AWS Project Series: Demonstrating Skills with AWS Services

Project 1: Launching an EC2 Instance

Objective: To learn how to launch and connect to an EC2 instance, which serves as a virtual server.

Resource Information: Amazon EC2 (Elastic Compute Cloud):

- **Purpose:** Provides scalable virtual servers for running applications.
- **Key Features:**
  - Various instance types for different workloads.
  - Flexible OS and software configurations.
  - Elastic IPs for consistent public IP access.
- **Real-World Use Case:** Hosting web applications, development environments, or running batch processing jobs.

Real-World Use Case: Organizations often need to quickly deploy virtual servers to host applications or services.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **EC2** service.
3. Click on **Launch Instance**.
4. Choose an Amazon Machine Image (AMI) – select "Amazon Linux 2".
5. Select an instance type – choose "t2.micro" (free tier eligible).
6. Configure instance details – keep the default settings.
7. Add storage – retain the default 8 GB size.
8. Add tags – click **Add Tag** and specify a key-value pair (e.g., `Name: MyFirstEC2`).
9. Configure security group – create a new security group with an inbound rule allowing SSH access (port 22) from your IP.

10. Review and launch the instance.
11. Create or select an existing key pair for SSH access and download the `.pem` file.

Connect to the instance using an SSH client:

12. ssh -i "your-key.pem" ec2-user@<instance-public-ip>
13. Verify the instance by running `uname -a` on the terminal.

Challenges and Solution:

- **Challenge:** Ensuring secure access to the EC2 instance.
- **Solution:** This project demonstrates the use of key pairs and security group rules to provide secure access.

Expected Outcome: At the end of this project, you will have a running EC2 instance that you can securely access via SSH.

Project 2: Storing and Accessing Data in S3

Objective: To learn how to create an S3 bucket, upload objects, and retrieve them.

Resource Information: Amazon S3 (Simple Storage Service):

- **Purpose:** Offers scalable object storage with high durability and availability.
- **Key Features:**
  - Tiered storage classes for cost optimization.
  - Lifecycle policies for automated data management.
  - Versioning to protect against accidental overwrites.
- **Real-World Use Case:** Hosting website files, storing backups, or archiving data.

Real-World Use Case: S3 is widely used for scalable data storage solutions for applications and backups.

Steps to Complete the Project:

1. Log in to the AWS Management Console.

2. Navigate to the **S3** service.
3. Click **Create Bucket**.
4. Provide a unique bucket name (e.g., `my-first-s3-bucket`) and choose a region.
5. Keep default settings and click **Create Bucket**.
6. Open the newly created bucket and click **Upload**.
7. Add files from your local system and click **Upload**.
8. Enable public access for the file:
   - Click the file name in the bucket.
   - Navigate to the **Permissions** tab.
   - Edit the **Object** permissions to allow public access.
9. Retrieve the public URL and open it in a browser to verify access.
10. (Optional) Set up lifecycle rules to manage storage costs by transitioning files to other storage classes.

Challenges and Solution:

- **Challenge:** Managing public and private access to files.
- **Solution:** This project demonstrates how to enable and restrict public access to files for secure data storage.

Expected Outcome: You will have a functional S3 bucket with uploaded files that can be accessed based on permissions.

Project 3: Hosting a Static Website on S3

Objective: To host a static website using an S3 bucket.

Real-World Use Case: Many businesses need low-cost and reliable hosting solutions for static websites.

Steps to Complete the Project:

1. Log in to the AWS Management Console.

2. Navigate to the **S3** service.
3. Create a new bucket (e.g., `my-static-site-bucket`) and disable block public access settings.
4. Upload your static website files (e.g., `index.html`, `styles.css`).
5. Enable static website hosting:
   - Go to the **Properties** tab of the bucket.
   - Under **Static website hosting**, select **Enable**.
   - Specify `index.html` as the index document.
6. Save the configuration and copy the bucket's endpoint URL.
7. Test the URL in a browser.

Challenges and Solution:

- **Challenge:** Configuring static website hosting in a secure and scalable way.
- **Solution:** This project demonstrates how to leverage S3's built-in website hosting feature to serve static content.

Expected Outcome: You will have a live static website hosted on S3, accessible via the bucket's endpoint. The endpoint is public.

Project 4: Setting Up a DynamoDB Table

Objective: To create a DynamoDB table and perform basic operations.

Resource Information: Amazon DynamoDB

- **Purpose:** Fully managed NoSQL database for key-value and document storage.
- **Key Features:**
  - High performance with single-digit millisecond latency.
  - Built-in security, backup, and restore capabilities.
  - On-demand and provisioned capacity modes.
- **Real-World Use Case:** Storing user profiles, IoT data, or product catalogs.

Real-World Use Case: Applications often require a fast and scalable NoSQL database for structured data storage.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **DynamoDB** service.
3. Click **Create Table**.
4. Provide a table name (e.g., `UserData`) and a primary key (e.g., `UserID` as a string).
5. Leave the default settings for capacity mode and click **Create Table**.
6. Insert data into the table:
   - Go to the **Items** tab and click **Create Item**.
   - Use the JSON editor to add an item (e.g., `{"UserID": "001", "Name": "John Doe"}`).
7. Query the table by specifying the primary key.

Challenges and Solution:

- **Challenge:** Efficiently managing structured data in a scalable way.
- **Solution:** This project demonstrates how DynamoDB can be used to store and query data with high performance.

Expected Outcome: You will have a functional DynamoDB table with stored data that can be queried efficiently.

Project 5: Building a Serverless Web Application with AWS Lambda, API Gateway, and DynamoDB

Objective: To build a serverless web application that uses API Gateway to expose endpoints, AWS Lambda for backend logic, and DynamoDB for database storage.

Resource Information: AWS Lambda and Amazon API Gateway

- **AWS Lambda Purpose:** Enables serverless computing to run code in response to events without managing servers.

