PySpark On AWS Glue

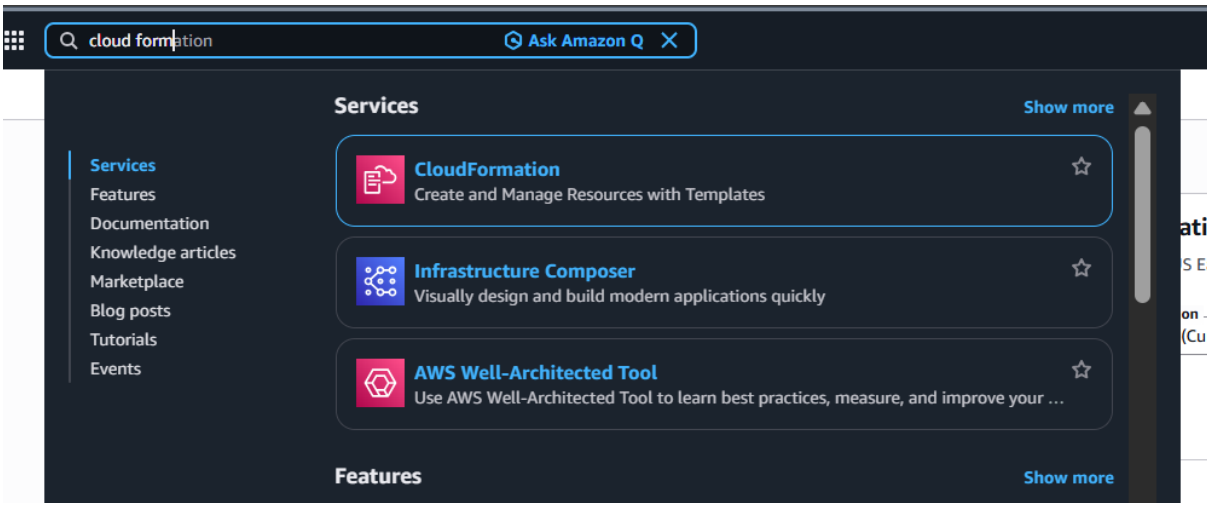
Creating Resources using supplied cloud formation template in aws

A **CloudFormation template** is a file (YAML or JSON) that describes:

- **What AWS resources to create**
- **How they are configured**
- **How they relate to each other**

When you choose **Create resources using supplied CloudFormation template**, AWS uses that template to create all required resources **in one operation**.

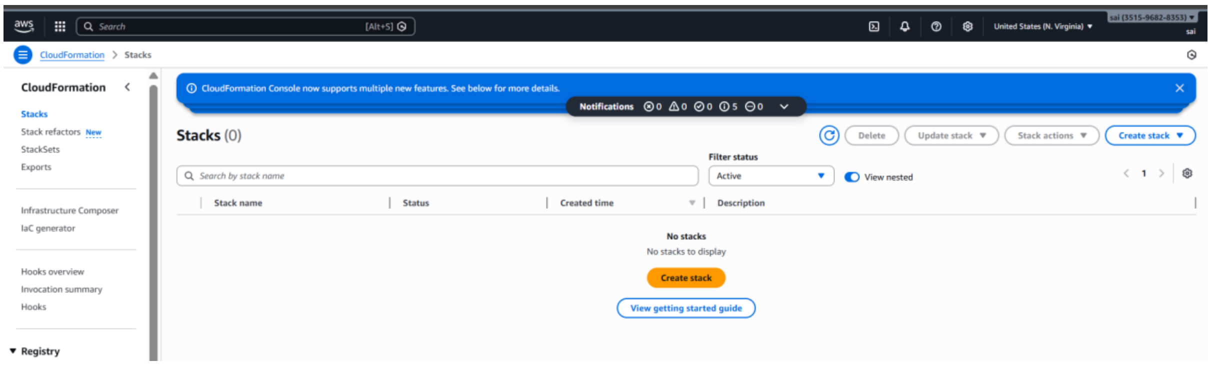
Go to AWS console and type cloud formation

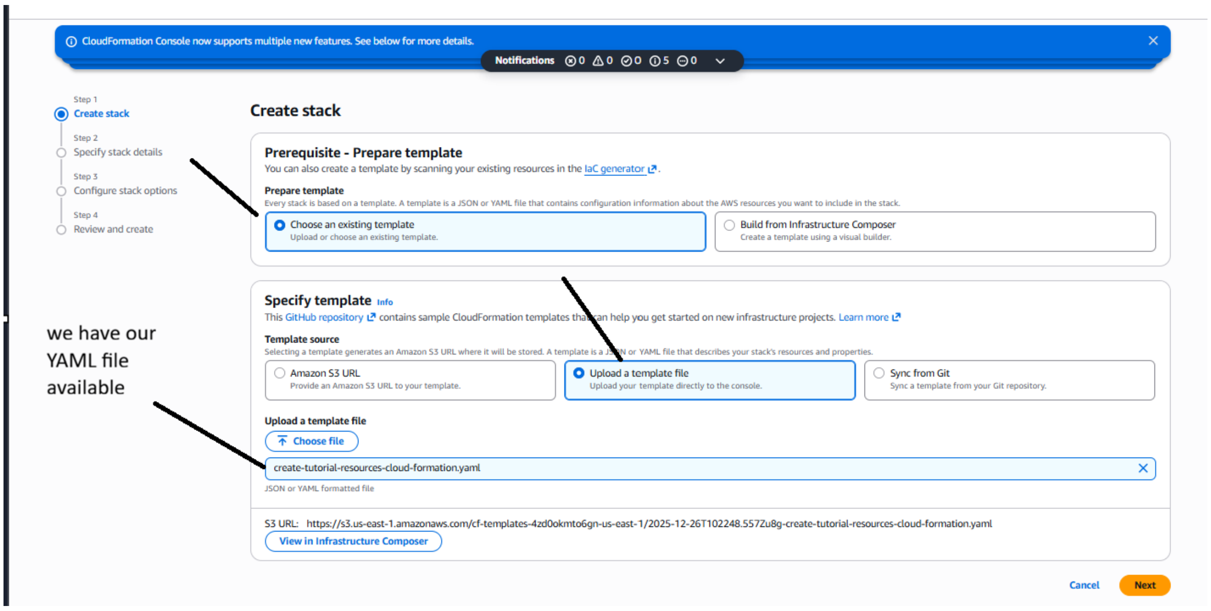

cloud formation
Ask Amazon Q
Services
Features
Documentation
Knowledge articles
Marketplace
Blog posts
Tutorials
Events
Services
Show more
CloudFormation
Create and Manage Resources with Templates
Infrastructure Composer
Visually design and build modern applications quickly
AWS Well-Architected Tool
Use AWS Well-Architected Tool to learn best practices, measure, and improve your ...
Features
Show more

U must be a user that has permissions to run cloud formation,quick glue,resources create ion poilicies and create s3 buckets etc

I am using my IAM user account where I have my admin access

Now the cloudformation interface looks like the below, click on create stack





we have our YAML file available

CloudFormation Console now supports multiple new features. See below for more details.

Notifications ⊗ 0 ⚠ 0 ⊘ 0 ⓘ 5 ⊖ 0

Step 1
Create stack
Step 2
Specify stack details
Step 3
Configure stack options
Step 4
Review and create

Create stack

Prerequisite - Prepare template
You can also create a template by scanning your existing resources in the IaC generator.

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Choose an existing template — Upload or choose an existing template.]
Build from Infrastructure Composer — Create a template using a visual builder.

Specify template Info
This GitHub repository contains sample CloudFormation templates that can help you get started on new infrastructure projects. Learn more

Template source
Selecting a template generates an Amazon S3 URL where it will be stored. A template is a JSON or YAML file that describes your stack's resources and properties.

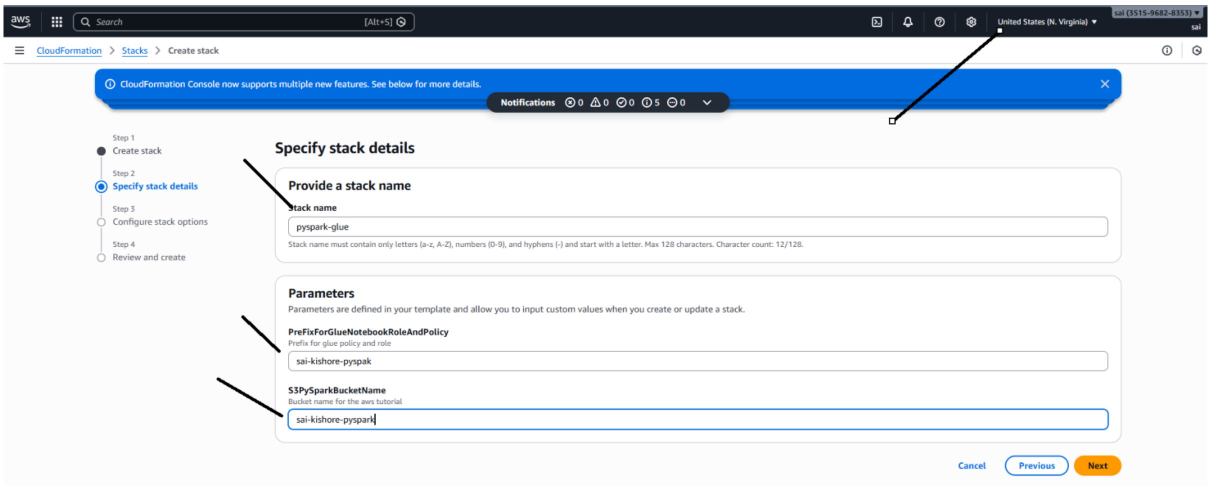
Amazon S3 URL — Provide an Amazon S3 URL to your template.
[✓ Upload a template file — Upload your template directly to the console.]
Sync from Git — Sync a template from your Git repository.

Upload a template file
Choose file
create-tutorial-resources-cloud-formation.yaml
JSON or YAML formatted file

S3 URL: https://s3.us-east-1.amazonaws.com/cf-templates-4zd0okmto6gn-us-east-1/2025-12-26T102248.557Zu8g-create-tutorial-resources-cloud-formation.yaml
View in Infrastructure Composer

Cancel Next

Click on next



CloudFormation > Stacks > Create stack

CloudFormation Console now supports multiple new features. See below for more details.

Notifications ⊗ 0 ⚠ 0 ⊘ 0 ⓘ 5 ⊖ 0

Step 1
Create stack
Step 2
Specify stack details
Step 3
Configure stack options
Step 4
Review and create

Specify stack details

Provide a stack name

Stack name
pyspark-glue
Stack name must contain only letters (a-z, A-Z), numbers (0-9), and hyphens (-) and start with a letter. Max 128 characters. Character count: 12/128.

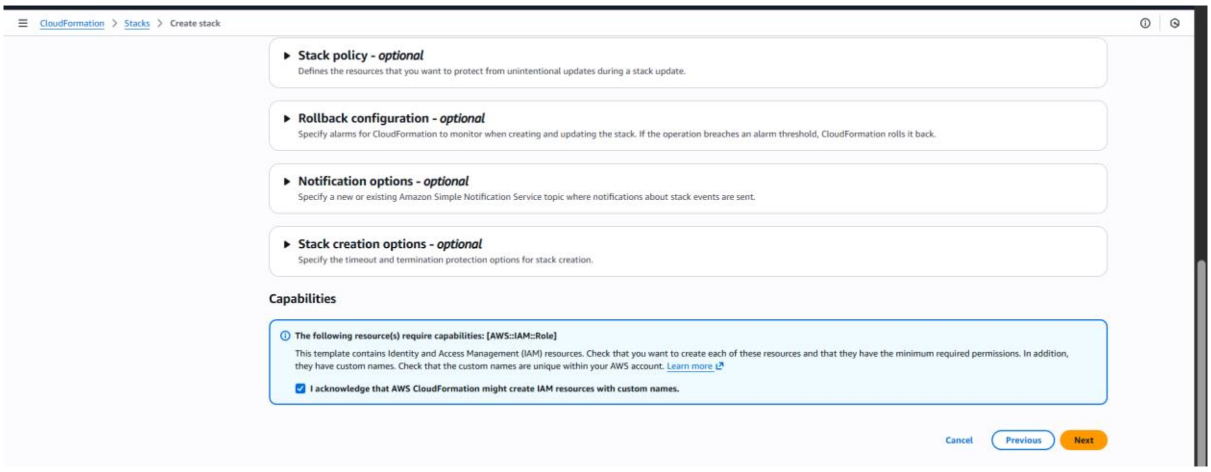
Parameters
Parameters are defined in your template and allow you to input custom values when you create or update a stack.

PreFixForGlueNotebookRoleAndPolicy
Prefix for glue policy and role
sai-kishore-pyspak

S3PySparkBucketName
Bucket name for the aws tutorial
sai-kishore-pyspark

Cancel Previous Next

Click next



CloudFormation > Stacks > Create stack

▸ **Stack policy** - ***optional***
Defines the resources that you want to protect from unintentional updates during a stack update.

▸ **Rollback configuration** - ***optional***
Specify alarms for CloudFormation to monitor when creating and updating the stack. If the operation breaches an alarm threshold, CloudFormation rolls it back.

▸ **Notification options** - ***optional***
Specify a new or existing Amazon Simple Notification Service topic where notifications about stack events are sent.

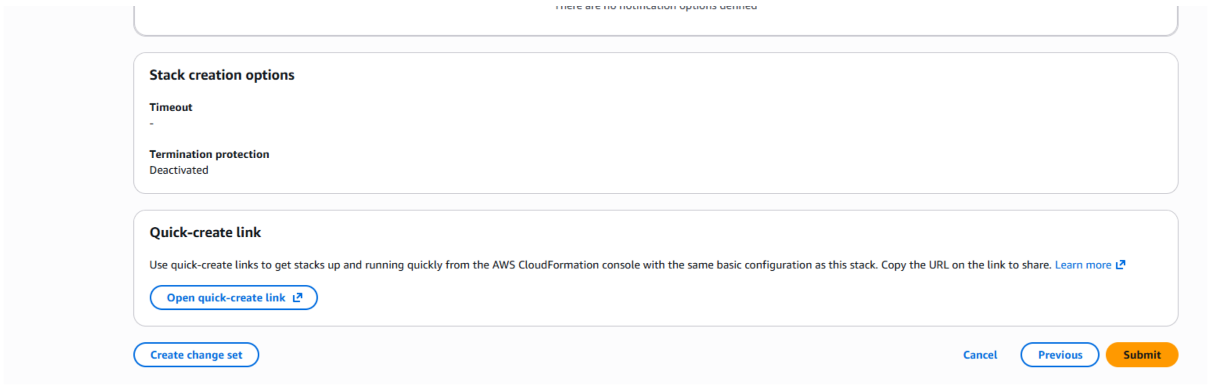
▸ **Stack creation options** - ***optional***
Specify the timeout and termination protection options for stack creation.

Capabilities

The following resource(s) require capabilities: [AWS::IAM::Role]
This template contains Identity and Access Management (IAM) resources. Check that you want to create each of these resources and that they have the minimum required permissions. In addition, they have custom names. Check that the custom names are unique within your AWS account. Learn more

☑ **I acknowledge that AWS CloudFormation might create IAM resources with custom names.**

Cancel Previous Next



Stack creation options

Timeout
-

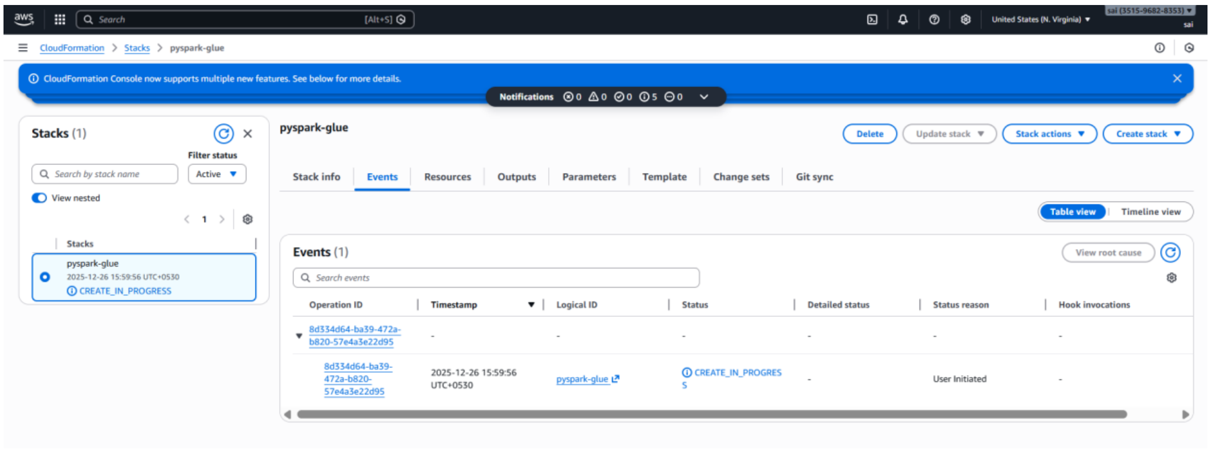
Termination protection
Deactivated

Quick-create link

Use quick-create links to get stacks up and running quickly from the AWS CloudFormation console with the same basic configuration as this stack. Copy the URL on the link to share. Learn more

Click submit

The interface looks as follows



CloudFormation > Stacks > pyspark-glue

CloudFormation Console now supports multiple new features. See below for more details.

