AWS PARTNER CERTIFICATION READINESS

Content Review Session

Week 2

Domain 1: Data Ingestion and Transformation

OPTIONAL AWS Skill Builder Subscription

The Skill Builder subscription provides access to official AWS Certification practice exams, self-paced digital training content including open-ended challenges, self-paced labs, and game-based learning. **Please note, the Skill Builder subscription is not required for this Accelerator program.**

Free digital training

LINK HERE

Special features include:

- 600+ digital courses
- Learning plans
- 10 Practice Question Sets
- *AWS Cloud Quest (Foundational)*

Individual subscription

LINK HERE

Everything in free digital training, plus:

- AWS Cloud Quest (Intermediate - Advanced)
- AWS Certification Official Practice Exams
- Enhanced Exam Prep Courses
- Unlimited access to 1000+ hands-on labs
- AWS Jam Journeys (lab-based challenges)
- AWS Digital Classroom (Annual only)

Individual subscriptions are priced at $29 USD per month (*Flexibility to cancel anytime*) or $449 USD per year.

Access **65** Data Engineer - Associate Practice Exam Questions with feedback on your answer choices

Today's Learning Outcomes

During this session, we will cover:

- Performing data ingestion
- Transforming and processing data
- Orchestrating data pipelines
- Applying programming concepts

AWS PARTNER CERTIFICATION READINESS

Domain 1: Data Ingestion and Transformation

Perform data ingestion

Perform data ingestion

Knowledge of:

- Throughput and latency characteristics for AWS services that ingest data
- Data ingestion patterns (for example, frequency and data history)
- Streaming data ingestion
- Batch data ingestion (for example, scheduled ingestion, event-driven ingestion)
- Replayability of data ingestion pipelines
- Stateful and stateless data transactions

Skills in:

- Reading data from streaming & batch sources
- Implementing appropriate configuration options for batch ingestion
- Consuming data APIs, setting up event
- Setting up schedulers by using Amazon EventBridge, Apache Airflow, or time-based schedules for jobs and crawlers
- Calling a Lambda function from Amazon Kinesis
- Creating allowlists for IP addresses to allow connections to data sources
- Implementing throttling and overcoming rate limits • Managing fan-in and fan-out for streaming data distribution

Data Analytics Overview on AWS

Data Ingestion


Real time


Batch


Amazon Kinesis Data/ Amazon Kinesis Video Streams


Amazon AppFlow


Amazon Managed Streaming for Kafka


Amazon Kinesis Data Firehose

Use Cases

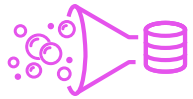
- Audio, video, web clickstreams, app log, devices, social media

Data Processing


Batch


Interactive


AWS Glue


AWS Glue DataBrew


Amazon EMR


AWS Lambda

- Data cleaning, validation, enrichment & transformation

Data Lake


Storage


Setup, Catalog, and Security


Amazon S3 / Amazon S3 Glacier


AWS Lake Formation


AWS Data Exchange


AWS Glue Data Catalog

- Centralized repository for all structured & unstructured data
- Role-based access and ownership for data lake objects

Data Analytics


Query Engine


Visualization


Search


Amazon Athena


QuickSight


Amazon Elasticsearch Service


Amazon Redshift + Analytics Workbench


Amazon EMR

- Lake house approach; minimal data moves
- Personalized search experience across applications, websites

Batch and Stream Processing Architectures

Batch processing is the method computers use to periodically complete high-volume, repetitive data jobs. Certain data processing tasks, such as backups, filtering, and sorting, can be compute intensive and inefficient to run on individual data transactions.

In contrast, **stream processing** requires ingesting a sequence of data, and incrementally updating metrics, reports, and summary statistics in response to each arriving data record. It is better suited for **real-time** monitoring and response

	Streaming analytics	Batch analytics
Scope	Filter most recent event (stateless) or, over a specified time window (stateful), aggregate counts, rolling metrics	Query or processing over the entire dataset
Size	Individual events or microbatches consisting of fewer records	Large batches of data
Performance	Results required in milliseconds to seconds	Latencies in minutes to hours are acceptable

Stream Processing on AWS


Amazon Kinesis Data Analytics and KDA Studio (both MSK and Kinesis Data Steams)


Amazon MSK Connect for sink from to Amazon MSK


Kinesis Data Firehose from Kinesis Data Streams, Amazon CloudWatch, etc.


AWS Lambda


Amazon EMR


AWS Glue


Custom consumers using stream storage-specific SDKs/Client Library

Streaming data options on AWS

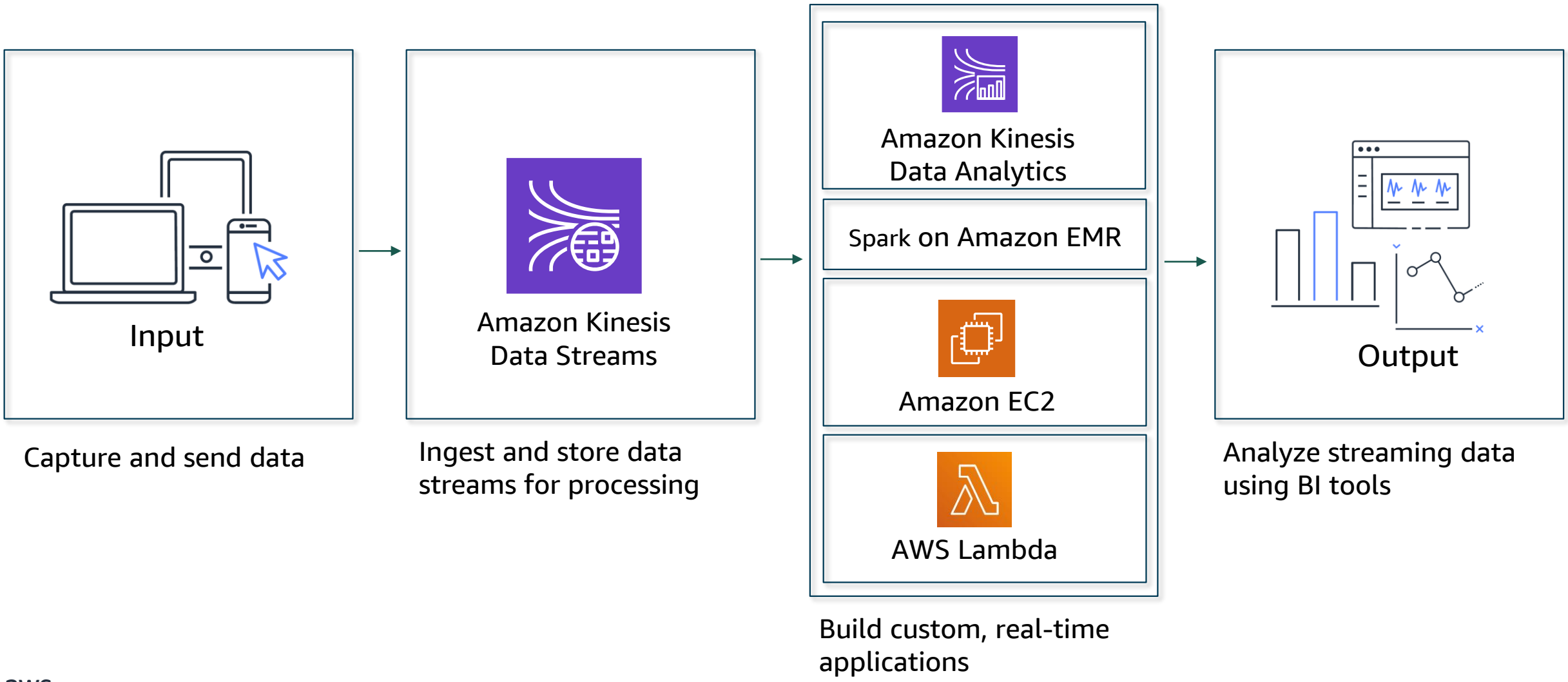
AWS provides several options to work with real-time data streaming:

- **Amazon Kinesis Data Streams** is a scalable and durable real-time data streaming service that can continuously capture gigabytes of data per second from hundreds of thousands of sources.
- **Amazon Data Firehose** captures, transforms, and loads data streams into AWS data stores for near real-time analytics with existing business intelligence tools with just a few clicks.
- **Amazon Managed Service for Apache Flink** transforms and analyzes streaming data in real time with Apache Flink, an open-source framework and engine for processing data streams.
- **Amazon Managed Streaming for Apache Kafka** is a fully managed service that makes it easy for you to build and run applications that use Apache Kafka to process streaming data.

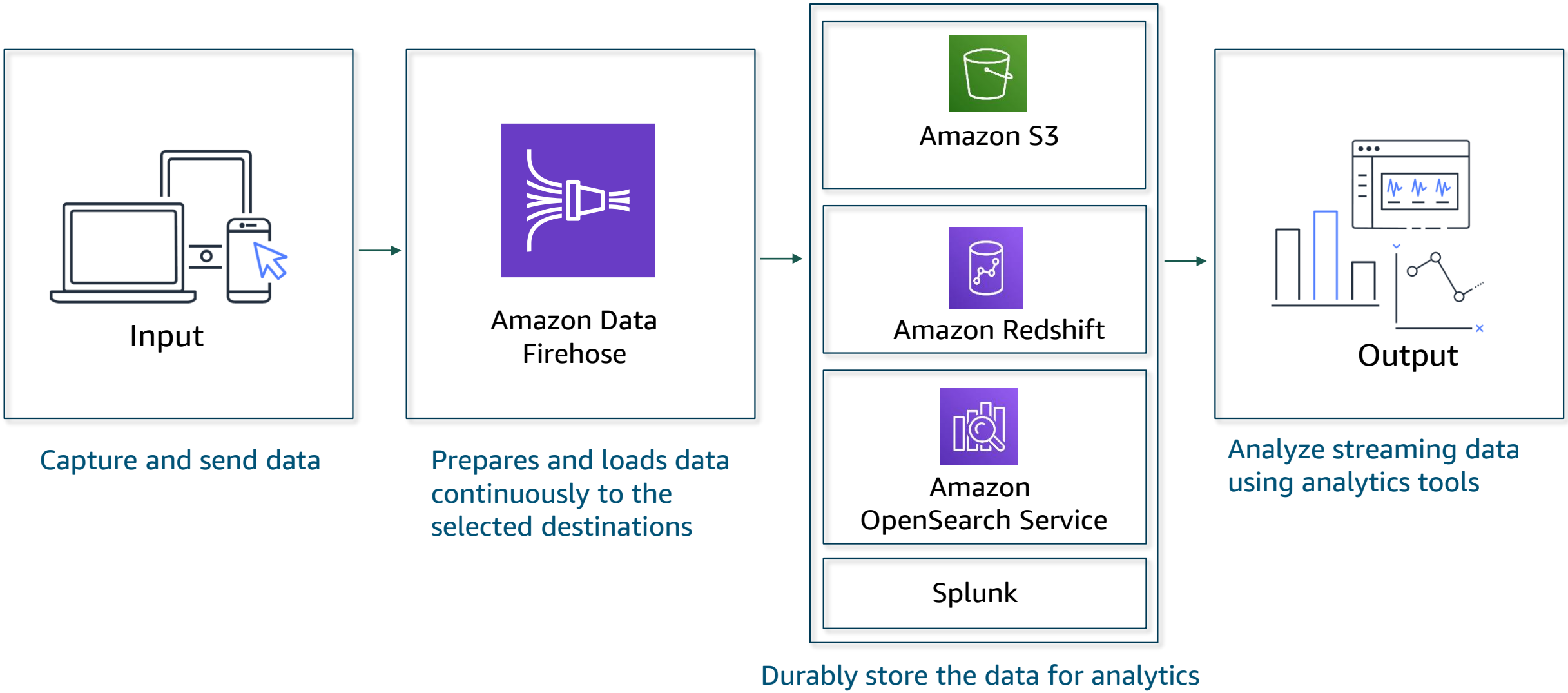
Choosing the right streaming service for your use case

