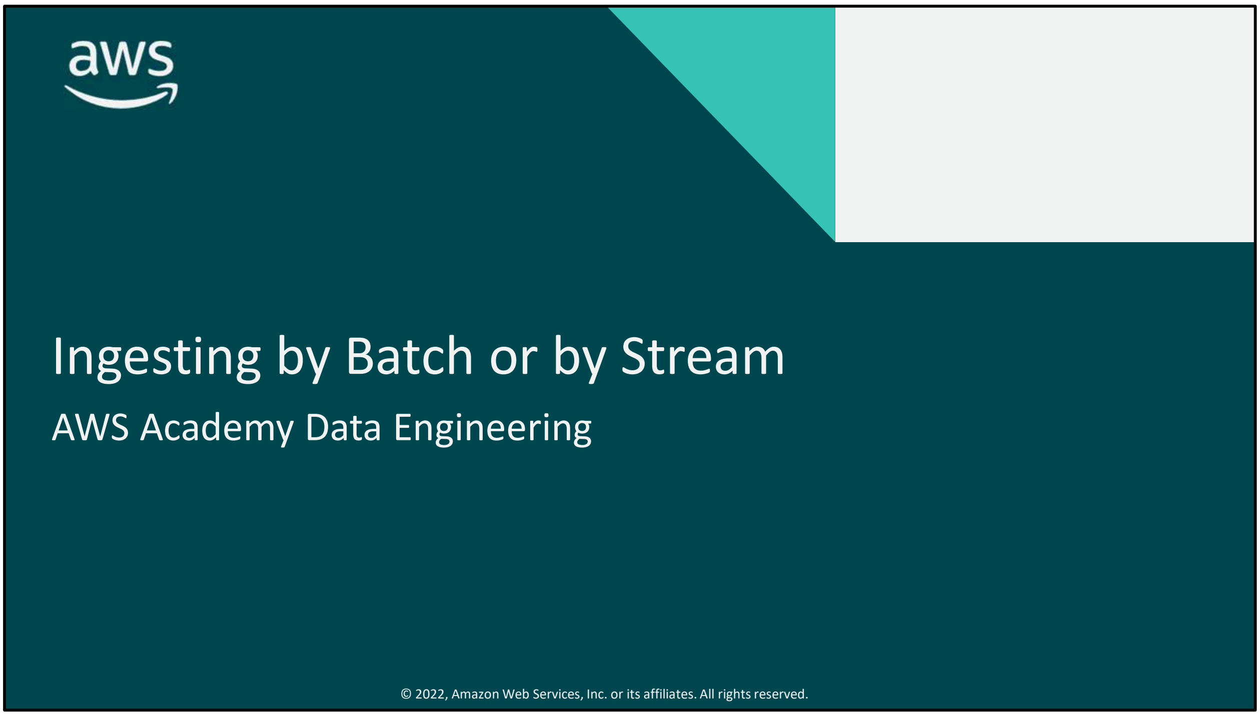


Ingesting by Batch or by Stream

AWS Academy Data Engineering

|Slide number 1

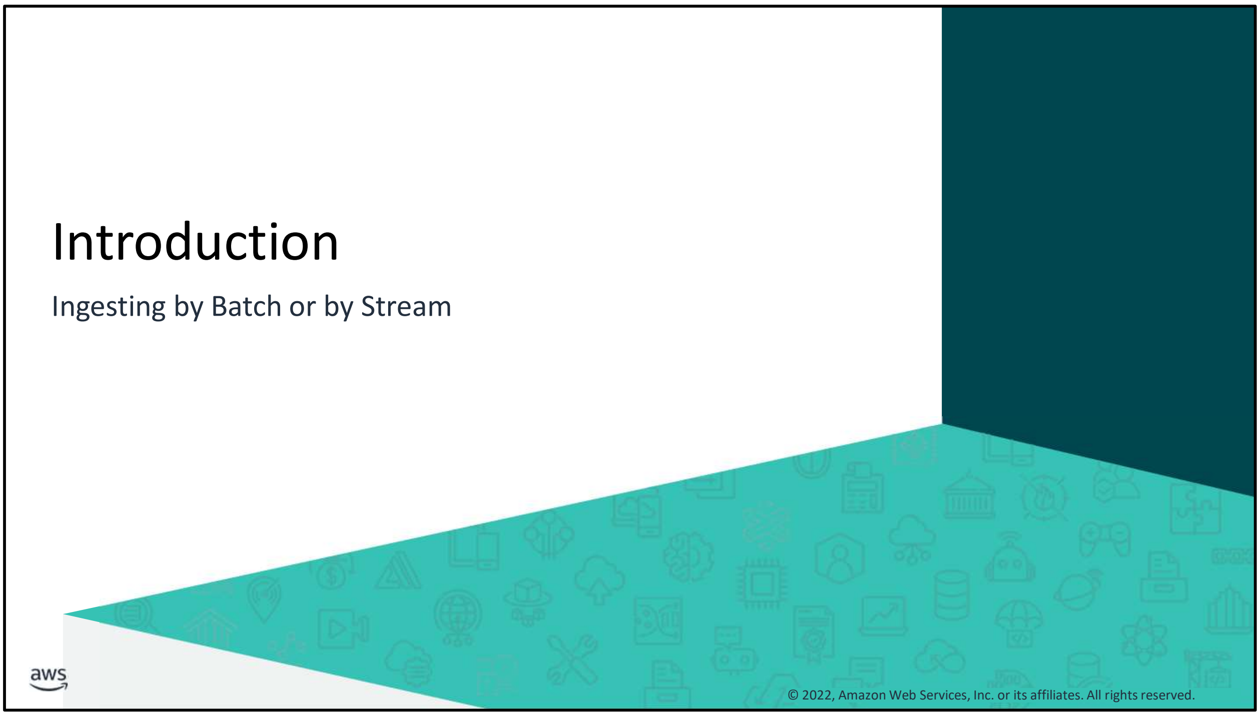
|Instructor notes

|This module extends the topics from module 6. If you skipped module 6, you might do a bit more context setting at the beginning of this module with respect to comparing ETL and ELT and to the ingestion layer of the pipeline.

|

|Student notes

This module focuses on concepts that are related to ingesting data by batch or by stream.

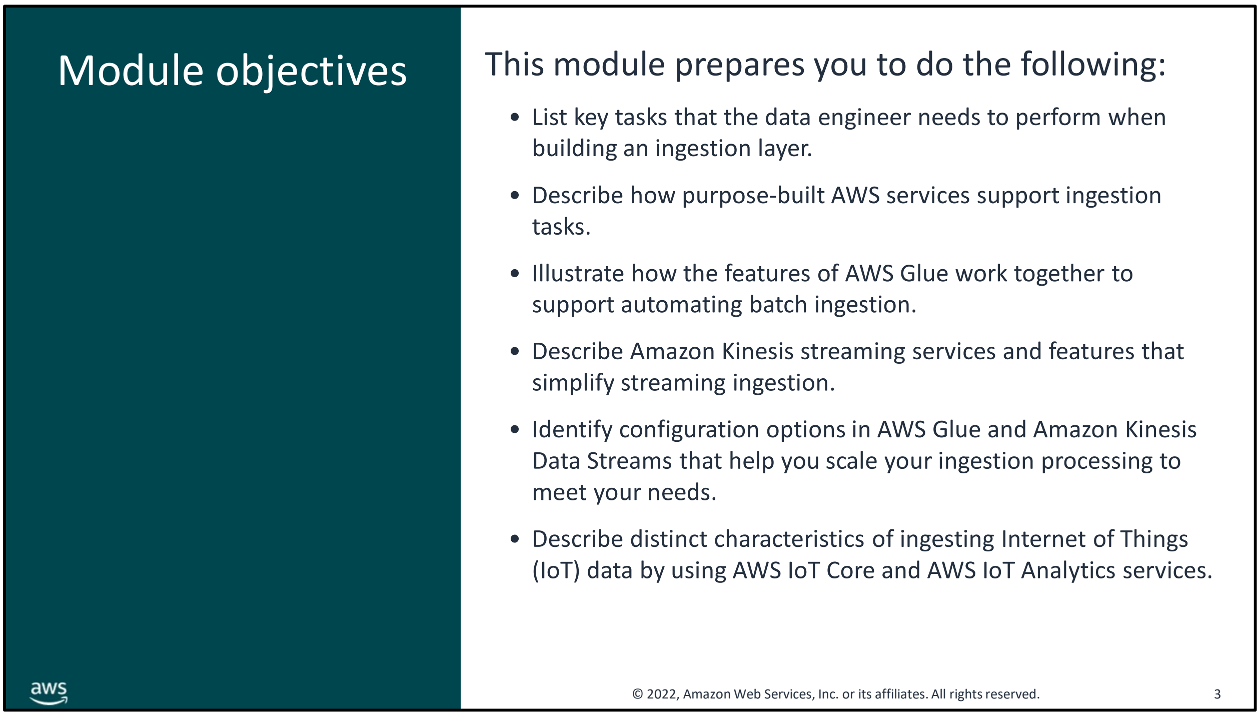


Introduction

Ingesting by Batch or by Stream

|Slide number 2
|Instructor notes
|
|Student notes

This introduction section describes the content of this module.



Module objectives

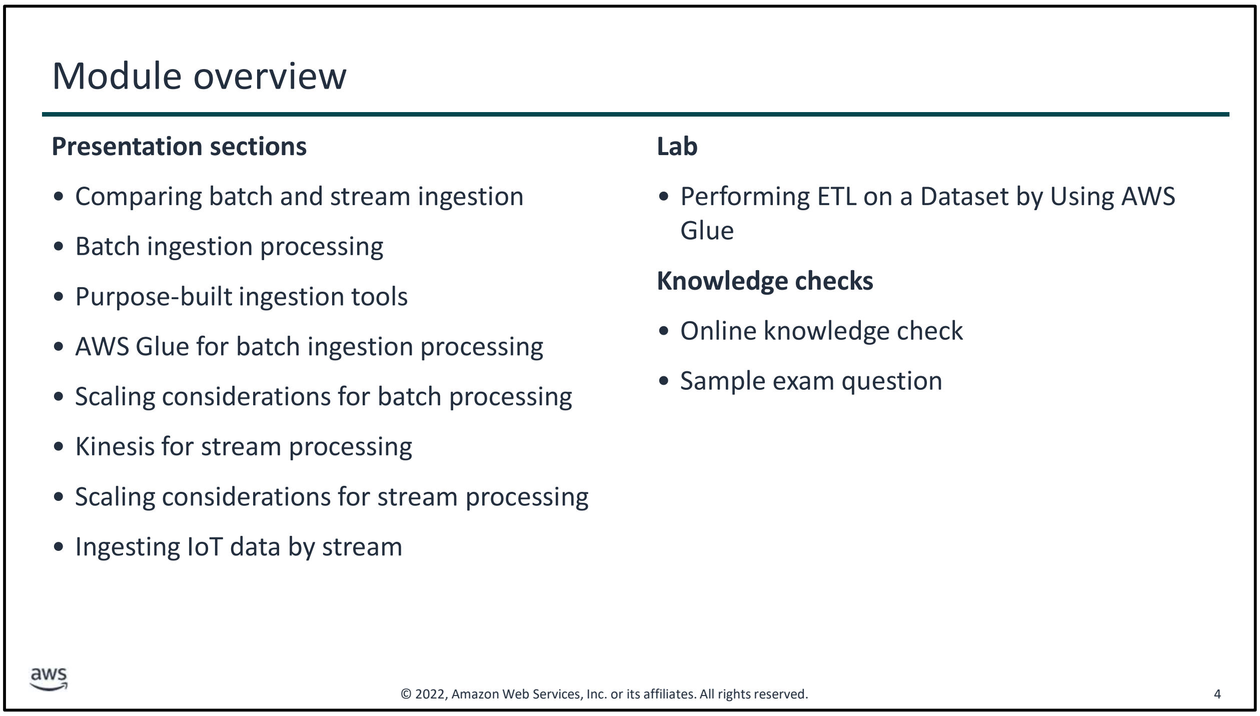
This module prepares you to do the following:

- List key tasks that the data engineer needs to perform when building an ingestion layer.
- Describe how purpose-built AWS services support ingestion tasks.
- Illustrate how the features of AWS Glue work together to support automating batch ingestion.
- Describe Amazon Kinesis streaming services and features that simplify streaming ingestion.
- Identify configuration options in AWS Glue and Amazon Kinesis Data Streams that help you scale your ingestion processing to meet your needs.
- Describe distinct characteristics of ingesting Internet of Things (IoT) data by using AWS IoT Core and AWS IoT Analytics services.

|Slide number 3
|Instructor notes
|
|Student notes

This module builds on the concepts that were presented in the prior modules. In this module, the focus is on the processes to build an ingestion layer that supports batch or stream processing.

This module reviews AWS services that simplify performing batch ingestion tasks, including Amazon AppFlow, AWS Database Migration Service, AWS DataSync, and AWS Data Exchange. You will learn how AWS Glue can help you to automate batch ingestion, and you will take a closer look at features of Amazon Kinesis streaming products that support streaming ingestion. You will also hear about options for scaling the throughput of AWS Glue jobs and Amazon Kinesis Data Streams. The last section introduces characteristics of ingesting streaming Internet of Things (IoT) data for use in analytics pipelines.



Module overview

Presentation sections

- Comparing batch and stream ingestion
- Batch ingestion processing
- Purpose-built ingestion tools
- AWS Glue for batch ingestion processing
- Scaling considerations for batch processing
- Kinesis for stream processing
- Scaling considerations for stream processing
- Ingesting IoT data by stream

Lab

- Performing ETL on a Dataset by Using AWS Glue

Knowledge checks

- Online knowledge check
- Sample exam question

|Slide number 4

|Instructor notes

|Each module has an introduction, content sections, and a wrap-up. The wrap-up for this module contains a sample exam question for you to review with the students.

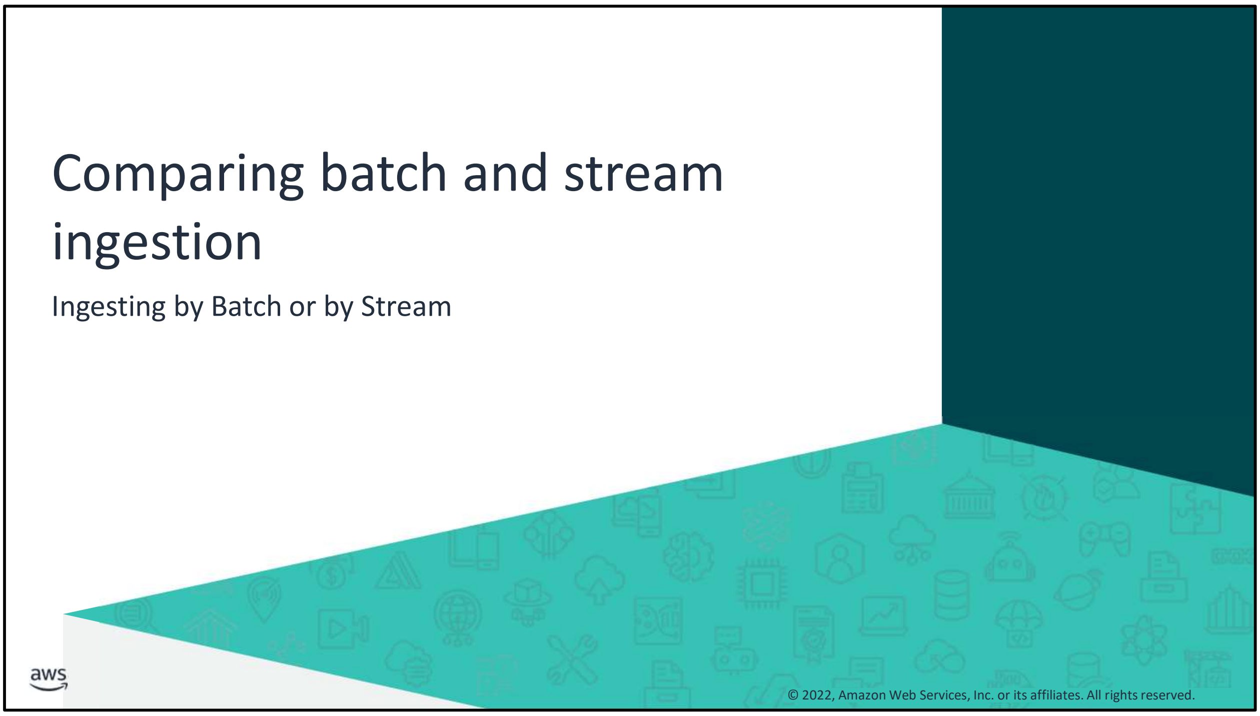
|

|Student notes

The objectives of this module are presented across multiple sections.

You will also complete a hands-on lab that uses AWS Glue to perform ETL on a dataset.

The module wraps up with a sample exam question and an online knowledge check that covers the presented material.



Comparing batch and stream ingestion

Ingesting by Batch or by Stream

|Slide number 5

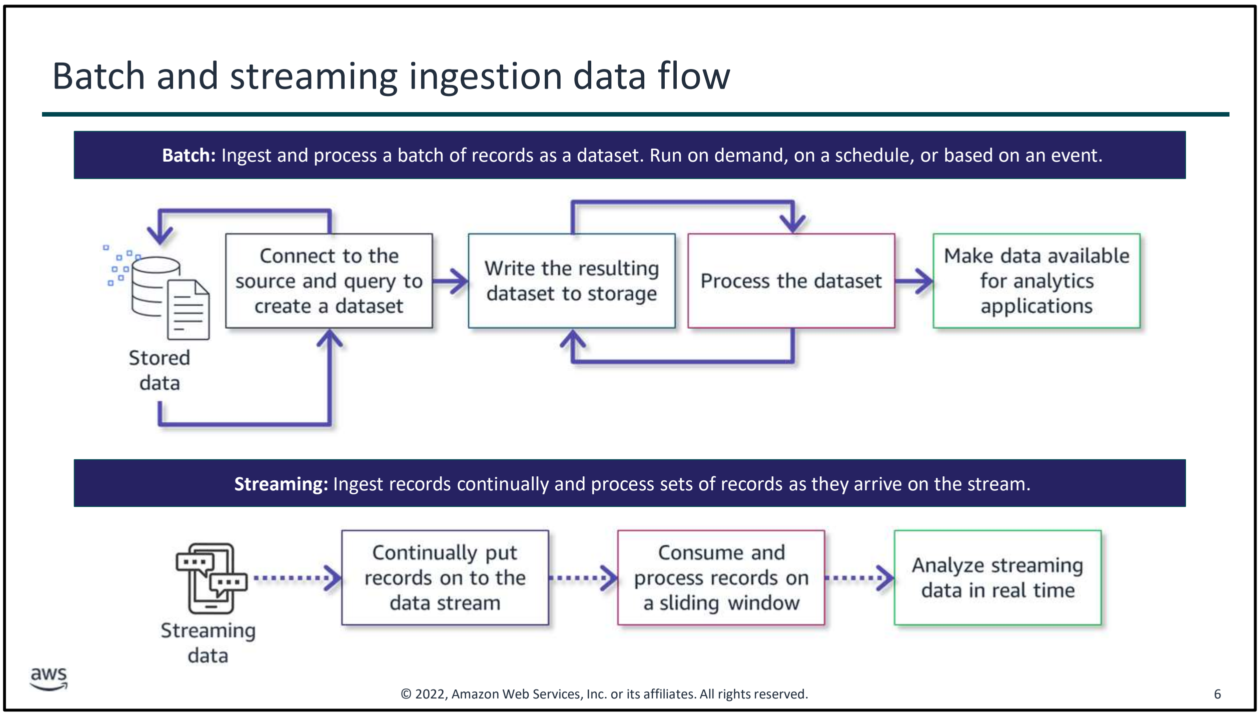
|Instructor notes

|This section is brief. You might choose to start with a discussion to compare batch and streaming to gauge understanding of the general concepts before going into the slides.

|

|Student notes

This section looks at the high-level processes that are used to ingest data into an analytics pipeline by using batch or stream processing.



Batch and streaming ingestion data flow

Batch: Ingest and process a batch of records as a dataset. Run on demand, on a schedule, or based on an event.

Stored data

Connect to the source and query to create a dataset

Write the resulting dataset to storage

Process the dataset

Make data available for analytics applications

Streaming: Ingest records continually and process sets of records as they arrive on the stream.

Streaming data

Continually put records on to the data stream

Consume and process records on a sliding window

Analyze streaming data in real time

|Slide number 6

|Instructor notes

|The key thing to highlight is the nature of how the data gets into the pipeline. With batch processing, you start a job, which must connect to a source, and then perform a query that selects a filtered subset of data and brings it into the pipeline. Whereas with streaming, producers push data onto the stream. The stream is more of a holding place for others to pick up the data.

|

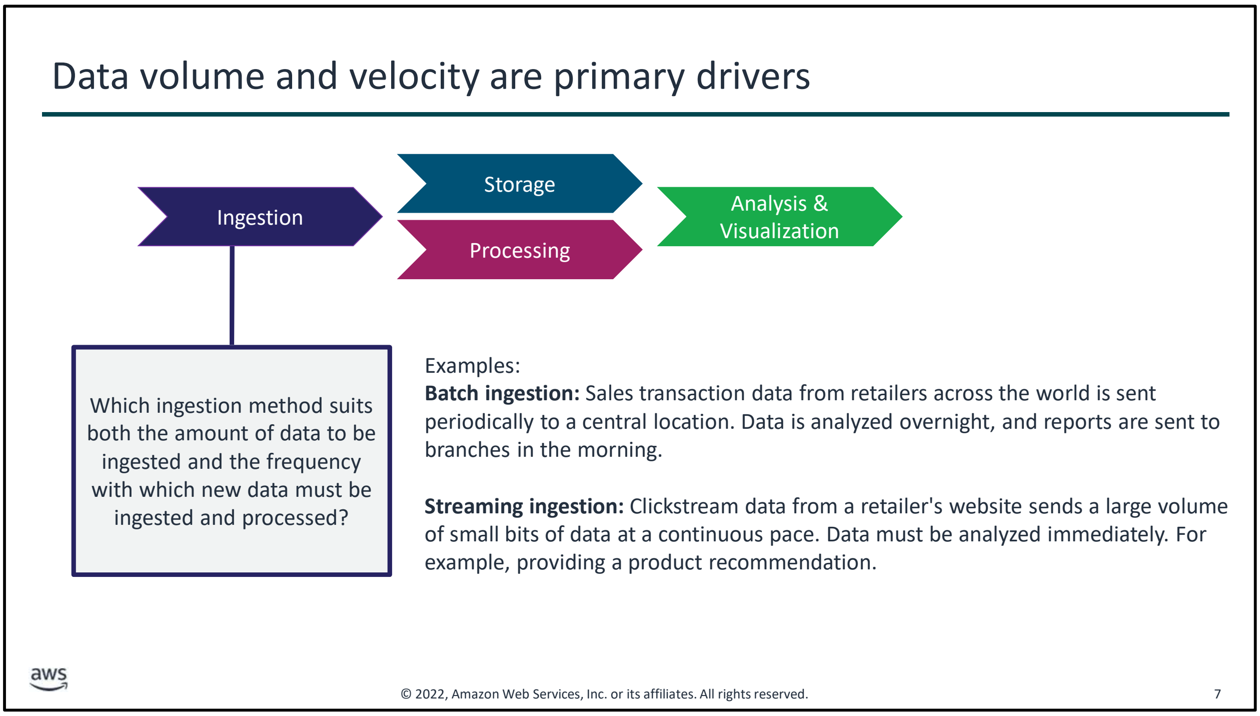
|Student notes

To generalize the characteristics of batch processing, batch ingestion involves running batch jobs that query a source, move the resulting dataset or datasets to durable storage in the pipeline, and then perform whatever transformations are required for the use case. As noted in the Ingesting and Preparing Data module, this could be just cleaning and minimally formatting data to put it into the lake. Or, it could be more complex enrichment, augmentation, and processing to support complex querying or big data and machine learning (ML) applications. Batch processing might be started on demand, run on a schedule, or initiated by an event. Traditional extract, transform, and load (ETL) uses batch processing, but extract, load, and transform (ELT) processing might also be done by batch.

The advantage of batch processing is its ability to process a large set of transactions that are collected over a period of time and perform complex and long-running processing on the dataset. Batch processing is typically less costly and easier to manage if there’s no urgency to perform the analysis.

By contrast, once a stream is active, streaming records arrive and are put onto the stream continuously. Consumers continuously read records off of the stream and process them on a sliding window of time, often making them available for real-time analytics immediately. The processed records might also be written to durable storage for downstream consumption.

The big advantage of streaming ingestion is the opportunity to review results and take action immediately. Data is ready for analysis with very low latency. Also, the stream is designed to handle high-volume, high-velocity unstructured data.



Data volume and velocity are primary drivers

Ingestion

Storage

Processing

Analysis & Visualization

Which ingestion method suits both the amount of data to be ingested and the frequency with which new data must be ingested and processed?

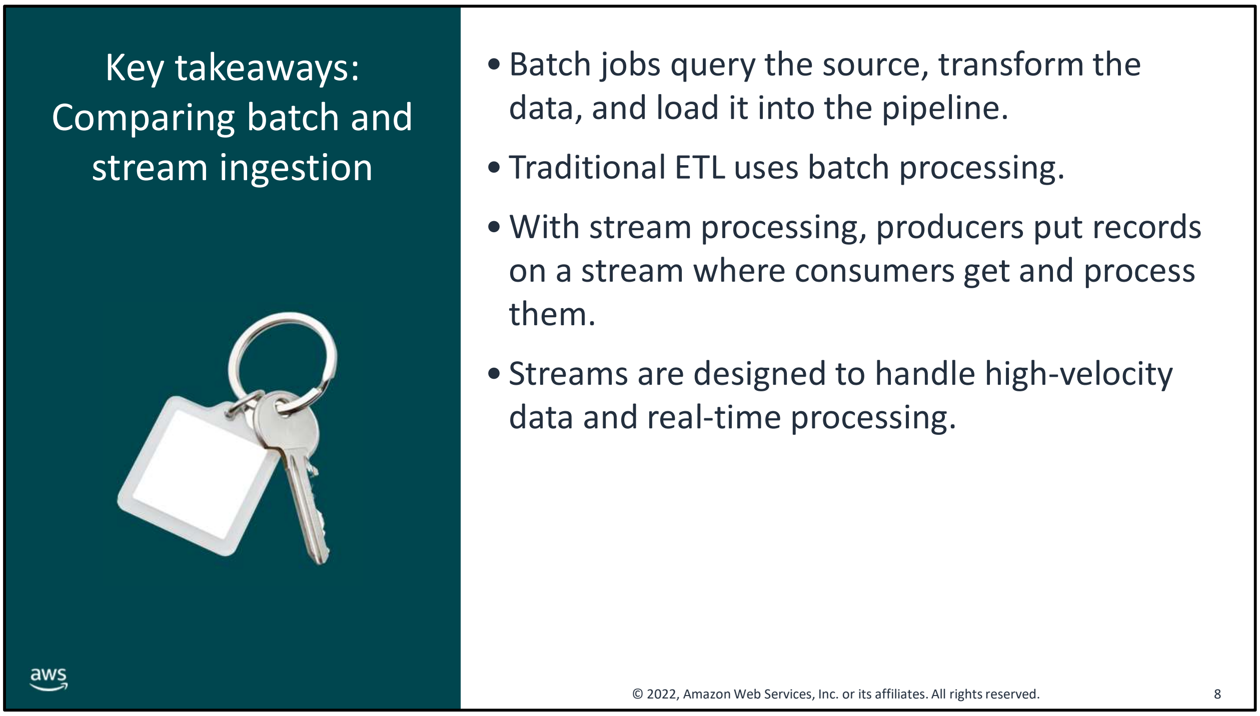
Examples:
Batch ingestion: Sales transaction data from retailers across the world is sent periodically to a central location. Data is analyzed overnight, and reports are sent to branches in the morning.

Streaming ingestion: Clickstream data from a retailer's website sends a large volume of small bits of data at a continuous pace. Data must be analyzed immediately. For example, providing a product recommendation.

|Slide number 7
|Instructor notes
|This slide is from the Elements of Data module. This is a good place to call back to it and remind learners about the five Vs. This module focuses on the methods to ingest data, in contrast with the previous module, which focused on what you need to do to the data.
|
|Student notes
This slide was presented in the Elements of Data module. The slide highlights how data volume and velocity drive your decisions about whether you need a batch or streaming ingestion process.

To choose ingestion methods for your pipeline, you need to understand the volume of data that the pipeline is expected to handle at a given interval. For example, you might need to ingest a continual stream of large volumes of small records that need to be processed as quickly as possible. Or you might need to ingest larger individual records that arrive periodically and are stored to process as a batch of records. Both examples include a high volume of data, but the velocity of the data’s arrival, and the speed with which it must be processed, will impact your pipeline design.

In this module, you will compare the factors that lead you to choose a particular ingestion method. You will also look at the configuration choices that refine how your batch or streaming solution handles the expected volume and velocity of data during ingestion.



Key takeaways: Comparing batch and stream ingestion

- Batch jobs query the source, transform the data, and load it into the pipeline.
- Traditional ETL uses batch processing.
- With stream processing, producers put records on a stream where consumers get and process them.
- Streams are designed to handle high-velocity data and real-time processing.

|Slide number 8
|Instructor notes
|
|Student notes

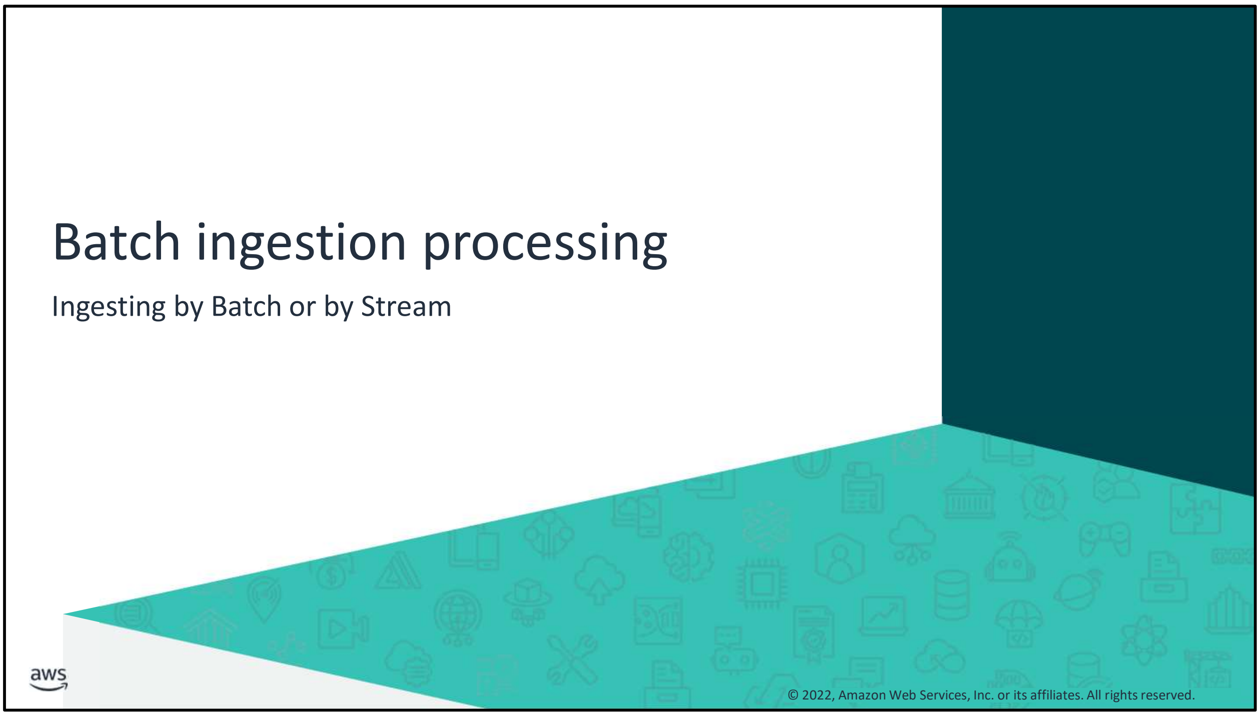
Here are a few key points to summarize this section.

Batch processing is about running jobs that query a source, transform data, and load it into the pipeline.

Traditional ETL uses batch processing because of its ability to process a large set of transactions that are collected over a period and perform complex and long-running processing on the dataset.

Streaming records arrive and are put on the stream continuously. Consumers process records from the stream and make them available for real-time analytics.

The big advantage of streaming ingestion is the opportunity to review results and take action immediately. Data is ready for analysis with low latency. Also, the stream is designed to handle high-volume, high-velocity unstructured data.



Batch ingestion processing

Ingesting by Batch or by Stream

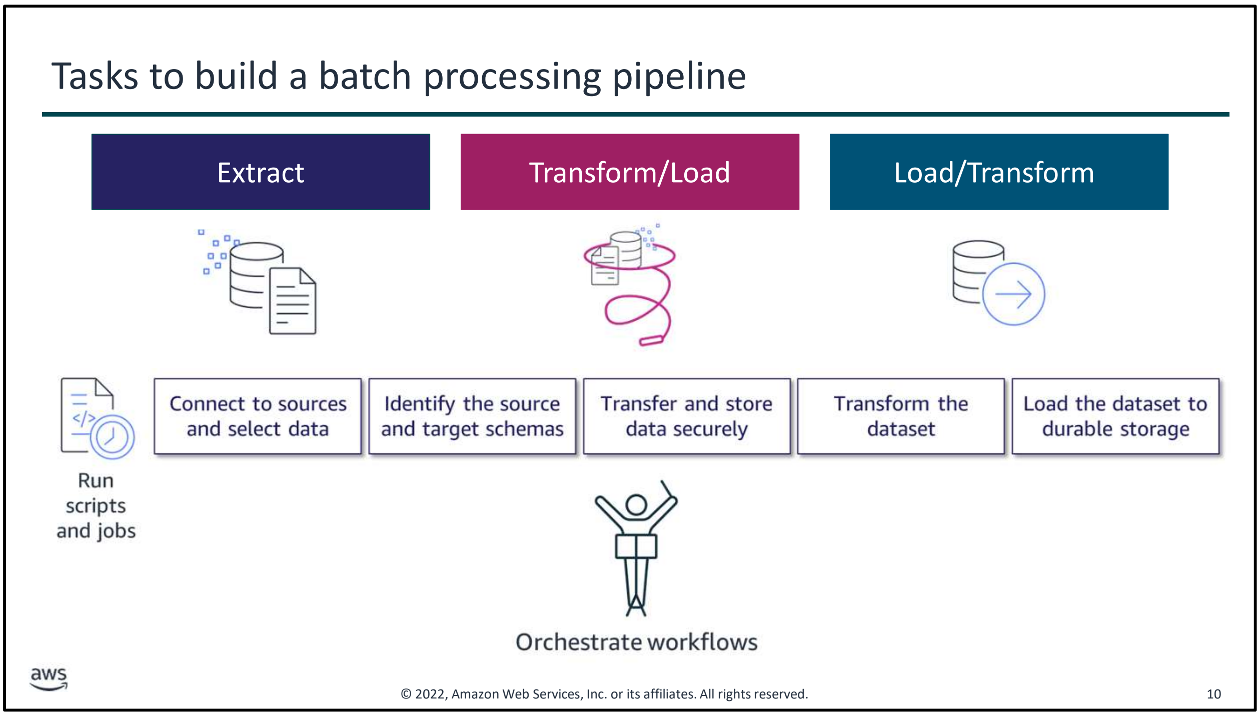
|Slide number 9

|Instructor notes

|

|Student notes

This section introduces the tasks and characteristics of batch ingestion processing.



Tasks to build a batch processing pipeline

Extract

Transform/Load

Load/Transform

Run scripts and jobs

Connect to sources and select data

Identify the source and target schemas

Transfer and store data securely

Transform the dataset

Load the dataset to durable storage

Orchestrate workflows

|Slide number 10

|Instructor notes

|The idea of this slide is to frame the concepts of ETL that were described in the prior module in the context of how a batch process performs ETL. The need for orchestration is important, and you might want to discuss with students the more detailed actions that a developer would have to take in each step beyond the data wrangling pieces that were discussed earlier. For example, the credentials and information that you need to connect to the source and how you would make sure that the transfer method is secure.

|

|Student notes

The Ingesting and Preparing Data module described a simplified example of extracting, transforming, and loading data for analysis. Support ticket data for the year 2020 was exported from a data source; imported into Microsoft Excel, where it was manipulated; and then saved as a .csv file, compressed, and uploaded to Amazon Simple Storage Service (Amazon S3). Consider this example and suppose that, after initial reviews of the dataset, your data analyst asks to get the support ticket data on an ongoing basis at the end of each month, starting from the last record in the dataset for 2020.

