AWS Glue

Serverless Data Integration Service

Extract, Transform, Load (ETL) and Data Catalog

AWS Training Series

What is AWS Glue?

AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easy to prepare and load your data for analytics.

Key Characteristics:

- **Serverless:** No infrastructure to manage or provision
- **Fully Managed:** AWS handles scaling, patching, and maintenance
- **Pay-per-use:** Only pay for resources consumed during job runs
- **Apache Spark-based:** Built on Apache Spark for distributed processing
- **Code Generation:** Automatically generates ETL code in Python or Scala

Main Components:

Data Catalog

Central metadata repository for all your data assets

ETL Engine

Serverless Apache Spark environment for data transformation

Crawlers

Automatically discover and catalog data schemas

Job Scheduler

Trigger jobs based on schedules or events

Why Do We Need AWS Glue?

Traditional ETL Challenges:

- **Infrastructure Management:** Setting up and maintaining ETL servers
- **Scaling Issues:** Manual scaling for varying workloads
- **Schema Discovery:** Manually cataloging data sources and schemas
- **Code Complexity:** Writing complex ETL code from scratch
- **Data Silos:** Difficulty in discovering and accessing distributed data
- **Cost Inefficiency:** Paying for idle infrastructure

AWS Glue Solutions:

🚀 Serverless

No infrastructure to manage, automatic scaling

📊 Auto Discovery

Crawlers automatically discover and catalog data

🔧 Code Generation

💰 Cost Effective

Automatically generates ETL code with visual editor

Pay only for what you use, no idle costs

AWS Glue Use Cases

1. Data Lake Analytics

Build and maintain data lakes by cataloging, cleaning, and transforming data from various sources into analytics-ready formats.

2. Data Warehouse Modernization

- Migrate from traditional on-premises data warehouses
- Transform and load data into Amazon Redshift or other cloud data warehouses
- Implement incremental data loading strategies

3. Real-time Analytics

- Process streaming data from Kinesis Data Streams
- Transform and enrich real-time data
- Load processed data into analytics services

4. Machine Learning Data Preparation

- Clean and prepare data for ML model training
- Feature engineering and data transformation
- Integration with Amazon SageMaker

5. Data Integration & Migration

- Integrate data from multiple sources (databases, files, APIs)
- Migrate data between different storage systems
- Standardize data formats across the organization

AWS Glue Data Catalog

The AWS Glue Data Catalog is a central metadata repository that stores structural and operational metadata for all your data assets.

What is the Data Catalog?

- **Metadata Repository:** Stores table definitions, schema information, and other metadata
- **Hive Metastore Compatible:** Works with Apache Spark, Hive, and Presto
- **Integrated with AWS Services:** Amazon Athena, EMR, Redshift Spectrum use it
- **Searchable:** Find and discover data assets across your organization

Key Benefits:

🔍 Data Discovery

Easily find and understand available data assets

📋 Schema Management

Centralized schema evolution and versioning

🔗 Service Integration

Single source of truth for all AWS analytics services

🏷️ Data Governance

Tag and classify data for compliance and governance

Data Catalog: Databases

What is a Database in Glue Data Catalog?

A database is a logical grouping of tables in the AWS Glue Data Catalog. It's similar to a schema in traditional databases.

Database Characteristics:

- **Logical Container:** Groups related tables together
- **Namespace:** Provides organization and access control
- **Metadata Storage:** Contains database-level properties and descriptions
- **Access Control:** IAM policies can be applied at database level

Creating a Database:

```
# AWS CLI
aws glue create-database --database-input '{
    "Name": "sales_database",
    "Description": "Database for sales and customer data"
}'
```

```
# Python (Boto3)
import boto3
glue = boto3.client('glue')

glue.create_database(
    DatabaseInput={
        'Name': 'sales_database',
        'Description': 'Database for sales and customer 
data'
    }
)
```

Best Practices:

- Use descriptive names that reflect the data domain
- Group related tables logically (e.g., by business unit, data source)
- Add meaningful descriptions for documentation
- Consider data governance and access patterns when organizing

Data Catalog: Tables

What is a Table in Glue Data Catalog?

> A table represents the metadata of your data, including schema, location, and format information. It doesn't store the actual data.

