

한국 마이크로소프트

Microsoft Technical Trainer

Enterprise Skills Initiative

Microsoft Azure

Azure Databricks

이 문서는 Microsoft Technical Trainer팀에서 ESI 교육 참석자분들에게 제공해 드리는 문서입니다.

요약

이 내용들은 표시된 날짜에 Microsoft에서 검토된 내용을 바탕으로 하고 있습니다. 따라서, 표기된 날짜 이후에 시장의 요구사항에 따라 달라질 수 있습니다. 이 문서는 고객에 대한 표기된 날짜 이후에 변화가 없다는 것을 보증하지 않습니다.

이 문서는 정보 제공을 목적으로 하며 어떠한 보증을 하지는 않습니다.

저작권에 관련된 법률을 준수하는 것은 고객의 역할이며, 이 문서를 마이크로소프트의 사전 동의 없이 어떤 형태(전자 문서, 물리적인 형태 막론하고) 어떠한 목적으로 재 생산, 저장 및 다시 전달하는 것은 허용되지 않습니다.

마이크로소프트는 이 문서에 들어있는 특허권, 상표, 저작권, 지적 재산권을 가집니다. 문서를 통해 명시적으로 허가된 경우가 아니면, 어떠한 경우에도 특허권, 상표, 저작권 및 지적 재산권은 다른 사용자에게 허여되지 않습니다.

© 2022 Microsoft Corporation All right reserved.

Microsoft®는 미합중국 및 여러 나라에 등록된 상표입니다.

이 문서에 기재된 실제 회사 이름 및 제품 이름은 각 소유자의 상표일 수 있습니다.

문서 작성 연혁

| 날짜 | 버전 | 작성자 | 변경 내용 |
|------------|-------|-----|--------------------------|
| 2022.02.13 | 0.3.0 | 우진환 | TASK 01 작성 |
| 2022.02.14 | 0.9.0 | 우진환 | TASK 02 작성 |
| 2022.02.27 | 1.0.0 | 우진환 | 문서 편집 및 서식 정의 |
| 2022.04.04 | 1.1.0 | 우진환 | 리소스 그룹 이름 변경, TASK 03 추가 |

목차

| | |
|---|----|
| APACHE SPARK 생태계..... | 5 |
| AZURE DATABRICKS 플랫폼 | 6 |
| DATABRICKS 시나리오 | 7 |
| TASK 01. AZURE OPEN DATASET을 사용하여 SPARK SQL 작업 실행 | 8 |
| TASK 02. AZURE BLOB 스토리지를 사용하여 SPARK 작업 실행 | 17 |
| TASK 03. 리소스 정리..... | 27 |

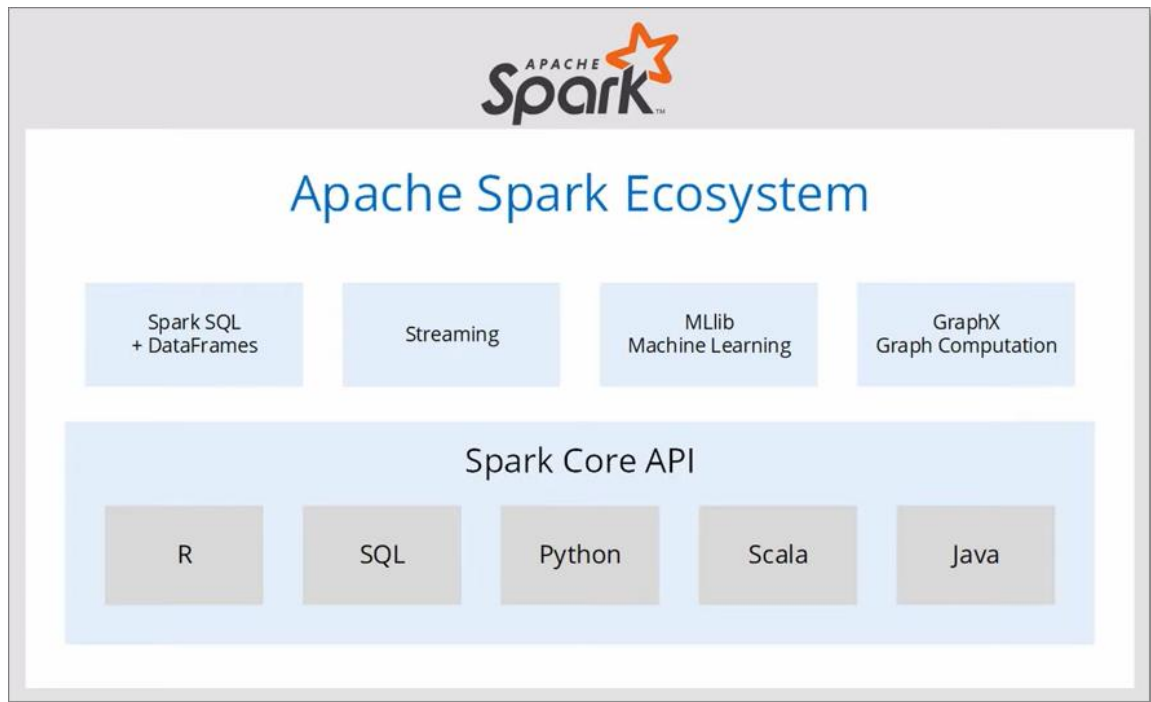
Azure Databricks는 Apache Spark 기반의 분석 플랫폼이며 Azure 클라우드 서비스 플랫폼에 최적화되어 있습니다. Apache Spark를 만든 사람과 함께 디자인된 Databricks는 Azure와 통합되어 원 클릭 설정, 간소화된 워크플로를 제공하며 데이터 과학자, 데이터 엔지니어, 비즈니스 분석가 간의 협업을 가능하게 하는 대화형 워크스페이스를 제공합니다.

Apache Spark 생태계

- Spark SQL + DataFrame: Databricks는 Apache Spark를 기반으로 하기 때문에 가장 주요한 기능은 Spark SQL과 데이터 프레임(DataFrame)입니다. DataFrame은 구조화된 데이터를 이미 작업했던 모든 시스템의 테이블처럼 작업할 수 있게 해주는 라이브러리입니다. 즉 이는 구조적인 데이터를 처리하는 Spark SQL 모듈입니다. DataFrame은 열(column) 단위로 데이터를 저장하는 집합이며 일반적인 관계형 데이터베이스의 테이블과 동일한 개념입니다.
- Streaming: 데이터 스트리밍을 허용하는 Streaming 서비스가 있습니다. 따라서 IoT 혹은 라이브 이벤트 애플리케이션을 수행하는 경우 라이브 시스템에서 변환을 수행할 수 있습니다. HDFS, Flume, Kafka와 통합해서 실시간 데이터를 처리할 수 있습니다.
- MLlib: Spark를 사용하여 모델을 준비하고 트레이닝하는 변환의 머신 러닝 유형을 수행할 수 있는 머신 러닝 라이브러리가 제공됩니다.
- GraphX: GraphX를 통해 소셜 미디어 유형의 애플리케이션을 수행할 수 있습니다.
- 이러한 모든 것은 Spark Core API를 기반으로 하기 때문에 R, Spark SQL (일반 SQL과 약간 다름), Python, Scala, Java를 사용할 수 있습니다.

즉 Azure Databricks는 Apache Spark, Databricks, 클라우드 환경을 통합적인 서비스로 제공하는 환경이라고 이해할 수 있습니다.

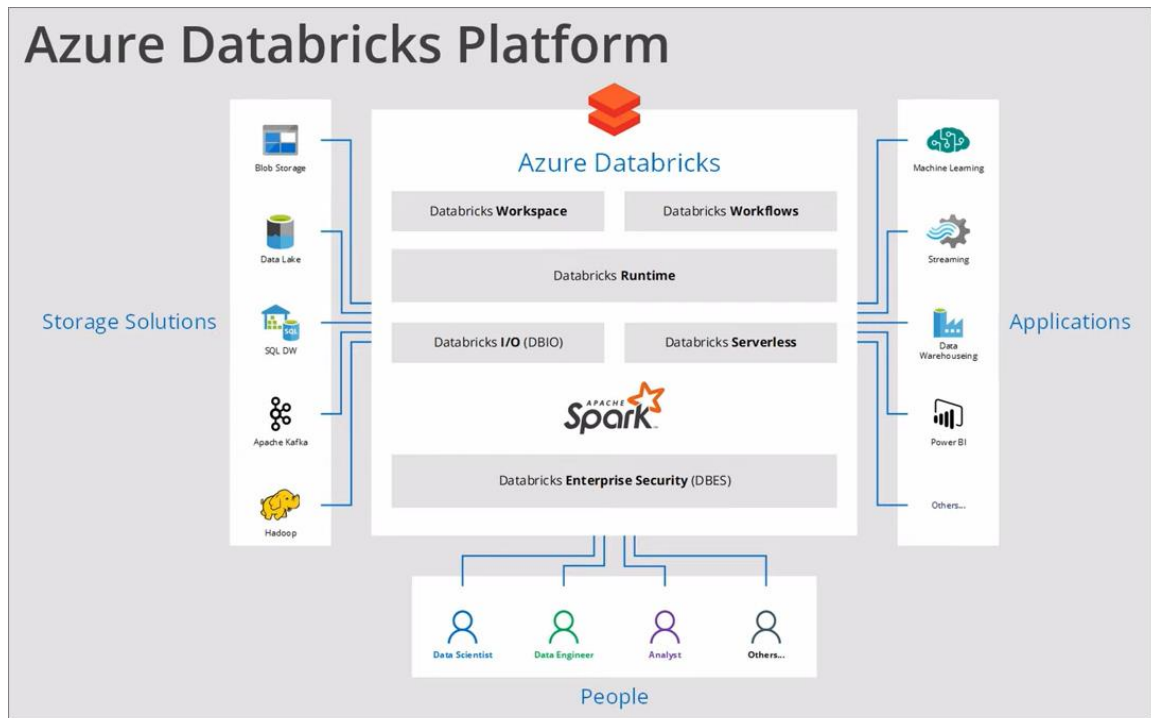
Z



Azure Databricks 플랫폼

플랫폼으로써의 Databricks에는 다음과 같은 여러 기능이 포함되어 있습니다.

