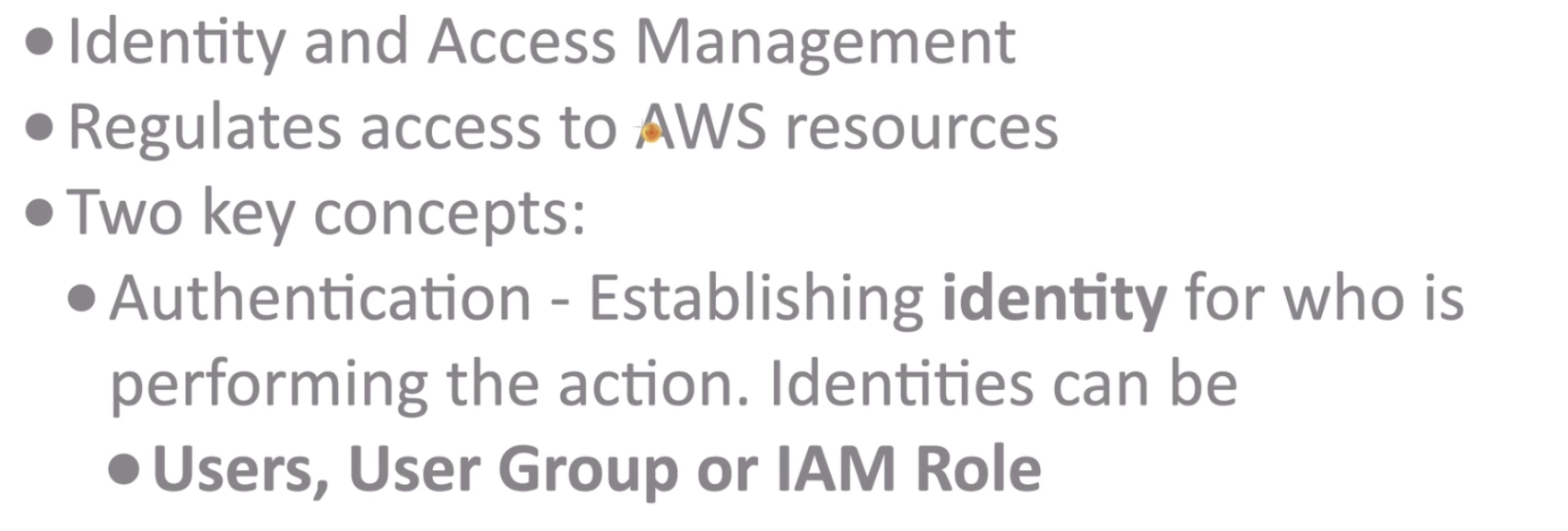
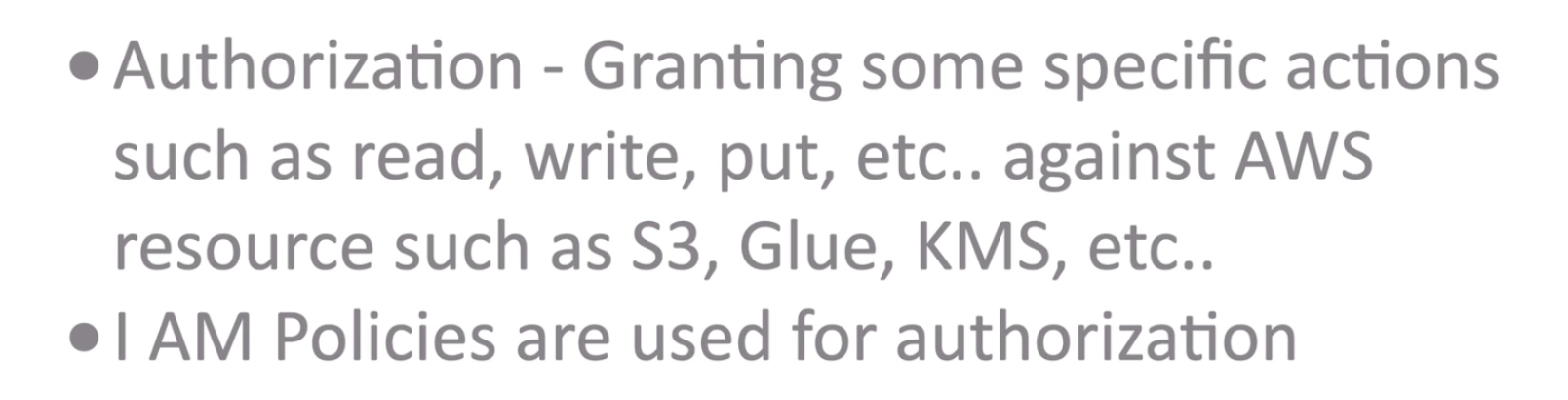
IAM:





Trusted Identity:

A **trusted entity** in AWS is someone or something you allow to use a specific IAM role. This "trusted entity" could be:

1. **An AWS service** like EC2 or Lambda.
2. **A user or role from another AWS account.**
3. **An external identity provider** like Google or your company's login system.