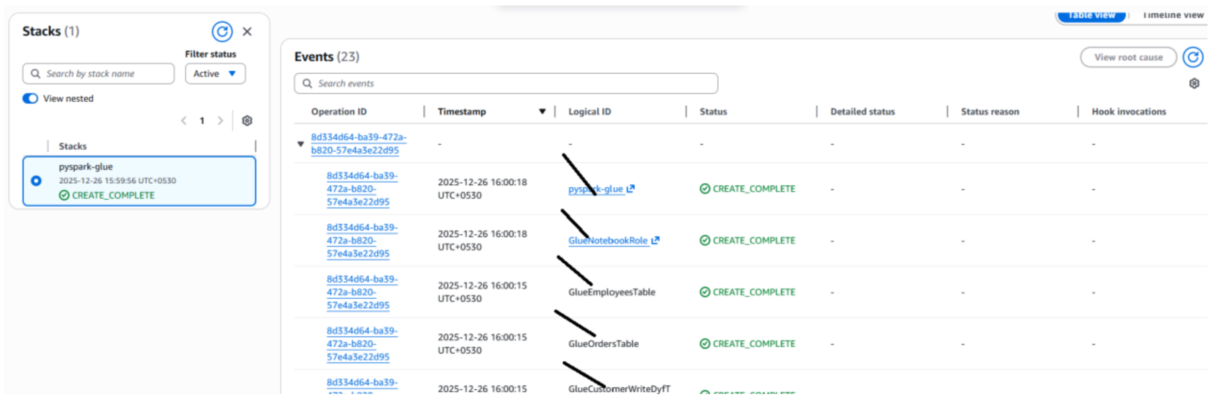
Stacks

pyspark-glue
2025-12-26 15:59:56 UTC+0530
CREATE_IN_PROGRESS

pyspark-glue

Events (1)

Operation ID	Timestamp	Logical ID	Status	Detailed status	Status reason	Hook invocations
8d334d64-ba39-472a-b820-57e4a3e22d95	-	-	-	-	-	-
8d334d64-ba39-472a-b820-57e4a3e22d95	2025-12-26 15:59:56 UTC+0530	pyspark-glue	CREATE_IN_PROGRESS	-	User Initiated	-

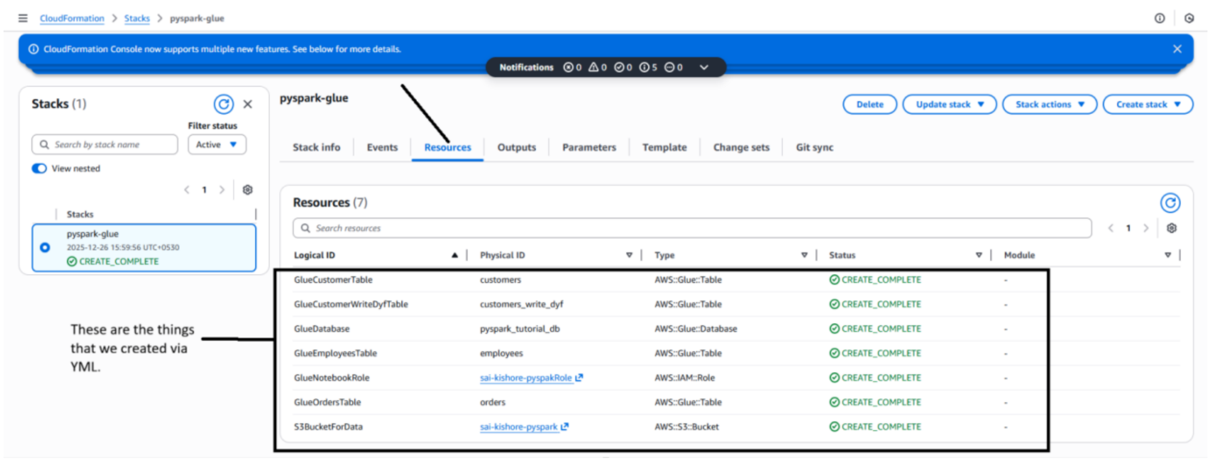


Stacks

pyspark-glue
2025-12-26 15:59:56 UTC+0530
CREATE_COMPLETE

Events (23)

Operation ID	Timestamp	Logical ID	Status	Detailed status	Status reason	Hook invocations
8d334d64-ba39-472a-b820-57e4a3e22d95	-	-	-	-	-	-
8d334d64-ba39-472a-b820-57e4a3e22d95	2025-12-26 16:00:18 UTC+0530	pyspark-glue	CREATE_COMPLETE	-	-	-
8d334d64-ba39-472a-b820-57e4a3e22d95	2025-12-26 16:00:18 UTC+0530	GlueNotebookRole	CREATE_COMPLETE	-	-	-
8d334d64-ba39-472a-b820-57e4a3e22d95	2025-12-26 16:00:15 UTC+0530	GlueEmployeesTable	CREATE_COMPLETE	-	-	-
8d334d64-ba39-472a-b820-57e4a3e22d95	2025-12-26 16:00:15 UTC+0530	GlueOrdersTable	CREATE_COMPLETE	-	-	-
8d334d64-ba39-	2025-12-26 16:00:15	GlueCustomerWriteDyfT				



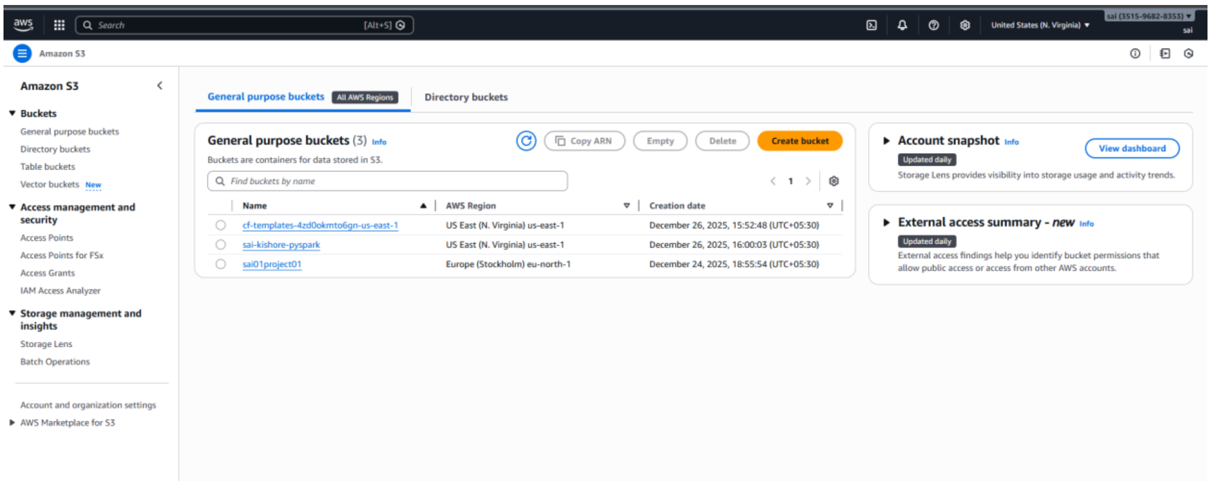
pyspark-glue
2025-12-26 15:59:56 UTC+0530
CREATE_COMPLETE

Resources (7)

Logical ID	Physical ID	Type	Status	Module
GlueCustomerTable	customers	AWS::Glue::Table	CREATE_COMPLETE	-
GlueCustomerWriteDyfTable	customers_write_dyf	AWS::Glue::Table	CREATE_COMPLETE	-
GlueDatabase	pyspark_tutorial_db	AWS::Glue::Database	CREATE_COMPLETE	-
GlueEmployeesTable	employees	AWS::Glue::Table	CREATE_COMPLETE	-
GlueNotebookRole	sai-kishore-pyspakRole	AWS::IAM::Role	CREATE_COMPLETE	-
GlueOrdersTable	orders	AWS::Glue::Table	CREATE_COMPLETE	-
S3BucketForData	sai-kishore-pyspark	AWS::S3::Bucket	CREATE_COMPLETE	-

These are the things that we created via YML.

Now when u go to s3



General purpose buckets (3) Info

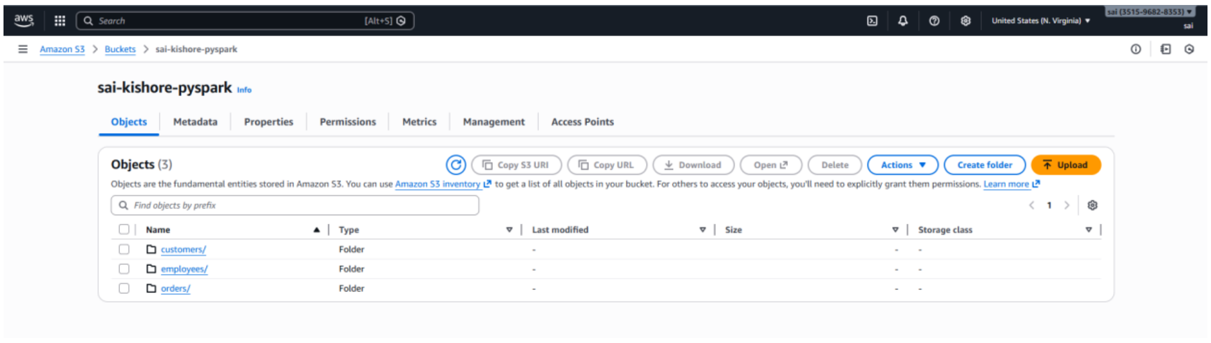
Buckets are containers for data stored in S3.

Name	AWS Region	Creation date
cf-templates-4zd0okmto6gn-us-east-1	US East (N. Virginia) us-east-1	December 26, 2025, 15:52:48 (UTC+05:30)
sai-kishore-pyspark	US East (N. Virginia) us-east-1	December 26, 2025, 16:00:03 (UTC+05:30)
sai01project01	Europe (Stockholm) eu-north-1	December 24, 2025, 18:55:54 (UTC+05:30)

Account snapshot Info
Updated daily
Storage Lens provides visibility into storage usage and activity trends.

External access summary - *new* Info
Updated daily
External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

Uploaded folders with each file in it in csv format in sai-kishore-pyspark



sai-kishore-pyspark Info

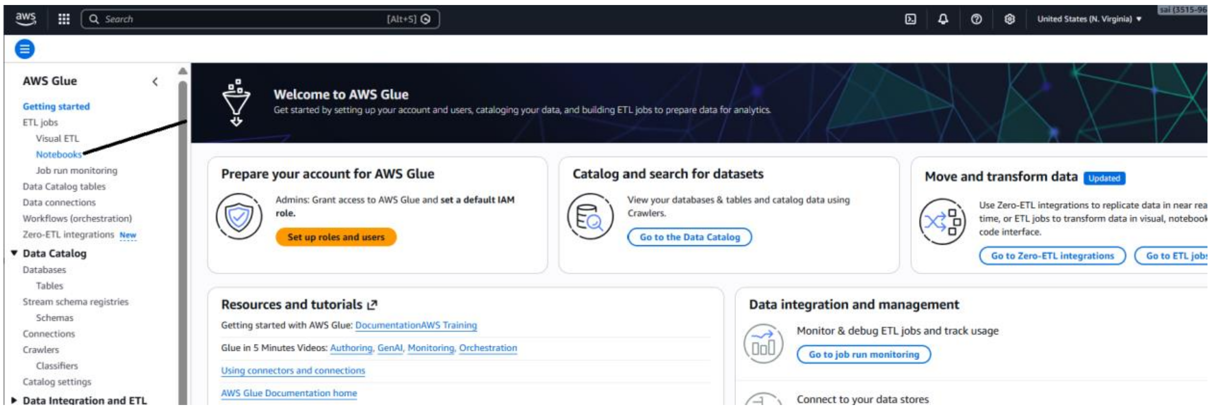
Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
customers/	Folder	-	-	-
employees/	Folder	-	-	-
orders/	Folder	-	-	-

Now lets create a notebook server that we can use the glue interactive session.

Lets go to glue



Welcome to AWS Glue

Get started by setting up your account and users, cataloging your data, and building ETL jobs to prepare data for analytics.

Prepare your account for AWS Glue

Admins: Grant access to AWS Glue and **set a default IAM role.**

Catalog and search for datasets

View your databases & tables and catalog data using Crawlers.

Move and transform data

Use Zero-ETL integrations to replicate data in near rea time, or ETL jobs to transform data in visual, notebook code interface.

Resources and tutorials

Getting started with AWS Glue: Documentation AWS Training

Glue in 5 Minutes Videos: Authoring, GenAI, Monitoring, Orchestration

Using connectors and connections

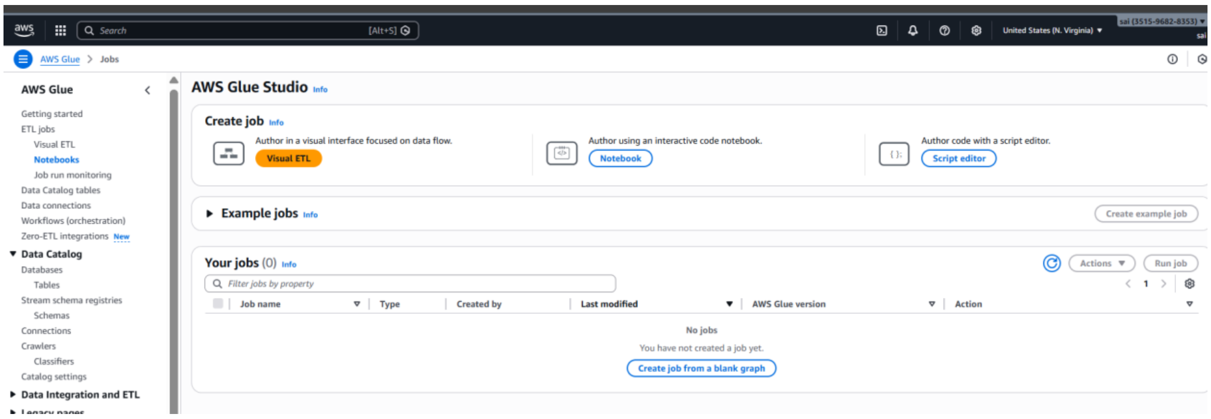
AWS Glue Documentation home

Data integration and management

Monitor & debug ETL jobs and track usage

Connect to your data stores

Click on notebook in below img



AWS Glue Studio Info

Create job Info

Author in a visual interface focused on data flow.

Author using an interactive code notebook.

Author code with a script editor.

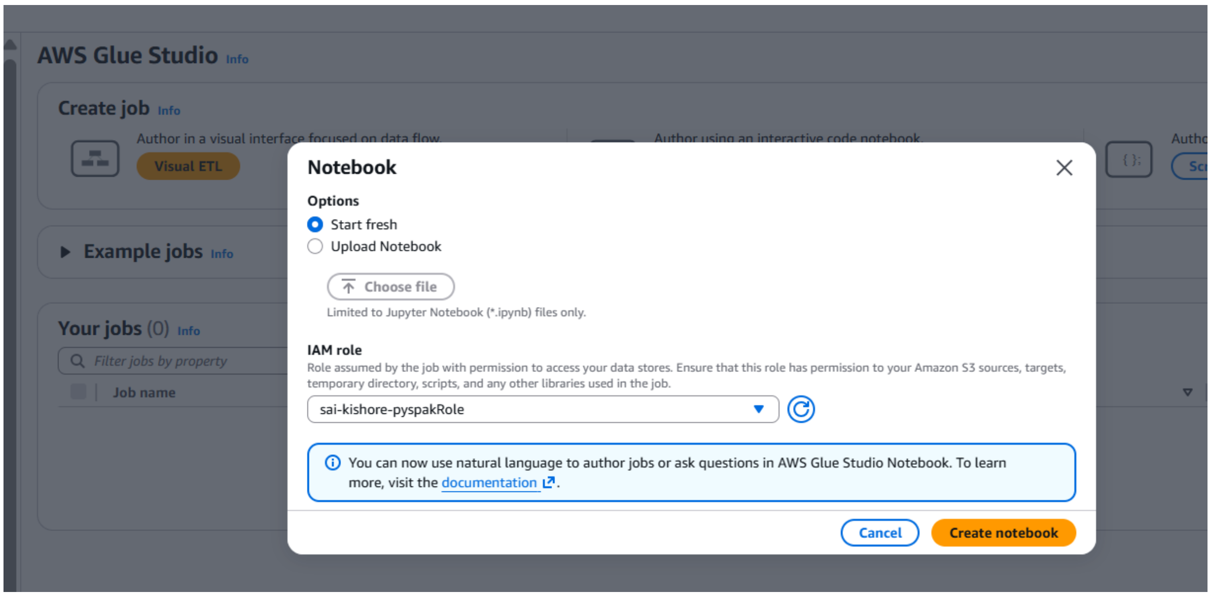
Example jobs Info

Your jobs (0) Info

No jobs

You have not created a job yet.

Click create notebook



Notebook

Options

Start fresh [✓]

Upload Notebook

Limited to Jupyter Notebook (*.ipynb) files only.

