Introduction to AWS Glue

AWS Glue automates the undifferentiated heavy lifting of ETL

Discover Automatically discover and categorize your data making it immediately searchable and queryable across data sources

Develop Generate code to clean, enrich, and reliably move data between various data sources; you can also use their favorite tools to build ETL jobs

Deploy Run your jobs on a serverless, fully managed, scale-out environment. No compute resources to provision or manage.

AWS Glue: Components

Data Catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon Redshift Spectrum

Job Authoring

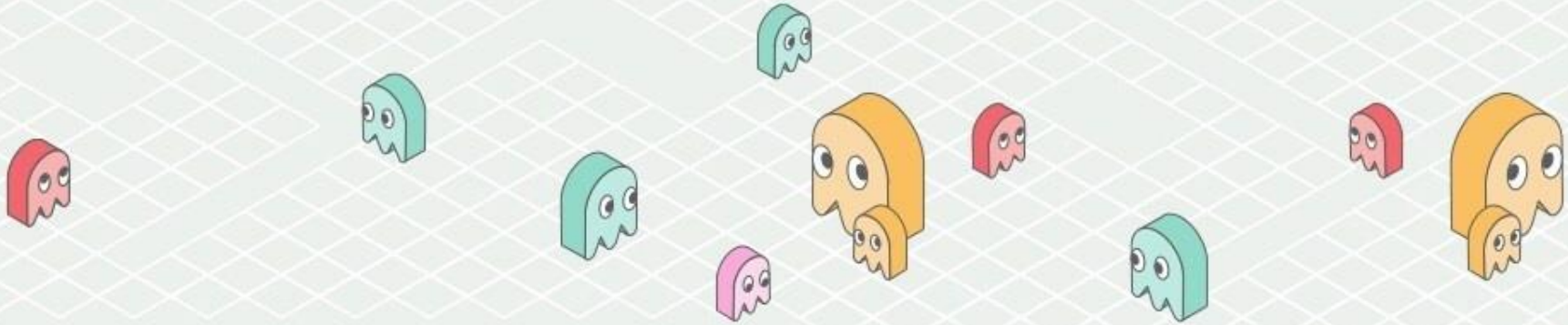
- Auto-generates ETL code
- Build on open frameworks – Python and Spark
- Developer-centric – editing, debugging, sharing

Job Execution

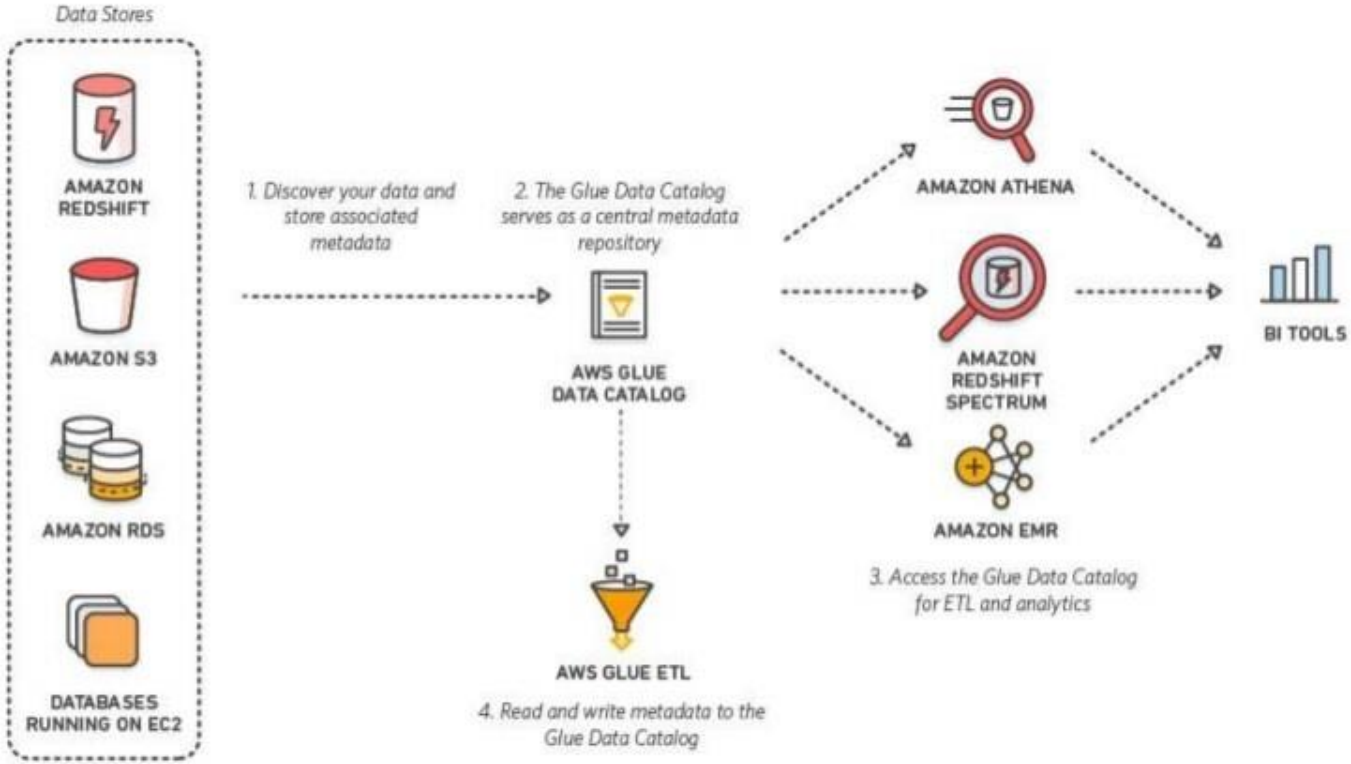
- Run jobs on a serverless Spark platform
- Provides flexible scheduling
- Handles dependency resolution, monitoring and alerting

amazon web services | Pop-up Loft LONDON

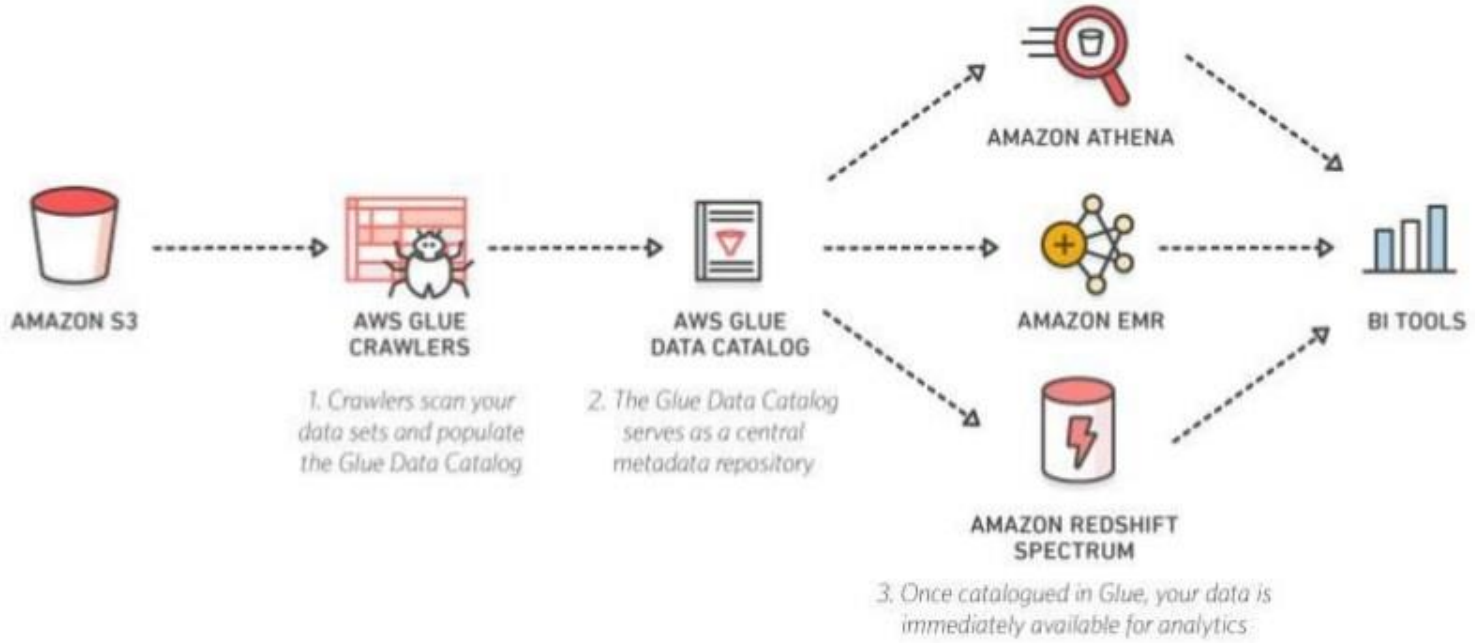
Common Use Cases for AWS Glue



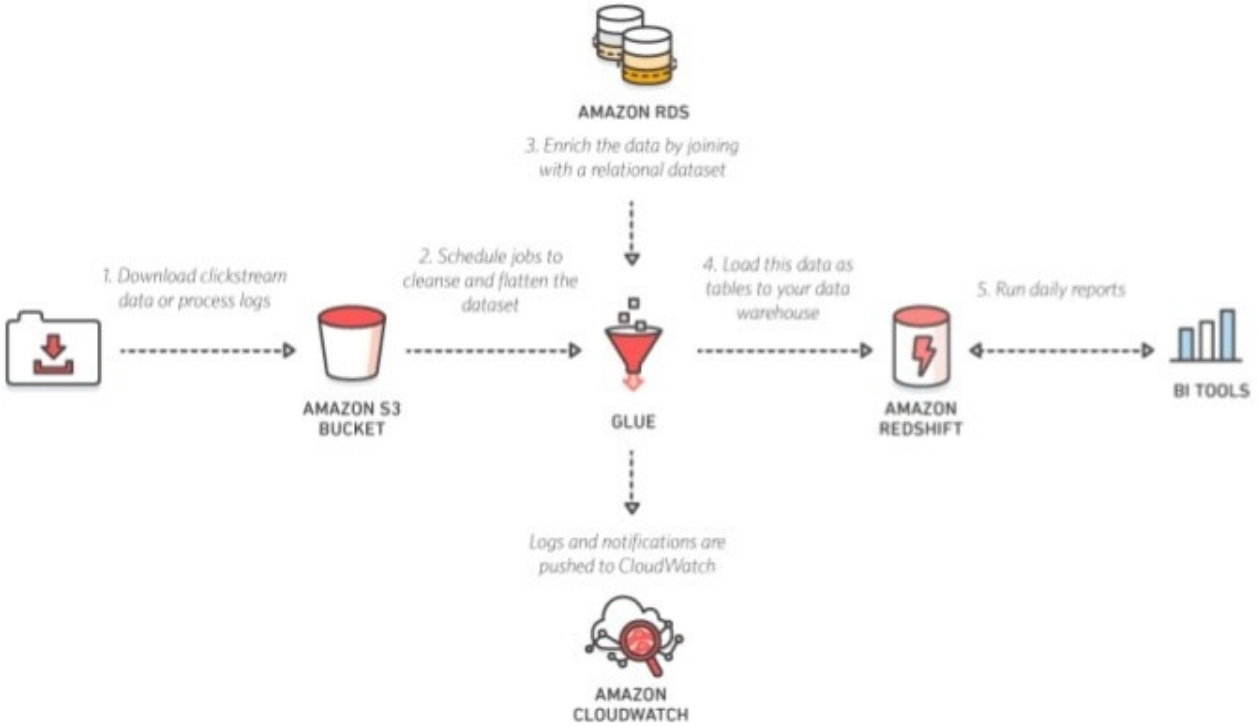
Understand your data assets



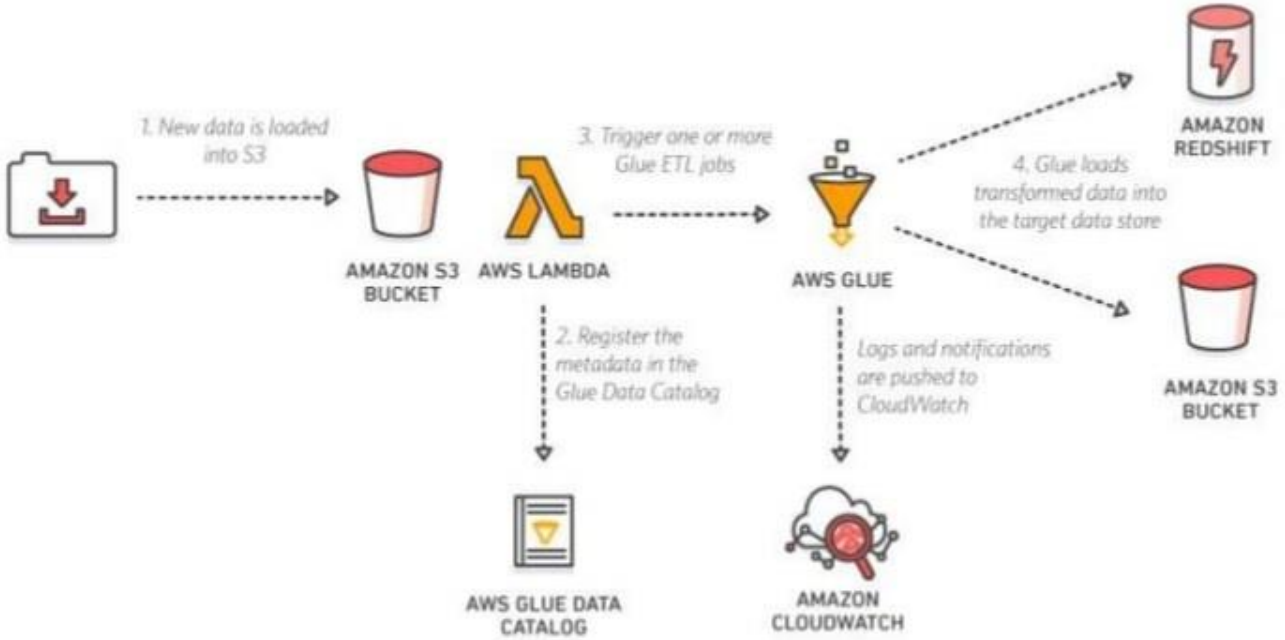
Instantly query your data lake on Amazon S3



ETL data into your data warehouse



Build event-driven ETL pipelines



amazon web services | Pop-up Loft LONDON

AWS Glue Data Catalog

Glue data catalog

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**.

