­­



**SeUng Jung**

**Data Life Cycle on AWS**

*Lab2-1. RDS to S3 with Glue­­*

**Table of Contents**

Lab 설명 4

Lab Architecture 4

RDS to S3 with Glue 6

# Lab 설명

이번 Lab은 ETL / Datawarehouse 관련 실습을 진행합니다.

서비스 운영을 위해 사용되는 DBMS(Amazon Aurora)와 NoSQL(DynamoDB)에 저장되어 있는 데이터를 원본 저장용 S3에 저장을 하고, 분석을 위해 가공된 데이터를 분석용 S3에 저장합니다. 데이터의 이동과 처리를 위해 서버리스 ETL 서비스인 Glue ETL을 사용합니다.

또한, 일부 분석용 데이터를 활용하여 Datawarehouse(Amazon Redshift)를 구성합니다.

분석을 위해 준비된 데이터는 Glue crawler 를 통해 Data Catalog를 생성하여 Athena, Redshift를 통해서 쿼리를 실행 합니다.

# Lab Architecture



단계별 상세 Lab은 다음과 같습니다.

**Lab 2-1. RDS to S3 with Glue**

Lab 2-2. DDB to S3

Lab 2-3. Data Transforming with Glue

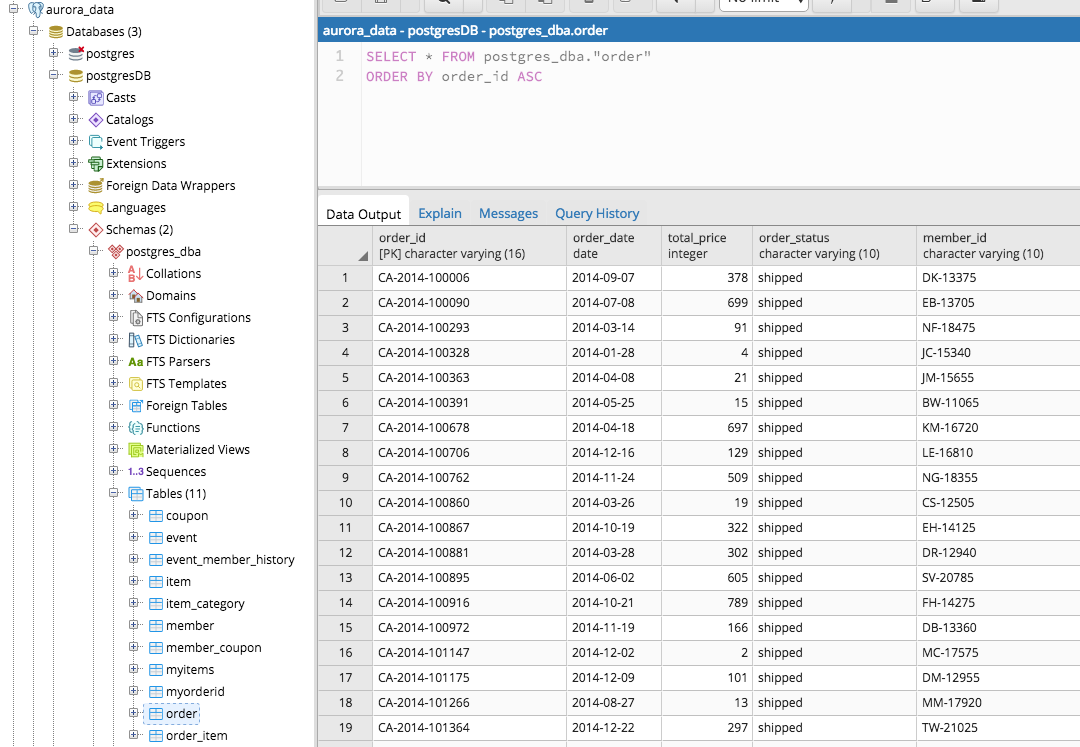
Lab 2-4. ETL to Redshift with Glue

Lab 2-5. Glue Data Catalog for Analytics

Lab 2-6. Query with Athena and Redshift spectrum

# RDS to S3 with Glue

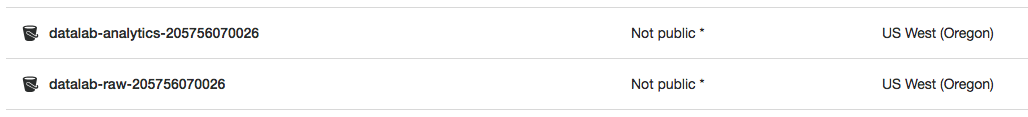
실습을 위한 환경에서는 원본이 되는 데이터는 Amazon Aurora(postgreSQL) DB에 저장되어 있습니다. 먼저 Aurora Aurora에 저장되어 있는 샘플 데이터의 구조는 다음과 같습니다.



AWS Glue 에서 Crawler를 생성할때 데이터베이스이름, 스키마 이름, 테이블 명 관련 내용을 include path 항목에 넣어서 필터링 할 수 있으므로 미리 확인이 필요합니다.  (예제에서는 데이터베이스명은 postgresDB, 스키마 이름은 postgres\_dba, 테이블 명은 order 입니다.)

다음으로는 Aurora의 데이터를 추출 하기 위한Glue Crawler를 생성해보겠습니다.

1. 우측 상단의 리전을 **Oregon (US-WEST)**으로 선택합니다.
2. AWS Management Console에서 S3를 선택하여 버킷을 확인합니다.