Attribute	Apache Kafka	MSK	Kinesis Streams
Ease of use	Advanced setup required	Get started in minutes	Get started in minutes
Management Overhead	High	Low (Amazon MSK Serverless) to Medium (Amazon MSK Provisioned)	Low
Scalability	Difficult to scale	Scale in minutes with one click	Scale in seconds with one click
Throughput	Very large	Very large	Scale with Kinesis Data Streams on-demand
Infrastructure	You manage	AWS manages	AWS manages
Open-sourced?	Yes	Yes (managed service for Apache Kafka)	No
Latency	Low	Lowest	Low (70ms with Enhance Fan Out)

Amazon Kinesis Data Streams



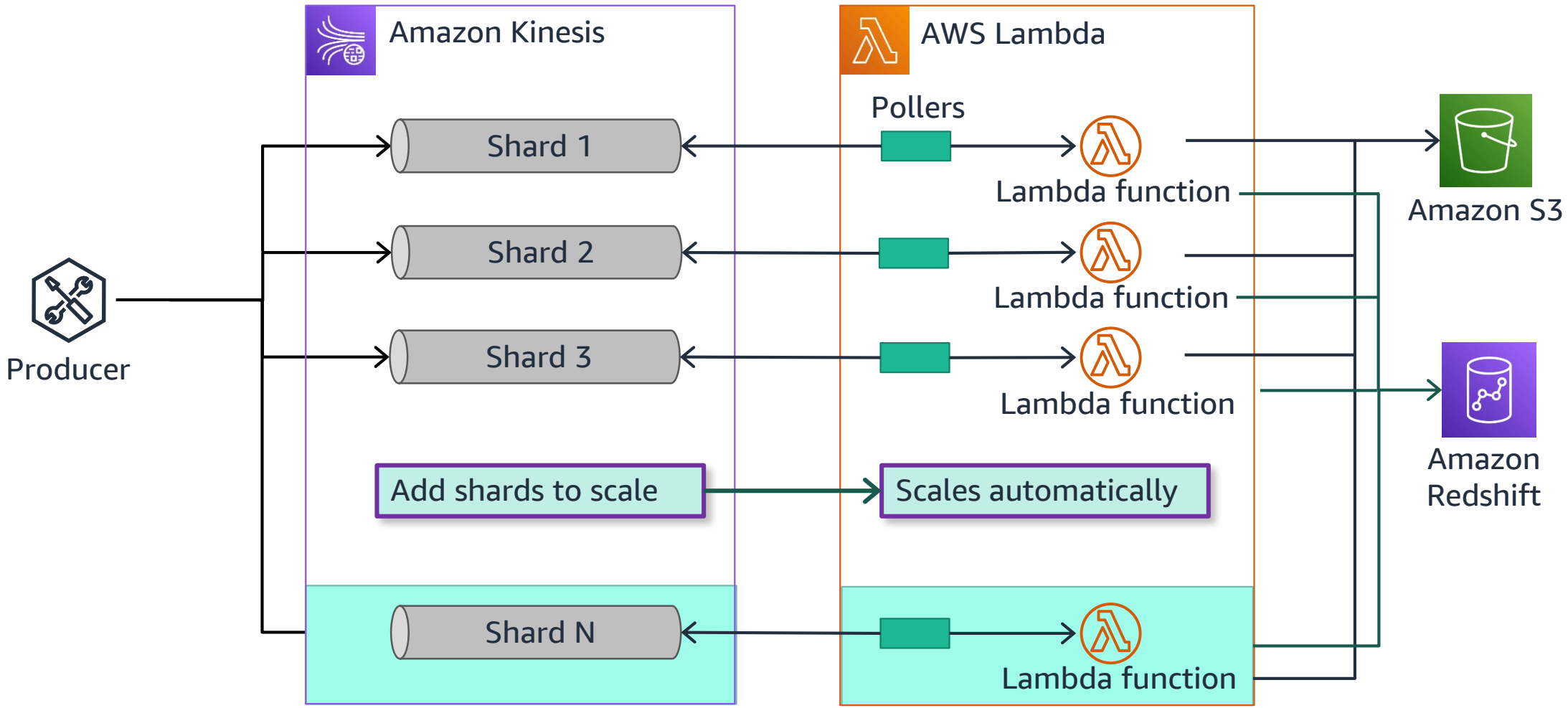
Amazon Data Firehose



Choosing the right service for your use case

Characteristics	Amazon Kinesis Data Streams	Amazon Data Firehose
Processing time	As fast as 70 milliseconds after ingestion	Between 60–900 seconds
Stream storage and duration	In shards, default 24 hours and up to 365 days	Max buffer size 128 MB and max time 900 seconds
Data transformation and conversion	None	Uses AWS Lambda and AWS Glue
Data producer	Amazon Kinesis Agent, applications using Amazon Kinesis Producer Library (KPL), AWS SDK for Java, Amazon CloudWatch Logs and CloudWatch Events, AWS IoT	
Data consumer	AWS Lambda, Amazon Kinesis Data Analytics, Amazon Kinesis Data Firehose, Applications using the Kinesis Client Library (KCL) and SDK for Java	AWS Lambda, Amazon Kinesis Data Analytics, and Kinesis Data Firehose, apps using the KCL and SWK for Java, Amazon S3, Amazon Redshift, Amazon ES, Splunk, and Amazon Kinesis Data Analytics
Data compression	None	gzip, Snappy, Zip, or no data compression

Using AWS Lambda with Amazon Kinesis Data Streams

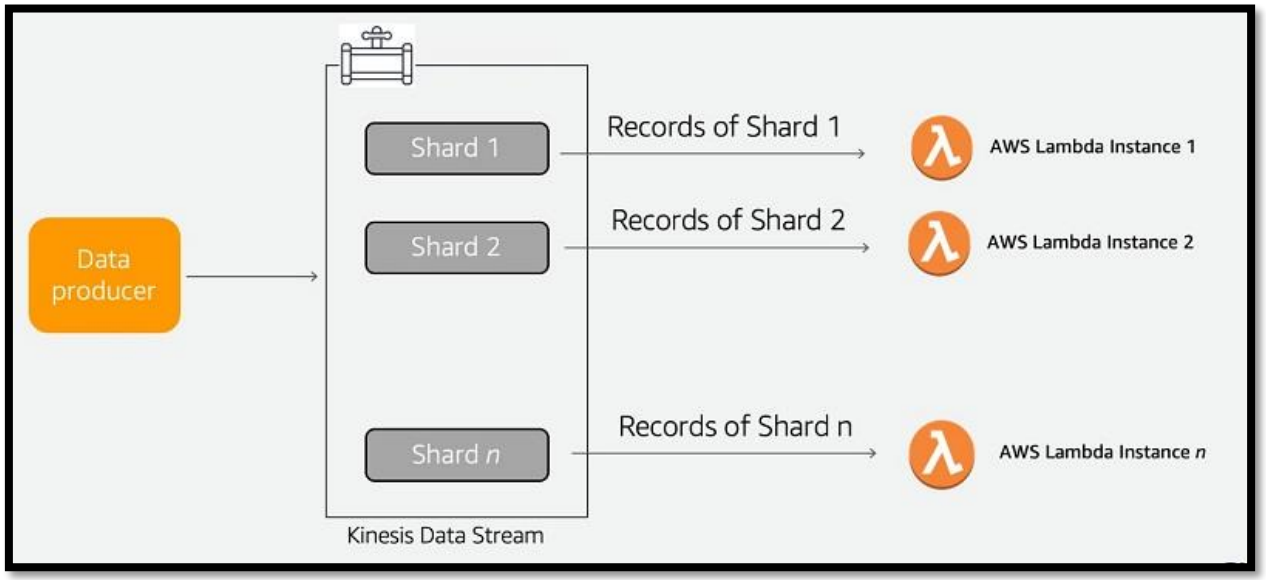


Resharding an Amazon Kinesis Data Stream

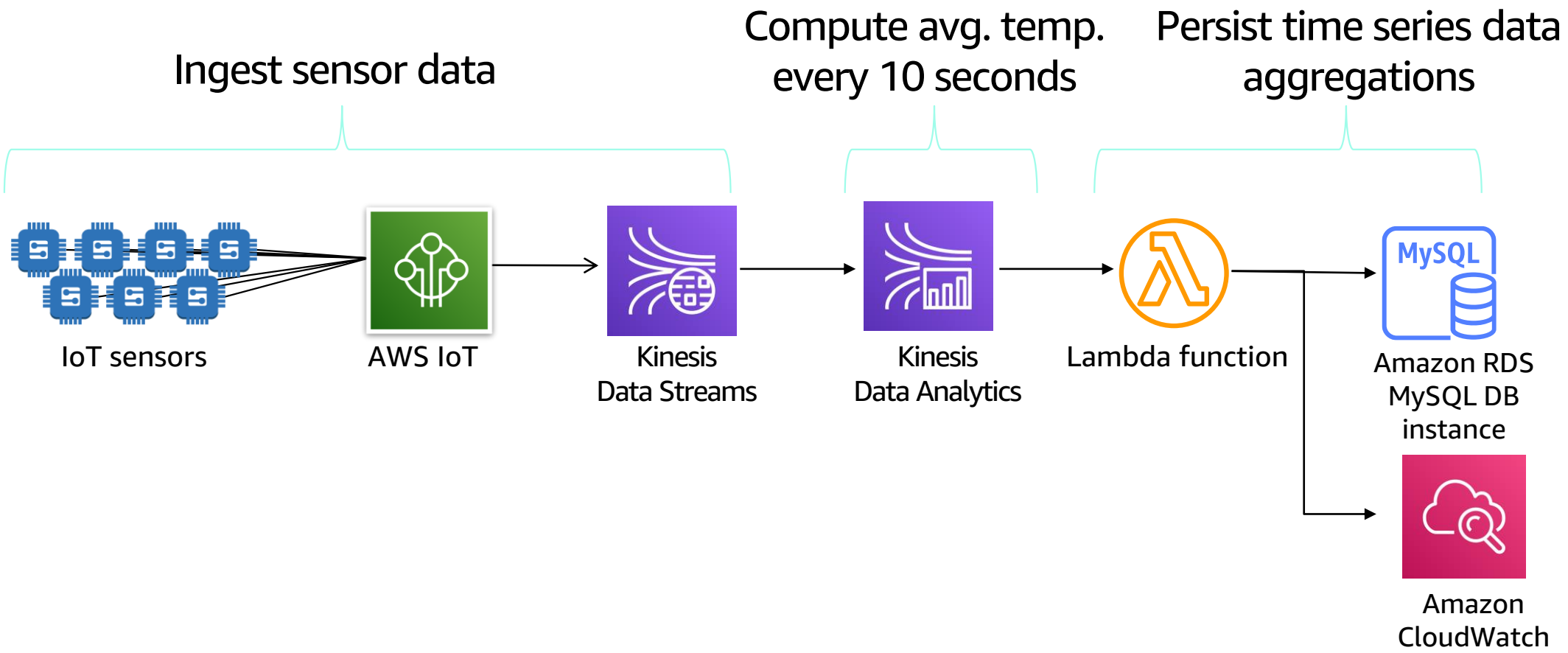
Resharding enables you to increase or decrease the number of shards in a stream in order to adapt to changes in the rate of data flowing through the stream. There are two types of resharding operations: shard **split** and shard **merge**.