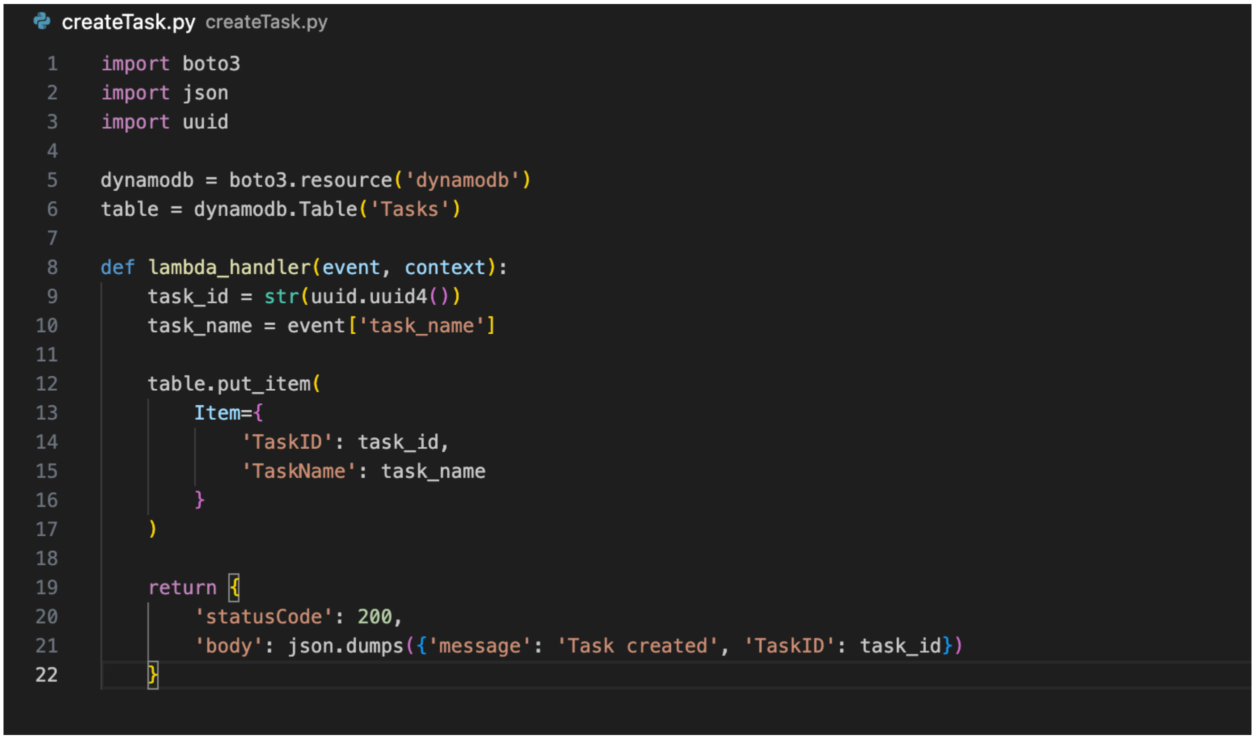
    - **Key Features:**
        - Supports multiple programming languages.
        - Automatically scales with the workload.
        - Integration with other AWS services like DynamoDB, S3, and API Gateway.
- **API Gateway Purpose:** Creates and manages APIs to enable communication between clients and backend services.
    - **Key Features:**
        - Supports RESTful and WebSocket APIs.
        - Integration with AWS Lambda for serverless APIs.
        - API monitoring and throttling for performance management.

Real-World Use Case: Develop a serverless task management application where users can create, update, retrieve, and delete tasks.

Steps to Complete the Project:

1. **Set Up DynamoDB Table:**
    - Go to the **DynamoDB** service in AWS Management Console.
    - Create a table named `Tasks` with the primary key `TaskID` (string type).
2. **Create AWS Lambda Functions:**
    - Navigate to the **Lambda** service.
    - Create the following functions:
        - `createTask`: Inserts a new task into the `Tasks` table.
        - `getTasks`: Retrieves all tasks from the table.
        - `updateTask`: Updates an existing task based on `TaskID`.
        - `deleteTask`: Deletes a task based on `TaskID`.
    - Use the AWS SDK (Boto3 for Python or equivalent) in the Lambda functions to interact with DynamoDB.

Example code for `createTask`:



createTask.py createTask.py

```python
import boto3
import json
import uuid

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Tasks')

def lambda_handler(event, context):
    task_id = str(uuid.uuid4())
    task_name = event['task_name']

    table.put_item(
        Item={
            'TaskID': task_id,
            'TaskName': task_name
        }
    )

    return {
        'statusCode': 200,
        'body': json.dumps({'message': 'Task created', 'TaskID': task_id})
    }
```

3. **Set Up API Gateway:**
    - Navigate to the **API Gateway** service.
    - Create a new **REST API** and name it `TaskAPI`.
    - Define methods (`POST`, `GET`, `PUT`, `DELETE`) corresponding to each Lambda function:
        - For `POST`, link it to `createTask`.
        - For `GET`, link it to `getTasks`.
        - For `PUT`, link it to `updateTask`.
        - For `DELETE`, link it to `deleteTask`.
    - Deploy the API to a stage (e.g., `dev`) and obtain the endpoint URL.
4. **Test the Application:**
    - Use tools like Postman or curl to interact with the API Gateway endpoints.
    - For example:
        - **Create a Task:** Send a `POST` request with a JSON body (`{"task_name": "Test Task"}`) to the `/tasks` endpoint.

        - **Get Tasks:** Send a `GET` request to `/tasks`.

5. **Monitor and Debug:**
    - Use AWS CloudWatch Logs to monitor Lambda execution and troubleshoot any issues.
    - Review API Gateway metrics for insights into API usage.

Challenges and Solution:

- **Challenge:** Ensuring low latency and high availability without managing servers.
- **Solution:** By combining API Gateway, Lambda, and DynamoDB, this project showcases a highly scalable serverless architecture.

Project 6: Setting Up a Custom VPC with Public and Private Subnets

Objective: To create a Virtual Private Cloud (VPC) with public and private subnets for hosting resources securely.

Resource Information: Amazon VPC (Virtual Private Cloud)

- **Purpose:** Enables network isolation for resources.
- **Key Features:**
    - Subnet segregation (public/private).
    - Custom routing and security.
    - Integration with other AWS services.
- **Real-World Use Case:** Hosting web applications with public-facing servers and private databases.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **VPC** service and create a new VPC.
3. Add subnets:
    - Create a **public subnet** and associate it with an internet gateway.

    - Create a **private subnet** and associate it with a NAT gateway for outgoing internet traffic.
4. Configure route tables:
    - Public subnet: Add a route to the internet gateway.
    - Private subnet: Add a route to the NAT gateway.
5. Launch EC2 instances:
    - Launch one instance in the public subnet (e.g., web server).
    - Launch another in the private subnet (e.g., database server).
6. Test communication between subnets.

Challenges and Solution:

- **Challenge:** Securely managing communication between public and private resources.
- **Solution:** Use security groups and route tables to control traffic flow.

Expected Outcome: At the end of this project, you will have a functional VPC with public and private subnets configured for secure resource hosting.

Project 7: Deploying a Relational Database with Amazon RDS

Objective: To deploy and connect to a relational database using Amazon RDS.

Resource Information: Amazon RDS (Relational Database Service)

- **Purpose:** Managed relational database service.
- **Key Features:**
    - Automated backups and software patching.
    - Scalability and high availability with Multi-AZ deployments.
    - Supports multiple database engines (e.g., MySQL, PostgreSQL, SQL Server).
- **Real-World Use Case:** Hosting a relational database for web applications.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **RDS** service.

3. Click **Create Database** and select **Standard Create**.
4. Choose a database engine (e.g., MySQL) and specify version.
5. Configure instance settings:
   - Select the instance class (e.g., `db.t3.micro`).
   - Set storage type and size.
6. Configure credentials:
   - Set master username and password.
7. Choose connectivity options:
   - Enable public access if needed and specify the VPC and subnet group.
8. Launch the RDS instance and wait for its status to become "Available."
9. Connect to the database:
   - Use an SQL client or application and provide the endpoint, username, and password.

Challenges and Solution:

- **Challenge:** Securing database access from unauthorized users.
- **Solution:** Use security groups to restrict access to specific IP addresses or instances.

Expected Outcome: You will have a functional RDS database ready for application integration.

Project 8: Configuring Aurora Database for High Availability

Objective: To deploy and test an Amazon Aurora database cluster with high availability.

