Lambda

Introduction

* Lambda is a Function as a Service in AWS cloud. It is creating Runtime environment for executing your code without managing any infrastructure.
* It is a Serverless, Event driven compute Service.
* Simply you write your code and upload it as a Zip file or Container Image.
* It collects only Compute charge for executing your code, not for the infrastructure used.

How it works?

File Processing

* Use S3 to trigger AWS Lambda data processing in real time after an upload or connect to an existing Amazon EFS file system to enable massively parallel shared access for large-scale file processing.

How to trigger Lambda?

* Synchronous invocation (Push)
  + Using CLI or API Calls to invoke
  + Elastic Load Balancing (Application Load Balancer)
  + Amazon Cognito
  + Amazon Lex
  + Amazon Alexa
  + Amazon API Gateway
  + Amazon CloudFront (Lambda@Edge)
  + Amazon Kinesis Data Firehose
* Asynchronous invocation (Event)
  + If your function returns an error, AWS will automatically retry the invoke twice, for a total of three invocations.
  + It place your invoke request in Lambda service queue and we process the requests as they arrive. You Can Monitor the queue using AWS X-Ray.

- Amazon S3

- Amazon SNS

- Amazon Simple Email Service (SES)

- AWS CloudFormation

- Amazon CloudWatch Logs

- Amazon CloudWatch Events

- AWS CodeCommit

- AWS Config

* Poll-Based invocation (Poll-Based)
  + Lambda will be get invoked when the particular configuration is reached.
  + Amazon Kinesis
  + Amazon SQS
  + Amazon DynamoDB Streams

A screenshot of a computer

Description automatically generated

How to add External libraries?

* Zip the Libraries and Upload it to Lambda
* Lambda Layers

Orchestrate Multiple Lambda through Step Functions

Limitation of Lambda

* Each Lambda function has a maximum execution time of 15 minutes.
* Limited Execution memory size can range from 128 MB to 10,240 MB (10 GB). CPU scales based on the amount of memory.
* Deployment Package Size has max. of 250 MB (uncompressed).
* Total size of all environment variables is limited to 4 KB.
* Function Layers - 5

Soft Limits

* Concurrent Executions – 1000
* Function and Layer Storage - 75 GB

Hard Limits

* Function Memory Limit - 128 MB to 3008 MB, in 64 MB increments
* Function Timeout - 900 Seconds (15 mints)
* Function Environment Variables - 4 KB
* Function Layers – 5
* Deployment Package Size 🡪 50 MB (Zipped, for direct upload)

🡪 250 MB (Unzipped, Including Layers)

Performance tuning for Lambda

* Limit the logging in your lambda function. Excessive logging can impact performance.
* Use Execution Environment Variables to Avoid hardcoding configuration values in your code.
* If you have large functionality, then split the requirement multiple function and run it as Micro services.
* Use Lambda Layers to maintain the common functionality to access from multiple function.

Database 🡪 Relational Database Service (RDS)

DynamoDB – NoSQL Database Service

Introduction

* It is a Fully Managed NoSQL database provided by AWS. It’s a serverless and key value based.
* Whatever features and Limitation of No SQL will be applicable here as well. like Schema free. you can add you add/delete your schema whenever it is required. It will not support any Joins etc.
* It is a Region-Specific Service.

Features

* Flexible Capacity modes (throughput capacity)
  + Provisioned Throughput:
  + On-Demand Capacity:
* Continuous backups and restore
* Automated multi-Region replication
* In-memory caching
* Data import and export tools.
* Import data from Amazon S3 directly into a new DynamoDB table without writing any code
* Export data from DynamoDB to S3 and use other AWS services such as Amazon Athena
* Use PartiQL, a SQL-compatible query language, to query, insert, update, and delete table data in DynamoDB.
* Use Amazon Kinesis Data Streams to capture item-level changes in your DynamoDB tables.

Service quota basics (Limitations)

Table limits

* Table size – Unlimited
* Number of Tables - For any AWS account, there is an initial quota of 2,500 tables per AWS Region.
* Page size limit for query and scan - There is a limit of 1 MB per page, per query or scan.

Indexes

* Local secondary indexes (LSIs) – You can define a maximum of five local secondary indexes. LSIs are primarily useful when an index must have strong consistency with the base table.
* Global secondary indexes (GSIs) – There is a default quota of 20 global secondary indexes per table.
* Projected secondary index attributes per table – You can project a total of up to 100 attributes into all table's local and global secondary indexes. This only applies to user-specified projected attributes.

Partition keys

* The minimum length of a partition key value is 1 byte. The maximum length is 2048 bytes. (2 MB)

Items, attributes, and expression parameters

* The maximum item size in DynamoDB is 400 KB
* There is no limit on the number of values in a list, map, or set, if the item that contains the values fits within the 400-KB item size limit.
* For expression parameters, the maximum length of any expression string is 4 KB.

Concepts

Read/write capacity mode

* On-demand
  + In On-Demand Capacity mode, DynamoDB automatically adjusts your table's throughput capacity based on the actual traffic to your table.
  + RCU and WCU will be considered from DynamoDB perspective.
  + On-demand is perfect for unpredictable workloads with sudden spikes of traffic
* Provisioned (default, free-tier eligible)
  + In Provisioned Throughput mode, you specify the maximum number of reads and writes per second (capacity units) that you expect your application to require.
  + Suitable for predictable workloads.
  + provisioned mode is cheaper and better suited for workloads with predictable traffic.

You can switch between Provisioned Throughput and On-Demand Capacity at any time.

Capacity Units

* Capacity unites are used to control how many people can read or write data from your DynamoDB table at the same time.

Types of Capacity Units

* Read capacity unit (RCU)
  + One strongly consistent read per second, or two eventually consistent reads per second, for items up to 4 KB in size
* Write capacity unit (WCU)
  + One writes per second, for items up to 1 KB in size.

Partition Key

* It is like a Table's Primary Key. It is a hash value that will be used to retrieve items from the table.

Sort Key

* It is a Second part of Primary key. used to sort the items and can be applied to filter the items.

Indexes

* DynamoDB supports two different kinds of indexes
  + Global secondary indexes 🡪 The primary key of the index can be any two attributes from its table.
  + Local secondary indexes 🡪 The partition key of the index must be the same as the partition key of its table. However, the sort of key can be any other attribute.

Global Table

* Global tables replicate your DynamoDB tables automatically across your choice of AWS Regions.

Backup --> Can be scheduled per table level. ????

Time to Live (TTL) - Automatically delete the expiry Items from table

Default - Disabled.

Export and Streams

- Explore Items

Scan

* While scan query you have to scan whole table then apply filter on every single row to find the right result.

Query

* Get query you are using a primary key in where condition
* Query is much better than Scan - performence wise.