Data Catalog: Crawlers

Crawlers automatically build your Data Catalog and keep it in sync

- Automatically discover new data, extracts 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

AWS Glue Data Catalog

Bring in metadata from a variety of data sources (Amazon S3, Amazon Redshift, etc.) into a single categorized list that is searchable

Data Catalog: Table details

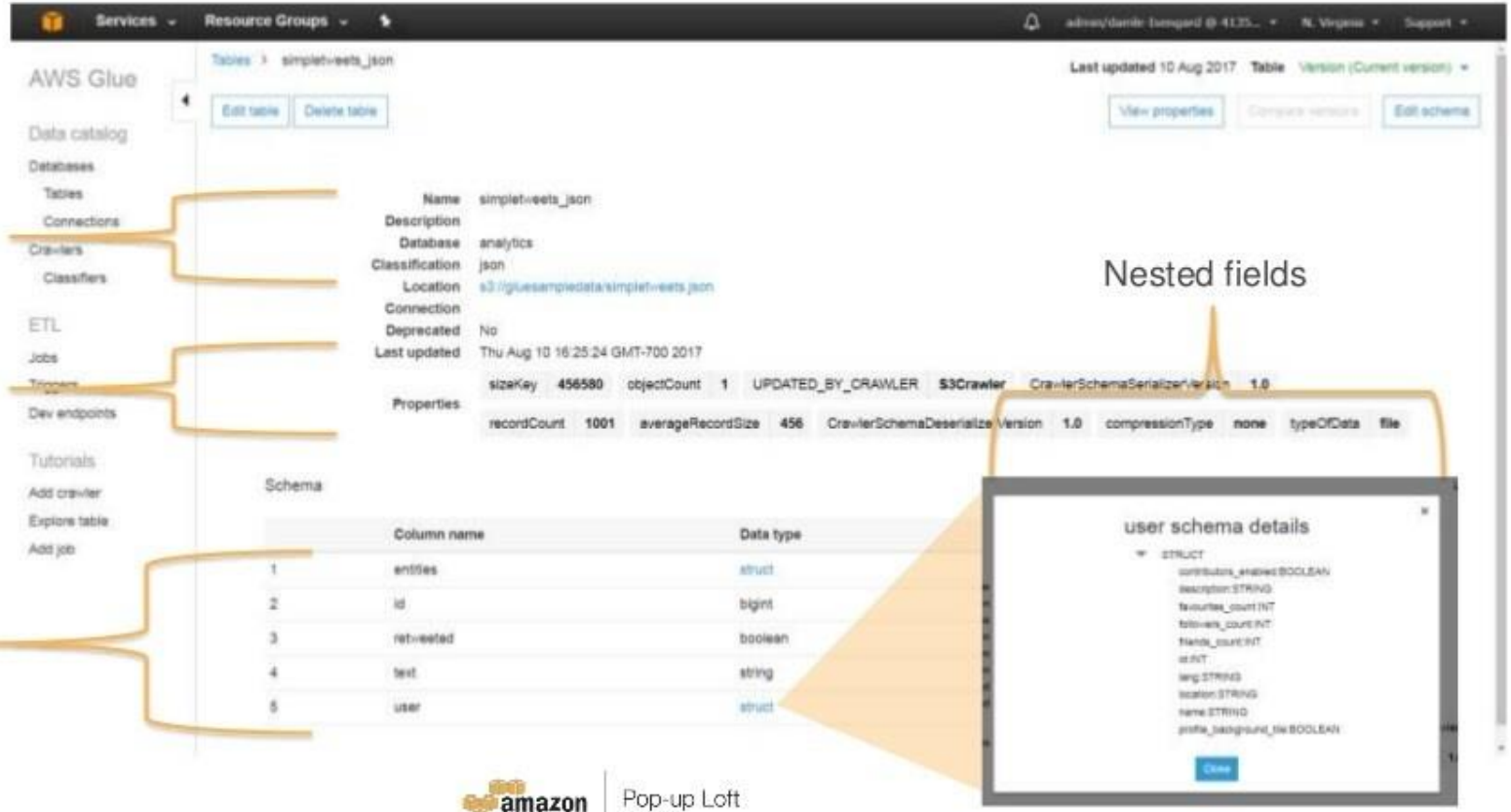


Table properties

Data statistics

Table schema

Data Catalog: Version control

Data Catalog: Detecting partitions

S3 bucket hierarchy

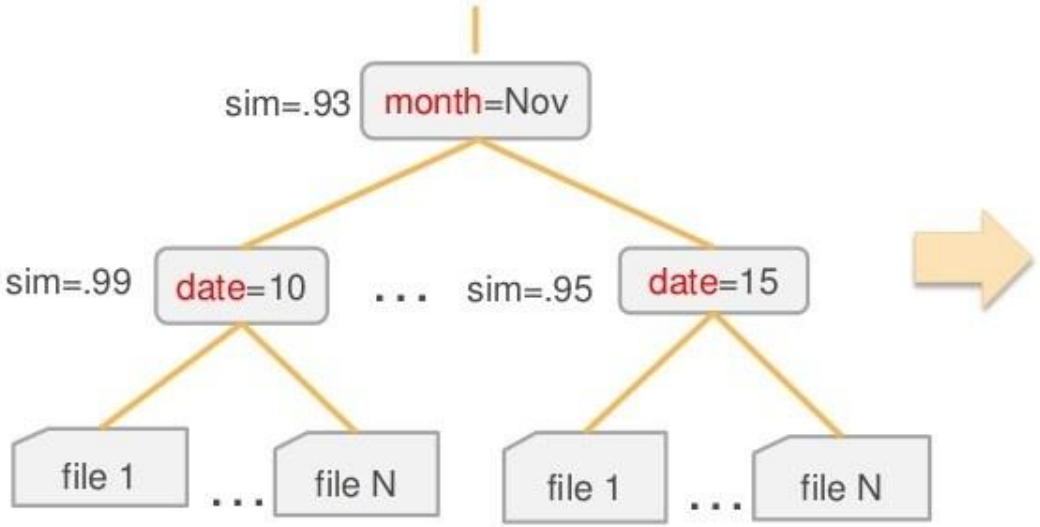


Table definition

Column	Type
month	str
date	str
col 1	int
col 2	float
⋮	⋮

Estimate schema similarity among files at each level to handle semi-structured logs, schema evolution…

Data Catalog: Automatic partition detection

12	errorcode	string	
13	region	string	Partition (0)
14	year	string	Partition (1)
15	month	string	Partition (2)
16	day	string	Partition (3)

Table partitions

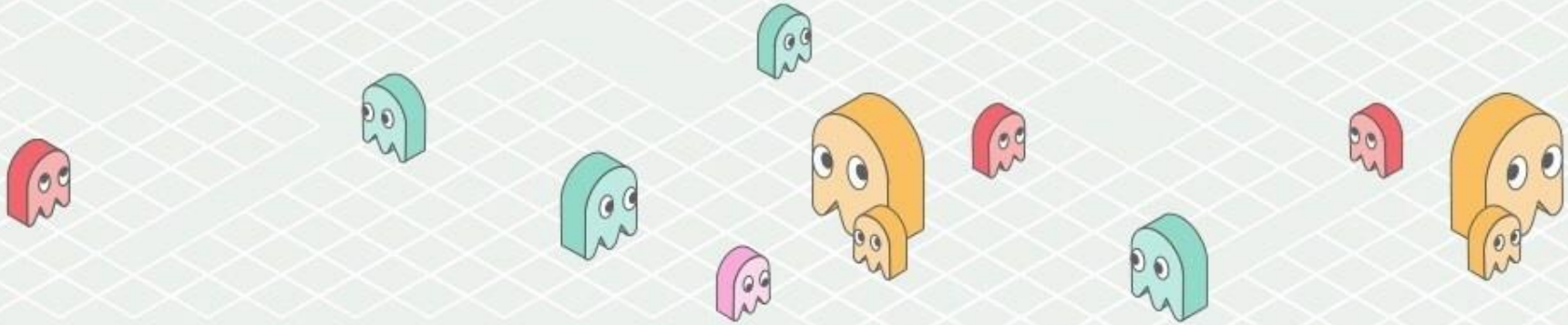
Automatically register available partitions

Showing: 1 - 3

region	year	month	day		
us-west-2	2016	02	18	View files	View properties
us-west-2	2016	02	17	View files	View properties
us-west-2	2016	02	16	View files	View properties

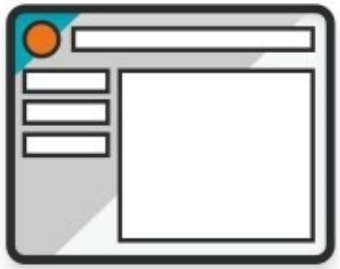
amazon web services | Pop-up Loft LONDON

Job Authoring in AWS Glue



Job Authoring Choices

- Python code generated by AWS Glue
- Connect a notebook or IDE to AWS Glue
- Existing code brought into AWS Glue



Job authoring: Automatic code generation

1. Customize the mappings
2. 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

```
sc = SparkContext()
glueContext = GlueContext(sc)
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [name_space = "nytaxianalysis", table_name = "taxi_303e40bd", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(name_space = namespace, table_name = tablename, transformation_ctx = "datasource0")
RenameField0 = RenameField.apply(frame = datasource0, old_name="lpep_pickup_datetime", new_name="pickup_datetime", transformation_ctx = "RenameField0")
RenameField1 = RenameField.apply(frame = RenameField0, old_name="lpep_dropoff_datetime", new_name="dropoff_datetime", transformation_ctx = "RenameField1")
RenameField2 = RenameField.apply(frame = RenameField1, old_name="ratecodeid", new_name="ratecode", transformation_ctx = "RenameField2")
```

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

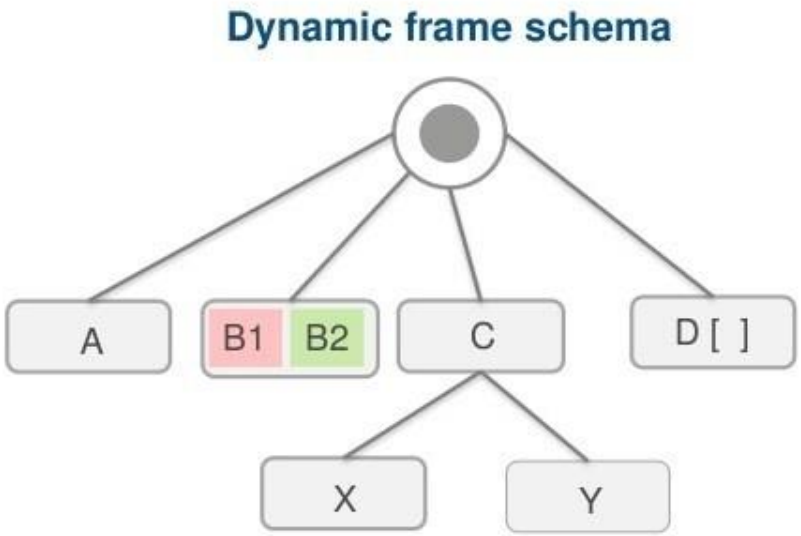
- **Customizable:** Use native PySpark, import custom libraries, and/or leverage Glue's libraries

```
##################################################
##
##  PySpark Logic to do lots of custom stuff...
##
##################################################
DataFrame0 = DynamicFrame.toDF(SelectFields0)

DataFrame0 = DataFrame0.withColumn("pickup_datetime", DataFrame0["pickup_datetime"].cast("timestamp"))
DataFrame0 = DataFrame0.withColumn("dropoff_datetime", DataFrame0["dropoff_datetime"].cast("timestamp"))
DataFrame0 = DataFrame0.withColumn("type", lit(recordtype))
```

- **Collaborative:** share code snippets via GitHub, reuse code across jobs

Job Authoring: 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**

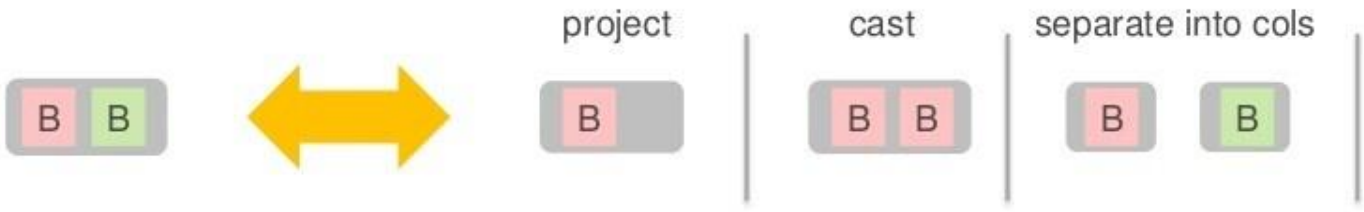
Easy to handle the unexpected:

- Tracks new fields, and inconsistent changing data types with **choices**, e.g. integer or string
- Automatically mark and separate error records

