Storage System

Backup

- AWS를 이용한 백업
  - ✓ 백업의 형태
    - 온프레미스 환경의 데이터를 AWS로 백업
    - AWS에 구축한 시스템을 백업하는 경우
  - ✓ 백업의 개요
    - 사내 시스템은 온프레미스 환경에 기업 사이트는 AWS 환경에 위치
    - 백업 대상은 용량이 큰 이미지 파일, 데이터베이스, 각 서버의 로그 파일, 파일 서버에 있는 파일
    - 로그 파일과 데이터베이스 파일은 장기간 보관
    - 비용을 절감하면서도 운영에 손이 많이 가지 않는 방식으로 진행
  - ✓ 인프라 핵심 설계 사항
    - 스토리지 게이트웨이를 이용한 자동 백업: 온프레미스 환경에 스토리지 게이트웨이를 사용하여 백업용 스토리지를 만들어 S3에 자동 백업
    - S3과 글레이셔로 수명 주기 관리: 로그 파일을 S3의 기능으로 백업하면서 온라인 보관 기간을 넘은 파일을 글레이셔에 아카이브
    - 용량의 대부분을 차지하는 이미지 파일과 데이터베이스는 S3에 백업: 스토리지 게이트웨이에 부과되는 종량제 요금을 줄이기 위해 용량이 큰 데이터는 스크립트를 사용하여 S3에 백업

Backup

- 온프레미스 환경의 데이터 백업
  - ✓ 온프레미스 환경의 데이터를 AWS에 백업하는 경우는 백업 대상이나 예산에 따라 적절한 설계 패턴이 달라짐
  - ✓ 일반적인 방법
    - 오브젝트 스토리지 서비스인 S3(Amazon Simple Storage Service)를 명령행 인터페이스를 통해 이용하는 패턴
      - 백업 대상이 파일인 경우에 유효하며 간편하면서도 저렴
      - S3는 범용적인 파일 저장 서비스로서 온라인 파일 서버처럼 사용할 수 있음
      - S3는 서로 다른 가용 영역(AZ)에 3중화되어 있으며 99.999999999% 라고 하는 높은 내구성
    - 가상 서버인 EC2와 가상 스토리지인 EBS로 구축한 백업용 서버에 데이터를 동기화시키는 패턴
      - 첫 번째 방법보다는 비용이 들지만 데이터베이스의 백업을 유연하게 실행할 수 있음
      - 백업 대상 EC2에서 유틸리티를 실행시키면 DB를 온라인 백업할 수 있음
    - 자동 백업 서비스인 AWS 스토리지 게이트웨이(Storage Gateway)를 이용하는 패턴
      - 첫 번째나 두 번째 방법보다 적은 노력으로 자동 백업 환경을 구축 할 수 있으면서도 백업 관리의 일원화도 가능
      - 스토리지 게이트웨이는 온프레미스 환경에 있는 서버에 스토리지 관리 전용 소프트웨어를 설치하여 AWS와 접속하는 서비스

S3

- S3 개요
  - ✓ S3(Simple Storage Service)는 인터넷 스토리지 서비스
  - ✓ 용량에 관계없이 파일을 저장할 수 있고 웹 (HTTP 프로토콜)에서 파일에 접근할 수 있으며 안정성이 뛰어나고 가용성이 높으며 무제한 확장이 가능
  - ✓ 대용량의 파일 저장을 EC2 와 EBS를 통해 구축한다면 상당히 많은 비용이 들고 노력이 요구되지만 S3는 저장 용량이 무한대이고 파일 저장에 최적화되어 있기 때문에 용량을 추가하거나 성능을 높이는 작업을 하지 않아도 되고 비용도 EC2 와 EBS로 구축하는 것보다 훨씬 저렴
  - ✓ EC2 와 EBS로 정적 웹 서비스(HTML과 JavaScript로만 구성된 웹사이트)를 구축한다면 일일이 EC2 와 EBS를 생성해 높은 요금을 낼 필요 없이 S3에서 바로 정적 웹 서비스(정적 웹 호스팅)를 사용하는 것이 가능
  - ✓ S3 자체가 수천 대 이상의 매우 성능이 좋은 웹 서버로 구성되어 있어서 EC2와 EBS로 구축했을 때처럼 자동 횡적 확장(Auto Scaling)이나 부하 분산(Load Balancing) 신경 쓰지 않아도 됨
  - ✓ 동적 웹 페이지(ASP, JSP, PHP. Ruby on Rails 등) 와 정적 웹 페이지가 섞여 있다면 동적 웹 페이지만 EC2에서 서비스하고 정적 웹 페이지는 S3를 이용하면 성능도 높일 수 있고 비용도 절감할 수 있음
  - ✓ S3는 파일 업로드/다운로드를 모두 HTTP 프로토콜로 처리하기 때문에 별도의 클라이언트나 ActiveX 설치없이 웹 브라우저에서 바로 사용 가능
  - ✓ Amazon S3는 대부분의 회사에서 사용해서 오래전 구축한 파일 서버 형태를 탈피하고 있는데 넷플릭스가 수십억 시간에 달하는 콘텐츠를 Amazon S3를 통해 서비스하고 있으며 Airbnb는 10 페타 바이트가 넘는 사용자 사진 그리고 백업 데이터 와 정적 파일을 모두 Amazon S3에 저장해서 사용하고 있음

S3

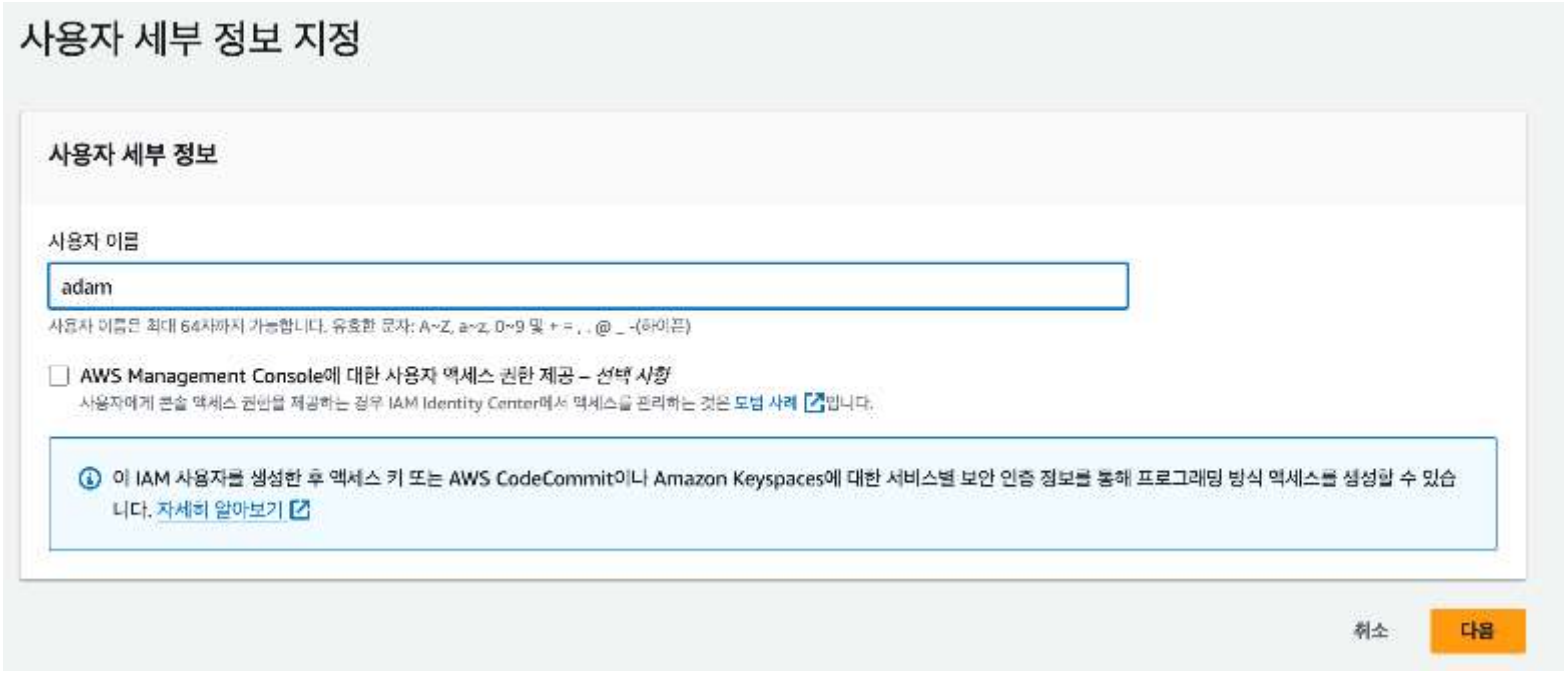
- S3 개요
  - ✓ 기본 개념
    - 객체
      - 파일과 메타 데이터로 이루어진 데이터가 저장되는 기본 단위
      - 기본적으로 키(Key)가 객체의 이름이며 값(Value)이 객체의 데이터
      - 객체 하나의 크기는 1Byte ~ 5TB
      - 메타 데이터는 MIME형식으로 확장자를 통해 자동 설정되고 사용자가 임의로 지정
      - MIME(Multipurpose Internet Mail Extensions)이란 파일 변환을 의미하는데 이메일과 함께 동봉할 파일을 텍스트 문자로 전환해서 이메일 시스템을 통해 전달하기 위해 개발되었기 때문에 이름에 Internet Mail Extension이 들어가며 현재는 웹을 통해 여러 형태의 파일을 전달하는 데 쓰이고 있음
    - Bucket
      - Amazon S3에서 생성할 수 있는 최상위 디렉토리
      - Bucket은 리전 별로 생성되며 계정 별로 100개까지 생성이 가능
      - Bucket 안에 객체가 저장되고 디렉토리 생성 가능(객체 이름이 디렉터리 경로까지 포함)
      - 저장 가능한 객체의 개수 와 저장 가능한 용량은 무제한
      - 접속 제어 및 권한 관리가 가능
      - http://examplebucket.s3.amazonaws.com/helloworld.jpg 처럼 URL로 접근 가능

S3

- S3 개요
  - ✓ 기본 개념
    - 내구성과 가용성: 1년 기준으로 99.999999999% 내구성, 99.99% 가용성을 가지고 있음
    - 내구성은 데이터가 유실되지 않는 것을 의미하고 가용성은 언제나 정상적으로 사용 가능한 상태
    - 내구성이 다른 2가지 스토리지 옵션
      - 표준 스토리지(Standard Storage): 일반적인 스토리지 옵션으로 99.999999999% 내구성을 가지고 있지만 AWS 내부적으로도 이런 높은 내구성을 유지하려면 그만큼 비용이 많이 들게 되고 요금도 높아지기 때문에 유실되면 안 되는 중요한 데이터 저장에 권장
      - 낮은 중복 스토리지(RRS - Reduced Redundancy Storage): 표준 스토리지보다는 낮은 99.99%의 내구성을 가지고 있는데 데이터를 복제한 사본의 수를 줄여 비용을 낮추었기 때문에 요금이 저렴한데 원본 데이터를 다른 곳에 가지고 있거나 동영상이나 이미지의 썸네일 등 원본에서 다시 생성할 수 있는 데이터 저장에 적합
    - 요금: 저장 용량과 데이터 전송량, HTTP 요청(Request) 개수로 책정

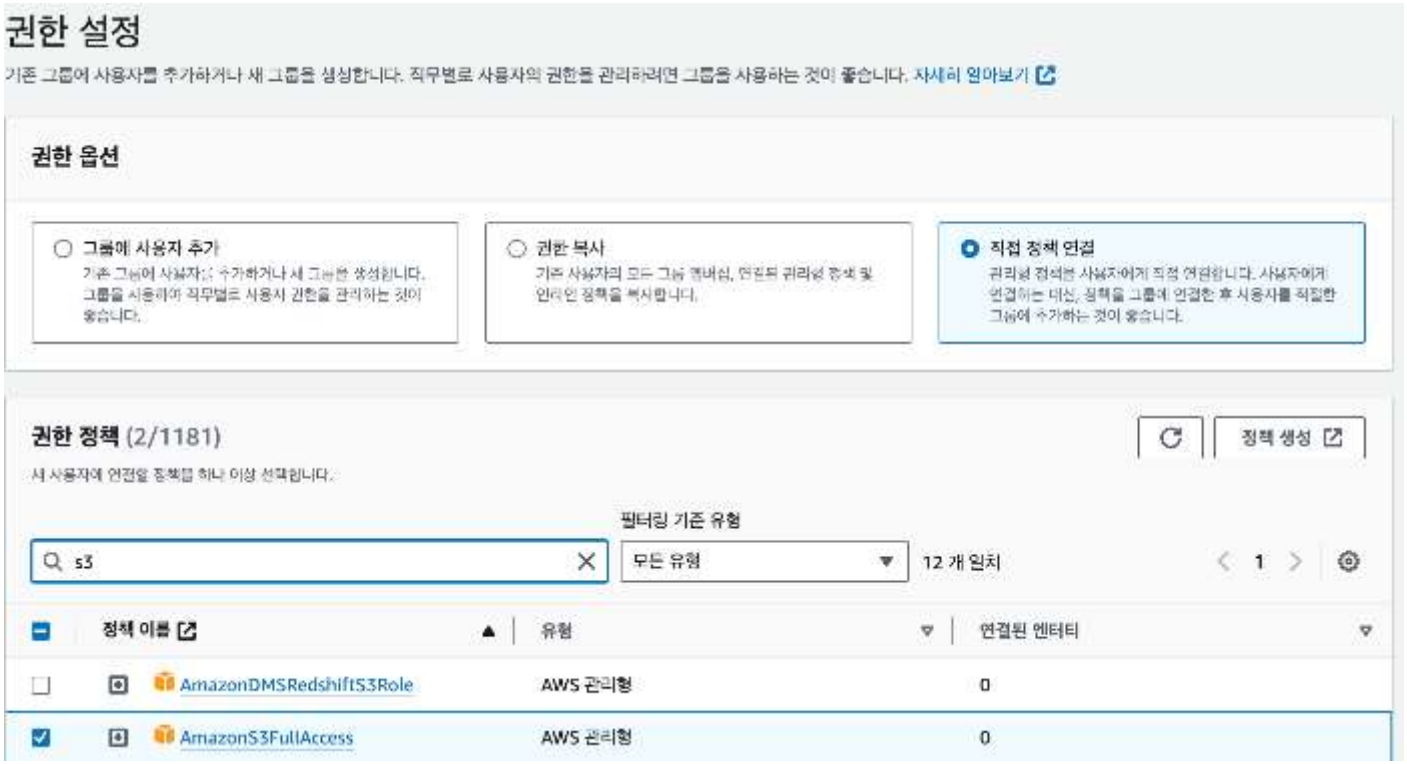
S3

- S3 Bucket 생성
  - ✓ IAM 사용자 생성 및 권한 설정 - 외부에서 사용을 하고자 하는 경우에 생성



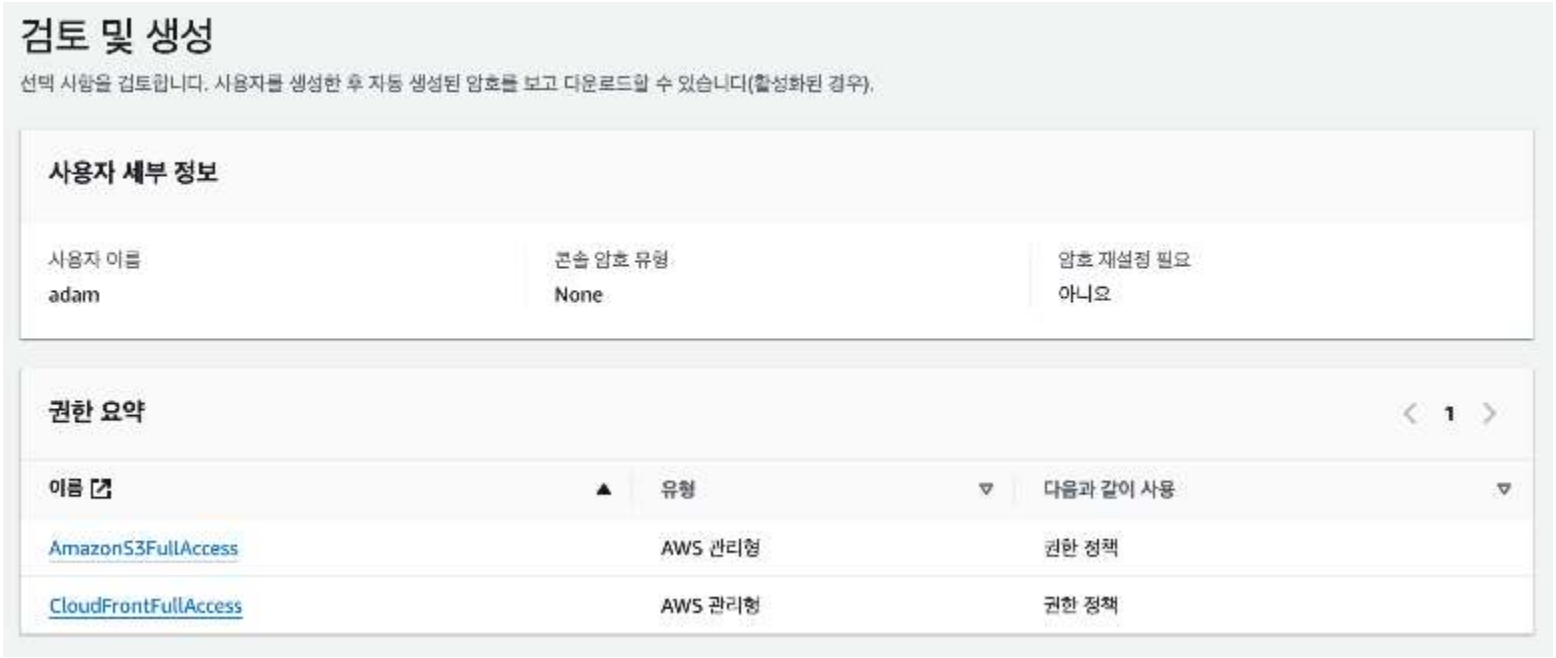
S3

- S3 Bucket 생성
  - ✓ IAM 사용자 생성 및 권한 설정 - 외부에서 사용을 하고자 하는 경우에 생성



S3

- S3 Bucket 생성
  - ✓ IAM 사용자 생성 및 권한 설정 - 외부에서 사용을 하고자 하는 경우에 생성



S3

- S3 Bucket 생성
  - ✓ IAM 사용자 생성 및 권한 설정 - 외부에서 사용을 하고자 하는 경우에 생성
    - ✓ IAM 사용자 Access Key 및 Secret Key 생성
      - https://console.aws.amazon.com/iam/에서 IAM 콘솔 열기
      - 탐색 메뉴에서 사용자를 선택
      - IAM 사용자 이름(확인란이 아님)을 선택
      - Security credentials(보안 자격 증명) 탭을 연 다음 Create access key(액세스 키 생성)를 선택
      - 새 액세스 키를 보려면 [Show]를 선택
        - 액세스 키 ID: AKIAIOSFODNN7EXAMPLE
        - 보안 액세스 키: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      - 키 페어 파일을 다운로드하려면 [Download .csv file]을 선택한 후 안전한 위치에 키와 함께 .csv 파일을 저장

