01. Explain the difference between AWS Regions, Availability Zone and Edge Locations. Why is this important for data analysis and latency-sensitive applications?

- **AWS Regions:**

Geographic locations hosting multiple Availability Zones (AZs), each isolated to provide high availability and fault tolerance. Examples include US East (N. Virginia), EU (Ireland), and Asia Pacific (Mumbai).

Availability Zones (AZs):

Isolated data centres within a region, connected through low-latency networks. AZs ensure high availability, fault tolerance, and redundancy. Each AZ has its own power, cooling, and physical security.

Edge Locations:

Globally distributed caching endpoints for Amazon CloudFront, a content delivery network (CDN). Edge locations cache content at edge sites closer to users, reducing latency and improving performance for content delivery.

Importance for data analysis and latency-sensitive applications:

1. Low Latency: Edge locations reduce latency for global content delivery, while AZs ensure low-latency access to resources within a region.

2. High Availability: AZs provide redundancy and fault tolerance, ensuring applications remain available even if one AZ experiences issues.

3. Data Residency: Regions allow data to be stored in specific geographic locations, complying with data sovereignty and regulatory requirements.

4. Performance: Strategically using Regions, AZs, and Edge Locations optimizes application performance, reliability, and user experience.

By understanding and leveraging these components, you can design and deploy applications that meet specific requirements for latency, availability, and data residency.

02. Using the AWS CU, list all available AWS regions. Share the command used and the output.

- To list all available AWS regions using the AWS CLI, you can use the following command:
  bash
  ```
  AWS ec2 describe-regions --query "Regions[].{Name: RegionName}" --output table
  ```
  This command will output a table of region names. Here's a sample output based on available data:

  Available AWS Regions

  - Region Name
    - af-south-1: Africa (Cape Town)
    - ap-east-1: Asia Pacific (Hong Kong)
    - ap-northeast-1: Asia Pacific (Tokyo)
    - ap-northeast-2: Asia Pacific (Seoul)

- ap-northeast-3: Asia Pacific (Osaka)
- ap-south-1: Asia Pacific (Mumbai)
- ap-south-2: Asia Pacific (Hyderabad)
- ap-southeast-1: Asia Pacific (Singapore)
- ap-southeast-2: Asia Pacific (Sydney)
- ap-southeast-3: Asia Pacific (Jakarta)
- ap-southeast-4: Asia Pacific (Melbourne)
- ap-southeast-5: Asia Pacific (Malaysia)
- ap-southeast-6: Asia Pacific (New Zealand)
- ap-southeast-7: Asia Pacific (Thailand)
- ca-central-1: Canada (Central)
- ca-west-1: Canada West (Calgary)
- eu-central-1: Europe (Frankfurt)
- eu-central-2: Europe (Zurich)
- eu-north-1: Europe (Stockholm)
- eu-south-1: Europe (Milan)
- eu-south-2: Europe (Spain)
- eu-west-1: Europe (Ireland)
- eu-west-2: Europe (London)
- eu-west-3: Europe (Paris)
- il-central-1: Israel (Tel Aviv)
- me-central-1: Middle East (UAE)
- me-south-1: Middle East (Bahrain)
- mx-central-1: Mexico (Central)
- sa-east-1: South America (Sao Paulo)
- us-east-1: US East (N. Virginia)
- us-east-2: US East (Ohio)
- us-west-1: US West (N. California)
- us-west-2: US West (Oregon)

There are currently 34 AWS regions available, spread across the globe, with more being added regularly [1].

03. Create a new IAM user with least privilege access to Amazon S3. Share your attached policies (JSON or screenshot).

- Here's how to create a new IAM user with least privilege access to Amazon S3:

Step 1: Create a new IAM policy

Create a new policy with the following JSON:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3ReadOnlyAccess",
      "Effect": "Allow",
            "Action": [
               "s3:GetObject",
               "s3:ListBucket"
            ],
            "Resource": [
               "arn:aws:s3:::my-bucket",
               "arn:aws:s3:::my-bucket/*"
            ]
         }
      ]
   }
```

This policy grants read-only access to a specific S3 bucket (my-bucket).

Step 2: Create a new IAM user

1. Go to the IAM dashboard and click "Users" > "Create user".
2. Choose "Programmatic access" and click "Next".
3. Attach the policy you created in Step 1.
4. Review and create the user.

This IAM user has least privilege access to Amazon S3, with read-only access to a specific bucket.

04. Compare different Amazon S3 storage (Standard, Intelligent-Tiering, Glacier). When should each be used in data analytics workflows?

- Here's a comparison of different Amazon S3 storage classes:

1. S3 Standard

- High availability and durability: Designed for frequently accessed data.

- Low latency: Fast data retrieval.

- Use cases:

  - Frequently accessed data.

  - Real-time analytics.

  - Active datasets.

2. S3 Intelligent-Tiering

- Automatic tiering: Moves data between frequent and infrequent access tiers.

- Cost-effective: Optimizes storage costs based on access patterns.

- Use cases:

  - Unpredictable access patterns.

  - Data with varying access frequencies.

- Cost optimization.

3. <u>**S3 Glacier**</u>

- Low-cost archival storage: Designed for long-term data retention.
- Data retrieval times: Vary from minutes to hours.
- Use cases:
  - Infrequently accessed data.
  - Data archiving.
  - Compliance and regulatory requirements.

When to use each in data analytics workflows:

1. <u>**S3 Standard:**</u>
   - Store raw data, processed data, or intermediate results for real-time analytics.
   - Use for frequently accessed datasets, such as data mart or data warehouse.
2. <u>**S3 Intelligent-Tiering:**</u>
   - Store data with unpredictable access patterns, such as historical data or logs.
   - Use for datasets with varying access frequencies.
3. <u>**S3 Glacier:**</u>
   - Archive historical data, logs, or other infrequently accessed data.
   - Use for long-term data retention, compliance, or regulatory requirements.

Considerations:

1. <u>Data access patterns:</u> Choose storage classes based on how frequently data is accessed.
2. <u>Data retention:</u> Consider data retention requirements and choose storage classes accordingly.
3. <u>Cost optimization:</u> Use Intelligent-Tiering for cost-effective storage.

By choosing the right S3 storage class, you can optimize storage costs, improve data accessibility, and meet data retention requirements in your data analytics workflows.

05. Create an S3 bucket and upload a sample dataset (CSV or JSON). Enable versioning and show at least two version of one file.

- Here's a step-by-step guide to creating an S3 bucket, uploading a sample dataset, enabling versioning, and showing two versions of a file:

<u>**Step 1: Create an S3 bucket**</u>

1. Log in to the AWS Management Console.
2. Navigate to the S3 dashboard.

3. Click "Create bucket".

4. Enter a unique bucket name (e.g., "my-sample-bucket").

5. Choose a region (e.g., "US East (N. Virginia)").

6. Click "Create bucket".

Step 2: Upload a sample dataset

1. Click on the bucket name ("my-sample-bucket").

2. Click "Upload".

3. Select a sample CSV or JSON file (e.g., "data.csv").

4. Click "Upload".

Step 3: Enable versioning

1. Click on the bucket name ("my-sample-bucket").

2. Click "Properties".

3. Click "Versioning".

4. Click "Enable versioning".

5. Click "Save".

Step 4: Upload a new version of the file

1. Click on the file name ("data.csv").

2. Click "Upload".

3. Select the same file ("data.csv") with changes (e.g., updated data).

4. Click "Upload".

Step 5: View versions of the file

1. Click on the file name ("data.csv").

2. Click "Versions".

3. You should see at least two versions of the file:

   - Version 1: The original file uploaded.

   - Version 2: The updated file uploaded.

Versioning benefits:

1. Data protection: Versioning protects your data from accidental deletion or overwrite.

2. Data recovery: You can recover previous versions of your data.

3. Audit trail: Versioning provides an audit trail of changes made to your data.

By enabling versioning on your S3 bucket, you can ensure that your data is protected and recoverable in case of accidental changes or deletions.

06. Write and apply a lifecycle policy to move files to Glacier after 30 days and delete them after 90. Share the policy JSON or Screenshot.

- Step-by-step guide to creating and applying a lifecycle policy to move files to Glacier after 30 days and delete them after 90 days:

Step 1: Create a lifecycle policy

Create a new lifecycle policy with the following JSON:

```
{
  "Rules": [
    {
      "ID": "Move to Glacier and Delete",
      "Filter": {},
      "Status": "Enabled",
      "Transitions": [
        {
          "Days": 30,
          "StorageClass": "GLACIER"
        }
      ],
      "Expiration": {
        "Days": 90
      }
    }
  ]
}
```

This policy moves files to Glacier after 30 days and deletes them after 90 days.

Step 2: Apply the lifecycle policy

1. Go to the S3 dashboard.

2. Click on the bucket name.

3. Click "Management".

4. Click "Lifecycle rules".

5. Click "Create lifecycle rule".

6. Enter a rule name (e.g., "Glacier and Delete").

7. Choose "Whole bucket" or specify a prefix.

8. Click "Next".

9. Configure the rule:

   - Transition to Glacier after 30 days.

   - Delete after 90 days.

10. Click "Review and create".

Policy details:

- Rule name: Glacier and Delete

- Transition: Move to Glacier after 30 days

- Expiration: Delete after 90 days

Benefits:

1. Automated data management: The lifecycle policy automates data management tasks.

2. Cost optimization: Move infrequently accessed data to Glacier to reduce costs.

3. Data retention: Delete data after a specified period to comply with data retention policies.

By applying this lifecycle policy, you can automate the movement of files to Glacier and deletion after a specified period, reducing storage costs and ensuring compliance with data retention policies.

07. Compare RDS, DynamoDB, and Redshift for use in different stages of a data pipeline. Give one use case for each.

Here's a comparison of RDS, DynamoDB, and Redshift for use in different stages of a data pipeline:

1. RDS (Relational Database Service)
- Use case: Transactional data storage for e-commerce platform
- **Characteristics:**
  - Relational database
  - Supports various database engines (e.g., MySQL, PostgreSQL)
  - Suitable for OLTP (Online Transaction Processing)
- **Benefits:**
  - Supports complex transactions
  - Ensures data consistency and integrity

2. DynamoDB
- Use case: Real-time analytics for gaming platform
- **Characteristics:**
  - NoSQL database
  - Fast performance and seamless scalability
  - Suitable for high-traffic applications
- **Benefits:**
  - Handles large amounts of data

- Provides low-latency performance

3. **Redshift**

- Use case: Data warehousing and analytics for customer behavior analysis
- **Characteristics:**
  - Columnar storage
  - Optimized for analytics and reporting
  - Suitable for OLAP (Online Analytical Processing)
- **Benefits:**
  - Fast query performance
  - Supports complex analytics and aggregations

Comparison:

Service	Use Case	Data Model	Performance	Scalability
RDS	Transactional data	Relational	Consistent	Vertical
DynamoDB	Real-time analytics	NoSQL	Fast	Horizontal
Redshift	Data warehousing	Columnar	Fast	Horizontal

Data Pipeline Stages:

1. Data ingestion: RDS or DynamoDB for transactional data, Redshift for analytics.
2. Data processing: Use services like AWS Glue or EMR for data transformation and processing.
3. Data storage: Choose based on data model and performance requirements (RDS, DynamoDB, or Redshift).
4. Data analytics: Redshift for complex analytics, DynamoDB for real-time insights.

By choosing the right service for each stage, you can build an efficient and scalable data pipeline.

08. What is serverless computing? Discuss pros and cons of using AWS Lambda for data pipelines.
Serverless Computing:

- Serverless computing is a cloud computing model where the cloud provider manages the infrastructure and dynamically allocates resources as needed. This approach eliminates the need to provision or manage servers, allowing developers to focus on writing code.

AWS Lambda:

AWS Lambda is a serverless compute service that runs code in response to events, such as changes to data or API calls. It's ideal for real-time data processing, event-driven architectures, and scalable applications.

Pros of using AWS Lambda for data pipelines:

1. Scalability: Lambda automatically scales to handle large workloads.

2. Cost-effective: Pay only for the compute time consumed.

3. Event-driven: Trigger Lambda functions in response to events.

4. Low maintenance: No server management required.

5. Flexibility: Supports multiple programming languages.

Cons of using AWS Lambda for data pipelines:

1. Function duration limits: 15-minute execution time limit.

2. Memory constraints: Limited memory allocation (up to 10 GB).

3. Cold start: Initial invocation may experience latency.

4. Vendor lock-in: Dependence on AWS services.

5. Monitoring and debugging: Requires additional tools and configuration.

Use cases for AWS Lambda in data pipelines:

1. Real-time data processing: Process streaming data from sources like Kinesis or IoT devices.

2. Data transformation: Transform and format data for analytics or storage.

3. Event-driven workflows: Trigger workflows based on events, such as data arrival or changes.

By understanding the pros and cons of AWS Lambda, you can effectively utilize serverless computing for scalable and efficient data pipelines.

09. Create a Lambda function triggered by S3 uploads that tags file name, size, and timestamp to Cloud watch. Share code and a log screenshot.

- Here's a step-by-step guide to creating a Lambda function triggered by S3 uploads that logs file name, size, and timestamp to CloudWatch:

Step 1: Create an IAM role for Lambda

Create an IAM role with the necessary permissions for Lambda to access S3 and CloudWatch.

Step 2: Create a Lambda function

1. Go to the AWS Lambda dashboard.

2. Click "Create function".

3. Choose "Author from scratch".

4. Enter a function name (e.g., "S3UploadLogger").

5. Choose a runtime (e.g., Python 3.9).

6. Set the handler to your Python script.

Lambda function code (Python):