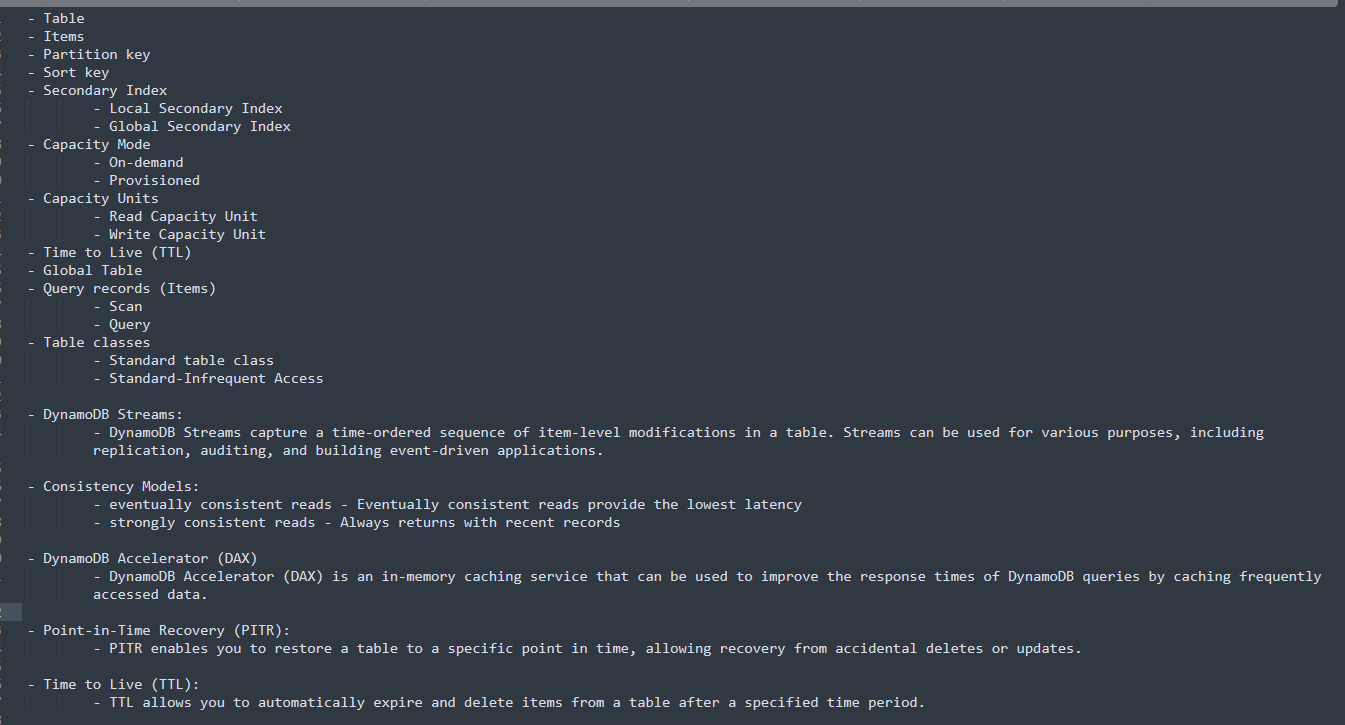
Table classes

* DynamoDB Standard table class is the default
* DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) table class is optimized for tables where storage is the dominant cost

Default setting of Table Creation

* Capacity mode – Provisioned
* Read Capacity - 5 RCU
* Write Capacity - 5 WCU
* Autoscaling – ON
* Local Secondary Index
* Global Secondary Index
* Table Class - DynamoDB Standard

Main Concepts



Analytical Services – EMR (Elastic MapReduce)

Introduction

* It is fully managed big data processing service which helps to processing and analyzing large datasets using popular frameworks such as Apache Hadoop, Apache Spark, Apache Hive, Apache HBase, Apache Flink, and more.
* EMR makes it easy to set up, operate, and scale distributed clusters for big data processing.

Features of EMR

* Fully Managed
  + AWS takes care of cluster provisioning, configuration, and maintenance. This allows users to focus on data processing tasks rather than cluster management.
* Elastic Scaling
  + EMR clusters can be easily scaled up or down based on the processing requirements. Auto Scaling features automatically adjust the number of instances in the cluster based on workload.
* Integration with Popular Big Data Frameworks
  + EMR supports a many big data processing frameworks, including Apache Hadoop, Apache Spark, Apache Hive, Apache HBase, Apache Flink, and more.
* Managed Notebooks
  + EMR Notebooks provide a managed environment for developing and running Spark and Hive queries.
* Integration with AWS Glue Data Catalog
  + EMR integrates with the AWS Glue Data Catalog, making it easy to discover, catalog, and share metadata about data sources.

Components of EMR

* EMR Cluster
  + EMR cluster consists of master and worker nodes that run big data processing frameworks.
* Master Node
  + Master node is responsible for managing the cluster and coordinating the distribution of tasks to worker nodes.
* Core Nodes
  + Core nodes run tasks and store data in the HDFS.
* Task Nodes
  + Task nodes run tasks but do not store data. They are useful for temporary processing and can be added or removed as needed.
* EMR File System (EMRFS)
  + EMRFS allows integration with Amazon S3 as a data storage layer. It enables data to be stored in S3 and accessed by EMR clusters.
* EMR Notebooks
  + Managed notebooks for interactive data analysis and visualization

Use Cases

* Data Analysis
  + Analyzing and processing large volumes of data using frameworks like Apache Spark.
* Data Warehousing
  + Running complex SQL queries and aggregations on large datasets stored in Amazon S3.
* Machine Learning
  + Building and training machine learning models using frameworks like Apache Spark MLlib and TensorFlow.

AWS Glue (Analytical Service)

Introduction

* It is a Fully Managed and Serverless Data integration service which helps to discover, prepare, ETL, integrate data from multiple data sources to Target systems.
* It is a managed ETL service for Apache Spark
* Supports only History Load Batch process (History Load). not suitable for incremental load, Streaming applications.
* Cost: Pay only for what you use.

Features

* Data Catalog
* Automated Schema Recognition using Crawlers
* Automated ETL Code Generation
* Scheduling Glue Jobs through Triggers
* Orchestration through Workflows.

Limitations

* Supports only History Load Batch process, not suitable for incremental load, Streaming applications.
* Glue supports only Structured databases. Not for No-SQL, Timeseries database.
* Difficult to combine stream and batch.
* Reliance on Spark (Glue is tightly coupled with Spark)
* AWS Glue runs jobs in Apache Spark. This means that the engineers who need to customize the generated ETL job must know Spark well.

How to trigger Glue job?

