AWS Cloud Data Ingestion Patterns and Practices

Patterns and Considerations for using AWS Services to Move Data into a Lake House Architecture

July 23, 2021

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Introduction 1
Data ingestion patterns 2
Homogeneous data ingestion patterns 4
Homogeneous relational data ingestion 4
Homogeneous data files ingestion 11
Heterogeneous data ingestion patterns 16
Heterogeneous data files ingestion 16
Streaming data ingestion 18
Relational data ingestion 27
Conclusion 34
Contributors 35
Further reading 35
Document history 36

Abstract

Today, many organizations want to gain further insight using the vast amount of data they generate or have access to. They may want to perform reporting, analytics and/or machine learning on that data and further integrate the results with other applications across the organization. More and more organizations have found it challenging to meet their needs with traditional on-premises data analytics solutions, and are looking at modernizing their data and analytics infrastructure by moving to cloud. However, before they can start analyzing the data, they need to ingest this data into cloud and use the right tool for the right job. This paper outlines the patterns, practices, and tools used by AWS customers when ingesting data into AWS Cloud using AWS services.

Introduction

As companies are dealing with massive surge in data being generated, collected and stored to support their business needs, users are expecting faster access to data to make better decisions quickly as changes occur. Such agility requires that they integrate terabytes to petabytes and sometimes exabytes of data, along with the data that was previously siloed, in order to get a complete view of their customers and business operations.

To analyze these vast amounts of data and to cater to their end user's needs, many companies create solutions like data lakes and data warehouses. They also have purpose-built data stores to cater to specific business applications and use cases—for example, relational database systems to cater to transactional systems for structured data or technologies like Elasticsearch to perform log analytics and search operations.

As customers use these data lakes and purpose-built stores, they often need to also move data between these systems. For example, moving data from the lake to purpose-built stores, from those stores to the lake, and between purpose-built stores.

At re:Invent 2020, we walked through a new modern approach to called the Lake House architecture. This architecture is shown in the following diagram.

Figure 1: Lake House architecture on AWS

As data in these data lakes and purpose-built stores continues to grow, it becomes harder to move all this data around. We call this *data gravity*.

Data movement is a critical part of any system. To design a data ingestion pipeline, it is important to understand the requirements of data ingestion and choose the appropriate approach which meets performance, latency, scale, security, and governance needs.

This whitepaper provides the patterns, practices and tools to consider in order to arrive at the most appropriate approach for data ingestion needs, with a focus on ingesting data from outside AWS to the AWS Cloud. This whitepaper is not a programming guide to handle data ingestion but is rather intended to be a guide for architects to understand the options available and provide guidance on what to consider when designing a data ingestion platform. It also guides you through various tools that are available to perform your data ingestion needs.

The whitepaper is organized into several high-level sections which highlight the common patterns in the industry. For example, *homogeneous data ingestion* is a pattern where data is moved between similar data storage systems, like Microsoft SQL Server to Microsoft SQL Server or similar formats like Parquet to Parquet. This paper further breaks down different use cases for each pattern—for example, migration from on-premises system to AWS Cloud, scaling in the cloud for read-only workloads and reporting, or performing change data capture for continuous data ingestion into the analytics workflow.

Data ingestion patterns

Organizations often want to use the cloud to optimize their data analytics and data storage solutions. These optimizations can be in the form of reducing costs, reducing the undifferentiated heavy lifting of infrastructure provisioning and management, achieving better scale and/or performance, or using innovation in the cloud.

Depending upon the current architecture and target Lake House architecture, there are certain common ingestion patterns that can be observed.

Homogeneous data ingestion patterns: These are patterns where the primary objective is to move the data into the destination in the same format or same storage engine as it is in the source. In these patterns, your primary objectives may be speed of data transfer, data protection (encryption in transit and at rest), preserving the data integrity and automating where continuous ingestion is required. These patterns usually

fall under EL piece of extract, transform, load (ETL) and can be an intermediary step before transformations are done after the ingestion.

This paper covers the following use cases for this pattern:

- Relational data ingestion between same data engines (for example, Microsoft SQL Server to Amazon RDS for SQL Server or SQL Server on Amazon EC2, or Oracle to Amazon RDS for Oracle.) This use case can apply to migrating your peripheral workload into the AWS Cloud or for scaling your workload to expand on new requirements like reporting.
- Data files ingestion from on-premises storage to an AWS Cloud data lake (for example, ingesting parquet files from Apache Hadoop to Amazon Simple Storage Service (Amazon S3) or ingesting CSV files from a file share to Amazon S3). This use case may be one time to migrate your big data solutions, or may apply to building a new data lake capability in the cloud.
- Large objects (BLOB, photos, videos) ingestion into Amazon S3 object storage.

Heterogeneous data ingestion patterns: These are patterns where data must be transformed as it is ingested into the destination data storage system. These transformations can be simple like changing the data type/format of the data to meet the destination requirement or can be as complex as running machine learning to derive new attributes in the data. This pattern is usually where data engineers and ETL developers spend most of their time to cleanse, standardize, format, and shape the data based on the business and technology requirements. As such, it follows a traditional ETL model. In this pattern, you may be integrating data from multiple sources and may have a complex step of applying transformation. The primary objectives here are same as in homogeneous data ingestion, with the added objective of meeting the business and technology requirements to transform the data.

This paper covers the following use cases for this pattern:

- Relational data ingestion between different data engines (for example, Microsoft SQL Server to Amazon Aurora relational database or Oracle to Amazon RDS for MySQL).
- Streaming data ingestion from data sources like Internet of Things (IoT) devices or log files to a central data lake or peripheral data storage.
- Relational data source to non-relational data destination and vice versa (for example, Amazon DocumentDB solution to Amazon Redshift or MySQL to Amazon DynamoDB).

- File format transformations while ingesting data files (for example, changing CSV format files on file share to Parquet on Amazon S3).

The tools that can be used in each of the preceding patterns depend upon your use case. In many cases, the same tool can be used to meet multiple use cases. Ultimately, the decision on using the right tool for the right job will depend upon your overall requirements for data ingestion in the Lake House architecture. An important aspect of your tooling will also be workflow scheduling and automation.

Homogeneous data ingestion patterns

Homogeneous relational data ingestion

To build a Lake House architecture, one of the most common scenarios is where you must move relational data from on-premises to the AWS Cloud. This section focuses on structured relational data where the source may be any relational database, such as Oracle, SQL Server, PostgreSQL, or MySQL and the target relational database engine is also the same. The target databases can be hosted either on Amazon Elastic Compute Cloud (Amazon EC2) or Amazon Relational Database Service (Amazon RDS). The following use cases apply when ingesting relational data in a homogeneous pattern:

- Migrating on-premises relational database or a portion of a database to AWS Cloud. This may be a one-time activity, with some data sync requirements till the cutover happens
- Continuous ingestion of relational data to keep a copy of the data in the cloud for reporting, analytics, or integration purposes.

Some of the most common challenges when migrating relational data from on-premises environments is the size of the data, available bandwidth of the network between on-premises and cloud, and downtime requirements. With a broad breadth of tools and options available to solve these challenges, it can be difficult for a migration team to sift through these options. This section addresses ways to overcome those challenges.

Moving from on-premises relational databases to databases hosted on Amazon EC2 instances and Amazon RDS

The volume of the data and the type of target (that is, Amazon EC2 or Amazon RDS) can affect the overall planning of data movement. Understanding the target type becomes critical as you can approach an Amazon EC2 target just like a server in-

house and use the same tools that were used while moving data from on-premises to on-premises. The exception to this scenario is thinking about network connectivity between on-premises and the AWS Cloud. Because you cannot access the operating system (OS) layer of the server hosting the Amazon RDS instance, not all data ingestion tools will work as is. Therefore, it requires careful selection of the ingestion toolset. The following section addresses these ingestion tools.

Initial bulk ingestion tools

Oracle Database as a source and a target

- **Oracle SQL Developer** is a graphical Java tool distributed free of charge by Oracle. It is best suited for smaller sized databases. Use this tool if the data size is up to 200 MB. Use the Database Copy command from the Tools menu to copy data from the source database to the the target database. The target database can be either Amazon EC2 or Amazon RDS. With any Oracle native tools, make sure to check the database version compatibility especially when moving from a higher to a lower version of the database.
- **Oracle Export and Import** utilities help you migrate databases that are smaller than 10 GB and don't include binary float and double data types. The import process creates the schema objects, so you don't have to run a script to create them beforehand. This makes the process well-suited for databases that have a large number of small tables. You can use this tool for both Amazon RDS for Oracle and Oracle databases on Amazon EC2.
- **Oracle Data Pump** is a long-term replacement for the Oracle Export/Import utilities. You can use Oracle Data Pump for any size database. Up to 20 TB of the data can be ingested using Oracle Data Pump. The target database can be either Amazon EC2 or Amazon RDS.
- AWS Database Migration Service (AWS DMS) is a managed service that helps you move data to and from AWS easily and securely. AWS DMS supports most commercial and open-source databases, and facilitates both homogeneous and heterogeneous migrations. AWS DMS offers both one-time full database copy and change data capture (CDC) technology to keep the source and target databases in sync and to minimize downtime during a migration. There is no limit to the source database size. For more information how to create a DMS task, see Creating a task.