To meet this data request, consider the ETL tasks that you must perform:

- Connect to your data source and select the data to be ingested.

- Identify the source schema and define the target schema so that you can map the fields properly.
- Transfer the data securely into the pipeline. Make sure that you establish mechanisms to audit the ingestion process to ensure the veracity of the data in the pipeline.
- Perform transformations and load the processed data into durable storage, where it can be accessed. As discussed earlier, you might use a transform/load approach, where the majority of transformations occur prior to loading into the storage layer. Or, you might use the load/transform approach by performing minimal transformations before loading the raw data into your storage layer.

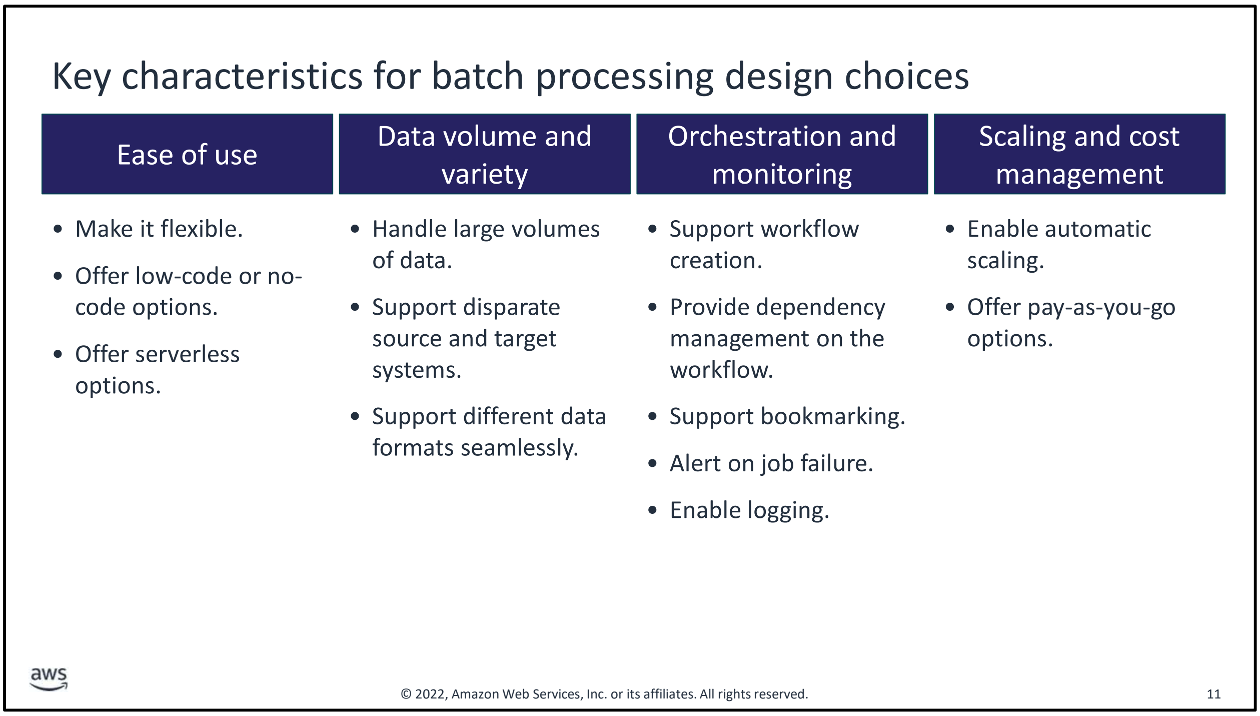
To perform the set of tasks each month, you would write scripts to perform the steps, rather than entering them at a command line each time. For example, you might write scripts to securely connect to the source, run queries to get the desired data, and transfer that data to some type of staging storage where the transformations could be done. You would also look for ways to script the manual steps that are needed to structure, clean, enrich, validate, and publish the dataset.

So now you have a set of scripts to get the data that you want, transfer it to temporary storage, wrangle it, and write it to your storage layer. The next step is to create batch jobs to automate running the set of ETL scripts that you have created. You might schedule the batch jobs to run on a schedule or based on an event. In the example scenario, you would run the batch ingestion process each month.

Secure your scripts and batch jobs by using the principle of least privilege. For example, data engineers might have permission to access, edit, or run the ETL scripts and jobs, but those same engineers might not have permissions to access the data as it moves through the pipeline. Likewise, data analysts might not be granted permissions to edit or run the ETL jobs, but they would have access to work with the stored data.

The last piece to making your batch ingestion pipeline production ready is to create a workflow to orchestrate when each job runs. You also need to handle interdependencies between jobs and related actions and parameters. For example, you need to manage what happens if one job fails or if a resource isn’t available.

Orchestration is especially important when you move beyond the simple example to a more complex ingestion process. For example, what if you were now asked to collect and include customer comments from the tickets to feed an ML model as part of analyzing the relationship between customer support experiences and contract renewals? Now you need to collect and wrangle an additional subset of unstructured data, and you would need to label it and take other steps to use it in an ML application. A workflow helps to manage the complexities of this type of ingestion.



Key characteristics for batch processing design choices

Ease of use	Data volume and variety	Orchestration and monitoring	Scaling and cost management
• Make it flexible. • Offer low-code or no-code options. • Offer serverless options.	• Handle large volumes of data. • Support disparate source and target systems. • Support different data formats seamlessly.	• Support workflow creation. • Provide dependency management on the workflow. • Support bookmarking. • Alert on job failure. • Enable logging.	• Enable automatic scaling. • Offer pay-as-you-go options.

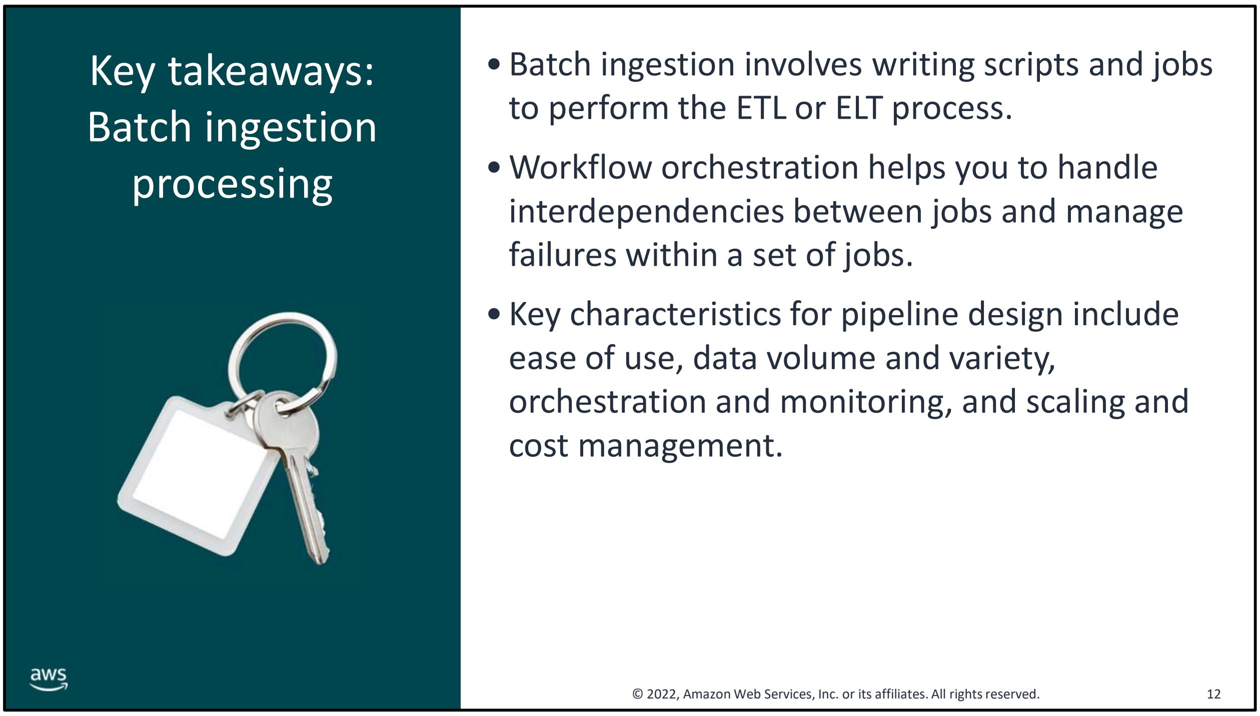
|Slide number 11
|Instructor notes
|This slide summarizes the characteristics that are listed in *Data Analytics Lens: AWS Well-Architected Framework*, in the **Scenarios** section, under **Batch data processing**. The direct link is in the course resources.
|
|Student notes
This slide summarizes the characteristics for pipelines to process batch data, as described in the *Data Analytics Lens: AWS Well-Architected Framework*.

Choose a development framework that provides flexibility for developers and also lets nondevelopers to build and run jobs easily. Offering low-code or no-code options to write ETL jobs and serverless options to run those jobs reduces the operational overhead for the data engineer.

Select tools that allow for the variety of data types, sources, and formats that you might need to process. Where possible, choose tools that are purpose built to ingest the data type that you are using.

To ensure the efficiency of your processing and also the veracity of your data, choose orchestration tools that handle dependencies within the workflow and can bookmark where a process left off when a failure occurred. Include monitoring that identifies and alerts on failures. Additionally, instrument your jobs to log metrics, timeouts, and alarms that can identify potential issues within your batch processing pipelines. Then, you will be able to respond to changes in the system's performance as jobs are authored or changed and as data source volumes change.

Your solution should be able to scale up quickly to handle peak volumes but also scale down to save resources and costs when they aren’t needed. Choose pay-as-you-go options for both authoring and processing to limit spending on idle resources.



Key takeaways: Batch ingestion processing

- Batch ingestion involves writing scripts and jobs to perform the ETL or ELT process.
- Workflow orchestration helps you to handle interdependencies between jobs and manage failures within a set of jobs.
- Key characteristics for pipeline design include ease of use, data volume and variety, orchestration and monitoring, and scaling and cost management.

|Slide number 12

|Instructor notes

|

|Student notes

Here are a few key points to summarize this section.

To perform batch ingestion, you write scripts to perform ETL tasks and create jobs to run the set of scripts for a given ETL or ELT process.

Using workflow orchestration helps you manage the interdependencies of jobs and handle failure management, particularly as your ETL jobs become more complex.

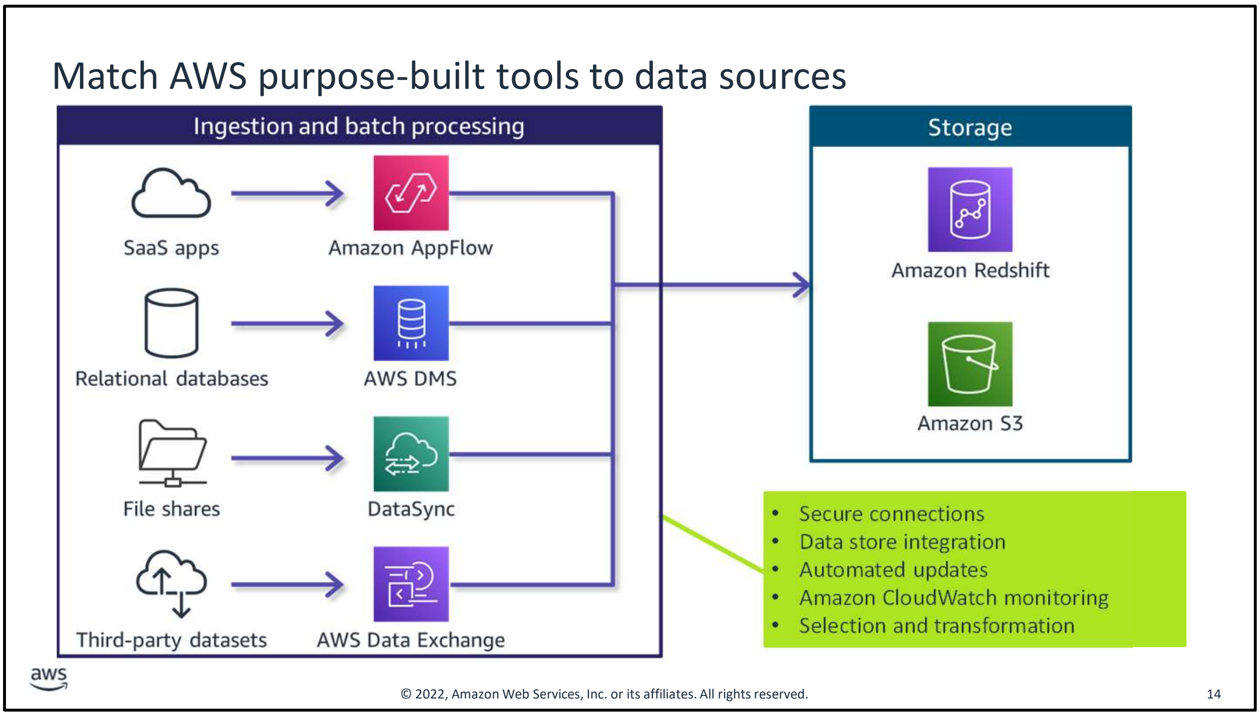
When you select components for batch ingestion, consider ease of use for the team who needs to use the system, the volume and type of data to be ingested, options for orchestration and monitoring, and whether the scaling and cost structure make sense for your intended use case.

Purpose-built ingestion tools

Ingesting by Batch or by Stream

|Slide number 13
|Instructor notes
|
|Student notes

This section introduces AWS tools that are purpose built to ingest different data types.



Match AWS purpose-built tools to data sources

Ingestion and batch processing

SaaS apps → Amazon AppFlow

Relational databases → AWS DMS

File shares → DataSync

Third-party datasets → AWS Data Exchange

Storage

Amazon Redshift

Amazon S3

- Secure connections
- Data store integration
- Automated updates
- Amazon CloudWatch monitoring
- Selection and transformation

|Slide number 14

|Instructor notes

|This diagram is a subset of one shared in the Design Principles and Patterns for Data Pipelines module as part of the reference architecture for the modern data architecture. This slide is a good spot to highlight that these are the AWS options, but the point is not so much about knowing these specific services but recognizing that tools exist that are designed for specific types of data. Many of these tools include interfaces that simplify common data wrangling tasks that would otherwise require scripting.

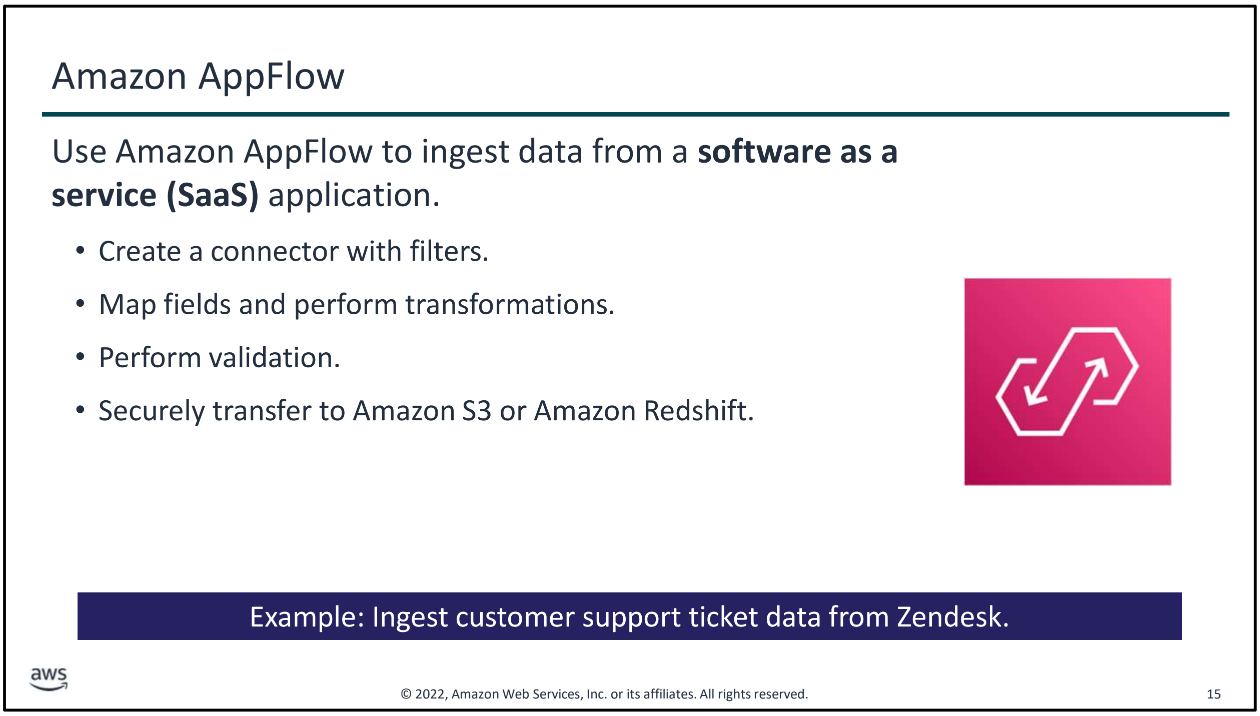
|

|Student notes

For accessibility: Data source types and batch processing options diagram shows several AWS services integrate with different data source types. Use Amazon AppFlow with SaaS apps; AWS DMS with relational databases; DataSync with file shares; and AWS Data Exchange with third-party datasets. These services also integrate with AWS storage services such as Amazon Redshift and Amazon S3. **End of accessibility description.**

This slide highlights a few of the options within AWS to simplify the tasks that are required to build and maintain batch ingestion processing. A purpose built tool is a tool that’s particularly suited to its use. The intent of highlighting these services is to provide examples of how the characteristics of your data and the skills and requirements of the teams that will build and use the pipeline should guide your choices. Several AWS services integrate with a variety of data source types and also integrate with AWS storage offerings. Other cloud vendors might also have similar offerings. The key is to consider your specific use case and make "best-fit" choices per data source instead of trying for a one-size-fits-all solution.

The next few slides summarize the features of each of these services. The content resources section of your online course provides links to learn more about each service.



Amazon AppFlow

Use Amazon AppFlow to ingest data from a **software as a service (SaaS)** application.

- Create a connector with filters.
- Map fields and perform transformations.
- Perform validation.
- Securely transfer to Amazon S3 or Amazon Redshift.

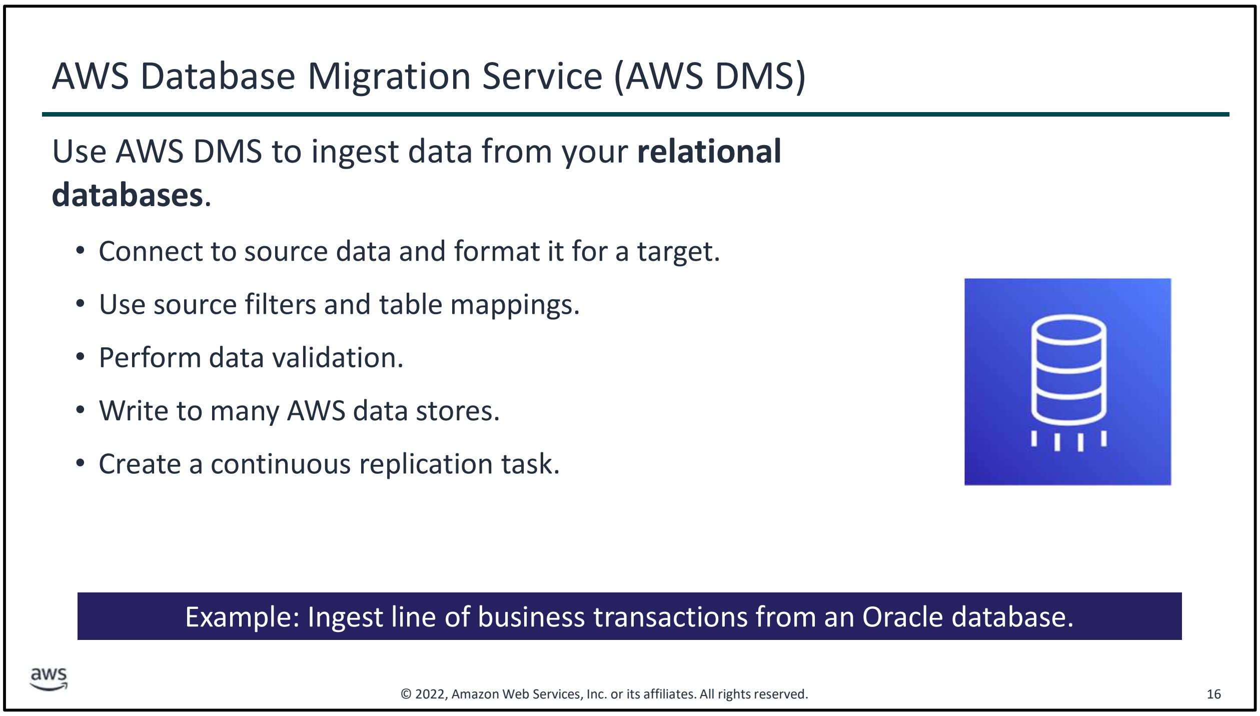
Example: Ingest customer support ticket data from Zendesk.

|Slide number 15
|Instructor notes
|
|Student notes

Use Amazon AppFlow to ingest data from a software as a service (SaaS) application. You can do the following with Amazon AppFlow:

- Create a connector that reads from a SaaS source and includes filters.
- Map fields in each source object to fields in the destination and perform transformations.
- Perform validation on records to be transferred.
- Securely transfer to Amazon S3 or Amazon Redshift. You can trigger an ingestion on demand, on event, or on a schedule.

An example use case for Amazon AppFlow is to ingest customer support ticket data from the Zendesk SaaS product.



AWS Database Migration Service (AWS DMS)

Use AWS DMS to ingest data from your **relational databases**.

- Connect to source data and format it for a target.
- Use source filters and table mappings.
- Perform data validation.
- Write to many AWS data stores.
- Create a continuous replication task.

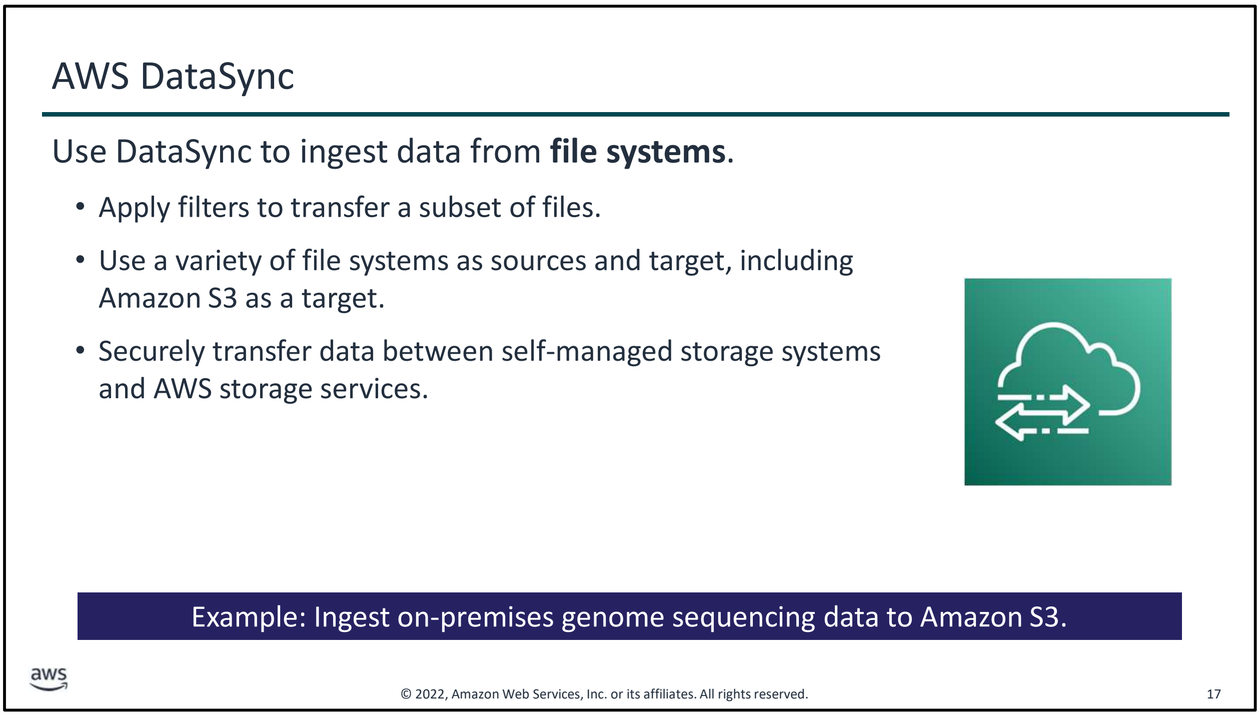
Example: Ingest line of business transactions from an Oracle database.

|Slide number 16
|Instructor notes
|You might want to note that this service can migrate a database to the cloud.
|
|Student notes

Use AWS DMS to ingest data from your relational databases. You can do the following with AWS DMS:

- Connect to source data, format it for a target, and use source filters.
- Use table mappings to select and transform data.
- Perform data validation to ensure that your data was migrated accurately.
- Write to many AWS data stores, including Amazon Redshift, Amazon S3, Amazon Aurora, and Amazon DynamoDB by using encrypted connections. You can create a continuous replication task to stay in sync.

An example use case for AWS DMS is to ingest line of business transactions from an Oracle database that is used within the organization. Data is ingested on an ongoing basis by using a continuous replication task.



AWS DataSync

Use DataSync to ingest data from **file systems**.

- Apply filters to transfer a subset of files.
- Use a variety of file systems as sources and target, including Amazon S3 as a target.
- Securely transfer data between self-managed storage systems and AWS storage services.

Example: Ingest on-premises genome sequencing data to Amazon S3.

|Slide number 17

|Instructor notes

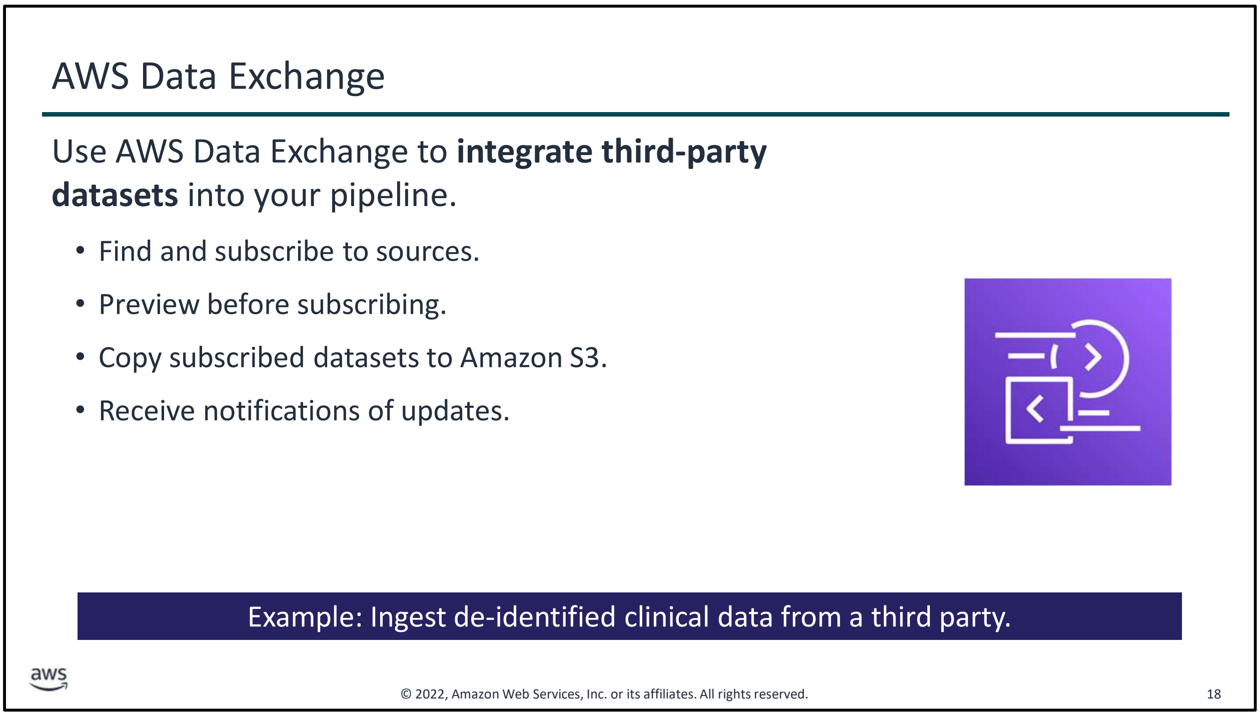
|

|Student notes

Use AWS DataSync to ingest data from file systems. You can do the following with DataSync:

- Apply filters to transfer a subset of files.
- Use a variety of file systems as sources and targets, including Amazon S3 as a target.
- Securely transfer data between self-managed storage systems and AWS storage services.
- Run on a schedule.
- Use Amazon CloudWatch metrics to monitor the performance of file transfers.

An example use case for DataSync is to ingest data into Amazon S3 from a genome-sequencing machine that creates and stores data on promises. The data in Amazon S3 can be used for analytics or ML applications.



AWS Data Exchange

Use AWS Data Exchange to **integrate third-party datasets** into your pipeline.

- Find and subscribe to sources.
- Preview before subscribing.
- Copy subscribed datasets to Amazon S3.
- Receive notifications of updates.

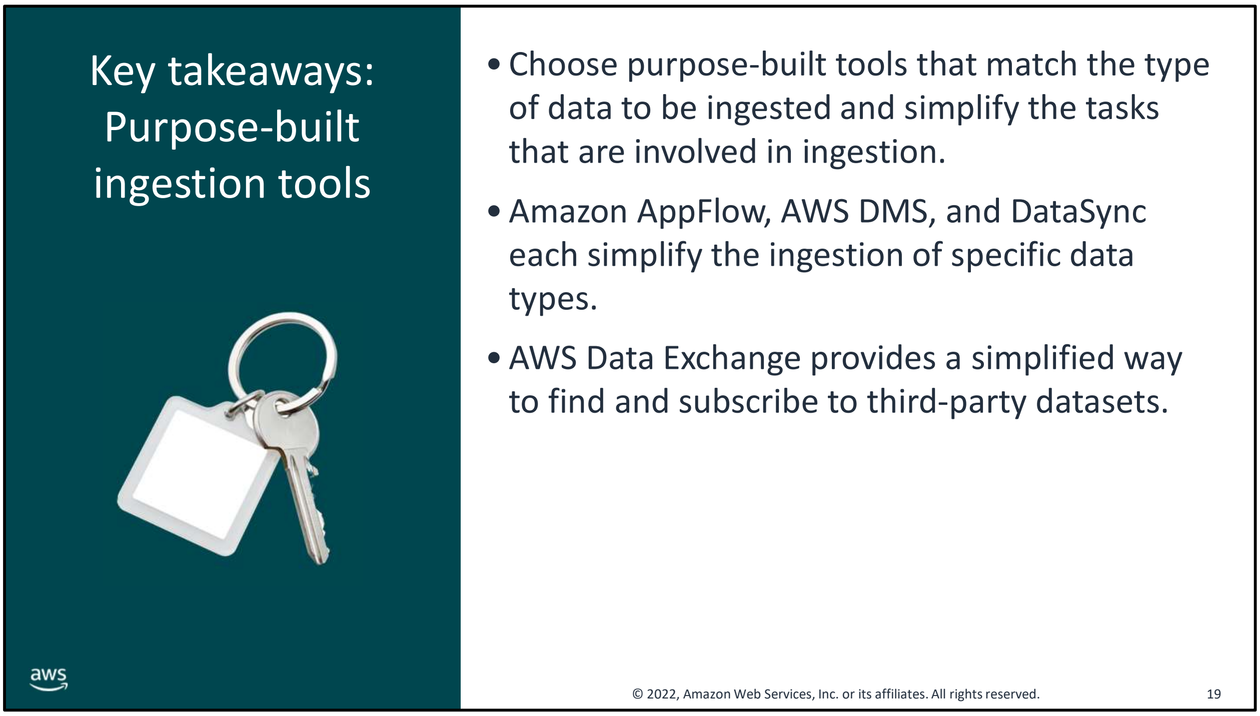
Example: Ingest de-identified clinical data from a third party.

|Slide number 18
|Instructor notes
|
|Student notes

Use AWS Data Exchange to integrate third-party datasets into your pipeline. You can do the following with AWS Data Exchange:

- Find and subscribe to sources.
- Preview a data dictionary and sample data before subscribing.
- Copy subscribed datasets to Amazon S3.
- Receive an event from AWS Data Exchange every time the provider publishes revisions or adds new datasets to a subscribed product.

An example use case for AWS Data Exchange is to ingest a dataset that provides de-identified clinical data from a hospital system.



Key takeaways: Purpose-built ingestion tools

- Choose purpose-built tools that match the type of data to be ingested and simplify the tasks that are involved in ingestion.
- Amazon AppFlow, AWS DMS, and DataSync each simplify the ingestion of specific data types.
- AWS Data Exchange provides a simplified way to find and subscribe to third-party datasets.

|Slide number 19

|Instructor notes

|

|Student notes

Here are a few key points to summarize this section.

By matching the ingestion tool with the type of data store being ingested, you can simplify the work to get the data into your pipeline. Examples include Amazon AppFlow for SaaS data, AWS DMS for relational databases, and AWS DataSync for file systems.

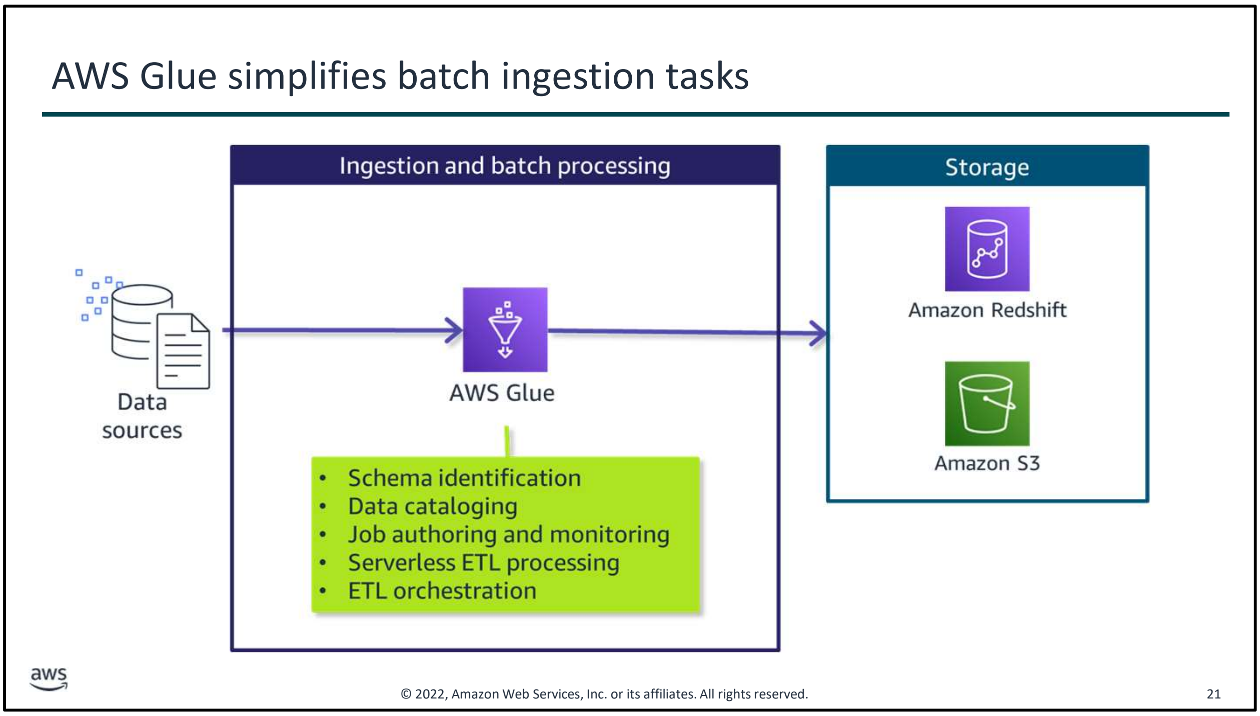
AWS Data Exchange simplifies finding and using third-party datasets that you can ingest into your pipeline.