Table Components:

- **Schema:** Column names, data types, and constraints
- **Location:** S3 path or database connection information
- **Format:** Parquet, JSON, CSV, Avro, ORC, etc.
- **Partitioning:** How data is organized for performance
- **Properties:** Additional metadata and configuration

Table Types:

Type	Description	Use Case
S3 Tables	Metadata for files stored in S3	Data lakes, file-based analytics

Type	Description	Use Case
Database Tables	Metadata for tables in RDS, Redshift	Traditional database integration
Streaming Tables	Metadata for Kinesis streams	Real-time data processing

Data Catalog Tables

Data Catalog: Connections

What are Connections?

Connections store connection information for data stores that require authentication or network configuration, such as databases, data warehouses, and other services.

Connection Types:

JDBC Connections

MySQL, PostgreSQL, Oracle, SQL Server, Redshift

MongoDB

NoSQL document database connections

Kafka

Apache Kafka streaming platform

Network

VPC, subnet, and security group configurations

Connection Properties:

- **Connection String:** Database URL and port
- **Credentials:** Username/password or IAM role
- **Network Configuration:** VPC, subnet, security groups
- **SSL/TLS Settings:** Encryption in transit configuration

Example JDBC Connection:

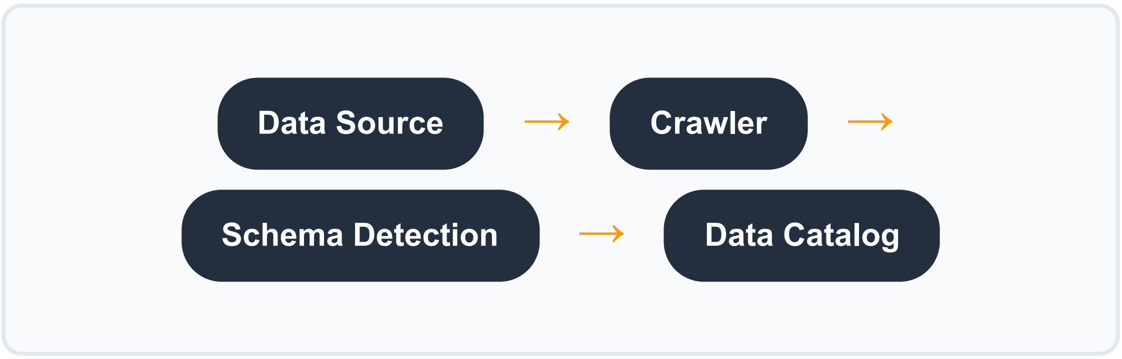
```
{
    "Name": "mysql-production",
    "ConnectionType": "JDBC",
    "ConnectionProperties": {
        "JDBC_CONNECTION_URL": "jdbc:mysql://prod-
db.example.com:3306/sales",
        "USERNAME": "glue_user",
        "PASSWORD": "stored-in-secrets-manager"
    },
    "PhysicalConnectionRequirements": {
        "SubnetId": "subnet-12345678",
        "SecurityGroupIdList": ["sg-87654321"],
        "AvailabilityZone": "us-east-1a"
    }
}
```

Data Catalog: Crawlers

What are Crawlers?

Crawlers are programs that connect to data stores, determine the schema of your data, and create metadata tables in the AWS Glue Data Catalog.

How Crawlers Work:



Crawler Capabilities:

- **Schema Inference:** Automatically detects data types and structure
- **Partition Discovery:** Identifies partitioning schemes
- **Schema Evolution:** Handles changes in data structure over time
- **Multiple Formats:** Supports JSON, Parquet, CSV, Avro, ORC, etc.
- **Incremental Updates:** Only processes new or changed data

Supported Data Stores:

Amazon S3

Files and folders in S3 buckets

JDBC Databases

RDS, Redshift, on-premises databases

DynamoDB

NoSQL tables and indexes

DocumentDB

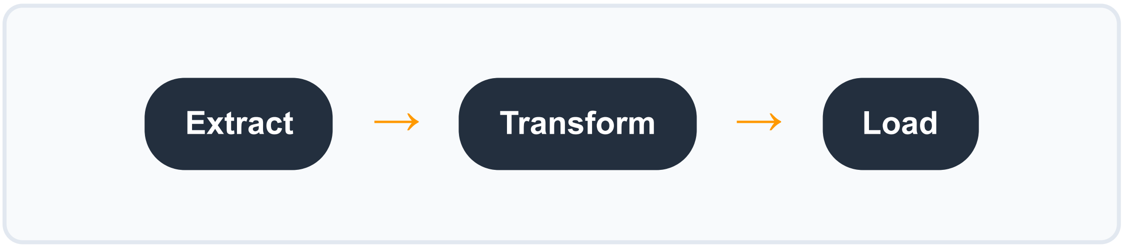
MongoDB-compatible document database

Data Integration & ETL

What is ETL?