- **Oracle GoldenGate** is a tool for replicating data between a source database and one or more destination databases with minimal downtime. You can use it to build high availability architectures, and to perform real-time data integration, transactional change data capture (CDC), replication in heterogeneous environments, and continuous data replication. There is no limit to the source database size.
- **Oracle Data Guard** provides a set of services for creating, maintaining, monitoring, and managing Oracle standby databases. You can migrate your entire Oracle database from on-premises to Amazon EC2 with minimal downtime by using Oracle Recovery Manager (RMAN) and Oracle Data Guard. With RMAN, you restore your database to the target standby database on Amazon EC2, using either backup/restore or the duplicate database method. You then configure the target database as a physical standby database with Oracle Data Guard, allowing all the transaction/redo data changes from the primary on-premises database to the standby database. There is no limit to the source database size.
- **AWS Application Migration Service** (AWS MGN)– is a highly automated lift-and-shift (rehost) solution that simplifies, expedites, and reduces the cost of migrating applications to AWS. It enables companies to lift-and-shift a large number of physical, virtual, or cloud servers without compatibility issues, performance disruption, or long cutover windows. MGN replicates source servers into your AWS account. When you're ready, it automatically converts and launches your servers on AWS so you can quickly benefit from the cost savings, productivity, resilience, and agility of the Cloud. Once your applications are running on AWS, you can leverage AWS services and capabilities to quickly and easily replatform or refactor those applications – which makes lift-and-shift a fast route to modernization.

  For more prescriptive guidance, see Migrating Oracle databases to the AWS Cloud.

Microsoft SQL Server Database as a source and a target

- **Native SQL Server backup/restore -** Using native backup (*.bak) files is the simplest way to back up and restore SQL Server databases. You can use this method to migrate databases to or from Amazon RDS. You can back up and restore single databases instead of entire database (DB) instances. You can also move databases between Amazon RDS for SQL Server DB instances. When you use Amazon RDS, you can store and transfer backup files in Amazon Simple Storage Service (Amazon S3), for an added layer of protection for disaster recovery. You can use this process to back up and restore SQL Server databases to Amazon EC2 as well (see the following diagram).

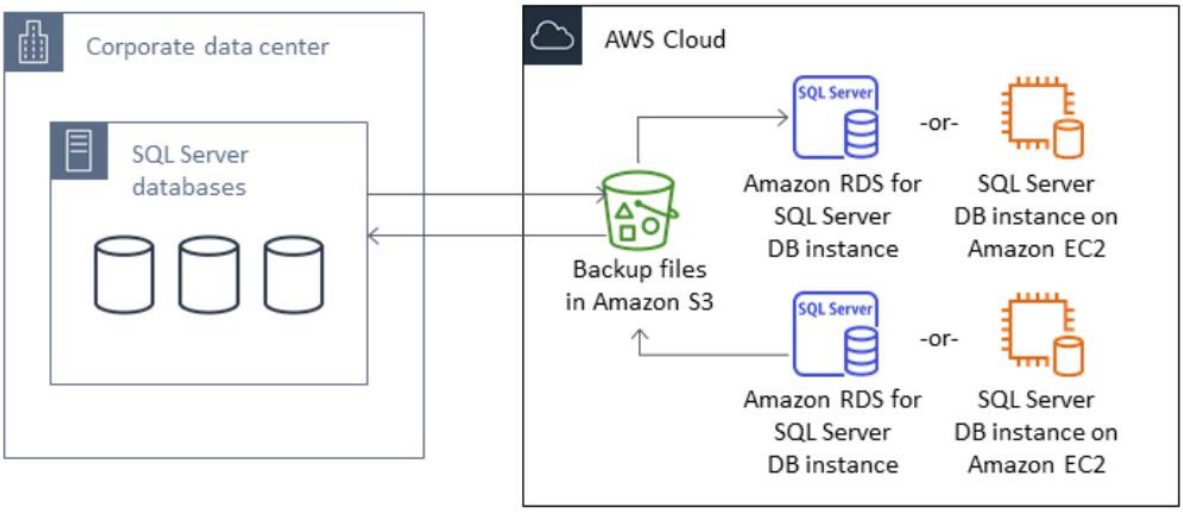

Figure 2: Microsoft SQL Server to Amazon RDS or Amazon EC2

- **Database Mirroring -** You can use database mirroring to set up a hybrid cloud environment for your SQL Server databases. This option requires SQL Server Enterprise edition. In this scenario, your principal SQL Server database runs on-premises, and you create a warm standby solution in the cloud. You replicate your data asynchronously, and perform a manual failover when you're ready for cutover.
- **Always On availability groups -** SQL Server Always On availability groups is an advanced, enterprise-level feature to provide high availability and disaster recovery solutions. This feature is available if you are using SQL Server 2012 and later versions. You can also use an Always On availability group to migrate your on-premises SQL Server databases to Amazon EC2 on AWS. This approach enables you to migrate your databases either with downtime or with minimal downtime.

- **Distributed availability groups -** A distributed availability group spans two separate availability groups. You can think of it as an availability group of availability groups. The underlying availability groups are configured on two different Windows Server Fail over Clustering (WSFC) clusters. The availability groups that participate in a distributed availability group do not need to share the same location. They can be physical or virtual systems. The availability groups in a distributed availability group don't have to run the same version of SQL Server. The target database (DB) instance can run a later version of SQL Server than the source DB instance.
- **AWS Snowball Edge** - You can use AWS Snowball Edge to migrate very large databases (up to 80 TB in size). Any type of source relational database can use this method when the database size is very large. Snowball has a 10 Gb Ethernet port that you plug into your on-premises server and place all database backups or data on the Snowball device. After the data is copied to Snowball, you send the appliance to AWS for placement in your designated S3 bucket. Data copied to Snowball Edge is automatically encrypted. You can then download the backups from Amazon S3 and restore them on SQL Server on an Amazon EC2 instance, or run the `rds_restore_database` stored procedure to restore the database to Amazon RDS. You can also use AWS Snowcone for databases up to 8 TB in size.
- **AWS DMS** – Apart from the above tools, AWS Database Migration Service (AWS DMS) can be used to migrate initial bulk and/or change data capture as well.

For more prescriptive guidance, see [Homogeneous database migration for SQL Server.]

PostgreSQL Database as a source and a target

- **Native PostgreSQL Tools** - `pg_dump`/`pg_restore` - database migration tools can be used under the following conditions:
    - You have a homogeneous migration, where you are migrating from a database with the same database engine as the target database.
    - You are migrating an entire database.
    - The native tools allow you to migrate your system with minimal downtime.

The `pg_dump` utility uses the COPY command to create a schema and data dump of a PostgreSQL database. The dump script generated by `pg_dump` loads data into a database with the same name and recreates the tables, indexes, and foreign keys. You can use the `pg_restore` command and the `-d` parameter to restore the data to a database with a different name.

- AWS DMS – If native database toolsets cannot be used, performing a database migration using AWS Database Migration Service (AWS DMS) is the best approach. AWS DMS can migrate databases without downtime and, for many database engines, continue ongoing replication until you are ready to switch over to the target database.

For more prescriptive guidance, see Homogeneous database migration for PostgreSQL Server.

MySQL/MariaDB Database as a source and a target

- **Native MySQL tools** - `mysqldbcopy` and `mysqldump` **-** You can also import data from an existing MySQL or MariaDB database to a MySQL or MariaDB DB instance. You do so by copying the database with `mysqldump` and piping it directly into the MySQL or MariaDB DB instance. The `mysqldump` command-line utility is commonly used to make backups and transfer data from one MySQL or MariaDB server to another. This utility is included with MySQL and MariaDB client software.
- **AWS DMS** – You can use AWS DMS as well for the initial ingestion.

Change Data Capture (CDC) tools

To read ongoing changes from the source database, AWS DMS uses engine-specific API actions to read changes from the source engine's transaction logs. The following examples provide more details on how AWS DMS does these reads:

- For Oracle, AWS DMS uses either the Oracle LogMiner API or binary reader API (`bfile` API) to read ongoing changes. AWS DMS reads ongoing changes from the online or archive redo logs based on the system change number (SCN).
- For Microsoft SQL Server, AWS DMS uses MS-Replication or MS-CDC to write information to the SQL Server transaction log. It then uses the `fn_dblog()` or `fn_dump_dblog()` function in SQL Server to read the changes in the transaction log based on the log sequence number (LSN).

- For MySQL, AWS DMS reads changes from the row-based binary logs (binlogs) and migrates those changes to the target.
- For PostgreSQL, AWS DMS sets up logical replication slots and uses the `test_decoding` plugin to read changes from the source and migrate them to the target.
- For Amazon RDS as a source, we recommend ensuring that backups are enabled to set up CDC. We also recommend ensuring that the source database is configured to retain change logs for a sufficient time—24 hours is usually enough.

For more prescriptive guidance, see Replication using DMS

Read-only workloads strategy for a database hosted on an Amazon EC2 instance and Amazon RDS

Every commercial vendor provides native tool sets to achieve read-only standby databases. Common examples include Oracle Active Data guard, Oracle GoldenGate, MS SQL Server Always on availability groups, Hot Standby for PostgreSQL and MySQL. Refer to vendor specific documentation if you are interested in pursuing read-only native scenarios on an Amazon EC2 instance.

Amazon RDS Read Replicas provide enhanced performance and durability for Amazon RDS database (DB) instances. They make it easy to elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads. You can create one or more replicas of a given source DB Instance and serve high-volume application read traffic from multiple copies of your data, thereby increasing aggregate read throughput. Read replicas can also be promoted when needed to become standalone DB instances. Read replicas are available in Amazon RDS for MySQL, MariaDB, PostgreSQL, Oracle, and MS SQL Server as well as for Amazon Aurora.

For the MySQL, MariaDB, PostgreSQL, Oracle, and MS SQL Server database engines, Amazon RDS creates a second DB instance using a snapshot of the source DB instance. It then uses the engines' native asynchronous replication to update the read replica whenever there is a change to the source DB instance. The read replica operates as a DB instance that allows only read-only connections; applications can connect to a read replica just as they would to any DB instance. Amazon RDS replicates all databases in the source DB instance. Check vendor specific licensing before using the read replicas feature in Amazon RDS.