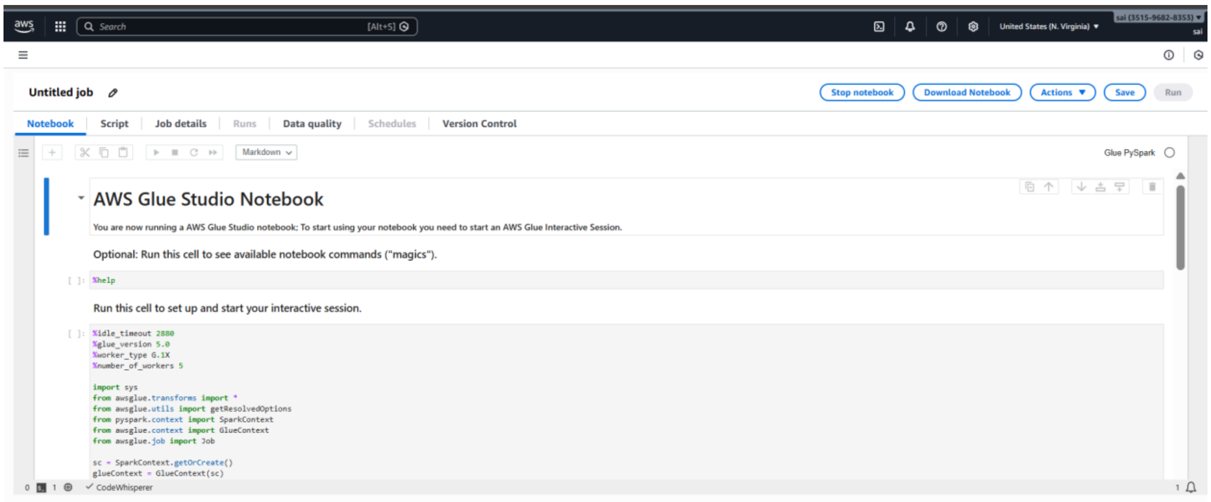
IAM role

Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any other libraries used in the job.

sai-kishore-pyspakRole

You can now use natural language to author jobs or ask questions in AWS Glue Studio Notebook. To learn more, visit the documentation.

U see the interface like this



AWS Glue Studio Notebook

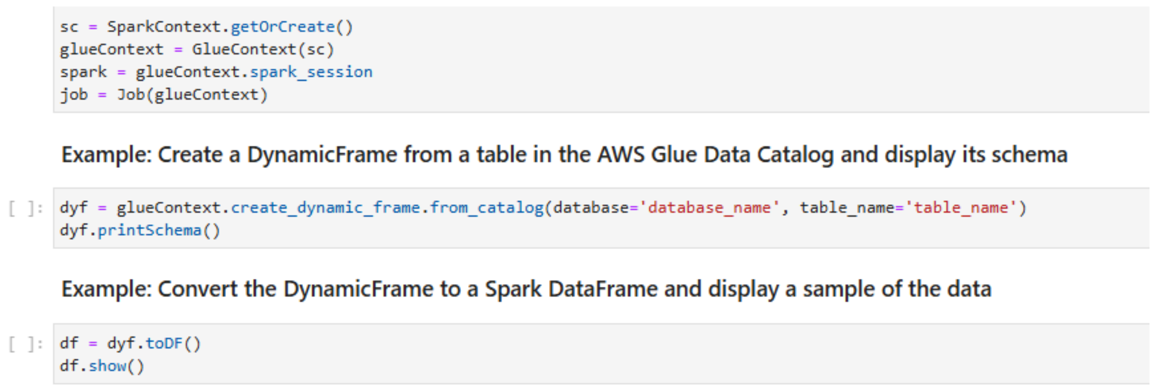
You are now running a AWS Glue Studio notebook; To start using your notebook you need to start an AWS Glue Interactive Session.

Optional: Run this cell to see available notebook commands ("magics").

```
[ ]: %help
```

Run this cell to set up and start your interactive session.

```
[ ]: %idle_timeout 2880
%glue_version 5.0
%worker_type G.1X
%number_of_workers 5

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

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
```



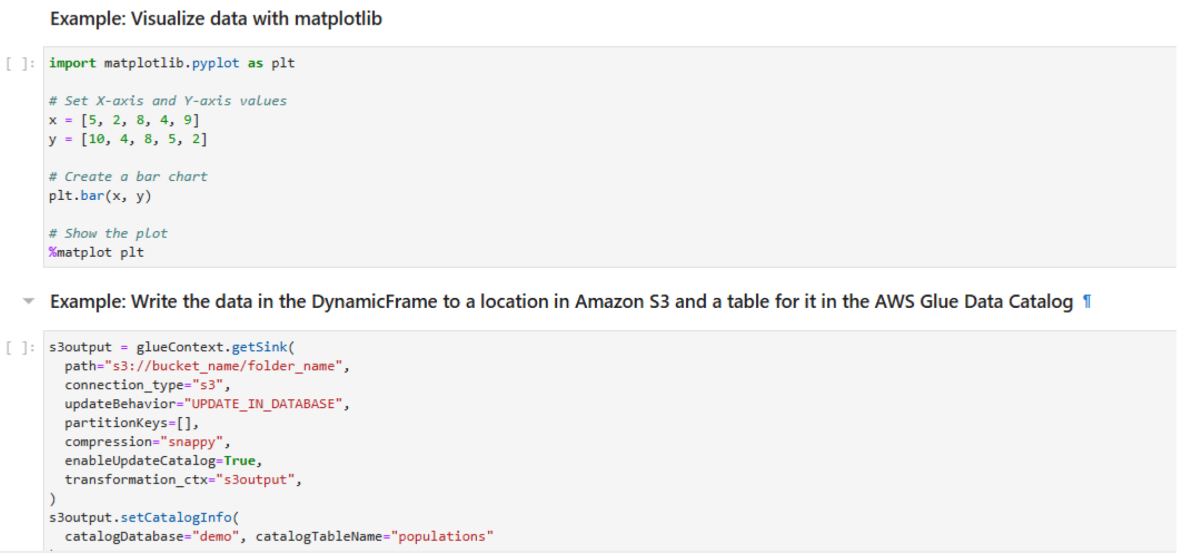
```
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

Example: Create a DynamicFrame from a table in the AWS Glue Data Catalog and display its schema

```
[ ]: dyf = glueContext.create_dynamic_frame.from_catalog(database='database_name', table_name='table_name')
dyf.printSchema()
```

Example: Convert the DynamicFrame to a Spark DataFrame and display a sample of the data

```
[ ]: df = dyf.toDF()
df.show()
```

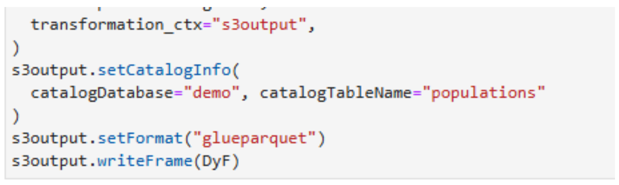


Example: Visualize data with matplotlib

```
[ ]: import matplotlib.pyplot as plt

# Set X-axis and Y-axis values
x = [5, 2, 8, 4, 9]
y = [10, 4, 8, 5, 2]

# Create a bar chart
plt.bar(x, y)

# Show the plot
%matplot plt
```

Example: Write the data in the DynamicFrame to a location in Amazon S3 and a table for it in the AWS Glue Data Catalog ¶

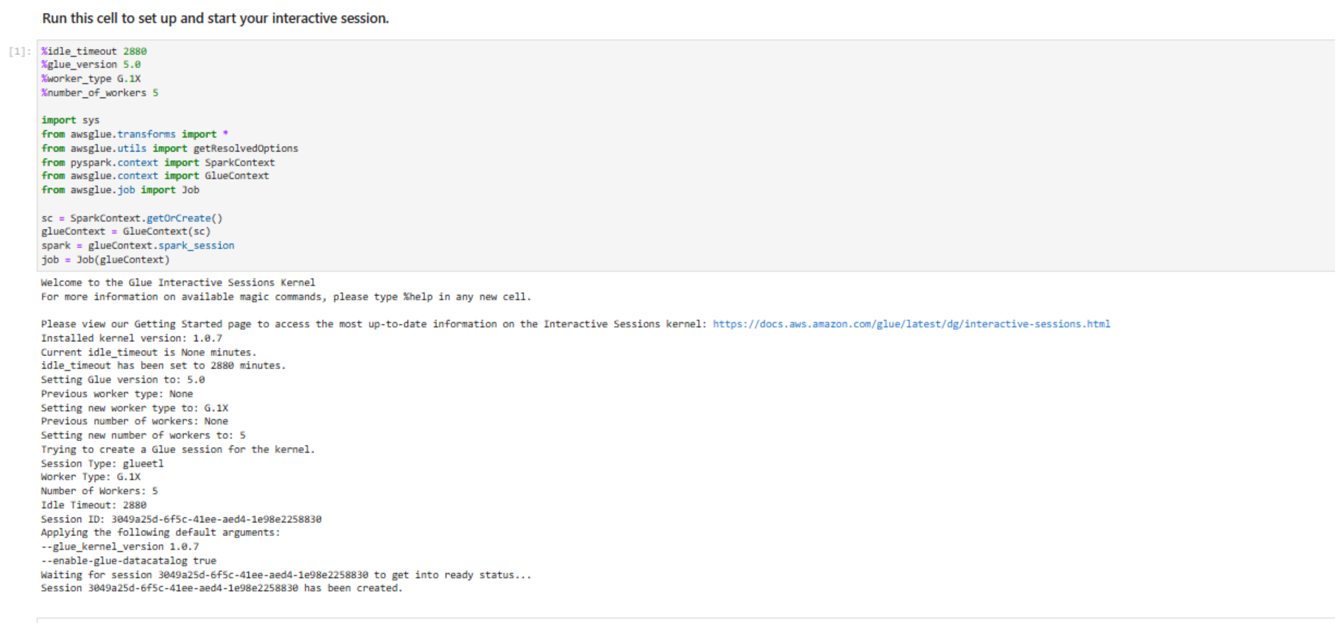
```
[ ]: s3output = glueContext.getSink(
  path="s3://bucket_name/folder_name",
  connection_type="s3",
  updateBehavior="UPDATE_IN_DATABASE",
  partitionKeys=[],
  compression="snappy",
  enableUpdateCatalog=True,
  transformation_ctx="s3output",
)
s3output.setCatalogInfo(
  catalogDatabase="demo", catalogTableName="populations"
```



```
  transformation_ctx="s3output",
)
s3output.setCatalogInfo(
  catalogDatabase="demo", catalogTableName="populations"
)
s3output.setFormat("glueparquet")
s3output.writeFrame(DyF)
```

The above block of code appears by default

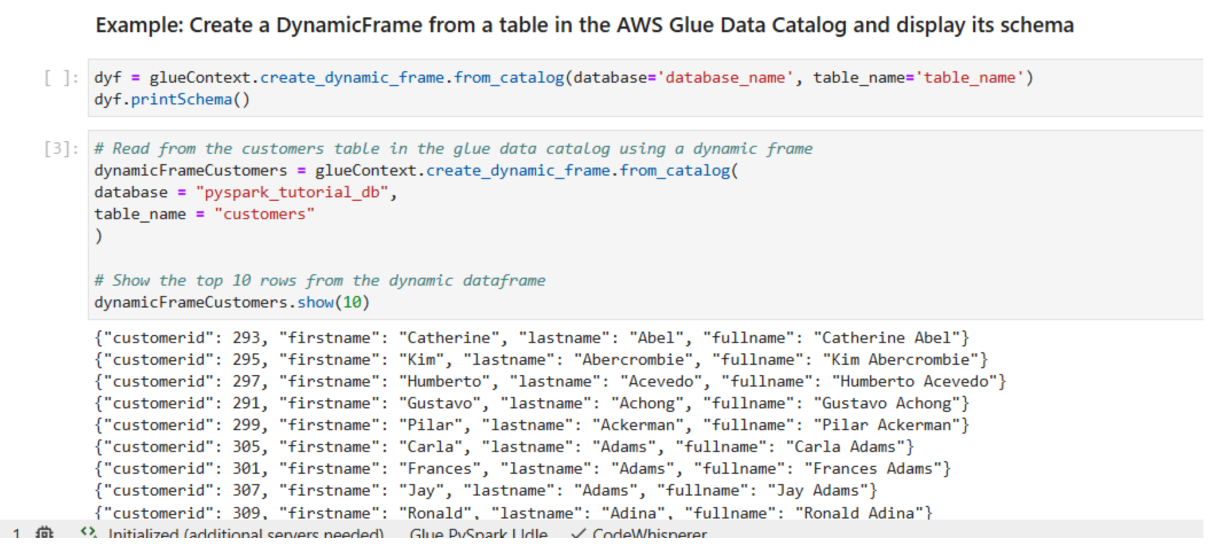
When I run this cell, spark session is created



Run this cell to set up and start your interactive session.

```
[1]: %idle_timeout 2880
     %glue_version 5.0
     %worker_type G.1X
     %number_of_workers 5

     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

     sc = SparkContext.getOrCreate()
     glueContext = GlueContext(sc)
     spark = glueContext.spark_session
     job = Job(glueContext)
```

```
Welcome to the Glue Interactive Sessions Kernel
For more information on available magic commands, please type %help in any new cell.

Please view our Getting Started page to access the most up-to-date information on the Interactive Sessions kernel: https://docs.aws.amazon.com/glue/latest/dg/interactive-sessions.html
Installed kernel version: 1.0.7
Current idle_timeout is None minutes.
idle_timeout has been set to 2880 minutes.
Setting Glue version to: 5.0
Previous worker type: None
Setting new worker type to: G.1X
Previous number of workers: None
Setting new number of workers to: 5
Trying to create a Glue session for the kernel.
Session Type: glueetl
Worker Type: G.1X
Number of Workers: 5
Idle Timeout: 2880
Session ID: 3049a25d-6f5c-41ee-aed4-1e98e2258830
Applying the following default arguments:
--glue_kernel_version 1.0.7
--enable-glue-datacatalog true
Waiting for session 3049a25d-6f5c-41ee-aed4-1e98e2258830 to get into ready status...
Session 3049a25d-6f5c-41ee-aed4-1e98e2258830 has been created.
```



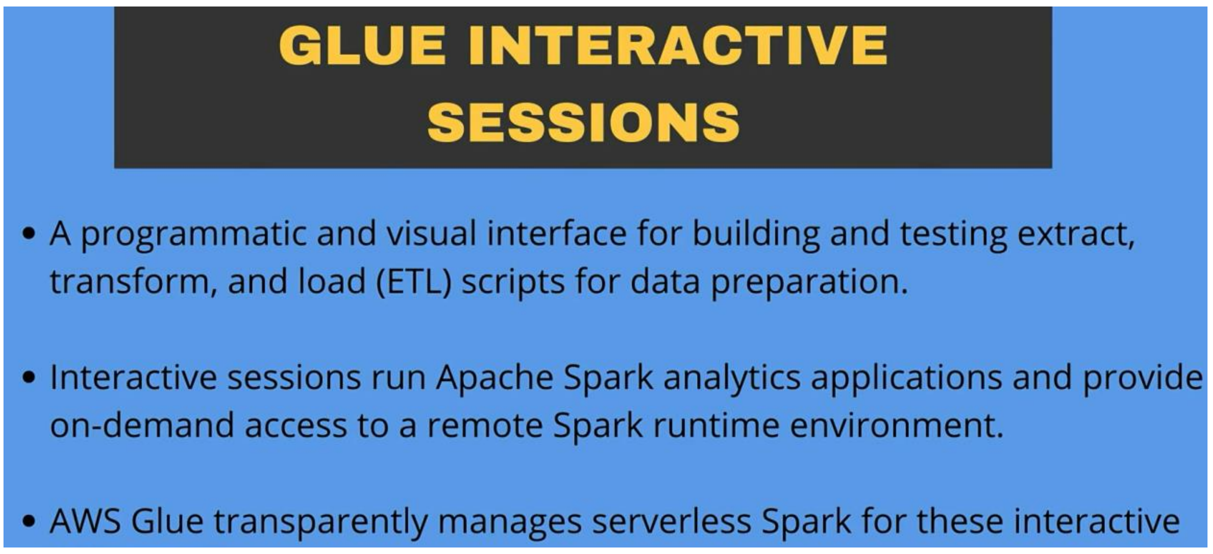
Example: Create a DynamicFrame from a table in the AWS Glue Data Catalog and display its schema

```
[ ]: dyf = glueContext.create_dynamic_frame.from_catalog(database='database_name', table_name='table_name')
     dyf.printSchema()
```

```
[3]: # Read from the customers table in the glue data catalog using a dynamic frame
     dynamicFrameCustomers = glueContext.create_dynamic_frame.from_catalog(
     database = "pyspark_tutorial_db",
     table_name = "customers"
     )