- In a shard **split**, you divide a single shard into two shards – to increase the capacity (and cost) of your stream.
- In a shard **merge**, you combine two shards into a single shard – to reduce the cost (and capacity) of your stream.

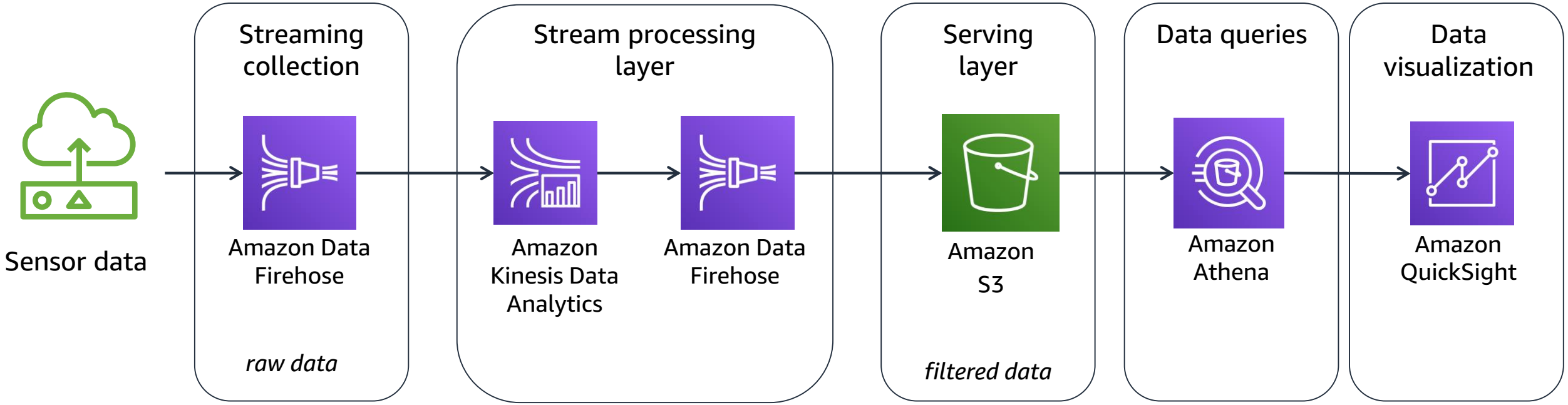
One approach to resharding could be to split every shard in the stream—which would double the stream's capacity. However, this might provide more additional capacity than you actually need and therefore create unnecessary cost.



Example: Using Amazon Kinesis to monitor IoT devices



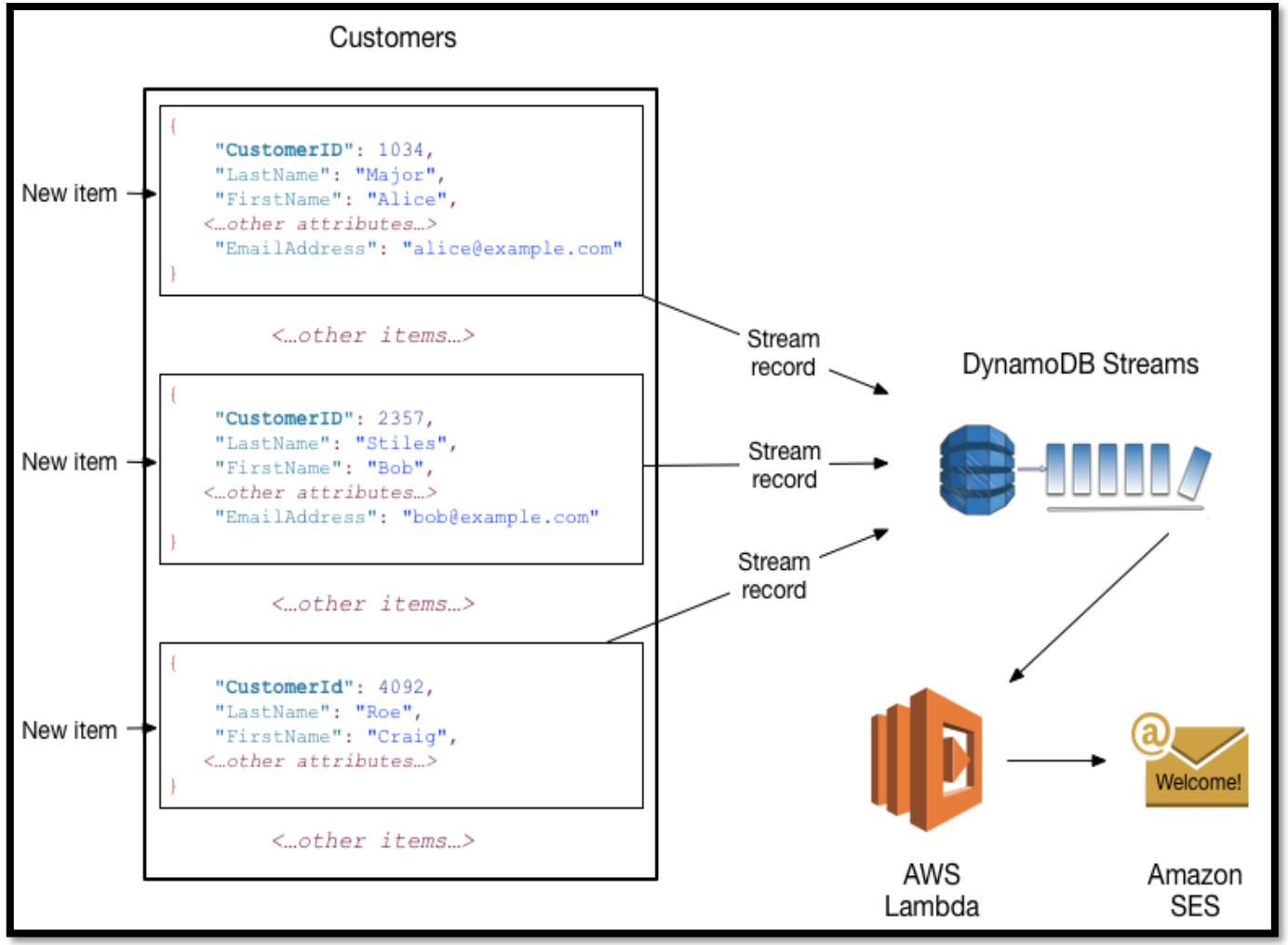
Example: Stream Processing Architecture on AWS



Amazon DynamoDB Streams

Amazon DynamoDB Streams

- DynamoDB Streams is an optional feature that captures data modification events in DynamoDB tables.
- You can use DynamoDB Streams together with AWS Lambda to create a trigger—code that runs automatically whenever an event of interest appears in a stream.



AWS PARTNER CERTIFICATION READINESS

Domain 1: Data Ingestion and Transformation

Transform and Process data

Transform and process data

Knowledge of:

- Creation of ETL pipelines based on business requirements
- Volume, velocity, and variety of data (for example, structured data, unstructured data)
- Cloud computing and distributed computing
- How to use Apache Spark to process data
- Intermediate data staging locations

Skills in:

- Optimizing container usage for performance
- Connecting to different data sources
- Integrating data from multiple sources
- Optimizing costs while processing data
- Implementing data transformation services based on requirements
- Transforming data between formats
- Troubleshooting and debugging common transformation failures and performance issues
- Creating data APIs to make data available to other systems by using AWS services

What is ETL (Extract Transform Load)?

Extract, transform, and load (ETL) is the process of combining data from multiple sources into a large, central repository called a data warehouse. ETL uses a set of business rules to clean and organize raw data and prepare it for storage, data analytics, and machine learning (ML).

Why is ETL important?

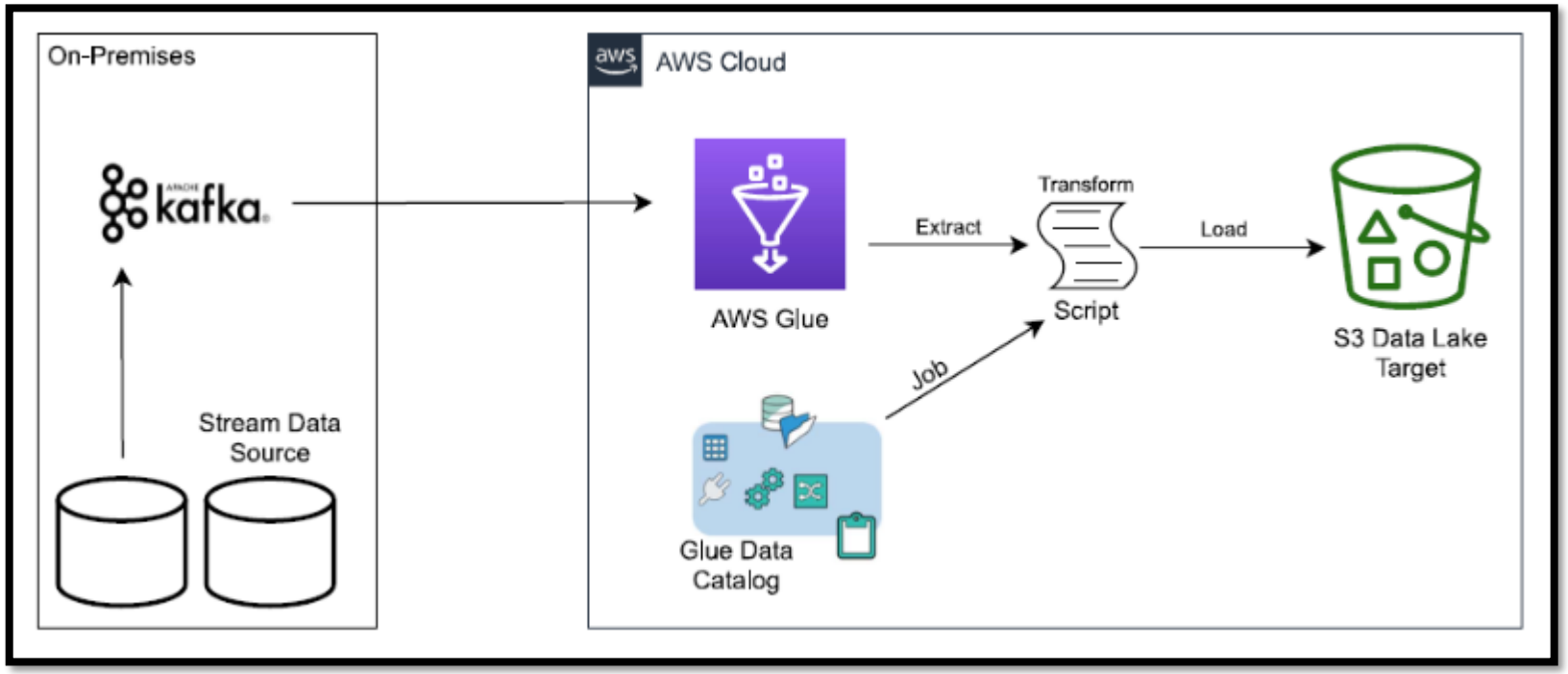
Organizations today have both structured and unstructured data from various sources including:

- Customer data from online payment and customer relationship management (CRM) systems
- Inventory and operations data from vendor systems
- Sensor data from Internet of Things (IoT) devices
- Marketing data from social media and customer feedback

How does ETL work?