- Databricks Runtime: Apache Spark 기반 외의 다른 런타임도 포함되어 있으며 런타임은 이러한 모든 기능을 단일 플랫폼으로 결합하는 워크스페이스를 제공합니다. 이 워크스페이스에서 스크립트를 통해 동료와 공동 작업을 할 수 있습니다. 또한 스크립트가 여러 개인 경우 이를 워크플로로 결합할 수 있습니다. 워크플로는 스크립트의 중첩일 수 있으며 스크립트는 기본적으로 간단한 ETL을 제공합니다.
- Databricks I/O (DBIO): Databricks의 input/output 라이브러리이며 Apache Kafka, Hadoop 뿐 아니라 Azure에서 여러 서비스를 쉽게 연결할 수 있도록 해줍니다.
- Databricks Serverless: Databricks 작업을 할 때 원하는 서버의 종류, 성능, 수를 지정하기만 하면 모든 것은 Azure에서 관리합니다. Databricks는 클러스터를 관리할 필요 없이 클러스터의 처리와 생성만 담당합니다.
- Databricks Enterprise Security (DBES): Databricks는 Azure 및 Azure AD와 잘 통합되어 있기 때문에 모든 액세스, 자격 증명, 권한 부여는 모두 Azure AD를 기반으로 처리됩니다. 따라서 회사 자격 증명을 사용하여 Databricks를 사용할 수 있습니다.
- Databricks에 연결할 수 있는 많은 솔루션이 있지만 5가지 주요 솔루션은 Blob 스토리지, Data Lake, SQL Datawarehouse, Apache Kafka, Hadoop입니다.
- 또한 Databricks를 사용할 수 있는 여러 애플리케이션도 있으며 가장 일반적인 것은 머신 러닝, 스트리밍 시나리오, 데이터 웨어하우스, ETL, Power BI입니다.

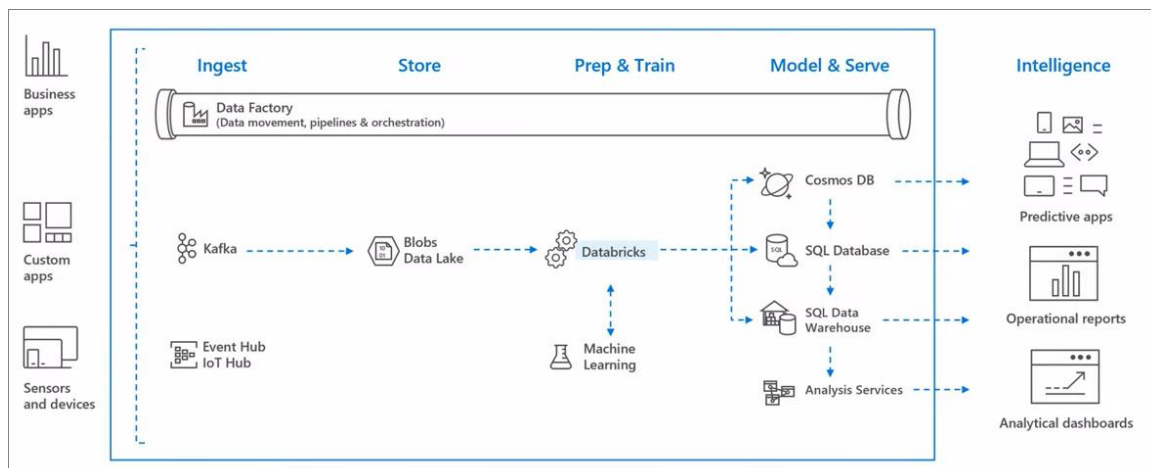


마지막으로 Databricks는 협업 플랫폼이기 때문에 사용하기 쉬운 UI가 제공됩니다.

Databricks 시나리오

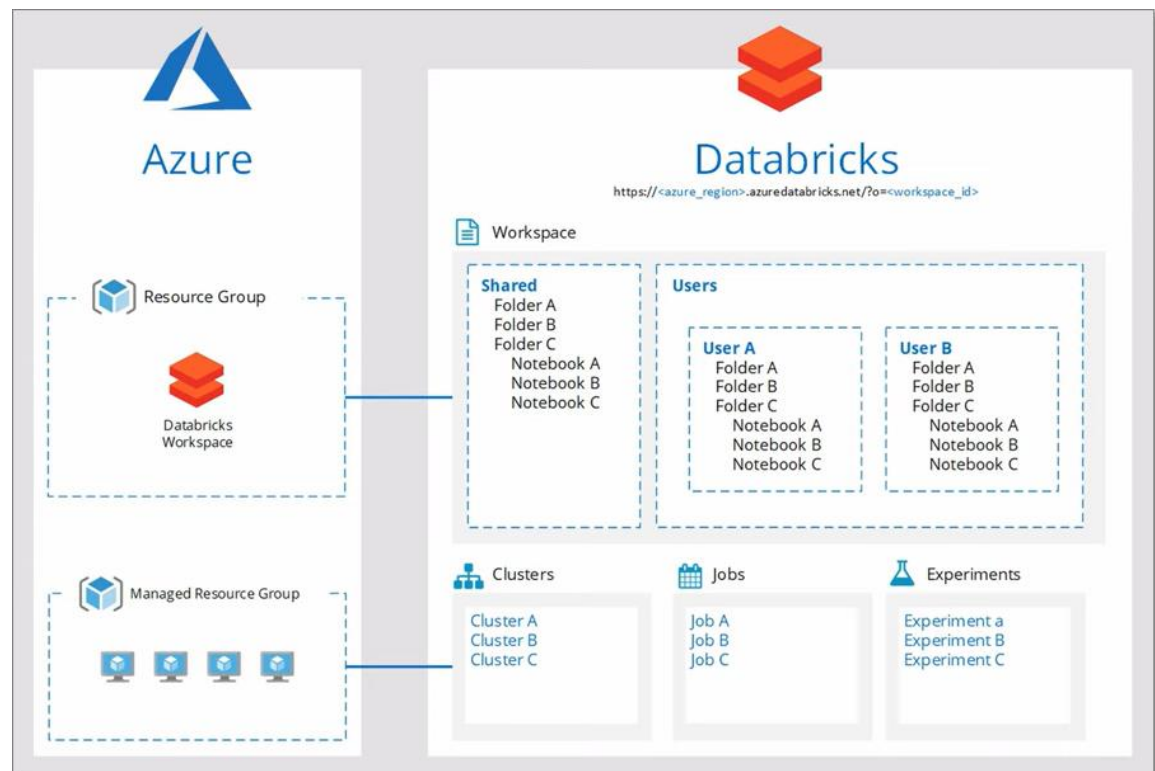
Databricks의 일반적인 시나리오는 머신 러닝 혹은 ETL의 일부인 일반적인 준비와 트레이닝에서 사용하는 것입니다.

- 일반적으로 Ingestion 레이어가 있고 Data Factory, Kafka, IoT Hub, Event Hub 혹은 외부 시스템에서 데이터를 수집하여 Blob 스토리지 혹은 Data Lake에 저장합니다.
- Databricks는 머신 러닝 시나리오의 경우 Blob 스토리지나 Data Lake에서 데이터를 가져와 변환하거나 모델을 트레이닝합니다.
- 그런 다음 이를 Cosmos DB, SQL Database, Data Warehouse, 분석 서비스에서 사용합니다. 물론 원하는 경우 이를 다시 Blob 스토리지에 저장할 수 있습니다.



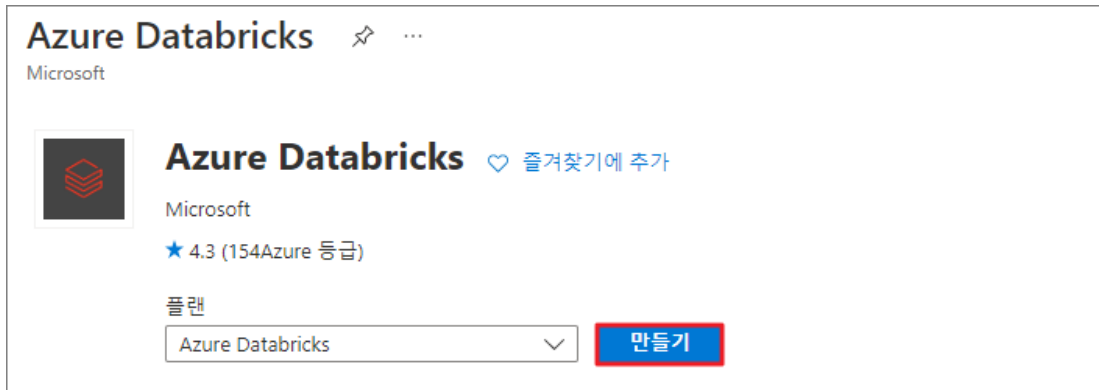
Azure Databricks를 만들게 되면 Azure 리소스 그룹을 생성하고 이 리소스 그룹에 Databricks 워크스페이스를 만들게 됩니다. Azure Kubernetes Service와 마찬가지로 Databricks 워크스페이스를 만들게 되면 Databricks 리소스를 관리하기 위한 별도의 관리되는 리소스 그룹이 생성됩니다. Databricks 워크스페이스는 **azuredatabricks.net**이라는 별도의 포털 서비스가 있고 워크스페이스 ID가 할당됩니다. 이 별도의 포털에서 Databricks와 관련된 거의 모든 작업을 수행할 수 있습니다.