AWS Glue for batch ingestion processing

Ingesting by Batch or by Stream

|Slide number 20
|Instructor notes
|
|Student notes

This section reviews the features of AWS Glue and how to use them to automate ETL processing.



AWS Glue simplifies batch ingestion tasks

Data sources

Ingestion and batch processing

AWS Glue

- Schema identification
- Data cataloging
- Job authoring and monitoring
- Serverless ETL processing
- ETL orchestration

Storage

Amazon Redshift

Amazon S3

|Slide number 21

|Instructor notes

|The next few slides look more closely at the features of AWS Glue. The overarching takeaway is that the data engineer should look for cloud-based ETL tools that simplify data wrangling tasks and tasks to write and manage ETL jobs.

|Tools such as AWS Glue include features that help to reduce human error and automate time-consuming tasks. Individual components of AWS Glue are called out on their own, but they are all part of the AWS Glue service. These components include AWS Glue Studio, crawlers, the AWS Glue Data Catalog, the AWS Glue Spark runtime engine, and workflows.

|AWS Documentation includes the *AWS Glue Developer Guide* and the *AWS Glue Studio User Guide*. In the developer guide, under **How it works**, the **Concepts** and **Components** sections provide a good introduction to the service. The *AWS Glue Studio User Guide* provides details about creating, editing, and monitoring jobs.

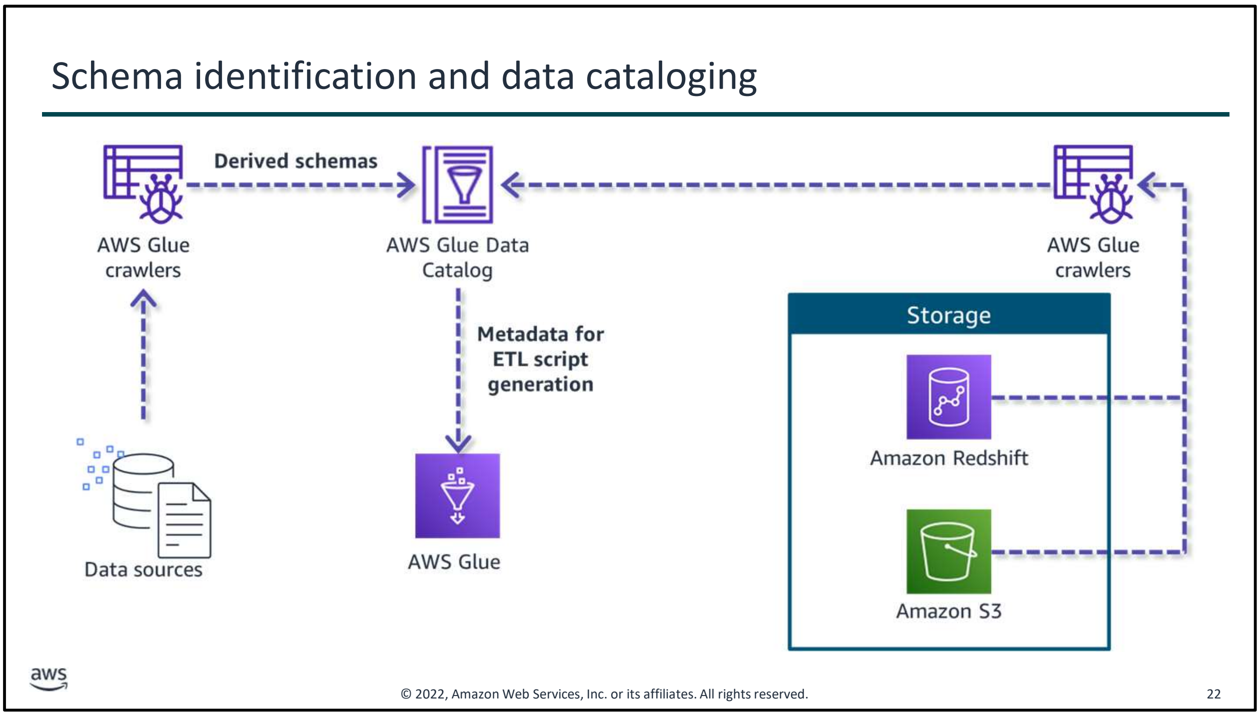
|

|Student notes

AWS Glue is a data integration service that helps you to automate and perform ETL tasks as part of ingesting data into your pipeline. You can also use the service to facilitate transformation and movement between data stores. The focus here is on its use as part of a batch ingestion process.

The next few slides will look at the following AWS Glue features in a bit more detail:

- Automatically generate schemas from semistructured data by using crawlers.
- Catalog data and get a unified view with the Data Catalog.
- Use AWS Glue Studio to author ETL jobs that bring data in from different sources and load it into a target.
- Perform serverless ETL processing in the AWS Glue Spark runtime engine.
- Orchestrate complex ETL tasks with interdependencies by using workflows.
- Monitor, troubleshoot, and optimize ETL jobs and workflows.



Schema identification and data cataloging

|Slide number 22

|Instructor notes

|The Design Principles and Patterns for Data Pipelines module introduced the concepts of AWS Glue crawlers and the Data Catalog as part of the storage layer. Crawlers can derive schema from semistructured data. In the context of ingestion, crawlers could be used to derive schema information from both the source and target so long as both are types of data stores the AWS Glue crawlers support. The metadata that is stored in the Data Catalog is used for "schema-on-read" so that the schema is retrieved when being queried.

|The diagram shows the potential flow from both data sources and data stores that are targets of the ingestion step. For more information about the Data Catalog and crawlers, see the *AWS Glue Developer Guide*.

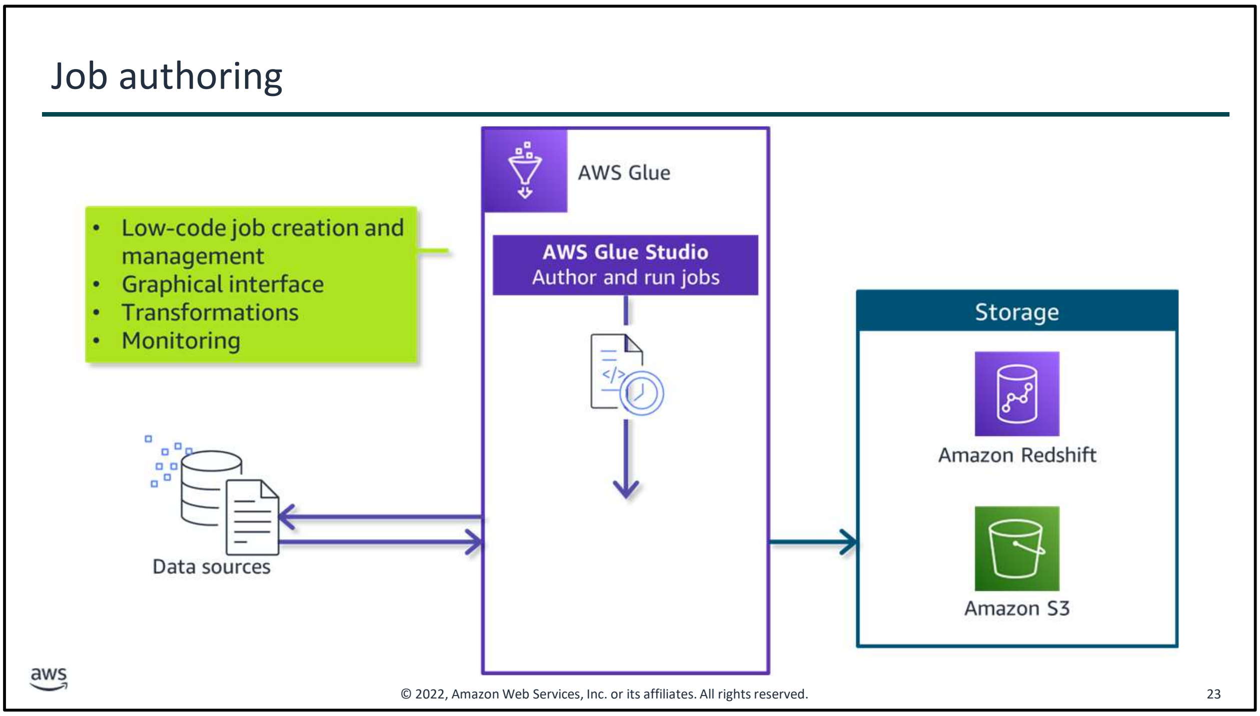
|

|Student notes

The AWS Glue Data Catalog stores metadata including schema information about data sources and targets of your ETL jobs. The Data Catalog also appears as part of the modern data architecture presented in the Design Principles and Patterns for Data Pipelines.

AWS Glue crawlers run on your data stores, derive schema from them, and populate the Data Catalog. Crawlers can run on many data stores, including Amazon S3, Amazon Redshift, most relational databases, and DynamoDB. By using the metadata in the Data

Catalog, you can also automatically generate scripts with AWS Glue extensions. You can use these as your starting point to write your AWS Glue jobs.
The data wrangling example scenario in the Ingesting and Preparing Data module uses crawlers as part of publishing data to Amazon S3 and making it available to users.



Job authoring

- Low-code job creation and management
- Graphical interface
- Transformations
- Monitoring

AWS Glue

AWS Glue Studio
Author and run jobs

Data sources

Storage

Amazon Redshift

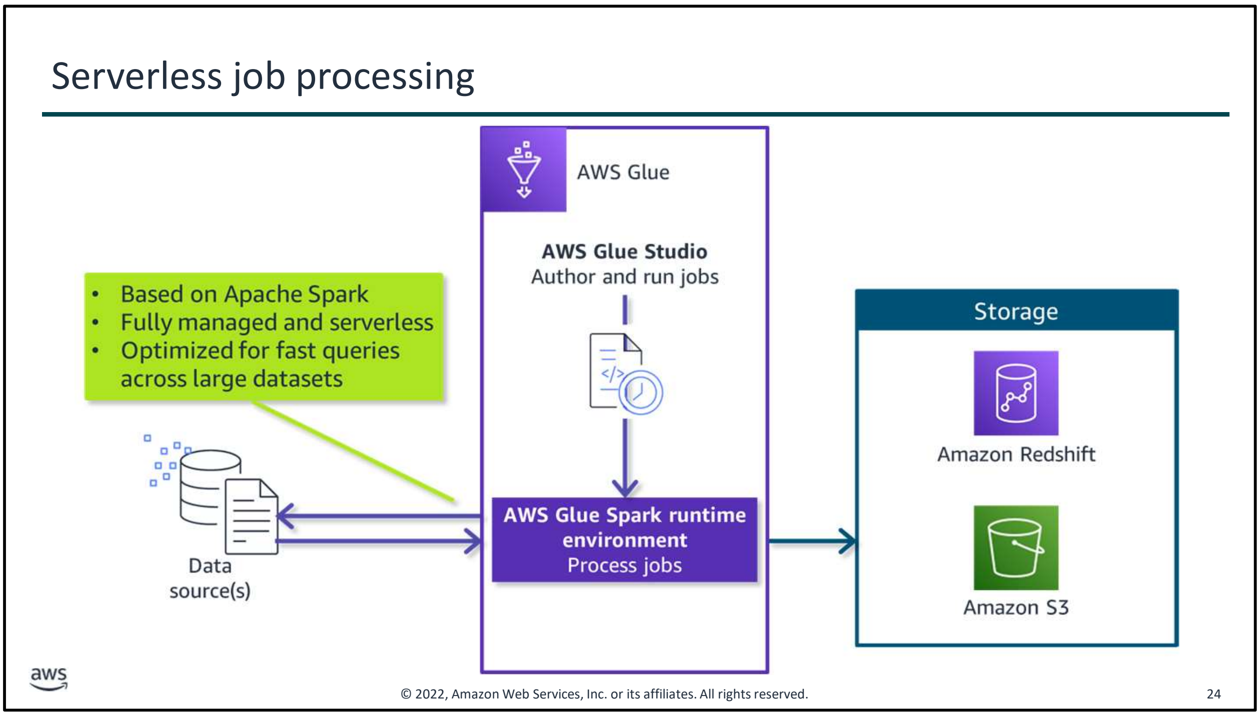
Amazon S3

|Slide number 23
|Instructor notes
|The visuals on this and the next few slides are intended to show the components of AWS Glue and how they can be used for each step in batch processing: authoring, running/processing, and orchestration.
|Depending on the audience, you might want to dive more into how scripts are written. For information, see the **Editing jobs** section of the *AWS Glue Studio User Guide*.
|You might note that other options to author jobs within AWS include Amazon SageMaker notebooks, Amazon EMR Studio, or AWS Lake Formation blueprints, as described in the reference architecture for batch data processing in the *Data Analytics Lens: AWS Well-Architected Framework*. The Processing Data for ML module will discuss SageMaker notebooks, and the Processing Big Data module will discuss Amazon EMR Studio. This course does not cover Lake Formation blueprints, but they are a good option when you need to quickly create batch data ingestion jobs to rapidly build a data lake in AWS.
|
|Student notes
Whether you use the scripts that the AWS Glue Data Catalog generates, you can use AWS Glue Studio, a feature of AWS Glue, to write your scripts and author jobs. You can also use the tool to run the jobs that you have created. You can run jobs on a schedule, on demand, or based on a specific event.

AWS Glue Studio provides an easy-to-use graphical interface to author ETL jobs. The visual interface helps users without a lot of coding experience to design jobs. The interface also accelerates the process for users who have coding experience.

You can do the following with AWS Glue Studio:

- Pull data from an Amazon S3, Amazon Kinesis, or Java Database Connectivity(JDBC) source.
- Configure a transformation that joins, samples, or transforms the data.
- Specify a target location for the transformed data.
- View the schema or a sample of the dataset at each point in the job.
- Run, monitor, and manage the jobs that were created in AWS Glue Studio.



Serverless job processing

- Based on Apache Spark
- Fully managed and serverless
- Optimized for fast queries across large datasets

Data source(s)

AWS Glue

AWS Glue Studio
Author and run jobs

AWS Glue Spark runtime environment
Process jobs

Storage

Amazon Redshift

Amazon S3

|Slide number 24

|Instructor notes

|This module purposefully limits discussions of how Apache Spark or other distributed processing systems work. The Processing Big Data module will cover big data processing. That module might be a better place to discuss how such systems work.

|The runtime engine that is referenced is based on the Amazon EMR runtime for Apache Spark, which was announced in 2019 in “Amazon EMR Introduces EMR Runtime for Apache Spark” in the *AWS Big Data Blog*. This engine is used with AWS Glue, and the engine is optimized for performance.

|Other options for job processing are available, as noted in the *Data Analytics Lens: AWS Well-Architected Framework*. One option is the Amazon EMR engine. The Lens mentions AWS Glue Elastic Views, but the course does not cover this feature. AWS Glue DataBrew is also mentioned, and it does some of processing for you behind the scenes. Amazon Redshift is also noted as a processing option. The Storing and Organizing Large Datasets module and Analyzing and Visualizing Data module cover Amazon Redshift in more detail.

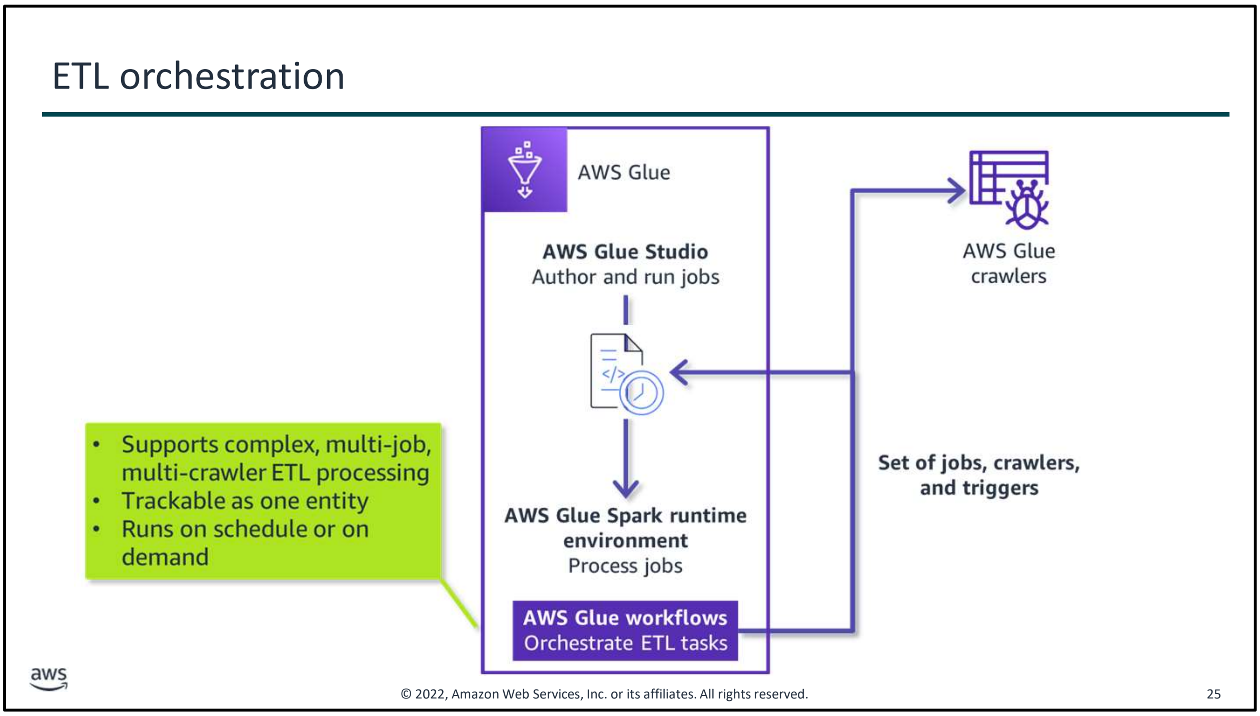
|

|Student notes

When you run your AWS Glue job, the service extracts the data from the identified source or sources and performs the specified transformations in a runtime engine in a managed serverless environment. You configure some aspects of the runtime environment as part of setting up your jobs, but you don’t need to provision or manage servers. AWS Glue will also load the resulting dataset to the target storage location that you specify in your job.

The AWS Glue runtime engine is Apache Spark based. Spark is an open-source, distributed processing system for big data. Spark is optimized to run fast queries across large datasets.

AWS developed a performance-optimized runtime for Spark for use with Amazon EMR. AWS Glue uses that same runtime engine. You will learn more about Amazon EMR in the Processing Big Data module.



ETL orchestration

- Supports complex, multi-job, multi-crawler ETL processing
- Trackable as one entity
- Runs on schedule or on demand

AWS Glue

AWS Glue Studio
Author and run jobs

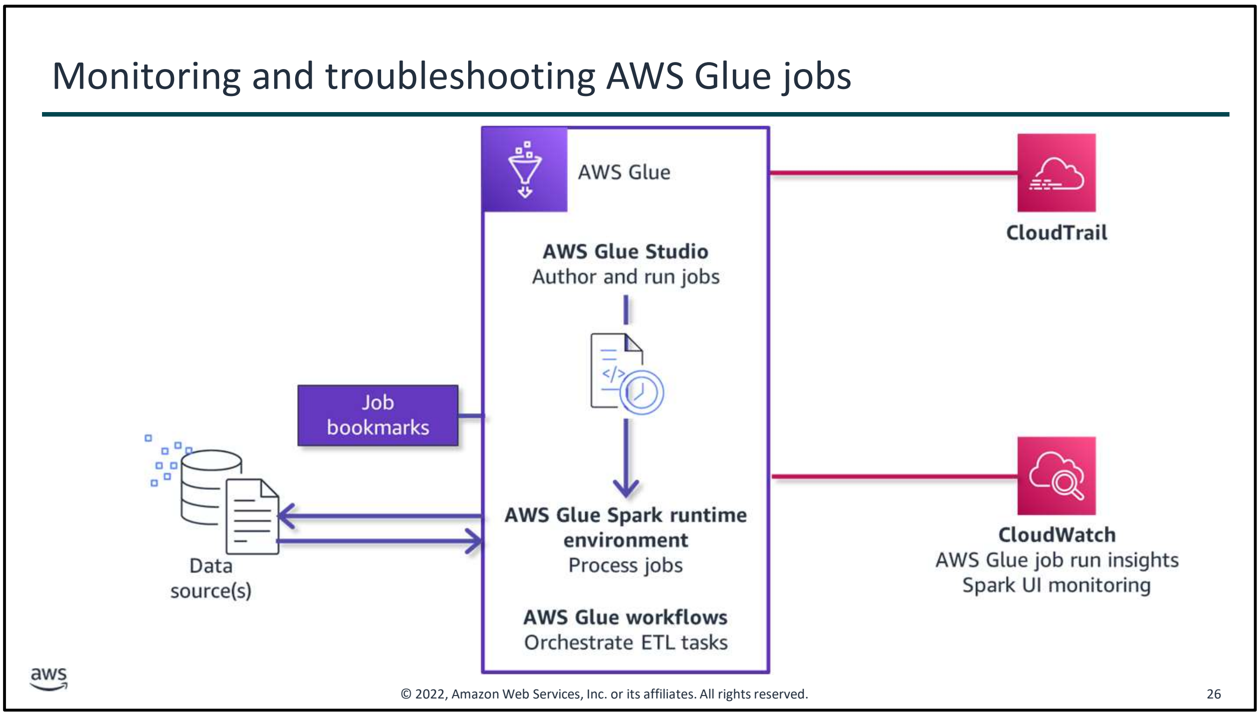
AWS Glue Spark runtime environment
Process jobs

AWS Glue workflows
Orchestrate ETL tasks

AWS Glue crawlers

Set of jobs, crawlers, and triggers

|Slide number 25
|Instructor notes
|ETL orchestration is an important concept for managing ETL processing, particularly where you have more complex sets of jobs and crawlers that need to run in a particular order with a set of parameters. You might note that the reference architecture for batch data processing in the *Data Analytics Lens: AWS Well-Architected Framework* notes other options for ETL orchestration. The reference architecture highlights the three orchestration options of AWS Glue, AWS Step Functions, and Amazon Managed Workflows for Apache Airflow (Amazon MWAA). This course does not describe Amazon MWAA, but the Automating the Pipeline module discusses Step Functions. Students will also use Step Functions in the Build and orchestrate ETL pipelines using Athena and Step Functions lab.
|
|Student notes
To orchestrate complex ETL processes, use AWS Glue workflows. Workflows work with the runtime engine to make it easier to design multi-job, multi-crawler ETL processes. You can you run and track a workflow as a single entity. After you create a workflow and specify the jobs, crawlers, and triggers that are associated with it, you can run the workflow on demand or on a schedule.



Monitoring and troubleshooting AWS Glue jobs

AWS Glue

AWS Glue Studio
Author and run jobs

AWS Glue Spark runtime environment
Process jobs

AWS Glue workflows
Orchestrate ETL tasks

Job bookmarks

Data source(s)

CloudTrail

CloudWatch
AWS Glue job run insights
Spark UI monitoring

|Slide number 26
|Instructor notes
|
|Student notes

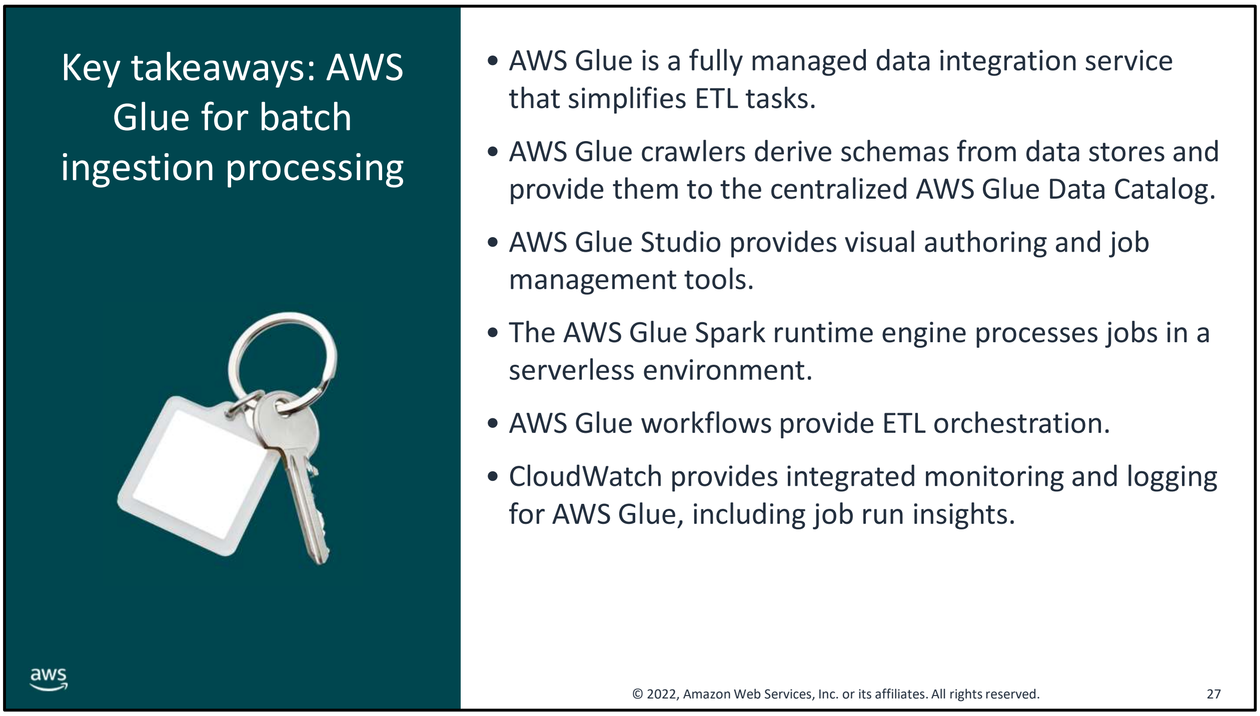
AWS Glue includes features that help you to handle issues in your jobs and track down the cause of issues.

You can view the status of a job in the AWS Glue console or by using the AWS Command Line Interface (AWS CLI).

With job bookmarks, you can process new data when rerunning on a scheduled interval. The bookmarks maintain the state of elements from the prior run so that you can successfully pick up where you left off without reprocessing old data.

AWS Glue job run insights is a feature in AWS Glue that simplifies job debugging and optimization for your AWS Glue jobs. Within AWS Glue, you can use the Apache Spark web UI and Amazon CloudWatch logs and metrics to monitor your jobs. This feature gives you the line number of an AWS Glue job script that had a failure. You can collect CloudWatch metrics about jobs and visualize them on the AWS Glue console to identify and fix issues. You can enable this option as part of defining your job. You can profile multiple AWS Glue jobs together and monitor the flow of data between them.

AWS Glue is also integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or AWS service. The Securing and Scaling the Data Pipeline module introduced CloudTrail.



Key takeaways: AWS Glue for batch ingestion processing

- AWS Glue is a fully managed data integration service that simplifies ETL tasks.
- AWS Glue crawlers derive schemas from data stores and provide them to the centralized AWS Glue Data Catalog.
- AWS Glue Studio provides visual authoring and job management tools.
- The AWS Glue Spark runtime engine processes jobs in a serverless environment.
- AWS Glue workflows provide ETL orchestration.
- CloudWatch provides integrated monitoring and logging for AWS Glue, including job run insights.

|Slide number 27

|Instructor notes

|

|Student notes

Here are a few key points to summarize this section.

AWS Glue simplifies ETL through features that can generate and catalog schema information, automatically create ETL scripts, and run batch jobs in an Apache Spark runtime engine.

AWS Glue workflows and AWS Glue job run insights simplify orchestrating and monitoring your AWS Glue jobs.

Scaling considerations for batch processing

Ingesting by Batch or by Stream

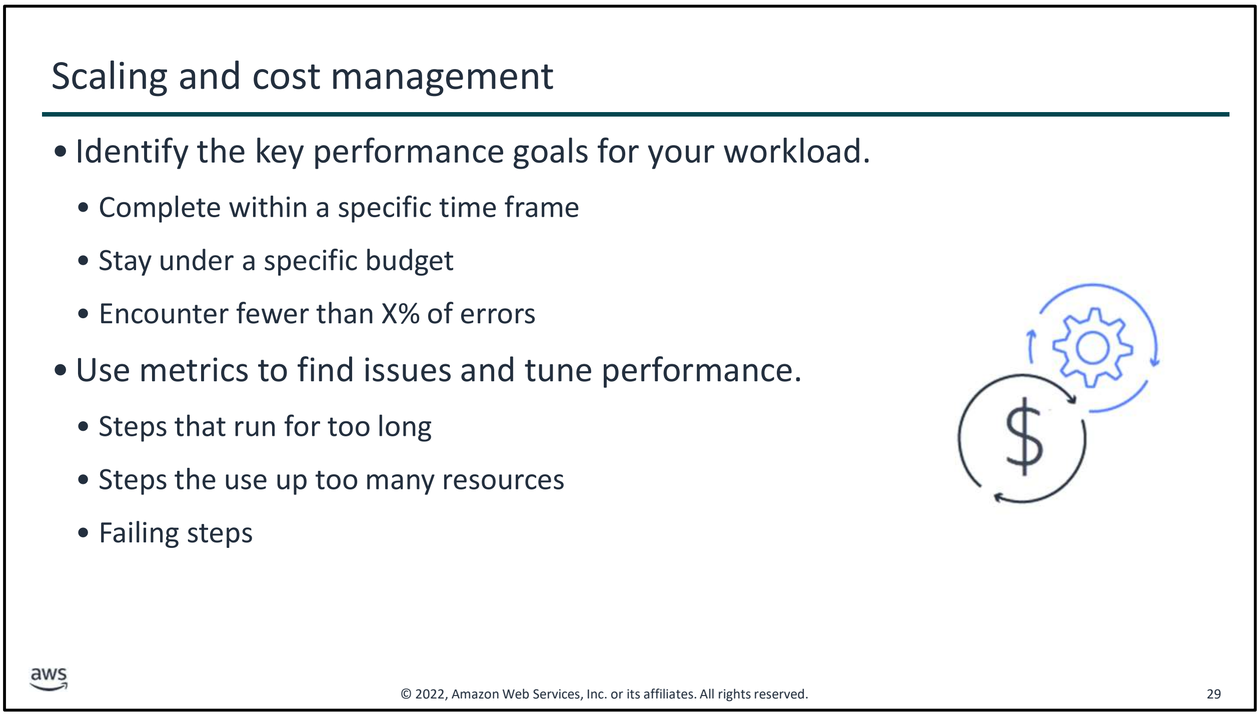
|Slide number 28

|Instructor notes

|

|Student notes

This section looks briefly at scaling considerations for batch processing pipelines.



Scaling and cost management

- Identify the key performance goals for your workload.
  - Complete within a specific time frame
  - Stay under a specific budget
  - Encounter fewer than X% of errors
- Use metrics to find issues and tune performance.
  - Steps that run for too long
  - Steps the use up too many resources
  - Failing steps

|Slide number 29
|Instructor notes
|
|Student notes

The Securing and Scaling the Data Pipeline module discussed designing a scalable pipeline. This slide looks at scaling decisions in batch processing.

Batch goals are often about completing within a specific time frame, within a given budget, or with a certain threshold of accuracy. For example, your primary goal might be to make sure that daily batches are completed by a specific time to support a daily analysis task. Therefore, you would want to look at how long each step takes. Or you might be more focused on making sure that the resource usage of a job is within a specified budget, so you might focus on processing overnight over a longer period of time to reduce the cost of resources.

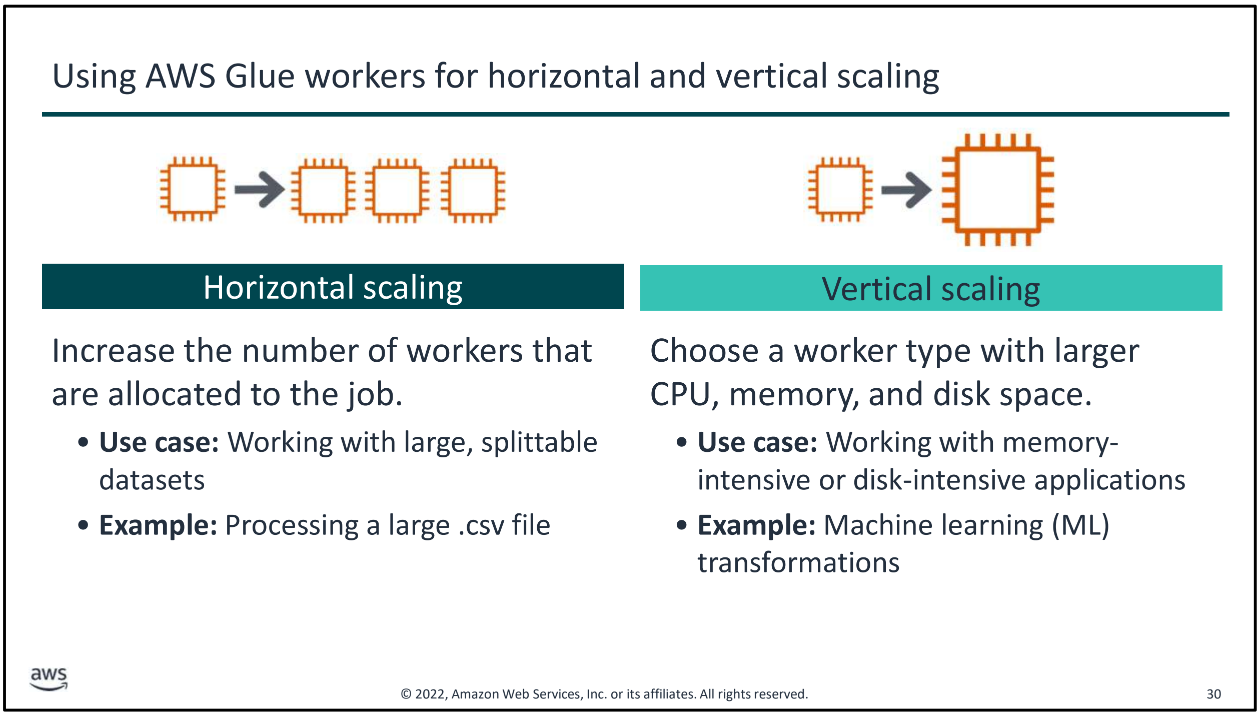
Of course, you want to look for steps that fail and determine if failures are related to scaling. For example, if a step times out while waiting for another step to complete or if a step fails because a resource is overwhelmed with requests, you might need to find ways to decouple those processes or scale individual resources up or out.

When you configure your batch jobs, it’s important to know how much data you are going to process, how often it’s updated, and how many different types of data you have. It's also important to consider the type of transformations that need to occur on that data. For example, are you performing complex transformations that require a lot of data to be kept in memory at once, or are there quick individual transformations that happen per record?

You can use your best guess when initially setting up your jobs, but you need to use real metrics to understand where the bottlenecks and opportunities are. From there, you can tackle each bottleneck. From the Data-Driven Organizations module, it’s important to remember that making choices is always about balancing the trade-offs of speed, cost, and accuracy of your outcomes. You can work to reduce one bottleneck, and that might create a different bottleneck or another type of problem. Your job is to weigh the benefits and choose the options that meet your specific goals. Then, continually monitor and optimize to make sure you are still meeting those goals.

Much of the tuning of your AWS Glue jobs will depend on how Spark distributes and processes your data based on the volume and type of data that is being ingested and the type of processing being performed on it. Spark is a distributed processing system, which means that it allocates processing across nodes. Spark takes steps to use its infrastructure to keep up with processing requests. This module doesn’t examine the details of how Spark and AWS Glue scale up and down to meet the needs of your data. However, the module covers a few general concepts to start from when trying to right size your processing components.

One relatively simple way to improve processing when you are using AWS Glue is to make sure you are on the latest available version of the AWS Glue Spark runtime engine. Both AWS and Spark continue to evolve their features, so sometimes making sure that you have the latest version will ease your bottlenecks.



Using AWS Glue workers for horizontal and vertical scaling

Horizontal scaling

Increase the number of workers that are allocated to the job.

- **Use case:** Working with large, splittable datasets
- **Example:** Processing a large .csv file

Vertical scaling

Choose a worker type with larger CPU, memory, and disk space.

