



# **Simplify Data Integration and Preparation using AWS Glue**

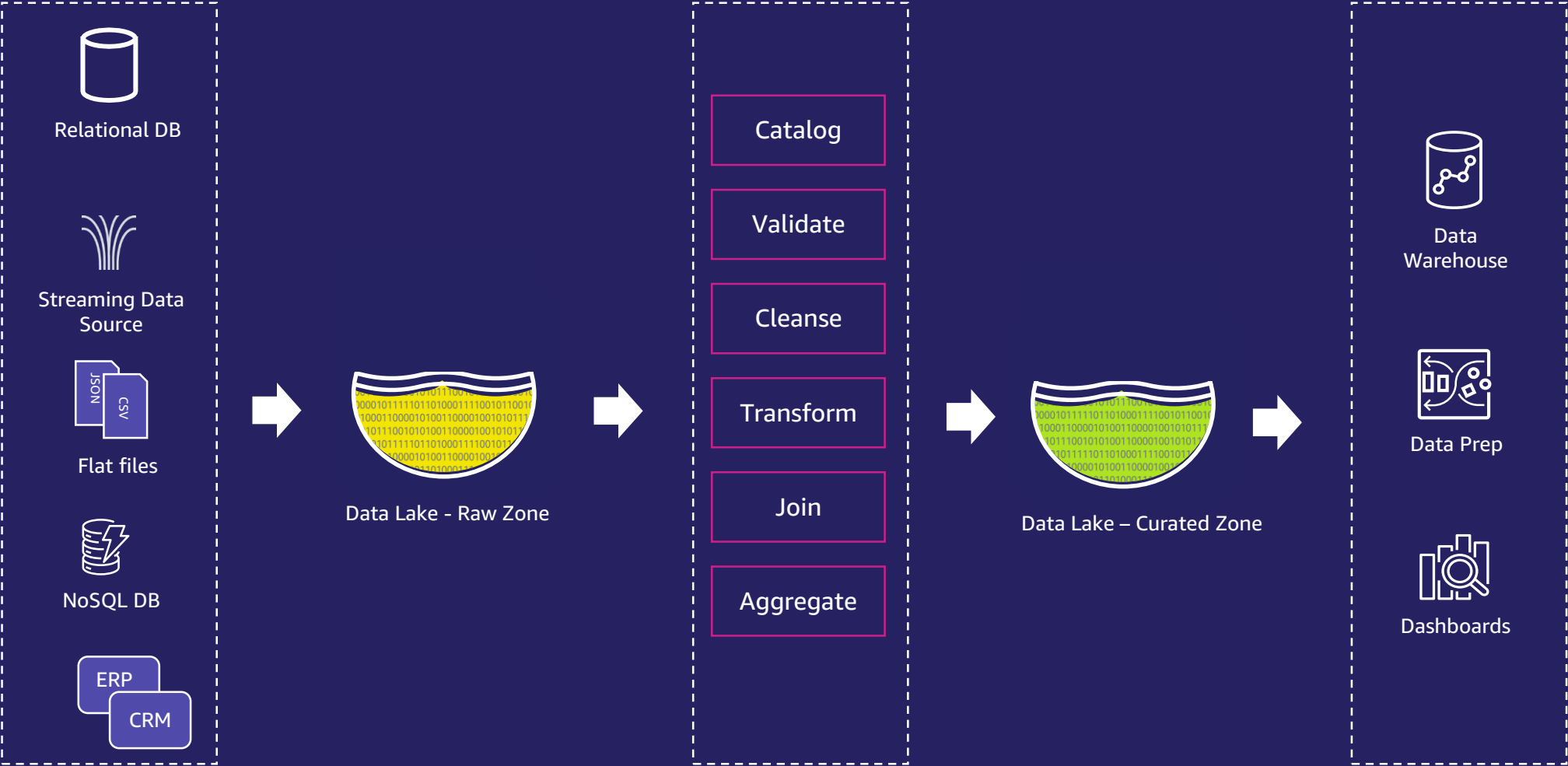
**Shiv Narayanan**, Product Manager – AWS Glue