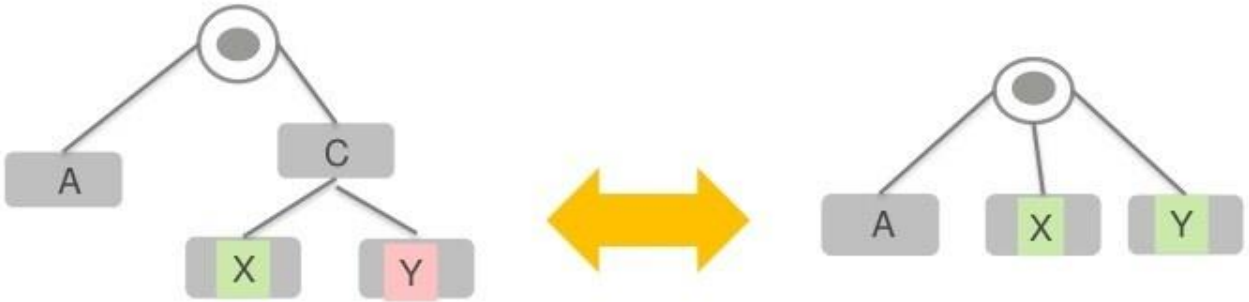
Job Authoring: Glue transforms

Adaptive and flexible

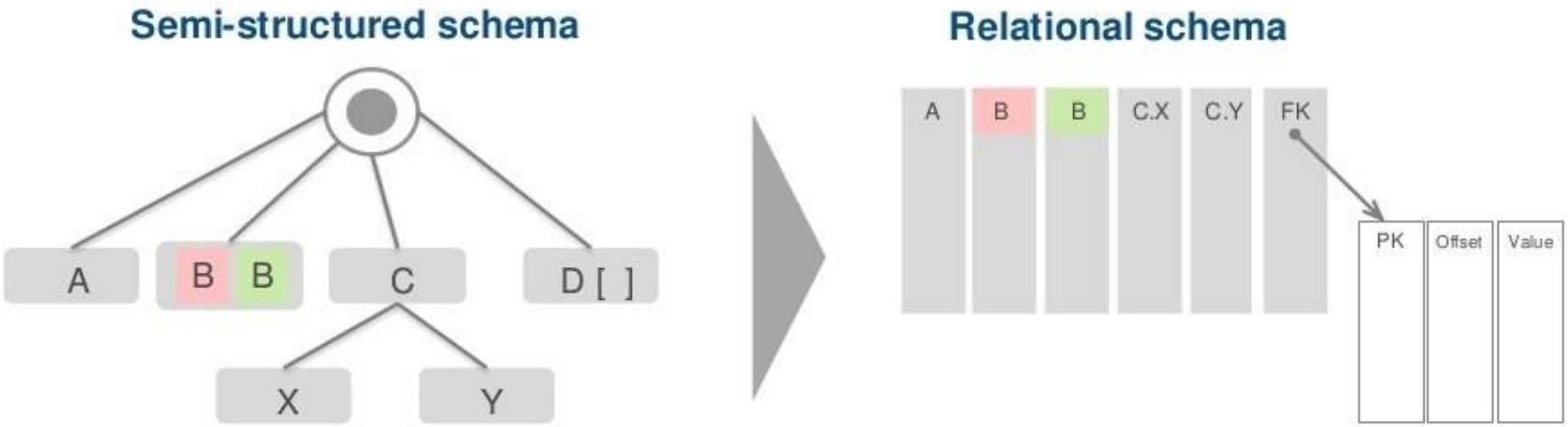
ResolveChoice()



Apply Mapping()



Job authoring: Relationalize() transform



- Transforms and **adds new** columns, types, and tables on-the-fly
- Tracks **keys** and **foreign keys** across runs
- SQL on the relational schema is orders of **magnitude faster** than JSON processing

Job authoring: Glue transformations

Add transform

	Name	Description
○	DropFields	Drop fields from a DynamicFrame
○	DropNullFields	DynamicFrame without null fields
○	Join	Join two DynamicFrames
○	MapToCollection	Apply a transform to each DynamicFrame in this DynamicFrameCollection
○	Relationalize	Flatten nested schema and pivot out array columns from the flattened frame
○	RenameField	Rename a field within a DynamicFrame
○	SelectFields	Select fields from a DynamicFrame
○	SelectFromCollection	Select one DynamicFrame from a DynamicFrameCollection
○	SplitFields	Split fields within a DynamicFrame
○	SplitRows	Split rows within a DynamicFrame based on comparators
○	Unbox	Unbox a string field

- **Prebuilt transformation:** Click and add to your job with simple configuration
- **Spigot** writes sample data from DynamicFrame to S3 in JSON format
- **Expanding…** more transformations to come

Job authoring: Write your own scripts

fromDF

`fromDF(dataframe, glue_ctx, name)`

Converts a DataFrame to a DynamicFrame by converting DynamicRecords to Rows. Returns the new DynamicFrame.

- `dataframe` – The spark SQL dataframe to convert. Required
- `glue_ctx` – The GlueContext (p. 164) object that specifies the context for this transform. Required.
- `name` – The name of the resulting DynamicFrame. Required

toDF

`toDF(options)`

Converts a DynamicFrame to an Apache DataFrame. Returns the new DataFrame.

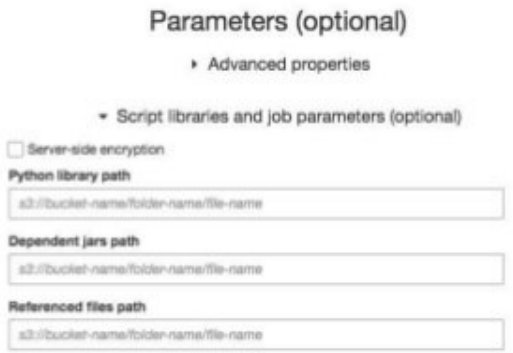
- `options` – A list of options. Please specify the target type if you choose the Project and Cast action type. Examples are:

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```

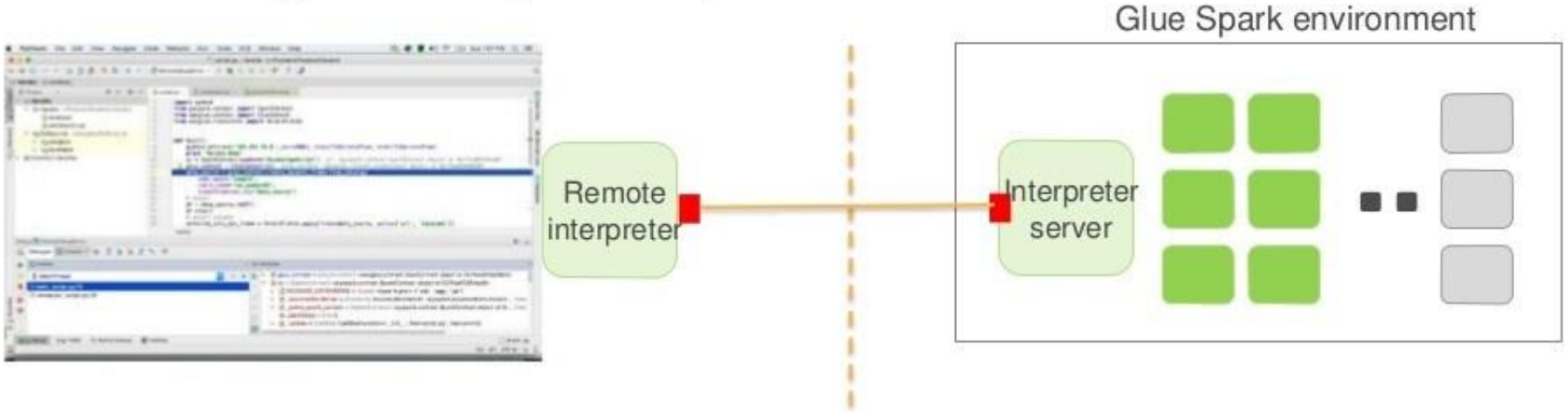
Convert to a Spark Data Frame
for complex SQL-based ETL

Convert back to Glue Dynamic Frame
for semi-structured processing and AWS Glue connectors

Import custom libraries required by your code



Job authoring: Developer endpoints



- Environment to **iteratively develop** and test ETL code.
- Connect your IDE or notebook (e.g. **Zeppelin**) to a Glue development endpoint.
- When you are satisfied with the results you can create an ETL job that runs your code.

Job Authoring: Leveraging the community

No need to start from scratch.

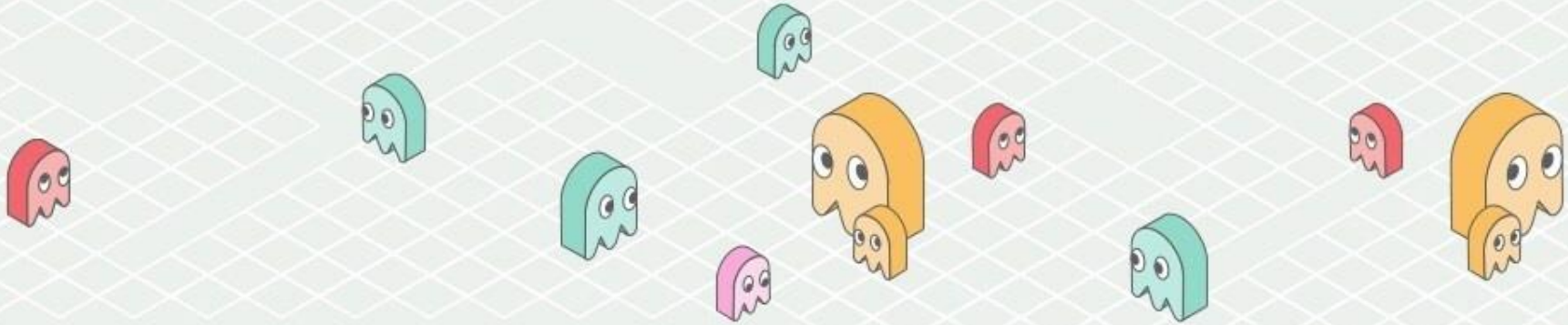
Use **Glue samples** stored in Github to share, reuse, contribute: https://github.com/awslabs/aws-glue-samples

- 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 Glue's Python ETL library

Download **Glue's Python ETL library** to start developing code in your IDE: https://github.com/awslabs/aws-glue-libs

amazon web services | Pop-up Loft LONDON

Orchestration and Job Scheduling



Job execution: Scheduling and monitoring

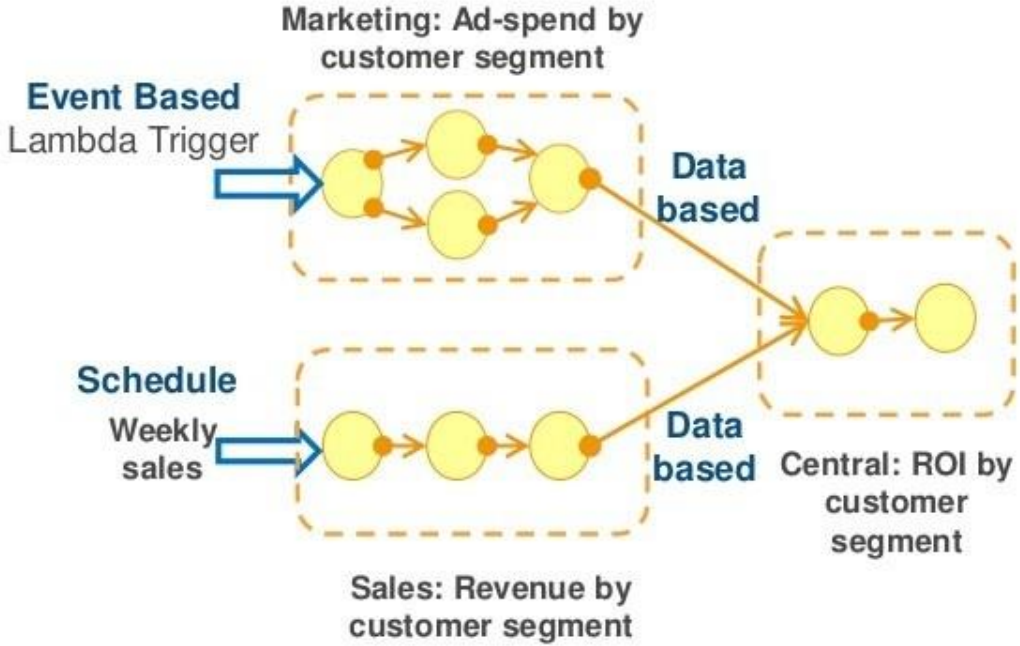
Compose jobs globally with event-based dependencies

- 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., AWS Lambda
- **More coming soon:** Data Catalog based events, S3 notifications and Amazon CloudWatch events

Logs and alerts are available in Amazon CloudWatch

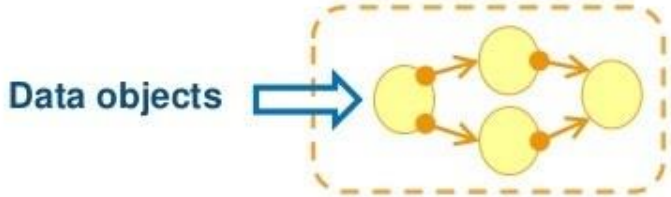


Job execution: Job bookmarks

Glue keeps track of data that has already been processed by a previous run of an ETL job. This persisted state information is called a **bookmark.**

Option	Behavior
Enable	Pick up from where you left off
Disable	Ignore and process the entire dataset every time
Pause	Temporarily disable advancing the bookmark

For example, you get new files everyday in your S3 bucket. By default, AWS Glue keeps track of which files have been successfully processed by the job to prevent data duplication.