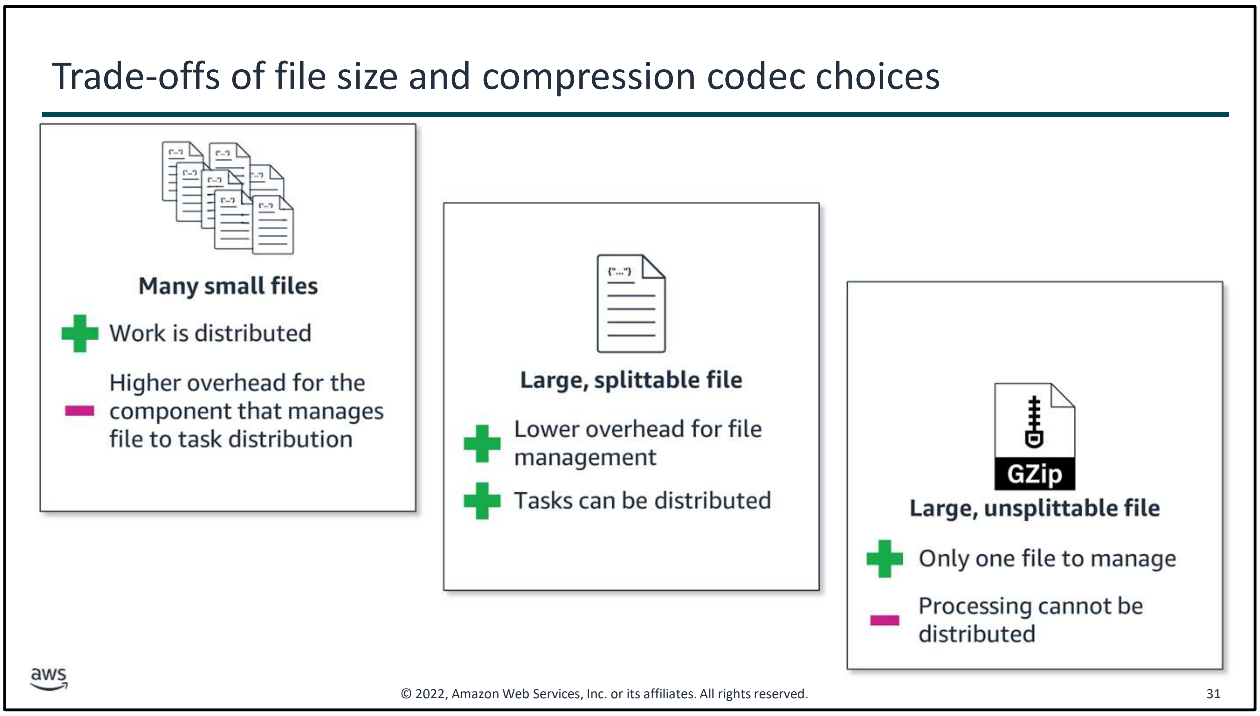
- **Use case:** Working with memory-intensive or disk-intensive applications
- **Example:** Machine learning (ML) transformations

|Slide number 30
|Instructor notes
|This slide provides a simplified look at a topic that quickly becomes complicated. A blog post in the content resources section of the course goes much deeper into this topic.
|
|Student notes
AWS Glue is designed to scale automatically. However, a data engineer might often need to make tweaks to better handle the scaling requirements of the specific workload.

With AWS Glue, you only pay for the time that your ETL job takes to run. *Data processing units* (DPUs) are the unit of measurement. A single DPU is also referred to as a *worker*. AWS Glue has different worker types, which are geared toward specific types of workloads to help you manage job latency and cost requirements. You choose the worker type and the number of workers as part of configuring your job.

You can increase the number of workers to accommodate more parallel processing. This is a good idea if you are working with large datasets that are "splittable.“ The next slide discusses more about splitting files. However, the general idea is that the file is organized in a way that makes it easy to process in chunks of records. The chunks can be processed in parallel with no impact on the outcomes.

You might choose a larger worker type if you are getting “out of memory” errors while your batch job processes. If you are working with memory-intensive applications where a lot of related data needs to be held in memory and processed together, you might need to vertically scale to a larger worker type, which increases CPU, memory, and disk space in the runtime environment.



Trade-offs of file size and compression codec choices

Many small files

+ Work is distributed

− Higher overhead for the component that manages file to task distribution

Large, splittable file

+ Lower overhead for file management

+ Tasks can be distributed

Large, unsplittable file

+ Only one file to manage

− Processing cannot be distributed

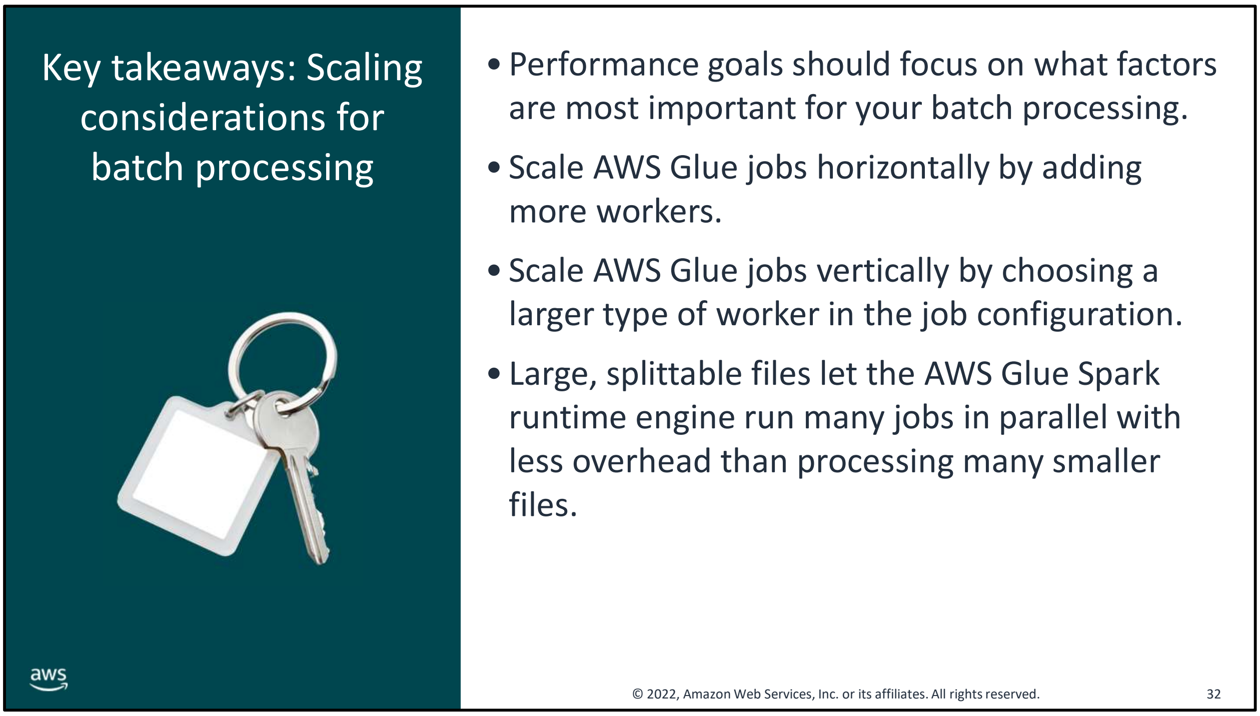
|Slide number 31
|Instructor notes
|This slide provides a simplified look related to how Spark processes files. The goal is awareness of how these types of choices come into play when you structure your batch ingestion pipelines.
|
|Student notes
For accessibility: The graphic shows that the benefit of using many small files is that work is distributed. But it has higher overhead for the component that manages file to task distribution. The benefits of using large and splittable file are that it lowers overhead for file management and the tasks can be distributed. The benefit of using large and unsplittable file is that you only have one file to manage, but the processing cannot be distributed. **End of accessibility description.**

The Ingesting and Preparing Data module discussed the need to consider file size management as part of structuring your dataset. Batch processing with the AWS Glue Spark runtime engine provides a good example of where that decision comes into play. Typically, you use a compression technology (or codec) to compress the files that you are processing to keep costs low for transfer and storage. Both the file format and the codec that you choose can impact performance.

Generally speaking, choose file formats and codecs that Spark can split. Without going into details of how Apache Spark manages and distributes work, if the file is in a format that Spark can split, then Spark can distribute the work of processing one large file and perform the work in parallel. Spark could also process many small files in parallel, but that adds file management overhead when compared to submitting a single large file for processing. However, if you choose a file format or codec that Spark cannot split, there's no way for the tasks to be distributed. In this case, processing the single large file might be inefficient.

Whenever possible, choose both file formats and compression codecs that are splittable. Apache Parquet is a good choice for a splittable file format (if you are not dependent on using a specific file format). Parquet is an open-source column-oriented data file format that is designed for efficient data storage and retrieval. Examples of codecs that support file splitting are bzip2, LZO, and Snappy.

In terms of scaling options for AWS Glue, if you have files that aren't splittable and you need to improve performance, you might scale AWS Glue workers vertically by modifying the worker type. If you are looking for faster performance and your file format and codec are splittable, you could scale AWS Glue workers horizontally by increasing the number of workers.



Key takeaways: Scaling considerations for batch processing

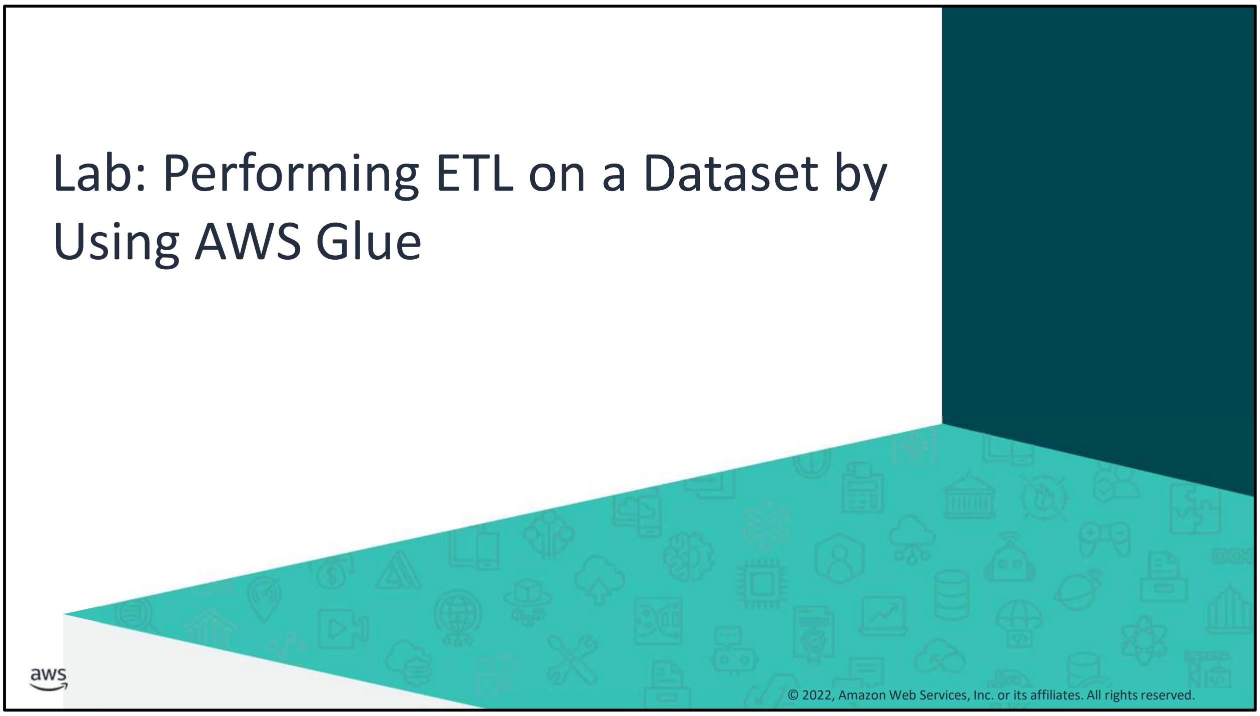
- Performance goals should focus on what factors are most important for your batch processing.
- Scale AWS Glue jobs horizontally by adding more workers.
- Scale AWS Glue jobs vertically by choosing a larger type of worker in the job configuration.
- Large, splittable files let the AWS Glue Spark runtime engine run many jobs in parallel with less overhead than processing many smaller files.

|Slide number 32
|Instructor notes
|
|Student notes

Here are a few key points to summarize this section.

Performance goals for batch processing should focus on the use case.

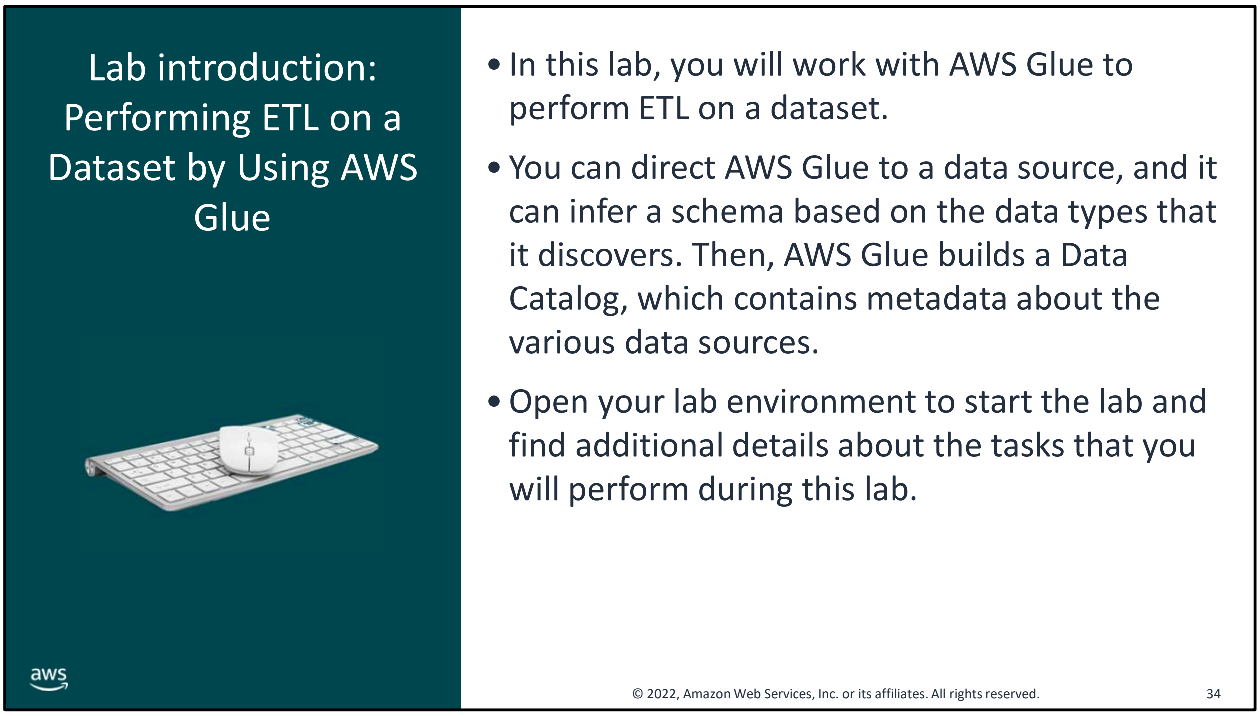
AWS Glue jobs have options for scaling horizontally or vertically based on how workers are configured for a job. Choices that you make for file type and compression codec impact how quickly AWS Glue can process your jobs.



Lab: Performing ETL on a Dataset by Using AWS Glue

|Slide number 33
|Instructor notes
|
|Student notes

You will now complete a lab. The next slide summarizes what you will do in the lab, and you will find the detailed instructions in the lab environment.

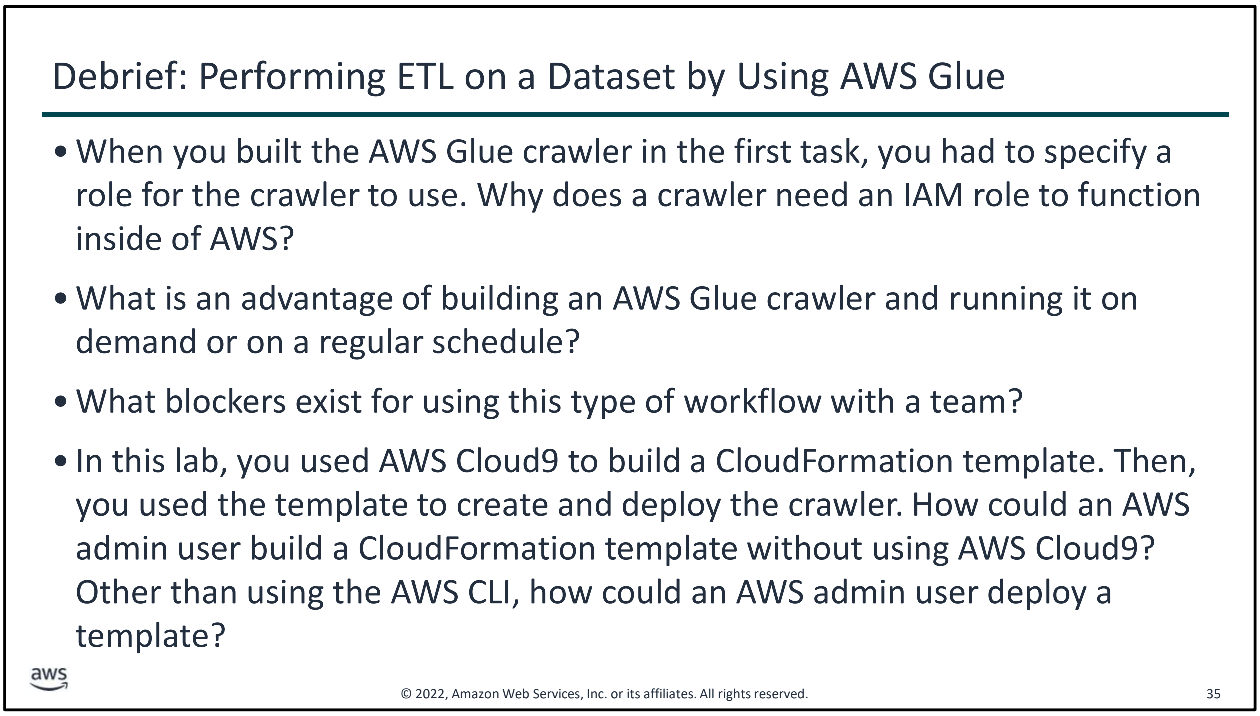


Lab introduction: Performing ETL on a Dataset by Using AWS Glue

- In this lab, you will work with AWS Glue to perform ETL on a dataset.
- You can direct AWS Glue to a data source, and it can infer a schema based on the data types that it discovers. Then, AWS Glue builds a Data Catalog, which contains metadata about the various data sources.
- Open your lab environment to start the lab and find additional details about the tasks that you will perform during this lab.

|Slide number 34
|Instructor notes
|<Add directions to find the lab guide information.>
|
|Student notes

Access the lab environment through your online course to get additional details and complete the lab.



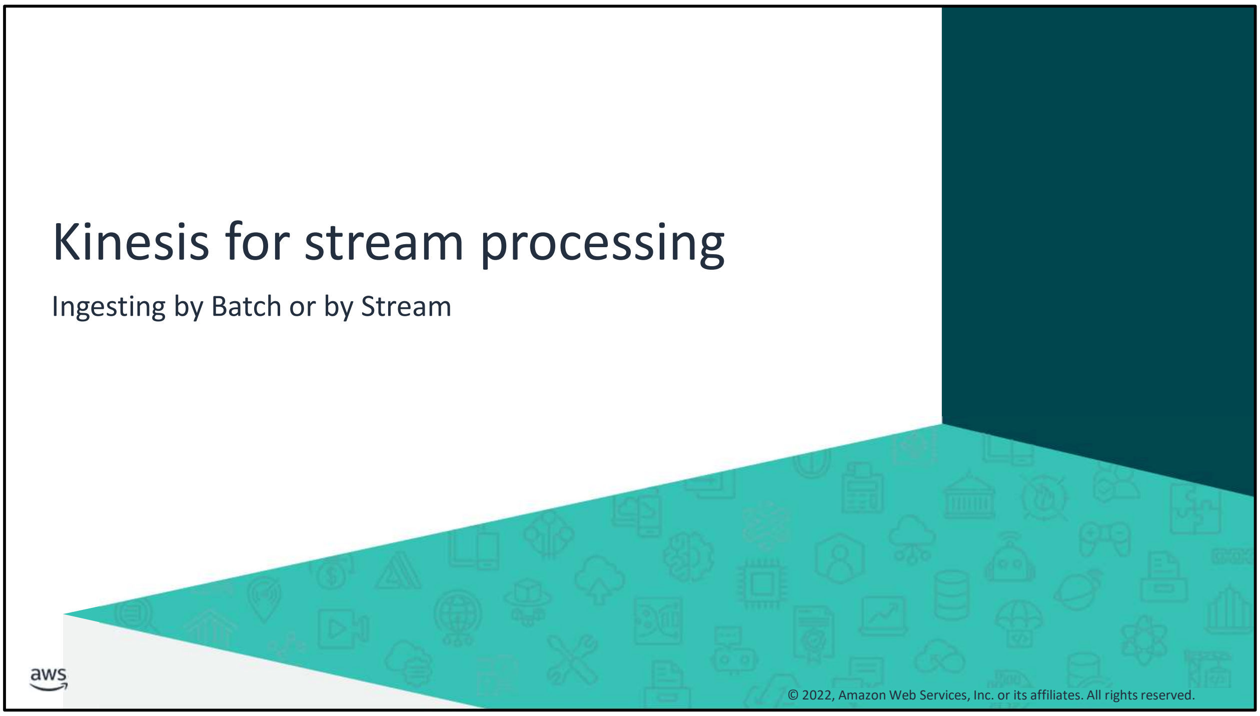
Debrief: Performing ETL on a Dataset by Using AWS Glue

- When you built the AWS Glue crawler in the first task, you had to specify a role for the crawler to use. Why does a crawler need an IAM role to function inside of AWS?
- What is an advantage of building an AWS Glue crawler and running it on demand or on a regular schedule?
- What blockers exist for using this type of workflow with a team?
- In this lab, you used AWS Cloud9 to build a CloudFormation template. Then, you used the template to create and deploy the crawler. How could an AWS admin user build a CloudFormation template without using AWS Cloud9? Other than using the AWS CLI, how could an AWS admin user deploy a template?

|Slide number 35
|Instructor notes
|Q1 – **Example strong response:** The crawler needs IAM permissions to perform actions on a user's behalf when calling other services, such as retrieving data from Amazon S3. In addition to crawlers, jobs and endpoints in AWS Glue need a role to perform any action.
|Q2 – **Example strong response:** Production datasets can change quickly—this is the velocity part of data. By using a crawler, you can include new data within the overall dataset as it changes. When you run a crawler on demand or on a schedule, all calculations that are made from the data and all insights in reports will factor in the latest data.
|Q3 – **Example strong response:** Team members would need to understand and be proficient with SQL to be able to build queries in Athena. They would also need to understand the schema of each database and its tables to be able to write effective queries. Team members would need permissions to access the services that are used in the workflow. They also need to understand what they can do, when, and why. Team members would also need to understand where data is stored and how that data is pulled into a crawler and then presented in services, such as Athena, for analysis.
|Q4 – **Example strong response:** You can use CloudFormation Designer or any text editor to create and save templates. To deploy a template, you could use the CloudFormation console.
|

|Student notes

Your instructor might review these questions with you, or you might review them on your own. Use this opportunity to extend your thinking about the tasks that you performed during the lab.



Kinesis for stream processing

Ingesting by Batch or by Stream

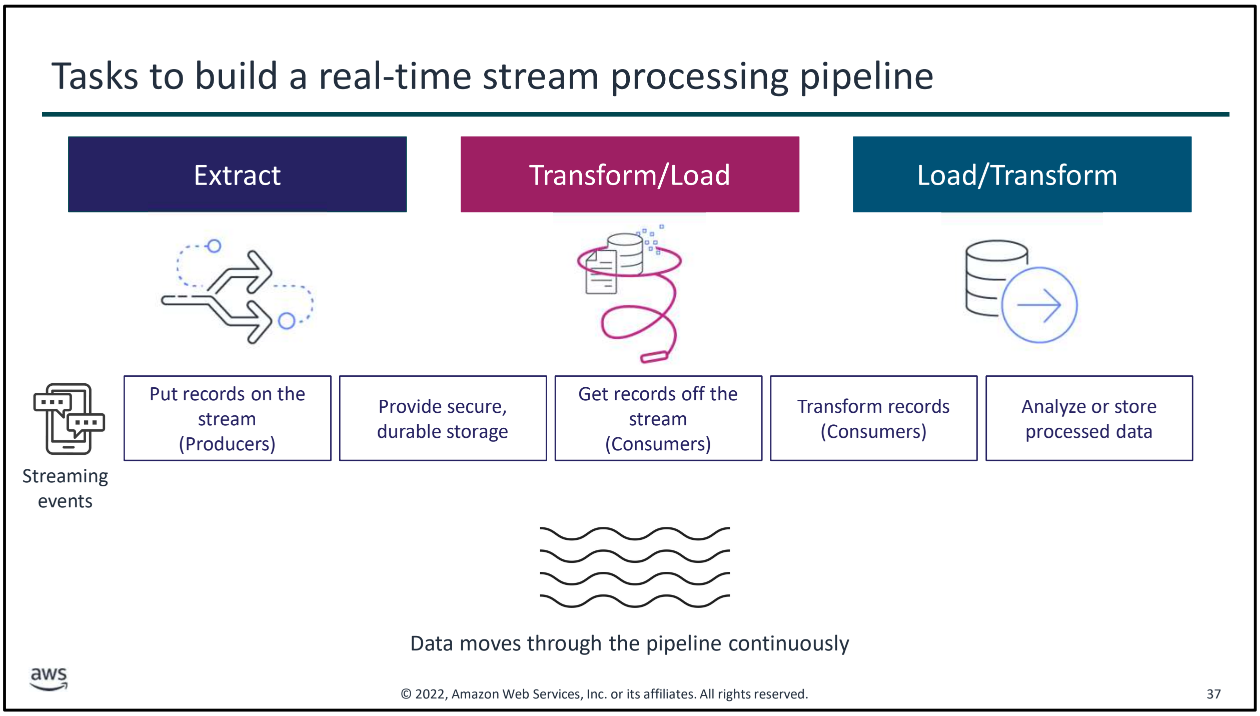
|Slide number 36

|Instructor notes

|

|Student notes

This section introduces the steps in streaming ingestion and the tools that can help you to perform them.



Tasks to build a real-time stream processing pipeline

Extract

Transform/Load

Load/Transform

Streaming events

Put records on the stream (Producers)

Provide secure, durable storage

Get records off the stream (Consumers)

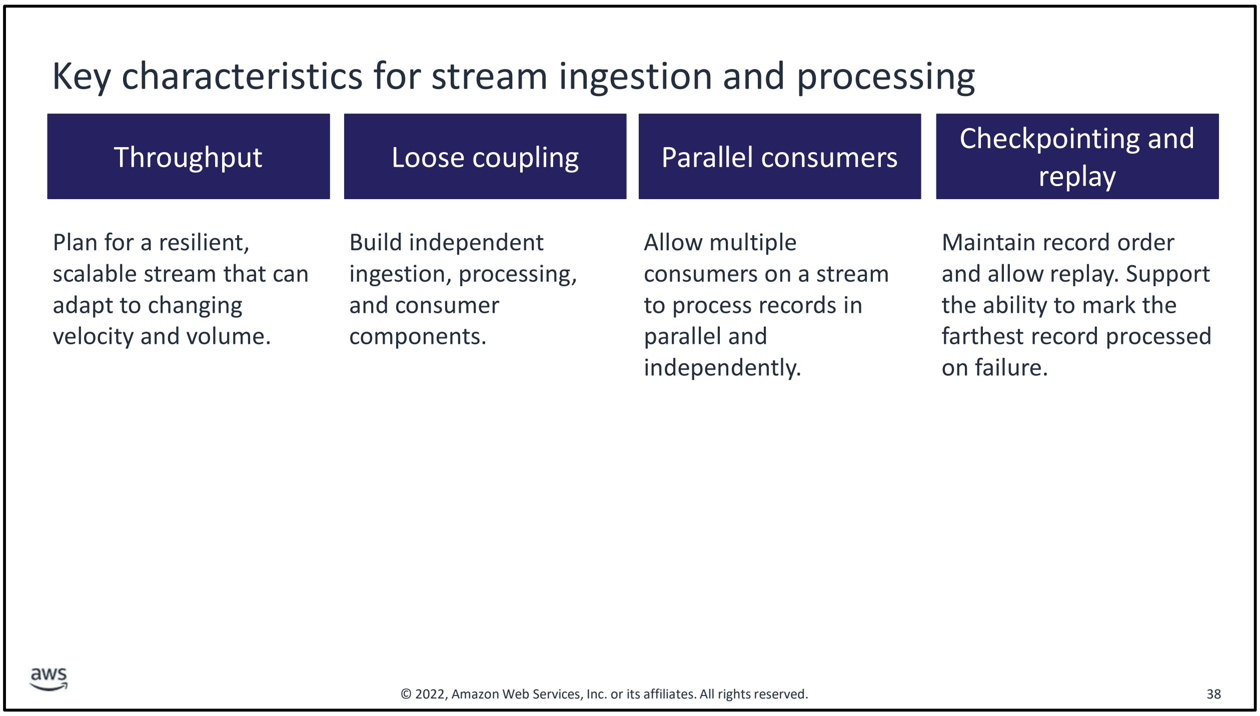
Transform records (Consumers)

Analyze or store processed data

Data moves through the pipeline continuously

|Slide number 37
|Instructor notes
|
|Student notes

With streaming pipelines, the main steps of the ingestion process involve producers, a data stream, and consumers. *Producers* put records on the stream, and the stream provides durable storage from which consumers can get and process records. *Consumers* read records from the stream and process them in batches. The results of the processing are made available to analysis and visualization tools for real-time analytics or might be stored for later analysis.



Key characteristics for stream ingestion and processing

Throughput

Plan for a resilient, scalable stream that can adapt to changing velocity and volume.

Loose coupling

Build independent ingestion, processing, and consumer components.

Parallel consumers

Allow multiple consumers on a stream to process records in parallel and independently.

Checkpointing and replay

Maintain record order and allow replay. Support the ability to mark the farthest record processed on failure.

|Slide number 38

|Instructor notes

|This slide summarizes the characteristics that are listed in the **Streaming ingest and stream processing** section of the *Data Analytics Lens: AWS Well-Architected Framework*. The direct link is in the course resources. Some topics that are beyond the scope of the module were excluded from this list.

|

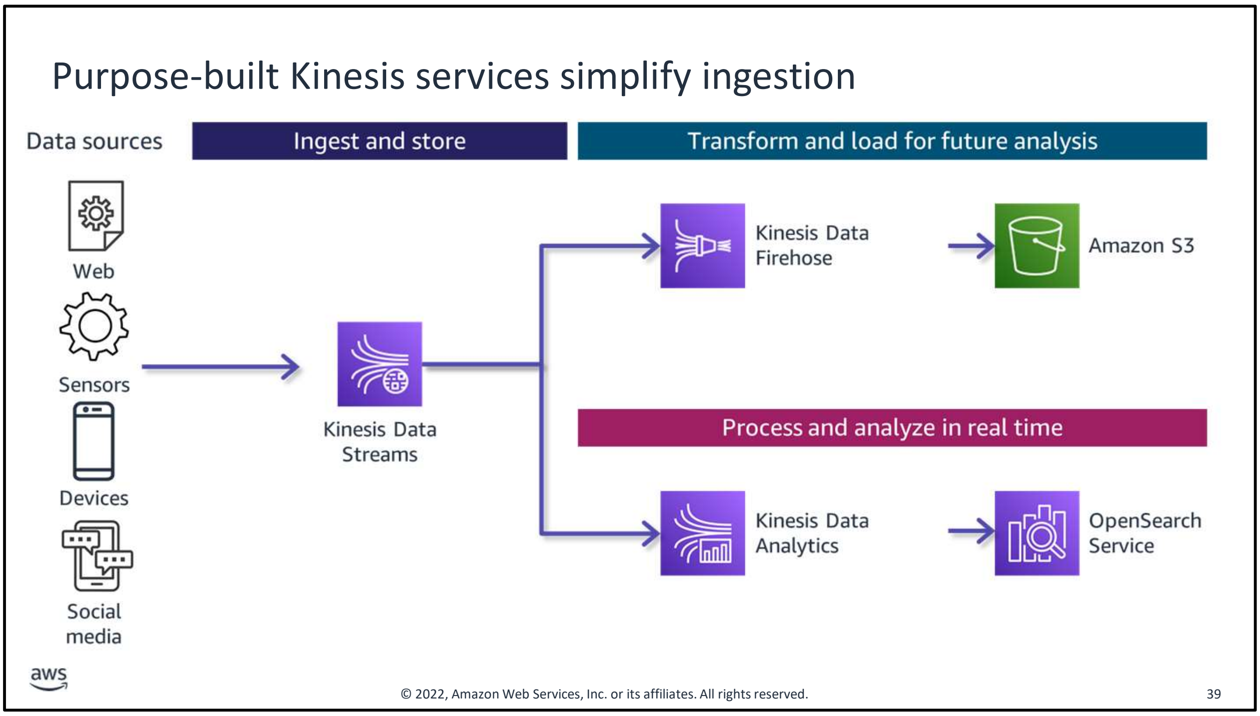
|Student notes

This slide summarizes the characteristics for streaming ingest and stream processing pipelines, as described in the *Data Analytics Lens: AWS Well-Architected Framework*.

Streaming pipelines support real-time processing of high-velocity data. Streaming pipelines need to be able to scale quickly to meet increased throughput requirements. Your solution should automatically discover new data shards or partitions that are added to handle incoming velocity.

Although data moves through the ingest, store, process, and analysis phases quickly, build components independently and loosely couple them. Your stream design should also allow for more than one consumer to process the same stream of data at the same time, independent of each other.

Given the continuous nature of the data, streams need to be highly durable to avoid losing records. To handle failures, stream processing techniques should maintain record order, provide methods to replay a failed record, and use checkpointing to mark the farthest record in the stream that was successfully processed.



Purpose-built Kinesis services simplify ingestion

Data sources

Ingest and store

Transform and load for future analysis

Web

Sensors

Devices

Social media

Kinesis Data Streams

Kinesis Data Firehose

Amazon S3

Process and analyze in real time

Kinesis Data Analytics

OpenSearch Service

|Slide number 39
|Instructor notes
|
|Student notes

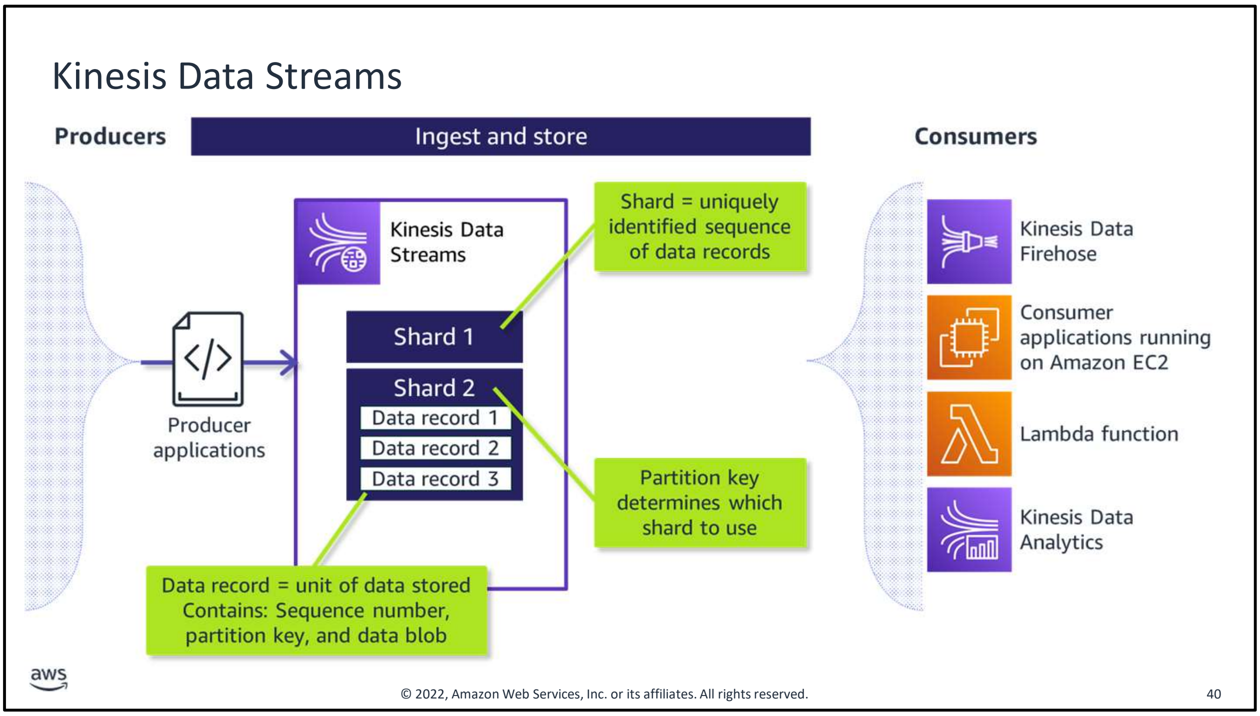
In the Design Principles and Patterns for Data Pipelines module, you looked briefly at the unique characteristics of streaming pipelines. AWS and other providers offer services that are specifically designed to ingest, process, and store streaming data for analytics. This slide looks specifically at streaming ingestion by using Amazon Kinesis services.