* Trigger one or more Glue jobs from an external source such as an AWS Lambda function.
  + Glue Triggers
  + Glue Workflows
* Orchestrate your AWS Glue jobs using
  + Step Functions
  + Workflows

Concepts

Data Catalog

* Databases
  + Tables
    - The Table contains the Metadata definition including Schema which can be used as Source or Target in Job definition.
    - Add table via Manual providing database name, table name, S3 location
    - Add table via Crawlers, choose data source and select source table to Crawl.
* Stream Schema Registries
  + Schemas
    - It is a Centralized Schema registry for Streaming Applications, where you can Integrate this schema with Apache Kafka, Amazon Managed Streaming for Apache Kafka, Amazon Kinesis Data Streams, Amazon Kinesis Data Analytics for Apache Flink, and AWS Lambda.
    - Schema registry Supports Avro and json format files.
    - AWS Glue Schema Registry is serverless and free to use.
* Connections
  + Connections are used for making connectivity for different data sources to Crawl or Discover data sources.
  + Connectors (Built in Connectors and Custom Connectors)
  + Provides built-in support for the most used data stores (such as Amazon Redshift, Amazon Aurora, Microsoft SQL Server, MySQL, MongoDB, and PostgreSQL) using JDBC connections
  + AWS Glue also allows you to use custom JDBC drivers in your extract, transform, and load (ETL) jobs. For data stores that are not natively supported, such as SaaS applications, you can use connectors.
* Crawlers
  + Crawlers are used to Crawl the multiple data source in Single run and create one or more tables in AWS Glue catalog under database. These tables will be used as Source or Target in your job definition.
  + It will access your data store, extracts metadata, and creates table definitions in the AWS Glue Data Catalog
  + We can schedule Crawl jobs to trigger Crawl to Update the Table definition.
* Classifiers
  + Classifiers are triggered during the Crawl task. It checks whether given file format Crawl can handle or not.
  + A classifier reads the data in a data store. If it recognizes the format of the data, it generates a schema in Data Catalog.
* Catalog Settings
  + Metadata Encryption
  + Encrypt Connection Passwords
* Data Integration and ETL
  + AWS Glue Studio
    - Jobs
* Interactive Sessions
  + Notebooks
    - Notebooks provides the interactive session for development, testing for your ETL scripts on Dev endpoint.
    - AWS Glue provides an interface to Sage Maker notebooks and Apache Zeppelin notebook servers.
    - We can run Spark also in Jupiter Notebooks
* Data Classification Tools
  + Sensitive Data detections
  + Records Matching
* Triggers
  + Triggers are used to trigger the glue jobs in Different ways.
  + It can be triggered by Manual, Schedule, On Job Completion Event. Multiple Jobs can be triggered in parallel or sequentially by triggering them on a job completion event.
  + Using triggers, you can design a chain of dependent jobs and crawlers
* Workflows
  + Blueprints

Dynamic Dataframe

* A Dynamic DataFrame is like a DataFrame, except that each record is self-describing, so no schema is required initially.
* You can use Dynamic Frame when Data that does not conform to a fixed schema.
* You can also convert the Dynamic Frame to DataFrame using toDF()
* Dynamic frames provide a set of advanced transformations for data cleaning and ETL.

Redshift

What is redshift.?

* Redshift is a data warehouse service which is designed to handle large volume of data and complex queries.

Concepts

* Cluster, Leader Node, Compute Node, Node Slices (Each compute node in Redshift is divided into slices)
* MPP
* Columnar Storage
* Distribution Style and Distribution Key
* Sort Key
* Redshift Spectrum
* Materialized Views and Late binding views.
* How to Share Redshift data
* Performance Tunning

Why we go for redshift?

Pros/features and Cons of Redshift?

Architecture of Redshift.?

Components

* Client Applications
  + Client application can communicate with various ETL tools and custom applications through JDBC and ODBC drivers.
* Leader Node
  + Leader node manages communication between Client applications and Compute nodes.
  + Leader node is responsible for parsing the queries and determining Execution plan.
  + Leader nodes know where table data is distributed in Compute nodes, based on that it sends the query to appropriate compute nodes and get the result.
* Compute Node
  + Compute node is responsible for execution of queries, building results sets and sends result back to Leader node.
  + Each Compute node is a independent machine that is having own storage, memory and computational capacity.
  + Compute nodes are designed for Horizontal scalable. Compute nodes are added when storage and CPU demand is required.
* Databases
  + Redshift cluster will have one or many databases. Databases are stored in Compute nodes and distributed across multiple nodes.

Architecture/Execution flow

* Client will submit the query request to the Leader node through JDBC/ODBC connection.
* Leader node will split the query into Multiple Small queries and assign that into Compute nodes.
* Compute node again split that into small queries and assign them into Node Slices.
* Node Slices will run the query on top of the Storage and fetch the result and give it to Compute node. Compute node will combine all the data and give it to leader node.
* Leader node will combine all records and send back to Client.

Limitations of Redshift.?

* Scale limitations
  + Redshift is designed for large-scale data processing, there are still limits on the size of individual nodes and the total number of nodes in a cluster. This can be a constraint for extremely large datasets or high-concurrency workloads.
* Real-time data processing:
  + Redshift is designed for batch-oriented processing and may not be the best solution for real-time data analytics.
* Limited support for complex transactions:
  + It is not well-suited for OLTP (Online Transaction Processing) workloads.
* Concurrency limits:
  + While Redshift is designed to handle concurrent queries, Heavy concurrent workloads may experience contention and affect overall system performance.
* Backup and restore limitations:
  + While Redshift provides automated backup and restore functionality, the granularity of backup options may not be as fine-tuned. Additionally, the restore process may take some time, impacting recovery time objectives.

How to load data into Redshift.?

* Redshift Data Transfer Service
  + Redshift Data Transfer Service helps to transfer data from an on-premises data warehouse to Redshift.
* Glue
* Data Pipeline
  + We can use Data Pipeline to schedule and manage the data transfer process into Redshift.
* Redshift Spectrum
  + Redshift Spectrum to query data directly from data stored in Amazon S3 without the need to load it into Redshift.
* S3 Copy
  + If your data is already in Amazon S3 and doesn't require complex transformations, you can use the COPY command directly from S3 without an intermediate staging table:

A white text on a black background

Description automatically generated

* Third-Party ETL Tools
  + Apache NiFi, Talend, or Stitch.
* Custom Scripting
  + Python

Types of Tables

* Regular Tables
* External Tables
* Temporary Tables
  + Temporary tables are created and exist only for the duration of a session or a transaction. They are useful for storing intermediate results during complex queries or for breaking down a complex task into smaller steps.



* Partitioned Tables
  + Partitioned tables are tables that are logically divided into smaller and partitioned based on a specified column.



Types of Views

* Regular views
  + Simple Views are ideal for small datasets and infrequent querying
* Materialized Views
  + Materialized Views are like Regular Views, but they store the results of the SQL query in a physical table rather than as a virtual table.
  + The view can be refreshed on a regular schedule to ensure that has updated records.
  + Materialized Views are ideal for large datasets and frequent querying

A screenshot of a computer screen

Description automatically generated

* Late-Binding Views



* Note that the placeholders {{table1}}, {{table2}}, and {{value}} are used in the SQL query, and must be replaced with actual values when the view is queried
* the WITH NO SCHEMA BINDING clause specifies that the view isn’t bound to the underlying database objects, such as tables and user-defined functions.
* Use case - Late-Binding Views are used where the underlying data is complex and frequently changing.