Extract, transform, and load (ETL) works by moving data from the source system to the destination system at periodic intervals. The ETL process works in three steps:

- Extract the relevant data from the source database
- Transform the data so that it is better suited for analytics
- Load the data into the target database



What is data extraction?

In **data extraction**, extract, transform, and load (ETL) tools extract or copy raw data from multiple sources and store it in a staging area.

Data extraction commonly happens in one of the three following ways.

Update notification

- In **update notification**, the source system notifies you when a data record changes. You can then run the extraction process for that change.

Incremental extraction

- In this case, the system checks for changes at periodic intervals, such as once a week, once a month, or at the end of a campaign. You only need to extract data that has changed.

Full extraction

- This extraction method requires you to keep a copy of the last extract to check which records are new. Because this approach involves high data transfer volumes, we recommend you use it only for small tables.

What is data transformation?

In **data transformation**, extract, transform, and load (ETL) tools transform and consolidate the raw data in the staging area to prepare it for the target data warehouse.

Basic data transformation

- Basic transformations improve data quality by removing errors, emptying data fields, or simplifying data. Examples of these transformations are:

Data cleansing

Data cleansing removes errors and maps source data to the target data format. For example, you can map empty data fields to the number 0, map the data value “Parent” to “P,” or map “Child” to “C.”

Data deduplication

Deduplication in data cleansing identifies and removes duplicate records.

Data format revision

Format revision converts data, such as character sets, measurement units, and date/time values, into a consistent format. For example, a food company might have different recipe databases with ingredients measured in kilograms and pounds. ETL will convert everything to pounds.

What is data loading?

In **data loading**, extract transform, and load (ETL) tools move the transformed data from the staging area into the target data warehouse. For most organizations that use ETL, the process is automated, well defined, continual, and batch driven. Two methods for loading data are below:

Full load
In full load, the entire data from the source is transformed and moved to the data warehouse. The full load usually takes place the first time you load data from a source system into the data warehouse.

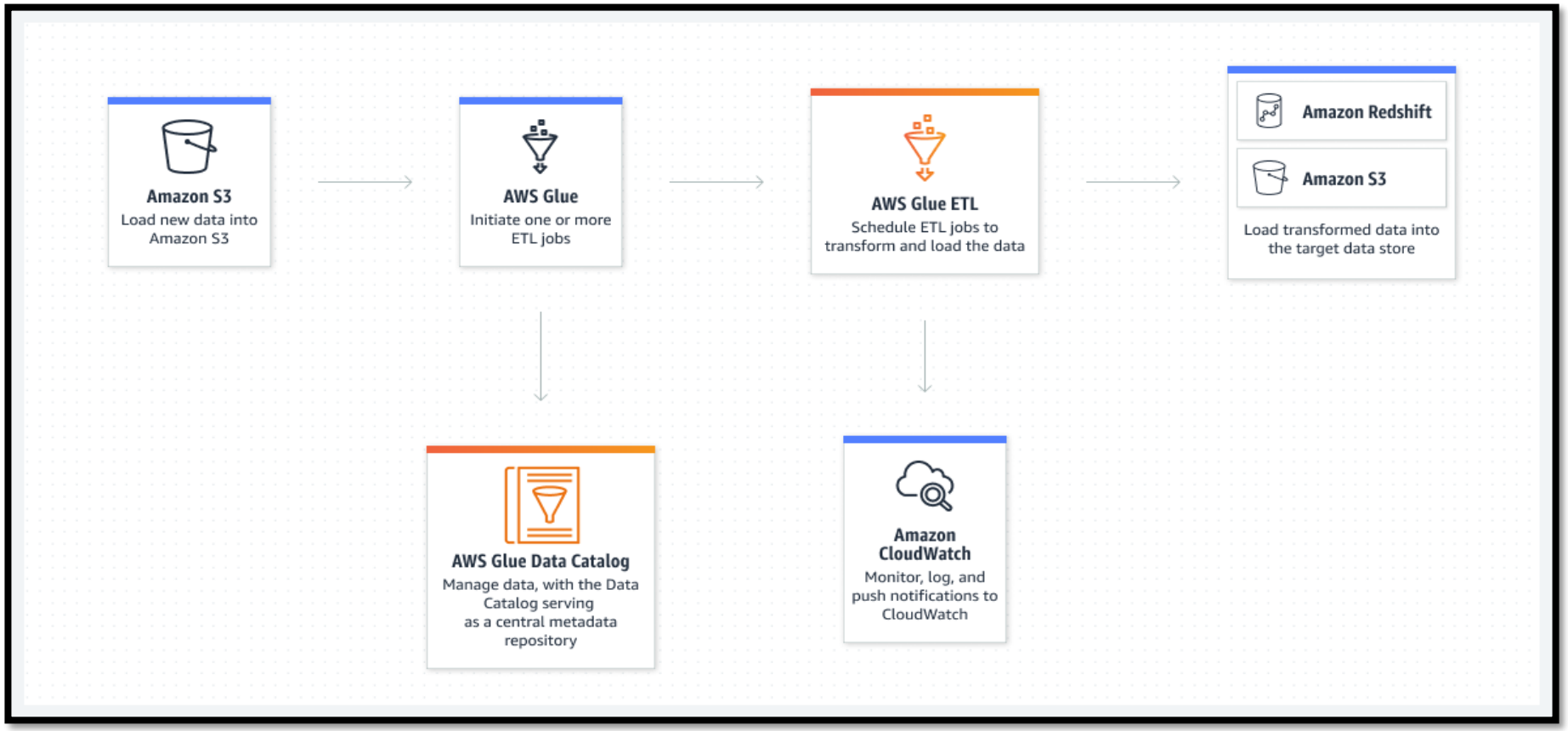
Incremental load
In incremental load, the ETL tool loads the delta (or difference) between target and source systems at regular intervals. There are two ways to implement incremental load:

Streaming incremental load
If you have small data volumes, you can stream continual changes over data pipelines to the target data warehouse. When the speed of data increases to millions of events per second, you can use event stream processing to monitor and process the data streams to make more-timely decisions.

Batch incremental load
If you have large data volumes, you can collect load data changes into batches periodically. During this set period of time, no actions can happen to either the source or target system as data is synchronized.

Using AWS Glue to process your ETL jobs

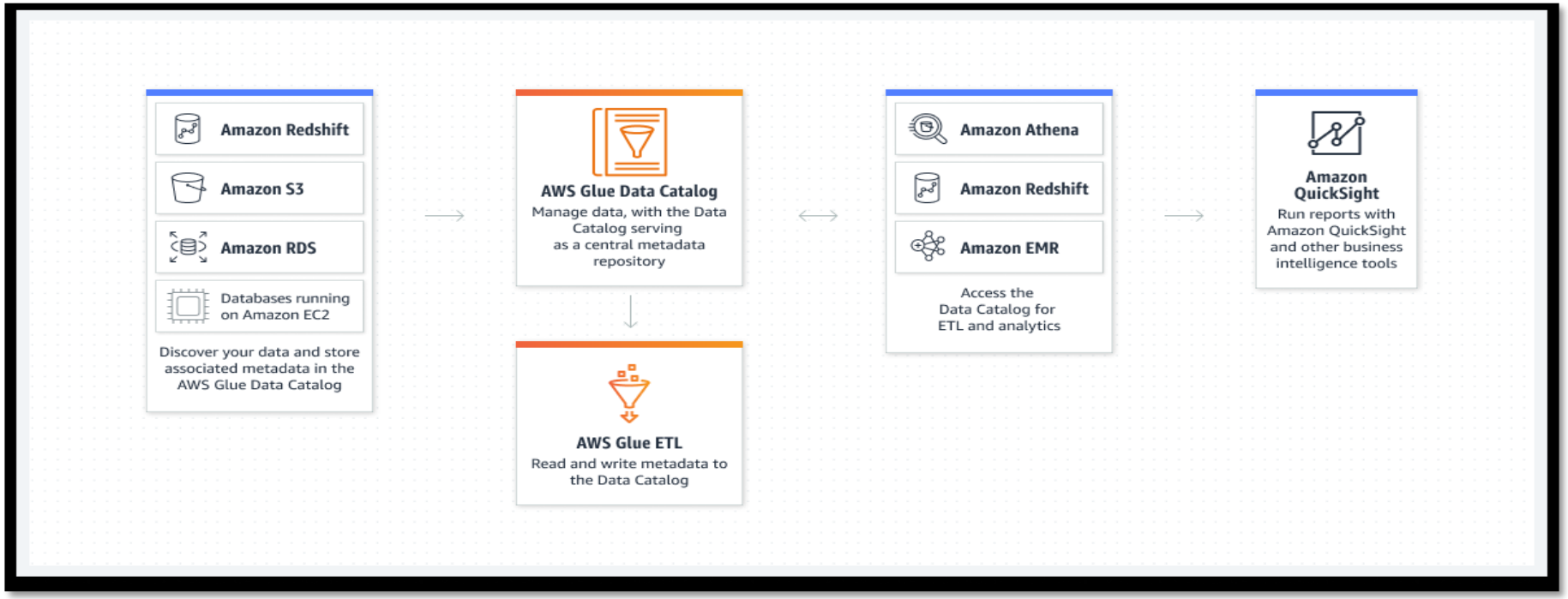
AWS Glue can run your extract, transform, and load (ETL) jobs as new data arrives. For example, you can configure AWS Glue to initiate your ETL jobs to run as soon as new data becomes available in Amazon Simple Storage Service (S3).



Data sources and streams for AWS Glue

- AWS Glue can integrate with more than 80 data sources on AWS, on premises, and on other clouds, such as:
  - Amazon Aurora, Amazon RDS for MySQL, Oracle, PostgreSQL, or SQL Server
  - Amazon Redshift, Amazon DynamoDB, Amazon S3
  - MySQL, Oracle, Microsoft SQL Server, and PostgreSQL

- AWS Glue also supports data streams from Amazon Managed Streaming for Apache Kafka (Amazon MSK), Amazon Kinesis Data Streams, and Apache Kafka.



JDBC or ODBC connections across AWS Services

Many AWS services support integrated AWS-service-to-AWS-service data connectivity

- If you have a need to connect to data outside of a native AWS service (i.e. to on-premises data), there are other data connection options such as ODBC and JDBC:
  - **ODBC (Open Database Connectivity)** is a standard API that allows applications to connect to various databases using a uniform interface. ODBC is widely supported by various DBMS, including Microsoft SQL Server, Oracle, MySQL, PostgreSQL, and many others.
  - **JDBC (Java Database Connectivity)** is a Java-based API that provides a standardized way for Java applications to connect to databases.

