# Automated Secret Rotation using AWS Secrets Manager

## Revision History

| Version | Author | Description of version | Date completed |
|---|---|---|---|
| v1.0 | | | |

## ● Introduction

This project introduces an automated secret rotation system using AWS Secrets Manager, focusing on multiple MySQL root user passwords. The primary goal is to enhance security by regularly updating sensitive credentials. Leveraging AWS

Lambda functions and cloud-native architecture, the system ensures operational efficiency, scalability, and maintainability.

---

- ## Problem statement

Traditional methods of managing static credentials, especially in scenarios involving privileged accounts like MySQL root users, pose significant security risks. The persistent use of unaltered passwords increases the likelihood of unauthorized access, data breaches, and system vulnerabilities.

1. **Manual password management:** Manual password management for clients is inefficient and prone to errors.
2. **Security concerns:** Storing passwords in plain text within the database for a longer period poses significant security risks if the database is compromised.
3. **Notifications and on-demand changes:** Need for automated password rotation, notification, and on-demand password change capabilities.

---

- ## Goals

1. **Automate password rotation:** Implement a Python script to automatically rotate passwords of the multiple root users for MySQL database every n days.
2. **Secure password storage:** Store/Retrieve passwords from AWS Secret Manager for enhanced security and access control.
3. **User notifications:** Send notifications to users before and immediately after changes are made.
4. **Admin on-demand changes:** Enable admins to trigger password changes for specific clients as needed, with immediate notifications and resetting of the n-day rotation cycle.
5. **Integration:** Integrate with existing MySQL databases.
6. **Development:** Develop a reliable and error-resistant Python script

---

## Design Solution

### Key Components

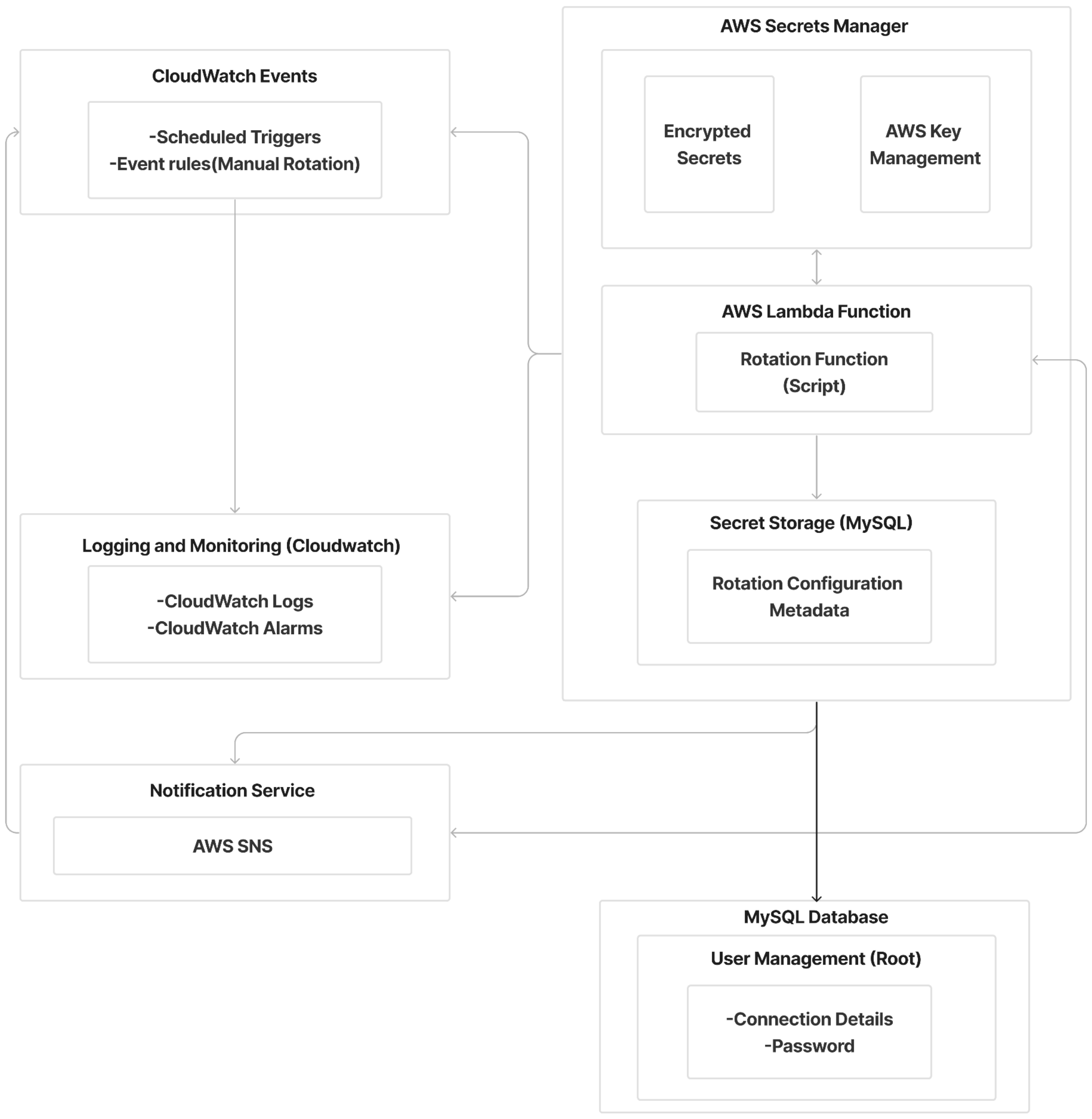
The automated secret rotation system comprises several key components, each playing a crucial role in achieving the project's objectives.