Amazon Kinesis Data Streams is a streaming service that meets the characteristics from the previous slide. The service continuously captures gigabytes of data and supports multiple sources and targets in parallel. A stream can scale up and down and encrypt sensitive data automatically. Kinesis Data Streams also provides mechanisms to manage the success or failure of stream processing.

This high-level diagram highlights two paths for streaming ingestion. In the top path, streaming data is collected into the stream and stored for future use. In the second path, data is processed as it is ingested to feed real-time analytics. In the example on the slide, a combination of services from the Amazon Kinesis family provides the pipeline infrastructure for these streaming applications.

Producers put data onto a Kinesis Data Stream stream, which provides durable storage for the data. More than one producer can write to the stream. Consumers get the data from the stream and process it. The stream is a kind of buffer between the incoming data and the processors of that data. A stream can have multiple consumers.

AWS has developed additional streaming services that are purpose built to consume stream data for common scenarios: Amazon Kinesis Data Firehose for streaming data to data stores and Amazon Kinesis Data Analytics for real-time analytics.



Kinesis Data Streams

Producers

Ingest and store

Consumers

Producer applications

Kinesis Data Streams

Shard 1

Shard 2

Data record 1

Data record 2

Data record 3

Shard = uniquely identified sequence of data records

Partition key determines which shard to use

Data record = unit of data stored
Contains: Sequence number, partition key, and data blob

Kinesis Data Firehose

Consumer applications running on Amazon EC2

Lambda function

Kinesis Data Analytics

|Slide number 40
|Instructor notes
|
|Student notes

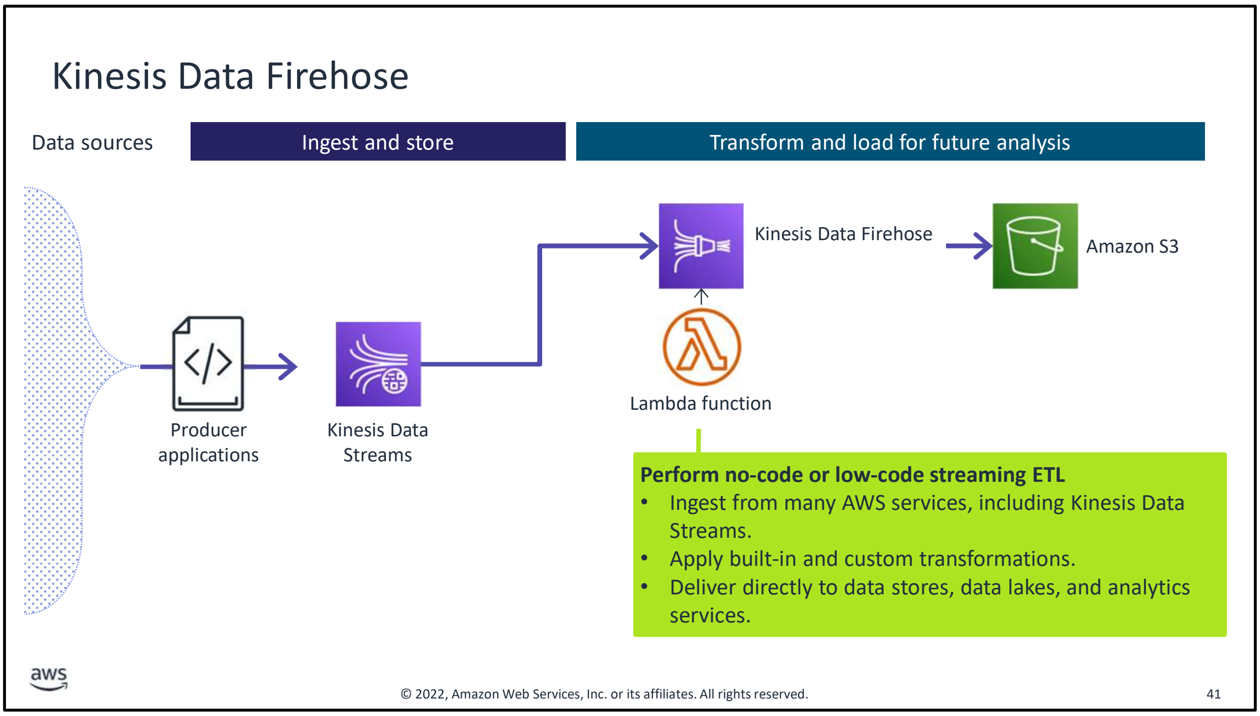
Some AWS services can directly write to the stream. For other producers, you need a producer application to capture streaming events from the producer and put them on the stream. The Amazon Kinesis Producer Library (KPL) simplifies the coding that is required to write a producer application.

Data is put onto a Kinesis data stream as data records, and records are stored in shards. Each record includes a unique sequence number, partition key, and data blob. The partition key will determine which shard the record will be written to.

In addition to the purpose-built stream consumers that were mentioned earlier, you can also build consumer applications by using the Amazon Kinesis Client Library (KCL) to read data from the stream and process it. Custom consumer applications can run on Amazon Elastic Compute Cloud (Amazon EC2) instances.

AWS Lambda functions can also be consumers of a stream. This option provides serverless processing of stream data, and Lambda handles aspects of error handling and stream management for you.

Streaming ingest and processing systems need to be secure by default. You can configure Kinesis data streams to use AWS Identity and Access Management (IAM) policies to grant access and encrypt sensitive data automatically. You will need to use the principle of least privilege to grant access to the streaming APIs and infrastructure, and encrypt data at rest and in transit.



Kinesis Data Firehose

Data sources

Ingest and store

Transform and load for future analysis

Producer applications

Kinesis Data Streams

Kinesis Data Firehose

Amazon S3

Lambda function

Perform no-code or low-code streaming ETL

- Ingest from many AWS services, including Kinesis Data Streams.
- Apply built-in and custom transformations.
- Deliver directly to data stores, data lakes, and analytics services.

|Slide number 41

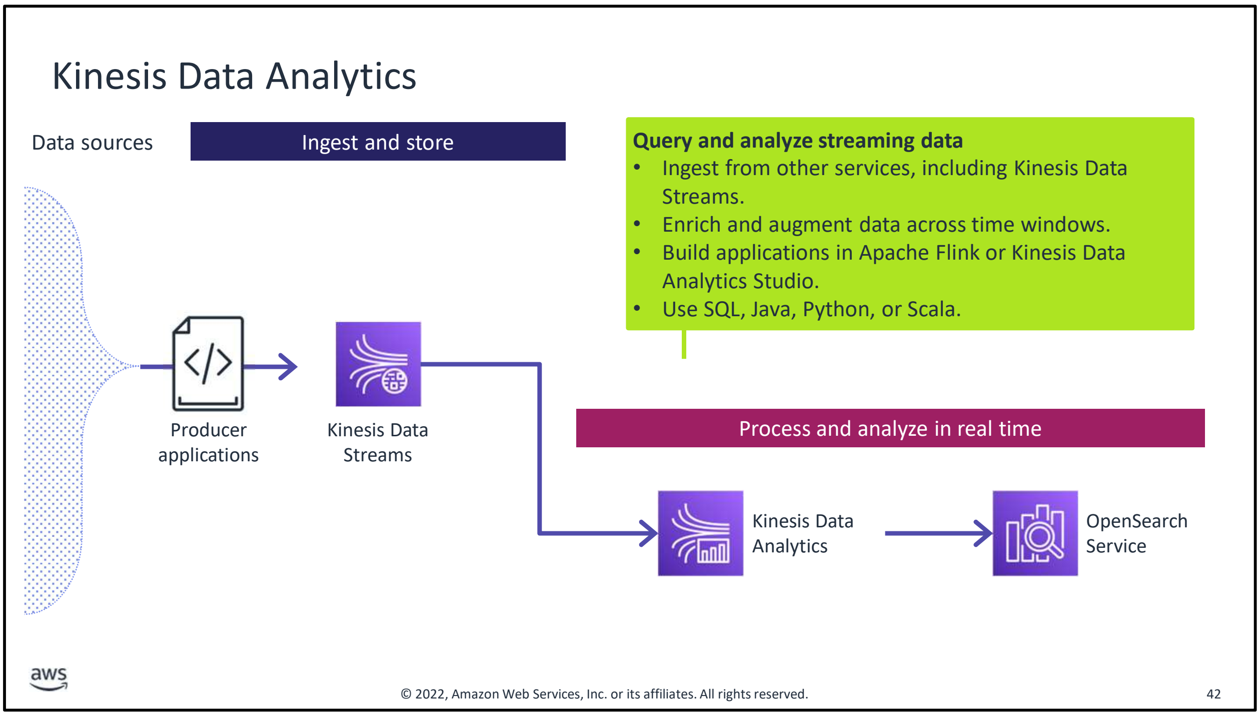
|Instructor notes

|

|Student notes

Kinesis Data Firehose is designed to ingest and deliver streaming data directly to data lakes on Amazon S3, as well as data stores and analytics services such as Amazon Redshift and Amazon OpenSearch Service. Kinesis Data Firehose also has the ability to run built-in transformations or custom transformations that you create by using a Lambda function that runs on the streaming data before it is delivered to the specified destination. If you need to get streaming data and clean or reformat it before storing it to your data lake for additional processing or analysis, you can use Kinesis Data Firehose without writing much code.

A common use case for Kinesis Data Firehose is to stream data to your data lake and convert the data into the format that is required for analysis without needing to build any other pipeline components.



Kinesis Data Analytics

Data sources

Ingest and store

Query and analyze streaming data

- Ingest from other services, including Kinesis Data Streams.
- Enrich and augment data across time windows.
- Build applications in Apache Flink or Kinesis Data Analytics Studio.
- Use SQL, Java, Python, or Scala.

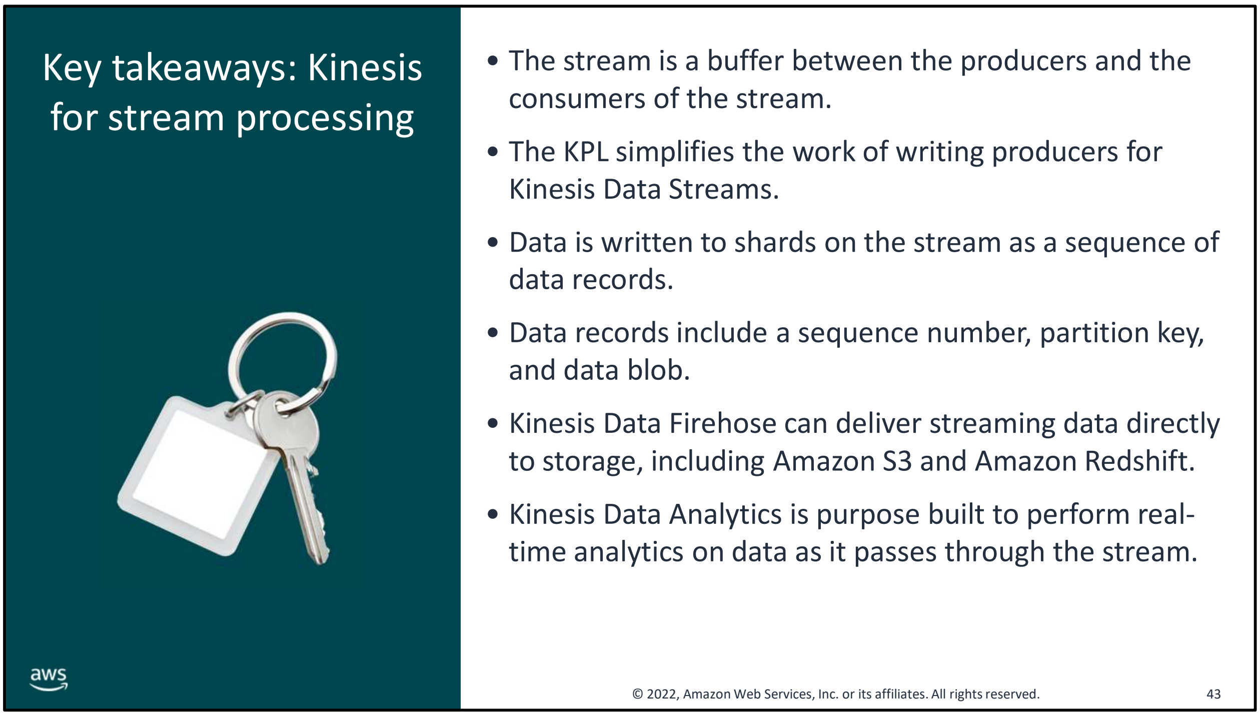
Process and analyze in real time

|Slide number 42
|Instructor notes
|
|Student notes

Kinesis Data Analytics is purpose built to perform analytics on streaming data as it passes through the stream and to deliver data directly to tools for real-time analytics. For example, you might perform calculations to derive new fields or to aggregate values over a sliding time window within the incoming stream. You could then write the results to OpenSearch Service, where the results are available for Amazon OpenSearch Dashboards or other analysis and visualizations tools. You can integrate a data source or destination with minimal code by using Kinesis Data Analytics libraries to integrate with AWS services including Amazon S3, OpenSearch Service, DynamoDB, Kinesis Data Streams, and Kinesis Data Firehose.

Apache Flink is a popular framework and engine to process data streams, and Kinesis Data Analytics provides the underlying infrastructure for your Flink applications. It handles core capabilities, such as provisioning compute resources and automatic scaling. You can use Java, Scala, or SQL for your Flink applications.

Kinesis Data Analytics Studio runs on and produces Flink applications and provides the ability to code in notebooks by using SQL, Python, and Scala. A notebook is a web-based development environment that provides a simple interactive development experience combined with the advanced capabilities that Flink provides. This makes it easier to perform more advanced analytics without as much coding expertise.



Key takeaways: Kinesis for stream processing

- The stream is a buffer between the producers and the consumers of the stream.
- The KPL simplifies the work of writing producers for Kinesis Data Streams.
- Data is written to shards on the stream as a sequence of data records.
- Data records include a sequence number, partition key, and data blob.
- Kinesis Data Firehose can deliver streaming data directly to storage, including Amazon S3 and Amazon Redshift.
- Kinesis Data Analytics is purpose built to perform real-time analytics on data as it passes through the stream.

|Slide number 43
|Instructor notes
|
|Student notes

Here are a few key points to summarize this section.

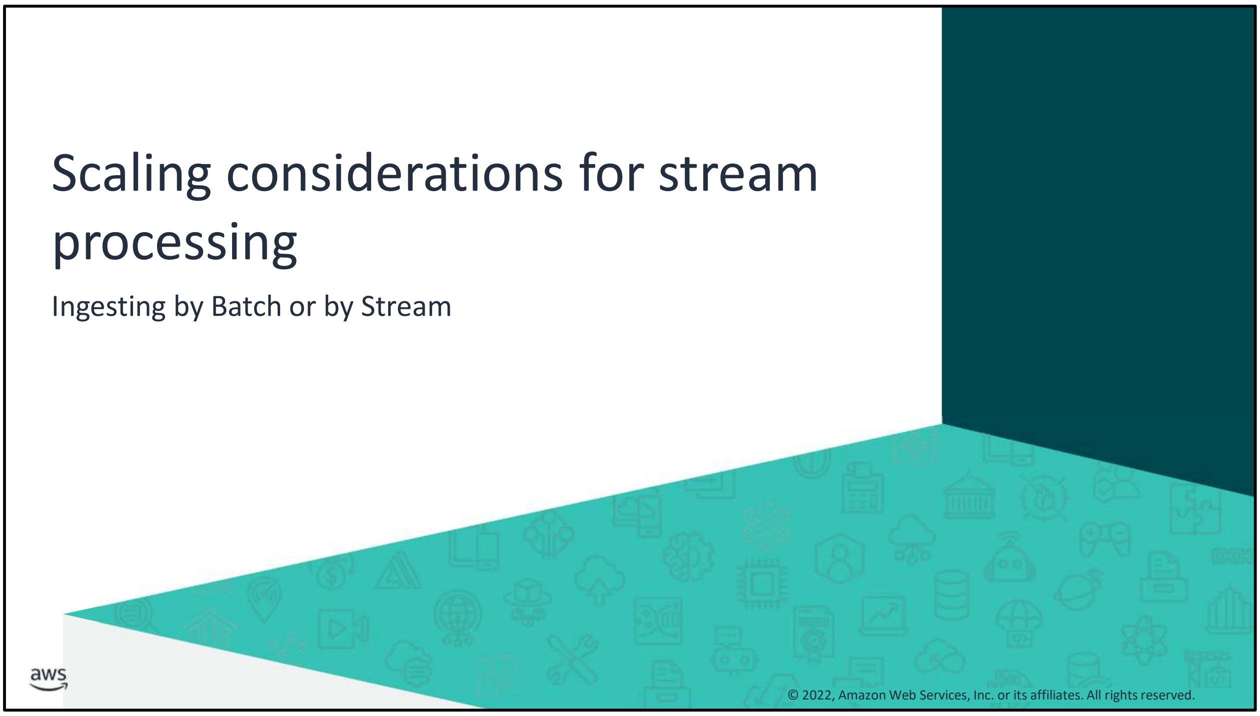
In streaming scenarios, the stream provides temporary storage and is a buffer between the producers, who put data on the stream, and the consumers, who read and process data from the stream.

Stream records are stored in shards and include a sequence number, partition key, and a data blob.

The Kinesis Producer Library (KPL) simplifies writing producers that put records on a Kinesis data stream.

Kinesis Data Firehose is designed to deliver streaming data directly to storage, and the service can store data to Amazon S3 and Amazon Redshift.

Kinesis Data Analytics is purpose built to perform analytics on streaming data for real-time analytics.



Scaling considerations for stream processing

Ingesting by Batch or by Stream

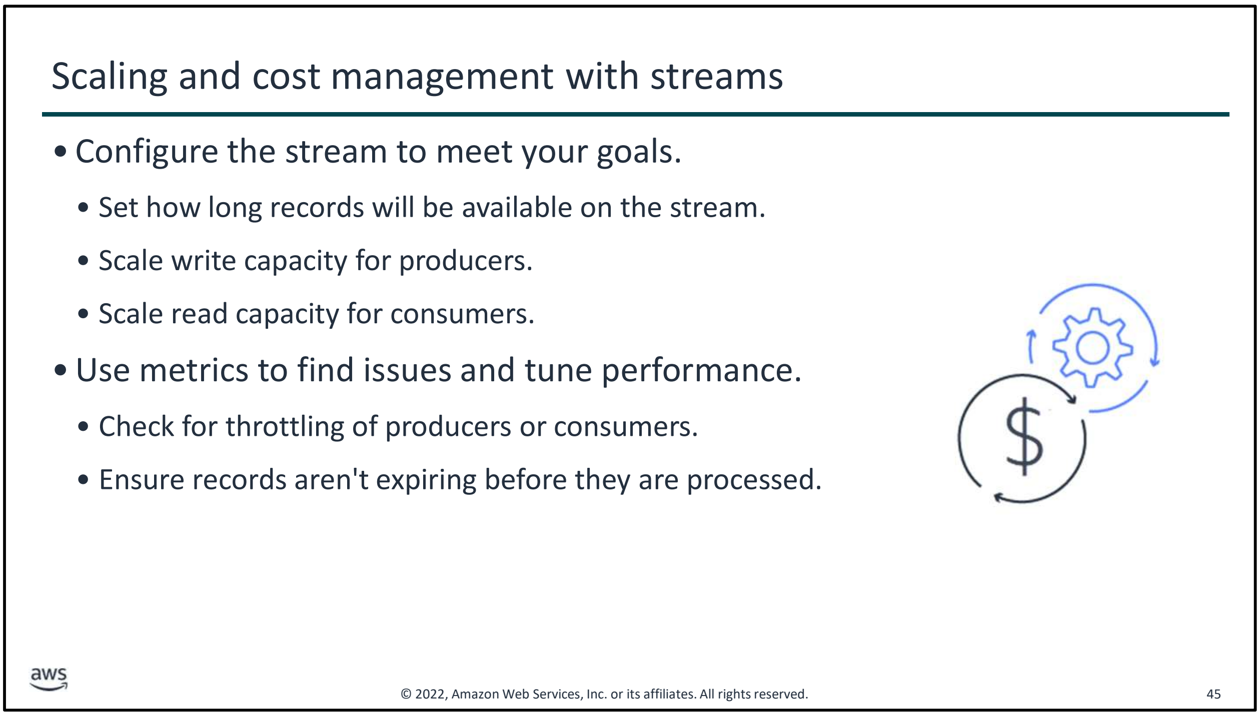
|Slide number 44

|Instructor notes

|

|Student notes

This section looks briefly at scaling considerations for stream processing pipelines.



Scaling and cost management with streams

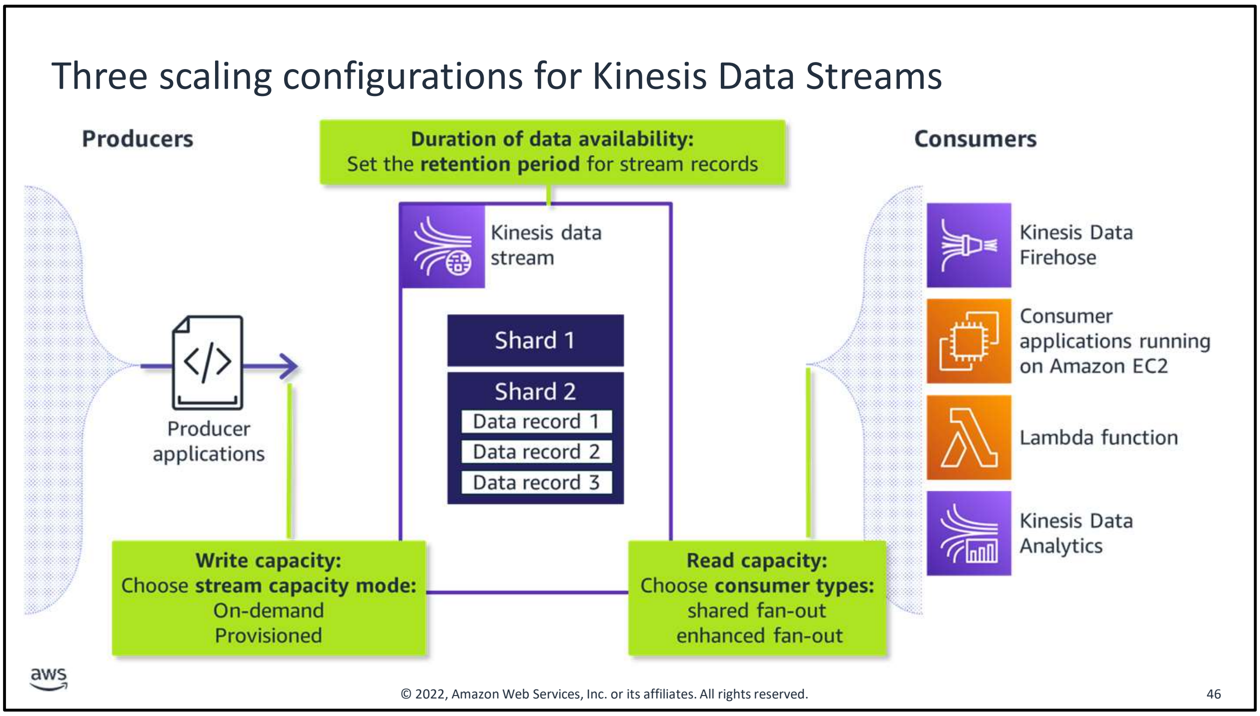
- Configure the stream to meet your goals.
  - Set how long records will be available on the stream.
  - Scale write capacity for producers.
  - Scale read capacity for consumers.
- Use metrics to find issues and tune performance.
  - Check for throttling of producers or consumers.
  - Ensure records aren't expiring before they are processed.

|Slide number 45
|Instructor notes
|
|Student notes

Managing scale for your streaming pipeline is about managing throughput. Remember that the stream itself provides temporary storage, so you want to make sure that all the consumers that need to act on records in the stream can complete their work before the record expires and is gone from the stream. If consumers are having difficulty completing processing before records expire, you could increase the time that records are available on the stream.

Beyond that, you want to optimize how quickly you can write records to the stream and consume them off the stream to make them available for analysis. You need to think about how much data you are going to write to the stream and at what velocity, as well as how many consumers need to read data from the stream and how quickly they need to get the data.

When you start using your stream, use metrics to check that processing is not being throttled. This would indicate the need to address the write or read capacity of the stream. Metrics that highlight expiring records can help you to ensure that your stream is keeping records long enough for your consumers to get to them and complete processing.



Three scaling configurations for Kinesis Data Streams

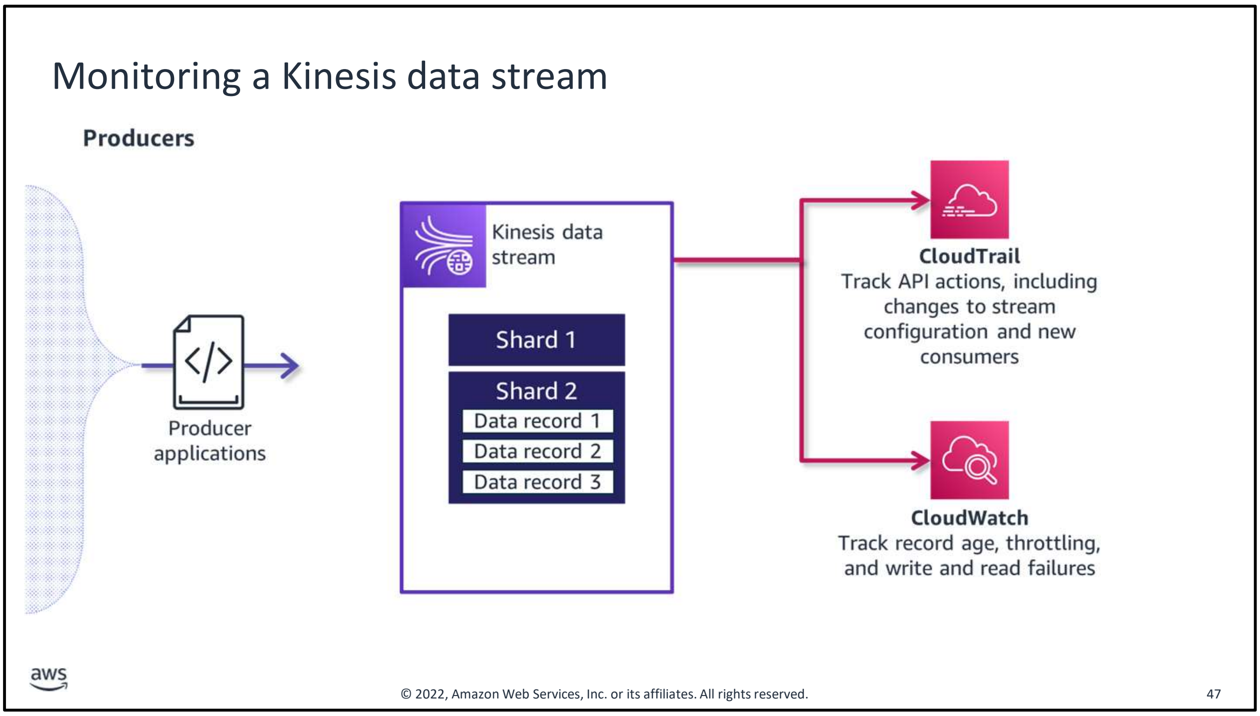
|Slide number 46
|Instructor notes
|
|Student notes

Kinesis Data Streams provides configuration options to help you address the scaling needs of your pipeline. Here is a summary of the three primary controls called out on the slide:

The retention period tells the stream how long to keep records on the stream before they expire. This is configurable, but costs increase as the retention period increases. A stream’s retention period is set to a default of 24 hours after creation. You can increase the retention period up to 8,760 hours (365 days).

The write capacity is controlled by how many shards are allocated to the stream, which sets the limit on how much data can be written to the stream in a given period. You can choose either on-demand or provisioned capacity mode. Data streams that use *on-demand mode* require no capacity planning and automatically scale to handle gigabytes of write and read throughput per minute. With the on-demand mode, Kinesis Data Streams automatically manages the shards to provide the necessary throughput. If you choose *provisioned mode*, you must specify the number of shards for the data stream. The total capacity of a data stream is the sum of the capacities of its shards. You can increase or decrease the number of shards in a data stream as needed.

Read capacity scales the throughput for consumers of your stream. You can choose between shared fan-out and enhanced fan-out consumer types. When a consumer uses the *shared fan-out* configuration, it shares a single pipe of read capacity with other consumers of the stream. The *enhanced fan-out* configuration gives the consumer its own pipe, which increases the throughput for that consumer.



Monitoring a Kinesis data stream

Producers

Producer applications

Kinesis data stream

Shard 1

Shard 2

Data record 1

Data record 2

Data record 3

CloudTrail
Track API actions, including changes to stream configuration and new consumers

CloudWatch
Track record age, throttling, and write and read failures

|Slide number 47
|Instructor notes
|
|Student notes

For accessibility: The Graphic shows that both CloudTrail and CloudWatch are integrated with Kinesis Data Streams. CloudTrail tracks API actions, including changes to stream configuration and new consumers. CloudWatch tracks record age, throttling, and write and read failures. **End of accessibility description.**

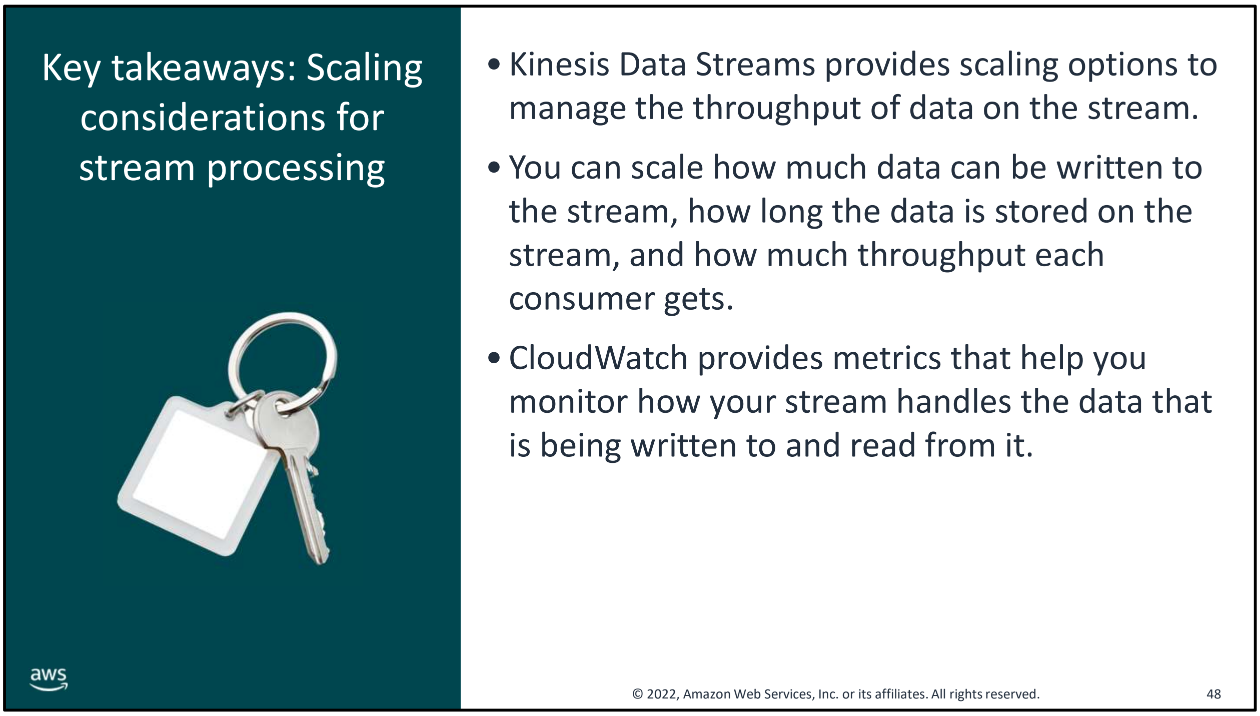
Both CloudTrail and CloudWatch are integrated with Kinesis Data Streams. The metrics that you configure for your streams are automatically collected and pushed to CloudWatch every minute.

The following metrics are recommended for your Kinesis data streams and align to the scaling considerations that were discussed on the previous slide:

- **GetRecords.IteratorAgeMilliseconds:** This metric indicates whether there's a risk of data loss due to record expiration.
- **ReadProvisionedThroughputExceeded:** This metric can help you determine whether your consumers are being throttled when they try to read from the stream.
- **WriteProvisionedThroughputExceeded:** This metric can help you determine whether

your producers are being throttled when they try to write to the stream.

- **PutRecord.Success** or **PutRecords.Success:** This metric indicates when records are failing to the stream.
- **GetRecords.Success:** This metric indicates when records are failing when a consumer tries to read them from the stream.



Key takeaways: Scaling considerations for stream processing

- Kinesis Data Streams provides scaling options to manage the throughput of data on the stream.
- You can scale how much data can be written to the stream, how long the data is stored on the stream, and how much throughput each consumer gets.
- CloudWatch provides metrics that help you monitor how your stream handles the data that is being written to and read from it.

|Slide number 48

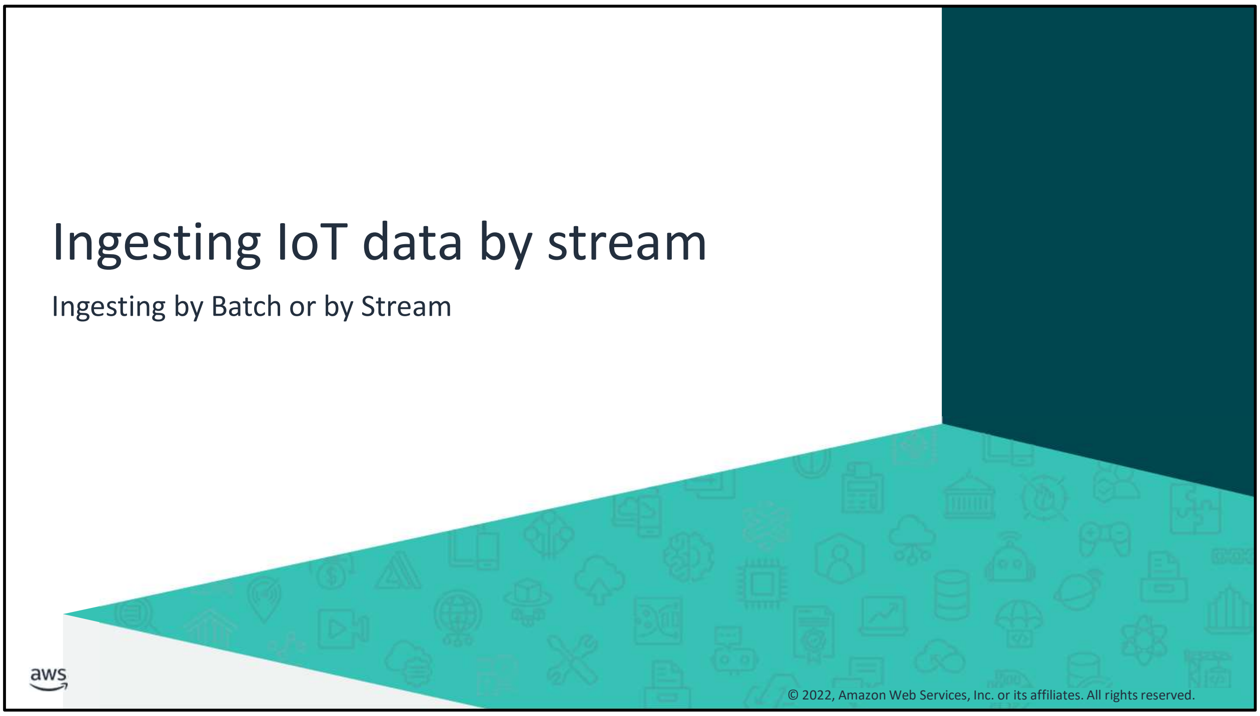
|Instructor notes

|

|Student notes

Here are a few key points to summarize this section.

