Blob 2 - Python으로 Blob Storage 연동

빠른 시작: Python용 클라이언트 라이브러리 Azure Blob Storage

연결 문자열로 선택해서 실습 진행

Python용 Azure Blob Storage 클라이언트 라이브러리를 시작하여 Blob 및 컨테이너를 관리합니다. 다음 단계에 따라 패키지를 설치하고 대화형 콘솔 앱에서 기본 작업에 대한 예제 코드를 사용해 보세요.

API 참조 설명서 | 라이브러리 소스 코드 | 패키지(PyPi) | 샘플

Python용 Azure 라이브러리(SDK) 사용

프로젝트를 만듭니다.

*Blob-quickstart*라는 Python 애플리케이션을 만듭니다.

1. 콘솔 창(예: PowerShell 또는 Bash)에서 프로젝트에 대한 새 디렉터리를 만듭니다.

   ```
   mkdir blob-quickstart
   ```

2. 새로 만든 *Blob-quickstart* 디렉터리로 전환합니다.

   ```
   cd blob-quickstart
   ```

패키지 설치

프로젝트 디렉터리에서 명령을 사용하여 `pip install` Azure Blob Storage 및 Azure Identity 클라이언트 라이브러리에 대한 패키지를 설치합니다. Azure 서비스에 대한 암호

없는 연결에는 **azure-identity** 패키지가 필요합니다.

```
pip install azure-storage-blob azure-identity
```

앱 프레임워크 설정

프로젝트 디렉터리에서 단계에 따라 앱의 기본 구조를 만듭니다.

1. 코드 편집기에서 새 텍스트 파일을 엽니다.
2. 문을 추가하고 `import` , 프로그램의 구조를 만들고, 아래와 같이 기본 예외 처리를 포함합니다.
3. *blob-quickstart* 디렉터리에 *새 파일을 blob-quickstart.py* 저장합니다.

```
import os, uuid
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient, BlobClien
t, ContainerClient

try:
    print("Azure Blob Storage Python quickstart sample")

    # Quickstart code goes here

except Exception as ex:
    print('Exception:')
    print(ex)
```

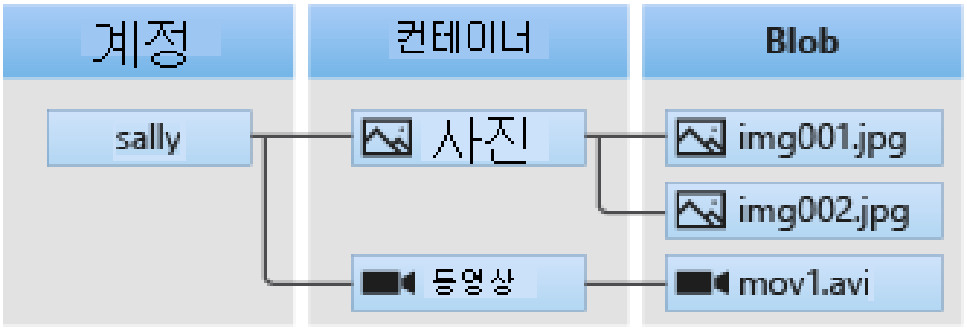
개체 모델

Azure Blob Storage는 대량의 비정형 데이터를 저장하는 데 최적화되어 있습니다. 비정형 데이터는 텍스트 또는 이진 데이터와 같은 특정 데이터 모델이나 정의를 따르지 않는 데이터입니다. Blob Storage는 다음 세 가지 유형의 리소스를 제공합니다.

- 스토리지 계정
- 스토리지 계정의 컨테이너

- 컨테이너의 blob

다음 다이어그램은 이러한 리소스 간의 관계를 보여 줍니다.



다음 Python 클래스를 사용하여 이러한 리소스와 상호 작용합니다.

- BlobServiceClient: `BlobServiceClient` 클래스를 사용하여 Azure Storage 리소스 및 blob 컨테이너를 조작할 수 있습니다.
- ContainerClient: `ContainerClient` 클래스를 사용하여 Azure Storage 컨테이너 및 해당 blob을 조작할 수 있습니다.
- BlobClient: `BlobClient` 클래스를 사용하여 Azure Storage blob을 조작할 수 있습니다.

