Build and automate a modern serverless data lake on AWS

Aditya Challa
AWS Solutions Architect
Amazon Web Services
A **data lake** is a system or repository of data stored in its natural/raw format, usually object blobs or files. A data lake is usually a single store of all enterprise data including raw copies of source system data and transformed data used for tasks such as reporting, visualization, advanced analytics and machine learning. A data lake can include structured data from relational databases (rows and columns), semi-structured data (CSV, logs, XML, JSON), unstructured data (emails, documents, PDFs) and binary data (images, audio, video). A data lake can be established "on premises" (within an organization's data centers) or "in the cloud" (using cloud services from vendors such as Amazon Web Services).

-- Wikipedia

**Serverless computing** is a cloud computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity. It can be a form of utility computing.

-- Wikipedia
Typical steps of building a data lake

1. Set up storage
2. Move data
3. Cleanse, prep, and catalog data
4. Configure and enforce security and compliance policies
5. Make data available for analytics
Defining the AWS data lake

Data lakes provide:

- Relational and nonrelational data
- Scale-out to Amazon EBS
- Diverse set of analytics and machine learning tools
- Work on data without any data movement
- Designed for low-cost storage and analytics
Why use AWS for big data & analytics?

- Agility
- Scalability
- Broadest and deepest capabilities
- Low cost
- Get to insights faster
- Data migrations made easy
Data lake on AWS

Catalog & search
- Amazon DynamoDB
- Amazon Elasticsearch Service (Amazon ES)
- AWS Glue

Central storage
- Scalable, secure, cost-effective
  - S3
- AWS Glue

Data ingestion
- AWS Snowball
- Amazon Kinesis Data Firehose
- AWS Direct Connect
- AWS Database Migration Service (AWS DMS)
- AWS Storage Gateway

Manage & secure
- AWS KMS
- IAM
- AWS CloudTrail
- Amazon CloudWatch

Access & user interfaces
- AWS AppSync
- Amazon API Gateway
- Amazon Cognito

Analytics & serving
- Amazon Athena
- Amazon EMR
- AWS Glue
- Amazon Redshift
- Amazon QuickSight
- Amazon Kinesis
- Amazon ES
- Amazon Neptune
- Amazon RDS
Modern serverless data lake components

Amazon S3

AWS Glue

AWS Lambda

Amazon CloudWatch Events
Amazon S3 is the best place for data lakes

- Unmatched durability, availability, and scalability
- Best security, compliance, and audit capabilities
- Object-level controls
- Business insights into your data
- Most ways to bring data in
A data lake needs to accommodate a wide variety of concurrent data sources

Rapidly ingest all data sources

- IoT, sensor data, clickstream data, social media feeds, streaming logs
- Oracle, MySQL, MongoDB, DB2, SQL Server, Amazon RDS
- On-premises ERP, mainframes, lab equipment, NAS storage
- Offline sensor data, NAS, on-premises Hadoop
- On-premises data lakes, EDW, large-scale data collection

Ingest methods:
- Kinesis Data Firehose
- AWS DMS
- Storage Gateway
- Snowball Edge
- DX

A data lake needs to accommodate a wide variety of concurrent data sources
AWS Transfer for SFTP

Fully managed service enabling transfer of data over SFTP while stored in Amazon S3

- Seamless migration of existing workflows
- Fully managed in AWS
- Native integration with AWS services
- Secure and compliant
- Cost-effective
- Simple to use
AWS DataSync

Transfer service that simplifies, automates, and accelerates data movement

Transfers up to 10 Gbps per agent
Simple data movement to Amazon S3 or Amazon EFS
Secure and reliable transfers
AWS integrated
Pay as you go

Combines the speed and reliability of network acceleration software with the cost-effectiveness of open-source tools

Migrate active application data to AWS
Transfer data for timely in-cloud analysis
Replicate data to AWS for business continuity
Choosing the right data formats

There is no such thing as the “best” data format

• All involve tradeoffs, depending on workload & tools
  • CSV, TSV, JSON are easy but not efficient
    • Compress & store or archive as raw input
  • Columnar compressed are generally preferred
    • Parquet or ORC
    • Smaller storage footprint = lower cost
    • More efficient scan & query
• Row-oriented (AVRO) good for full data scans
• Organize into partitions
• Coalescing to larger partitions over time

Key considerations are cost, performance, and support
Serverless ETL using AWS Glue
Data prep is ~80% of data lake work

- Building training sets
- Cleaning and organizing data
- Collecting datasets
- Mining data for patterns
- Refining algorithms
- Other
Set up a catalog, ETL, and data prep with AWS Glue