**Deenbandhu Prasad**, Sr. Analytics Specialist SA

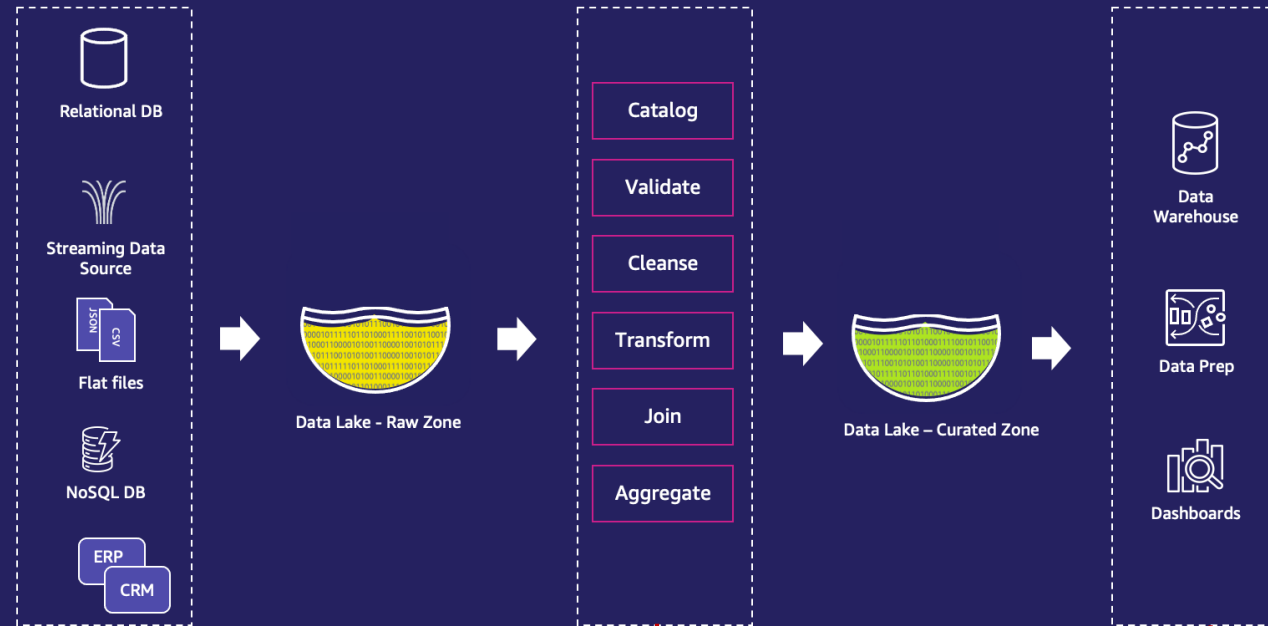
# What are we going to cover today

1. A use case that you can relate to
2. Challenges to implement this use case
3. AWS Glue and how it addresses these challenges

# A simple yet common data integration architecture pattern



# Simple is not really simple in data integration...



- 1 Expensive connectors
- 2 Non-unified tools
- 3 Challenging to handle semi structured data

- 4 Legacy methods to cleanse data
- 5 Locked in proprietary tools
- 6 Challenging to scale

- 7 One tool fits all approach
- 8 Expensive licenses for self service
- 9 Non scalable infrastructure

# AWS Glue Serverless Data Integration for complex workloads



## Serverless

No infrastructure to maintain. Allocate needed compute power and run jobs



## Data Integration for every user

Development environments catered to different skillsets - visual ETL development for Data Engineers, notebook styled development for Data Scientists, and no code development for Data Analysts



## Cost-effective

All-in-one pricing model is 55% cheaper than other cloud data integration solutions



## Handles complex workloads

Connect to 65+ data sources, process petabytes of data in real-time, includes batch and event driven modes



## No lock-in

Develop data integration pipelines in open source SparkSQL, PySpark, Python, Scala



# BMW powers self-service Cloud HUB data platform with AWS Glue



## PROBLEM

- On-premises Hadoop, Scoop used to perform data ingestion and processing
- Processes ran on hard-to-scale platform, leading to attrition of internal customers moving back to original data platform
- Reliance on DevOps to constantly provision clusters and tuning was time-consuming
- Lack of self-service options for data ingestion led to longer data ingestion cycles

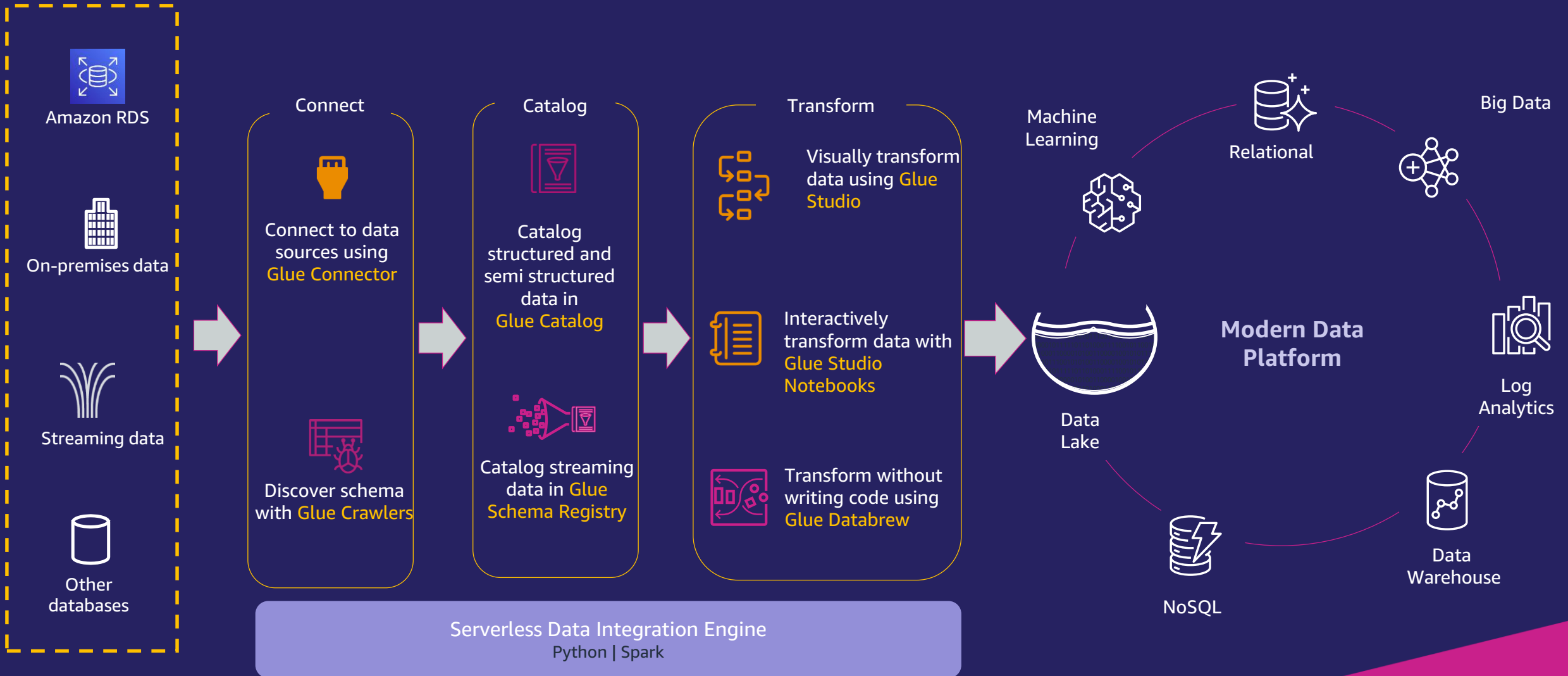
## SOLUTION

- Built self-service cloud hub platform and used AWS Glue for structured data ingestion
- Created self-service features with a customized UI and used Glue APIs to automatically provision AWS Glue jobs for data ingestion
- Build a customized level data catalog on top of AWS Glue Catalog and Catalog APIs

