Change Data Capture (CDC)

What is Change Data Capture? Example and tools to do Change Data Capture

Change Data Capture (CDC) is a technique used to capture changes made to a database so that they can be propagated to other systems in near real-time. CDC is often used in data integration, data replication, and data warehousing scenarios to ensure that the target systems have up-to-date and accurate data.

CDC works by monitoring the transaction log of a database and capturing changes made to the data. The captured changes can then be transformed and loaded into another system, such as a data warehouse or reporting database. CDC can capture changes at the row level, which means that only the changed data is propagated, reducing the amount of data transferred and increasing efficiency.

There are several tools and technologies that can be used to implement CDC, including:

1. Microsoft SQL Server Change Data Capture - This feature is built into Microsoft SQL Server and can be used to capture and propagate changes made to SQL Server databases. It provides a reliable and efficient way to capture changes and propagate them to other systems.
2. Oracle GoldenGate - This tool is used to capture and replicate changes made to Oracle databases. It provides real-time data integration and can capture changes made to databases on-premises or in the cloud.
3. Apache Kafka - Kafka is a distributed streaming platform that can be used for real-time data integration, including CDC. It can capture changes made to databases and propagate them to other systems in real-time, making it a popular choice for data streaming use cases.
4. Attunity Replicate - This tool is used for data integration, replication, and data warehousing scenarios. It can capture changes made to databases and propagate them to other systems in near real-time, and it supports a wide variety of data sources and targets.

An example of using CDC might be capturing changes made to a customer database, such as new orders or changes to customer details, and propagating those changes to a data warehouse for analysis and reporting purposes. CDC can also be used to keep databases in sync across different systems or to replicate data across different environments.

What programming language to use in CDC?

The programming language used to implement CDC (Change Data Capture) depends on the technology being used to capture and process changes made to a database.

For example, if you are using the built-in CDC feature in Microsoft SQL Server, you can use SQL scripts or T-SQL (Transact-SQL) code to capture changes and process them.

If you are using a tool like Apache Kafka, you can use programming languages such as Java, Python, or Scala to implement the CDC solution.

Similarly, if you are using a cloud-based CDC solution like AWS DMS or Google Cloud Dataflow, you can use programming languages such as Java, Python, or Go to implement the solution.

Overall, the choice of programming language depends on the technology being used and the specific requirements of the CDC

Techniques to implement CDC?

There are several techniques to implement Change Data Capture (CDC) in a data integration process. Here are a few commonly used techniques:

1. Database Triggers: One of the simplest ways to implement CDC is through the use of database triggers. A trigger can be set up on a table to automatically capture any changes to the data in the table. When a change occurs, the trigger can be configured to write the change details to a separate CDC table or a change file.
2. Change Tracking: Many relational database management systems (RDBMS) provide built-in mechanisms to track changes to data. For example, Microsoft SQL Server has a feature called Change Tracking that captures changes to selected tables in the database. Change Tracking can be used to identify new, updated, and deleted rows.
3. Log-based CDC: Log-based CDC involves capturing changes to a database through the transaction log. This method requires a more complex setup but offers more robust and reliable CDC. Log-based CDC can be implemented using tools such as Apache Kafka, Debezium, or Oracle GoldenGate.
4. Third-party Tools: There are many third-party tools available in the market that can be used to implement CDC. These tools offer features such as automated change capture, data mapping, data transformation, and data delivery. Some examples of CDC tools are Attunity Replicate, HVR, and Qlik Replicate.

The choice of technique for implementing CDC will depend on factors such as the type of data source, the complexity of the integration process, the level of data accuracy and completeness required, and the budget and resources available.

CDC setup in AWS

Change Data Capture (CDC) can be used in AWS S3 and Google BigQuery to capture changes made to databases and propagate them to other systems.

In both AWS S3 and Google BigQuery, CDC can be used to capture changes made to databases in real-time and propagate those changes to other systems. This can be useful for data warehousing, analytics, and reporting scenarios where up-to-date data is required for analysis and decision-making.

AWS S3 - To use CDC in AWS S3, you can use AWS Database Migration Service (DMS). DMS can capture changes made to databases, including AWS RDS, Aurora, and other databases, and replicate those changes to S3. To set up CDC with DMS, you will need to create a replication instance, source endpoint, and target endpoint. You can then configure DMS to capture changes made to the database using the CDC feature. The changes can be stored in S3 in Apache Parquet format, which is a columnar storage format optimized for analytics workloads.

