**Project 1: Scalable Web Application with ALB and Auto Scaling**

**Architecture:** EC2-based

**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.

**Key AWS Services Used:**

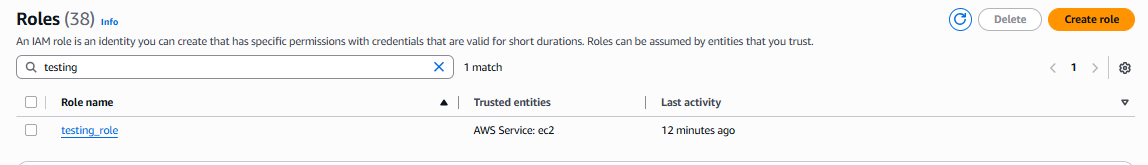
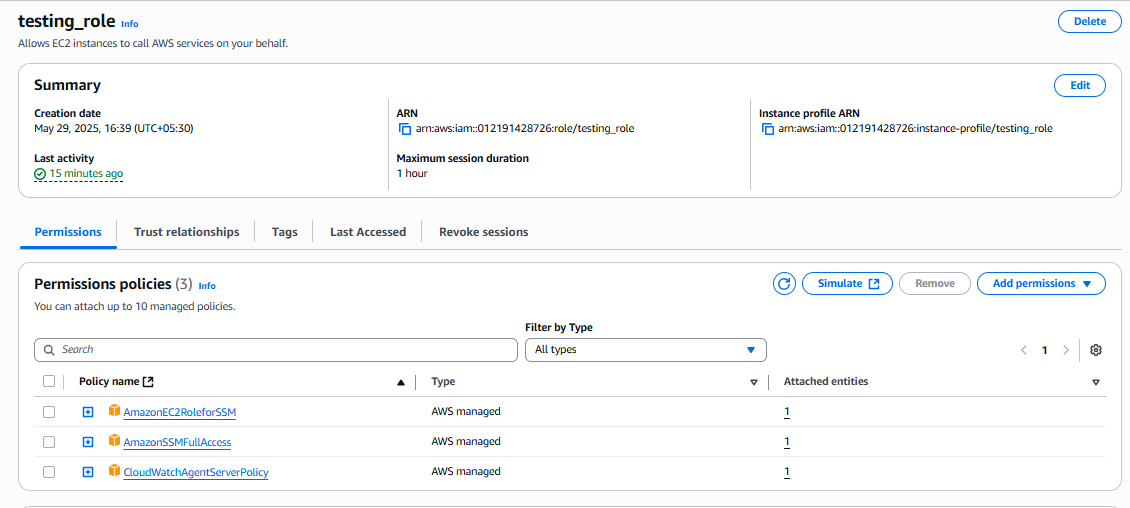
* **EC2:** Launch instances for the web app.
* **Application Load Balancer (ALB):** Distributes traffic across multiple instances.
* **Auto Scaling Group (ASG):** Ensures instances scale based on demand.
* **Amazon RDS (Optional):** Backend database (MySQL/PostgreSQL) with Multi-AZ.
* **IAM:** Role-based access to instances.
* **CloudWatch & SNS:** Monitor performance and send alerts.

**Learning Outcomes:**

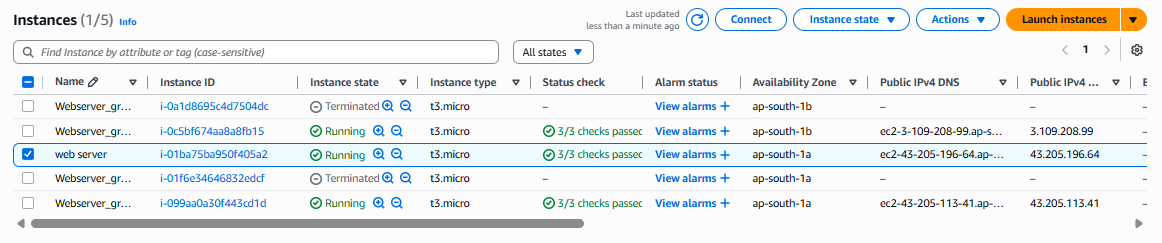
* Setting up **secure and scalable** EC2-based web applications.
* Implementing **high availability** using ALB and ASG.
* Optimizing **costs and performance** using Auto Scaling policies

