AWS Glue

Learning objectives

At the end of this topic you should have a grasp of the following topics:

- Understand what AWS Glue is and how it works;
- Describe some of the basic features of AWS Glue; and
- Build a simple ETL pipeline using AWS Glue.

Introduction to AWS Glue

One of the tools that AWS offers for ETL processes is AWS Glue. Reading its introductory page, AWS Glue is described as a serverless *data integration service* designed to provide the full spectrum of solutions one may need when building a data-driven project. This includes data preparation and joining that may be used for application development, analytics, and machine learning.

Glue forms part of the Platform as a Service (PaaS) category of ETL tools that enable data transformation on top of an existing cloud service provider's infrastructure. Using such PaaS-based ETL tools allows us to easily integrate with the other services hosted by that same cloud service provider. Another example of a PaaS-based ETL tool is Azure Data Factory. The idea here is that, due to the seamless integration with other cloud services, one may focus on the output of a data pipeline instead of its actual underlying technologies.

The serverless nature of AWS Glue can rapidly speed up the data development cycle, as no prior consideration around infrastructure, workflow isolation, or maintenance is required at a project's inception. In terms of pricing, this also has an upside as resources only incur costs during a workflow's execution - leading to significant savings for pipelines that seldomly run or execute on an irregular schedule.

How AWS Glue works?

So how does AWS Glue work? Let's take a deeper look.

Since AWS Glue is an orchestration tool, it uses other AWS services to perform ETL jobs. AWS Glue calls these services using API operations that transform data, store the logic of and creates various runtime jobs, and manages notifications on these jobs. This coordination can be configured within the AWS Glue console, connecting multiple services into a managed application that frees data engineers to focus purely on creating and monitoring ETL work.

AWS Glue supports all of the usual functionality that one would need in an ETL pipeline, such as using triggers to initiate jobs based on either events or a defined schedule. Furthermore, it supports many different data sources and configurable targets, with the capability to automatically form transformation scripts based on the inputs and outputs present within a workflow.

Why use AWS Glue?

So why should a team use AWS Glue? The features of AWS Glue that we've highlighted previously, including its serverless architecture, automated transformation scripts, and rich support of other AWS services, enable us to rapidly iterate over and deliver a customised ETL pipeline in minutes. This is substantial, as typically setting up the infrastructure for a robust ETL pipeline alone can take a significant amount of time. The ability to deeply customise a generated workflow also means that every solution can be tailored exactly to business requirements.

Let's list the benefits of AWS Glue to further clarify its use:

- **AWS Glue speeds up data integration** - AWS Glue is integrated across a wide range of AWS services, meaning that regardless of the format of our data we can quickly process them within AWS Glue and push the results to other services.
- **Automation of ones' data integration at scale** - this is due to AWS Glue automating most of the effort around building and maintaining the ETL jobs through the use of *crawlers* to identify source datasets and dataset formats, and to suggest schemas and transformations related to this data.
- **Serverless maintenance** - this benefit comes from Glues serverless infrastructure. Simply pay for Glue and all the infrastructure is maintained without user developer input.

Use case: event-driven ETL pipelines

Within this section, let's consider an example of an event-driven ETL pipeline with AWS Glue. We've covered event-driven pipelines in the previous sections of this course. As a reminder, an event-driven pipeline is activated or triggered based on some criteria. Usually, the criteria would be a certain file becoming present within a designated area within our system. Once located, our data pipeline will start running its processing scripts.

In the example presented in Figure 1, our specified directory takes the form of an S3 bucket. A Lambda function is used to search the S3 bucket for files and trigger the Glue ETL jos, which are based upon AWS Glue Data Catalog metadata. Once the Glue ETL processing has been completed the data is loaded into Amazon Redshift and S3. This example also contains CloudWatch integration so that logs and notifications about ETL jobs get pushed to CloudWatch.

Let's ground the above pipeline in a real-world example. Imagine that we have data about climatic conditions that need to be processed as part of a natural disaster early warning system. The climate data gets dropped at random times of the day based on its availability from various APIs. Under these circumstances, we'd need an event-based trigger that starts our data pipeline once a new piece of data gets uploaded to a monitored storage service, such as S3. Once the new climate data is present in S3, a lambda function is invoked, triggering ETL jobs we've configured in Glue. These jobs would be responsible for calculating the risk score of a developing weather event. Glue jobs could be further used to process this result into a relational format, where it could be stored in Amazon Redshift for structured analysis, or captured back into S3 for logging purposes.

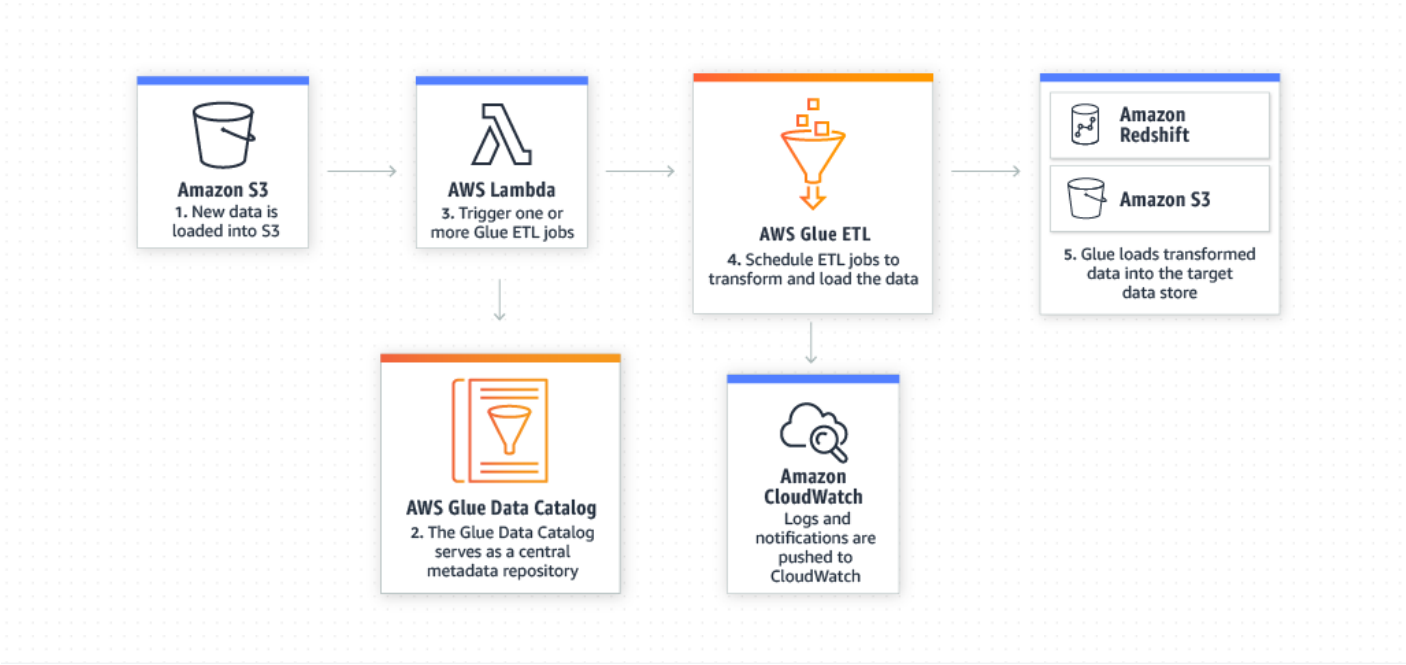

Figure 1: AWS Glue Architecture. Image Source

Other common use cases for AWS Glue include:

- **Creating a unified catalogue to find data across multiple data stores** This speaks to the use of Glue's Data Catalog, which allows us to store data from multiple data stores such as S3, RDS, Redshift and more in a central location for inspection and processing.
- **Enabling analysts or non-technical personnel to run and monitor ETL processes without coding** In some cases we may not have a team full of technical experts but still need to process and store data. This is where Glue has great utility, allowing non-technical users to create a full pipeline via point-and-click interactions on AWS Glue Studio's UI.
- **Explore data with self-service visual data preparation**. We can use AWS Glue's Databrew service to explore data directly from our data lake, data warehouse or databases. Databrew has 250 pre-built transformations, allowing us to gain insights rapidly around an extensive number of questions we may seek to investigate.
- **Build Views to combine and replicate data from multiple data stores using AWS Glue Elastic Views** This service is currently in preview, and requires registration before it's availbable for use. Sometimes we have data stored over multiple locations but need some combination thereof to create a new data source or perform analysis. AWS Elastic Views provides functionality to combine data originating from both relational and non-relational sources, and keeps them continuously updated in a SQL-like virtual table known as a material view.

Head over to the AWS Glue introductory page to get more detail on the use cases provided above.

AWS Glue Components

In the subsections that follow we cover some of the concepts necessary to understand how we may use AWS Glue. We frame this in the context of a practical example, where we take data from a specified source, transform it with a script, and load it into a specified target (ETL flow). This example, seen in Figure 2, helps convey the various stages that form part of Glue's functioning.

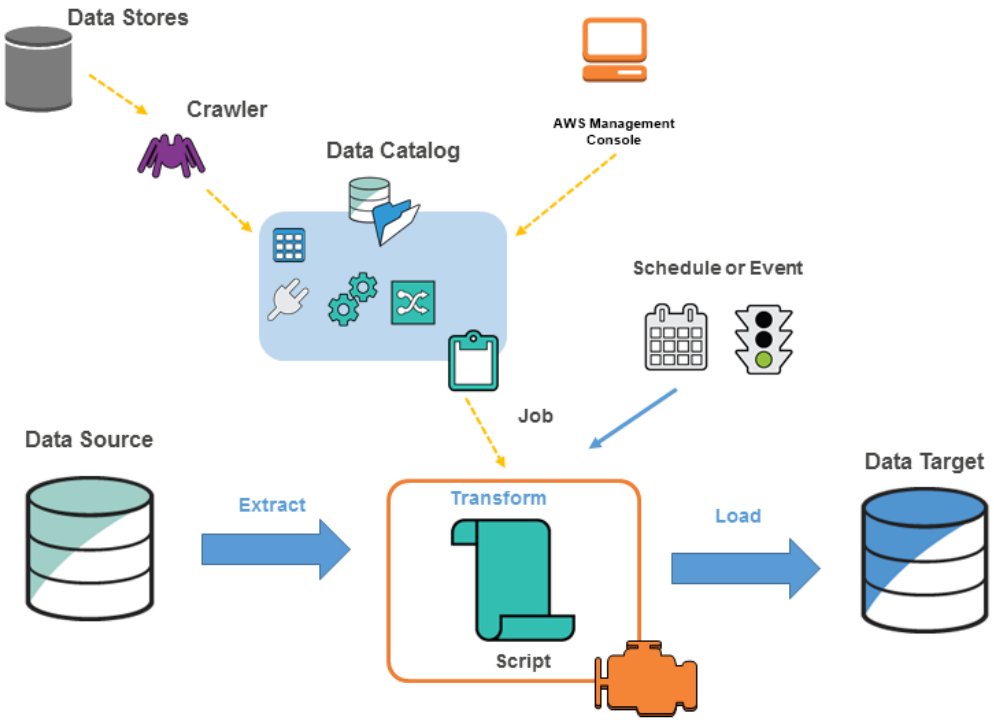

Figure 2: AWS Glue Architecture. Image Source

Data Discovery

The **AWS Glue Data Catalog** contains the metadata for all of our connected datasets, regardless of the AWS storage services they are stored within. Data within Catalog are provided by **AWS Glue Crawlers**, which are automated programs responsible for connecting to our data sources and determining their schema. For data streaming there is the option to use **AWS Glue Schema Registry**, a serverless component that enables validation and control over how the schema of streamed data evolves.