```
import boto3

import logging


s3 = boto3.client('s3')

cloudwatch = boto3.client('logs')
```

```
log_group_name = 'S3UploadLogs'


def lambda_handler(event, context):
    for record in event['Records']:
        bucket_name = record['s3']['bucket']['name']
        object_key = record['s3']['object']['key']
        object_size = record['s3']['object']['size']
        timestamp = record['eventTime']


        log_message = f'File uploaded: {object_key}, Size: {object_size} bytes, Timestamp:
{timestamp}'


        cloudwatch.put_log_events(
            logGroupName=log_group_name,
            logStreamName='S3UploadLogStream',
            logEvents=[
                {
                    'timestamp': int(timestamp_to_millis(timestamp)),
                    'message': log_message
                }
            ]
        )


    return {
        'statusCode': 200,
        'statusMessage': 'OK'
    }


def timestamp_to_millis(timestamp):
    import datetime
    dt = datetime.datetime.strptime(timestamp, '%Y-%m-%dT%H:%M:%S.%fZ')
    return dt.timestamp() * 1000
```

Step 3: Configure S3 trigger

1. Go to the Lambda function.

2. Click "Add trigger".

3. Choose "S3".

4. Select the S3 bucket.

5. Set the event type to "Object created".

Step 4: Test the Lambda function

Upload a file to the S3 bucket. The Lambda function should trigger and log the file name, size, and timestamp to CloudWatch.

CloudWatch log:

File uploaded: example.txt, Size: 1024 bytes, Timestamp: 2023-03-01T12:00:00.000Z

This Lambda function logs S3 upload events to CloudWatch, providing valuable insights into file uploads.

10. Use AWS Glue to crow your S3 dataset, create a Data CatLog table, and run a Glue job to convert CSV data to parquet. Share job code and output location.

- Here's a step-by-step guide to using AWS Glue to crawl an S3 dataset, create a Data CatLog table, and run a Glue job to convert CSV data to Parquet:

Step 1: Create an AWS Glue Crawler

1. Go to the AWS Glue dashboard.

2. Click "Crawlers".

3. Click "Create crawler".

4. Enter a crawler name (e.g., "S3Crawler").

5. Choose "S3" as the data source.

6. Specify the S3 path (e.g., "s3://my-bucket/data/").

7. Create an IAM role for the crawler.

Step 2: Run the Crawler

1. Run the crawler to populate the Data CatLog.

2. The crawler will create a table in the Data CatLog.

Step 3: Create a Glue Job

1. Go to the AWS Glue dashboard.

2. Click "Jobs".

3. Click "Create job".

4. Enter a job name (e.g., "CSVtoParquet").

5. Choose "Spark" as the type.

6. Specify the script location (e.g., S3 bucket).

Glue Job Code (Python):

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
# Read CSV data
datasource0 = glueContext.create_dynamic_frame.from_CatLog(
    database="my_database",
    table_name="my_table",
    transformation_ctx="datasource0"
)
# Convert to Parquet
datasink4 = glueContext.write_dynamic_frame.from_options(
    frame=datasource0,
    connection_type="s3",
    connection_options={
        "path": "s3://my-bucket/parquet-data/"
    },
    format="parquet",
    transformation_ctx="datasink4"
)
```

job.commit()