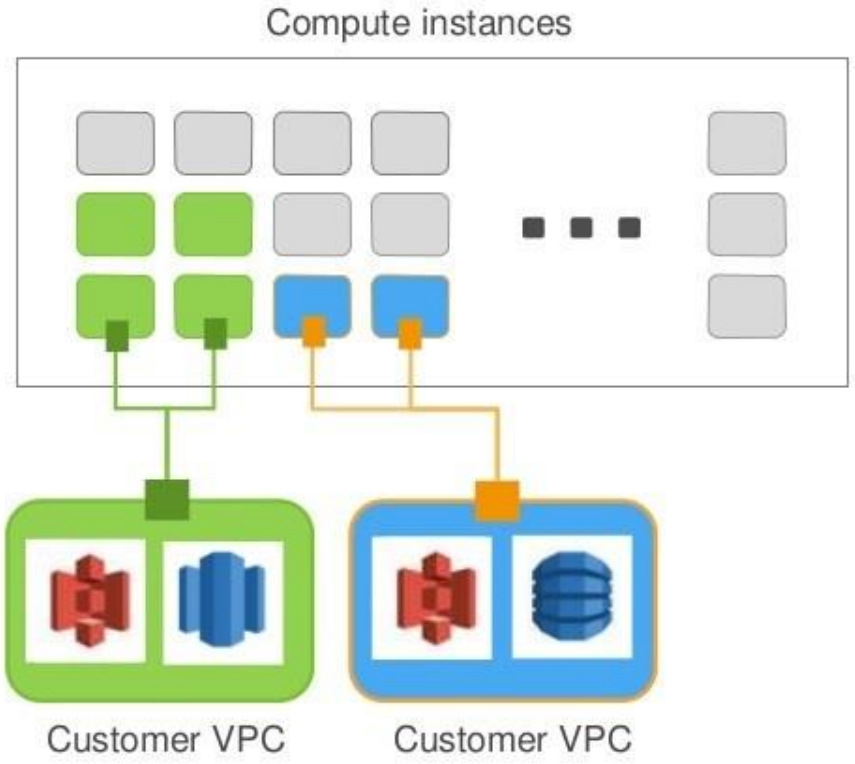


Marketing: Ad-spend by customer segment

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



Introduction to Amazon Athena

What to Expect from the Session

Introduction to Serverless

Big data challenges

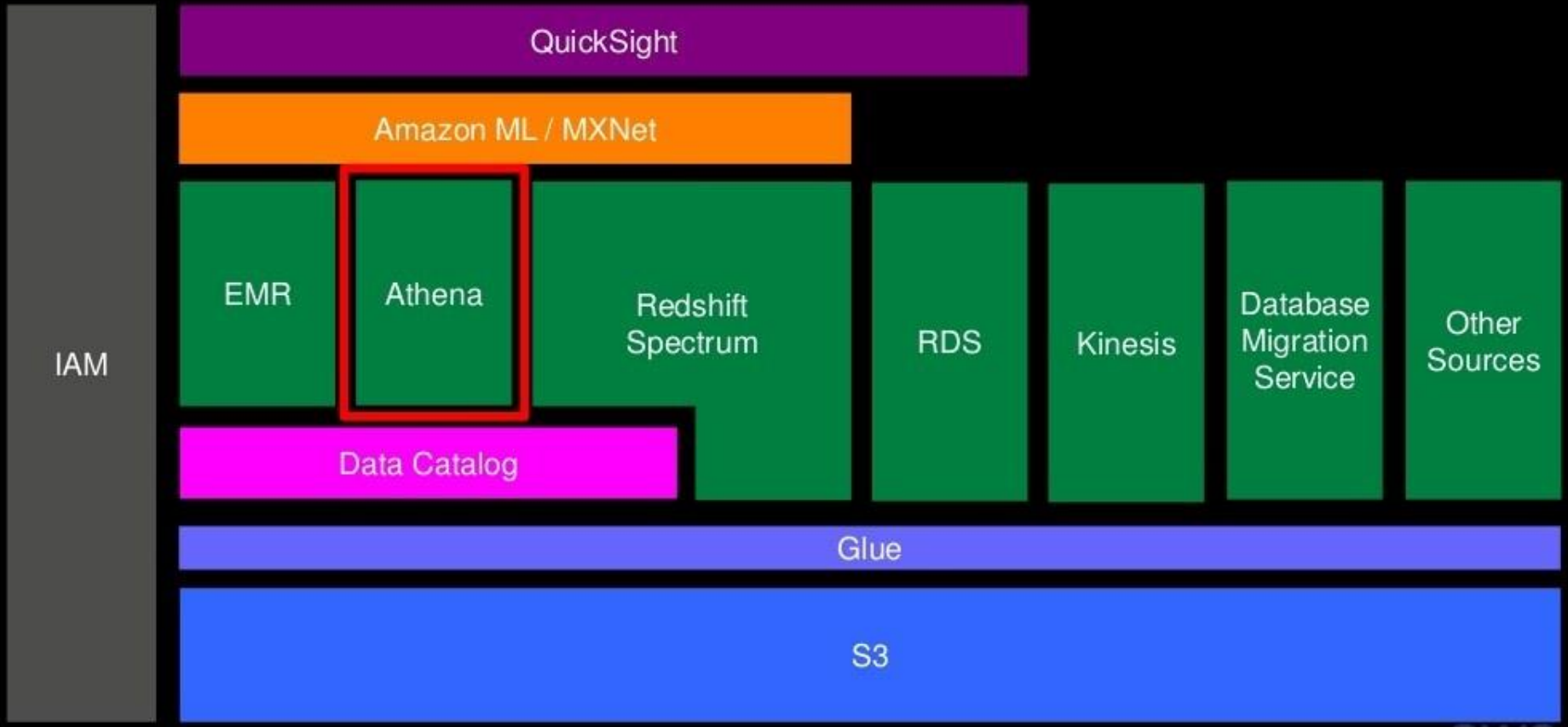
Introduction to Athena

Demo

Customer Use Cases

aws

Amazon Analytics End to End Architecture



© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Cloud Services Evolution

Virtual machines

Managed services

Serverless

Serverless

Serverless characteristics

No servers to provision or manage

Scales with usage

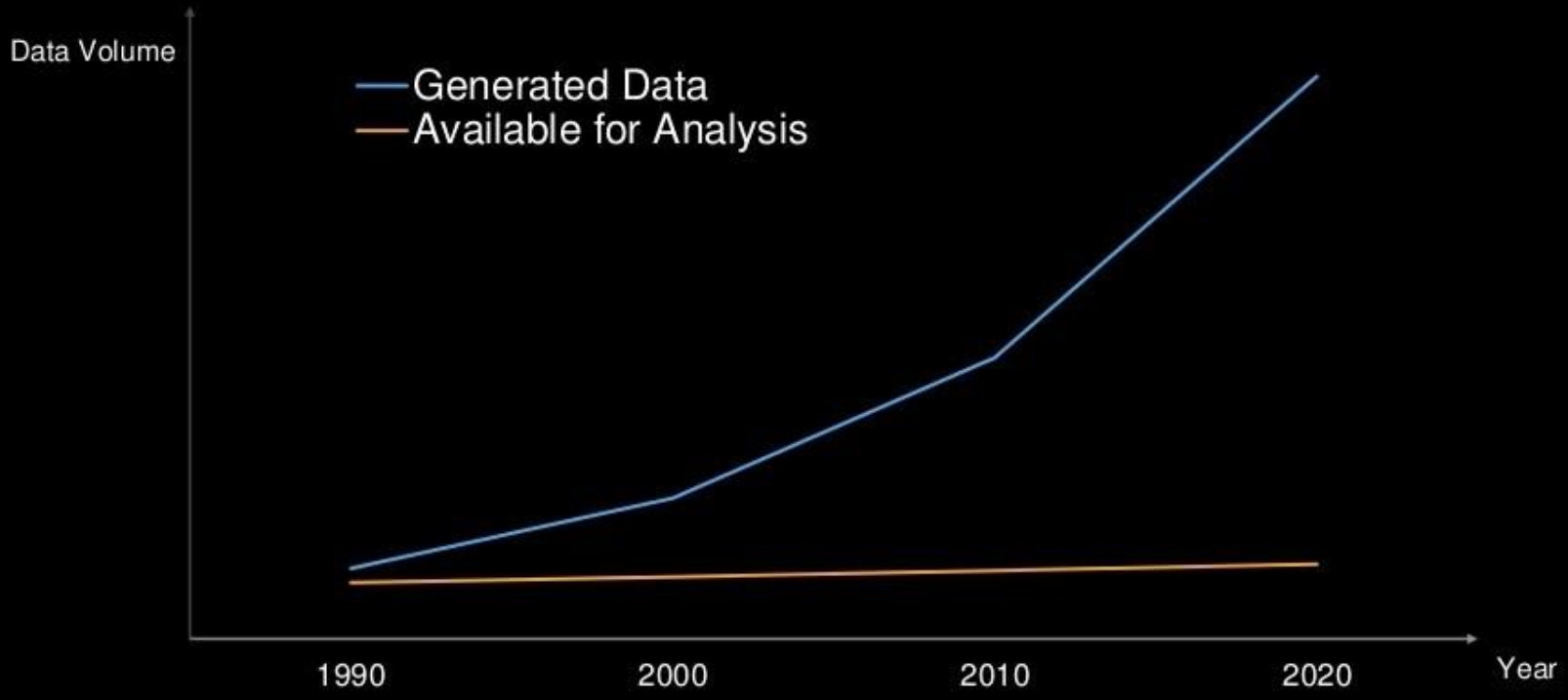
Never pay for idle

Availability and fault tolerance built in

aws

The Dark Data Problem

Most generated data is unavailable for analysis



Sources:
Gartner: User Survey Analysis: Key Trends Shaping the Future of Data Center Infrastructure Through 2011
IDC: Worldwide Business Analytics Software 2012–2016 Forecast and 2011 Vendor Shares

aws

Amazon Athena

Amazon Athena is an interactive query service that makes it easy to analyze data directly from Amazon S3 using Standard SQL

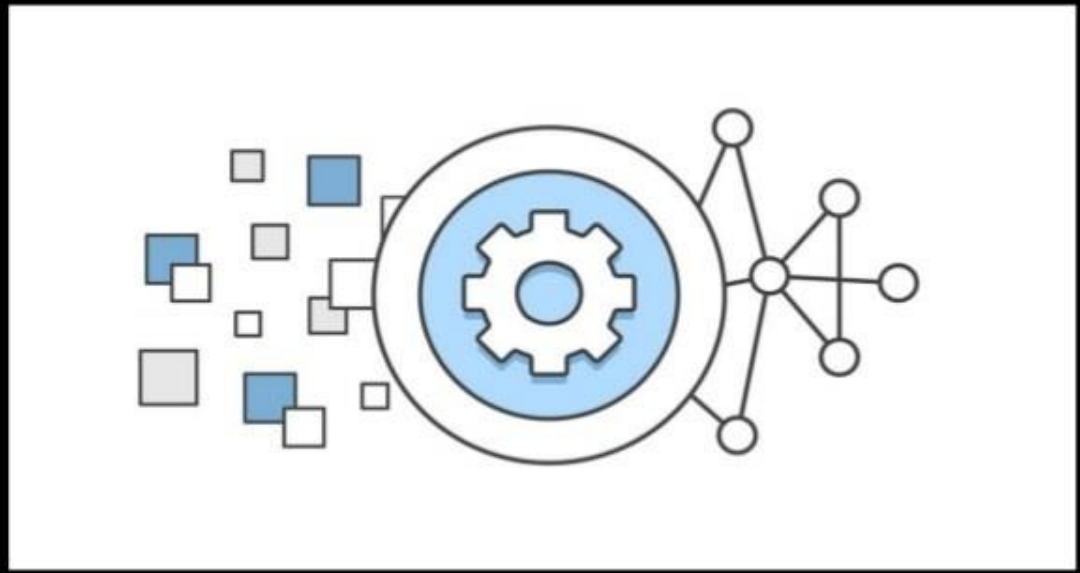
Value Proposition

- Decouple storage from compute
- Serverless – No infrastructure or resources to manage
- Pay only for data scanned
- Schema on read – Same data, many views
- Secure – IAM for authentication; Encryption at rest
- Standard compliant and open storage formats
- Built on powerful community supported OSS solutions

aws

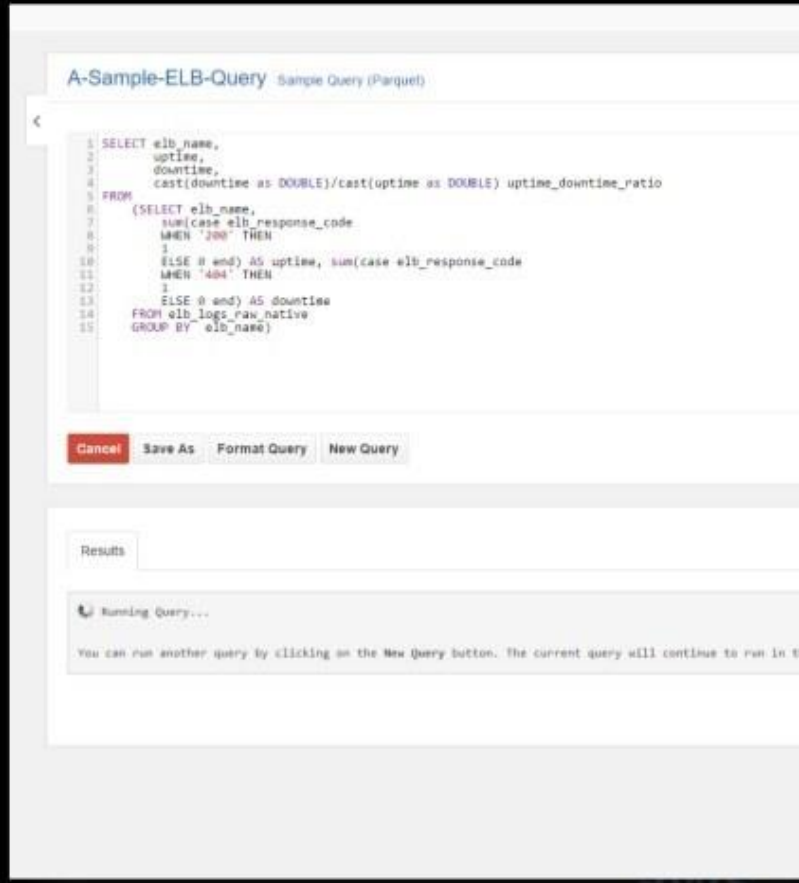
Athena is Serverless

- No Infrastructure or administration
- Zero Spin up time
- Transparent upgrades



Easy To Use

- Log into the Console
- Create a table
  - Type in a Hive DDL Statement
  - Use Glue Data Catalog
  - Use the console Add Table wizard
- Start querying