Kinesis Data Streams allows you to scale how long data remains on the stream, the stream's read capacity, and the stream's write capacity. Use CloudWatch metrics to help you monitor how your stream is performing.



Ingesting IoT data by stream

Ingesting by Batch or by Stream

|Slide number 49

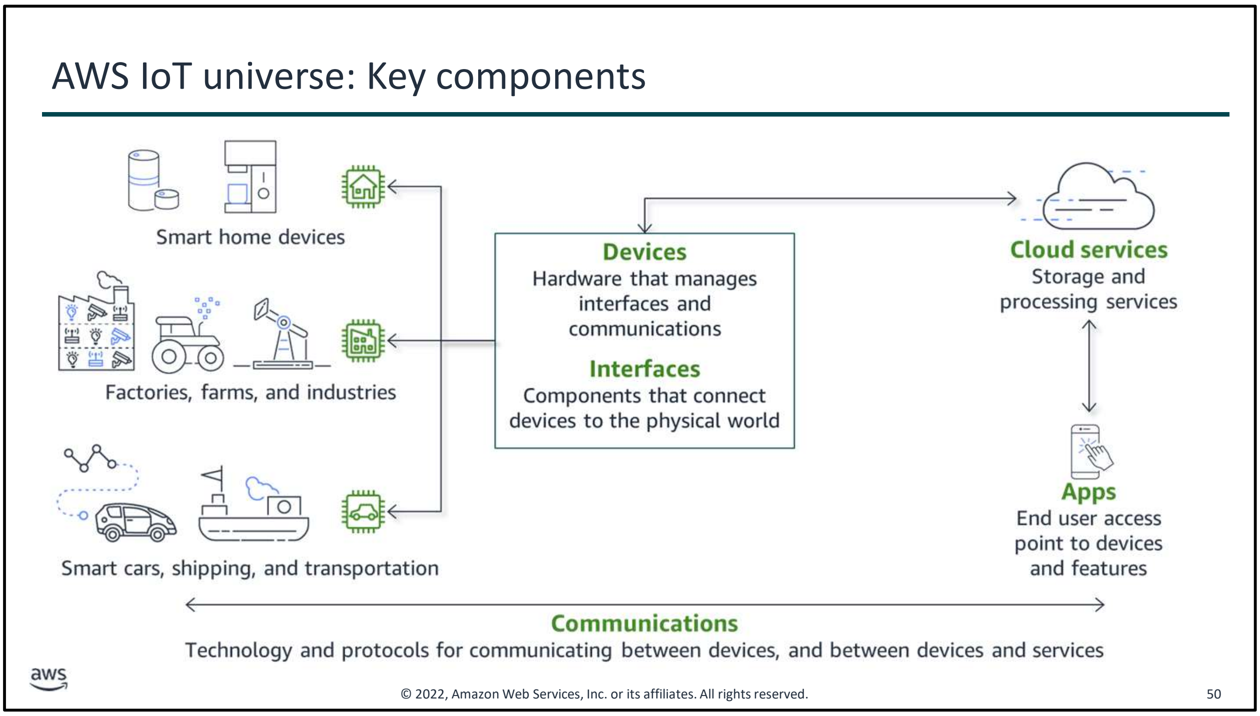
|Instructor notes

|A supplemental deck called IoT Use Case Slides is provided, and other modules don’t directly reference this supplemental deck. It is designed to provide a real-world example of an IoT pipeline, with sections for each phase of the pipeline, including ingesting and preparing, storing, and analyzing and visualization. You might choose to review each part as you deliver the related module, or you might walk through the use case as a separate review after covering all modules.

|

|Student notes

IoT data is often ingested by stream, although you could ingest this data by batch. This section introduces key concepts for ingesting IoT data by stream using AWS services.



AWS IoT universe: Key components

Smart home devices

Factories, farms, and industries

Smart cars, shipping, and transportation

Devices
Hardware that manages interfaces and communications

Interfaces
Components that connect devices to the physical world

Cloud services
Storage and processing services

Apps
End user access point to devices and features

Communications
Technology and protocols for communicating between devices, and between devices and services

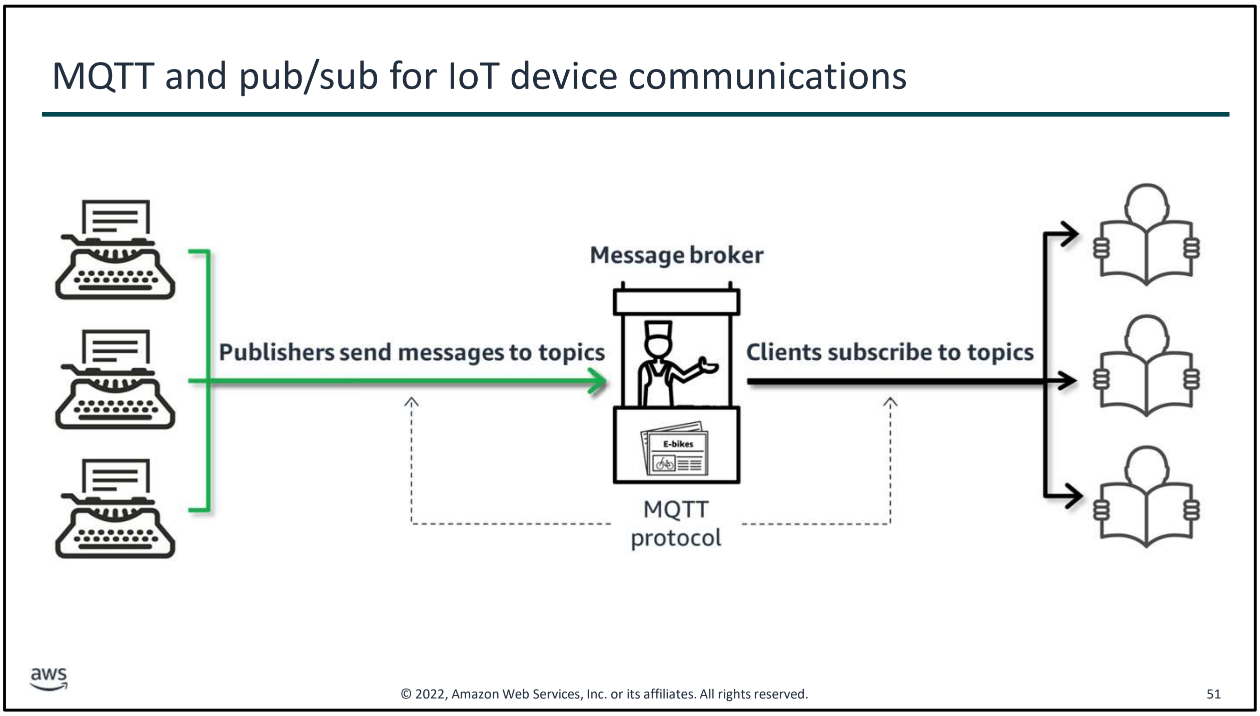
|Slide number 50
|Instructor notes
|
|Student notes

The IoT universe is made up of many different devices. They communicate to cloud resources to provide information about what is happening in their environment, and these devices can receive commands from cloud resources to modify their behavior. For example, a smart plug sends information about itself (where it is, what it is, and whether it’s on or off) to the internet, where you can access it on your phone. You use the app to tell the smart plug to turn itself off, and that is communicated back to the device.

You can break the main components of the IoT universe into five categories:

- **Devices:** A device is a type of hardware that manages interfaces and communications. Devices are usually located close to the real-world interfaces that they monitor and control. Devices can include computing and storage resources, such as microcontrollers, a CPU, and memory.
- **Interfaces:** Interfaces connect a device to the physical world. This might be a user interface to give input or receive output, sensors that detect information about their environment, or actuators that the device can use to control something in the outside world.

- **Communications:** Devices need to send information to each other and to cloud services. They also need to receive communications from cloud services. Devices do this through a variety of technologies and protocols. For example, a device might send information by using a protocol, such as HTTP, traveling over the cellular network. The contents of the messages that IoT devices send might be semistructured or unstructured.
- **Cloud services:** Cloud services like those that AWS provides are distributed, large-scale data storage and processing services that are connected to the internet. This is how data from IoT devices connects with the services where you can perform analytics or ML on the data.
- **Apps:** Apps give end users access to IoT devices and to the features of the cloud services to which those devices are connected. For example, an app can provide the ability to control your lights from your phone.



MQTT and pub/sub for IoT device communications

Message broker

Publishers send messages to topics

Clients subscribe to topics

MQTT protocol

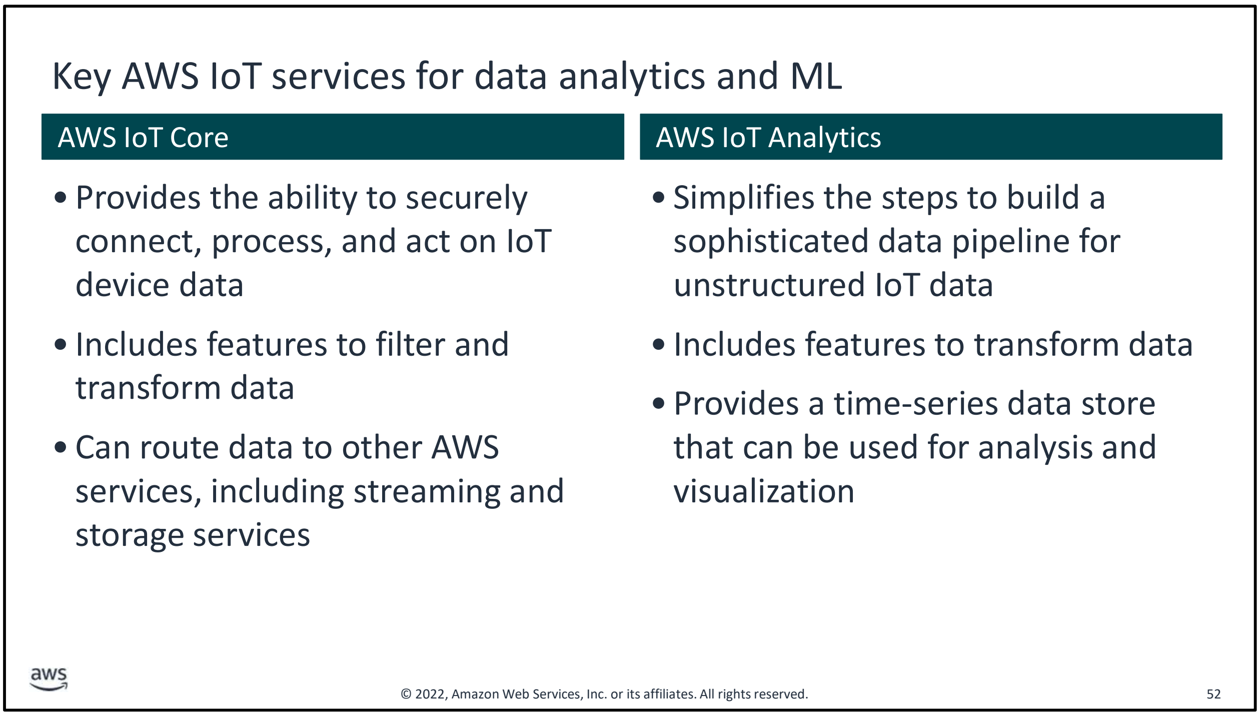
|Slide number 51
|Instructor notes
|
|Student notes

One common protocol for IoT is Message Queuing Telemetry Transport (MQTT). This lightweight protocol uses a publish/subscribe, or pub/sub, model to send and receive messages. The advantage of using MQTT for IoT is that it requires a small code footprint and very little bandwidth, which is important for small, remote devices.

The advantage of a pub/sub model is that it decouples the message creators from the message consumers. Publishers don’t need to know about subscribers— they just publish to the topic. Likewise, subscribers don't need to be directly connected to the publisher. They subscribe to topics of interest. This is especially helpful for IoT devices that are not continuously connected to the internet, or for those that experience unplanned disconnections. With this model, devices send and receive messages when they are online.

With IoT devices, a device might be a publisher and a subscriber. For example, the device on a light sends a message that it’s on, and your smart home app is a subscriber to that topic. When you send a command to turn the light off, the device is a subscriber to a topic that gets the message and turns the light off.

To discuss ingesting IoT data into an analytics pipeline, the rest of this section looks only at the messages that come from a device and are delivered to cloud services, where they can be used for analytics and ML applications. IoT data is often ingested as streaming data, given that many devices usually feed into a single pipeline and that messages from those devices are generated every few seconds or minutes.



Key AWS IoT services for data analytics and ML

AWS IoT Core

- Provides the ability to securely connect, process, and act on IoT device data
- Includes features to filter and transform data
- Can route data to other AWS services, including streaming and storage services

AWS IoT Analytics

- Simplifies the steps to build a sophisticated data pipeline for unstructured IoT data
- Includes features to transform data
- Provides a time-series data store that can be used for analysis and visualization

|Slide number 52

|Instructor notes

|AWS provides many IoT services. To focus on the data pipeline discussion, this module only looks at AWS IoT Core and AWS IoT Analytics. Resources for learning more about IoT are linked in the content resources section.

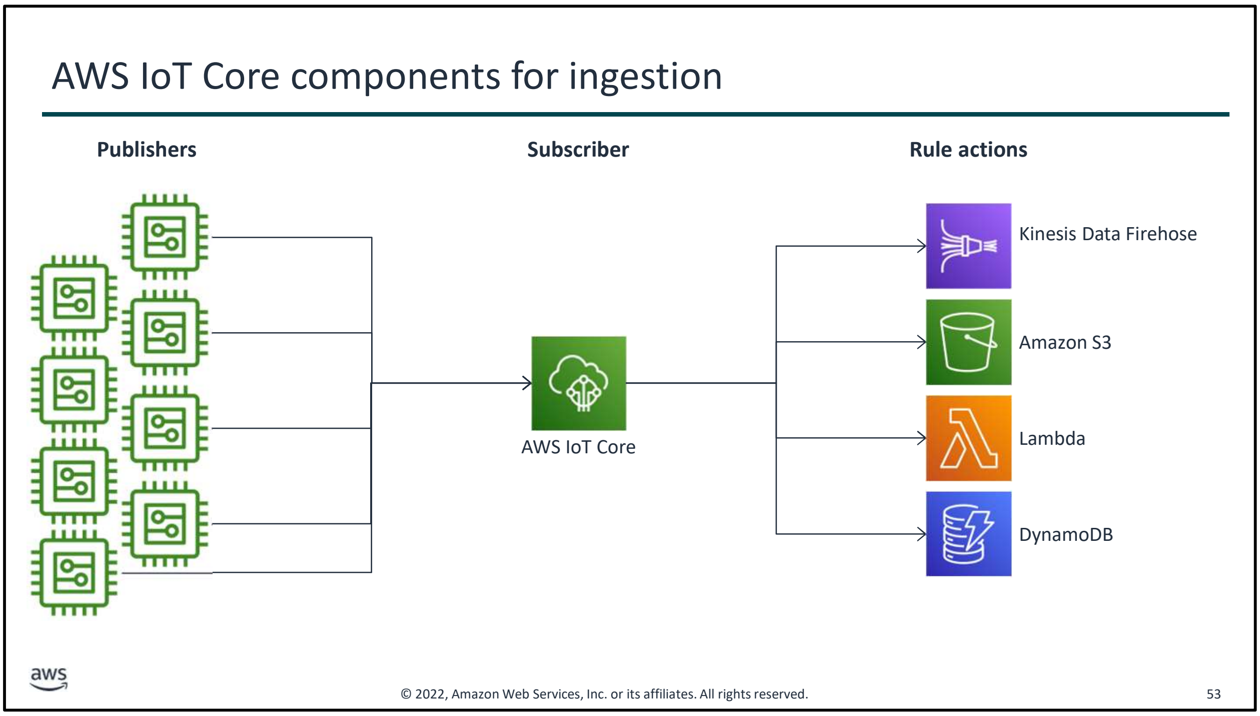
|

|Student notes

The full set of AWS services for IoT are described in the “AWS IoT Services Overview” section of the *AWS IoT Core Developer Guide*. A link is included in the course resources. The two services on this slide are especially relevant to ingesting data for analytics.

With AWS IoT Core, you can securely connect, process, and act upon device data even when devices are offline. AWS IoT Core is at the heart of IoT solutions that are built to run on AWS

The AWS IoT Analytics service simplifies the steps to build a sophisticated data pipeline for unstructured IoT data. The service has features to perform the types of data transformations that this course describes and to store time-series data that other tools in your pipeline can analyze. The next few slides look at these two AWS services in the context of ingesting IoT data.



AWS IoT Core components for ingestion

Publishers

Subscriber

Rule actions

AWS IoT Core

Kinesis Data Firehose

Amazon S3

Lambda

DynamoDB

|Slide number 53

|Instructor notes

|

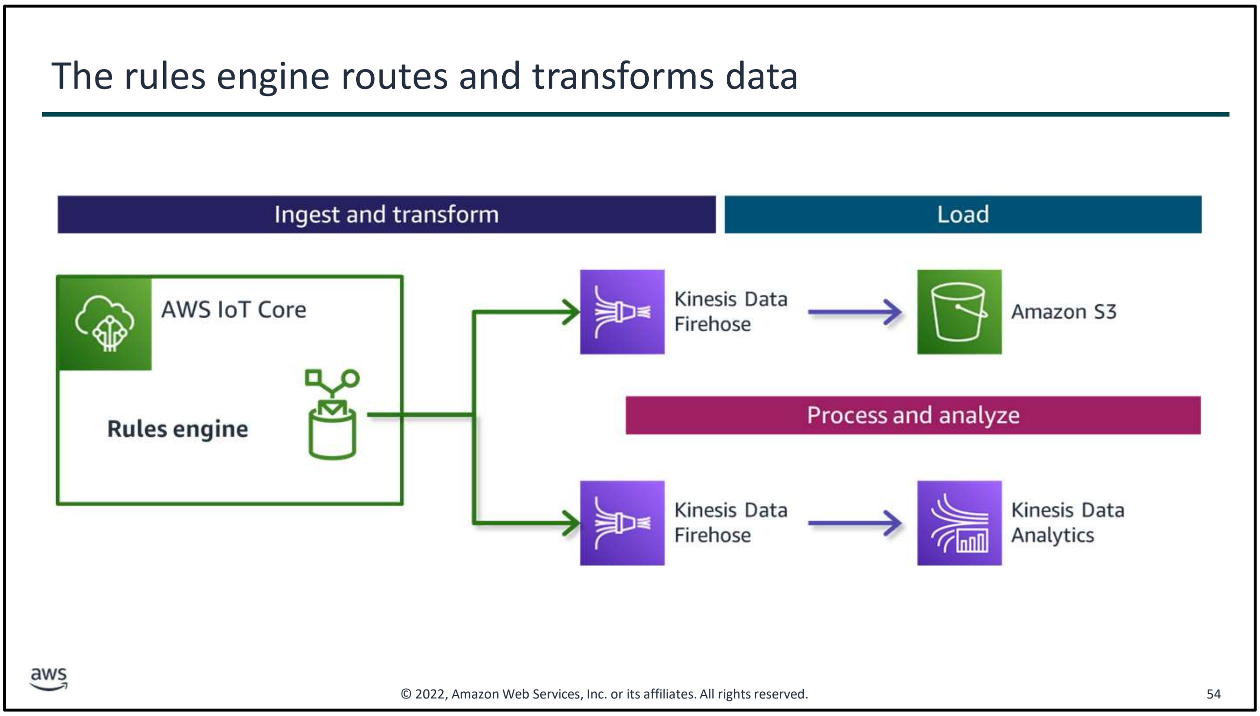
|Student notes

AWS IoT provides the cloud services that connect your IoT devices to other devices and AWS cloud services. AWS IoT Core is a subscriber to messages that IoT devices publish. The service uses rule actions to deliver messages to other AWS services.

AWS IoT Core includes the following features:

- The device gateway serves as the entry point for IoT devices to connect to AWS.
- The message broker provides a secure mechanism for devices and AWS IoT applications to publish and receive messages from each other.
- The registry is a catalog of static metadata and attributes about the devices, such as serial numbers and firmware versions.
- The rules engine uses rule actions to route messages from the message broker to other AWS services for storage and additional processing.
- The identity service provides shared responsibility for security in the AWS Cloud. Devices must keep their credentials safe to securely send data to the message broker. The message broker and rules engine use AWS security features to send data securely to devices or other AWS services.

The content resources section of your course provides links to go deeper into these and other AWS IoT Core topics. The next slide looks more closely at the rules engine.



The rules engine routes and transforms data

Ingest and transform

Load

AWS IoT Core

Rules engine

Kinesis Data Firehose

Amazon S3

Process and analyze

Kinesis Data Firehose

Kinesis Data Analytics

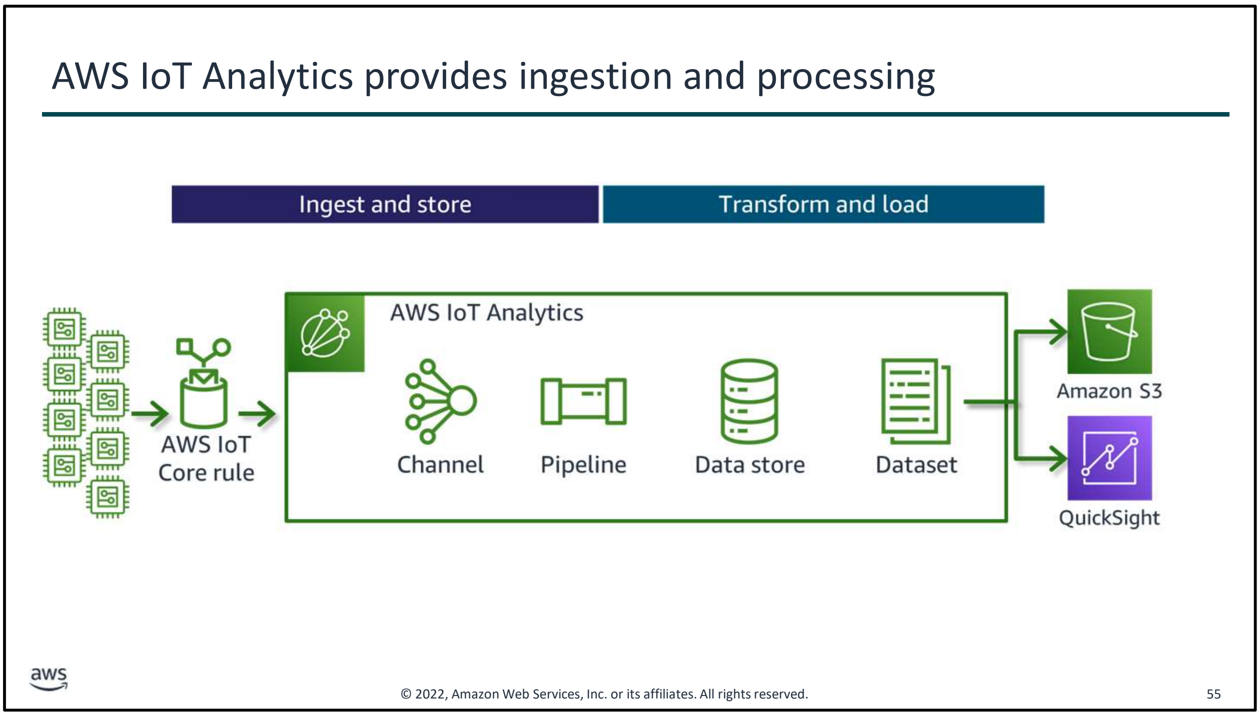
|Slide number 54
|Instructor notes
|
|Student notes

For this module's discussion of ingesting data into a pipeline, the key feature of AWS IoT Core is the rules engine. The rules engine listens for topics coming into AWS IoT Core and applies any rules that you have created for that topic. You create rules that include a name, a SQL statement that indicates the relevant topic, and one or more actions to be taken. The SQL statement provides a way to filter and transform the original messages that come into a specified topic. You can limit which message attributes to include and can include conditions about which messages to include. You can also augment messages with additional attributes.

Examples of actions are writing the selected messages to Kinesis Data Firehose or inserting data into a DynamoDB table. Although you can select to send data to Amazon S3 as a rule action, as shown in the prior slide, it wouldn’t be common to send individual messages to Amazon S3. Each message would create a new object on Amazon S3, and that would be an inefficient way to store the data.

A more common scenario for streaming IoT data to an Amazon S3 data lake would be to stream it to a Kinesis streaming service, such as Kinesis Data Firehose. Then, set up Amazon S3 as a consumer of the Kinesis Data Firehose stream. This method would be efficient for delivering a lot of messages, and you could also use Kinesis Data Firehose features to transform your data further. For example, you could run a Lambda function on the messages and convert the file format before delivering the data to Amazon S3.

Consider the streaming pipeline scenario that is described in the **Kinesis for stream processing** section of this module. The messages from AWS IoT Core are producers of data that is destined for a stream, which is then consumed by another service. In this example, Kinesis Data Firehose is configured as a rule action for a topic that AWS IoT Core is subscribed to. Using the terminology that you learned earlier about streams, AWS IoT Core is the producer, and Amazon S3 is the consumer. Similarly, for real-time analysis, Kinesis Data Analytics could be a consumer of a Kinesis Data Firehose delivery steam and could perform real-time analytics to feed real-time dashboards or other applications.



AWS IoT Analytics provides ingestion and processing

Ingest and store

Transform and load

AWS IoT Core rule

AWS IoT Analytics

Channel

Pipeline

Data store

Dataset

Amazon S3

QuickSight

|Slide number 55
|Instructor notes
|
|Student notes

AWS IoT Analytics is a fully managed service that automates the steps that are required to analyze data from IoT devices. The service filters, transforms, and enriches data before storing it in a time-series data store for analysis.

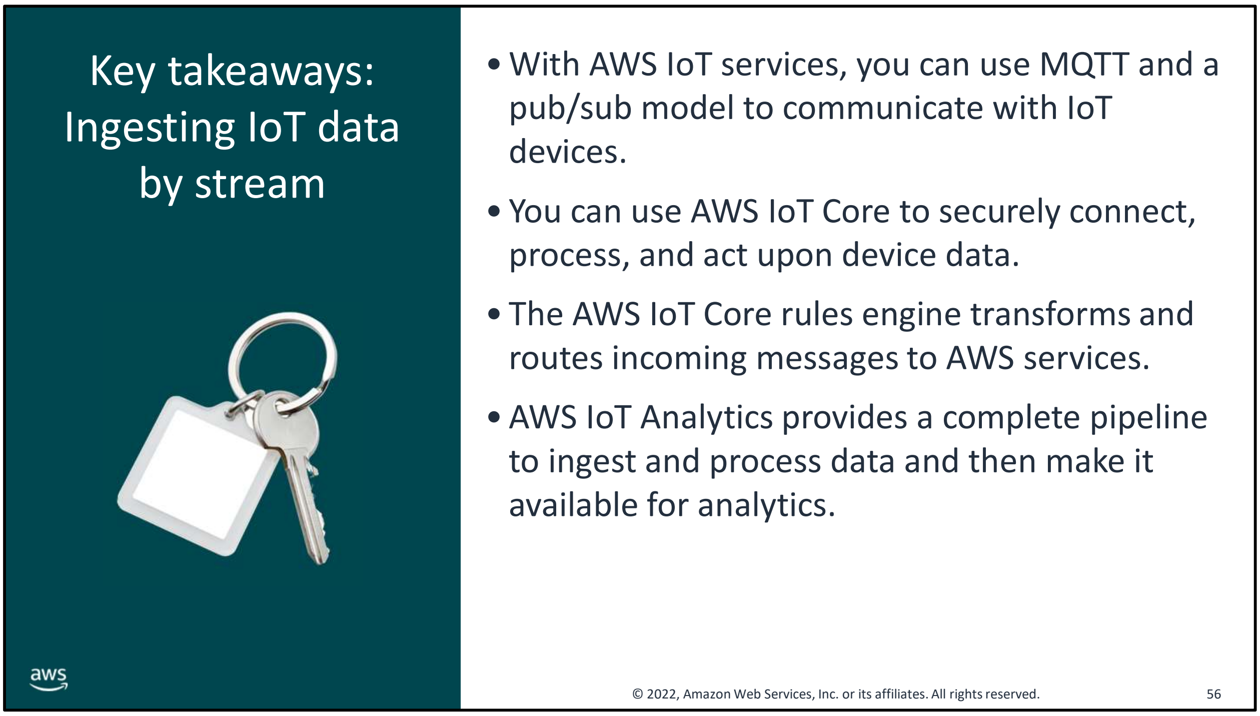
The service has multiple components. A *channel* collects data from a topic (in this example, from an AWS IoT Core rule) and archives unprocessed messages before publishing the data to a pipeline. The unprocessed messages are stored in an S3 bucket that you or AWS IoT Analytics manages.

A *pipeline* consumes messages from a channel and performs processing on the messages before storing them in a data store. The processing steps are called *activities* and can include performing transformations, such as removing, renaming, or adding message attributes; filtering messages based on attribute values; or invoking Lambda functions for advanced transformations.

Pipelines store their processed messages in a *data store*. The data store is not a database, but it is a scalable and queryable repository of messages. A data store's processed messages are stored in an S3 bucket that you or AWS IoT Analytics manages.

You retrieve data from a data store by creating a *dataset*. The dataset contains the SQL that you used to query the data store. You can send the dataset contents to an S3 bucket to integrate it with your data lake.

AWS IoT Analytics also supports direct access to your datasets from Amazon QuickSight so that you can easily visualize your dataset without additional services. The Analyzing and Visualizing Data module includes a demonstration of using AWS IoT Analytics and QuickSight.



Key takeaways: Ingesting IoT data by stream

- With AWS IoT services, you can use MQTT and a pub/sub model to communicate with IoT devices.
- You can use AWS IoT Core to securely connect, process, and act upon device data.
- The AWS IoT Core rules engine transforms and routes incoming messages to AWS services.
- AWS IoT Analytics provides a complete pipeline to ingest and process data and then make it available for analytics.

|Slide number 56
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

IoT devices use a pub/sub model to communicate with each other and with cloud-based services. In a pub/sub model, publishers send messages to a topic, and subscribers subscribe to that topic. A message broker routes published messages to the correct subscribers.

AWS provides many services to support the IoT universe, including AWS IoT Core and AWS IoT Analytics. AWS IoT Core provides features to support ingesting and transforming IoT data into your pipeline. AWS IoT Analytics provides a complete ingestion and processing pipeline, including direct access by visualization tools, such as QuickSight.

Module wrap-up

Ingesting by Batch or by Stream

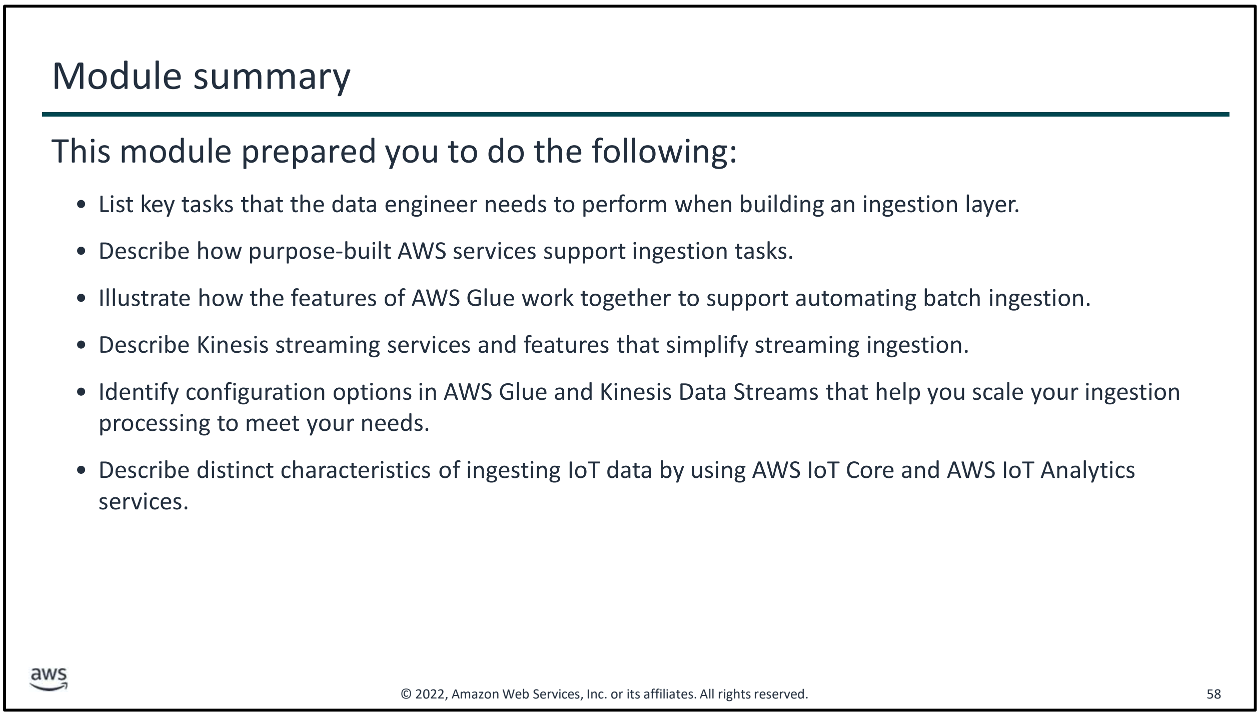
|Slide number 57

|Instructor notes

|

|Student notes

This section summarizes what you have learned and brings the Ingesting by Batch or by Stream module to a close.