Resource Information: Amazon Aurora

- **Purpose:** Managed relational database optimized for high performance and availability.
- **Key Features:**
  - Distributed, fault-tolerant storage.
  - Supports MySQL and PostgreSQL compatibility.
  - Automatic failover within seconds.
- **Real-World Use Case:** Powering critical web applications that require low-latency database access.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **RDS** service and choose **Create Database**.
3. Select **Amazon Aurora** and specify engine compatibility (e.g., MySQL).
4. Configure the Aurora cluster:
   - Choose instance class (e.g., `db.r6g.large`).
   - Enable Multi-AZ deployment for high availability.
5. Set credentials:
   - Provide a master username and password.
6. Define connectivity settings:
   - Select the VPC and subnet group.
7. Launch the Aurora cluster and wait for the instances to become available.
8. Connect to the cluster endpoint using an SQL client.
9. Simulate a failover:
   - Stop the primary instance to observe failover to the replica.

Challenges and Solution:

- **Challenge:** Ensuring database availability during failures.
- **Solution:** Aurora's automatic failover ensures minimal downtime.

Expected Outcome: You will have a highly available Aurora database cluster ready for production use.

Project 9: Hosting a Multi-Tier Application Using VPC and RDS

Objective: To deploy a multi-tier application with a front-end web server, application server, and database server in a secure VPC.

Resource Information:

- **VPC:** Provides network isolation for resources.
- **RDS:** Manages the database layer.
- **EC2:** Hosts the application and web servers.

Steps to Complete the Project:

1. Set up a custom VPC with public and private subnets.
2. Deploy EC2 instances:
   - Web server in the public subnet.
   - Application server in the private subnet.
3. Configure an RDS database in the private subnet.
4. Connect the application server to the RDS database using the endpoint.
5. Configure security groups to allow traffic:
   - Web server: Allow HTTP and HTTPS traffic.
   - Application server: Allow traffic from the web server.
   - RDS: Allow traffic only from the application server.
6. Test the application:
   - Access the web server and verify end-to-end functionality.

Challenges and Solution:

- **Challenge:** Securing inter-tier communication.
- **Solution:** Use VPC security groups and network ACLs to restrict traffic.

Expected Outcome: You will have a fully functional multi-tier application hosted securely in a VPC.

Project 10: Implementing VPC Peering for Cross-VPC Communication

Objective: To set up VPC peering between two VPCs and enable resource sharing.

Resource Information: Amazon VPC Peering

- **Purpose:** Enables communication between two VPCs.
- **Key Features:**
  - Low-latency, high-throughput network connectivity.
  - Works across accounts and regions.
- **Real-World Use Case:** Connecting applications hosted in separate VPCs for security or organizational purposes.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **VPC** service and create two separate VPCs (e.g., `VPC-A` and `VPC-B`).
3. Configure subnets and route tables for each VPC.
4. Create a VPC peering connection between `VPC-A` and `VPC-B`.
5. Update the route tables in both VPCs to allow traffic to the peered VPC.
6. Test connectivity by deploying EC2 instances in each VPC and pinging between them.

Challenges and Solution:

- **Challenge:** Ensuring secure and efficient cross-VPC communication.
- **Solution:** Use VPC peering to establish direct connectivity without exposing resources to the internet.

Expected Outcome: You will have two VPCs successfully connected via peering for resource sharing.

Project 11: Using AWS Transit Gateway for Centralized VPC Management

Objective: To implement AWS Transit Gateway to manage multiple VPCs and on-premises connections.

Resource Information: AWS Transit Gateway

- **Purpose:** Simplifies VPC-to-VPC and hybrid cloud connectivity.
- **Key Features:**
  - Centralized routing.
  - Scalable interconnect for multiple VPCs.
- **Real-World Use Case:** Organizations with multiple VPCs benefit from simplified network architecture.

Steps to Complete the Project:

1. Log in to the AWS Management Console.

2. Navigate to the **Transit Gateway** service and create a Transit Gateway.
3. Attach multiple VPCs to the Transit Gateway.
4. Update route tables for the attached VPCs to direct traffic through the Transit Gateway.
5. Test connectivity between VPCs via the Transit Gateway.

Challenges and Solution:

- **Challenge:** Managing connectivity between numerous VPCs and on-premises networks.
- **Solution:** Transit Gateway reduces complexity with centralized routing.

Expected Outcome: You will have a Transit Gateway connecting multiple VPCs efficiently.

Project 12: Setting Up an Application Load Balancer

Objective: To create and configure an Application Load Balancer (ALB) for distributing traffic across multiple EC2 instances.

Resource Information: Elastic Load Balancing (Application Load Balancer)

- **Purpose:** Distributes incoming application traffic across multiple targets.
- **Key Features:**
  - Layer 7 routing based on URL paths or hostnames.
  - Integration with AWS services like EC2, ECS, and Lambda.
  - Enhanced security with SSL/TLS termination.
- **Real-World Use Case:** Managing traffic for a scalable web application.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **EC2** service and launch two or more EC2 instances in the same VPC.
3. Configure a security group for the EC2 instances to allow HTTP traffic.
4. Navigate to the **Load Balancers** section under EC2 and click **Create Load Balancer**.
5. Choose **Application Load Balancer** and provide details:
   - Name: `MyAppLoadBalancer`
   - Scheme: Internet-facing

    - Listeners: HTTP (port 80)
6. Configure the load balancer’s VPC and subnets.
7. Set up the target group:
    - Create a target group for EC2 instances.
    - Add the previously launched EC2 instances as targets.
8. Review and create the ALB.
9. Test the load balancer by accessing its DNS name in a web browser.

Challenges and Solution:

- **Challenge:** Ensuring high availability of web applications.
- **Solution:** Use ALB to distribute traffic and handle failures automatically.

Expected Outcome: You will have a functional Application Load Balancer distributing traffic across EC2 instances.

Project 13: Configuring Auto Scaling Groups with EC2 Instances

Objective: To set up an Auto Scaling Group (ASG) that dynamically adjusts the number of EC2 instances based on demand.

Resource Information: Auto Scaling Groups

- **Purpose:** Ensures optimal number of instances to handle load.
- **Key Features:**
    - Dynamic scaling policies.
    - Health checks and automatic instance replacement.
    - Integration with Elastic Load Balancing.
- **Real-World Use Case:** Maintaining application availability during traffic spikes or drops.

Steps to Complete the Project:

1. Log in to the AWS Management Console.
2. Navigate to the **Launch Templates** section under EC2 and create a new launch template:

    - Specify instance details (e.g., AMI, instance type).
    - Include user data for initializing instances if needed.
3. Navigate to the **Auto Scaling Groups** section and click **Create Auto Scaling Group**.
4. Provide details:
    - Launch template: Select the previously created template.
    - VPC and subnets: Select appropriate subnets.
    - Load balancer: Attach the Application Load Balancer created in Project 1.
5. Configure scaling policies:
    - Define minimum, maximum, and desired number of instances.
    - Set up target tracking scaling policies (e.g., maintaining CPU utilization at 50%).
6. Review and create the Auto Scaling Group.
7. Test scaling by simulating load on the instances (e.g., using a stress testing tool).

Challenges and Solution:

- **Challenge:** Maintaining application performance during fluctuating demand.
- **Solution:** Use ASG with scaling policies to adjust resources dynamically.

Expected Outcome: You will have an Auto Scaling Group that adjusts the number of EC2 instances based on demand.

Project 14: Implementing a Secure ALB with SSL/TLS

Objective: To configure an Application Load Balancer with SSL/TLS for secure HTTPS traffic.

