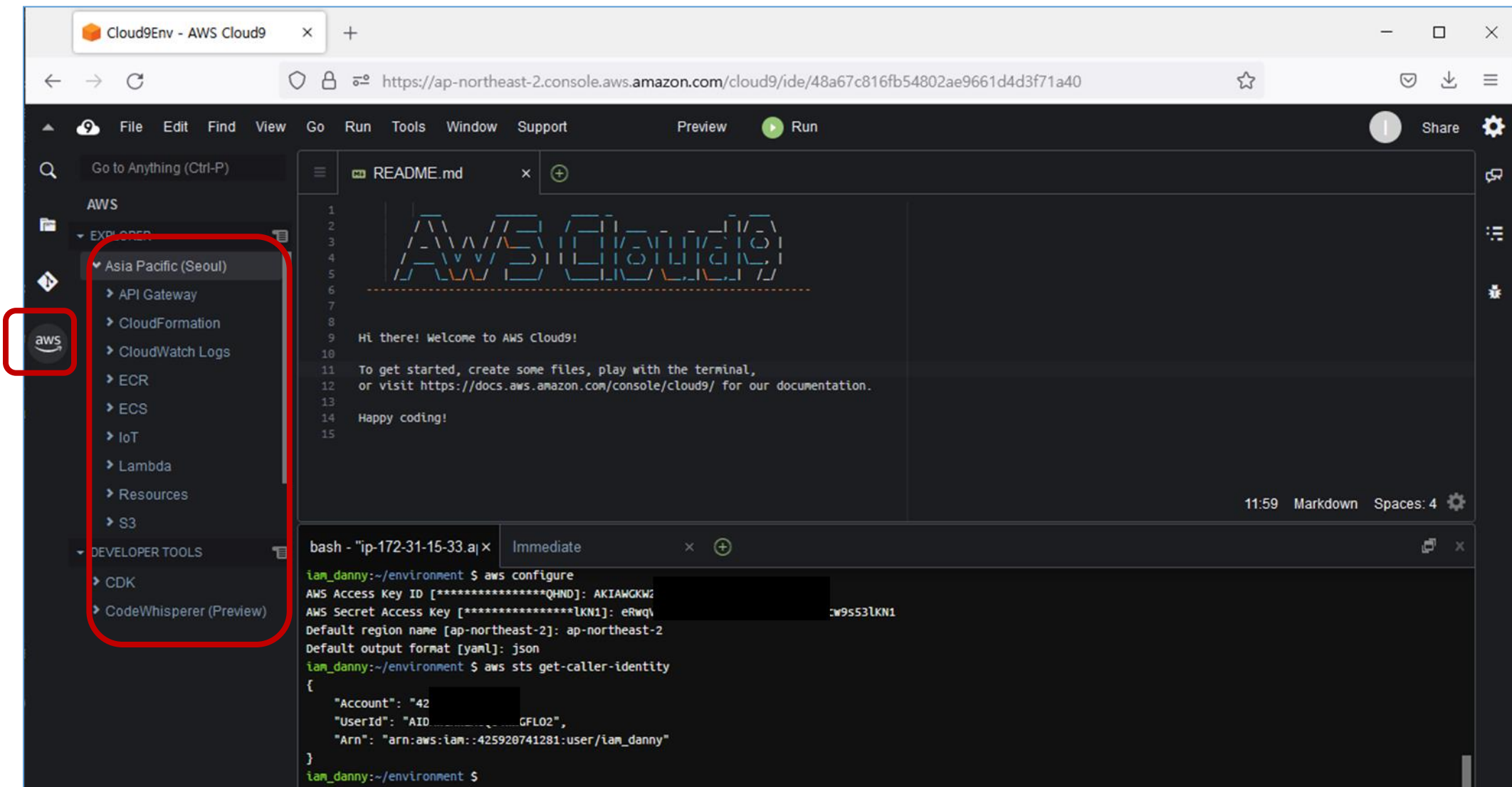
{"Note": "DynamoDB is NoSQL", "NoteId": "1"}
{"Note": "A DynamoDB table is schemaless", "NoteId": "2"}
{"Note": " PartiQL is a SQL compatible language for DynamoDB", "NoteId": "3"}
{"Note": "I love DyDB", "NoteId": "4"}
{"Note": "The maximum item size in DynamoDB is 400 KB", "NoteId": "5"}

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 editor is using Python 3.10.5 64-bit with Prettier formatting.

클라우드 환경 : AWS Cloud9

AWS Explorer를 사용하여 다양한 AWS 서비스의 리소스를 확인하고, 생성하고, 업데이트하고, 삭제

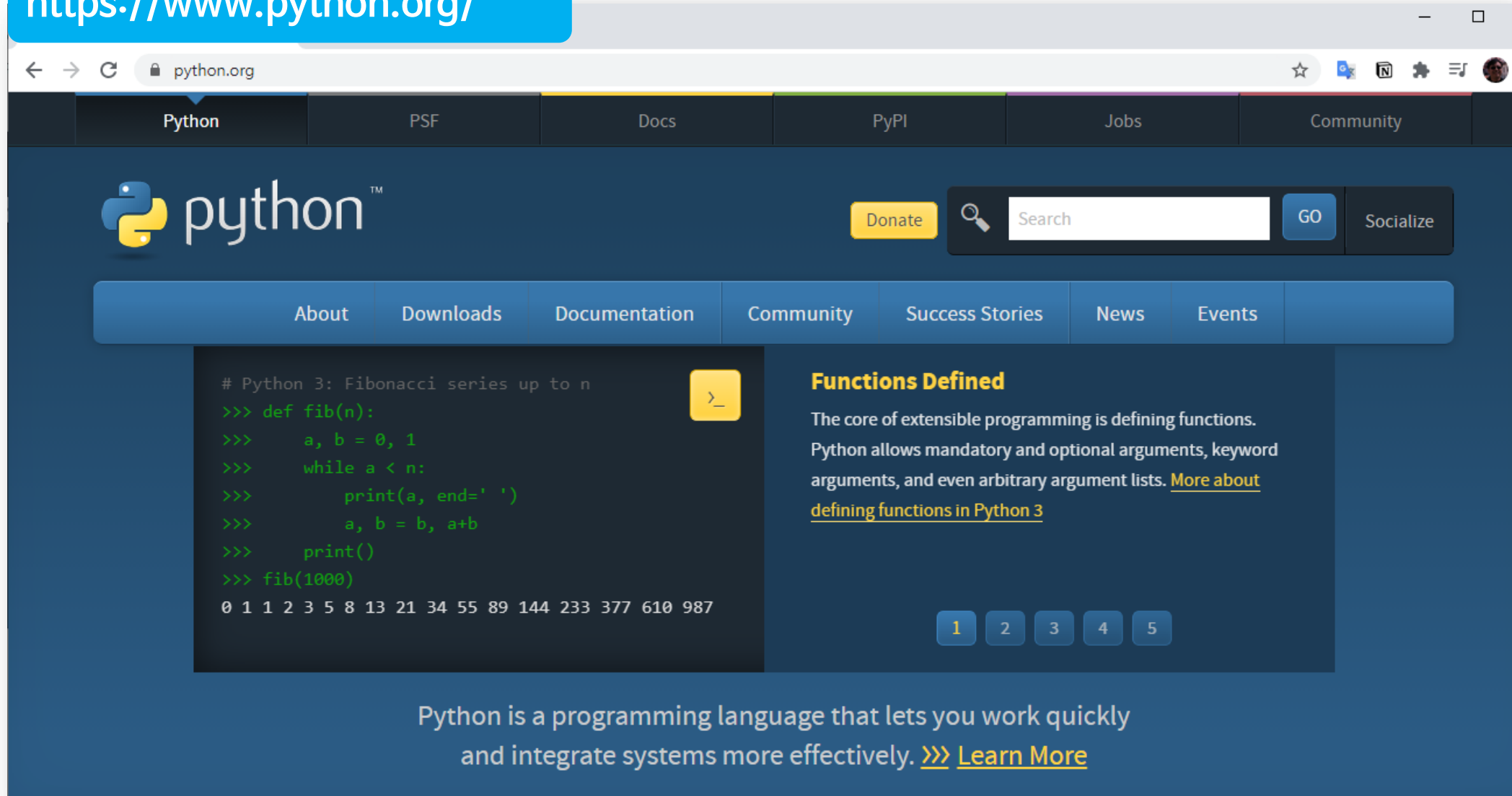


2. 로컬 개발 환경

Python 설치
VS Code 설치
Git 설치

Python 사이트

https://www.python.org/



The screenshot shows the Python.org homepage with a dark blue header and navigation bar. The main content area features a code editor on the left, a text block on the right, and a footer with a promotional message.

Navigation Bar: Python, PSF, Docs, PyPI, Jobs, Community

Header: python™, Donate, Search, GO, Socialize

Secondary Navigation: About, Downloads, Documentation, Community, Success Stories, News, Events

