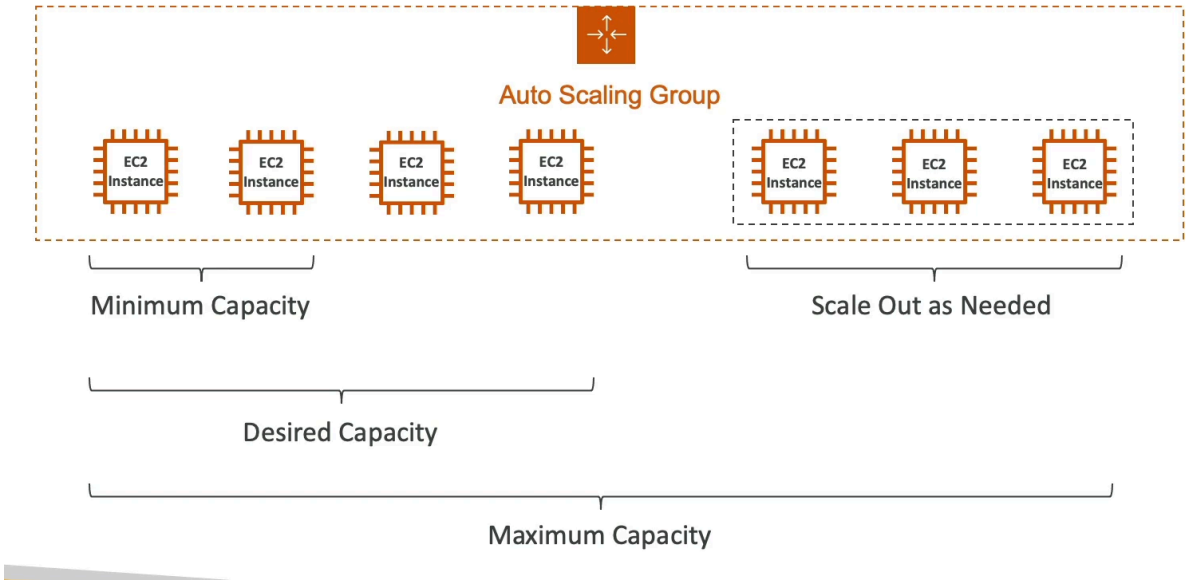
🚀 AWS Auto Scaling Groups (ASG)

An Auto Scaling Group (ASG) is an AWS service that automatically adjusts the number of EC2 instances in response to demand. It ensures that the right number of instances are running to handle traffic while optimizing costs. ASG can scale out (add instances) when demand increases and scale in (remove instances) when demand decreases.



Goal of Auto Scaling Groups (ASG) 🎯

An **Auto Scaling Group (ASG)** helps maintain the right number of EC2 instances to handle varying levels of traffic. It ensures **high availability**, **cost optimization**, and **fault tolerance**. ASG can automatically **scale in** (reduce instances) or **scale out** (add instances) based on demand.

How ASG Works with AWS Load Balancer ⚖️

ASGs work seamlessly with **Elastic Load Balancers (ELB)** to distribute traffic evenly across EC2 instances. The Load Balancer:

- **Routes traffic** to healthy instances ✅
- **Removes unhealthy instances** ❌
- **Works with auto-scaling policies** to ensure **dynamic traffic handling** 📈

📘 What is an Auto Scaling Group?

An **Auto Scaling Group (ASG)** helps you automatically launch or terminate EC2 instances based on defined conditions such as:

- CPU usage 📈
- Number of requests 📊
- Time schedule 🕒

✅ **Benefits of ASG**:

- High availability
- Fault tolerance
- Cost optimization

🧩 Components of ASG

1. **Launch Template** 🧾
   Defines instance configuration like AMI, instance type, key pair, security groups, and user data.

2. **Scaling Policies** 📏
   Rules that define when to scale in or scale out (e.g., when CPU > 70%).

3. **Group Size** 👥
   Minimum, maximum, and desired number of EC2 instances.

4. **Health Checks** 🩺
   AWS checks if instances are healthy and replaces unhealthy ones automatically.

5. **Load Balancer (Optional)** ⚖️
   Can be attached to distribute traffic across instances.

🛠️ Hands-On: AWS Auto Scaling Group with Dynamic Scaling and CloudWatch Alarms

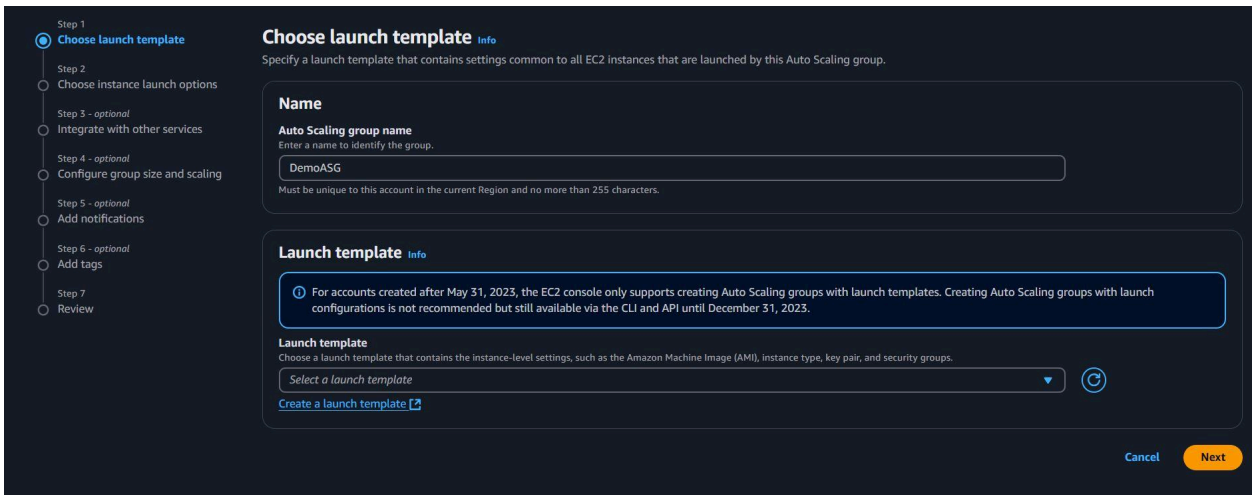
Step-by-step:

1. Go to **EC2** Dashboard ➡️ **Auto Scaling Groups**
2. Click on **Create Auto Scaling Group**

📝 Step 1: Choose Launch Template

Name and Launch Template

- Enter **ASG Name** (e.g., `DemoASG`)
- Click **Create a Launch Template**



Inside Launch Template:

- Name: `DemoLT`
- Description: `Launch template for ASG demo`
- ✅ **Application and OS Images (AMI)**: Choose Amazon Linux 2

- 🖥️ **Instance type**: e.g., `t2.micro`
- 🔑 **Key pair**: Choose existing or create new
- 🔒 **Security group**: Allow HTTP (port 80) and SSH (port 22)

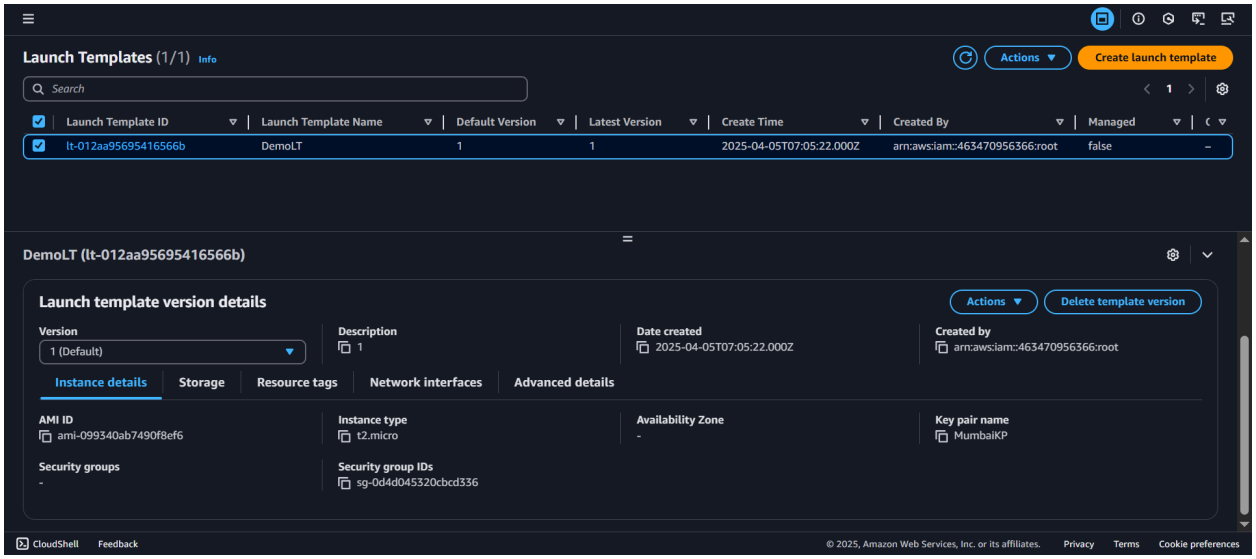
⚙️ Advanced Details – User Data

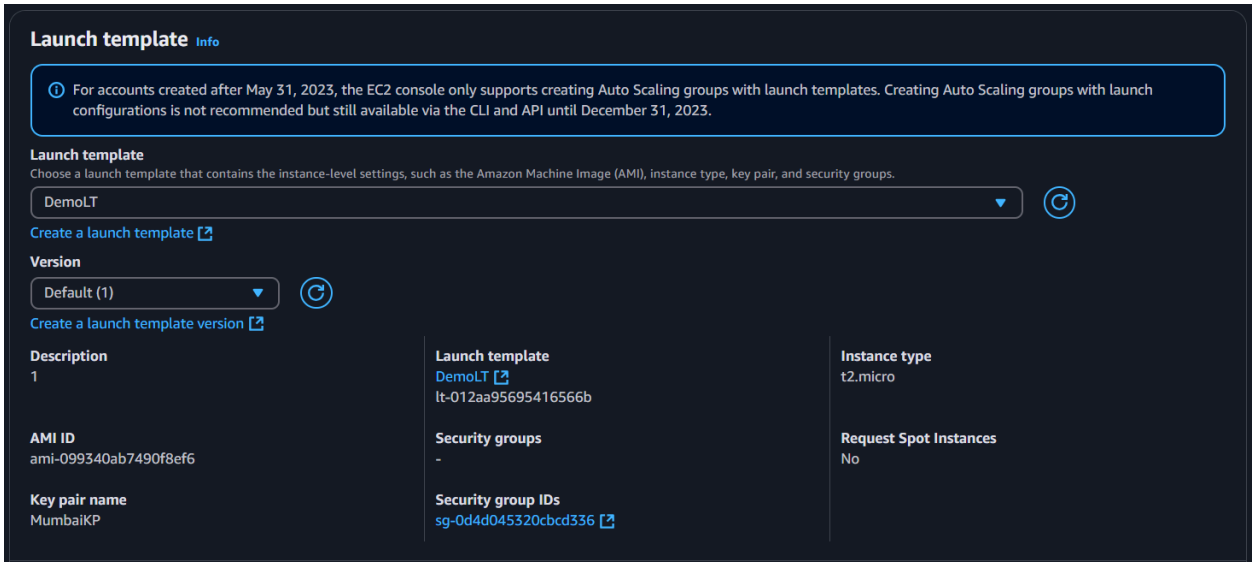
Paste the script below:

bash

```
#!/bin/bash
yum update -y
yum install httpd -y
systemctl start httpd
systemctl enable httpd
echo "<h1>Hello World from $(hostname -f)</h1>" >
/var/www/html/index.html
```

Click **Create Launch Template**
Then you can see you **“DemoLT”** Launch template in LT Dashboard.