Highly Available

- You connect to a service endpoint or log into the console
- Athena uses warm compute pools across multiple Availability Zones
- Your data is in Amazon S3, which is also highly available and designed for 99.999999999% durability

aws

Query Data Directly from Amazon S3

- No loading of data
- Query data in its raw format
  - Text, CSV, JSON, weblogs, AVRO, AWS service logs
  - Convert to an optimized form like ORC or Parquet for the best performance and lowest cost
- No ETL required (but it can help!)
- Stream data from directly from Amazon S3
- Take advantage of Amazon S3 durability and availability

aws

Simple Pricing

- DDL operations – FREE
- SQL operations – FREE
- Query concurrency – FREE
- Data scanned - $5 / TB
- Standard S3 rates for storage, requests, and data transfer apply

Familiar Technologies Under the Covers

Used for DDL functionality

Complex data types

Multitude of formats

Supports data partitioning

presto

Used for SQL Queries

In-memory distributed query engine

ANSI-SQL compatible with extensions

aws

Presto SQL

- ANSI SQL compliant
- Complex joins, nested queries & window functions
- Complex data types (arrays, structs, maps)
- Partitioning of data by any key
  - date, time, custom keys
- Presto built-in functions

```
WITH q21_tmp1_cached AS
 (SELECT l_orderkey,
         count(DISTINCT l_suppkey) AS count_suppkey,
         max(l_suppkey) AS max_suppkey
  FROM lineitem_parq
  WHERE l_orderkey IS NOT NULL
  GROUP BY l_orderkey),
     q21_tmp2_cached AS
 (SELECT l_orderkey,
         count(DISTINCT l_suppkey) count_suppkey,
                                   max(l_suppkey) AS max_suppkey
  FROM lineitem_parq
  WHERE l_receiptdate > l_commitdate
    AND l_orderkey IS NOT NULL
  GROUP BY l_orderkey)
SELECT s_name,
       count(1) AS numwait
FROM
  (SELECT s_name
   FROM
     (SELECT s_name,
             t2.l_orderkey,
             l_suppkey,
             count_suppkey,
             max_suppkey
      FROM q21_tmp2_cached t2
      RIGHT OUTER JOIN
        (SELECT s_name,
                l_orderkey,
                l_suppkey
         FROM
           (SELECT s_name,
                   t1.l_orderkey,
                   l_suppkey,
                   count_suppkey,
                   max_suppkey
            FROM q21_tmp1_cached t1
            JOIN
              (SELECT s_name,
                      l_orderkey,
                      l_suppkey
               FROM orders_parq o
               JOIN
                 (SELECT s_name,
                         l_orderkey,
                         l_suppkey
                  FROM nation_parq n
                  JOIN supplier s ON s.s_nationkey = n.n_nationkey
                  AND n.n_name = 'SAUDI ARABIA'
                  JOIN lineitem_parq l ON s.s_suppkey = l.l_suppkey
                  WHERE l.l_receiptdate > l.l_commitdate
                    AND l.l_orderkey IS NOT NULL) l1 ON o.o_orderkey = l1.l_orderkey
               AND o.o_orderstatus = 'F') l2 ON l2.l_orderkey = t1.l_orderkey) a
         WHERE (count_suppkey > 1)
           OR ((count_suppkey=1)
               AND (l_suppkey <> max_suppkey))) l3 ON l3.l_orderkey = t2.l_orderkey) b
   WHERE (count_suppkey IS NULL)
     OR ((count_suppkey=1)
         AND (l_suppkey = max_suppkey))) c
GROUP BY s_name
ORDER BY numwait DESC,
         s_name LIMIT 100;
```

aws

Simple Pricing - $5/TB Scanned

- Pay by the amount of data scanned per query
- Ways to save costs
  - Compress
  - Convert to Columnar format
  - Use partitioning
- Free: DDL Queries, Failed Queries

```
SELECT elb_name,
       uptime,
       downtime,
       cast(downtime as DOUBLE)/cast(uptime as DOUBLE) uptime_downtime_ratio
FROM
    (SELECT elb_name,
        sum(case elb_response_code
        WHEN '200' THEN
        1
        ELSE 0 end) AS uptime, sum(case elb_response_code
        WHEN '404' THEN
        1
        ELSE 0 end) AS downtime
    FROM elb_logs_raw_native
    GROUP BY  elb_name)
```

Dataset	Size on Amazon S3	Query Run time	Data Scanned	Cost
Logs stored as Text files	1 TB	237 seconds	1.15TB	$5.75
Logs stored in Apache Parquet format*	130 GB	5.13 seconds	2.69 GB	$0.013
Savings	**87% less with Parquet**	**34x faster**	**99% less data scanned**	**99.7% cheaper**

CloudTrail

- CloudTrail tracks all Athena APIs
- Can be queried by Athena
- Includes IAM user and account ID of whom executed queries
- Helpful in tracking query usage

aws

Athena or Redshift Spectrum (or both)?

- **Athena**
  - Separation of storage and compute
  - Good query performance
  - Text and optimized file formats – help reduce cost and improve performance
  - Serverless

- **Redshift Spectrum**
  - Best cost/query – Best query performance
  - Data Warehouse capabilities
  - Join Redshift and S3 data
  - Cluster management overhead

aws

How to connect

AWS Glue Data Catalog

Discover and organize your data

aws

Glue data catalog

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**.

Data Catalog: Crawlers

Crawlers automatically build your Data Catalog and keep it in sync

- Automatically discover new data, extracts 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

aws

Amazon QuickSight

Amazon QuickSight is a Business Analytics Service that lets business users quickly and easily visualize, explore, and share insights from their data.

Who Is QuickSight For?

QuickSight is designed to give everyday business users the **flexibility** to do easy, self-serve analysis on their data.

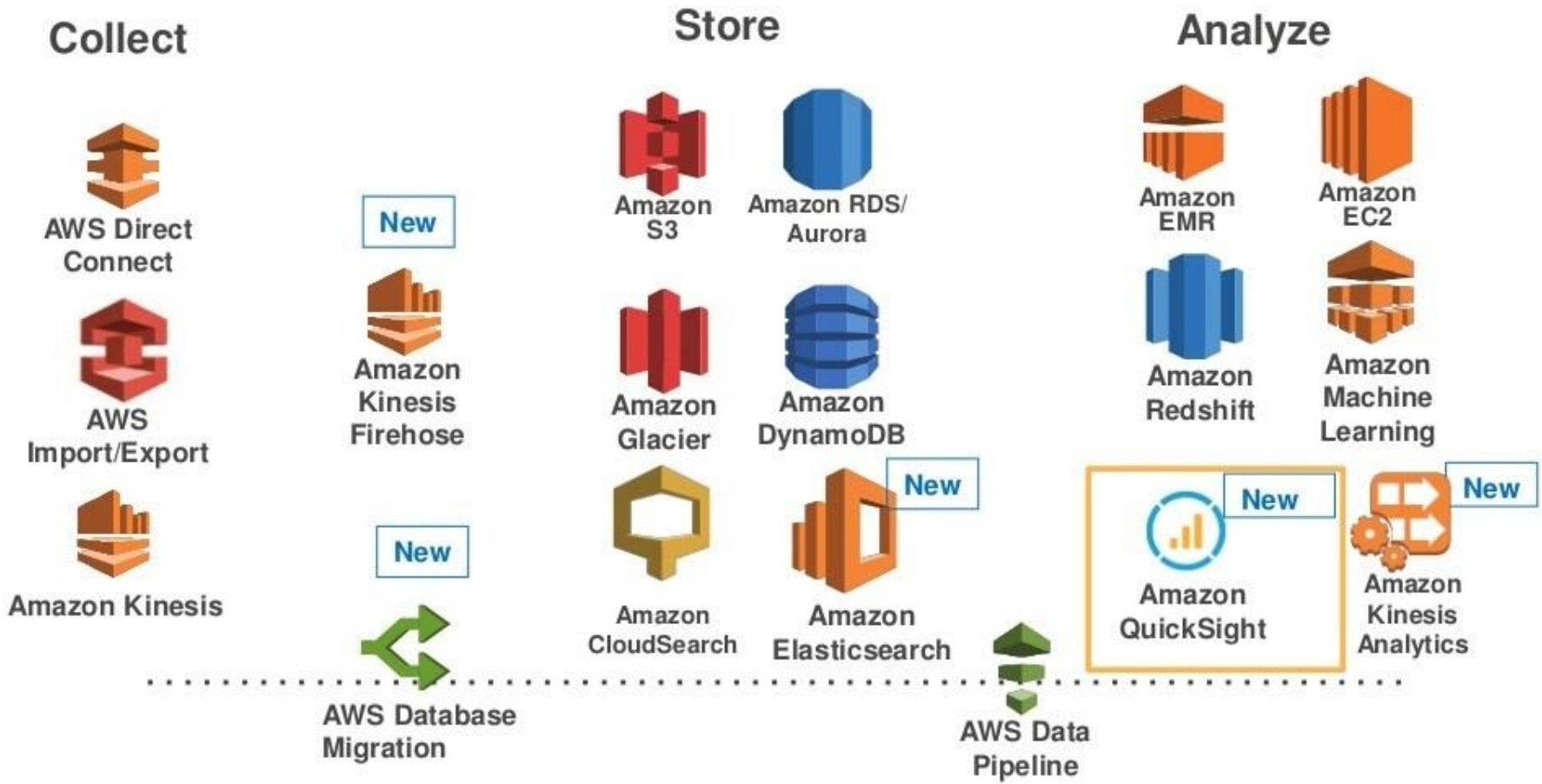
QuickSight is also **perfect** for delivering published dashboards throughout the organization.

Deep Integration with AWS Data Sources

QuickSight is **deeply integrated** with AWS data sources like Redshift, RDS, S3, Athena and others, as well as third-party sources like Excel, Salesforce, as well as on-premises databases.

Getting Started with Amazon QuickSight

AWS Big Data portfolio



What are the Big Data challenges our customers face?

How is my marketing campaign performing?

Which devices are showing time for maintenance?

Why is my most profitable region not growing?

Who are my top customers and what are they buying?

Lots of data

Lots and lots of questions

Few insights

How much inventory do I have?

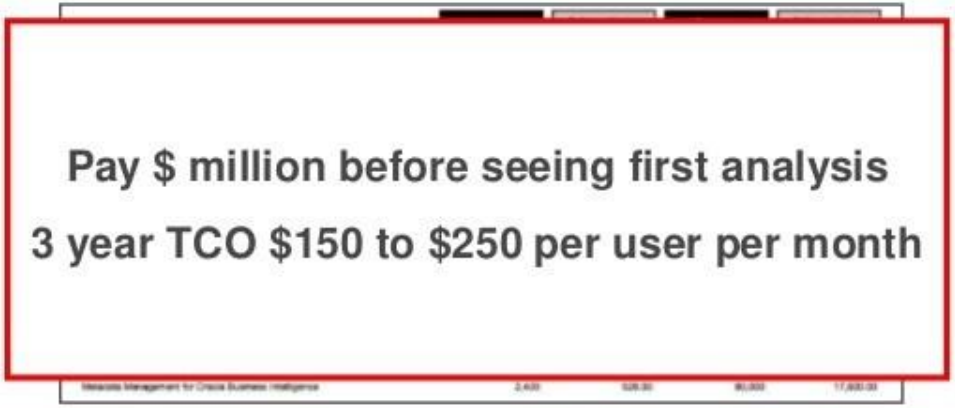
How is my employee satisfaction trending?

What is my product profitability by region?

Has my fraud account expense increased?