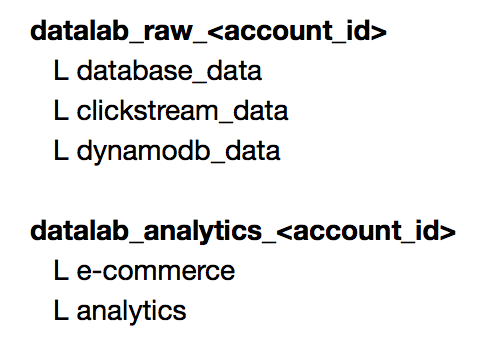
다음처럼

**datalab-raw-[account no]**

**datalab-analytics-[account no]**

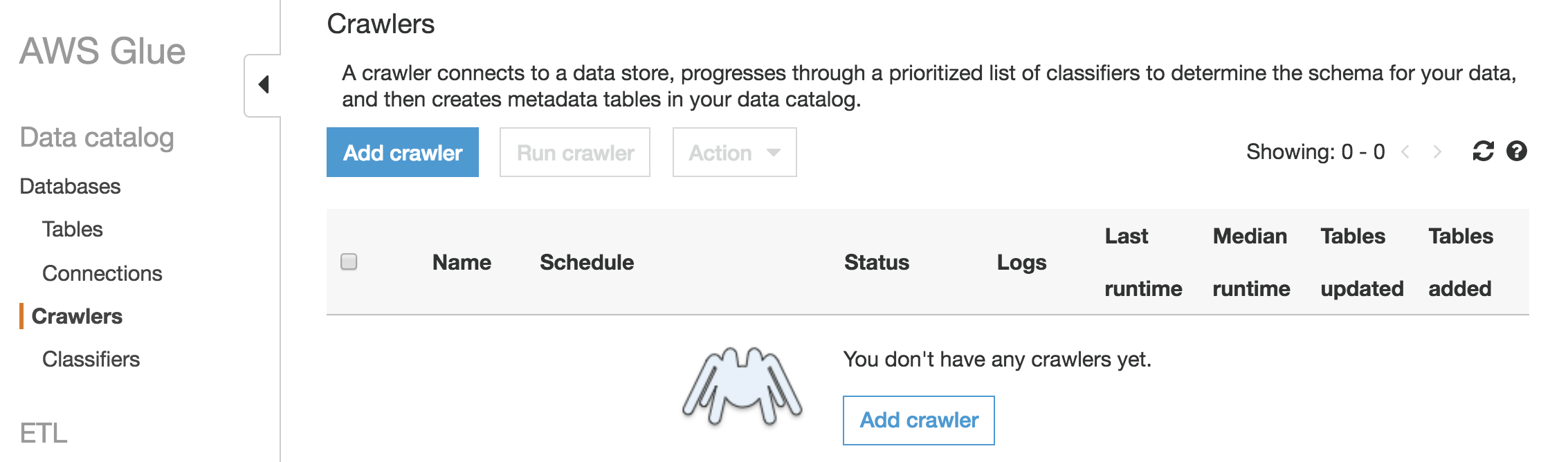
형태로 버킷이 만들어져 있습니다.

1. 다음처럼 그 아래에 폴더를 만들어 줍니다.





1. AWS Management Console에서 **AWS Glue** 서비스 화면으로 이동

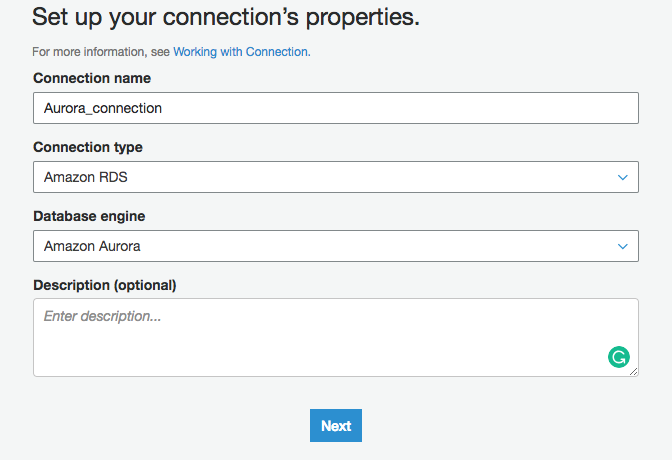


1. 좌측 메뉴의 Database – Connections 로 이동하여 Add connection 선택

Connection name : Aurora\_connection

Connection type : Amazon RDS

Database engine : Amazon Aurora



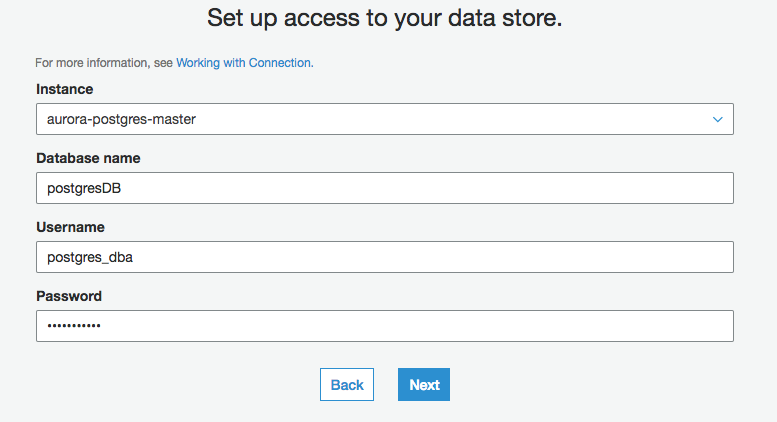
1. [Next]를 클릭하고, 다음 화면에서 상세 접속 정보 입력

Instance : aurora-postgres-master

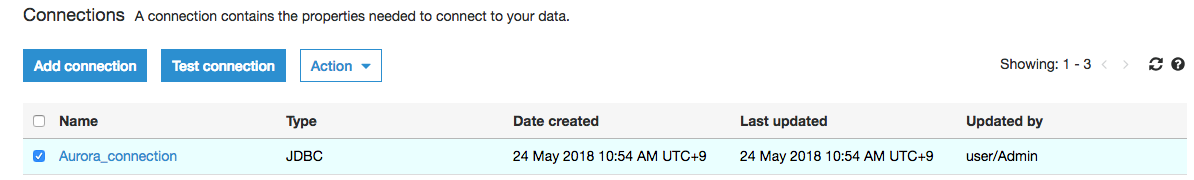
Database name : postgresDB

Username : postgres\_dba

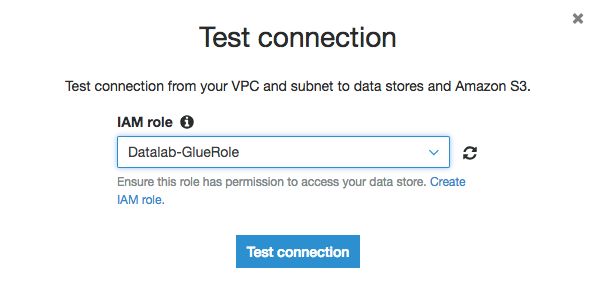
Password : postgres123



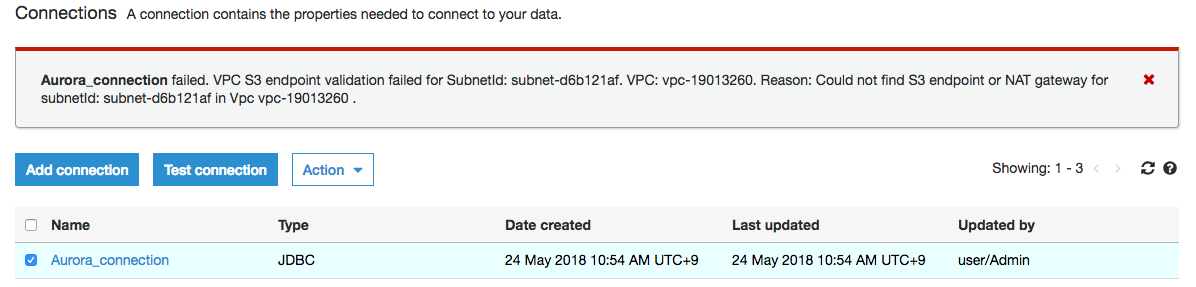
1. 생성된 리스트에서 **Aurora\_connection**에 체크를 하고 **[Test connection]** 실행