> 👍 If you use Google BigQuery - To use CDC in Google BigQuery, you can use a tool like Alooma or Stitch Data. These tools can capture changes made to databases, including PostgreSQL, MySQL, and MongoDB, and replicate those changes to BigQuery. To set up CDC with Alooma or Stitch Data, you will need to create a data source connection to your database, select the tables to replicate, and configure the CDC settings. The changes can be stored in BigQuery in a format optimized for analytics workloads, such as columnar or partitioned formats.

Example of CDC in Python in AWS S3 using DMS (MySQL)

Here's an example of how to set up CDC in Python using AWS DMS and S3:

1. First, you'll need to set up AWS DMS and configure the source and target endpoints. In this example, we'll assume you have an AWS DMS endpoint set up to capture changes from a MySQL database.
2. Next, you'll need to create an S3 bucket to store the CDC data. You can use the AWS Management Console or the AWS CLI to create the bucket. Or can implement this example code

```
import boto3

# set up boto3 client for S3
s3 = boto3.client('s3', region_name='us-east-1')

# create an S3 bucket for storing CDC data
bucket_name = 'my-cdc-bucket'
s3.create_bucket(Bucket=bucket_name)

```

3. Once your bucket is set up, we need to set up a DMS replication instance and endpoint, and then create a CDC task. You can configure AWS DMS to write the CDC data to S3. In the AWS DMS console, go to the task settings and select the "Target" tab. Here, you can select "Amazon S3" as the target endpoint and configure the S3 settings (e.g. bucket name, folder structure, etc.).
4. After you've configured the S3 settings, you can start the data migration task to begin capturing changes from your MySQL database and writing them to S3.
5. Once the changes are captured, you can use Python to query and process the data in S3. Here's some sample code to get started:

```
import boto3
import pandas as pd
# Initialize a client object
s3 = boto3.client('s3', region_name='us-east-1')

# Define the bucket and folder names
bucket_name = 'my-bucket'
folder_name = 'my-folder'

# List all the objects in the folder
objects = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_name)['Contents']

# Loop through each object and load the CDC data into a dataframe
dfs = []
for obj in objects:
    obj_key = obj['Key']
    obj_body = s3.get_object(Bucket=bucket_name, Key=obj_key)['Body']
    df = pd.read_csv(obj_body, delimiter=',')
    dfs.append(df)

# Concatenate all the dataframes into one
df = pd.concat(dfs)

```

```
# Filter the data by date range
start_date = '2022-01-01'
end_date = '2022-03-01'
df = df[(df['timestamp'] >= start_date) & (df['timestamp'] < end_date)]

```

This code creates a client object to connect to S3, defines the bucket and folder names, and queries the folder to load the CDC data into dataframes. It then concatenates all the dataframes into one and filters the data by a date range.

You can then use Python libraries like Pandas and Matplotlib to analyze and visualize the data as needed.

Note that this is just a simple example and the specific implementation will depend on your AWS DMS settings, S3 bucket structure, and other factors.

Other way, Here is the explaination code that combine step 3,4,5 together : We create a replication instance, subnet group, and endpoint using the `create_replication_instance`, `create_replication_subnet_group`, and `create_endpoint` methods, respectively. We then create a CDC task using the `create_replication_task` method, specifying the source and target endpoints, replication instance, and table mappings file. The `CdcStartPosition` parameter is set to `TRUNCATE_BEFORE_UPDATE` to truncate the target table before updates, and `CdcStopPosition` is set to `LAST_RECORD` to stop the task after processing the last record in the source database.

```
import boto3

# set up boto3 client for DMS
dms = boto3.client('dms')

# create a replication instance
replication_instance_id = 'my-replication-instance'
dms.create_replication_instance(
    ReplicationInstanceIdentifier=replication_instance_id,
    AllocatedStorage=100,
    EngineVersion='3.3.1',
    ReplicationInstanceClass='dms.t2.micro',
    VpcSecurityGroupIds=['sg-12345678'],
    AvailabilityZone='us-west-2a',
    PubliclyAccessible=False
```

```
)

# create a replication subnet group
subnet_group_id = 'my-subnet-group'
dms.create_replication_subnet_group(
    ReplicationSubnetGroupIdentifier=subnet_group_id,
    ReplicationSubnetGroupDescription='My replication subnet group',
    SubnetIds=['subnet-12345678', 'subnet-23456789']
)

# create a replication endpoint
endpoint_id = 'my-endpoint'
dms.create_endpoint(
    EndpointIdentifier=endpoint_id,
    EndpointType='source',
    EngineName='aurora',
    EngineDisplayName='Amazon Aurora',
    Username='my-username',
    Password='my-password',
    ServerName='my-aurora-cluster.cluster-abcdefg.us-west-2.rds.amazonaws.com',
    Port=3306,
    DatabaseName='my-database',
    SslMode='none',
    ReplicationInstanceIdentifier=replication_instance_id,
    Tags=[
        {'Key': 'environment', 'Value': 'prod'}
    ]
)

# create a CDC task
task_id = 'my-cdc-task'
dms.create_replication_task(
    ReplicationTaskIdentifier=task_id,
    SourceEndpointArn='arn:aws:dms:us-west-2:123456789012:endpoint:my-endpoint',
    TargetEndpointArn='arn:aws:dms:us-west-2:123456789012:endpoint:s3-target',
    ReplicationInstanceArn='arn:aws:dms:us-west-2:123456789012:rep:my-replicatio
    MigrationType='cdc',
    TableMappings='file://table-mappings.json',
    CdcStartPosition='TRUNCATE_BEFORE_UPDATE',
    CdcStopPosition='LAST_RECORD',
    Tags=[
        {'Key': 'environment', 'Value': 'prod'}
    ]
)