## IMPACT

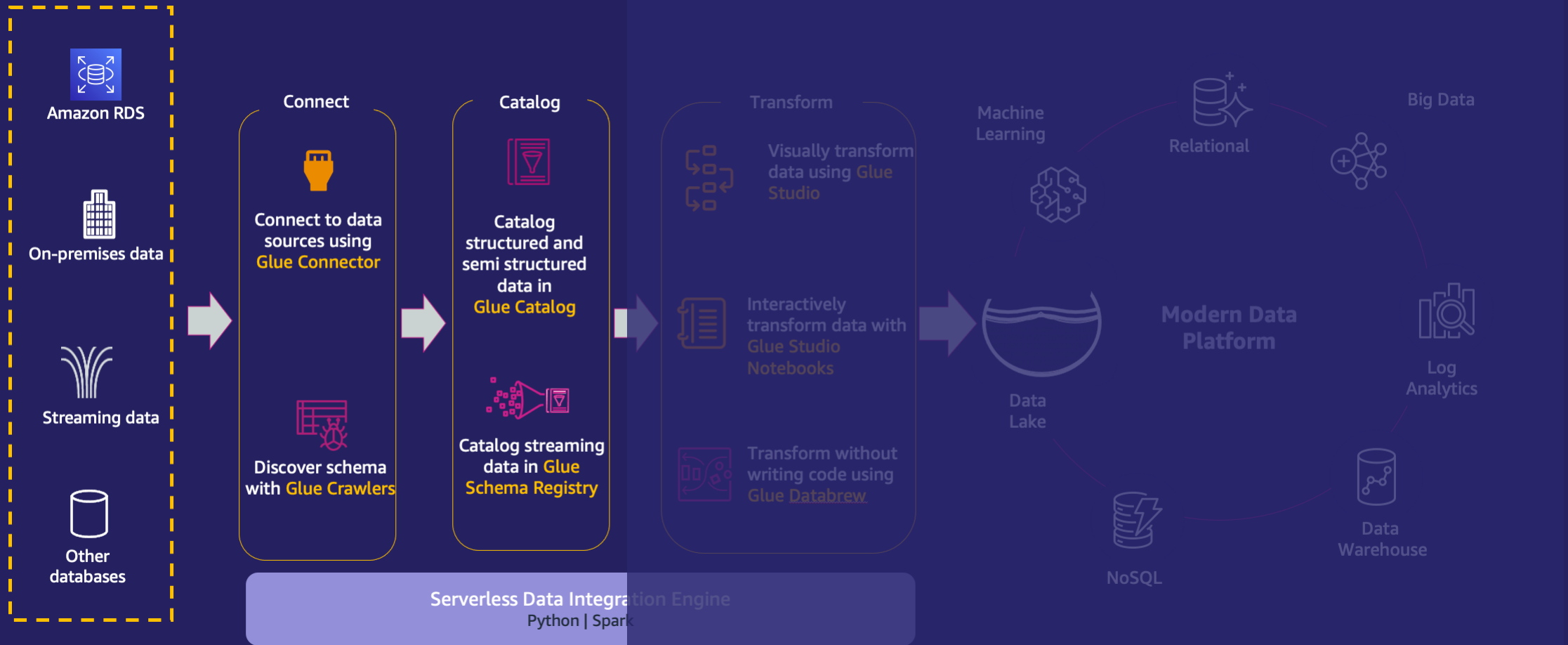
- Significant adoption of the Cloud HUB data platform with internal users
- Users self-serve data instead of relying on data engineers, shortening data ingestion cycles

# A Data Integration ecosystem for building Modern Data Platforms





# Data Ingestion



# Glue Connectors Marketplace



splunk > + Many more...