Code Editor:

```
# 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](#)

Footer: 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

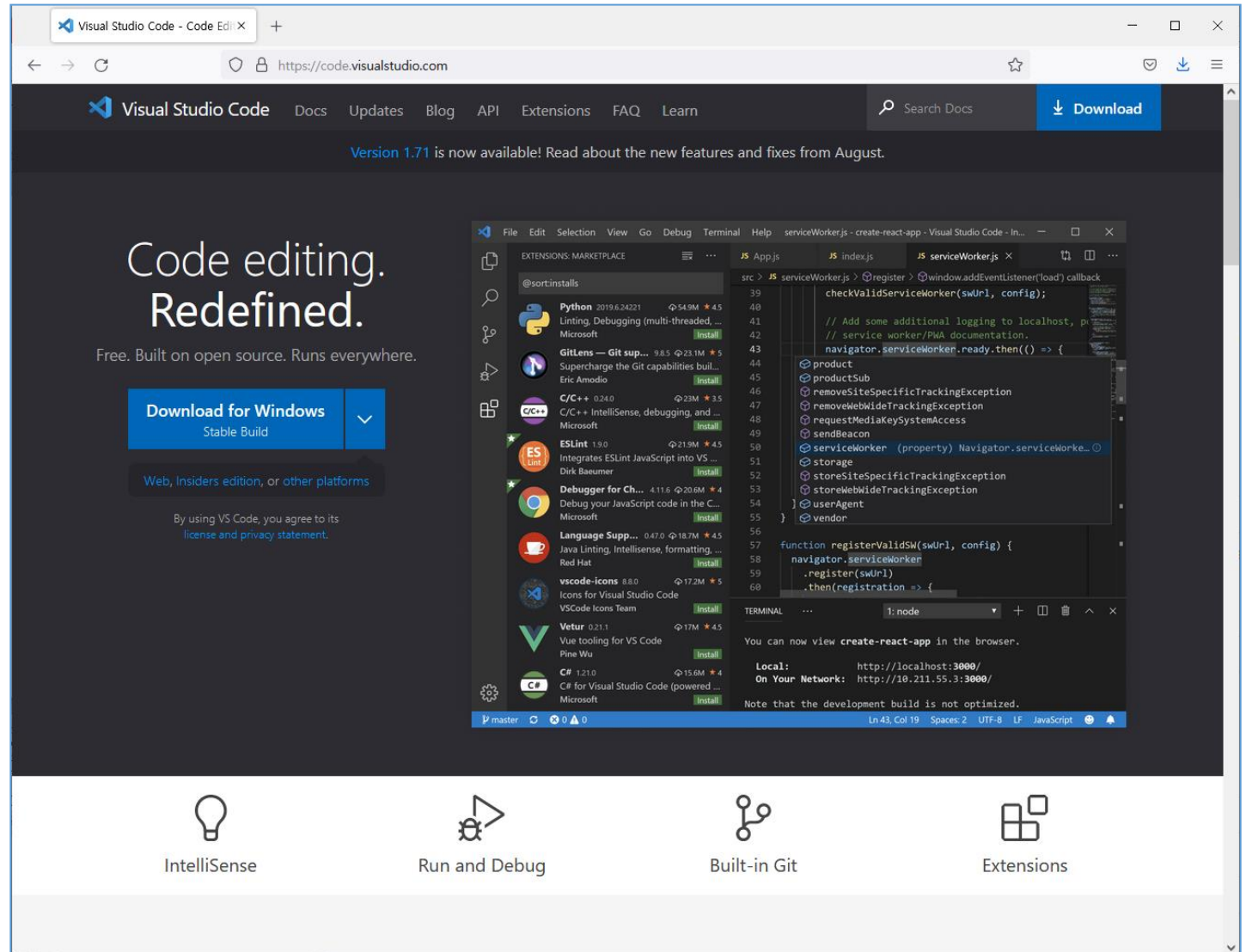
가상 환경(virtual environment)으로 프로젝트별로 독립된 파이썬 실행 환경을 사용할 수 있습니다.

- 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/>

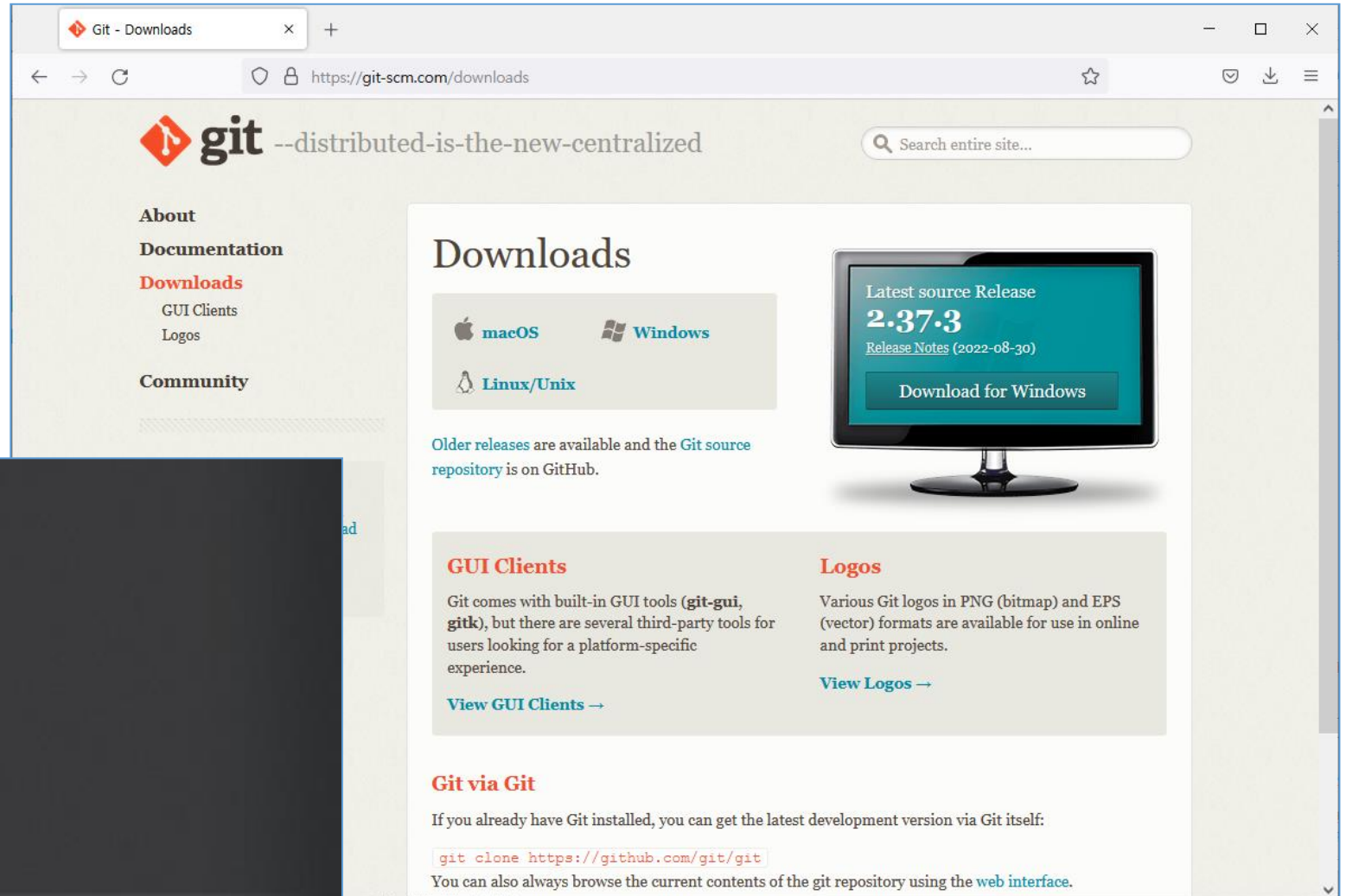
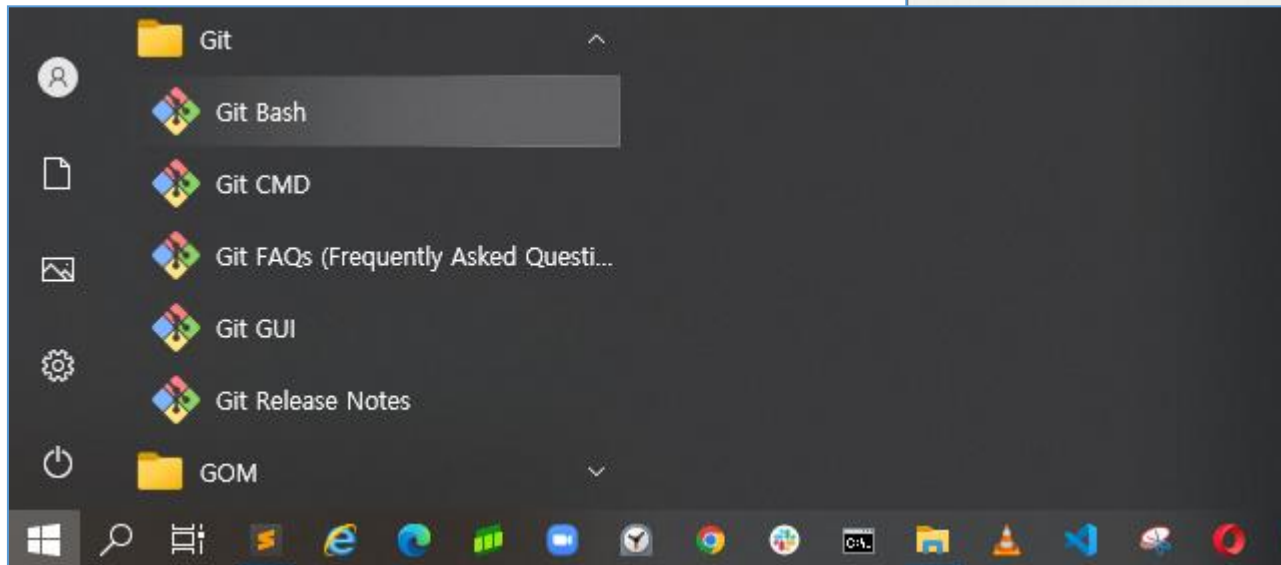


Git 설치

■ 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 top navigation bar includes the AWS logo, service links, a search bar, and the user profile 'iam_danny @ 4259-2074-1281'. The left sidebar shows the 'Identity and Access Management(IAM)' menu with options like 'Dashboard', 'Access Management', 'Users', 'Groups', 'Roles', 'Policies', 'Account Settings', 'Reports & Access', 'Access Analyzer', 'SCP', and 'IAM Search'. The main content area is titled '내 보안 자격 증명' (My Security Credentials) and shows account details for 'iam_danny'. Below this, there are tabs for 'AWS IAM 자격 증명', 'AWS CodeCommit 자격 증명', and 'Amazon MCS 자격 증명'. The 'AWS IAM 자격 증명' tab is active, showing a section for '콘솔 액세스를 위한 암호' (Console Access Keys) and a section for 'CLI, SDK 및 API 액세스를 위한 액세스 키' (Access Keys for CLI, SDK, and API). The 'Access Keys for CLI, SDK, and API' section contains a table of access keys. A red box highlights the '액세스 키 만들기' (Create New Access Key) button. Another red box highlights the user profile in the top right corner.

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

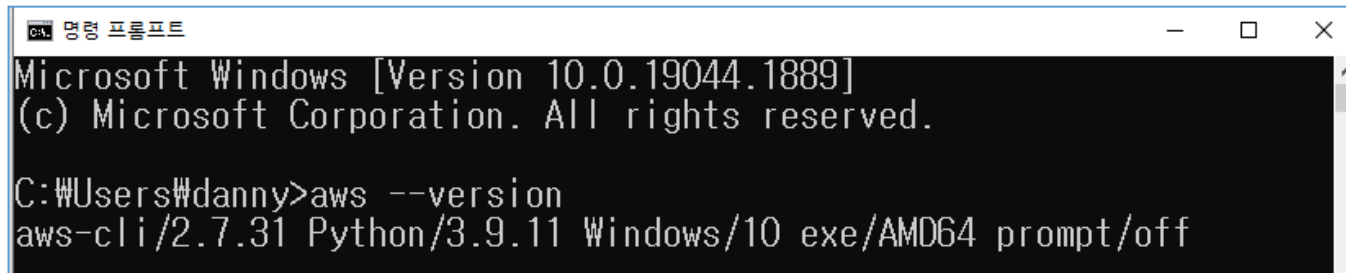
Access Keys for CLI, SDK, and API:

액세스 키 ID	상태	생성 완료	마지막 사용	작업
AKIAWGKW2X6Q4Q4NQHND	(활성)	2022-09-08 00:00 UTC+0900	2022-09-09 00:17 UTC+0900	비활성화 삭제

AWS CLI 설치

■ AWS CLI 설치

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



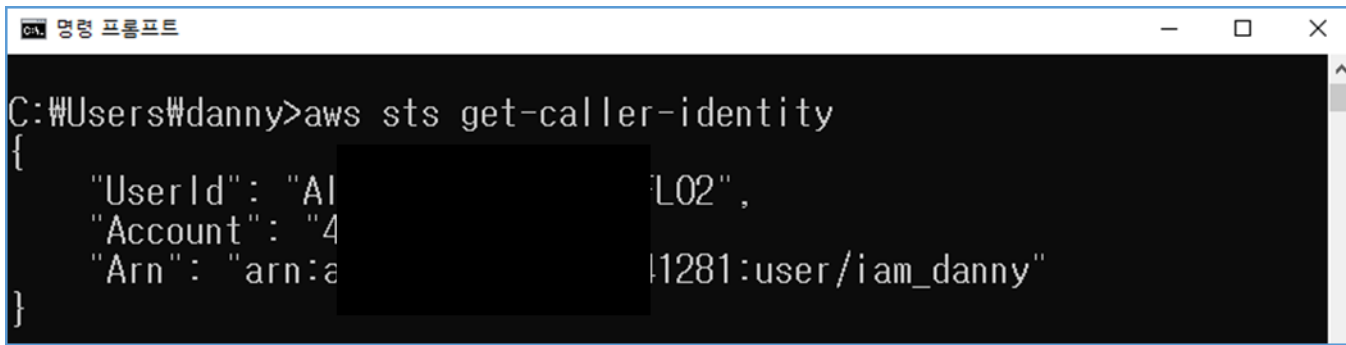
```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

