# EC2 + ALB + Launch Template + Target Group + ASG + S3 — GUI Lab

**Purpose:** Hands-on AWS Console (GUI) lab that launches EC2 instances via a Launch Template and Auto Scaling Group (ASG), places them behind an Application Load Balancer (ALB), and uploads a startup file to S3 using an IAM role.

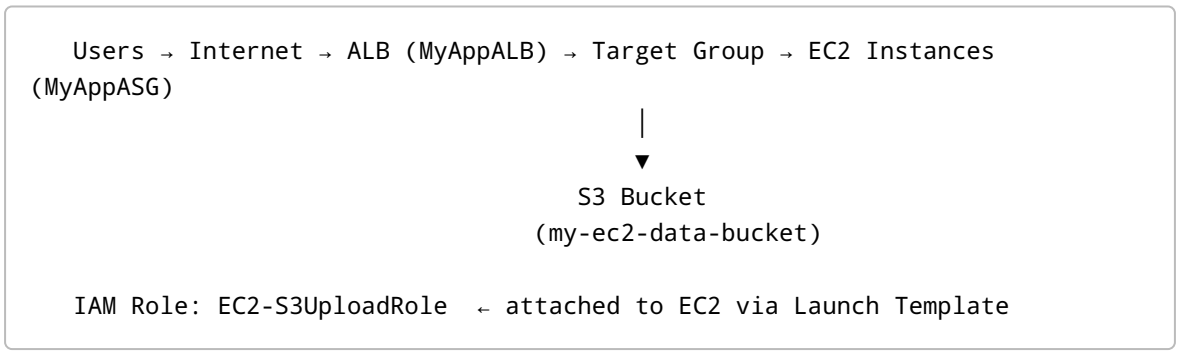
---

## Prerequisites

- AWS account with permissions for EC2, IAM, S3, ELB, and Auto Scaling.
- Default VPC (or your custom VPC) with **at least 2 public subnets** in different AZs.
- SSH key pair (optional — only for testing SSH).

---

## Architecture (text diagram)




---

## Resources to create (GUI)

- S3 Bucket: `my-ec2-data-bucket`
- IAM Role: `EC2-S3UploadRole` (EC2 role with S3 access)
- Launch Template: `MyAppTemplate` (with user-data script)
- Target Group: (Instances, HTTP:80)
- Auto Scaling Group: `MyAppASG` (Desired:2, Min:1, Max:3)
- Application Load Balancer: `MyAppALB` (Internet-facing, HTTP:80)

---

## Step-by-step Execution (AWS Console GUI)

### STEP 1 — Create S3 bucket & IAM role

1. **S3** → Create bucket → name: `my-ec2-data-bucket` → Region: your choice → Create.

2. **IAM** → Roles → Create role

3. Trusted entity: **AWS service** → **EC2**.

4. Attach policy: **AmazonS3FullAccess** (demo only). In production, create a least-privilege policy allowing `s3:PutObject` on `arn:aws:s3:::my-ec2-data-bucket/*`.
5. Name the role: `EC2-S3UploadRole` → Create role.

   **Note (secure example policy)** — Use this custom policy instead of `AmazonS3FullAccess` for production:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": ["arn:aws:s3:::my-ec2-data-bucket/*"]
    }
  ]
}
```

---

## STEP 2 — Create Launch Template (MyAppTemplate)

1. **EC2** → Launch Templates → Create launch template.
2. Name: `MyAppTemplate`.
3. AMI: **Amazon Linux 2** (or preferred AMI).
4. Instance type: `t3.micro` (demo).
5. Key pair: select existing (optional).
6. IAM instance profile: **EC2-S3UploadRole** (select the role created earlier).
7. Security group: create/select a group allowing **HTTP (80)** and **SSH (22)** (SSH only for testing).
8. Expand **Advanced details** → **User data**: paste the script below (this will run on instance boot and upload a file to S3):