Databricks 워크스페이스에서 공유 워크스페이스, 사용자별 워크스페이스를 만들 수 있으며 클러스터와 작업(job)을 관리하게 됩니다. 클러스터는 기본적으로 작업을 실행하고 작업 후 즉시 삭제하기 위해 생성하는 컴퓨터 리소스입니다.

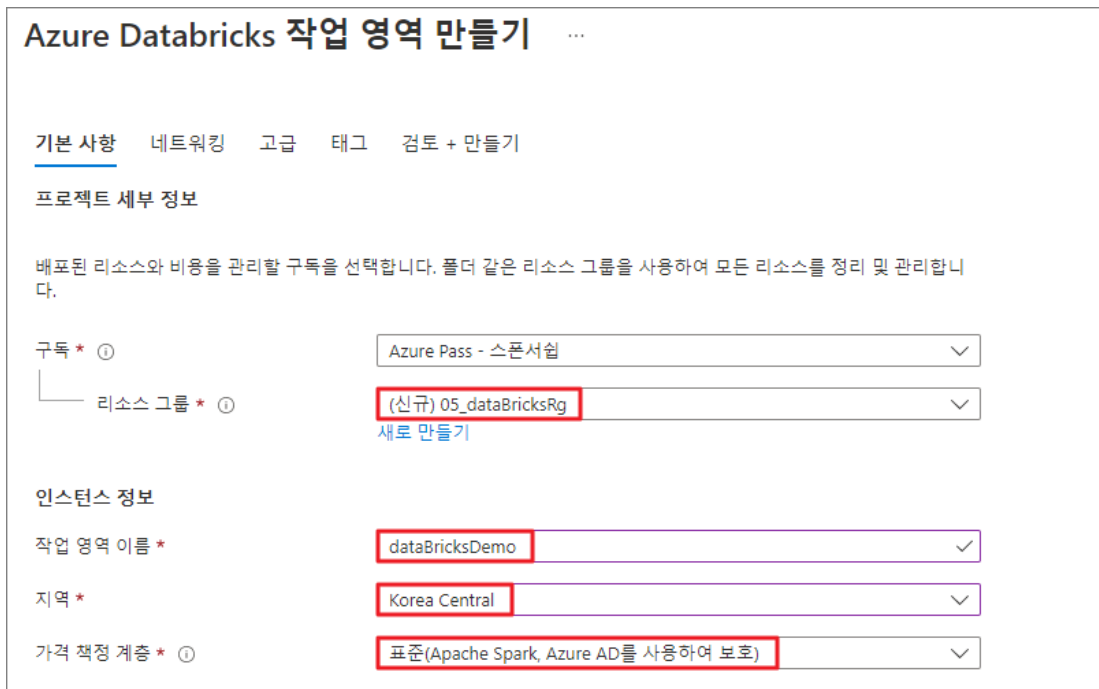


TASK 01. Azure Open Dataset을 사용하여 Spark SQL 작업 실행

1. Azure 포털에서 [리소스 만들기]를 클릭한 후 "Azure Databricks"를 검색하고 클릭합니다. [Azure Databricks] 블레이드에서 [만들기]를 클릭합니다.



2. [Azure Databricks 작업 영역 만들기] 블레이드에서 아래와 같이 구성하고 [다음]을 클릭합니다.
 - [프로젝트 세부 정보 - 리소스 그룹]: "새로 만들기"를 클릭한 후 "05_dataBricksRg"를 입력합니다.
 - [인스턴스 정보 - 작업 영역 이름]: dataBricksDemo
 - [인스턴스 정보 - 지역]: Korea Central
 - [인스턴스 정보 - 가격 책정 계층]: 표준(Apache Spark, Azure AD를 사용하여 보호)



3. [네트워킹] 탭에서 기본 설정을 유지하고 [다음]을 클릭합니다.



4. [고급] 탭에서 기본 설정을 유지하고 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 탭에서 [만들기]를

클릭합니다.

5. Blob 스토리지에서 데이터를 가져올 것(**ingest**)이기 때문에 스토리지 계정을 만들어야 합니다. Azure 포털에서 [리소스 만들기]를 클릭한 후 "스토리지 계정"을 검색하고 클릭합니다. [스토리지 계정] 블레이드에서 [만들기]를 클릭합니다.

6. [저장소 계정 만들기] 블레이드의 [기본] 탭에서 아래와 같이 구성한 후 [검토 + 만들기]를 클릭합니다. [검토 + 만들기] 블레이드에서 [만들기]를 클릭합니다.
- [프로젝트 정보 - 리소스 그룹]: 05_dataBricksRg
 - [인스턴스 정보 - 스토리지 계정 이름]: 중복되지 않는 고유한 이름을 입력합니다.
 - [인스턴스 정보 - 지역]: (Asia Pacific) Korea Central
 - [인스턴스 정보 - 성능]: 표준
 - [인스턴스 정보 - 중복]: LRS(로컬 중복 스토리지)

저장소 계정 만들기

기본 고급 네트워킹 데이터 보호 암호화 태그 검토 + 만들기

Azure Storage는 가용성, 보안, 내구성, 확장성 및 중복성이 뛰어난 클라우드 스토리지를 제공하는 Microsoft 관리 서비스입니다. Azure Storage는 Azure Blob(개체), Azure Data Lake Storage Gen2, Azure Files, Azure 큐 및 Azure 테이블을 포함합니다. 스토리지 계정의 비용은 사용량 및 아래에서 선택한 옵션에 따라 다릅니다. [Azure Storage 계정에 대한 자세한 정보](#)

프로젝트 정보

새 스토리지 계정을 만들 구독을 선택합니다. 다른 리소스와 함께 스토리지 계정을 구성하고 관리할 새 리소스 그룹 또는 기존 리소스 그룹을 선택합니다.

구독 * Azure Pass - 스폰서십

리소스 그룹 * 05_dataBricksRg
[새로 만들기](#)

인스턴스 정보

레거시 스토리지 계정 유형을 만들어야 하는 경우 다음을 클릭하세요. [여기](#).

스토리지 계정 이름 * kormttdatabricks

지역 * (Asia Pacific) Korea Central

성능 * ☒ 표준: 대부분 시나리오에 권장됨(범용 v2 계정)
☐ 프리미엄: 짧은 대기 시간이 필요한 경우에 권장됩니다.

중복 * LRS(로컬 중복 스토리지)

7. 앞서 만든 Azure Databricks 블레이드로 이동합니다. [dataBricksDemo Azure Databricks Service] 블레이드의 [개요]에서 [작업 영역 시작]을 클릭합니다. Databricks 작업 영역이 실제로 작업을 진행하는 별도의 포털입니다. 기본 Azure 포털에서는 가상 네트워크 피어링을 제외하면 Databricks와 관련하여 실제로 할 수 있는 작업은 거의 없습니다.

dataBricksDemo Azure Databricks Service

검색(Ctrl+/) << 삭제

개요

활동 로그

역세스 제어(IAM)

태그

설정

가상 네트워크 피어링

암호화

속성

잠금

자동화

작업(미리 보기)

템플릿 내보내기

기본 정보

상태 : Active 관리되는 리소스 그룹 : databricks-rg-dataBricksDemo-ffcowmjrma35y

리소스 그룹 : 05_dataBricksRg URL : https://adb-6431880263135207.7.azure.databricks.net

위치 : Korea Central 가격 책정 계층 : standard

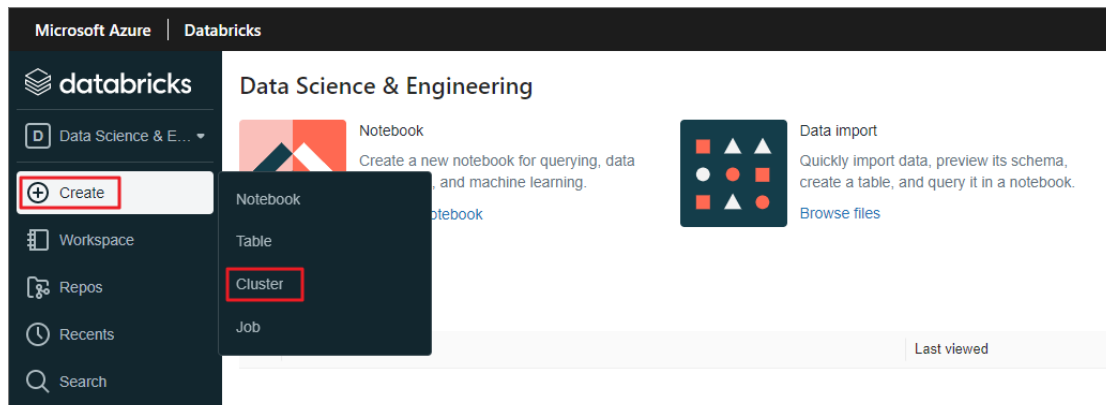
구독 : Azure Pass - 스폰서십

구독 ID : 1e38264e-a5cb-4cb2-b882-43f52ddb6f63

태그 (편집) : 태그를 추가하려면 여기를 클릭

작업 영역 시작

8. Azure Databricks 워크스페이스에서 가장 먼저 해야 할 일은 스크립트를 실행할 워크로드인 클러스터를 만드는 것입니다. 좌측 메뉴에서 [Create - Cluster]를 클릭합니다. 혹은 [Compute] 메뉴로 이동한 후 새 클러스터를 생성할 수도 있습니다.