S3

- S3 Bucket 생성
  - ✓ IAM 사용자 생성 및 권한 설정 - 외부에서 사용을 하고자 하는 경우에 생성
    - ✓ IAM 사용자 Access Key 및 Secret Key 생성

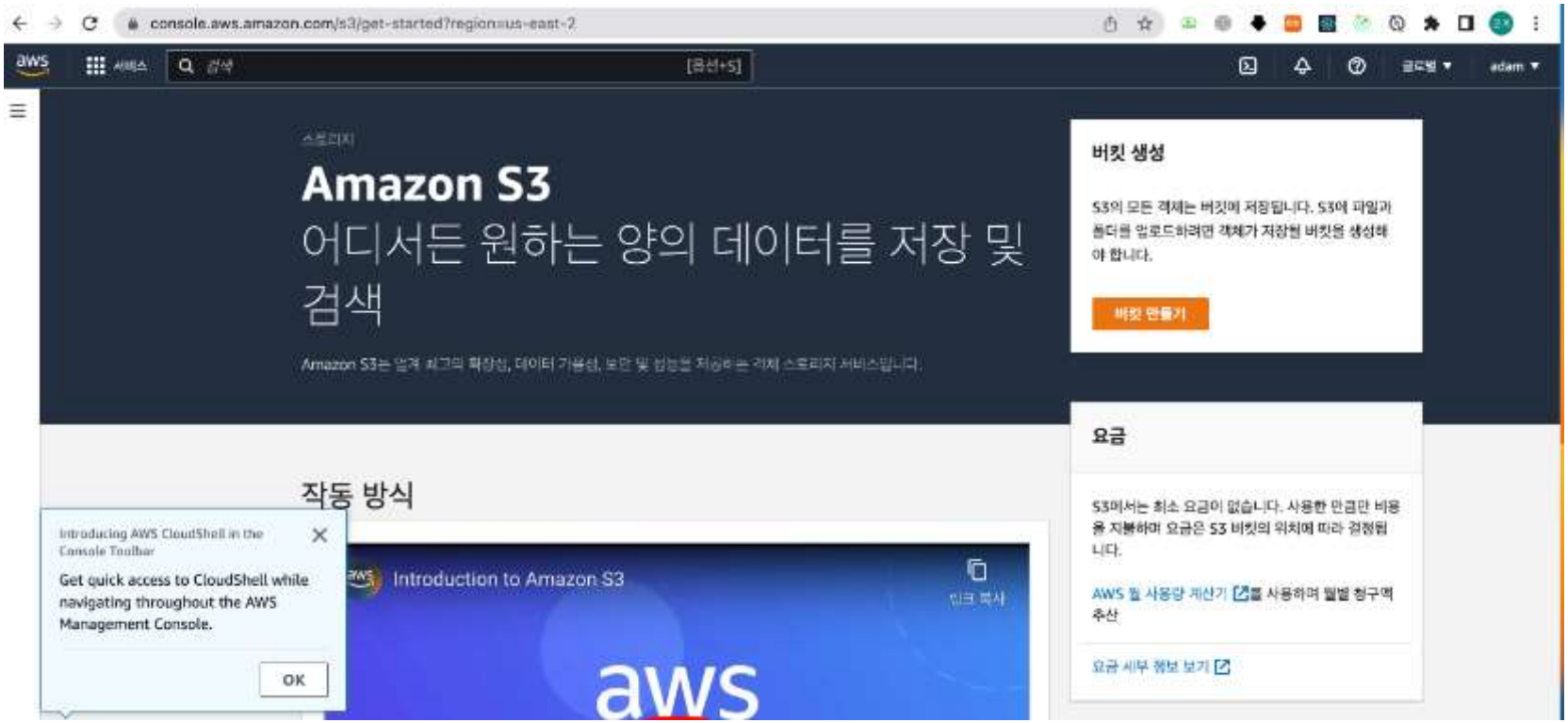
S3

- ❖ S3 Bucket 생성
  - ✓ IAM 사용자 생성 및 권한 설정 - 외부에서 사용을 하고자 하는 경우에 생성
    - ● IAM 사용자 Access Key 및 Secret Key 생성

Access key ID	Secret access key			
AKIAZKP7ENH	Rje6Hea+o8FTlxO0/taoRTNG1WT3vfuLAcfZhKrt			

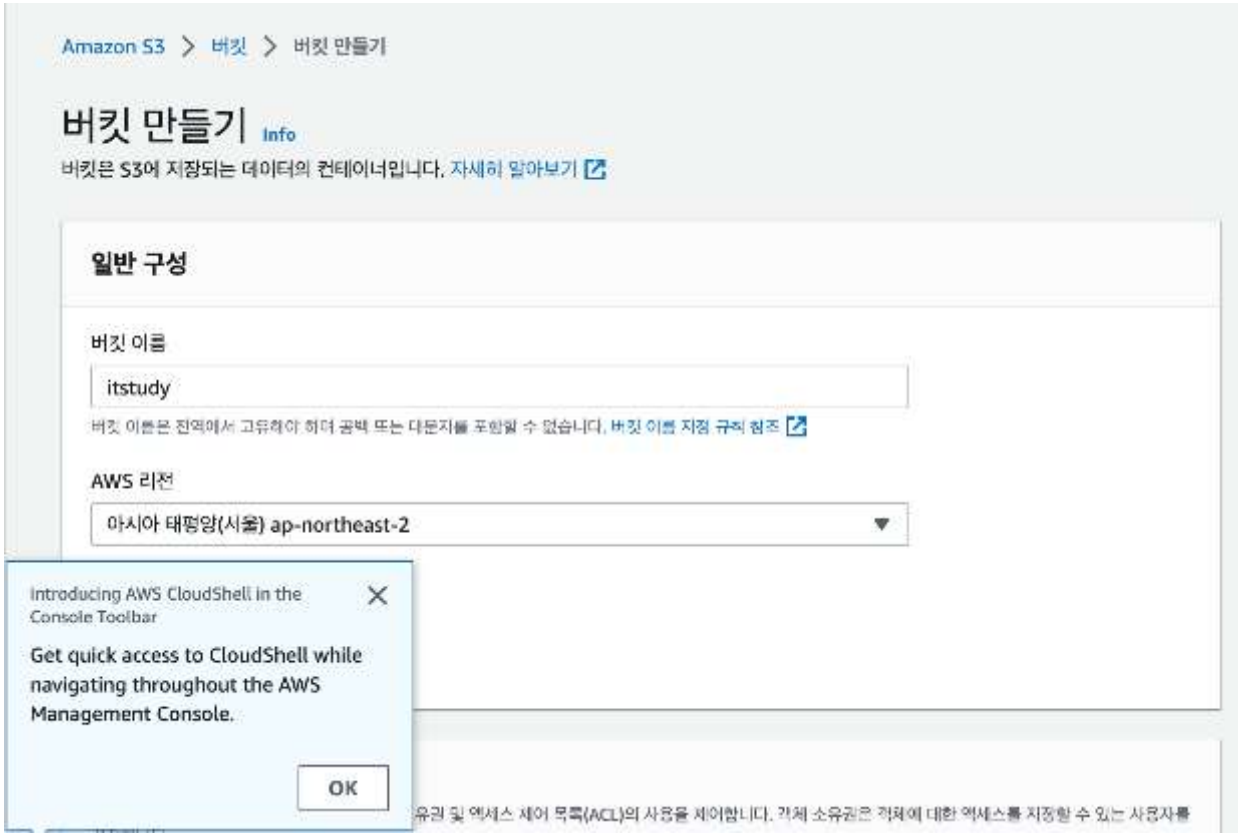
S3

- S3 Bucket 생성
  - ✓ Bucket 생성
    - https://console.aws.amazon.com/s3



S3

- ❖ S3 Bucket 생성
  - ✓ Bucket 생성
    - ● Bucket 만들기 – 이름 과 리전 설정

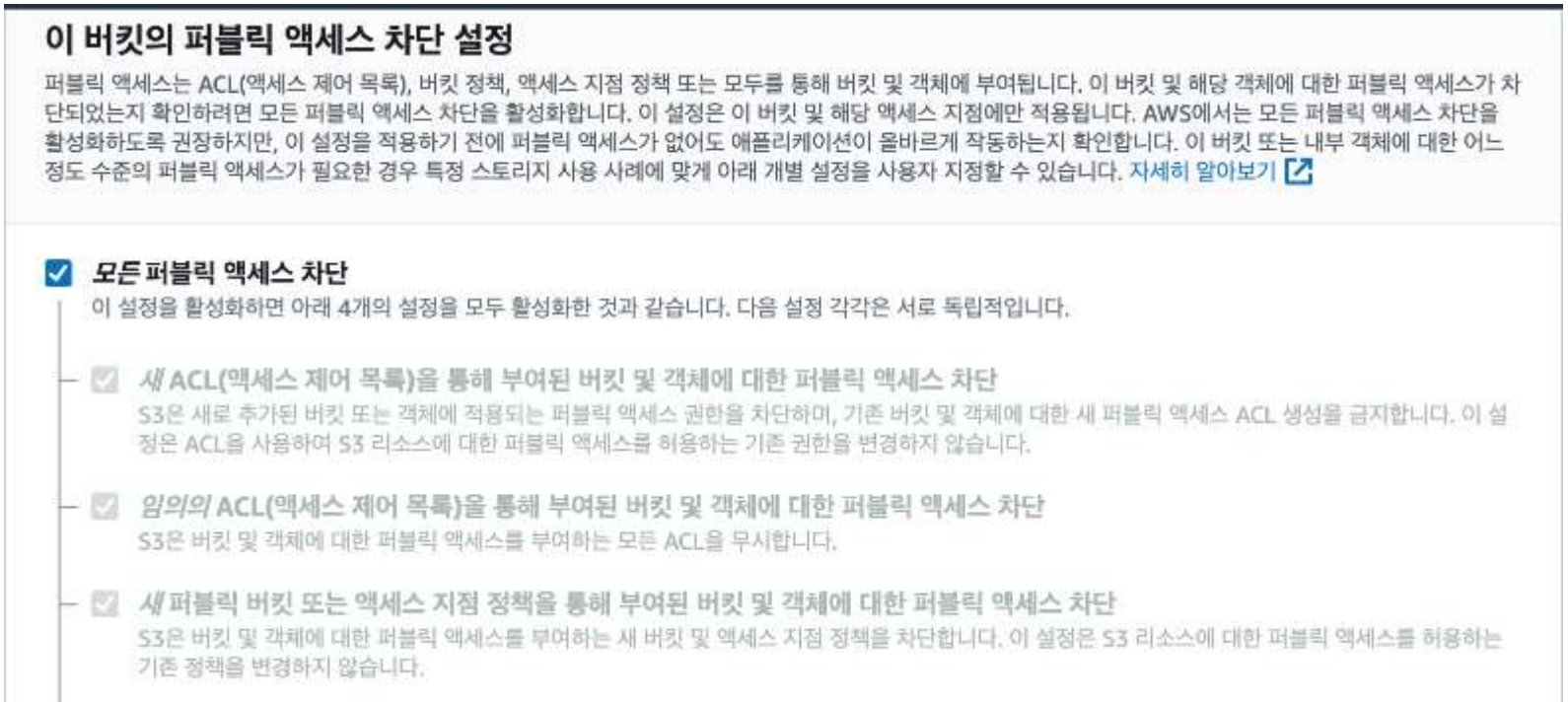


S3

- ❖ S3 Bucket 생성
  - ✓ Bucket 생성
    - Bucket 만들기 – ACL 설정(ACL 활성화됨 선택)

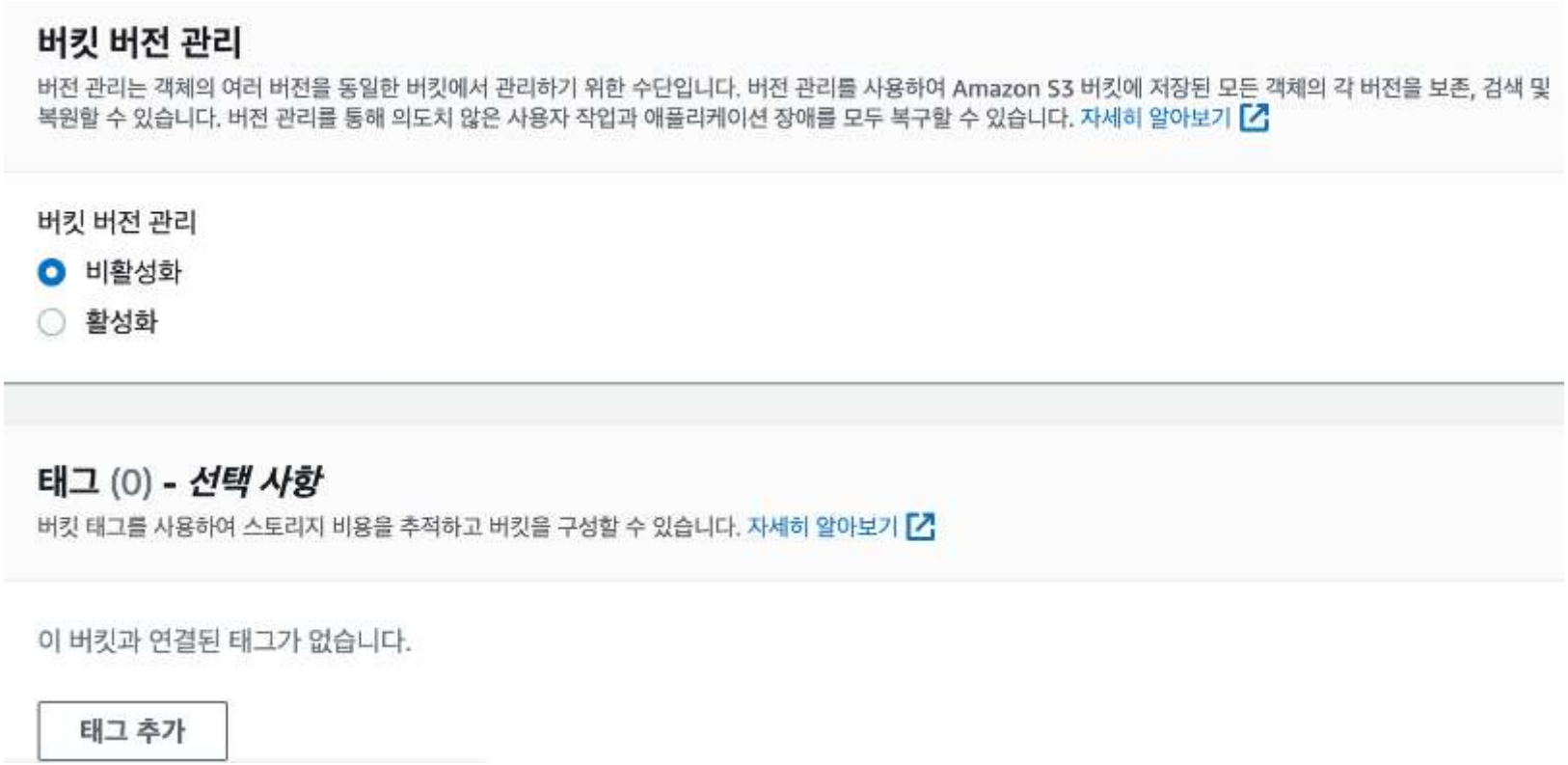
S3

- S3 Bucket 생성
  - ✓ Bucket 생성
    - Bucket 만들기 – 퍼블릭 엑세스 차단 설정(차단 설정 해제)