     # Show the top 10 rows from the dynamic dataframe
     dynamicFrameCustomers.show(10)
```

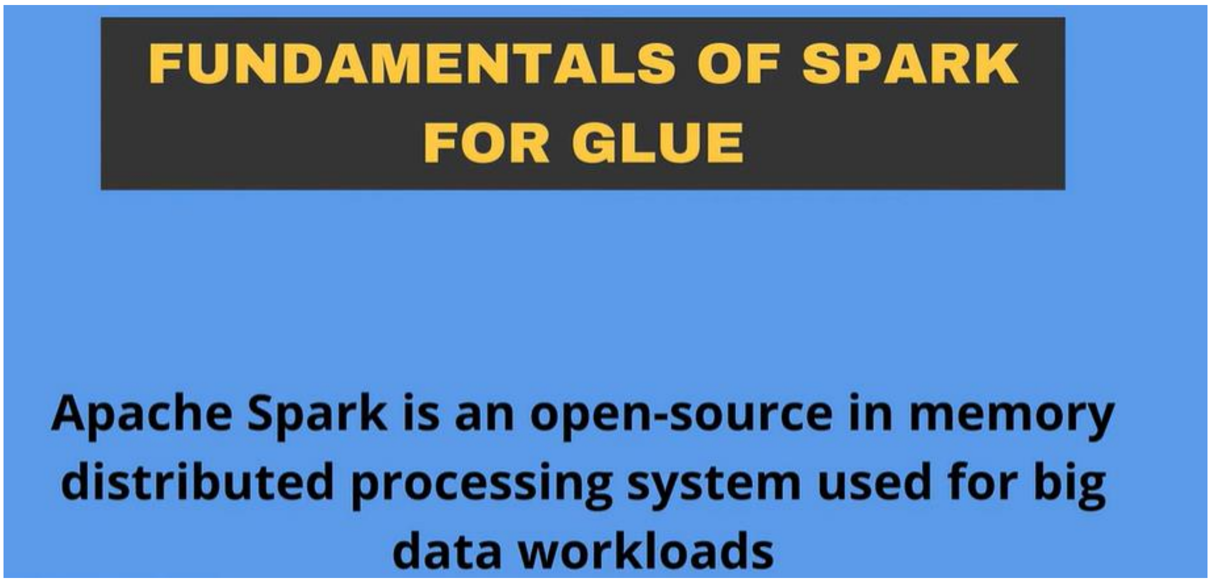
```
{"customerid": 293, "firstname": "Catherine", "lastname": "Abel", "fullname": "Catherine Abel"}
{"customerid": 295, "firstname": "Kim", "lastname": "Abercrombie", "fullname": "Kim Abercrombie"}
{"customerid": 297, "firstname": "Humberto", "lastname": "Acevedo", "fullname": "Humberto Acevedo"}
{"customerid": 291, "firstname": "Gustavo", "lastname": "Achong", "fullname": "Gustavo Achong"}
{"customerid": 299, "firstname": "Pilar", "lastname": "Ackerman", "fullname": "Pilar Ackerman"}
{"customerid": 305, "firstname": "Carla", "lastname": "Adams", "fullname": "Carla Adams"}
{"customerid": 301, "firstname": "Frances", "lastname": "Adams", "fullname": "Frances Adams"}
{"customerid": 307, "firstname": "Jay", "lastname": "Adams", "fullname": "Jay Adams"}
{"customerid": 309, "firstname": "Ronald", "lastname": "Adina", "fullname": "Ronald Adina"}
```

1 Initialized (additional servers needed) Glue PySpark | Idle ✓ CodeWhisperer



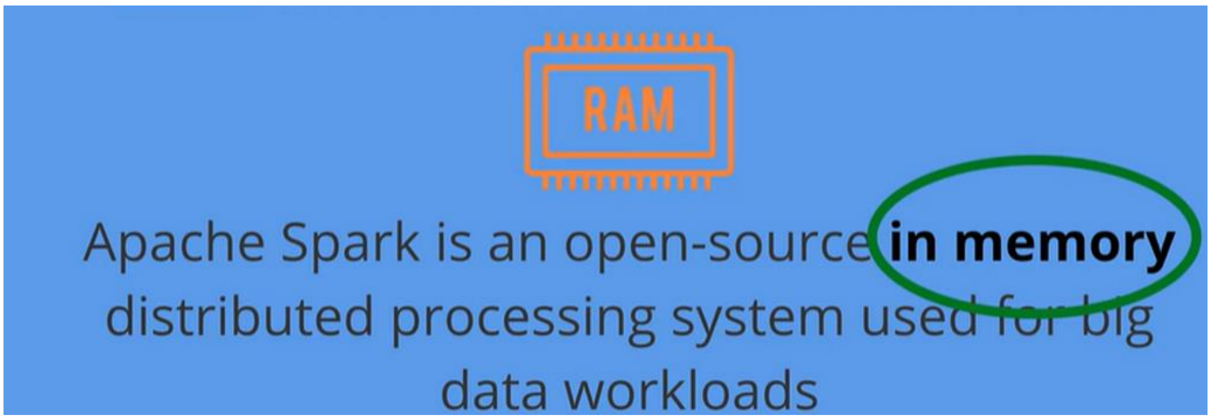
GLUE INTERACTIVE SESSIONS

- A programmatic and visual interface for building and testing extract, transform, and load (ETL) scripts for data preparation.
- Interactive sessions run Apache Spark analytics applications and provide on-demand access to a remote Spark runtime environment.
- AWS Glue transparently manages serverless Spark for these interactive

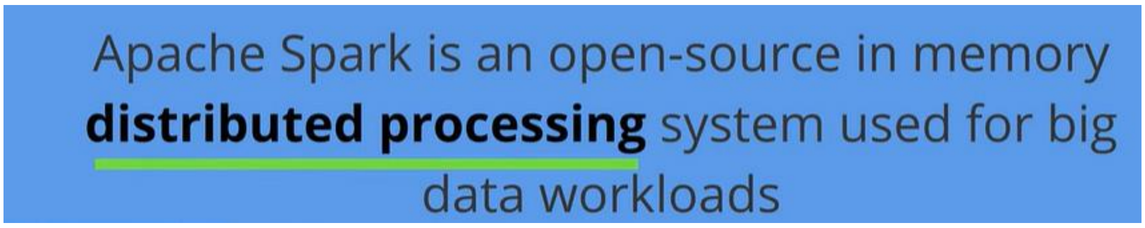


FUNDAMENTALS OF SPARK FOR GLUE

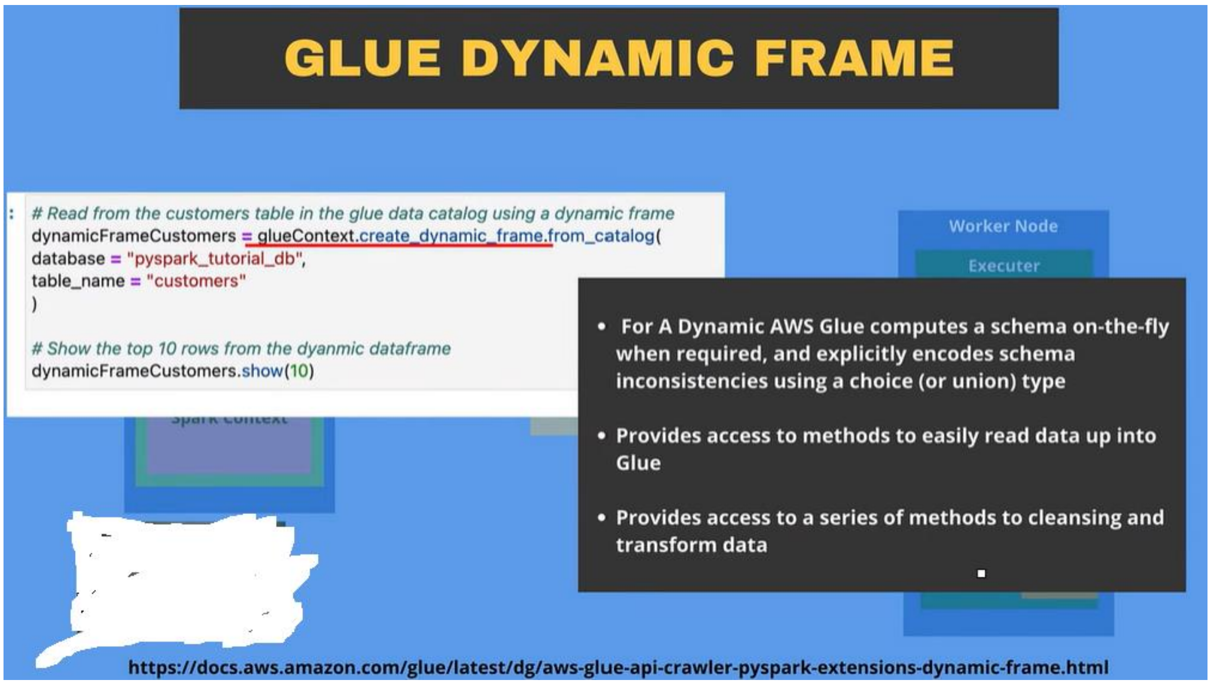
Apache Spark is an open-source in memory distributed processing system used for big data workloads



Apache Spark is an open-source **in memory** distributed processing system used for big data workloads



Apache Spark is an open-source in memory **distributed processing** system used for big data workloads



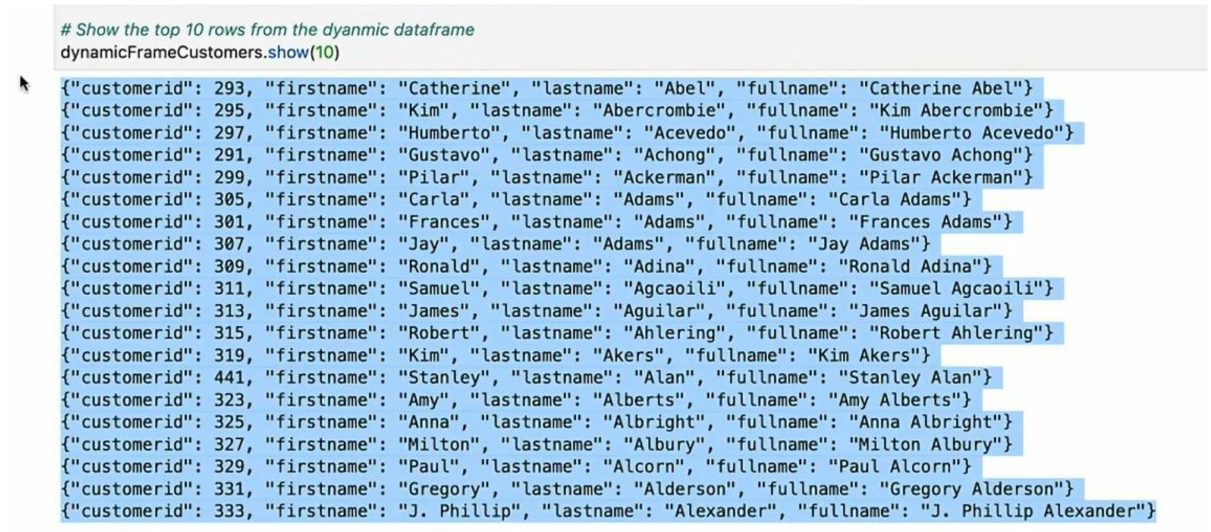
GLUE DYNAMIC FRAME

```python
# Read from the customers table in the glue data catalog using a dynamic frame
dynamicFrameCustomers = glueContext.create_dynamic_frame.from_catalog(
database = "pyspark_tutorial_db",
table_name = "customers"
)

# Show the top 10 rows from the dyanmic dataframe
dynamicFrameCustomers.show(10)
```

- For A Dynamic AWS Glue computes a schema on-the-fly when required, and explicitly encodes schema inconsistencies using a choice (or union) type
- Provides access to methods to easily read data up into Glue
- Provides access to a series of methods to cleansing and transform data

[redacted]

https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-crawler-pyspark-extensions-dynamic-frame.html



```python
# Show the top 10 rows from the dyanmic dataframe
dynamicFrameCustomers.show(10)
```

```
{"customerid": 293, "firstname": "Catherine", "lastname": "Abel", "fullname": "Catherine Abel"}
{"customerid": 295, "firstname": "Kim", "lastname": "Abercrombie", "fullname": "Kim Abercrombie"}
{"customerid": 297, "firstname": "Humberto", "lastname": "Acevedo", "fullname": "Humberto Acevedo"}
{"customerid": 291, "firstname": "Gustavo", "lastname": "Achong", "fullname": "Gustavo Achong"}
{"customerid": 299, "firstname": "Pilar", "lastname": "Ackerman", "fullname": "Pilar Ackerman"}
{"customerid": 305, "firstname": "Carla", "lastname": "Adams", "fullname": "Carla Adams"}
{"customerid": 301, "firstname": "Frances", "lastname": "Adams", "fullname": "Frances Adams"}
{"customerid": 307, "firstname": "Jay", "lastname": "Adams", "fullname": "Jay Adams"}
{"customerid": 309, "firstname": "Ronald", "lastname": "Adina", "fullname": "Ronald Adina"}
{"customerid": 311, "firstname": "Samuel", "lastname": "Agcaoili", "fullname": "Samuel Agcaoili"}
{"customerid": 313, "firstname": "James", "lastname": "Aguilar", "fullname": "James Aguilar"}
{"customerid": 315, "firstname": "Robert", "lastname": "Ahlering", "fullname": "Robert Ahlering"}
{"customerid": 319, "firstname": "Kim", "lastname": "Akers", "fullname": "Kim Akers"}
{"customerid": 441, "firstname": "Stanley", "lastname": "Alan", "fullname": "Stanley Alan"}
{"customerid": 323, "firstname": "Amy", "lastname": "Alberts", "fullname": "Amy Alberts"}
{"customerid": 325, "firstname": "Anna", "lastname": "Albright", "fullname": "Anna Albright"}
{"customerid": 327, "firstname": "Milton", "lastname": "Albury", "fullname": "Milton Albury"}
{"customerid": 329, "firstname": "Paul", "lastname": "Alcorn", "fullname": "Paul Alcorn"}
{"customerid": 331, "firstname": "Gregory", "lastname": "Alderson", "fullname": "Gregory Alderson"}
{"customerid": 333, "firstname": "J. Phillip", "lastname": "Alexander", "fullname": "J. Phillip Alexander"}
```

We call it dynamic frame when we have many records together.

Aws gives us numerous methods to use with a dynamic frame

Dynamic frame has different methods to read the data like

RDD

JDBC

S3

Glue Data Catalog – from_catalog is used in the code

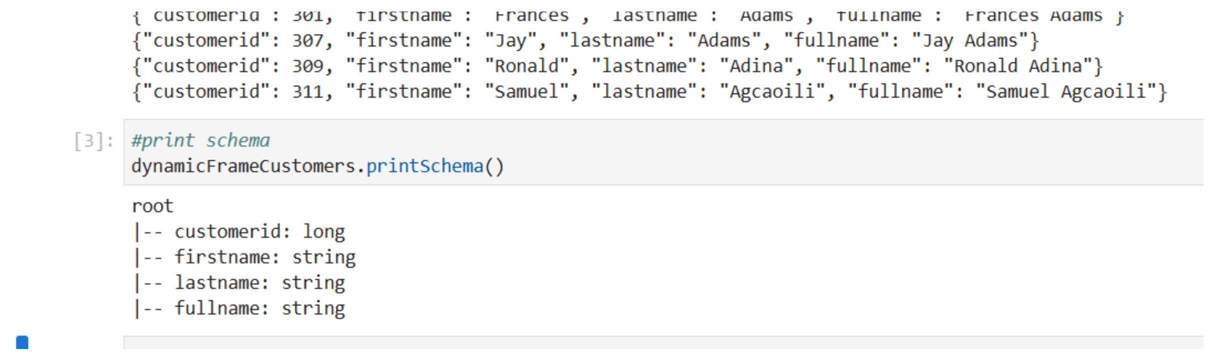
Here is the documentation link for Dynamicframe class:

https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-crawler-pyspark-extensions-dynamic-frame.html

U will also have practical examples in it, so go through documentation

Let's start our code

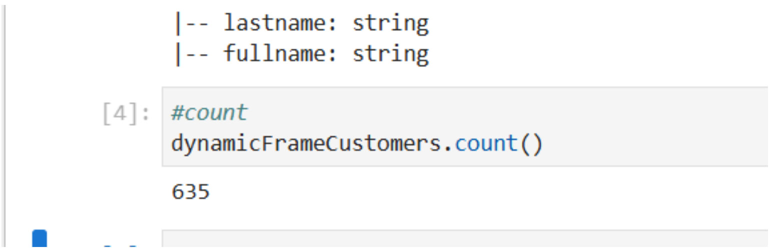
Print schema: S is capital in code



```
{"customerid": 301, "firstname": "Frances", "lastname": "Adams", "fullname": "Frances Adams"}
{"customerid": 307, "firstname": "Jay", "lastname": "Adams", "fullname": "Jay Adams"}
{"customerid": 309, "firstname": "Ronald", "lastname": "Adina", "fullname": "Ronald Adina"}
{"customerid": 311, "firstname": "Samuel", "lastname": "Agcaoili", "fullname": "Samuel Agcaoili"}
```

```
[3]: #print schema
     dynamicFrameCustomers.printSchema()
```

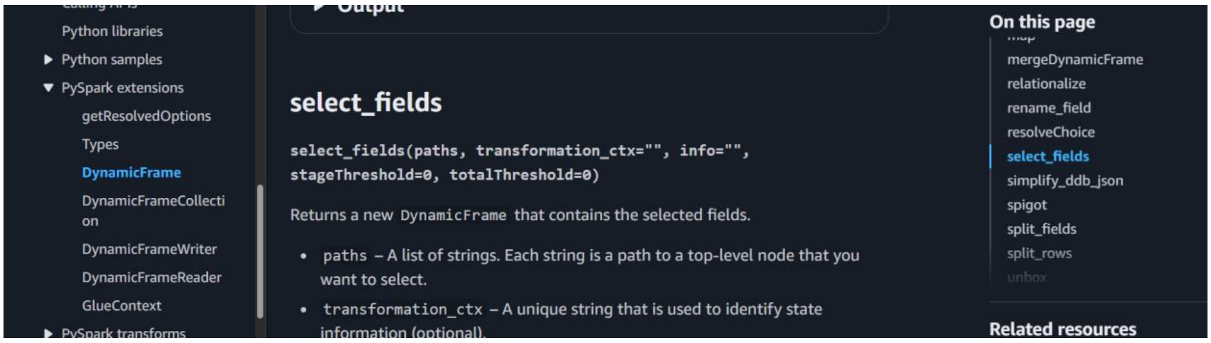
```
root
|-- customerid: long
|-- firstname: string
|-- lastname: string
|-- fullname: string
```