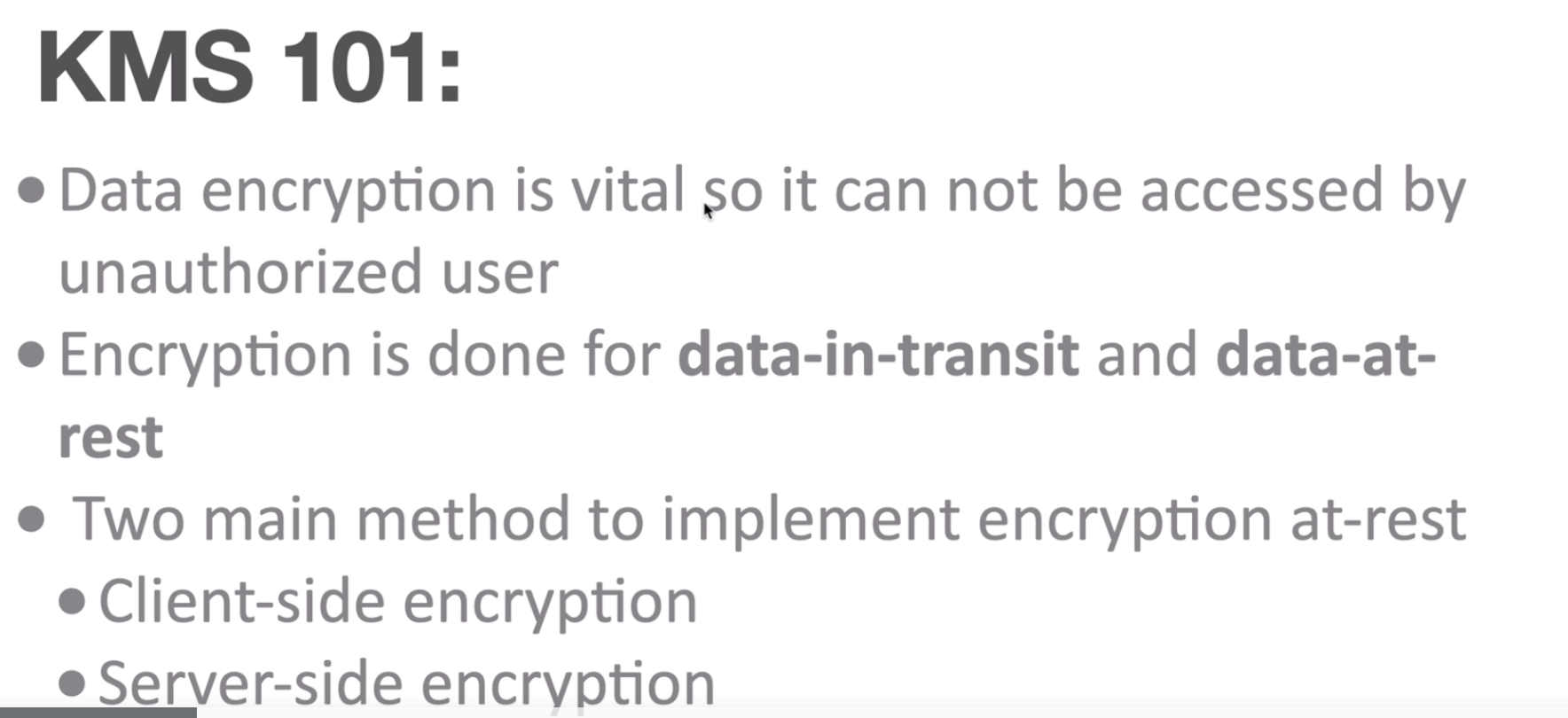
**Inline Policy:**

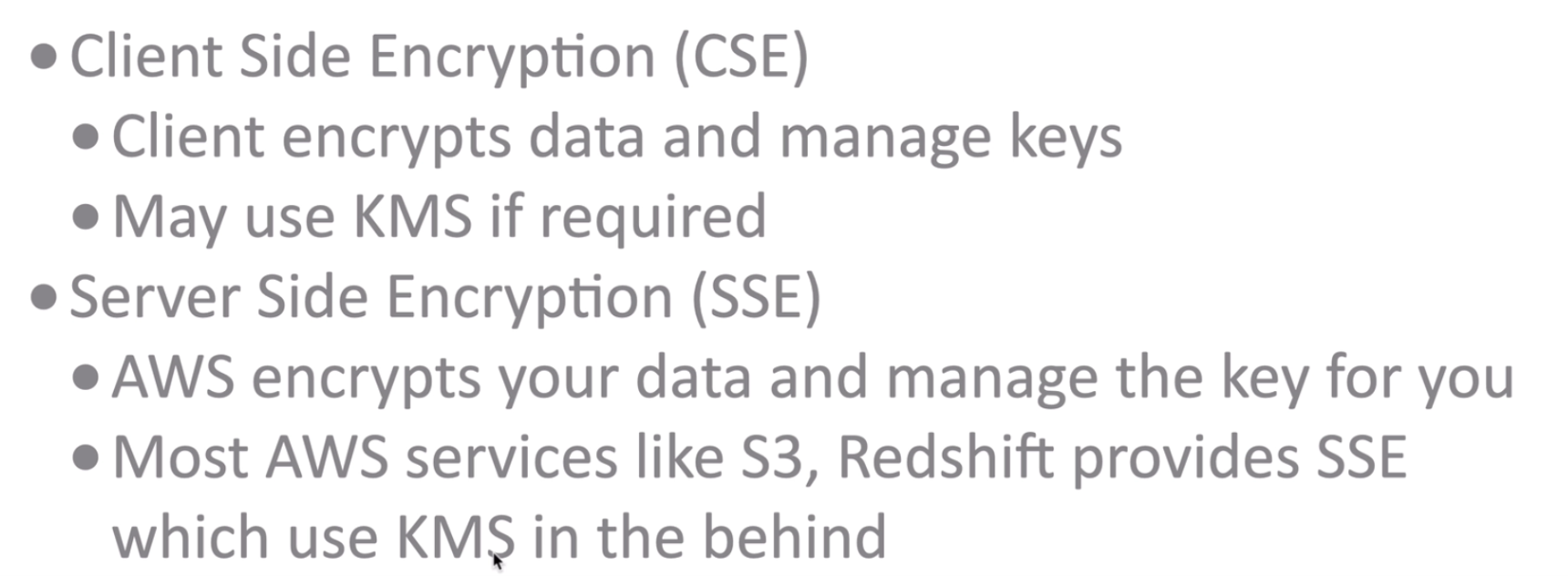
* It's like a **custom rule written directly inside one specific user or role**.
* Think of it as writing a personal to-do list for one person only.
* Example: You create a rule that says, "This role can only access S3 bucket my-bucket." This rule (policy) is attached directly to the role.

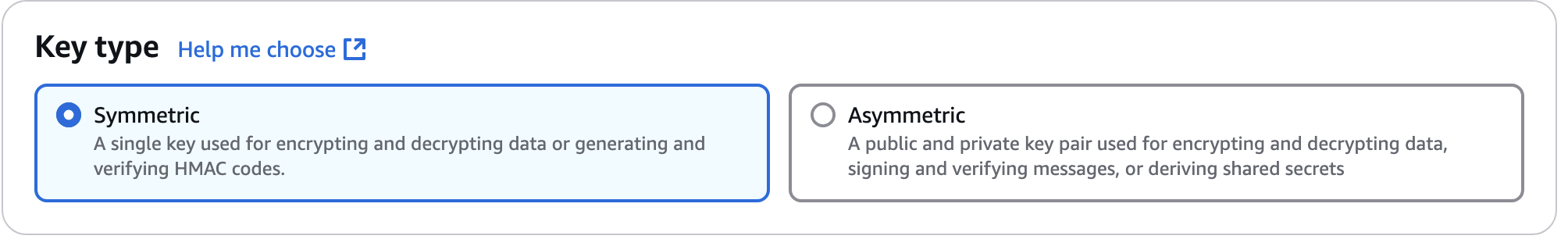
**Managed Policy:**

* It's a **shared policy** you can use for multiple users, groups, or roles.
* Think of it as a **template or reusable rule** that anyone can follow.
* AWS has **AWS-managed policies** (pre-built templates) and **Customer-managed policies** (templates you create).
* Example: You create a policy like, "Read-only access to all S3 buckets" and apply it to 10 users or roles. Instead of writing the rule 10 times, you use this single managed policy.

KMS:



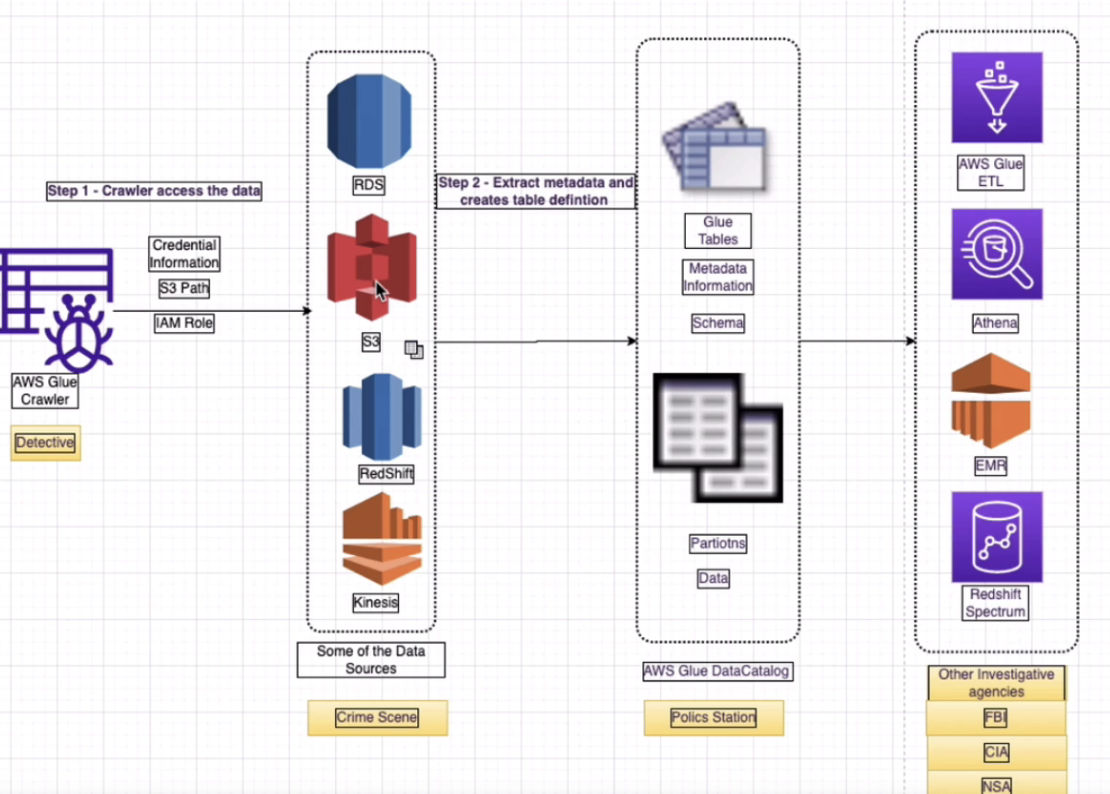
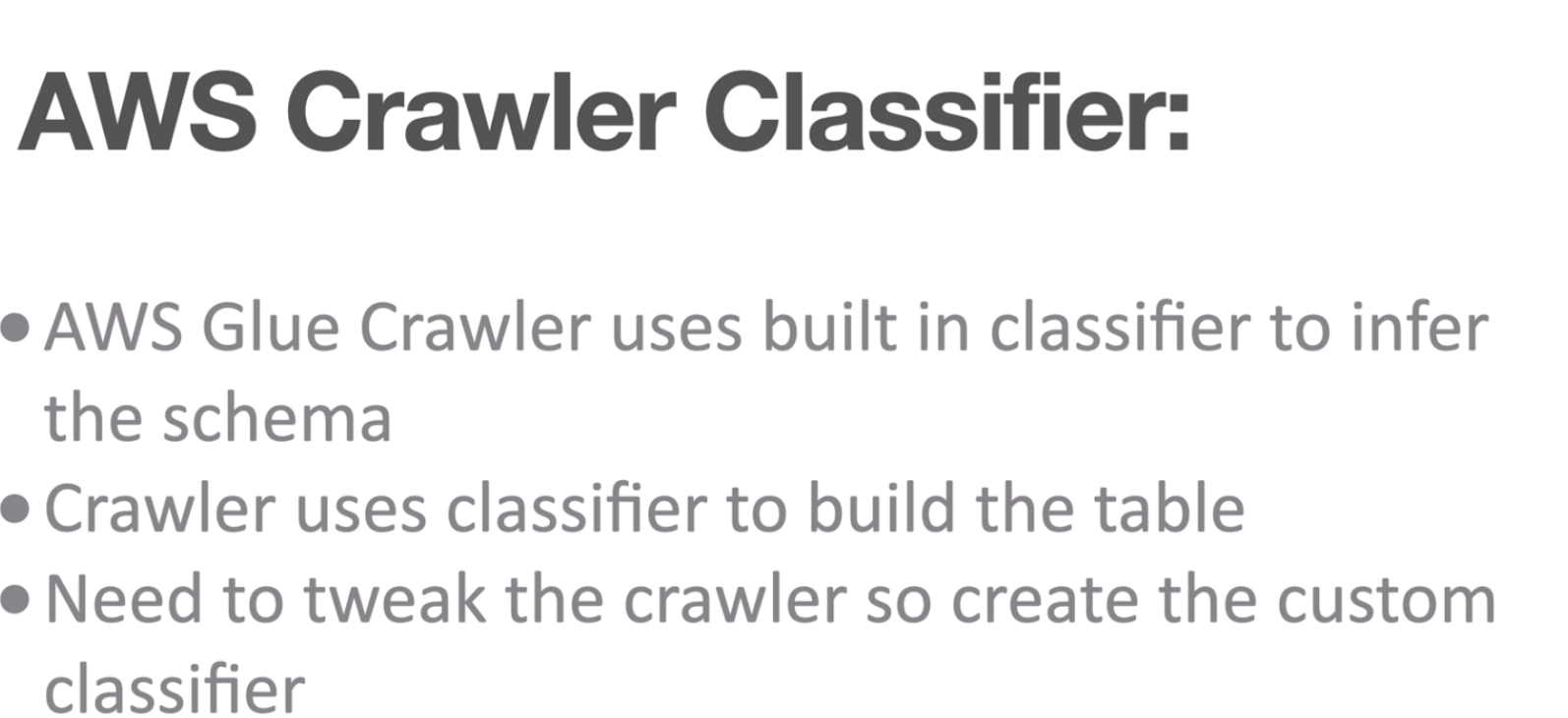
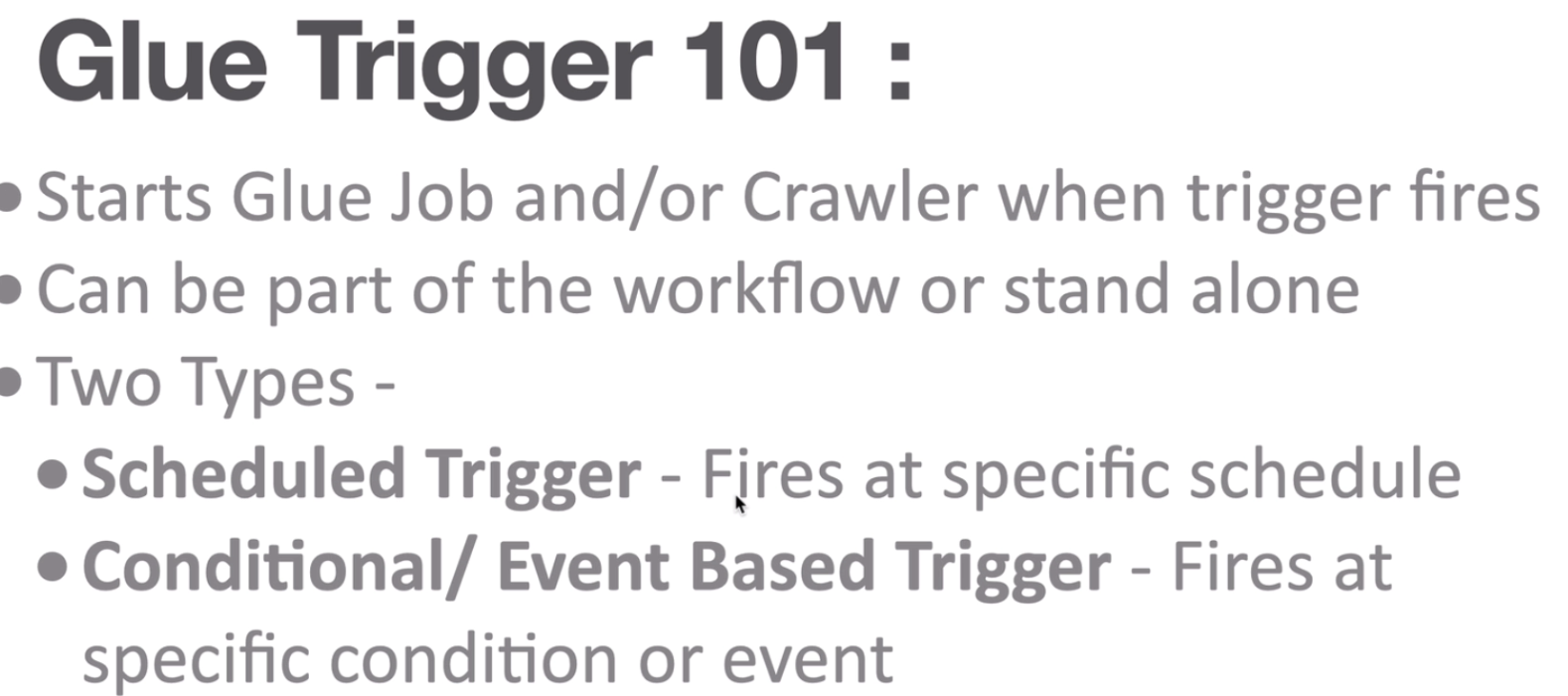
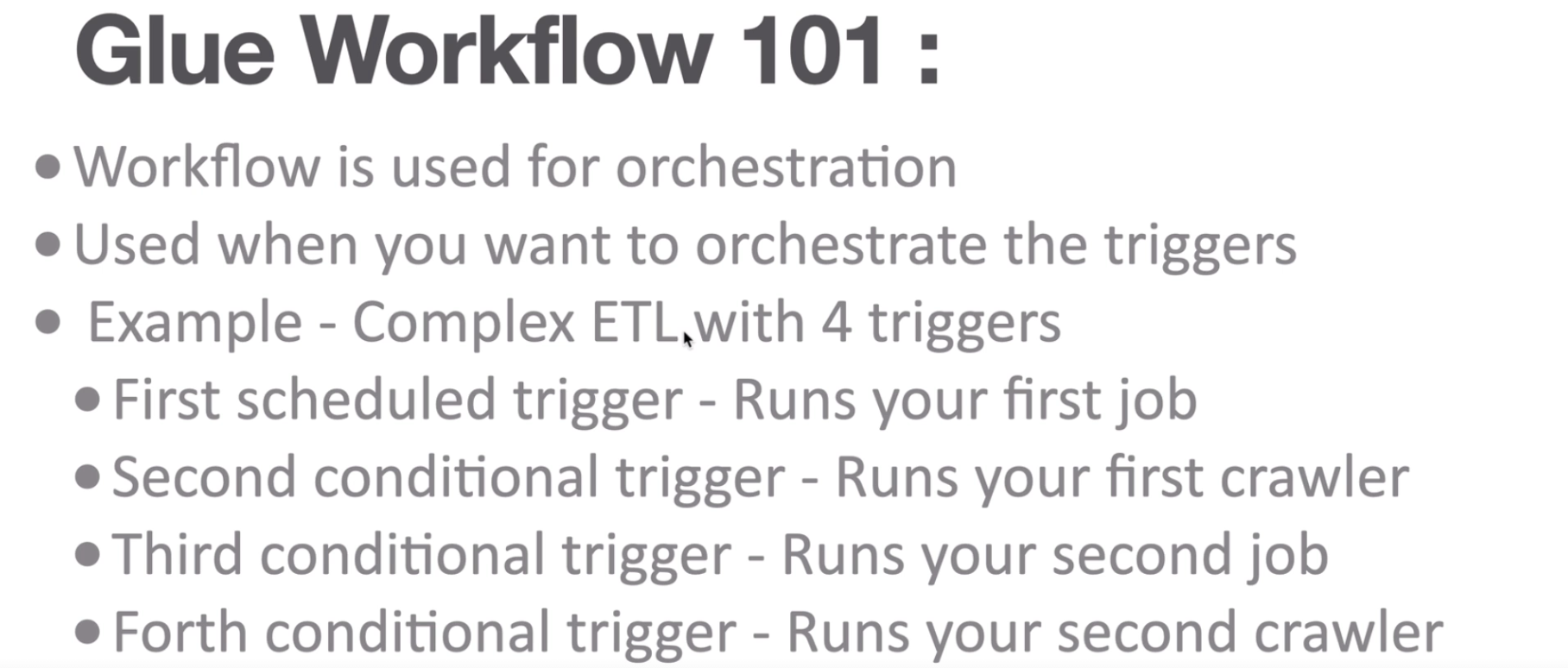
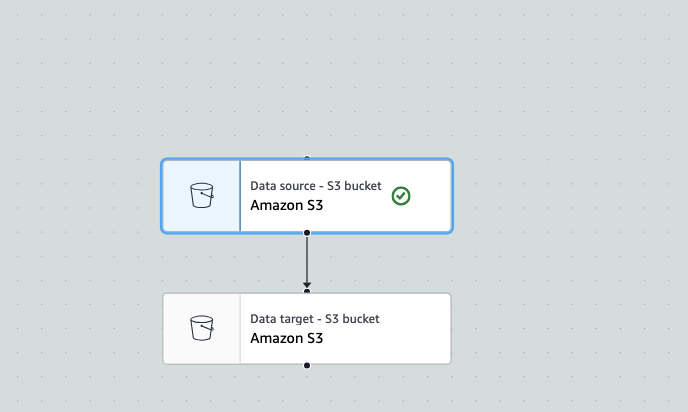