- AWS Lambda Functions:
  - Rotation Function: Executes the rotation logic for MySQL root user passwords triggered by predefined intervals or manual requests.
  - Reset Cycle Function: Manages the reset of the rotation cycle when a manual rotation is initiated.
- AWS Secrets Manager:
  - Secret Storage: Stores MySQL root user credentials securely, allowing for centralized management.
  - Rotation Configuration: Manages rotation policies, intervals, and metadata associated with each secret.
- MySQL Database:
  - User Management: MySQL root user accounts and their associated permissions are configured and maintained.
  - Connection Details: Hostname, username, and initial password details are stored securely.
- CloudWatch Events:
  - Scheduled Triggers: Initiates the rotation function at specified intervals to ensure timely updates.
  - Event Rules: Listens for manual rotation requests, triggering the necessary Lambda functions.
- Logging and Monitoring:
  - CloudWatch Logs: Captures logs generated by Lambda functions for auditability and troubleshooting.
  - CloudWatch Alarms: Monitors critical system metrics, triggering alerts for any anomalies or issues.
- IAM Roles:
  - Lambda Execution Role: Grants necessary permissions for Lambda functions to interact with Secrets Manager, CloudWatch, and other services.
  - Secrets Manager Access Role: Ensures secure communication between Secrets Manager and the MySQL database.
- Configuration Management:

    - Environment Variables: Houses configuration parameters like database connection details, rotation intervals, and reset cycle flags.
    - Parameter Store: Stores and retrieves sensitive configuration parameters securely.
- AWS Simple Notification Service (SNS):
    - Sends notifications on successful MySQL secret rotations.
    - Manages topics for publishing messagesSubscribers receive notifications through various protocols (Lambda)

---

- Architecture Overview:



In this architecture overview:

- AWS Lambda Functions handle rotation and reset cycle logic.
- AWS Secrets Manager stores and manages MySQL root user credentials and rotation configuration.
- MySQL Database manages user accounts and connection details.
- AWS SNS manages the notification system for the secret notification process.

- CloudWatch Events trigger scheduled rotations and listen for manual rotation requests.
- Logging and Monitoring using CloudWatch capture logs and monitor system metrics.

---

- Steps Involved:

## Phase 1: Initialization and Setup

AWS Account Setup:

- Create or use an existing AWS account.
- Ensure appropriate IAM user roles and permissions.

Secrets Manager Configuration:

- Create a new secret in AWS Secrets Manager for storing MySQL root user credentials.
- Configure rotation policies and rotation frequency.

IAM Role for Lambda:

- Create an IAM role with the necessary permissions for Lambda functions to interact with Secrets Manager, CloudWatch, and other services.

## Phase 2: Lambda Function Development

Lambda Rotation Function:

- Develop a Lambda function (lambda_function.py) to execute the rotation logic for MySQL root user passwords.
- Integrate with AWS Secrets Manager to retrieve and update credentials.

## Phase 3: MySQL Database Configuration

MySQL User Management:

- Configure MySQL root user accounts with the necessary permissions.
- Ensure appropriate connection details are stored securely.

