

# Amazon Simple Storage Service(S3)

## 1. S3에 대하여

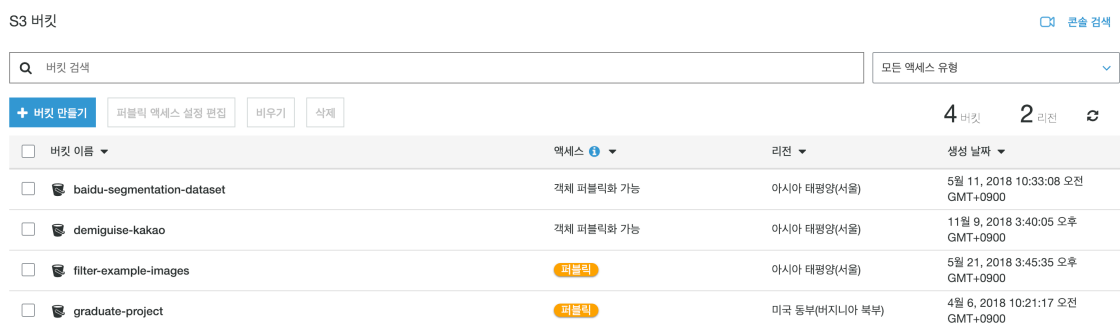
아래 대부분 내용들은 [Amazon Simple Storage Service 안내서](#)를 바탕으로 구성

- 개요

S3는 99.99999999%의 내구성을 실현한 **스토리지**다. Stoarge는 컴퓨터에 있는 File System과 유사한 것으로, 여러가지 파일들, 즉 비정형화된 데이터(영상, 음악, pdf파일 등 **일반적 파일**)을 저장하고 관리하는 데에 특화된 서비스를 지칭한다. S3는 REST API와 같은 단순한 **웹 서비스 인터페이스**를 사용한다. 저장한 데이터 양에 대한 비용도 저렴하고, 저장할 수 있는 데이터 양이 무한하다. S3는 FTP(file transfer server) 서버와 같이 단순히 파일 저장 영역으로도 사용할 수 있으며, 다양한 AWS 서비스와 연동해서 쓸 수 있다. 그리고 재미있는 것 중 하나가 이것 만으로도 **정적 웹 사이트 호스팅**도 가능하다.

- S3을 다룰 때 내부 개념

- 버킷(Bucket)



The screenshot shows the Amazon S3 console interface. At the top, there's a search bar and a dropdown for '모든 액세스 유형'. Below that, there are buttons for '+ 버킷 만들기', '퍼블릭 액세스 설정 편집', '비우기', and '삭제'. The main table lists several buckets:

버킷 이름	액세스	리전	생성 날짜
baidu-segmentation-dataset	객체 퍼블릭화 가능	아시아 태평양(서울)	5월 11, 2018 10:33:08 오전 GMT+0900
demiguise-kakao	객체 퍼블릭화 가능	아시아 태평양(서울)	11월 9, 2018 3:40:05 오후 GMT+0900
filter-example-images	퍼블릭	아시아 태평양(서울)	5월 21, 2018 3:45:35 오후 GMT+0900
graduate-project	퍼블릭	미국 동부(버지니아 북부)	4월 6, 2018 10:21:17 오전 GMT+0900

버킷은 S3에 저장된 객체에 대한 컨테이너입니다. 모든 객체는 특정 버킷에 포함됩니다.

example)

photos/puppy.jpg 로 명명된 객체는 johnsmith 버킷에 저장되며, 다음 URL을 사용하여 주소를 지정할 수 있습니다. `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`

=> 기본적으로 파일 시스템과 같이, 디렉토리 형태로 객체를 관리한다.

버킷은 가장 높은 수준에서 Amazon S3 네임스페이스를 구성하고, 스토리지 및 데이터 전송 요금에 대한 책임이 있는 계정을 식별하며, 액세스 제어에서 역할을 하고, 사용량 보고에 대한 집계 단위로 사용되는 등 여러 목적으로 사용됩니다.

- 객체(Object)



The screenshot shows the Amazon S3 console interface for a specific bucket. At the top, there's a search bar and a dropdown for '아시아 태평양(서울)'. Below that, there are buttons for '업로드', '+ 폴더 만들기', '다운로드', and '작업'. The main table lists several objects:

이름	마지막 수정	크기	스토리지 클래스
baidu-segmentation	--	--	--
crawled_selfie	--	--	--
detection_dataset	--	--	--
dataset.h5	8월 15, 2018 8:21:54 오후 GMT+0900	1.2 GB	스탠다드
train.csv	7월 8, 2018 10:47:13 오후 GMT+0900	74.2 KB	스탠다드

객체는 S3에 저장되는 기본 개체입니다. 객체는 객체 데이터와 메타데이터로 구성됩니다. 데이터 부분은 S3에서 볼 수 없습니다. 메타데이터는 객체를 설명하는 이름-값 페어의 집합입니다. 여기에는 마지막으로 수정한 날짜와 같은 몇 가지 기본 메타데이터 및 콘텐츠 형식과 같은 표준 HTTP 메타데이터가 포함됩니다. 객체를 저장할 때 사용자 정의 메타데이터를 지정할 수도 있습니다.