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



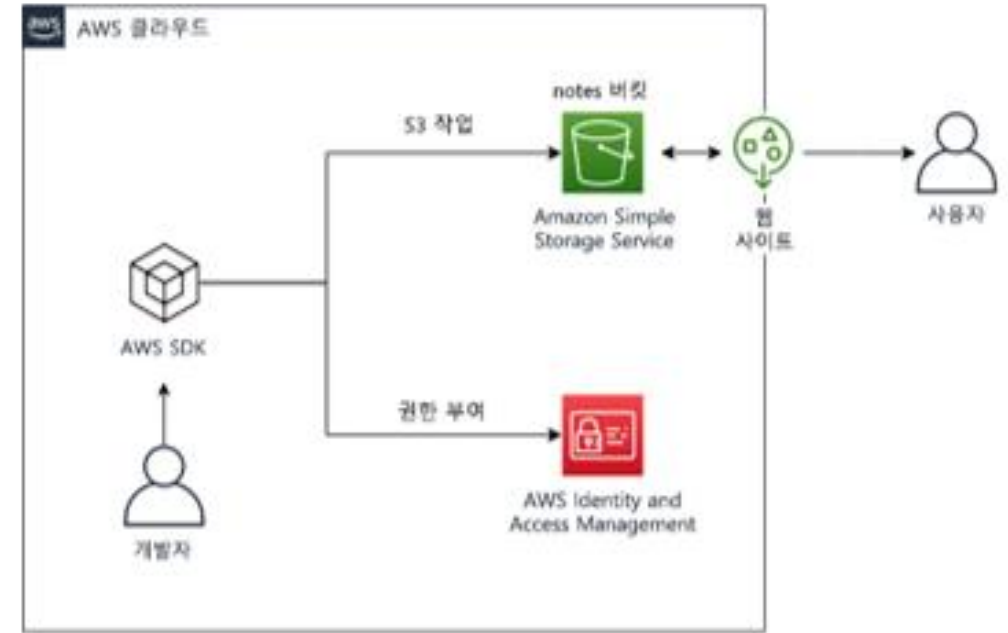
```
C:\Users\danny>aws sts get-caller-identity
{
  "UserId": "A1XXXXXXXXXXXXL02",
  "Account": "411111111111",
  "Arn": "arn:aws:iam::411111111111:user/iam_danny"
}
```


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

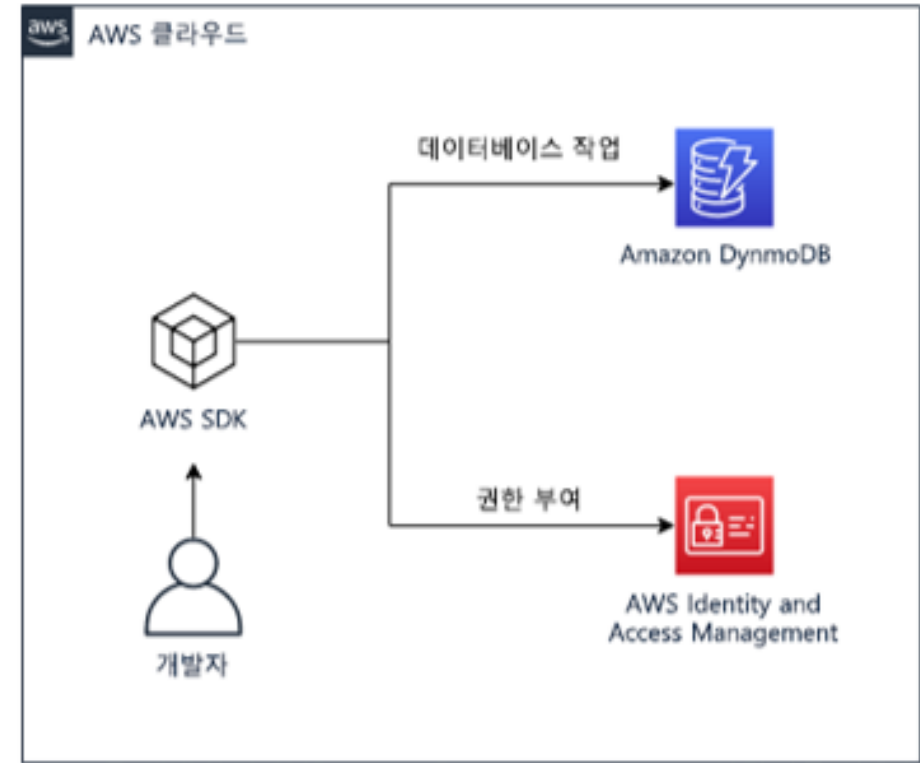
DynamoDB

■ 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 updateNewAttribute(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...

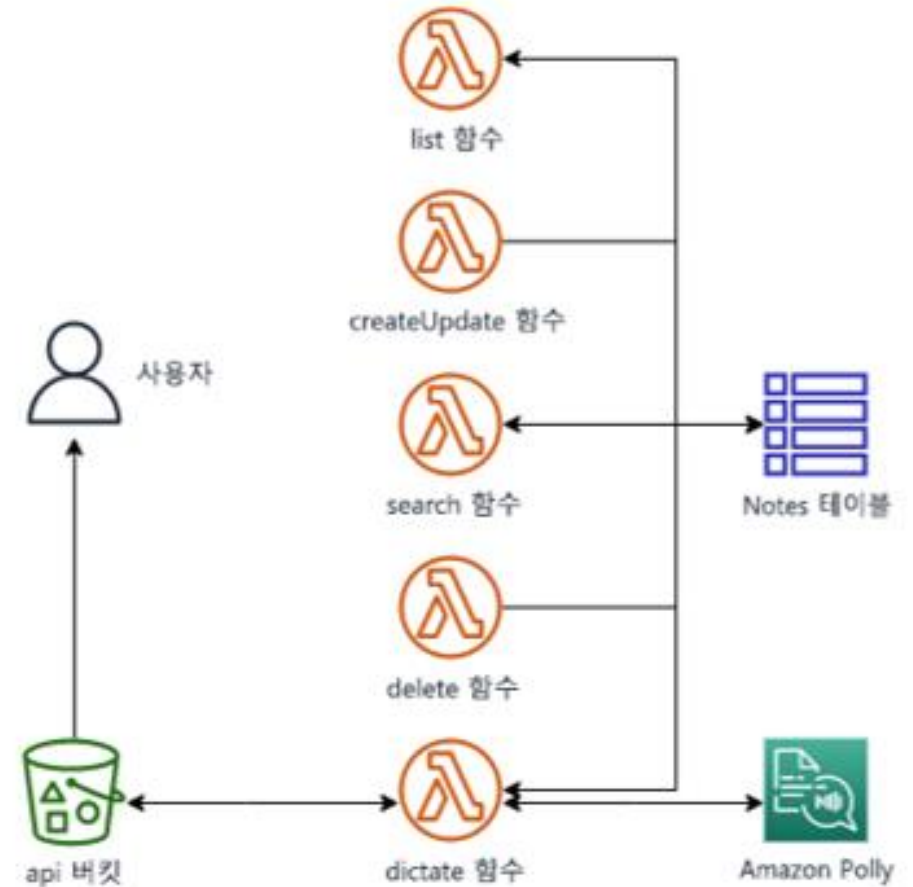
```
{"Note": "The maximum item size in DynamoDB is 400 KB", "UserId": "student", "Noteld": "5", "Is_Incomplete": "No"}
```

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`

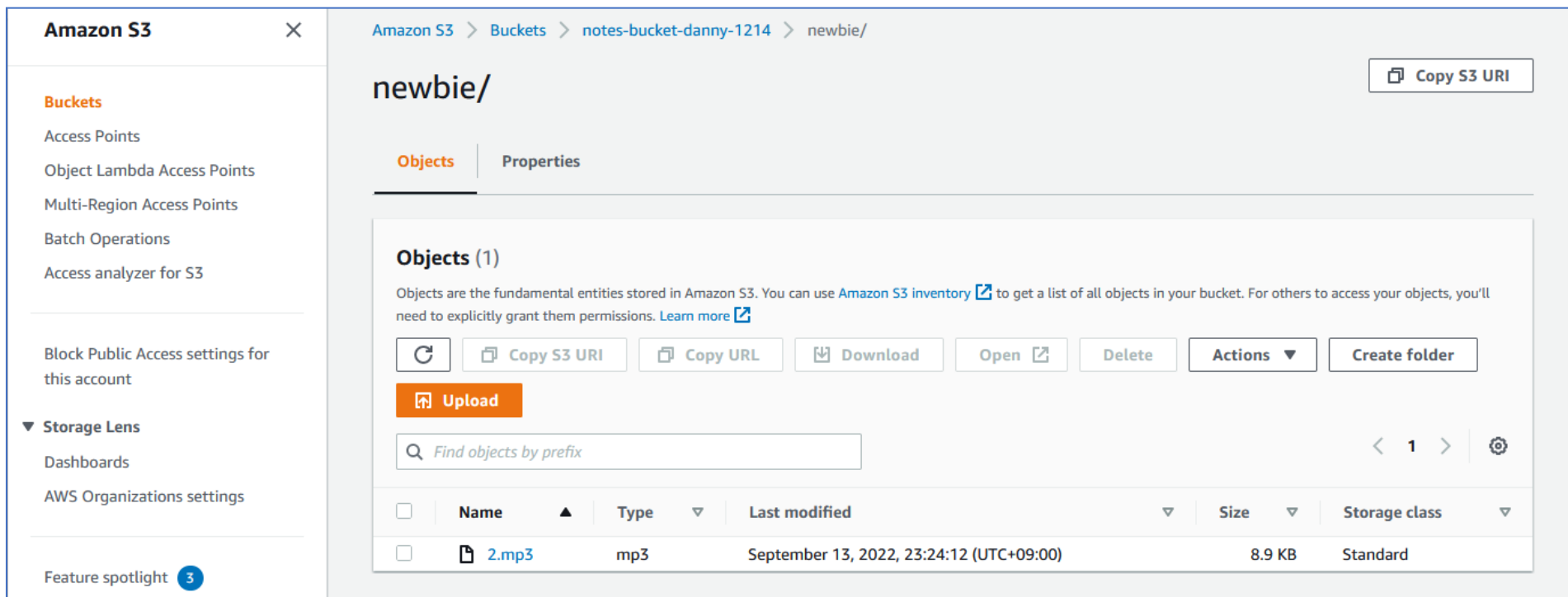
- o 함수 핸들러 업데이트

`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

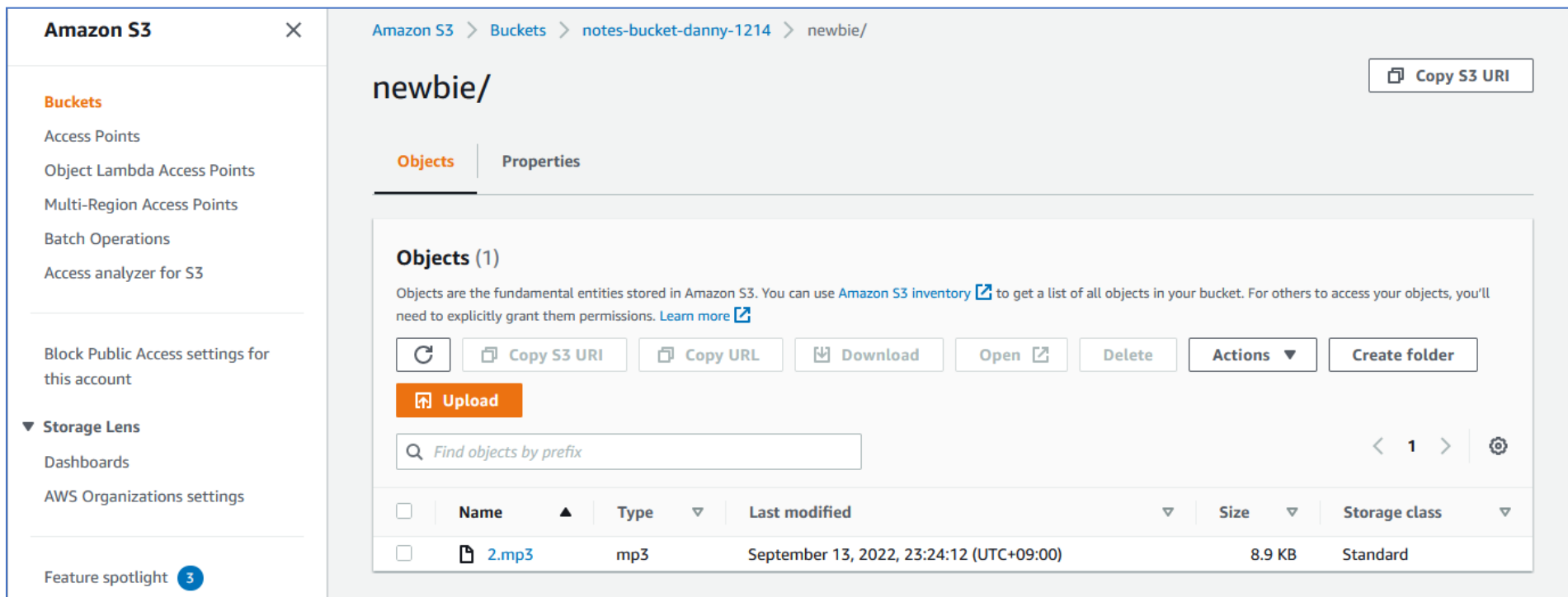
```
{ "UserId": "newbie", "NotelId": "2", "VoicelId": "Joey" }
```
- Lambda 함수 호출
`aws lambda invoke --function-name dictate-function --payload fileb://event.json response.txt`
- 결과 확인