Step 4: Run the Glue Job

1. Run the Glue job to convert the CSV data to Parquet.

2. The Parquet files will be stored in the specified S3 location (e.g., "s3://my-bucket/parquet-data/").

Output Location:

The Parquet files will be stored in the S3 location specified in the Glue job code (e.g., "s3://my-bucket/parquet-data/").

This Glue job converts CSV data to Parquet, optimizing storage and query performance for analytics and data processing.

11. Explain the difference between Kinesis Data Streams, Kinesis Firehose, and Kinesis Data Analytics. Provide a real-world example of how each would be used.
- Here's a comparison of Kinesis Data Streams, Kinesis Firehose, and Kinesis Data Analytics:
1. Kinesis Data Streams
- Real-time data processing: Capture and process large amounts of data in real-time.
- Custom processing: Use custom code to process and analyse data.
- Example: Real-time analytics for IoT sensor data.
Real-world example: A manufacturing company uses Kinesis Data Streams to collect sensor data from its production line. The data is processed in real-time to detect anomalies and predict equipment failures.

2. Kinesis Firehose
- Data ingestion and delivery: Capture and deliver data to destinations like S3, Redshift, and Elasticsearch.
- Near real-time: Data is delivered in near real-time.
- Example: Log data ingestion and delivery to S3.
Real-world example: A web application uses Kinesis Firehose to collect log data from its users. The data is delivered to S3 for storage and analysis.
3. Kinesis Data Analytics
- Real-time analytics: Analyse data in real-time using SQL or Apache Flink.
- Windowed aggregations: Perform aggregations over time windows.
- Example: Real-time analytics for social media data.
Real-world example: A social media platform uses Kinesis Data Analytics to analyse user engagement metrics in real-time. The platform uses SQL to calculate metrics like engagement rates and trending topics.
Comparison:

Service	Purpose	Data Processing	Destination
Kinesis Data Streams	Real-time data processing	Custom code	Custom
Kinesis Firehose	Data ingestion and delivery	Pre-built	S3, Redshift, Elasticsearch
Kinesis Data Analytics	Real-time analytics	SQL or Apache Flink	In-application

Each Kinesis service serves a specific purpose, from real-time data processing to data ingestion and analytics. By choosing the right service, you can build scalable and efficient data pipelines.:

12. What is columnar storage and how does it benefit Redshift performance for analytics workloads?

- **Columnar Storage:**
  Columnar storage is a data storage format that stores data in columns instead of rows. This approach optimizes data storage and retrieval for analytics workloads.

  Benefits for Redshift Performance:

  1. Faster query performance: Columnar storage reduces the amount of data that needs to be scanned, leading to faster query performance.
  2. Improved compression: Columnar storage enables better compression, reducing storage costs and improving query performance.
  3. Efficient data retrieval: Columnar storage allows for efficient retrieval of specific columns, reducing the amount of data that needs to be processed.

  How Columnar Storage Benefits Analytics Workloads:

  1. Aggregation queries: Columnar storage optimizes aggregation queries, such as SUM and AVG, by only accessing the relevant columns.
  2. Filtering and sorting: Columnar storage enables efficient filtering and sorting of data, reducing the amount of data that needs to be processed.
  3. Join operations: Columnar storage optimizes join operations by allowing for efficient retrieval of specific columns.

  Redshift's Columnar Storage:
  Redshift's columnar storage is optimized for analytics workloads, providing:

  1. High-performance querying: Redshift's columnar storage enables fast query performance, even for large datasets.
  2. Efficient data compression: Redshift's columnar storage compresses data efficiently, reducing storage costs.
  3. Scalability: Redshift's columnar storage scales horizontally, allowing for efficient processing of large datasets.