The screenshot shows the AWS S3 console interface. On the left, a list of objects in the 'dataset.h5' bucket is displayed. The 'dataset.h5' object is selected. On the right, a detailed view of the 'dataset.h5' object is shown, including its key, size, last modified date, and various metadata fields.

dataset.h5	
다운로드	정로 복사
Select 소스	
최신 버전	
개요	<p>키: dataset.h5</p> <p>크기: 1.2 GB</p> <p>만료 날짜: N/A</p> <p>만료 규칙: N/A</p> <p>ETag: 718187eb3ecb6f5d9324876f7d5e5834-74</p> <p>마지막 수정: 8월 15, 2018 8:21:54 오후 GMT+0900</p> <p>객체 URL: https://s3.ap-northeast-2.amazonaws.com/baidu-segmentation-dataset/data/dataset.h5</p>
속성	<p>스토리지 클래스: 스탠다드</p> <p>암호화: 없음</p> <p>메타데이터: 1</p> <p>태그: 0 태그</p> <p>객체 잠금: 비활성</p>
권한	<p>소유자: 객체</p> <p>읽기: 1 피부여자</p> <p>쓰기: 1 피부여자</p> <p>객체 권한: 읽기: 1 피부여자, 쓰기: 1 피부여자</p>

## ○ 키(key)

키는 버킷 내 객체의 고유한 식별자입니다. 버킷 내 모든 객체는 **정확히 하나의 키**를 갖습니다. 버킷, 키 및 버전 ID의 조합이 각 객체를 고유하게 식별하기 때문에 S3는 "버킷 + 키 + 버전"과 객체 자체 사이의 기본 데이터 맵으로 간주할 수 있습니다.

## ● 주요 기능들

### 1. REST API 제공

- 버킷 만들기(C) : 객체를 저장할 고유한 버킷을 만들고 여기에 이름을 지정
- 객체 작성(C&U) : 객체를 만들거나 덮어써서 데이터를 저장.
- 객체 읽기(R) : 데이터를 읽어옴
- 객체 삭제(D) : 일부 데이터를 삭제
- 키 나열 : 버킷 중 하나에 포함된 키를 나열.

### 2. 스토리지 클래스

- standard : 기본 스토리지 클래스
- Stadard\_IA & onezone-IA : 자주 액세스하지 않지만, 빨리 가져와야 하는 데이터들 ( standard 보다 저장 비용은 싸고(1/2배), 가져오는 비용은 비쌈(10배))
- Intelligent\_tiering : 자동으로 트래픽을 파악하여, 자동 비용 절감 효과를 제공
- glacier : 데이터 액세스가 잦지 않은 데이터의 보관에 적합. 매우 싸지만 실시간으로 액세스가 불가능

스토리지 클래스	다음으로 설계됨	가용성 (설계상)	가 용 영 역	최 소 스 토 리 지 기 간	최소 요 금 객체 크기	기타 고려 사항
STANDARD	자주 액세스하는 데이터	99.99%	>= 3	없음	없음	없음
STANDARD_IA	수명이 길고 자주 액세스하지 않는 데이터	99.9%	>= 3	30 일	128KB	GB당 검색 요금이 적용됩니다.
INTELLIGENT_TIERING	변경 또는 알 수 없는 액세스 패턴으로 수명이 긴 데이터	99.9%	>= 3	30 일	없음	객체당 모니터링 및 자동화 비용이 적용됩니다. 검색 요금이 없습니다.
ONEZONE_IA	수명이 긴 데이터에 자주 액세스하지 않는 중요하지 않은 데이터	99.5%	1	30 일	128KB	GB당 검색 요금이 적용됩니다. 가용 영역의 손실에 대한 복원력이 없습니다.
GLACIER	분에서 시간 단위로 검색 시간을 지원하는 장기 데이터 보관	99.99% (객체 복원 후)	>= 3	90 일	없음	GB당 검색 요금이 적용됩니다. 이 객체에 액세스하려면 먼저 보관된 객체를 복원해야 합니다. 자세한 내용은 <a href="#">보관된 객체의 복원</a> 단원을 참조하십시오.