Example AWS Services supporting ODBC or JDBC connections

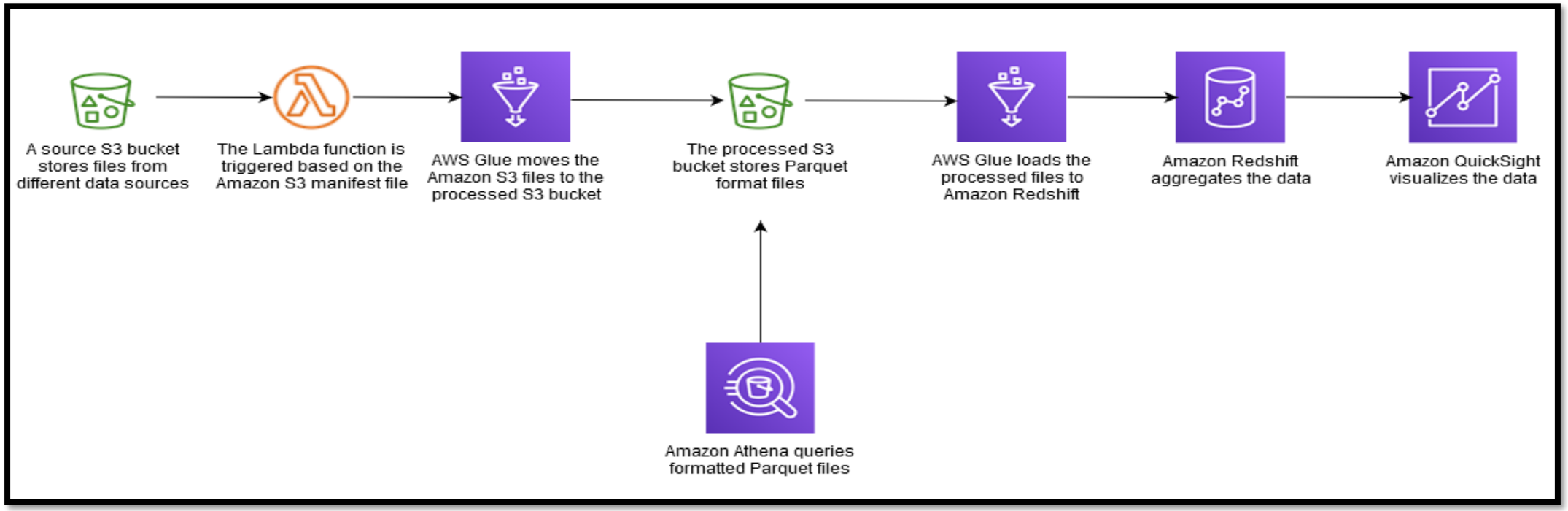
Amazon Redshift, Amazon Athena
- Supports ODBC and JDBC connections

AWS Glue
- Natively supports JDBC connections

Build an ETL pipeline with Amazon S3 and AWS Glue

- This pattern below shows how to load incremental data changes from Amazon S3 into Amazon Redshift by using AWS Glue, performing extract, transform, and load (ETL) operations.

- The source files in Amazon S3 can have different formats, including comma-separated values (CSV), XML, and JSON files. This pattern describes how you can use AWS Glue to convert the source files into a cost-optimized and performance-optimized format like Apache Parquet.



Orchestrate data pipelines

Knowledge of:

- How to integrate various AWS services to create ETL pipelines
- Event-driven architecture
- How to configure AWS services for data pipelines based on schedules or dependencies
- Serverless workflows

Skills in:

- Using orchestration services to build workflows for data ETL pipelines
- Building data pipelines for performance, availability, scalability, resiliency, and fault tolerance
- Implementing and maintaining serverless workflows
- Using notification services to send alerts

Why orchestrate data pipelines on AWS?

AWS data orchestration services provide the scalability, reliability, and availability needed to successfully manage your data processing on AWS.

ETL workflows often involve orchestrating and monitoring the execution of many sequential and parallel data processing tasks.

Simplifying ETL Workflow Management on AWS

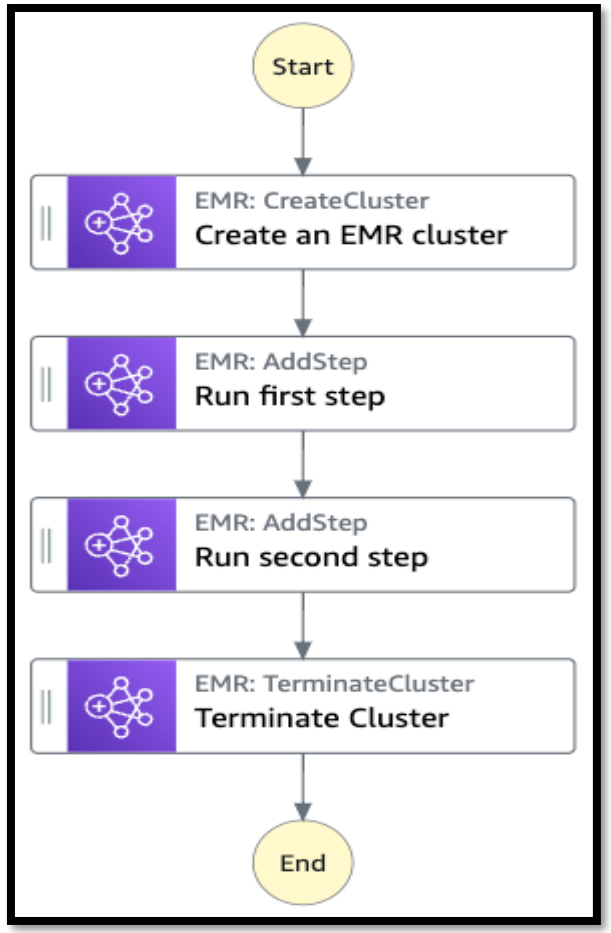
- AWS managed orchestration services such as **AWS Step Functions** and **Amazon Managed Workflows for Apache Airflow (MWAA)** are managed workflow orchestration services that help simplify ETL workflow management that involves a diverse set of technologies.

AWS Step Functions as a data orchestration tool

Depending on your data processing needs, Step Functions directly integrates with other data processing services provided by AWS, such as AWS Batch for batch processing, Amazon EMR for big data processing, AWS Glue for data preparation, Athena for data analysis, and AWS Lambda for compute.

AWS Step Functions with AWS analytics services examples

- You can use AWS Step Functions and the Amazon Redshift Data API together to run an ETL/ELT workflow that loads data into an Amazon Redshift data warehouse
- You can also integrate Amazon EMR and AWS Step Functions together (example on the right)



AWS Step Functions workflow types

With AWS Step Functions you can process data faster using parallel transformations or dynamic parallelism, and it lets you easily retry failed transformations, or choose a specific way to handle errors without the need to manage a complex process.

Step Functions has two workflow types:

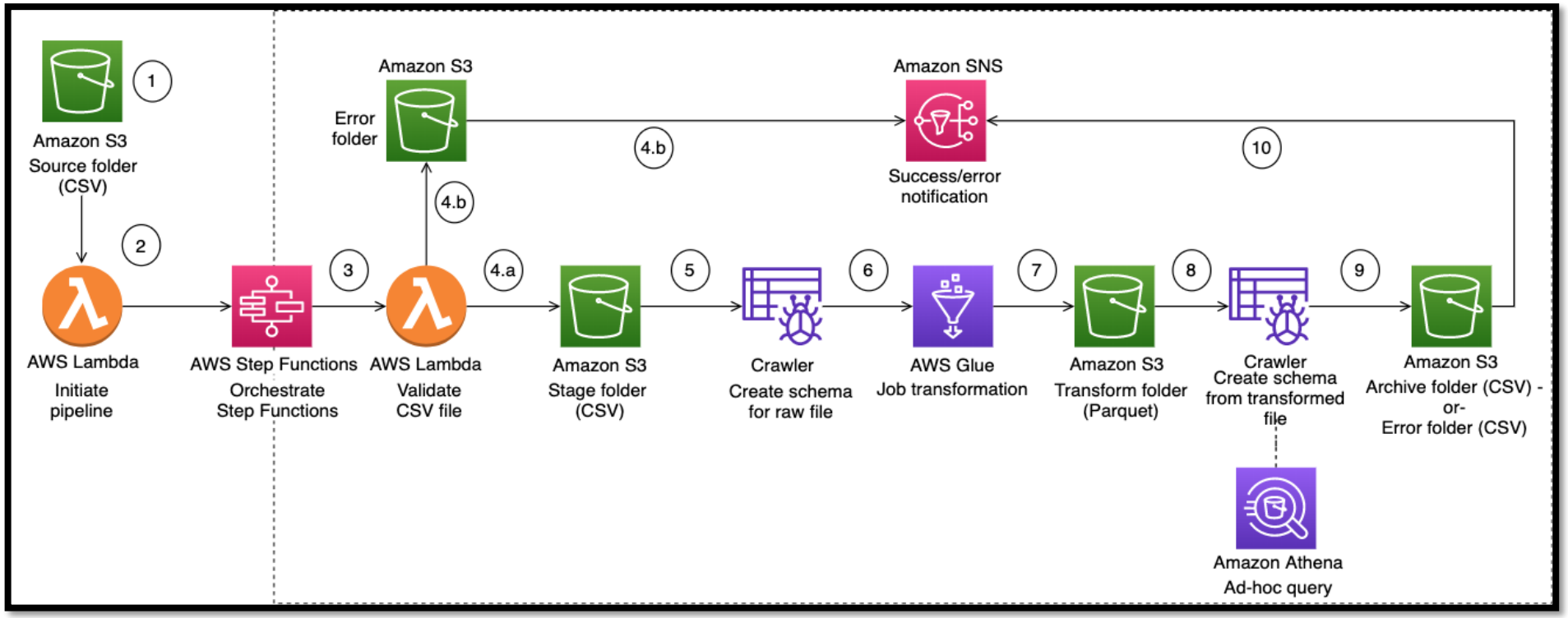
- **Standard Workflows** have exactly-once workflow transformation, and can run for up to one year.
- **Express Workflows** have at-least-once workflow transformation, and can run for up to five minutes.

Which workflow type to use?

- Standard Workflows are ideal for long-running, auditable workflows, as they show execution history and visual debugging.
- Express Workflows are ideal for high-event-rate workloads, such as streaming data processing.

ETL pipeline orchestration with AWS Step Functions

- This pattern below describes a serverless extract, transform, and load (ETL) pipeline to validate, transform, compress, and partition a large CSV dataset for performance and cost optimization.
- The pipeline is orchestrated by AWS Step Functions and includes error handling, automated retry, and user notification features.

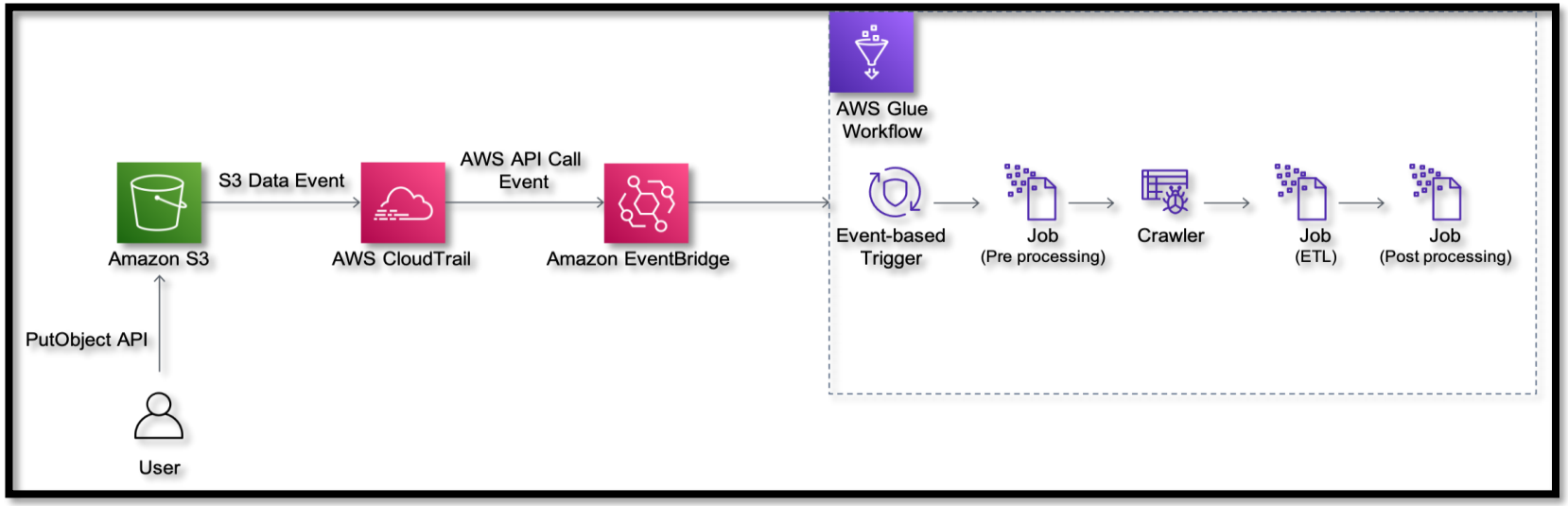


- When a CSV file is uploaded to an Amazon Simple Storage Service (Amazon S3) bucket source folder, the ETL pipeline starts to run.