9. [New Cluster] 페이지에서 아래와 같이 구성한 후 [Create Cluster]를 클릭합니다.
- Cluster name: "demoCluster"를 입력합니다. 프로덕션 환경에서는 의미 있는 이름을 지정하는 것이 좋습니다.
 - Cluster mode: "Standard"를 선택합니다. 여러 사용자가 동일한 클러스터에서 작업하는 경우 **High Concurrency** 모드를 사용하는 것이 좋으며 ETL 변환을 위한 경우 **Standard** 모드를 사용하는 것이 좋습니다. **Single Node**는 워커 없이 클러스터를 생성하고자 하는 경우 사용합니다.
 - Databricks runtime version: 기본으로 표시되는 LTS 버전을 선택합니다.
 - [Autopilot options - Enable autoscaling]: 옵션 선택을 해제합니다. 이 옵션을 선택하는 경우 프로세싱 요구 사항에 따라 워커 수를 지정한 수에 따라 증가 및 감소시킵니다.
 - [Autopilot options - Terminate after]: 기본값을 유지합니다. 이 옵션은 클러스터가 워크로드 처리를 완료하고 더 이상 진행할 작업이 없을 때 자동으로 삭제되는 시간입니다.
 - Worker type: "Standard_DS3_v2"를 선택합니다.
 - Workers: "1"대 만 설정합니다. 이 실습에서 사용할 스크립트는 매우 작기 때문에 더 많은 서버를 사용하는 경우 더 많은 비용이 필요할 뿐 아니라 작업을 지정한 워커 수로 분할하고 다시 결합하는 작업이 실제 수행하는 작업보다 더 커질 수 있습니다.
 - Driver type: "Same as worker"를 선택합니다.

Create Cluster

New Cluster Cancel Create Cluster DBU / hour: 1.5 1 Workers:14 GB Memory, 4 Cores 1 Driver:14 GB Memory, 4 Cores

Cluster name
demoCluster

Cluster mode
Standard

Databricks runtime version
Runtime: 9.1 LTS (Scala 2.12, Spark 3.1.2)

50% promotional discount applied to Photon during preview

Autopilot options
☐ Enable autoscaling
☒ Terminate after 120 minutes of inactivity

Worker type
Standard_DS3_v2 14 GB Memory, 4 Cores

Workers
1 ☐ Spot instances

Driver type
Same as worker 14 GB Memory, 4 Cores

DBU / hour: 1.5 Standard_DS3_v2

Advanced options

10. 클러스터가 생성되면 좌측 메뉴에서 [Compute]로 이동합니다. 앞서 만든 클러스터가 표시되고 2대의 노드가 실행 중인 것을 확인할 수 있습니다. 2대의 노드는 마스터와 워커입니다. 워커는 서버가 생성한 모든 스크립트를 실행하는 가상 머신이며 마스터는 모든 스크립트 실행을 조정하고 두 대 이상의 워커가 있는 경우 작업을 여러 워커로 분할하는 역할을 합니다.

Compute

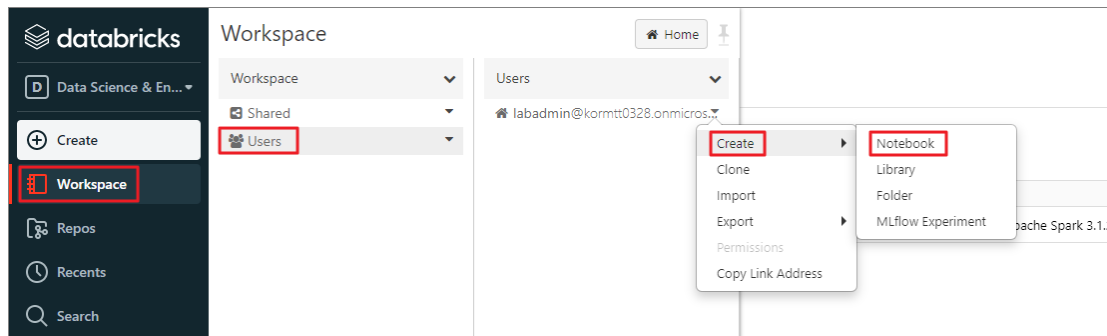
All-purpose clusters Job clusters Pools

Create Cluster All Created by me Accessible by me

| | Name | State | Nodes | Runtime | Driver | Worker | Creator | | Actions |
|--|-------------|---------|-------|-------------------|-----------------|-----------------|------------------------|---|---------|
| | demoCluster | Running | 2 | 9.1 LTS (includes | Standard_DS3_v2 | Standard_DS3_v2 | labadmin@kormtt0328... | 0 | ... |

1 - 1 of 1 < > 20 / Page Go to 1

11. 좌측 메뉴에서 [Workspace - Users - 자신의 계정]을 선택하고 [Create - Notebook]을 클릭합니다. Notebook은 기본 스크립트 언어입니다.



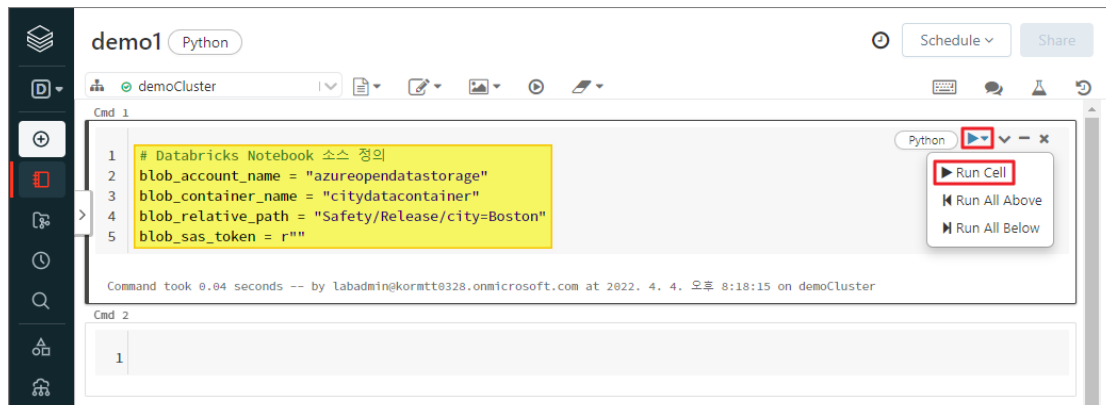
12. [Create Notebook] 창에서 아래와 같이 입력하고 [Create]를 클릭합니다.

- Name: demo1
- Default Language: Python
- Cluster: demoCluster

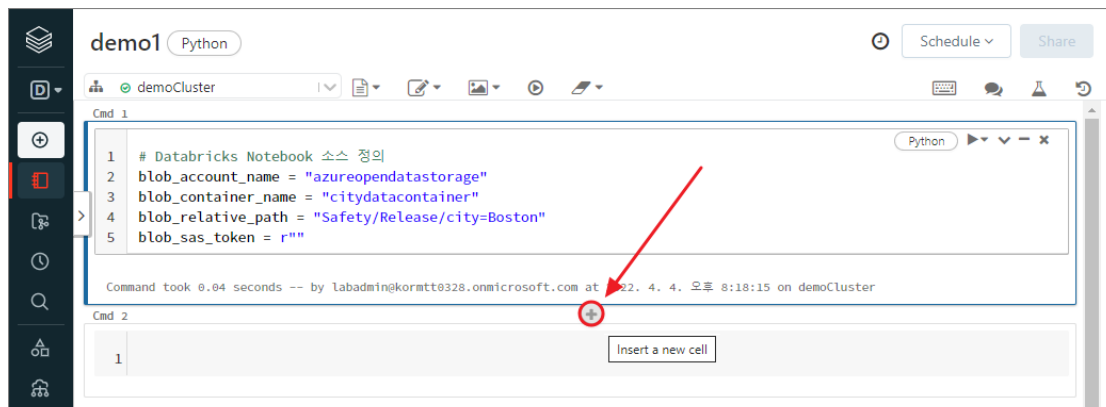
13. 첫 번째 Python 데모는 Microsoft에서 제공하는 Open Dataset을 사용하여 Databricks가 작동하는 방법을 확인해 볼 것입니다. Notebook의 셀에 다음과 같은 코드를 입력하고 <Shift + Enter> 키를 누르거나 우측 상단의 메뉴에서 [Run Cell]을 클릭합니다.