❓ Why Do Launch Templates Have Versions?

📦 Launch Templates in AWS are versioned so that you can:

- **🔁 Iterate Safely: Make changes without affecting existing Auto Scaling Groups.**
- **🔒 Maintain Stability: Continue using an older version if a newer one causes issues.**
- **🧪 Test New Configurations: Easily switch between different versions to test instance types, AMIs, or user data.**
- **📂 Track Changes: Each version stores full details of what changed for audit or rollback purposes.**

✅ Best Practice: Create a new version instead of modifying the original template.

🧠 Step 2: Choose Instance Launch Options

🔄 What is "Override Launch Template"?

🔧 Override Launch Template lets you change specific settings like:

- **Instance type**
- **Weighted capacity**
- **Launch template version**

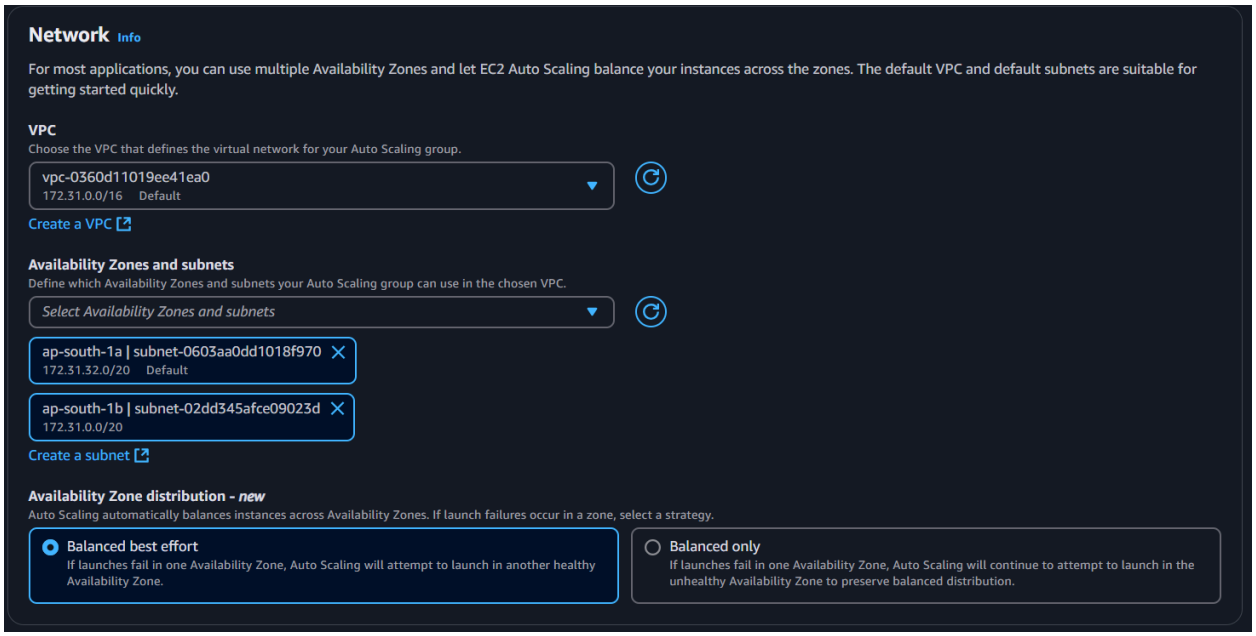
This is helpful when:

- **You want to test different instance types in a single ASG.**
- **You're doing cost optimization by mixing instance sizes.**

🎯 Example: You can run 70% `t3.medium` and 30% `t2.micro` in one group using overrides.

🌐 Network Settings

- **VPC: Choose the VPC where your instances will run.**
- **Availability Zones & Subnets:**
  ✅ Select at least two AZs for high availability and fault tolerance.
  📌 Tip: Multi-AZ improves uptime and helps during zone-specific failures.



- **Click Next ⏭️**

🔌 Step 3 (Optional): Integrate with Other Services

⚖️ Load Balancing

- **Choose: Attach to a new load balancer**
- **Name the load balancer: `DemoASG-ALB`**