# Custom Connectors with AWS Glue

## Data Sources

On-premises DBs

Proprietary stores

SaaS applications

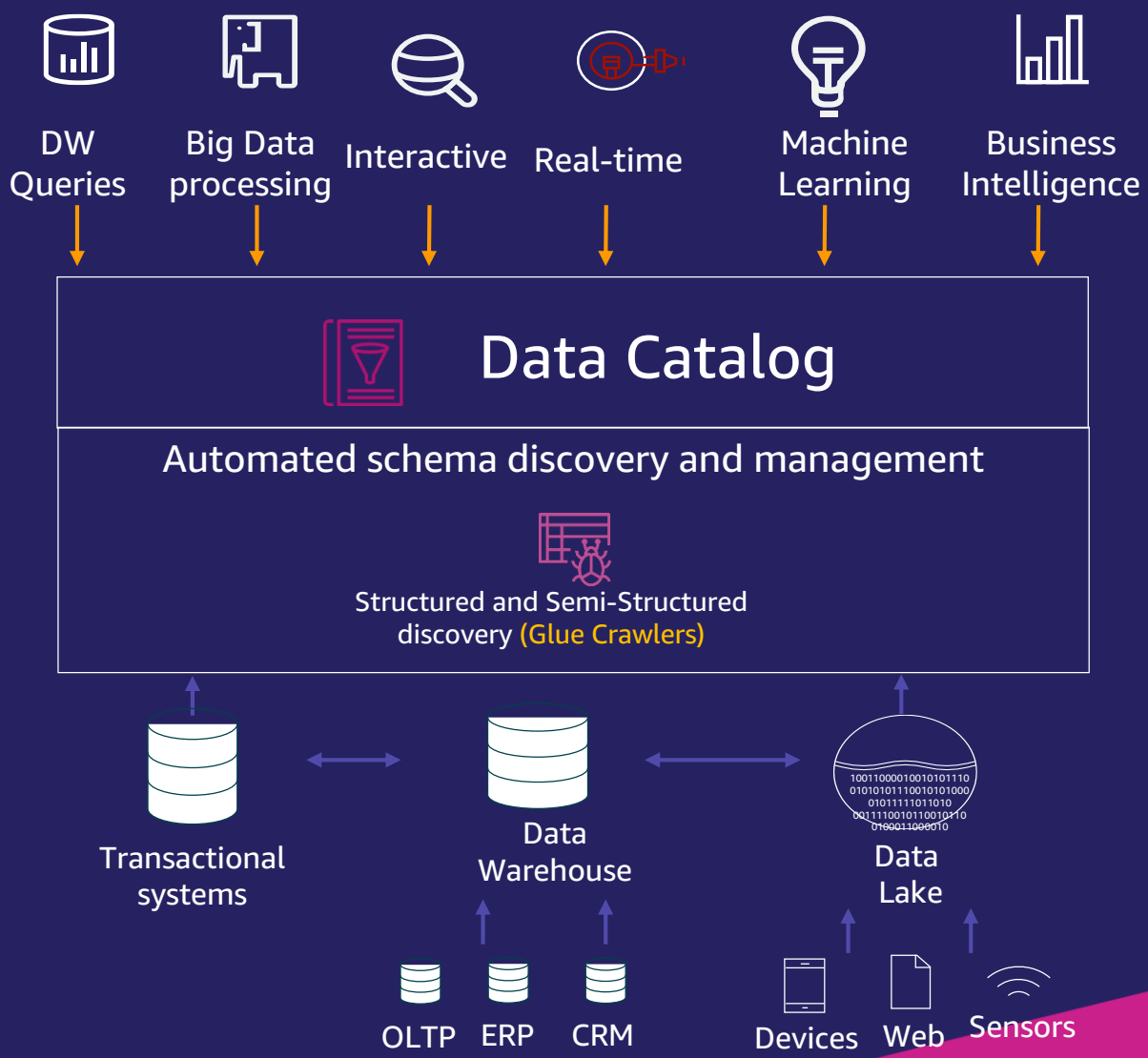


**CUSTOM CONNECTOR**

- No additional cost for connecting to sources
- Flexible and easy to build connectors

# Unified Data Catalog with automated schema discovery

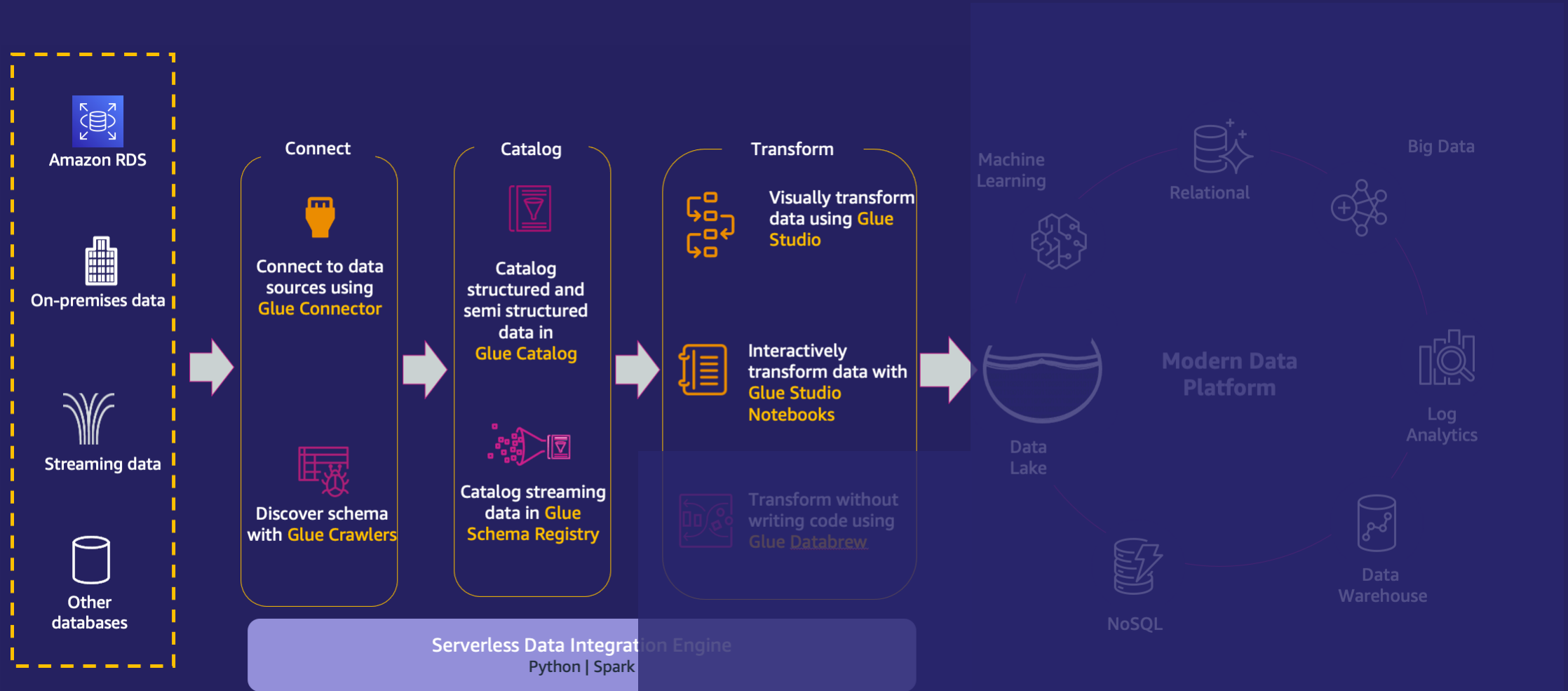
- No movement of data = Low Costs/Admin
- All metadata centrally available for search and query = Productivity
- Unify structured, semi-structured data = Speed to Insight
- Automate data discovery = Productivity