Azure에 인증하고 Blob 데이터에 대한 액세스 권한 부여

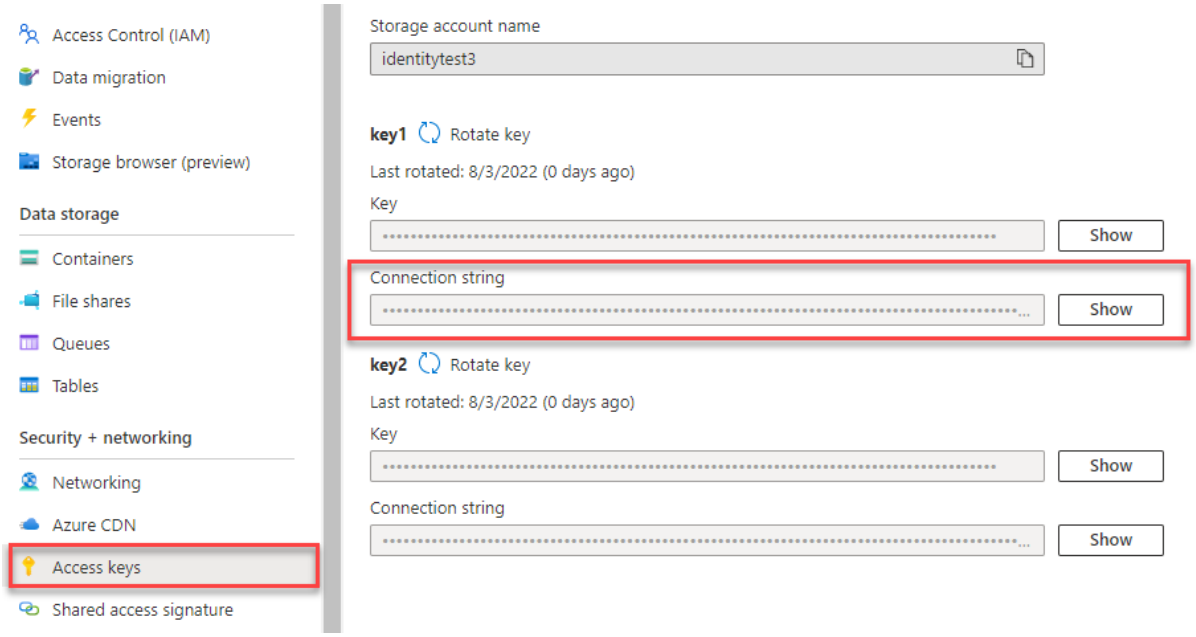
암호 없이 DefaultAzureCredential로 사용하는 방법(권장)은 아래 링크 참고

https://learn.microsoft.com/ko-kr/azure/storage/blobs/storage-quickstart-blobs-python?tabs=managed-identity%2Croles-azure-portal%2Csign-in-azure-cli#authenticate-to-azure-and-authorize-access-to-blob-data

연결 문자열에는 스토리지 계정 액세스 키가 포함되며, 이를 사용하여 요청에 대한 권한을 부여합니다. 키를 안전하지 않은 위치에 노출하지 않도록 항상 주의해야 합니다.

> **참고**
>
> 스토리지 계정 액세스 키를 사용하여 데이터 액세스 권한을 부여하려면 다음 Azure RBAC 작업에 대한 권한이 필요합니다. **Microsoft.Storage/storageAccounts/listkeys/action**. 이 작업에 대한 권한이 있는 최소 권한 있는 기본 제공 역할은 **읽기 권한자 및 데이터 액세스**이지만 이 작업을 포함하는 모든 역할이 작동합니다.

1. Azure Portal에 로그인합니다.
2. 스토리지 계정을 찾습니다.
3. 스토리지 계정 메뉴 창의 **보안 + 네트워킹**에서 **액세스 키**를 선택합니다.
4. **액세스 키** 창에서 **키 표시**를 선택합니다.
5. **key1** 섹션에서 **연결 문자열** 값을 찾습니다. **클립보드에 복사** 아이콘을 선택하여 연결 문자열을 복사합니다. 다음 단계에서 연결 문자열 값을 환경 변수에 추가합니다.



스토리지 연결 문자열 구성

이후 작업 시에는 연결 문자열을 시스템 환경 변수를 활용해서 코드에 직접 입력하지 않도록 합니다.