Serverless event-driven workflow example

Here's an AWS Glue workflow that listens to S3 PutObject data events captured by AWS CloudTrail.

This workflow is configured to run when five new files are added or the batching window time of 900 seconds expires after first file is added. The following diagram illustrates the architecture.



The flow on the left would follow these steps:

1. Create an AWS Glue workflow with a starting trigger of EVENT type and configure the batch size on the trigger to be five and batch window to be 900 seconds.
2. Configure Amazon S3 to log data events, such as PutObject API calls to CloudTrail.
3. Create a rule in EventBridge to forward the PutObject API events to AWS Glue when they are emitted by CloudTrail.
4. Add an AWS Glue event-driven workflow as a target to the EventBridge rule.
5. To start the workflow, upload files to the S3 bucket.

Data pipelines on AWS

An efficient and well-designed data integration pipeline is critical for making the data available and trusted amongst the analytics consumers.

Here are some considerations to review when designing data pipelines:

Factor	AWS Glue Workflow	AWS Step Function	Amazon Managed Workflow for Apache Airflow (MWAA)
Use case	Suitable when your pipeline consists of mostly AWS Glue jobs and crawlers.	Suitable when there is a need to integrate with different services, including AWS Lambda, SSM, and so on.	Compatible with open-source Airflow and suitable when you want to reuse existing Airflow assets.
Infrastructure	Serverless	Serverless	Managed service
Building a data pipeline	Build a data pipeline using an AWS Glue job written in Python or /Scala and crawlers.	Build a data pipeline using the Step Functions console. Possible to integrate with non-supported services using Lambda.	Workflows are created as DAGs, which are defined within a Python file that defines the DAG's structure as code.

Apply programming concepts

Knowledge of:

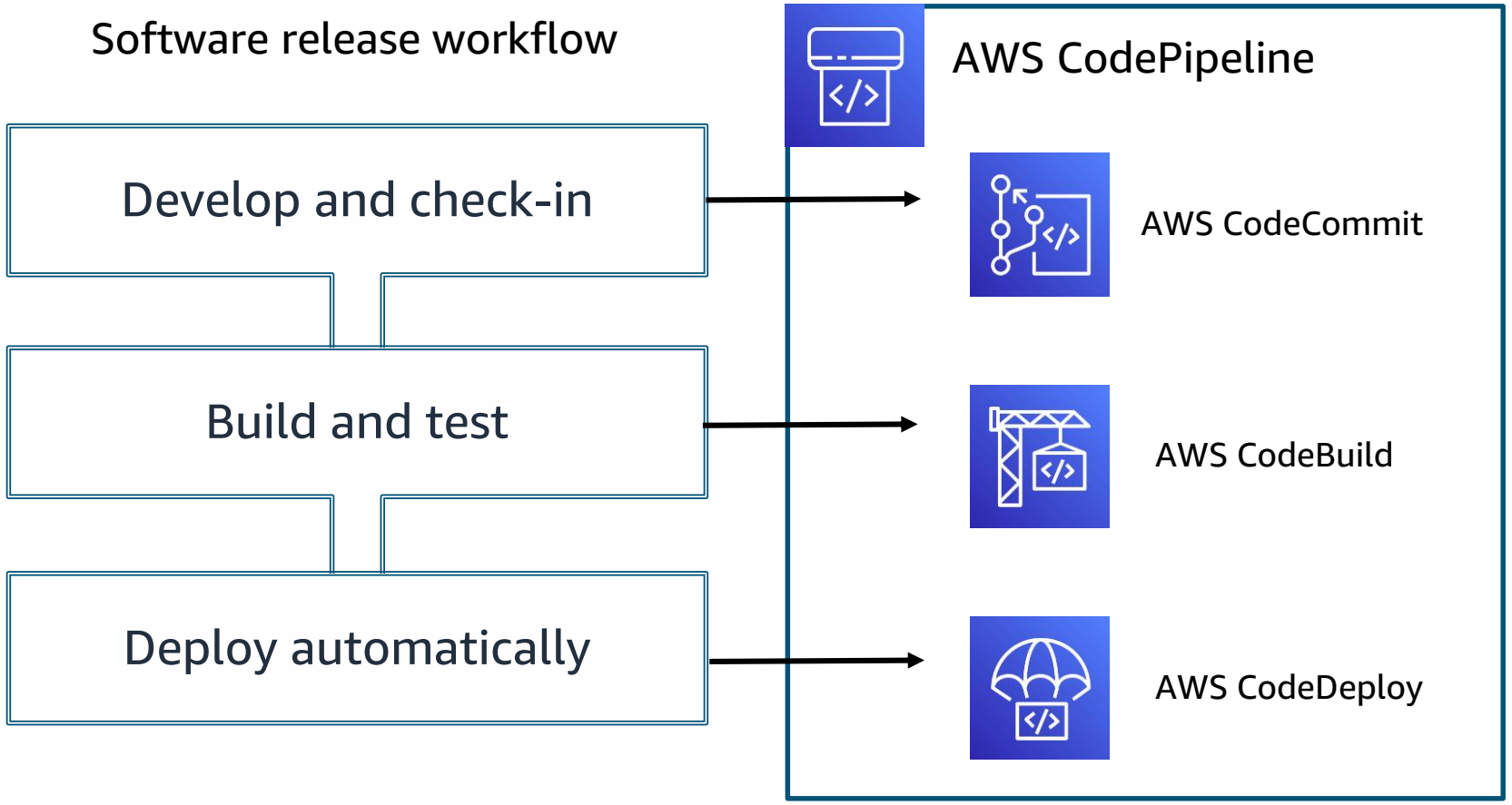
- Continuous integration and continuous delivery (CI/CD) (implementation, testing, and deployment of data pipelines)
- SQL queries (for data source queries and data transformations)
- Infrastructure as code (IaC) for repeatable deployments
- Distributed computing
- Data structures and algorithms
- SQL query optimization

Skills in:

- Optimizing code to reduce runtime for data ingestion and transformation
- Configuring Lambda functions to meet concurrency and performance needs
- Performing SQL queries to transform data and meet data pipeline requirements
- Using Git commands to perform actions such as creating, updating, cloning, and branching repositories
- Using the AWS Serverless Application Model (AWS SAM) to package and deploy serverless data pipelines
- Using and mounting storage volumes from within Lambda functions

Continuous integration and continuous delivery (CI/CD)

A pipeline helps you automate steps in your software delivery process, such as initiating automatic builds and then deploying to Amazon EC2 instances.

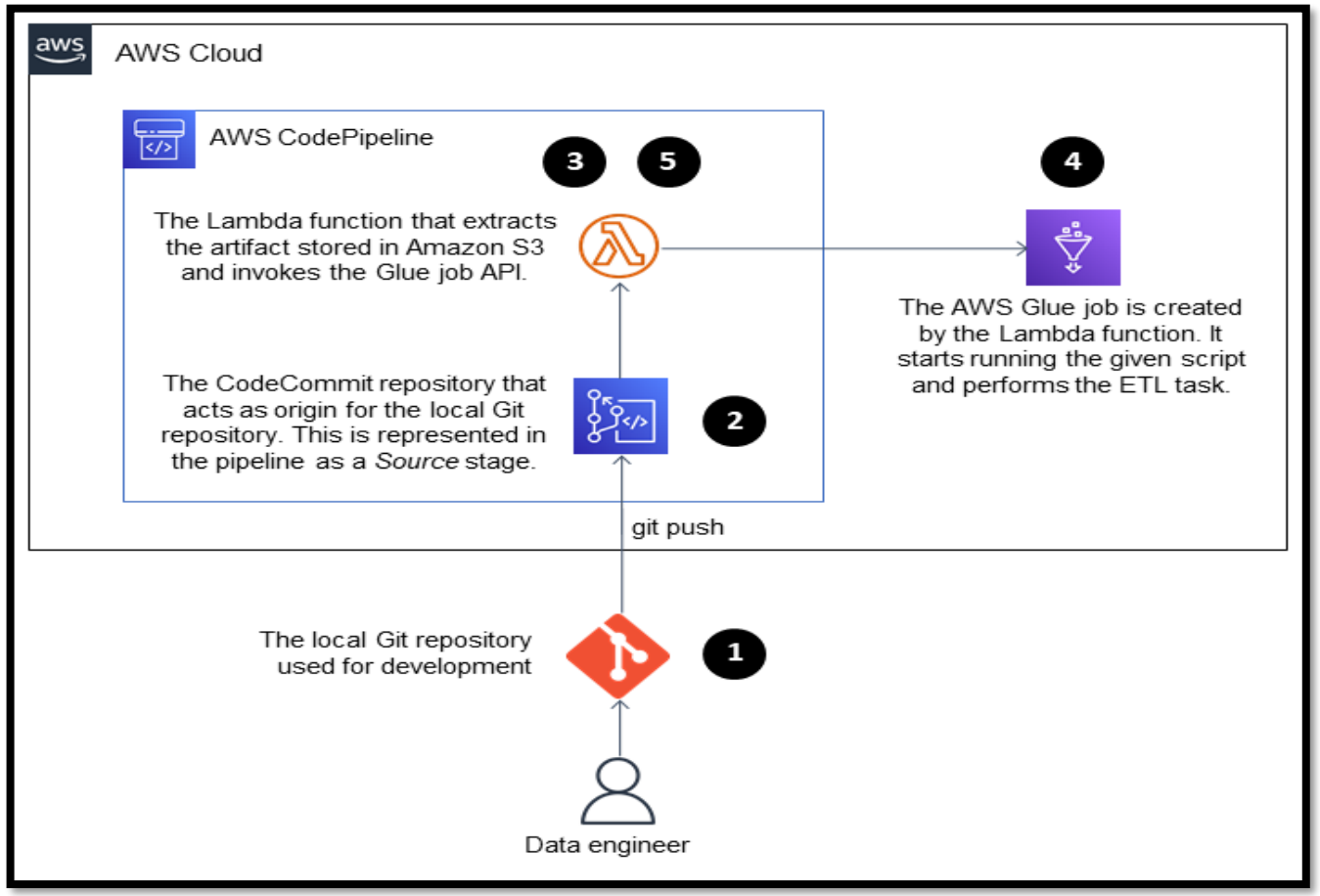


Deploy an AWS Glue job with AWS CodePipeline CI/CD

This pattern is helpful in the situation where businesses, developers, and data engineers want to launch jobs as soon as changes are committed and pushed to the target repositories. It helps achieve a higher level of automation and reproducibility, therefore avoiding errors during the job launch and lifecycle.

The process on the right consists of these steps:

1. The developer or data engineer makes a modification in the ETL code, commits, and pushes the change to AWS CodeCommit.
2. The push initiates the pipeline.
3. The pipeline initiates a Lambda function, which calls codecommit:GetFile on the repository and uploads the file to Amazon Simple Storage Service (Amazon S3).
4. The Lambda function launches a new AWS Glue job with the ETL code.
5. The Lambda function finishes the pipeline.