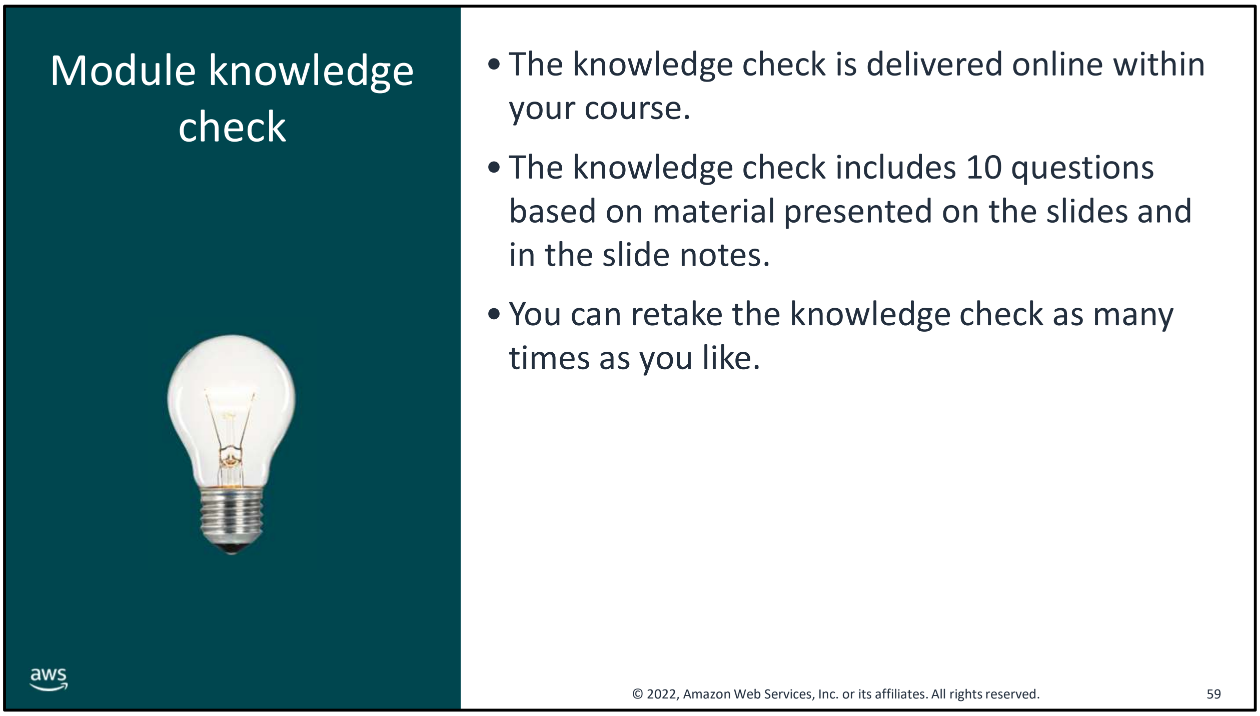


Module summary

This module prepared you to do the following:

- List key tasks that the data engineer needs to perform when building an ingestion layer.
- Describe how purpose-built AWS services support ingestion tasks.
- Illustrate how the features of AWS Glue work together to support automating batch ingestion.
- Describe Kinesis streaming services and features that simplify streaming ingestion.
- Identify configuration options in AWS Glue and Kinesis Data Streams that help you scale your ingestion processing to meet your needs.
- Describe distinct characteristics of ingesting IoT data by using AWS IoT Core and AWS IoT Analytics services.

|Slide number 58
|Instructor notes
|This is a good opportunity to use an online group or discussion board to ask students to reflect on what they have learned. You might ask the students to recall a point from the module that aligns to one of the listed objectives. This provides a good segue to the knowledge check and sample exam question.
|
|Student notes
This module focused on the processes to build an ingestion layer that supports batch or stream processing. This module reviewed AWS services that simplify performing batch ingestion tasks, including Amazon AppFlow, AWS DMS, DataSync, and AWS Data Exchange. You learned how AWS Glue can help you to automate batch ingestion, and you looked at features of Kinesis streaming products that support streaming ingestion. You also heard briefly about the options for scaling the throughput of AWS Glue jobs and Kinesis data streams. Finally, you learned how AWS IoT Core and AWS IoT Analytics can be used as part of ingesting and transforming streaming IoT data.



Module knowledge check

- The knowledge check is delivered online within your course.
- The knowledge check includes 10 questions based on material presented on the slides and in the slide notes.
- You can retake the knowledge check as many times as you like.

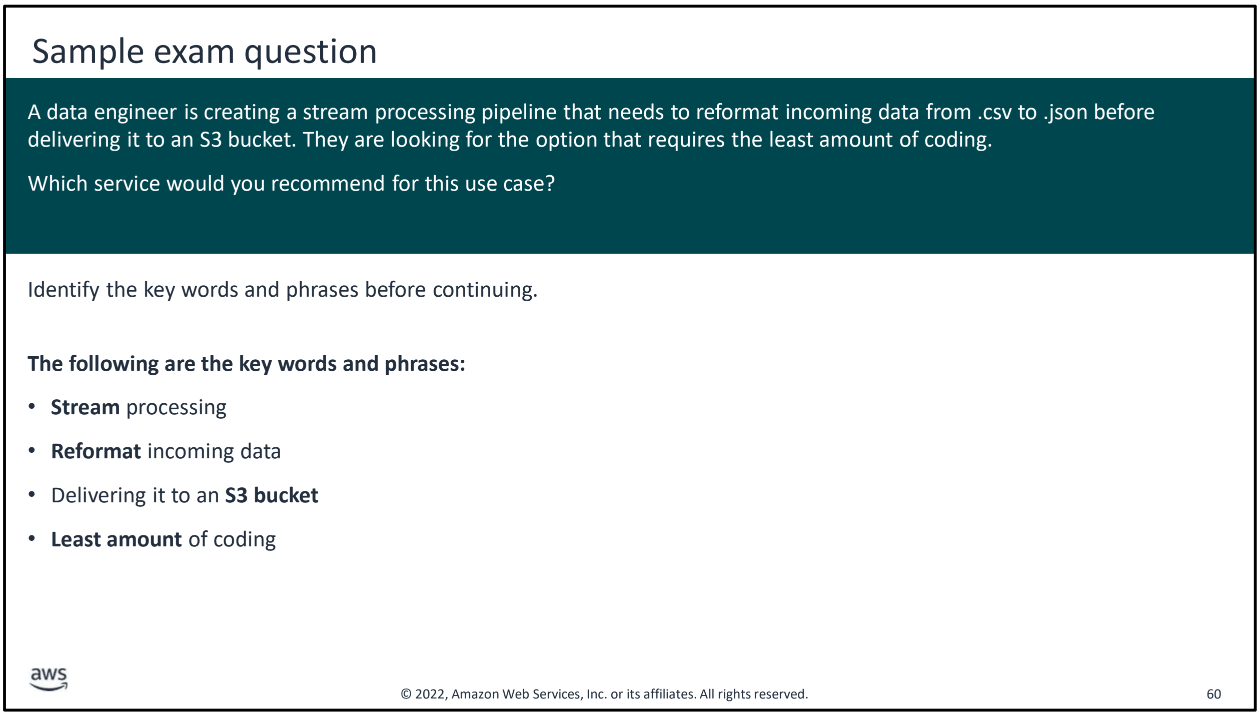
|Slide number 59

|Instructor notes

|

|Student notes

Use your online course to access the knowledge check for this module.



Sample exam question

A data engineer is creating a stream processing pipeline that needs to reformat incoming data from .csv to .json before delivering it to an S3 bucket. They are looking for the option that requires the least amount of coding.

Which service would you recommend for this use case?

Identify the key words and phrases before continuing.

The following are the key words and phrases:

- **Stream** processing
- **Reformat** incoming data
- Delivering it to an **S3 bucket**
- **Least amount** of coding

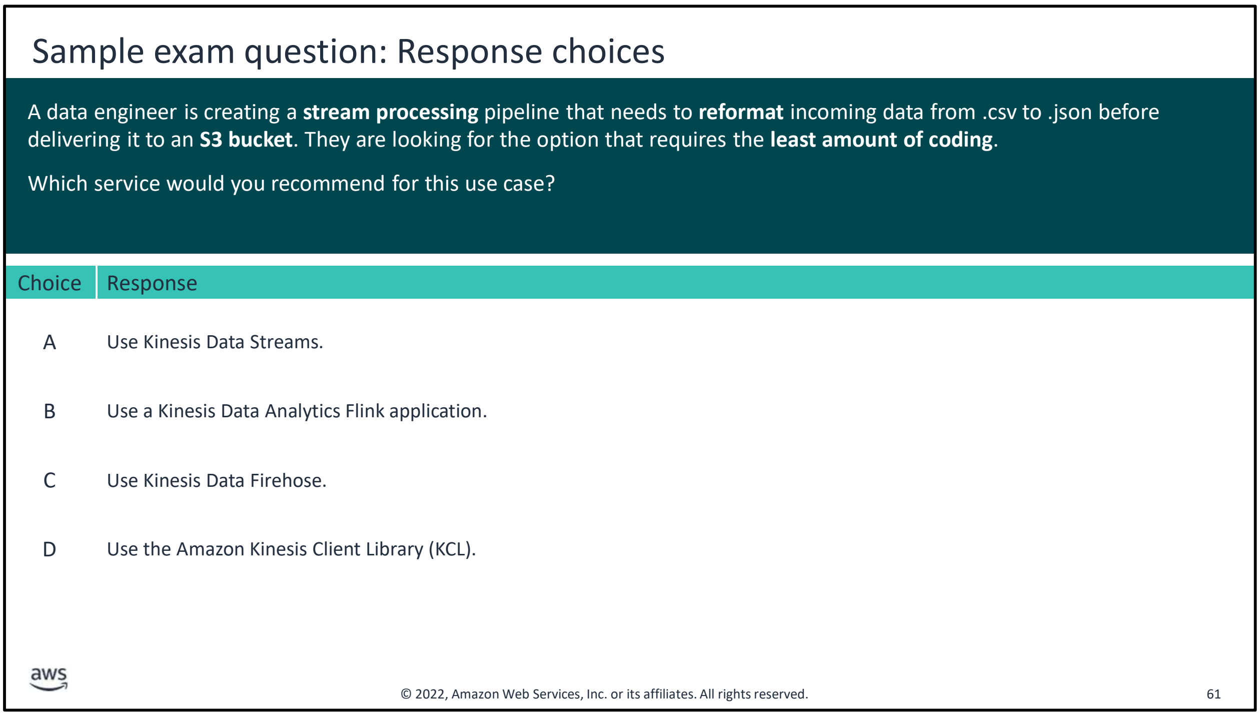
|Slide number 60

|Instructor notes: The key words section is animated to be revealed on click.

|

|Student notes

The question notes that you are building a streaming solution and that you need to reformat the data before saving it to Amazon S3. The question also notes that the data engineer is looking for the option that requires the least amount of coding.



Sample exam question: Response choices

A data engineer is creating a **stream processing** pipeline that needs to **reformat** incoming data from .csv to .json before delivering it to an **S3 bucket**. They are looking for the option that requires the **least amount of coding**.

Which service would you recommend for this use case?

Choice	Response
A	Use Kinesis Data Streams.
B	Use a Kinesis Data Analytics Flink application.
C	Use Kinesis Data Firehose.
D	Use the Amazon Kinesis Client Library (KCL).

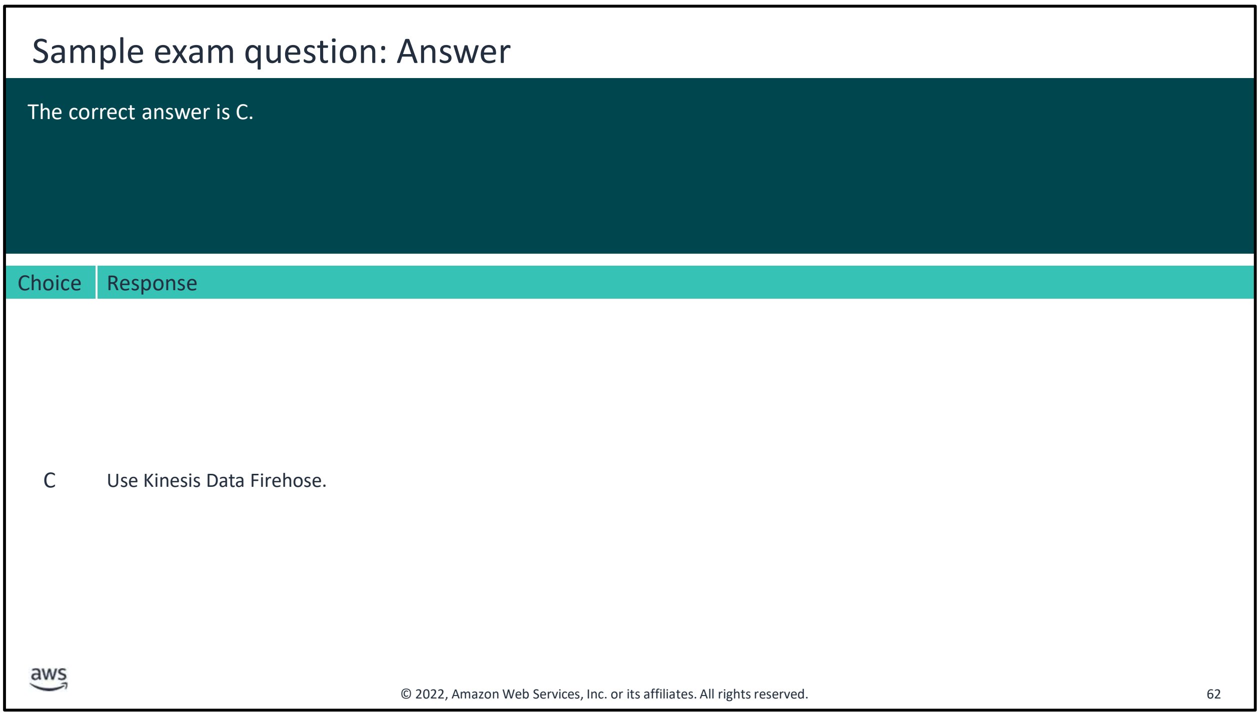
|Slide number 61

|Instructor notes

|

|Student notes

Use the key words that you identified on the previous slide, and review each of the possible responses to determine which one best addresses the question.



Sample exam question: Answer

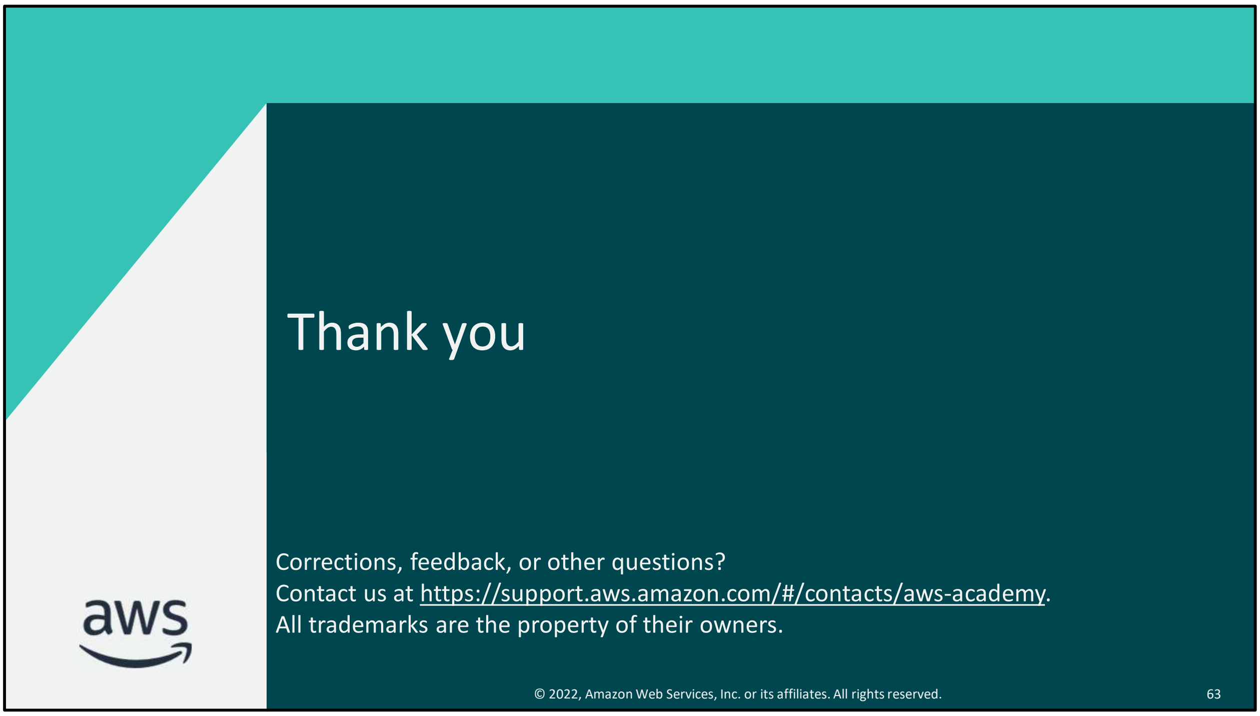
The correct answer is C.

Choice	Response
C	Use Kinesis Data Firehose.

|Slide number 62
|Instructor notes
|
|Student notes
The data engineer could build a consumer for Kinesis Data Streams to perform the transformation by using the Kinesis Client Library (KCL), but this is not a low-code option. This solution also wouldn't enable saving data directly to Amazon S3. This means neither choice A (Use Kinesis Data Streams) or D (Use the KCL) is the best option.

Kinesis Data Analytics applications can be written by using Flink. Such applications are used to perform queries and analysis on streaming data as it passes through the stream. However, this service wouldn't be the best choice for the simple format transformation and delivery to an S3 bucket, so choice B (Use a Kinesis Data Analytics Flink application) is not the best option.

Kinesis Data Firehose is designed to perform common transformations on streaming data before delivering it to destinations including Amazon S3, while requiring little or no coding. Therefore, choice C is the best option.

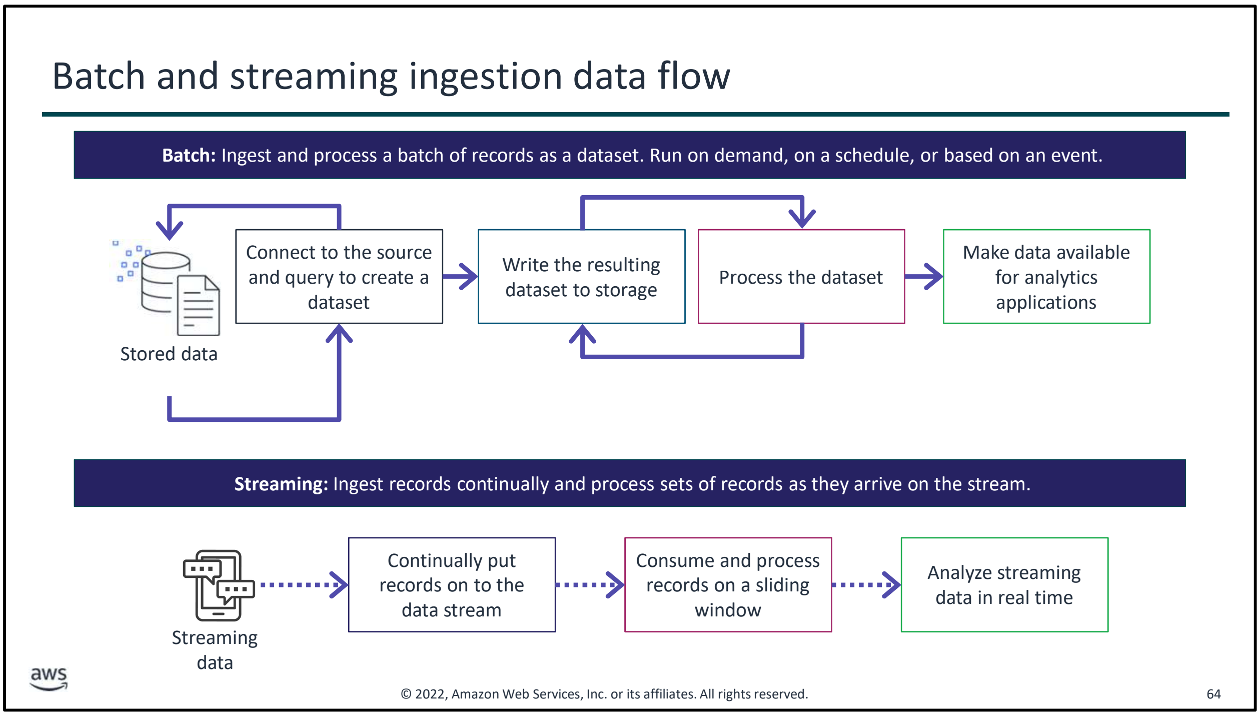


Thank you

Corrections, feedback, or other questions?
Contact us at https://support.aws.amazon.com/#/contacts/aws-academy.
All trademarks are the property of their owners.

|Slide number 63
|Instructor notes
|
|Student notes

That concludes this module. The Content Resources page of your course includes links to additional resources that are related to this module.



Batch and streaming ingestion data flow

Batch: Ingest and process a batch of records as a dataset. Run on demand, on a schedule, or based on an event.

Stored data

Connect to the source and query to create a dataset

Write the resulting dataset to storage

Process the dataset

Make data available for analytics applications

Streaming: Ingest records continually and process sets of records as they arrive on the stream.

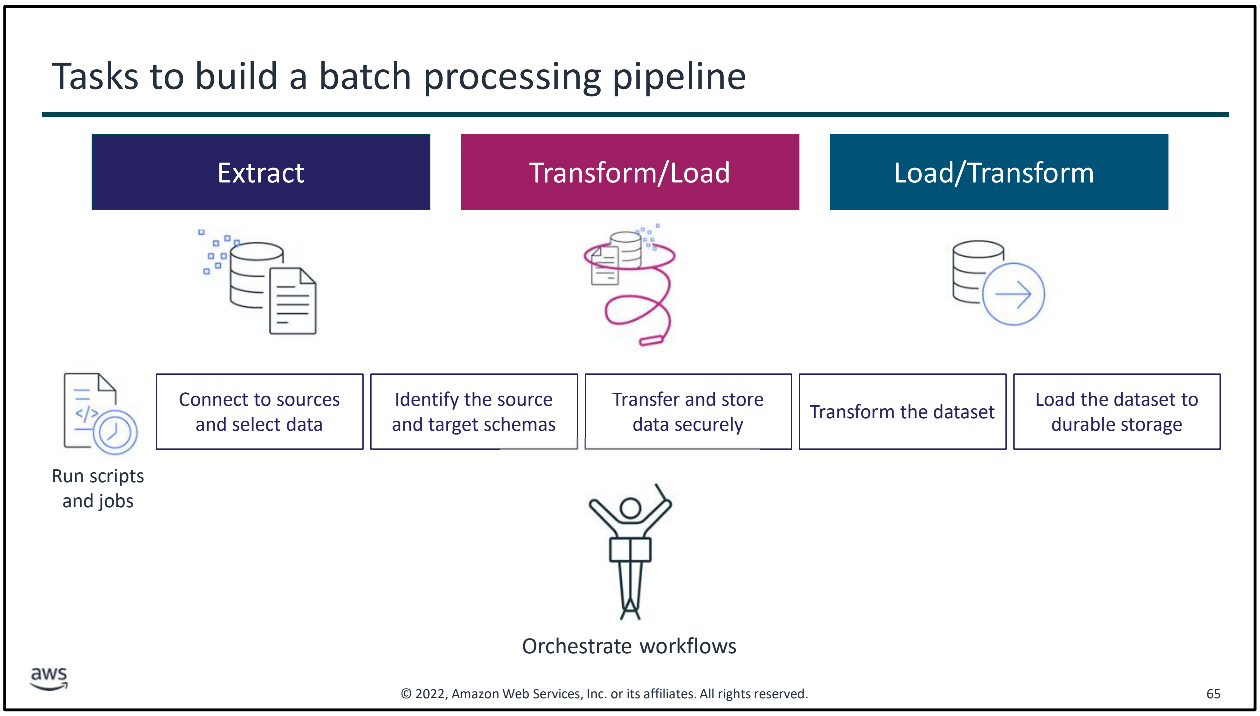
Streaming data

Continually put records on to the data stream

Consume and process records on a sliding window

Analyze streaming data in real time

Source for slide 6



Tasks to build a batch processing pipeline

Extract

Transform/Load

Load/Transform

Run scripts and jobs

Connect to sources and select data

Identify the source and target schemas

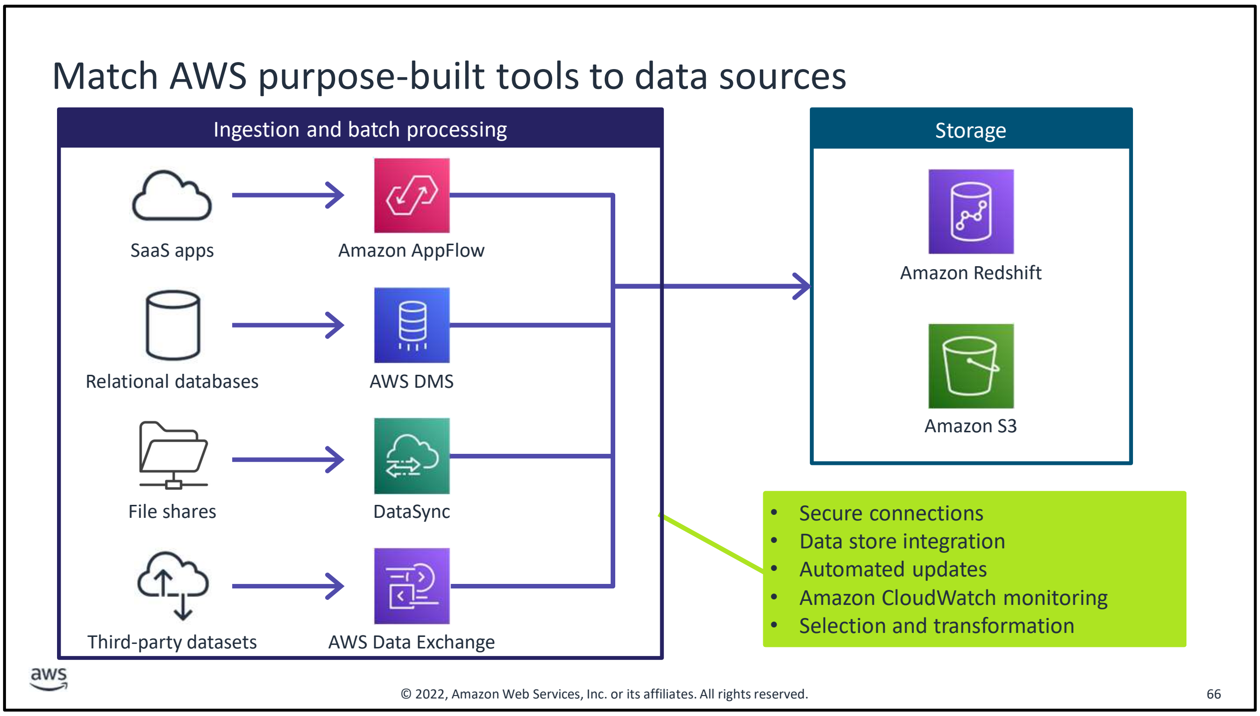
Transfer and store data securely

Transform the dataset

Load the dataset to durable storage

Orchestrate workflows

Source for slide 10



Match AWS purpose-built tools to data sources

Ingestion and batch processing

SaaS apps → Amazon AppFlow

Relational databases → AWS DMS

File shares → DataSync

Third-party datasets → AWS Data Exchange

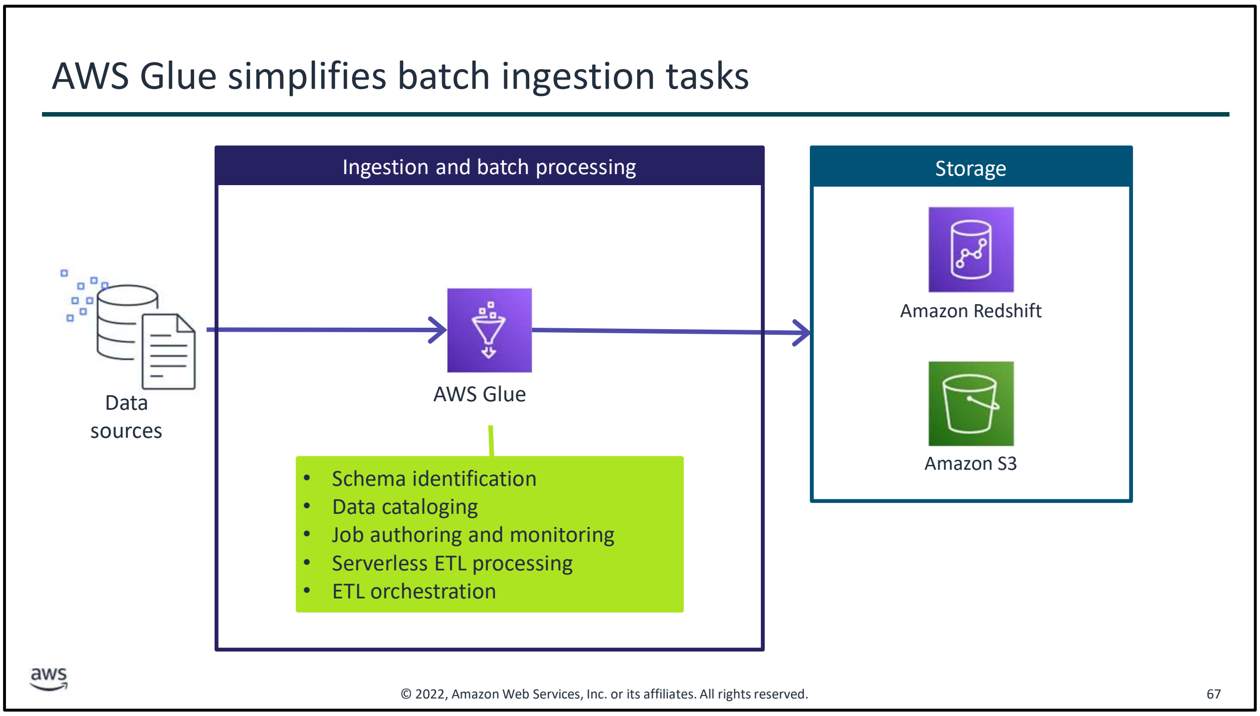
Storage

Amazon Redshift

Amazon S3

- Secure connections
- Data store integration
- Automated updates
- Amazon CloudWatch monitoring
- Selection and transformation

Source for slide 14



AWS Glue simplifies batch ingestion tasks

Data sources

Ingestion and batch processing

AWS Glue

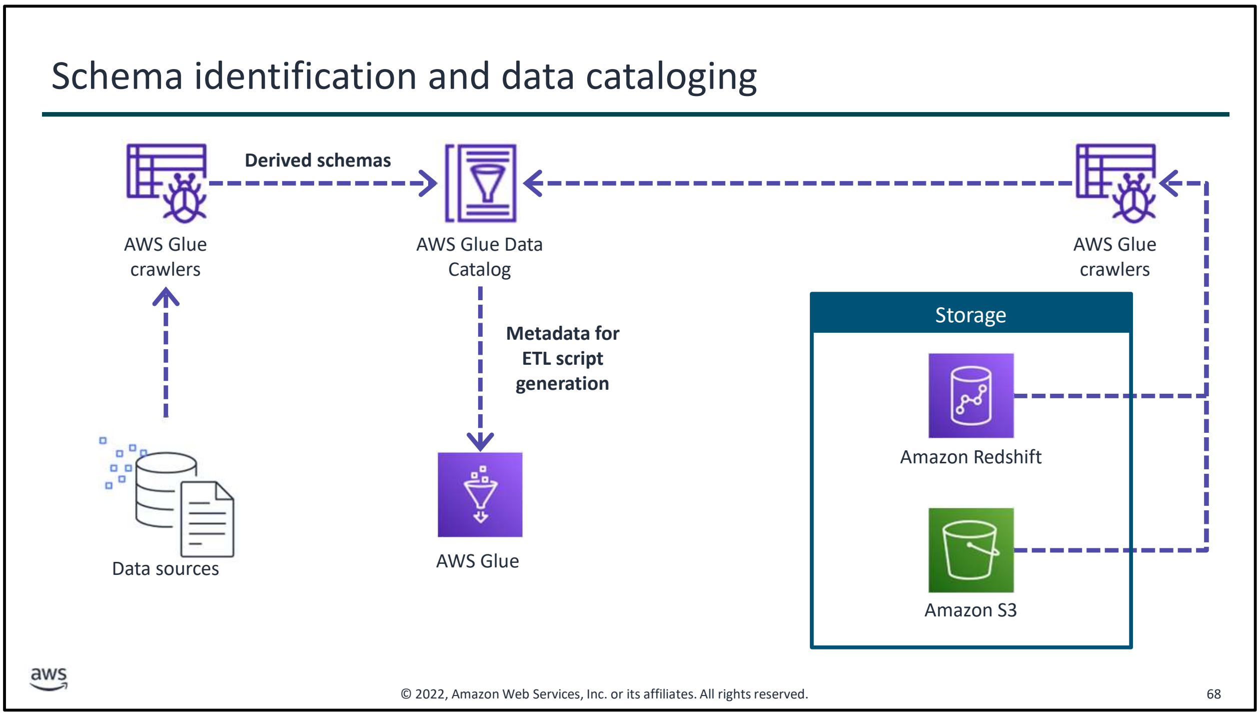
- Schema identification
- Data cataloging
- Job authoring and monitoring
- Serverless ETL processing
- ETL orchestration

Storage

Amazon Redshift

Amazon S3

Source for slide 21

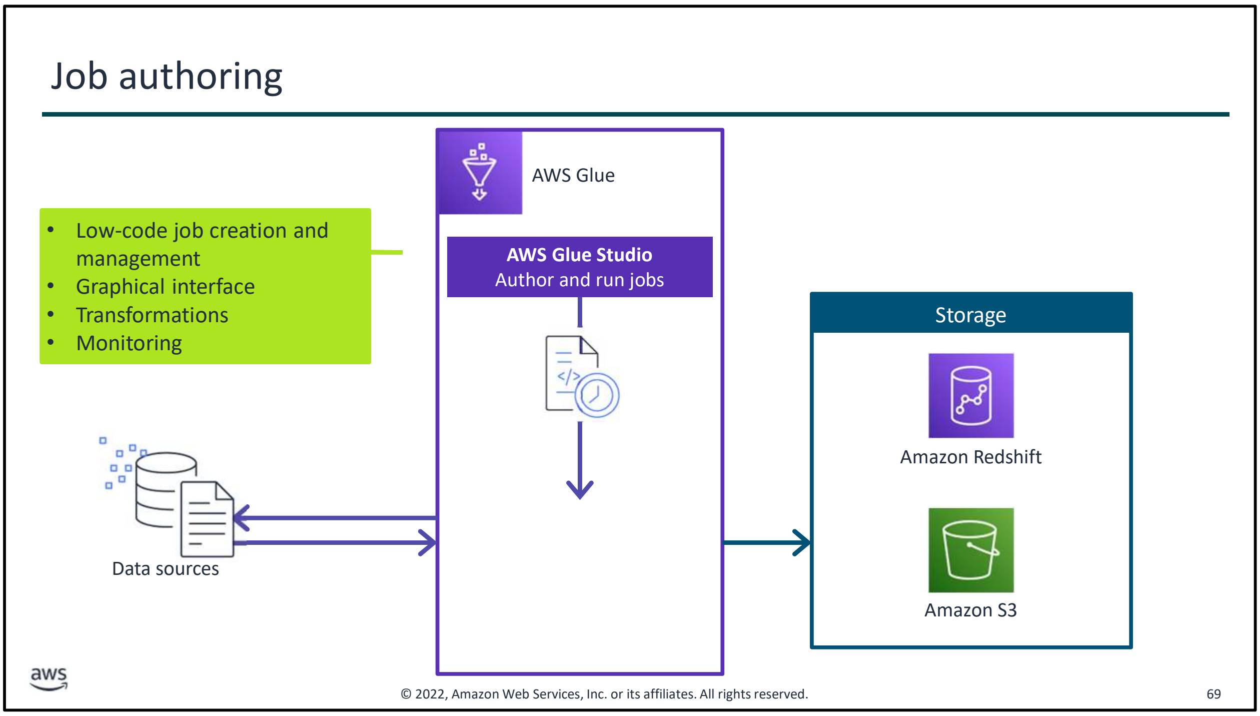


Schema identification and data cataloging

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

68

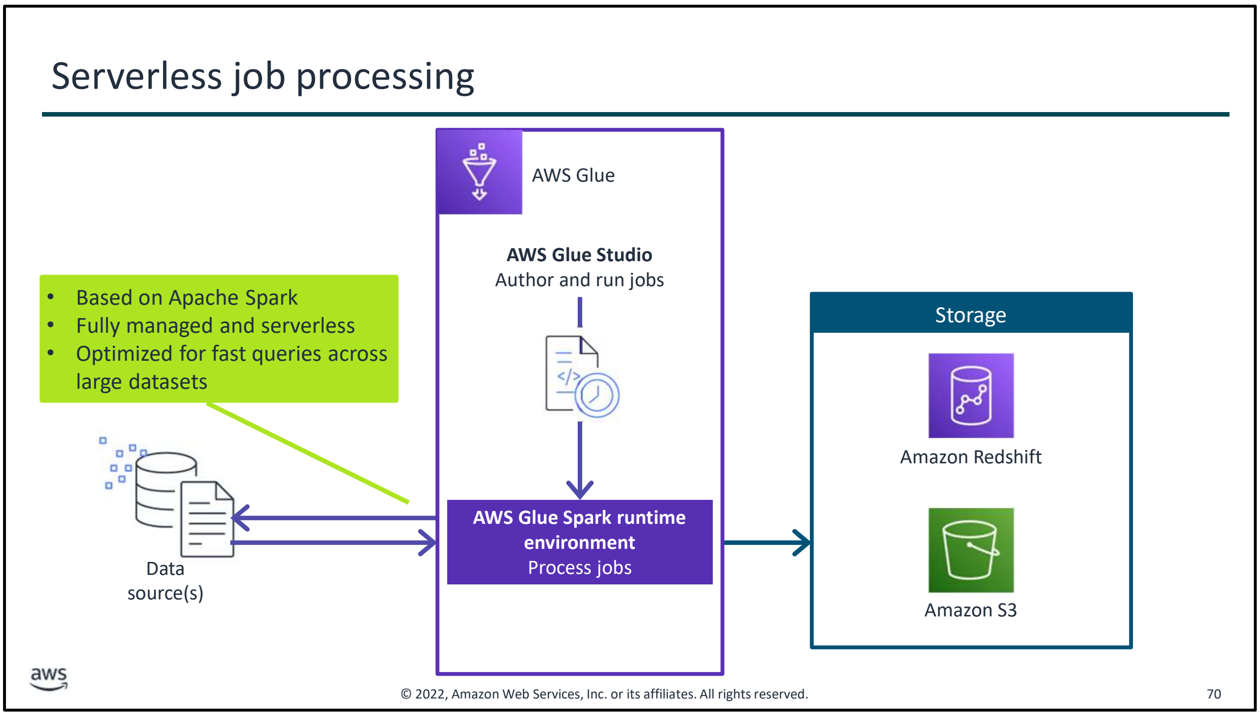
Source for slide 22



Job authoring

- Low-code job creation and management
- Graphical interface
- Transformations
- Monitoring

