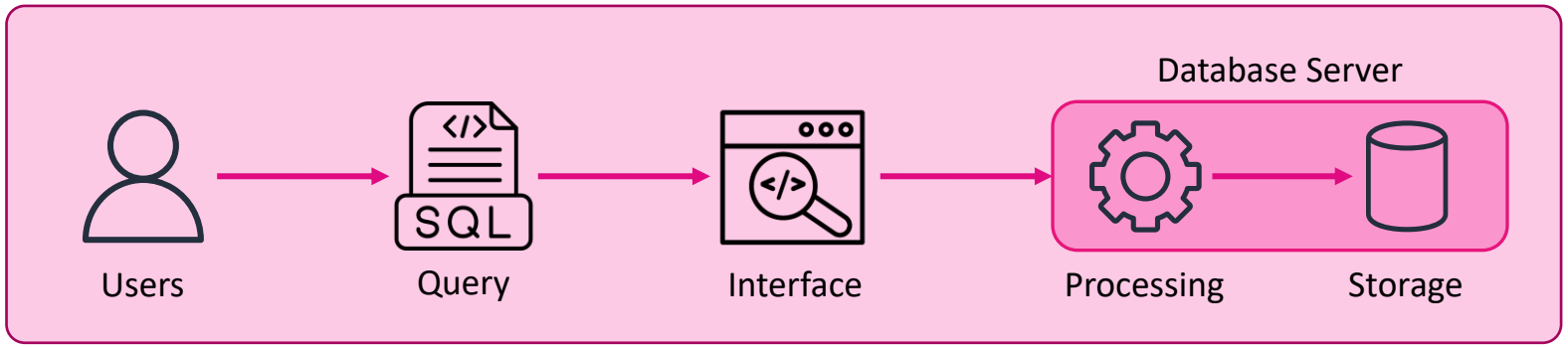
Querying Data with Amazon Athena

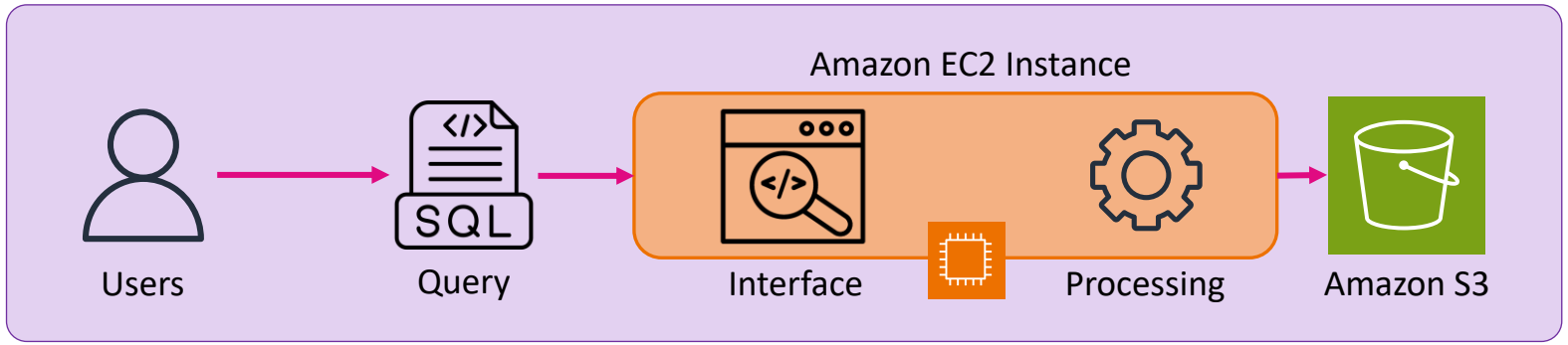
Amazon Athena

Querying data using SQL

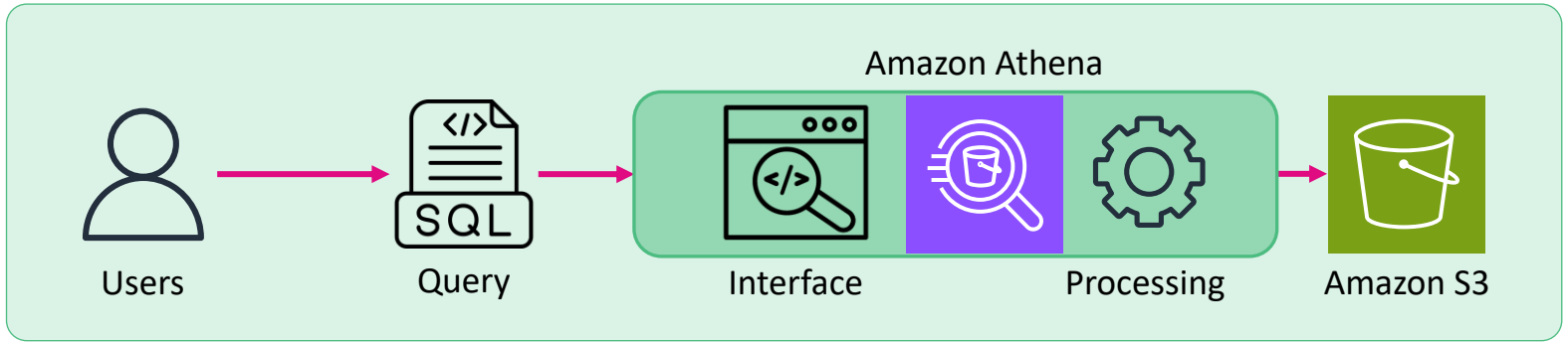
Querying data from a Database



Querying data from Amazon S3



Querying data from Amazon S3 using Amazon Athena



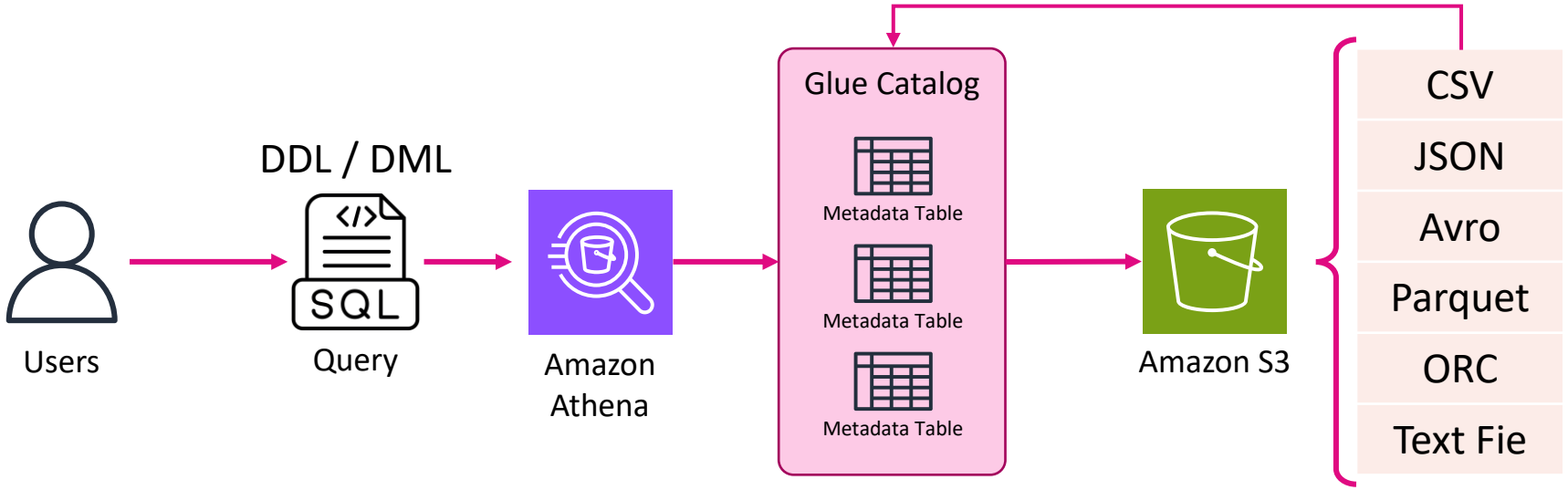
Amazon Athena

- Athena helps you analyze unstructured, semi-structured, and structured data stored in Amazon S3.

SIMPLE, INSTANT START	INTERACTIVE, ADVANCED ANALYTICS	OPEN AND FLEXIBLE	COST EFFECTIVE
Serverless, no setup	Federated queries across 35+ data stores	ANSI SQL, Apache Spark	Pay only for what you use
Instant start, optimized runtimes for fast results	Use PySpark ecosystem	Multiple formats, compression types	SQL: Save 30%–90% on per-query costs through compression
Point to S3 and start querying	Simplified notebooks on console for PySpark	Complex joins and data types	Python: minimize idle compute charges

How it works?

- Athena integrates with the AWS Glue Data Catalog, which offers a persistent metadata store for your data in Amazon S3.
- For each dataset that you'd like to query, Athena must have an underlying table it will use for obtaining and returning query results. Therefore, before querying data, a table must be registered in Athena.
- You can create a table manually or automatically (using an Glue crawler).