🛠️ Load Balancer Settings

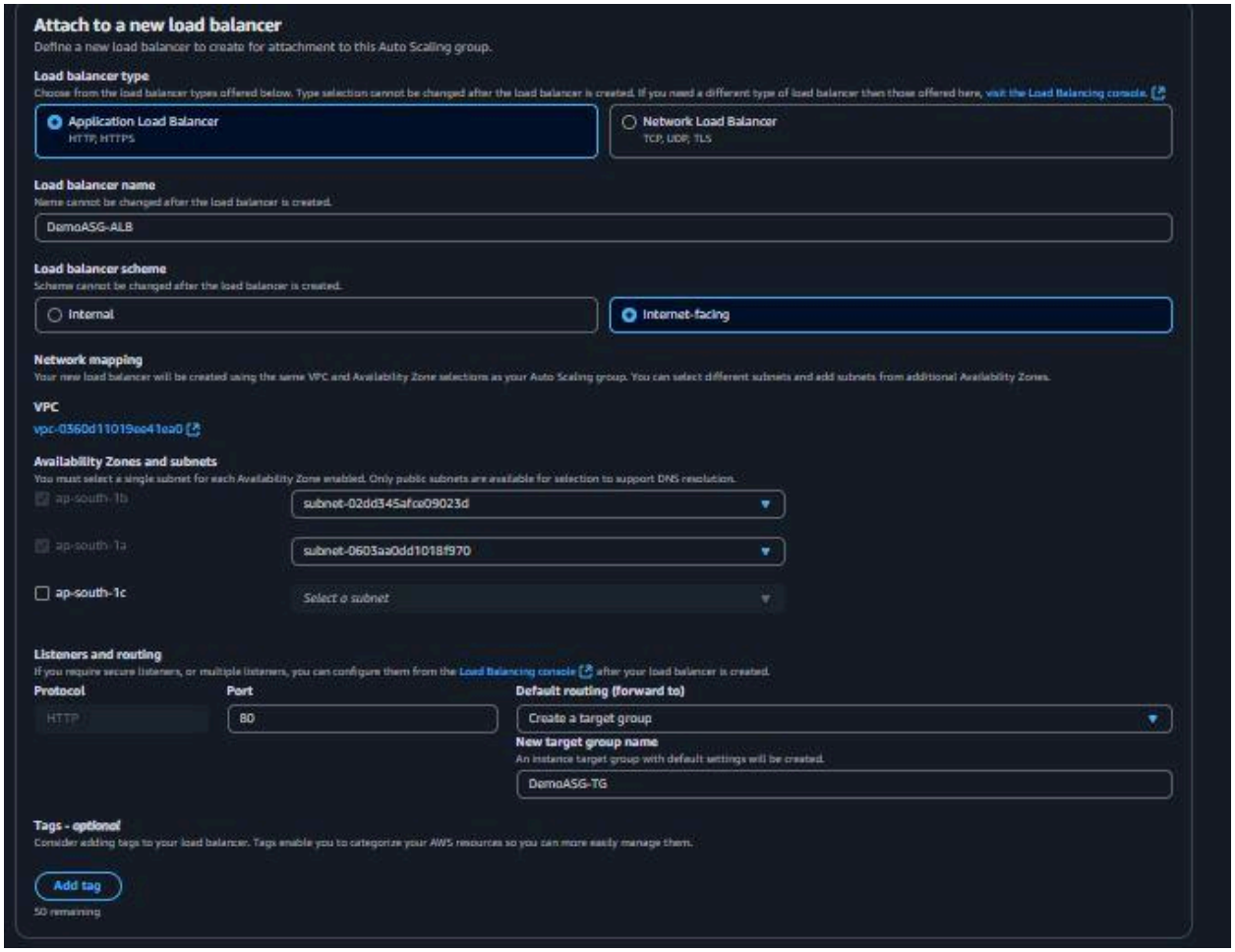
- **Scheme: Choose Internet-facing (for public traffic) or Internal (for internal apps).**
- **VPC: Same VPC as your ASG**

🎯 Create Target Group

- **Name: `DemoASG-TG`**
- **Target Type: Instance**
- **Protocol: HTTP**
- **Port: 80**

✅ Enable Health Checks

- **Turn on Elastic Load Balancing health checks (✅ Recommended)**
  These checks ensure that only healthy instances receive traffic.



- **Click Next ⏭️ to proceed!**

Step 4 – Configure Group Size and Scaling (ASG)

👥 Group Size Configuration

Auto Scaling Groups require you to define how many instances to run at different times.

🎯 Desired Capacity

- **The number of EC2 instances you want running most of the time.**
- **✅ AWS will try to maintain this number unless scaling policies say otherwise.**

Example: If desired capacity is set to 2, the ASG will keep 2 EC2 instances running at all times.

🔽 Minimum Capacity

- **The smallest number of instances the ASG can scale down to.**
- **Ensures you always have at least this many instances running.**

🔼 Maximum Capacity

- **The largest number of instances the ASG can scale up to.**
- **Prevents over-scaling and controls costs.**

🤖 Automatic Scaling Options

❌ No Scaling Policies

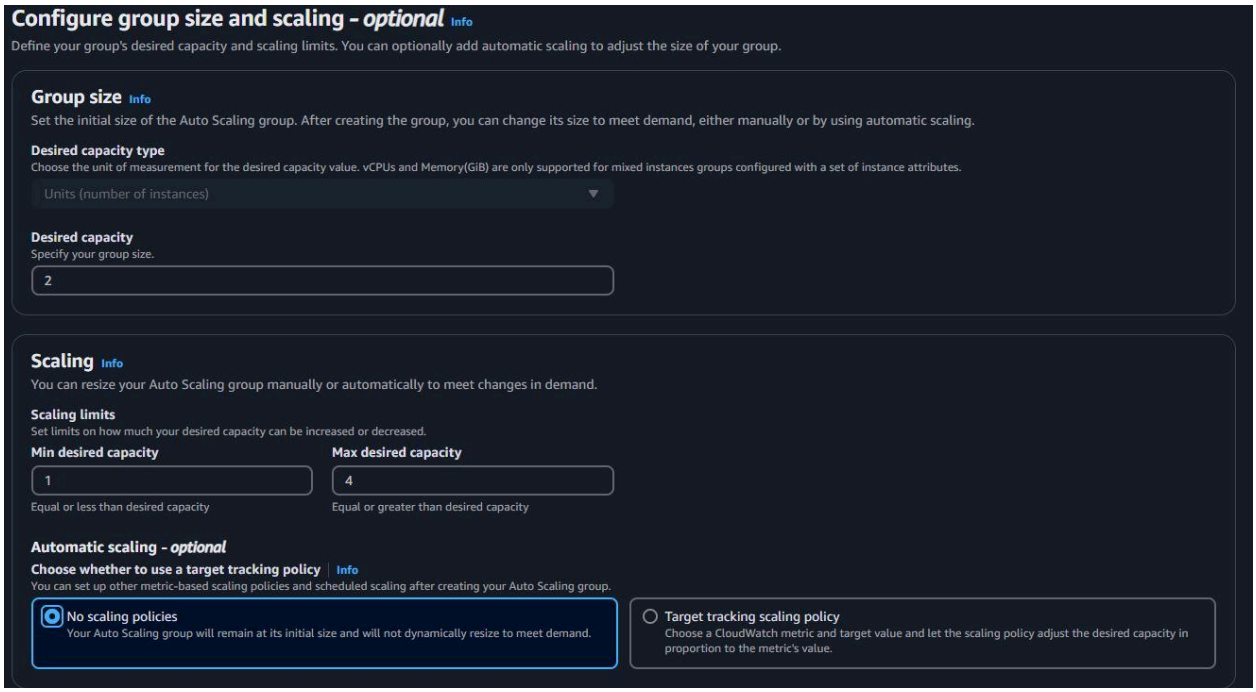
- 🚫 **ASG will not scale automatically.**
- **It will stay at the desired capacity no matter what the load or traffic is.**
- **Useful for manual control or testing setups.**

📈 Target Tracking Scaling Policy

- **AWS uses CloudWatch metrics like CPU utilization, request count, etc.**
- **You set a target value, and AWS adjusts the number of instances to match the demand.**

Example:
Set CPU target at 50% ➡️
If usage goes above that, ASG adds instances.
If usage drops, ASG removes instances.