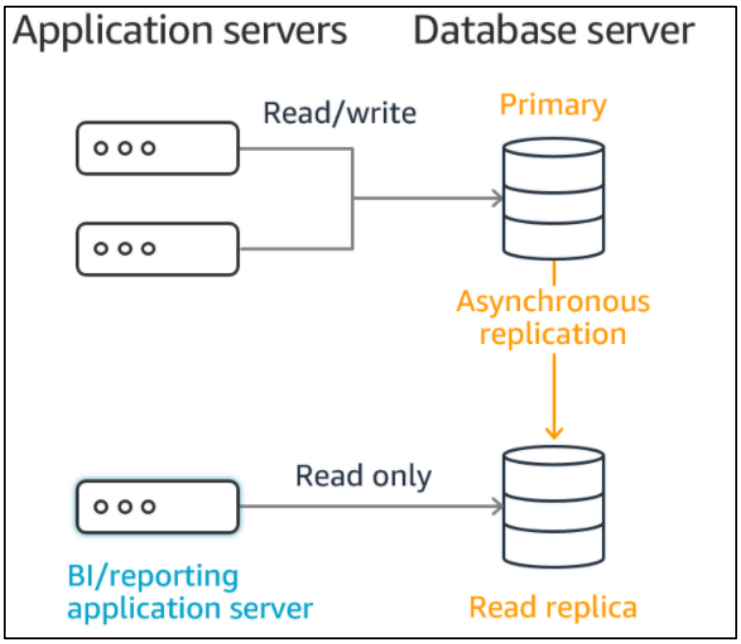

Figure 3: Common OLTP and AWS Read Replica Scenario

Homogeneous data files ingestion

Many organizations use structured or semi-structured files formats to store data. These formats include structured files like CSV, Flat Files, Parquet, and Avro, or semi-structured files like JSON. These formats are also common in the Lake House architecture and produced as outputs from an existing application or database storage, from processing in big data systems, or from data transfer solutions. Data engineers and developers are already familiar with these formats and use them in building systems for data migration solutions. When data is loaded into the data lake from various source systems, known as an *outside-in* approach, it may initially be stored as-is, before it is transformed into a standardized format into the data lake. Or, the files may be transformed into the preferred format when loaded into the data lake. For movement of the data within the Lake House architecture, known as *inside-out* use cases, the data lake processing engine outputs data for loading into purpose-built databases for scenarios like machine learning, reporting, business intelligence, and more. In *peripheral* data movement scenarios, where connection between data sources cannot be established, the data in extracted in common structured formats, placed in S3 storage, and used as intermediary for ETL operation.

Depending upon the use case, they could be classified as homogeneous or heterogeneous data ingestion. When they are ingested into data lake in as-is format, the classification is homogeneous. While if the format is changed or data destination is

anything other than file or object storage, it is classified as heterogeneous ingestion. Data ingestions are usually completed as batch processes and may or may not involve any transformations. Further data ingestion mechanisms may also depend upon the slice of data received (e.g delta, full copy) (.

This section covers the pattern where the format of the files is not changed from its source, nor is any transformation applied when the files are being ingested. This scenario is a fairly common outside-in data movement pattern. This pattern is used for populating a so-called landing area in the data lake where all of the original copies of the ingested data are kept.

If the ingestion must be done only one time, which could be the use case for migration of an on-prem big-data or analytics systems to AWS or a one-off bulk ingestion job, then depending upon the size of the data, the data can be ingested over the wire or using the Snow family of devices.

These options are used when no transformations are required while ingesting the data.

If you determine that the bandwidth you have over the wire is sufficient and data size manageable to meet the SLAs, then following set of tools can be considered for one time or continuous ingestion of files. Note that you need to have the right connectivity between your data center and AWS setup using options like AWS Direct Connect. For all available connectivity options, see the Amazon Virtual Private Cloud Connectivity Options whitepaper.

The following tools can cater to both one-time or continuous file ingestion needs, with some tools catering to other uses cases around backup or disaster recovery (DR) as well.

Integrating on-premises data processing platforms

AWS Storage Gateway can be used to integrate on-premises data processing platforms with an Amazon S3-based data lake. The File Gateway configuration of Storage Gateway offers on-premises devices and applications a network file share via a Network File System (NFS) connection. Files written to this mount point are converted to objects stored in Amazon S3 in their original format without any proprietary modification. This means that you can easily integrate applications and platforms that don’t have native Amazon S3 capabilities—such as on-premises lab equipment, mainframe computers, databases, and data warehouses—with S3 buckets, and then use tools such as Amazon EMR or Amazon Athena to process this data.

File Gateway can be deployed either as a virtual or hardware appliance. It enables for you to present a NFS such as Server Message Block (SMB) in your on-premises environment, which interfaces with Amazon S3. In Amazon S3, File Gateway preserves file metadata, such as permissions and timestamps for objects it stores in Amazon S3.

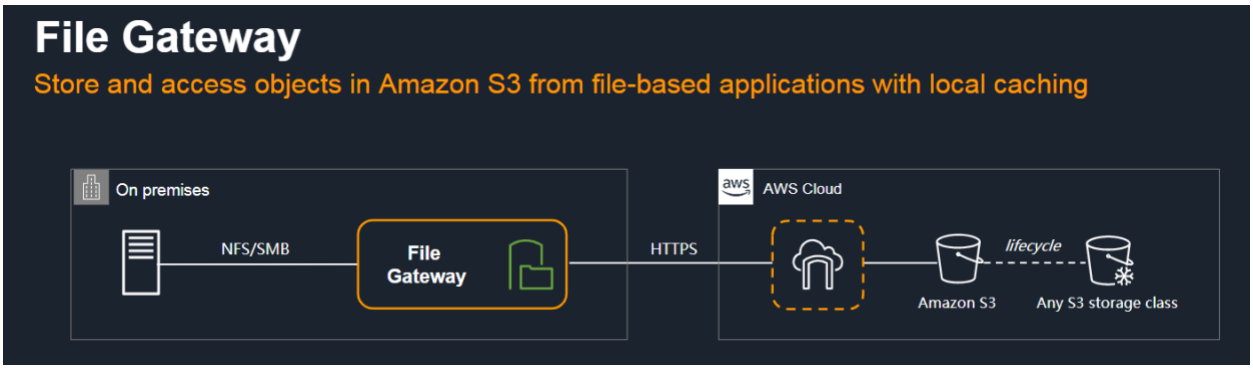

Figure 4: File Gateway architecture illustration

Data synchronization between on-premises data platforms and AWS

AWS DataSync is an online data transfer service that simplifies, automates, and accelerates moving data between on-premises storage systems and AWS storage services, as well as between AWS storage services. You can use DataSync to transfer data to the cloud for analysis and processing.

For details on how AWS DataSync can be used to transfer files from on-premises to AWS securely, see AWS re:Invent recap: Quick and secure data migrations using AWS DataSync. This blog post also references the re:invent session which details the aspects around end-to-end validation, in-flight encryption, scheduling, filtering, and more.

DataSync provides built-in security capabilities such as encryption of data in-transit, and data integrity verification in-transit and at-rest. It optimizes use of network bandwidth, and automatically recovers from network connectivity failures. In addition, DataSync provides control and monitoring capabilities such as data transfer scheduling and granular visibility into the transfer process through Amazon CloudWatch metrics, logs, and events.

DataSync can copy data between Network File System (NFS) shares, Server Message Block (SMB) shares, self-managed object storage and Amazon Simple Storage Service (Amazon S3) buckets.

It's simple to use. You deploy the DataSync agent as a virtual appliance in a network that has access to AWS, where you define your source, target, and transfer options per transfer task. It allows for simplified data transfers from your SMB and NFS file shares, and self-managed object storage, directly to any of the Amazon S3 storage classes. It also supports Amazon Elastic File System (Amazon EFS) and Amazon FSx for Windows File Server for data movement of file data, where it can preserve the file and folder attributes.

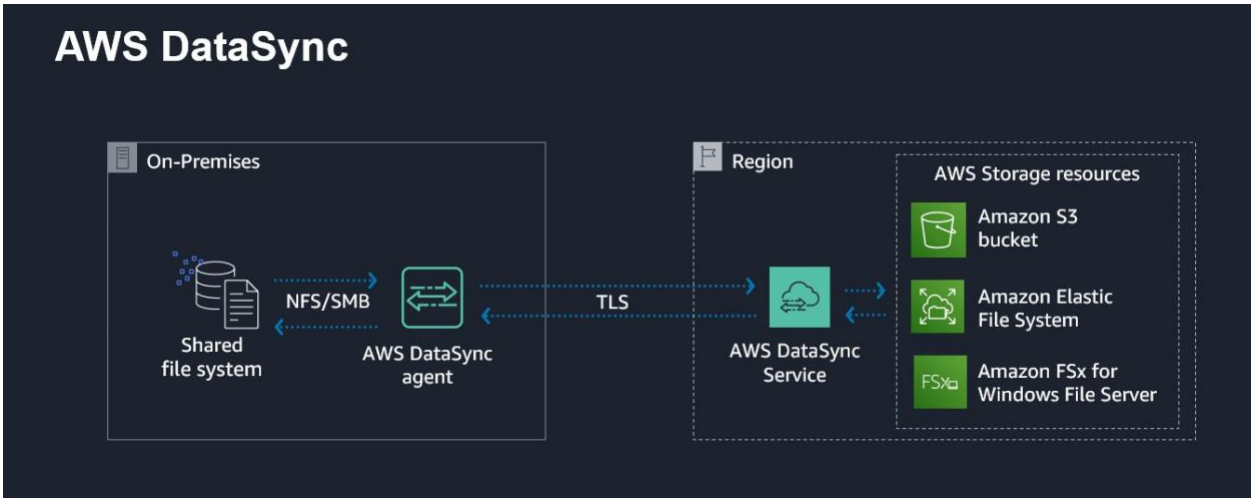

Figure 5: AWS Data Sync Architecture illustration

Data synchronization between on-premises environments and AWS

The AWS Transfer Family provides a fully managed option to ingest files into Amazon S3 via SFTP, FTP and FTPS protocols.