### 3. 객체 버전 관리

버전 관리를 통해 한 버킷 내에 여러 개의 객체 버전을 유지할 수 있습니다. 예를 들어, 하나의 버킷에 `my-image.jpg` (버전 111111)와 `my-image.jpg` (버전 222222)를 저장할 수 있을 것입니다. 버전 관리를 사용하면 의도하지 않은 덮어쓰기 및 삭제의 결과로부터 보호를 받을 수 있습니다. 또, 버전 관리를 사용해 객체를 보관함으로써 이전 버전에 액세스할 수 있습니다.

## 2. 다른 경쟁사에서 제공하고 있는 동일한 서비스와의 공통점 및 차이점

- Reference :

- [Deep dive on AWS vs. Azure vs. Google cloud storage options](#)
- [S3 VS BLOBS VS Cloud Storage](#)
- 

- 공통점

Object Storage는 가장 기본이 되는 서비스로 Google, Azure에서도 당연히 제공된다. 그렇기 때문에, 차이는 사실상 가격에만 초점이 있어, 가격적인 면에서만 비교를 해보겠다.

	AWS	Azure	Google
Service Name	S3	Blobs	Cloud Storage
Availability SLA	99.95%	99.99%	99.95%
HOT	S3 Standard	Hot Blob Storage	GCS
Cool	S3 Standard_IA	Cool Blob Storage	GCS Nearline
Cold	Glacier	Use Cool Blob Storage	GCS Coldline
# Object Limits	Unlimited	Unlimited	Unlimited
Size Limit	5TB/Object	500 TB/account	5TB per object

- 차이점(가격)

#### Monthly per-GB Prices for the first TB stored

	AWS	AZURE	Google
HOT	0.023	0.0208	0.026
COOL	0.0125	0.0152	0.01
COLD	0.004	0.002	0.007

### 3. 데이터 사이언스 관점에서 해당 AWS의 활용 방법

- 고려 사항 : 최종일관성(Eventual Consistency)

S3는 데이터를 저장할 때 자동으로 여러 개의 데이터 센터에서 데이터를 동기화한다. 덕분에 높은 견고성과 내결함성을 보장하는데, 이렇게 동기화할 때 이슈는, 동시에 동기화하기 위해서는(Writing) 그 사이에는 Reading을 할 수 없다는 문제가 있다. 즉, 견고성과 내결함성을 얻기 위해서 성능을 잃어버리는 이슈가 발생하는데, S3는 기본적으로 최종 일관성, 즉, "업데이트 작업은 시간이 어느정도 지나면 반드시 반영한다"라는 전제로 동작한다. 이와 반대된 말이 엄밀한 일관성(Strong Consistency)으로, "어떤 때 데이터를 빼도 일관된 값이 나온다"이다.

- S3 이벤트 알림

S3는 새로운 객체 생성, 객체 제거, 복원 등의 이슈가 터졌을 때, 이런 이벤트를 알려서 다른 프로그램이 동작하도록 지원한다. 이벤트 개시 대상으로는

- Amazon SNS : 푸시 메시징 시스템, 분산 서비스로 메시지를 푸시
- Amazon SQS : 메시지 대기열 시스템
- AWS Lambda : 애플리케이션을 구축할 수 있는 컴퓨팅 서비스

### 4. 요금 부과 체계(직접 실습해볼 때 요금폭탄을 맞지 않기 위해).

- 비용 체계

AWS 비용 체계는 기본적으로 종량제 과금 형태로, 시간 기반, 용량 기반, 횟수 기반이라는 기준으로 계산된다. 또한 네트워크에 따라 차등적으로 가격을 부여하는데, 외부의 네트워크에서 AWS로 통신하는 것을 In이라고 하고, AWS에서 외부의 네트워크로 나가는 것을 Out이라고 한다. 기본적으로 In은 무료이고 Out에서 돈을 받는 형식이다. 그리고 동일 Region간의 송수신은 무료고, 타 Region은 외부 네트워크보다는 좀 더 싸게 돈을 받는다.

S3에서는 크게 3가지 액션에 따라 돈을 따로 받는다.