🛑 For Now: Select "No Scaling Policies"



We will not enable auto-scaling in this setup.
The ASG will stay at the desired number of instances.

Click Next ⏭️ to continue!

📩 Step 5 – Add Notifications (Optional)

🔔 You can configure your ASG to send notifications via Amazon SNS (Simple Notification Service).

Notification Examples:

- **✅ Instance launch success**
- **❌ Instance termination**
- **⚠️ Failed health check**

> **You can skip this step if you don't need alerts right now.**

🏷️ Step 6 – Add Tags (Optional)

🏷️ Tags help with resource organization and cost tracking.

Example Tags:

- `Name`: `DemoASG`
- `Environment`: `Dev` or `Production`
- `Owner`: Your name/team

> **These are helpful but optional — feel free to skip for now.**

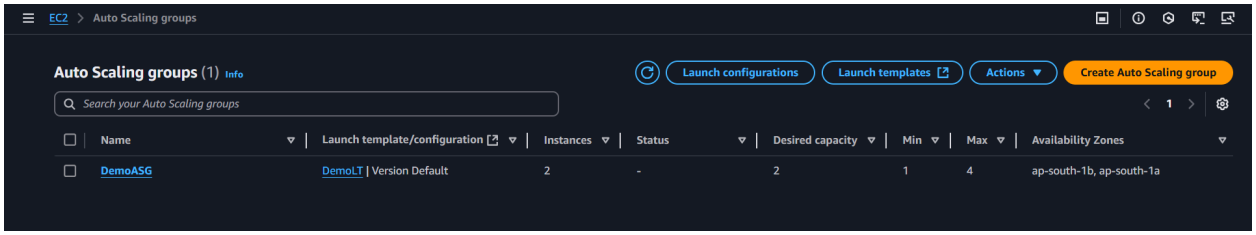
🔍 Step 7 – Review & Create

✅ Review Your Settings:

- **Launch Template selected**
- **Network and Subnets configured**
- **Group Size defined**
- **Scaling policies: No scaling for now**
- **Load Balancer attached (if any)**

🛠️ Double-check all your configurations!

🚀 Click "Create Auto Scaling Group"



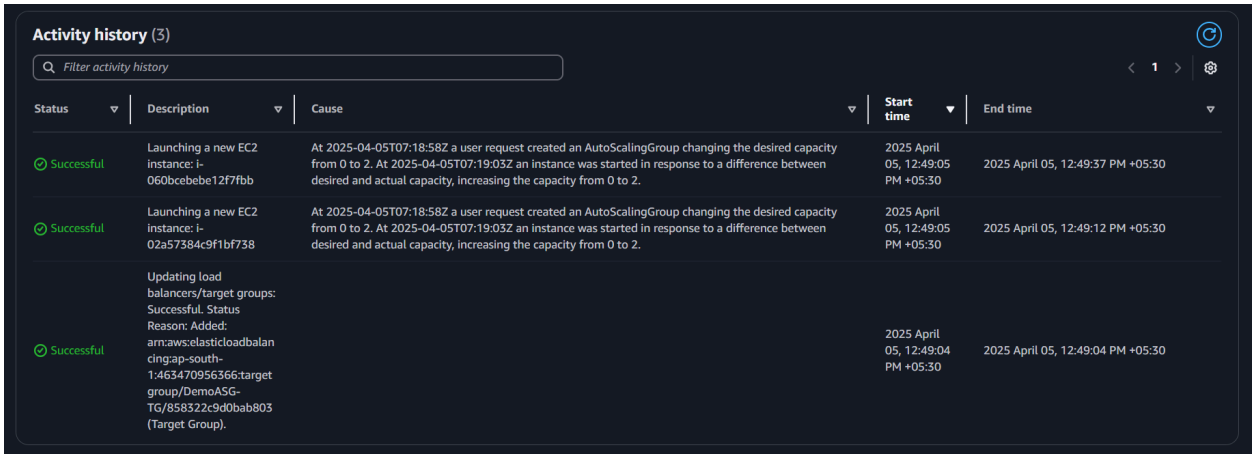
🚀 After Creating Your ASG

Once you've successfully created your Auto Scaling Group (ASG), AWS will **automatically start creating EC2 instances** based on your **desired capacity**.