## Phase 4: Notification System Integration

SNS Topic Setup:

- Create an SNS (Simple Notification Service) topic in the AWS Management Console.

Subscribe Endpoints:

- Subscribe relevant endpoints (e.g., email addresses, Lambda function ARNs) to the SNS topic.

Update Lambda Function:

- Modify the Lambda rotation function code to include notification functionality using the SNS topic.

Test Notification System:

- Trigger the Lambda function and validate that notifications are sent to the subscribed endpoints.

## Phase 5: CloudWatch Events and Triggers

CloudWatch Event Rules:

- Set up CloudWatch Event Rules to trigger the rotation function at scheduled intervals (e.g., every 30 days).
- Implement event rules to listen for manual rotation requests.

## Phase 6: Logging and Monitoring

CloudWatch Logging:

- Configure CloudWatch Logs to capture logs generated by the Lambda functions for auditability and troubleshooting.

CloudWatch Alarms:

- Set up CloudWatch Alarms to monitor critical system metrics, triggering alerts for any anomalies or issues.

## Phase 7: Testing and Validation

Testing the System:

- Manually trigger rotations to validate the rotation function.
- Confirm that logs are generated and alarms are triggered as expected.

System Validation:

- Monitor the rotation cycles in Secrets Manager.
- Validate that the MySQL root user passwords are being updated according to the defined rotation frequency.

## Phase 8: Documentation and Knowledge Transfer

Implementation Guide:

- Document the steps for setting up and configuring the automated secret rotation system.
- Include details on Lambda function code, Notification system, configuration parameters, and troubleshooting steps.

Knowledge Transfer:

- Share documentation and implementation details with the relevant team members.
- Conduct training sessions if necessary.

## Phase 9: Deployment to Production

Deployment Considerations:

- Plan the deployment to production environments, considering any downtime or impact on operations.
- Ensure proper communication and coordination with stakeholders.

Monitoring in Production:

- Continuously monitor the system in the production environment.
- Review logs, alarms, and rotation cycles regularly.

## Phase 10: Continuous Improvement

Feedback and Iteration:

- Gather feedback from users and stakeholders.
- Iterate on the system based on lessons learned and evolving security requirements.

Scaling Considerations:

- Assess the scalability of the solution as the number of systems and credentials grows.
- Implement additional measures for handling increased load.

By following these phased steps, We can systematically implement an automated secret rotation system, ensuring security, efficiency, and scalability in managing MySQL root user passwords.

---

- ## Data Flow Overview:

The data flow within the automated secret rotation system involves the seamless exchange of information between various components. Below is a step-by-step description of the data flow:

- Initialization:

The process begins with the initialization phase where AWS Lambda functions, AWS Secrets Manager, MySQL database, and the SNS topic are set up.

- Secrets Manager Initialization:

AWS Secrets Manager is configured with a new secret to store MySQL root user credentials.

Rotation policies, including rotation frequency, are defined.

- IAM Role Creation:

An IAM role is created to provide AWS Lambda functions with the necessary permissions to interact with AWS Secrets Manager, CloudWatch, and other services.

- Rotation Function Trigger (Scheduled or Manual):

The rotation function is triggered either based on scheduled intervals set by CloudWatch Events or manually triggered through CloudWatch Event Rules.

- Lambda Rotation Function Execution:

The rotation function (rotation_function.py) is executed, retrieving the current MySQL root user credentials from Secrets Manager.

The Lambda function establishes a secure connection to the MySQL database using stored connection details.

The root user password is updated in the database.

- Password Update in Secrets Manager:

The new root user password is securely stored in AWS Secrets Manager, replacing the previous one.

Rotation metadata, including the last rotation timestamp, is updated.

- Notification Integration:

After the password update, the Lambda function triggers the send_notification function.

The send_notification function publishes a message to the configured SNS topic.

- Reset Cycle Function Trigger (Manual):

In case of a manual rotation, the reset cycle function (reset_cycle_function.py) is triggered.

- Lambda Reset Cycle Function Execution:

The reset cycle function updates rotation metadata, indicating that the next rotation should occur based on the configured frequency.

- Logging and Monitoring:

CloudWatch Logs capture detailed logs generated during the rotation process, providing auditability.

CloudWatch Alarms monitor key metrics, triggering alerts for any anomalies or issues.