1. 저장

스토리지 요금

리전:	미국 동부(오하이오) ⚙
요금	
S3 Standard 스토리지	
처음 50TB/월	0.023 USD/GB
다음 450TB/월	0.022 USD/GB
500TB/월 초과	0.021 USD/GB

2. S3 이벤트

# 요청 요금

아래에 달리 명시되지 않은 요청의 경우

## S3 Standard

리전:	미국 동부(오하이오) ▾
	요금
S3 Select에서 반환한 데이터	0.0007 USD/GB
S3 Select에서 스캔한 데이터	0.002 USD/GB
PUT, COPY, POST 또는 LIST 요청	0.005 USD/요청 1,000건
GET, SELECT 및 기타 모든 요청	0.0004 USD/요청 1,000건
Standard로 수명 주기 전환 요청 - Infrequent Access 또는 One Zone - Infrequent Access 또는 Intelligent-Tiering	0.01 USD/요청 1,000건

### 3. 네트워크 통신

## 데이터 전송 요금

아래 요금은 Amazon S3에서 퍼블릭 인터넷을 통해 "수신" 및 "송신"되는 데이터를 기준으로 합니다<sup>1,2,3</sup>. AWS Direct Connect 요금은 [여기 >](#)서 확인할 수 있습니다.

같은 AWS 리전 내에서 S3 버킷 간에 또는 Amazon S3에서 다른 서비스로 데이터 전송은 무료입니다.

리전:	미국 동부(오하이오) ▾
요금	
인터넷에서 Amazon S3로 데이터 수신	
모든 데이터 수신	GB당 0.00 USD
Amazon S3에서 인터넷으로 데이터 송신	
최대 1GB/월	GB당 0.00 USD
다음 9.999TB/월	GB당 0.09 USD
다음 40TB/월	GB당 0.085 USD
다음 100TB/월	GB당 0.07 USD
150TB 초과/월	GB당 0.05 USD
Amazon S3에서 데이터 송신	
CloudFront	GB당 0.00 USD
미국 동부(버지니아 북부)	GB당 0.01 USD

으로 계산된다.

[가격표](#)

## 5. 이 외(boto3)

AWS는 기본적으로 `boto3` 라는 라이브러리를 지원해주는데, 이 덕분에 좀 더 편하게 파이썬 환경에서 AWS 서비스를 제어할 수 있다. 아래는 관련된 코드이다.

```
import boto3
```

```

from botocore.exceptions import ProfileNotFound, NoRegionError
from tqdm import tqdm
import os

```

```

REGION_NAME_DICT = {
    "아시아 태평양(싱가포르)": "ap-southeast-1",
    "아시아 태평양(시드니)": "ap-southeast-2",
    "아시아 태평양(도쿄)": "ap-northeast-1",
    "아시아 태평양(서울)": "ap-northeast-2",
    "아시아 태평양(오사카-로컬)": "ap-northeast-3",
    "아시아 태평양(뭄바이)": "ap-south-1",
    "캐나다(중부)": "ca-central-1",
    "중국(베이징)": "cn-north-1",
    "중국(닝샤)": "cn-northwest-1",
    "EU(프랑크푸르트)": "eu-central-1",
    "EU(아일랜드)": "eu-west-1",
    "EU(런던)": "eu-west-2",
    "EU(파리)": "eu-west-3",
    "남아메리카(상파울루)": "sa-east-1",
    "미국 동부(버지니아 북부)": "us-east-1",
    "미국 동부(오하이오)": "us-east-2",
    "미국 서부(캘리포니아 북부 지역)": "us-west-1",
    "미국 서부(오레곤)": "us-west-2",
}

```

```

class S3Manager:

```

```

    """

```

S3 내 다양한 기능 중에서 크게 5가지 기능만을 중심으로, 보다 간편한 방식으로 통신할 수 있도록 만든 Manager 클래스

1. LIST - S3 내 Object들의 List를 반환받음
2. DOWNLOAD - S3 내 Object를 download 받음
3. UPLOAD - S3 로 Object를 upload 함
4. DELETE - S3 내 Object를 Delete 함
5. RENAME - S3 내 Object의 Path를 바꿈

```

    """

```

```

def __init__(
    self, bucket_name=None,
    profile_name='default',
    s3_location='ap-northeast-2'):
    self.profile_name = profile_name
    self.s3_location = s3_location
    if not self.verify_profile_name(profile_name):
        raise ProfileNotFound

    self.s3 = self.get_s3_client(profile_name)

