AWS Assignment Solutions

## 1. Explain the difference between AWS Regions, Availability Zones, and Edge Locations.

AWS Regions are geographic areas that contain multiple isolated locations known as Availability Zones (AZs). Each AZ consists of one or more data centers with redundant power and networking. Edge Locations are endpoints for AWS services like CloudFront, used for content delivery to reduce latency.  
  
Importance:  
- Regions let users deploy resources close to their users to reduce latency.  
- AZs help build high availability and fault-tolerant applications.  
- Edge Locations reduce content delivery times for latency-sensitive apps.

## 2. Using the AWS CLI, list all available AWS regions. Share the command used and the output.

Command:  
aws ec2 describe-regions --query "Regions[\*].RegionName" --output table  
  
Sample Output:  
+---------------------+  
| DescribeRegions |  
+---------------------+  
| ap-south-1 |  
| eu-west-1 |  
| us-east-1 |  
| us-west-2 |  
+---------------------+

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

Sample Policy JSON:  
{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Action": [  
 "s3:ListBucket",  
 "s3:GetObject",  
 "s3:PutObject"  
 ],  
 "Resource": [  
 "arn:aws:s3:::your-bucket-name",  
 "arn:aws:s3:::your-bucket-name/\*"  
 ]  
 }  
 ]  
}

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

Comparison:  
- Standard: Used for frequently accessed data. High durability and availability.  
- Intelligent-Tiering: Automatically moves data between access tiers based on usage.  
- Glacier: Designed for long-term archival. Low cost, but retrieval takes time.  
  
Analytics Use Cases:  
- Standard: For real-time or high-frequency data processing.  
- Intelligent-Tiering: When access patterns are unpredictable.  
- Glacier: Store raw or processed historical data for compliance or long-term storage.

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

Commands:  
aws s3 mb s3://my-analytics-bucket  
aws s3api put-bucket-versioning --bucket my-analytics-bucket --versioning-configuration Status=Enabled  
aws s3 cp sample-v1.csv s3://my-analytics-bucket/  
aws s3 cp sample-v2.csv s3://my-analytics-bucket/sample-v1.csv  
  
List Versions:  
aws s3api list-object-versions --bucket my-analytics-bucket --prefix sample-v1.csv

## 6. Write and apply a lifecycle policy to move files to Glacier after 30 days and delete them after 90.

Sample Lifecycle Policy JSON:  
{  
 "Rules": [  
 {  
 "ID": "MoveToGlacierAndDelete",  
 "Prefix": "",  
 "Status": "Enabled",  
 "Transitions": [  
 {  
 "Days": 30,  
 "StorageClass": "GLACIER"  
 }  
 ],  
 "Expiration": {  
 "Days": 90  
 }  
 }  
 ]  
}

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

- RDS : Managed relational DB for transactional workloads. Use Case: Store user profiles in a web app.  
- DynamoDB : NoSQL DB for key-value and document data. Use Case: Real-time user session tracking.  
- Redshift : Columnar DB for analytics. Use Case: Run queries on large-scale sales data for BI dashboards.

## 8. Create a DynamoDB table and insert 3 records manually. Then write a Lambda function that adds records when triggered by S3 uploads.

Create Table:  
aws dynamodb create-table --table-name AnalyticsData --attribute-definitions AttributeName=ID,AttributeType=S --key-schema AttributeName=ID,KeyType=HASH --billing-mode PAY\_PER\_REQUEST  
  
Insert Records:  
aws dynamodb put-item --table-name AnalyticsData --item '{"ID":{"S":"1"},"Name":{"S":"Record1"}}'  
...  
  
Lambda Function (Python):  
import json  
import boto3  
def lambda\_handler(event, context):  
 dynamodb = boto3.resource('dynamodb')  
 table = dynamodb.Table('AnalyticsData')  
 for record in event['Records']:  
 file\_name = record['s3']['object']['key']  
 table.put\_item(Item={'ID': file\_name, 'Name': 'Uploaded'})

## 9. What is serverless computing? Discuss pros and cons of using AWS Lambda for data pipelines.

Serverless computing lets developers run code without managing servers. AWS Lambda is a key example.  
  
 Pros:   
- No infrastructure management  
- Scales automatically  
- Pay-per-use  
  
 Cons:   
- Cold start latency  
- Limited execution time (15 min max)  
- Debugging and monitoring can be harder

## 10. Create a Lambda function triggered by S3 uploads that logs file name, size, and timestamp to CloudWatch.

Lambda Function:  
import json  
import logging  
def lambda\_handler(event, context):  
 logger = logging.getLogger()  
 logger.setLevel(logging.INFO)  
 for record in event['Records']:  
 logger.info(f"File: {record['s3']['object']['key']}, Size: {record['s3']['object']['size']}, Time: {record['eventTime']}")

## 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.

- Kinesis Data Streams : Real-time streaming data ingestion and custom processing.  
 - \*Example\*: Real-time log collection and custom transformation before storing in S3.  
- Kinesis Firehose : Fully managed service that loads streaming data directly to destinations like S3, Redshift.  
 - \*Example\*: Send IoT sensor data directly to S3 for storage.  
- Kinesis Data Analytics : Real-time SQL queries on streaming data.  
 - \*Example\*: Run real-time analytics on clickstream data from a website.

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

Columnar storage stores data by columns instead of rows. In analytics, queries often access only a few columns.  
  
 Benefits in Redshift:   
- Reduces I/O by reading only required columns  
- Improves compression efficiency  
- Faster query performance for large datasets

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

Table Schema:   
CREATE TABLE sales (  
 sale\_id INT,  
 product\_name VARCHAR(100),  
 quantity INT,  
 sale\_date DATE  
);  
  
COPY Command:  
COPY sales FROM 's3://my-bucket/sales.csv'  
CREDENTIALS 'aws\_iam\_role=arn:aws:iam::123456789012:role/MyRedshiftRole'  
CSV IGNOREHEADER 1;  
  
Sample Query:  
SELECT product\_name, SUM(quantity) FROM sales GROUP BY product\_name;

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

Glue Data Catalog : Acts as a central metadata repository. Athena uses it to interpret structure of data stored in S3.  
  
 Schema-on-read : You define structure at query time, not when data is stored. This allows flexibility with unstructured or semi-structured data.

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

Steps:  
- Crawl S3 data with AWS Glue to create the catalog table.  
- Use Athena to query using the table.  
  
SQL Example:  
SELECT \* FROM glue\_catalog\_db. sales data LIMIT 10;