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



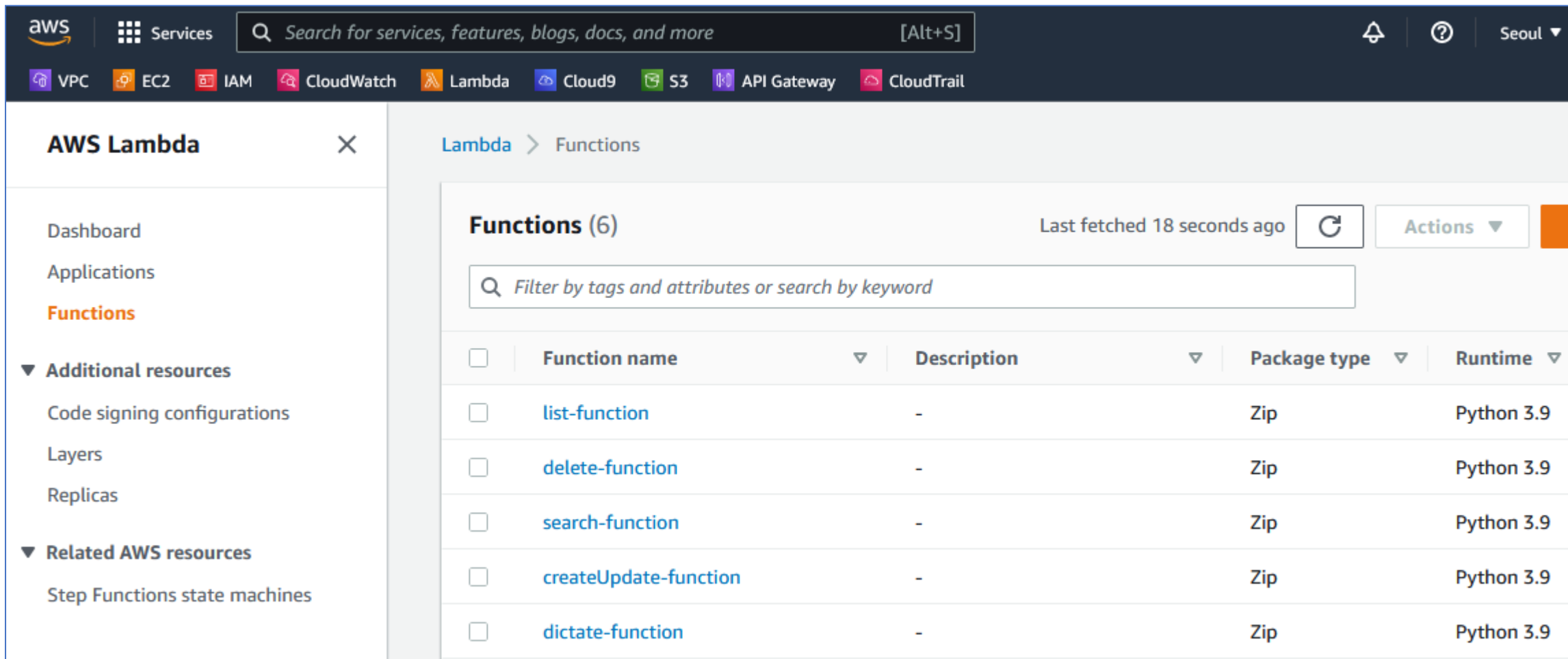
추가 Lambda 함수 생성

- 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 displays 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').

<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

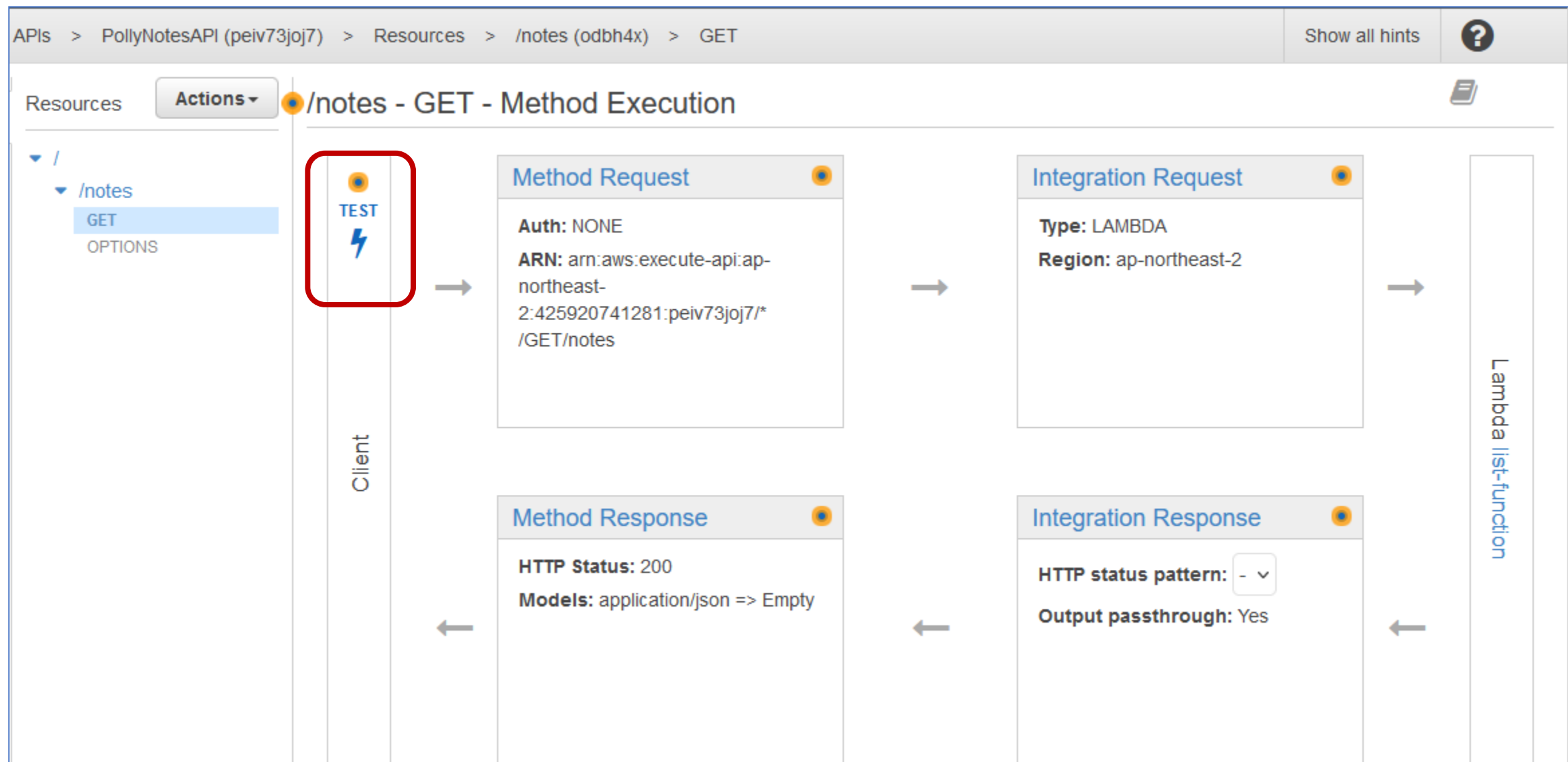
REST API 및 리소스 생성

- 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



GET 메서드 - 테스트 결과

The screenshot displays the AWS Management Console interface for the Amazon API Gateway service. The breadcrumb navigation at the top indicates the path: APIs > PollyNotesAPI (6890l8t38b) > Resources > /notes (h1yev0) > GET. The left-hand navigation pane shows the 'Resources' section for the 'PollyNotesAPI' selected. The main content area is titled '/notes - GET - Method Test' and includes a description: 'Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method'. Below this, there are sections for 'Path', 'Query Strings', and 'Headers'. The 'Response Body' section, which is highlighted with a red border, displays the following JSON array:

```
[{"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": "newbie", "NoteId": 1}, {"Note": "Free swag code: 1234", "UserId": "newbie", "NoteId": 1}, {"Note": "I love DynamoDB", "UserId": "newbie", "NoteId": 2}]
```

GET 메서드 – UserId 전달 구성

oj7) > Resources > /notes (odbh4x) > GET

← Method Execution **/notes - GET - Integration Request**

Provide information about the target backend that this method will call and whether the incoming request data

▼ Mapping Templates

Request body passthrough

- ☐ When no template matches the request Content-Type header ⓘ
- ☒ When there are no templates defined (recommended) ⓘ
- ☐ Never ⓘ

Content-Type

application/json

+ Add mapping template

application/json

Generate template:

1 {"UserId": "student"}

Request: /notes

Status: 200

Latency: 850 ms

Response Body

```
[{"Note": "DynamoDB is NoSQL", "UserId": "student", "Note Id": 1}, {"Note": "A DynamoDB table is schemaless", "User Id": "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, "Is_Incomplete": "No"}]
```

■ list-function/app.py

```
27 def getDatabaseItems(dynamoDBResource, ddbTable, event):
28     print("getDatabaseItems Function")
29
30     # Create our DynamoDB table resource
31     table = dynamoDBResource.Table(ddbTable)
32
33     # If a userId was passed, query the table for that user's items
34     if "UserId" in event:
35         UserId = event['UserId']
36         records = table.query(KeyConditionExpression=Key("UserId").eq(UserId))
37     else:
38         # if not, scan the table and return all items
39         records = table.scan()
40
41     return records["Items"]
```

Response 데이터 제한

Integration Response

HTTP status pattern: - v

Output passthrough: Yes



Method Execution /notes - GET - Integration Response

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

Lambda Error Regex	Method response status	Output model
-	200	

Map the output from your Lambda function to the headers and output model of the 200 method response.

Lambda Error Regex

default

Content handling

Passthrough

Header Mappings

Mapping Templates

Content-Type

application/json

application/json

Add mapping template

1

2

3

4

5

6

7

8

#set(\$inputRoot = \$input.path('\$')) [

#foreach(\$elem in \$inputRoot) {

"NotId" : "\$elem.NotId",

"Note" : "\$elem.Note"

}

#if(\$foreach.hasNext),#end

#end

]

Response 데이터 (응답 크기 축소로 성능 향상 및 데이터 전송 비용 절감)

Request: /notes

Status: 200

Latency: 805 ms

Response Body

```
[
  {
    "NoteId" : "1",
    "Note" : "DynamoDB is NoSQL"
  },
  {
    "NoteId" : "2",
    "Note" : "A DynamoDB table is schemaless"
  }
]
```

```
#set($inputRoot = $input.path('$')) [
  #foreach($elem in $inputRoot) {
    "NotId" : "$elem.NotId",
    "Note" : "$elem.Note"
  }
  #if($foreach.hasNext),#end
#end
]
```

POST 메서드 - 구성

- /notes 리소스를 선택하고 Actions ▼를 선택한 다음, Create Method를 선택
 - /notes 리소스 다음에 표시되는 상자에서 POST를 선택하고 (확인 표시)를 선택
 - **Lambda Function에서 createUpdate-function을 입력하거나 선택**
- 나머지 모든 값을 기본값으로 유지하고 Save를 선택
- Add Permission to Lambda Function 모달에서 OK를 선택
- ./notes - POST - Method Execution 패널에서 TEST를 선택
- 복사/붙여넣기: Request Body 코드 블록에 다음 JSON 객체를 복사

```
{  
  "Note": "This is your new note added using the POST method",  
  "NoteId": 3,  
  "UserId": "student"  
}
```
- Test를 선택
- GET 메서드를 사용하여 새 노트가 추가되었는지 확인

Request Body

```
1 {  
2   "Note": "This is your new note  
   added using the POST method",  
3   "NoteId": 3,  
4   "UserId": "student"  
5 }
```

POST 메서드 - Request Body 스키마 적용

API Gateway 모델을 사용하여 Request Body 데이터를 확인할 수 있습니다.