- 이 셀에는 Databricks에서 사용할 Open Dataset을 정의합니다.
- Open Dataset이 제공되는 Azure Blob 스토리지의 이름과 사용할 컨테이너 경로를 지정합니다.
- Open Dataset에 액세스하기 위한 별도의 SAS 토큰은 필요하지 않습니다.

```
# Databricks Notebook 소스 정의
blob_account_name = "azureopendatastorage"
blob_container_name = "citydatacontainer"
blob_relative_path = "Safety/Release/city=Boston"
blob_sas_token = r""
```

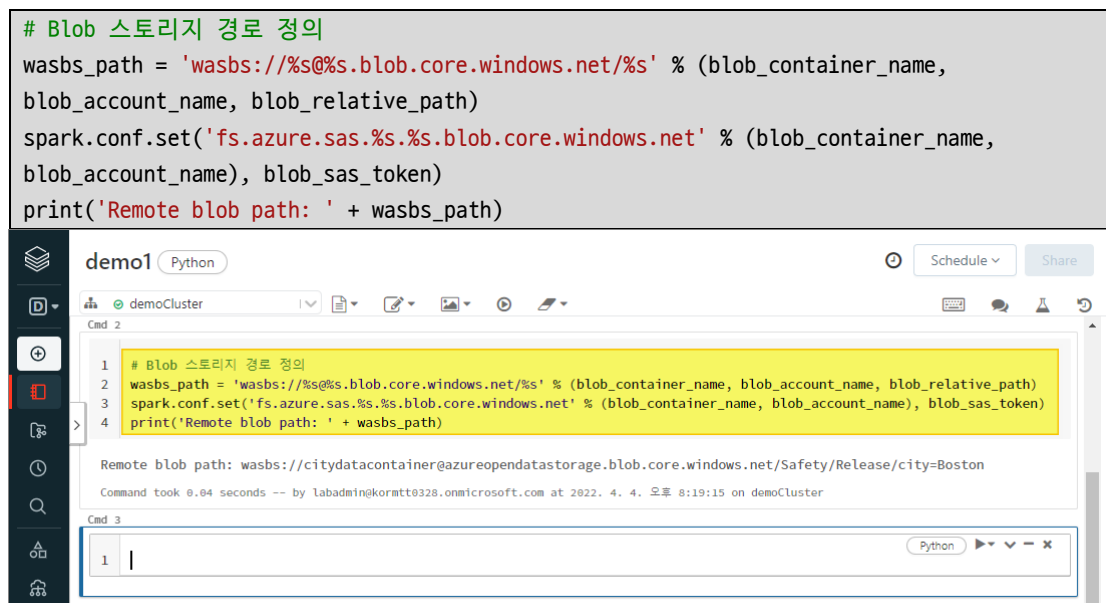


14. 셀이 실행되면 셀 하단에 실행 시간이 표시됩니다. 다음 셀이 자동으로 추가되며 만약 셀이 자동으로 추가되지 않는 경우 셀 하단의 [+] 키를 눌러 새 셀을 추가할 수 있습니다. 셀은 Python으로 작성된 일련의 명령이며 여러 셀을 단일 Notebook으로 결합할 수 있습니다.



15. 다음 셀에 아래와 같은 내용을 입력하고 실행합니다.

- 이 명령은 Blob 스토리지 경로에 대한 설정이며 셀을 실행하면 설정한 Blob 경로가 표시됩니다.
- 즉 원격 경로가 Azure Blob 스토리지를 의미하는 **wasbs** 프로토콜이며 사용할 컨테이너 경로가 표시됩니다.



16. 다음 셀에서 컨테이너의 데이터를 가져오기 위해 다음 명령을 실행합니다.

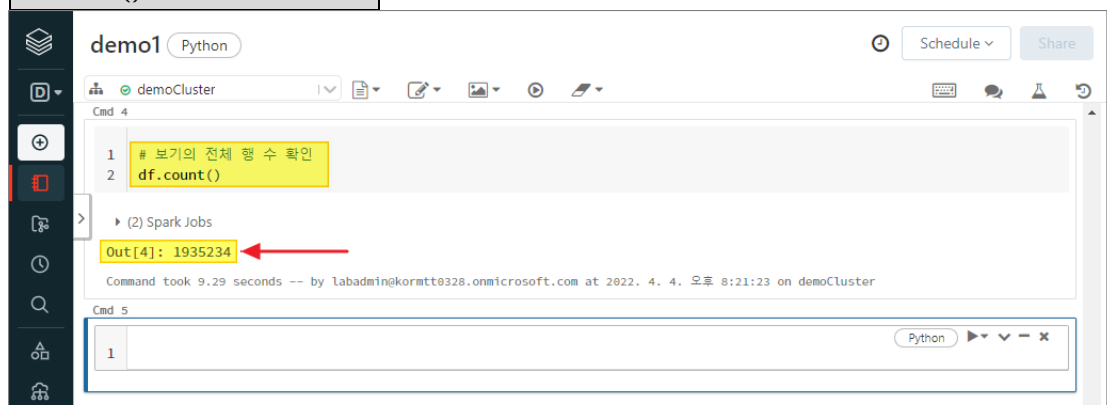
- 첫 번째 명령은 `spark.read`를 통해 앞서 지정했던 Azure Blob 스토리지의 데이터를 Spark가 읽어오도록 합니다. `parquet`은 CSV, JSON 파일과 같은 데이터 형식입니다.
- 두 번째 명령에서 앞 단계에서 실행한 일부 정보를 프린트합니다.
- 마지막 명령에서 새 보기(view)를 만듭니다.

```
# 데이터 가져오기 및 보기 만들기
df = spark.read.parquet(wasbs_path)
print('Register the DataFrame as a SQL temporary view: source')
df.createOrReplaceTempView('source')
```



17. 다음 셀에서 아래와 같은 명령을 실행하여 앞서 만든 보기의 전체 행 수를 확인합니다.

```
# 보기의 전체 행 수 확인
df.count()
```



18. 앞서 "source"라는 뷰를 만들었기 때문에 SQL로 언어를 전환하여 작업을 실행할 수 있습니다. 다음과 같은 명령을 실행합니다.

- `%sql`: SQL 쿼리 언어로 전환하는 명령입니다.
- `SELECT~`: 일반적인 SQL 쿼리를 사용하여 처음 10개의 행만 반환하는 쿼리를 작성합니다.
- 즉 SQL을 사용하여 외부 Blob 스토리지와 연결한 후 데이터를 쿼리하고 결과를 화면에 표시할 수 있습니다.

```
%sql
SELECT * FROM source LIMIT 10
```

demo1 Python

demoCluster

Cmd 5

```
1 %sql
2 SELECT * FROM source LIMIT 10
```

(1) Spark Jobs

Table Data Profile

| | dataType | dataSubtype | dateTime | category | subcategory |
|---|----------|-------------|------------------------------|----------------------------------|----------------------------|
| 1 | Safety | 311_All | 2021-06-23T14:35:00.000+0000 | Highway Maintenance | StreetLight Pole WO |
| 2 | Safety | 311_All | 2021-04-23T11:06:00.000+0000 | Signs & Signals | Sign Repair |
| 3 | Safety | 311_All | 2019-02-07T10:30:17.000+0000 | Highway Maintenance | Request for Pothole Repai |
| 4 | Safety | 311_All | 2013-08-26T13:56:38.000+0000 | Health | Unsanitary Conditions - Es |
| 5 | Safety | 311_All | 2015-02-06T07:51:12.000+0000 | Highway Maintenance | Request for Pothole Repai |
| 6 | Safety | 311_All | 2019-10-30T17:16:01.000+0000 | Enforcement & Abandoned Vehicles | Parking Enforcement |
| 7 | Safetv | 311 All | 2020-08-30T09:48:14.000+0000 | Code Enforcement | Poor Conditions of Proper |

Showing all 10 rows.

Command took 5.99 seconds -- by labadmingkoramt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:22:12 on demoCluster

19. 셀의 출력 내용 하단에서 사용할 수 있는 추가 메뉴가 있습니다.

- ① [Display as Bar Chart] 아이콘을 클릭하여 결과를 여러 종류의 차트로 시각화할 수 있습니다.
- ② [Download CSV] 아이콘을 클릭하여 결과를 CSV 파일로 다운로드하고 다른 사용자에게 공유할 수 있습니다.

demo1 Python

demoCluster

Cmd 5

```
1 %sql
2 SELECT * FROM source LIMIT 10
```

(1) Spark Jobs

Table Data Profile

| | dataType | dataSubtype | dateTime | category | subcategory |
|---|----------|-------------|------------------------------|----------------------------------|----------------------------|
| 1 | Safety | 311_All | 2021-06-23T14:35:00.000+0000 | Highway Maintenance | StreetLight Pole WO |
| 2 | Safety | 311_All | 2021-04-23T11:06:00.000+0000 | Signs & Signals | Sign Repair |
| 3 | Safety | 311_All | 2019-02-07T10:30:17.000+0000 | Highway Maintenance | Request for Pothole Repai |
| 4 | Safety | 311_All | 2013-08-26T13:56:38.000+0000 | Health | Unsanitary Conditions - Es |
| 5 | Safety | 311_All | 2015-02-06T07:51:12.000+0000 | Highway Maintenance | Request for Pothole Repai |
| 6 | Safety | 311_All | 2019-10-30T17:16:01.000+0000 | Enforcement & Abandoned Vehicles | Parking Enforcement |
| 7 | Safetv | 311 All | 2020-08-30T09:48:14.000+0000 | Code Enforcement | Poor Conditions of Proper |

Showing all 10 rows.