```
|-- lastname: string
|-- fullname: string
```

```
[4]: #count
     dynamicFrameCustomers.count()
```

```
635
```

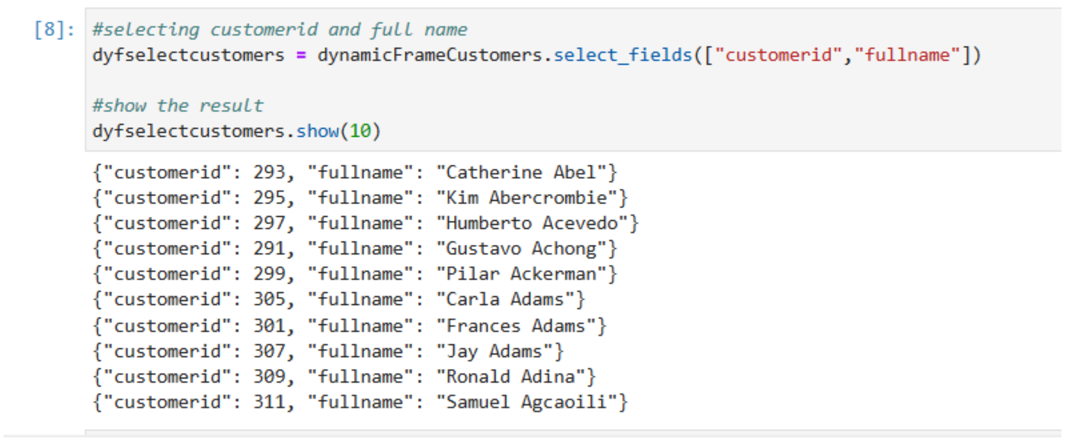


select_fields

`select_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)`

Returns a new `DynamicFrame` that contains the selected fields.

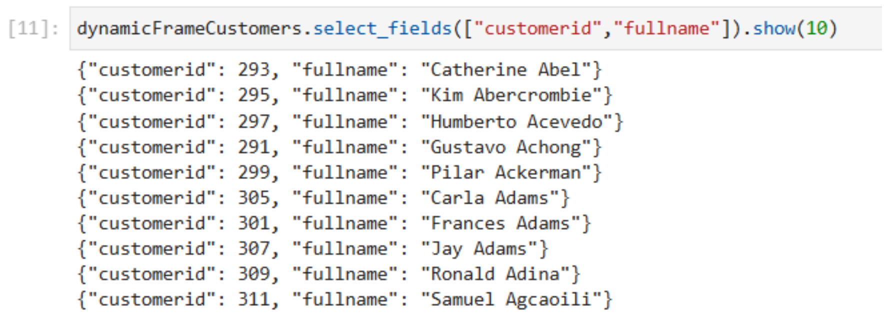
- `paths` – A list of strings. Each string is a path to a top-level node that you want to select.
- `transformation_ctx` – A unique string that is used to identify state information (optional).



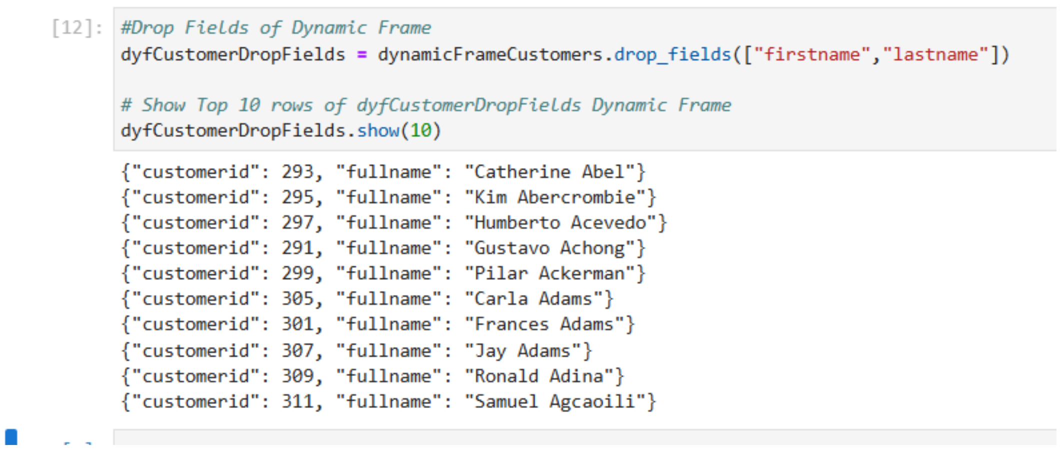
```
[8]: #selecting customerid and full name
     dyfselectcustomers = dynamicFrameCustomers.select_fields(["customerid","fullname"])

     #show the result
     dyfselectcustomers.show(10)
```

```
{"customerid": 293, "fullname": "Catherine Abel"}
{"customerid": 295, "fullname": "Kim Abercrombie"}
{"customerid": 297, "fullname": "Humberto Acevedo"}
{"customerid": 291, "fullname": "Gustavo Achong"}
{"customerid": 299, "fullname": "Pilar Ackerman"}
{"customerid": 305, "fullname": "Carla Adams"}
{"customerid": 301, "fullname": "Frances Adams"}
{"customerid": 307, "fullname": "Jay Adams"}
{"customerid": 309, "fullname": "Ronald Adina"}
{"customerid": 311, "fullname": "Samuel Agcaoili"}
```

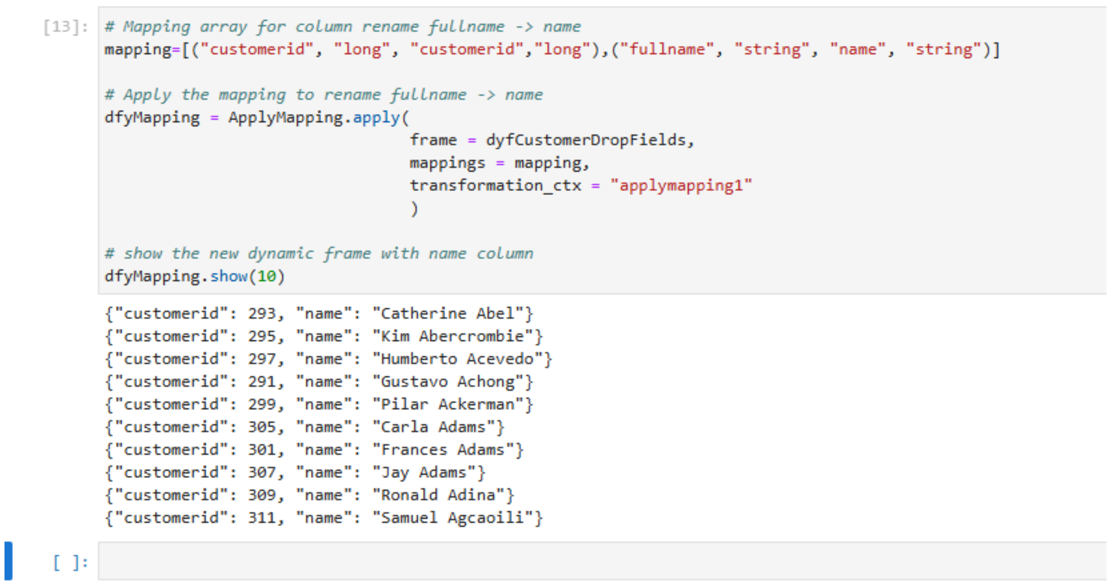
Note: always follow the documentation if u need any help



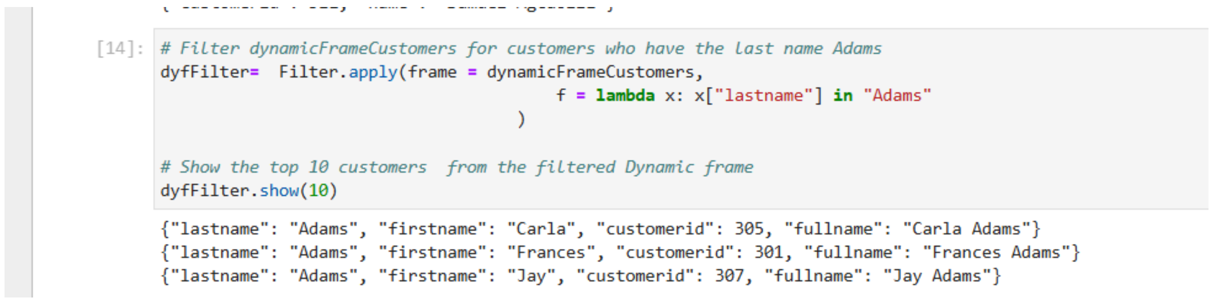
```
[11]: dynamicFrameCustomers.select_fields(["customerid","fullname"]).show(10)

{"customerid": 293, "fullname": "Catherine Abel"}
{"customerid": 295, "fullname": "Kim Abercrombie"}
{"customerid": 297, "fullname": "Humberto Acevedo"}
{"customerid": 291, "fullname": "Gustavo Achong"}
{"customerid": 299, "fullname": "Pilar Ackerman"}
{"customerid": 305, "fullname": "Carla Adams"}
{"customerid": 301, "fullname": "Frances Adams"}
{"customerid": 307, "fullname": "Jay Adams"}
{"customerid": 309, "fullname": "Ronald Adina"}
{"customerid": 311, "fullname": "Samuel Agcaoili"}
```



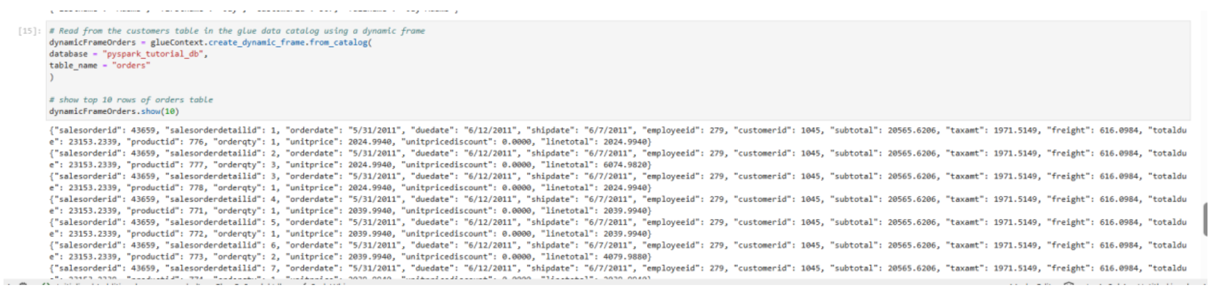
```
[12]: #Drop Fields of Dynamic Frame
      dyfCustomerDropFields = dynamicFrameCustomers.drop_fields(["firstname","lastname"])

      # Show Top 10 rows of dyfCustomerDropFields Dynamic Frame
      dyfCustomerDropFields.show(10)

{"customerid": 293, "fullname": "Catherine Abel"}
{"customerid": 295, "fullname": "Kim Abercrombie"}
{"customerid": 297, "fullname": "Humberto Acevedo"}
{"customerid": 291, "fullname": "Gustavo Achong"}
{"customerid": 299, "fullname": "Pilar Ackerman"}
{"customerid": 305, "fullname": "Carla Adams"}
{"customerid": 301, "fullname": "Frances Adams"}
{"customerid": 307, "fullname": "Jay Adams"}
{"customerid": 309, "fullname": "Ronald Adina"}
{"customerid": 311, "fullname": "Samuel Agcaoili"}
```



```
[13]: # Mapping array for column rename fullname -> name
      mapping=[("customerid", "long", "customerid","long"),("fullname", "string", "name", "string")]

      # Apply the mapping to rename fullname -> name
      dfyMapping = ApplyMapping.apply(
                                      frame = dyfCustomerDropFields,
                                      mappings = mapping,
                                      transformation_ctx = "applymapping1"
                                      )

      # show the new dynamic frame with name column
      dfyMapping.show(10)

{"customerid": 293, "name": "Catherine Abel"}
{"customerid": 295, "name": "Kim Abercrombie"}
{"customerid": 297, "name": "Humberto Acevedo"}
{"customerid": 291, "name": "Gustavo Achong"}
{"customerid": 299, "name": "Pilar Ackerman"}
{"customerid": 305, "name": "Carla Adams"}
{"customerid": 301, "name": "Frances Adams"}
{"customerid": 307, "name": "Jay Adams"}
{"customerid": 309, "name": "Ronald Adina"}
{"customerid": 311, "name": "Samuel Agcaoili"}

[ ]:
```

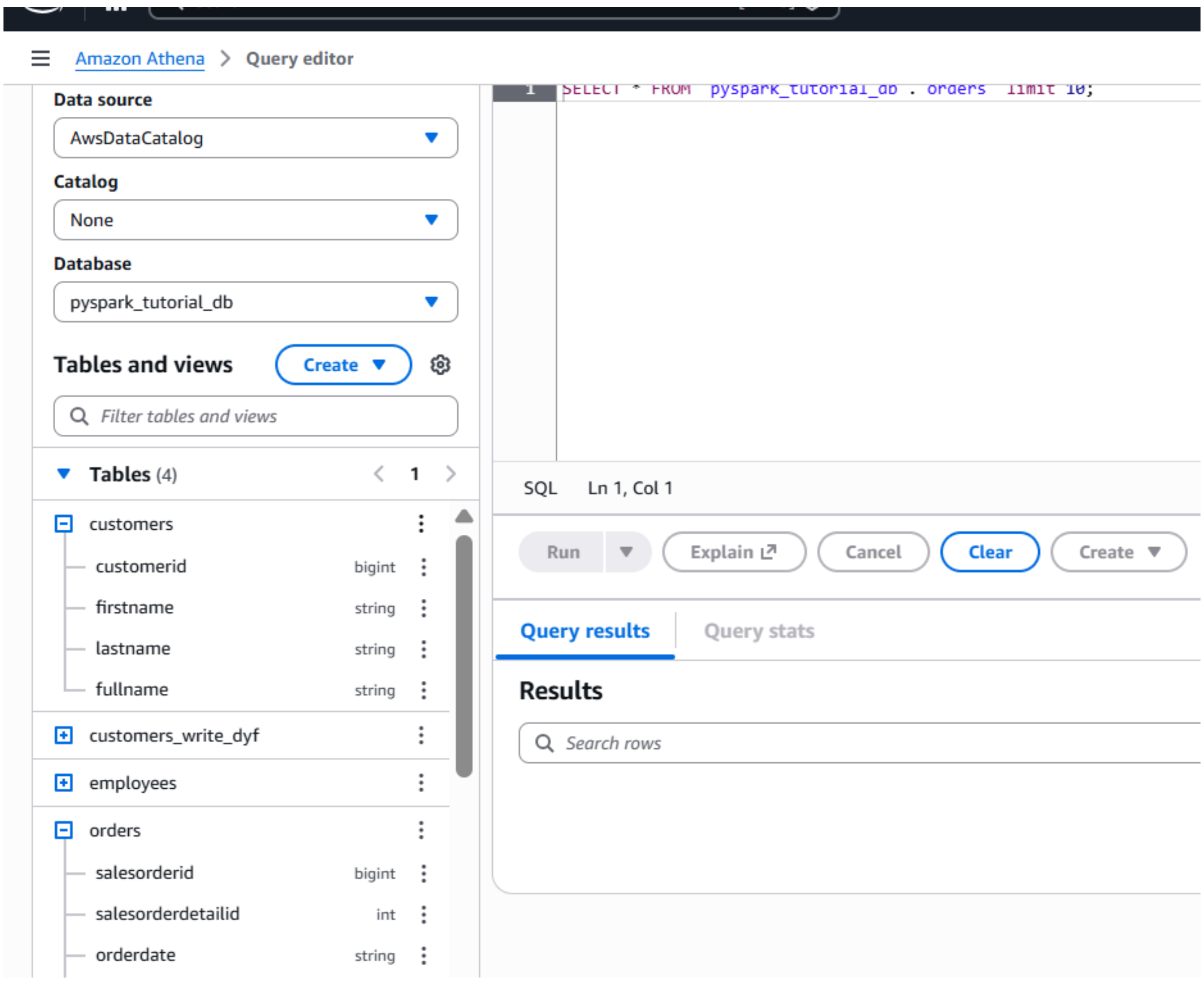


```
[14]: # Filter dynamicFrameCustomers for customers who have the last name Adams
      dyfFilter=  Filter.apply(frame = dynamicFrameCustomers,
                                           f = lambda x: x["lastname"] in "Adams"
                                     )

      # Show the top 10 customers  from the filtered Dynamic frame
      dyfFilter.show(10)

{"lastname": "Adams", "firstname": "Carla", "customerid": 305, "fullname": "Carla Adams"}
{"lastname": "Adams", "firstname": "Frances", "customerid": 301, "fullname": "Frances Adams"}
{"lastname": "Adams", "firstname": "Jay", "customerid": 307, "fullname": "Jay Adams"}
```



```
[15]: # Read from the customers table in the glue data catalog using a dynamic frame
      dynamicFrameOrders = glueContext.create_dynamic_frame.from_catalog(
      database = "pyspark_tutorial_db",
      table_name = "orders"
      )