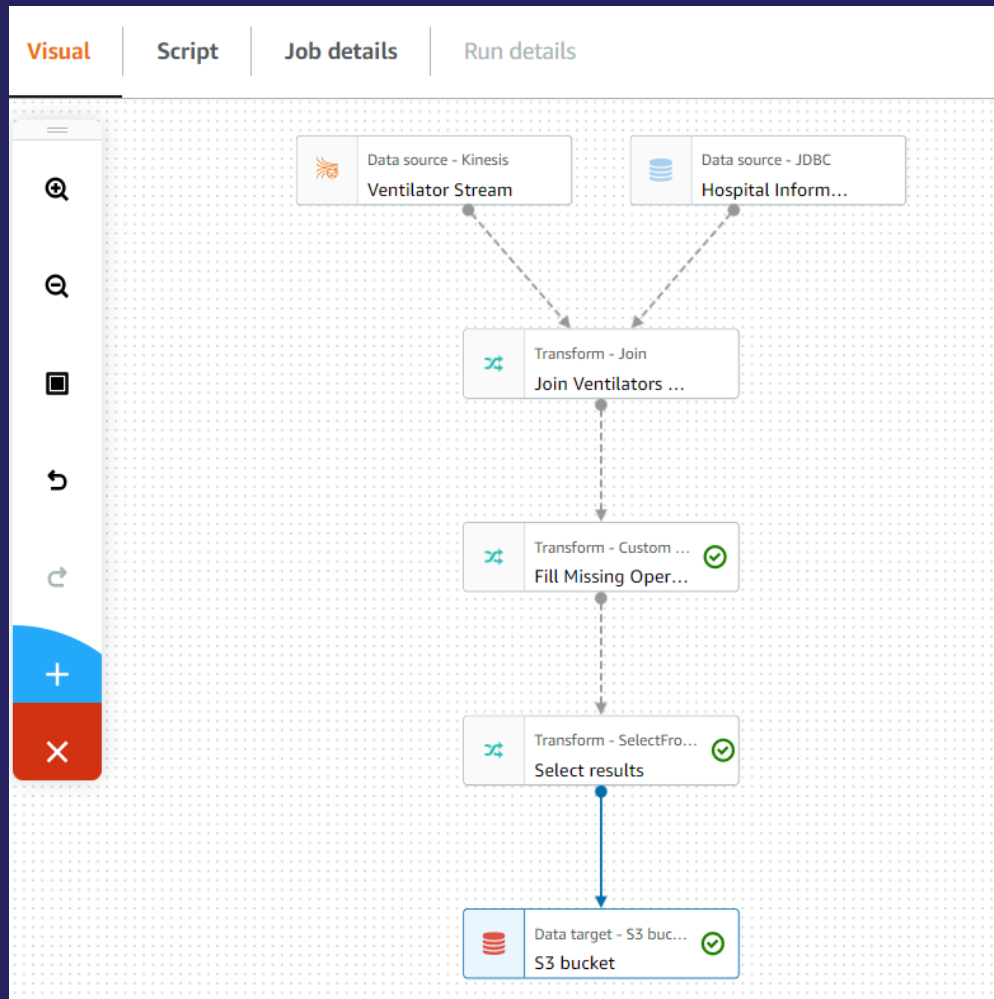
# Data Ingestion Demo



# Data Transformation



# AWS Glue Studio: Visual ETL interface



Author AWS Glue jobs **visually** without coding

Monitor **1000s of jobs** through a **single pane of glass**

**Distributed processing** without the learning curve

Advanced transforms **though code snippets**

# AWS Glue Studio Notebook (Preview)

Preview

The screenshot shows the AWS Glue Studio Notebook interface. At the top, there are tabs for 'Notebook', 'Job details', 'Runs', and 'Schedules'. Below the tabs, there are icons for file operations and a 'Download' button. The main area contains a code cell with the following content:

```
[3]: # Execute this cell to configure and start your interactive session.
%%session_id_prefix my-session-bt2qj
%%configure
{
  "region": "us-east-1",
  "iam_role": "arn:aws:iam:590186200215:role/NotebookLifecycleTestRole"
}
...

[6]: %%sql
select * from `covid-19`.`country_codes` limit 10
```

The output of the SQL query is displayed as a table:

country	alpha-2 code	alpha-3 code	numeric code	latitude	longitude
Afghanistan	AF	AFG	4	33	65
Albania	AL	ALB	8	41	20
Algeria	DZ	DZA	12	28	3
American Samoa	AS	ASM	16	-14	-170
Andorra	AD	AND	20	42	1
Angola	AO	AGO	24	-12	18