Command took 5.99 seconds -- by labadmingkoramt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:22:12 on demoCluster

Cmd 6

1

Shift+

Bar

Scatter

Map

Line

Area

Pie

Quantile

Histogram

Box plot

Q-Q plot

Pivot

Legacy charts

TASK 02. Azure Blob 스토리지를 사용하여 Spark 작업 실행

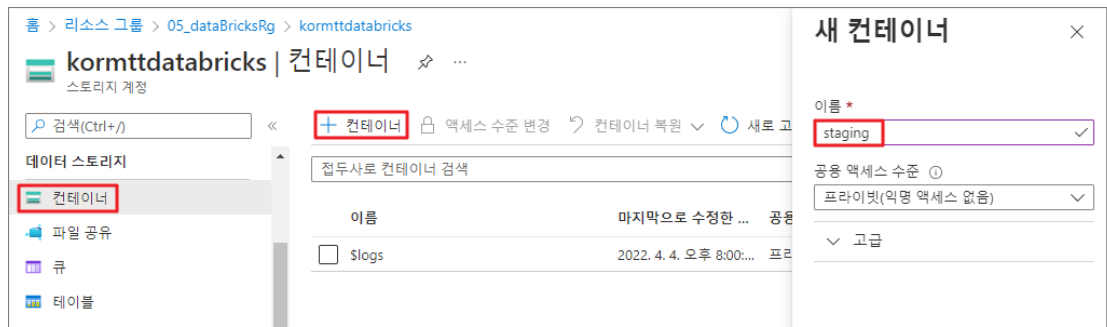
이 작업에서는 Scala를 사용하여 스크립트를 작성하고 앞서 만든 스토리지 계정의 데이터를 분석합니다.

실습에서는 스토리지 계정에 분석할 데이터를 업로드하고 Azure Databricks로 데이터를 변환한 후 변환된 데이터를 다시 스토리지 계정에 저장합니다.

1. 먼저 Azure Databricks를 통해 분석할 데이터를 Azure 스토리지 계정에 업로드합니다. 앞서 만든 [스토리지 계정] 블레이드로 이동합니다. [데이터 스토리지 - 컨테이너]로 이동한 후 메뉴에서 [컨테이너]를

클릭합니다. [새 컨테이너]에서 아래와 같이 입력한 후 [만들기]를 클릭합니다.

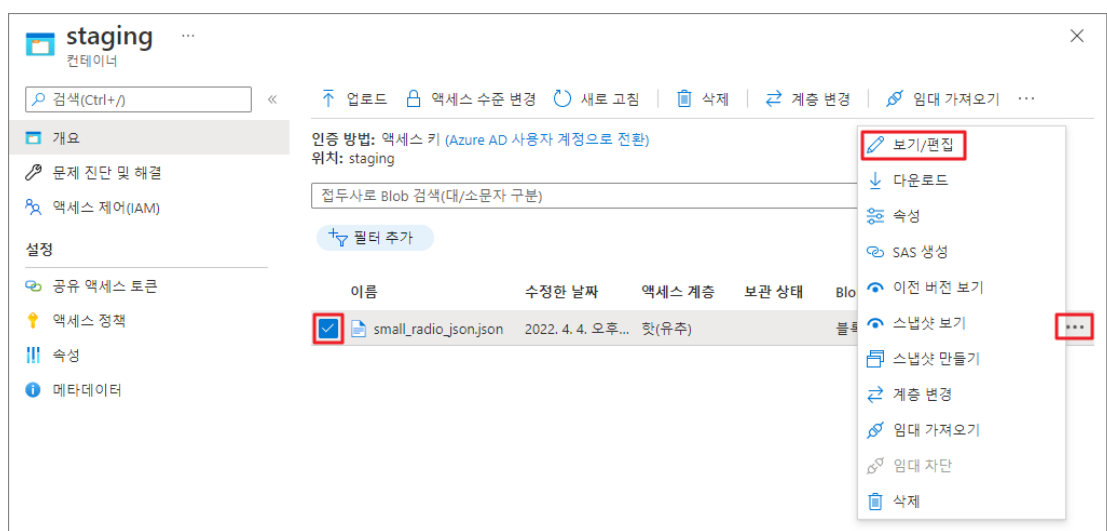
- 이름: staging
- 공용 액세스 수준: 프라이빗(익명 액세스 없음)



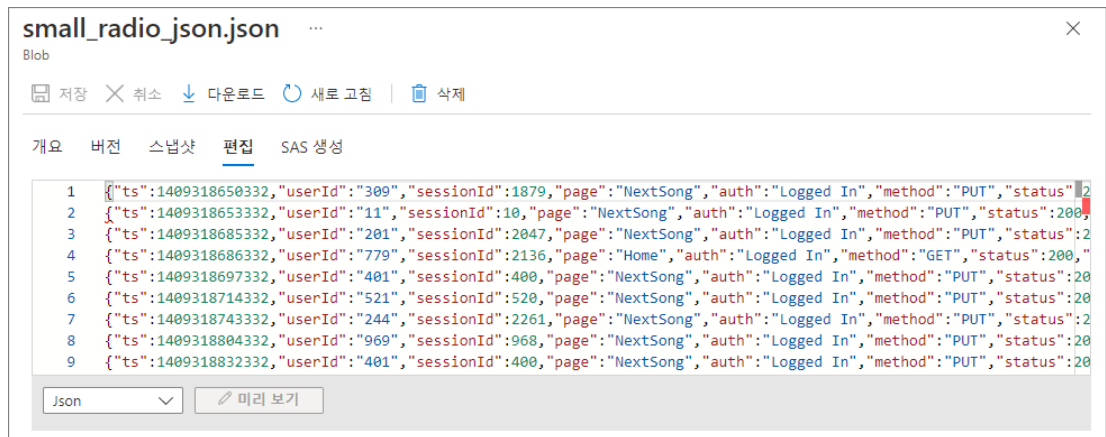
- 새로 만든 [staging 컨테이너] 블레이드로 이동한 후 메뉴에서 [업로드]를 클릭합니다. [Blob 업로드]에서 [찾아보기]를 클릭한 후 "small_radio_json.json" 파일을 선택하고 [업로드]를 클릭합니다.



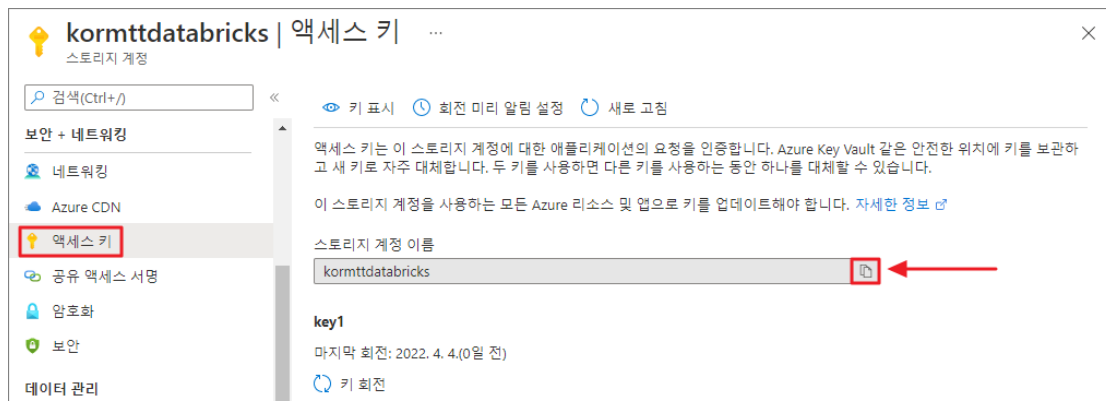
- 업로드한 파일을 선택하고 [... - 보기/편집]을 클릭합니다.



- 업로드한 파일을 클릭하여 아래와 같이 JSON 형식의 파일이 업로드 된 것을 확인합니다.



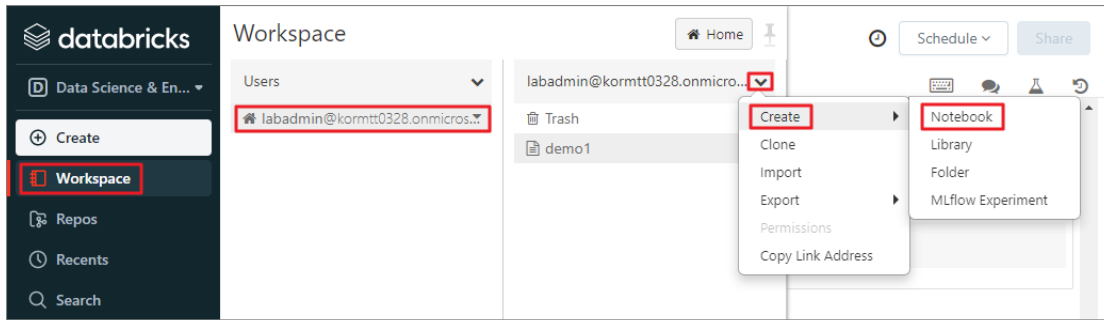
5. 이제 Databricks에서 이 스토리지 계정의 컨테이너에 연결할 때 사용할 스토리지 계정 이름과 SAS 토큰을 생성해야 합니다. [스토리지 계정] 블레이드로 이동한 후 [보안 + 네트워킹 - 액세스 키]를 클릭합니다. 표시되는 스토리지 계정 이름을 메모장에 복사합니다.