Old-guard BI

Costs Too Much



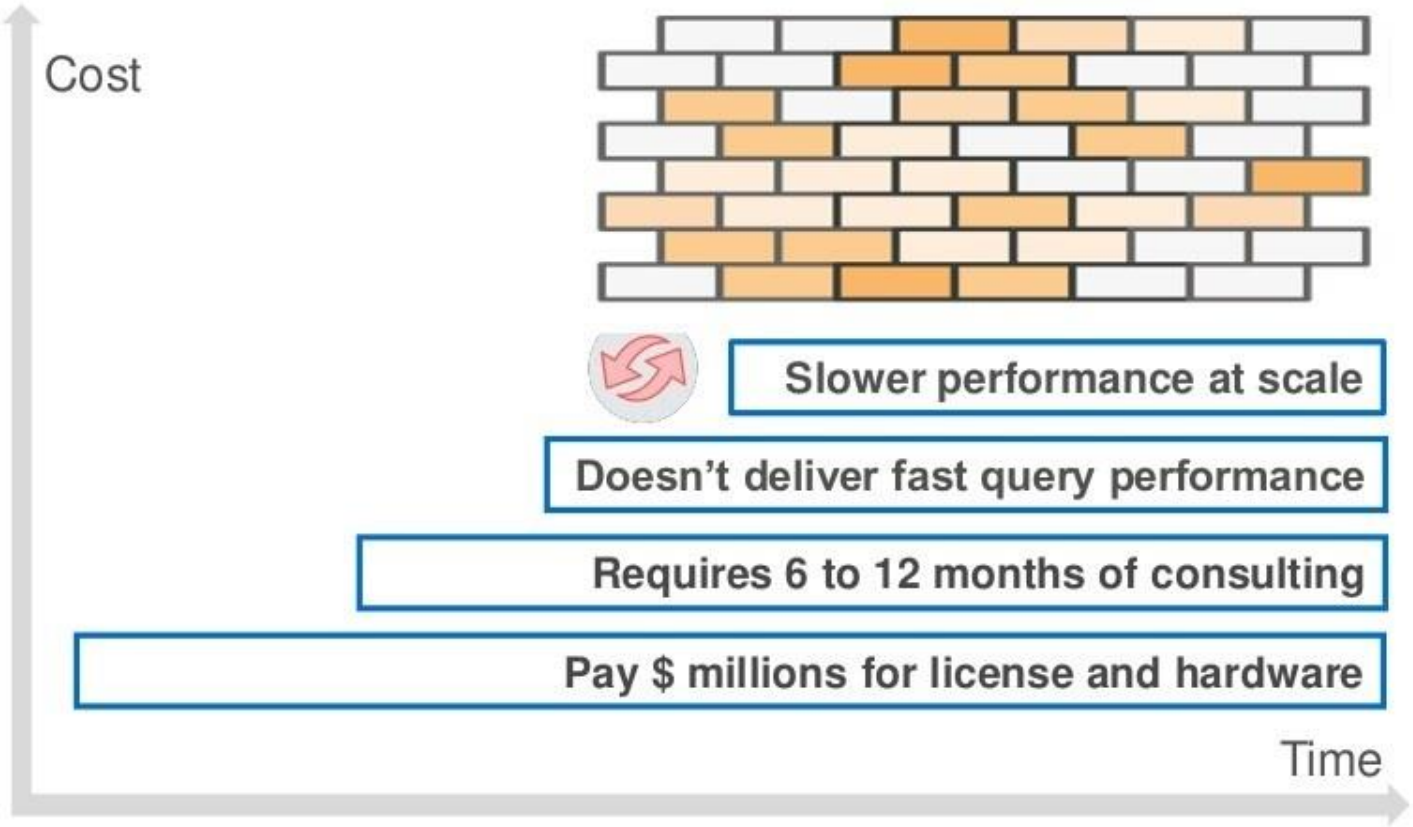
Pay $ million before seeing first analysis

3 year TCO $150 to $250 per user per month

Takes Too Long

Spend 6 to 12 months of consulting and SW implementation time

Old-guard BI

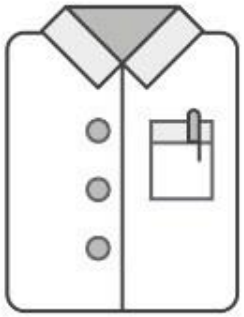

Cost
Slower performance at scale
Doesn't deliver fast query performance
Requires 6 to 12 months of consulting
Pay $ millions for license and hardware
Time

Freedom to get the real value out of the data you have

Introducing Amazon QuickSight

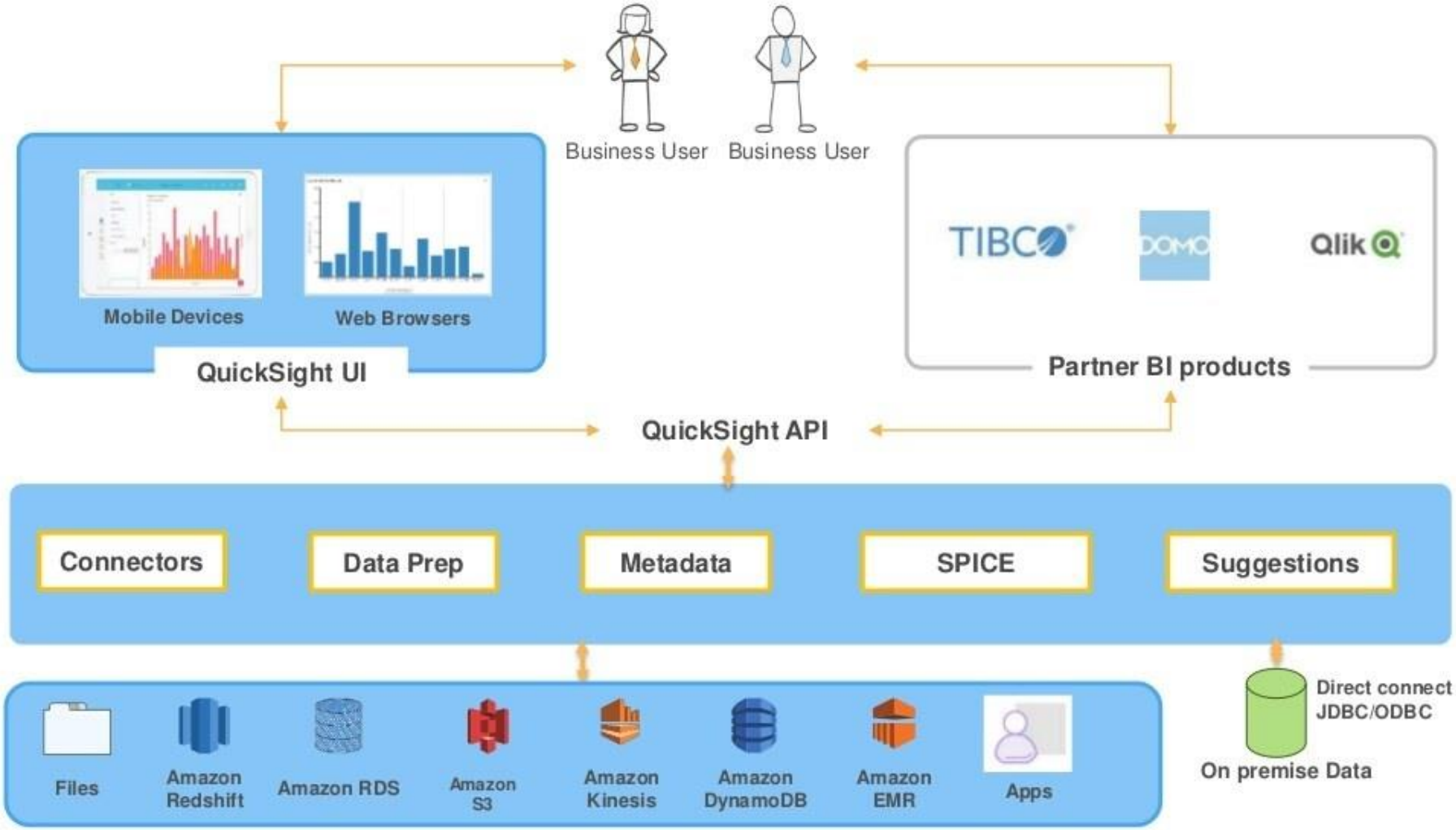
A fast, cloud-powered, BI service for 1/10th the cost of traditional BI software

First analysis in about 60 seconds



Business user

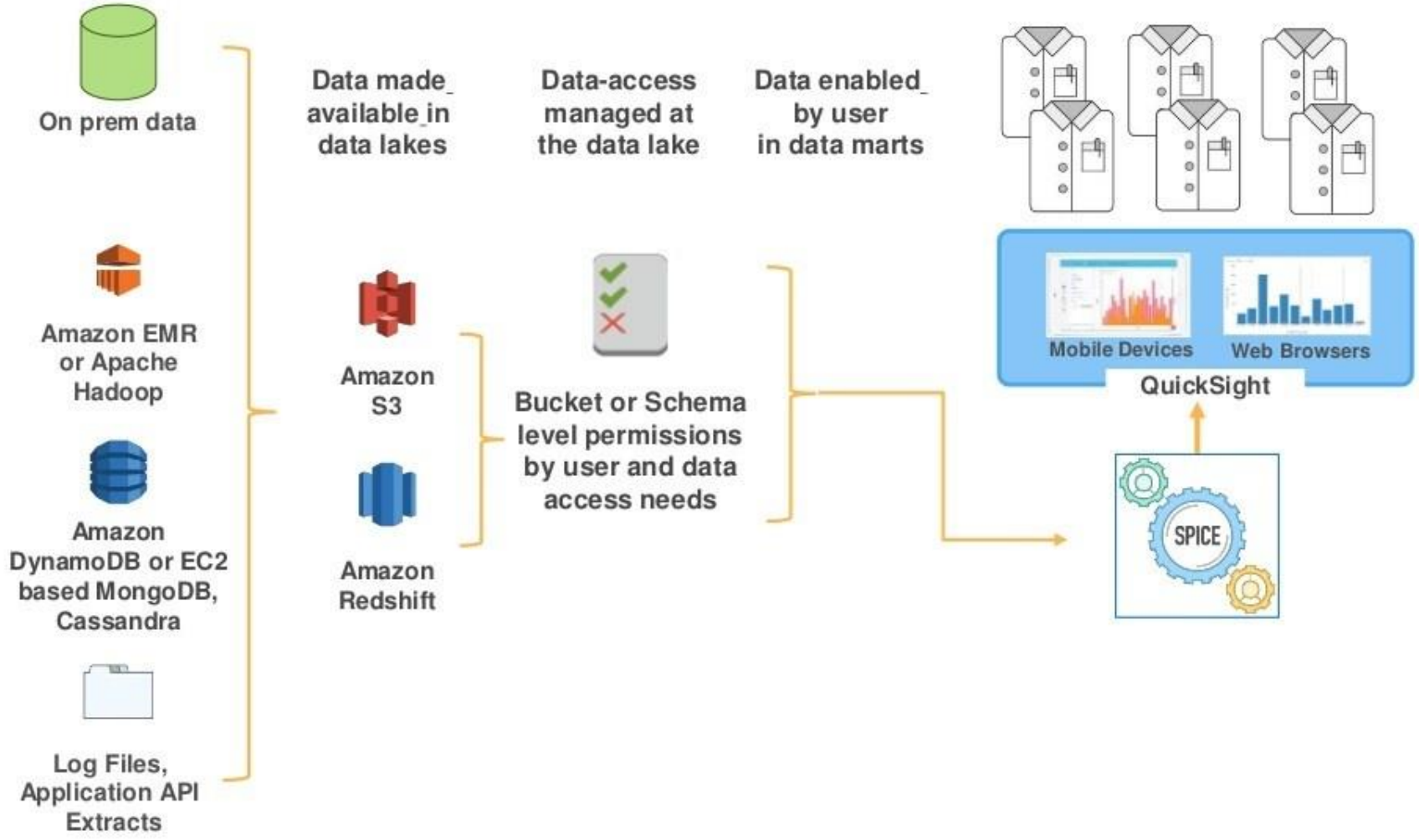
Sign-in


Business User
Business User
Mobile Devices
Web Browsers
QuickSight UI
TIBCO
Qlik
Partner BI products
QuickSight API
Connectors
Data Prep
Metadata
SPICE
Suggestions
Files
Amazon Redshift
Amazon RDS
Amazon S3
Amazon Kinesis
Amazon DynamoDB
Amazon EMR
Apps
Direct connect JDBC/ODBC
On premise Data

I have multiple data sets both on-premise and on AWS from different sources, and I need to make data available and enable access via Amazon QuickSight.

How do I do this?

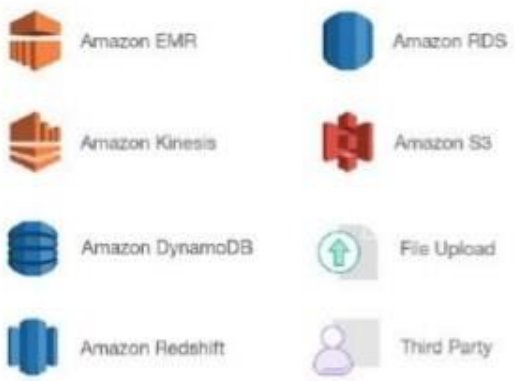
1. Data made available in “Data Lakes” using Amazon S3 or Amazon Redshift
2. Data access managed with bucket or schema level policies
3. Data enabled via Amazon QuickSight


On prem data
Amazon EMR or Apache Hadoop
Amazon DynamoDB or EC2 based MongoDB, Cassandra
Log Files, Application API Extracts
Data made available in data lakes
Amazon S3
Amazon Redshift
Data-access managed at the data lake
Bucket or Schema level permissions by user and data access needs
Data enabled by user in data marts
Mobile Devices
Web Browsers
QuickSight
SPICE

Innovations

- Easy exploration of AWS data
- Fast insights with SPICE (**S**uper-fast, **P**arallel, **I**n-memory, **C**alculation **E**ngine)
- Intuitive visualizations and transitions with AutoGraph
- Native mobile experience
- Secure sharing and collaboration through StoryBoards

Easy exploration of AWS data



Securely discover and connect to AWS data

Quickly explore AWS data sources

- Relational databases
- NoSQL databases
- Amazon EMR, Amazon S3, files
- Streaming data sources

Easily import data from any table or file

Automatic detection of data types

How do I SPICE up my data?