Resource Information: Application Load Balancer with SSL/TLS

- **Purpose:** Enhances security by encrypting traffic between clients and the load balancer.
- **Key Features:**
    - SSL/TLS certificate management.
    - Integration with AWS Certificate Manager (ACM).
- **Real-World Use Case:** Securely serving a web application to end-users.

Steps to Complete the Project:

1. Obtain an SSL/TLS certificate:
    - Navigate to **AWS Certificate Manager (ACM)**.
    - Request a public certificate for your domain.
    - Validate the domain using DNS or email verification.
2. Navigate to the **Load Balancers** section under EC2 and select the ALB created in Project 1.
3. Add an HTTPS listener:
    - Select HTTPS (port 443) as the protocol.
    - Attach the ACM certificate.
    - Configure the listener rules to forward traffic to the target group.
4. Update the security group for the load balancer to allow HTTPS traffic.
5. Test the HTTPS configuration by accessing the ALB's DNS name with `https://`.

Challenges and Solution:

- **Challenge:** Ensuring secure communication with minimal configuration complexity.
- **Solution:** Use ACM for certificate management and integrate it with ALB.

Expected Outcome: You will have a secure Application Load Balancer serving HTTPS traffic.

Project 15: Deploying a Highly Available Web Application with ALB and ASG

Objective: To deploy a highly available web application using an Application Load Balancer and Auto Scaling Group.

Resource Information:

- **ALB:** Distributes traffic across multiple targets.
- **ASG:** Ensures scalability and availability.

Steps to Complete the Project:

1. Set up an Application Load Balancer (refer to Project 1).
2. Create an Auto Scaling Group (refer to Project 2).
3. Launch EC2 instances with a web server application in the ASG.
4. Attach the ASG to the ALB’s target group.
5. Configure health checks in the target group to monitor instance availability.
6. Test high availability:
    - Simulate instance failure and observe traffic redistribution.
    - Simulate traffic spikes and verify instance scaling.

Challenges and Solution:

- **Challenge:** Achieving high availability and fault tolerance.
- **Solution:** Combine ALB and ASG to ensure traffic distribution and scaling.

Expected Outcome: You will have a highly available web application with traffic distributed and scaled automatically.

Project 16: Implementing Weighted Target Groups with ALB

Objective: To use weighted target groups in an Application Load Balancer for traffic routing.

Resource Information: Weighted Target Groups

- **Purpose:** Enables gradual traffic shifting for blue/green deployments.
- **Key Features:**
    - Weighted routing rules.
    - Support for canary releases.
- **Real-World Use Case:** Deploying application updates with minimal risk.

Steps to Complete the Project:

1. Create two target groups:
    - Group A for the existing application version.
    - Group B for the new application version.

2. Configure the ALB with weighted routing:
    - Navigate to the **Listeners** tab.
    - Add a rule to distribute traffic between Group A and Group B with specified weights (e.g., 80% to Group A, 20% to Group B).
3. Deploy the new application version to instances in Group B.
4. Test traffic distribution and monitor application performance.

Challenges and Solution:

- **Challenge:** Managing risk during application updates.
- **Solution:** Use weighted routing to gradually shift traffic to the new version.

Expected Outcome: You will have an ALB configured for weighted traffic distribution between target groups.

Project 17: Setting Up a Custom Domain Name with Route 53

Objective: To register a custom domain name and configure it for a web application hosted on EC2.

Resource Information: Amazon Route 53

- **Purpose:** Domain Name System (DNS) and domain registration service.
- **Key Features:**
    - Domain registration and DNS management.
    - Health checks and traffic policies.
    - Integration with AWS services.
- **Real-World Use Case:** Hosting a website with a custom domain name.

Steps to Complete the Project:

1. Log in to the AWS Management Console and navigate to **Route 53**.
2. Register a domain name (e.g., `mywebsite.com`).
    - Go to the **Domains** section and select **Register Domain**.
    - Complete the registration process.

3. Set up a hosted zone for the domain:
    - Navigate to the **Hosted Zones** section and create a new hosted zone.
    - Note the nameservers provided by Route 53.
4. Update DNS records:
    - Add an A record to map the domain to the IP address of your EC2 instance.
5. Test the domain by entering it in a browser to confirm it resolves to the web application.

Challenges and Solution:

- **Challenge:** Ensuring proper DNS propagation.
- **Solution:** Allow up to 48 hours for DNS changes to propagate fully.

Expected Outcome: You will have a custom domain name pointing to your web application.

Project 18: Configuring a Subdomain in Route 53

Objective: To create and manage a subdomain (e.g., `api.mywebsite.com`) for a specific application endpoint.

Resource Information: Subdomain Management

- **Purpose:** Organize and route traffic to specific endpoints or services.
- **Key Features:**
    - Flexible record types (A, CNAME, etc.).
    - Easy integration with existing hosted zones.
- **Real-World Use Case:** Separating APIs or different services under a single domain.

Steps to Complete the Project:

1. Log in to the AWS Management Console and navigate to the **Route 53** hosted zone for your domain.
2. Add a new record set:
    - Name: `api.mywebsite.com`.
    - Type: A or CNAME (depending on the resource).
    - Value: Enter the IP address or hostname of the resource.

3. Save the record and test the subdomain by accessing it in a browser or using a tool like `nslookup`.

Challenges and Solution:

- **Challenge:** Avoiding conflicts with existing records.
- **Solution:** Ensure the subdomain name does not overlap with other records.

Expected Outcome: You will have a functional subdomain pointing to a specific application or resource.

Project 19: Configuring Health Checks for a Web Application

Objective: To set up health checks in Route 53 to monitor the availability of a web application.

Resource Information: Route 53 Health Checks

- **Purpose:** Monitors the health of endpoints and integrates with DNS failover.
- **Key Features:**
  - HTTP, HTTPS, and TCP checks.
  - Alarms using CloudWatch.
- **Real-World Use Case:** Ensuring high availability with automatic failover.

Steps to Complete the Project:

1. Log in to the AWS Management Console and navigate to **Route 53**.
2. Go to the **Health Checks** section and click **Create Health Check**.
3. Configure the health check:
   - Specify the endpoint (e.g., `mywebsite.com`).
   - Choose the protocol (HTTP/HTTPS/TCP).
   - Set the threshold for failed requests.
4. Associate the health check with a DNS record:
   - Navigate to the hosted zone for the domain.
   - Edit the record set and enable failover.
5. Test the health check by simulating an endpoint failure.

Challenges and Solution:

- **Challenge:** Avoiding false positives or negatives in health checks.
- **Solution:** Configure appropriate thresholds and response intervals.

Expected Outcome: You will have a health check monitoring your application and routing traffic based on endpoint availability.

Project 20: Implementing Weighted Routing Policy

Objective: To configure a weighted routing policy to distribute traffic between two application versions hosted on different EC2 instances.

Resource Information: Weighted Routing in Route 53

- **Purpose:** Distributes traffic based on specified weights.
- **Key Features:**
  - Gradual traffic shifting.
  - Flexible traffic management.
- **Real-World Use Case:** Blue/green deployment or A/B testing.