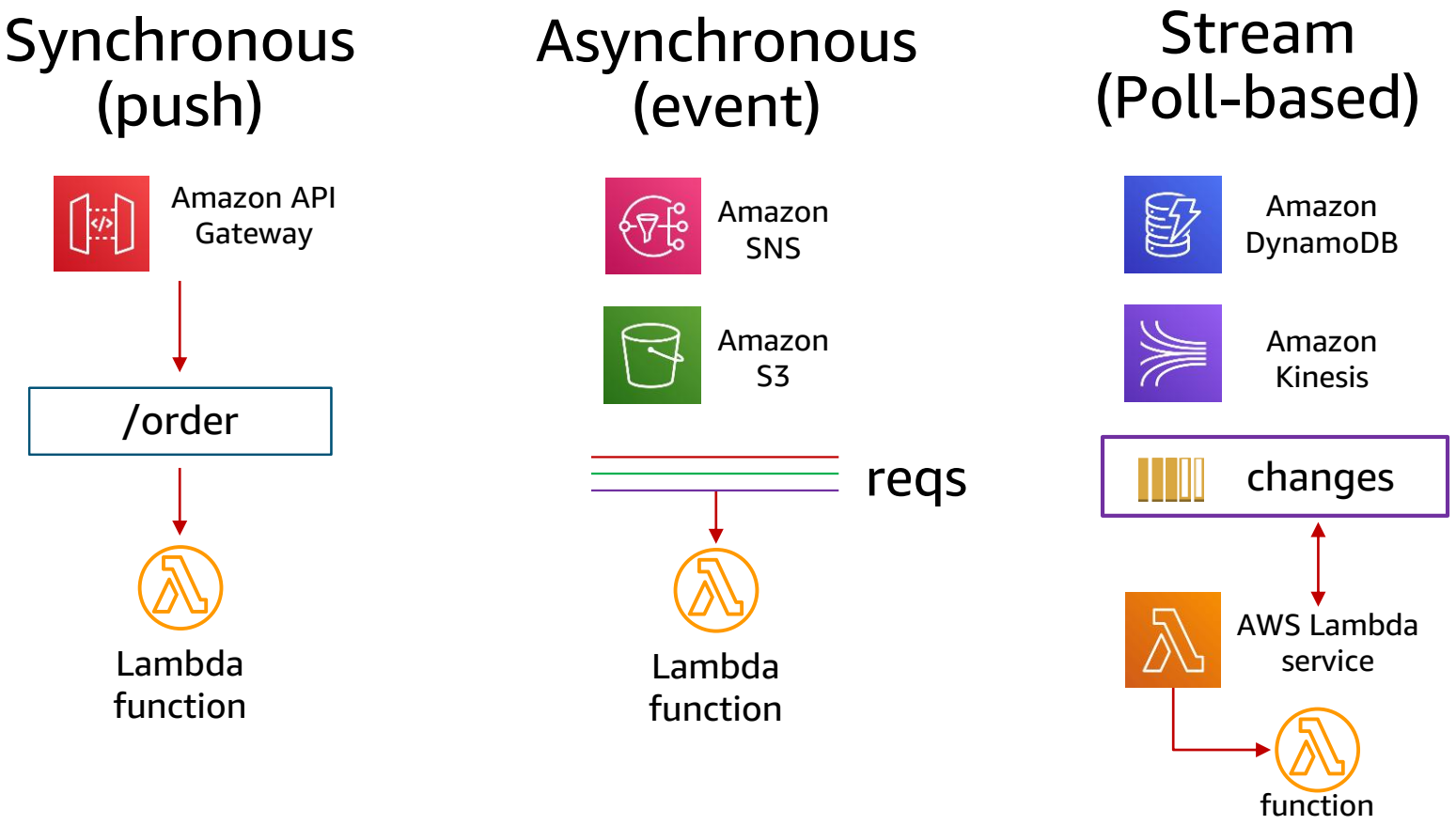
Optimizing AWS Lambda functions in your environment

Memory is the principal lever available to Lambda developers for controlling the performance of a function.

- The **amount of memory also determines the amount of virtual CPU available to a function**. Adding more memory proportionally increases the amount of CPU, increasing the overall computational power available.
- If a function is CPU-, network- or memory-bound, then changing the memory setting can dramatically improve its performance.
- For example, 1000 invocations of a function that computes prime numbers may have the following average durations at different memory levels:

Memory	Duration	Cost
128 MB	11.722 s	$0.024628
256 MB	6.678 s	$0.028035
512 MB	3.194 s	$0.026830
1024 MB	1.465 s	$0.024638

AWS Lambda execution models



Best practices for preventing Lambda function timeouts

Verify that your Lambda function has enough system resources

→ The amount of **network bandwidth and CPU allocated** to a Lambda function invocation is determined by the function's memory configuration

Verify that your Lambda function is configured to work within the maximum timeout settings of any integrated AWS service

→ An AWS Lambda function's **maximum invocation timeout limit is 15 minutes**

Configure provisioned concurrency for your Lambda function

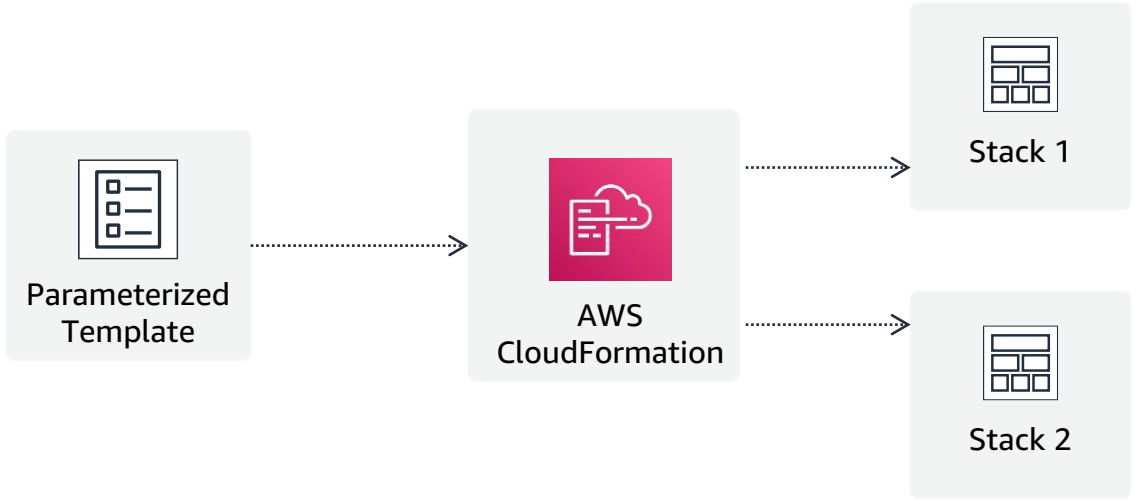
→ Concurrency is the number of requests your function can handle at the same time

→ Provisioned concurrency initializes a requested number of runtime environments so that they're prepared to respond immediately to your function's invocations

Infrastructure-as-code on AWS options

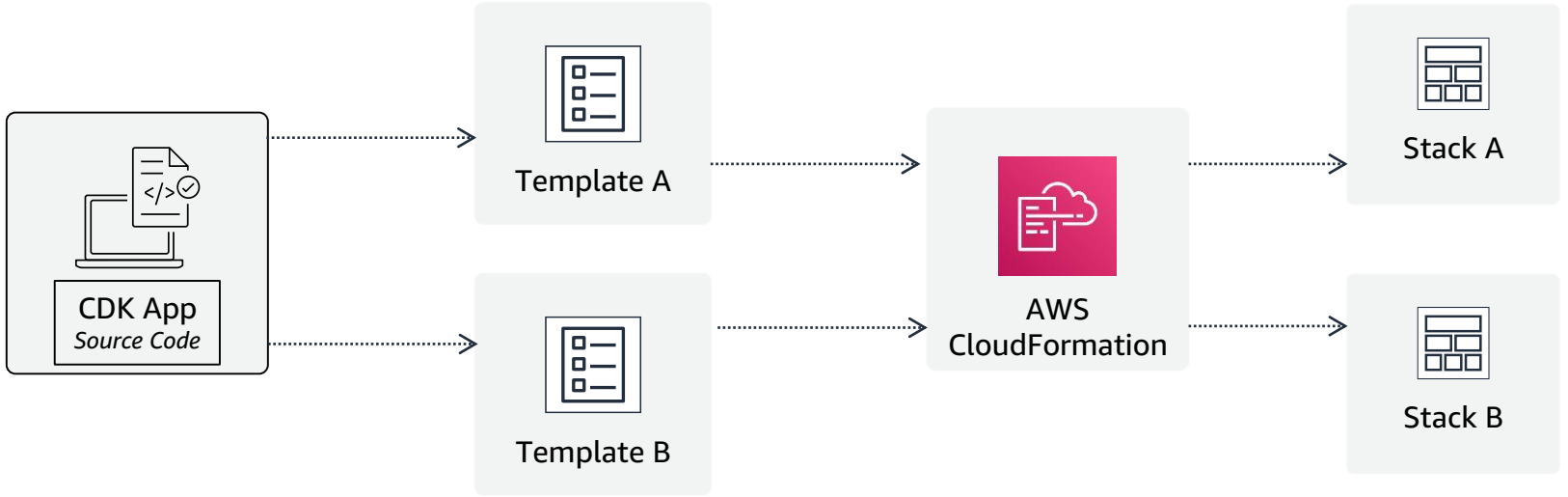
AWS CloudFormation

Parameters and intrinsic functions



AWS CDK

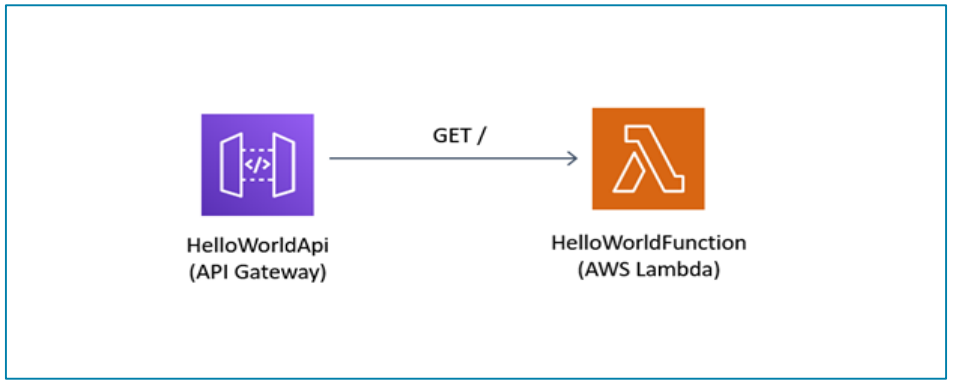
Put your infrastructure, application code, and configuration all in one place



AWS Serverless Application Model (AWS SAM)

The AWS Serverless Application Model (AWS SAM) is an open-source framework that you can use to build serverless applications on AWS.

- The example below consists of an Amazon API Gateway endpoint and an AWS Lambda function.
- When you send a GET request to the API Gateway endpoint, the Lambda function is invoked.