The AWS Transfer Family supports common user authentication systems, including Microsoft Active Directory and Lightweight Directory Access Protocol (LDAP). Alternatively, you can also choose to store and manage users' credentials directly within the service. By connecting your existing identity provider to the AWS Transfer Family service, you assure that your external users continue to have the correct, secure level of access to your data resources without disruption.

The AWS Transfer Family enables seamless migration by allowing you to import host keys, use static IP addresses, and use existing hostnames for your servers. With these features, user scripts and applications that use your existing file transfer systems continue working without changes.

AWS Transfer Family is simple to use. You can deploy an AWS Transfer for SFTP endpoint in minutes with a few clicks. Using the service, you simply consume a

managed file transfer endpoint from the AWS Transfer Family, configure your users and then configure an Amazon S3 bucket to use as your storage location.

Data copy between on-premises environments and AWS

Additionally, Amazon S3 natively supports DistCP (distributed copy) tool, which is a standard Apache Hadoop data transfer mechanism. This allows you to run DistCP jobs to transfer data from an on-premises Hadoop cluster to an S3 bucket. The command to transfer data typically looks like the following:

```
hadoop distcp hdfs://source-folder s3a://destination-bucket
```

For continued file ingestion, you can use a scheduler, such as `cron jobs`.

Programmatic data transfer using APIs and SDKs

You can also use the Amazon S3 REST APIs using AWS CLI or use your favorite programming languages with AWS SDKs to transfer the files into Amazon S3. Again, you will have to depend upon the schedulers available on the source systems to automate continuous ingestion. Note that the AWS CLI includes two namespaces for S3 – s3 and s3api. The `s3` namespace includes the common commands for interfacing with S3, whereas `s3api` includes a more advanced set of commands. Some of the syntax is different between these two APIs.

Transfer of large data sets from on-premises environments into AWS

If you are dealing with large amounts of data, from terabytes to even exabytes, then ingesting over the wire may not be ideal. In this scenario, you can use the AWS Snow Family of devices to securely and efficiently migrate bulk data from on-premises storage platforms and Hadoop clusters to S3 buckets. Usually, you can use them for one-time ingestion of data, but periodic data transfers can also be scheduled for continuous ingestion.

The AWS Snow Family, comprising AWS Snowcone, AWS Snowball, and AWS Snowmobile, offers a number of physical devices and capacity points. For a feature comparison of each device, see AWS Snow Family. When choosing a particular member of the AWS Snow Family, it is important to consider not only the data size but also the supported network interfaces and speeds, device size, portability, job lifetime, and supported APIs. Snow Family devices are owned and managed by AWS and

integrate with AWS security, monitoring, storage management, and computing capabilities.

Heterogeneous data ingestion patterns

Heterogeneous data files ingestion

This section covers use cases where you are looking to ingest the data and change the original file format and/or load it into a purpose-built data storage destination and/or perform transformations while ingesting data. The use case for this pattern usually falls under outside-in or inside-out data movement in the Lake House architecture. Common use cases for inside-out data movement include loading the data warehouse storage (for example, Amazon Redshift) or data indexing solutions (for example, Amazon Elasticsearch Service) from data lake storage. Common use cases for outside-in data movement are ingesting CSV files from on-premises to an optimized parquet format for querying or to merge the data lake with changes from the new files. These may require complex transformations along the way, which may involve processes like changing data types, performing lookups, cleaning, and standardizing data, and so on before they are finally ingested into the destination system. Consider the following tools for these use cases.

Data extract, transform, and load (ETL)

AWS Glue is a serverless data integration service that makes it easy to discover, prepare, and combine data for analytics, machine learning, and application development. It provides various mechanisms to perform extract, transform, and load (ETL) functionality. AWS Glue provides both visual and code-based interfaces to make data integration easier. Data engineers and ETL developers can visually create, run, and monitor ETL workflows with a few clicks in AWS Glue Studio.

You can build event-driven pipelines for ETL with AWS Glue ETL. See the following example.

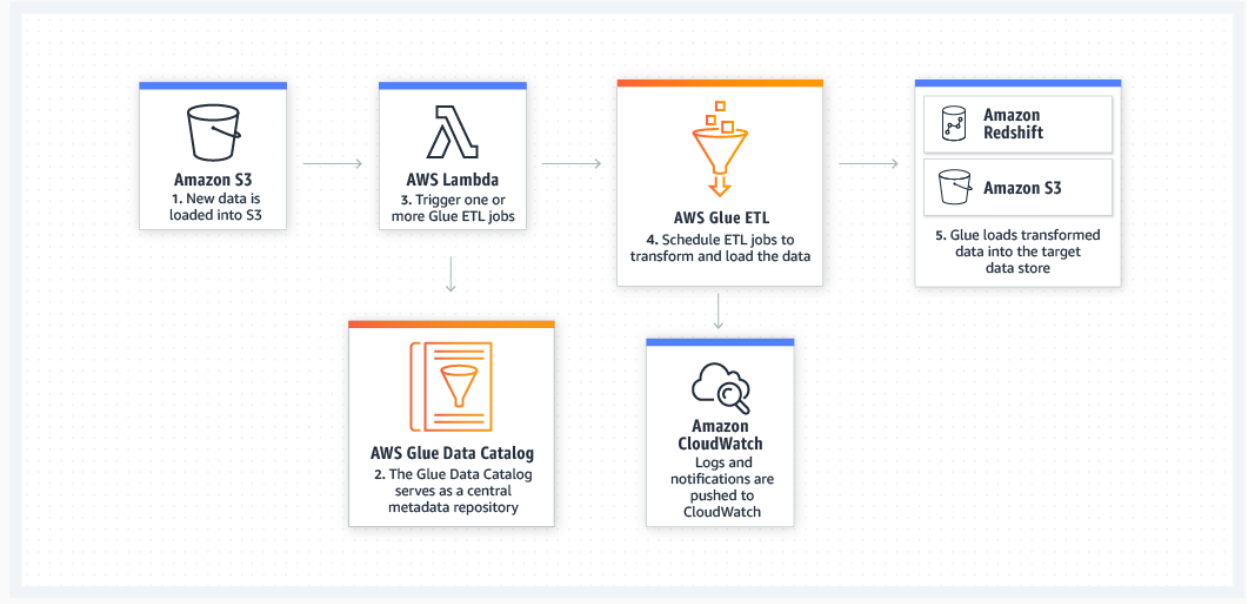

Figure 6 : AWS Glue ETL architecture Illustration

You can use AWS Glue as a managed ETL tool to connect to your data centers for ingesting data from files while transforming data and then load the data into your data storage of choice in AWS (or example, Amazon S3 data lake storage or Amazon Redshift). For details on how to set up AWS Glue in a hybrid environment when you are ingesting data from on-premises data centers, see How to access and analyze on-premises data stores using AWS Glue.

AWS Glue supports various format options for files both as input and as output. These formats include avro, csv, ion, orc, and more. For a complete list of supported formats, see AWS Glue programming ETL format.

AWS Glue provide various connectors to connect to the different source and destination targets. For a reference of all tconnectors and their usage as source or sink, see Connection Types and Options for ETL in AWS Glue.

AWS Glue supports Python and Scala for programming your ETL. As part of the transformation, AWS Glue provides various transform classes for programming with both PySpark and Scala.

You can use AWS Glue to meet the most complex data ingestion needs for your Lake House architecture. Most of these ingestion workloads must be automated in enterprises and can follow a complex workflow. You can use Workflows in AWS Glue to achieve orchestration of AWS Glue workloads. For more complex workflow orchestration and automation, use AWS Data Pipeline.

Using native tools to ingest data into data management systems

AWS services may also provide native tools/APIs to ingest data from files into respective data management systems. For example, Amazon Redshift provides the `COPY` command which uses the Amazon Redshift massively parallel processing (MPP) architecture to read and load data in parallel from files in Amazon S3, from an Amazon DynamoDB table, or from text output from one or more remote hosts. For a reference to the Amazon Redshift COPY command, see Using a COPY command to load data. Note that the files need to follow certain formatting to be loaded successfully. For details, see Preparing your input data. You can use AWS Glue ETL to perform the required transformation to bring the input file in the right format.

Amazon Keyspaces (for Apache Cassandra) is a scalable, highly available, managed Cassandra-compatible database service that provides a `cqlsh` copy command to load data into an Amazon Keyspaces table. For more details, including best practices and performance tuning, see Loading data into Amazon Keyspaces with cqlsh.

Using third-party vendor tools

Many customers may already be using third-party vendor tools for ETL jobs in their data centers. Depending upon the access, scalability, skills, and licensing needs, customers can choose to use these tools to ingest files into the Lake House architecture for various data movement patterns. Some third-party tools include MS SQL Server Integration Services (SSIS), IBM DataStage, and more. This whitepaper does not cover those options here. However, it is important to consider aspects like native connectors provided by these tools (for example, having connectors for Amazon S3, or Amazon Athena) as opposed to getting a connector from another third-party. Further considerations include security scalability, manageability, and maintenance of those options to meet your enterprise data ingestion needs.

Streaming data ingestion

One of the core capabilities of a Lake House architecture is the ability to ingest streaming data quickly and easily. Streaming data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes). Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers.

This data must be processed sequentially and incrementally on a record-by-record basis or over sliding time windows, and used for a wide variety of analytics including correlations, aggregations, filtering, and sampling. When ingesting streaming data, the use case may require to first load the data into your data lake before processing it, or it may need to be analyzed as it is streamed and stored in the destination data lake or purpose-built storage.

Information derived from such analysis gives companies visibility into many aspects of their business and customer activity—such as service usage (for metering/billing), server activity, website clicks, and geo-location of devices, people, and physical goods—and enables them to respond promptly to emerging situations. For example, businesses can track changes in public sentiment on their brands and products by continuously analyzing social media streams and respond in a timely fashion as the necessity arises.

AWS provides several options to work with streaming data. You can take advantage of the managed streaming data services offered by Amazon Kinesis or deploy and manage your own streaming data solution in the cloud on Amazon EC2.