Source for slide 23



Serverless job processing

- Based on Apache Spark
- Fully managed and serverless
- Optimized for fast queries across large datasets

AWS Glue

AWS Glue Studio
Author and run jobs

AWS Glue Spark runtime environment
Process jobs

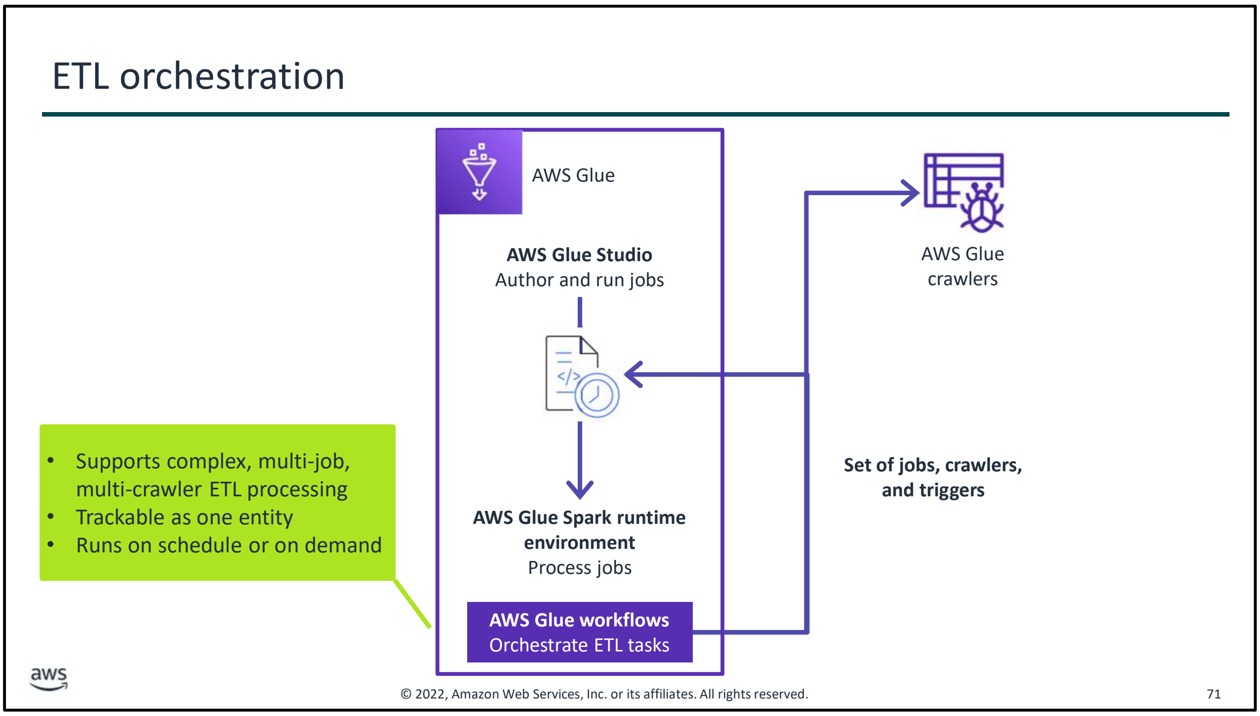
Data source(s)

Storage

Amazon Redshift

Amazon S3

Source for slide 24



ETL orchestration

- Supports complex, multi-job, multi-crawler ETL processing
- Trackable as one entity
- Runs on schedule or on demand

AWS Glue

AWS Glue Studio
Author and run jobs

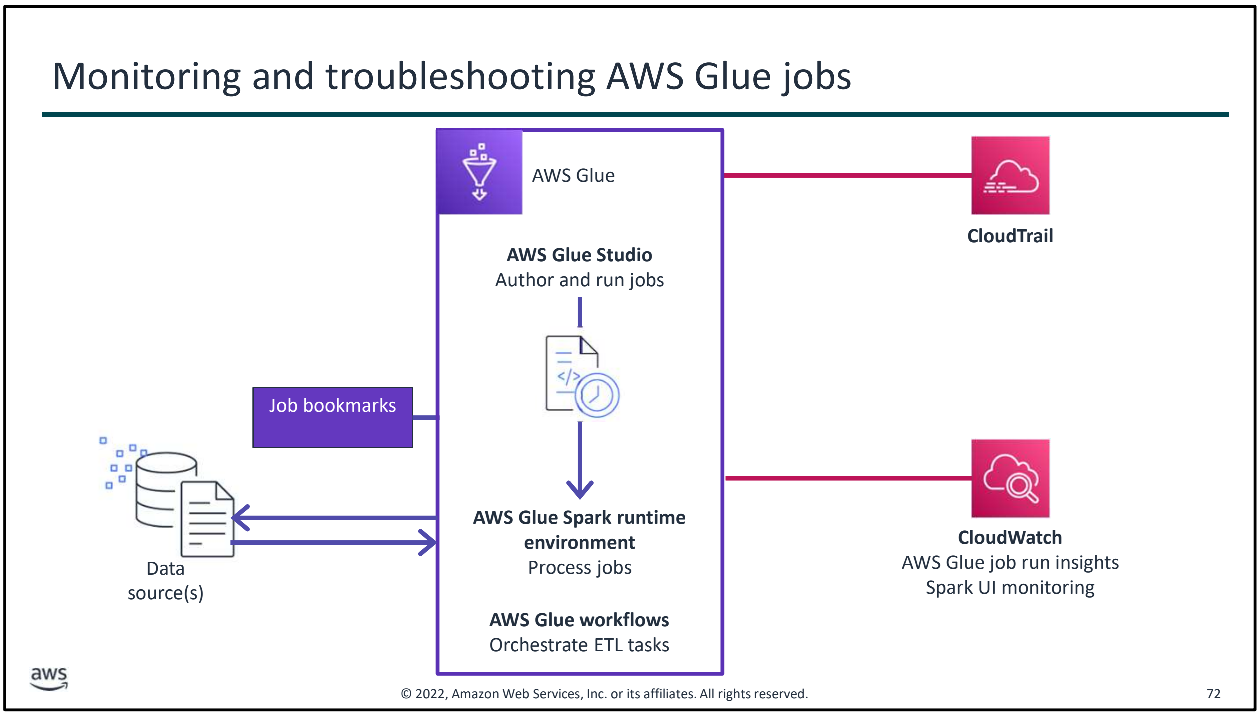
AWS Glue Spark runtime environment
Process jobs

AWS Glue workflows
Orchestrate ETL tasks

AWS Glue crawlers

Set of jobs, crawlers, and triggers

Source for slide 25



Monitoring and troubleshooting AWS Glue jobs

AWS Glue

AWS Glue Studio
Author and run jobs

AWS Glue Spark runtime environment
Process jobs

AWS Glue workflows
Orchestrate ETL tasks

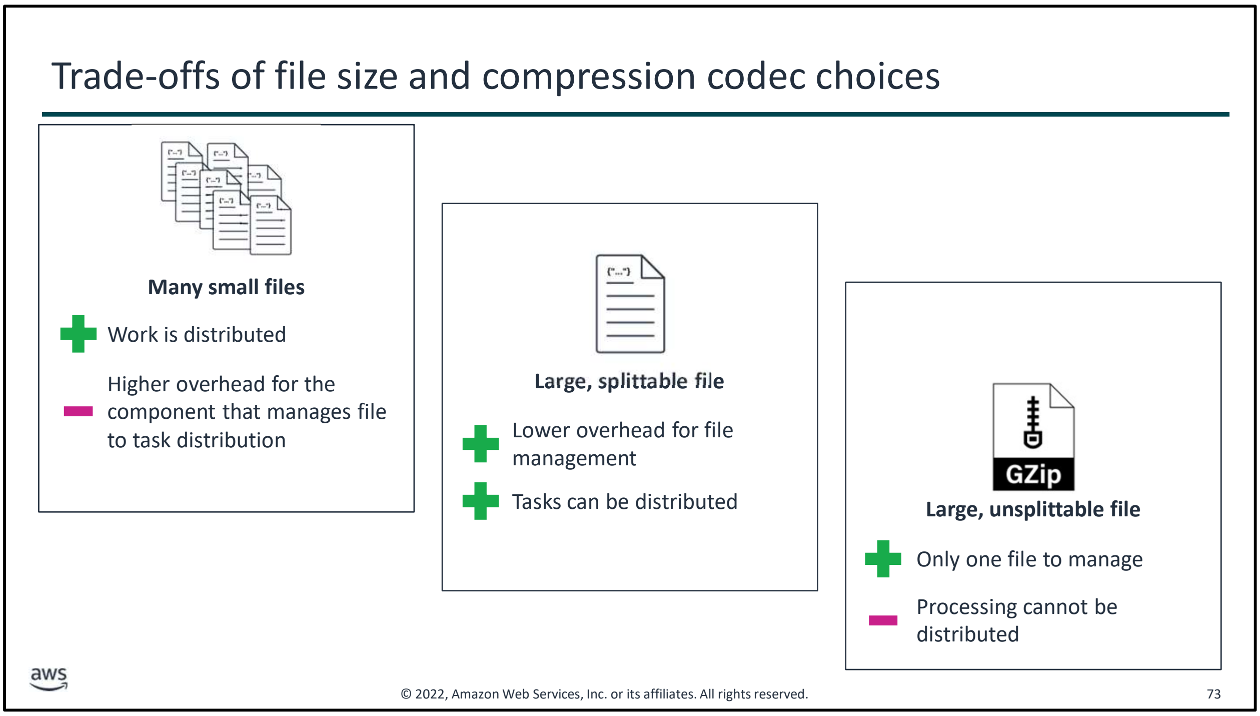
Job bookmarks

Data source(s)

CloudTrail

CloudWatch
AWS Glue job run insights
Spark UI monitoring

Source for slide 26



Trade-offs of file size and compression codec choices

Many small files

- + Work is distributed
- − Higher overhead for the component that manages file to task distribution

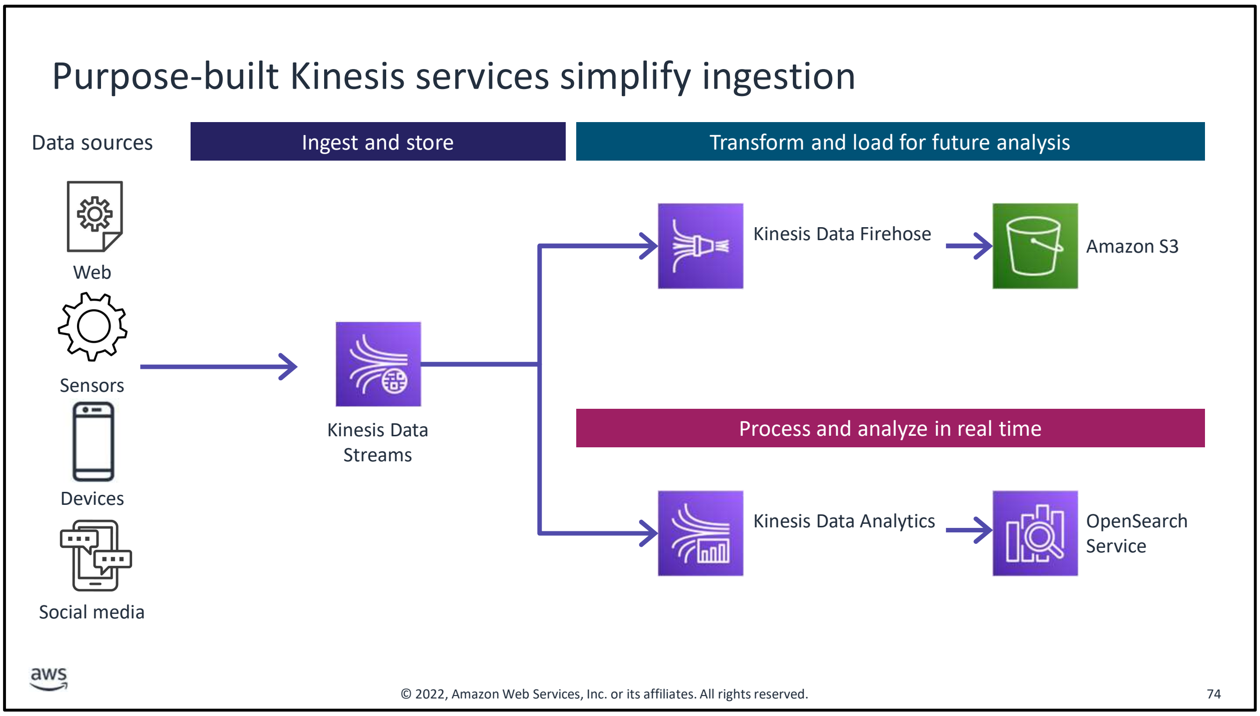
Large, splittable file

- + Lower overhead for file management
- + Tasks can be distributed

Large, unsplittable file

- + Only one file to manage
- − Processing cannot be distributed

Source for slide 31



Purpose-built Kinesis services simplify ingestion

Data sources

Ingest and store

Transform and load for future analysis

Web

Sensors

Devices

Social media

Kinesis Data Streams

Kinesis Data Firehose

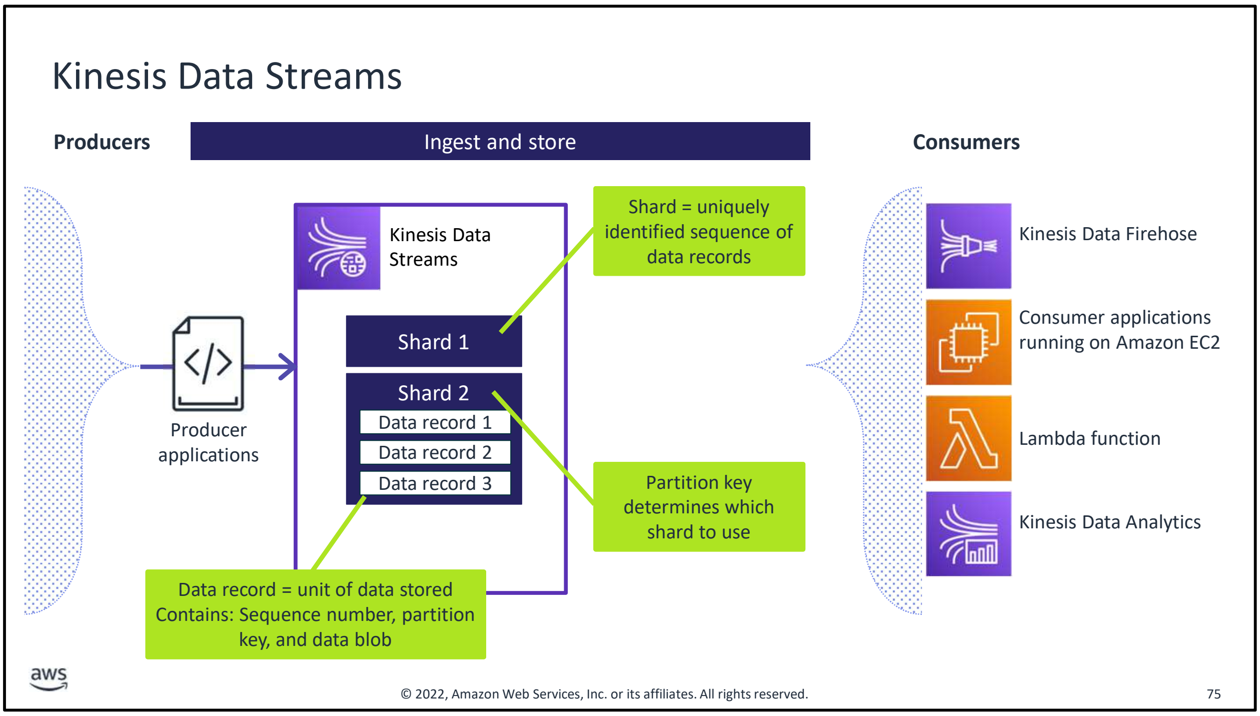
Amazon S3

Process and analyze in real time

Kinesis Data Analytics

OpenSearch Service

Source for slide 39



Kinesis Data Streams

Producers

Ingest and store

Consumers

Producer applications

Kinesis Data Streams

Shard 1

Shard 2

Data record 1

Data record 2

Data record 3

Shard = uniquely identified sequence of data records

Partition key determines which shard to use

Data record = unit of data stored
Contains: Sequence number, partition key, and data blob

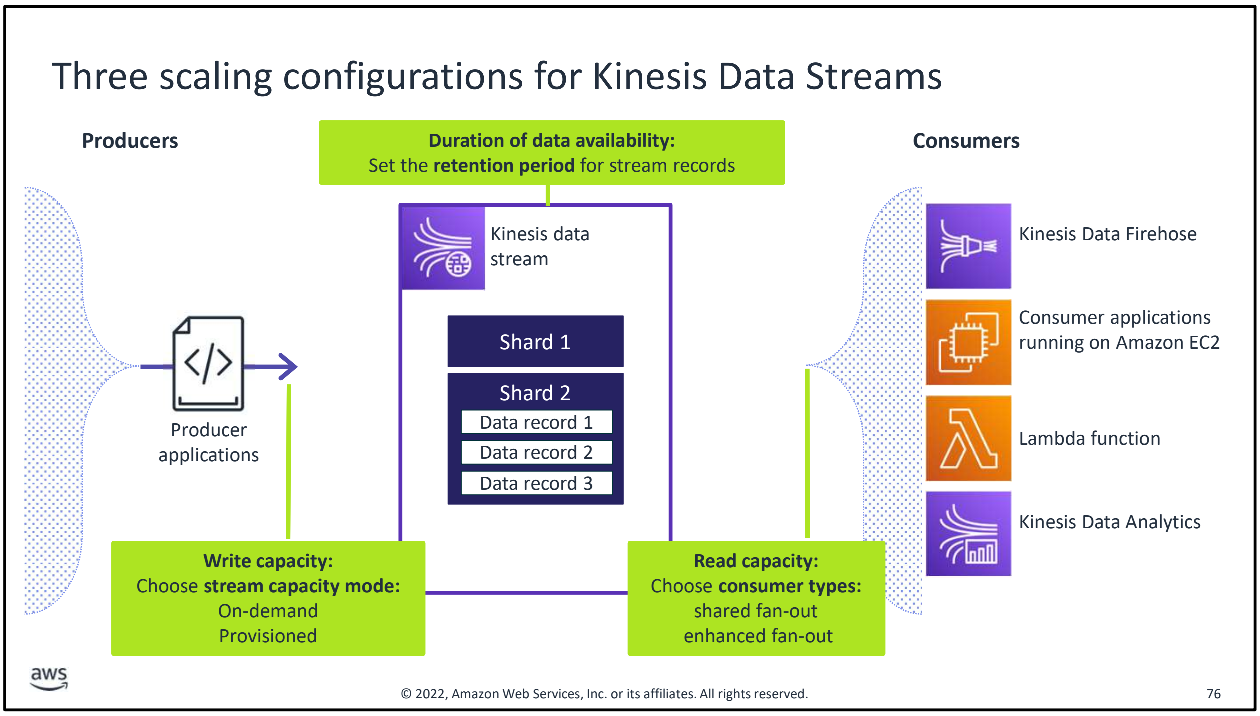
Kinesis Data Firehose

Consumer applications running on Amazon EC2

Lambda function

Kinesis Data Analytics

Source for slide 40



Three scaling configurations for Kinesis Data Streams

Producers

Duration of data availability:
Set the **retention period** for stream records

Consumers

Kinesis data stream

Shard 1

Shard 2

Data record 1

Data record 2

Data record 3

Producer applications

Kinesis Data Firehose

Consumer applications running on Amazon EC2

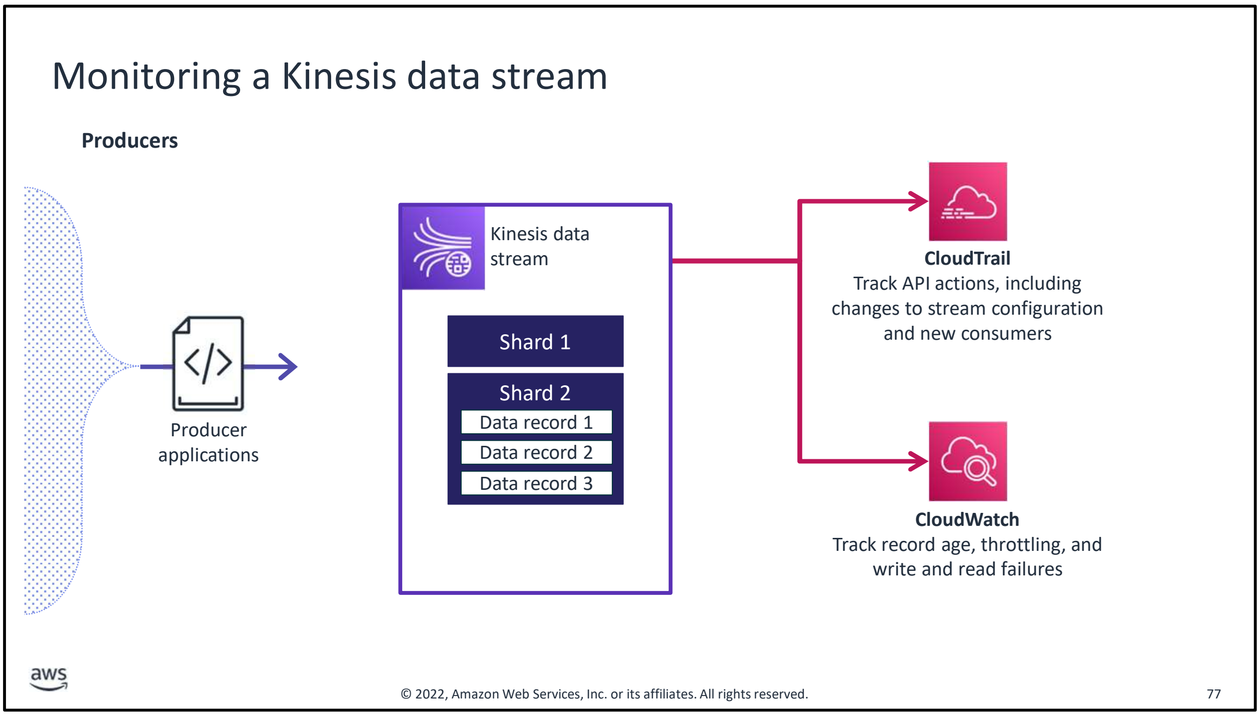
Lambda function

Kinesis Data Analytics

Write capacity:
Choose **stream capacity mode:**
On-demand
Provisioned

Read capacity:
Choose **consumer types:**
shared fan-out
enhanced fan-out

Source for slide 46



Monitoring a Kinesis data stream

Producers

Producer applications

Kinesis data stream

Shard 1

Shard 2

Data record 1

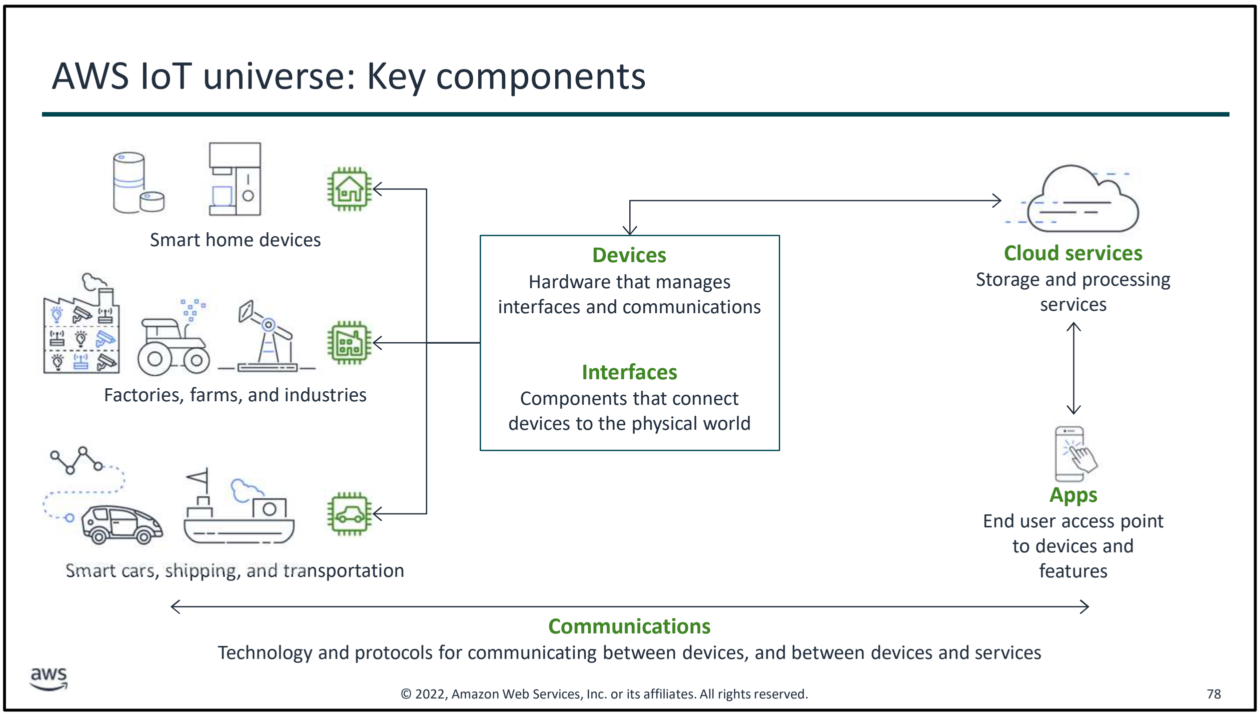
Data record 2

Data record 3

CloudTrail
Track API actions, including changes to stream configuration and new consumers

CloudWatch
Track record age, throttling, and write and read failures

Source for slide 47



AWS IoT universe: Key components

Smart home devices

Factories, farms, and industries

Smart cars, shipping, and transportation

Devices
Hardware that manages interfaces and communications

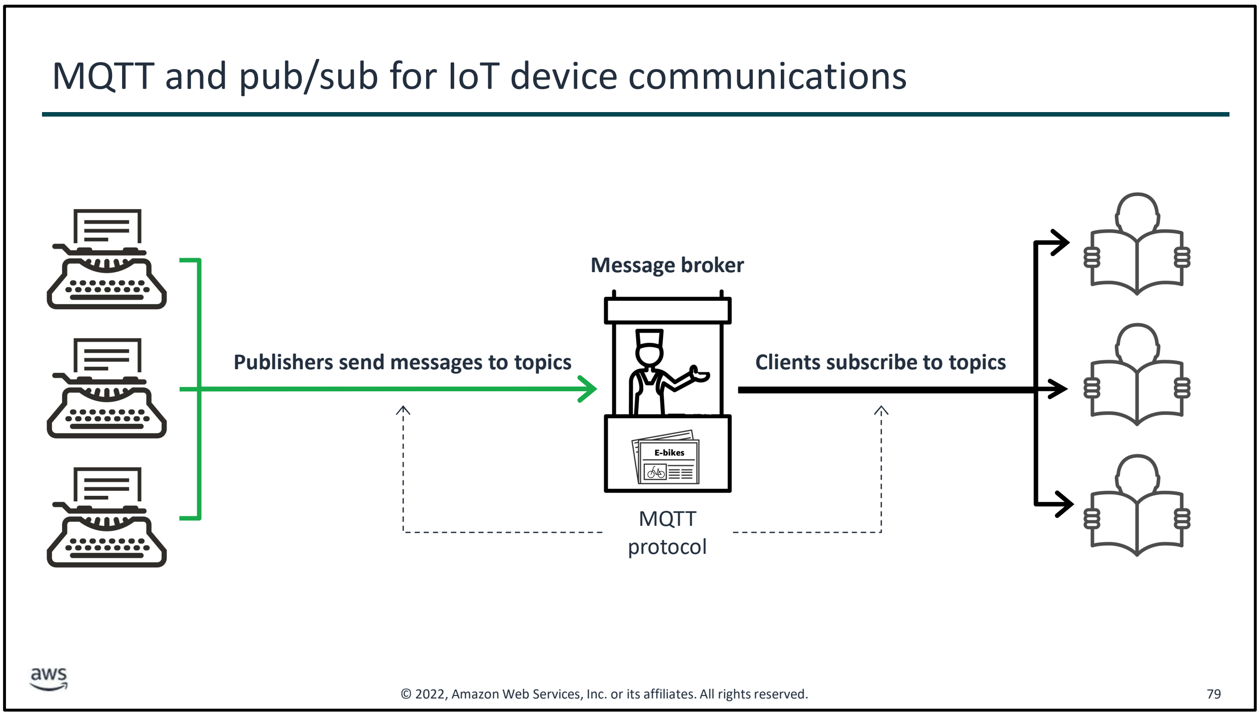
Interfaces
Components that connect devices to the physical world

Cloud services
Storage and processing services

Apps
End user access point to devices and features

Communications
Technology and protocols for communicating between devices, and between devices and services

Source for slide 50



MQTT and pub/sub for IoT device communications

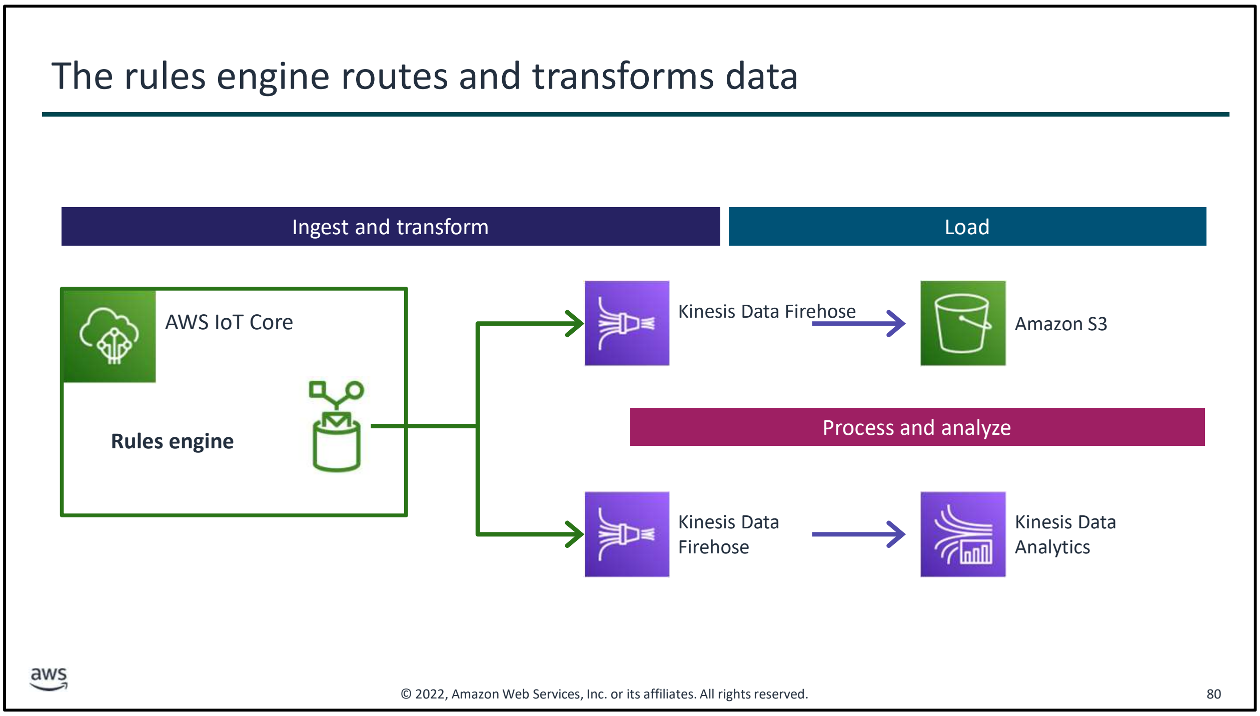
Message broker

Publishers send messages to topics

Clients subscribe to topics

MQTT protocol

Source for slide 51



The rules engine routes and transforms data

Ingest and transform

Load

AWS IoT Core

Rules engine

Kinesis Data Firehose

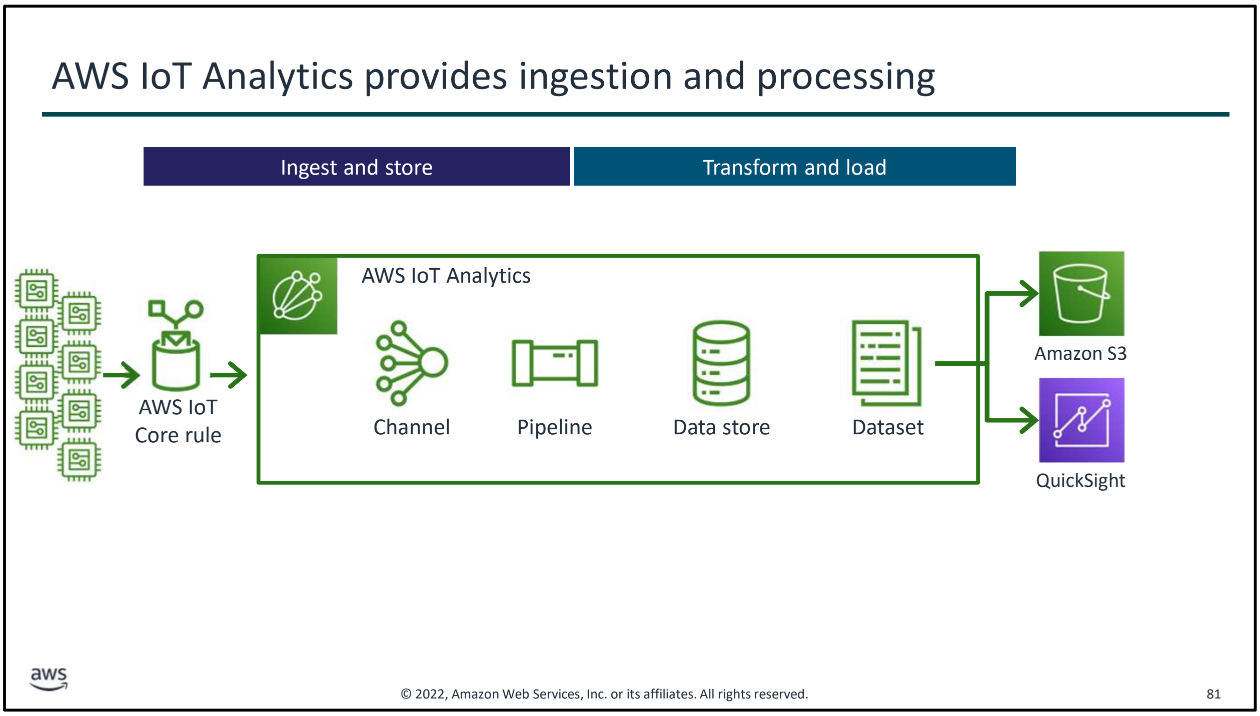
Amazon S3

Process and analyze

Kinesis Data Firehose

Kinesis Data Analytics

Source for slide 54



AWS IoT Analytics provides ingestion and processing

Ingest and store

Transform and load

AWS IoT Core rule

AWS IoT Analytics

Channel

Pipeline

Data store

Dataset

Amazon S3

QuickSight

Source for slide 55