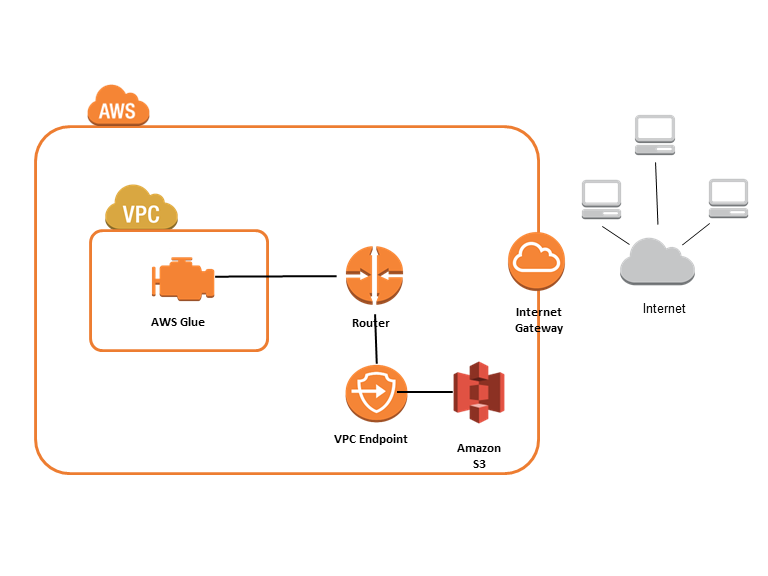
1. IAM Role에서 **Datalab-GlueRole**을 선택하고 **[Test Connection]**실행



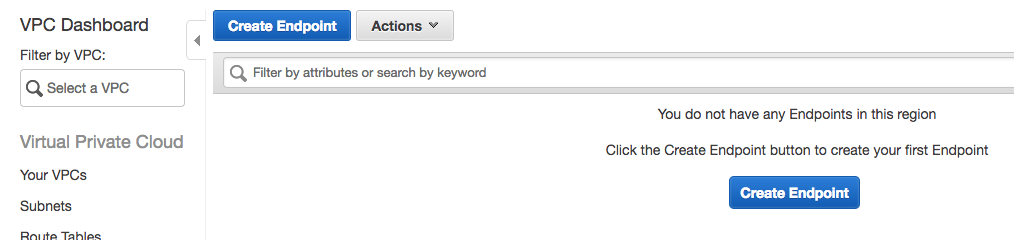
Note : 다음과 같은 에러 화면을 발견하면 추가 작업 필요



Glue에서 S3의 데이터 접근을 위해 VPC 엔드포인트를 생성해 주어야 합니다. VPC 엔드포인트를 이용하면 Glue를 통해 S3에 저장하는 데이터가 내부 네트웤 망을 사용하여 안전하게 접근 할 수 있도록 해줍니다.



1. 콘솔에서 VPC 서비스 콘솔로 이동,
2. **VPC – endpoint** 메뉴로 이동하여 Create Endpoint 클릭

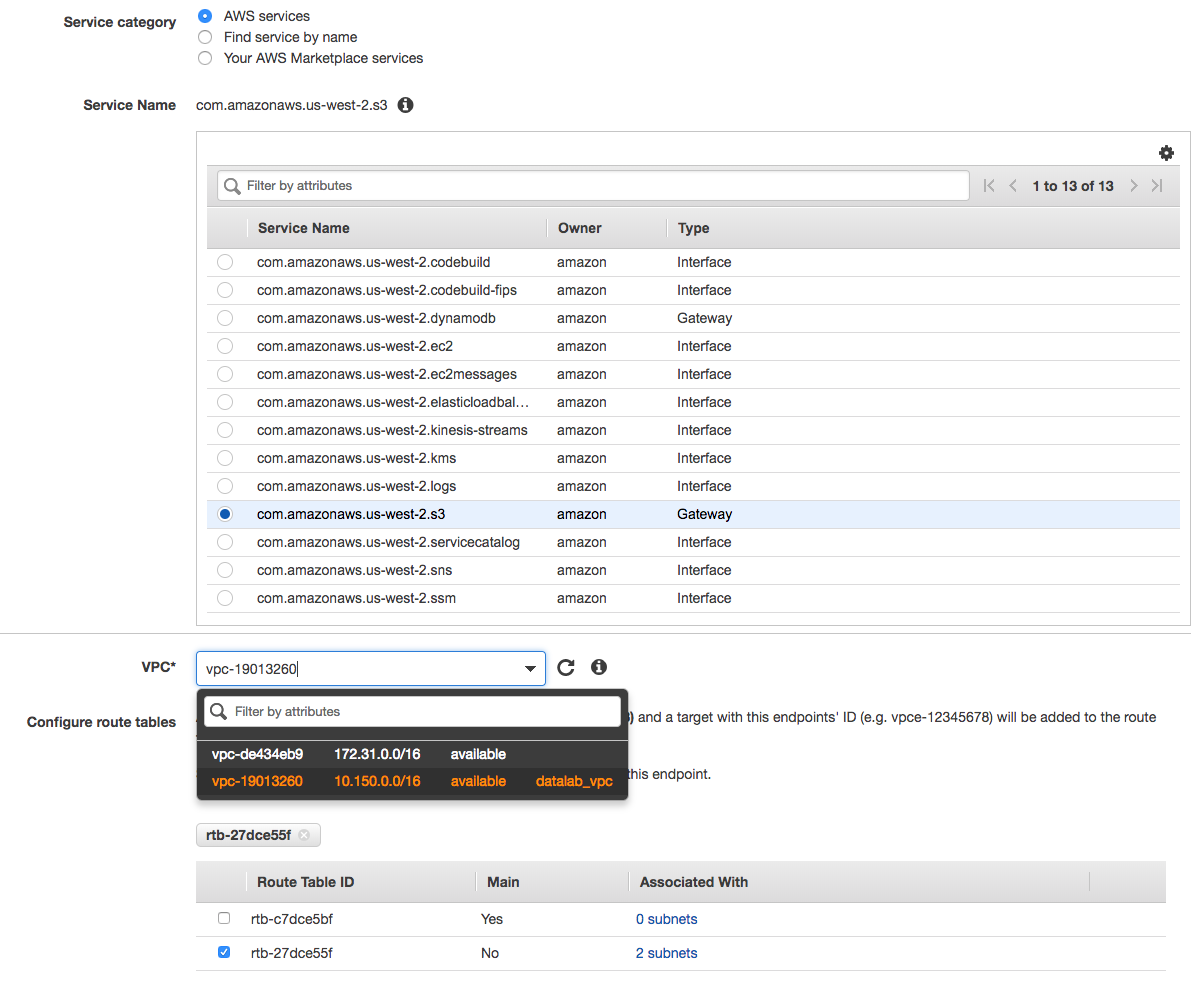


1. 다음과 같이 **s3로 끝나는 서비스명**을 선택하여 엔드포인트 생성

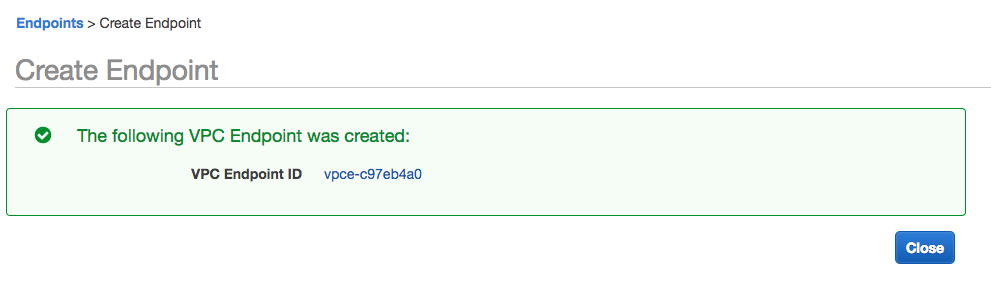
Service Name : com.amazonaws.us-west-2.s3