  
**Primary Components of AWS Glue**

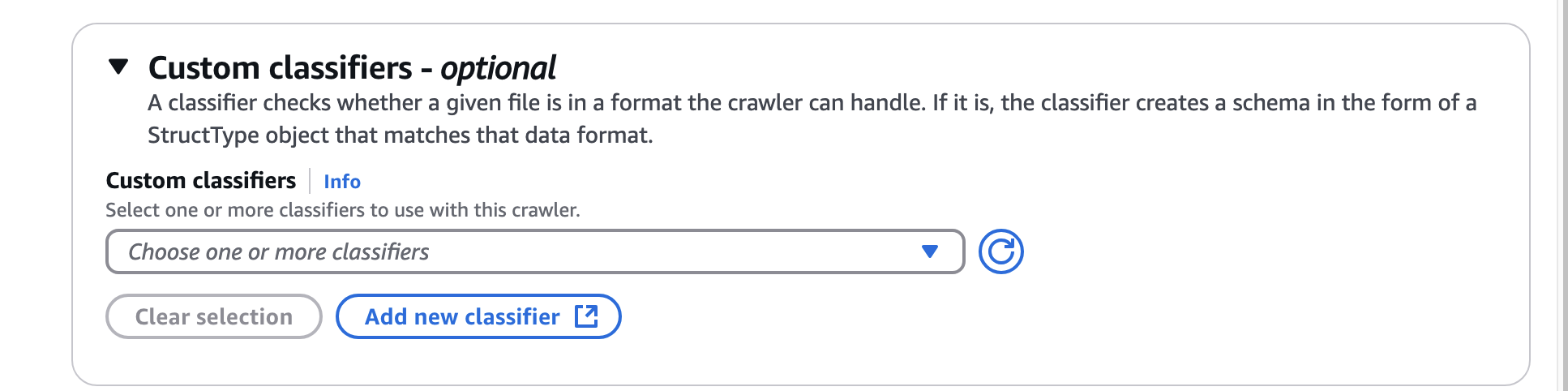
1. **AWS Glue Data Catalog**:
   * A metadata repository to store information about your datasets.
2. **AWS Glue Crawlers**:
   * Scans your data sources and automatically infers schema and other metadata.
3. **ETL Jobs**:
   * Customizable jobs that can extract, transform, and load data using Python or Scala.
4. **AWS Glue Studio**:
   * A visual interface to create, run, and monitor ETL jobs.
5. **Triggers**:
   * Used to schedule and automate ETL workflows.

