Step-by-Step Implementation Plan

1. Launch a Web Application

- Create a simple HTML/PHP site (can even be a static HTML page)
- Package it in an **AMI** for EC2 instances

2. EC2 Auto Scaling Group

- Launch EC2 instances in **2 Availability Zones** for high availability
- Use a **Launch Template/Configuration** with the AMI

3. Application Load Balancer (ALB)

- Create ALB → forward traffic to EC2 instances in the ASG
- Configure health checks → ensure failed instances are replaced

4. Optional RDS Database

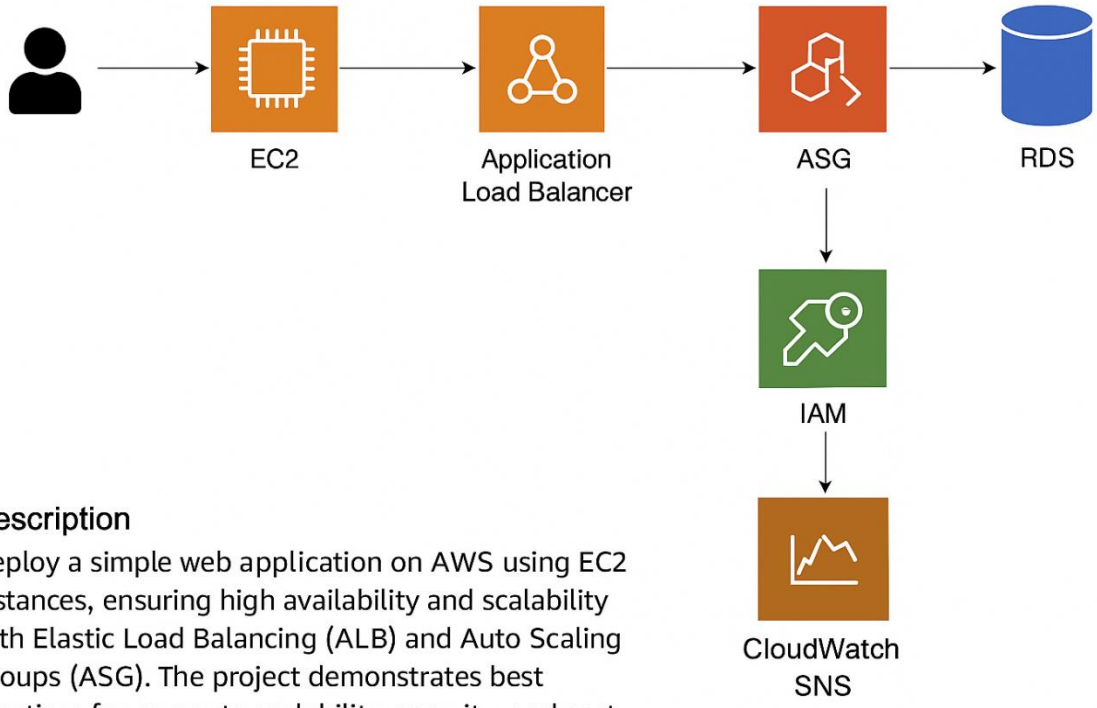
- Add **Amazon RDS MySQL/PostgreSQL** (Multi-AZ) if your app needs a database
- Configure security groups to allow EC2 access only

5. IAM Roles & Security Groups

- EC2 role → minimal permissions (read from S3 if needed)
- Security groups → allow HTTP/HTTPS from ALB only

6. Monitoring & Alerts

- CloudWatch → monitor CPU/Memory, scale-out/scale-in policies
- SNS → alert notifications



Description

Deploy a simple web application on AWS using EC2 instances, ensuring high availability and scalability with Elastic Load Balancing (ALB) and Auto Scaling Groups (ASG). The project demonstrates best practices for compute scalability, security, and cost optimization.

Project 1: Scalable Web App with ALB & Auto Scaling

Overview

Deploy a highly available and scalable web application on AWS using EC2, ALB, and Auto Scaling.

Key Components:

- **EC2:** Web servers
- **ALB:** Distribute traffic
- **Auto Scaling Group:** Automatic scaling based on demand
- **IAM Roles:** Secure access to AWS services from EC2
- **CloudWatch:** Monitor performance metrics
- **SNS:** Send alerts based on CloudWatch alarms

Step 1: Launch Template

- **Name:** WebAppTemplate
- **AMI:** Amazon Linux 2 / Ubuntu
- **Instance Type:** t2.micro / t3.medium
- **Key Pair:** Existing or new
- **User Data:** Install Apache & sample webpage

```
#!/bin/bash
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
echo "<h1>Welcome to Scalable Web App</h1>" | sudo tee
/var/www/html/index.html
```

- **Security Group:** Allow SSH(22), HTTP(80), HTTPS(443)
- **IAM Role:** Attach role with CloudWatch & S3 access