Distribution Styles

* Distribution Style helps how data is distributed across the nodes in the Redshift cluster.
* EVEN Distribution: Data is distributed evenly across all nodes.
  + EVEN distribution is appropriate when a table doesn't participate in joins.
* KEY Distribution: Data is distributed based on a specific column (the distribution key).
* ALL Distribution: A full copy of the table is stored on each node.
  + ALL distribution is appropriate only for relatively slow-moving tables; that is, tables that are not updated frequently or extensively.
* AUTO - Redshift assigns an optimal distribution style based on the size of the table data. For example, if AUTO distribution style is specified, Amazon Redshift initially assigns the ALL-distribution style to a small table. When the table grows larger, Amazon Redshift might change the distribution style to KEY, choosing the primary key (or a column of the composite primary key) as the distribution key. If the table grows larger and none of the columns are suitable to be the distribution key, Amazon Redshift changes the distribution style to EVEN.



How to share data to others from Redshift.?

* Redshift Data Sharing
  + Redshift Data Sharing allows you to share live, read-only data with other AWS accounts. You can grant access to specific schemas or tables, and consumers can query the shared data using their own Redshift clusters.
* Steps:
  + Enable Data Sharing on your Redshift cluster.
  + Share specific schemas or tables with the desired AWS accounts.
  + Consumers can then connect to your cluster and query the shared data.
* S3 Data Sharing
  + Store your Redshift data in Amazon S3 and share it directly with others.
  + Steps
    - Export data from Redshift to Amazon S3 using the UNLOAD command or a data pipeline.
    - Share the S3 bucket or specific folders with other AWS accounts
    - Consumers can access the shared S3 data using tools like Athena, Redshift Spectrum, or directly through the S3 API.

Export to CSV/Parquet Files

* Export data from Redshift tables to CSV or Parquet files, and share these files with others
* Steps
  + Use the UNLOAD command to export data to CSV files in an S3 bucket.
  + Share the S3 bucket or download the files and share them through other means.

Materialized Views

* Create materialized views in Redshift and then share the views with others
* Steps
  + Create materialized views using SQL queries
  + Share the SQL statements for creating the materialized views or provide access to the views

APIs and Applications

* Build APIs or applications that query Redshift data and serve the results to authorized users
* Steps
  + Develop an API or application that connects to Redshift.
  + Implement user authentication and authorization mechanisms.
  + Users can interact with the API or application to access the required data

Performance tuning.? / Best Practices?

* Use appropriate distribution key and Style based on the requirement.

A close up of a text

Description automatically generated

* Use Sort key for Sorting records of the table to improve the performance
* Data Loading Best Practices
  + Use the COPY command for efficient data loading from Amazon S3
  + Run the VACUUM and ANALYZE commands periodically to reclaim space and update statistics.
* Query Optimization
  + Use the EXPLAIN command to analyze query plans and identify areas for optimization
* Redshift Advisor
  + Redshift Advisor to get recommendations for optimizing your cluster, including suggestions for table design, distribution, and queries.
* Use Redshift Spectrum for External Tables

Data Pipeline

* Data pipeline is a ETL Service which helps to perform Extract, Transform, Load data between different AWS compute and storage services (S3, RDS, DynamoDB,and EMR), as well as on-premises data sources, at specified intervals.

How it works.?

Step 1: Create Pipeline definition file which contains the Business logics and Transformations to be performed during the data Movement.

Step 2: Upload Pipeline definition file to Pipeline and Activate the Pipeline.

Step 3: Pipeline Schedules and Runs the Tasks By Creating EC2 instances to perform the activities.

* Task Runner Pools for the tasks and perform those Tasks.
* For Example, Task Runner could copy log files to Amazon S3 and launch Amazon EMR clusters.
* Task Runner is installed and runs automatically on resources created by your pipeline definitions.

Features

* Distributed
* Fault Tolerant (Automatically Retries Failed tasks)
* Providing options for Scheduling and Monitoring
* Low Expensive Cost.

Limitations

* Data pipeline is designed to do Data movement between AWS storage services. Not suitable for different 3rd Party services
* It is a Hard to Setup and Configurations on the Compute Resources
* Defining Complex pipelines are difficult for Beginner, It will be easy in other Workflow services like Airflow.

AWS data Pipeline Components

Pipeline Definition

* How to Communicate Business logics to Data Pipelines

Data Nodes

* Names, Locations, formats of your Source and Target Data sources like S3, DynamoDB, RDS, Redshift
* SqlDataNode: An SQL database and table query that represents data for a pipeline activity to use.
* DynamoDBDataNode: A table that contains data for EmrActivity or HiveActivity to use.
* RedshiftDataNode: A Redshift table that contains data for RedshiftCopyActivity to use.
* S3DataNode: An Amazon S3 location that contains files for a pipeline activity to use.

Activities

* Actual work that is being performed on the data.
* Transform your data. Ex: Have to run any queries on your beloved data.
* CopyActivitiy: Used when data needs to be copied from one data node to another.
* EmrActivity: Activity for starting and running an EMR cluster.
* HiveActivity: Runs a hive query.
* HiveCopyActivity: Runs a pig script in the AWS EMR cluster.
* RedshiftCOpyActivity: Runs a copy operation to the Redshift table.
* ShellCommandActivity: For executing a Linux shell command or a script.
* SQLActivity: Runs an SQL command on supported databases. The data pipeline supports JDBC databases, AWS RDS databases, and Redshift.

Schedules

* Schedule these activities

PreConditions

* Must be satisfied before your activity get started.

Compute Resources

* Resources for an AWS Data pipeline are usually an EMR or an EC2 instance.

Actions

* used to Share the status of the data pipeline like sending notifications/Alarams

Pipeline

* Schedules and Run the tasks to perform the defined Activities
* Pipeline Components
* Instances
  + AWS Data pipeline runs the data pipeline where it compiles all pipeline components into create set of actionable instances. that instance contains all information about the task.
* Attempts
  + No of times the tasks to be executed on the instances.

Task Runner

* Polls AWS Data pipeline for tasks and then performs those tasks.

Pipeline Schedules the tasks 🡪Task runner polls the tasks 🡪 Task runner reports the task is done

How Data Pipeline works

A diagram of a data pipeline

Description automatically generated

Example of Data Pipeline

A diagram of a data pipeline

Description automatically generated

Lake Formation

Introduction

* Lake formation helps to simplify the process of Setting up and managing the data lakes. Data lakes are centralized repositories for storing and analyzing large volume of raw and processed data.

Why we go for Lake formation.?

* If we want to build a data lake with Secure manner, grant and revoke access through simplified approach, easy way of sharing data to other account or end users. then we need to go for Lake formation.

Internally Connected

S3 🡨 🡪 Glue (Jobs, Data Catalog) 🡨 🡪 Athena 🡨 🡪 IAM

Permissions

Grant, Revoke, View permissions for the different IAM roles.

Options

-

- Data filters are used to apply column level security, Row level security for the data lake resources to the End users.

- Data Sharing - Sharing data to the End users thorugh Redshift and HMS which data resides in S3.

- Crawlers - Use Glue Crawlers to ingest and update metadata from sources to data Catalog.