- API Gateway 탐색 패널에서 API: PollyNotesAPI 아래에서 Models를 선택
- Create를 선택 : Model name에 NoteModel 입력, Content type에 application/json 입력
- Model schema에 다음 내용 입력

```
{  
  "title": "Note",  
  "type": "object",  
  "properties": {  
    "UserId": {"type": "string"},  
    "NoteId": {"type": "integer"},  
    "Note": {"type": "string"}  
  },  
  "required": ["UserId", "NoteId", "Note"]  
}
```

- 왼쪽 탐색 패널의 API: PollyNotesAPI 아래에서 Resources를 선택
- Resources 패널에서 /notes 리소스에 대해 POST를 선택
- . /notes - POST - Method Execution 패널에서 Method Request를 선택
 - Request Validator에서 (편집)을 선택
 - Validate body를 선택하고 (확인 표시)를 선택
 - ► Request Body를 확장
 - ⊕ Add model을 선택
 - Content type에 application/json을 입력
 - Model name에서 NoteModel을 선택

POST 메서드 - Request Body 스키마 적용

Amazon API Gateway

APIs > PollyNotesAPI (peiv73joj7) > Models > Create

APIs

Custom Domain Names

VPC Links

API: **PollyNotesAPI**

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Models

Create

Empty

Error

New Model

Provide a name, content type, and a schema for your model. Models use [JSON schema](#).

Model name*

Content type*

Model description

Model schema*

```
1 {  
2   "title": "Note",  
3   "type": "object",  
4   "properties": {  
5     "UserId": {"type": "string"},  
6     "NoteId": {"type": "integer"},  
7     "Note": {"type": "string"},  
8   },  
9   "required": ["UserId", "NoteId", "Note"],  
10 }  
11
```

```
{  
  "title": "Note",  
  "type": "object",  
  "properties": {  
    "UserId": {"type": "string"},  
    "NoteId": {"type": "integer"},  
    "Note": {"type": "string"}  
  },  
  "required": ["UserId", "NoteId", "Note"]  
}
```

POST 메서드 - Request Body 스키마 적용

The screenshot displays the Amazon API Gateway console interface. The breadcrumb navigation at the top indicates the path: APIs > PollyNotesAPI (peiv73joj7) > Resources > /notes (odbh4x) > POST. The left-hand navigation pane shows the 'API: PollyNotesAPI' and 'Resources' section highlighted with a red box. The main content area is titled '/notes - POST - Method Request' and includes a 'Settings' section. Within the settings, 'Authorization' is set to 'NONE', 'Request Validator' is set to 'Validate body' (highlighted with a red box), and 'API Key Required' is set to 'false'. Below these, there are expandable sections for 'URL Query String Parameters' and 'HTTP Request Headers'. The 'Request Body' section is expanded and highlighted with a red box, showing a table with two columns: 'Content type' and 'Model name'. The 'Content type' is set to 'application/json' and the 'Model name' is set to 'NoteModel'.

Amazon API Gateway

APIs > PollyNotesAPI (peiv73joj7) > Resources > /notes (odbh4x) > POST

Show all hints ?

APIs

Custom Domain Names

VPC Links

API: PollyNotesAPI

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Resources

Actions

Method Execution /notes - POST - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Settings

Authorization NONE

Request Validator Validate body

API Key Required false

URL Query String Parameters

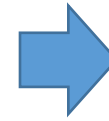
HTTP Request Headers

Request Body

Content type	Model name
application/json	NoteModel

POST 메서드 - 테스트

```
{  
  "Note": "This is your updated note using the Model validation",  
  "UserId": "student",  
  "id": 3  
}
```



Request: /notes
Status: 400
Latency: 62 ms
Response Body

```
{"message": "Invalid request body"}
```

```
{  
  "Note": "This is your updated note using the Model validation",  
  "UserId": "student",  
  "Notelid": 3  
}
```



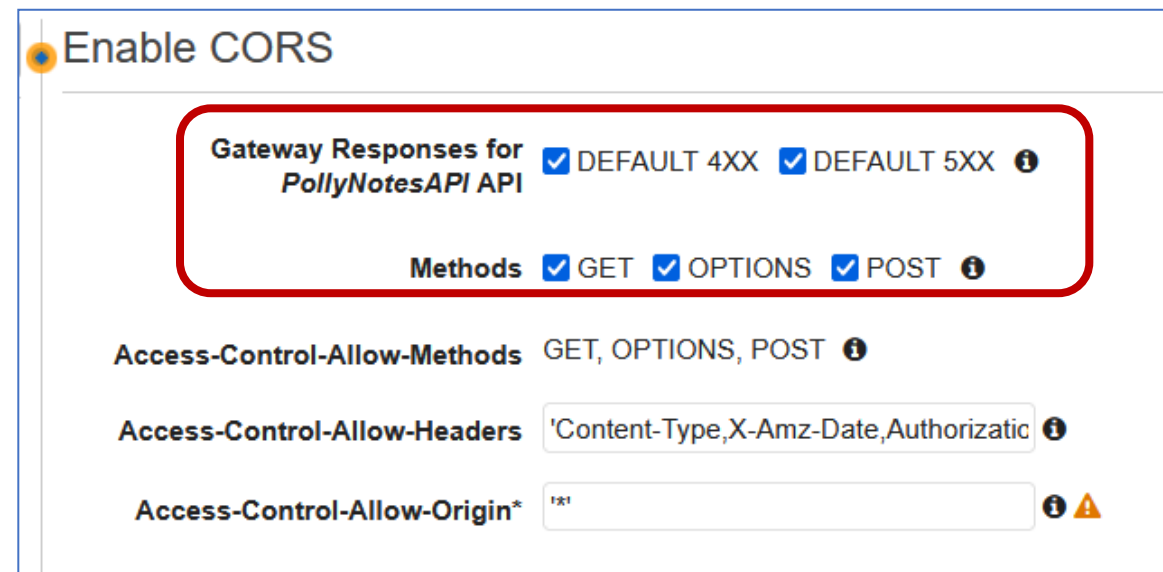
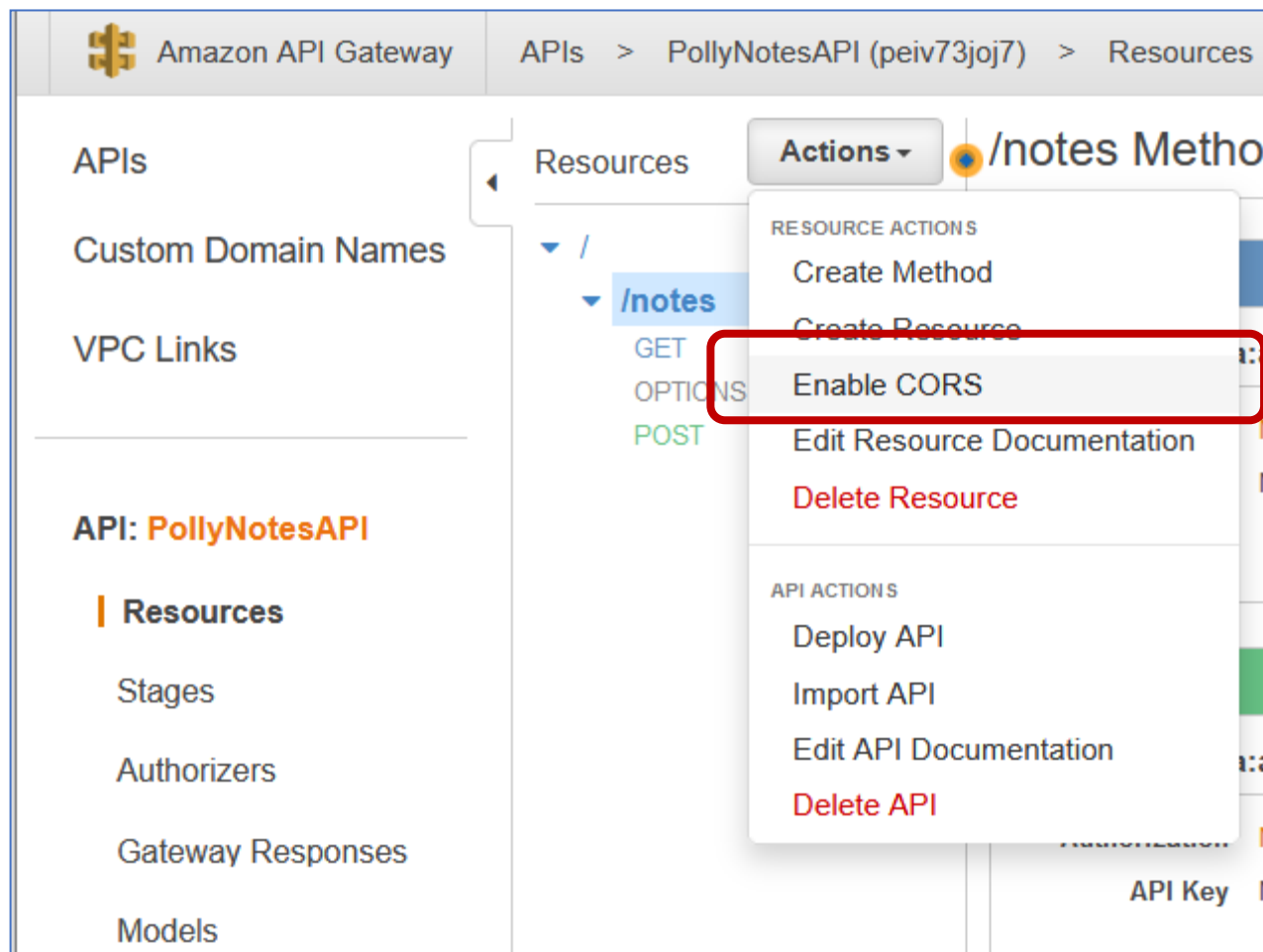
Request: /notes
Status: 200
Latency: 326 ms
Response Body

```
3
```