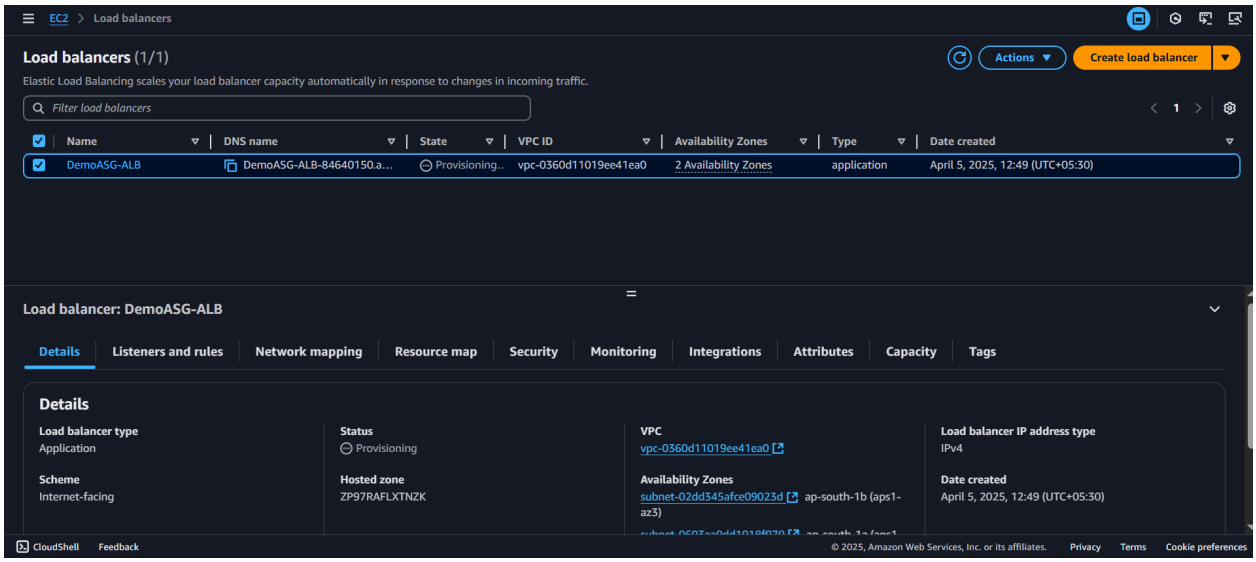
🔍 Where to Check?

Go to the **Activity** tab in your ASG dashboard:

➡️ **EC2 Dashboard**
➡️ **Auto Scaling Groups**
➡️ Select your ASG (e.g., `DemoASG`)
➡️ Click on the **Activity** tab 📋



You can also check for Load balancers Target Groups that we configured while creating ASG. **They will also be created parallelly.**



Note :-

Keep the Security Group Same for Load balancer and launch Templates.

Auto Scaling Groups launch instances based on either a Launch Template or Launch Configuration. If "Auto-assign Public IP" is disabled, the launched instances will not get a public IP or DNS.

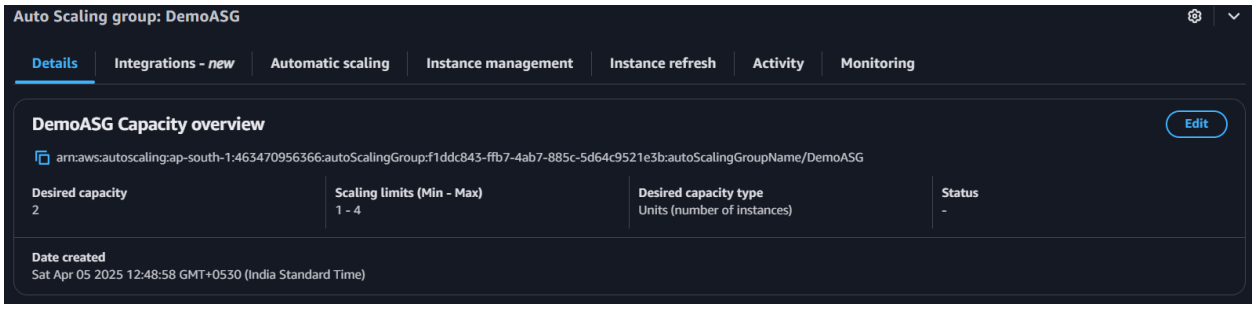
✅ **Fix:**

- **Go to Launch Template or Launch Configuration used by the ASG.**
- **Under Network Interfaces, check if "Auto-assign Public IP" is enabled.**
- **If not, you'll need to:**
    - **Create a new version of the Launch Template with the public IP enabled, or**
    - **Create a new Launch Configuration, if you're still using that.**
    - **Then update your ASG to use the new version.**

🏗️ Components Involved

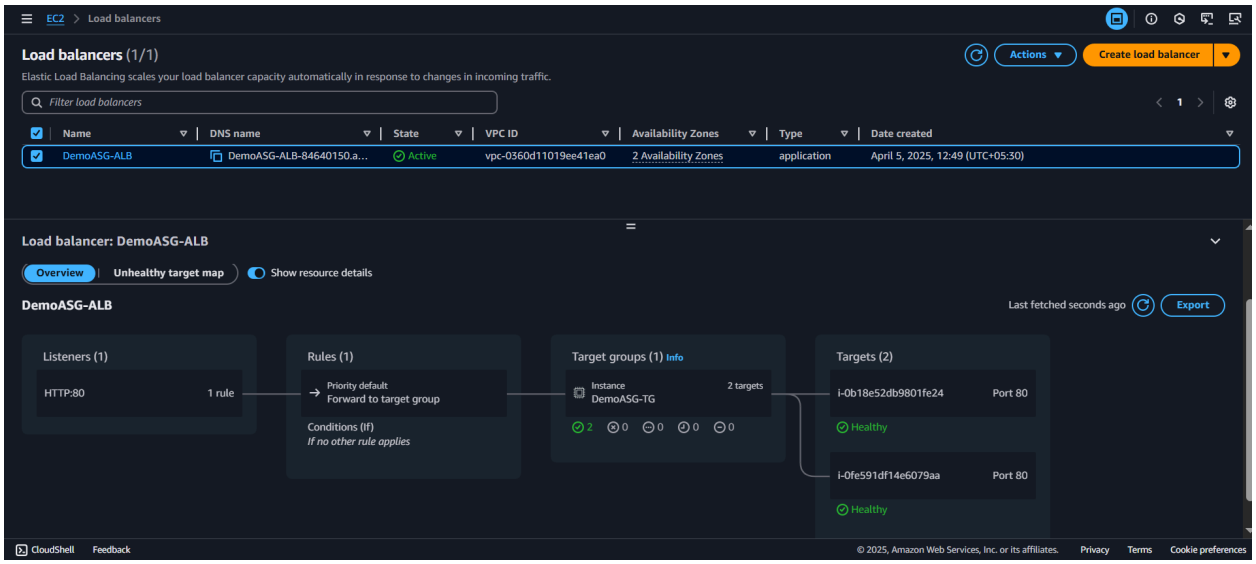
1. Auto Scaling Group (ASG)

- 🔢 **Desired Capacity: 2**
  → ASG always tries to maintain 2 running EC2 instances.
- 🔻 **Minimum Capacity: 2**
  → ASG won't go below 2 instances.
- 🔺 **Maximum Capacity: 4**
  → ASG can scale out up to 4 instances if needed (e.g., due to high load).