Step by step process

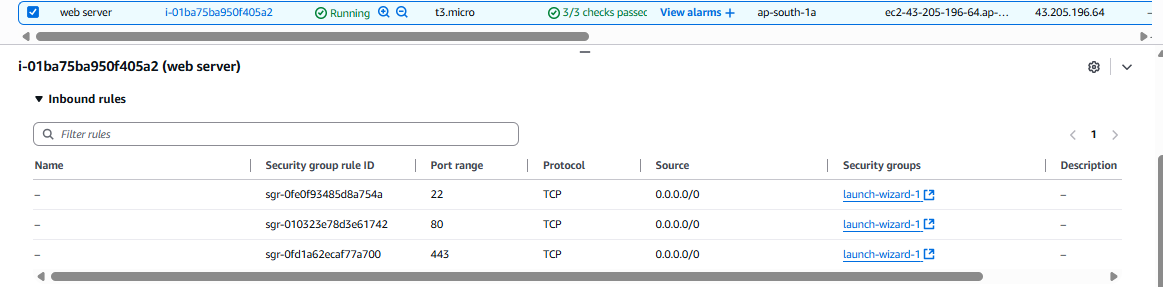
### ****Step 1: IAM Roles****

* Create an **IAM role** for EC2 to allow access to CloudWatch and S3 (if needed).
  + Attach AmazonEC2RoleforSSM, CloudWatchAgentServerPolicy.

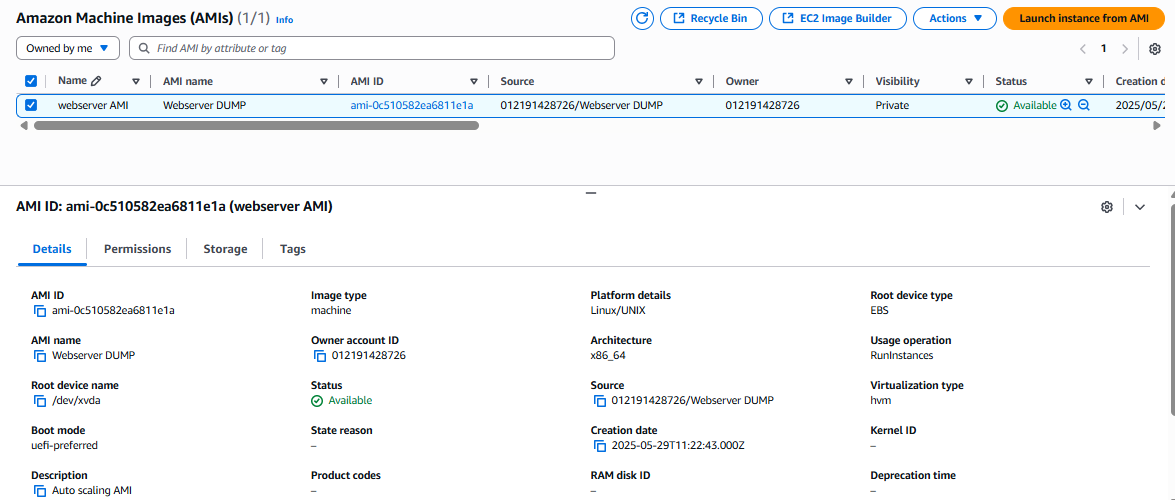
### 💻 ****Step 2: EC2 Instance Setup****

* Launch a **t2.micro or t3.micro** instance with:
  + Amazon Linux 2 or Ubuntu
  + A web server (e.g., install Apache: sudo yum install -y httpd and start the service)
* Install and configure your **web app** (can be a static HTML or dynamic app).