      # show top 10 rows of orders table
      dynamicFrameOrders.show(10)

{"salesorderid": 43659, "salesorderdetailid": 1, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
e": 23153.2339, "productid": 776, "orderqty": 1, "unitprice": 2024.9940, "unitpricediscount": 0.0000, "linetotal": 2024.9940}
{"salesorderid": 43659, "salesorderdetailid": 2, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
e": 23153.2339, "productid": 777, "orderqty": 3, "unitprice": 2024.9940, "unitpricediscount": 0.0000, "linetotal": 6074.9820}
{"salesorderid": 43659, "salesorderdetailid": 3, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
e": 23153.2339, "productid": 778, "orderqty": 1, "unitprice": 2024.9940, "unitpricediscount": 0.0000, "linetotal": 2024.9940}
{"salesorderid": 43659, "salesorderdetailid": 4, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
e": 23153.2339, "productid": 771, "orderqty": 1, "unitprice": 2039.9940, "unitpricediscount": 0.0000, "linetotal": 2039.9940}
{"salesorderid": 43659, "salesorderdetailid": 5, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
e": 23153.2339, "productid": 772, "orderqty": 1, "unitprice": 2039.9940, "unitpricediscount": 0.0000, "linetotal": 2039.9940}
{"salesorderid": 43659, "salesorderdetailid": 6, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
e": 23153.2339, "productid": 773, "orderqty": 2, "unitprice": 2039.9940, "unitpricediscount": 0.0000, "linetotal": 4079.9880}
{"salesorderid": 43659, "salesorderdetailid": 7, "orderdate": "5/31/2011", "duedate": "6/12/2011", "shipdate": "6/7/2011", "employeeid": 279, "customerid": 1045, "subtotal": 20565.6206, "taxamt": 1971.5149, "freight": 616.0984, "totaldu
```

Note: u see that the interface in athena of ur tables in s3, means the data got crawled by yaml file and data crawling is done in prev AWS project



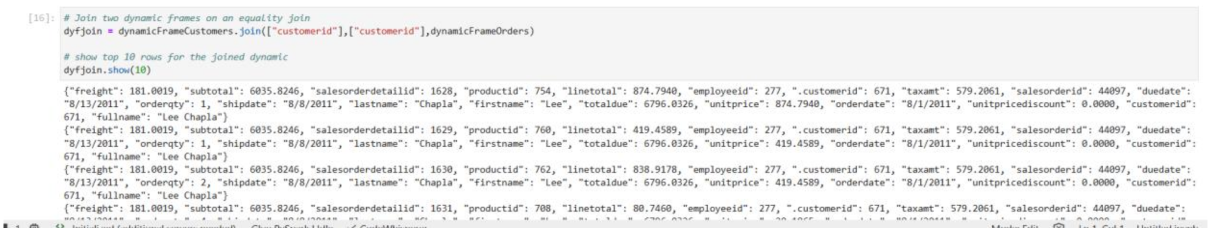
```
SELECT * FROM pyspark_tutorial_db . orders limit 10;
```

Data source: AwsDataCatalog

Catalog: None

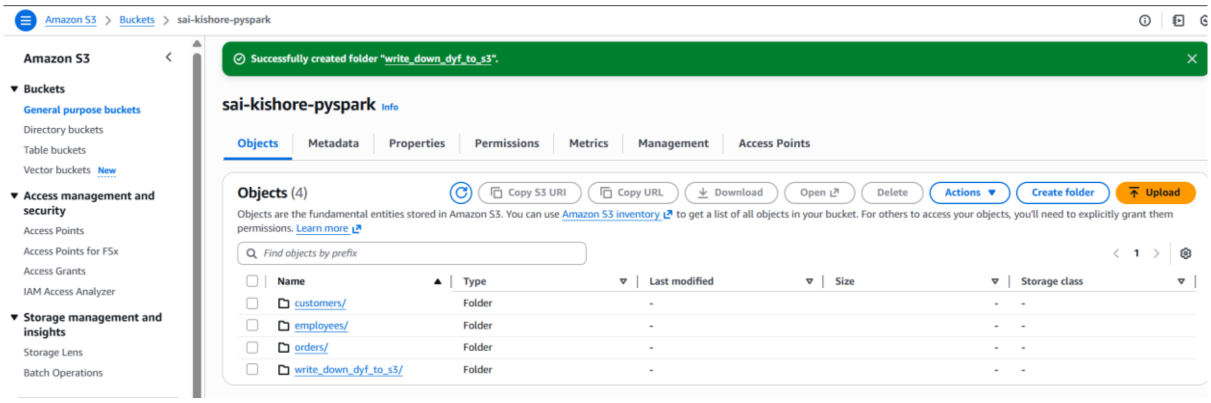
Database: pyspark_tutorial_db

customers	
customerid	bigint
firstname	string
lastname	string
fullname	string
customers_write_dyf	
employees	
orders	
salesorderid	bigint
salesorderdetailid	int
orderdate	string



```
[16]: # Join two dynamic frames on an equality join
      dyfjoin = dynamicFrameCustomers.join(["customerid"],["customerid"],dynamicFrameOrders)

      # show top 10 rows for the joined dynamic
      dyfjoin.show(10)

{"freight": 181.0019, "subtotal": 6035.8246, "salesorderdetailid": 1628, "productid": 754, "linetotal": 874.7940, "employeeid": 277, ".customerid": 671, "taxamt": 579.2061, "salesorderid": 44097, "duedate":
"8/13/2011", "orderqty": 1, "shipdate": "8/8/2011", "lastname": "Chapla", "firstname": "Lee", "totaldue": 6796.0326, "unitprice": 874.7940, "orderdate": "8/1/2011", "unitpricediscount": 0.0000, "customerid":
671, "fullname": "Lee Chapla"}
{"freight": 181.0019, "subtotal": 6035.8246, "salesorderdetailid": 1629, "productid": 760, "linetotal": 419.4589, "employeeid": 277, ".customerid": 671, "taxamt": 579.2061, "salesorderid": 44097, "duedate":
"8/13/2011", "orderqty": 1, "shipdate": "8/8/2011", "lastname": "Chapla", "firstname": "Lee", "totaldue": 6796.0326, "unitprice": 419.4589, "orderdate": "8/1/2011", "unitpricediscount": 0.0000, "customerid":
671, "fullname": "Lee Chapla"}
{"freight": 181.0019, "subtotal": 6035.8246, "salesorderdetailid": 1630, "productid": 762, "linetotal": 838.9178, "employeeid": 277, ".customerid": 671, "taxamt": 579.2061, "salesorderid": 44097, "duedate":
"8/13/2011", "orderqty": 2, "shipdate": "8/8/2011", "lastname": "Chapla", "firstname": "Lee", "totaldue": 6796.0326, "unitprice": 419.4589, "orderdate": "8/1/2011", "unitpricediscount": 0.0000, "customerid":
671, "fullname": "Lee Chapla"}
{"freight": 181.0019, "subtotal": 6035.8246, "salesorderdetailid": 1631, "productid": 708, "linetotal": 80.7460, "employeeid": 277, ".customerid": 671, "taxamt": 579.2061, "salesorderid": 44097, "duedate":
```

Creating a folder name “write_down_dyf_to_s3”


Amazon S3 > Buckets > sai-kishore-pyspark

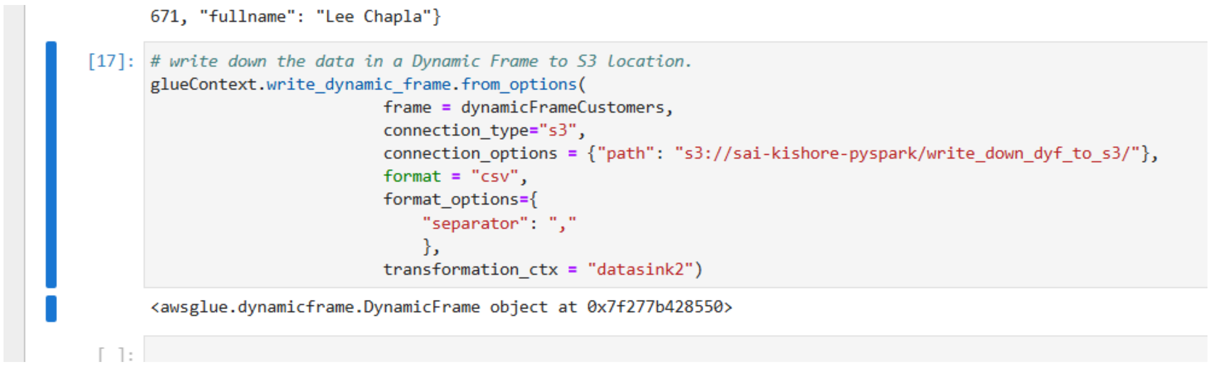
Successfully created folder "write_down_dyf_to_s3".

sai-kishore-pyspark

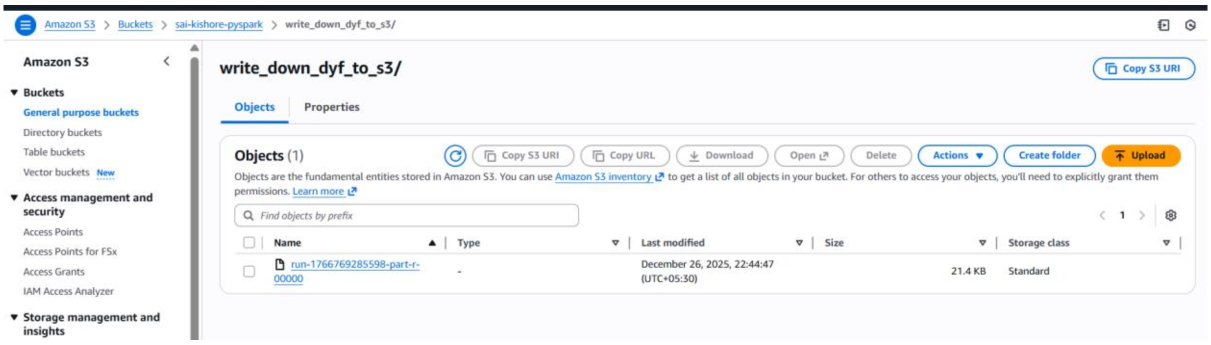
Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
customers/	Folder	-	-	-
employees/	Folder	-	-	-
orders/	Folder	-	-	-
write_down_dyf_to_s3/	Folder	-	-	-


```
671, "fullname": "Lee Chapla"}

[17]: # write down the data in a Dynamic Frame to S3 location.
      glueContext.write_dynamic_frame.from_options(
                                  frame = dynamicFrameCustomers,
                                  connection_type="s3",
                                  connection_options = {"path": "s3://sai-kishore-pyspark/write_down_dyf_to_s3/"},
                                  format = "csv",
                                  format_options={
                                      "separator": ","
                                      },
                                  transformation_ctx = "datasink2")

      <awsglue.dynamicframe.DynamicFrame object at 0x7f277b428550>

[ ]:
```

Now check ur s3 bucket


Amazon S3 > Buckets > sai-kishore-pyspark > write_down_dyf_to_s3/

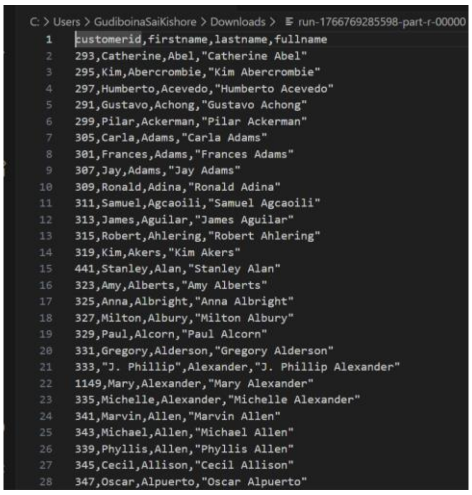
write_down_dyf_to_s3/

Objects (1)

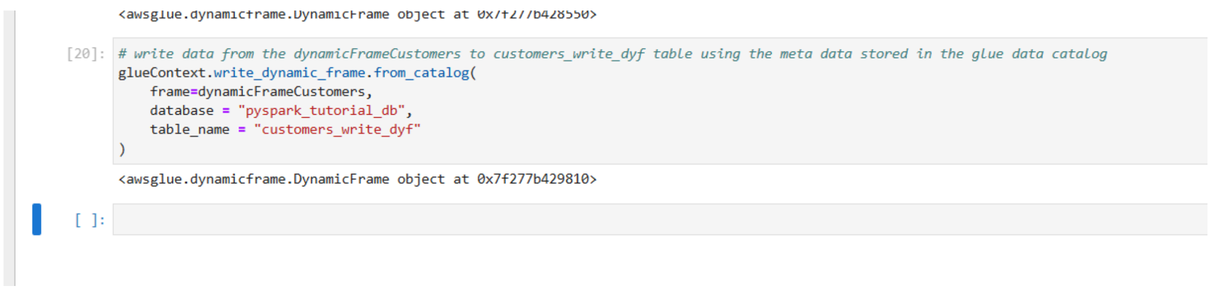
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
run-1766769285598-part-r-00000	-	December 26, 2025, 22:44:47 (UTC+05:30)	21.4 KB	Standard

Download the file and check the data


C: > Users > GudiboinaSaiKishore > Downloads > run-1766769285598-part-r-00000

```
customerid,firstname,lastname,fullname
293,Catherine,Abel,"Catherine Abel"
295,Kim,Abercrombie,"Kim Abercrombie"
297,Humberto,Acevedo,"Humberto Acevedo"
291,Gustavo,Achong,"Gustavo Achong"
299,Pilar,Ackerman,"Pilar Ackerman"
305,Carla,Adams,"Carla Adams"
301,Frances,Adams,"Frances Adams"
307,Jay,Adams,"Jay Adams"
309,Ronald,Adina,"Ronald Adina"
311,Samuel,Agcaoili,"Samuel Agcaoili"
313,James,Aguilar,"James Aguilar"
315,Robert,Ahlering,"Robert Ahlering"
319,Kim,Akers,"Kim Akers"
441,Stanley,Alan,"Stanley Alan"
323,Amy,Alberts,"Amy Alberts"
325,Anna,Albright,"Anna Albright"
327,Milton,Albury,"Milton Albury"
329,Paul,Alcorn,"Paul Alcorn"
331,Gregory,Alderson,"Gregory Alderson"
333,"J. Phillip",Alexander,"J. Phillip Alexander"
1149,Mary,Alexander,"Mary Alexander"
335,Michelle,Alexander,"Michelle Alexander"
341,Marvin,Allen,"Marvin Allen"
343,Michael,Allen,"Michael Allen"
339,Phyllis,Allen,"Phyllis Allen"
345,Cecil,Allison,"Cecil Allison"
347,Oscar,Alpuerto,"Oscar Alpuerto"
```

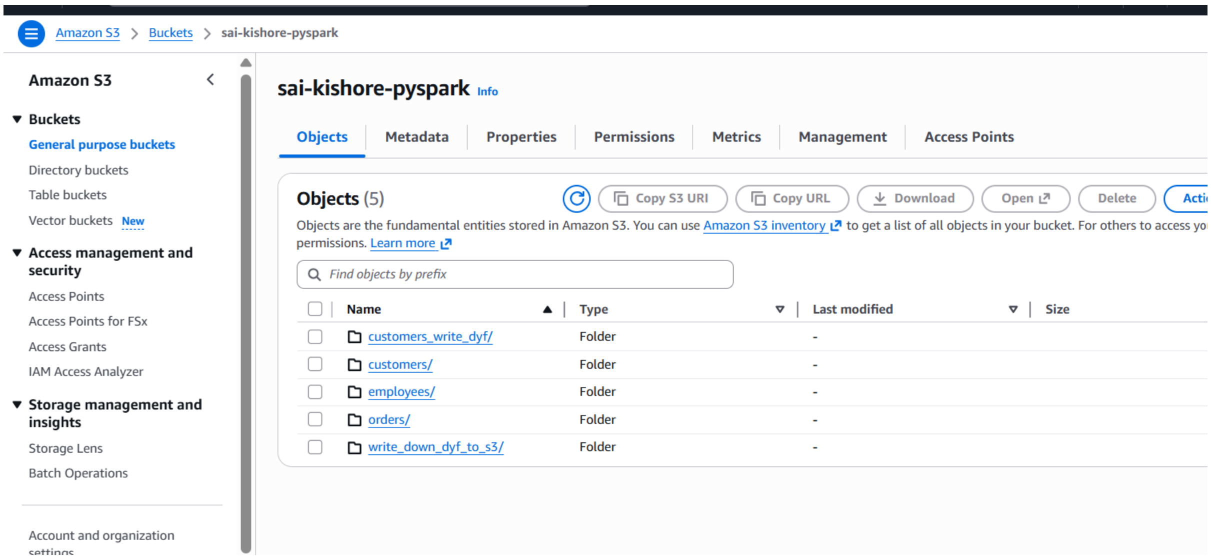


```
<awsglue.dynamicframe.DynamicFrame object at 0x7f277b428550>
```

```
[20]: # write data from the dynamicFrameCustomers to customers_write_dyf table using the meta data stored in the glue data catalog
glueContext.write_dynamic_frame.from_catalog(
    frame=dynamicFrameCustomers,
    database = "pyspark_tutorial_db",
    table_name = "customers_write_dyf"
)
```