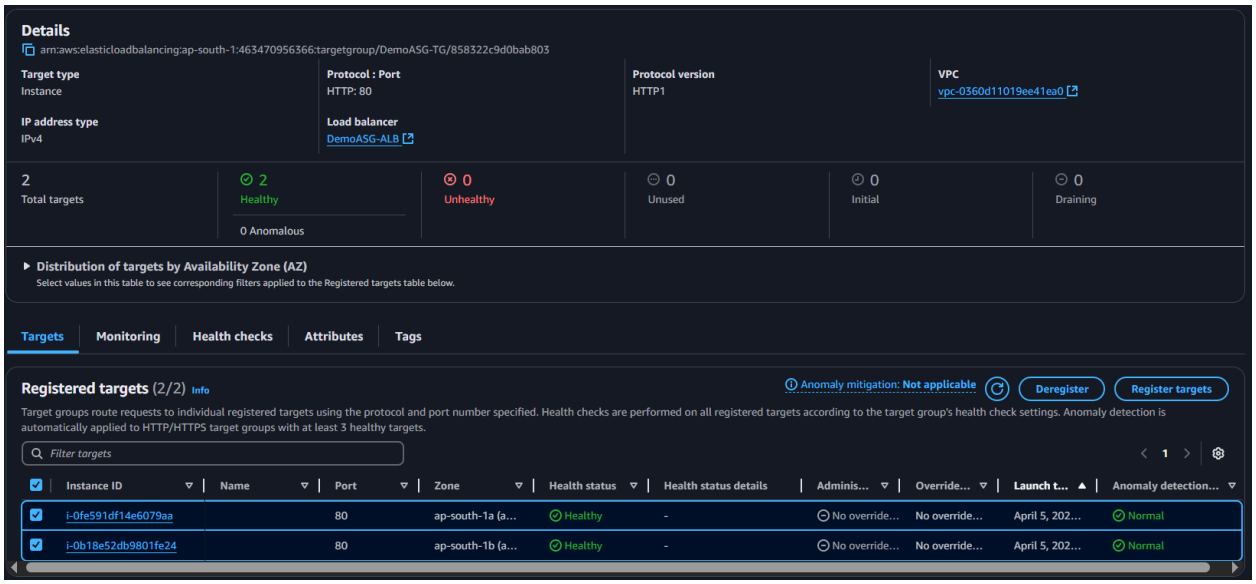
2. Elastic Load Balancer (ELB)

- 📦 **Distributes incoming traffic across multiple EC2 instances.**
- 🔁 **Ensures high availability and fault tolerance.**



3. Target Group

- 🎯 **Contains the list of registered EC2 instances.**
- 💡 **Instances launched by the ASG are automatically added to this target group.**



🔍 What You See in Action

✅ In ASG Activity:

- 🛠️ **EC2 instances are being created automatically as per the desired capacity.**
- 🔗 **Once running, they are registered with the Load Balancer through the Target Group.**

📊 Real-time Behavior:

- **If one instance fails ❌ → ASG launches a new one ✅.**
- **If traffic increases 🌐 → ASG may add instances (up to max = 4).**
- **If traffic drops 📉 → ASG reduces instances (not below min = 2).**

🌐 Check Web App Availability Using AWS Load Balancer

Step-by-Step:

1. **You have a web app deployed on two EC2 instances, each in a different Availability Zone (AZ).**
2. **An Application Load Balancer (ALB) is set up in front of these instances.**
3. **Copy the DNS name of the load balancer**
   (e.g., `my-load-balancer-1234567890.us-west-2.elb.amazonaws.com`)

🔎 Testing the Setup:

- **Open your browser and paste the Load Balancer DNS.**
- **Refresh the page a few times.**

🖥️ First request → goes to EC2 Instance 1

🖥️ Second request → goes to EC2 Instance 2
...and so on.

✅ What It Means:

The Load Balancer is working correctly — it's distributing traffic evenly between the two EC2 instances in separate AZs.

This ensures:

- **High availability**
- **Fault tolerance**
- **Better performance**

AWS Auto Scaling Group (ASG) – Capacity Changes Explained

🔧 Initial Configuration

- **Desired Capacity:2️⃣**
- **Minimum Capacity:1️⃣**
- **Maximum Capacity:4️⃣**

👉 ASG will try to maintain 2 instances running (as per desired capacity), but can scale:

- **🔼 Up to 4 instances**
- **🔽 Down to 1 instance**

🔄 Updated Configuration

- **Desired Capacity:1️⃣**
- **Minimum Capacity:1️⃣**
- **Maximum Capacity:3️⃣**

📉 ASG sees that the new desired capacity is lower than the current number of instances (which was 2).

⚙️ ASG will:

- **Terminate 1 instance to reach the new desired count of 1️⃣.**

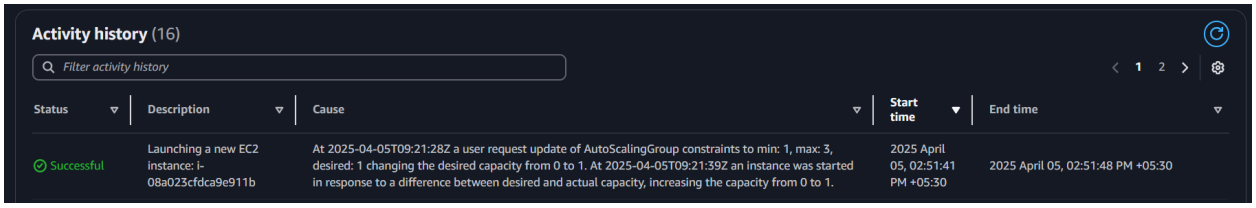
✅ This action will be recorded in the Activity Tab of the ASG.