Steps to Complete the Project:

1. Launch two EC2 instances hosting different versions of an application.
2. Navigate to the **Route 53** hosted zone for your domain.
3. Add two A records with the following configurations:
   - Record 1: Points to the IP of Instance A with weight 70.
   - Record 2: Points to the IP of Instance B with weight 30.
4. Test traffic distribution by accessing the domain repeatedly and verifying the response.
5. Adjust weights as needed for traffic management.

Challenges and Solution:

- **Challenge:** Balancing traffic effectively during deployments.
- **Solution:** Use weighted routing to gradually shift traffic to the new version.

Expected Outcome: You will have a domain distributing traffic between two application versions based on weights.

Project 21: Configuring Geolocation Routing Policy

Objective: To set up geolocation-based routing to direct users to region-specific application endpoints.

Resource Information: Geolocation Routing in Route 53

- **Purpose:** Directs traffic based on user location.
- **Key Features:**
  - Customized user experiences.
  - Enhanced application performance.
- **Real-World Use Case:** Providing localized content or services.

Steps to Complete the Project:

1. Launch application endpoints in multiple regions (e.g., US-East and EU-West).
2. Navigate to the **Route 53** hosted zone for your domain.
3. Add A records with geolocation routing:
   - Record 1: Points to the US endpoint, with location set to North America.
   - Record 2: Points to the EU endpoint, with location set to Europe.
4. Test routing by accessing the domain from different locations using VPNs or proxies.

Challenges and Solution:

- **Challenge:** Handling users from undefined locations.
- **Solution:** Set a default record for undefined locations.

Expected Outcome: You will have geolocation-based routing directing users to the nearest endpoint.

Project 22: Analyzing Log Data with Amazon Athena

Objective: Use Amazon Athena to query and analyze log data stored in Amazon S3.

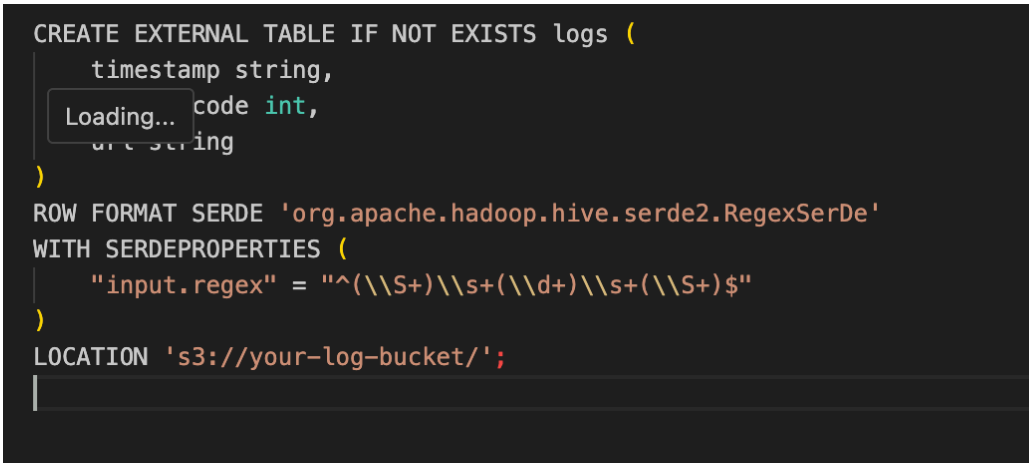
Resources Information:

- **Amazon Athena:** A serverless query service for analyzing data directly in Amazon S3 using SQL.
- **Amazon S3:** Provides scalable object storage for storing log data.

Steps to Complete the Project:

1. **Set Up Log Storage in S3:**
   - Create an S3 bucket and upload sample log files (e.g., web server logs).
2. **Set Up Athena:**
   - Open the Athena console.
   - Configure the query result location by selecting an S3 bucket or folder.
3. **Create a Table in Athena:**
   - Define the schema for the log data using SQL DDL.

     Example table creation query:

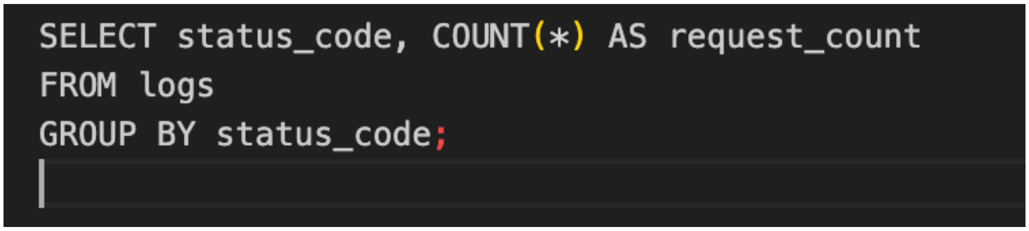


```
CREATE EXTERNAL TABLE IF NOT EXISTS logs (
    timestamp string,
 Loading... code int,
    url string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    "input.regex" = "^(\\S+)\\s+(\\d+)\\s+(\\S+)$"
)
LOCATION 's3://your-log-bucket/';
```

4. **Run Queries in Athena:**

   - Query the data to analyze trends like frequent errors or peak traffic times.

Example query:



```
SELECT status_code, COUNT(*) AS request_count
FROM logs
GROUP BY status_code;
```

Challenges and Solution:

- **Challenge:** Handling large-scale log files efficiently.
- **Solution:** Use partitioning by date or log type to improve query performance.

Expected Outcome:
You will be able to analyze log data directly in S3 using SQL, gaining insights into traffic patterns and error occurrences.

Project 23: Creating Interactive Dashboards with Amazon QuickSight

Objective: Use Amazon QuickSight to create interactive dashboards based on data queried from Amazon Athena.

Resources Information:

- **Amazon QuickSight:** A business intelligence service to build dashboards and visualizations.
- **Amazon Athena:** Provides the data source for QuickSight dashboards.
- **Amazon S3:** Stores the raw data for analysis.

Steps to Complete the Project:

1. **Prepare Data in Athena:**
   - Use the log table from **Project 1** and clean/aggregate the data for visualization.

2. **Set Up Amazon QuickSight:**
   - Sign up for QuickSight and choose the Standard or Enterprise edition.
   - Connect QuickSight to Athena by setting up a new data source.
3. **Create a Dataset in QuickSight:**
   - Select the Athena database and table created in the previous project.
   - Perform data transformations if needed (e.g., filtering for specific time periods).
4. **Build Visualizations:**
   - Create charts like bar graphs, pie charts, or line charts to represent traffic trends, status codes, or error distribution.
   - Add filters to enable interactive exploration of the data.
5. **Publish the Dashboard:**
   - Save the visualizations and create a dashboard.
   - Share it with stakeholders or embed it into applications.

Challenges and Solution:

- **Challenge:** Ensuring data refreshes automatically for real-time insights.
- **Solution:** Configure scheduled refreshes in QuickSight to pull updated data from Athena.

Expected Outcome:
You will create an interactive dashboard visualizing key metrics from your log data, providing actionable insights to stakeholders.

Project 24: Querying CloudTrail Logs with Amazon Athena

Objective: Use Amazon Athena to query AWS CloudTrail logs for auditing and security analysis.