AWS offers streaming and analytics managed services such as Amazon Kinesis Data Firehose, Amazon Kinesis Data Streams, Amazon Kinesis Data Analytics, and Amazon Managed Streaming for Apache Kafka (Amazon MSK).

In addition, you can run other streaming data platforms, such as Apache Flume, Apache Spark Streaming, and Apache Storm, on Amazon EC2 and Amazon EMR.

Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose is the easiest way to load streaming data into AWS. You can use Kinesis Data Firehose to quickly ingest real-time clickstream data and move the data into a central repository, such as Amazon S3, which can act as a data lake. Not only this, as you ingest data, you have the ability to transform the data as well as integrate with other AWS services and third-party services for use cases such as log analytics, IoT analytics, security monitoring, and more

Amazon Kinesis Data Firehose is a fully managed service for delivering real-time streaming data directly to Amazon S3. Kinesis Data Firehose automatically scales to match the volume and throughput of streaming data, and requires no ongoing administration. Kinesis Data Firehose can also be configured to transform streaming data before it’s stored in Amazon S3. Its transformation capabilities include compression, encryption, data batching, and AWS Lambda functions.

Kinesis Data Firehose can compress data before it's stored in Amazon S3. It currently supports GZIP, ZIP, and SNAPPY compression formats. GZIP is the preferred format because it can be used by Amazon Athena, Amazon EMR, and Amazon Redshift. Kinesis Data Firehose encryption supports Amazon S3 server-side encryption with AWS Key Management Service (AWS KMS) for encrypting delivered data in Amazon S3. You can choose not to encrypt the data or to encrypt with a key from the list of AWS KMS keys that you own (see Encryption with AWS KMS). Kinesis Data Firehose can concatenate multiple incoming records, and then deliver them to Amazon S3 as a single S3 object. This is an important capability because it reduces Amazon S3 transaction costs and transactions per second load.

Finally, Kinesis Data Firehose can invoke AWS Lambda functions to transform incoming source data and deliver it to Amazon S3. Common transformation functions include transforming Apache Log and Syslog formats to standardized JSON and/or CSV formats. The JSON and CSV formats can then be directly queried using Amazon Athena. If using a Lambda data transformation, you can optionally back up raw source data to another S3 bucket, as shown in the following figure.

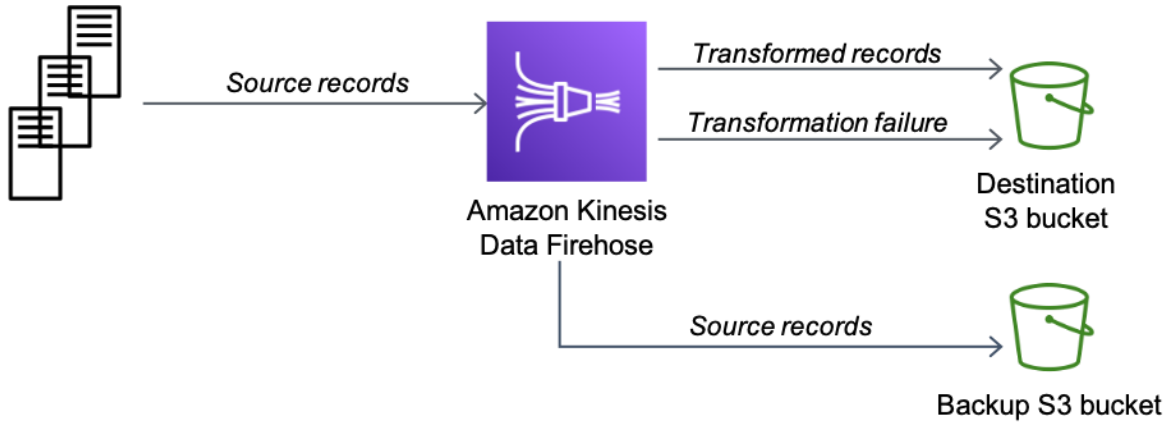

Figure 7: Delivering real-time streaming data with Amazon Kinesis Data Firehose to Amazon S3 with optional backup

Sending data to an Amazon Kinesis Data Firehose Delivery Stream

There are several options to send data to your delivery stream. AWS offers SDKs for many popular programming languages, each of which provides APIs for Kinesis Data Firehose. AWS has also created a utility to help send data to your delivery stream.

Using the API

The Kinesis Data Firehose API offers two operations for sending data to your delivery stream: `PutRecord` sends one data record within one call, `PutRecordBatch` can send multiple data records within one call.

In each method, you must specify the name of the delivery stream and the data record, or array of data records, when using the method. Each data record consists of a data BLOB that can be up to 1,000 KB in size and any kind of data.

For detailed information and sample code for the Kinesis Data Firehose API operations, see Writing to a Firehose Delivery Stream Using the AWS SDK.

Using the Amazon Kinesis Agent

The Amazon Kinesis Agent is a stand-alone Java software application that offers an easy way to collect and send data to Kinesis Data Streams and Kinesis Data Firehose. The agent continuously monitors a set of files and sends new data to your stream. The agent handles file rotation, checkpointing, and retry upon failures. It delivers all of your data in a reliable, timely, and simple manner. It also emits Amazon CloudWatch metrics to help you better monitor and troubleshoot the streaming process.

You can install the agent on Linux-based server environments such as web servers, log servers, and database servers. After installing the agent, configure it by specifying the files to monitor and the destination stream for the data. After the agent is configured, it durably collects data from the files and reliably sends it to the delivery stream.

The agent can monitor multiple file directories and write to multiple streams. It can also be configured to pre-process data records before they're sent to your stream or delivery stream.

If you're considering a migration from a traditional batch file system to streaming data, it's possible that your applications are already logging events to files on the file systems of your application servers. Or, if your application uses a popular logging library (such as Log4j), it is typically a straight-forward task to configure it to write to local files. Regardless of how the data is written to a log file, you should consider using the agent in this scenario. It provides a simple solution that requires little or no change to your existing system. In many cases, it can be used concurrently with your existing batch solution. In this scenario, it provides a stream of data to Kinesis Data Streams, using the log files as a source of data for the stream.

In our example scenario, we chose to use the agent to send streaming data to the delivery stream. The source is on-premises log files, so forwarding the log entries to Kinesis Data Firehose was a simple installation and configuration of the agent. No additional code was needed to start streaming the data.

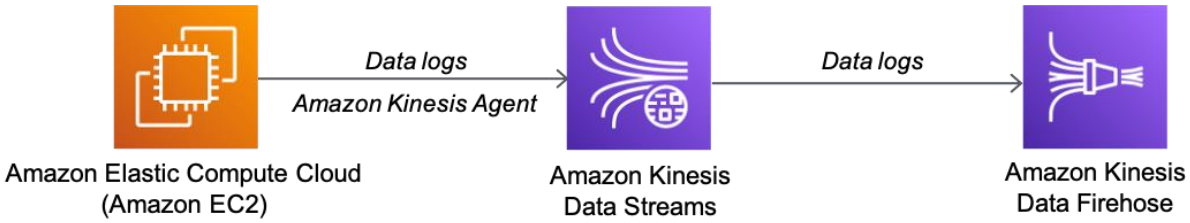

Figure 8: Kinesis Agent to monitor multiple file directories

Data transformation

In some scenarios, you may want to transform or enhance your streaming data before it is delivered to its destination. For example, data producers might send unstructured text in each data record, and you may need to transform it to JSON before delivering it to Amazon Elasticsearch Service.

To enable streaming data transformations, Kinesis Data Firehose uses an AWS Lambda function that you create to transform your data.

Data transformation flow

When you enable Kinesis Data Firehose data transformation, Kinesis Data Firehose buffers incoming data up to 3 MB or the buffering size you specified for the delivery stream, whichever is smaller. Kinesis Data Firehose then invokes the specified Lambda function with each buffered batch asynchronously. The transformed data is sent from Lambda to Kinesis Data Firehose for buffering. Transformed data is delivered to the destination when the specified buffering size or buffering interval is reached, whichever happens first. The following figure illustrates this process for a delivery stream that delivers data to Amazon S3.

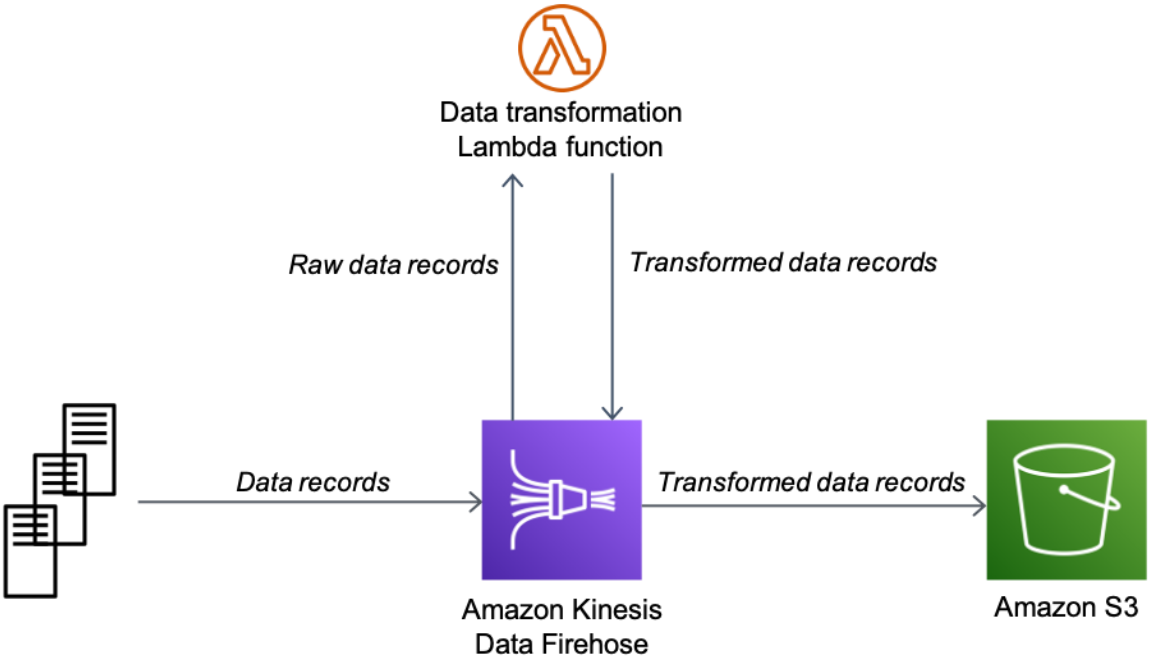

Figure 9: Kinesis Agent to monitor multiple file directors and write to Kinesis Data Firehose

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams enables you to build your own custom applications that process or analyze streaming data for specialized needs. It can continuously capture and store terabytes of data per hour from hundreds of thousands of sources. You can use Kinesis Data Streams to run real-time analytics on high frequency event data such as stock ticker data, sensor data, and device telemetry data to gain insights in a matter of minutes versus hours or days.