We created glue job to convert CSV to parquet

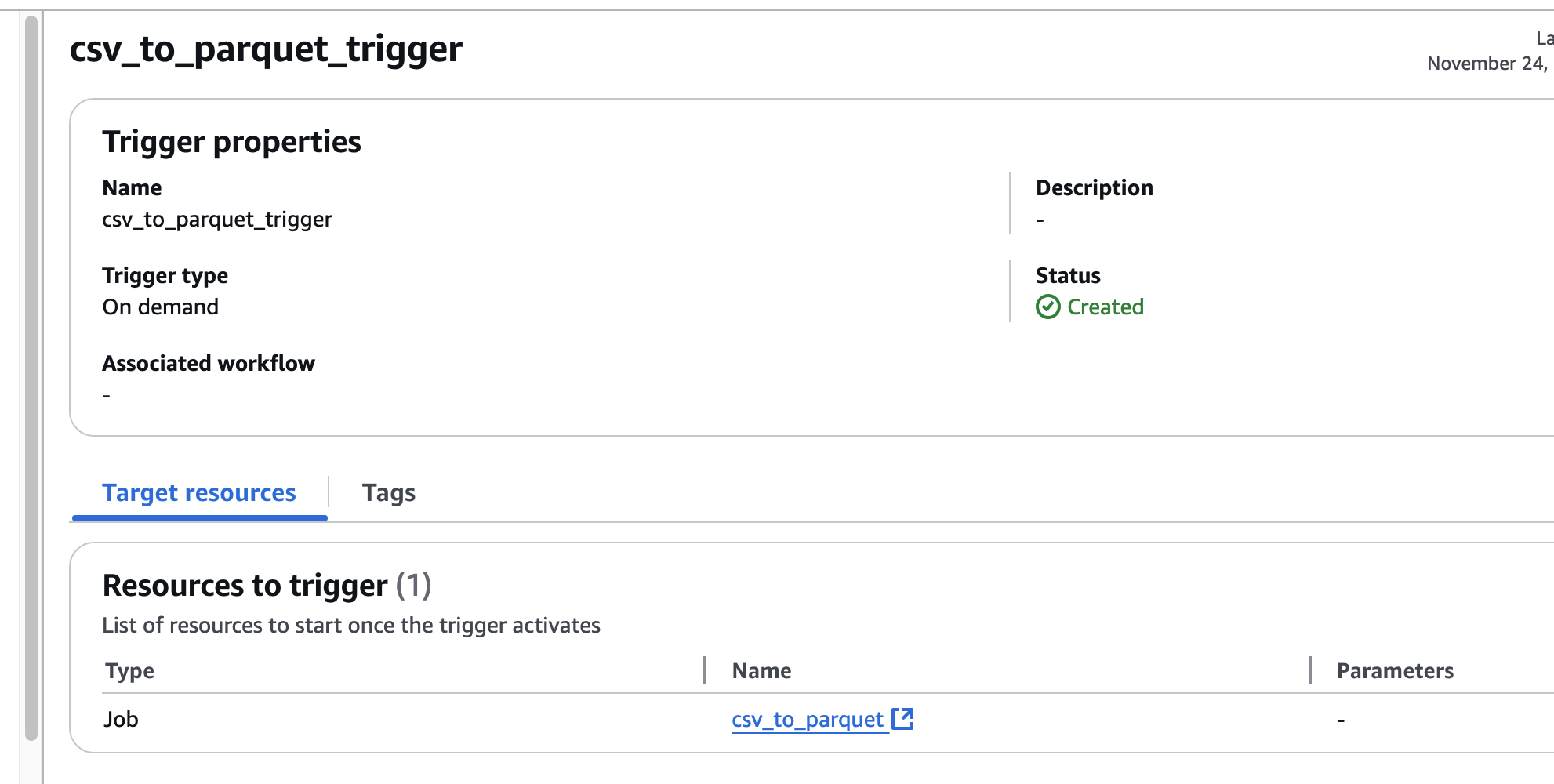
Glue Crawler:

* 
* 
* 
* 
* 

We can define custom classifier as well

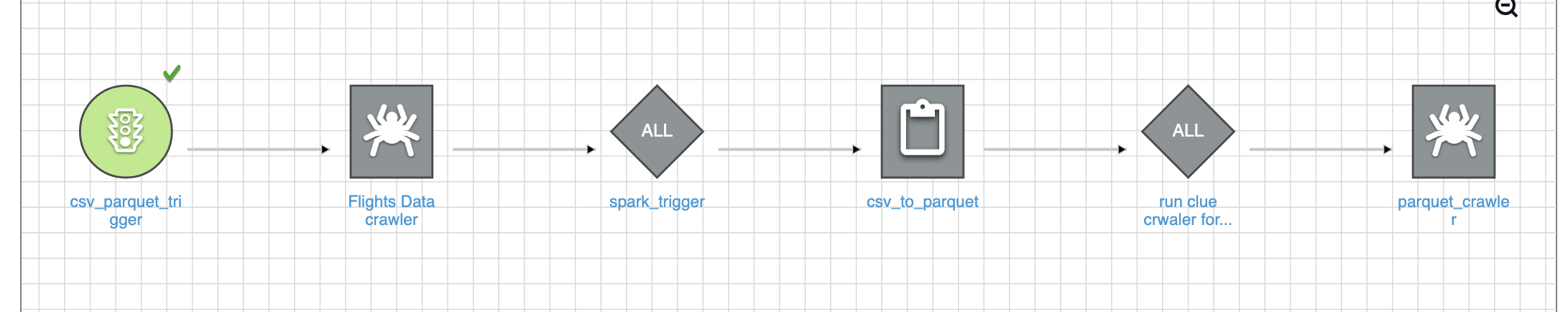


And later we added trigger for it



Workflow means orchestrated pipelines

Workflows in AWS Glue are used to **orchestrate and manage a sequence of ETL jobs, crawlers, and other processes** to build complex data pipelines. Workflows allow you to define dependencies, triggers, and the order of execution for different Glue components, ensuring seamless coordination between tasks.

* 

Setting Spark History Server in Glue

Mainly to troubleshoot during failure

**Search: launch spark history server glue**

**github repository for glue samples**

LOG\_DIR="s3://aws-glue-assets-816069129520-ap-south-1/sparkHistoryLogs/"

AWS\_ACCESS\_KEY\_ID="AKIA34AMCWUYJDEG4X6M"

AWS\_SECRET\_ACCESS\_KEY="WdEe9cv5MgggyqtoSoF9OlAEbNMtxJiwy2osoUul"

docker run -itd -v ~/.aws:/root/.aws -e AWS\_PROFILE=$PROFILE\_NAME -e SPARK\_HISTORY\_OPTS="$SPARK\_HISTORY\_OPTS -Dspark.history.fs.logDirectory=$LOG\_DIR -Dspark.hadoop.fs.s3a.aws.credentials.provider=com.amazonaws.auth.DefaultAWSCredentialsProviderChain" -p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class org.apache.spark.deploy.history.HistoryServer"

**docker start** sparkui

**Once the container is built make sure to add required environment variables**[**as mentioned in the instructions**](https://www.google.com/url?q=https://github.com/aws-samples/aws-glue-samples/tree/master/utilities/Spark_UI%23start-the-spark-history-server&sa=D&source=editors&ust=1629529826545000&usg=AOvVaw2nzAwX8XLmnVqTZpwIrdBs)**before starting the docker container with Spark UI Server.**

* **Set LOG\_DIR by replacing s3a://path\_to\_eventlog with your event log directory**
* **Set AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY with your valid AWS credentials.**