VPC : vpc-\*\*\*\*\*\*\*\* \*\*.\*\*\*.\*\*\*.\*\*\*, datalab\_vpc

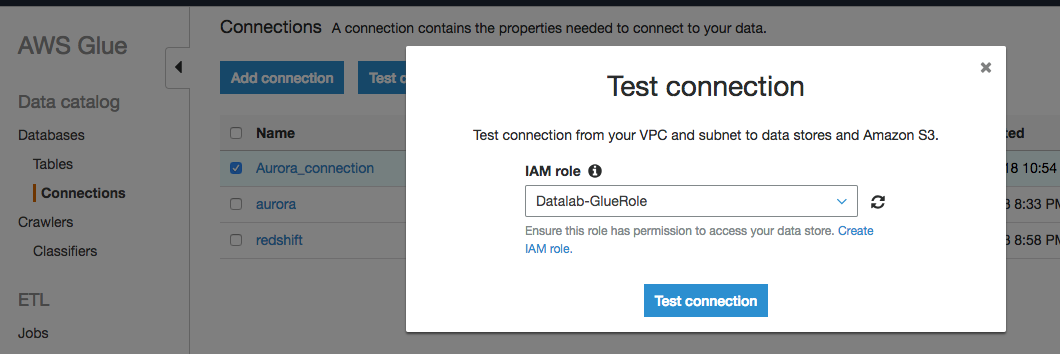
Route Table : 2개의 subnet을 포함하는 테이블 선택, 또는 위쪽 에러에 포함된 서브넷 명을 포함하는 Route Table 선택



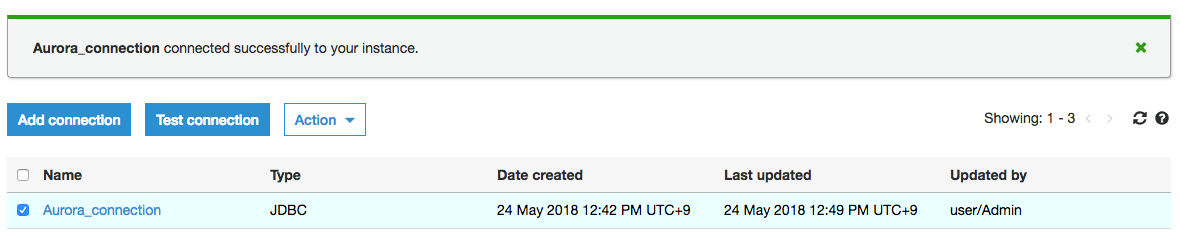
1. 다음과 같이 생성되었다는 메시지 확인



1. 다시 Glue 화면의 **Database – connection** 으로 이동하여 **[Test Connection]**

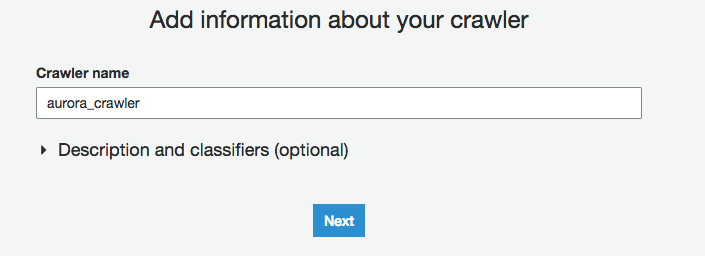


1. 접속 테스트가 성공하면 다음 단계로 넘어갑니다.

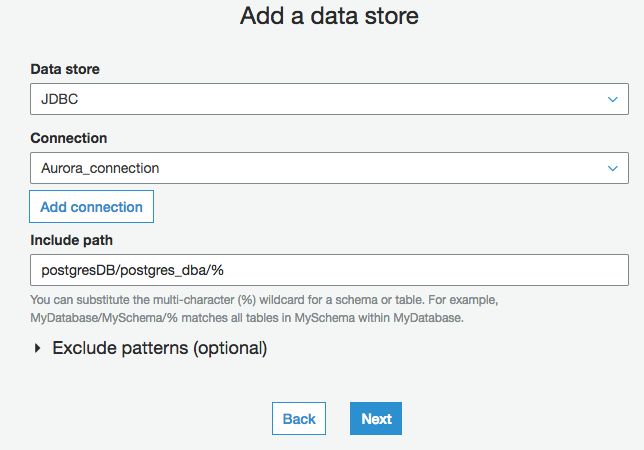


1. 좌측 네비게이션에서 Crawlers 메뉴를 선택하면 Add crawler 버튼을 통해 신규로 Crawler를 생성,

Crawler name : aurora\_crawler



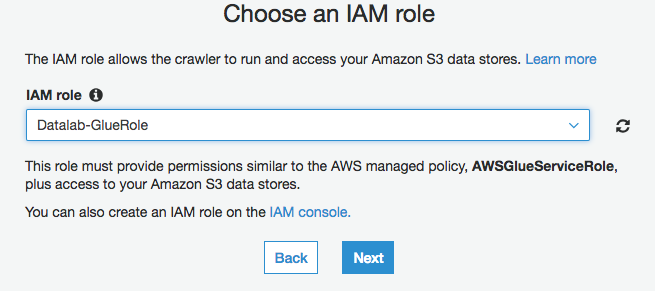
1. **[Next]** 버튼을 누르고 다음 화면에서 Data Store는 **JDBC**로 선택



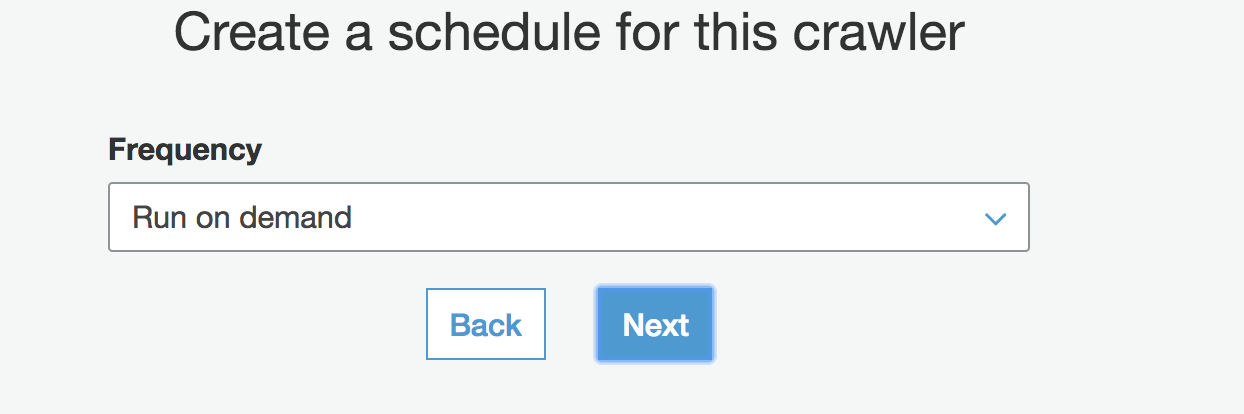
1. 앞에서 만든 **Aurora\_connection**을 선택하고 Include path에 대상이 되는 테이블의 경로를 적어줍니다.  Include path에 들어가는 데이터는 Database Name/Schema name/Table name 형태입니다.