Serverless provisioning, configuration, and scaling to run your ETL jobs on Apache Spark

Pay only for the resources used for jobs

Crawl your data sources, identify data formats, and suggest schemas and transformations

Automates the effort in building, maintaining, and running ETL jobs
AWS Glue In Action
AWS Glue: Components

Data Catalog
- Hive metastore compatible with enhanced functionality
- Crawlers automatically extract metadata and create tables
- Integrated with Athena, Amazon Redshift Spectrum

Job Authoring
- Auto-generates ETL code
- Builds on open frameworks—Python and Spark
- Developer-centric—editing, debugging, sharing

Job Execution
- Runs jobs on a serverless Spark platform
- Provides flexible scheduling
- Handles dependency resolution, monitoring, and alerting
Manage table metadata through a Hive metastore API or Hive SQL. Supported by tools like Hive, Presto, Spark, etc.

We added a few extensions:

- **Search** over metadata for data discovery
- **Connection info**—JDBC URLs, credentials
- **Classification** for identifying and parsing files
- **Versioning** of table metadata as schemas evolve and other metadata are updated

Populate using Hive DDL, bulk import, or automatically through crawlers
AWS Glue Data Catalog: Crawlers

Crawlers automatically build your Data Catalog and keep it in sync

- Automatically discover new data, extract schema definitions
  - Detect schema changes and version tables
  - Detect Hive style partitions on Amazon S3

- Built-in classifiers for popular types; custom classifiers using Grok expressions

- Run ad hoc or on a schedule; serverless—only pay when crawler runs
Data Catalog: Detecting partitions

S3 bucket hierarchy

- sim=.93
  - month=Nov
    - sim=.99
      - date=10
      - sim=.95
        - date=15

- file 1
  - ... file N
  - sim=.95
    - file 1
      - ... file N

Table definition

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>str</td>
</tr>
<tr>
<td>date</td>
<td>str</td>
</tr>
<tr>
<td>col 1</td>
<td>int</td>
</tr>
<tr>
<td>col 2</td>
<td>float</td>
</tr>
</tbody>
</table>

Estimate schema similarity among files at each level to handle semi-structured logs, schema evolution...
Data Catalog: Table details

Table properties

Data statistics

Table schema
Job authoring in AWS Glue

- You have choices on how to get started
- Python code generated by AWS Glue
  - Connect a notebook or IDE to AWS Glue
  - Existing code brought into AWS Glue
1. Customize the mappings
2. AWS Glue generates transformation graph and **Python** code
3. Connect your **notebook** to development endpoints to customize your code
Job authoring: ETL code

- **Human-readable**, editable, and portable PySpark code

```python
from pyspark.sql import SparkSession

sc = SparkContext()
glueContext = GlueContext(sc)
job = Job(glueContext)
job.init(args=['JOB_NAME'], args)

# Type: DataSource
# Ori: [
#   name_space = "nytaxianalysis", table_name = "taxi\_hails4\_08", transformation_ctx = "datasource0"]
# @return: datasource0

datasource0 = glueContext.create_dynamic_frame.from_catalog(name_space = name_space, table_name = table_name, transformation_ctx = "datasource0")

RenameField0 = RenameField.apply(frame = datasource0, old_name = "trip\_pickup\_datetime", new_name = "pickup\_datetime", transformation_ctx = "RenameField0")

RenameField1 = RenameField.apply(frame = RenameField0, old_name = "trip\_dropoff\_datetime", new_name = "dropoff\_datetime", transformation_ctx = "RenameField1")

RenameField2 = RenameField.apply(frame = RenameField1, old_name = "cont\_code", new_name = "rate\_code", transformation_ctx = "RenameField2")
```

- **Flexible**: AWS Glue’s ETL library simplifies manipulating complex, semi-structured data

- **Customizable**: Use native PySpark, import custom libraries, and/or leverage AWS Glue’s libraries

- **Collaborative**: Share code snippets via GitHub, reuse code across jobs
Job authoring: AWS Glue Dynamic Frames

Like Spark’s Data Frames, but better for:
• Cleaning and (re)-structuring semi-structured data sets, e.g., JSON, Avro, Apache logs...

No upfront schema needed:
• Infers schema on the fly, enabling transformations in a single pass

Easy to handle the unexpected:
• Tracks new fields and inconsistent changing data types with choices, e.g., integer or string
• Automatically marks and separates error records
Job authoring: Leveraging the community

No need to start from scratch.