Below the table, the output of the Python code is shown:

```
[10]: dyf = glue_context.create_dynamic_frame.from_catalog(
        database='covid-19',
        table_name='country_codes')
dyf.show()

{"country": "Afghanistan", "alpha-2 code": "AF", "alpha-3 code": "AFG", "numeric code": 4, "latitude": 33, "longitude": 65}
{"country": "Albania", "alpha-2 code": "AL", "alpha-3 code": "ALB", "numeric code": 8, "latitude": 41, "longitude": 20}
{"country": "Algeria", "alpha-2 code": "DZ", "alpha-3 code": "DZA", "numeric code": 12, "latitude": 28, "longitude": 3}
{"country": "American Samoa", "alpha-2 code": "AS", "alpha-3 code": "ASM", "numeric code": 16, "latitude": -14, "longitude": -170}
{"country": "Andorra", "alpha-2 code": "AD", "alpha-3 code": "AND", "numeric code": 20, "latitude": 42, "longitude": 1}
{"country": "Angola", "alpha-2 code": "AO", "alpha-3 code": "AGO", "numeric code": 24, "latitude": -12, "longitude": 18}
{"country": "Anguilla", "alpha-2 code": "AI", "alpha-3 code": "AIA", "numeric code": 660, "latitude": 18.25, "longitude": -63.1667}
{"country": "Antarctica", "alpha-2 code": "AQ", "alpha-3 code": "ATA", "numeric code": 10, "latitude": -90, "longitude": 0}
{"country": "Antigua and Barbuda", "alpha-2 code": "AG", "alpha-3 code": "ATG", "numeric code": 28, "latitude": 17.05, "longitude": -61.8}
{"country": "Argentina", "alpha-2 code": "AR", "alpha-3 code": "ARG", "numeric code": 32, "latitude": -34, "longitude": -64}
{"country": "Armenia", "alpha-2 code": "AM", "alpha-3 code": "ARM", "numeric code": 51, "latitude": 40, "longitude": 45}
{"country": "Aruba", "alpha-2 code": "AW", "alpha-3 code": "ABW", "numeric code": 533, "latitude": 12.5, "longitude": -69.9667}
{"country": "Australia", "alpha-2 code": "AU", "alpha-3 code": "AUS", "numeric code": 36, "latitude": -27, "longitude": 133}
{"country": "Austria", "alpha-2 code": "AT", "alpha-3 code": "AUT", "numeric code": 40, "latitude": 47.3333, "longitude": 13.3333}
{"country": "Azerbaijan", "alpha-2 code": "AZ", "alpha-3 code": "AZE", "numeric code": 31, "latitude": 40.5, "longitude": 47.5}
{"country": "Bahamas", "alpha-2 code": "BS", "alpha-3 code": "BHS", "numeric code": 44, "latitude": 24.25, "longitude": -76}
{"country": "Bahrain", "alpha-2 code": "BH", "alpha-3 code": "BHR", "numeric code": 48, "latitude": 26, "longitude": 50.55}
{"country": "Bangladesh", "alpha-2 code": "BD", "alpha-3 code": "BGD", "numeric code": 50, "latitude": 24, "longitude": 90}
{"country": "Barbados", "alpha-2 code": "BB", "alpha-3 code": "BRB", "numeric code": 52, "latitude": 13.1667, "longitude": -59.5333}
{"country": "Belarus", "alpha-2 code": "BY", "alpha-3 code": "BLR", "numeric code": 112, "latitude": 53, "longitude": 28}
```

Interactive AWS Glue jobs development

Submit AWS Glue jobs from the AWS Glue Studio notebook

Use notebook magic to define transforms in SQL and control cost

Built-in monitoring support





# Glue Studio offers out of box transformations...

Node properties 1 Transform Output schema Data preview

Name  
TRANSFORM

Node type  
Choose which type of node to add to the job.

FillMissingValues  
Impute missing values in a selected data field.

Join  
Join two sources into one output using a column header.

SplitFields  
Split data into two datasets, separating the specified fields.

SelectFromCollection  
Choose a single DynamicFrame from a collection of DynamicFrames.

Filter  
Filter data based on different sets of rules.

Union  
Union two sources into one output.

Aggregate  
Applying aggregate functions on specified fields.

FillMissingValues  
Impute missing values in a selected data field.

Custom transform  
Custom operation written as code.

Spark SQL  
Custom Spark SQL code to execute.

Code block Info  
Enter a custom script to add to your job.

```
1 object MyTransform {  
2   def execute(glueContext : GlueContext, input : Seq[DynamicFrame]) : Seq[DynamicFrame] = {  
3  
4   }  
5 }
```

Code block Info  
Enter a custom script to add to your job.

```
1 def MyTransform (glueContext, dfc) -> DynamicFrameCollection:  
2   |
```

Code block  
Enter SQL code to add to your job.

```
1 select * from myDataSource  
2   |
```



built-in  
transform

# ... and advanced transforms like PII detection for advanced cleansing



# AWS Glue streaming for real-time ETL

Prepare streaming data and make it available for analytics in seconds



Easily integrate with  
**Amazon MSK,**  
**Amazon Kinesis,** and  
**Apache Kafka**