**Include path :** **postgresDB/postgres\_dba/%**

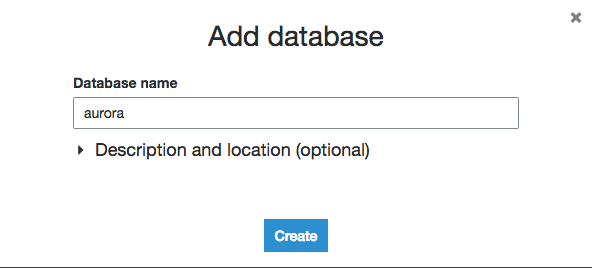
1. Add another data store는 **[No]**를 선택합니다.
2. IAM Role에서 **[Datalab-GlueRole]**을 선택합니다.



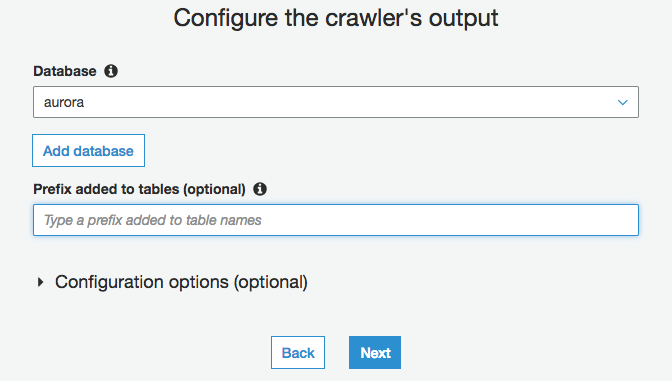
1. Schedule은 필요할 때만 실행하도록 **[Run on Demand]** 를 선택합니다.



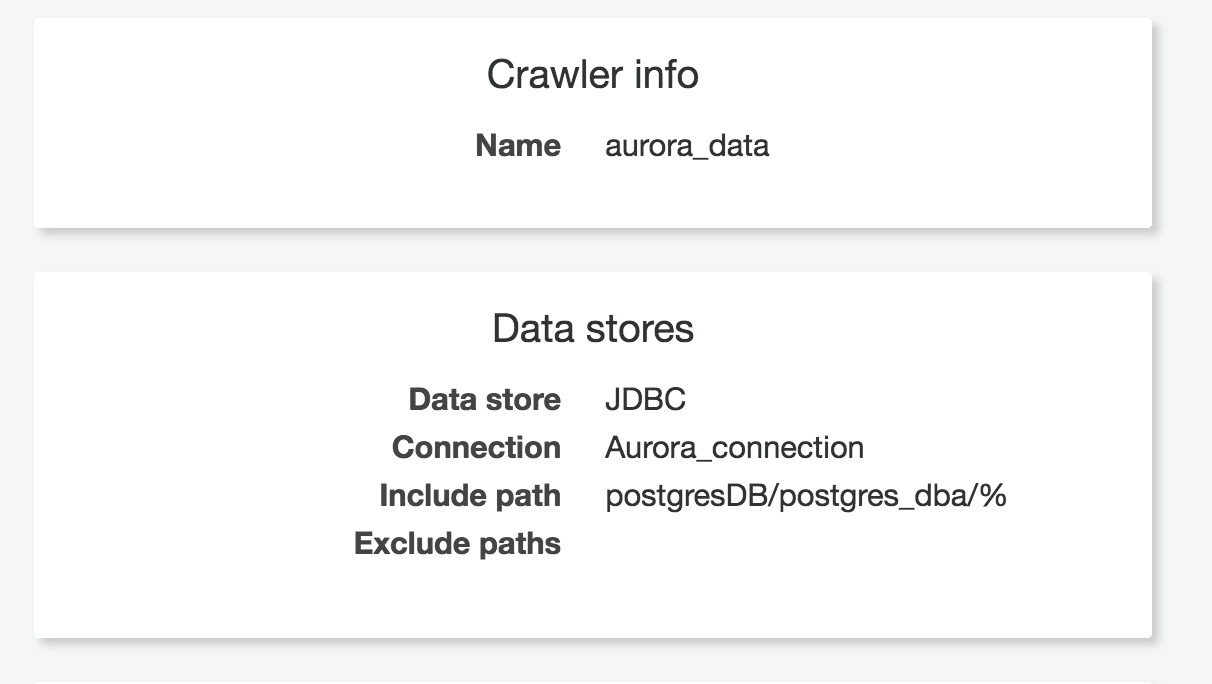
1. 만들고 있는 크롤러가 실행되면 수집되는 테이블정보를 어디에 저장할지 정해주는 부분입니다. Database이름을 정해주면 그 아래에 Aurora에 있는 테이블 정보가 저장됩니다. **[Add Database]**를 선택합니다. Database 이름은 **aurora** 로 정해줍니다.