Fast insights with SPICE

- **S**uper-fast, **P**arallel, **I**n-memory optimized, **C**alculation **E**ngine
- 2 to 4x compression columnar data
- Compiled queries with machine code generation
- Rich calculations
- SQL-like syntax
- Very fast response time to queries
- Fully managed – No hardware or software to license

Data Sources

Connect to AWS data services; upload files; or connect to apps such as Salesforce

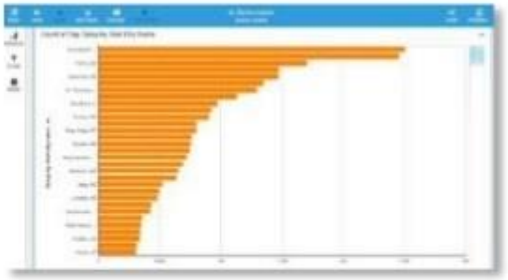
In-Memory Calculation Engine

The Super-fast, Parallel, In-memory, Calculation Engine ("SPICE") generates answers on large datasets and returns rapid responses

QuickSight UI

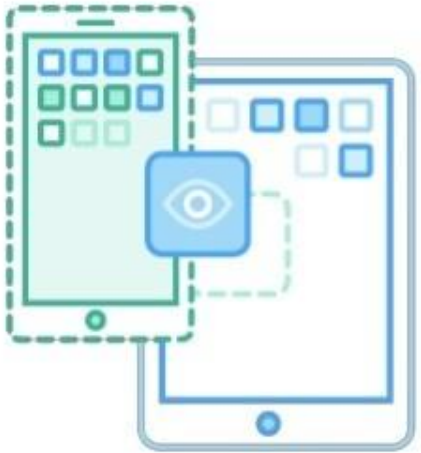
SPICE allows for very fast analysis and smart visualizations for sharing and collaboration

Intuitive visualizations with AutoGraph



- Automatic detection of data types
- Optimal query generation
- Appropriate graph type selection
- Ability to customize the graph type
- Very fast response

Native mobile experience

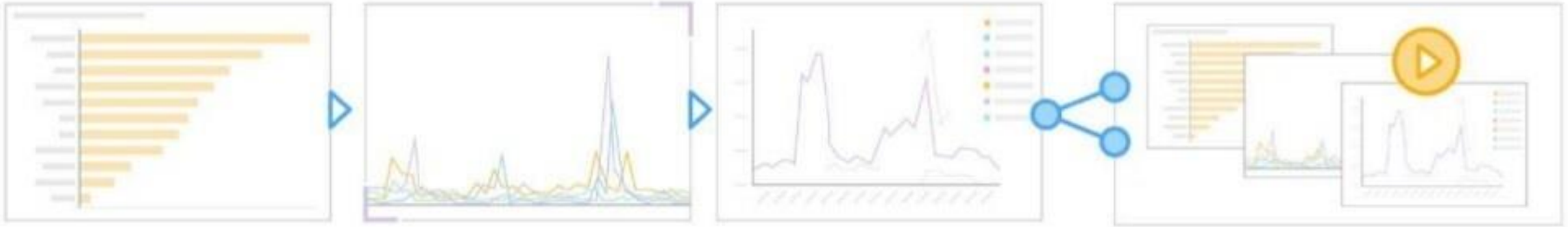


- iOS, Android
- Full experience on tablets
- Consumption experience on smart phones
- Very fast response

Tell a story with your data

- Capture the critical snapshot of analysis
- Build a sequence of analysis
- Share it securely

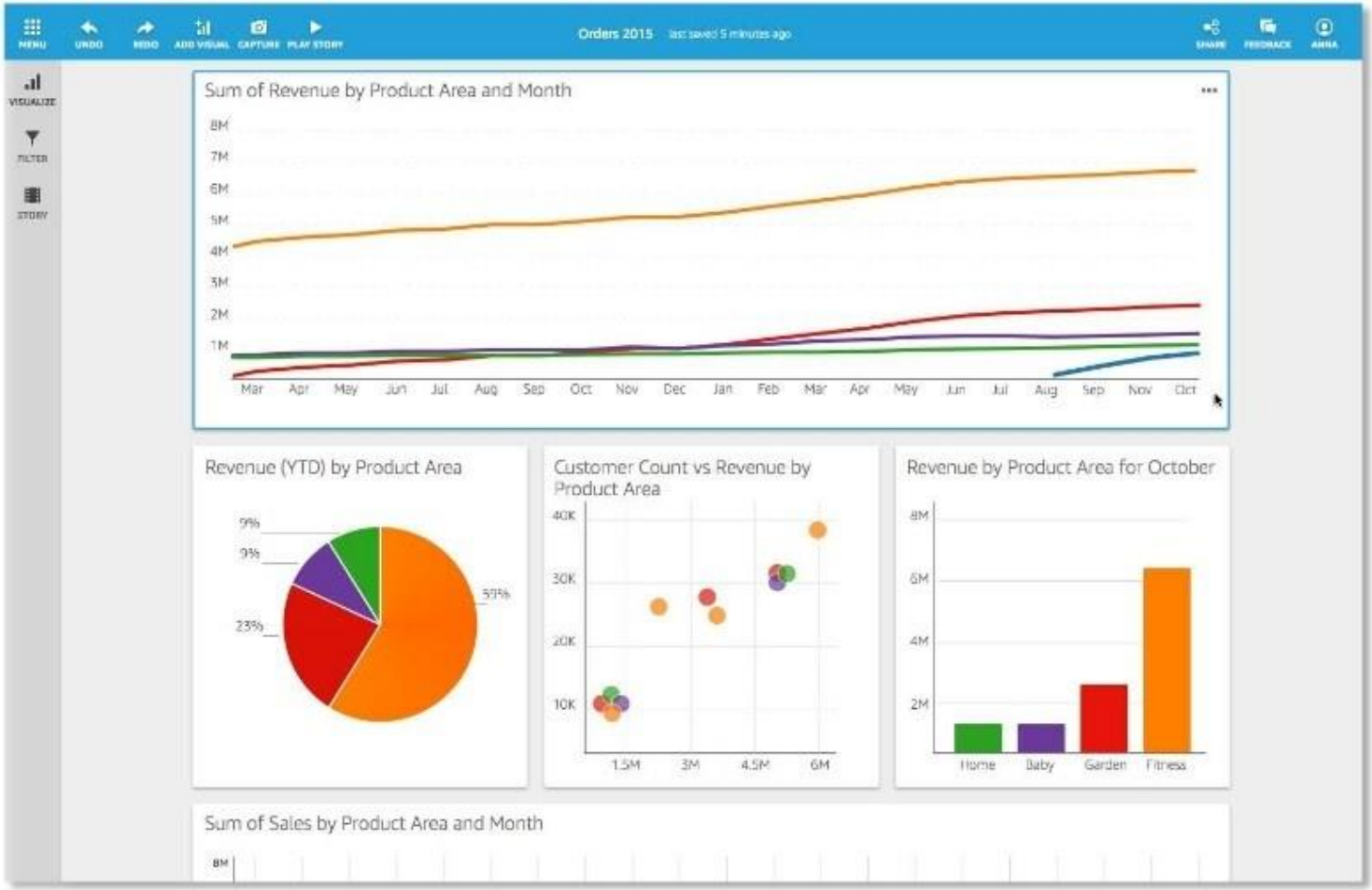
- Enable interactive exploration
- Very fast response



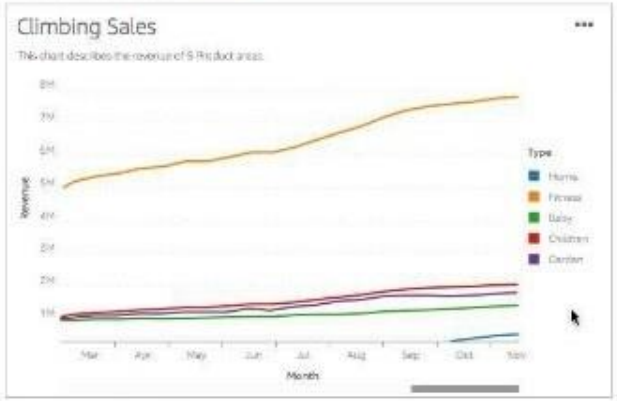
Conduct your analysis and capture the visualizations as 'scenes' with annotations and comments

Share the scenes as a story to be played back anytime for further analysis

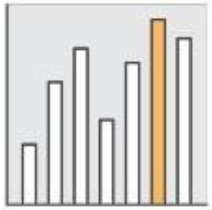
Dashboard


MENU
UNDO
REDO
ADD VISUAL
CAPTURE
PLAY STORY
Orders 2015
last saved 5 minutes ago
SHARE
FEEDBACK
ANNA
VISUALIZE
FILTER
STORY
Sum of Revenue by Product Area and Month
8M
7M
6M
5M
4M
3M
2M
1M
Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Nov Oct
Revenue (YTD) by Product Area
9%
9%
23%
59%
Customer Count vs Revenue by Product Area
40K
30K
20K
10K
1.5M
3M
4.5M
6M
Revenue by Product Area for October
8M
6M
4M
2M
Home
Baby
Garden
Fitness
Sum of Sales by Product Area and Month
8M