  By leveraging columnar storage, Redshift provides fast and efficient query performance for analytics workloads, making it an ideal choice for data warehousing and business intelligence applications.

13. Load a CSV file from S3 into Redshift using the COPY command. Share table schema, command used, and sample output from a query.

- Here's a step-by-step guide to loading a CSV file from S3 into Redshift using the COPY command:

  Table Schema:

```
CREATE TABLE sales (
   id INTEGER,
   product_name VARCHAR(255),
   sales_date DATE,
   amount DECIMAL(10, 2)
);
```

COPY Command:

```
COPY sales (
    id,
    product_name,
    sales_date,
    amount
)
FROM 's3://my-bucket/sales_data.csv'
IAM_ROLE 'arn:aws:iam::123456789012:role/my-redshift-role'
CSV QUOTE '"' DELIMITER ','
REGION 'us-west-2';
```

Explanation:

1. COPY sales: Specifies the table to load data into.
2. FROM 's3://my-bucket/sales_data.csv': Specifies the S3 location of the CSV file.
3. IAM_ROLE 'arn:aws:iam::123456789012:role/my-redshift-role': Specifies the IAM role with necessary permissions to access S3.
4. CSV QUOTE '"' DELIMITER ',': Specifies the CSV format options.

Sample Output from a Query:

```
SELECT * FROM sales LIMIT 5;
```

Output:

```
 id | product_name | sales_date | amount
----+--------------+------------+--------
  1 | Product A    | 2022-01-01 | 100.00
  2 | Product B    | 2022-01-02 | 200.00
  3 | Product C    | 2022-01-03 | 300.00
  4 | Product D    | 2022-01-04 | 400.00
  5 | Product E    | 2022-01-05 | 500.00
```

This COPY command efficiently loads data from S3 into Redshift, enabling fast querying and analysis of the data.

14. What is the role of the AWS Glue Data CatLog in Athena? How does schema-on-read work?
AWS Glue Data CatLog:

- The AWS Glue Data CatLog is a centralized metadata repository that stores information about data sources, tables, and databases. In Athena, the Data CatLog plays a crucial role in:

  1. Metadata management: Stores metadata about data sources, tables, and databases.
  2. Schema management: Defines the structure of data, including column names, data types, and relationships.
  3. Data discovery: Enables users to discover and search for data assets.

  Schema-on-Read:
  Schema-on-read is a data processing approach where the schema is applied to the data at query time, rather than at data ingestion time. In Athena, schema-on-read works as follows:

  1. Data ingestion: Data is stored in S3 without a predefined schema.
  2. Schema definition: A schema is defined in the Data CatLog, which describes the structure of the data.
  3. Query execution: When a query is executed, Athena applies the schema to the data, allowing for efficient querying and analysis.

  Benefits of Schema-on-Read:

  1. Flexibility: Allows for changes to the schema without modifying the underlying data.
  2. Efficient querying: Enables fast querying and analysis of data without requiring data transformation or loading.
  3. Cost-effective: Reduces storage and processing costs by avoiding data duplication and transformation.

  How Athena uses Schema-on-Read:

  1. Query optimization: Athena optimizes queries based on the schema, reducing the amount of data that needs to be scanned.
  2. Data pruning: Athena prunes data that is not relevant to the query, reducing the amount of data that needs to be processed.
  3. Fast query performance: Athena's schema-on-read approach enables fast query performance, even for large datasets.

  By leveraging the AWS Glue Data CatLog and schema-on-read approach, Athena provides a flexible and efficient way to query and analyse data in S3.

15. Create an Athena table from S3 data using Glue CatLog. Run a query and share the SQL + result screenshot.

- To create an Athena table from S3 data using Glue CatLog, follow these steps [1]:

  Step 1: Create a Database in Glue CatLog

  - Navigate to the AWS Glue console and select "Databases" from the left-hand menu.
  - Click "Add database" and name your database uniquely, like "activitylogs3000db".