Concepts

* Data Catalog
  + It is a centralized metadata repository that stores information about the data assets in the data lake. Lake Formation uses Glue Data Catalog as its underlying data catalog service.
* Blueprint
  + Blueprint helps to ingest data from multiple data sources to data lake either bulk load or incremental load. It will internally creates glue jobs to pull data from sources to data lake and it will catalog the data automatically.
* Workflow
  + Workflows are instances to execute Blueprint jobs.
* Metadata Crawling
  + Metadata crawling is the process of automatically discovering and extracting metadata from data sources, such as databases and files. AWS Glue, integrated with Lake Formation, performs metadata crawling to populate the data catalog.
* A data lake resource refers to the various components within the data lake, such as databases, tables, and objects. Access controls and permissions can be applied at the resource level.

Pros of Lake formation

* Simplified Data Lake Setup
* Integrated Data Catalog
  + Lake formation integrates with Glue Data Catalog. So that User can explore data through Catalog.
* Fine-Grained Access Control
  + Admin can define and enforce permissions at various levels, such as databases, tables, and columns.
* Cross-Account Access
  + Allows users from different AWS accounts to access and interact with the data lake resources
* Blueprints for Automation and Workflows

Cons of Lake formation

* Service Dependencies
  + Lake formation is tightly copupled with Glue features such Jobs, Data catalog and workflows. So, Users should be aware of the dependencies and potential limitations associated with AWS Glue.

How to do?

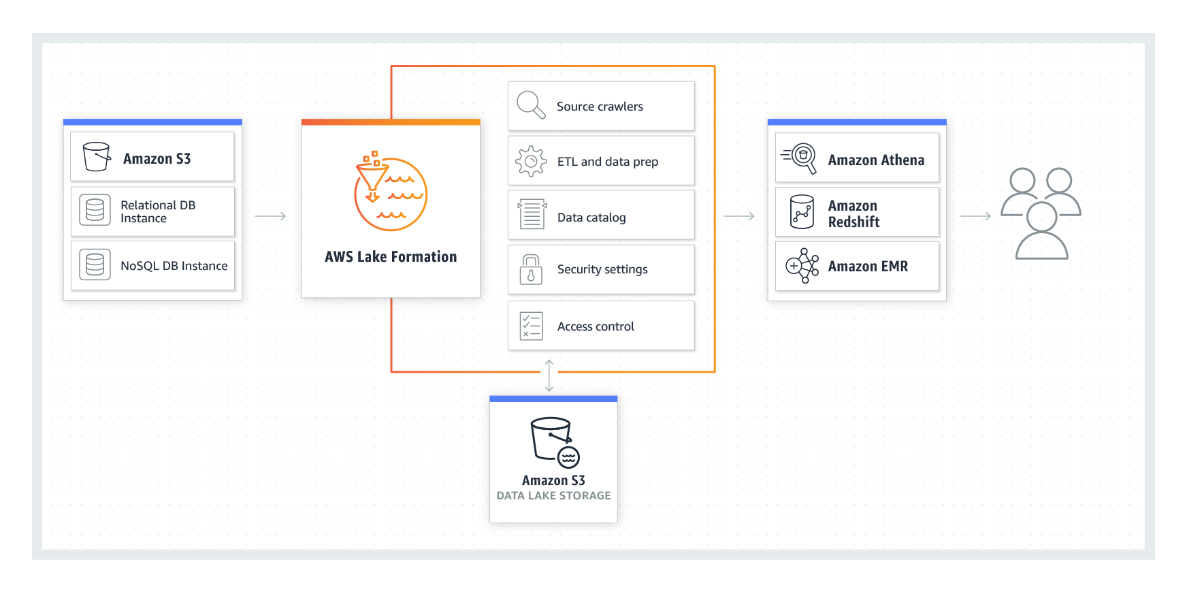
* Register S3 Buckets and Path where your data lake will reside.
* Use Jobs option to Build Glue ETL to ingest, cleanse, transform, and organize the raw data.
* Use Crawlers to Crawl metadata about data sources and data in the data lake.
* Define Data Access Control to Metadata and Data resides in Data lake.

Features of Lake formation

* Data Lake Setup
  + Using Lake formation, we can create a data lake by providing tools for setting up storage in S3 and configuring data ingestion pipelines.
* Data Ingestion
  + Lake Formation supports easy data ingestion from various sources into the data lake. This includes batch and real-time data ingestion capabilities.
* Data Catalog
  + Lake Formation includes a centralized data catalog that organizes metadata about your data. The catalog provides a unified view of data assets
* Fine-Grained Access Control
  + Lake Formation allows you to set up fine-grained access control policies for data stored in the data lake. This includes defining access permissions based on users, groups, or roles.
* Integration with AWS Services (Glue and Athena)
  + Lake Formation integrates with other AWS services, such as AWS Glue for data cataloging and ETL (Extract, Transform, Load) processes, and Amazon Athena for querying data using SQL.
* Data Row and Column-Level Security
  + Lake Formation supports row and column-level security, allowing you to control access to specific data rows or columns based on user roles.
* Data Lake Governance
  + Lake Formation helps in enforcing governance policies for data in the lake
* Data Lake Cross-Account Sharing
  + Lake Formation supports cross-account sharing of data, allowing you to share data across different AWS accounts securely.

Workflow for Lake formation

* Setup (Create Data Lake)
  + Create an AWS Lake Formation Data Lake, specifying the S3 bucket where your data will be stored.
* Ingest Data
  + Ingest data into the lake using various methods, such as direct uploads, AWS Glue jobs, or other data transfer tools.
* Data Cataloging
  + Use AWS Glue to catalog the data, defining schema and metadata for better data discovery and management
* Access Control
  + Define fine-grained access control policies using Lake Formation to control who can access the data and what actions they can perform.
* Query and Analysis
  + Use Athena or other query tools to analyze and query the data stored in the lake.



A screenshot of a computer

Description automatically generated

Kinesis

Introduction

* Kinesis is a Streaming data Analytical Service which helps to Collect, process and Analyze the Real time data, Streaming data.

Features

* Fully Managed 🡪 Kinesis is fully managed and runs your streaming applications without infrastructure Management.
* Scalable 🡪 Kinesis can handle any amount of streaming data and process data from hundreds of thousands of sources with very low latencies.

Kinesis Capabilities

Kinesis Data Streams

* Kinesis Data Streams is the real-time data streaming service in Kinesis
* It helps to Continuously capturing multiple gigabytes of data every second from multiple sources.
* It will be ideal choice for developers involved in developing custom applications or streaming data according to special needs.