1. **LOG\_DIR="s3a://path\_to\_eventlog/"**
2. **AWS\_ACCESS\_KEY\_ID="AKIAxxxxxxxxxxxx"**
3. **AWS\_SECRET\_ACCESS\_KEY="yyyyyyyyyyyyyyy"**
4. **docker run -itd \**
5. **-e SPARK\_HISTORY\_OPTS="$SPARK\_HISTORY\_OPTS \**
6. **-Dspark.history.fs.logDirectory=$LOG\_DIR \**
7. **-Dspark.hadoop.fs.s3a.access.key=$AWS\_ACCESS\_KEY\_ID \**
8. **-Dspark.hadoop.fs.s3a.secret.key=$AWS\_SECRET\_ACCESS\_KEY" \**
9. **-p 18080:18080 glue/sparkui:latest \**
10. **"/opt/spark/bin/spark-class org.apache.spark.deploy.history.HistoryServer"**

**Prerequisites for Glue Catalog Tables**

**Let us go through the prerequisites for Glue Crawler to crawl the metadata and create Glue Catalog Tables..**

* **We need to have data in s3 or other supported data stores to use Glue Crawler to crawl the metadata to create tables.**
* **It is highly recommended to have structure to the data. If we are reading the data from text files, then the data should have a header.**
* **Glue should have appropriate permissions via IAM Role to access the s3 buckets.**

**Let us download GitHub activity data so that we can upload to s3 to learn data engineering using AWS Analytics Services.**

* **You can use wget to download the data into a local machine.**
* **We will download 3 day’s worth of data. Here are the dates.**
  + **2020-01-13**
  + **2020-01-14**
  + **2020-01-15**
* **We can download the files using instructions provided as part of**[**gharchive.org**](https://www.google.com/url?q=https://www.gharchive.org&sa=D&source=editors&ust=1629529826548000&usg=AOvVaw2bRNpv3PNFcTv4-ZhEY5t8)**. I will be downloading these files using terminal on my Mac under ~/Downloads/ghactivity**

1. **mkdir ~/Downloads/ghactivity**
2. **cd ~/Downloads/ghactivity**
3. **wget https://data.gharchive.org/2021-01-13-{0..23}.json.gz**
4. **wget https://data.gharchive.org/2021-01-14-{0..23}.json.gz**
5. **wget https://data.gharchive.org/2021-01-15-{0..23}.json.gz**

* **You can then use the s3 web console or upload using AWS CLI to upload the data.**

**Let us upload GitHub activity data into s3. You can use s3 web console to create a bucket and then copy the data into the bucket. Based up on the bandwidth, this action will take a considerable amount of time.**

* **Make sure to have a bucket by name itv-github**
* **Make sure to create folder by name landing/ghactivity/**
* **Upload files into the folder using AWS s3 Web Console**

**You can also use the below commands to upload the files using CLI. You need to have AWS CLI installed and configured against your AWS account with credentials which have write permissions to s3 bucket. I am using credentials which have write permissions on s3 bucket itv-github.**

**cd ~/Downloads/ghactivity**

**# Single thread**

**aws s3 cp . s3://itv-github/landing/ghactivity/ \**

**--recursive \**

**--profile itvgithub**

**# 3 threads in parallel using nohup**

**nohup aws s3 cp . s3://itv-github/landing/ghactivity/ \**

**--exclude "\*" \**

**--include "2021-01-13\*" \**

**--recursive &**

**nohup aws s3 cp . s3://itv-github/landing/ghactivity/ \**

**--exclude "\*" \**

**--include "2021-01-14\*" \**

**--recursive &**

**nohup aws s3 cp . s3://itv-github/landing/ghactivity/ \**

**--exclude "\*" \**

**--include "2021-01-15\*" \**

**--recursive &**

We can manage Glue Catalog using AWS Glue CLI. Let us understand how to trigger the crawler and also how to validate whether the tables are created or not using AWS Glue CLI commands.

* We need to make sure that the user has required permissions to manage Glue Catalog via CLI.
* We can assign Glue Service Role to the user whose credentials are configured as part of AWS CLI Profile.
* Here are the commands to get the details about a crawler.

1. aws glue list-crawlers --profile itvgithub
2. aws glue get-crawler --name "Retail Crawler" --profile itvgithub
3. aws glue start-crawler --name "Retail Crawler" --profile itvgithub

* Here are the commands to list the tables.

1. aws glue get-databases --profile itvgithub
2. aws glue get-tables --database-name retail\_db --profile itvgithub
3. aws glue get-table \
4. --database-name retail\_db \
5. --name orders \
6. --profile itvgithub

We can manage Glue Catalog using Python boto3. Let us understand how to trigger the crawler and also how to validate whether the tables are created or not using AWS boto3 code.

* We need to make sure that the user has required permissions to manage Glue Catalog via CLI.
* We can assign Glue Service Role to the user whose credentials are configured as part of AWS CLI Profile.
* Here is the code to get the details about crawler.

1. import boto3
2. import os
3. os.environ.setdefault('AWS\_PROFILE', 'itvgithub')
4. os.environ.setdefault('AWS\_DEFAULT\_REGION', 'us-east-1')
6. glue\_client = boto3.client('glue')
7. crawler = glue\_client.list\_crawlers()['CrawlerNames'][3]
9. glue\_client.get\_crawler(
10. Name=crawler
11. )
13. glue\_client.start\_crawler(
14. Name=crawler
15. )
17. glue\_client.get\_crawler(
18. Name=crawler
19. )['Crawler']['State']

* Here is the code to get the details about databases.

1. import boto3
2. import os
3. os.environ.setdefault('AWS\_PROFILE', 'itvgithub')
4. os.environ.setdefault('AWS\_DEFAULT\_REGION', 'us-east-1')
6. glue\_client = boto3.client('glue')
8. for db in glue\_client.get\_databases()['DatabaseList']:
9. print(db['Name'])

* Here is the code to get the details about all the tables as well as a single table.

1. import boto3
2. import os
3. os.environ.setdefault('AWS\_PROFILE', 'itvgithub')
4. os.environ.setdefault('AWS\_DEFAULT\_REGION', 'us-east-1')
6. glue\_client = boto3.client('glue')
8. for table in glue\_client.get\_tables(
9. DatabaseName='retail\_db'
10. )['TableList']:
11. print(table['Name'])
13. glue\_client.get\_table(
14. DatabaseName='retail\_db',
15. Name='orders'
16. )

To convert an AWS Glue **DynamicFrame** into a **Spark DataFrame**, you can use the toDF() method provided by the Glue library. Below is a step-by-step guide:

**Steps to Convert Glue DynamicFrame to Spark DataFrame**

1. **Import Necessary Libraries**: Ensure you have imported the Glue and Spark libraries:

python

Copy code

from awsglue.context import GlueContext

from pyspark.context import SparkContext

1. **Initialize Glue and Spark Contexts**:

python

Copy code

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

1. **Create a DynamicFrame**: Assume you have created or loaded a DynamicFrame using Glue's APIs:

python

Copy code

dynamic\_frame = glueContext.create\_dynamic\_frame.from\_catalog(

database="your\_database",

table\_name="your\_table"

)

1. **Convert to Spark DataFrame**: **Use the toDF() method to convert the DynamicFrame to a Spark DataFrame:**

python

**Copy code**

**spark\_df = dynamic\_frame.toDF()**

dynamic\_frame = DynamicFrame.fromDF(df, glueContext, "dynamic\_frame")

1. **Perform Operations on Spark DataFrame**: Once converted, you can use Spark DataFrame operations on spark\_df:

python

Copy code

spark\_df.show()

spark\_df.printSchema()