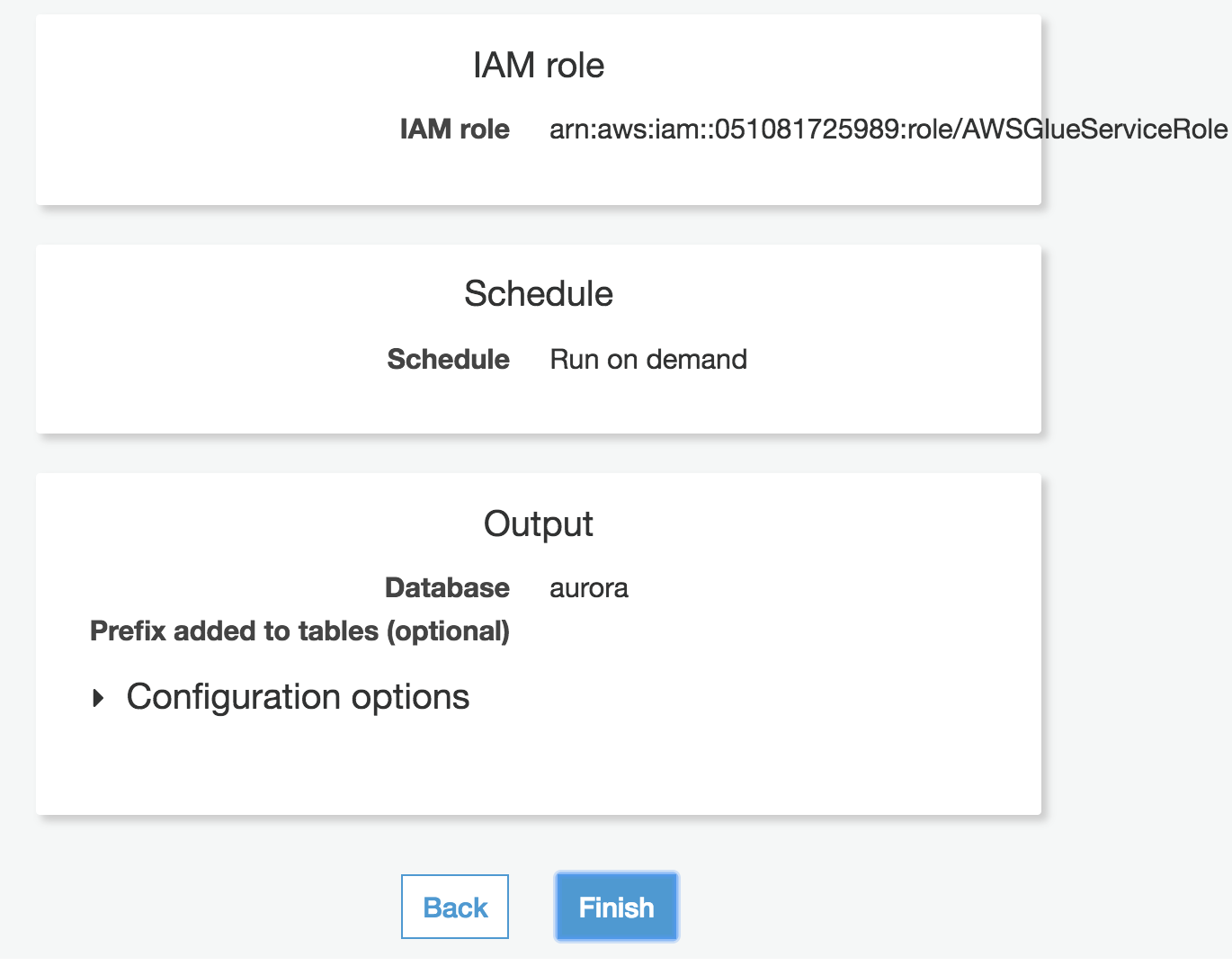


1. 생성한 Database를 선택하고 Prefix 부분은 생략하고 다음으로 넘어갑니다.

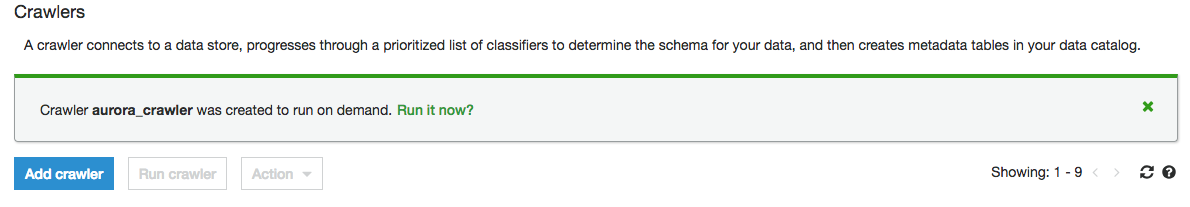


1. 설정된 정보를 확인하고 **[Finish]**를 누르면 crawler가 생성됩니다.





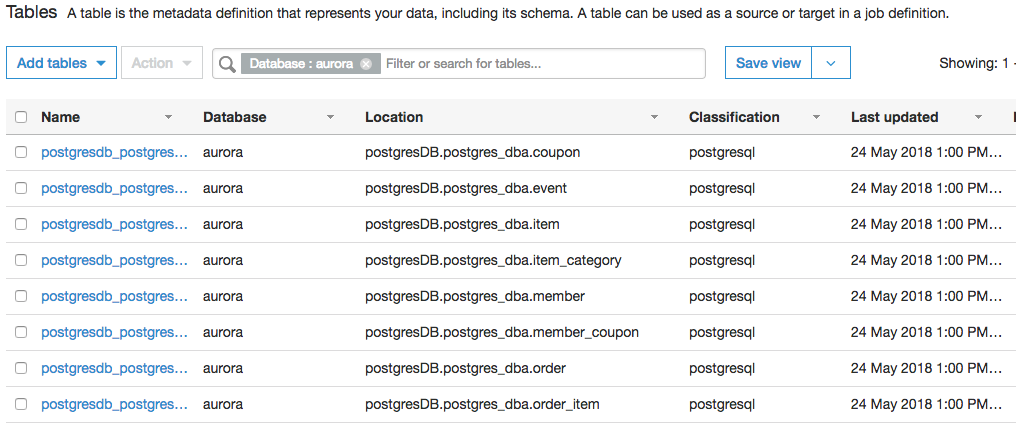
1. Crawler가 생성되면 다음처럼 실행 여부를 물어봅니다.



1. **[Run it now]**를 누르면 바로 Crawler가 실행됩니다.
2. 수분 이내로 Crawler 실행이 완료되면 Tables added에 추가된 테이블 수가 업데이트 됩니다.



1. 좌측 메뉴의 Databases를 선택하고 설정해준 이름(aurora)를 선택하고 아래 **[Tables in aurora]**를 선택하면 Table List를 볼 수 있습니다.



Note : 지금까지 작업은 Aurora에 있는 데이터를 Glue에서 식별하기 위한 정보를 수집한 부분입니다. 해당 정보를 통해 S3로 데이터를 수집하기 위한 Job을 생성해 보겠습니다.

1. 좌측 메뉴에서 Jobs를 선택하고 Add Jobs 버튼을 누릅니다.