S3

- S3 Bucket 생성
  - ✓ Bucket 생성
    - Bucket 만들기 – 기타 옵션 설정

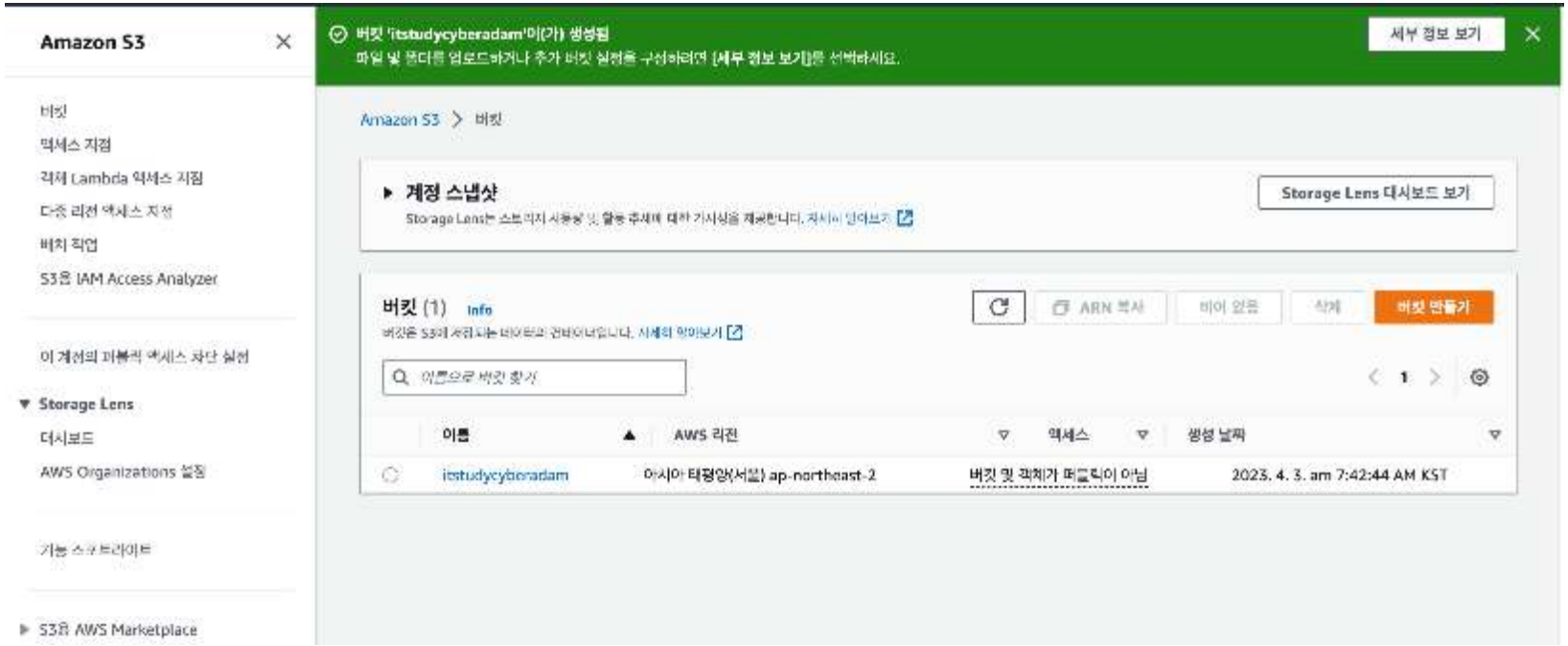


S3

- S3 Bucket 생성
  - ✓ Bucket 생성
    - Bucket 만들기 – 기타 옵션 설정

S3

- S3 Bucket 생성
  - ✓ Bucket 생성
    - Bucket 만들기



S3

- S3 Bucket 생성
  - ✓ 설정
    - 외부에서 Bucket을 사용할 수 있도록 [권한] – [버킷 정책] 에서 [편집]을 눌러서 Bucket 정책 추가

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicListGet",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:List*",
                "s3:Get*",
                "s3:Put*",
                "s3:Delete*"
            ],

            "Resource": [
                "arn:aws:s3:::Bucket이름",
                "arn:aws:s3:::Bucket이름/*"
            ]
        }
    ]
}
```

S3

- S3 Bucket 생성
  - ✓ 설정
    - [권한] – [CORS] 에서 CORS 정책 추가

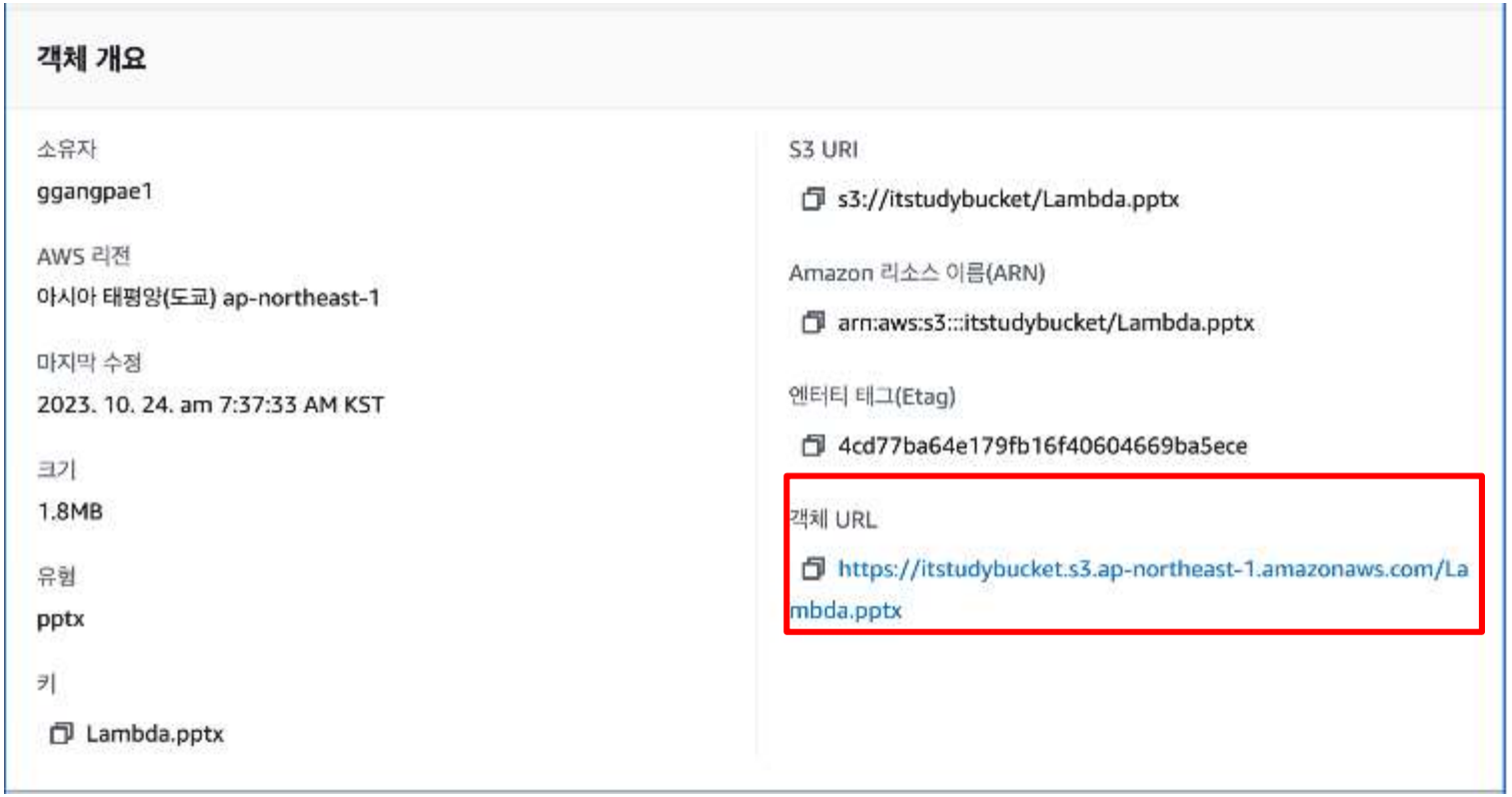
```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": ["Access-Control-Allow-Origin"]
    }
]
```

S3

- S3 Bucket 생성
  - ✓ 파일 업로드

S3

- S3 Bucket 생성
  - ✓ 파일 다운로드



S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트 생성
    - 프로젝트 디렉토리 생성: pythons3
    - 가상 환경 생성
      - ◆ python3 -m venv myvenv
      - ◆ source myvenv/bin/activate
    - 필요한 패키지 설치
      - ◆ pip install Django
      - ◆ pip install django-cors-headers
      - ◆ pip install mysqlclient
    - 프로젝트 생성: django-admin startproject s3project .
    - 애플리케이션 생성: python manage.py startapp s3application

S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트 설정
    - ● s3project/settings.py 파일을 수정

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['*']
# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "s3application",
    "corsheaders", #CORS 관련 추가
]
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트 설정
    - ● s3project/settings.py 파일을 수정

```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware', # CORS 관련 추가 (가장 상단에 위치)
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",

]
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트 설정
    - s3project/settings.py 파일을 수정

```
#CORS 관련 추가
CORS_ALLOW_ALL_ORIGINS = True #(모든 포트 허용)

#HTTP methods 추가
CORS_ALLOW_METHODS = (
"DELETE",
"GET",
"OPTIONS",
"PATCH",
"POST",
"PUT",
)
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트 설정
    - ● s3project/settings.py 파일을 수정

```
# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'adam',
        'USER': 'root',
        'PASSWORD': 'wnddkd', # mariaDB 설치 시 입력한 root 비밀번호 입력
        'HOST': '127.0.0.1',
        'PORT': ''
    }
}

TIME_ZONE = "Asia/Seoul"
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트 설정
    - ● M1 맥북을 사용하는 경우
      - ◆ pip install pymysql
      - ◆ myproject/settings.py 파일에 추가

```
import pymysql
pymysql.install_as_MySQLdb()
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 모델 생성
    - ● s3application 디렉토리의 models.py 파일 수정 – 데이터베이스 테이블 과 연동할 모델 생성

```
from django.db import models

class FileContent(models.Model):
    title = models.TextField(max_length=40, null=True)
    imgfile = models.ImageField(null=True, upload_to="", blank=True)
    content = models.TextField()

    def __str__(self):
        return self.title
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 모델 생성
    - migration

      # 마이그레이션 파일 만들기
      python manage.py makemigrations

      # 마이그레이션 적용
      python manage.py migrate

    - 데이터베이스에 테이블이 만들어 졌는지 확인

S3

- ❖ django 에서의 파일 업로드
  - ✓ 요청 처리
    - ● s3application의 views.py 파일을 수정

```
from .models import FileContent
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def fileUpload(request):
    if request.method == 'POST':
        title = request.POST['title']
        content = request.POST['content']
        img = request.FILES["imgfile"]
        fileContent = FileContent(
            title=title,
            content=content,
            imgfile=img,
        )
        print(fileContent)
        fileContent.save()
        return JsonResponse({"result":"success"})
    else:
        return JsonResponse({"result":"fail"})
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ 요청 처리
    - urls.py 파일을 수정

```
from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings

# fileupload/urls.py
from django.urls import path
from fileupload import views

urlpatterns = [
        path('fileupload/', views.fileUpload, name="fileupload"),
]
```

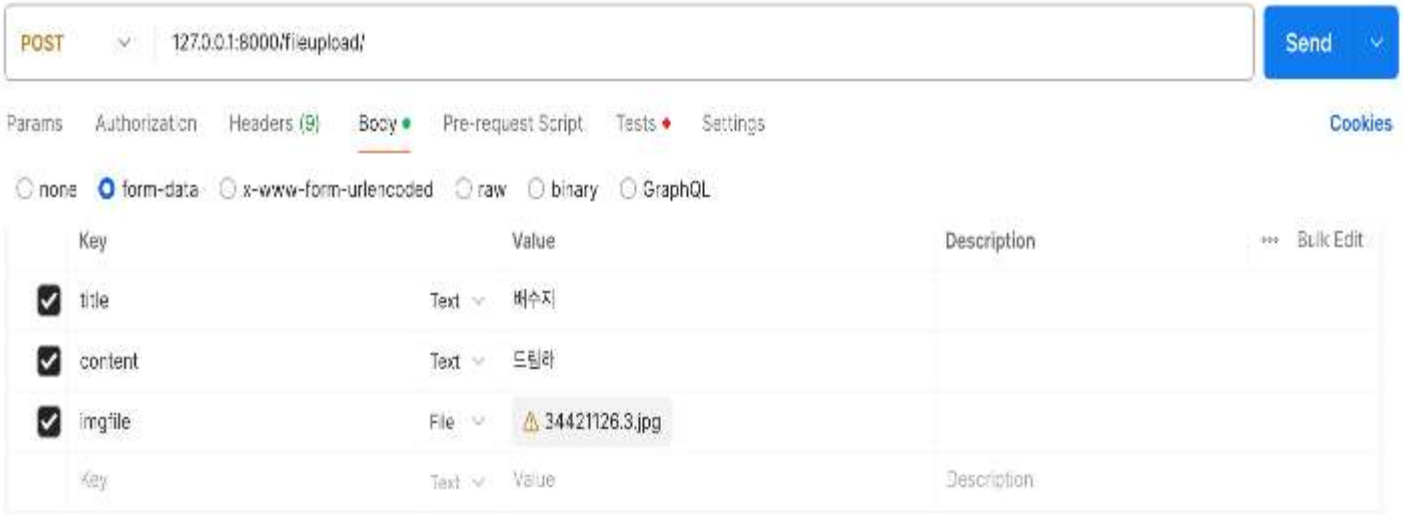
S3

- ❖ django 에서의 파일 업로드
  - ✓ 요청 처리
    - settings.py 파일에 파일을 저장하기 위한 옵션을 추가

```
import os
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

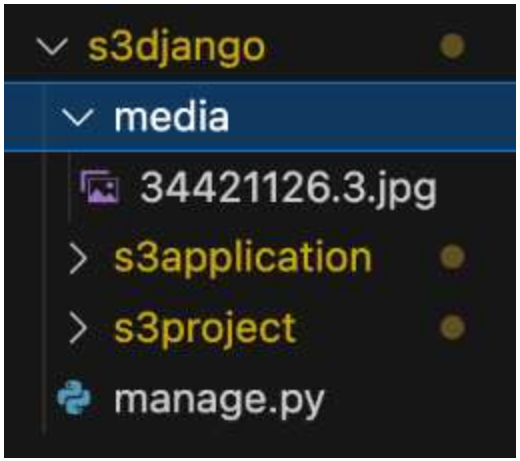
S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트를 실행해서 파일 업로드 확인
    - 실행: python manage.py runserver
    - Postman 에서 테스트



S3

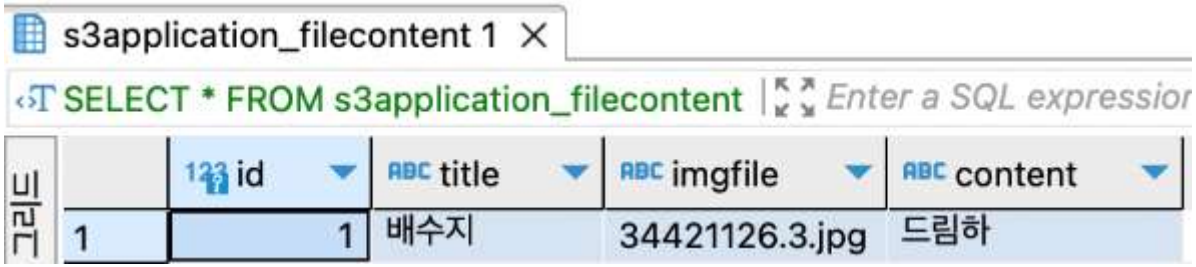
- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트를 실행해서 파일 업로드 확인
    - ● 프로젝트에서 파일 업로드 확인



S3

- ❖ django 에서의 파일 업로드
  - ✓ 프로젝트를 실행해서 파일 업로드 확인
    - ● 데이터베이스 확인

```
SELECT  *
FROM s3application_filecontent;
```



S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드
    - ● 패키지 설치

      pip install boto3
      pip install django-storages

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드
    - ● settings.py 파일에 추가

```
# AWS Setting
AWS_REGION = 'ap-northeast-1' #AWS서버의 지역
AWS_STORAGE_BUCKET_NAME = 'itstudybucket' #생성한 Bucket 이름
AWS_ACCESS_KEY_ID = 'AKIA264FV5MWPNFTR67R' #액서스 키 ID
AWS_SECRET_ACCESS_KEY = 'Dkhqua9VeWRvYINfP5yZQxuUc/ZHLI6W5lJ++0D2' #액
서스 키 PW
#Bucket이름.s3.AWS서버지역.amazonaws.com 형식
AWS_S3_CUSTOM_DOMAIN = '%s.s3.%s.amazonaws.com' %
(AWS_STORAGE_BUCKET_NAME, AWS_REGION)

# Static Setting
STATIC_URL = "https://%s/static/" % AWS_S3_CUSTOM_DOMAIN
STATICFILES_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드
    - ● settings.py 파일에 추가

```
# Media Setting
MEDIA_URL = "https://%s/meida/" % AWS_S3_CUSTOM_DOMAIN
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드
    - ● AWS 와 연결: python manage.py collectstatic
      - ◆ Bucket을 확인하면 static 파일이 업로드 되고 admin 이라는 디렉토리가 생성된 것을 확인할 수 있음

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드
    - ● 프로젝트를 실행하고 파일 업로드를 수행한 후 버킷을 확인

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드되는 파일 이름 수정
    - views.py 파일의 업로드 처리 메서드 수정

```
import uuid

@csrf_exempt
def fileUpload(request):
    if request.method == 'POST':
        title = request.POST['title']
        content = request.POST['content']
        img = request.FILES["imgfile"]
        idx = img.name.rfind(".")
        if idx != 1:
            img.name = img.name[:idx] + str(uuid.uuid4()) + img.name[idx:]
        else:
            img.name = img.name + str(uuid.uuid4())
        fileupload = FileUpload(
            title=title,
            content=content,
            imgfile=img,
        )
        fileupload.save()
        return redirect('fileupload')
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드되는 파일 이름 수정
    - ● views.py 파일의 업로드 처리 메서드 수정

```
else:
    fileuploadForm = FileUploadForm
    context = {
        'fileuploadForm': fileuploadForm,
    }
    return render(request, 'fileupload.html', context)
```

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS에 업로드되는 파일 이름 수정
    - ● 프로젝트를 실행하고 파일 업로드를 수행한 후 버킷을 확인

S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS 이미지 출력
    - views.py 파일에 기본 요청 처리 메서드 추가