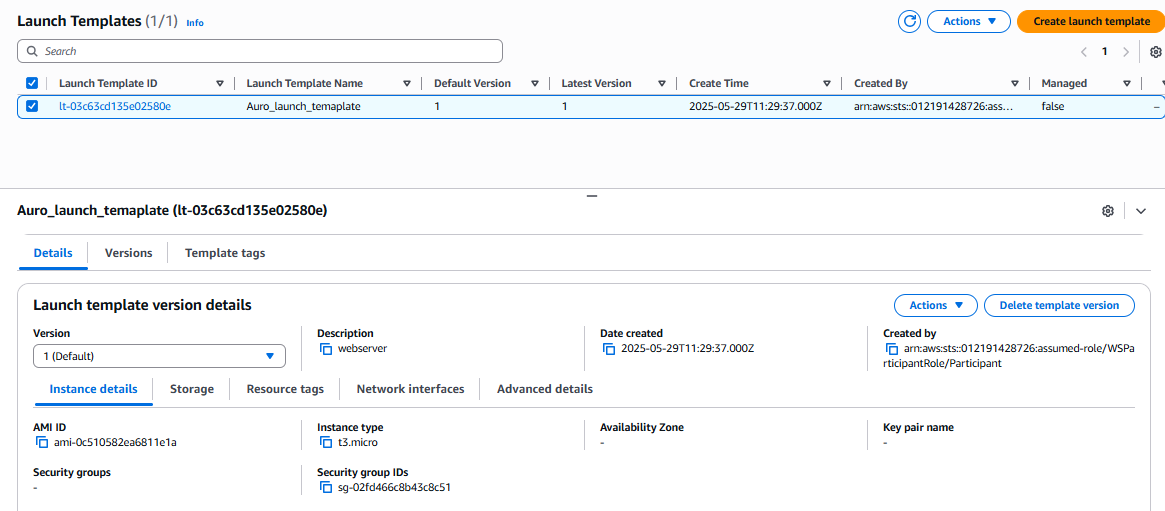
### 🛑 ****Step 3: Security Groups****

* ALB: Allow inbound on port **80/443**
* EC2: Allow inbound from ALB **only**, outbound allowed for updates

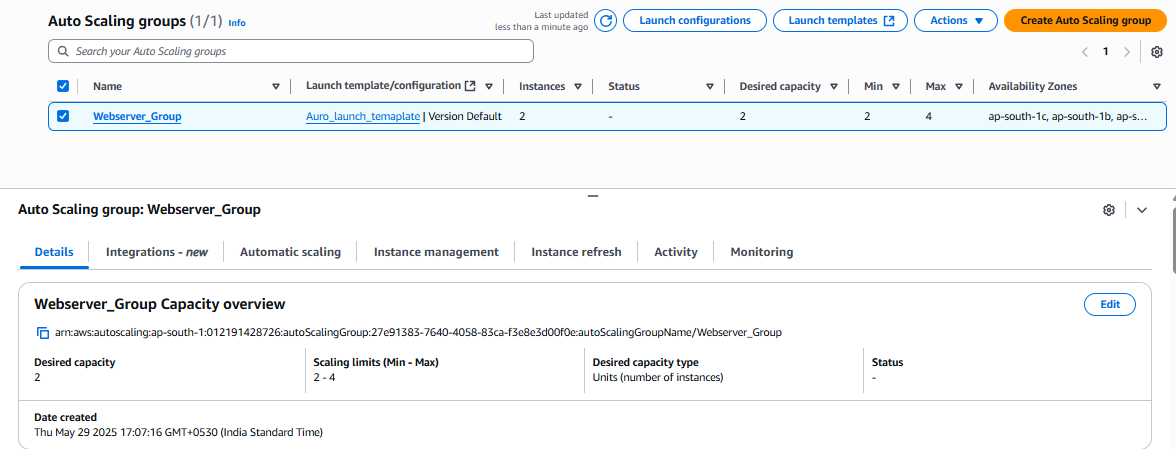
### 🖼️ ****Step 4: Create AMI****

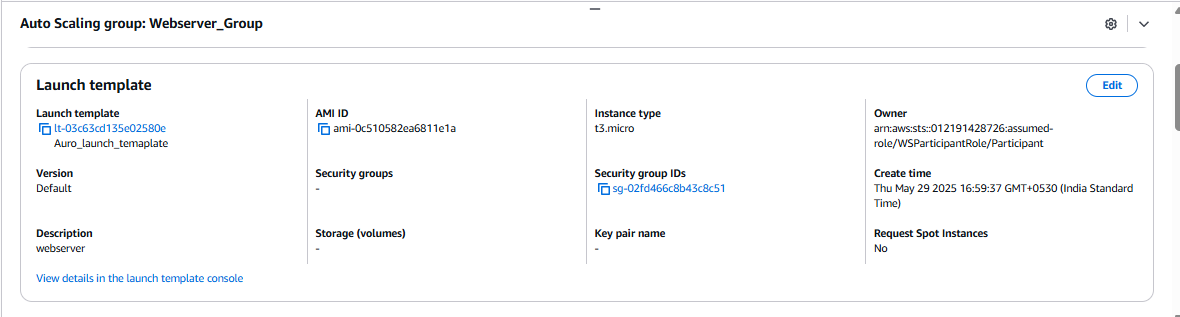
* After installing and configuring your app, create an **Amazon Machine Image (AMI)** to reuse in Auto Scaling.