Data Transformation

Data transformation can often be a complex task. To deal with this complexity,**AWS Glue Studio** allows us to create data transformation jobs via a user interface without writing any code. The jobs are run using Apache Spark, which typically requires technical expertise to code and configure, but in this case AWS does everything in the background. Glue Studio generates code in either Scala or Python for use in Spark - allowing for further extensibility by data engineers who are comfortable in these languages.

Glue also provides the option to build ETL pipelines that run on a schedule, on-demand, or in an event-driven manner (such as new data drops). The ETL jobs can have multiple dependencies or run in parallel. Logging is automatically set up and logs for each job is pushed to **Amazon CloudWatch**, without the need to configure Cloudwatch when setting up an AWS Glue ETL pipeline. Concerning data streaming, AWS Glue can continuously consume data from sources such as **Amazon Kinesis** and **Amazon MSK**.

Data Replication

AWS Glue has the option to use **AWS Glue Elastic Views**, which enables us to create SQL-like views on data that is stored in different types of AWS storage. The service component uses an open-source SQL-compatible query language called **PartiQL**. PartiQL allows for queries and data manipulation across nested, semi-structured and structured data. This feature provides a simpler way of automating connections between multiple data sources and keeping them up to date in a specified target. AWS Glue Elastic Views supports DynamoDB, Aurora, and RDS as data sources, and Redshift, Elasticsearch Service, and S3 as targets.

Elastic Views allows for easy data replication of raw data across our data stores and saving this data to a target of our choice. Alternatively, we can use this connection to multiple data stores as a method of extracting and replicating valuable information in a target data store.

Data Preparation

AWS Glue can help clean and prepare our data for analysis and consumption by machine learning applications. Take the **FindMatches** feature for example, which finds imperfect duplicate data entries and asks the developer to confirm whether the match is correct. It systematically learns the criteria for matching a pair of records and builds that into an ETL job that you can run to find duplicates and matches accross entire databases.

The AWS Glue dashboard also provides script templates and interactive developer endpoints. Using the tools provided you can compose custom reading, writing and transformation functions. You can intreactively edit, debug and test code it generates for you before importing them as custom libraries into your AWS Glue ETL jobs.

AWS Glue Databrew provides an interactive visual interface where analysts and scientists can clean and normalise data without writing code. Using this service, data can be visualised directly from data stores such as data lakes, data warehouses, and databases.

For more detail on AWS Glue's features follow thislink.

AWS Glue exercise

Now that we have an overview of AWS Glue, and how it works at a high level, let's go through a practical example of a *load* process within an ETL

job, as implemented with AWS Glue. This example is focused on connecting the data and loading it into the database, but remember that in reality the Extract and Transform sections are likely to be more complicated.

Go ahead and follow the step-by-step instructions outlined below.

Exercise overview

For convenience, we provide links to the various exercise steps:

Setup AWS Glue

Step 1: Add source data

Back to exercise overview

- Start by opening up AWS Glue from the AWS management console.
- Upload the *books.csv* file to an S3 bucket you have created. This file can be foundhere

Step 2: Create a PostgreSQL RDS instance

Back to exercise overview

- Create a **Free Tier** PostgreSQL RDS instance, ensuring it has the following attributes:
  - Version: PostgreSQL 12.5-R1.
  - Under **DB instance class**, select the *Burstable classes* radio button > db.t2.micro.
  - Under **Connectivity**, use default VPC & Security group settings. Make sure that the instance is publicly accessible by selecting the appropriate radio button.
  - Use a SQL Client to connect to the RDS database. If you have forgotten how to do so, you may use this resource for reference.
  - Create a database called **BookCatalog**.
  - Create a table in the database using thecreate_table.sql file found in the GitHub repository.

Step 3: Create a VPC Endpoint to access your S3 bucket

Back to exercise overview

- In the AWS console, go to**Services** and navigate to **VPC**, which can be found under the**Networking & Content Delivery** section.
- In the navigation pane found on the left, click on**Endpoints**.
- Click on **Create Endpoint**.

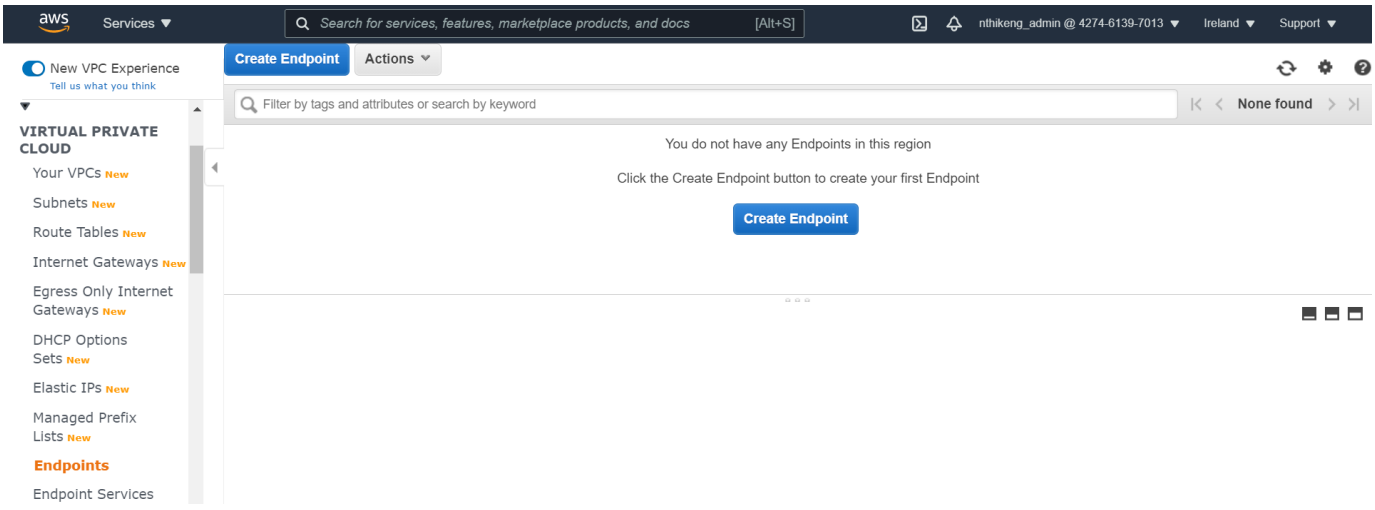

Figure 3: VPC endpoint dashboard.

- In the search bar, search for 's3'. This should bring up two options, make sure you select the one with a**Gateway** type.

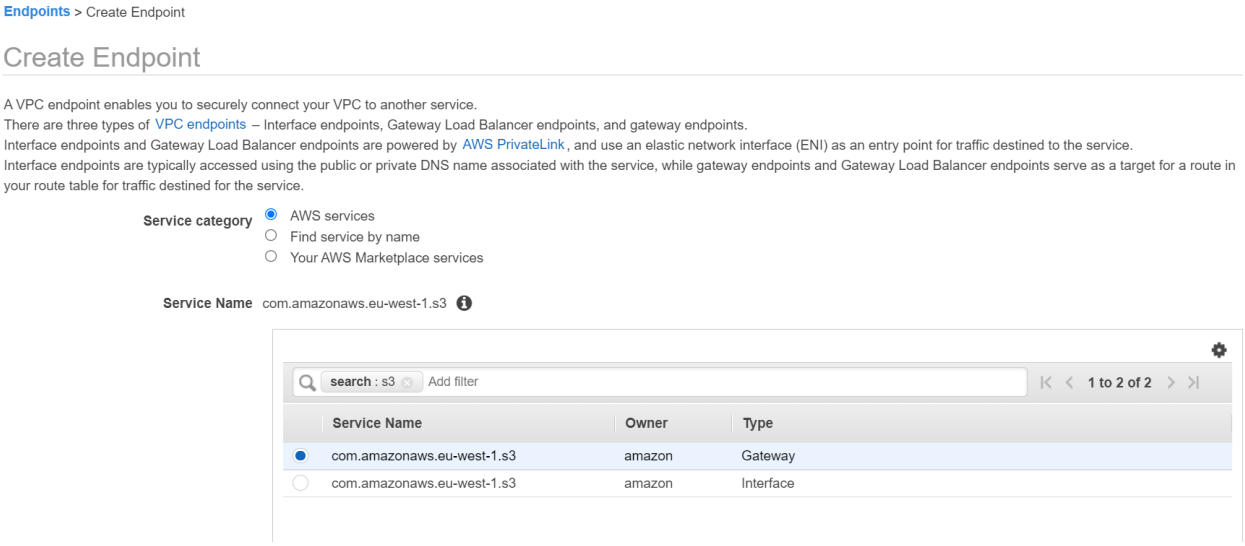

Figure 4: VPC endpoint service configuration.

- Make sure that your default VPC is chosen and select the default routing table that is provided.