**To add additional pip modules to Glue job we need zip all the folders and files inside site package and create a zip folder and upload to s3 and add it as part of library path**

A white line with a black stripe

Description automatically generated with medium confidence

To add additional jars to pyspark

A screenshot of a computer

Description automatically generated

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

from awsglue.transforms import ApplyMapping, ResolveChoice, DropNullFields

from awsglue.context import GlueContext

from pyspark.context import SparkContext

from awsglue.dynamicframe import DynamicFrame

from pyspark.sql.functions import year, month, dayofmonth, to\_date,to\_timestamp

from pyspark.sql.functions import \*

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Script generated for node Amazon S3

AmazonS3\_node1732428595301 = glueContext.create\_dynamic\_frame.from\_options(format\_options={"quoteChar": "\"", "withHeader": True, "separator": ",", "optimizePerformance": False}, connection\_type="s3", format="csv", connection\_options={"paths": ["s3://atul.data/customers/"], "recurse": True}, transformation\_ctx="AmazonS3\_node1732428595301")

df = AmazonS3\_node1732428595301.toDF()

df = df.withColumn("ModifiedDate", col("ModifiedDate").substr(1, 10))

df = df.withColumn("ModifiedDate", to\_date(df["ModifiedDate"], "yyyy-MM-dd"))

df=df.withColumn("year", year(df["ModifiedDate"])).withColumn("month", month(df["ModifiedDate"])).withColumn("day", dayofmonth(df["ModifiedDate"]))

# Convert back to DynamicFrame

dynamic\_frame = DynamicFrame.fromDF(df, glueContext, "dynamic\_frame")

# Write data back to S3 with partitioning

AmazonS3\_node1732428678984 = glueContext.write\_dynamic\_frame.from\_options(

frame=dynamic\_frame,

connection\_type="s3",

format="glueparquet",

connection\_options={

"path": "s3://atul.data/parquet\_landing/",

"partitionKeys": ["year", "month", "day"]

},

format\_options={"compression": "snappy"},

transformation\_ctx="AmazonS3\_node1732428678984"

)

job.commit()

**Glue Bookmark:**

**Key Concepts of AWS Glue Bookmarks**

1. **Purpose:  
   Bookmarks enable an ETL job to remember previously processed data from a source (e.g., S3 bucket, database) and only process new or changed data in subsequent runs.**
2. **State Management:  
   Bookmarks maintain a checkpoint or "state" of the data processed in the last successful run. This state is stored in the AWS Glue Data Catalog or job metadata.**
3. **Incremental Processing:  
   With bookmarks, only new data (e.g., new files in S3 or newly added rows in a database table) is processed, reducing duplication and improving efficiency.**
4. **Applicability:  
   Bookmarks are typically used with data sources that support incremental data discovery, such as:**
   * **Amazon S3 (e.g., files with a timestamp or incremental naming pattern)**
   * **Amazon RDS/Athena/Redshift**
   * **JDBC data sources**

Let us get an overview of AWS CLI related to Glue.

* As part of Glue we have several components such as crawler, catalog, jobs, triggers etc.
* We should be able to manage all the Glue components using the below main command.

aws glue

* We can see the list of supported commands for each sub component using help.

aws glue help

* You can explore help on specific commands by using help on top of the sub component’s action.

aws glue list-jobs help

* For each command you can pass the specific set of credentials using profile.

1. aws glue list-jobs
2. aws glue list-jobs --profile itvgithub --region us-east-1

Let us understand more about AWS CLI to get information about job, job runs as well as current bookmark.

* Listing jobs

1. aws glue list-jobs \
2. --profile itvgithub \
3. --region us-east-1

* Get job details

1. aws glue \
2. get-job \
3. --job-name github\_json\_to\_parquet \
4. --profile itvgithub \
5. --region us-east-1

* Get job run ids. The latest one will be typically at top.

1. aws glue \
2. get-job-runs \
3. --job-name github\_json\_to\_parquet \
4. --profile itvgithub \
5. --region us-east-1

* Get job run details to verify if job is successful or not.

1. aws glue \
2. get-job-run \
3. --job-name github\_json\_to\_parquet \
4. --run-id jr\_a350197ce2d5cc3168160813e28bef293e0edd4fc2fe8f458191885d0bb32f96 \
5. --profile itvgithub \
6. --region us-east-1

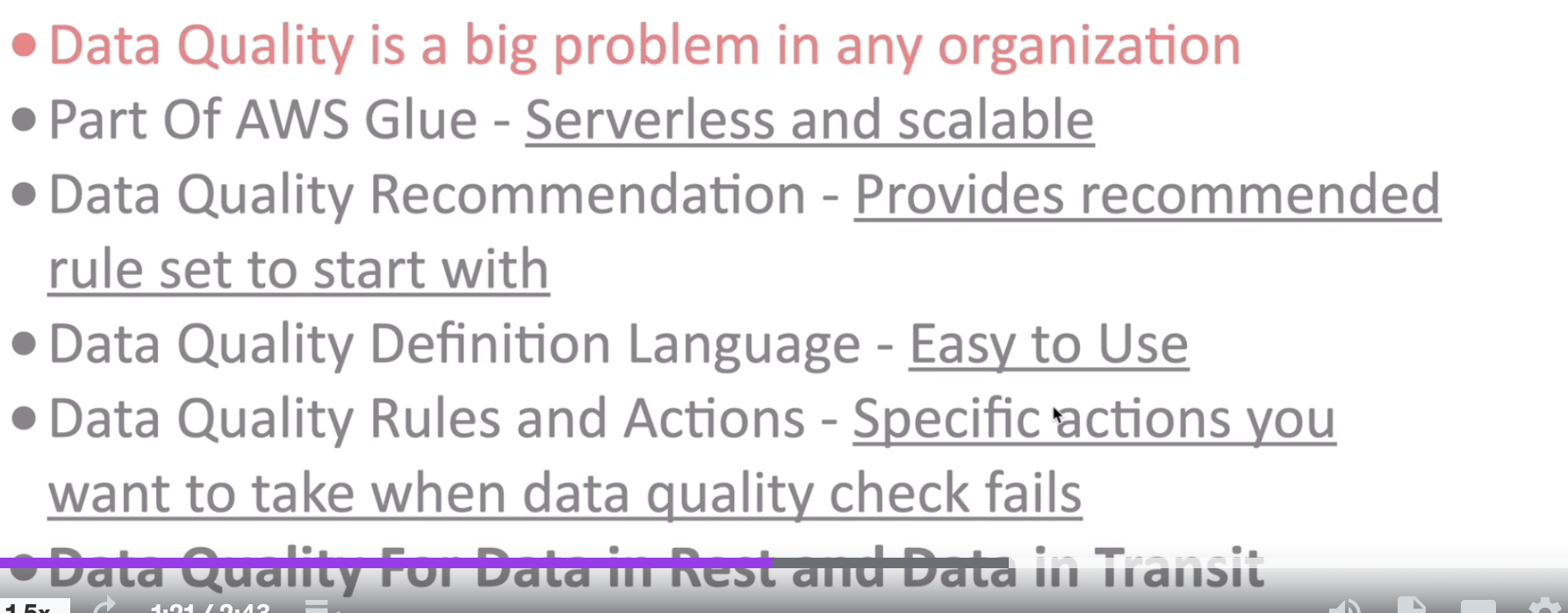
* Get job bookmark details. This information will be used to read the data in incremental fashion in subsequent runs. Make sure to keep track of it to compare with subsequent runs.