Features

* Going to write custom code
* Data storage (Retention Period) for 1 to 7 days, replay capability, multi consumers.
* distributed across 3 Availability Zones.
* User must manage scaling (shard splitting/merging)
* Use with Lambda to insert data in real-time to Elasticsearch
* Producers 🡪 need to write a code for Producer. Supports SDK, Kinesis Agent, KPL.
* Data Sources 🡪 Any data source (servers, mobile devices, IoT devices, etc) that can call the Kinesis API to send data.
* Consumers 🡪 Kinesis Data Analytics, Amazon EMR, Amazon EC2, AWS Lambda
* Use cases 🡪 Log and event data collection, Real-time analytics, Mobile data capture, Gaming data feed

Kinesis Video Streams

Kinesis Data Firehose

Concepts

* It is Data transfer Service which helps to Capture, transform, and deliver streaming data into data lakes, data stores, and analytics services.
* Firehose handles loading data streams directly into AWS products for processing
* Fully managed, send to S3, Splunk, Redshift, Elasticsearch
* Serverless data transformations with Lambda
* Automated Scaling
* No data storage
* Kinesis Firehose is used to LOAD streaming data to a target destination (S3, Elasticsearch, Splunk, etc).
* You can also transform streaming data (by using Lambda) before loading it to destination. Data from failed attempts will be saved to S3.

Producers 🡪 need to write a code for Producer. Supports SDK, Kinesis Agent, KPL.

Data Sources 🡪 Any data source (servers, mobile devices, IoT devices, etc) that can call the Kinesis API to send data.

Consumers 🡪 Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, generic HTTP endpoints, Datadog, New Relic, MongoDB, and Splunk

Use cases 🡪 IoT Analytics, Clickstream Analytics, Log Analytics, Security monitoring

Kinesis Data Analytics

* Transform and analyze streaming data in real time with Apache Flink.

Data Sources 🡪 Amazon MSK, Amazon Kinesis Data Streams, servers, mobile devices, IoT devices, etc.

Data Consumers 🡪 Analysis results can be sent to another Kinesis stream, a Kinesis Data Firehose delivery stream, or a Lambda function.

Use cases 🡪 Streaming ETL, Real-time analytics, Stateful event processing

A screenshot of a computer screen

Description automatically generated

AWS CI-CD Process

What is CI-CD

* It is process of store your code and build, Test and deploy your code automatically to other environments.
* It would require few Manual approvals required to promote to next stage.

Services Available in AWS

* Code Commit 🡪 Storing our code.
* Code Build 🡪 Building and Testing your code.
* Code Deploy 🡪 Deploying the code to EC2 instances (Not Elastic Beanstalk)
* Code Pipeline 🡪 Automating pipeline from Code to Elastic Beanstalk
* Code Star 🡪 Manage Software development activities in one place.
* Code Artifact 🡪 Store, Publish, and share Software packages.
* Code Guru 🡪 Automated Code reviews by ML

What is Continuous Integration (CI)

* Developers pushing Code to Code repository often (Eg: GitHub, BitBucket, CodeCommit)
* Build Server (Build and test) fetch the code ASAP and do Build and Test. (Jenkins, CodeBuild etc,)
* Share the Feedback to development about the tests whether they are passed/Failed.
* Developer 🡪 Push Code 🡪 Code Repo 🡪 Fetch Code 🡪 Build Server 🡪 Build and Tests results 🡪 Developer

What is Continuous Delivery (CD)

* Build Server push the build to deployment server and deployed to Multiple Application servers
* Developer 🡪 Push Code 🡪 Code Repo 🡪 Fetch Code 🡪 Build Server (Build and Test) 🡪 Deploy Every passed build 🡪 Deployment Server 🡪 Different Application Server.
* CD Tools (Jenkins CD, Code Deploy, Spinnaker etc)

What is Code Pipeline

* To Orchestrate Everything, we can use Code Pipeline.

What is Code Commit

* Code is committed to Online
* Code backup.
* Git Repositories can be expensive
* The industry includes, Github, GitLab, BitBucket

Benefits of Code Commit

* Private Git Repositories (Particular Region, VPC)
* No Size limit on Repositories (Scale Seamlessly)
* Fully Managed, Highly Available
* Code only on Your AWS Cloud account (Increased Security and Compliance)
* Security (Encrypted, Access Control via IAM)
* Integrated with Jenkins, CodeBuild, Other CI Tools.

Security

* Authentication
  + SSH keys (AWS Users can configure SSH keys into Their IAM Console
  + HTTPS 🡪
* Authorization
  + IAM policies to manage
* Encryption
  + Repos are automatically encrypted by AWS KMS
* Cross Account Access
  + Don't share SSH keys to other account for access, Use IAM policies to Manage it.

Demo

* Supports both SSH keys and HTTPS Git Credentials for AWS Code Commit
* Generate HTTPS Git Credentials from IAM
* Install Git and Open Git CLI
* git clone https://repo link

CodeBuild (buildspec.yaml and CodeBuild Agent)

* Source (CodeCommit, GitHub, S3, ECR)
* Build instructions (Code file buildspec.yaml or insert manually in Console)
* Output logs are stored in Cloudwatch logs and S3
* use CloudWatch Metrics to Monitor the Build statistics
* Use CloudWatch Events to detect the failure build and trigger Notifications.
* Use CloudWatch Alarms to Notify if you need any threshold for Failures.

CodeBuild - How it works

* Code along with buildspec.yaml will be pushed to Code Commit.
* CodeBuild will fetch that buildspec.yaml file and execute the instructions from yaml file in CodeBuild Own Containers.

CodeBuild Agent

* Execute Yaml file locally in your docker and for this use CodeBuild Agent.

CodeBuild - Inside VPC

* By default, CodeBuild containers are launched outside your VPC. It cannot access your VPC services
* You need to give your VPC details to CodeBuild to execute containers inside your VPC.
  + VPC id
  + Subnet ID
  + Security group ID.

Code Deploy (appspec.yaml and Code deploy Agent)

* Used to deploy your code to EC2 instances.
* There are several other open-source tools to handle deployments (Terraform, Ansible, Chef, Pubbrt)
* Code deploy Agent should run on all EC2 instances where you want to deploy your code.
* Developers Commits the code and appspec.yaml file to Code Commit and trigger Code deploy
* Code deploy Agents keep polling Code deploy for any work to do.
* Once Code Agent receives request from Code deploy, then Download files from S3/GitHub and appspec.yml.
* Code deploy agent will execute the instructions in appspec.yml file into EC2 instances.

Code Pipeline

* It is a Visual workflow tool to orchestrate yout CICD
* Source 🡪 Code resides in CodeCommit, ECR, S3, GitHub, BitBucket
* Build 🡪 Codebuild, Jenkins, CloudBees, teamCity
* Test 🡪 CodeBuild, AWS DeviceFarm, 3rd Party tools
* Deploy --> CodeDeploy, Elastic Beanstalk, CLoudformation, ECS, S3 etc.

Consists of Stages:

* Build 🡪 test 🡪 Deploy 🡪 Load Testing
* Manual Approvals can be defined at any stage.

Code Pipeline – Artifacts

* Each Pipeline stage can create Artifacts.
* Artifacts (input/Output) stored in S3 bucket and passed to next stage.