```
<awsglue.dynamicframe.DynamicFrame object at 0x7f277b429810>
```

```
[ ]:
```



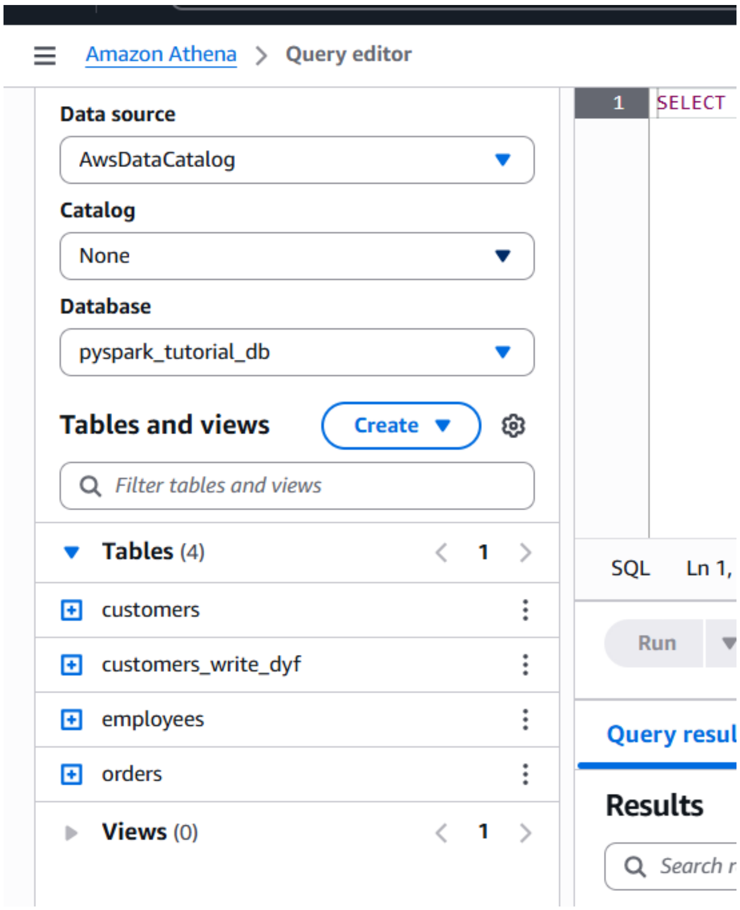
Amazon S3 > Buckets > sai-kishore-pyspark

sai-kishore-pyspark Info

Objects (5)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access yo permissions. Learn more

Name	Type	Last modified	Size
customers_write_dyf/	Folder	-	
customers/	Folder	-	
employees/	Folder	-	
orders/	Folder	-	
write_down_dyf_to_s3/	Folder	-	



Amazon Athena > Query editor

Data source: AwsDataCatalog
Catalog: None
Database: pyspark_tutorial_db

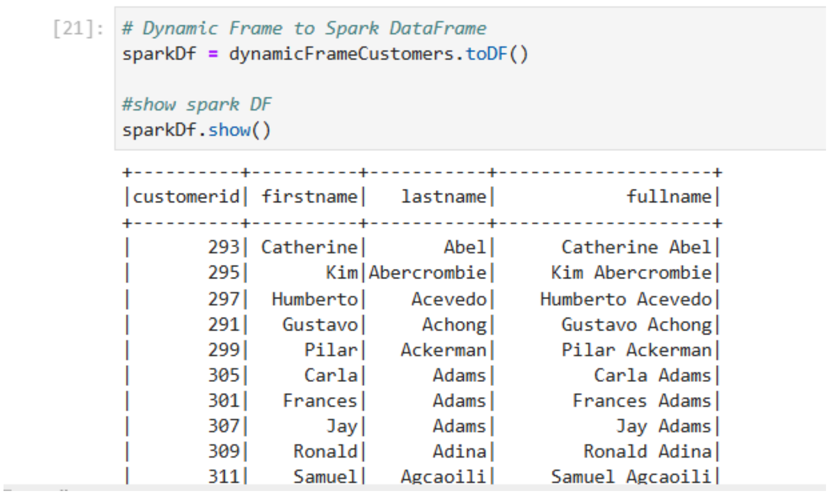
Tables and views

Tables (4)
- customers
- customers_write_dyf
- employees
- orders

Views (0)

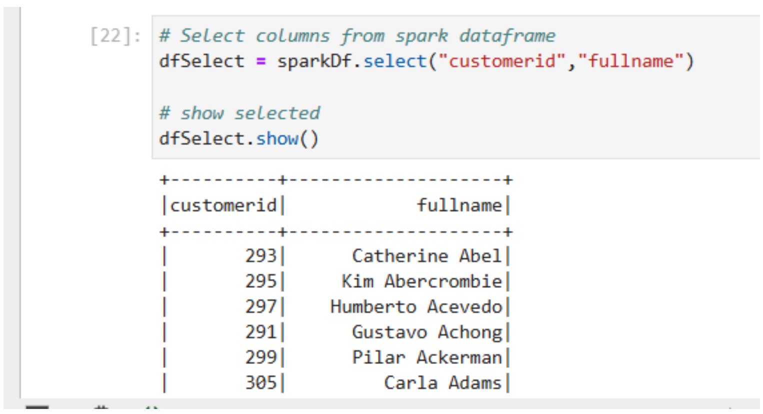
```
1 SELECT
```

Note: write_down_dyf_to_s3 didn’t come to athena



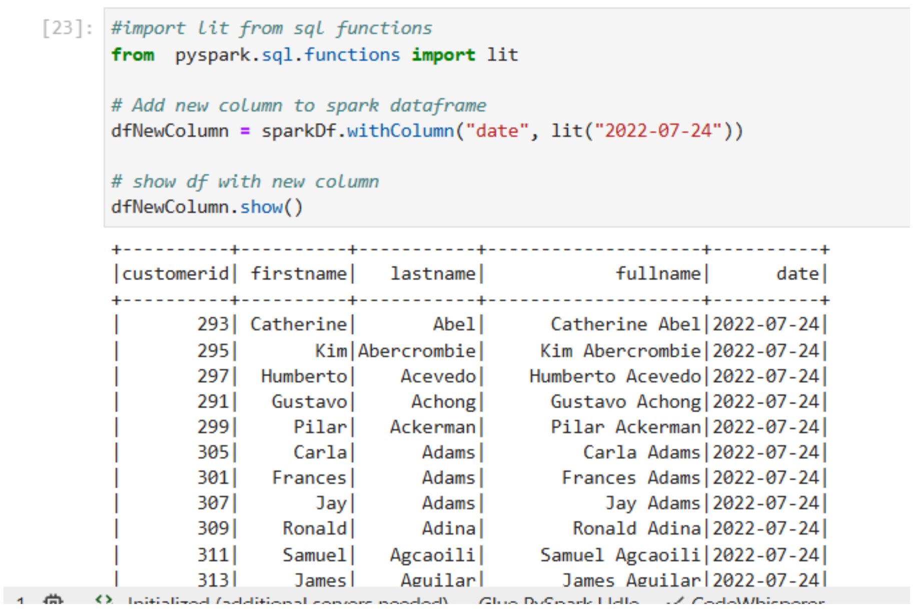
```
[21]: # Dynamic Frame to Spark DataFrame
      sparkDf = dynamicFrameCustomers.toDF()

      #show spark DF
      sparkDf.show()
```

```
+----------+---------+-----------+-------------------+
|customerid|firstname|   lastname|           fullname|
+----------+---------+-----------+-------------------+
|       293|Catherine|       Abel|     Catherine Abel|
|       295|      Kim|Abercrombie|    Kim Abercrombie|
|       297| Humberto|    Acevedo|   Humberto Acevedo|
|       291|  Gustavo|     Achong|     Gustavo Achong|
|       299|    Pilar|   Ackerman|     Pilar Ackerman|
|       305|    Carla|      Adams|        Carla Adams|
|       301|  Frances|      Adams|      Frances Adams|
|       307|      Jay|      Adams|          Jay Adams|
|       309|   Ronald|      Adina|       Ronald Adina|
|       311|   Samuel|   Agcaoili|    Samuel Agcaoili|
```



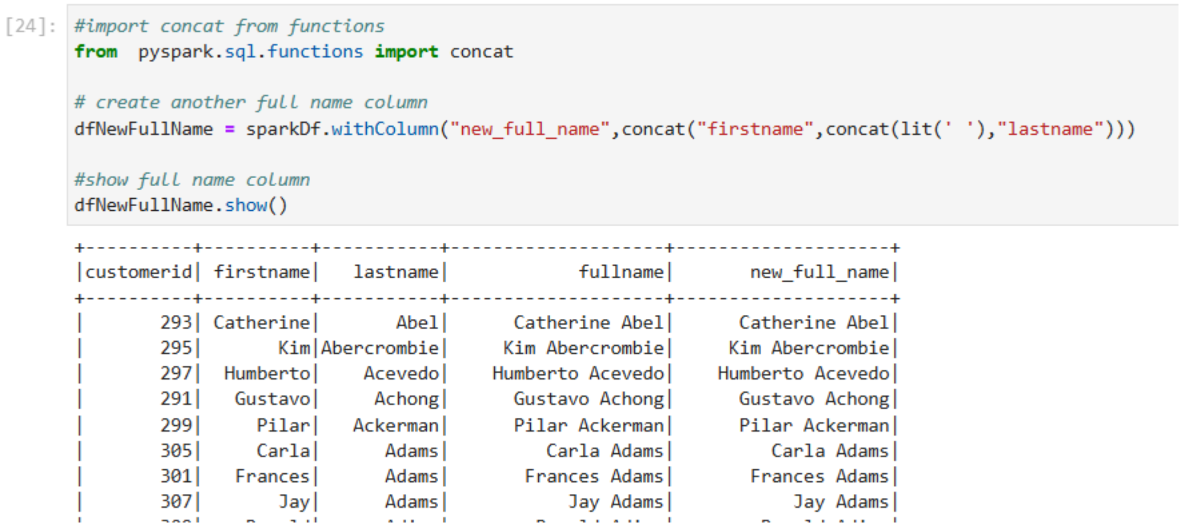
```
[22]: # Select columns from spark dataframe
      dfSelect = sparkDf.select("customerid","fullname")

      # show selected
      dfSelect.show()
```

```
+----------+-------------------+
|customerid|           fullname|
+----------+-------------------+
|       293|     Catherine Abel|
|       295|    Kim Abercrombie|
|       297|   Humberto Acevedo|
|       291|     Gustavo Achong|
|       299|     Pilar Ackerman|
|       305|        Carla Adams|
```



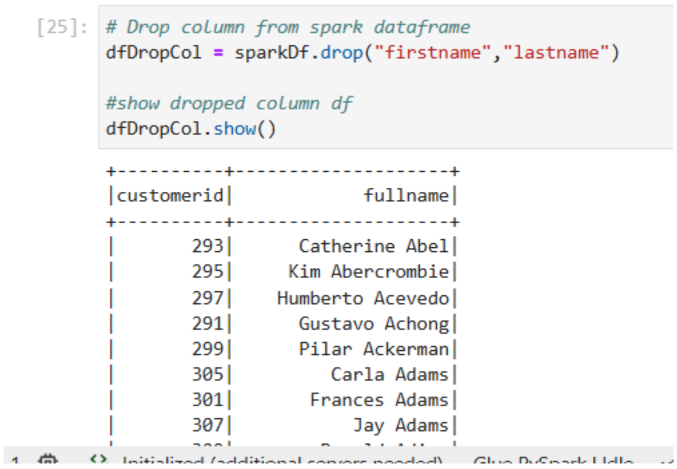
```
[23]: #import lit from sql functions
      from  pyspark.sql.functions import lit

      # Add new column to spark dataframe
      dfNewColumn = sparkDf.withColumn("date", lit("2022-07-24"))

      # show df with new column
      dfNewColumn.show()
```

```
+----------+----------+-----------+-------------------+----------+
|customerid| firstname|   lastname|           fullname|      date|
+----------+----------+-----------+-------------------+----------+
|       293| Catherine|       Abel|     Catherine Abel|2022-07-24|
|       295|       Kim|Abercrombie|    Kim Abercrombie|2022-07-24|
|       297|  Humberto|    Acevedo|   Humberto Acevedo|2022-07-24|
|       291|   Gustavo|     Achong|     Gustavo Achong|2022-07-24|
|       299|     Pilar|   Ackerman|     Pilar Ackerman|2022-07-24|
|       305|     Carla|      Adams|        Carla Adams|2022-07-24|
|       301|   Frances|      Adams|      Frances Adams|2022-07-24|
|       307|       Jay|      Adams|          Jay Adams|2022-07-24|
|       309|    Ronald|      Adina|       Ronald Adina|2022-07-24|
|       311|    Samuel|   Agcaoili|    Samuel Agcaoili|2022-07-24|
|       313|     James|    Aguilar|      James Aguilar|2022-07-24|
```



```
[24]: #import concat from functions
      from  pyspark.sql.functions import concat

      # create another full name column
      dfNewFullName = sparkDf.withColumn("new_full_name",concat("firstname",concat(lit(' '),"lastname")))

      #show full name column
      dfNewFullName.show()
```

```
+----------+----------+-----------+-------------------+-------------------+
|customerid| firstname|   lastname|           fullname|      new_full_name|
+----------+----------+-----------+-------------------+-------------------+
|       293| Catherine|       Abel|     Catherine Abel|     Catherine Abel|
|       295|       Kim|Abercrombie|    Kim Abercrombie|    Kim Abercrombie|
|       297|  Humberto|    Acevedo|   Humberto Acevedo|   Humberto Acevedo|
|       291|   Gustavo|     Achong|     Gustavo Achong|     Gustavo Achong|
|       299|     Pilar|   Ackerman|     Pilar Ackerman|     Pilar Ackerman|
|       305|     Carla|      Adams|        Carla Adams|        Carla Adams|
|       301|   Frances|      Adams|      Frances Adams|      Frances Adams|
|       307|       Jay|      Adams|          Jay Adams|          Jay Adams|
```



```
[25]: # Drop column from spark dataframe
      dfDropCol = sparkDf.drop("firstname","lastname")

      #show dropped column df
      dfDropCol.show()
```

```
+----------+-------------------+
|customerid|           fullname|
+----------+-------------------+
|       293|     Catherine Abel|
|       295|    Kim Abercrombie|
|       297|   Humberto Acevedo|
|       291|     Gustavo Achong|
|       299|     Pilar Ackerman|
|       305|        Carla Adams|
|       301|      Frances Adams|
|       307|          Jay Adams|
```

[26]:
```python
# Rename column in Spark dataframe
dfRenameCol = sparkDf.withColumnRenamed("fullname","full_name")

#show renamed column dataframe
dfRenameCol.show()
```

```
+----------+---------+-----------+-------------------+
|customerid| firstname|   lastname|          full_name|
+----------+---------+-----------+-------------------+
|       293| Catherine|       Abel|     Catherine Abel|
|       295|       Kim|Abercrombie|    Kim Abercrombie|
|       297|  Humberto|    Acevedo|   Humberto Acevedo|
|       291|   Gustavo|     Achong|     Gustavo Achong|
|       299|     Pilar|   Ackerman|     Pilar Ackerman|
|       305|     Carla|      Adams|        Carla Adams|
|       301|   Frances|      Adams|      Frances Adams|
```

[27]:
```python
# Group by lastname then print counts of lastname and show
sparkDf.groupBy("lastname").count().show()
```

```
+--------+-----+
|lastname|count|
+--------+-----+
|  Achong|    1|
|  Bailey|    1|
|   Caron|    1|
|   Casts|    1|
|   Curry|    1|
| Desalvo|    1|
| Dockter|    1|
|    Dyck|    1|
|  Farino|    1|
| Fluegel|    1|
|   Ganio|    1|
```

```
only showing top 20 rows
```

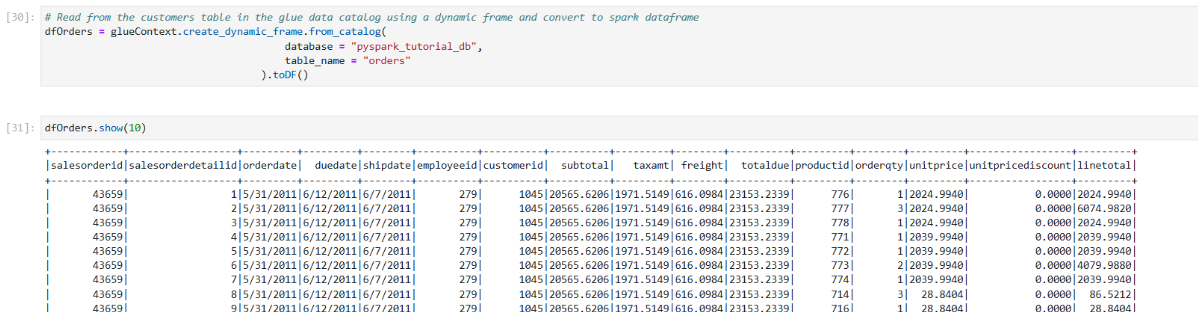
[28]:
```python
# Filter spark DataFrame for customers who have the last name Adams
sparkDf.filter(sparkDf["lastname"] == "Adams").show()
```

```
+----------+---------+--------+-------------+
|customerid|firstname|lastname|     fullname|
+----------+---------+--------+-------------+
|       305|    Carla|   Adams|  Carla Adams|
|       301|  Frances|   Adams|Frances Adams|
|       307|      Jay|   Adams|    Jay Adams|
+----------+---------+--------+-------------+
```

[29]:
```python
# Where clause spark DataFrame for customers who have the last name Adams
sparkDf.where("lastname =='Adams'").show()
```