```



```

        if not self.verify_s3_location(s3_location):
            raise NoRegionError

    if bucket_name is None:
        raise ValueError("Bucket name must exists!")
    self.bucket_name = bucket_name
    if not self.verify_bucket_existence():
        print("There is no Bucket({}) in S3. You should create bucket
first".format(self.bucket_name))

#####
# Session 관리
#   - get_s3_client
#   - verify_profile_name
#   - verify_s3_location
#####
@staticmethod
def get_s3_client(profile_name):
    session = boto3.session.Session(profile_name=profile_name)
    return session.client('s3')

@staticmethod
def verify_profile_name(profile_name):
    return profile_name in boto3.session.Session().available_profiles

@staticmethod
def verify_s3_location(s3_location):
    return s3_location in REGION_NAME_DICT.values()

#####
# Bucket 관리
#   - verify_bucket_existence
#   - create_bucket
#   - delete_bucket
#####
def verify_bucket_existence(self):
    res = self.s3.list_buckets()
    return self.bucket_name in [bucket['Name']
                                for bucket in res['Buckets']]

def create_bucket(self):
    self.s3.create_bucket(
        Bucket=self.bucket_name,
        CreateBucketConfiguration={
            "LocationConstraint": self.s3_location
        })

def delete_bucket(self):
    self.s3.delete_bucket(Bucket=self.bucket_name)

```

```

#####
# File 관리
# - list_s3_file_paths
# - upload_file
# - download_file
# - delete_file
# - rename_file
#####
def list_s3_file_paths(self, prefix='', suffix=''):
    """
    Generate the s3_file_paths in an S3 bucket.

    :param prefix: Only fetch s3_file_paths that start with this prefix
(optional).
    :param suffix: Only fetch s3_file_paths that end with this suffix
(optional).
    """
    kwargs = {'Bucket': self.bucket_name}
    # If the prefix is a single string (not a tuple of strings), we can
    # do the filtering directly in the S3 API.
    if isinstance(prefix, str):
        kwargs['Prefix'] = prefix
    s3_file_paths = []
    while True:
        # The S3 API response is a large blob of metadata.
        # 'Contents' contains information about the listed objects.
        resp = self.s3.list_objects_v2(**kwargs)
        for obj in resp['Contents']:
            s3_file_path = obj['Key']
            if s3_file_path.startswith(prefix) and\
                s3_file_path.endswith(suffix) and\
                s3_file_path[-1] != "/":
                s3_file_paths.append(s3_file_path)
        # The S3 API is paginated, returning up to 1000 s3_file_paths at
a time.

        # Pass the continuation token into the next response, until we
        # reach the final page (when this field is missing).
        try:
            kwargs['ContinuationToken'] = resp['NextContinuationToken']
        except KeyError:
            break

    return s3_file_paths

def upload_file(self, local_file_path, s3_file_path):
    self.s3.upload_file(local_file_path, self.bucket_name, s3_file_path)

def upload_files(self, local_file_paths, s3_file_paths):

```

```

"""
upload the file list to bucket

:param local_file_paths: the list of file path in local directory
:param s3_file_paths : the list of key in s3

local_file_paths와 s3_file_paths는 서로 같은 인덱스 순으로 매칭이 된다.
"""

for local_file_path, s3_file_path in tqdm(
    zip(local_file_paths, s3_file_paths)):
    self.upload_file(local_file_path, s3_file_path)

def download_file(self, s3_file_path, local_file_path):
    self.s3.download_file(self.bucket_name, s3_file_path,
local_file_path)

def download_files(self, s3_file_paths, local_file_paths):
    """
    download the file list from bucket

    :param s3: s3 client
    :param s3_file_paths: the list of key in s3
    :param local_file_paths: the list of file path in local directory
    """
    for s3_file_path, local_file_path in tqdm(
        zip(s3_file_paths, local_file_paths)):
        file_dir = os.path.split(local_file_path)[0]
        if file_dir == "":
            pass
        else:
            os.makedirs(file_dir, exist_ok=True)

        self.download_file(s3_file_path, local_file_path)

def delete_file(self, s3_file_path):
    """
    delete the file from bucket

    :param s3_file_path:
    :return:
    """
    self.s3.delete_object(Bucket=self.bucket_name, Key=s3_file_path)

def delete_files(self, s3_file_paths):
    """
    delete the file list from bucket

    :param s3_file_paths:
    :return:

```

```

"""
for s3_file_path in tqdm(s3_file_paths):
    self.delete_file(s3_file_path)

def rename_file(self, old_file_path, new_file_path):
    """
    rename the file from bucket

    :param old_file_path:
    :param new_file_path:
    :return:
    """
    self.s3.copy_object(
        Bucket=self.bucket_name,
        Key=new_file_path,
        CopySource={
            'Bucket': self.bucket_name,
            'Key': old_file_path})
    self.s3.delete_object(Bucket=self.bucket_name, Key=old_file_path)

def rename_files(self, old_file_paths, new_file_paths):
    """
    rename the file list from bucket

    :param old_file_paths:
    :param new_file_paths:
    :return:
    """
    for old_file_path, new_file_path in tqdm(
        zip(old_file_paths, new_file_paths)):
        self.rename_file(old_file_path, new_file_path)

```