Demo

* Create CodePipeline
* Stages
  + Add Source stage (CodeCommit, GitHub, S3, ECR)
  + Add Build Stage (CodeBuild, Jenkins)
  + Add Deploy stage (Cloud formation, Codedeploy, EBeans, ECS, S3)
* New Stage
  + ActionGroups
    - Actions - Action provides

Application Services – Step Function

Introduction

* Step Functions is a fully managed, serverless orchestration service which helps to coordinate multiple aws services. with step function, you can define and execute workflow as series of steps involved in workflow.
* Step Functions is a serverless orchestration service which helps to coordinate flexible workflow flows.

Concepts

* State Machines (Workflows)
  + Workflow in step function is called as State machine. Each step in the workflow is called as State.
* States (Each Step in the workflow)
  + Each step in the workflow is called as State. It has a various type of states, including task states, choice states, pass states, parallel states, and more.
* Execution History
  + Step Functions maintain a detailed execution history for each workflow. which contains input and output data, and any errors that occurred during execution.
* Integration with AWS Services
  + Step Functions can integrate with a variety of AWS services, such as S3, Lambda, Glue, Redshift, DynamoDB and more.
* Serverless Execution
  + you don't need to provision or manage servers. It automatically scales to handle the execution of workflows, and you pay only for the actual state transitions and the time your code is running.
* Visual Workflow Editor
  + Step Functions provides a visual workflow editor in the Console, which allowing you to design and edit state machines using a graphical interface.

Use cases

* Microservices Orchestration
  + Coordinate the execution of microservices in a distributed application
* Data Processing Pipelines
  + Create and manage data processing workflows that involve multiple AWS services.

Different States in Step functions

* Task
  + Single unit of work in the workflow.
* Choice
  + It will add branching logic to your workflow.
* Pass
  + Pass their output as Input to the next state
* Wait
  + A Wait state adds a delay to your workflow. You can specify a fixed time to wait or use a timestamp to wait until a specific time.
* Succeed
  + stops the execution of the state machine successfully. you can use it as an end state for a successful workflow.
* Fail
  + stops the execution of the state machine with a specified error. You can use it as an end state for a failed workflow.
* Parallel
  + A Parallel state enables you to execute multiple task of your workflow in parallel. Each branch is specified as a separate state.

Core Concepts

* The workflows which we build with Step Functions are called state machines, and each step of your workflow is called a state.
* Task perform work or either coordinating with another AWS service or another application.
* "Pass States" will pass their output as Input to the next state. we can delay execution using "wait states"
* "Parallel" states begin with Multiple branches of the Execution at a same time, such as running a multiple lambda function at once.
* "Choice state" will add branching logic to your state machine and make decisions based on their input.
* "State Transition" - When you execute your state machine, each move from one state to another state is called as State transition.

SQS (Simple Queue Service)

* SQS is a Managed queuing service which is used to manage the messages in the queue.

How it works.?

* Producer (One or more Producers) sends the messages to the queue. Consumers will consume the messages, process it and deletes the messages in the queue.

Concepts

* Producer
* Consumer
* Retention Period
* Visibility Timeout
* Delivery Delay Queue
* Dead Letter Queue
* Long Polling
* Default Retention period of the message in the Queue is 4 days and Max. 14 days. Limitation of 256KB per message.
* Message order will be maintained.

SQS Producer

* Producer can send message to the SQS queue through SendMessage API.
* The message will be persisted in SQS until Consumer deletes it.
* Message Retention: 4 days by default and 14 days as Max.

SQS Consumer

* Consumers (Applications running on the EC2, On-Prem Servers, lambda)
* Consumer can poll/Receive for messages (Up to 10 messages at a time)
* Process the Messages (Example: Insert the message into the RDS)
* Delete the message using Delete Message API.

SQS with Multiple EC2 instances

* Consumers can receive and Process the Message in Parallel.
* Consumers can delete the messages after processing.
* We can scale Consumers Horizontally to improve the throughput of the Processing.

SQS with ASG

* Consumers are EC2 instances which are running inside the ASG, It will Scale up/down based on Some Metrics.
* If the No of message to be polled is getting increased, CloudWatch Alarms will be Notified by CloudWatch Metrics (Approximate Number of message is configured already), Alarm will Increase/decrease the ASG.

Configurations

* Visibility Timeout (0 seconds to 12 Hours)
  + Once the Customer polled the message, The Visibility Timeout Period will start. During Visibility Timeout, the message will not be visible to other customers.
  + Default Visibility time out is 30 seconds, which means message to be processed in 30 Seconds.
  + After the Visibility Timeout period, the message is Visible in SQS. Needs to be processed Twice.
  + If Consumer don't want to Re-process the data, Use "ChangeMessageVisiblity" API to get more time.
  + If Time out period is too low, then we may get duplicates.
* Message Retention Period
  + How long the message would be retained in the queue.
* Delivery Delay Queue
  + Delay message (Consumers don't see the message Immediately) up to 15 minutes. Default 0 seconds.
  + Can override the default by using Delay Seconds parameter.
* Maximum Message size (1 kb to 256 kb)
* Receive message Wait time (Long Polling)
* Dead Letter Queue
  + If the Consumer fails to process the message with in the Visibility Timeout, The Message goes back to the queue.
  + We can set a threshold of how many times; the message can go back to the queue. After the "MaximumReceives" threshold is exceeded, the message goes into the Dead letter queue.
  + Send Undeliverable messages to the Dead Letter Queue.
  + Make sure to process the message in the DLQ before they expire. (Good to set retention period is 14 days)
* SQS Long Polling
  + When Customer is requesting the message from SQS queue, If none of the messages are there, then there is option for the waiting in the queue. This is called as long polling.
  + The Wait time can be 1 sec to 20 seconds.
  + Long Polling Can be enabled at the queue level or at API level by using "WaitTimeSeconds".
* SQS Extended Client.
  + Max. Size is 256KB, how do we send 1 GB of message?
  + Using SQS Extended client (Java library)
  + Producers send large file to the S3 and send Small metadata message to SQS, Consumer will receive that metadata file and take data from S3.
* SQS - Must know API
  + CreateQueue (MessageretentionPeriod), DeleteQueue.
  + PurgeQueue - Delete all message from the queue.
  + SendMessage (DelaySeconds), ReceiveMessage, DeleteMessage.
  + MaxNumberofMessages - default 1 and Max. 10 (for ReceiveMessageAPI)
  + Receive message Wait time (Long Polling)
  + ChangeMessageVisiblity - Change the message Timeout

SNS - Simple Notification Service

Introduction

* Simple Notification Service is a messaging service which helps to make easy to set up, operate, and send notifications from the cloud.

Concepts

* Publisher
* Subscriber
* SNS Topic

If you want to send one message to many receivers. There are two ways,

* Directly integration
* Using Pub/Sub model

Event producer only send messages to the SNS topic.

Many Event receivers (Subscriptions) as we want to listen the SNS topic notifications.

Subscribers can be,

* SQS
* Lambda
* Email
* SMS Messages
* Mobile Notifications.

SNS Integrates a Lot of AWS Services

* Many services can send data directly to the SNS for Notifications.
* CloudWatch (For Alarms)
* Autoscaling group Notifications
* Amazon S3 (Bucket events)
* CloudFormation (Upon Stage changes ==> Failed to Build etc..)