Query Optimization

Optimizing Athena Query

Use Columnar Data

Query performance improves if you convert your data into columnar formats. Use Athena's CTAS feature or use AWS Glue jobs to convert your existing data to a columnar format

Supported Types
- Parquet
- ORC

Partition Data

By partitioning your data, you can restrict the amount of data scanned by each query, thus improving performance and reducing cost

Available Methods
- Partition Projection
- Glue Partition Indexing for multi-engine support
- Bucketing

Compression

Athena bills on compressed bytes, so you can save up to 90% on queries that scan data through compression

Supported Types
- BZIP2
- DEFLATE
- GZIP
- LZ4
- LZO
- SNAPPY
- ZLIB
- ZSTD

Result Reuse

When you re-run a query in Athena, you can optionally choose to reuse the last stored query result.
This option can increase performance and reduce costs.

Key features
- The maximum age for reusing query results can be specified in minutes, hours, or days.

Amazon Athena Workgroup

Workgroups

WORKLOAD ISOLATION

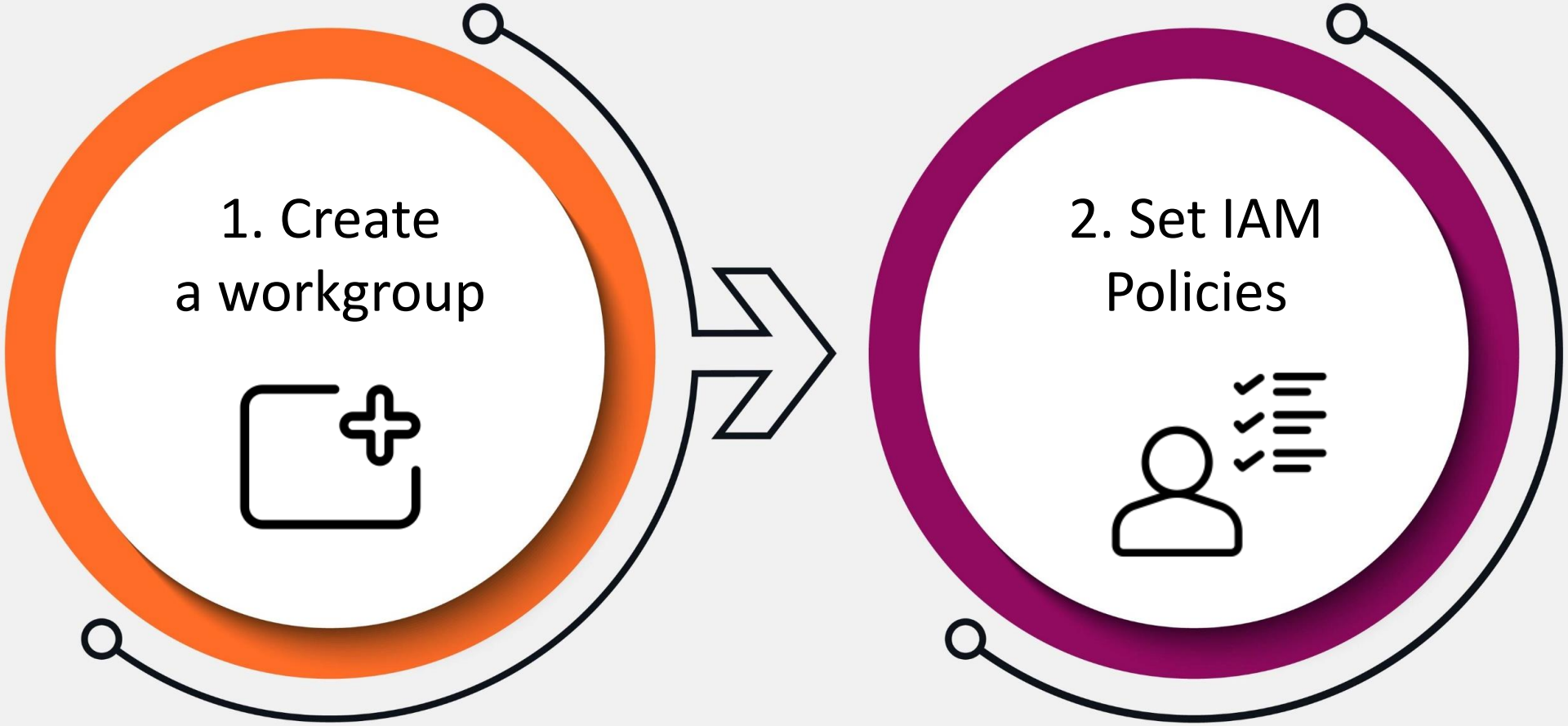
QUERY METRICS

COST CONTROLS

- In Amazon Athena you can use workgroups to separate users, teams, applications, or workloads, to set limits on amount of data each query or the entire workgroup can process.
- By default, if you have not created any workgroups, all queries in your account run in the primary workgroup.
- You can also use workgroups to track costs.
- Because workgroups act as resources, you can use resource-level identity-based policies to control access to a specific workgroup.