Use **AWS Glue samples** stored in GitHub to share, reuse, contribute: [https://github.com/awslabs/aws-glue-samples](https://github.com/awslabs/aws-glue-samples)

- Migration scripts to import existing Hive metastore data into AWS Glue Data Catalog
- Examples of how to use Dynamic Frames and Relationalize() transform
- Examples of how to use arbitrary PySpark code with AWS Glue’s Python ETL library

Download **AWS Glue’s Python ETL library** to start developing code in your IDE: [https://github.com/awslabs/aws-glue-libs](https://github.com/awslabs/aws-glue-libs)
Job execution: Scheduling and monitoring

Compose jobs globally with event-based dependencies

- Easy to reuse and leverage work across organization boundaries

Multiple triggering mechanisms

- **Schedule-based**: e.g., time of day
- **Event-based**: e.g., job completion
- **On-demand**: e.g., Lambda
- More: Amazon S3 notifications, and Amazon CloudWatch Events

Logs and alerts are available in CloudWatch
Job execution: Serverless

There is no need to provision, configure, or manage servers

- Auto-configure VPC and role-based access
- Customers can specify the capacity that gets allocated to each job
- Automatically scale resources (on post-GA roadmap)
- You pay only for the resources you consume while consuming them
Common customer use cases
Log aggregation with AWS Glue ETL

AWS Glue ETL

Update table partition
Create partition on Amazon S3

Glue Crawler

AWS Glue Data Catalog

AWS Service Logs
Web Application Logs
Server Logs

Amazon S3
Amazon S3

Query data

Amazon Athena
Real-Time data collection with Glue ETL

AWS Glue ETL

Create partition ON Amazon S3

Update table partition

Glue Crawler

Amazon S3

Query data

Amazon Athena

Real-time events

Amazon Kinesis

Store in

AWS Glue Data Catalog
Data import using Glue database connectors

AWS Glue ETL
Update table partition
AWS Glue Data Catalog

Database operation
AWS Glue JDBC Connector
Save data

AWS Glue ETL
Update table partition
AWS Glue Data Catalog

Amazon RDS
Amazon Redshift
Amazon S3
Amazon Athena
Serverless processing using Lambda
Benefits of Lambda

Productivity-focused compute platform to build powerful, dynamic, modular applications in the cloud

1. **No infrastructure to manage**
   - Focus on business logic

2. **Cost-effective and efficient**
   - Pay only for what you use

3. **Bring your own code**
   - Run code in standard languages
Application components for serverless apps

**EVENT SOURCE**
- Changes in data state
- Requests to endpoints
- Changes in resource state

**FUNCTION**
- Node
- Python
- Java
- … more coming soon

**SERVICES (ANYTHING)**
Event sources that integrate with Lambda

**DATA STORES**
- Amazon S3
- DynamoDB
- Kinesis
- Amazon Cognito
- Amazon RDS Aurora

**NEW**
- Amazon Aurora

**ENDPOINTS**
- Amazon Alexa
- API Gateway
- AWS IoT

**REPOSITORIES**
- AWS CloudFormation
- CloudTrail
- CloudWatch

**EVENT/MESSAGE SERVICES**
- Amazon SES
- Amazon SNS
- Cron events

**ORCHESTRATION AND STATE MANAGEMENT**
- AWS Step Functions

... and the list will continue to grow!
Lambda use case for streaming data ingestion

- **Lambda**: Transformations & enrichment
  - **Lookup**
  - **DynamoDB**: Lookup tables
  - **Amazon S3**: Buffered files
  - **Amazon Redshift**: Table loads
  - **Amazon ES**: Domain loads
  - **Amazon S3**: Source record backup

- **Amazon Kinesis Firehose**: Delivery stream
- **CloudWatch**: Delivery metrics

- **Record Producers**
- **Amazon Kinesis Agent**
- **Raw records** → **Transformed records**
Amazon Kinesis Streams and Lambda

- Number of Amazon Kinesis Streams **shards** corresponds to **concurrent invocations** of Lambda function

- **Batch size** sets maximum number of records per Lambda function invocation
Serverless data lake architecture
Serverless data lake architecture

Diagram showing integration with AWS services like S3, Lambda, CloudWatch, Glue, and SQS, illustrating the process flow.
Steps in building a serverless data lake

1. Ingest data into Amazon S3
2. Configure an Amazon S3 event trigger
3. Automate the Data Catalog with an AWS Glue crawler
4. Author ETL jobs
5. Automate ETL job execution
6. Monitor with CloudWatch Events
Serverless data lake blog post reference

Data lakes and analytics
More than 10,000 data lakes on AWS
AWS Partners

[Logos of various companies]
Thank you!

Aditya Challa
aditchal@amazon.com