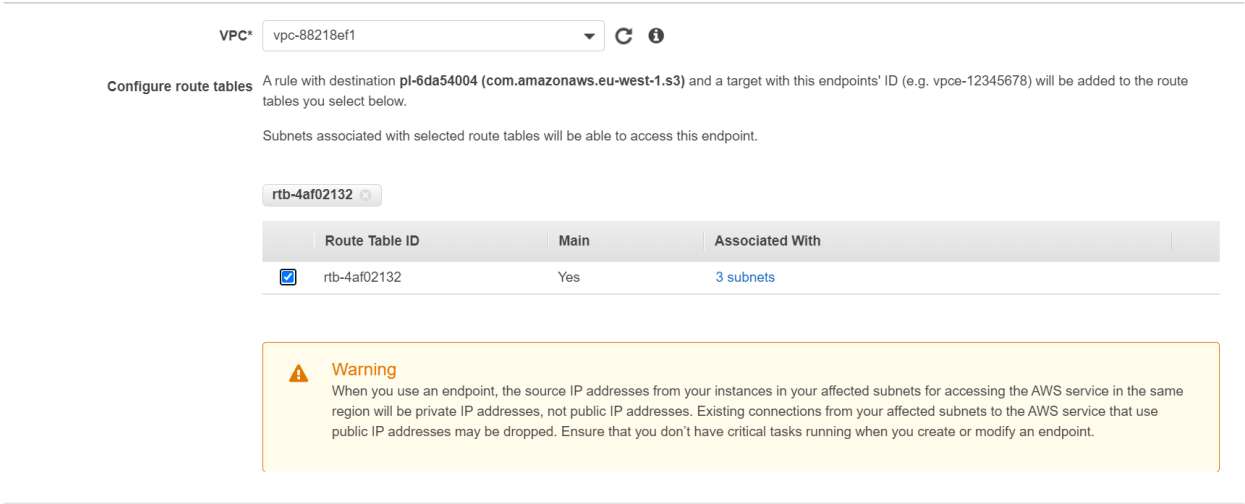

Figure 5: VPC endpoint routing table configuration.

- For policy settings, make sure that you allow full access and proceed to create the endpoint.

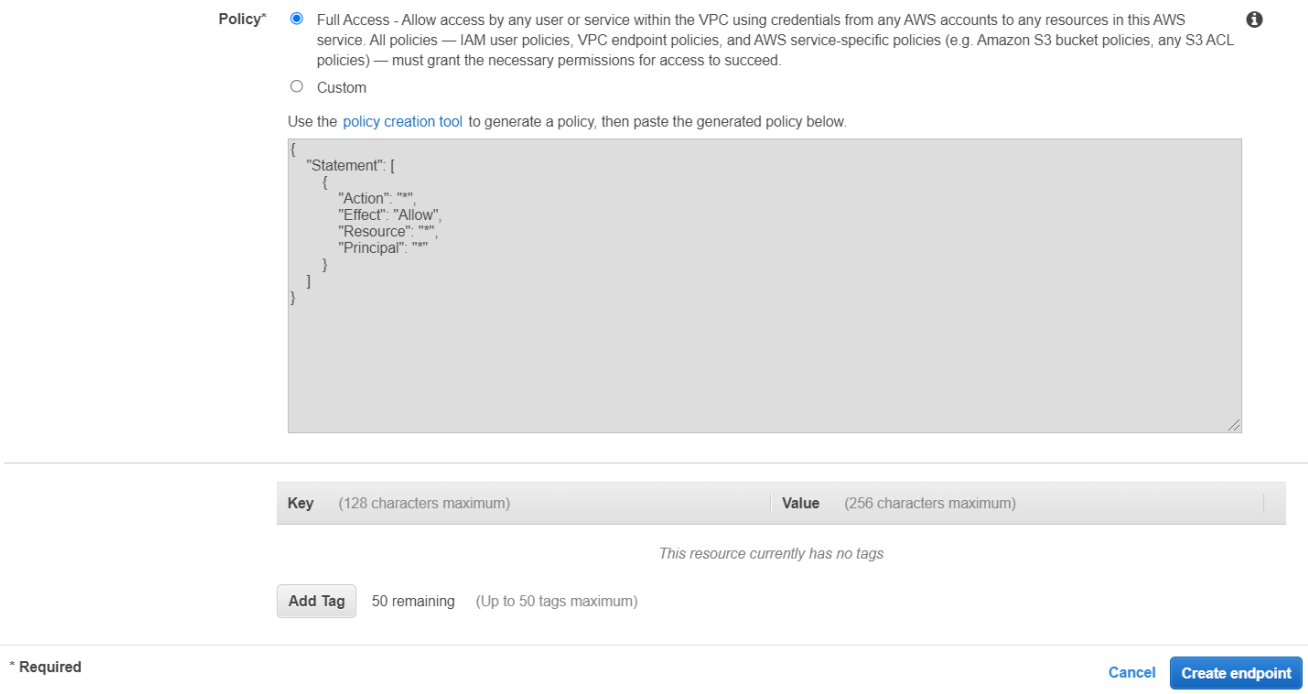

Figure 6: VPC endpoint policy assignment.

Step 4: Add AWS Glue classifier for the source data

Back to exercise overview

An AWS Glue classifier determines the schema of data sources in our pipeline. There are two choices when creating a classifier; we either write our own or select from a catalogue of pre-built versions. For this exercise, we will use the latter. Here we need to refer to our CSV data in the S3 bucket so that Glue knows where to look for our data.

- In the management console, click on **Services** and navigate to **AWS Glue** under **Analytics**.
- In the navigation bar on your left, click on **Classifiers**.
- Click on **Add classifier**.

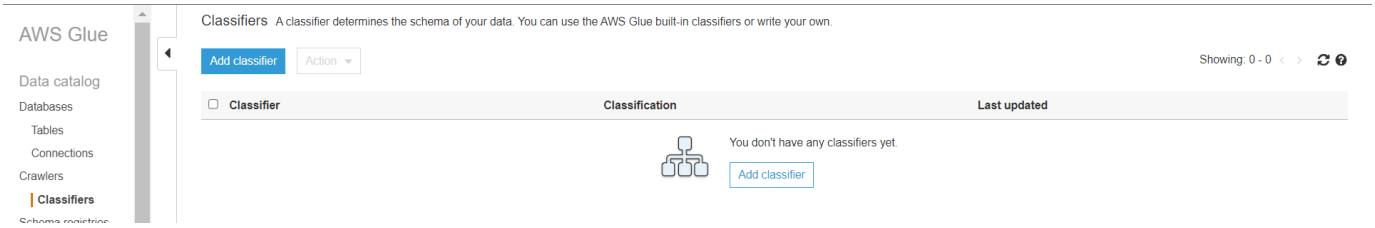

Figure 7: Classifier dashboard.

- Give the classifier a name
- Select **CSV** for classifier type
- Select **Has Headings** for the column headings option
- Leave the other settings as default
- Click **Create** to complete classifier creation

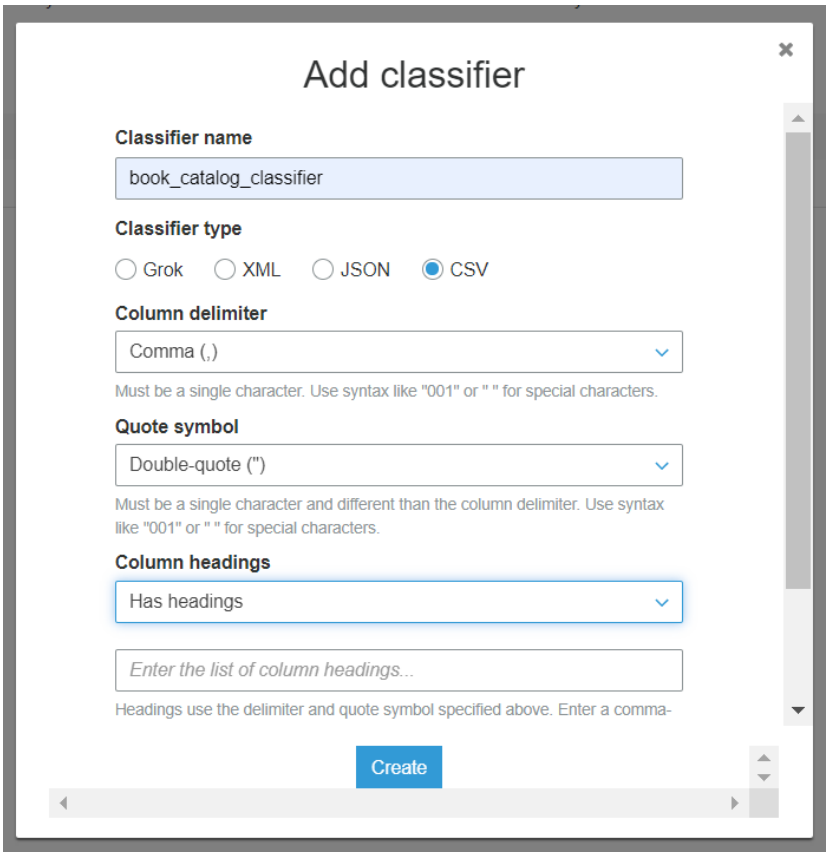

Figure 8: Classifier configuration window.

Step 5: Create crawler for CSV data source

Back to exercise overview

A crawler connects to a data store, iterates through a list of classifiers to determine the schema of the associated data, and uses the resulting findings to create metadata tables in our project's data catalogue. Knowing this, let's create a crawler for our source data and refer to the classifier created in the previous step.

- In the navigation bar on your left, click on "Crawlers".
- Click on "Add Crawler".

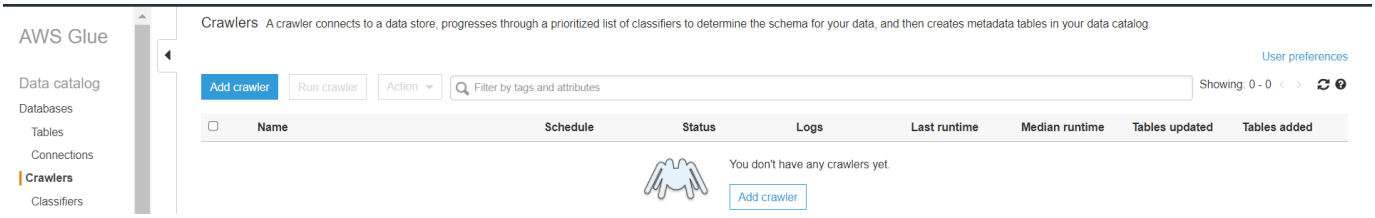

Figure 9: Crawler dashboard.

- Give the crawler a name.
- Click the arrow to expand additional options. Under custom classifiers, add the classifier created in the previous step.
- Click on **Next** to proceed.