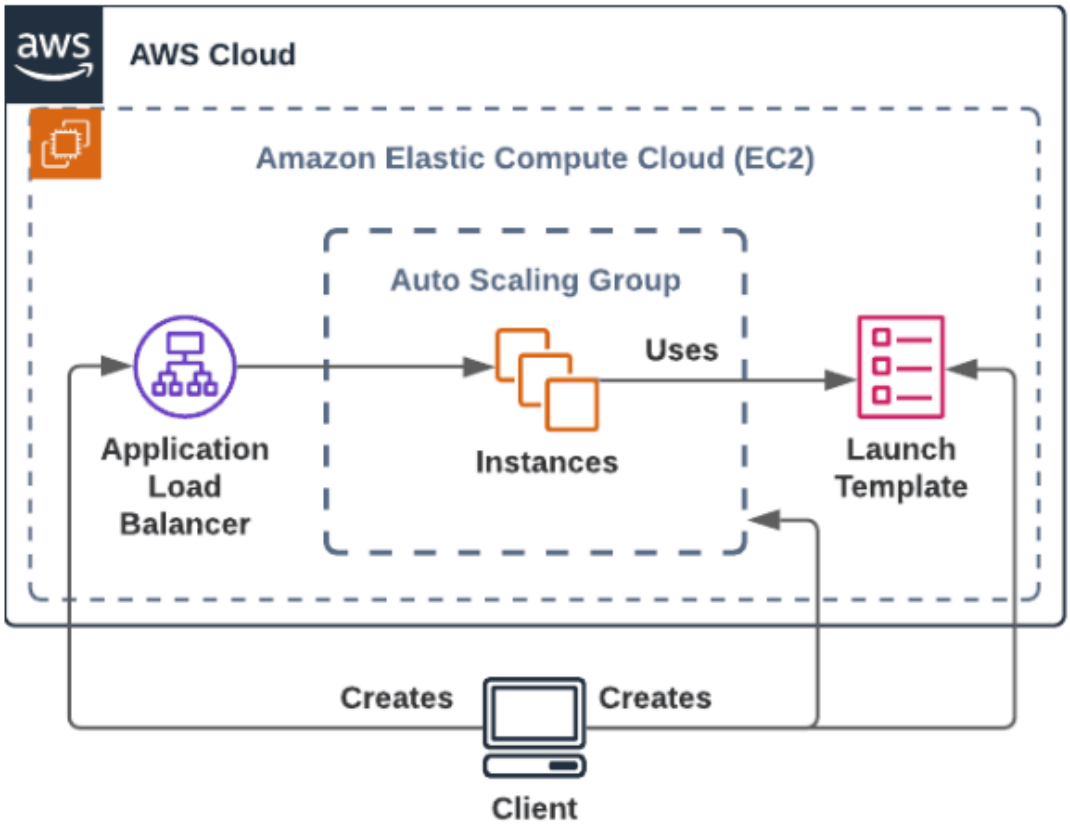
Step 2: Auto Scaling Group

- **Launch Template:** WebAppTemplate
- **Group Size:** Desired=2, Min=1, Max=4
- **Network:** Select VPC & 2+ AZs
- **Load Balancer:** Attach ALB
- **Target Group:** Protocol=HTTP, Health Check=/
- **Scaling Policies:** CPU > 60% → scale out, CPU < 40% → scale in

---

Step 3: Application Load Balancer (ALB)

- **Name:** WebAppALB
- **Scheme:** Internet-facing
- **VPC:** Same as ASG
- **AZs:** 2+ availability zones
- **Listeners:** HTTP 80
- **Target Group:** ASG instances
- **Security Group:** Allow HTTP(80)



Step 4: CloudWatch & SNS Integration

- **CloudWatch Metrics:** CPU, Memory, Disk, Network, HTTP requests
- **CloudWatch Logs:** Apache/Nginx logs or custom app logs
- **SNS Alerts:** Create SNS topic and subscribe email for alarms
- **CloudWatch Alarm Example:**
    - Metric: CPUUtilization

- Threshold: > 70% for 5 min
- Action: Notify SNS topic

Step 5: Test & Verify

1. Access ALB DNS: `http://your-alb-dns-name` → Should see welcome page
2. Load Test: `ab -n 1000 -c 50 http://your-alb-dns-name/`
3. Verify Auto Scaling: Stop instance → Auto Scaling replaces it
4. Check CloudWatch metrics and SNS email alerts

✅ Key Takeaways

- **High Availability:** ALB + multi-AZ
- **Scalability:** Auto Scaling handles traffic spikes
- **Redundancy:** Multiple instances across AZs
- **Security:** IAM roles for safe access
- **Monitoring & Alerts:** CloudWatch + SNS for proactive management