```
def main(request):
    pictures = FileUpload.objects.all()

    return render(request, 'main.html', {'result': pictures})
```

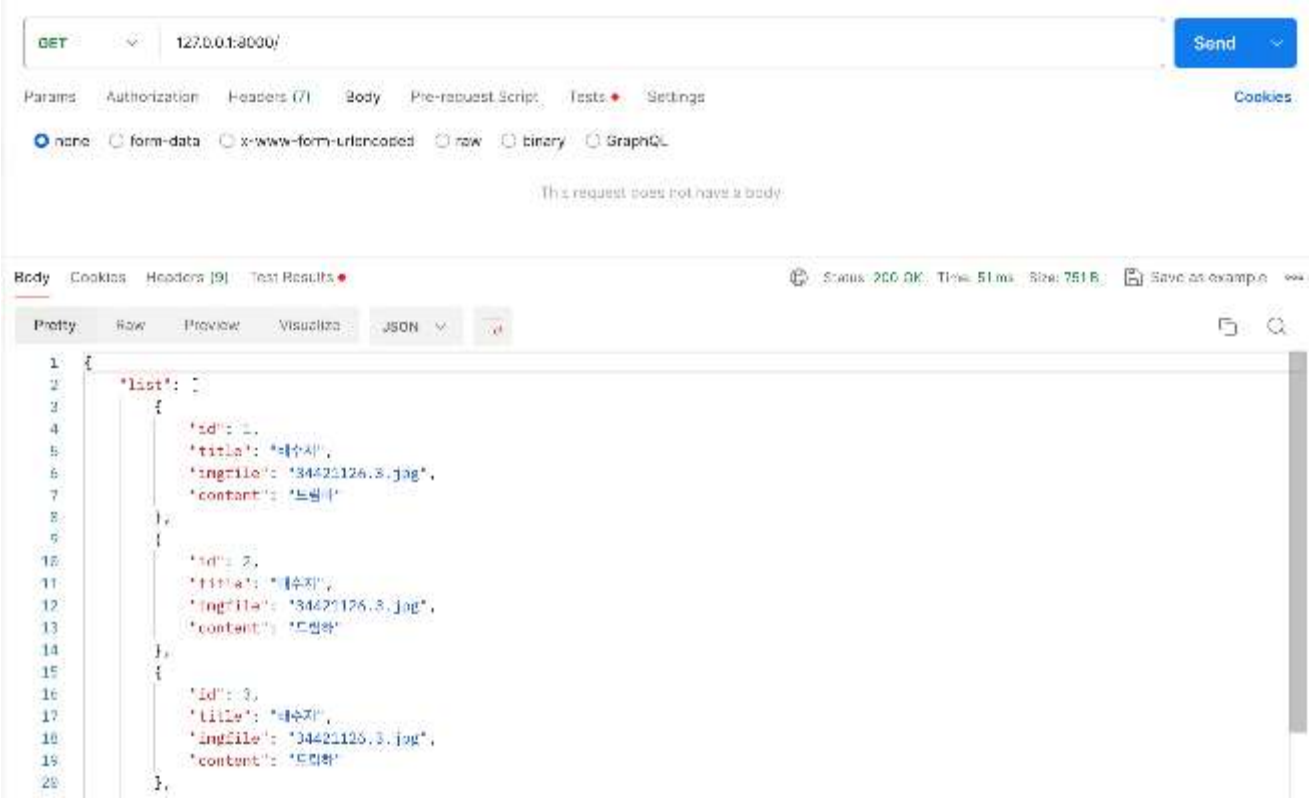
S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS 이미지 출력
    - ● urls.py 파일 수정

```
from django.urls import path
from s3application import views

urlpatterns = [
    path('', views.main),
    path('fileupload/', views.fileUpload, name="fileupload"),
]
```

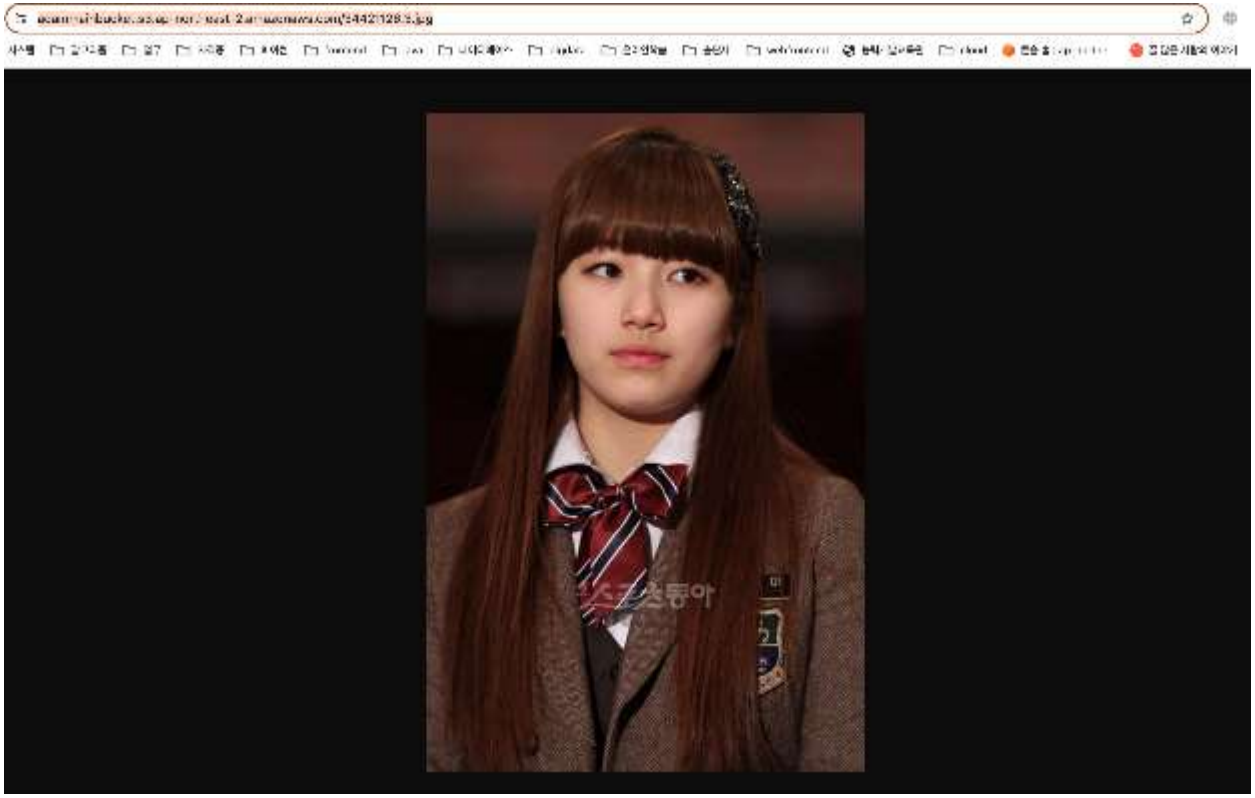
S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS 이미지 출력
    - ● 프로젝트 수행 후 기본 요청 수행 후 결과 확인



S3

- ❖ django 에서의 파일 업로드
  - ✓ AWS 이미지 출력
    - ● 브라우저에서 imgfile 앞에 버킷의 EndPoint를 추가해서 다운로드 되는지 확인



S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - Spring Boot 프로젝트 생성 – Spring Web, Lombok 의 의존성만 설정
    - 프로젝트의 build.gradle에 S3 사용을 위한 의존성 추가

      implementation 'io.awspring.cloud:spring-cloud-starter-aws:2.3.1'

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - application.properties 파일에 S3 설정을 추가

```
# AWS Account Credentials (AWS 접근 키)
cloud.aws.credentials.accessKey=AKIAWN5SS7IWEYL55SWG
cloud.aws.credentials.secretKey=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

# AWS S3 bucket Info (S3 Bucket정보)
cloud.aws.s3.bucket=adamsoft
cloud.aws.region.static=us-east-1
cloud.aws.stack.auto=false

# file upload max size (파일 업로드 크기 설정)
spring.servlet.multipart.max-file-size=20MB
spring.servlet.multipart.max-request-size=20MB
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 업로드 용량 초과 예외를 처리할 예외 처리 클래스 생성

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.ErrorResponse;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.multipart.MaxUploadSizeExceededException;
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 업로드 용량 초과 예외를 처리할 예외 처리 클래스 생성

```
@Component
@RestControllerAdvice
public class FileUploadFailedException{
    @ExceptionHandler(MaxUploadSizeExceededException.class)
    protected ResponseEntity<ErrorResponse>
     handleMaxUploadSizeExceededException(
            MaxUploadSizeExceededException e) {

        ErrorResponse response = ErrorResponse.builder(e, HttpStatus.BAD_REQUEST,
     "용량 초과").build();
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }
}
```

S3

- Spring Boot 프로젝트
  - ✓ 파일 업로드
    - 업로드할 파일 경로를 위한 클래스를 생성

```
public class CommonUtils {
    private static final String FILE_EXTENSION_SEPARATOR = ".";
    private static final String CATEGORY_PREFIX = "/";
    private static final String TIME_SEPARATOR = "_";

    public static String buildFileName(String category, String originalFileName) {
        int fileExtensionIndex =
     originalFileName.lastIndexOf(FILE_EXTENSION_SEPARATOR);
        String fileExtension = originalFileName.substring(fileExtensionIndex);
        String fileName = originalFileName.substring(0, fileExtensionIndex);
        String now = String.valueOf(System.currentTimeMillis());

        return category + CATEGORY_PREFIX + fileName + TIME_SEPARATOR + now
     + fileExtension;
    }
}
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 서비스 클래스를 생성

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.io.InputStream;
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 서비스 클래스를 생성

```
@Slf4j
@RequiredArgsConstructor
@Service
public class AwsS3Service {
    private AmazonS3 amazonS3Client;

    @Value("${cloud.aws.credentials.accessKey}")
    private String accessKey;

    @Value("${cloud.aws.credentials.secretKey}")
    private String secretKey;

    @Value("${cloud.aws.s3.bucket}")
    private String bucketName;

    @Value("${cloud.aws.region.static}")
    private String region;
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 서비스 클래스를 생성

```
@PostConstruct
public void setS3Client() {
    AWSCredentials credentials = new BasicAWSCredentials(this.accessKey,
 this.secretKey);

    amazonS3Client = AmazonS3ClientBuilder.standard()
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .withRegion(this.region)
            .build();
}
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 서비스 클래스를 생성

```
public String uploadFileV1(String category, MultipartFile multipartFile) {
    boolean result = validateFileExists(multipartFile);
    if(result == false){
        return null;
    }
    String fileName = CommonUtils.buildFileName(category,
multipartFile.getOriginalFilename());
    ObjectMetadata objectMetadata = new ObjectMetadata();
    objectMetadata.setContentType(multipartFile.getContentType());
    try (InputStream inputStream = multipartFile.getInputStream()) {
        amazonS3Client.putObject(new PutObjectRequest(bucketName, fileName,
inputStream, objectMetadata)
                .withCannedAcl(CannedAccessControlList.PublicRead));
    } catch (IOException e) {
        return null;
    }
    return amazonS3Client.getUrl(bucketName, fileName).toString();
}
}
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 서비스 클래스를 생성

```
    private boolean validateFileExists(MultipartFile multipartFile) {
        boolean result = true;
        if (multipartFile.isEmpty()) {
            result = false;
        }
        return result;
    }
}
```

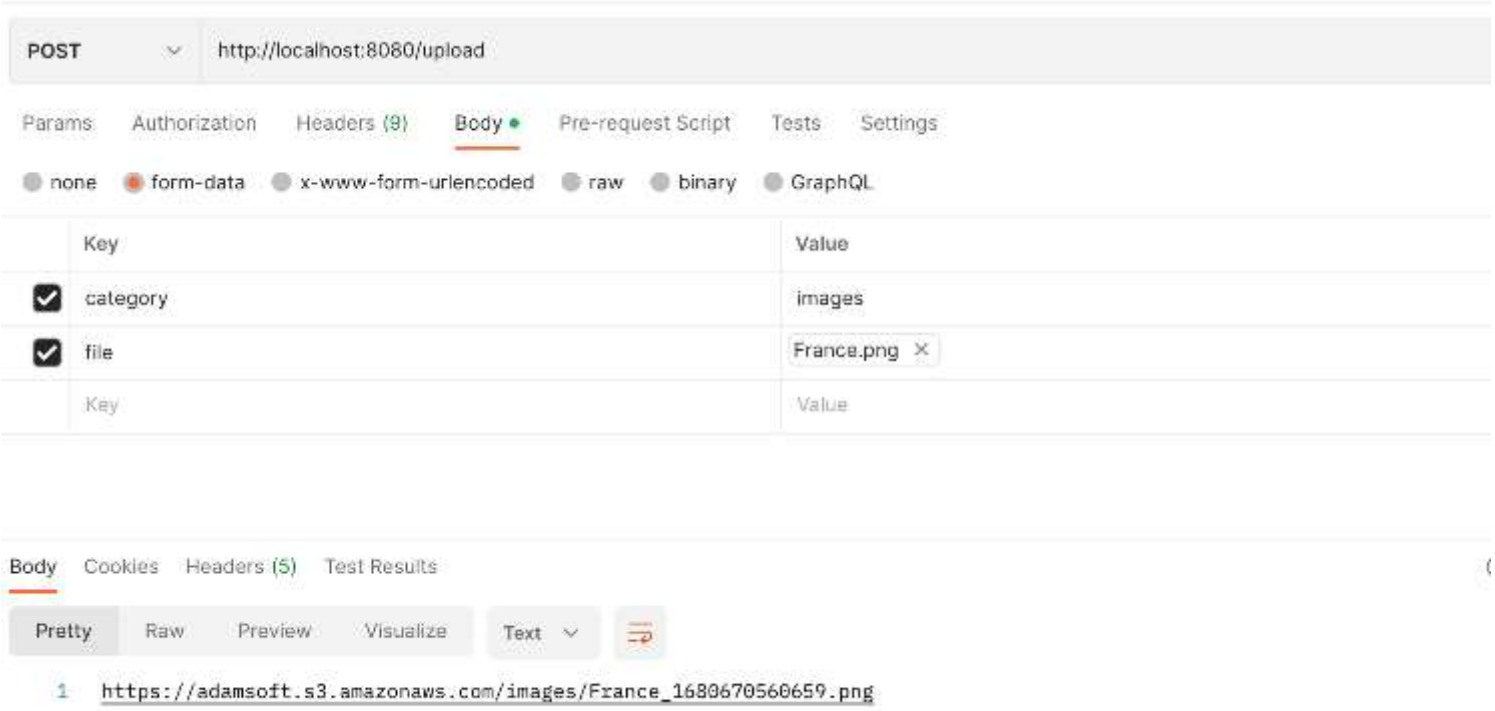
S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 컨트롤러 클래스를 생성

```
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

@RestController
@RequiredArgsConstructor
public class S3FileController {

    private final AwsS3Service awsS3Service;
    @PostMapping("/upload")
    public String uploadFile(
            @RequestParam("category") String category,
            @RequestPart(value = "file") MultipartFile multipartFile) {
        return awsS3Service.uploadFileV1(category, multipartFile);
    }
}
```

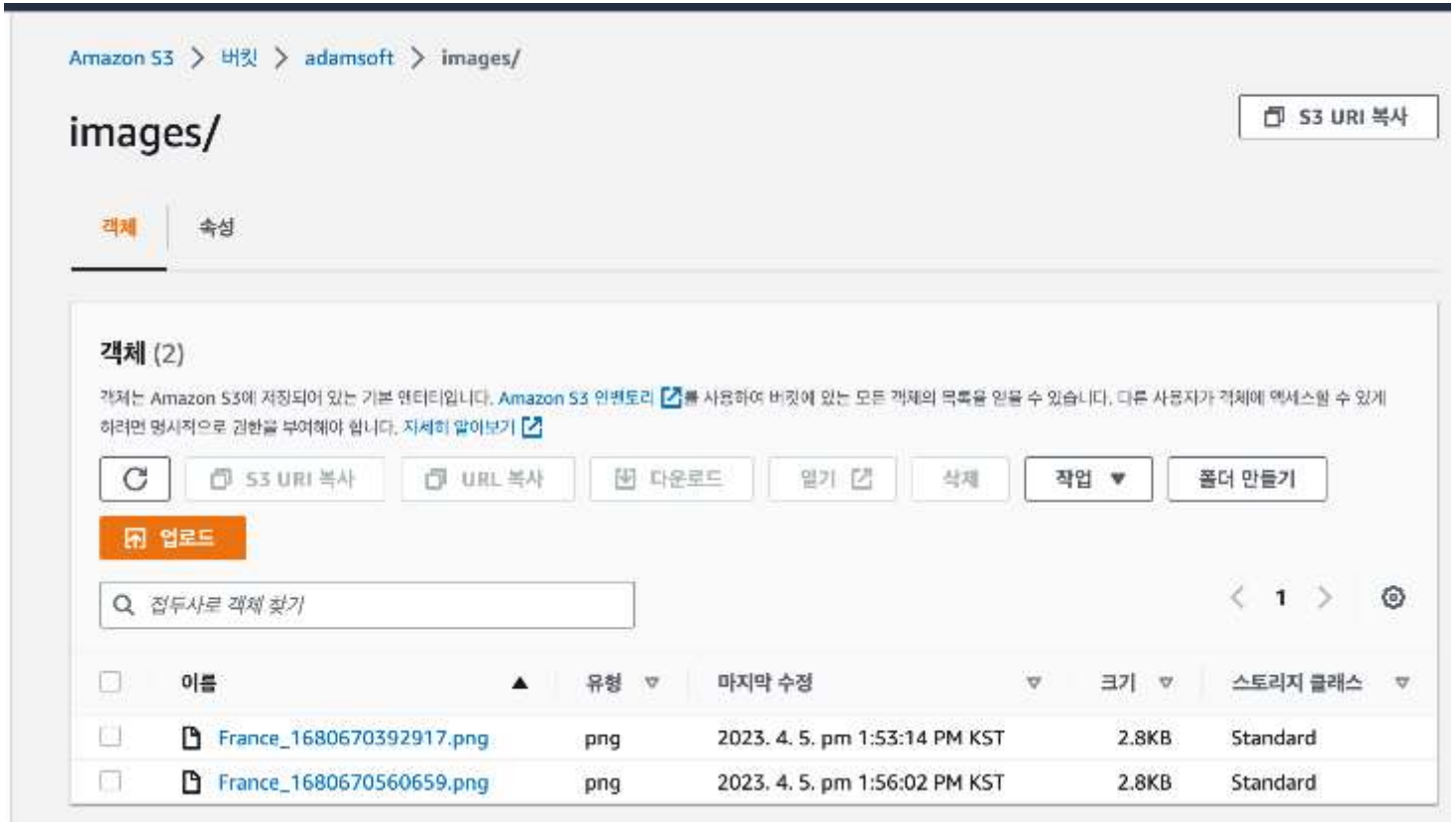
S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 테스트