Setting Up Workgroups

Two step process:



Workgroups – Cost Controls

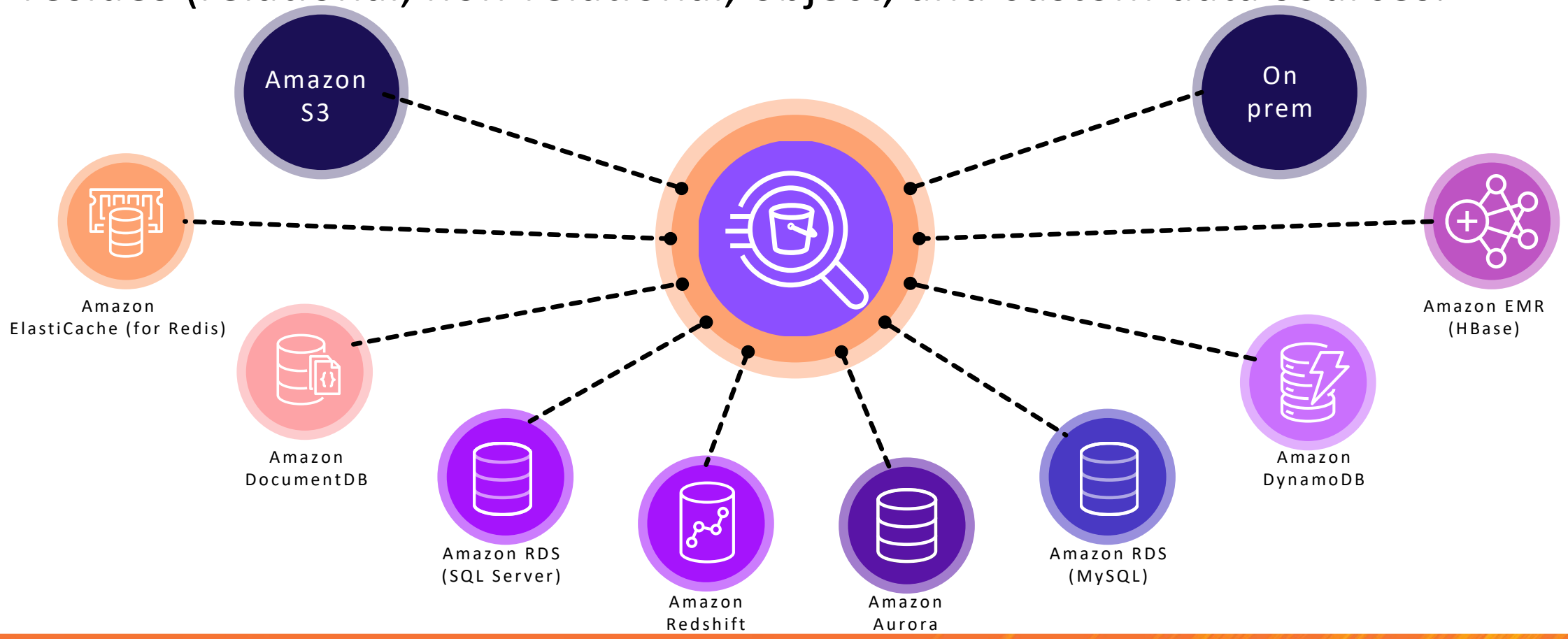
- Per query data scanned threshold; exceeding, will cancel query
- Trigger alarms to notify of increasing usage and cost
- Disable Workgroup when all queries exceed a maximum threshold
- Any Athena metric: successful/failed & total queries, query run time, etc.

Data limit	Time period	Action
10 Gigabytes	Not applicable	Query will be cancelled.
1 Terabytes	24 hours	Send notification to topic : arn:aws:sns:us-east-1:9 9:AthenaAlarm
10 Gigabytes	1 hour	Send notification to topic : arn:aws:sns:us-east-1:9 9:AthenaAlarm