Step 2: Create a Table in Athena Using Glue Data CatLog

- Open Athena and create a table using a DDL statement like the following:

```
CREATE EXTERNAL TABLE IF NOT EXISTS activitylogs3000db.activity_log (
  `date` date,
  `time` string,
  `location_code` string,
  `placeholder1` string,
  `ip_address` string,
  `log_type` string,
  `domain` string,
  `activity_path` string,
  `response_code` begin,
  `placeholder2` string,
  `placeholder3` string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION 's3://your-bucket-name/log-file/';
```

Step 3: Run a Query

- After creating the table, you can run a query like:

```
SELECT * FROM activitylogs3000db.activity_log LIMIT 10;
```

Result:

The result will display the first 10 rows of your data, depending on the data stored in your S3 bucket. Here's a sample output:

```
date       | time       | location_code | placeholder1 | ip_address | log_type | domain |
activity_path | response_code | placeholder2 | placeholder3
-----------|------------|---------------|--------------|------------|----------|--------|---------------|-
--------------|--------------|-------------
2022-01-01 | 12:00:00   | NYC           |              | 192.0.2.1  | GET      | example | /index.html  |
200           |              |
```

16. Describe how Amazon Quick sight supports business intelligence in a serverless data architecture. What are SPICE and embedded dashboards?

- Amazon Quick Sight is a serverless business intelligence service that supports fast and easy data analysis. Here's how it works:

Key Features:

- Serverless Architecture: Automatically scales to hundreds of thousands of users without requiring server setup, configuration, or management.
- SPICE (Super-fast, Parallel, In-memory Calculation Engine): Achieves blazing-fast performance at scale, replicating data for high availability and allowing thousands of users to perform interactive analysis simultaneously.
- Embedded Dashboards: Allows embedding interactive dashboards and visualizations into applications, portals, and websites without extra coding or licensing.

Benefits:

- Scalability: Automatically scales to tens of thousands of users without infrastructure management.
- Fast Performance: SPICE enables fast query performance and interactive analysis.
- Cost-Effective: Pay-per-session pricing model means you only pay for what you use.
- Unified Business Intelligence: Provides a single source of truth for data-driven insights across interactive dashboards, reports, and embedded analytics.

Embedded Dashboards:

- Easy Embedding: Embed dashboards in minutes with a simple copy-paste process.
- Customizable: Match embedded content with your application's look and feel using built-in themes and JavaScript SDK.
- Scalable: Automatically scales to hundreds of thousands of users without managing servers [1] [2].

SPICE:

- In-Memory Calculation Engine: Processes data quickly and efficiently.
- High Availability: Replicates data for high availability and fast query performance.
- Large Dataset Support: Supports datasets up to a billion rows, enabling fast analysis of large datasets [2] [3].

17. Connect Quick sight to Athena or Redshift and build a dashboard with at least one calculated field and one filter. Share a screenshot of your final dashboard.

- Here's a step-by-step guide to connecting Quick Sight to Athena and building a dashboard:

Step 1: Connect Quick Sight to Athena

1. Go to Quick Sight and click on "Datasets" > "New dataset".
2. Select "Athena" as the data source.
3. Enter the database and table name, and configure the connection settings.

Step 2: Create a Calculated Field

1. Go to the dataset and click on "Add calculated field".

2. Create a calculated field, such as "Revenue Growth" = (current_revenue - previous_revenue) / previous_revenue * 100.

Step 3: Create a Filter

1. Go to the analysis and click on "Filter" > "Create filter".
2. Create a filter, such as "Region" = "North America".

Step 4: Build the Dashboard

1. Create a new analysis and add visuals, such as charts and tables.
2. Use the calculated field and filter to create interactive visuals.

Final Dashboard:

Here's a sample dashboard with a calculated field and filter:

[Dashboard Screenshot]

The dashboard shows revenue growth by region, with a filter applied to show only data for North America. The calculated field "Revenue Growth" is used to display the percentage change in revenue.

Dashboard Components:

1. Revenue Growth Chart: A bar chart showing revenue growth by region.
2. Region Filter: A dropdown filter allowing users to select specific regions.
3. Calculated Field: The "Revenue Growth" calculated field is used to display the percentage change in revenue.