```
#!/bin/bash
yum update -y
yum install -y awscli
echo "Hello from EC2 at $(date)" > /tmp/instance-data.txt
aws s3 cp /tmp/instance-data.txt s3://my-ec2-data-bucket/
```

1. Create launch template.

---

## STEP 3 — Create Target Group and Auto Scaling Group (ASG)

### Create Target Group

1. **EC2** → Target Groups → Create target group.
2. Target type: **Instances**.

3. Name: `web-target-group`.
4. Protocol: **HTTP**, Port: **80**.
5. VPC: select your VPC.
6. Health check path: `/`.
7. Create target group. (Do not register targets manually — ASG will register them.)

**Create Auto Scaling Group (ASG)**

1. **EC2** → Auto Scaling Groups → Create Auto Scaling group.
2. Choose launch template: **MyAppTemplate**.
3. Name: `MyAppASG`.
4. VPC and subnets: select **at least 2 public subnets** across different AZs.
5. Set group size: Desired = **2**, Min = **1**, Max = **3**.
6. Load balancing: Attach to an existing load balancer → choose **Application Load Balancer** and select the target group `web-target-group` (or create a new target group here if you prefer).
7. Skip scaling policies for now (or create target tracking later).
8. Review and create ASG.

   After creation, ASG will launch EC2 instances automatically using the launch template.

---

## STEP 4 — Create Application Load Balancer (ALB)

1. **EC2** → Load Balancers → Create Load Balancer → Choose **Application Load Balancer**.
2. Name: `MyAppALB`.
3. Scheme: **Internet-facing**.
4. IP address type: **IPv4**.
5. Network mappings: choose **at least 2 public subnets** (one in each AZ).
6. Security groups: create/select `alb-sg` that allows **HTTP (80)** from `0.0.0.0/0`.
7. Listeners and routing:
8. Listener: **HTTP (80)** → Default action: Forward to **web-target-group**.
9. Create ALB.

   Wait until ALB is **Active**.

---

## STEP 5 — Verification & Result

1. Wait until ASG finishes launching instances (Desired = 2).
2. **EC2** → Instances: verify that instances are in **running** state and have the IAM role attached.
3. **Target Groups** → `web-target-group` → Targets: instances should show as **healthy**.
4. **Load Balancers** → `MyAppALB` → Description → **DNS name**: open `http://<ALB-DNS>` in a browser to access the app.
5. **S3** → `my-ec2-data-bucket`: check that the file `instance-data.txt` (or multiple files named by instance) exists. Each instance writes `/tmp/instance-data.txt` on its first boot and uploads it to the bucket.

---

## Troubleshooting

- **No objects in S3:**
- Verify the instance has the IAM role attached (EC2 → Instance → Description → IAM role).
- Check user-data execution logs on the instance: `/var/log/cloud-init-output.log` or `/var/log/messages` (if you can SSH).
- Confirm region of S3 bucket matches instance region.
- **Targets unhealthy:**
- Security groups: ensure `alb-sg` can reach instance SG on port 80 and instance SG allows traffic from ALB.
- Confirm web server started successfully in user-data.
- **ASG not launching instances:**
- Check launch template for valid AMI and instance type, and that subnets have capacity/quota.

---

## Cleanup (to avoid charges)

1. **Auto Scaling Groups** → Set desired capacity to 0 → Delete the ASG.
2. **EC2** → Terminate any running instances left behind.
3. **Load Balancers** → Delete `MyAppALB`.
4. **Target Groups** → Delete `web-target-group`.
5. **Launch Templates** → Delete `MyAppTemplate`.
6. **IAM** → Delete the role `EC2-S3UploadRole` (detach policies first).
7. **S3** → Empty and delete bucket `my-ec2-data-bucket`.
8. Delete any security groups created for this lab.

---

## Optional Enhancements

- Replace `AmazonS3FullAccess` with the least-privilege S3 policy shown above.
- Use private subnets + NAT gateway for production-style deployments.

---

*End of lab — enjoy!*