# AWS RDS MySQL Vertical Autoscaling

## Objective

Implement a system for vertical autoscaling of an Amazon RDS MySQL instance, allowing the instance to scale up when CPU utilization exceeds 80% and scale down when CPU utilization falls below 60%, while adhering to the principle of least privilege.

## Steps for Vertical Autoscaling

### 1. **Set Up CloudWatch Alarms**

#### a. Create a High CPU Utilization Alarm:

* Navigate to **CloudWatch** > **Alarms**.
* Create a new alarm with:
  + Metric: CPUUtilization
  + Condition: CPUUtilization > 80% for 5 minutes.
  + Action: Trigger the Lambda function for scaling up.

#### b. Create a Low CPU Utilization Alarm:

* Metric: CPUUtilization
* Condition: CPUUtilization < 60% for 5 minutes.
* Action: Trigger the Lambda function for scaling down.

### 2. **Create a Custom IAM Policy for Lambda**

#### a. Policy JSON:

{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Action": "rds:ModifyDBInstance",  
 "Resource": "arn:aws:rds:<region>:<account-id>:db:<db-instance-id>",  
 "Condition": {  
 "StringEquals": {  
 "rds:TargetDBInstanceClass": [  
 "db.t3.medium",  
 "db.m5.large"  
 ]  
 }  
 }  
 },  
 {  
 "Effect": "Allow",  
 "Action": "rds:DescribeDBInstances",  
 "Resource": "\*"  
 }  
 ]  
}

#### b. Replace Placeholders:

* <region>: AWS region (e.g., us-east-1).
* <account-id>: Your AWS account ID.
* <db-instance-id>: RDS instance identifier.

#### c. Attach the Policy:

* Navigate to **IAM Roles**.
* Attach the policy to the Lambda function’s execution role.

### 3. **Create the Lambda Function**

#### a. Python Script for Scaling:

import boto3  
  
def lambda\_handler(event, context):  
 rds\_client = boto3.client('rds')  
 db\_instance\_identifier = "your-db-instance-id"  
  
 # Define Alarm Names  
 high\_cpu\_alarm\_name = "HighCPUAlarm" # Alarm for scaling up  
 low\_cpu\_alarm\_name = "LowCPUAlarm" # Alarm for scaling down  
  
 # Define Instance Classes  
 instance\_classes = {  
 "scale\_up": "db.m5.large", # Larger instance class for scaling up  
 "scale\_down": "db.t3.medium" # Smaller instance class for scaling down  
 }  
  
 # Identify which alarm triggered the event  
 alarm\_name = event['detail']['alarmName']  
  
 if alarm\_name == high\_cpu\_alarm\_name:  
 target\_instance\_class = instance\_classes["scale\_up"]  
 elif alarm\_name == low\_cpu\_alarm\_name:  
 target\_instance\_class = instance\_classes["scale\_down"]  
 else:  
 print("Unknown alarm, no scaling action taken.")  
 return  
  
 # Modify the RDS instance  
 try:  
 response = rds\_client.modify\_db\_instance(  
 DBInstanceIdentifier=db\_instance\_identifier,  
 DBInstanceClass=target\_instance\_class,  
 ApplyImmediately=True  
 )  
 print(f"Scaling operation to {target\_instance\_class} initiated:", response)  
 except Exception as e:  
 print("Error scaling instance:", e)

#### b. Deployment:

* Deploy this Lambda function in the AWS Lambda console.
* Configure it to be triggered by the CloudWatch Alarms.

### 4. **Testing and Validation**

1. Simulate high CPU utilization to trigger the **HighCPUAlarm**.
2. Verify that the RDS instance scales up to the larger class.
3. Simulate low CPU utilization to trigger the **LowCPUAlarm**.
4. Confirm the RDS instance scales down to the smaller class.

### **Key Considerations**

1. **Downtime**:
   * Modifying the instance class will cause a brief downtime. Plan changes during low-traffic periods.
2. **Scaling Limits**:
   * Ensure the target instance classes support your workload.
3. **Cost Management**:
   * Monitor costs to avoid unexpected charges for larger instance classes.
4. **Tagging Resources**:
   * Use tags to manage and restrict policies to specific resources.

## Summary

This setup enables automated vertical scaling of RDS MySQL instances based on CPU utilization, ensuring optimal performance and cost efficiency while adhering to the principle of least privilege for resource access.