Kinesis Data Streams provide many more controls in terms of how you want to scale the service to meet high demand use cases, such as real-time analytics, gaming data feeds, mobile data captures, log and event data collection, and so on. You can then build applications that consume the data from Amazon Kinesis Data Streams to power real-time dashboards, generate alerts, implement dynamic pricing and advertising, and more. Amazon Kinesis Data Streams supports your choice of stream processing framework including Kinesis Client Library (KCL), Apache Storm, and Apache Spark Streaming.

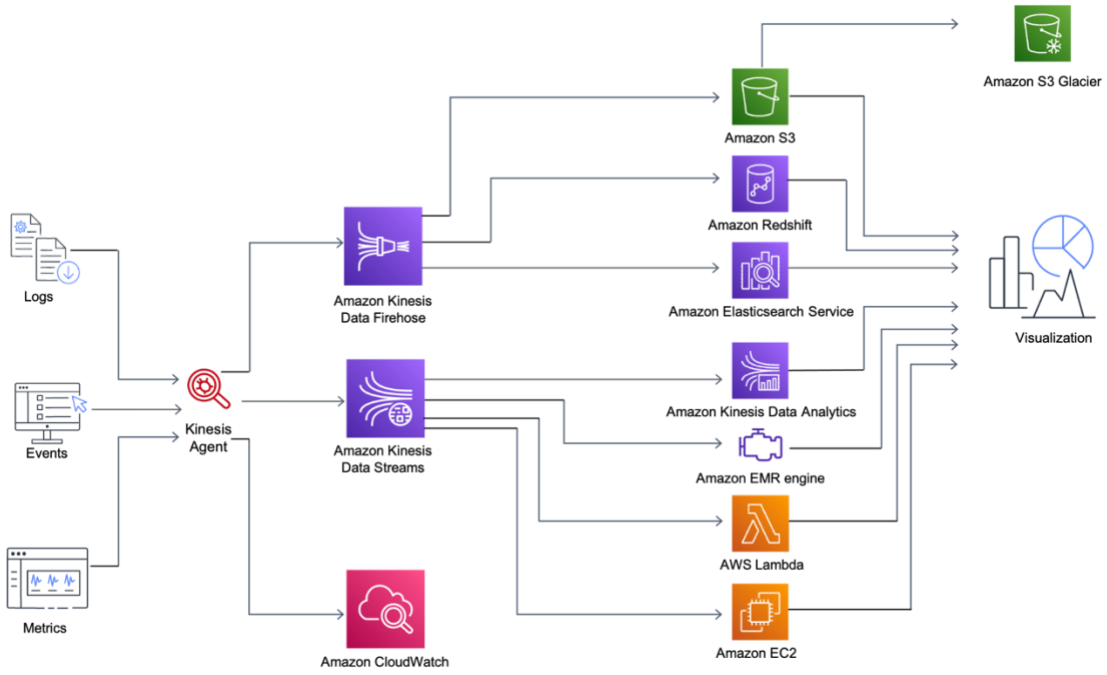

Figure 10: Custom real-time pipelines using stream-processing frameworks

Sending data to Amazon Kinesis Data Streams

There are several mechanisms to send data to your stream. AWS offers SDKs for many popular programming languages, each of which provides APIs for Kinesis Data Streams. AWS has also created several utilities to help send data to your stream.

Amazon Kinesis Agent

The Amazon Kinesis Agent can be used to send data to Kinesis Data Streams. For details on installing and configuring the Kinesis agent, see Writing to Amazon Kinesis Firehose Using Amazon Kinesis Agent.

Amazon Kinesis Producer Library (KPL)

The KPL simplifies producer application development, allowing developers to achieve high write throughput to one or more Kinesis streams. The KPL is an easy-to-use, highly configurable library that you install on your hosts that generate the data that you want to stream to Kinesis Data Streams. It acts as an intermediary between your producer application code and the Kinesis Data Streams API actions.

The KPL performs the following primary tasks:

- Writes to one or more Kinesis streams with an automatic and configurable retry mechanism

- Collects records and uses `PutRecords` to write multiple records to multiple shards per request
- Aggregates user records to increase payload size and improve throughput
- Integrates seamlessly with the Amazon Kinesis Client Library (KCL) to de-aggregate batched records on the consumer
- Submits Amazon CloudWatch metrics on your behalf to provide visibility into producer performance

The KPL can be used in either synchronous or asynchronous use cases. We suggest using the higher performance of the asynchronous interface unless there is a specific reason to use synchronous behavior. For more information about these two use cases and example code, see Writing to your Kinesis Data Stream Using the KPL.

Using the Amazon Kinesis Client Library (KCL)

You can develop a consumer application for Kinesis Data Streams using the Kinesis Client Library (KCL). Although you can use the Kinesis Streams API to get data from an Amazon Kinesis stream, we recommend using the design patterns and code for consumer applications provided by the KCL.

The KCL helps you consume and process data from a Kinesis stream. This type of application is also referred to as a consumer. The KCL takes care of many of the complex tasks associated with distributed computing, such as load balancing across multiple instances, responding to instance failures, checkpointing processed records, and reacting to resharding. The KCL enables you to focus on writing record-processing logic.

The KCL is a Java library; support for languages other than Java is provided using a multi-language interface. At run time, a KCL application instantiates a worker with configuration information, and then uses a record processor to process the data received from a Kinesis stream. You can run a KCL application on any number of instances. Multiple instances of the same application coordinate on failures and load-balance dynamically. You can also have multiple KCL applications working on the same stream, subject to throughput limits. The KCL acts as an intermediary between your record processing logic and Kinesis Streams.

For detailed information on how to build your own KCL application, see Developing Amazon Kinesis Streams Consumers Using the Amazon Kinesis Client Library.

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Amazon Managed Streaming for Apache Kafka (Amazon MSK) is a fully managed service that makes it easy for you to build and run applications that use Apache Kafka to process streaming data. Apache Kafka is an open-source platform for building real-time streaming data pipelines and applications. With Amazon MSK, you can use native Apache Kafka APIs to populate data lakes, stream changes to and from databases, and power machine learning and analytics applications. Amazon MSK is tailor made for use cases that require ultra-low latency (less than 20 milliseconds) and higher throughput through a single partition. With Amazon MSK, you can offload the overhead of maintaining and operating Apache Kafka to AWS which will result in significant cost savings when compared to running a self-hosted version of Apache Kafka.

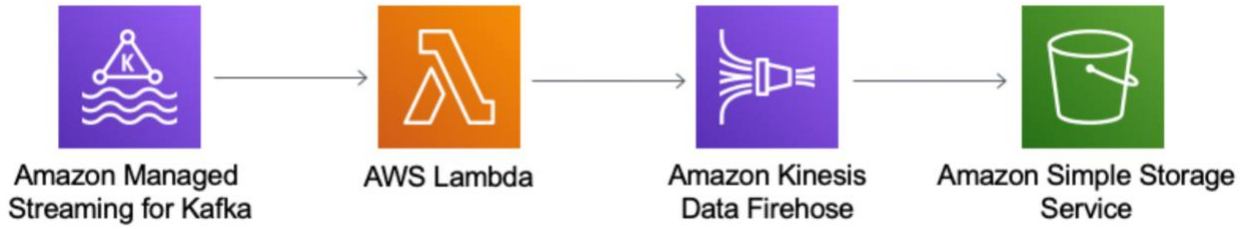

Figure 10: Managed Kafka for storing streaming data in an Amazon S3 data lake

Other streaming solutions in AWS

You can install streaming data platforms of your choice on Amazon EC2 and Amazon EMR, and build your own stream storage and processing layers. By building your streaming data solution on Amazon EC2 and Amazon EMR, you can avoid the friction of infrastructure provisioning, and gain access to a variety of stream storage and processing frameworks. Options for streaming the data storage layer include Amazon MSK and Apache Flume. Options for streaming the processing layer include Apache Spark Streaming and Apache Storm.