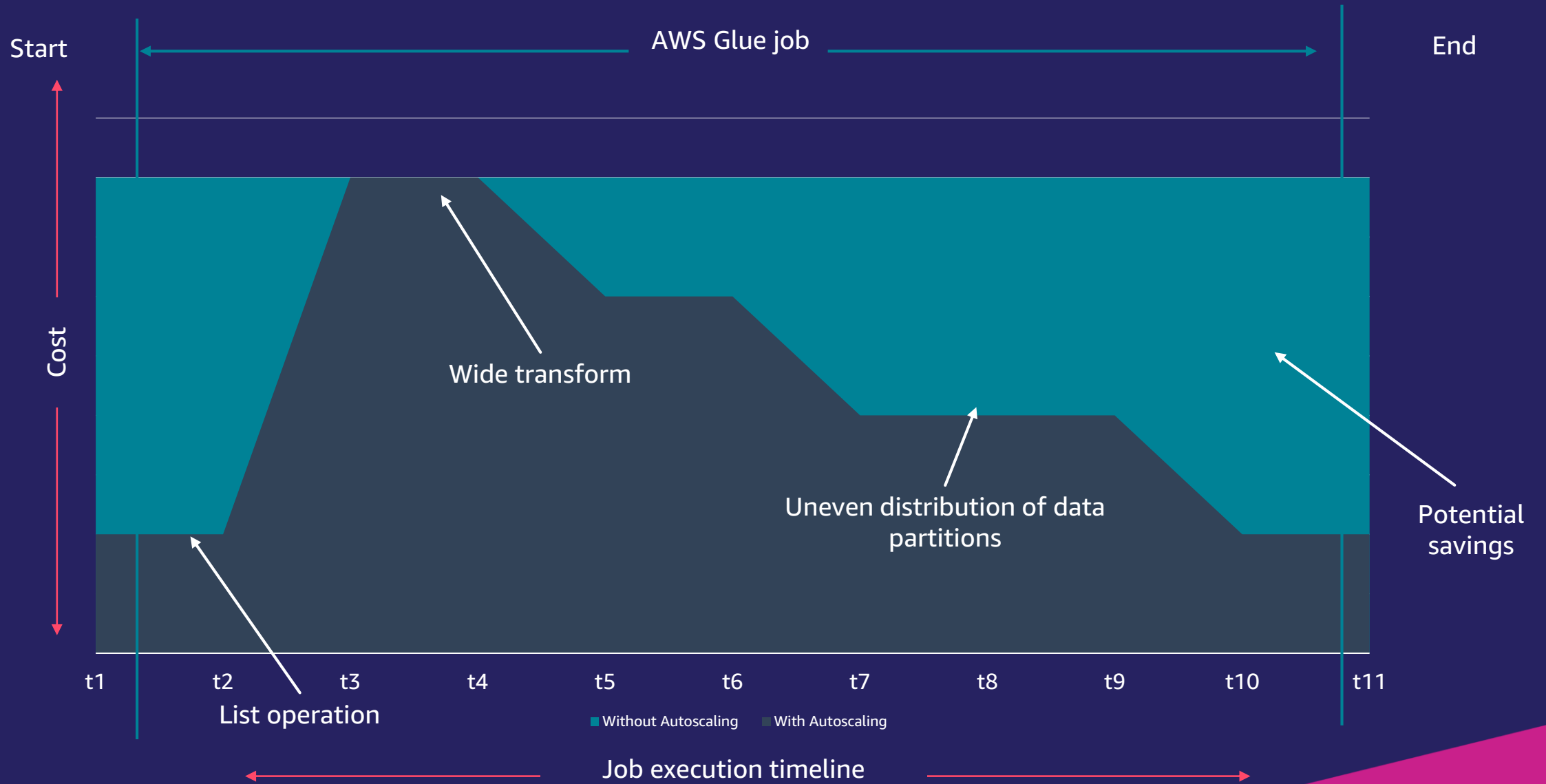


Combine, enrich,  
and transform **data**  
**in flight** using  
**Glue Studio**