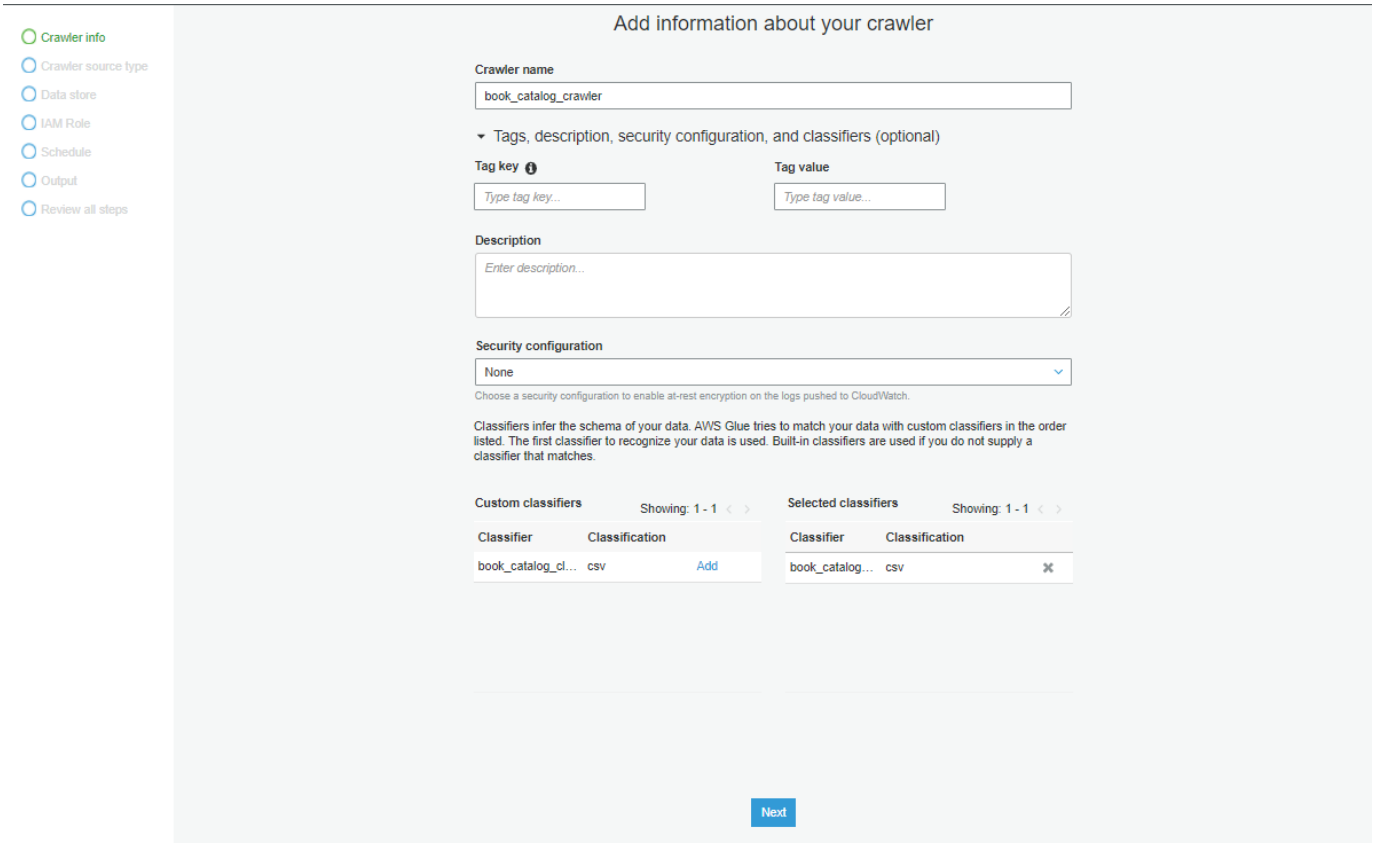

Figure 10: Crawler information.

- Leave defaults for crawler source type.
- Click on **Next** to proceed.

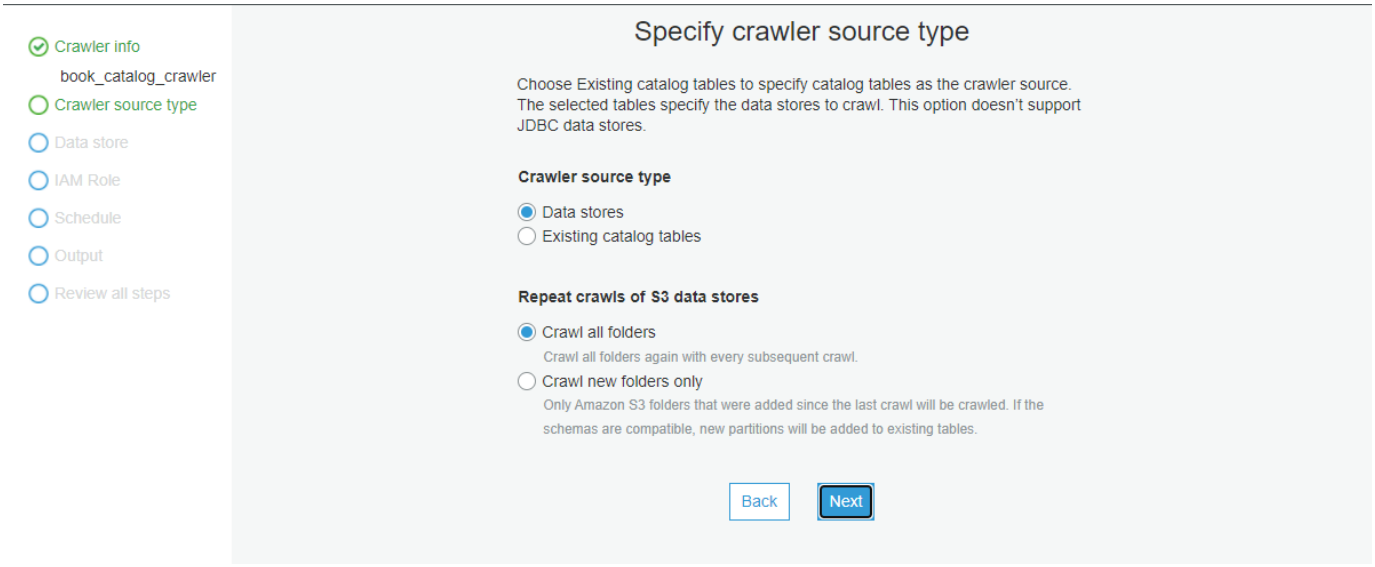

Figure 11: Crawler source type configuration.

- Choose S3 as the data store.
- Choose the file path of your CSV data by clicking the folder icon selecting the uploaded CSV file.
- Click on **Next**.

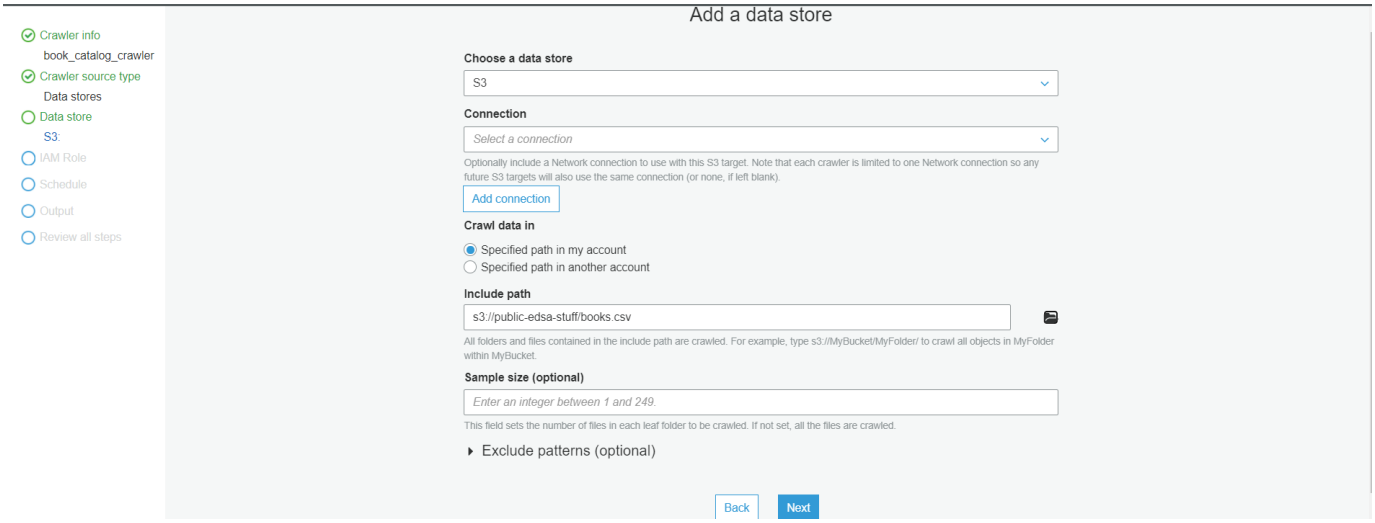

Figure 12: Crawler data store configuration.

- When asked whether to add another data store, select **No** and then click **Next** to proceed.
- Click on the **Create an IAM role** radio button and give the new IAM role a name.
- Briefly open a new tab and go to the **IAM Role** settings in the AWS Console and attach the following policies to your newly created role:
    - AmazonS3FullAccess.
    - AmazonRDSDataFullAccess.
- Resume the crawler configuration and click on **Next** to proceed.

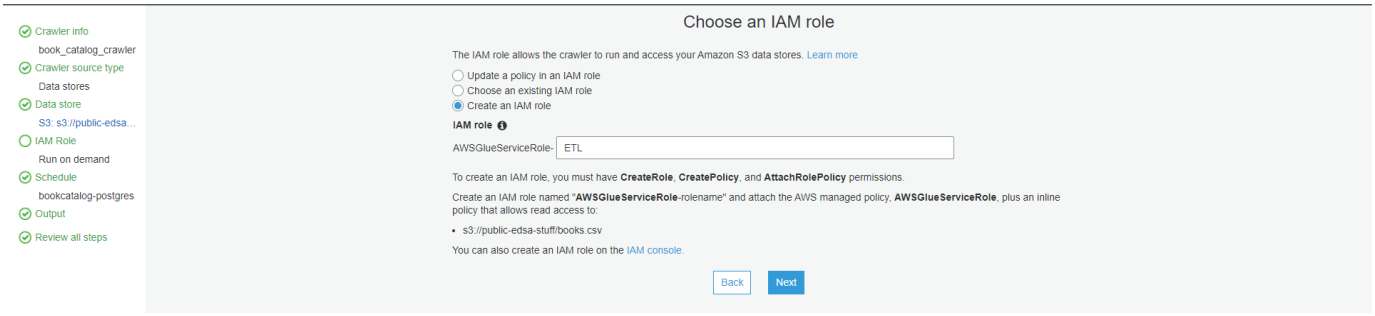

Figure 13: Crawler IAM role configuration.