ETL (Extract, Transform, Load) is the process of extracting data from various sources, transforming it into a suitable format, and loading it into a target system for analysis.

ETL Process:



Extract:

- Pull data from various sources (databases, files, APIs, streams)
- Handle different data formats and structures
- Manage connection and authentication

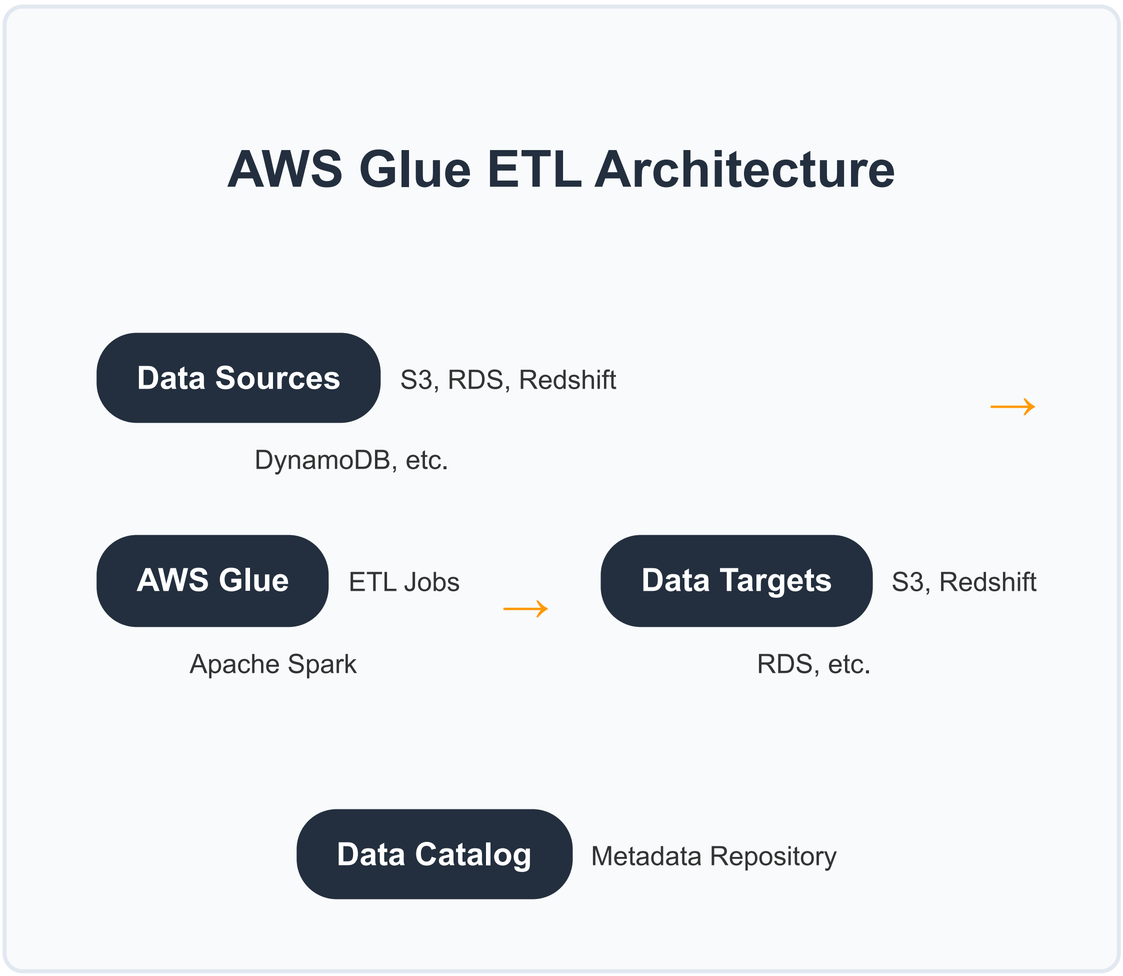
Transform:

- Clean and validate data quality
- Apply business rules and calculations
- Join data from multiple sources
- Aggregate and summarize data
- Convert data types and formats

Load:

- Write transformed data to target systems
- Handle incremental vs. full loads
- Optimize for query performance

AWS Glue ETL Architecture



Key Components:

Job Execution Environment

Serverless Apache Spark clusters that auto-scale

Development Endpoints

Interactive development environment for testing ETL scripts based on workload

Job Scheduler

Trigger jobs on schedule, events, or job completion

Monitoring & Logging

CloudWatch integration for job monitoring and debugging

Zero ETL Integration

What is Zero ETL?

Zero ETL eliminates the need to build and maintain complex data pipelines by providing near real-time data replication and transformation capabilities.

AWS Zero ETL Integrations:

Source	Target	Use Case
Amazon RDS	Amazon Redshift	Real-time analytics on transactional data
Amazon Aurora	Amazon Redshift	Operational analytics without ETL complexity
Amazon DynamoDB	Amazon OpenSearch	Real-time search and analytics

Benefits of Zero ETL:

- **Reduced Complexity:** No ETL pipelines to build or maintain
- **Near Real-time:** Data available for analytics within minutes

- **Cost Effective:** Eliminates ETL infrastructure costs
- **Automatic Scaling:** Handles varying data volumes automatically
- **Data Consistency:** Maintains ACID properties during replication

When to Use Zero ETL vs Traditional ETL:

Use Zero ETL When:

- Minimal data transformation needed
- Real-time analytics required
- Simple replication scenarios

Use Traditional ETL When:

- Complex transformations required
- Multiple data sources to combine
- Custom business logic needed

Different Ways to Write ETL Jobs

1. Visual ETL Editor (AWS Glue Studio)

Drag-and-drop interface for creating ETL jobs without writing code.

- **No Code Required:** Visual interface for non-programmers
- **Pre-built Transforms:** Common transformations available as blocks
- **Auto Code Generation:** Generates Python/Scala code automatically
- **Job Monitoring:** Built-in job execution monitoring

2. Script Editor

- **Python/Scala:** Write custom ETL scripts
- **PySpark/Spark:** Full Apache Spark capabilities
- **AWS Glue Libraries:** Pre-built functions for common operations
- **Custom Logic:** Implement complex business rules

3. Jupyter Notebooks

- **Interactive Development:** Test and iterate on ETL logic
- **Data Exploration:** Analyze data before transformation
- **Collaboration:** Share notebooks with team members
- **Documentation:** Combine code with explanatory text

4. AWS Glue DataBrew

- **Visual Data Preparation:** Point-and-click data cleaning
- **250+ Transformations:** Pre-built data preparation functions
- **Data Profiling:** Automatic data quality assessment
- **No Code Required:** Business users can prepare data

ETL Job Example - Python Script

Sample ETL Job: Transform Sales Data

```
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

# Initialize Glue context
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Read data from Data Catalog
datasource = glueContext.create_dynamic_frame.from_catalog(
    database="sales_database",
    table_name="raw_sales_data"
)

# Apply transformations
# 1. Drop null values
cleaned_data = DropNullFields.apply(frame=datasource)

# 2. Rename columns
renamed_data = RenameField.apply(
    frame=cleaned_data,
    old_name="customer_id",
    new_name="cust_id"
)

# 3. Filter data (sales > 100)
filtered_data = Filter.apply(
    frame=renamed_data,
    f=lambda x: x["sales_amount"] > 100
)

# 4. Add calculated field
def add_tax(rec):
    rec["total_with_tax"] = rec["sales_amount"] * 1.08
    return rec

final_data = Map.apply(frame=filtered_data, f=add_tax)

# Write to S3
glueContext.write_dynamic_frame.from_options(
    frame=final_data,
    connection_type="s3",
    connection_options={
        "path": "s3://my-bucket/processed-sales/"
    },
    format="parquet"
)

job.commit()
```

ETL Job Example

AWS Glue Triggers

What are Triggers?

Triggers are mechanisms that start ETL jobs based on schedules, events, or job completion status.