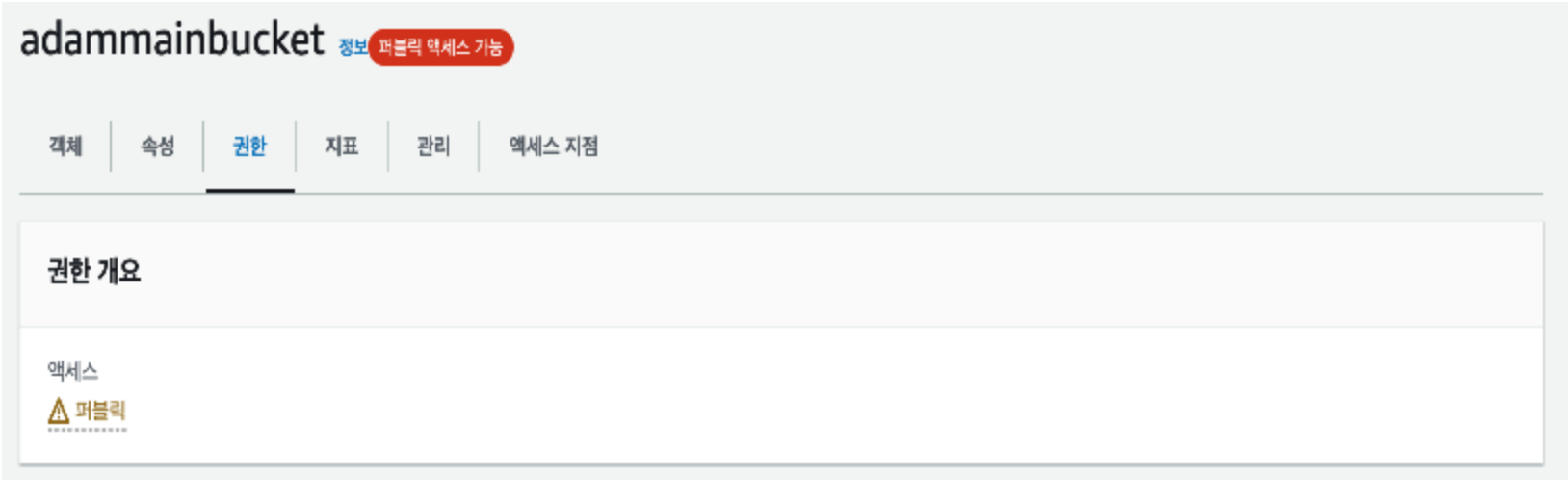
S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 업로드
    - ● 확인



S3

- ❖ Spring Boot 프로젝트
  - ✓ ACL 문제로 업로드가 안되는 경우 버킷의 객체 소유권 변경
    - 버킷의 권한 탭 클릭



S3

- Spring Boot 프로젝트
  - ✓ ACL 문제로 업로드가 안되는 경우 버킷의 객체 소유권 변경
    - 객체 소유권 편집 클릭

S3

- ❖ Spring Boot 프로젝트
  - ✓ ACL 문제로 업로드가 안되는 경우 버킷의 객체 소유권 변경
    - ● 객체 소유권 수정

S3

- Spring Boot 프로젝트
  - ✓ 파일 다운로드
    - 다운로드 될 파일의 속성을 설정할 메서드를 CommonUtils 클래스에 추가

```
private static final int UNDER_BAR_INDEX = 1;

public static ContentDisposition createContentDisposition(String
    categoryWithFileName) {
        String fileName = categoryWithFileName.substring(
                categoryWithFileName.lastIndexOf(CATEGORY_PREFIX) +
    UNDER_BAR_INDEX);
        return ContentDisposition.builder("attachment")
                .filename(fileName, StandardCharsets.UTF_8)
                .build();
}
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 다운로드
    - ● 다운로드 메서드를 Service 클래스에 추가

```
public byte[] downloadFileV1(String resourcePath) {
        boolean result = validateFileExistsAtUrl(resourcePath);
        if(result == false){
            return null;
        }

        S3Object s3Object = amazonS3Client.getObject(bucketName, resourcePath);
        S3ObjectInputStream inputStream = s3Object.getObjectContent();
        try {
            return IOUtils.toByteArray(inputStream);
        } catch (IOException e) {
            return null;
        }
    }
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 다운로드
    - ● 다운로드 메서드를 Service 클래스에 추가

```
private boolean validateFileExistsAtUrl(String resourcePath) {
    boolean result = true;
    if (!amazonS3Client.doesObjectExist(bucketName, resourcePath)) {
        result = false;
    }
    return result;
}
```

S3

- ❖ Spring Boot 프로젝트
  - ✓ 파일 다운로드
    - ● 컨트롤러 클래스에 추가

```
private HttpHeaders buildHeaders(String resourcePath, byte[] data) {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentLength(data.length);
        headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);

headers.setContentDisposition(CommonUtils.createContentDisposition(resourceP
ath));
        return headers;
}
```

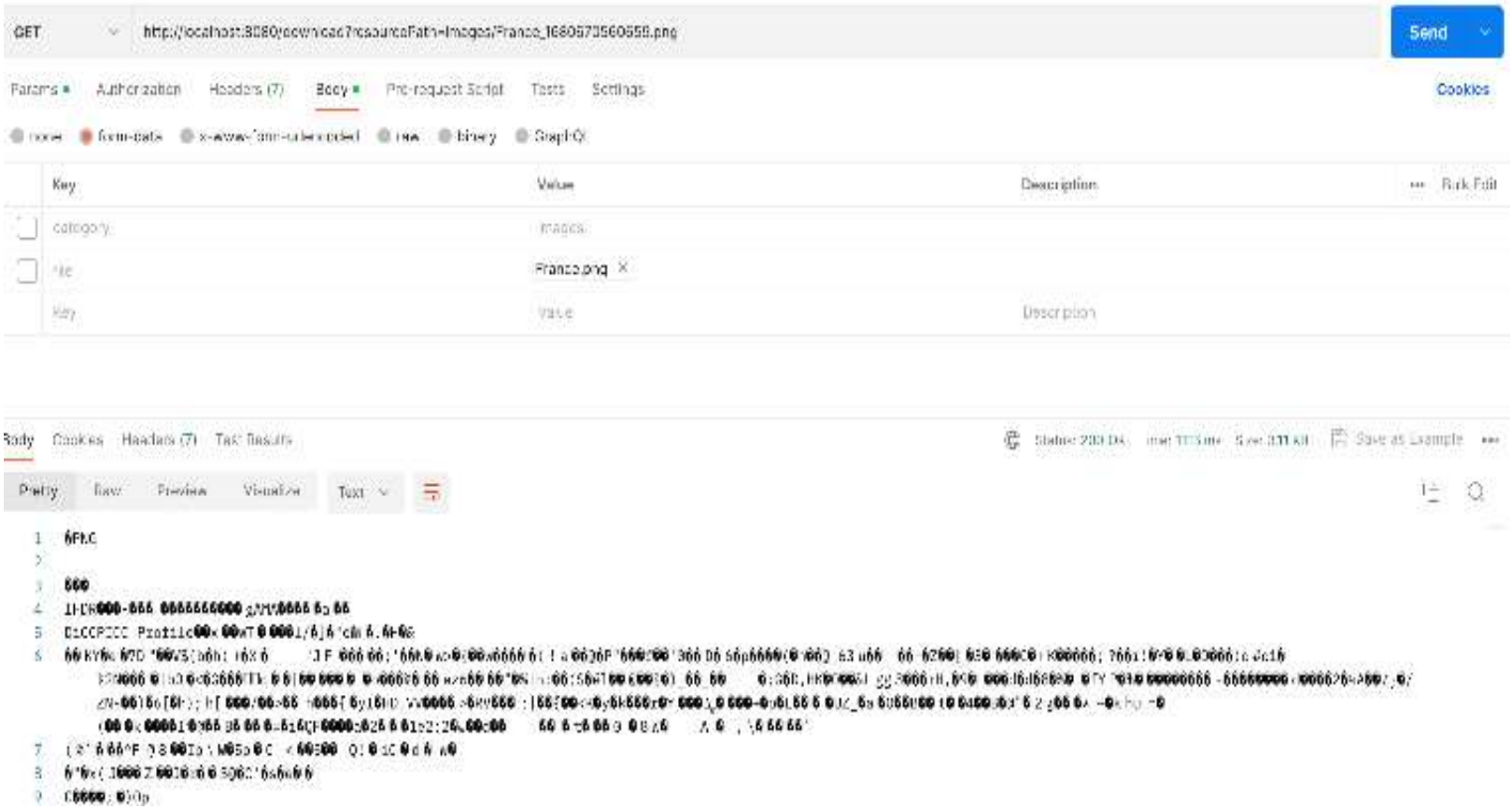
S3

- Spring Boot 프로젝트
  - ✓ 파일 다운로드
    - 컨트롤러 클래스에 추가

```
@GetMapping("/download")
public ResponseEntity<ByteArrayResource> downloadFile(
        @RequestParam("resourcePath") String resourcePath) {
    byte[] data = awsS3Service.downloadFileV1(resourcePath);
    ByteArrayResource resource = new ByteArrayResource(data);
    HttpHeaders headers = buildHeaders(resourcePath, data);

    return ResponseEntity
        .ok()
        .headers(headers)
        .body(resource);
}
```

S3

- Spring Boot 프로젝트
  - ✓ 파일 다운로드
    - 테스트



S3

- Spring Boot 프로젝트
  - ✓ 다중 파일 업로드
    - 서비스 클래스에 메서드 추가

```
public List<String> uploadFile(String category, List<MultipartFile> multipartFiles) {
        List<String> fileUrls = new ArrayList<>();

        for (MultipartFile multipartFile : multipartFiles) {

            String fileName = CommonUtils.buildFileName(category,
    multipartFile.getOriginalFilename());
            ObjectMetadata objectMetadata = new ObjectMetadata();
            objectMetadata.setContentType(multipartFile.getContentType());
```

S3

- Spring Boot 프로젝트
  - ✓ 다중 파일 업로드
    - 서비스 클래스에 메서드 추가

```
            try (InputStream inputStream = multipartFile.getInputStream()) {
                amazonS3Client.putObject(new PutObjectRequest(bucketName,
        fileName, inputStream, objectMetadata)
                        .withCannedAcl(CannedAccessControlList.PublicRead));
                fileUrls.add( bucketName + "/" + fileName);
            } catch (IOException e) {
                log.warn(e.getMessage());
            }
        }

        return fileUrls;
    }
```

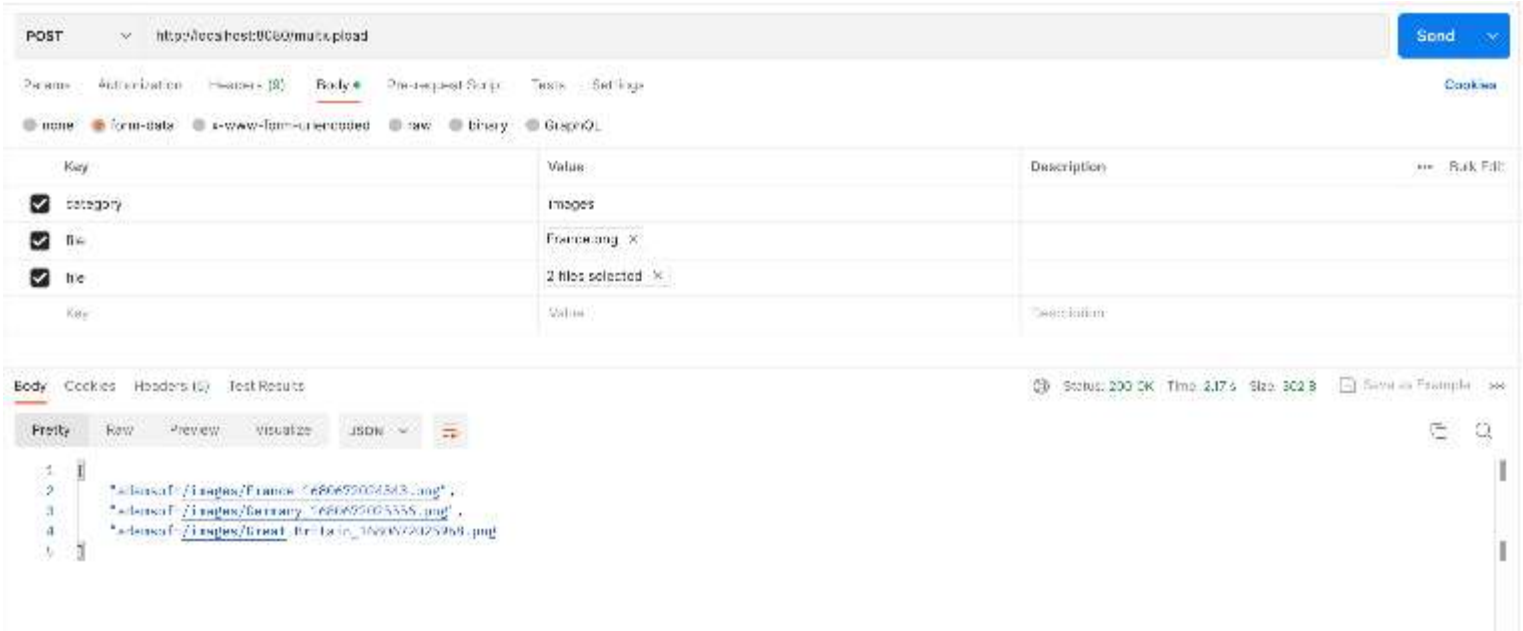
S3

- Spring Boot 프로젝트
  - ✓ 다중 파일 업로드
    - 컨트롤러 클래스에 메서드 추가

```
@PostMapping("/multiupload")
public List<String> uploadFile(
        @RequestParam("category") String category,
        @RequestPart(value = "file") List<MultipartFile> multipartFiles) {
    return awsS3Service.uploadFile(category, multipartFiles);
}
```

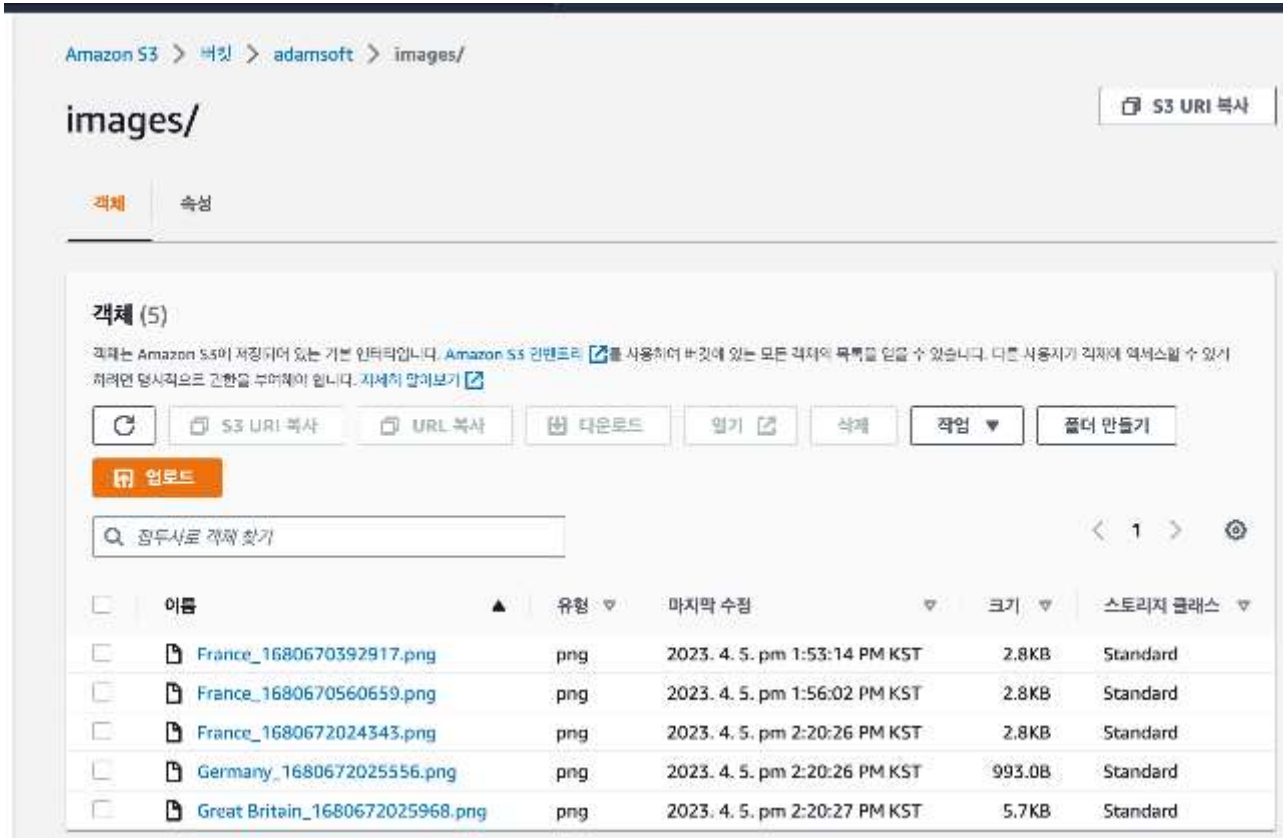
S3

- Spring Boot 프로젝트
  - ✓ 다중 파일 업로드
    - 테스트



S3

- ❖ Spring Boot 프로젝트
  - ✓ 다중 파일 업로드
    - 확인



S3

- react 에서 파일 업로드
  - ✓ 프로젝트 생성: npx create-react-app fileupload
  - ✓ cd fileupload
  - ✓ 라이브러리 설치: npm install aws-sdk

S3

- react 에서 파일 업로드
  - ✓ App.js 파일에 코드 작성

```
import React ,{useState} from 'react';
import AWS from 'aws-sdk'

const S3_BUCKET ='adamsoft';
const REGION ='us-east-1';

AWS.config.update({
    accessKeyId: 'AKIAWN5SS7IWEYL55SWG',
    secretAccessKey: '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'
})

const myBucket = new AWS.S3({
    params: { Bucket: S3_BUCKET},
    region: REGION,
})
```

S3

- react 에서 파일 업로드
  - ✓ App.js 파일에 코드 작성

```
function App() {
  const [progress , setProgress] = useState(0);
    const [selectedFile, setSelectedFile] = useState(null);

    const handleFileInput = (e) => {
        setSelectedFile(e.target.files[0]);
    }

    const uploadFile = (file) => {

        const params = {
            ACL: 'public-read',
            Body: file,
            Bucket: S3_BUCKET,
            Key: file.name
        };
```