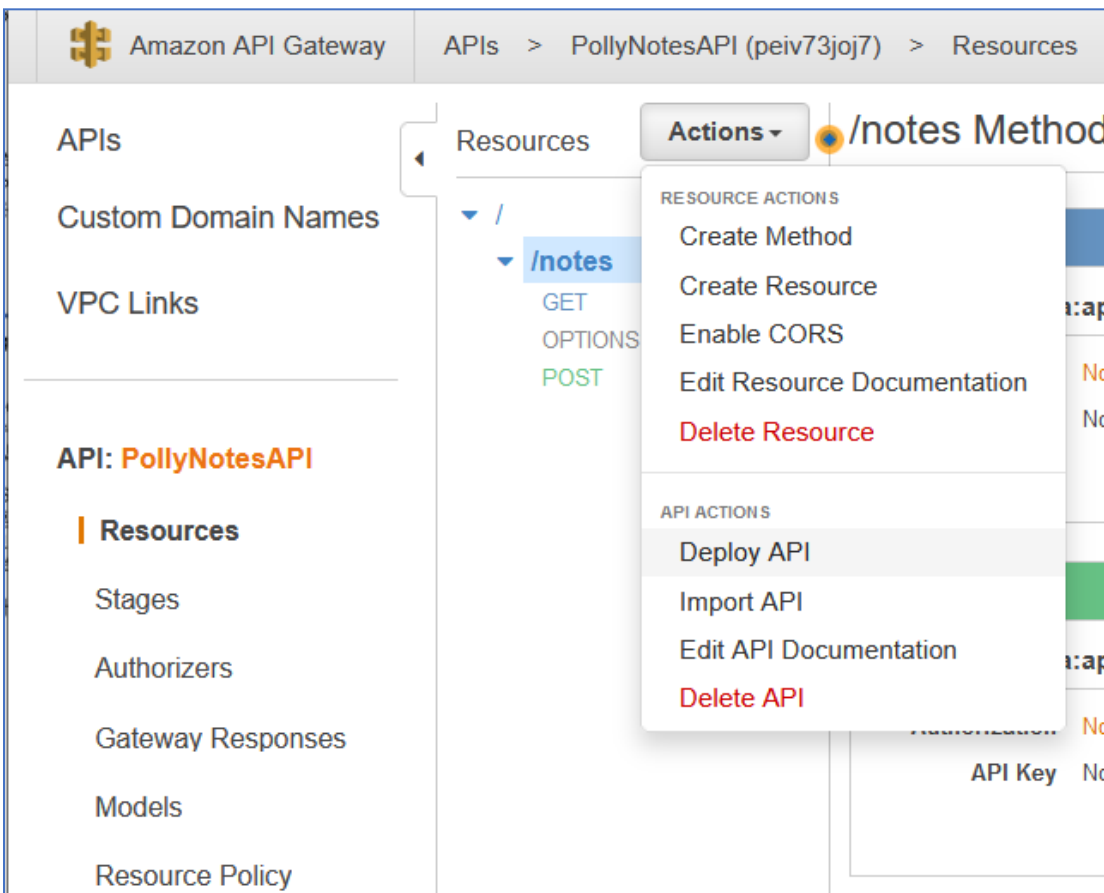

REST API 리소스에 대한 CORS 활성화

CORS(Cross-Origin Resource Sharing)는 웹브라우저에서 실행 중인 자바스크립트에서 시작되는 cross-origin HTTP 요청을 제한하는 브라우저 보안 기능입니다.

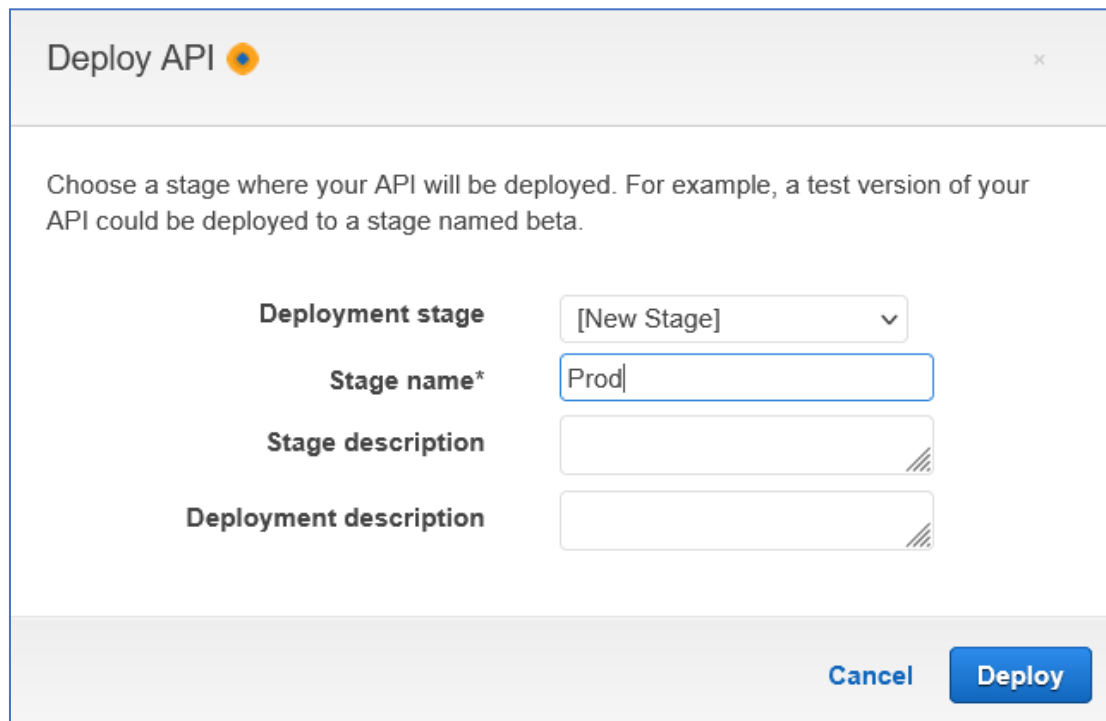
REST API의 리소스가 cross-origin HTTP 요청을 받을 경우 CORS 지원을 활성화해야 합니다.



API 배포



The screenshot shows the Amazon API Gateway console. The breadcrumb navigation is 'APIs > PollyNotesAPI (peiv73joj7) > Resources'. The left sidebar contains links for 'APIs', 'Custom Domain Names', 'VPC Links', 'API: PollyNotesAPI', 'Resources' (selected), 'Stages', 'Authorizers', 'Gateway Responses', 'Models', and 'Resource Policy'. The main content area shows the 'Resources' section for 'PollyNotesAPI'. A resource named '/notes' is selected, and the 'Actions' dropdown menu is open. The 'API ACTIONS' section of the menu is highlighted, and 'Deploy API' is the selected option.




The 'Deploy API' dialog box is shown. It contains the following fields:

- Deployment stage:** A dropdown menu with '[New Stage]' selected.
- Stage name*:** A text input field containing 'Prod'.
- Stage description:** A text input field.
- Deployment description:** A text input field.

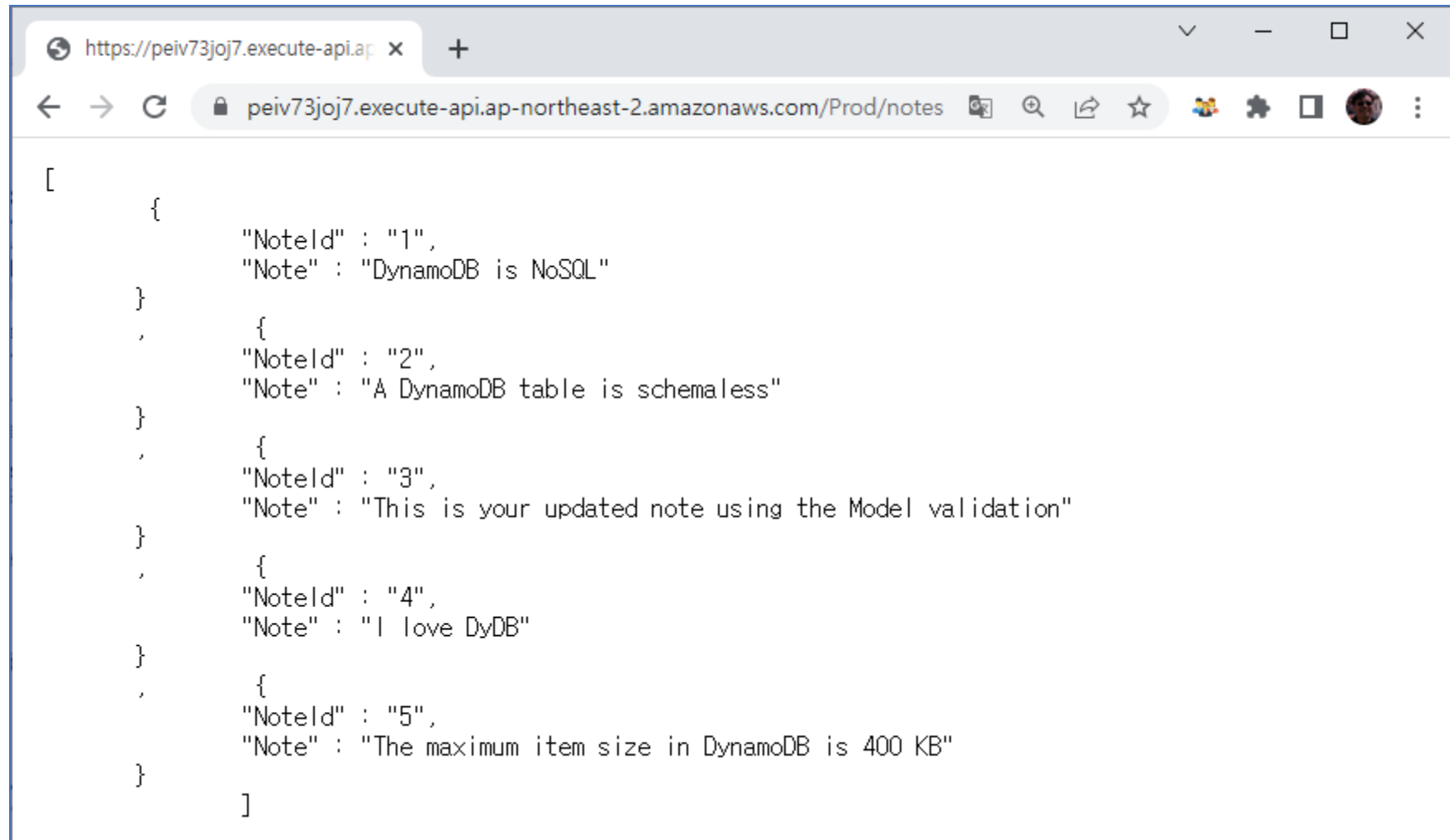
At the bottom right, there are 'Cancel' and 'Deploy' buttons.



 **Invoke URL:** <https://peiv73joj7.execute-api.ap-northeast-2.amazonaws.com/Prod>

API 배포 확인

- 웹브라우저에서 URL 접속 : <https://peiv73joj7.execute-api.ap-northeast-2.amazonaws.com/Prod/notes>

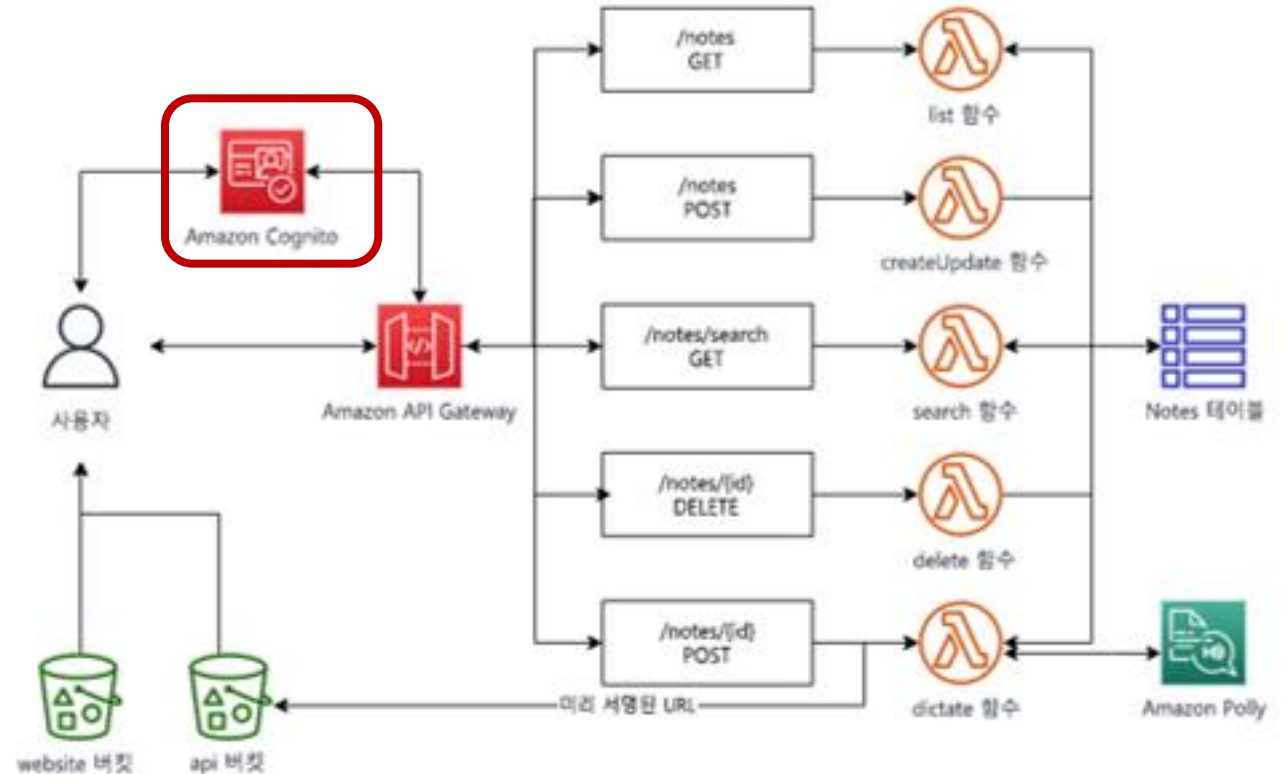


8. 캡스톤 애플리케이션 구축 완료

Cognito

■ 목표

- Cognito를 사용하여 웹 애플리케이션을 위한 사용자 풀과 앱 클라이언트를 생성
- 사용자를 추가하고 Cognito CLI를 사용하여 로그인할 수 있는지 확인
- Cognito를 권한 부여자로 사용하도록 API Gateway 메서드를 구성
- API 호출시 JWT 인증 토큰이 생성되는지 확인
- Swagger 가져오기 전략을 사용하여 API Gateway 리소스를 신속하게 개발
- Cognito 및 API Gateway 구성을 사용하도록 웹 애플리케이션 프론트엔드를 설정하고 전체 애플리케이션 기능을 확인