- Continuous Monitoring:

The system continues to monitor and log rotation activities, ensuring that scheduled and manual rotations occur smoothly.

- Testing and Validation:

During testing, manual rotations are initiated to validate the responsiveness and effectiveness of the system.

Logs and alarms are reviewed to confirm proper execution.

- Documentation and Knowledge Transfer:

Implementation details are documented in the implementation guide.

Knowledge is transferred through documentation and training sessions.

- Deployment to Production:

The system, having undergone testing and validation, is deployed to production environments, and monitoring in production is initiated.

- Continuous Improvement:

Feedback from users and stakeholders is gathered for continuous improvement.

The system is iterated upon based on lessons learned and changing security requirements

This data flow illustrates the coordinated exchange of information, ensuring the secure and automated rotation of MySQL root user passwords within the AWS environment.

---

## • Python Script

Below is a brief overview of the Python script for the automated secret rotation system using AWS Lambda functions:

"Lambda_function.py"

Purpose:

This script is the main AWS Lambda function responsible for rotating MySQL root user passwords.

Workflow:

- Initialization:

Retrieves the MySQL root user credentials from AWS Secrets Manager.

Establishes a secure connection to the MySQL database using stored connection details.

- Password Rotation:

Generates a new password for the root user (you may use a more secure method).

Updates the MySQL root user password in the database.

- Secrets Manager Update:

Updates the AWS Secrets Manager with the new password.

Updates rotation metadata, including the last rotation timestamp.

- Notification:

After the password update, triggers the `send_notification` function to publish a notification message to the configured AWS SNS topic.

- Logging:

Captures detailed logs, including the timestamp of the rotation, in CloudWatch Logs for auditability.

- Error Handling:

Includes error handling mechanisms to address potential issues during the rotation process.

Note:

- The scripts utilize the AWS SDK (boto3) for interaction with AWS services.
- Credentials for accessing AWS services are managed securely through the Lambda function's IAM role.
- The code includes comments for clarity and understanding.
- The scripts are designed to work in conjunction with the AWS Secrets Manager and CloudWatch Events based on the system's configuration.

Make sure to replace placeholder values with your actual configuration details and customize the script based on your specific requirements.

code sample:

```
import json
import boto3
import pymysql
from datetime import datetime, timedelta

secrets_manager = boto3.client('secretsmanager')
sns_client = boto3.client('sns')

def rotate_mysql_password(event, context):
    # Replace with your MySQL secret name
    secret_name = "prod/prodData/MySQL"

    # Check if rotation is requested manually
    is_manual_rotation = event and event.get('is_manual_rotation', False)

    # Calculate the rotation date 40 days from now
    rotation_date = (datetime.now() + timedelta(days=40)).strftime('%Y-%m-%d')

    # Retrieve the current secret value once
    current_secret = secrets_manager.get_secret_value(SecretId=secret_name)
    secret_data = json.loads(current_secret['SecretString'])
    users_data = secret_data.get("users", [])

    for user_info in users_data:
        username = user_info.get("username")
        current_password = user_info.get("password")

        # Generate a new password (you may want to use a more secure method)
        new_password = "new_password"
        # Update the MySQL secret in Secrets Manager
        secrets_manager.update_secret(
            SecretId=secret_name,
            SecretString=json.dumps(secret_data)
        )

        # Update the MySQL user password
        update_mysql_user(username, new_password)

        # Send notification immediately after rotation
        send_notification(f"Password for MySQL user '{username}' rotated successfully!",
subject='Rotation Completed')

    # Send notification 5 days before rotation
    send_notification(f"MySQL secret rotation is scheduled for {rotation_date}.", subject='Rotation
Scheduled')

    # Reset rotation cycle if done manually
    if is_manual_rotation:
        reset_rotation_cycle(secret_name)

    return {
        'statusCode': 200,
        'body': json.dumps('MySQL secret rotation successful!')
    }

def update_mysql_user(username, new_password):
    # Replace with your MySQL connection details
    connection = pymysql.connect(host="your-mysql-host",
                               user="your-mysql-user",
                               password="your-mysql-password",
                               database="your-mysql-database")

    with connection.cursor() as cursor:
        # Replace with your MySQL user and host details
        cursor.execute(f"ALTER USER '{username}'@'%' IDENTIFIED BY '{new_password}';")
        connection.commit()

def reset_rotation_cycle(secret_name):
    # Here the rotation type is stored as 'timestamp' in the Secrets Manager secret
    rotation_type = 'timestamp'

    # Retrieve the current secret value
    current_secret = secrets_manager.get_secret_value(SecretId=secret_name)
    secret_data = json.loads(current_secret['SecretString'])

    if rotation_type == 'timestamp':
        # Update the rotation timestamp to the current time
        secret_data['rotation_timestamp'] = datetime.now().isoformat()
    elif rotation_type == 'counter':
        # Update the rotation counter (assuming you have a 'rotation_counter' field)
        secret_data['rotation_counter'] += 1
    else:
        # Handle other rotation types if needed
        pass

    # Update the MySQL secret in Secrets Manager
    secrets_manager.update_secret(
        SecretId=secret_name,
        SecretString=json.dumps(secret_data)
    )

def send_notification(message, subject='MySQL Secret Rotation Notification'):
    # Replace 'your-topic-arn' with the actual ARN of your SNS topic
    topic_arn = 'arn:aws:sns:your-region:your-account-id:MySQLRotationNotification'

    sns_client.publish(
        TopicArn=topic_arn,
        Message=message,
        Subject=subject
    )

# Uncomment the line below to test the rotation function locally
# rotate_mysql_password(None, None)
```