6. [스토리지 계정] 블레이드의 [보안 + 네트워킹 - 공유 액세스 서명]으로 이동한 후 아래와 같이 구성하고 [SAS 및 연결 문자열 생성]을 클릭합니다.
- 허용되는 서비스: "Blob" 만 선택합니다.
 - 허용되는 리소스 종류: "서비스", "컨테이너", "개체"를 모두 선택합니다.
 - 다른 모든 옵션은 기본값으로 선택하고 시작 및 만료 날짜가 충분한 기간인지 확인합니다.

7. 생성된 SAS 토큰 값을 메모장에 복사합니다.

8. Azure Databricks 워크스페이스의 좌측 메뉴에서 [Workspace]를 클릭한 후 자신의 계정을 선택하고 [Create - Notebook]을 클릭합니다.



9. [Create Notebook] 창에서 아래와 같이 입력한 후 [Create]를 클릭합니다.

- Name: demo2
- Default Language: Scala
- Cluster: demoCluster

10. 첫 번째 셀에서 다음과 같은 Scala 코드를 입력합니다. 이 코드는 앞서 구성했던 Azure 스토리지 계정에 연결하기 위한 것입니다. 다음과 같이 구성한 후 셀을 실행합니다.

- `<container_name>` 값을 스토리지 계정의 컨테이너 이름인 "staging"으로 변경합니다.
- `<storage_account_name>` 값을 메모장에 복사했던 스토리지 계정 이름으로 변경합니다.
- `<sas_token>` 값을 메모장에 복사했던 스토리지 계정의 SAS 토큰 값으로 변경합니다.

```
// Databricks Notebook 소스 지정
val containerName = "<container_name>"
val storageAccountName = "<storage_account_name>"
val sas = "<sas_token>"

val url = "wasbs://" + containerName + "@" + storageAccountName +
".blob.core.windows.net/"
var config = "fs.azure.sas." + containerName + "." + storageAccountName +
".blob.core.windows.net"
```

```

demo2 Scala
demoCluster
Cmd 1
1 // Databricks Notebook 소스 지정
2 val containerName = "staging"
3 val storageAccountName = "kormtttdatabricks"
4 val sas = "?sv=2020-08-04&ss=bft&srt=sco&sp=rwdlacupitfx&se=2022-04-04T19:26:46Z&st=2022-04-04T11:26:46Z&spr=https&sig=PQP0jKhtxj0P2yBR2C6KDwmroTqoQ0BDCDXFbIyA6IU%3D"
5
6 val url = "wasbs://" + containerName + "@" + storageAccountName + ".blob.core.windows.net/"
7 var config = "fs.azure.sas." + containerName + "." + storageAccountName + ".blob.core.windows.net"

containerName: String = staging
storageAccountName: String = kormtttdatabricks
sas: String = ?sv=2020-08-04&ss=bft&srt=sco&sp=rwdlacupitfx&se=2022-04-04T19:26:46Z&st=2022-04-04T11:26:46Z&spr=https&sig=PQP0jKhtxj0P2yBR2C6KDwmroTqoQ0BDCDXFbIyA6IU%3D
url: String = wasbs://staging@kormtttdatabricks.blob.core.windows.net/
config: String = fs.azure.sas.staging.kormtttdatabricks.blob.core.windows.net
Command took 5.44 seconds -- by labadmin@kormtt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:31:25 on demoCluster

```

11. 다음 셀에서 아래와 같은 명령을 실행합니다. 이 명령은 Databricks 유틸리티를 사용하여 스토리지를 탑재합니다. 이는 Windows나 Linux에서 일반 드라이브를 매핑하는 것과 동일한 작업입니다. 차이점은 Databricks에서는 드라이브 매핑 대신 Blob 스토리지를 매핑한다는 것이며 일반 Linux 명령을 사용하여 데이터를 마운트한 볼륨으로 복사할 수 있습니다. 실행 결과에서 정상적으로 Azure 스토리지가 마운트되었는지 확인합니다.

```

// 스토리지 계정 마운트
dbutils.fs.mount(
  source = url,
  mountPoint = "/mnt/staging",
  extraConfigs = Map(config -> sas))

```

```

demo2 Scala
demoCluster
Cmd 2
1 // 스토리지 계정 마운트
2 dbutils.fs.mount(
3   source = url,
4   mountPoint = "/mnt/staging",
5   extraConfigs = Map(config -> sas))

(1) Spark Jobs
res0: Boolean = true
Command took 23.93 seconds -- by labadmin@kormtt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:31:56 on demoCluster

Cmd 3
1 |
Scala

```

12. 다음 셀에서 아래와 같은 명령을 실행하여 Azure 스토리지 계정에 업로드했던 JSON 파일을 읽고 이 파일의 데이터를 표시합니다.

```

// JSON 파일을 읽고 내용을 출력
val df = spark.read.json("/mnt/staging/small_radio_json.json")
display(df)

```

The screenshot shows the Azure Databricks interface. At the top, there's a toolbar with 'demo2' and 'Scala' selected. Below it, a command box contains the following Scala code:

```
1 // JSON 파일을 읽고 내용을 출력
2 val df = spark.read.json("/mnt/staging/small_radio_json.json")
3 display(df)
```

The output shows a table with 7 rows and 8 columns. A red arrow points to the 'gender' column header.

| | artist | auth | firstName | gender | itemInSession | lastName | length |
|---|------------------------------|-----------|-----------|--------|---------------|------------|-----------|
| 1 | El Arrebato | Logged In | Annalyse | F | 2 | Montgomery | 234.57914 |
| 2 | Creedence Clearwater Revival | Logged In | Dylann | M | 9 | Thomas | 340.87138 |
| 3 | Gorillaz | Logged In | Liam | M | 11 | Watts | 246.17751 |
| 4 | null | Logged In | Tess | F | 0 | Townsend | null |
| 5 | Otis Redding | Logged In | Margaux | F | 2 | Smith | 135.57506 |
| 6 | Slightly Stoopid | Logged In | Alan | M | 39 | Morse | 198.53016 |
| 7 | NOFX | Logged In | Gabriella | F | 1 | Shelton | 130.2722 |

Showing all 25 rows.

Command took 2.83 seconds -- by labadmingkoratt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:32:41 on demoCluster

13. 다음 셀에서 아래 명령을 실행합니다. `select` 문을 사용하여 가져온 데이터의 일부 열 값만 표시합니다.

```
// 일부 열 값만 표시
val specificColumnsDf = df.select("firstname", "lastname", "gender", "location", "level")
display(specificColumnsDf)
```

The screenshot shows the Azure Databricks interface. At the top, there's a toolbar with 'demo2' and 'Scala' selected. Below it, a command box contains the following Scala code:

```
1 // 일부 열 값만 표시
2 val specificColumnsDf = df.select("firstname", "lastname", "gender", "location", "level")
3 display(specificColumnsDf)
```

The output shows a table with 7 rows and 5 columns. A red arrow points to the 'location' column header.

| | firstname | lastname | gender | location | level |
|---|-----------|------------|--------|--|-------|
| 1 | Annalyse | Montgomery | F | Killeen-Temple, TX | free |
| 2 | Dylann | Thomas | M | Anchorage, AK | paid |
| 3 | Liam | Watts | M | New York-Newark-Jersey City, NY-NJ-PA | paid |
| 4 | Tess | Townsend | F | Nashville-Davidson--Murfreesboro--Franklin, TN | free |
| 5 | Margaux | Smith | F | Atlanta-Sandy Springs-Roswell, GA | free |
| 6 | Alan | Morse | M | Chicago-Naperville-Elgin, IL-IN-WI | paid |
| 7 | Gabriella | Shelton | F | San Jose-Sunnyvale-Santa Clara, CA | free |

Showing all 25 rows.

Command took 0.84 seconds -- by labadmingkoratt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:33:14 on demoCluster

14. 다음 셀에서 이전 셀에서 정의했던 `specificColumnsDf` 데이터 프레임에서 `level` 열의 이름을 `subscription_type` 열 이름으로 변경합니다.

```
// 데이터 프레임의 기존 열 이름 변경
val renamedColumnsDF = specificColumnsDf.withColumnRenamed("level", "subscription_type")
display(renamedColumnsDF)
```


demo2 Scala

demoCluster

Cmd 5

```
1 // 데이터 프레임의 기존 열 이름 변경
2 val renamedColumnsDF = specificColumnsDF.withColumnRenamed("level", "subscription_type")
3 display(renamedColumnsDF)
```

(1) Spark Jobs

renamedColumnsDF: org.apache.spark.sql.DataFrame = [firstname: string, lastname: string ... 3 more fields]

Table Data Profile

| | firstname | lastname | gender | location | subscription_type |
|---|-----------|------------|--------|--|-------------------|
| 1 | Annalyse | Montgomery | F | Killeen-Temple, TX | free |
| 2 | Dylann | Thomas | M | Anchorage, AK | paid |
| 3 | Liam | Watts | M | New York-Newark-Jersey City, NY-NJ-PA | paid |
| 4 | Tess | Townsend | F | Nashville-Davidson--Murfreesboro--Franklin, TN | free |
| 5 | Margaux | Smith | F | Atlanta-Sandy Springs-Roswell, GA | free |
| 6 | Alan | Morse | M | Chicago-Naperville-Elgin, IL-IN-WI | paid |
| 7 | Gabriella | Shelton | F | San Jose-Sunnyvale-Santa Clara, CA | free |

Showing all 25 rows.