- Select "Run on demand" for the schedule.
- Click on **Next** to proceed.

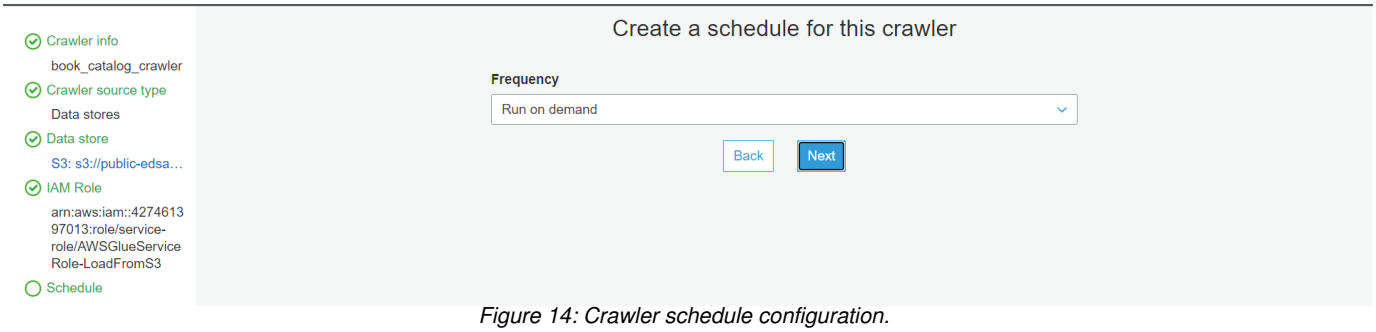

Figure 14: Crawler schedule configuration.

- Click on **Add database**. This should create a pop-up window enabling you to set up a database representing your data catalogue in AWS Glue. Create a name for your database and click **Create**. This is the database that will create and store tables in the **data catalog** for the items in the **S3 bucket**.

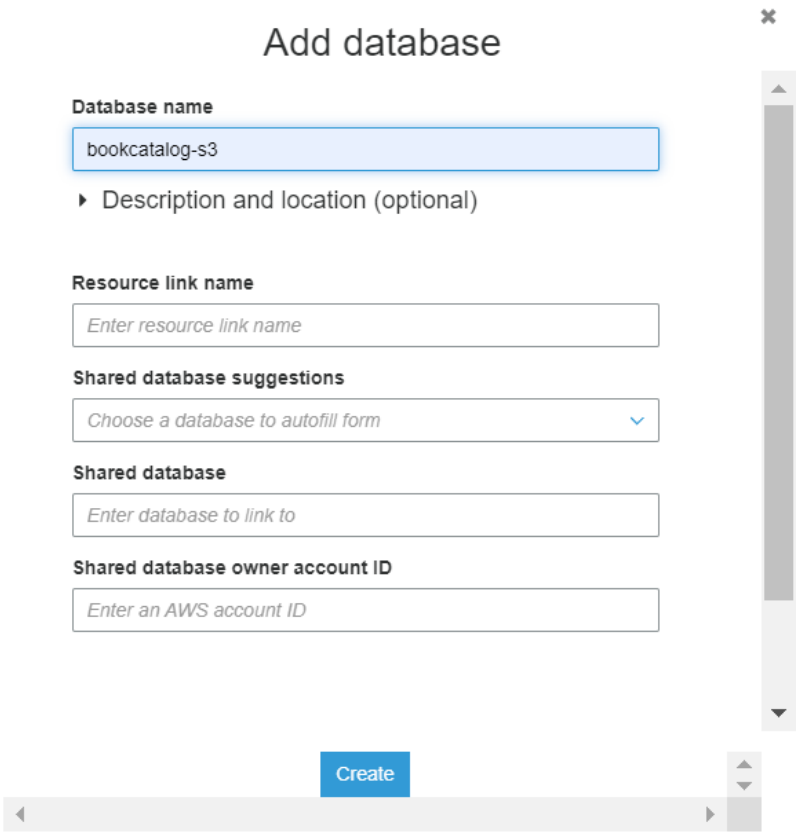

Figure 15: Crawler database creation.

- Make sure that the name of your database is selected in the*"Database name"* drop-down menu shown after this configuration.
- Click **Next** to review all steps.

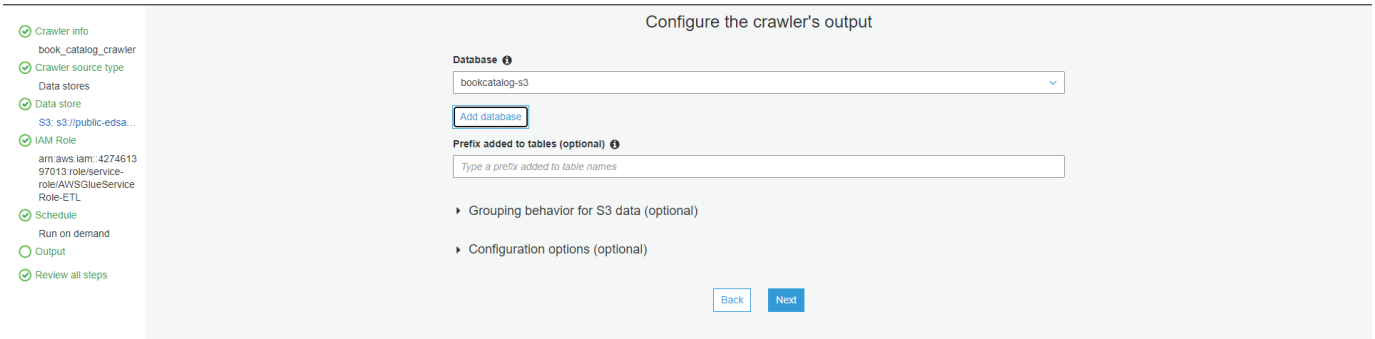

Figure 16: Crawler database configuration.

- Verify all the configuration details of your crawler and click**Finish** when done.

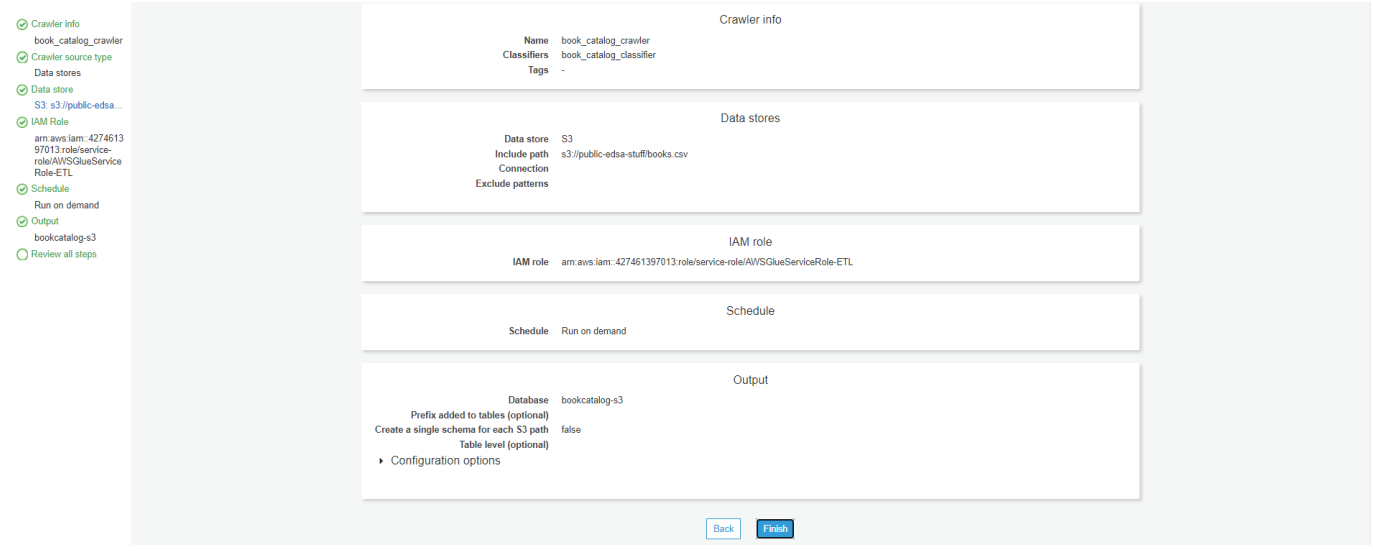

Figure 17: Review crawler configuration.

- Once created, select the crawler via its corresponding checkbox.

- Click on **Run crawler**. If you encounter any issues, please make sure that your endpoint and IAM Roles were set up correctly. Some additional help is provided within the additional resources section of the Train.

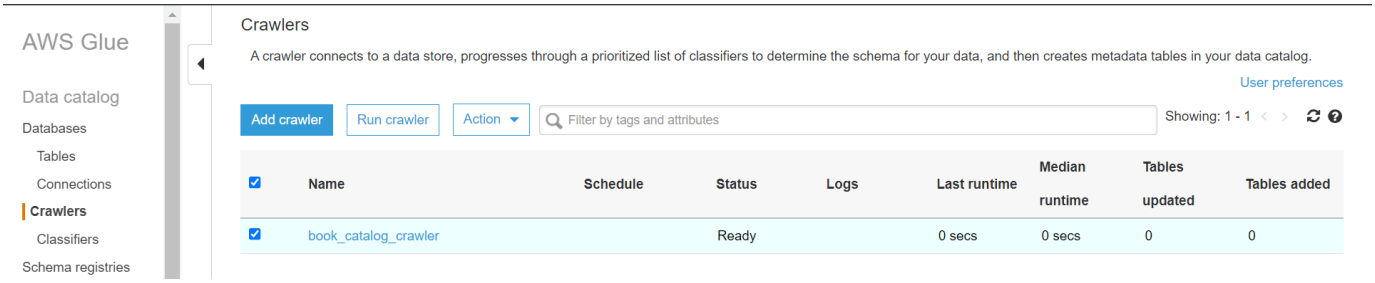

Figure 18: Run crawler.

- To see if your crawler creation was successful, navigate to**Tables** and ensure that a new table has been created.