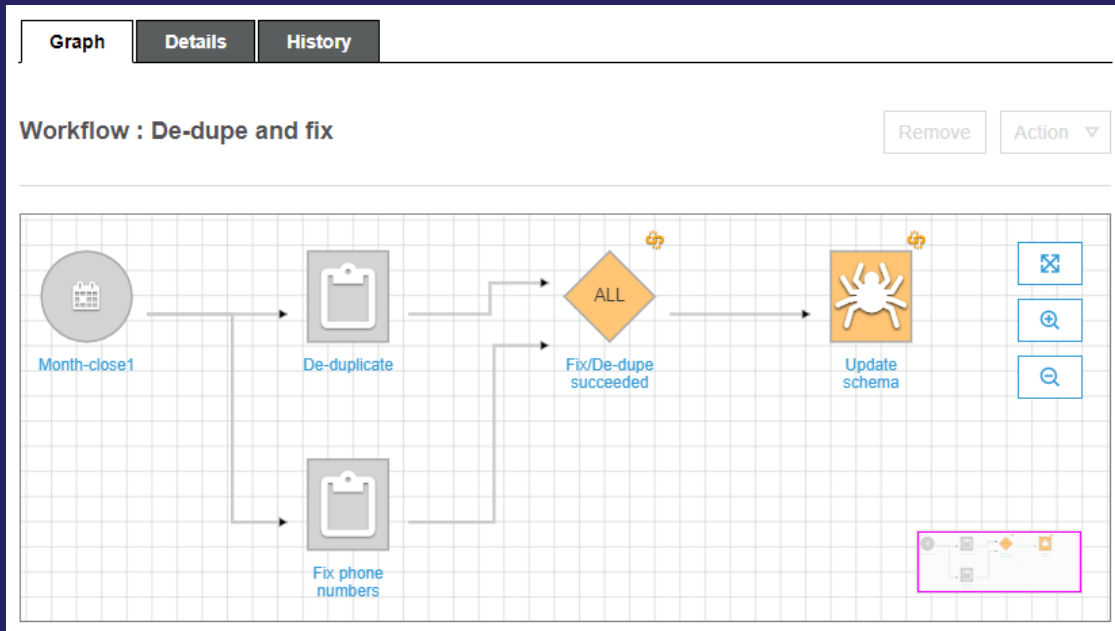


Adapt to **varied**  
and **changing**  
**schemas**

# Auto-scaling (Preview)



# Orchestrate jobs easily with AWS Glue workflows



Orchestrate Glue jobs and other AWS services

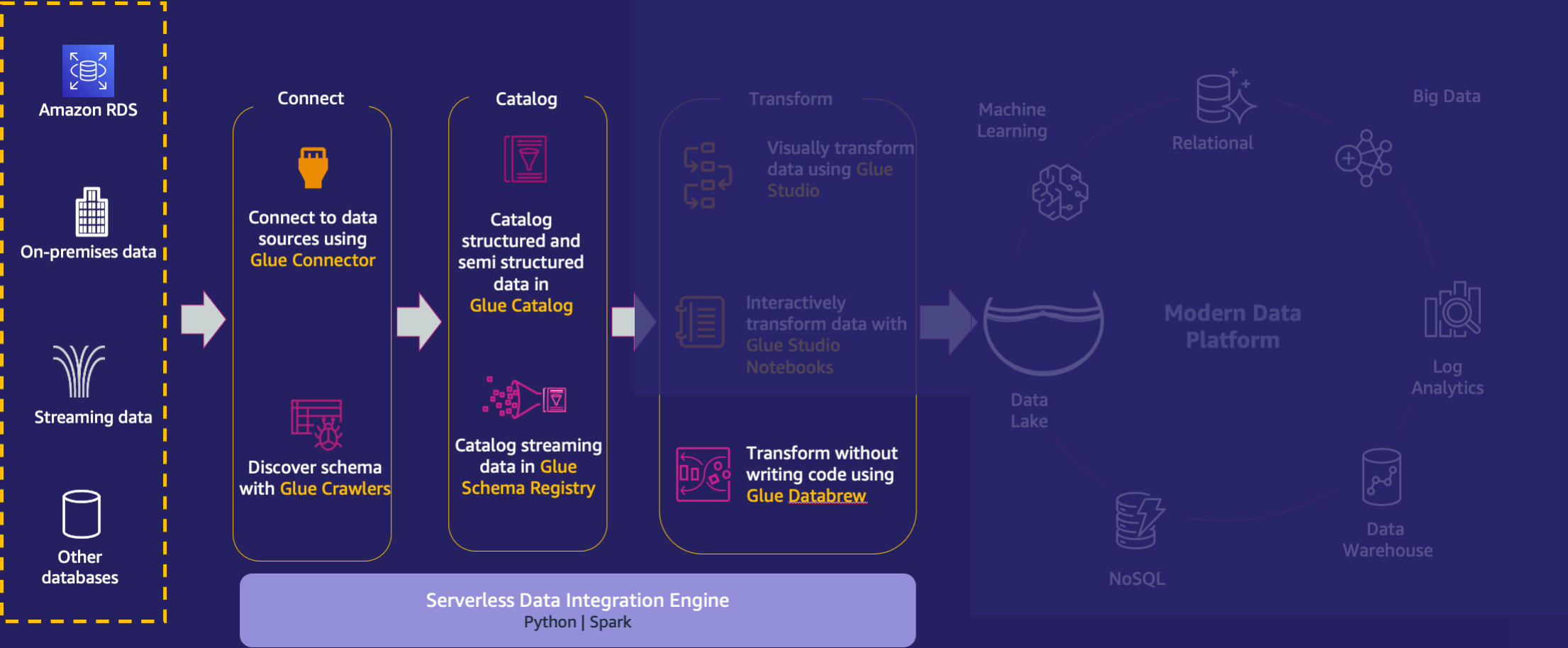
Schedule jobs or trigger based on events

Monitor execution of the workflows in one place

# Data Transformation Demo

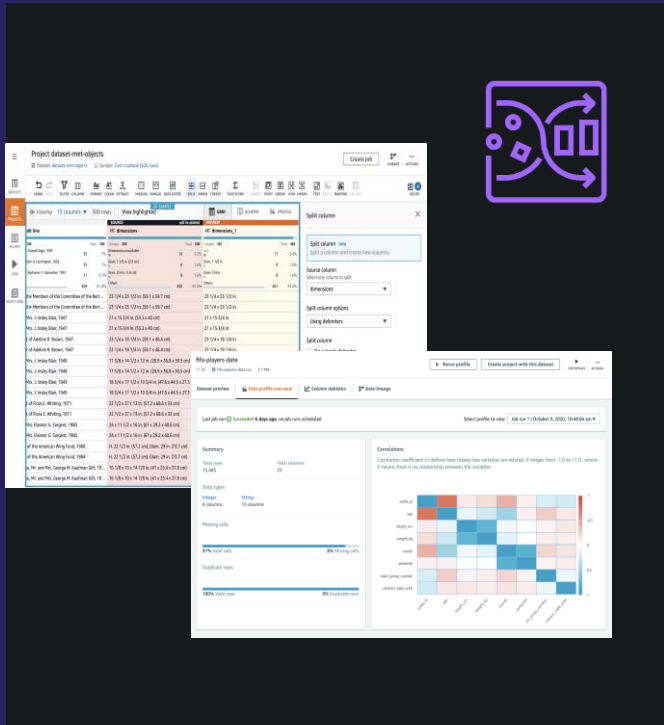


# Data Preparation



# AWS Glue DataBrew

## Visual data preparation for analytics and machine learning



Clean and normalize data with a visual interface

250+ built-in transformations without writing code

Profile data to understand data patterns and anomalies

Work on large datasets at scale



# Data Preparation Demo



# Migrate to AWS Glue



Build Data Lakes and Lake Houses for scalable data analysis



Migrate from expensive traditional ETL solutions to gain flexibility and reduce costs



Process data in real time using Glue Streaming



Process petabytes of data both in batch using Apache Spark



Prepare data for Machine Learning



**Thank you!**