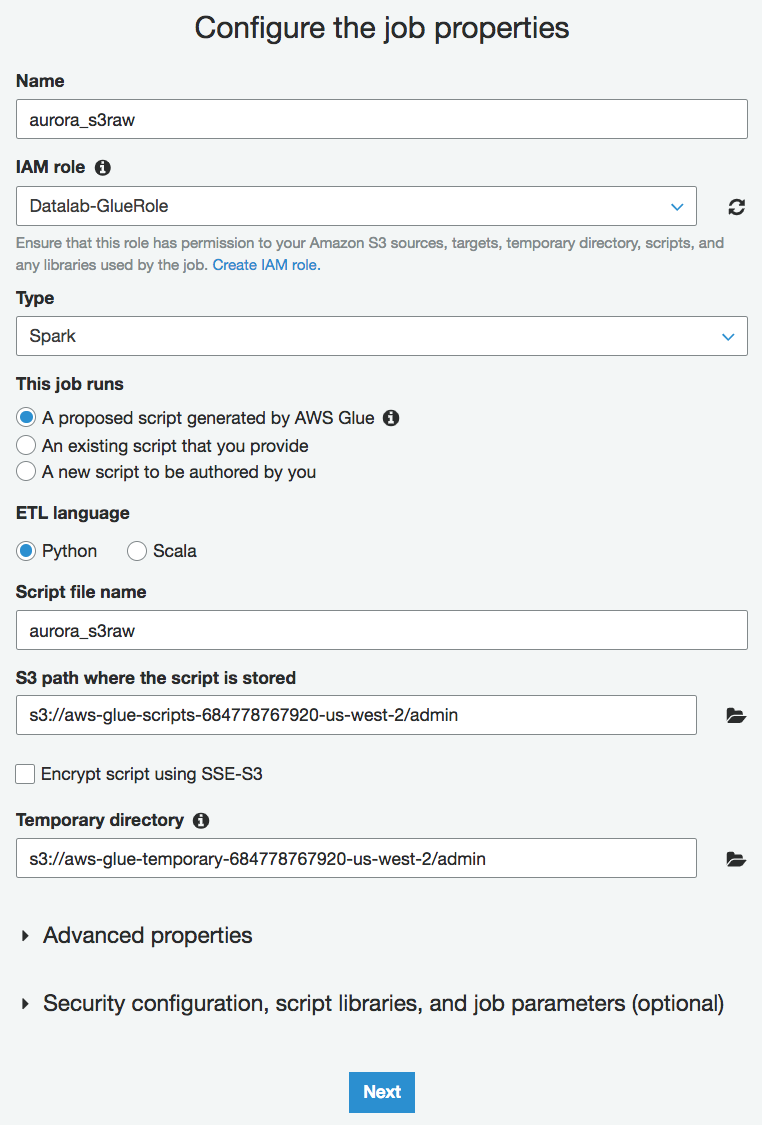
Name : aurora\_s3raw

IAM role : Datalab-GlueRole

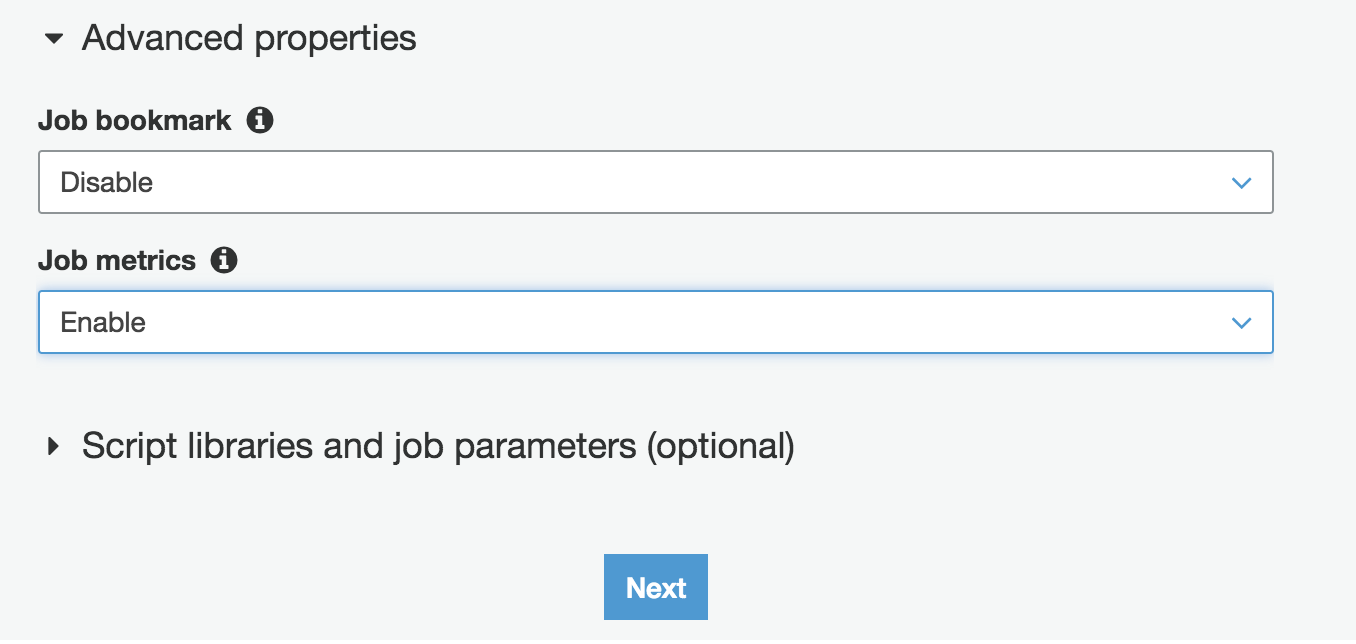
Type : Spark

This job runs : A new script to be authored by you

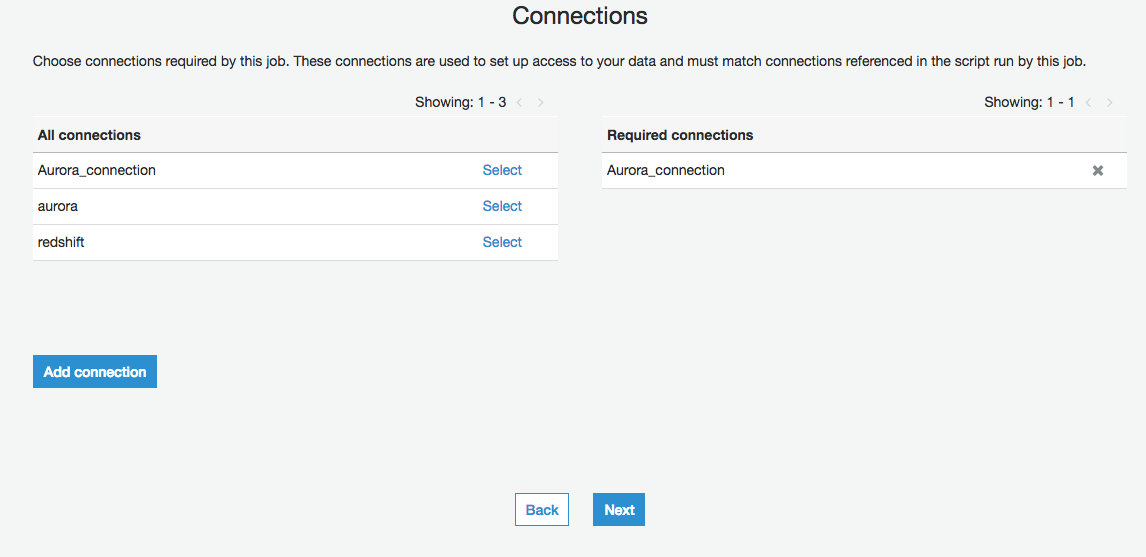
Other : default



1. 여기에서 Job Metrics를 Enable 하면, 이후 작업에 사용된 리소스와 데이터 사이즈 등의 지표를 확인 가능합니다.