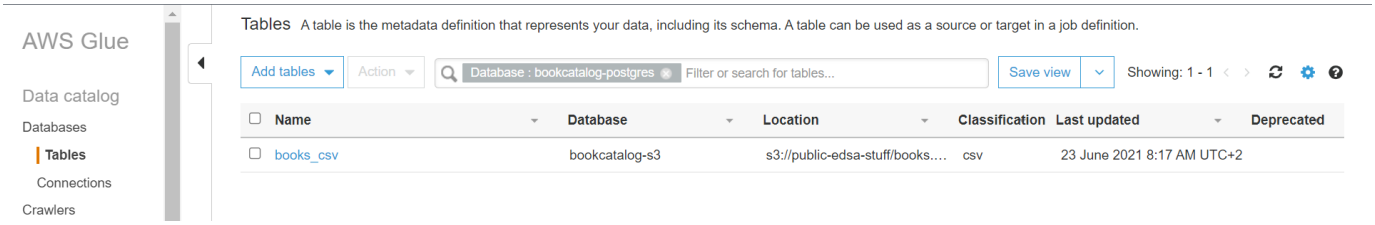

Figure 19: Tables dashboard.

Step 6: Create a connection to the SQL DB

Back to exercise overview

For AWS Glue to connect to your database, we need to create a connection via the Glue management console. This connection contains all the properties of the PostgreSQL database required to be successful.

- In the navigation pane, select**Connections** and proceed to **Add connection**

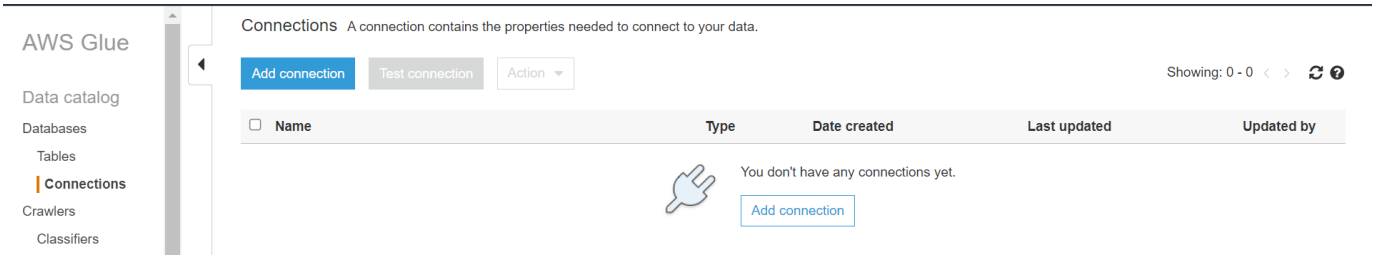

Figure 20: Connection dashboard.

- Provide a connection name.
- Select *'JDBC'* for "connection type".
- Click **Next** to go to the following page.

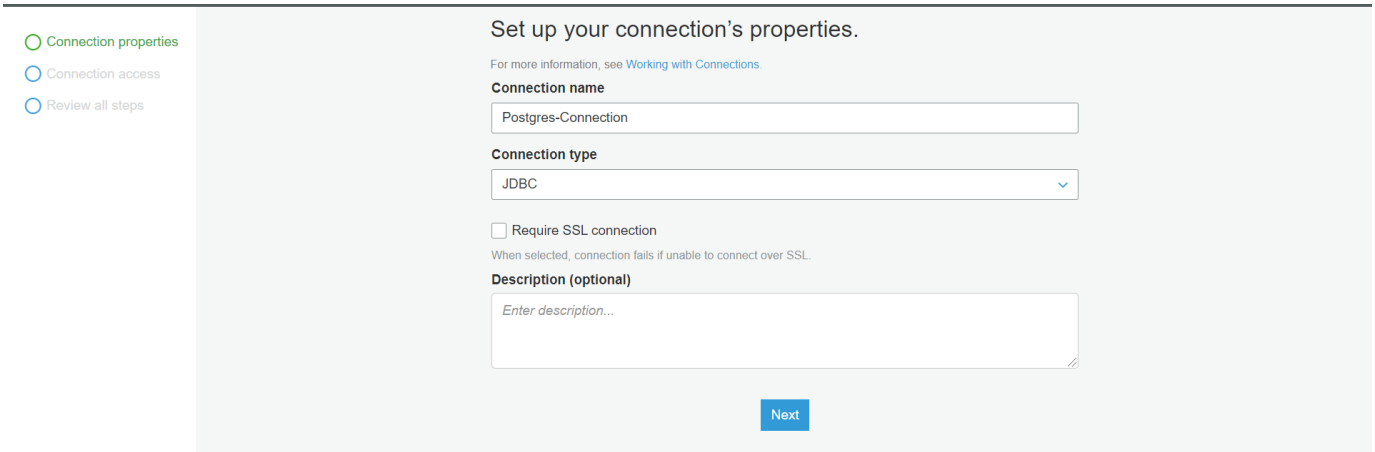

Figure 21: Connection properties.

- Within the *"JDBC URL"* text box paste the following string.

jdbc:postgresql://<database endpoint>:<port>/postgres

- Update the URL string by enter the database endpoint (obtained when setting up the PostgreSQL RDS instance inStep 2), username, and password for your database.
- Select default VPC and Security Group settings used when creating the database
  - **NB**: The RDS DB has to have an "All TCP" connection inbound rule connected to your default security group.
- Click **Next** to review all steps.

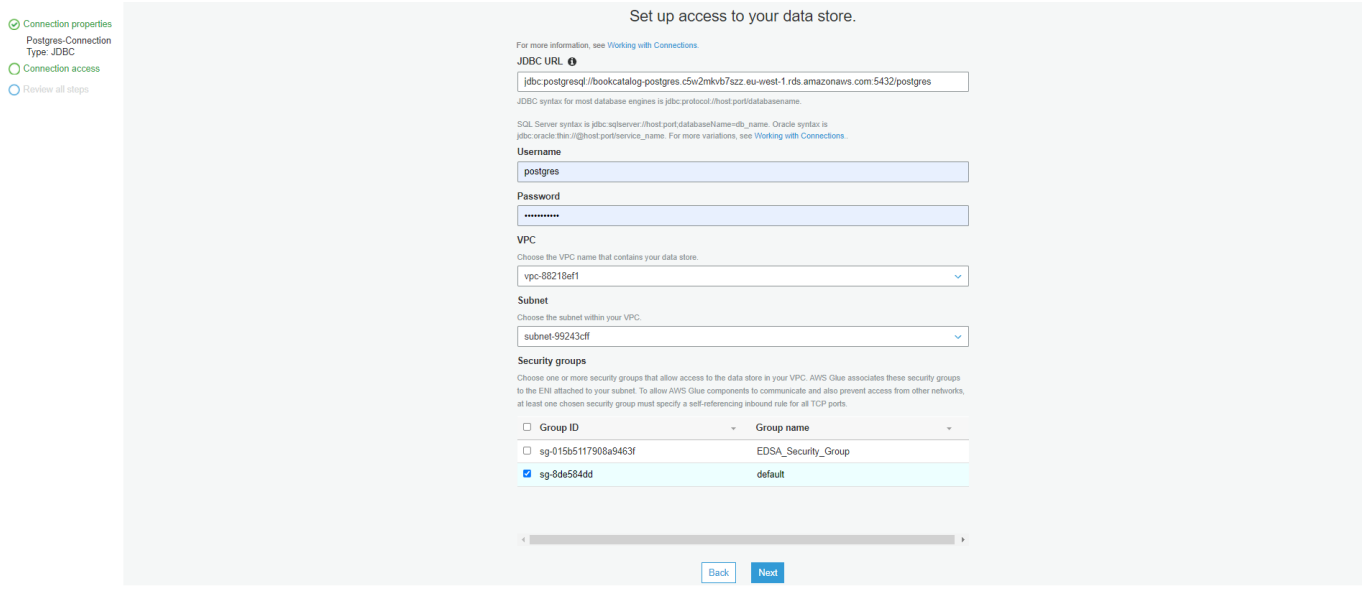

Figure 22: Connection access.

- Verify that all the connection details are correct.
- Click **Finish** to complete the setup.

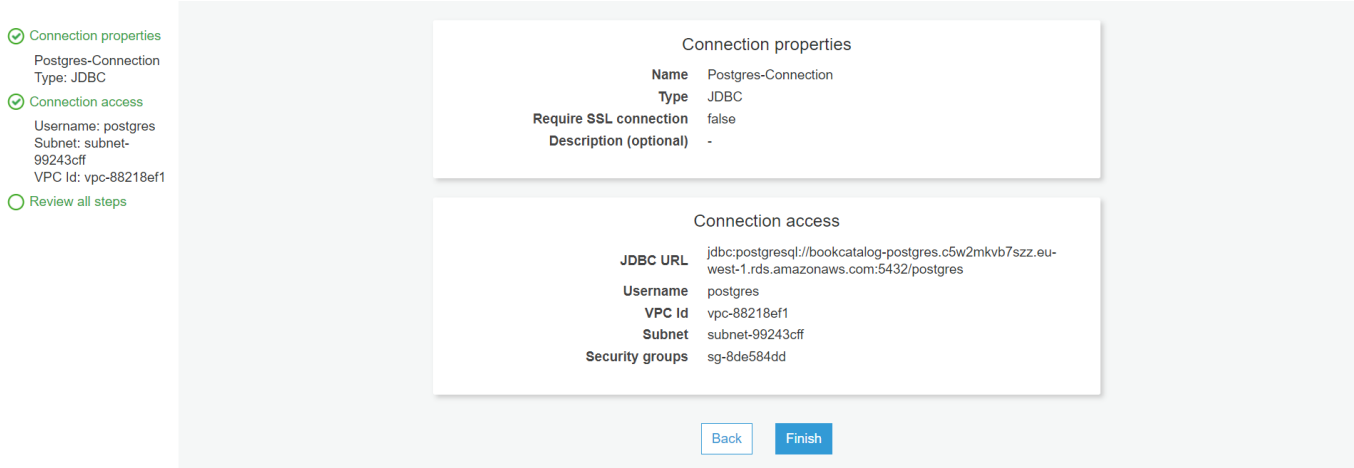

Figure 23: Review connection steps.

- Select the connection you've just created and click on**Test Connection**. This should bring up a pop-up window prompting you to select an IAM Role.
- Ensure you select the IAM Role created in thestep 5 (crawler configuration) and click **Test**.