```

Code implement for CDC in Python in AWS using AWS Lambda and CloudWatch (MySQL)

1. Create an Amazon RDS instance and a MySQL database.
2. Install the necessary Python libraries: boto3, mysql-connector-python, and logzero.

```
pip install boto3 mysql-connector-python logzero
```

3. Configure AWS credentials using boto3. You can use the following code to set up your credentials

```
import boto3
session = boto3.Session(profile_name='myprofile')
client = session.client('rds')

```

4. Create an S3 bucket to store the CDC logs

```
import boto3
session = boto3.Session(profile_name='myprofile')
s3 = session.client('s3')

bucket_name = 'my-cdc-bucket'
s3.create_bucket(Bucket=bucket_name)
```

5. Create a Lambda function that reads the changes made to the database and writes them to an S3 bucket.

```
import json
 import boto3
 import mysql.connector
```

```
from logzero import logger

def lambda_handler(event, context):
    # Get database connection details from environment variables
    host = os.environ['DB_HOST']
    user = os.environ['DB_USER']
    password = os.environ['DB_PASSWORD']
    database = os.environ['DB_NAME']

    # Create connection to the database
    connection = mysql.connector.connect(
        host=host,
        user=user,
        password=password,
        database=database
    )

    # Get a cursor to execute queries
    cursor = connection.cursor()

    # Get the latest change timestamp from the S3 bucket
    s3 = boto3.client('s3')
    latest_timestamp = s3.get_object(
        Bucket=os.environ['CDC_BUCKET'],
        Key='latest_timestamp.txt' )['Body'].read().decode('utf-8')

    # Get the changes made to the database since the latest timestamp
    cursor.execute(f"SELECT * FROM mytable WHERE modified_date > '{latest_times
    changes = cursor.fetchall()

    # Write the changes to an S3 bucket
    s3.put_object(
        Bucket=os.environ['CDC_BUCKET'],
        Key=f'{datetime.now().strftime("%Y-%m-%d-%H-%M-%S")}.json',
        Body=json.dumps(changes)
    )

    # Update the latest timestamp in the S3 bucket
    s3.put_object(
        Bucket=os.environ['CDC_BUCKET'],
        Key='latest_timestamp.txt',
        Body=datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    )

    logger.info(f"{len(changes)} changes made to the database")
```

6. Create a CloudWatch Events rule to trigger the Lambda function every time a change is made to the database.

```
import boto3
session = boto3.Session(profile_name='myprofile')
events = session.client('events')

rule_name = 'my-cdc-rule'
function_arn = 'arn:aws:lambda:us-east-1:123456789012:function:my-cdc-function'

events.put_rule(
    Name=rule_name,
    ScheduleExpression='cron(0/5 * * * ? *)', # run every 5 minutes
    State='ENABLED'
)

events.put_targets(
    Rule=rule_name,
    Targets=[
        {
            'Id': 'my-cdc-target',
            'Arn': function_arn
        }
    ]
 )

```

This code sets up a CloudWatch Events rule that triggers the Lambda function every 5 minutes. The Lambda function then reads the changes made to the database from the last run and writes them to an S3 bucket.

CDC in AWS using the boto3 library to connect to an Amazon RDS PostgreSQL database.

First, we need to set up an Amazon RDS PostgreSQL database and enable CDC for the desired tables. This can be done through the AWS Management Console or using the AWS CLI. Once CDC is enabled, we can use the boto3 library to create a connection to the database and retrieve the changes.