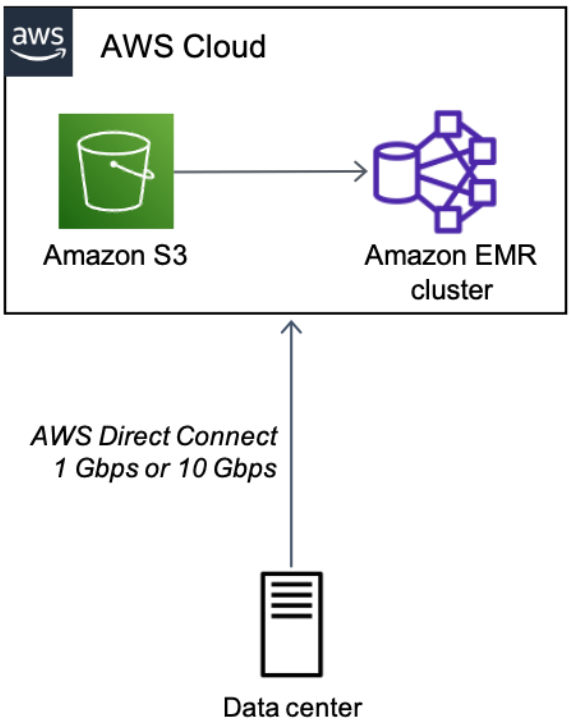

Figure 11: Moving data to AWS using Amazon EMR

Relational data ingestion

One of common scenarios for Lake House Architecture is where customers move relational data from on-premises data centers into the AWS Cloud or from within the AWS Cloud into managed relational database services offered by AWS, such as Amazon Relational Database Service (Amazon RDS), and Amazon Redshift – a cloud data warehouse. This pattern can be used for both onboarding the data into Lake House architecture and/or migrating data from commercial relational database engines into Amazon RDS. Also, AWS provides managed services for loading and migrating relational data from Oracle or SQL Server databases to Amazon RDS and Amazon Aurora – a fully managed relational database engine that's compatible with MySQL and PostgreSQL.

Customers migrating into Amazon RDS and Amazon Aurora managed database services gain benefits of operating and scaling a database engine without extensive administration and licensing requirements. Also, customers gain access to features such as backtracking where relational databases can be backtracked to a specific time, without restoring data from a backup, restoring database cluster to a specified time, and avoiding database licensing costs.

Customers with data in on-premises warehouse databases gain benefits by moving the data to Amazon Redshift – a cloud data warehouse database that simplifies administration and scalability requirements.

The data migration process across heterogeneous database engines is a two-step process:

1. Assessment and Schema Conversion
2. Loading of data into AWS managed database services

Once data is onboarded, you can decide whether to maintain a copy with up-to-date changes of a database in support of Lake House architecture or cut-over applications to use the managed database for both application needs and Lake House architecture.

Schema Conversion Tool

AWS Schema Conversion Tool (SCT) is used to facilitate heterogeneous database assessment and migration by automatically converting the source database schema and code objects to a format that's compatible with the target database engine. The custom code that it converts includes views, stored procedures, and functions. Any code that SCT cannot convert automatically is flagged for manual conversion.

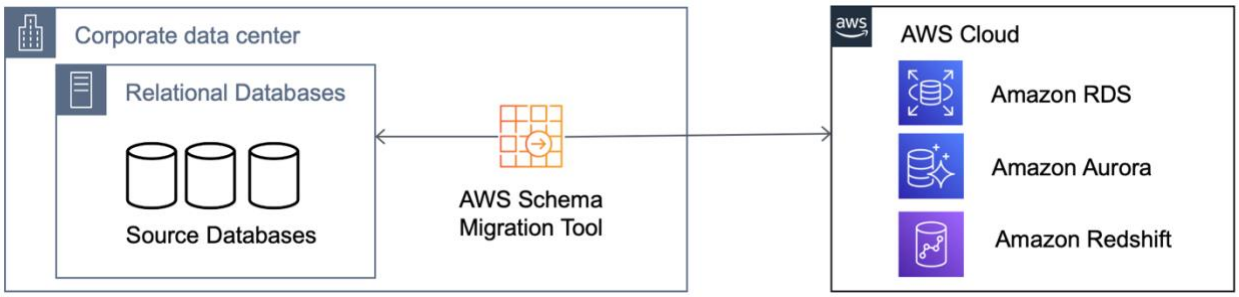

Figure 12: AWS Schema Conversion Tool

AWS Database Migration Service (AWS DMS)

AWS Database Migration Service (AWS DMS) is used to perform initial data load from on-premises database engine into target (Amazon Aurora). After the ingestion load is completed, depending on whether you need an on-going replication, AWS DMS or other migration and change data capture (CDC) solutions can be used for propagating the changes (deltas) from a source to the target database.

Network connectivity in a form of AWS VPN or AWS Direct Connect between on-premises data centers and the AWS Cloud must be established and sized accordingly to ensure secure and realizable data transfer for both initial and ongoing replications.

When loading large databases, especially in cases when there is a low bandwidth connectivity between the on-premises data center and AWS Cloud, it's recommended to use AWS Snowball or similar data storage devices for shipping data to AWS. Such physical devices are used for securely copying and shipping the data to AWS Cloud. Once devices are received by AWS, the data is securely loaded into Amazon Simple Storage Service (Amazon S3), and then ingested into an Amazon Aurora database engine. **N**etwork connectivity must be sized accordingly to that data can be initially loaded in a timely manner, and ongoing CDC does not incur latency lag.

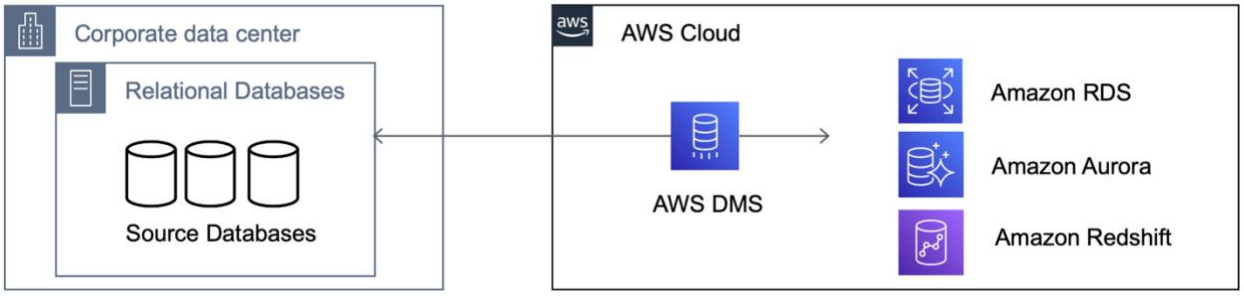

Figure 13: AWS Database Migration Service

When moving data from on-premises databases or storing data in the cloud, security and access control of the data is an important aspect that must be accounted for in any architecture. AWS services use Transport Level Security (TLS) for securing data in transit. For securing data at rest, AWS offers a large number of encryption options for encrypting data automatically using AWS provided keys, customer provided keys and even using Hardware Security Module (HSM). Once data is loaded in AWS and securely stored, the pattern must account for providing controlled and auditable access to the data at the right level of granularity. In AWS, a combination of AWS Identity & Access Management and AWS Lake Formation services can be used to achieve this requirement.

Ingestion between relational and non-relational data stores

Many organizations consider migrating from commercial relational data stores to non-relational data stores to align with the application and analytics modernization strategies. These AWS customers modernize their applications with microservices architecture. In cases of microservices architecture, customers choose to implement separate datastores for each microservice that is highly available and can scale to meet

the demand. In regard to analytics modernization strategies s, in many situations these customers move their data around the perimeter of the Lake House architecture from one purpose-built store like a relational database to a non-relational database and from a non-relational to a relational data store to derive insights from their data.

In addition, relational database management systems (RDBMSs) require up-front schema definition, and changing the schema later is very expensive. There are many use cases where it's very difficult to anticipate the database schema upfront that the business will eventually need. Therefore, RDBMS backends may not be appropriate for applications that work with a variety of data. However, NoSQL databases (like document databases) have dynamic schemas for unstructured data, and you can store data in many ways. They can be column-oriented, document-oriented, graph-based, or organized as a key-value store. The following section illustrates the ingestion pattern for these use cases.

Migrating or ingesting data from a relational data store to NoSQL data store

Migrating from commercial relational databases like Microsoft SQL Server or Oracle to Amazon DynamoDB is challenging because of their difference in schema. There are many different schema design considerations when moving from relational databases to NoSQL databases..

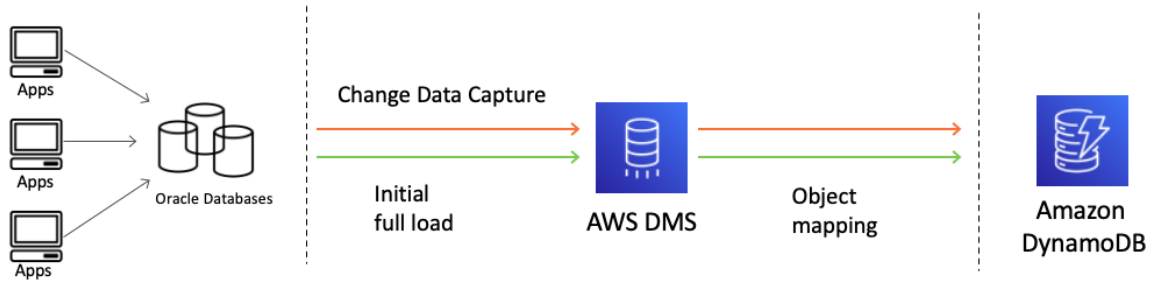

Figure 14: Migrating data from Relational data store to NoSQL data store

You can use AWS Database Migration Service (AWS DMS) to migrate your data to and from the most widely used commercial and open-source databases. It supports homogeneous and heterogeneous migrations between different database platforms. AWS DMS supports migration to a DynamoDB table as a target. You use object mapping to migrate your data from a source database to a target DynamoDB table.

Object mapping enables you to determine where the source data is located in the target. You can also create a DMS task that captures the ongoing changes from the source database and apply these to DynamoDB as target. This task can be full load plus change data capture (CDC) or CDC only.

One of the key challenges when refactoring to Amazon DynamoDB is identifying the access patterns and building the data model. There are many best practices for designing and architecting with Amazon DynamoDB. AWS provides NoSQL Workbench for Amazon DynamoDB. NoSQL Workbench is a cross-platform client-side GUI application for modern database development and operations and is available for Windows, macOS, and Linux. NoSQL Workbench is a unified visual IDE tool that provides data modeling, data visualization, and query development features to help you design, create, query, and manage DynamoDB tables.