다음 코드를 `try` 블록 내에 추가합니다. 들여쓰기를 잘 맞춰줍니다.

아래 'CONNECTION_STRING' 문자열 부분을 포탈에서 복사한 연결 문자열로 바꿔줍니다.

```
# Retrieve the connection string for use with the applicati
on. The storage
# connection string is stored in an environment variable on
the machine
# running the application called AZURE_STORAGE_CONNECTION_S
TRING. If the environment variable is
# created after the application is launched in a console or
with Visual Studio,
# the shell or application needs to be closed and reloaded
to take the
# environment variable into account.
connect_str = 'CONNECTION_STRING'

# Create the BlobServiceClient object
blob_service_client = BlobServiceClient.from_connection_str
ing(connect_str)
```

중요

계정 액세스 키는 신중하게 사용해야 합니다. 계정 액세스 키가 손실하거나 실수로 안전하지 않은 위치에 배치되는 경우 서비스가 취약해질 수 있습니다. 액세스 키가 있는 사람은 누구나 스토리지 계정에 대한 요청에 권한을 부여할 수 있으며 모든 데이터에 효과적으로 액세스할 수 있습니다. `DefaultAzureCredential` 는 향상된 보안 기능과 이점을 제공하며 Azure 서비스에 대한 권한 부여를 관리하는 데 권장되는 방법입니다.

컨테이너 만들기

새 컨테이너의 이름을 결정합니다. 아래 코드는 컨테이너 이름이 고유해질 수 있도록 이름에 UUID 값을 추가합니다.

> 💡 **중요**
>
> 컨테이너 이름은 소문자여야 합니다. 컨테이너 및 Blob 이름 지정에 대한 자세한 내용은 **컨테이너, Blob, 메타데이터 이름 지정 및 참조**를 참조하세요.

create_container 메서드를 호출하여 스토리지 계정에 컨테이너를 실제로 만듭니다.

이 코드를 `try` 블록의 끝에 추가합니다.

```
# Create a unique name for the container
container_name = str(uuid.uuid4())

# Create the container
container_client = blob_service_client.create_container(con
tainer_name)
```

컨테이너를 만드는 방법에 대해 자세히 알아보고 더 많은 코드 샘플을 살펴보려면 Python을 사용하여 Blob 컨테이너 만들기를 참조하세요.

포탈에서 스토리지 계정의 컨테이너에 추가로 생성된 걸 확인 할 수 있습니다.

컨테이너에 Blob 업로드

1. 데이터 파일을 저장할 로컬 디렉터리를 만듭니다.
2. 로컬 디렉터리에 텍스트 파일을 만듭니다.
3. 컨테이너 만들기 섹션에서 BlobServiceClient에 대해 get_blob_client 메서드를 호출하여 BlobClient 개체에 대한 참조를 가져옵니다.
4. upload_blob 메서드를 호출하여 로컬 텍스트 파일을 Blob에 업로드합니다.

이 코드를 `try` 블록의 끝에 추가합니다.

```
# Create a local directory to hold blob data
local_path = "./data"
os.mkdir(local_path)
```

```
# Create a file in the local data directory to upload and d
ownload
local_file_name = str(uuid.uuid4()) + ".txt"
upload_file_path = os.path.join(local_path, local_file_nam
e)

# Write text to the file
file = open(file=upload_file_path, mode='w')
file.write("Hello, World!")
file.close()

# Create a blob client using the local file name as the nam
e for the blob
blob_client = blob_service_client.get_blob_client(container
=container_name, blob=local_file_name)

print("\nUploading to Azure Storage as blob:\n\t" + local_f
ile_name)

# Upload the created file
with open(file=upload_file_path, mode="rb") as data:
    blob_client.upload_blob(data)
```

Blob 업로드에 대해 자세히 알아보고 더 많은 코드 샘플을 살펴보려면 Python을 사용하여 Blob 업로드를 참조하세요.

파이썬 작업 폴더(로컬)에 data폴더와 텍스트 파일이 생성되며

Blob에 컨테이너가 생성되면서 텍스트 파일이 업로드 됩니다.

포탈에서 확인 할 수 있습니다.