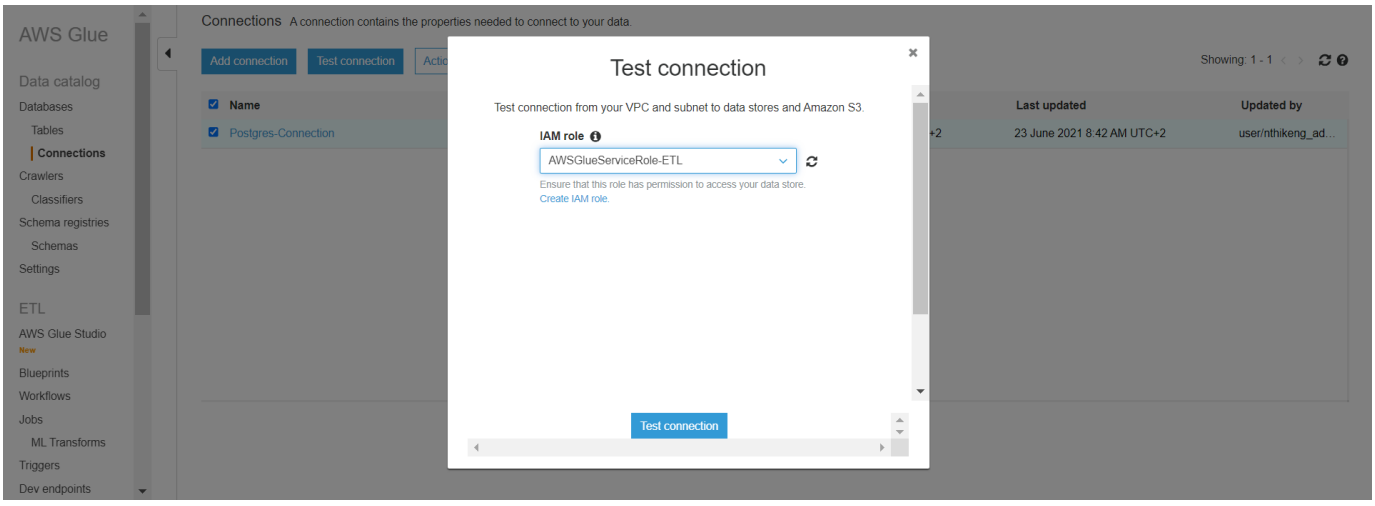

Figure 24: Test Connection.

Step 7: Create a SQL DB crawler and create a data store for the RDS DB

Back to exercise overview

In the same way that we configured a crawler for the S3 bucket containing the CSV data, we also need to create a crawler for our postgresql database. These actions are very similar to those of step 5, however, we don't need to have a classifier for the SQL database.

The following actions should be performed from the *"Crawlers"* view in the AWS Glue dashboard:

- Enter a crawler name, e.g. **DB_crawler**.
- Do not add a classifier to your crawler.
- Leave data store step as defaults.
- Select *"JDBC"* when adding a data store.
- Select the JDBC connection created in Step 6.
- Under **Include path**, give the name of your RDS database:**BookCatalog**.
- Select the same IAM role as above.
- Configure the crawler schedule as "run on demand".
- Add a new database name, e.g. **bookcatalog-rds**.
- Run the crawler. This should create a new table called **bookcatalog_public_book_data**.

Step 8: Create a job

Back to exercise overview

A job is where most of the work will go related to the actual ETL process. In this example, we keep things simple as we solely focus on loading data into our database. In reality, however, this would be the area to write scripts that perform manipulations on our data set.

AWS Glue provides almost unlimited options for what we can do with our data in the way of transforms, and since the service's generated code is fully editable, we can completely customise a script to meet our particular needs. We can further use this point to add triggers and schedules to your ETL pipeline.

- In the navigation pane, select **Jobs** and click on **Add jobs**.

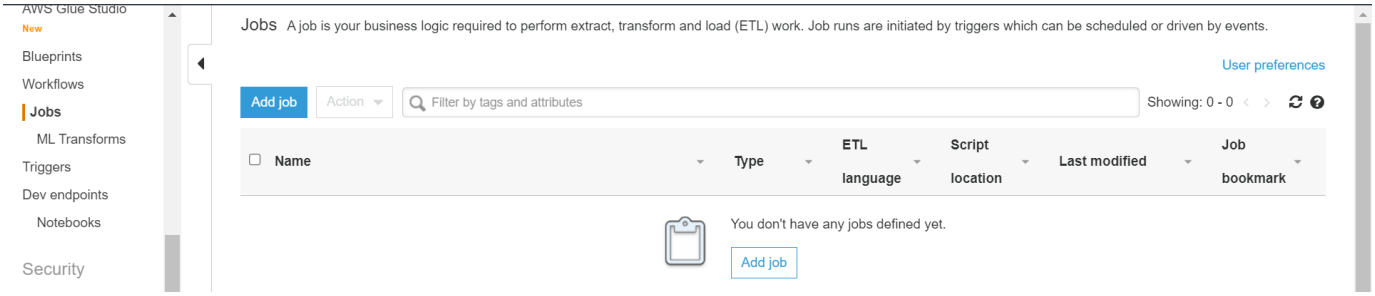

Figure 25: Job dashboard.

- Give your job a name.
- Select the same IAM Role you've been using for this exercise.
- Under **Type** select **Spark**.
- Select a location for your script path (S3 bucket).
- For the **Glue version**, select **Spark 2.4, Python 3 with improved job startup times (Glue Version 2.0)**
- Select the path where you would like the job script to reside.
  - Use the same location you used to store the CSV file.
- Select a temporary directory (Same S3 bucket as above).
- Click **Next** to proceed.

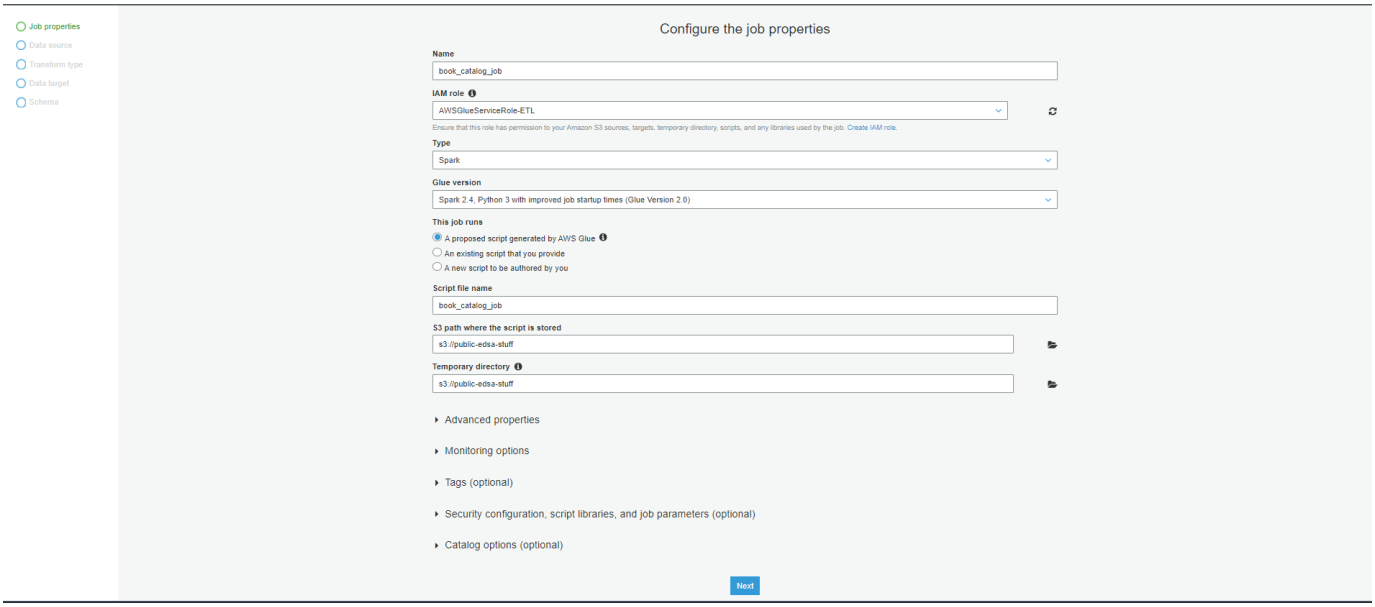

Figure 26: Job properties configuration.

- Select the table that was created by running your S3 crawler.
- Click **Next** to proceed.

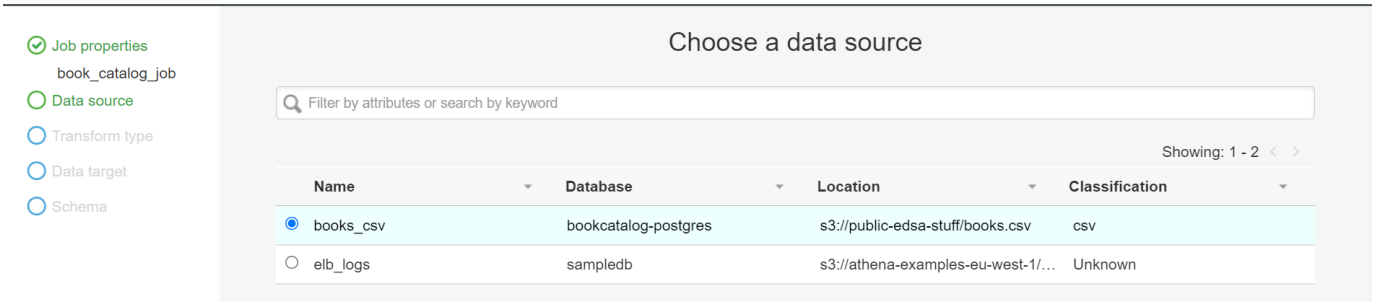

Figure 27: Data source configuration.

- For the **Transform Type** make sure **Change Schema** is selected.
- Click **Next** to proceed.

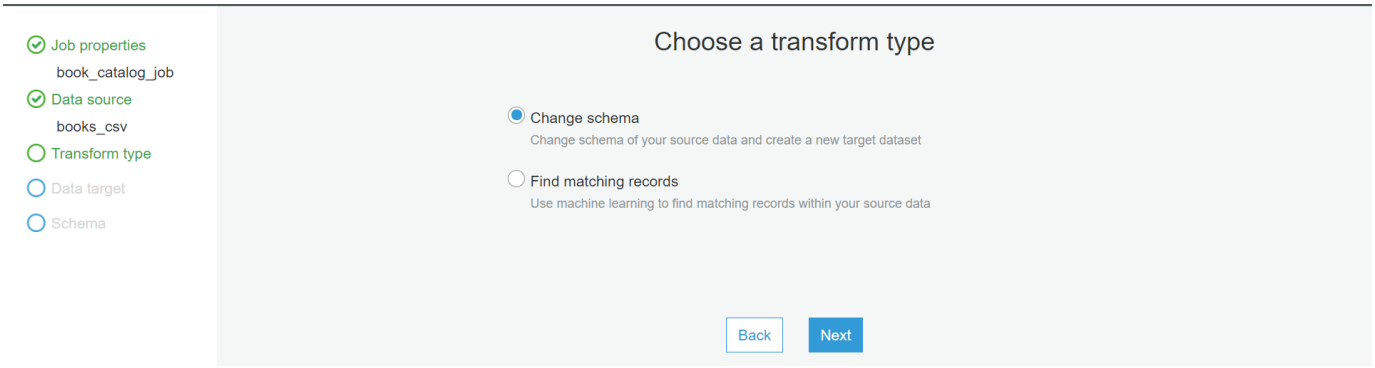

Figure 28: Transform type configuration.