Migrating or ingesting data from a relational data store to a document DB (such as Amazon DocumentDB [with MongoDB compatibility])

Amazon DocumentDB (with MongoDB compatibility) is a fast, scalable, highly available, and fully managed document database service that supports MongoDB workloads. As a document database, Amazon DocumentDB makes it easy to store, query, and index JSON data.

In this scenario, converting the relational structures to documents can be complex and may require building complex data pipelines for transformations. Amazon Database Migration Services (AWS DMS) can simplify the process of the migration and replicate ongoing changes.

AWS DMS maps database objects to Amazon DocumentDB in the following ways:

- A relational database, or database schema, maps to an Amazon DocumentDB database.
- Tables within a relational database map to collections in Amazon DocumentDB.
- Records in a relational table map to documents in Amazon DocumentDB. Each document is constructed from data in the source record.

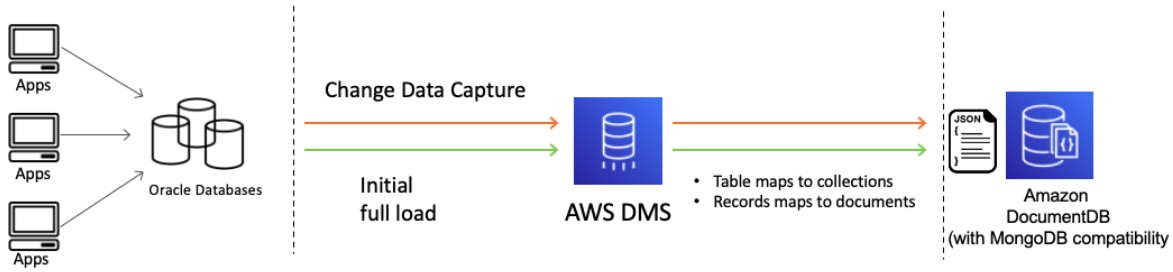

Figure 15 – Migrating data from a relational data store to DocumentDB

AWS DMS reads records from the source endpoint, and constructs JSON documents based on the data it reads. For each JSON document, AWS DMS determines an `_id` field to act as a unique identifier. It then writes the JSON document to an Amazon DocumentDB collection, using the `_id` field as a primary key.

Migrating or ingesting data from a document DB (such as Amazon DocumentDB [with MongoDB compatibility] to a relational database

AWS DMS supports Amazon DocumentDB (with MongoDB compatibility) as a database source. You can use AWS DMS to migrate or replicate changes from Amazon DocumentDB to relational database such as Amazon Redshift for data warehousing use cases. Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. AWS DMS supports two migration modes when using DocumentDB as a source, document mode and table mode.

In document mode, the JSON documents from DocumentDB are migrated as is. So, when you use a relational database as a target, the data is a single column named _doc in a target table. You can optionally set the extra connection attribute extractDocID to true to create a second column named "_id" that acts as the primary key. If you use change data capture (CDC), set this parameter to true except when using Amazon DocumentDB as the target.

In table mode, AWS DMS transforms each top-level field in a DocumentDB document into a column in the target table. If a field is nested, AWS DMS flattens the nested values into a single column. AWS DMS then adds a key field and data types to the target table's column set.

The change streams feature in Amazon DocumentDB (with MongoDB compatibility) provides a time-ordered sequence of change events that occur within your cluster's

collections. You can read events from a change stream using AWS DMS to implement many different use cases, including the following:

- Change notification
- Full-text search with Amazon Elasticsearch Service (Amazon ES)
- Analytics with Amazon Redshift

After change streams are enabled, you can create a migration task in AWS DMS that migrates existing data and at the same time replicates ongoing changes. AWS DMS continues to capture and apply changes even after the bulk data is loaded. Eventually, the source and target databases synchronize, minimizing downtime for a migration.

During a database migration when Amazon Redshift is the target for data warehousing use cases, AWS DMS first moves data to an Amazon Simple Storage Service (Amazon S3) bucket. When the files reside in an Amazon S3 bucket, AWS DMS then transfers them to the proper tables in the Amazon Redshift data warehouse.

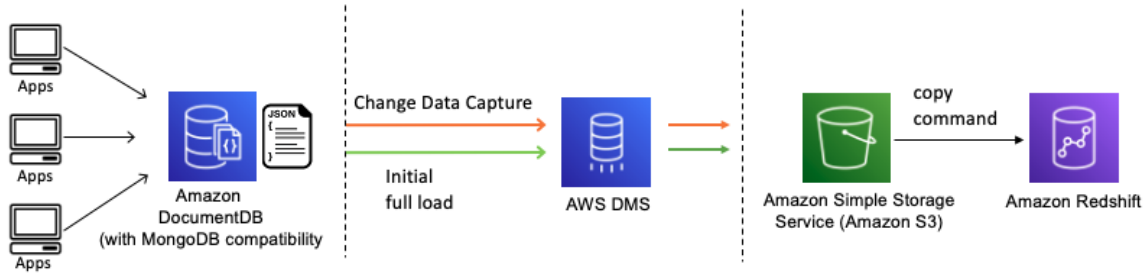

Figure 16 Migrating data from NoSQL store to Amazon Redshift

AWS Database Migration Service supports both full load and change processing operations. AWS DMS reads the data from the source database and creates a series of comma-separated value (.csv) files. For full-load operations, AWS DMS creates files for each table. AWS DMS then copies the table files for each table to a separate folder in Amazon S3. When the files are uploaded to Amazon S3, AWS DMS sends a copy command and the data in the files are copied into Amazon Redshift. For change-processing operations, AWS DMS copies the net changes to the .csv files. AWS DMS then uploads the net change files to Amazon S3 and copies the data to Amazon Redshift.

Migrating or ingesting data from a document DB (such as Amazon Document DB [with MongoDB compatibility]) to Amazon Elasticsearch Service

Taking the same approach as AWS DMS support for Amazon DocumentDB (with MongoDB) as a database source, you can migrate or replicate changes from Amazon DocumentDB to Amazon Elasticsearch Service (Amazon ES) as the target.

In Amazon ES, you work with indexes and documents. An *index* is a collection of documents, and a *document* is a JSON object containing scalar values, arrays, and other objects. Elasticsearch Service provides a JSON-based query language, so that you can query data in an index and retrieve the corresponding documents. When AWS DMS creates indexes for a target endpoint for Amazon ES, it creates one index for each table from the source endpoint.

AWS DMS supports multithreaded full load to increase the speed of the transfer, and multithreaded CDC load to improve the performance of CDC. For the task settings and prerequisites that are required to be configured for these modes, see Using an Amazon Elasticsearch Service cluster as a target for AWS Database Migration Service.

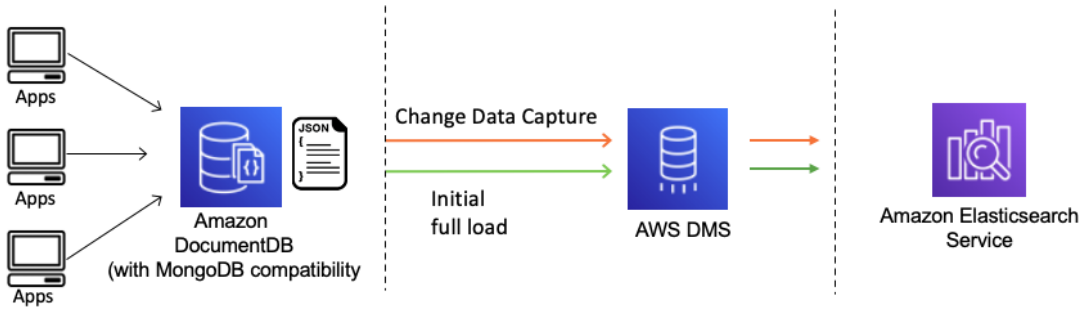

Figure 16 Migrating data from Amazon DocumentDB store to Amazon Elasticsearch Service

Conclusion

With the massive amount of data growth, organizations are focusing on driving greater efficiency in their operations by making data driven decisions. As data is coming from various sources in different forms, organizations are tasked with how to integrate terabytes to petabytes and sometimes exabytes of data that were previously siloed in order to get a complete view of their customers and business operations. Traditional on-

premises data analytics solutions can't handle this approach because they don't scale well enough and are too expensive. As a result, customers are looking to modernize their data and analytics infrastructure by moving to the cloud.

To analyze these vast amounts of data, many companies are moving all their data from various silos into a Lake House architecture. A cloud-based Lake House architecture on AWS allows customers to take advantages around scale, innovation, elasticity and agility to meet their data analytics and machine learning needs. As such, ingesting data into the cloud becomes an important step of the overall architecture. This whitepaper addressed the various patterns, scenarios, use cases and the right tools for the right job that an organization should consider while ingesting data into AWS.

Contributors

Contributors to this document include:

- Divyesh Sah, Sr. Enterprise Solutions Architect, Amazon Web Services
- Varun Mahajan, Solutions Architect, Amazon Web Services
- Mikhail Vaynshteyn, Solutions Architect, Amazon Web Services
- Sukhomoy Basak, Solutions Architect, Amazon Web Services
- Bhagvan Davanam, Solutions Architect, Amazon Web Services

Further reading

- Database Migrations Case Studies
- Architectural Patterns for Migrating Database Workloads
- Migrating Database Workloads Using AWS Database Migration Service (walkthrough)
- Best Practices for Migrating Oracle Databases to Amazon RDS PostgreSQL or Amazon Aurora PostgreSQL
- Large-Scale Database Migrations with AMS DMS and AWS Snowball
- Applying Record Level Changes from Relational Databases to Amazon S3 Data Lake Using Apache Hudi
- Streaming CDC into Amazon S3 Data Lake in Parquet Format with AWS DMS

- Loading Data Lake Changes with AWS DMS and AWS Glue
- Performing a Live Migration from MongoDB Cluster to Amazon DynamoDB

Document history

Date	Description
July 23, 2021	First publication