Trigger Types:

Trigger Type	Description	Use Case
Scheduled	Run jobs on a cron schedule	Daily/weekly batch processing
On-Demand	Manually triggered jobs	Ad-hoc data processing
Conditional	Trigger based on job completion	Job dependencies and workflows
Event-based	S3 events, CloudWatch events	Real-time data processing

Trigger Configuration Example:

```
# Scheduled Trigger - Daily at 2 AM
{
    "Name": "daily-sales-etl",
    "Type": "SCHEDULED",
    "Schedule": "cron(0 2 * * ? *)",
    "Actions": [
        {
            "JobName": "sales-data-processing",
            "Arguments": {
                "--job-bookmark-option": "job-bookmark-
enable"
            }
        }
    ]
}

# Conditional Trigger - Run after job completion
{
    "Name": "dependent-job-trigger",
    "Type": "CONDITIONAL",
    "Predicate": {
        "Conditions": [
            {
                "LogicalOperator": "EQUALS",
                "JobName": "extract-job",
                "State": "SUCCEEDED"
            }
        ]
    },
    "Actions": [
        {
            "JobName": "transform-job"
        }
    ]
}
```

AWS Glue Triggers

Job Bookmarks & Monitoring

Job Bookmarks

> Job bookmarks help AWS Glue maintain state information and prevent reprocessing of old data in subsequent job runs.

How Job Bookmarks Work:

- **State Tracking:** Keeps track of processed data
- **Incremental Processing:** Only processes new or changed data
- **Automatic Management:** No manual intervention required
- **Cost Optimization:** Reduces processing time and costs

Monitoring & Logging

CloudWatch Metrics

- Job execution duration
- Data processed volume

CloudWatch Logs

- Detailed job execution logs

- Error rates and success rates

- Error messages and stack traces
- Custom application logs

AWS Glue Console

- Job run history and status
- Performance metrics
- Data lineage visualization

CloudTrail Integration

- API call logging
- Security and compliance auditing
- Change tracking

Best Practices for Monitoring:

- Set up CloudWatch alarms for job failures
- Monitor job duration for performance optimization
- Use custom metrics for business-specific monitoring
- Implement proper error handling and retry logic

Practical Example: E-commerce Data Pipeline

Scenario:

An e-commerce company needs to process daily sales data from multiple sources and create analytics-ready datasets for business intelligence.

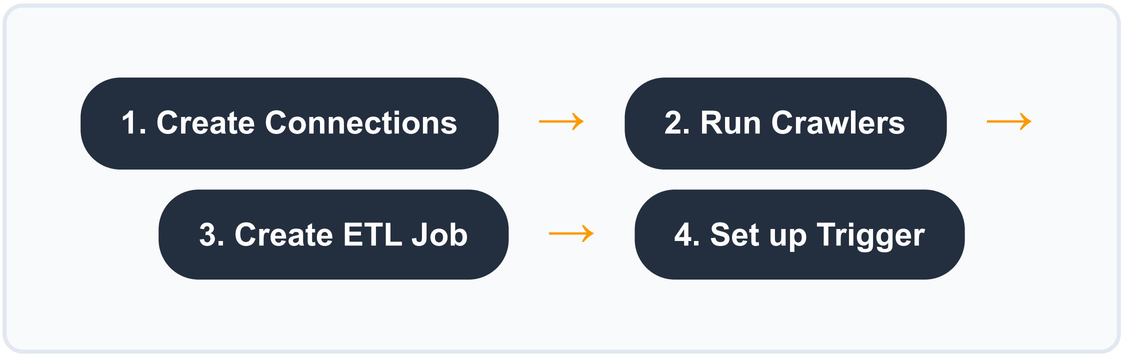
Data Sources:

- **MySQL Database:** Customer and product information
- **S3 Files:** Daily sales transactions (CSV format)
- **DynamoDB:** User behavior and clickstream data

Requirements:

- Combine data from all sources
- Clean and validate data quality
- Calculate daily sales metrics
- Store results in S3 for analytics
- Run daily at 3 AM

Implementation Steps:



Implementation Details

Step 1: Create Database Connection

```
# Create JDBC connection for MySQL
aws glue create-connection --connection-input '{
    "Name": "mysql-ecommerce",
    "ConnectionType": "JDBC",
    "ConnectionProperties": {
        "JDBC_CONNECTION_URL": "jdbc:mysql://ecommerce-
db.amazonaws.com:3306/ecommerce",
        "USERNAME": "glue_user",
        "PASSWORD": "secure_password"
    },
    "PhysicalConnectionRequirements": {
        "SubnetId": "subnet-12345678",
        "SecurityGroupIdList": ["sg-87654321"]
    }
}'
```