Embeddable



Fast to get started



Easily explore any AWS data

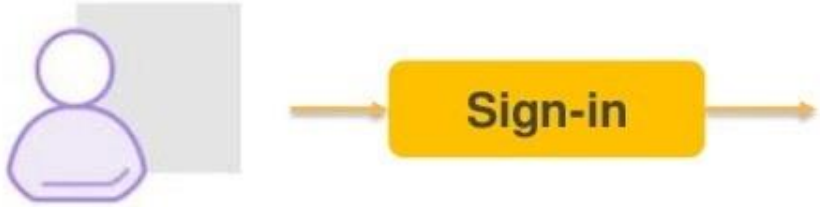
Fast insights with SPICE

Easy to use and share

Effortless scale

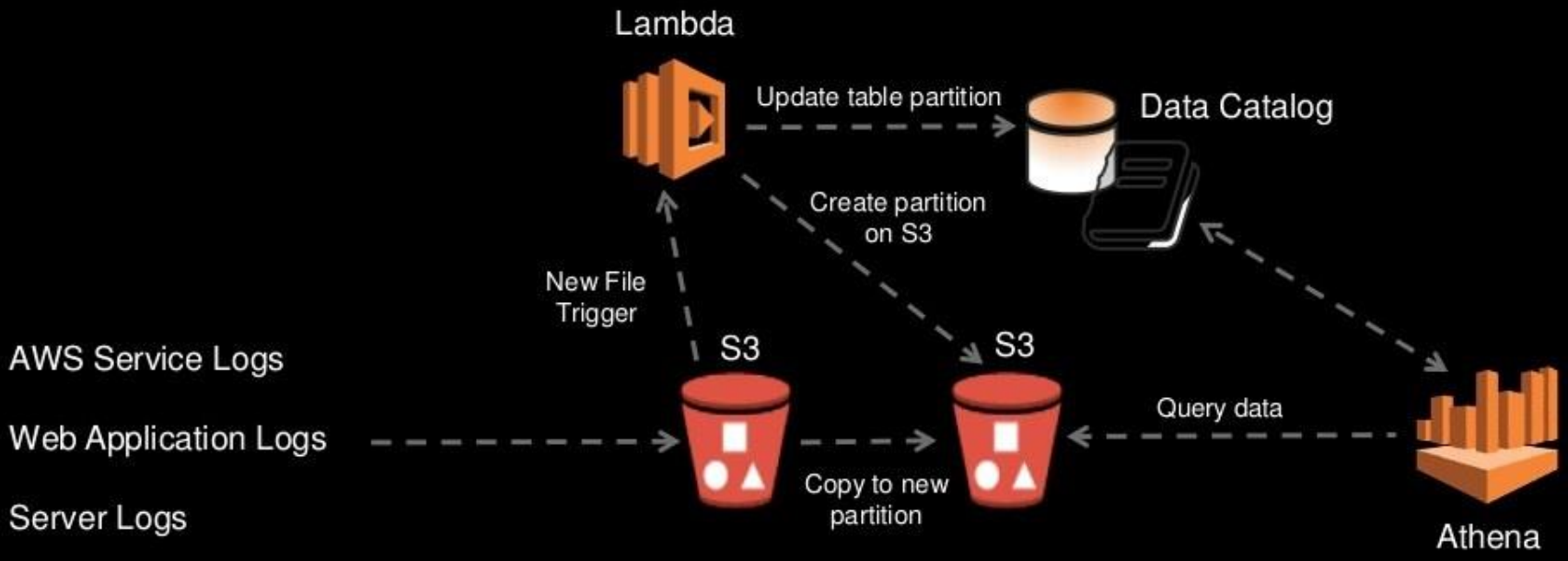
Low cost

Register for the Preview @ aws.amazon.com/quicksight



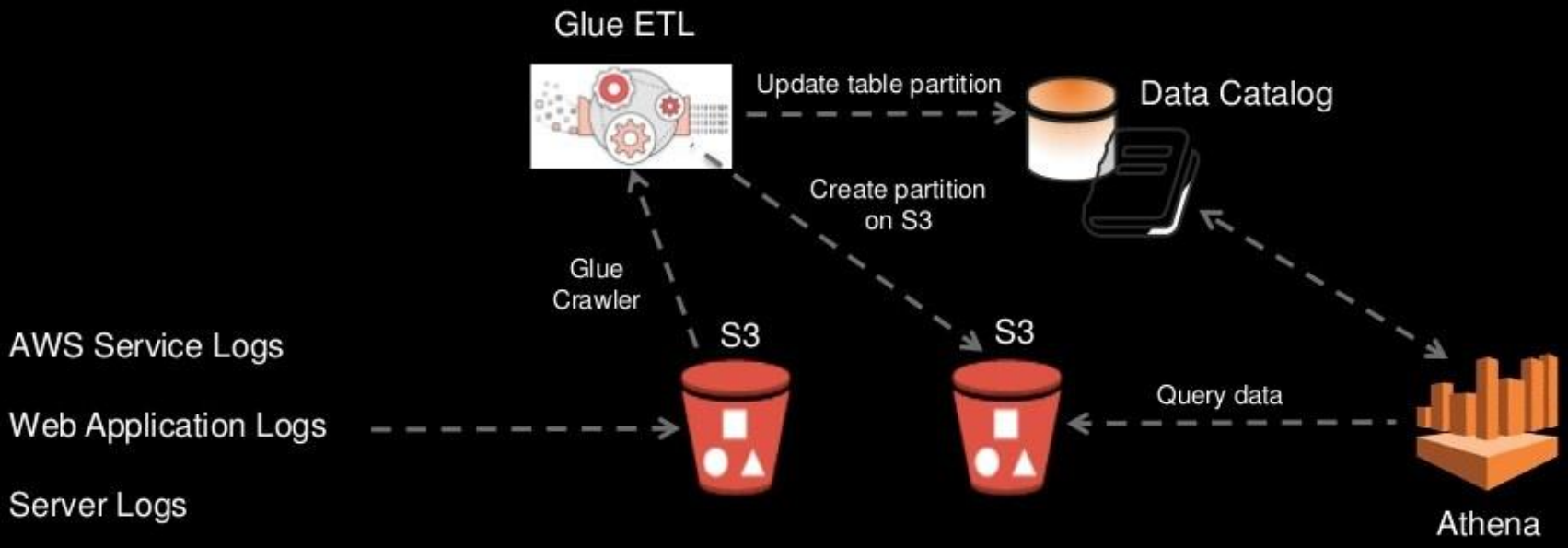
Popular Customer Use Cases

Log Aggregation



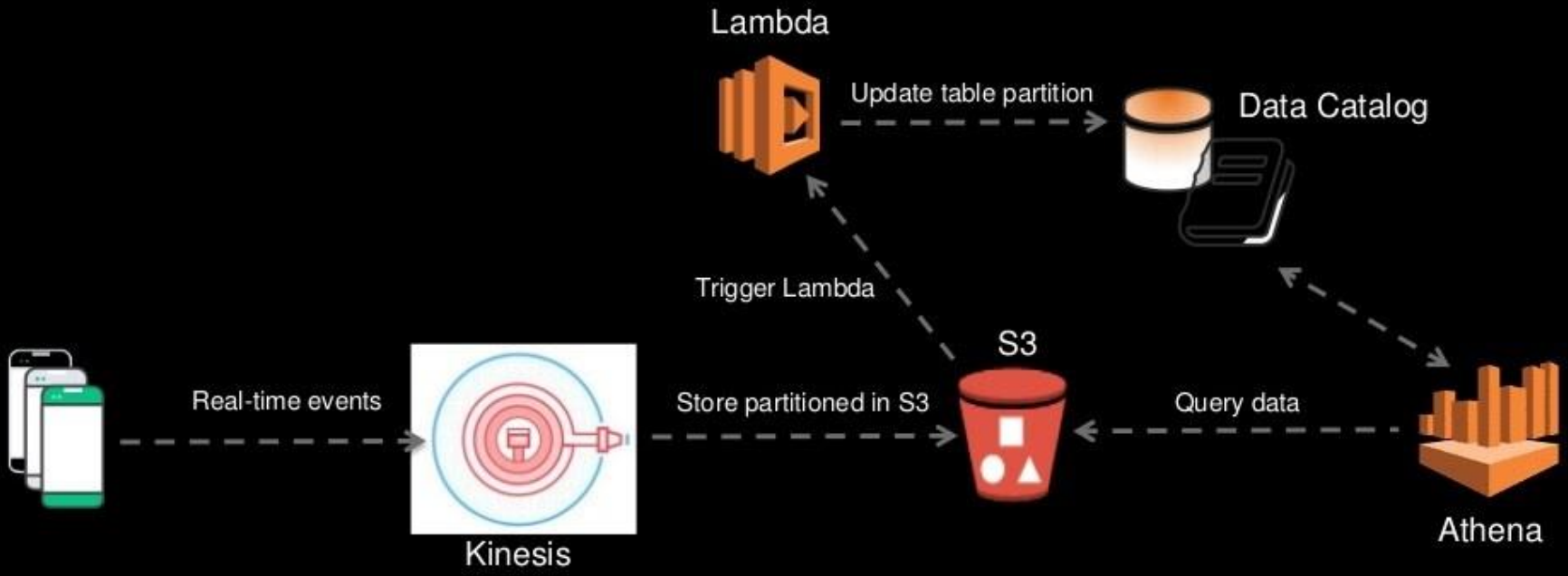
© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Log Aggregation with ETL



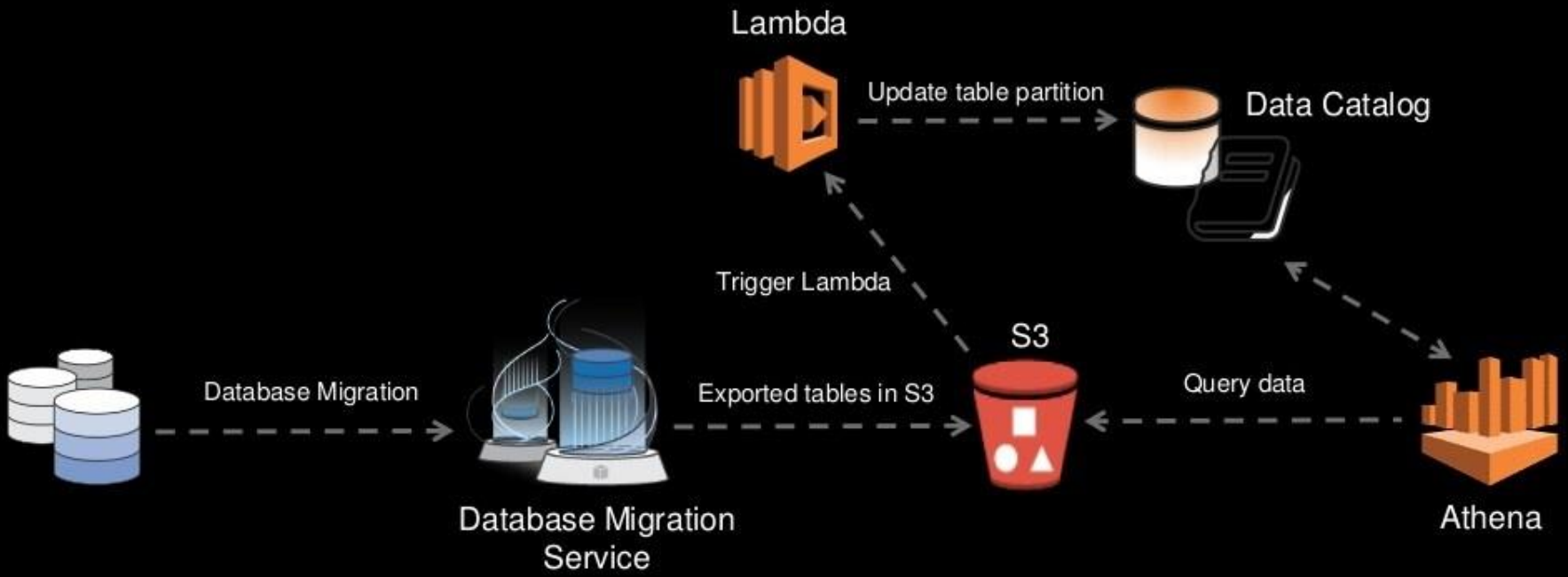
© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Real-Time Data Collection



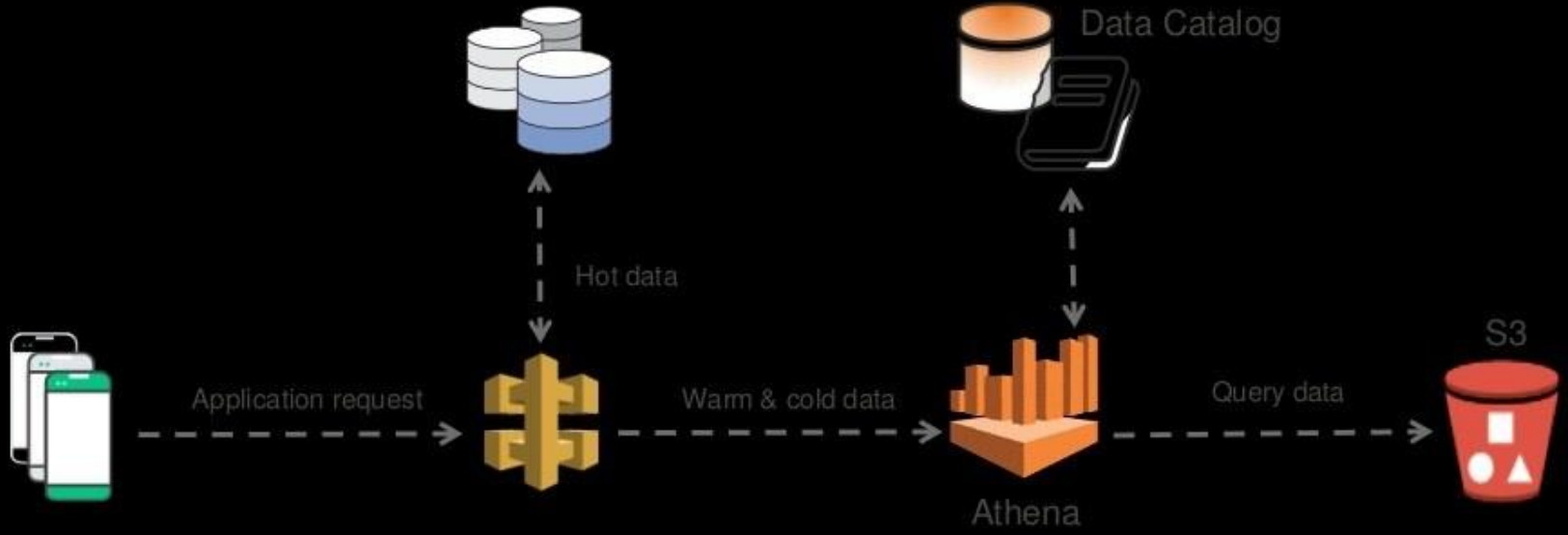
© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data Export



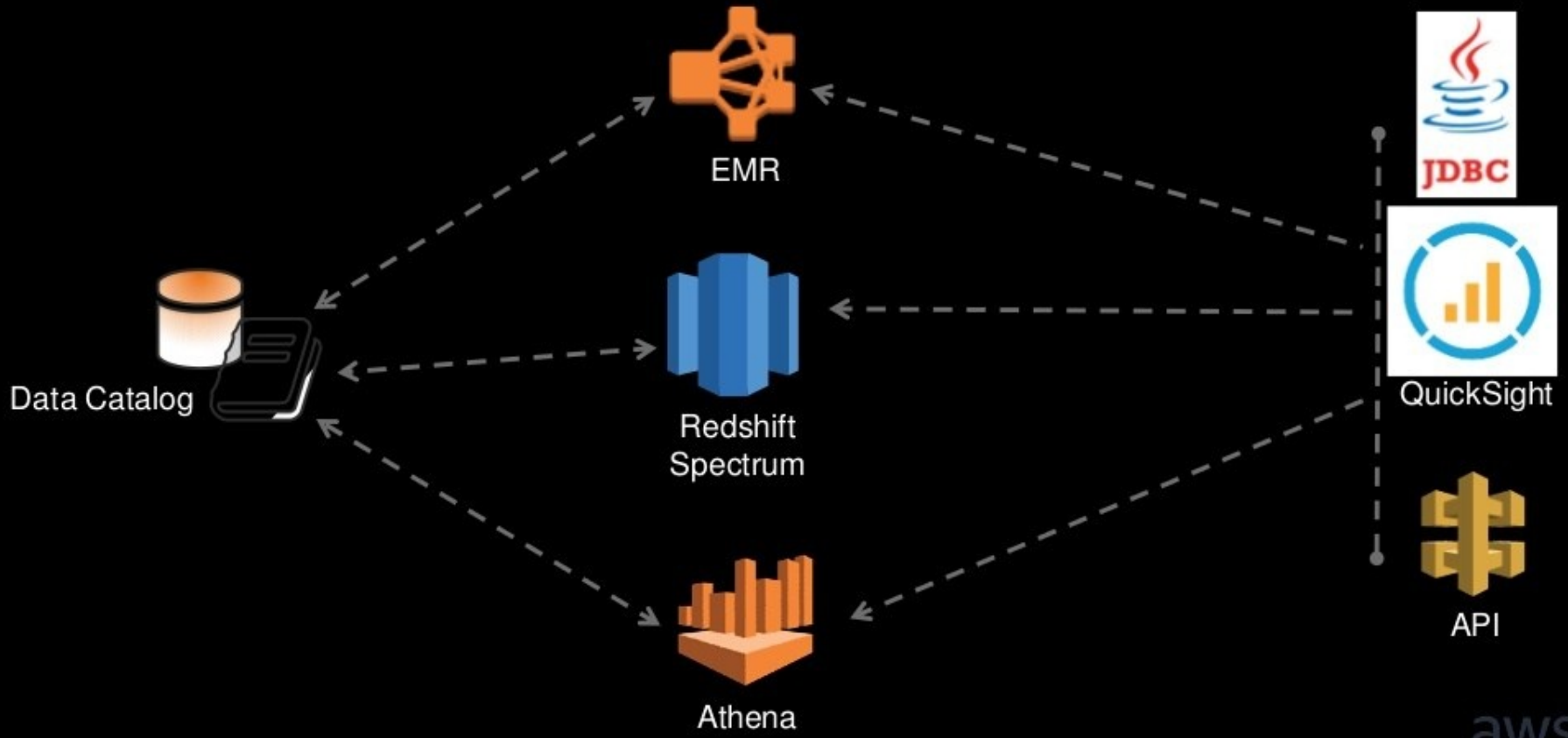
© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

SaaS Model



© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Analytics Reporting*



© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.