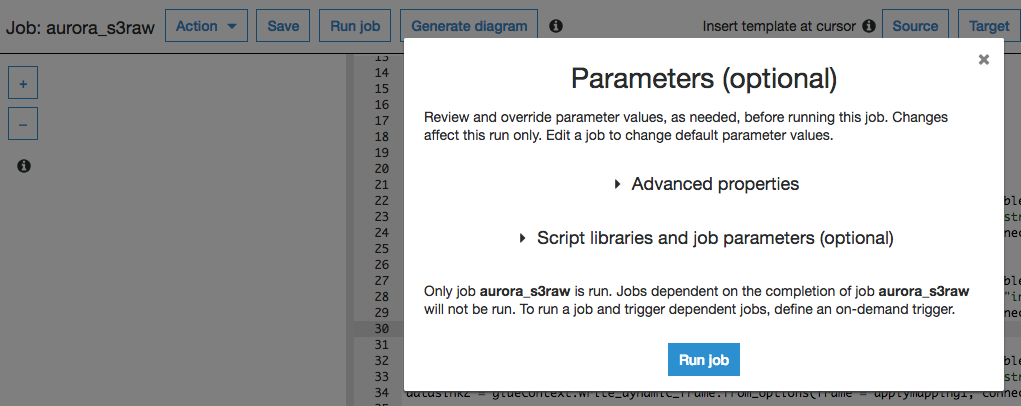
1. Connections 화면에서 앞에서 생성한 **[Aurora\_connection]** 을 선택합니다.



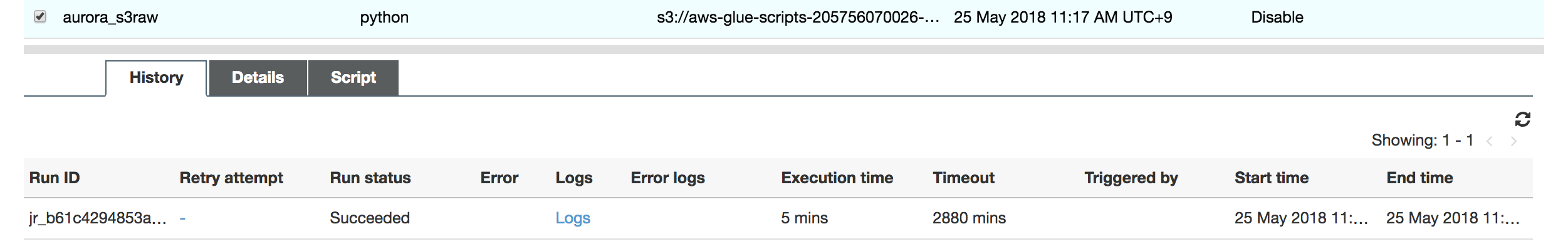
1. 코드 수정화면이 나오면 아래 코드 샘플을 붙여넣기 합니다. 그리고 반드시 최상단 S3bucket 명칭을 자신의 버킷명과 일치 시켜야 합니다.

|  |
| --- |
| import sys  from awsglue.transforms import \*  from awsglue.utils import getResolvedOptions  from pyspark.context import SparkContext  from awsglue.context import GlueContext  from awsglue.job import Job  ## @params: [JOB\_NAME]  args = getResolvedOptions(sys.argv, ['JOB\_NAME'])  sc = SparkContext()  glueContext = GlueContext(sc)  spark = glueContext.spark\_session  job = Job(glueContext)  job.init(args['JOB\_NAME'], args)  s3Bucket = "s3://datalab-raw-205756070026"  s3Folder ="/database\_data/"  ## order  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_order", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("member\_id", "string", "member\_id", "string"), ("order\_date", "string", "order\_date", "string"), ("order\_status", "string", "order\_status", "string"), ("country", "string", "country", "string"), ("shipping\_date", "date", "shipping\_date", "date"), ("total\_price", "int", "total\_price", "int"), ("city", "string", "city", "string"), ("order\_time", "string", "order\_time", "string"), ("state", "string", "state", "string"), ("postal\_code", "string", "postal\_code", "string"), ("region", "string", "region", "string"), ("order\_id", "string", "order\_id", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "order"}, format = "json", transformation\_ctx = "datasink2")  ## order\_item  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_order\_item", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("item\_count", "int", "item\_count", "int"), ("order\_date", "string", "order\_date", "string"), ("item\_id", "string", "item\_id", "string"), ("item\_price", "int", "item\_price", "int"), ("order\_time", "string", "order\_time", "string"), ("order\_id", "string", "order\_id", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "order\_item"}, format = "json", transformation\_ctx = "datasink2")  ## member  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_member", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("member\_id", "string", "member\_id", "string"), ("country", "string", "country", "string"), ("login\_id", "string", "login\_id", "string"), ("gender", "string", "gender", "string"), ("city", "string", "city", "string"), ("last\_login\_ymdt", "string", "last\_login\_ymdt", "string"), ("membership\_level", "string", "membership\_level", "string"), ("login\_password", "string", "login\_password", "string"), ("name", "string", "name", "string"), ("state", "string", "state", "string"), ("postal\_code", "string", "postal\_code", "string"), ("region", "string", "region", "string"), ("age", "string", "age", "string"), ("reg\_ymdt", "string", "reg\_ymdt", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "member"}, format = "json", transformation\_ctx = "datasink2")  ## member\_coupon  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_member\_coupon", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("member\_id", "string", "member\_id", "string"), ("coupon\_id", "int", "coupon\_id", "int")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "member\_coupon"}, format = "json", transformation\_ctx = "datasink2")  ## event  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_event", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("event\_id", "int", "event\_id", "int"), ("available\_count", "int", "available\_count", "int"), ("end\_ymdt", "string", "end\_ymdt", "string"), ("discount\_amount", "int", "discount\_amount", "int"), ("name", "string", "name", "string"), ("available\_yn", "string", "available\_yn", "string"), ("start\_ymdt", "string", "start\_ymdt", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "event"}, format = "json", transformation\_ctx = "datasink2")  ## item  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_item", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("item\_id", "string", "item\_id", "string"), ("price", "int", "price", "int"), ("name", "string", "name", "string"), ("description", "string", "description", "string"), ("reg\_ymdt", "string", "reg\_ymdt", "string"), ("item\_category\_id", "int", "item\_category\_id", "int")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "item"}, format = "json", transformation\_ctx = "datasink2")  ## item\_category  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_item\_category", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("category\_name", "string", "category\_name", "string"), ("category\_id", "int", "category\_id", "int"), ("parent\_category\_name", "string", "parent\_category\_name", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "item\_category"}, format = "json", transformation\_ctx = "datasink2")  ## coupon  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_coupon", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("coupon\_id", "int", "coupon\_id", "int"), ("end\_ymdt", "string", "end\_ymdt", "string"), ("discount\_amount", "int", "discount\_amount", "int"), ("name", "string", "name", "string"), ("start\_ymdt", "string", "start\_ymdt", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "coupon"}, format = "json", transformation\_ctx = "datasink2")  job.commit() |

1. **[Run job]**을 누르면 작업이 시작됩니다.



1. Glue Job은 리소스가 할당되는 준비시간이 약 10분정도 소요됩니다. 실질적인 작업 실행시간은 콘솔에서 확인 가능하며 작업 실행시간에 대해서만 과금이 됩니다. (최소 10분 단위 과금)
2. 작업이 완료되면 Jobs 리스트에서 다음과 같은 결과를 확인 가능합니다.



1. 직접 S3 버킷으로 이동하여 데이터가 잘 복제 되었는지 확인합니다.