Cognito - User Pool 생성

사용자 풀은 Amazon Cognito의 사용자 디렉터리입니다.

사용자 풀에서 사용자는 Amazon Cognito를 통해 웹 또는 모바일 앱에 로그인할 수 있습니다.

- AWS Management Console에서 Cognito 서비스를 선택
- 최신 Cognito 인터페이스를 사용 : Try out the new interface 링크에서 새 인터페이스를 사용 선택
- Create a user pool을 선택하고 다음 항목을 선택
 - Authentication providers:
Provider types: ☒ Cognito user pool
Cognito user pool sign-in options: ☒ User name
- Configure security requirements 섹션에서 암호에 필요한 세부 사항을 설정
- Password policy: 화면에서 다음 구성을 선택
 - Password policy mode: Custom
 - Password minimum length: 6 character(s)
 - Password requirements: 모두 선택 해제
- No MFA

Cognito - User Pool 생성

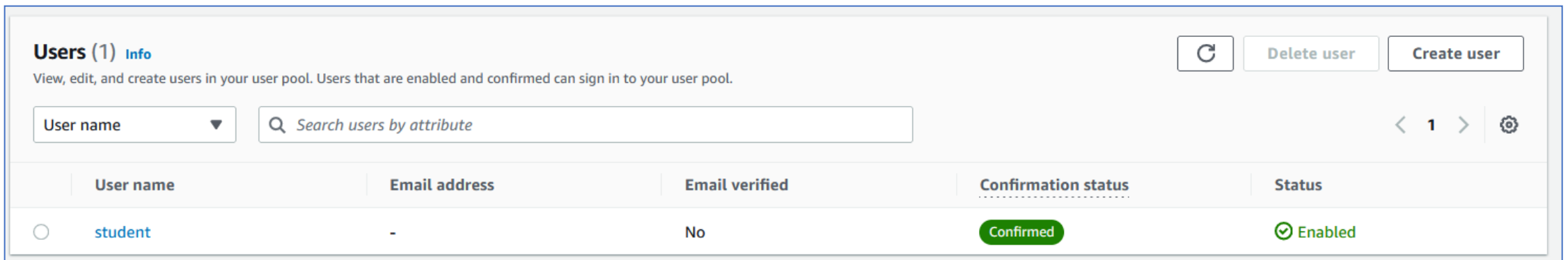
- Enable self-service account recovery - Recommended (선택 해제)
- Allow Cognito to automatically send messages to verify and confirm - Recommended (선택 해제)
- Configure message delivery에 다음 옵션을 설정
 - Email:
Email provider: Send email with Cognito
- Integrate your app에 다음 옵션 설정
 - User pool name:
User pool name: PollyNotesPool
 - Initial app client:
App type: Public client
App client name: PollyNotesWeb
Client secret: Don't generate a client secret

Create user pool

PollyNotesPool 링크를 선택 : Pool Id, Pool ARN, Client ID(App integration 탭)을 별도로 저장

Cognito - User Pool에 사용자 추가(git bash 에서 실행)

- 터미널 창에서 이후 명령에서 사용하게 될 3개의 변수를 생성
 - apiURL='https://peiv73joj7.execute-api.ap-northeast-2.amazonaws.com/Prod'
 - CognitoPoolId='ap-northeast-2_Al8WUSCfS'
 - AppClientId='171ddfva9v38pvct9cl8hp8d4t'
- student 사용자 생성 및 confirm
 - aws cognito-idp sign-up --client-id \$AppClientId --username student --password student
 - aws cognito-idp admin-confirm-sign-up --user-pool-id \$CognitoPoolId --username student
- Cognito 콘솔로 이동
- User pool name 필드 아래 PollyNotesPool 링크를 선택
- Users 탭 아래에 student라는 사용자 Confirmed 확인 상태가 표시되는 것을 확인



The screenshot shows the AWS Cognito console's 'Users' tab for a specific user pool. At the top, there's a header 'Users (1) Info' with a sub-header 'View, edit, and create users in your user pool. Users that are enabled and confirmed can sign in to your user pool.' To the right of the header are buttons for 'Delete user' and 'Create user'. Below the header is a search bar with the placeholder 'Search users by attribute' and a dropdown menu for 'User name'. A pagination bar shows '< 1 >' and a settings gear icon. The main content is a table with the following columns: 'User name', 'Email address', 'Email verified', 'Confirmation status', and 'Status'. There is one row of data for the user 'student', with an email address of '-', 'Email verified' set to 'No', 'Confirmation status' set to 'Confirmed' (indicated by a green pill), and 'Status' set to 'Enabled' (indicated by a green checkmark icon).

User name	Email address	Email verified	Confirmation status	Status
student	-	No	Confirmed	Enabled

Cognito - User Pool에 사용자 추가(git bash 에서 실행)

```
MINGW64:/c/Users/danny

danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ apiURL='https://peiv73joj7.execute-api.ap-northeast-2.amazonaws.com/Prod'
  AppClientId='7rboh4ram9lt0inqbpsq2g98dh'

danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ CognitoPoolId='ap-northeast-2_BJrzADFwa'

danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ AppClientId='7rboh4ram9lt0inqbpsq2g98dh'

danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ aws cognito-idp sign-up --client-id $AppClientId --username student --password
  student
{
  "UserConfirmed": false,
  "UserSub": "7078094b-0018-41c1-a6d6-49d07f552032"
}

danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ aws cognito-idp admin-confirm-sign-up --user-pool-id $CognitoPoolId --username
  student

danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ |
```

Cognito – 사용자 인증 테스트(추후 정리)

- 실습지침에 있는 **testLoginWebsite** 값을 복사하고 새 웹 브라우저 탭에서 해당 URL을 오픈
- 두 입력란에 이전 작업에서 복사해둔 Pool Id와 App Client Id를 입력
- 사용자 이름 및 암호 입력란 모두에 student를 입력
- Login 선택
- 이 창을 열어 둬. 다음 단계 테스트에서 이 토큰을 사용

Cognito를 권한 부여자로 설정

- API Gateway 서비스 선택
- PollyNotesAPI 선택
- 왼쪽 탐색 메뉴에서 Authorizers를 선택

+ Create New Authorizer

- 정보 입력
 - a. Name: PollyNotesPool
 - b. Type: Cognito
 - c. Cognito User Pool: PollyNotesPool
 - d. Token Source: Authorization
 - e. Token Validation: 공백
- Create 선택
- Test 선택
- 이전 실습에서 생성된 Authorization Token을 Authorization (header) 값을 찾아서 붙여넣음
- Test 선택

Swagger 파일로 API 리소스 생성

■ PollyNotesAPI-swagger.yaml

```
1 ---
2 securityDefinitions:
3   PollyNotesPool:
4     type: "apiKey"
5     name: "Authorization"
6     in: "header"
7     x-amazon-apigateway-authtype: "cognito_user_pools"
8     x-amazon-apigateway-authorizer:
9       type: "cognito_user_pools"
10      providerARNs:
11        - "[Cognito_Pool_ARN]"
12 swagger: "2.0"
13 info:
14   title: "PollyNotesAPI"
15   basePath: "/Prod"
16   schemes:
17     - "https"
18 paths:
19   /notes/search:
20     get:
21       consumes:
22         - "application/json"
23       produces:
24         - "application/json"
25       parameters:
26         - name: "text"
27           in: "query"
28           required: true
29           type: "string"
30       responses:
31         "200":
32           description: "200 response"
33           schema:
34             $ref: "#/definitions/Empty"
35           headers:
36             Access-Control-Allow-Origin:
37               type: "string"
38             Access-Control-Allow-Methods:
39               type: "string"
40             Access-Control-Allow-Headers:
41               type: "string"
42       security:
43         - PollyNotesPool: []
44     x-amazon-apigateway-integration:
45       httpMethod: "POST"
46       uri: "arn:aws:apigateway:[AWS_Region]:lambda:path/2015-03-31/functions/arn:aws:lambda:[AWS_Region]:[AWS_AccountId]:function:search-function/i
```

Swagger 파일로 API 리소스 생성

- PollyNotesAPI-swagger.yaml 파일의 표시자 [Cognito_Pool_Arn] [AWS_Region] [AWS_AccountId] 변경을 위한 변수 생성
 - region=\$(curl http://169.254.169.254/latest/meta-data/placement/region -s)
 - 또는 리전명 직접 지정 : region=ap-northeast-2
 - acct=\$(aws sts get-caller-identity --output text --query "Account")
 - poolId=\$(aws cognito-idp list-user-pools --max-results 1 --output text --query "UserPools[].Id")
 - poolArn="arn:aws:cognito-idp:\$region:\$acct:userpool/\$poolId"
- swagger 파일 표시자 업데이트
 - sed -i "s~₩[Cognito_Pool_Arn₩]~\$poolArn~g" PollyNotesAPI-swagger.yaml
 - sed -i "s~₩[AWS_Region₩]~\$region~g" PollyNotesAPI-swagger.yaml
 - sed -i "s~₩[AWS_AccountId₩]~\$acct~g" PollyNotesAPI-swagger.yaml
- apild 변수를 PollyNotesAPI ID로 설정
 - apild=\$(aws apigateway get-rest-apis --query "items[?name == 'PollyNotesAPI'].id" --output text)

Swagger 파일로 API 리소스 생성

- PollyNotesAPI-swagger.yaml 파일의 표시자 [Cognito_Pool_Arn] [AWS_Region] [AWS_AccountId] 변경을 위한 변수 생성
 - region=\$(curl http://169.254.169.254/latest/meta-data/placement/region -s)
 - 또는 리전명 직접 지정 : region=ap-northeast-2
 - acct=\$(aws sts get-caller-identity --output text --query "Account")
 - poolId=\$(aws cognito-idp list-user-pools --max-results 1 --output text --query "UserPools[].Id")
 - poolArn="arn:aws:cognito-idp:\$region:\$acct:userpool/\$poolId"
- swagger 파일 표시자 업데이트
 - sed -i "s~₩[Cognito_Pool_Arn₩]~\$poolArn~g" PollyNotesAPI-swagger.yaml
 - sed -i "s~₩[AWS_Region₩]~\$region~g" PollyNotesAPI-swagger.yaml
 - sed -i "s~₩[AWS_AccountId₩]~\$acct~g" PollyNotesAPI-swagger.yaml
- apild 변수를 PollyNotesAPI ID로 설정
 - apild=\$(aws apigateway get-rest-apis --query "items[?name == 'PollyNotesAPI'].id" --output text)

Swagger 파일로 API 리소스 생성

- 리소스를 API Gateway로 PUT

- `aws apigateway put-rest-api --rest-api-id $apild --mode merge --body 'fileb://PollyNotesAPI-swagger.yaml'`

```
MINGW64:/c/Users/danny
danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ aws apigateway put-rest-api --rest-api-id $apiId --mode merge --body 'fileb://PollyNotesAPI-swagger.yaml'
{
  "id": "peiv73joj7",
  "name": "PollyNotesAPI",
  "createdDate": "2022-09-18T15:40:25+09:00",
  "apiKeySource": "HEADER",
  "endpointConfiguration": {
    "types": [
      "REGIONAL"
    ]
  },
  "tags": {},
  "disableExecuteApiEndpoint": false
}
```

- 새 리소스를 API Gateway로 배포합니다.