S3

- react 에서 파일 업로드
  - ✓ App.js 파일에 코드 작성

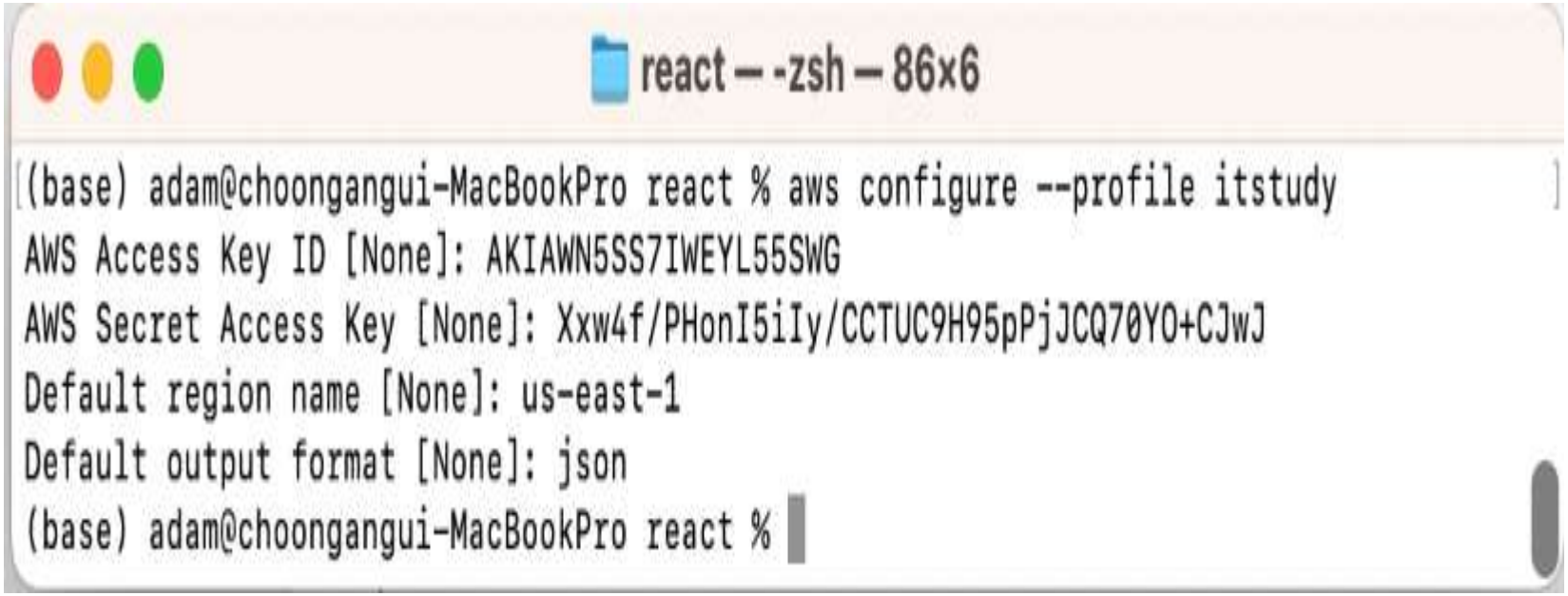
```
            myBucket.putObject(params)
                .on('httpUploadProgress', (evt) => {
                    setProgress(Math.round((evt.loaded / evt.total) * 100))
                })
                .send((err) => {
                    if (err) console.log(err)
                })
        }

      return <div>
            <div>Native SDK File Upload Progress is {progress}%</div>
            <input type="file" onChange={handleFileInput}/>
            <button onClick={() => uploadFile(selectedFile)}> Upload to S3</button>
        </div>
    }

    export default App;
```

S3

- react 에서 파일 업로드
  - ✓ 프로젝트 실행 후 확인: npm start

정적 웹 사이트 CI/CD

- 정적 웹 사이트 호스팅
  - ✓ AWS Command Interface 설치
    - https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/getting-started-install.html
  - ✓ 확인: aws --version
  - ✓ 유저 추가



정적 웹 사이트 CI/CD

- 정적 웹 사이트 호스팅
  - ✓ S3 Bucket 설정 – 속성 탭에서 수정

정적 웹 사이트 CI/CD

- 정적 웹 사이트 호스팅
  - ✓ react 프로젝트 생성 - npx create-react-app staticweb
  - ✓ 프로젝트 빌드 – npm run build
  - ✓ 업로드 - aws s3 sync ./build s3://Bucket이름 --profile=계정명
    - package.json 파일의 scripts에 추가하면 yarn deploly 나 npm run deploy로 배포 가능

      "deploy": "aws s3 sync ./build s3://Bucket이름 --profile=계정이름"
  - ✓ S3의 엔드 포인트에 접속해서 확인

정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● 프로젝트 내에 .github/workflows/?.yml

```
name: CI/CD hdev_client to AWS S3

on:
  push:
    branches:
      - main
```

정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● 프로젝트 내에 .github/workflows/?.yml

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: 코드 체크아웃
        uses: actions/checkout@v3

      - name: AWS IAM 사용자 설정
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_REGION }}
```

정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● 프로젝트 내에 ./github/workflows/?.yml

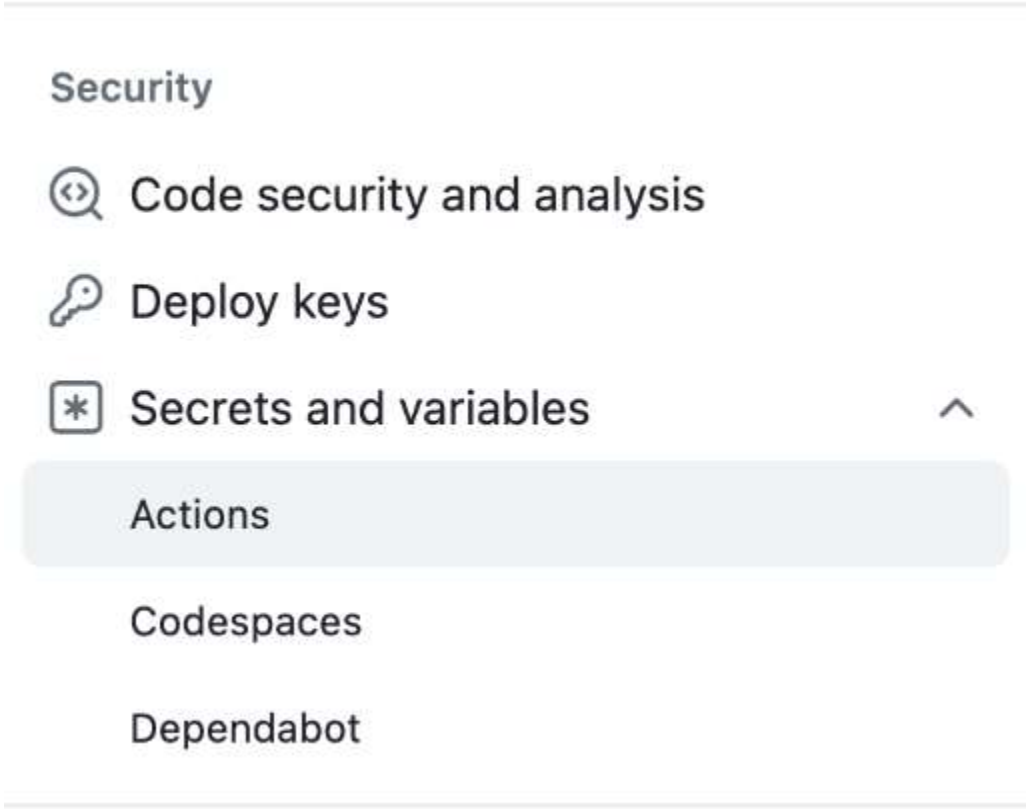
```
- name: 리액트 빌드
  run: |
    npm install
    npm run build

- name: 빌드한 파일 S3에 업로드
  run: aws s3 sync build/ s3://${{ secrets.AWS_S3_BUCKET }} --acl public-read
  env:
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - Git Hub 의 Settings 에 Secret Key 등록

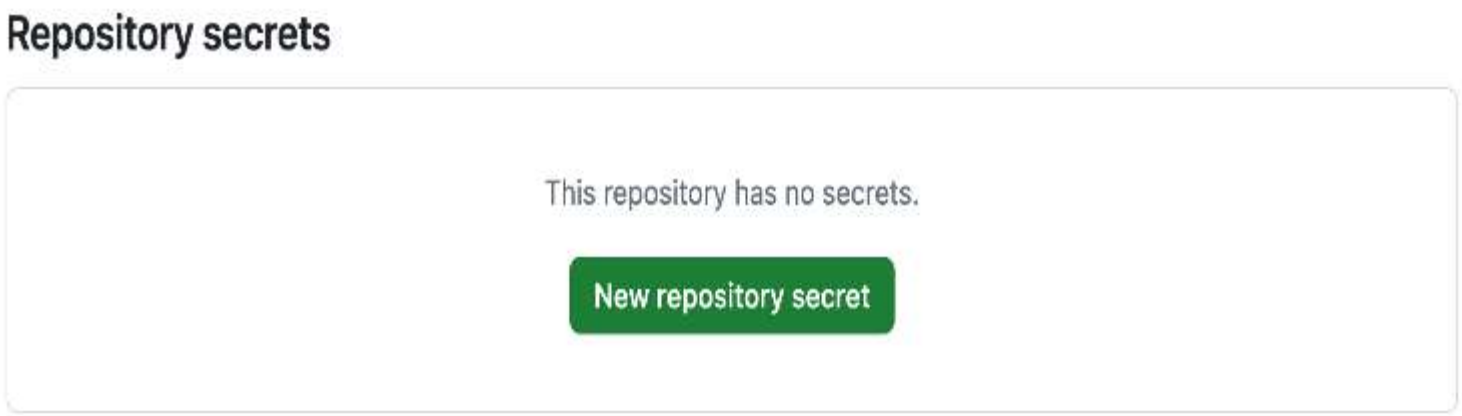
정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● Git Hub 의 Settings 에 Secret Key 등록



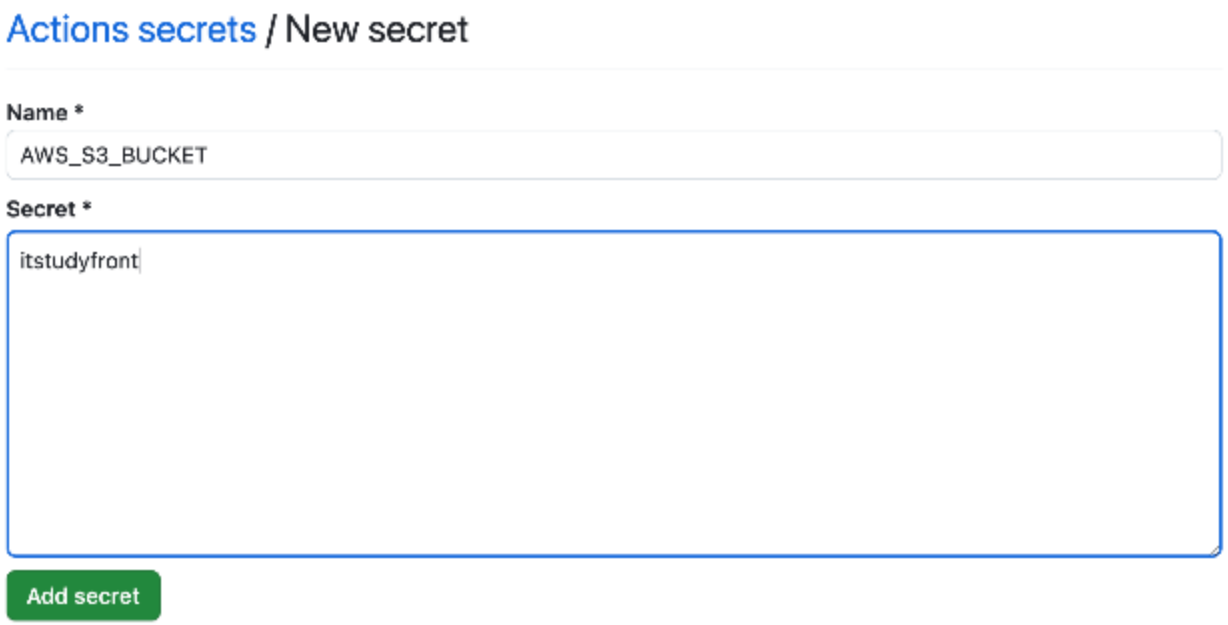
정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - Git Hub 의 Settings 에 Secret Key 등록



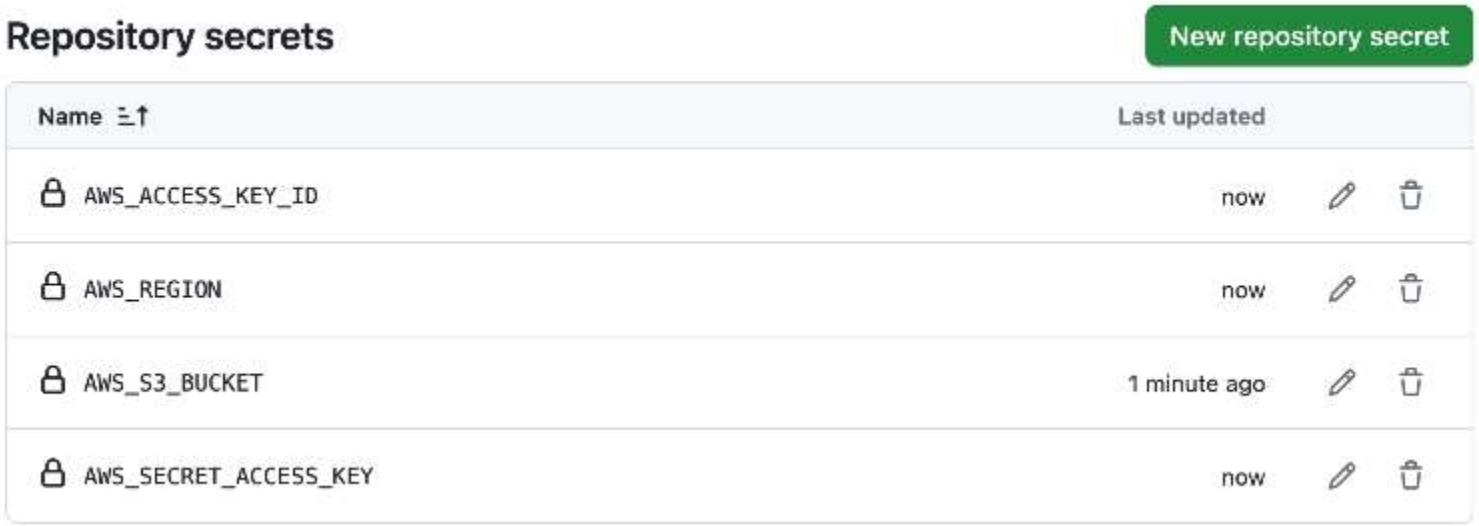
정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● Git Hub 의 Settings 에 Secret Key 등록



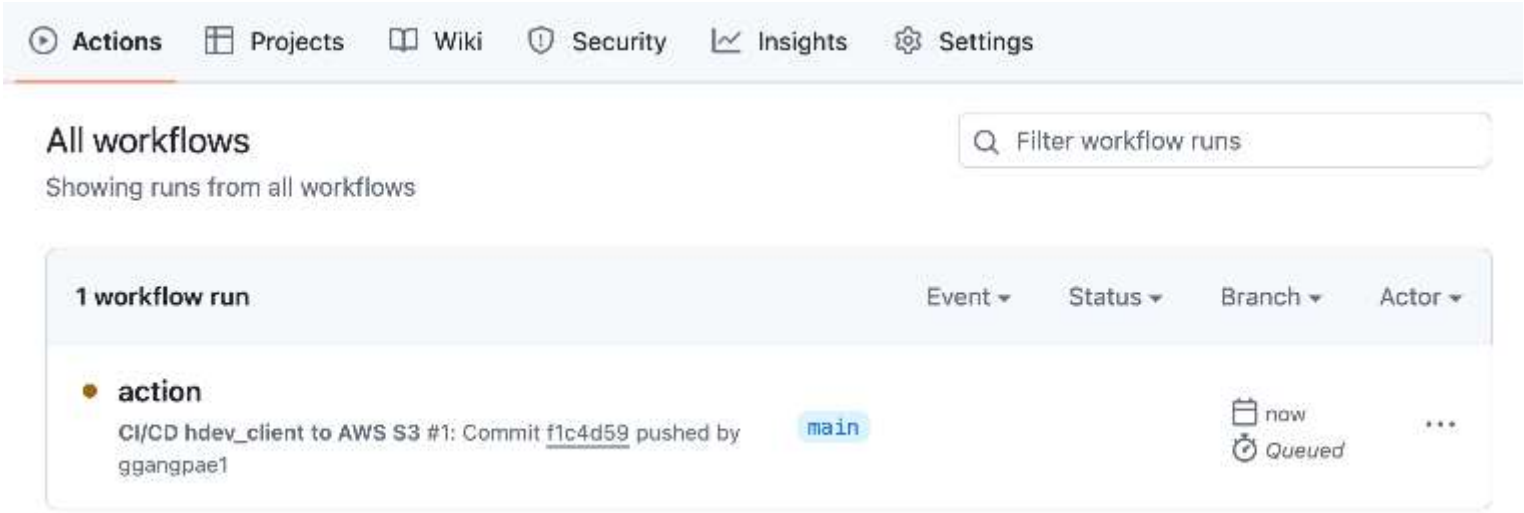
정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● Git Hub 의 Settings 에 Secret Key 등록



정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● 코드를 업로드하고 Git Hub에서 Action 확인



정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● ACL 문제가 발생하면 버킷의 객체 소유권 변경

정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● app.js 코드를 수정하고 다시 업로드하고 반영되었는지 확인