How to Publish

* Topic Publish
  + Create a Topic
  + Create a Subscription
  + Publish to the Topic
* Direct Publish (For Mobile apps SDK)

Security

* Encryption
  + In-flight Encryption
  + At rest encryption using KMS keys
  + Client-Side encryption if client wants to perform Encryption and Decryption by own.

Access Controls (IAM Policies)

- SNS Access Policies (Similar to S3 Bucket policies)

Migration and Data transfer Services

Migration – DMS (Data Migration Service)

Introduction

* It is a fully managed service which helps to migrate or replicate databases to and from the AWS Cloud with minimal downtime.
* It supports both homogeneous (e.g., Oracle to Oracle) and heterogeneous (e.g., Oracle to Amazon Aurora) migrations.
* The only requirement to use AWS DMS is that one of your endpoints must be on an AWS service. You can't use AWS DMS to migrate from an on-premises database to another on-premises database
* DMS can be used for a one-time migration or continuous replication using DMS’ change data capture.

Homogeneous Migration (Oracle to oracle)

* Homogeneous data migrations are serverless, which means that AWS DMS automatically scales the resources that are required for your migration.
* DMS homogeneous migrations do not require replication instances, you do not need to pay for storage. You only pay for the usage of AWS DMS service on an hourly basis.

Heterogeneous Migration (Oracle to PostgreSQL)

Features

* Homogeneous and Heterogeneous Migrations
* Minimize Downtime
  + DMS helps minimize downtime during migrations
* Continuous Data Replication
  + DMS supports ongoing replication for continuous data replication from the source to the target. This is useful for scenarios where you want to keep the source and target databases in sync.
* Schema Conversion
  + For heterogeneous migrations, AWS DMS includes a schema conversion tool that helps convert schema definitions and code between different database engines.
* CDC (Change Data Capture)
  + DMS uses Change Data Capture to capture changes in the source database. This ensures that only incremental changes are migrated to the target, reducing the overall migration time.
* Security
  + It supports SSL encryption for data in transit, and you can configure encryption for data at rest using AWS Key Management Service (KMS).

Workflow of AWS DMS Migration

Step 1. Create a Replication Instance

- Provision a replication instance, which is an EC2 instance used to run the replication tasks.

Step 2. Define Source and Target Endpoints

- Define the source and target database endpoints. This includes specifying the connection details for the source and target databases.

Step 3. Create a Replication Task

- Define a replication task that specifies the migration or replication settings. This includes selecting tables, defining transformation rules, and configuring other migration options.

Step 4. Run the Replication Task

- Start the replication task to begin the migration process. DMS will capture changes from the source and apply them to the target.

Step 5. Monitor and Troubleshoot

- Monitor the progress of the migration using CloudWatch. If any issues arise, you can use CloudWatch logs and DMS event notifications for troubleshooting.

Use Cases

* Database Migrations
* Continuous Replication
  + Keep databases in sync for real-time analytics or reporting
* Data Warehousing
  + Load data into Amazon Redshift for analytics and business intelligence
* Global Database Distribution
  + Distribute data globally for improved performance and availability.
* Disaster Recovery
  + Set up disaster recovery solutions by replicating data to a standby database

Steps to Migrate data from Oracle to S3 using DMS

* Setup Oracle database as a Source for AWS DMS
* Setup S3 as a Target for DMS
* Create DMS Endpoints (Source and Target Endpoints) and Launch DMS Replication Instance
* Create DMS Migration Task
* Use Athena for data Validation

Schema Conversion Tool (SCT)

* SCT helps to automatically convert the database schema of your source database into a format compatible with your target Amazon RDS instance
* AWS SCT can copy database schemas for homogeneous migrations and convert them for heterogeneous migrations.
* Once a schema has been created on an empty target, depending on the volume of data and/or supported engines, either AWS DMS or AWS SCT are then used to move the data. AWS DMS traditionally moves smaller relational workloads (<10 TB), whereas AWS SCT is primarily used to migrate large data warehouse workloads. AWS DMS supports ongoing replication to keep the target in sync with the source; AWS SCT does not.

Data Sync

Introduction

* Data Sync is used to transfer large amounts of data between on-premises storage, S3, EFS (Elastic File System).
* It is designed to simplify and accelerate data transfer workflows, making it easier to move data to and from AWS.

Features

* Data Transfer Acceleration
  + DataSync is mainly built for accelerate the data transfers
* Supported Data Sources and Destinations
  + DataSync supports transfers between on-premises storage, S3, EFS, and FSx. It allows for flexibility in choosing source and destination locations.
* Incremental Transfers
  + DataSync supports incremental transfers, meaning it only transfers the changes made to files, reducing the amount of data transferred during subsequent sync operations.
* Scheduling and Automation
  + Transfers can be scheduled and automated, allowing you to set up recurring data transfer workflows
* Integration with AWS Key Management Service (KMS)
  + DataSync integrates with KMS for encrypting data during transfer.
* Parallel Data Transfer
  + DataSync can transfer multiple files or objects in parallel, improving overall transfer efficiency
* Monitoring and Logging
  + DataSync provides monitoring capabilities through CloudWatch, allowing you to track the status of data transfer tasks.

Use Cases

* Data Migration
  + Migrating large datasets from on-premises storage to AWS storage services.
* Content Distribution
  + Distributing large media files or datasets to multiple locations for content delivery or collaboration
* Data Sync Between Storage Systems
  + Syncing data between different on-premises storage systems or between different AWS storage services
* Hybrid Cloud Workloads
  + Supporting hybrid cloud architectures by enabling data transfers between on-premises and the AWS Cloud.
* Data Archiving
  + Archiving large datasets to Amazon S3

How to Use AWS DataSync

* Create a DataSync Agent
  + Deploy a DataSync agent on your on-premises server or as an EC2 instance.
* Create a Location
  + Define source and destination locations, specifying the type of storage and access credentials.
* Create a Task
  + Create a DataSync task that specifies the source and destination locations, transfer options, and schedule if needed.
* Run the Task
  + Start the DataSync task to initiate the data transfer. Monitor the progress and check logs for any issues.
* Automate and Schedule
  + Optionally, set up automated and scheduled tasks for recurring data transfers.

Data transfer Options in AWS

Online

* VPN
  + You can connect securely between your data centers and AWS
  + Suitable for Small Data transfer and Connectivity
  + quick to set up and cost-efficient
* Direct Connect
  + provides a dedicated physical connection to transfers between data centers and AWS
  + Suitable for large data transfer
  + needs time to setup and not cost-efficient solution
* AWS DataSync
  + AWS DataSync is a data transfer service which helps to automatically move data between On-prem Storage, S3, EFS.
  + You can use DataSync to transfer data at speeds up to 10 times faster than open-source tools.
  + Suitable for One time data movement, recurring data transfer, Automated Replications.
* AWS S3 Transfer family
  + It is a fully managed support for file transfers directly into and out of Amazon S3.
  + there is no infrastructure to buy and set up.
  + Suitable for media uploads, backups, local files.

Offline

* Snowfamily