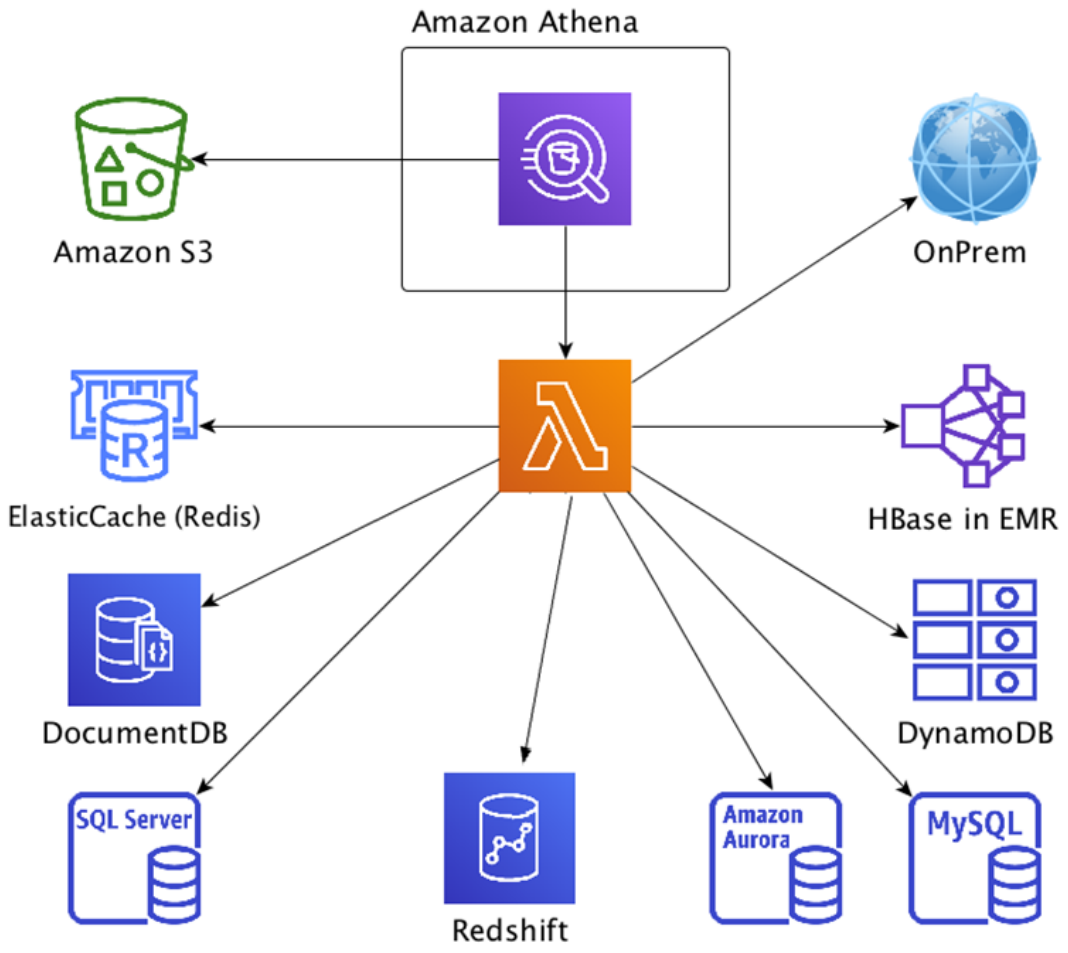
Federated Query

Federated Query in Amazon Athena

- As the number of data stores and applications increase, running analytics across multiple data sources can become challenging.
- Federated queries allow you to query the data in-place from wherever it resides (relational, non-relational, object, and custom data sources.