```
+----------+---------+--------+-------------+
|customerid|firstname|lastname|     fullname|
+----------+---------+--------+-------------+
|       305|    Carla|   Adams|  Carla Adams|
|       301|  Frances|   Adams|Frances Adams|
|       307|      Jay|   Adams|    Jay Adams|
+----------+---------+--------+-------------+
```

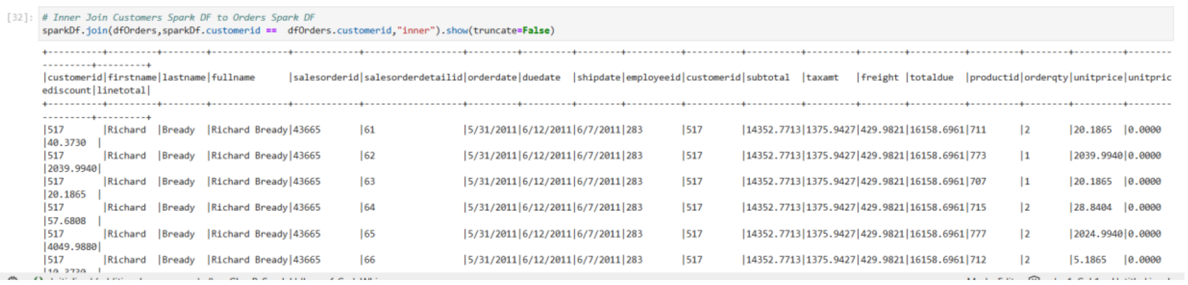
[ ]:



```
[30]: # Read from the customers table in the glue data catalog using a dynamic frame and convert to spark dataframe
      dfOrders = glueContext.create_dynamic_frame.from_catalog(
                                              database = "pyspark_tutorial_db",
                                              table_name = "orders"
                                          ).toDF()
```

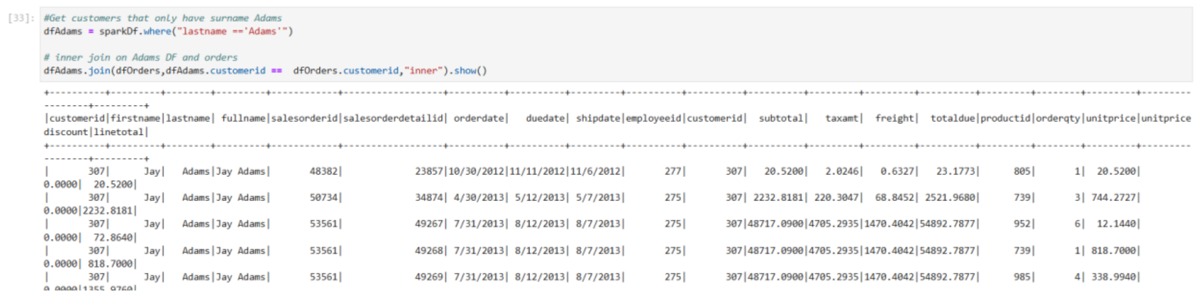
```
[31]: dfOrders.show(10)
```

```
+------------+------------------+---------+---------+--------+----------+----------+----------+---------+--------+----------+---------+--------+---------+----------------+---------+
|salesorderid|salesorderdetailid|orderdate|  duedate|shipdate|employeeid|customerid|  subtotal|   taxamt| freight|  totaldue|productid|orderqty|unitprice|unitpricediscount|linetotal|
+------------+------------------+---------+---------+--------+----------+----------+----------+---------+--------+----------+---------+--------+---------+----------------+---------+
|       43659|                 1|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      776|       1|2024.9940|          0.0000|2024.9940|
|       43659|                 2|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      777|       3|2024.9940|          0.0000|6074.9820|
|       43659|                 3|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      778|       1|2024.9940|          0.0000|2024.9940|
|       43659|                 4|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      771|       1|2039.9940|          0.0000|2039.9940|
|       43659|                 5|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      772|       1|2039.9940|          0.0000|2039.9940|
|       43659|                 6|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      773|       2|2039.9940|          0.0000|4079.9880|
|       43659|                 7|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      774|       1|2039.9940|          0.0000|2039.9940|
|       43659|                 8|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      714|       3|  28.8404|          0.0000|  86.5212|
|       43659|                 9|5/31/2011|6/12/2011|6/7/2011|       279|      1045|20565.6206|1971.5149|616.0984|23153.2339|      716|       1|  28.8404|          0.0000|  28.8404|
```



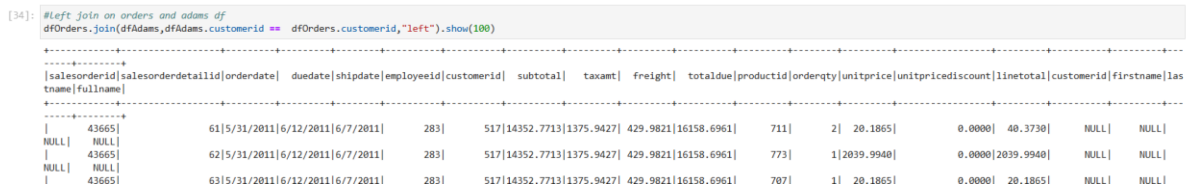
```
[32]: # Inner Join Customers Spark DF to Orders Spark DF
      sparkDf.join(dfOrders,sparkDf.customerid ==  dfOrders.customerid,"inner").show(truncate=False)
```

```
+----------+---------+--------+-------------+------------+------------------+---------+---------+--------+----------+----------+----------+---------+--------+----------+---------+--------+---------+-------
--------+---------+
|customerid|firstname|lastname|fullname     |salesorderid|salesorderdetailid|orderdate|duedate  |shipdate|employeeid|customerid|subtotal  |taxamt   |freight |totaldue  |productid|orderqty|unitprice|unitpric
ediscount|linetotal|
+----------+---------+--------+-------------+------------+------------------+---------+---------+--------+----------+----------+----------+---------+--------+----------+---------+--------+---------+-------
--------+---------+
|517       |Richard  |Bready  |Richard Bready|43665      |61                |5/31/2011|6/12/2011|6/7/2011|283       |517       |14352.7713|1375.9427|429.9821|16158.6961|711      |2       |20.1865  |0.0000
|40.3730  |
|517       |Richard  |Bready  |Richard Bready|43665      |62                |5/31/2011|6/12/2011|6/7/2011|283       |517       |14352.7713|1375.9427|429.9821|16158.6961|773      |1       |2039.9940|0.0000
|2039.9940|
|517       |Richard  |Bready  |Richard Bready|43665      |63                |5/31/2011|6/12/2011|6/7/2011|283       |517       |14352.7713|1375.9427|429.9821|16158.6961|707      |1       |20.1865  |0.0000
|20.1865  |
|517       |Richard  |Bready  |Richard Bready|43665      |64                |5/31/2011|6/12/2011|6/7/2011|283       |517       |14352.7713|1375.9427|429.9821|16158.6961|715      |2       |28.8404  |0.0000
|57.6808  |
|517       |Richard  |Bready  |Richard Bready|43665      |65                |5/31/2011|6/12/2011|6/7/2011|283       |517       |14352.7713|1375.9427|429.9821|16158.6961|777      |2       |2024.9940|0.0000
|4049.9880|
|517       |Richard  |Bready  |Richard Bready|43665      |66                |5/31/2011|6/12/2011|6/7/2011|283       |517       |14352.7713|1375.9427|429.9821|16158.6961|712      |2       |5.1865   |0.0000
|10.3730  |
```



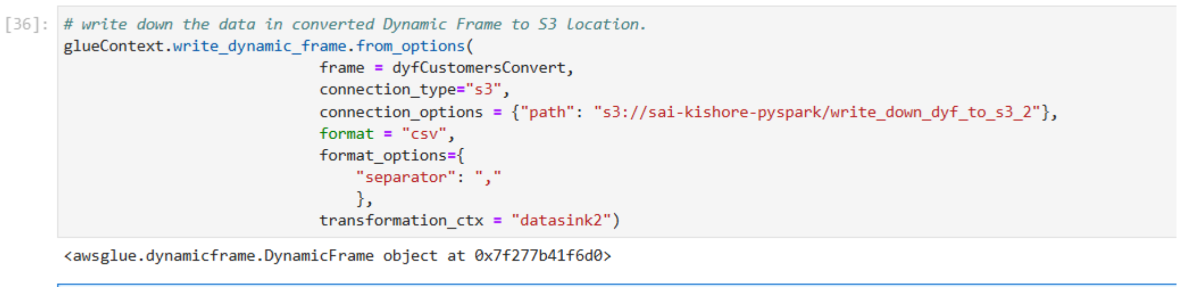
```
[33]: #Get customers that only have surname Adams
      dfAdams = sparkDf.where("lastname =='Adams'")

      # inner join on Adams DF and orders
      dfAdams.join(dfOrders,dfAdams.customerid ==  dfOrders.customerid,"inner").show()
```

```
+----------+---------+--------+----------+------------+------------------+----------+----------+---------+----------+----------+----------+---------+---------+----------+---------+--------+---------+--------
--------+---------+
|customerid|firstname|lastname|  fullname|salesorderid|salesorderdetailid| orderdate|   duedate| shipdate|employeeid|customerid|  subtotal|   taxamt|  freight|  totaldue|productid|orderqty|unitprice|unitprice
discount|linetotal|
+----------+---------+--------+----------+------------+------------------+----------+----------+---------+----------+----------+----------+---------+---------+----------+---------+--------+---------+--------
--------+---------+
|       307|      Jay|   Adams|Jay Adams|       48382|             23857|10/30/2012|11/11/2012|11/6/2012|       277|       307|   20.5200|   2.0246|   0.6327|   23.1773|      805|       1|  20.5200|
0.0000|  20.5200|
|       307|      Jay|   Adams|Jay Adams|       50734|             34874| 4/30/2013| 5/12/2013| 5/7/2013|       275|       307| 2232.8181| 220.3047|  68.8452| 2521.9680|      739|       3| 744.2727|
0.0000|2232.8181|
|       307|      Jay|   Adams|Jay Adams|       53561|             49267| 7/31/2013| 8/12/2013| 8/7/2013|       275|       307|48717.0900|4705.2935|1470.4042|54892.7877|      952|       6|  12.1440|
0.0000|  72.8640|
|       307|      Jay|   Adams|Jay Adams|       53561|             49268| 7/31/2013| 8/12/2013| 8/7/2013|       275|       307|48717.0900|4705.2935|1470.4042|54892.7877|      739|       1| 818.7000|
0.0000| 818.7000|
|       307|      Jay|   Adams|Jay Adams|       53561|             49269| 7/31/2013| 8/12/2013| 8/7/2013|       275|       307|48717.0900|4705.2935|1470.4042|54892.7877|      985|       4| 338.9940|
0.0000|1355.9760|
```

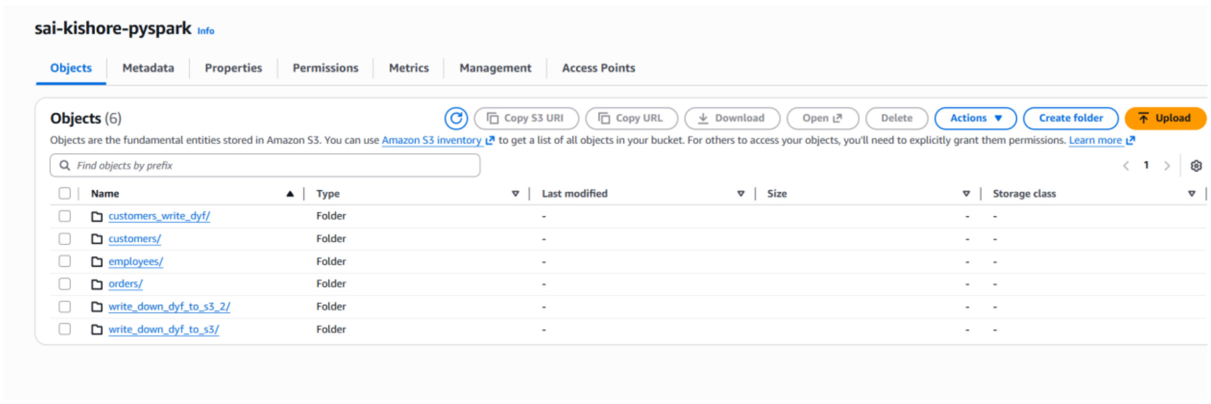


```
[34]: #Left join on orders and adams df
      dfOrders.join(dfAdams,dfAdams.customerid ==  dfOrders.customerid,"left").show(100)
```

```
+------------+------------------+---------+---------+--------+----------+----------+----------+---------+--------+----------+---------+--------+---------+----------------+---------+----------+---------+---
-----+--------+
|salesorderid|salesorderdetailid|orderdate|  duedate|shipdate|employeeid|customerid|  subtotal|   taxamt| freight|  totaldue|productid|orderqty|unitprice|unitpricediscount|linetotal|customerid|firstname|las
tname|fullname|
+------------+------------------+---------+---------+--------+----------+----------+----------+---------+--------+----------+---------+--------+---------+----------------+---------+----------+---------+---
-----+--------+
|       43665|                61|5/31/2011|6/12/2011|6/7/2011|       283|       517|14352.7713|1375.9427| 429.9821|16158.6961|      711|       2|  20.1865|          0.0000|  40.3730|      NULL|     NULL|
NULL|    NULL|
|       43665|                62|5/31/2011|6/12/2011|6/7/2011|       283|       517|14352.7713|1375.9427| 429.9821|16158.6961|      773|       1|2039.9940|          0.0000|2039.9940|      NULL|     NULL|
NULL|    NULL|
|       43665|                63|5/31/2011|6/12/2011|6/7/2011|       283|       517|14352.7713|1375.9427| 429.9821|16158.6961|      707|       1|  20.1865|          0.0000|  20.1865|      NULL|     NULL|
```



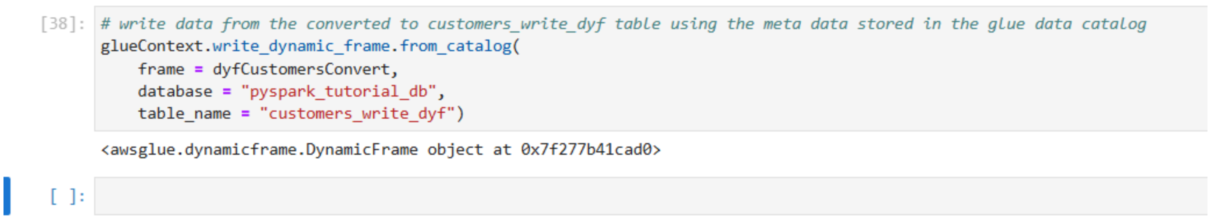
```
[36]: # write down the data in converted Dynamic Frame to S3 location.
      glueContext.write_dynamic_frame.from_options(
                                      frame = dyfCustomersConvert,
                                      connection_type="s3",
                                      connection_options = {"path": "s3://sai-kishore-pyspark/write_down_dyf_to_s3_2"},
                                      format = "csv",
                                      format_options={
                                          "separator": ","
                                          },
                                      transformation_ctx = "datasink2")

      <awsglue.dynamicframe.DynamicFrame object at 0x7f277b41f6d0>
```



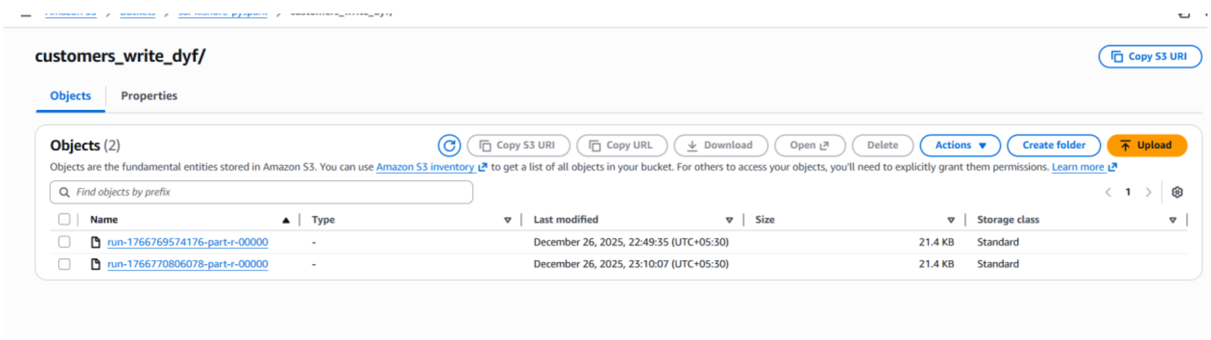
Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
customers_write_dyf/	Folder	-	-	-
customers/	Folder	-	-	-
employees/	Folder	-	-	-
orders/	Folder	-	-	-
write_down_dyf_to_s3_2/	Folder	-	-	-
write_down_dyf_to_s3/	Folder	-	-	-



```
[38]: # write data from the converted to customers_write_dyf table using the meta data stored in the glue data catalog
      glueContext.write_dynamic_frame.from_catalog(
          frame = dyfCustomersConvert,
          database = "pyspark_tutorial_db",
          table_name = "customers_write_dyf")

      <awsglue.dynamicframe.DynamicFrame object at 0x7f277b41cad0>

[ ]:
```



Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
run-1766769574176-part-r-00000	-	December 26, 2025, 22:49:35 (UTC+05:30)	21.4 KB	Standard
run-1766770806078-part-r-00000	-	December 26, 2025, 23:10:07 (UTC+05:30)	21.4 KB	Standard