컨테이너의 Blob 나열

list_blobs 메서드를 호출하여 컨테이너의 blob을 나열합니다. 이 경우 하나의 Blob만 컨테이너에 추가되었으므로 나열된 작업은 하나의 해당 Blob만 반환합니다.

이 코드를 `try` 블록의 끝에 추가합니다.

```
print("\nListing blobs...")

# List the blobs in the container
blob_list = container_client.list_blobs()
for blob in blob_list:
    print("\t" + blob.name)
```

Blob을 나열하고 더 많은 코드 샘플을 탐색하는 방법에 대한 자세한 내용은 Python을 사용하여 Blob 나열을 참조하세요.

Blob 다운로드

download_blob 메서드를 호출하여 이전에 만든 blob을 다운로드합니다. 예제 코드는 로컬 파일 시스템에서 두 파일을 볼 수 있도록 파일 이름에 "DOWNLOAD" 접미사를 추가합니다.

이 코드를 `try` 블록의 끝에 추가합니다.

```
# Download the blob to a local file
# Add 'DOWNLOAD' before the .txt extension so you can see b
oth files in the data directory
download_file_path = os.path.join(local_path, str.replace(l
ocal_file_name ,'.txt', 'DOWNLOAD.txt'))
container_client = blob_service_client.get_container_client
(container= container_name)
print("\nDownloading blob to \n\t" + download_file_path)

with open(file=download_file_path, mode="wb") as download_f
ile:
 download_file.write(container_client.download_blob(blob.na
me).readall())
```

Blob 다운로드에 대해 자세히 알아보고 더 많은 코드 샘플을 살펴보려면 Python을 사용하여 Blob 다운로드를 참조하세요.

파이썬 작업 폴더(로컬)의 data폴더 안에 DOWNLOAD파일이 다운로드 됩니다.

컨테이너 삭제

다음 코드는 delete_container 메서드로 전체 컨테이너를 제거하여 앱이 만든 리소스를 정리합니다. 원하는 경우 로컬 파일을 삭제할 수도 있습니다.

앱은 blob, 컨테이너 및 로컬 파일을 삭제하기 전에 `input()`을 호출하여 사용자 입력을 일시 중지합니다. 리소스가 삭제되기 전에 올바르게 만들어졌는지 확인합니다.

이 코드를 `try` 블록의 끝에 추가합니다.

```
# Clean up
print("\nPress the Enter key to begin clean up")
input()

print("Deleting blob container...")
container_client.delete_container()

print("Deleting the local source and downloaded files...")
os.remove(upload_file_path)
os.remove(download_file_path)
os.rmdir(local_path)

print("Done")
```

컨테이너 삭제에 대해 자세히 알아보고 더 많은 코드 샘플을 살펴보려면 Python을 사용하여 Blob 컨테이너 삭제 및 복원을 참조하세요.

Enter key 입력전에 이전 작업 내용을 먼저 확인합니다.

코드 실행

이 앱은 로컬 폴더에 테스트 파일을 만들고 Azure Blob Storage에 업로드합니다. 그런 다음, 예제에서 컨테이너의 Blob을 나열하고 새 이름으로 파일을 다운로드합니다. 이전 파일과 새 파일을 비교할 수 있습니다.

blob-quickstart.py 파일이 포함된 디렉터리로 이동한 다음, 다음 `python` 명령을 실행하여 앱을 실행합니다.

```
python blob-quickstart.py
```

앱의 출력은 다음 예제와 유사합니다(가독성을 위해 생략된 UUID 값).

```
Azure Blob Storage Python quickstart sample

Uploading to Azure Storage as blob:
        quickstartUUID.txt

Listing blobs...
        quickstartUUID.txt

Downloading blob to
        ./data/quickstartUUIDDOWNLOAD.txt

Press the Enter key to begin clean up

Deleting blob container...
Deleting the local source and downloaded files...
Done
```

정리 프로세스를 시작하기 전에 *data* 폴더에서 두 파일을 확인합니다. 비교하고 동일한지 확인할 수 있습니다.

리소스 정리

파일을 확인하고 테스트를 완료한 후 **Enter** 키를 눌러 스토리지 계정에서 만든 컨테이너와 함께 테스트 파일을 삭제합니다. Azure CLI를 사용하여 리소스를 삭제할 수도 있습니다.