Step 2: Create and Run Crawlers

```
# Crawler for S3 sales data
aws glue create-crawler --name "s3-sales-crawler" --role
"AWSGlueServiceRole" \
    --database-name "ecommerce_db" \
    --targets '{"S3Targets": [{"Path": "s3://ecommerce-
data/sales/"}]}'
```

```
# Crawler for MySQL tables
aws glue create-crawler --name "mysql-crawler" --role
"AWSGlueServiceRole" \
    --database-name "ecommerce_db" \
    --targets '{"JdbcTargets": [{"ConnectionName": "mysql-
ecommerce", "Path": "ecommerce/%"}]}'

# Start crawlers
aws glue start-crawler --name "s3-sales-crawler"
aws glue start-crawler --name "mysql-crawler"
```

Step 3: Key ETL Transformations

- **Join Operations:** Combine sales data with customer and product info
- **Data Cleaning:** Remove duplicates and handle missing values
- **Aggregations:** Calculate daily sales totals by product category
- **Data Quality:** Validate email formats and price ranges

AWS Glue Best Practices

Performance Optimization:

- **Partitioning:** Partition data by date, region, or other logical divisions
- **File Formats:** Use columnar formats (Parquet, ORC) for better performance
- **Compression:** Enable compression to reduce storage and I/O costs
- **Worker Allocation:** Right-size DPU (Data Processing Units) based on workload
- **Job Bookmarks:** Enable to avoid reprocessing data

Cost Optimization:

- **Spot Instances:** Use Glue Flex for non-time-critical workloads
- **Resource Monitoring:** Monitor and adjust DPU allocation
- **Data Lifecycle:** Implement S3 lifecycle policies for processed data
- **Incremental Processing:** Process only new/changed data

Security Best Practices:

- **IAM Roles:** Use least privilege principle for service roles
- **Encryption:** Enable encryption at rest and in transit

- **VPC Configuration:** Use VPC endpoints for secure communication
- **Secrets Management:** Store credentials in AWS Secrets Manager

Data Quality:

- **Data Validation:** Implement data quality checks in ETL jobs
- **Error Handling:** Proper exception handling and retry logic
- **Data Lineage:** Track data flow and transformations
- **Testing:** Test ETL jobs with sample data before production

AWS Glue Pricing

Pricing Model:

AWS Glue follows a pay-per-use model - you only pay for the resources consumed during job execution.

ETL Jobs Pricing:

Component	Unit	Price (US East)
ETL Jobs (Standard)	DPU-Hour	$0.44
ETL Jobs (Flex)	DPU-Hour	$0.29
Data Catalog	Per 100,000 requests	$1.00
Crawlers	DPU-Hour	$0.44

Cost Factors:

- **DPU Allocation:** Number of Data Processing Units allocated
- **Job Duration:** How long jobs run (billed per second, 1-minute minimum)

- **Data Volume:** Amount of data processed affects job duration
- **Frequency:** How often jobs are executed

Cost Optimization Tips:

- Use Glue Flex for non-urgent workloads (34% cost savings)
- Optimize job performance to reduce execution time
- Use job bookmarks to avoid reprocessing data
- Monitor and right-size DPU allocation

Key Takeaways

AWS Glue Advantages:

🚀 Serverless

No infrastructure management, automatic scaling

📊 Integrated

Works seamlessly with AWS analytics services

🔧 Flexible

Multiple ways to create ETL jobs (visual, code, notebooks)

💰 Cost-Effective

Pay only for what you use, no idle costs

When to Use AWS Glue:

- **Data Lake Analytics:** Building and maintaining data lakes
- **Data Warehouse ETL:** Loading data into Redshift or other warehouses
- **Data Integration:** Combining data from multiple sources
- **Schema Discovery:** Automatically cataloging data assets
- **Serverless Requirements:** No infrastructure management needed

Getting Started:

1. Set up IAM roles and permissions
2. Create connections to your data sources
3. Run crawlers to populate the Data Catalog
4. Create your first ETL job using Glue Studio
5. Set up triggers and monitoring

Next Steps: Start with a simple ETL job using the visual editor, then gradually move to more complex scenarios as you become comfortable with the service.