This dashboard provides insights into revenue growth by region, enabling users to analyze and explore the data in a visually appealing way.

18. Explain how AWS Cloud Watch and Cloud Trail differ. In a data analytics pile line, what role does each play in monitoring auditing, and troubleshooting?

- **AWS CloudWatch and CloudTrail:**

1. CloudWatch: Monitoring and logging service for AWS resources and applications.
2. CloudTrail: Auditing and logging service for AWS API calls and account activity.

Key differences:

1. Focus: CloudWatch focuses on performance, health, and monitoring, while CloudTrail focuses on auditing, compliance, and governance.
2. Data: CloudWatch collects metrics, logs, and events, while CloudTrail collects API call history and account activity.

Roles in a data analytics pipeline:

CloudWatch:

1. Monitoring: Tracks performance metrics for AWS resources, such as S3, Glue, and Athena.
2. Alerting: Sets alarms for anomalies or threshold breaches, enabling proactive issue resolution.
3. Troubleshooting: Provides logs and metrics for diagnosing issues in data processing and analysis.

CloudTrail:

1. Auditing: Tracks API calls and account activity, providing a record of changes and actions.
2. Compliance: Ensures regulatory compliance by maintaining a record of all API calls and account activity.
3. Security: Helps detect unauthorized access or malicious activity in the data analytics pipeline.

Benefits:

1. Improved visibility: CloudWatch and CloudTrail provide comprehensive monitoring and auditing capabilities.
2. Faster troubleshooting: CloudWatch metrics and logs enable quick issue resolution.
3. Enhanced security: CloudTrail's auditing capabilities help detect security threats and ensure compliance.

By using both CloudWatch and CloudTrail, organizations can ensure their data analytics pipeline is monitored, audited, and secured, enabling reliable and compliant data processing and analysis.

19. Describe a complete end-to-end data analysis pipe line using AWS services include services for data ingestion, storage, transformation, querying, and visualization (Example S3 – Lambda – Glue – Quick sight.
Explain why you would choose each service for stage it's used in.

- Here's a complete end-to-end data analysis pipeline using AWS services:

Pipeline:

1. Data Ingestion: Amazon S3 (Storage) → AWS Lambda (Trigger)
2. Data Transformation: AWS Glue (ETL)
3. Data Storage: Amazon S3 (Processed Data Storage)
4. Data Querying: Amazon Athena (SQL Queries)
5. Data Visualization: Amazon Quick Sight (Business Intelligence)

Pipeline Explanation:

1. Data Ingestion:

- Amazon S3: Collects data from various sources and stores it in S3 buckets.
- AWS Lambda: Triggered by S3 object creation, Lambda can perform initial data processing or validation.

2. Data Transformation:
   - AWS Glue: Performs ETL (Extract, Transform, Load) operations on the data, handling complex data transformations and loading data into processed data storage.
3. Data Storage:
   - Amazon S3: Stores processed data in S3 buckets, providing durable and scalable storage.
4. Data Querying:
   - Amazon Athena: Allows users to query data using SQL, providing fast and efficient querying capabilities.
5. Data Visualization:
   - Amazon Quick Sight: Provides business intelligence and data visualization capabilities, enabling users to gain insights from data.

Why choose each service?

1. **S3:** Scalable, durable, and widely used object storage service.
2. **Lambda**: Serverless compute service that can handle event-driven processing and validation.
3. **Glue:** Managed ETL service that simplifies data transformation and loading.
4. **Athena:** Serverless query service that provides fast and efficient SQL querying capabilities.
5. Quick Sight: Fast, cloud-powered business intelligence service that enables data visualization and insights.

Benefits:

1. Scalability: Each service scales automatically, handling large volumes of data and users.
2. Cost-Effective: Pay-per-use pricing model ensures costs are optimized.
3. Integration: AWS services integrate seamlessly, enabling a cohesive data analysis pipeline.
4. Security: AWS services provide robust security features, ensuring data protection and compliance.

This pipeline provides a comprehensive end-to-end data analysis solution, enabling organizations to collect, process, query, and visualize data efficiently.

Dear Instructor, Due to some technical issues I was unable to access AWS and hence I'm not able to attach the screenshot of the task.