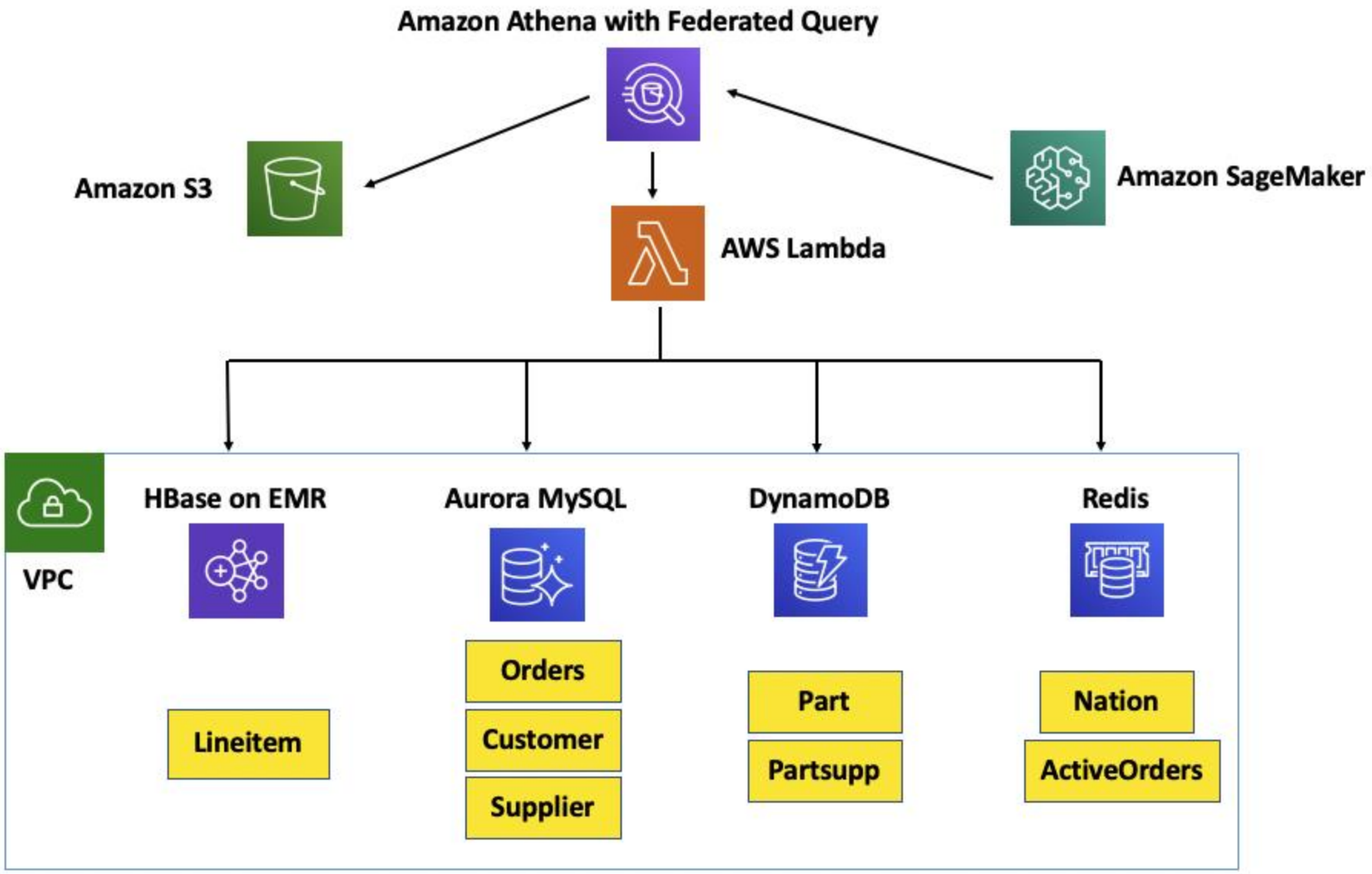


How it works?