The below is a preview of **commands** that you could use to initialize, build and deploy an application using SAM

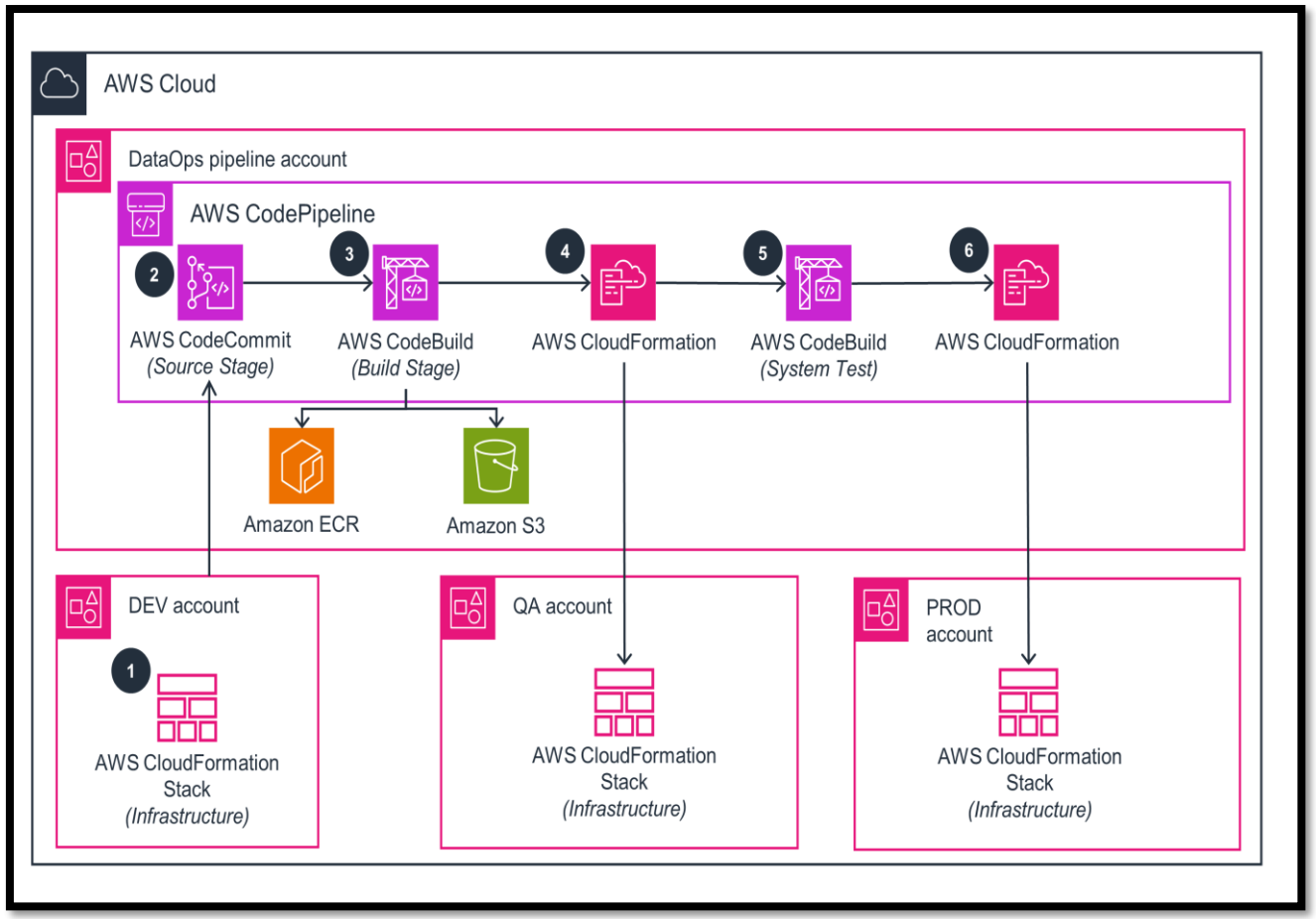
```
#Step 1 - Download a sample application
sam init

#Step 2 - Build your application
cd sam-app
sam build

#Step 3 - Deploy your application
sam deploy --guided
```

Game Analytics Pipeline on AWS Example

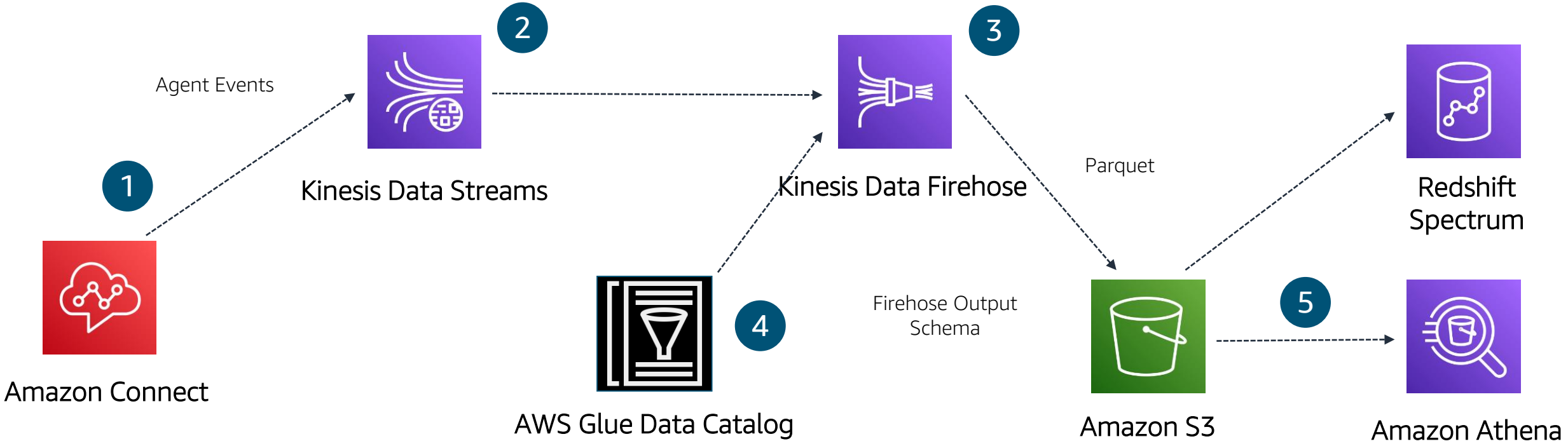
This architecture diagram shows the DataOps CI/CD pipeline for centralized game analytics on AWS



Steps to Build

1. Build and test the codified infrastructure using the AWS Cloud Development Kit (AWS CDK) to synthesize an AWS CloudFormation template.
2. Initiate the CI/CD pipeline when infrastructure code changes are committed to the AWS CodeCommit repository.
3. Store compiled infrastructure assets, such as a Docker container and CloudFormation templates, in Amazon Elastic Container Registry (Amazon ECR) and Amazon S3.
4. Deploy the infrastructure for integration and system testing into the quality assurance (QA) AWS account using the CloudFormation Stack.
5. Run automated testing scripts to verify that the deployed infrastructure is functional inside an AWS CodeBuild project.
6. Deploy the tested infrastructure into the Production (PROD) AWS account using the CloudFormation Stack

Enable faster queries compared to row-oriented formats like JSON



- Convert the format of your input data from JSON to columnar data format **Apache Parquet** or **Apache ORC** before storing the data in Amazon S3.
- Works in conjunction to the transform features to convert other format to JSON before the data conversion

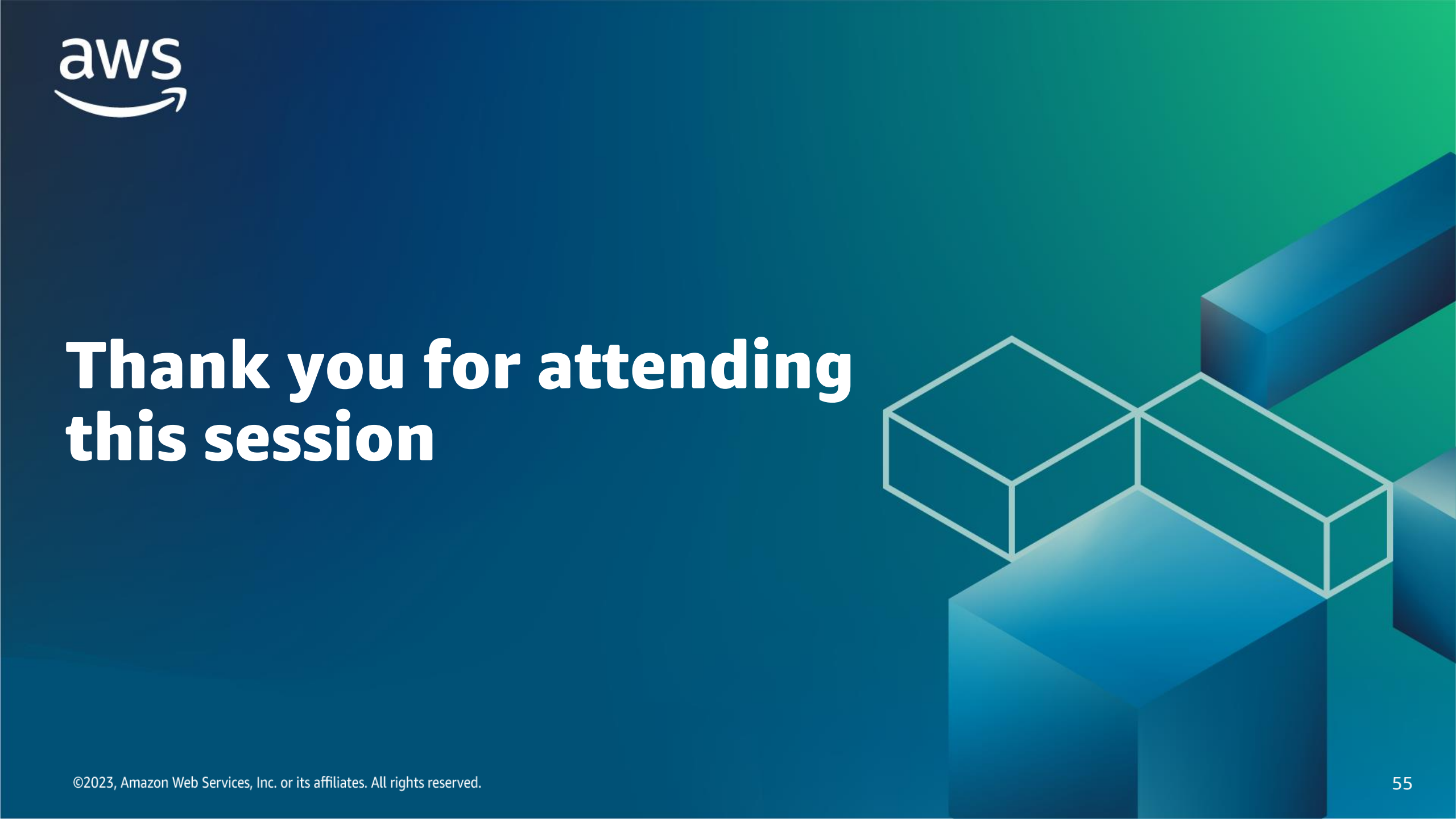
SQL query optimizations on Amazon Redshift

Amazon Redshift is built around industry-standard SQL, with added functionality to manage very large datasets and support high-performance analysis and reporting of those data.

To maximize query performance, follow these recommendations when creating queries:

- Avoid using select *. Include only the columns you specifically need.
- Don't use cross-joins unless absolutely necessary. These joins without a join condition result in the Cartesian product of two tables. Cross-joins are typically run as nested-loop joins, which are the slowest of the possible join types.
- Use subqueries in cases where one table in the query is used only for predicate conditions and the subquery returns a small number of rows (less than about 200). The following example uses a subquery to avoid joining the LISTING table.

```
select sum(sales.qtysold)
from sales
where salesid in (select listid from listing where listtime > '2008-12-26');
```



Thank you for attending this session