Here is an example code. This code uses the boto3 library to retrieve the database endpoint, port, and credentials from AWS, and the psycopg2 library to connect to the database and retrieve the changes from the CDC enabled tables. The changes can then be processed as needed.

```
import boto3
import psycopg2
 from psycopg2 import sql

 # Set up connection to AWS RDS PostgreSQL database
 client = boto3.client('rds')
 response = client.describe_db_instances(DBInstanceIdentifier='my-db-instance')
 db_endpoint = response['DBInstances'][0]['Endpoint']['Address']
 db_port = response['DBInstances'][0]['Endpoint']['Port']
 db_user = response['DBInstances'][0]['MasterUsername']
 db_password = 'my-db-password'

 # Connect to the database
 conn = psycopg2.connect(host=db_endpoint, port=db_port, dbname='my-db', user=db
 cursor = conn.cursor()

 # Get list of CDC enabled tables
 query = sql.SQL("SELECT table_name FROM information_schema.tables WHERE table_c
 cursor.execute(query)
 tables = cursor.fetchall()

 # Retrieve changes from CDC enabled tables
 for table in tables:
     query = sql.SQL("SELECT * FROM {} WHERE xmin::text::bigint > %s").format(sq
     cursor.execute(query, [0])
     changes = cursor.fetchall()

     # Process changes
     for change in changes:
         # Process change
         pass

 cursor.close()
 conn.close()

```

Bonus: Example of CDC in Python in Google Big Query using Stitch Data

Here's an example of how to set up CDC in Python using Stitch Data and Google BigQuery:

1. First, you'll need to sign up for Stitch Data and create a new data pipeline.
2. Next, you'll need to select your data source (e.g. MySQL, PostgreSQL, MongoDB, etc.) and configure the connection settings.
3. Once your data source is connected, you can configure the CDC settings. In the Stitch Data dashboard, go to the "Advanced" tab and select "CDC" to configure the CDC settings. Here, you can specify the tables to capture changes for, the frequency of updates, and other settings.
4. After you've configured the CDC settings, you can start the data pipeline to begin capturing changes from your database.
5. Once the changes are captured, you can use Python to query and process the data in Google BigQuery. Here's some sample code to get started:

```
from google.cloud import bigquery
# Initialize a client object
client = bigquery.Client()

# Define the dataset and table names
dataset_id = 'my_dataset'
table_id = 'my_table'
# Create a reference to the table
table_ref = client.dataset(dataset_id).table(table_id)
# Define the query to load the CDC data into BigQuery
query = f"""
     SELECT *  FROM `{table_ref}`
     WHERE _sdc_batched_at >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 DAY)
     """

# Execute the query and save the results to a dataframe
df = client.query(query).to_dataframe()

```

This code creates a client object to connect to Google BigQuery, defines the dataset and table names, and queries the table to load the CDC data into a dataframe.

You can then use Python libraries like Pandas and Matplotlib to analyze and visualize the data as needed.

Note that this is just a simple example and the specific implementation will depend on your data source, CDC settings, and other factors.

--

What alternatives of CDC in AWS, GCP and Azure?

There are several alternatives to Change Data Capture (CDC) available in cloud platforms like AWS, GCP, and Azure. Here are some of the alternatives:

1. AWS DMS (Database Migration Service): AWS DMS is a managed service that can be used to migrate data from one database to another. It supports both homogeneous and heterogeneous migrations, and also supports ongoing replication for CDC.
2. AWS Kinesis Data Streams: AWS Kinesis Data Streams is a real-time data streaming service that allows you to capture and process data in real time. You can use Kinesis Data Streams to ingest data from various sources and stream it to AWS services like S3, Redshift, or Elasticsearch.
3. GCP Dataflow: GCP Dataflow is a fully managed service for building batch and streaming data pipelines. It provides a unified programming model for both batch and streaming data processing and supports CDC.
4. GCP Cloud Functions: GCP Cloud Functions is a serverless compute service that allows you to run your code in response to events. You can use Cloud Functions to create a serverless CDC pipeline that triggers changes to your source database and writes the changes to your target database.
5. Azure Stream Analytics: Azure Stream Analytics is a fully managed service for real-time data stream processing. You can use Stream Analytics to ingest data from various sources, process the data using SQL-like queries, and output the results to Azure services like Blob storage or SQL Database.
6. Azure Functions: Azure Functions is a serverless compute service that allows you to run your code in response to events. You can use Functions to create a serverless CDC pipeline that triggers changes to your source database and writes the changes to your target database.

These are just a few examples of the CDC alternatives available in cloud platforms. The choice of tool depends on your specific use case and requirements.