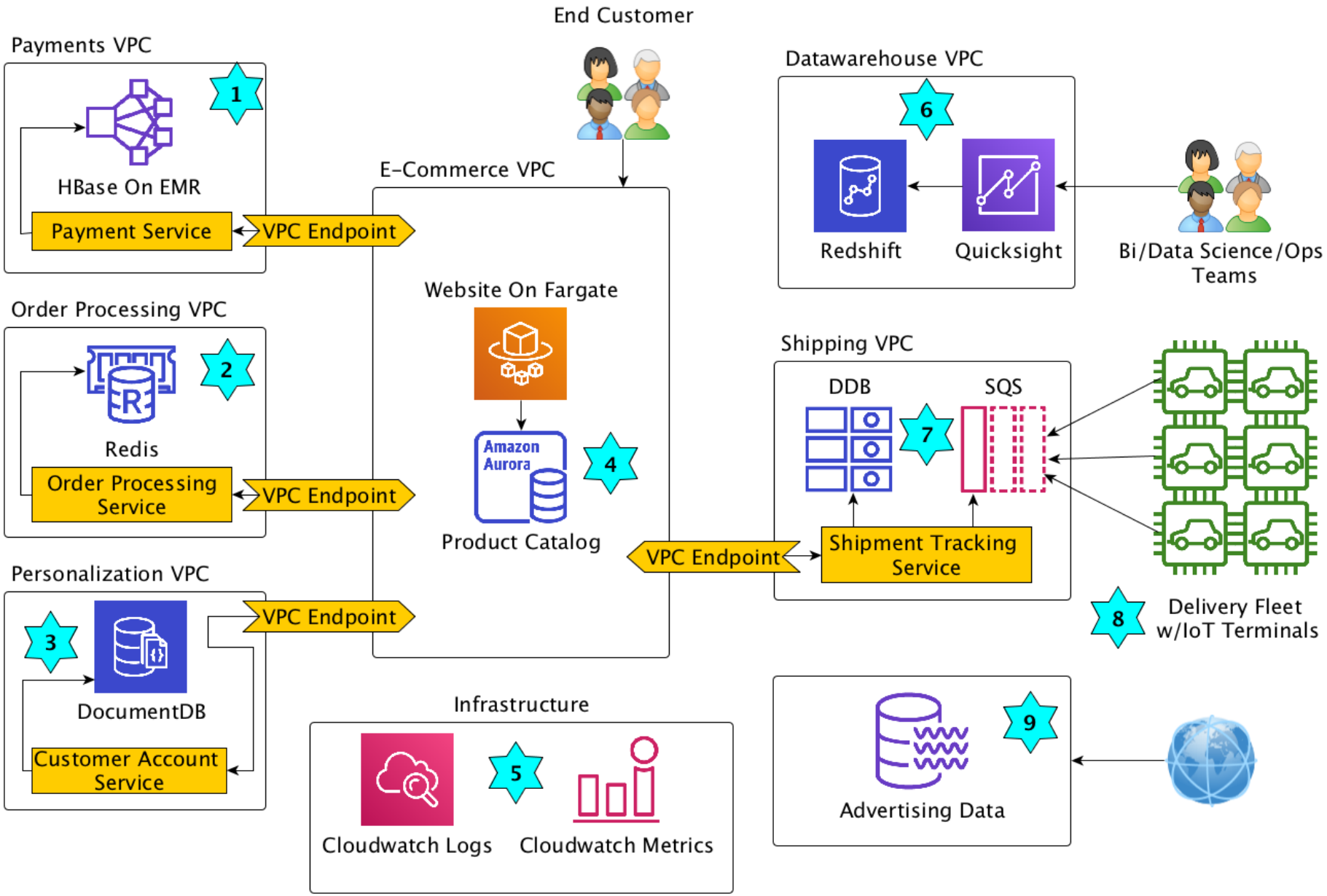


- Athena uses data source connectors that run on AWS Lambda to execute federated queries.
- A data source connector is a piece of code that can translate between your target data source and Athena. You can think of a connector as an extension of Athena's query engine.
- When a query is submitted against a data source, Athena invokes the corresponding connector to identify parts of the tables that need to be read, manages parallelism, and pushes down filter predicates.

A hypothetical e-commerce company who's architecture uses:



Another example



New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

lambda:cloudwatch

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

lambda:cloudwatch

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

lambda:redis

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

lambda:cmdb

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

lambda:docdb

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
WITH logs
    AS (SELECT log_stream,
               message                                          AS
               order_processor_log,
               Regexp_extract(message, '.*orderId=(\d+) .*', 1) AS orderId,
               Regexp_extract(message, '(.*):.*', 1)            AS log_level
        FROM
    "lambda:cloudwatch"."/var/ecommerce-engine/order-processor".all_log_streams
        WHERE  Regexp_extract(message, '(.*):.*', 1) != 'INFO'),
    active_orders
    AS (SELECT *
        FROM   "lambda:redis".redis_db.redis_customer_orders),
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status"
```

lambda:athena-dynamodb

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
13      order_processors
14      AS (SELECT instanceid,
15                 publicipaddress,
16                 state.NAME
17          FROM   "lambda:cmdb".ec2.ec2_instances),
18      customer
19      AS (SELECT id,
20                 email
21          FROM   "lambda:docdb"."customers".customer_info),
22      addresses
23      AS (SELECT id,
24                 is_residential,
25                 address.street AS street
26          FROM   "lambda:docdb"."customers".customer_addresses),
27      shipments
28      AS ( SELECT order_id,
29                  shipment_id,
30                  from_unixtime(cast(shipped_date as double)) as shipment_time,
31                  carrier
32         FROM "lambda:athena-dynamodb".default.order_shipments),
33      payments
34      AS ( SELECT "summary:order_id",
35                  "summary:status",
36                  "summary:cc_id",
37                  "details:network"
38         FROM "lambda:hbase".hbase_payments.transactions)
39
40 SELECT _key_              AS redis_order_id,
41        customer_id,
42        customer.email     AS cust_email,
43        "summary:cc_id"    AS credit_card,
44        "details:network"  AS CC_type,
45        "summary:status"   AS payment_status,
46        status             AS redis_status,
47        addresses.street   AS street_address,
```

lambda:hbase

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
    order_processors
    AS (SELECT instanceid,
               publicipaddress,
               state.NAME
        FROM   "lambda:cmdb".ec2.ec2_instances),
    customer
    AS (SELECT id,
               email
        FROM   "lambda:docdb"."customers".customer_info),
    addresses
    AS (SELECT id,
               is_residential,
               address.street AS street
        FROM   "lambda:docdb"."customers".customer_addresses),
    shipments
    AS ( SELECT order_id,
                shipment_id,
                from_unixtime(cast(shipped_date as double)) as shipment_time,
                carrier
       FROM "lambda:athena-dynamodb".default.order_shipments),
    payments
    AS ( SELECT "summary:order_id",
                "summary:status",
                "summary:cc_id",
                "details:network"
       FROM "lambda:hbase".hbase_payments.transactions)