---

- ## Multiple Scenarios:

- ### Positive Scenarios:

1. Scenario 1: Scheduled Rotation
    a. Description: The Lambda function is triggered by a scheduled event.

    b. Expected Outcome: Successfully rotates MySQL root user passwords, updates Secrets Manager, and logs activities.
2. Scenario 2: Manual Rotation
    a. Description: The Lambda function is manually triggered through CloudWatch Event Rules.
    b. Expected Outcome: Successfully rotates MySQL root user passwords, updates Secrets Manager, and logs activities.
3. Scenario 3: Reset Cycle Function
    a. Description: The reset cycle function is triggered manually.
    b. Expected Outcome: Successfully resets the rotation cycle metadata in AWS Secrets Manager.

- Negative Scenarios

1. Scenario 1: Secrets Manager Unavailable
    a. Description: AWS Secrets Manager is unavailable during rotation.
    b. Expected Outcome: Proper error handling, logging, and no disruption in existing passwords.
2. Scenario 2: Database Connection Failure
    a. Description: Unable to establish a connection to the MySQL database.
    b. Expected Outcome: Proper error handling and logging without impacting existing passwords.
3. Scenario 3: Password Update Failure
    a. Description: Fails to update the MySQL root user password in the database.
    b. Expected Outcome: Proper error handling and logging without disrupting existing passwords.

- Error Conditions

1. Scenario 1: Critical Error during Rotation
    a. Description: Encounters a critical error during rotation.
    b. Expected Outcome: Proper error handling, logging, and no disruption in existing passwords.
2. Scenario 2: Log Handling Failure
    a. Description: Fails to log rotation activities due to an error.

    b. Expected Outcome: Proper error handling and resolution without impacting existing passwords.

- Operational Scenarios:

1. Scenario 1: Continuous Rotation Monitoring
    a. Description: Continuous monitoring of scheduled rotations.
    b. Expected Outcome: Successful rotation at defined intervals with appropriate logging.
2. Scenario 2: Audit Log Review
    a. Description: Reviewing CloudWatch Logs for audit purposes.
    b. Expected Outcome: Log entries capturing rotation activities and relevant details.

- Edge Cases

1. Scenario 1: Password Generation Edge Case
    a. Description: Testing edge cases for password generation.
    b. Expected Outcome: Proper handling of edge cases, generating secure passwords.
2. Scenario 2: Database Interaction Edge Case
    a. Description: Testing edge cases for database interaction.
    b. Expected Outcome: Proper handling of edge cases, ensuring database integrity.

- Test Cases:

1. Test Case 1: Scheduled Rotation
    a. Description: Verify scheduled rotations occur at specified intervals.
    b. Expected Outcome: Successful rotation, updated Secrets Manager, and proper logging.
2. Test Case 2: Manual Rotation
    a. Description: Manually trigger rotation through CloudWatch Event Rules.

    b. Expected Outcome: Successful rotation, updated Secrets Manager, and proper logging.
3. Test Case 3: Reset Cycle Function
    a. Description: Manually trigger the reset cycle function.
    b. Expected Outcome: Successful reset of rotation cycle metadata and proper logging.
4. Test Case 4: Negative Scenarios
    a. Description: Execute tests for negative scenarios (e.g., unavailable Secrets Manager, database connection failure).
    b. Expected Outcome: Proper error handling, logging, and no disruption in existing passwords.
5. Test Case 5: Edge Cases
    a. Description: Test edge cases for password generation and database interaction.
    b. Expected Outcome: Proper handling of edge cases and ensuring system robustness.