- `aws apigateway create-deployment --rest-api-id $apild --stage-name Prod`

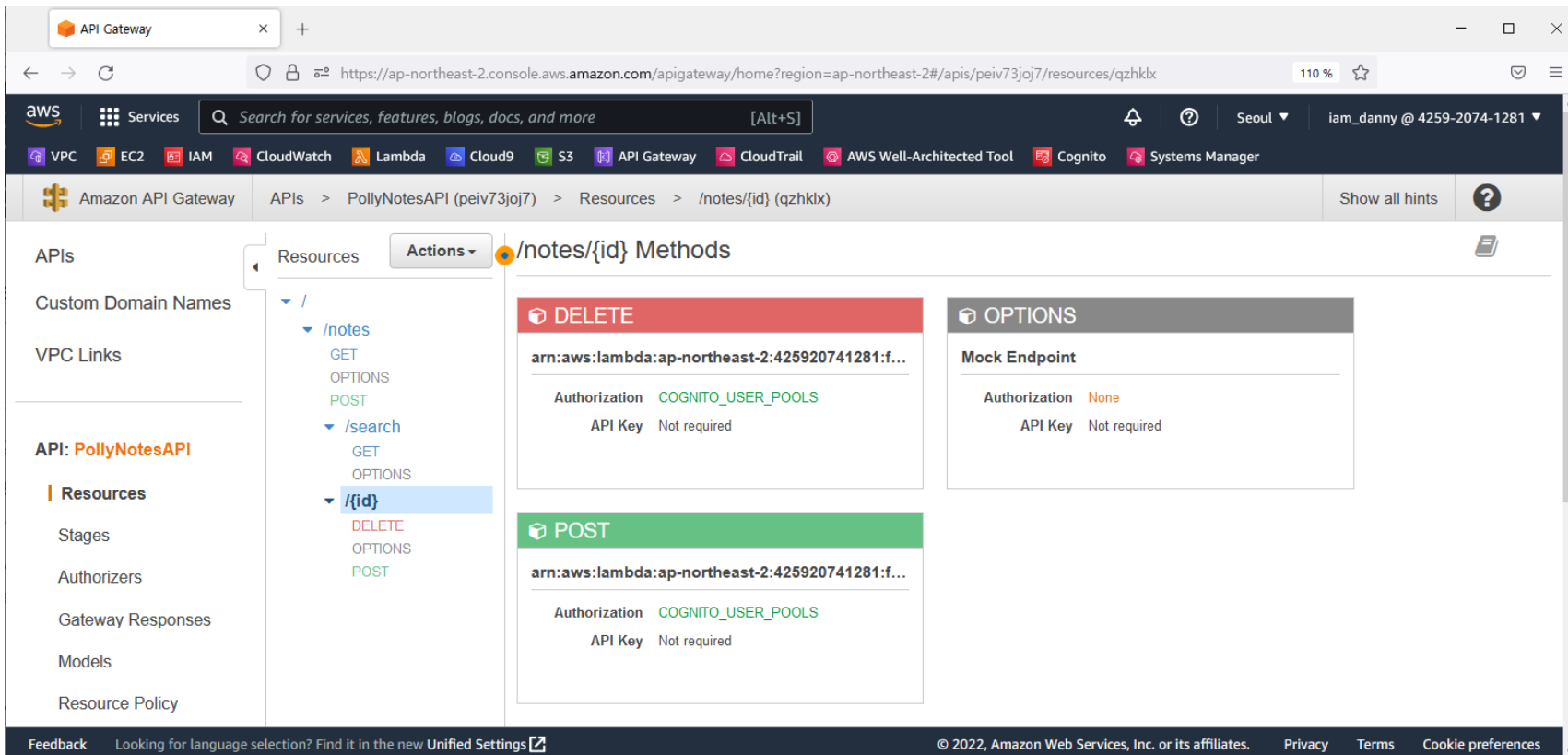
```
danny@DESKTOP-335KS8C MINGW64 ~ (master)
$ aws apigateway create-deployment --rest-api-id $apiId --stage-name Prod
{
  "id": "bnnact",
  "createdDate": "2022-09-19T00:23:56+09:00"
}
```

Swagger 파일로 API 리소스 생성

■ 함수에 Lambda 권한을 추가

- aws lambda add-permission --function-name delete-function --statement-id apilnvoke --action lambda:InvokeFunction --principal apigateway.amazonaws.com
- aws lambda add-permission --function-name dictate-function --statement-id apilnvoke --action lambda:InvokeFunction --principal apigateway.amazonaws.com
- aws lambda add-permission --function-name search-function --statement-id apilnvoke --action lambda:InvokeFunction --principal apigateway.amazonaws.com

■ 'API Gateway' 웹 브라우저 탭으로 이동하여 Resources 를 선택하고, Swagger로 생성된 새 리소스를 탐색



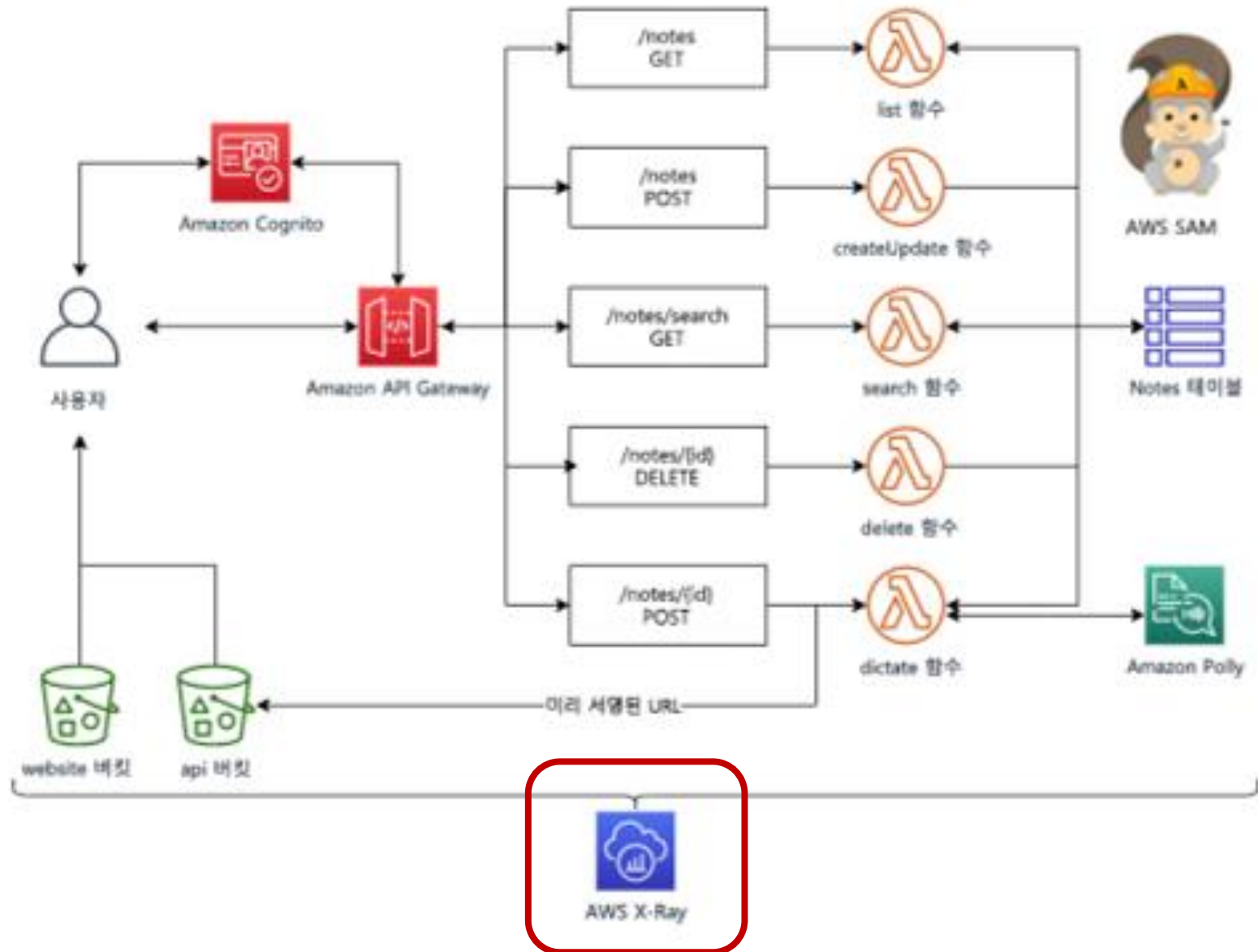
테스트용 웹 애플리케이션 구성



추가 예정

9. AWS X-Ray를 사용하여 애플리케이션 관찰

AWS X-Ray



함수에서 로깅 활성화

```
import logging
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

logger = logging.getLogger()
logger.setLevel(logging.INFO)
patch_all()
```

```
def add_annotation(UserId, NoteId):
    print('Adding annotation...')
    # TODO 2: add UserId and NoteId as annotations
    xray_recorder.begin_subsegment('Delete a note')
    xray_recorder.put_annotation("UserId", UserId)
    xray_recorder.put_annotation("NoteId", NoteId)
    xray_recorder.end_subsegment()
```

SAM 템플릿 사용자 지정 및 X-Ray Tracing 활성화

SAM 템플릿은 CloudFormation 템플릿의 확장에 속하며, 작업을 더 쉽게 처리하는 추가 구성 요소가 있습니다. 트레이싱은 리소스별로 활성화할 수 있지만 AWS SAM을 사용하면 전역적으로 활성화할 수 있습니다.

■ template.yml

- Globals 섹션 : 서버리스 함수와 API에 공통적인 속성을 정의
- Resources 섹션 : CloudFormation 리소스와 SAM 리소스의 조합을 포함
- pollyNotesAPI 리소스
 - 애플리케이션에서 사용할 API Gateway 리소스를 정의
 - Prod Stage 설정, CORS 활성화, 리전 배포 유형 지정, 사용할 권한 부여자 지정
- 5개의 AWS Lambda 함수는 AWS::Serverless::Function 리소스를 사용하여 정의
- Events 섹션 : AWS::Serverless::Function을 트리거하는 이벤트를 지정

```
Globals:
  Function:
    Tracing: Active
  Api:
    TracingEnabled: true
    MethodSettings:
      - LoggingLevel: INFO
        ResourcePath: '/*'
        HttpMethod: '*'
```

SAM을 사용하여 애플리케이션 배포

- 배포에 사용할 버킷 변수를 생성
`apiBucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `pollynotesapi`) == `true`].Name')`
- `sam build` 명령 실행
`sam build --use-container`
- `sam deploy` 명령 실행
`sam deploy --stack-name polly-notes-api --s3-bucket $apiBucket --parameter-overrides apiBucket=$apiBucket`

웹 브라우저에서 웹 사이트 테스트

추가 예정

X-Ray를 사용하여 애플리케이션 관찰

추가 예정

X-Ray를 사용하여 애플리케이션 관찰

추가 예정

애플리케이션 문제 해결 및 재배포

추가 예정

Thank you