SELECT _key_              AS redis_order_id,
       customer_id,
       customer.email     AS cust_email,
       "summary:cc_id"    AS credit_card,
       "details:network"  AS CC_type,
       "summary:status"   AS payment_status,
       status             AS redis_status,
       addresses.street   AS street_address,
```

Run query | Save as | Create | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
FROM    active_orders
        LEFT JOIN logs
               ON logs.orderid = active_orders._key_
        LEFT JOIN order_processors
               ON logs.log_stream = order_processors.instance_id
        LEFT JOIN customer
               ON customer.id = customer_id
        LEFT JOIN addresses
               ON addresses.id = address_id
        LEFT JOIN shipments
               ON shipments.order_id = active_orders._key_
        LEFT JOIN payments
               ON payments."summary:order_id" = active_orders._key_
```

Run query | Save as | Create | (Run time: 7.72 seconds, Data scanned: 2.07 KB) | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

Redis | Document DB | HBase | Document DB

	redis_order_id	customer_id	cust_email	credit_card	CC_type	payment_status	redis_status	street_address	shipment_time	shipment_carrier	ec2_order_
1	0001237	92053	sdoe@email.com	9001	VISA	PENDING	backordered	96110 Rufus Ave.		internal	
2	0001238	453	adoe@email.com	8827	MASTERCARD	FUNDED	preparing_to_ship	987 Rufus St.	1976-03-23 15:52:54.000	Shipper2	
3	0001236	12151	jadoe@email.com	5612	VISA	PENDING	pending_payment	456790 Rufus Lane		Shipper3	
4	0002234	9820	cdoe@email.com	1189	VISA	PENDING	backordered	13890 Rufus Road		Shipper1	52.14.89.109
5	0001235	11123	jdoe@email.com	4119	AMEX	FUNDED	shipping	13456 Rufus Dr.	2012-12-09 21:16:50.000	internal	52.14.89.109
6	0001234	11123	jdoe@email.com	4119	AMEX	FUNDED	preparing_to_ship	13456 Rufus Dr.	2013-06-17 19:14:01.000	internal	

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
FROM    active_orders
        LEFT JOIN logs
               ON logs.orderid = active_orders._key_
        LEFT JOIN order_processors
               ON logs.log_stream = order_processors.instance_id
        LEFT JOIN customer
               ON customer.id = customer_id
        LEFT JOIN addresses
               ON addresses.id = address_id
        LEFT JOIN shipments
               ON shipments.order_id = active_orders._key_
        LEFT JOIN payments
               ON payments."summary:order_id" = active_orders._key_
```

Run query | Save as | Create | (Run time: 7.72 seconds, Data scanned: 2.07 KB) | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

CMDB

ent_status	redis_status	street_address	shipment_time	shipment_carrier	ec2_order_processor	ec2_state	log_level	order_processor_log
G	backordered	96110 Rufus Ave.		internal				
	preparing_to_ship	987 Rufus St.	1976-03-23 15:52:54.000	Shipper2				
G	pending_payment	456790 Rufus Lane		Shipper3				
G	backordered	13890 Rufus Road		Shipper1	52.14.89.109	running	WARN	WARN: orderId=0002234 encountered a problem during shipping
	shipping	13456 Rufus Dr.	2012-12-09 21:16:50.000	internal	52.14.89.109	running	ERROR	ERROR: orderId=0001235 encountered a problem during shippir
	preparing_to_ship	13456 Rufus Dr.	2013-06-17 19:14:01.000	internal				

New query 1 | New query 2 | New query 3 | New query 4 | New query 5 | New query 6 | New query 7 | New query 8 | New query 10 | New query 9

```
FROM    active_orders
        LEFT JOIN logs
               ON logs.orderid = active_orders._key_
        LEFT JOIN order_processors
               ON logs.log_stream = order_processors.instance_id
        LEFT JOIN customer
               ON customer.id = customer_id
        LEFT JOIN addresses
               ON addresses.id = address_id
        LEFT JOIN shipments
               ON shipments.order_id = active_orders._key_
        LEFT JOIN payments
               ON payments."summary:order_id" = active_orders._key_
```

Run query | Save as | Create | (Run time: 7.72 seconds, Data scanned: 2.07 KB) | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

CMDB

ent_status	redis_status	street_address	shipment_time	shipment_carrier	ec2_order_processor	ec2_state	log_level	order_processor_log
G	backordered	96110 Rufus Ave.		internal				
	preparing_to_ship	987 Rufus St.	1976-03-23 15:52:54.000	Shipper2				
G	pending_payment	456790 Rufus Lane		Shipper3				
G	backordered	13890 Rufus Road		Shipper1	52.14.89.109	running	WARN	WARN: orderId=0002234 encountered a problem during shipping
	shipping	13456 Rufus Dr.	2012-12-09 21:16:50.000	internal	52.14.89.109	running	ERROR	ERROR: orderId=0001235 encountered a problem during shippir
	preparing_to_ship	13456 Rufus Dr.	2013-06-17 19:14:01.000	internal				