1. aws glue \
2. get-job-bookmark \
3. --job-name github\_json\_to\_parquet \
4. --profile itvgithub \
5. --region us-east-1

* We can use reset-job-bookmark to reset remove the bookmark. It comes handy to start the jobs from the beginning. We can also reset to a particular run using run id.

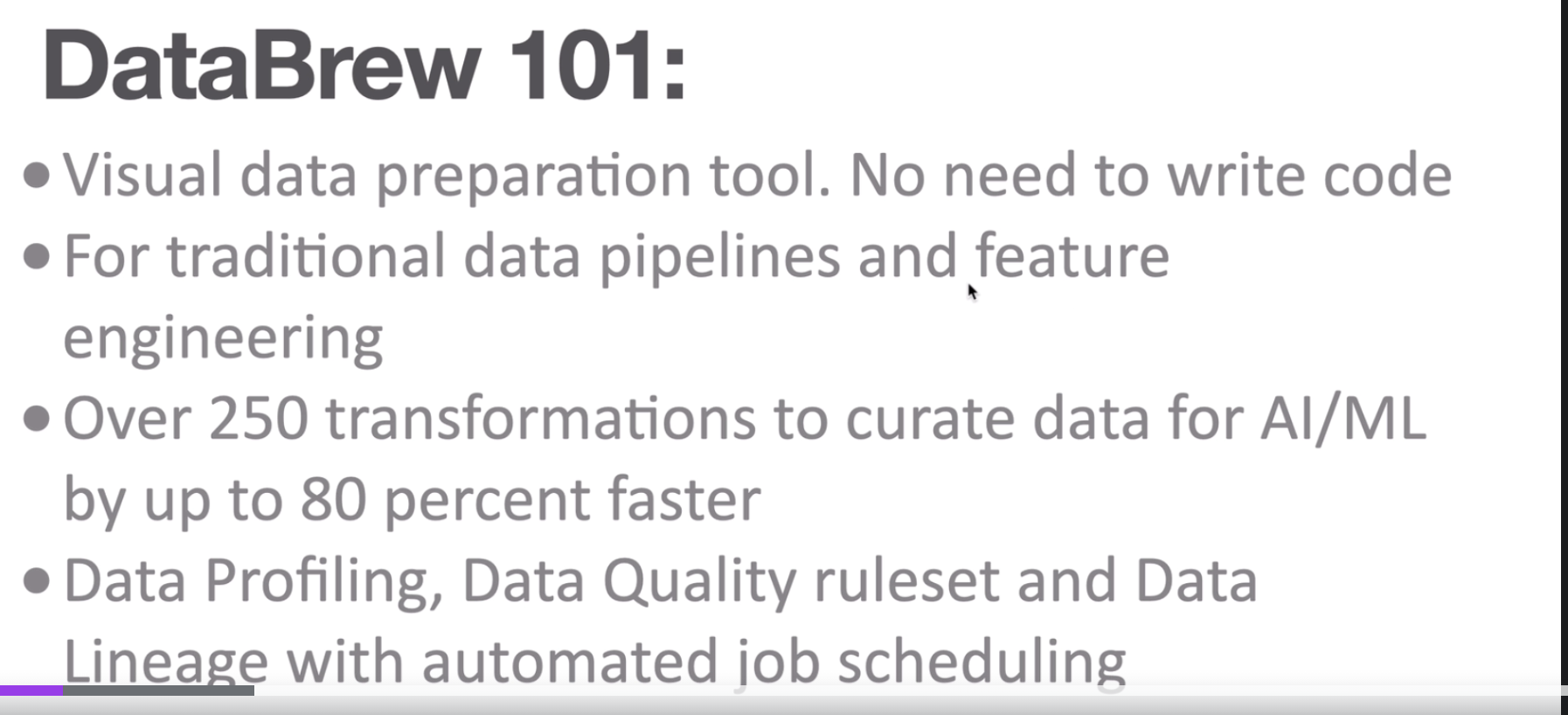
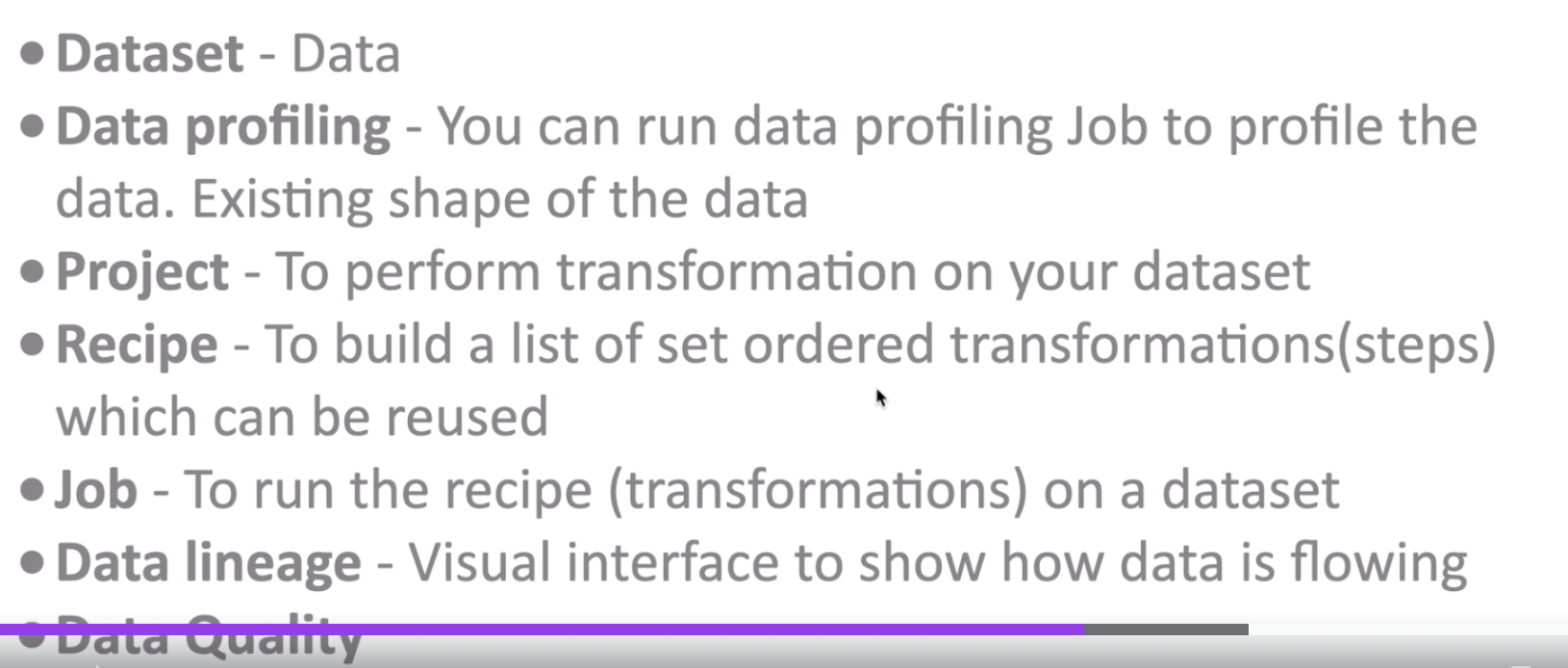
1. aws glue reset-job-bookmark \
2. --job-name github\_json\_to\_parquet \
3. --profile itvgithub \
4. --region us-east-1

**Data Quality:**

****

**We need to define Rule set using DQDL (Data Quality Definition Language)**

**DataBrew:**

* 
* 
* 

**Cron Expression in Glue:**

**A screenshot of a computer

Description automatically generated**

**We can’t have both day of the month and day of week**