Command took 0.69 seconds -- by labadmin@kormtt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:33:45 on demoCluster

15. 다음 셀에서 이전에 만들었던 데이터 프레임을 새로운 뷰(view)로 생성하기 위해 아래 명령을 실행합니다.

```
// 새로운 뷰 만들기
```

```
renamedColumnsDF.createOrReplaceTempView("renamed")
```

demo2 Scala

demoCluster

Cmd 6

```
1 // 새로운 뷰 만들기
2 renamedColumnsDF.createOrReplaceTempView("renamed")
```

Command took 0.21 seconds -- by labadmin@kormtt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:34:29 on demoCluster

Cmd 7

1

Shift+Enter to run

16. 새로 만든 뷰에서 **subscription_type** 열 값 기준으로 구독 수를 확인해 볼 수 있습니다. 다음 명령을 실행합니다.

```
// SQL 쿼리로 구독별 수 확인
```

```
%sql
SELECT
  count(*) as count,
  subscription_type
FROM renamed
GROUP BY
  subscription_type
```

demo2 Scala

demoCluster

Cmd 7

```
1 %sql
2 SELECT
3   count(*) as count,
4   subscription_type
5 FROM renamed
6 GROUP BY
7   subscription_type
```

(2) Spark Jobs

Table Data Profile

| | count | subscription_type |
|---|-------|-------------------|
| 1 | 10 | free |
| 2 | 15 | paid |

Showing all 2 rows.

Command took 1.33 seconds -- by labadmin@kormtt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:35:23 on demoCluster

17. 이전 셀에서 실행한 결과를 데이터 프레임의 결과로 저장하려면 앞서 실행했던 `SELECT` 문을 Spark SQL로 캡슐화하고 이를 변수에 할당하면 됩니다. 다음 명령을 실행합니다.

```
// 새 데이터 프레임 만들기
val aggregate = spark.sql("""
SELECT
  count(*) as count,
  subscription_type
FROM renamed
GROUP BY
  subscription_type
""")
```

demo2 Scala

demoCluster

Cmd 8

```
1 // 새 데이터 프레임 만들기
2 val aggregate = spark.sql("""
3 SELECT
4   count(*) as count,
5   subscription_type
6 FROM renamed
7 GROUP BY
8   subscription_type
9 """)
```

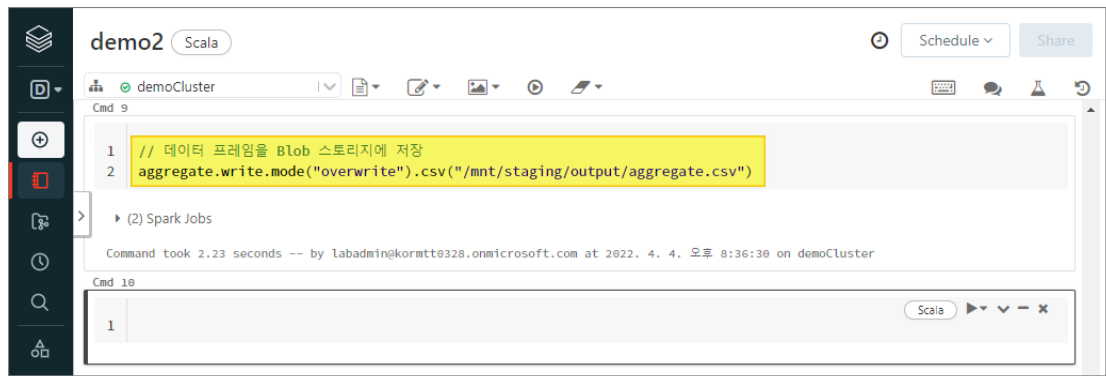
aggregate: org.apache.spark.sql.DataFrame = [count: long, subscription_type: string]

aggregate: org.apache.spark.sql.DataFrame = [count: bigint, subscription_type: string]

Command took 0.74 seconds -- by labadmin@kormtt0328.onmicrosoft.com at 2022. 4. 4. 오후 8:36:04 on demoCluster

18. 이제 데이터 프레임을 만들었기 때문에 이를 다시 Blob 스토리지에 저장할 수 있습니다. 이전 데이터 프레임 뒤에 `write.mode("overwrite")`를 사용하여 데이터 프레임의 결과를 CSV 파일 형식으로 Blob 스토리지에 덮어 씁니다.

```
// 데이터 프레임을 Blob 스토리지에 저장
aggregate.write.mode("overwrite").csv("/mnt/staging/output/aggregate.csv")
```



19. [스토리지 계정] 블레이드의 [데이터 스토리지 - 컨테이너]로 이동한 후 **staging** 컨테이너를 클릭합니다. [staging 컨테이너] 블레이드에서 Databricks에서 출력한 내용이 **output** 폴더에 생성된 것을 확인할 수 있습니다. **output** 폴더를 클릭합니다.

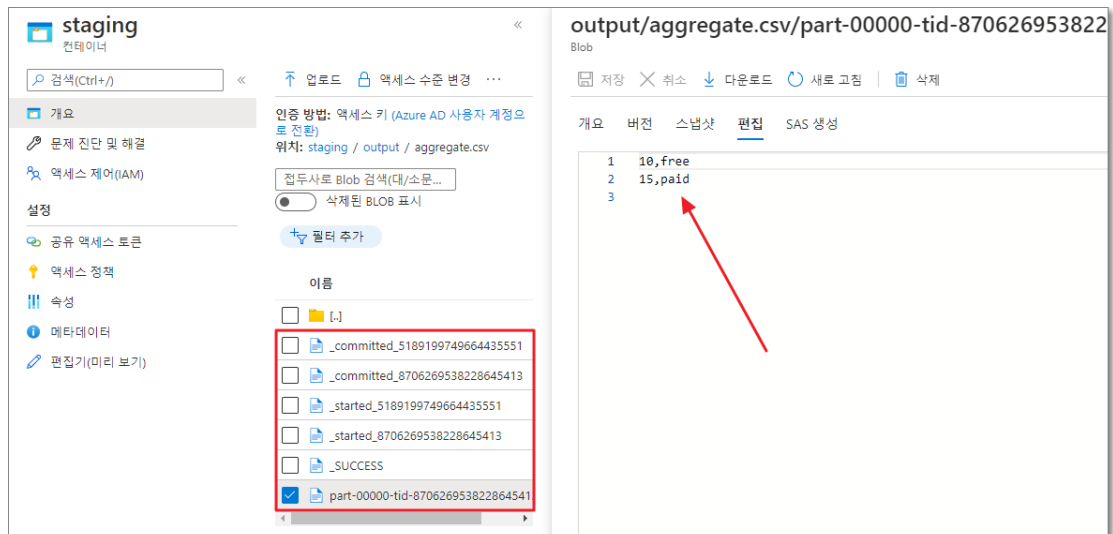


20. 이 폴더에 **aggregate.csv** 이름의 폴더와 메타데이터 파일이 생성된 것을 확인하고 **aggregate.csv** 폴더를 클릭합니다.



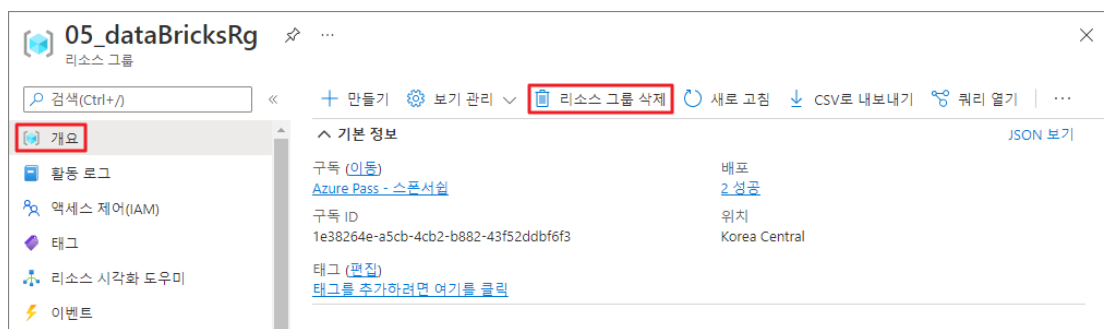
21. 다음과 같은 파일이 생성된 것을 확인할 수 있습니다.

- **part-xxxxx**: Databricks의 결과가 저장된 파일입니다.
- **_SUCCESS**: 파일 처리 상태를 표시하는 파일입니다.
- **_started_xxxxx**: 처리가 시작된 시간을 알려주는 파일입니다.
- **_committed_xxxxx**: 처리 중에 생성된 파일의 수를 나타내는 파일입니다.



TASK 03. 리소스 정리

1. Azure 포털에서 [05_dataBricksRg 리소스 그룹] 블레이드로 이동한 후 메뉴에서 [리소스 그룹 삭제]를 클릭합니다.



2. 리소스 그룹 삭제 확인 창에서 리소스 그룹 이름을 입력한 후 [삭제]를 클릭합니다.

 "05_dataBricksRg"을(를) 삭제하시겠습니까... ×



경고! "05_dataBricksRg" 리소스 그룹을 삭제하면 되돌릴 수 없습니다. 지금 수행하려는 작업은 취소할 수 없습니다. 계속 진행하면 이 리소스 그룹과 그 안의 모든 리소스가 영구적으로 삭제됩니다.

리소스 그룹 이름 입력:
 ✓

관련 리소스
이 리소스 그룹에 있는 2개의 리소스가 삭제됩니다.

| 이름 | 형식 | 위치 |
|--|------------------------|---------------|
|  dataBricksDemo | Azure Databricks Se... | Korea Central |
|  kormttdatabricks | 스토리지 계정 | Korea Central |