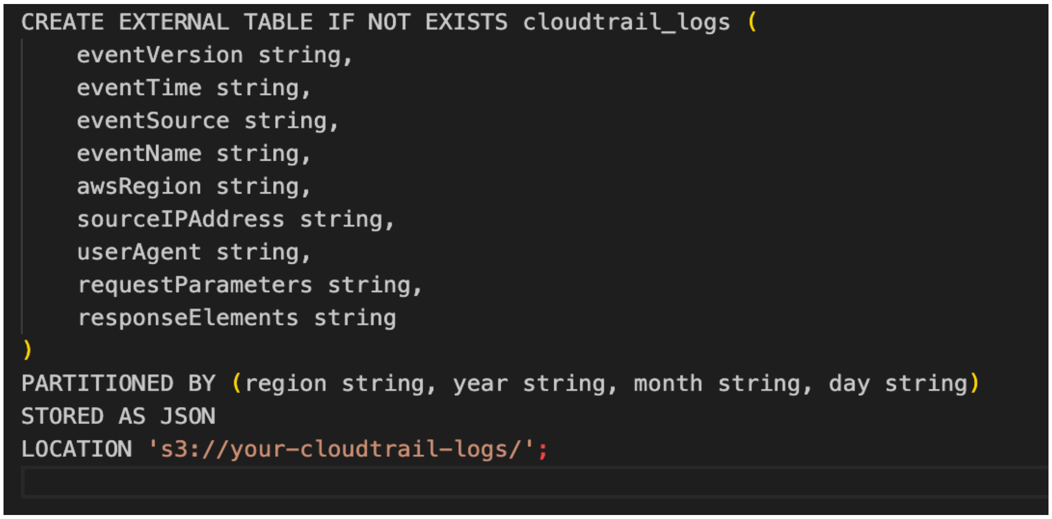
Resources Information:

- **Amazon Athena:** Enables querying CloudTrail logs stored in S3 using SQL.
- **Amazon CloudTrail:** Provides a history of AWS API calls for auditing and compliance.
- **Amazon S3:** Stores CloudTrail logs.

Steps to Complete the Project:

1. **Enable CloudTrail Logging:**
   - Go to the AWS CloudTrail console and enable logging.
   - Configure an S3 bucket as the destination for CloudTrail logs.
2. **Set Up Athena:**
   - Open the Athena console and configure the query results location.
3. **Create a Table in Athena for CloudTrail Logs:**
   - Define a schema for CloudTrail logs using SQL DDL.

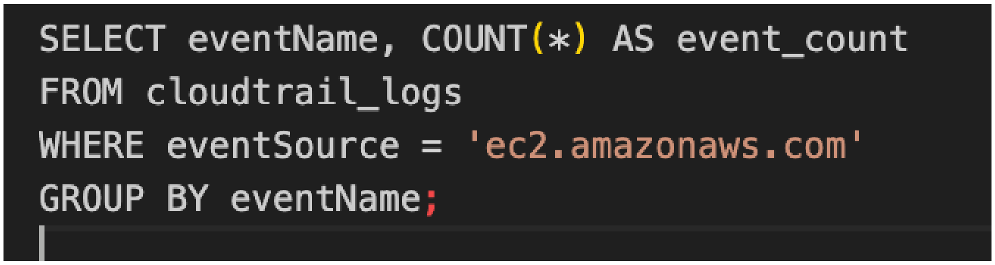
     Example table creation query:



```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudtrail_logs (
    eventVersion string,
    eventTime string,
    eventSource string,
    eventName string,
    awsRegion string,
    sourceIPAddress string,
    userAgent string,
    requestParameters string,
    responseElements string
)
PARTITIONED BY (region string, year string, month string, day string)
STORED AS JSON
LOCATION 's3://your-cloudtrail-logs/';
```

4. **Run Queries in Athena:**
   - Query the data to analyze security events or API activity.

     Example query:



```
SELECT eventName, COUNT(*) AS event_count
FROM cloudtrail_logs
WHERE eventSource = 'ec2.amazonaws.com'
GROUP BY eventName;
```

Challenges and Solution:

- **Challenge:** Querying large CloudTrail datasets efficiently.
- **Solution:** Use partitioning (e.g., by region and date) to improve performance.

Expected Outcome:
You will be able to analyze AWS API activity, identify security anomalies, and ensure compliance.

Project 25: Building Sales Dashboards with Amazon QuickSight

Objective: Create a sales dashboard with Amazon QuickSight to visualize sales data stored in S3.

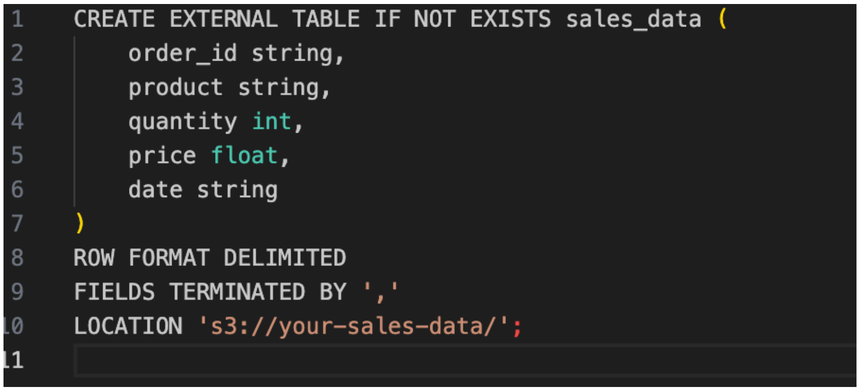
Resources Information:

- **Amazon QuickSight:** A tool for business intelligence and visualization.
- **Amazon S3:** Stores raw sales data in CSV or Parquet format.
- **Amazon Athena:** Queries sales data for QuickSight visualization.

Steps to Complete the Project:

1. **Upload Sales Data to S3:**
    - Store sales data (e.g., `sales.csv`) in an S3 bucket.
2. **Set Up Athena for Sales Data:**
    - Define a table schema in Athena for the sales data.

      Example table creation query:



```sql
CREATE EXTERNAL TABLE IF NOT EXISTS sales_data (
    order_id string,
    product string,
    quantity int,
    price float,
    date string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://your-sales-data/';

```

3. **Set Up Amazon QuickSight:**
    - Connect QuickSight to Athena and import the sales data table.
4. **Build Visualizations:**
    - Create charts such as total revenue, sales by region, and top products.
    - Add filters for date ranges or product categories.
5. **Share the Dashboard:**
    - Publish the dashboard and share it with stakeholders.

Challenges and Solution:

- **Challenge:** Handling large datasets.
- **Solution:** Use Parquet files for better compression and query performance.

Expected Outcome:
You will have an interactive sales dashboard providing insights into revenue trends and product performance.

Project 26: Performing Data Transformation with Athena and Visualizing in QuickSight

Objective: Transform raw data using Amazon Athena and visualize the results with Amazon QuickSight.

Resources Information:

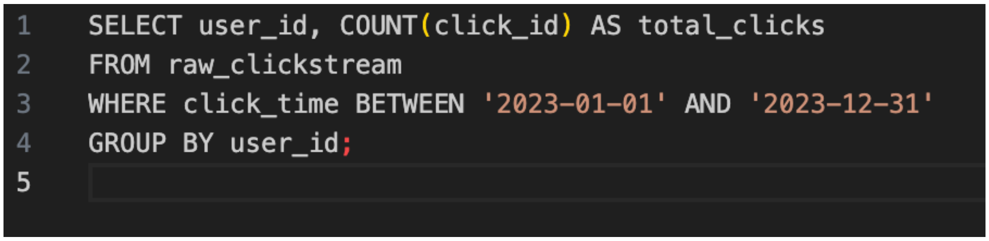
- **Amazon Athena:** Provides SQL-based data transformation.
- **Amazon QuickSight:** Visualizes transformed data.
- **Amazon S3:** Stores raw data for transformation.