### 📐 ****Step 5: Launch Template or Configuration****

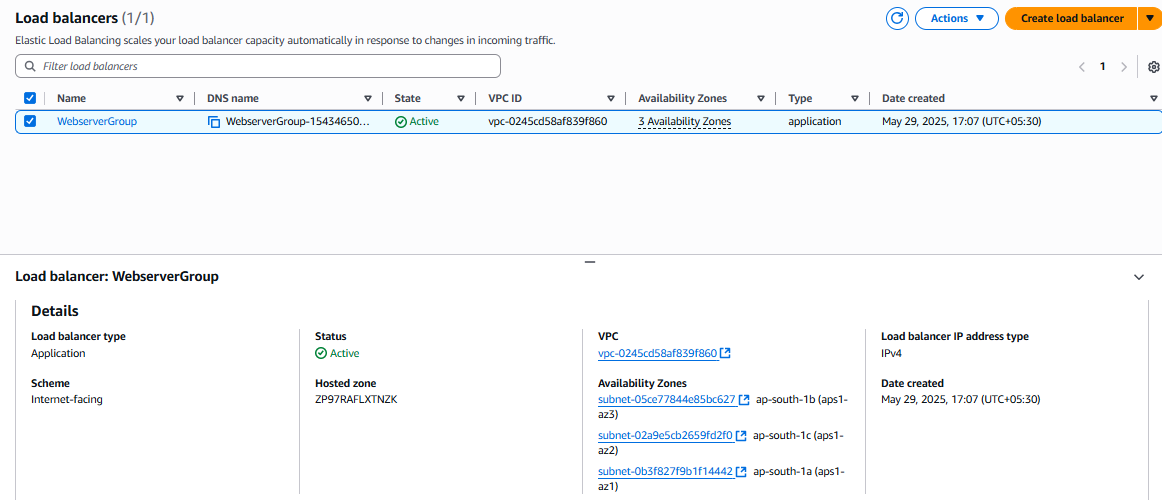
* Create a **Launch Template** using the AMI
* Set instance type, key pair, IAM role, and security group

### 🌀 ****Step 6: Create Auto Scaling Group****

* Define:
  + Min/Max/Desired capacity (e.g., 2/4/2)
  + Target group (connected to ALB)
  + Scaling policies (e.g., CPU > 70%)
  + Network (subnets across multiple AZs)



### ⚖️ ****Step 7: Create Application Load Balancer****

* ALB type: **Application**
* Listener: Port 80 → Target group (EC2 instances in ASG)
* Health checks: / or /health

### 📈 ****Step 9: Monitoring with CloudWatch****

* Monitor:
  + CPU utilization
  + ALB request count
  + Healthy/Unhealthy instance count
* Set up **CloudWatch Alarms + SNS topic** for email alerts

### 💰 ****Step 10: Cost Optimization****

* Use **t3/t4g instances** (burstable performance)
* Use **Auto Scaling** to terminate idle instances
* Use **Reserved Instances** for baseline capacity

## 🧪 **11. Testing**

* Access ALB DNS name from browser
* Simulate load with tools like **ApacheBench** or **Locust**
* Observe scaling actions in EC2 dashboard

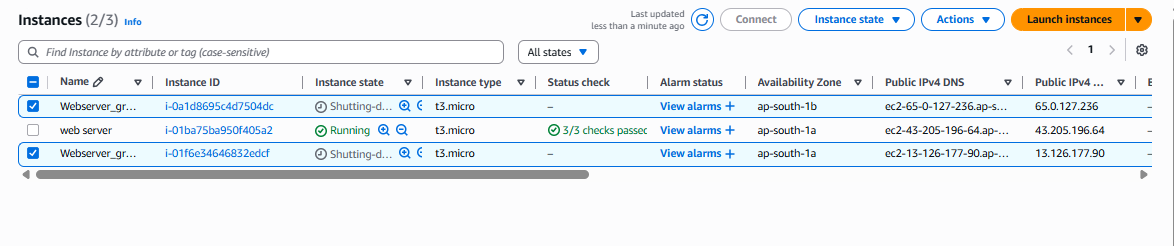
## 📦 **12. Optional: Infrastructure as Code (IaC)**

* Use **Terraform** or **CloudFormation** for repeatable setup.
* Modules:
  + ec2\_instance, alb, asg, rds, cloudwatch, sns

## 🎓 **Learning Outcomes Recap**

* ✅ Launched and secured EC2 instances
* ✅ Used ALB to distribute load
* ✅ Used ASG to scale based on demand
* ✅ Monitored metrics and set alerts
* ✅ (Optional) Used RDS for backend DB
* ✅ Practiced cost-saving strategies

Working autoscaling link below snapshot:

Before instance terminate snapshot

After auto scaling group created new instance as per configuration.