- On the **Data Target** page, select **Use tables in the data catalog and update your data target**
- Select the **bookcatalog_public_book_data** table in the given options.
- Click **Next** to proceed.

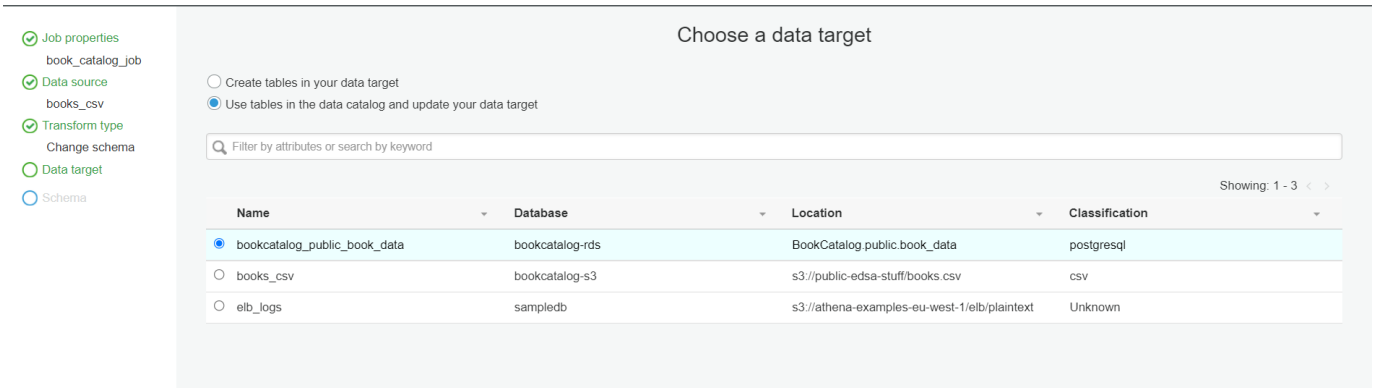

Figure 29: Data target configuration.

- You should now be able to see the**Output Schema Definition**, you can either choose to add or remove columns during the definition. For now, we will leave it as is.
- Click **Save job and edit script**

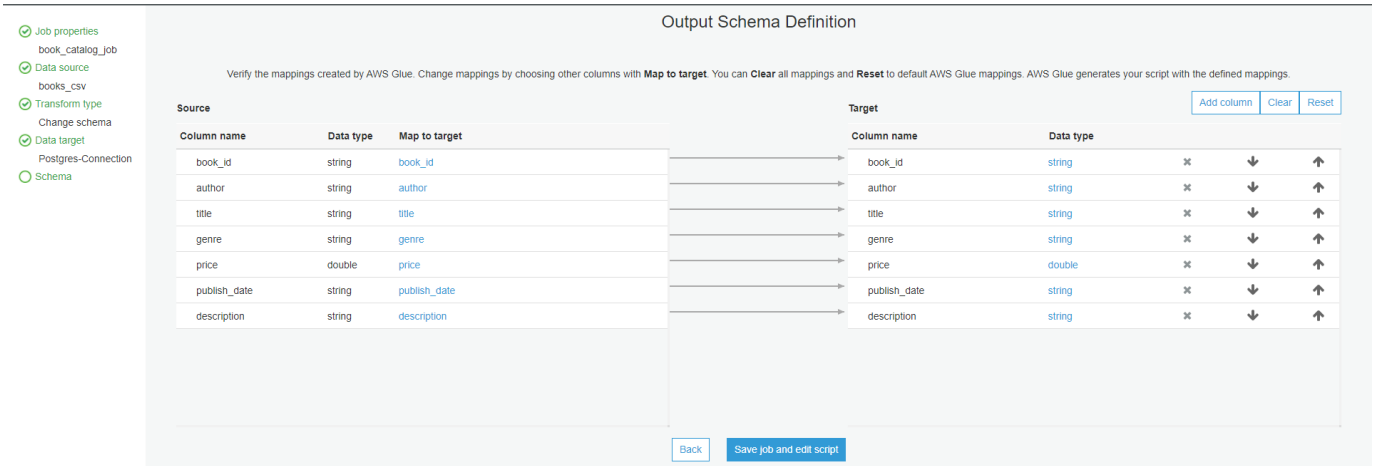

Figure 30: Output schema definition.

Following the above actions, you should be taken to a page that displays the job script that you have created. Notice that the script is written in Python and uses PySpark to load and transform the data.

On the left-hand side of the page, you should see a diagram that illustrates the flow of your job. It shows the data source, the transformations applied to your data, and finally, where the data is stored (data target).

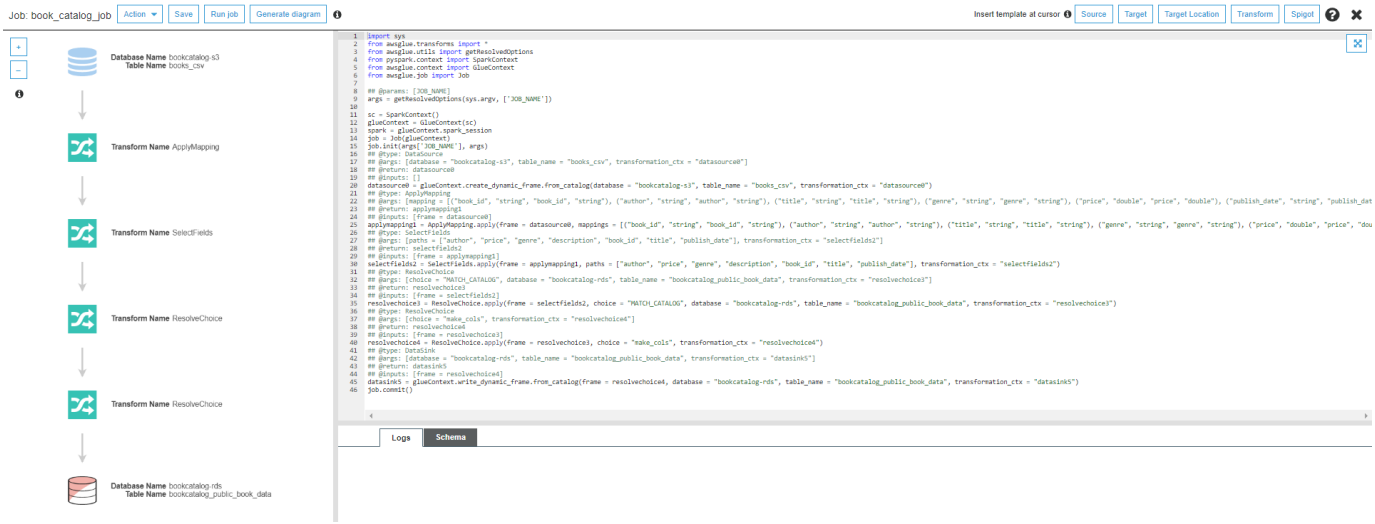

Figure 31: Job script.

- Click **Run job** to execute, a pop-up window will appear prompting you to configure parameters, leave defaults and click **Run job**.

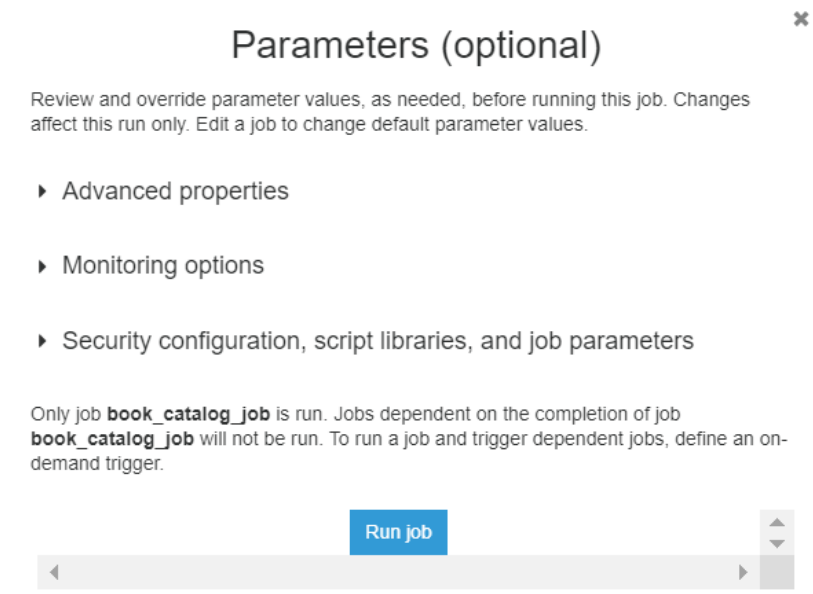

Figure 32: Job Parameters.

- Check that the data is in your PostgreSQL RDS database by running a SELECT query in your SQL client.

Conclusion

Well done! You have successfully created an ETL pipeline within AWS Glue! Go ahead and explore some of the other features if you are interested! In conclusion, we have covered the following topics:

- What AWS Glue is and how it functions;
- Why we would benefit from using AWS Glue;
- Features that make AWS Glue useful; and
- Building an ETL pipeline using AWS Glue.

Appendix

Additional resources

- AWS Glue Overview
- AWS Glue Features
- Why use AWS Glue?

IAM Roles & AWS Service Connectivity

For AWS Glue to run with various other AWS services, one needs to make sure the IAM roles, security groups and VPC configurations are set up correctly.

Security Group & VPC

- Add an "All TCP" Type inbound rule to the default security group.
- Create a VPC endpoint in the default VPC. Follow this link for more information on how to set up a VPC endpoint.

IAM Role Additions

- Make sure that the following AWS managed policies are attached to your Role

    1. AmazonS3FullAccess
    2. AWSGlueServiceRole
    3. AmazonRDSDataFullAccess

- The role created via the Glue interface should look something like the JSON file below:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::<s3 bucket url>/books.csv*"
            ]
        }
    ]
}
```