Steps to Complete the Project:

1. **Upload Raw Data to S3:**
    - Place raw data (e.g., clickstream data) into an S3 bucket.

2. **Transform Data with Athena:**
    - Write SQL queries to clean and aggregate the data.

      Example: Removing duplicates and calculating total clicks per user.



```sql
SELECT user_id, COUNT(click_id) AS total_clicks
FROM raw_clickstream
WHERE click_time BETWEEN '2023-01-01' AND '2023-12-31'
GROUP BY user_id;

```

3. **Set Up QuickSight for Visualization:**
    - Connect QuickSight to Athena and import the transformed data.
4. **Create a Dashboard:**
    - Build visualizations such as user activity trends and engagement metrics.

Challenges and Solution:

- **Challenge:** Ensuring accurate data transformation.
- **Solution:** Validate transformed data with sample queries in Athena.

Expected Outcome:
You will transform raw data into actionable insights and visualize it in QuickSight.

Project 27: Monitoring EC2 Performance with Amazon CloudWatch

Objective: Use Amazon CloudWatch to monitor the performance and health of an EC2 instance.

Resources Information:

- **Amazon CloudWatch:** Provides monitoring and observability services.
- **Amazon EC2:** Virtual servers used for hosting applications.

Steps to Complete the Project:

1. **Launch an EC2 Instance:**
   - Log in to the AWS Management Console.
   - Launch an EC2 instance with Amazon Linux 2 or Ubuntu as the AMI.
2. **Set Up CloudWatch Agent:**
   - Install the CloudWatch agent on the EC2 instance.
   - Configure the agent using the CloudWatch agent configuration file to monitor memory and disk usage.
3. **Monitor Metrics in CloudWatch:**
   - Open the CloudWatch console.
   - View metrics such as CPU utilization, disk reads/writes, and network traffic.
4. **Set Up Alarms:**
   - Create a CloudWatch alarm to monitor high CPU utilization.
   - Configure the alarm to send notifications via Amazon SNS.

Challenges and Solution:

- **Challenge:** Capturing custom metrics.
- **Solution:** Use the CloudWatch agent to send additional metrics like memory usage.

Expected Outcome:
You will monitor EC2 performance metrics and receive alerts for resource anomalies.

Project 28: Auditing API Activity with AWS CloudTrail

Objective: Enable AWS CloudTrail to audit and log API activity for governance and compliance.

Resources Information:

- **AWS CloudTrail:** Logs AWS API calls for security and auditing.
- **Amazon S3:** Stores CloudTrail logs.

Steps to Complete the Project:

1. **Enable CloudTrail:**
   - Open the AWS CloudTrail console.
   - Create a trail and configure an S3 bucket to store logs.
2. **Monitor API Activity:**
   - Log into the CloudTrail console to view recent events.
   - Filter events by source (e.g., `ec2.amazonaws.com`) or action (e.g., `RunInstances`).
3. **Set Up Insights for Anomalies:**
   - Enable CloudTrail Insights to detect unusual activity patterns.
4. **Query Logs with Athena:**
   - Use Amazon Athena to analyze the CloudTrail logs for specific actions or events.

Challenges and Solution:

- **Challenge:** Managing large volumes of logs.
- **Solution:** Use Athena to query logs efficiently and archive older data.

Expected Outcome:
You will have a detailed audit log of all AWS API activity and the ability to analyze it for security and compliance.

Project 29: Ensuring Resource Compliance with AWS Config

Objective: Use AWS Config to monitor resource compliance and detect non-compliant configurations.

Resources Information:

- **AWS Config:** Tracks and records AWS resource configurations for compliance.

Steps to Complete the Project:

1. **Set Up AWS Config:**
   - Open the AWS Config console and enable resource recording.
   - Choose the resources to monitor (e.g., EC2 instances, S3 buckets).
2. **Set Up Rules:**
   - Create rules to check compliance (e.g., ensuring S3 buckets are encrypted).
   - Use AWS Managed Rules or define custom rules with AWS Lambda.
3. **View Compliance Reports:**
   - Open the AWS Config dashboard to view compliance status.
   - Identify non-compliant resources and take corrective action.
4. **Remediate Issues:**
   - Automate remediation using AWS Config rules or manual updates to resource configurations.

Challenges and Solution:

- **Challenge:** Managing compliance for multiple resources.
- **Solution:** Use AWS Config aggregation to monitor compliance across accounts and regions.

Expected Outcome:
You will monitor resource configurations and ensure compliance with organizational policies.

Project 30: Creating Custom Dashboards in CloudWatch

Objective: Build a custom CloudWatch dashboard to visualize metrics from multiple AWS services.

Resources Information:

- **Amazon CloudWatch:** Provides monitoring and observability for AWS resources.

Steps to Complete the Project:

1. **Identify Metrics to Monitor:**
   - Select key metrics from EC2, RDS, and S3 for monitoring.
2. **Create a Dashboard:**
   - Open the CloudWatch console and create a new dashboard.
   - Add widgets for the selected metrics (e.g., CPU utilization for EC2, read/write latency for RDS).
3. **Customize Dashboard Layout:**
   - Arrange widgets to optimize readability and add annotations if needed.
4. **Share the Dashboard:**
   - Share the dashboard with team members or embed it in a webpage.

Challenges and Solution:

- **Challenge:** Ensuring real-time updates for critical metrics.
- **Solution:** Configure high-resolution metrics and refresh intervals in the dashboard.

Expected Outcome:
You will have a unified dashboard displaying real-time metrics for your AWS environment.

Project 31: Setting Up Real-Time Alerts with CloudWatch

Objective:
Use CloudWatch to create real-time alerts for abnormal behavior in AWS resources.

Resources Information:

- **Amazon CloudWatch:** Provides monitoring and alerting for AWS resources.

Steps to Complete the Project:

1. **Define Metrics to Monitor:**
   - Choose critical metrics like EC2 CPU utilization, S3 bucket requests, or RDS connection counts.

2. **Create Alarms in CloudWatch:**
   - Set thresholds for abnormal behavior (e.g., CPU usage > 80%).
   - Configure actions to send notifications via Amazon SNS.
3. **Test Alerts:**
   - Simulate a threshold breach and verify that alerts are triggered.
4. **Respond to Alerts:**
   - Use the alerts to troubleshoot and resolve issues promptly.

Challenges and Solution:

- **Challenge:** Managing false alarms.
- **Solution:** Use anomaly detection in CloudWatch to reduce false positives.

Expected Outcome:
You will set up real-time alerts for proactive monitoring and issue resolution.

Project 32: Predicting Housing Prices Using Amazon SageMaker

Objective: To develop a machine learning model to predict housing prices based on various features such as location, size, and amenities.

Resource Information: Amazon SageMaker

Purpose: Provides a scalable environment to train, deploy, and monitor machine learning models.

Key Features:

- Fully managed Jupyter notebooks.
- Built-in algorithms for regression models.

Real-World Use Case: Estimating property prices for real estate applications.