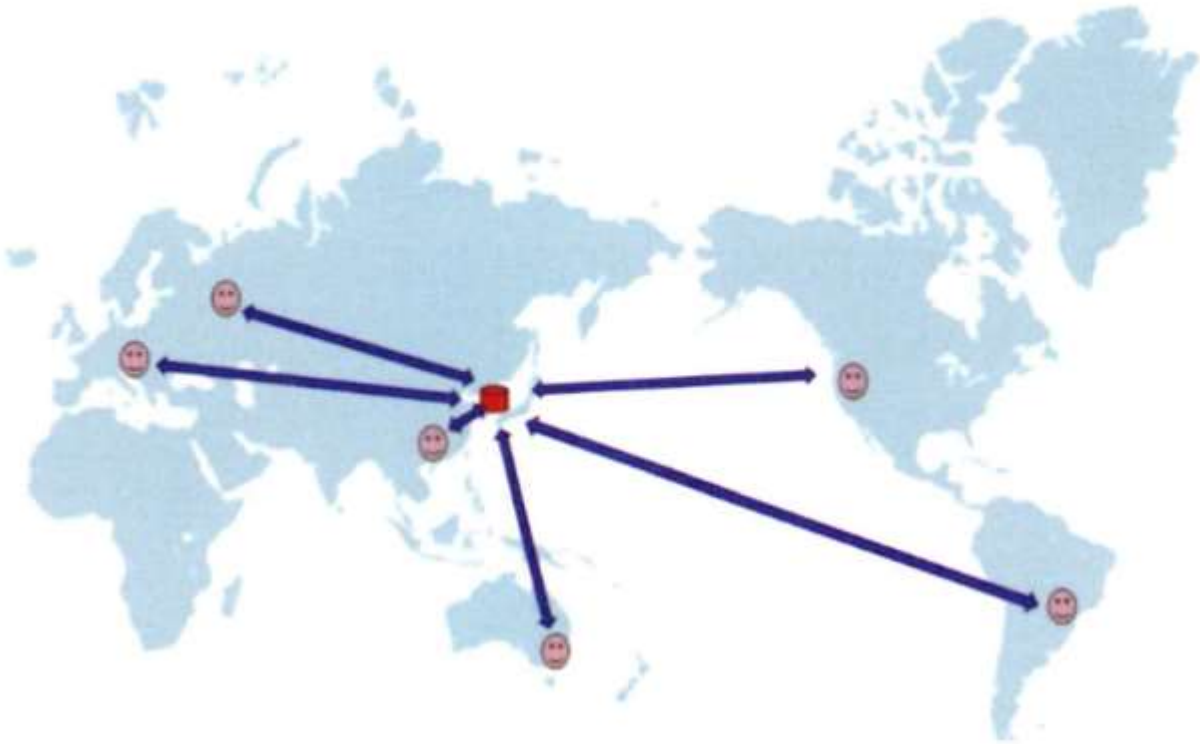
```
import React, {Fragment} from 'react';

function App() {
  return (
    <Fragment>
      <h1>70-79뉴프렌즈</h1>
    </Fragment>
  );
}
export default App;
```

정적 웹 사이트 CI/CD

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 웹 페이지 호스트 와 네트워크 서버 설정을 위한 서비스
    - ● CDN
      - ◆ 콘텐츠 전송 네트워크(Content Delivery Network)는 리소스 요청이 발생했을 때 사용자가 현재 어디에서 요청하는지에 따라 리소스를 전달해주는 분산 네트워크 시스템(Distributed Network System)
      - ◆ CDN의 목적은 Edge Location을 통해 사용자에게 최소한의 지연 시간과 최고의 성능으로 콘텐츠를 보여주기 위함
        - ▪ 최초로 웹사이트 호스팅이 이루어지고 있는 곳을 Origin이라고 하는데 이 웹사이트는 전 세계에서 다양한 사용자가 방문
        - ▪ 어떤 지역에서 사용자가 웹사이트 요청을 하는 경우 Origin에서 사용자 요청에 응답하여 콘텐츠를 전달하면 문제가 발생할 수 있음
        - ▪ 사용자가 Origin Server에서 멀어진다면 콘텐츠를 받는 데 지연 시간(latency)이 발생
        - ▪ 많은 사용자가 동시 다발적으로 요청하게 된다면 지연 시간은 더 길어지는데 이런 문제를 해결하기 위해서 CDN이 등장

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● CDN



S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● CDN
      - ◆ CDN은 많은 요청이 오가는 지역을 근거로 Edge Location을 생성
      - ◆ Edge Location은 Origin Server에서 가지고 있는 콘텐츠 복사본을 가지고 있으며 콘텐츠 요청을 전달받을 때 Origin Server에서 콘텐츠를 전달하는 것이 아니라 사용자와 가장 가까운 Edge Location을 통해 콘텐츠를 전송

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● Edge Location
      - ◆ Edge Location은 CloudFront CDN에 의해 전 세계 곳곳에 생성
      - ◆ 각각의 Edge Location은 전 세계 사용자에게 최소한의 지연 시간을 발생시키기 위해 존재
      - ◆ CloudFront에 Contents Server가 Hosting 되면 사용자의 요청을 받을 시 사용자가 위치한 가장 가까운 Edge Location을 참조하여 콘텐츠를 전달
      - ◆ Edge Location은 Origin Server에 있는 콘텐츠 원본을 캐시에 보관
      - ◆ 사용자가 웹사이트를 요청할 시 캐시에 요청 정보가 있지 않다면 일차적으로 Origin Server에서 콘텐츠를 가져온 후 캐시에 보관하여 사용자에게 전달
      - ◆ 추후 사용자가 똑같은 요청을 할 시 Origin Server가 아닌 Cache로부터 콘텐츠를 전달받기 때문에 속도가 빠른 장점을 가지고 있음

S3

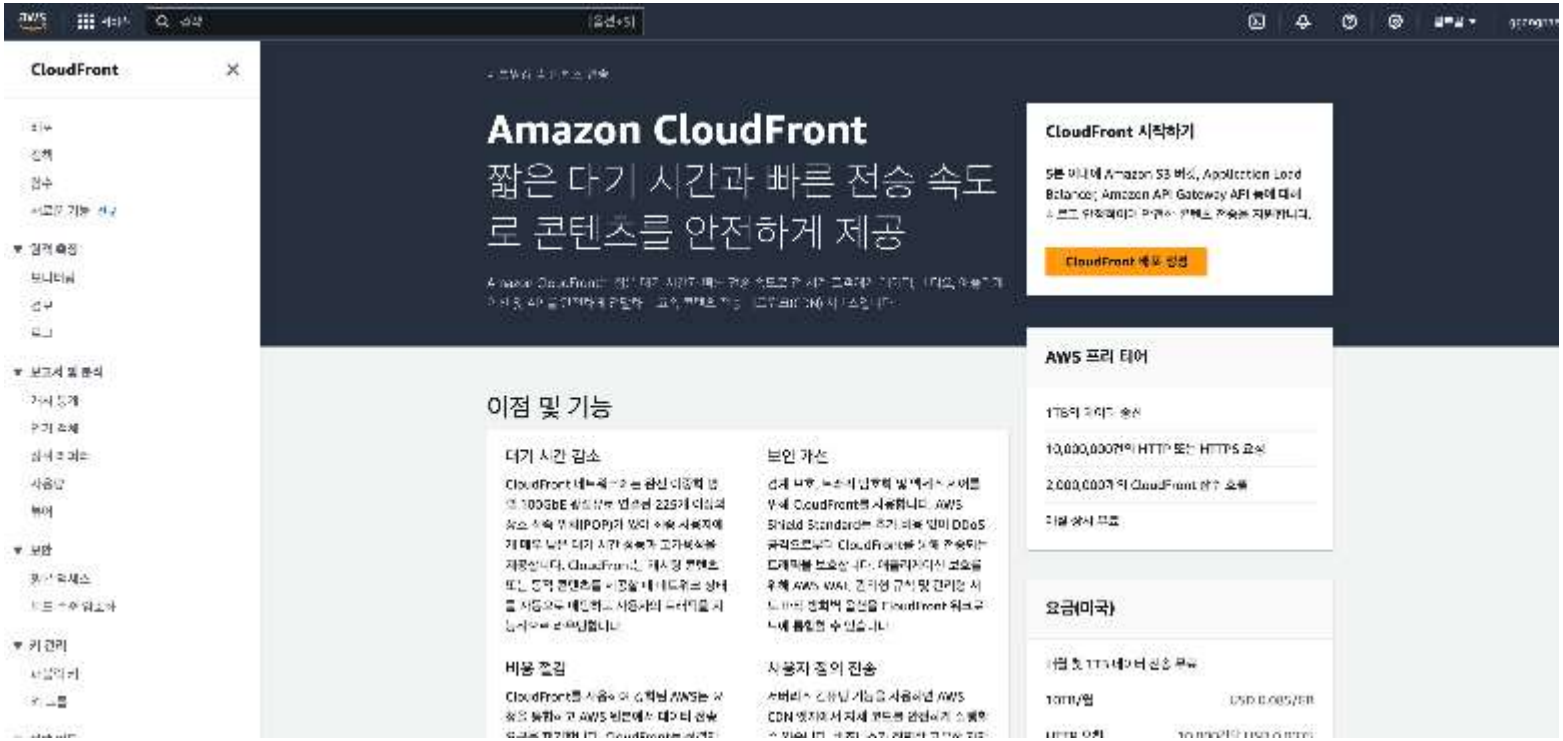
- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● Origin Server
      - ◆ 최초 웹 서버가 구현되고 돌아가고 있는 곳이 Origin Server
      - ◆ 웹 페이지를 꾸미기 위해 필요한 HTML. CSS, JavaScript 파일이 있는 S3 Bucket이나 EC2 인스턴스가 Origin Server
      - ◆ CloudFront는 Origin Server를 통해 콘텐츠를 전달받고 사용자에게 전달.
      - ◆ S3 Bucket 단독으로 웹사이트 호스팅을 하게 되면 정적 웹사이트 호스팅만 가능하지만 CloudFront를 사용하여 동적 웹사이트 호스팅을 구현할 수 있음

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 장점
      - ◆ Edge Location을 만들고 캐시에 원본 콘텐츠를 담고 있기 때문에 사용자에게 콘텐츠를 전달할 때 처리량이 줄고 지연 시간이 거의 없음
      - ◆ http 와 https를 모두 지원하기 때문에 보안이 뛰어남
      - ◆ 데이터를 전송할 때만 비용이 청구되기 때문에 웹사이트 호스팅 시 저렴

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - 설정 - Cloud Front 배포 생성을 선택



S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 설정 - Cloud Front 배포 생성을 선택
      - ◆ 원본 도메인 설정
        - ▪ 원본 도메인: 오리진을 어떤 것을 선택할 지 설정
        - ▪ 원본 경로: 하위 디렉토리가 존재하는 경우 선택
        - ▪ Bucket을 선택하게 되면 OAI 옵션을 설정할 수 있음

S3

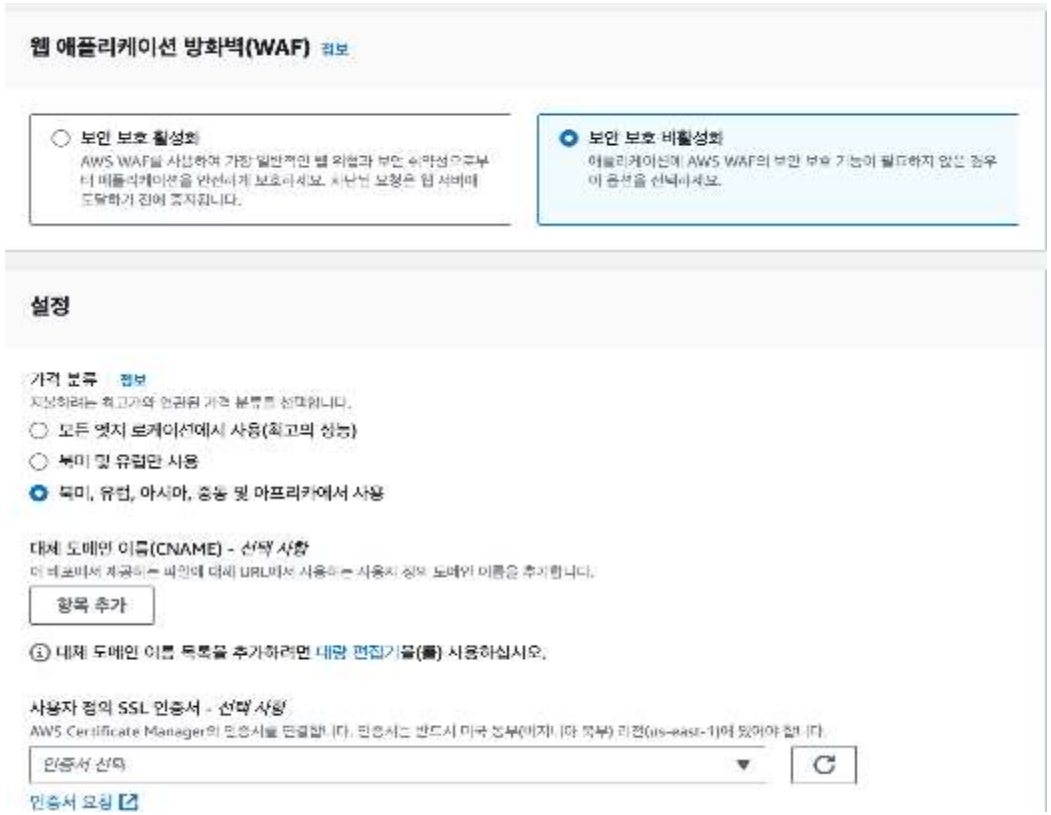
- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 설정 - Cloud Front 배포 생성을 선택
      - ◆ 뷰어 프로토콜 정책 수정

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 설정 - Cloud Front 배포 생성을 선택
      - ◆ 캐시 정책 설정
        - ▪ 처음 등장하는 경로 패턴은 수정할 수 없음
        - ▪ 자동으로 객체 압축은 콘텐츠를 오리진에서 전송 시 압축해서 보낼 지에 대한 여부를 정의
        - ▪ 뷰어 프로토콜 정책에는 HTTP와 HTTPS를 모두 허용할지 HTTP를 HTTPS로 재 전송할지 HTTPS만 허용할지 총 세 가지 선택 사항이 있음
        - ▪ 일반적으로 HTTPS가 HTTP보다 보안이 훨씬 뛰어난데 SSL(Secure Socket Layer) 인증서를 통해 서버에서 브라우저로 정보가 안전하게 전달될 수 있음
        - ▪ 허용된 HTTP 방법을 설정

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 설정 - Cloud Front 배포 생성을 선택
      - ◆ 방화벽 과 가격 분류 설정

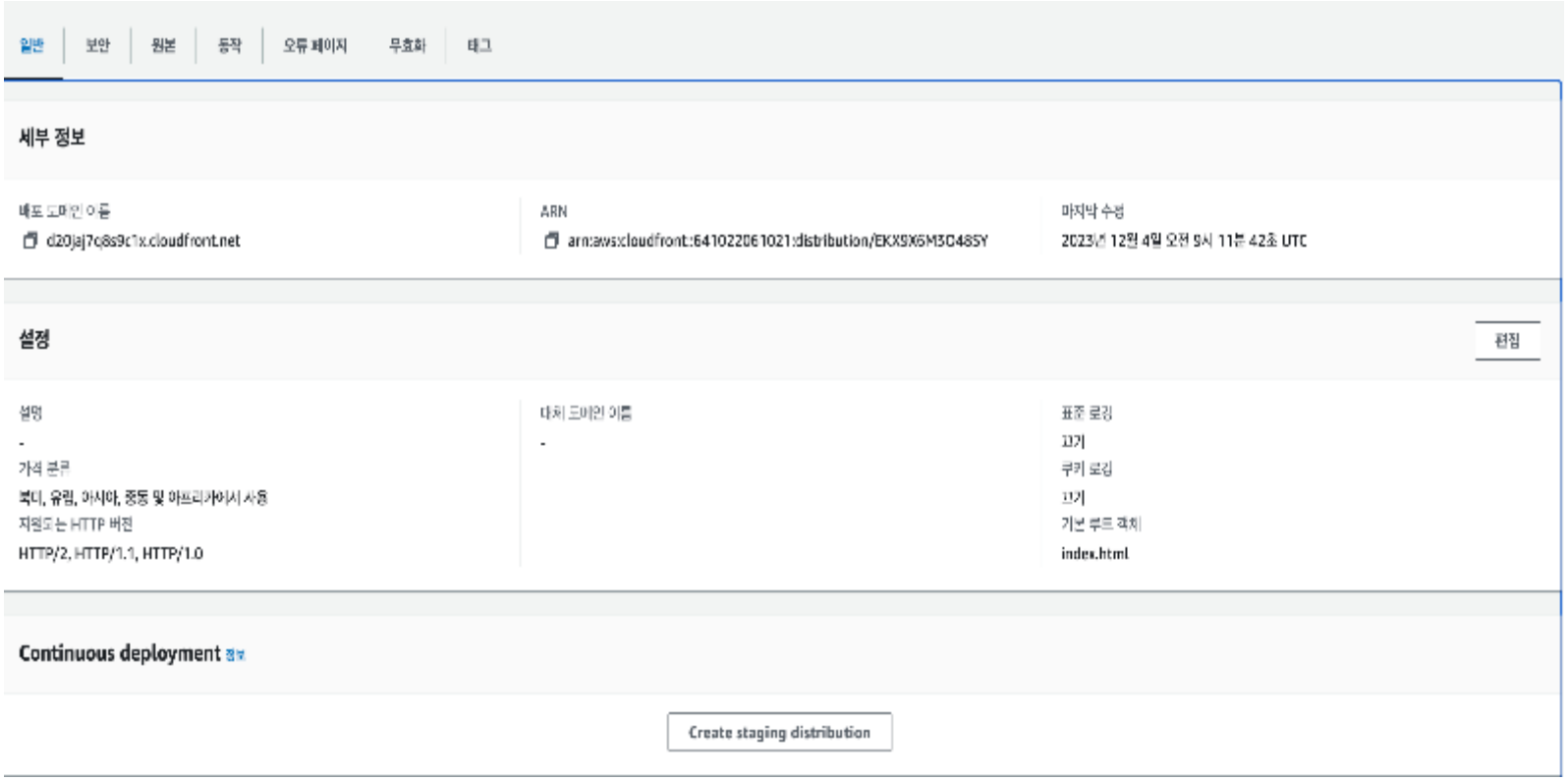


S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - ● 설정 - Cloud Front 배포 생성을 선택
      - ◆ 기본값 루트 객체 설정

