

Project Report: Implementing Auto Scaling using ASG in AWS

Project Title

Implementing Elastic and Scalable Architecture using AWS Auto Scaling Groups (ASG)

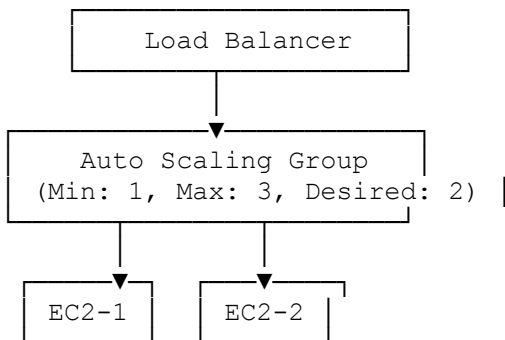
Objective

To configure an Auto Scaling Group (ASG) using a custom AMI and launch template that dynamically adjusts the number of EC2 instances based on traffic load or health checks, ensuring high availability and cost-efficiency.

Key Concepts & Services Used

- Amazon Machine Image (AMI)
 - Launch Template
 - Auto Scaling Group (ASG)
 - Target Tracking Policies
 - CloudWatch Alarms
 - EC2 Instances
 - Load Balancer (Optional)
-

Architecture Overview



Steps Implemented

1. Create Custom AMI

- Launch a base EC2 instance and install required packages (e.g., NGINX).
- Create an AMI from this configured instance to use for all future scaling.

2. Create Launch Template

- Launch Template includes:
 - AMI ID (from the custom image)
 - Instance type (e.g., t2.micro)
 - Security group
 - Key pair
 - User data (optional startup script)

3. Configure Auto Scaling Group (ASG)

- Set parameters:
 - Launch Template
 - VPC and Subnet(s)
 - Desired capacity: 2
 - Min size: 1, Max size: 3
 - Health check: EC2 or ELB-based
 - Replace unhealthy instances automatically

4. Attach Scaling Policy

- **Target Tracking Policy** (e.g., maintain average CPU utilization at 50%)
- AWS CloudWatch triggers scale in/out actions based on metrics

5. (Optional) Attach to Load Balancer

- Create an Application Load Balancer (ALB)
- Register target group with ASG for seamless traffic distribution

Testing Auto Scaling

- Simulated load using `stress` command or by launching test traffic
- Observed ASG scale-out action when CPU threshold crossed
- ASG scaled in when traffic reduced