Steps to Complete the Project:

1. **Prepare the Dataset:**
   - Collect housing data from a public dataset like Kaggle or Zillow.
   - Clean the data to remove invalid entries, handle missing values, and normalize numeric features.
   - Save the cleaned dataset as a CSV file and upload it to an S3 bucket.
2. **Set Up SageMaker Notebook Instance:**
   - Log in to the AWS Management Console and navigate to Amazon SageMaker.
   - Create a new Jupyter notebook instance and attach an IAM role with access to S3.
3. **Data Exploration and Preprocessing:**
   - Load the dataset from S3 into the notebook using Pandas.
   - Explore the data with visualizations using Matplotlib or Seaborn.
   - Split the data into training (80%) and testing (20%) datasets.
4. **Train the Model:**
   - Use SageMaker's built-in XGBoost algorithm for regression.
   - Create a training job by specifying the algorithm, training data location in S3, and hyperparameters.
   - Monitor the training process using the SageMaker console.
5. **Deploy the Model:**
   - Deploy the trained model as an endpoint using SageMaker’s model hosting services.
   - Test the endpoint by sending JSON requests with sample input data and validate predictions.
6. **Evaluate the Model:**
   - Use evaluation metrics such as Mean Squared Error (MSE) or R-squared to measure the model’s performance.
   - If needed, tune hyperparameters and retrain the model.

Challenges and Solution:

- **Challenge:** Handling missing values in the dataset.
- **Solution:** Use SageMaker preprocessing utilities or Pandas methods to impute missing values.

Project 33: Sentiment Analysis Using Amazon Comprehend

Objective: To analyze customer reviews and classify them as positive, neutral, or negative.
Resource Information: Amazon Comprehend
Purpose: Automates text analysis and natural language processing tasks.
Key Features:

- Pre-trained NLP models.
- Language detection and sentiment analysis.

Real-World Use Case: Understanding customer feedback for e-commerce platforms.

Steps to Complete the Project:

1. **Collect and Upload Data:**
   - Gather customer reviews from a source such as an e-commerce website or feedback forms.
   - Save the reviews in a CSV or JSON format and upload the file to an S3 bucket.
2. **Analyze Sentiment Using Amazon Comprehend:**
   - Log in to the AWS Management Console and navigate to Amazon Comprehend.
   - Use the “Analyze Text” feature to process the dataset for sentiment analysis.
   - Export the results to an S3 bucket.
3. **Visualize Results:**
   - Use Amazon QuickSight to create graphs showing the sentiment distribution.
   - Create dashboards to display trends over time.
4. **Generate Insights:**
   - Identify products with consistently negative reviews for improvement.
   - Highlight products with positive reviews for promotional campaigns.

Challenges and Solution:

- **Challenge:** Handling multilingual data in reviews.
- **Solution:** Use Comprehend's language detection API to preprocess reviews based on their language.

Here are AWS KMS (Key Management Service)-related projects in the detailed report format:

Project 34: Encrypting S3 Data Using AWS KMS

Objective: To secure sensitive data stored in S3 buckets by encrypting it with customer-managed keys (CMKs) from AWS KMS.
Resource Information: AWS Key Management Service (KMS), Amazon S3
Purpose: Protects data at rest and ensures compliance with security standards.
Key Features:

- Granular access control for encryption keys.
- Seamless integration with S3 for automatic encryption.

Real-World Use Case: Encrypting customer data in S3 for regulatory compliance.

Steps to Complete the Project:

1. **Create a Customer-Managed Key (CMK):**
   - Navigate to the AWS KMS console and create a new CMK.
   - Assign alias and description to the CMK for easy identification.
   - Set up key policies to grant access to specific IAM users or roles.
2. **Set Up an S3 Bucket:**
   - Create a new S3 bucket to store sensitive data.
   - Enable default encryption for the bucket and select AWS KMS as the encryption type.
   - Choose the CMK created earlier as the encryption key.

3. **Upload Encrypted Data:**
    - Upload files to the S3 bucket using the AWS Management Console, CLI, or SDK.
    - Verify that the files are encrypted by checking the "Server-Side Encryption" field in S3 properties.
4. **Access Encrypted Data:**
    - Assign IAM permissions to users or applications that need access to the bucket and the CMK.
    - Test access by downloading and decrypting files using the AWS CLI or SDK.
5. **Monitor Key Usage:**
    - Use AWS CloudTrail to track key usage and access patterns.
    - Set up Amazon CloudWatch alarms for unauthorized access attempts.

Challenges and Solution:

- **Challenge:** Managing access to sensitive keys.
- **Solution:** Use IAM policies and key policies to enforce strict access controls.

Project 35: Encrypting EBS Volumes with AWS KMS

Objective: To protect data on EBS volumes by encrypting them using AWS KMS keys.
Resource Information: AWS Key Management Service (KMS), Amazon Elastic Block Store (EBS)
Purpose: Ensures data security for compute instances using encrypted storage.
Key Features:

- Transparent encryption for EBS volumes.
- Support for customer-managed keys (CMKs).

Real-World Use Case: Encrypting sensitive application logs stored on EBS volumes.

Steps to Complete the Project:

1. **Create a Customer-Managed Key (CMK):**
    - Navigate to the KMS console and create a CMK for EBS encryption.
    - Configure key rotation and access policies.

2. **Launch an EC2 Instance:**
   - Create an EC2 instance and attach a new EBS volume.
   - Choose the option to enable encryption and select the CMK created earlier.
3. **Verify Encryption:**
   - Log in to the EC2 instance and create files on the encrypted volume.
   - Verify that the volume is encrypted in the EC2 console by checking the encryption field.
4. **Backup and Restore:**
   - Take a snapshot of the encrypted volume.
   - Restore the snapshot to create a new encrypted volume.
5. **Monitor Key Usage:**
   - Use CloudTrail and CloudWatch to monitor access to the CMK.
   - Set up alerts for unusual activities.

Challenges and Solution:

- **Challenge:** Key rotation for long-term security.
- **Solution:** Enable automatic key rotation for the CMK to minimize manual intervention.

Project 36: Securely Sharing Data Between AWS Accounts Using AWS KMS

Objective: To share encrypted data stored in S3 between AWS accounts while maintaining security using KMS.

Resource Information: AWS Key Management Service (KMS), Amazon S3

Purpose: Enables secure inter-account data sharing for collaborative projects.

Key Features:

- Cross-account key sharing.
- Fine-grained access control with KMS policies.

Real-World Use Case: Sharing financial reports between corporate departments in different AWS accounts.

Steps to Complete the Project:

1. **Create a Shared CMK:**
   - In the source account, create a CMK and modify its key policy to allow access to the destination account.
2. **Encrypt and Upload Data:**
   - Encrypt the data locally or use S3 encryption with the shared CMK.
   - Upload the encrypted data to an S3 bucket in the source account.
3. **Grant S3 Access:**
   - Update the S3 bucket policy to grant the destination account access to the bucket and objects.
4. **Access Data from Destination Account:**
   - Use the shared CMK to decrypt the data after downloading it from S3.
   - Test access permissions and ensure the decryption process works.
5. **Monitor Key and Data Access:**
   - Use CloudTrail to track API calls to KMS and S3.

Challenges and Solution:

- **Challenge:** Configuring cross-account access securely.
- **Solution:** Use resource-based policies and role assumptions to ensure minimal privilege access.