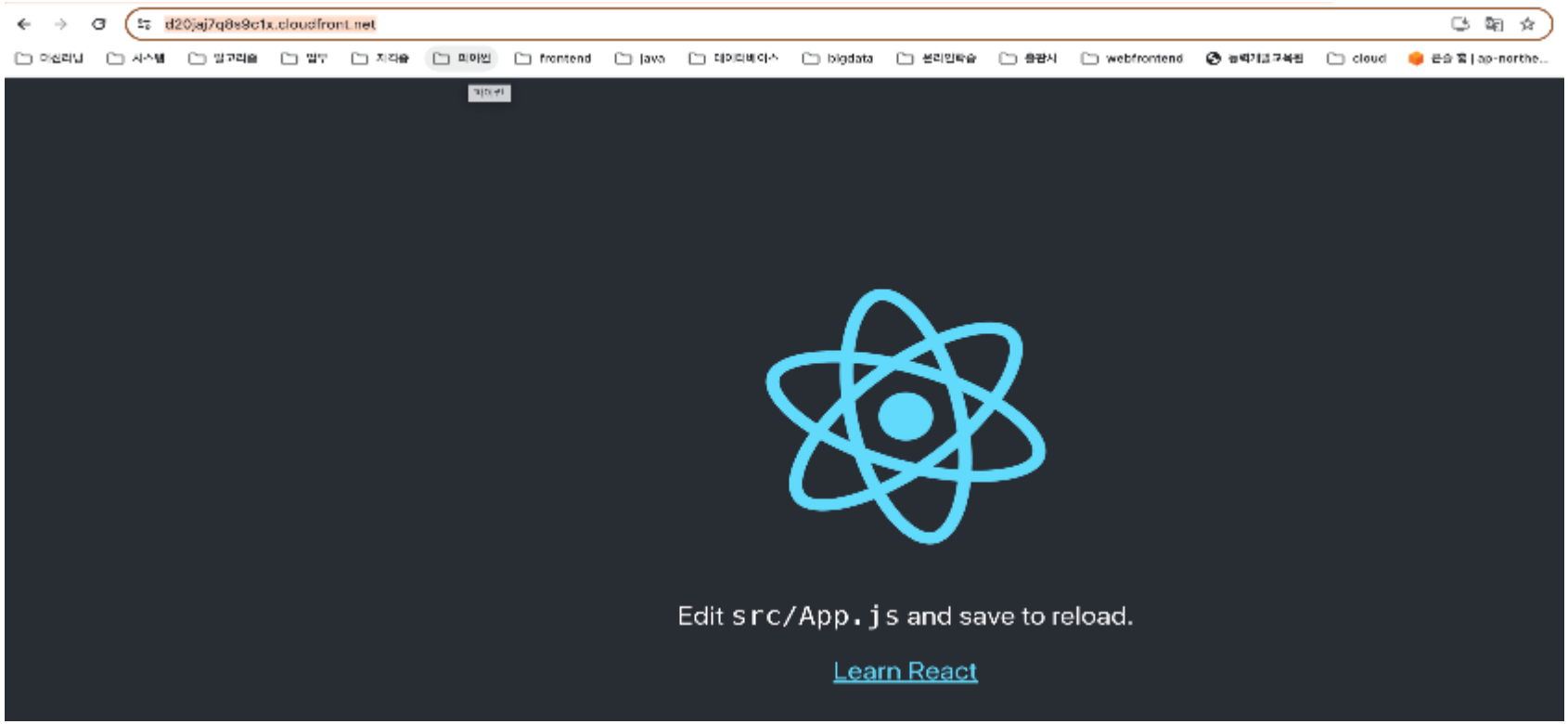
S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - 확인: CloudFront의 도메인 확인



S3

- 정적 웹 사이트 호스팅
  - ✓ CloudFront
    - 확인: 브라우저에서 확인



S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - ● 프로젝트 내에 ./github/workflows/?.yml

```
- name: CloudFront 캐시 무력화 설정
  uses: chetan/invalidate-cloudfront-action@v2
  env:
    DISTRIBUTION: ${{ secrets.AWS_CLOUDFRONT_ID }}
    PATHS: "/*"
    AWS_REGION: us-east-1
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ Git Hub Action 을 이용한 CICD 구현
    - AWS_CLOUDFRONT_ID 키를 추가하고 Cloud Front ID 추가
    - IAM 사용자에게 CloudFront 권한 추가

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 생성
    - AWS Certificate Manager 서비스에 접속해서 인증서 요청을 눌러서 인증서 생성

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 생성
    - AWS Certificate Manager 서비스에 접속해서 인증서 요청을 눌러서 인증서 생성

S3

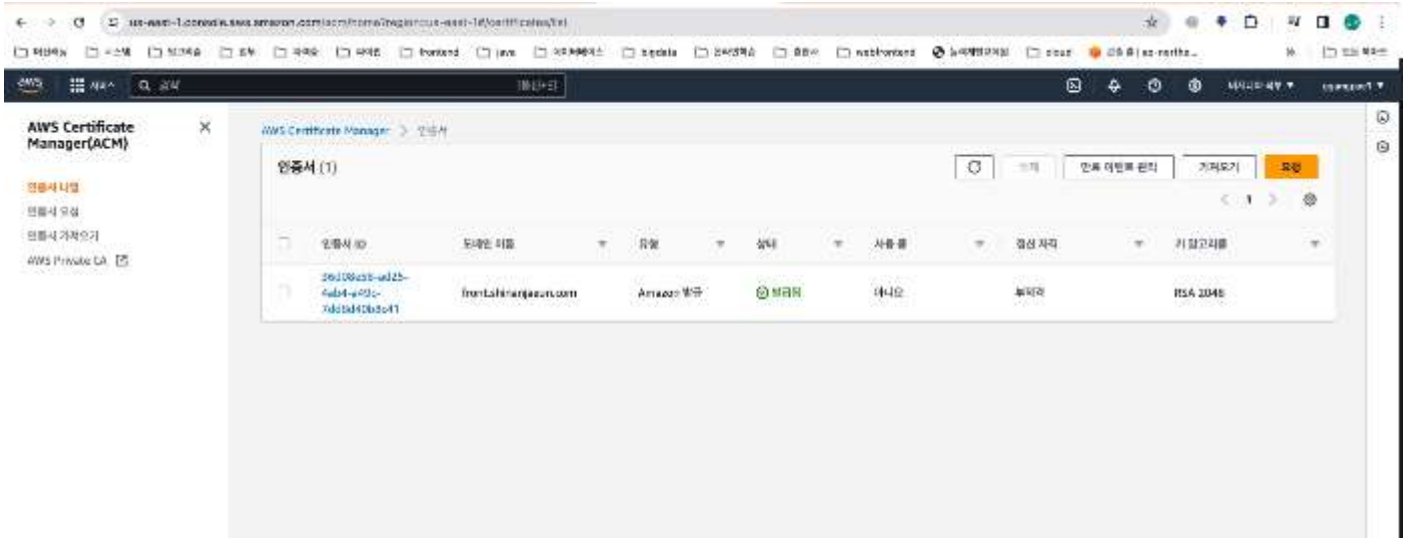
- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 생성
    - AWS Certificate Manager 서비스에 접속해서 인증서 요청을 눌러서 인증서 생성

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 생성
    - AWS Certificate Manager 서비스에서 인증서를 선택하고 인증서 검증 – Route53에서 레코드 생성 클릭

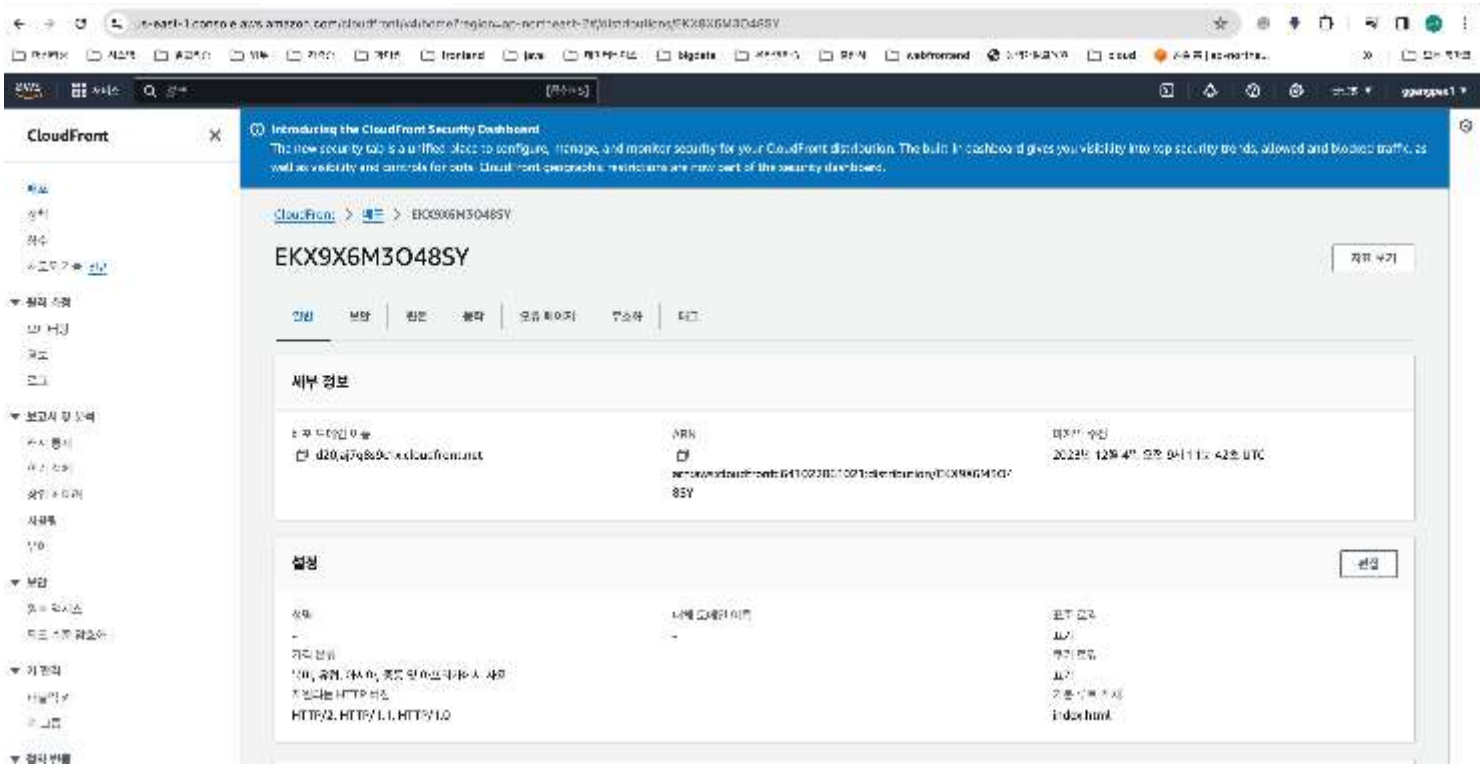
S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 생성
    - ● 인증서 생성 확인



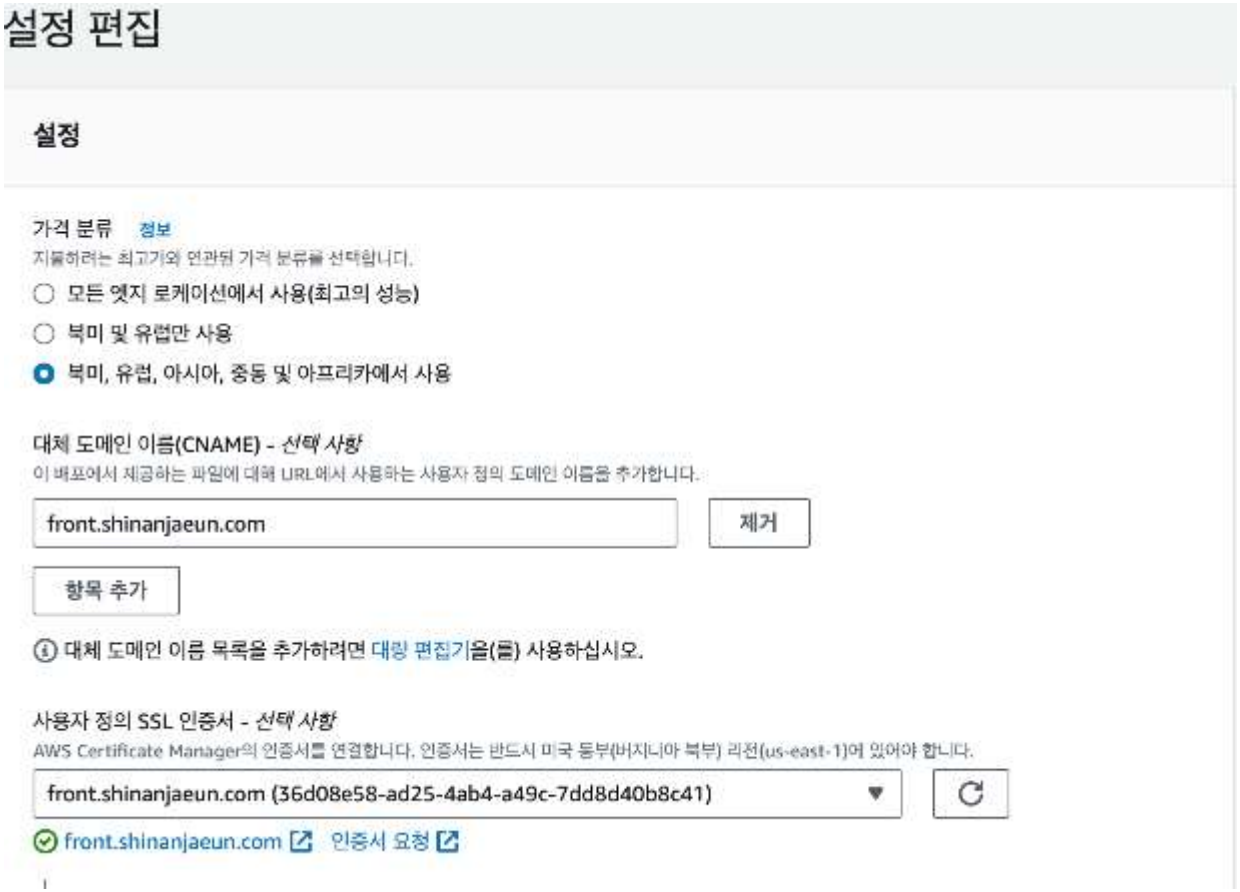
S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 와 Cloud Front 의 배포 연결
    - Cloud Front에서 배포된 곳에 접속해서 [일반] – [설정] – [편집] 클릭



S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ https 인증서 와 Cloud Front 의 배포 연결
    - Cloud Front에서 배포된 곳에 접속해서 [설정] – [편집]

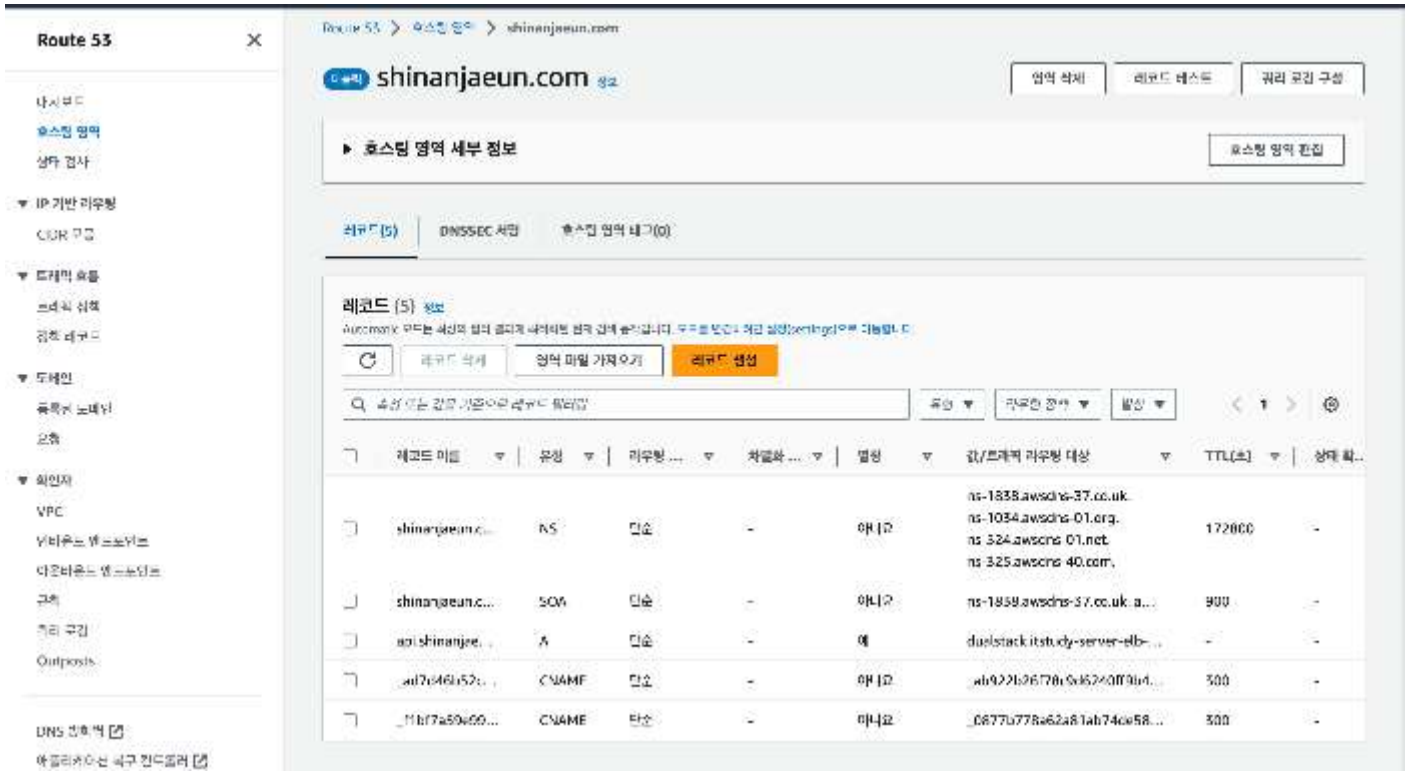


S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ Route53에서 도메인 과 Cloud Front 연결

S3

- ❖ 정적 웹 사이트 호스팅
  - ✓ Route53에서 도메인 과 Cloud Front 연결



S3

❖ 정적 웹 사이트 호스팅

✓ Route53에서 도메인 과 Cloud Front 연결