AWS Practical Guide: Auto Scaling Group with Load Balancer

This document provides detailed, step-by-step instructions to configure an **Application Load Balancer (ALB)** with an **Auto Scaling Group (ASG)** in AWS. The ASG will use a **Launch Template** (based on your custom AMI), and all instances will automatically register/deregister in the ALB Target Group.

---

1. Prerequisites

- A custom **AMI** created from your configured EC2 instance.
- A **VPC** with at least 2 public subnets in different Availability Zones.
- Proper **IAM role/instance profile** if your instances need S3, CloudWatch, or other AWS services.
- Security groups: one for ALB, one for EC2 instances.

---

2. Create a Launch Template

1. Go to **EC2 → Launch Templates → Create launch template**.
2. Enter a name (e.g., `my-webapp-template`).
3. Select your custom **AMI ID**.
4. Choose instance type (e.g., `t3.micro`).
5. Assign an **IAM role** (if required).
6. Configure **security group** (EC2 instances SG — allow HTTP from ALB SG only).
7. Add **User Data** script to install/start your web server:

```
#!/bin/bash
yum update -y
yum install -y httpd php php-fpm
systemctl enable httpd
systemctl start httpd
echo "<h1>Web server running on $(hostname)</h1>" > /var/www/html/
index.html
```

8. Save the template.

---

3. Create a Target Group

1. Go to **EC2 → Target Groups → Create target group**.
2. Target type: **Instances**.
3. Protocol: **HTTP**, Port: **80**.

4. VPC: select your VPC.
5. Health check: Protocol **HTTP**, Path `/` (or `/health` if your app has it).
6. Finish creation — note the Target Group ARN.

---

4. Create an Application Load Balancer

1. Go to **EC2 → Load Balancers → Create Load Balancer**.
2. Choose **Application Load Balancer**.
3. Name: `my-alb`.
4. Scheme: **Internet-facing**.
5. IP type: **IPv4**.
6. Select at least **2 public subnets**.
7. Security group: ALB SG (allow inbound 80/443 from internet).
8. Listener: Port 80 → forward to your Target Group.
9. Create ALB.

Once created, ALB will have a **DNS name** (e.g., `my-alb-123456.us-east-1.elb.amazonaws.com`). This is your app's entry point.

---

5. Create the Auto Scaling Group

1. Go to **EC2 → Auto Scaling Groups → Create Auto Scaling group**.
2. Name: `my-asg`.
3. Select your **Launch Template**.
4. Select VPC and **subnets** (choose at least 2 AZs).
5. Attach to existing **Target Group** (created in Step 3).
6. Group size: e.g., Min = 2, Desired = 2, Max = 6.
7. Health checks: choose **ELB** (so ALB health checks are used).
8. Health check grace period: e.g., **300 seconds**.
9. Scaling policies: choose **Target Tracking**.
10. Example: scale to keep **Average CPU Utilization = 60%**, or
11. Use **ALB Request Count per Target**.
12. Review and create.

---

6. Test the Setup

1. Open the ALB **DNS name** in your browser.
2. You should see your app (e.g., the hostname message).
3. Run a load generator (e.g., `stress` tool or ApacheBench):

```
ab -n 10000 -c 100 http://my-alb-123456.us-east-1.elb.amazonaws.com/
```

4. Watch **EC2 → Auto Scaling Groups → Activity history** to see scale-out events.
5. Check **Target Groups → Targets** to confirm new instances auto-register.

---

7. Scale In (Removing Instances)

- When load decreases, the ASG reduces desired capacity.
- Instances are deregistered from the Target Group, wait for **deregistration delay** (default ~300s), then terminated.
- ALB only routes traffic to healthy instances.

---

8. Best Practices

- Use **User Data** or **baked AMIs** for consistent bootstrapping.
- Keep health check path simple and fast (e.g., `/health`).
- Store sessions in an external store (Redis/DB) if you need sticky sessions.
- Use **CloudWatch Alarms** for monitoring scaling behavior.
- Enable **connection draining / deregistration delay** to avoid cutting in-flight requests.

---

9. Useful CLI Commands (Quick Reference)

```
# Create Target Group
aws elbv2 create-target-group
  --name my-tg --protocol HTTP --port 80 --vpc-id vpc-xxxx
  --health-check-protocol HTTP --health-check-path /health

# Create Load Balancer
aws elbv2 create-load-balancer
  --name my-alb --subnets subnet-a subnet-b --security-groups sg-alb

# Create Listener
aws elbv2 create-listener
  --load-balancer-arn <alb-arn> --protocol HTTP --port 80
  --default-actions Type=forward,TargetGroupArn=<tg-arn>

# Create Auto Scaling Group
aws autoscaling create-auto-scaling-group
  --auto-scaling-group-name my-asg
  --launch-template LaunchTemplateId=lt-xxxx,Version=1
  --min-size 2 --max-size 6 --desired-capacity 2
  --vpc-zone-identifier "subnet-a,subnet-b"
  --target-group-arns <tg-arn>
```

```
# Add Target Tracking Policy (CPU 60%)
aws autoscaling put-scaling-policy
  --auto-scaling-group-name my-asg
  --policy-name cpu-policy
  --policy-type TargetTrackingScaling
  --target-tracking-configuration '{"PredefinedMetricSpecification":
{"PredefinedMetricType":"ASGAverageCPUUtilization"},"TargetValue":60.0}'
```

---

10. Summary

- **Launch Template**: defines how to create instances.
- **ASG**: creates/destroys instances based on scaling policies.
- **Target Group**: holds registered instances.
- **ALB**: routes traffic to healthy targets.

When load increases, ASG creates new EC2s from the Launch Template → registers them to the Target Group → ALB automatically starts routing traffic to them after health checks.

When load decreases, ASG terminates instances → deregisters from Target Group → ALB stops sending traffic.

✅Following these steps, you'll have a fully functional **load-balanced, auto-scaled application** in AWS.