---

## ● Additional Considerations

1. Monitoring and Alerts:
    a. Implement comprehensive monitoring using AWS CloudWatch.
    b. Set up alarms to notify administrators in case of failures or abnormal behavior during rotations.
2. Secure IAM Roles:
    a. Ensure that IAM roles assigned to Lambda functions follow the principle of least privilege.
    b. Regularly review and update IAM policies based on evolving security requirements.
3. Rotation Frequency:
    a. Choose an appropriate rotation frequency based on security policies and compliance requirements.
    b. Consider the impact on applications and systems that use the rotated credentials.
4. Rotation Verification:
    a. Implement mechanisms to verify the success of each rotation, such as checking the last rotation timestamp in Secrets Manager.

5. Rollback Mechanism:
    a. Design a rollback mechanism in case of unexpected issues during a rotation.
    b. Ensure that the system can revert to the previous state without compromising security.
6. Testing Environment:
    a. Maintain a testing environment to simulate rotations, test new configurations, and ensure the system's resilience to changes.
7. Documentation and Runbooks:
    a. Maintain detailed documentation and runbooks for the system, including setup instructions, troubleshooting guides, and recovery procedures.
8. Rotation History and Auditing:
    a. Keep a historical record of rotation activities for auditing purposes.
    b. Regularly review rotation logs to detect any anomalies or unauthorized access.
9. Secure Transmission of Credentials:
    a. Ensure that credentials transmitted between services are encrypted.
    b. Use secure protocols for communication between the Lambda function and the MySQL database.
10. Cross-Account Considerations:
    a. If using multiple AWS accounts, carefully manage cross-account access and permissions for Secrets Manager and Lambda functions.
11. Concurrency and Throttling:
    a. Consider concurrency and throttling limits imposed by AWS services and adjust configurations accordingly.
12. Cross-Region Replication:
    a. If the MySQL database is deployed in multiple regions, implement cross-region replication for Secrets Manager to ensure availability and redundancy.
13. Backup and Restore Procedures:
    a. Establish backup and restore procedures for both the MySQL database and AWS Secrets Manager in case of data loss or corruption.
14. Regulatory Compliance:
    a. Ensure that the automated secret rotation system complies with relevant industry regulations and standards.
15. User Notifications:
    a. Implement a notification mechanism to inform users and administrators about upcoming rotations, especially for systems relying on the rotated credentials.
16. Dependency Versioning:

    a. Regularly update and version dependencies, such as the AWS SDK and third-party libraries, to benefit from security patches and improvements.

17. Integration with CI/CD Pipelines:
    a. Integrate the secret rotation system with CI/CD pipelines to ensure that changes are tested and deployed seamlessly.
18. Continuous Improvement:
    a. Establish a feedback loop to collect input and iteratively improve the system based on lessons learned and evolving requirements.

By considering these additional aspects, you can enhance the robustness, security, and maintainability of your automated secret rotation system. Adjust and customize these considerations based on your specific use case and organizational requirements.

---

## ● References

1. AWS services that use AWS Secrets Manager secrets
2. Set up alternating users rotation for AWS Secrets Manager
3. Troubleshooting IAM policies - AWS Identity and Access Management
4. Set up automatic rotation for Amazon RDS, Amazon Aurora, Amazon Redshift, or Amazon DocumentDB secrets using the console - AWS Secrets Manager
5. AWS Secrets Manager rotation function templates
6. MySQL Password Rotation with AWS Secrets Manager and Lambda
7. How aws secret manager notify the app about secret update? - Stack Overflow
8. Getting started with Amazon SNS - Amazon Simple Notification Service
9. Amazon Simple Notification Service Examples - AWS SDK for JavaScript.
10. Retrieve secrets from AWS Secrets Manager
11. https://docs.aws.amazon.com/secretsmanager/latest/userguide/create_database_secret.html?icmpid=docs_asm_help_panel

---

---