📋 Summary

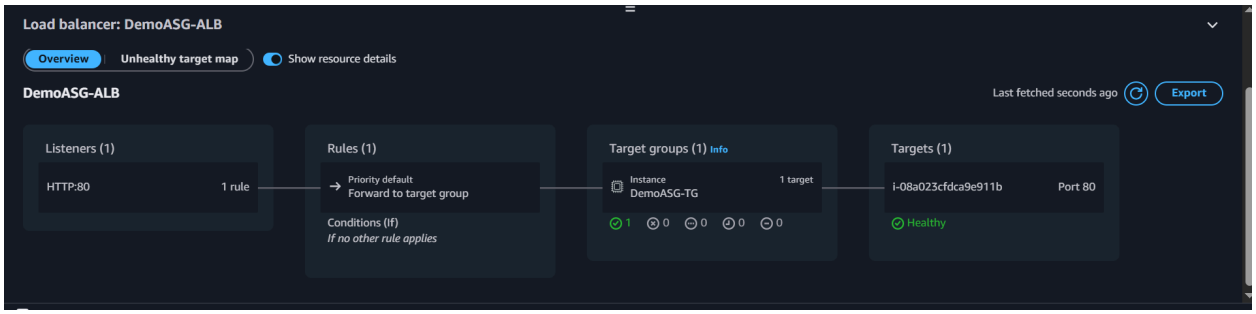
Setting	Before 🕑	After 🔁

Desired Capacity	2️⃣	1️⃣
Min Capacity	1️⃣	1️⃣
Max Capacity	4️⃣	3️⃣

🛠️ ASG Action: Reduces instances from 2 ➡️ 1
🕵️ Check: Activity Tab for logs of this change



Load Balancer :-



📘 AWS Auto Scaling – Automatic Scaling Explained

Auto Scaling helps manage instance counts automatically based on demand 📈📉

🚀 1. Automatic Scaling in ASG

Auto Scaling Group (ASG) can add/remove EC2 instances automatically to match load.
It uses Scaling Policies to decide when to scale in or out.

⚙️ 2. Dynamic Scaling Policies

These policies respond in real-time to CloudWatch alarms 📊

🔹 **a) Simple Scaling**

👉 **One action per alarm.**
🛎️ **Example: If CPU > 80%, add 1 instance.**

🔹 **b) Step Scaling**

📈 **Scale in/out in steps based on how much the metric breaches threshold.**
🛎️ **Example:**

- **CPU > 70% ➡️ add 1 instance**
- **CPU > 85% ➡️ add 2 instances**

🔹 **c) Target Tracking Scaling**

🎯 **Keep a metric at a target value automatically.**
🛎️ **Example: Maintain CPU at 50% by scaling in/out as needed.**

🔮 3. Predictive Scaling Policies

📅 **Uses machine learning to predict future traffic & scale in advance.**
Helps during known patterns, like daily traffic spikes.

⏰ 4. Scheduled Actions

🕑 Set specific times to scale in/out.
🔔 Example:

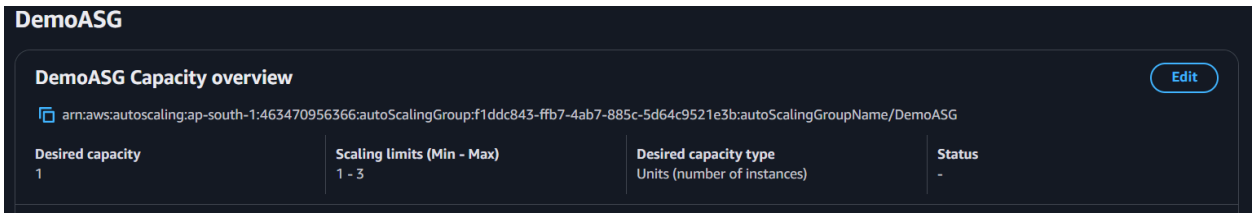
- **Add 2 instances at 8 AM**
- **Remove 2 instances at 8 PM**

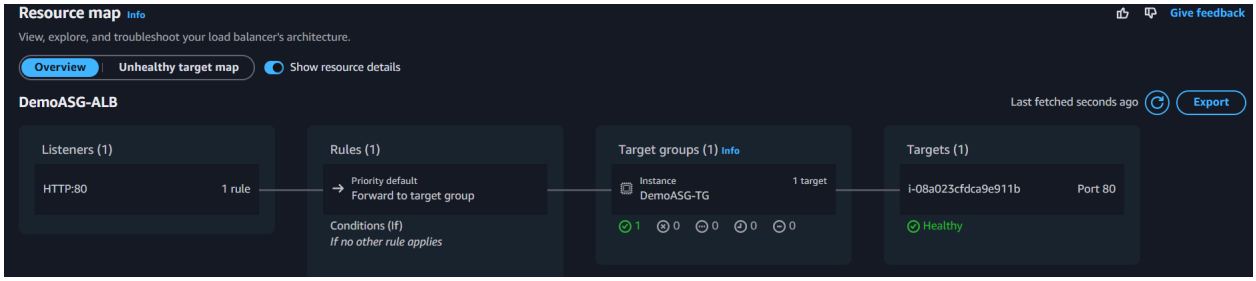
Great for predictable workloads (e.g., office hours).

⚙️ ASG Automatic Scaling – Example

We use Target Tracking Policy in ASG to monitor CPU Utilization 📊.
When CPU crosses the defined threshold 🔺, ASG automatically adds instances.
This ensures app performance by scaling out/in based on real-time load 🚀.

As currently we have Desired Capacity 1,Min 1 and Max 3.



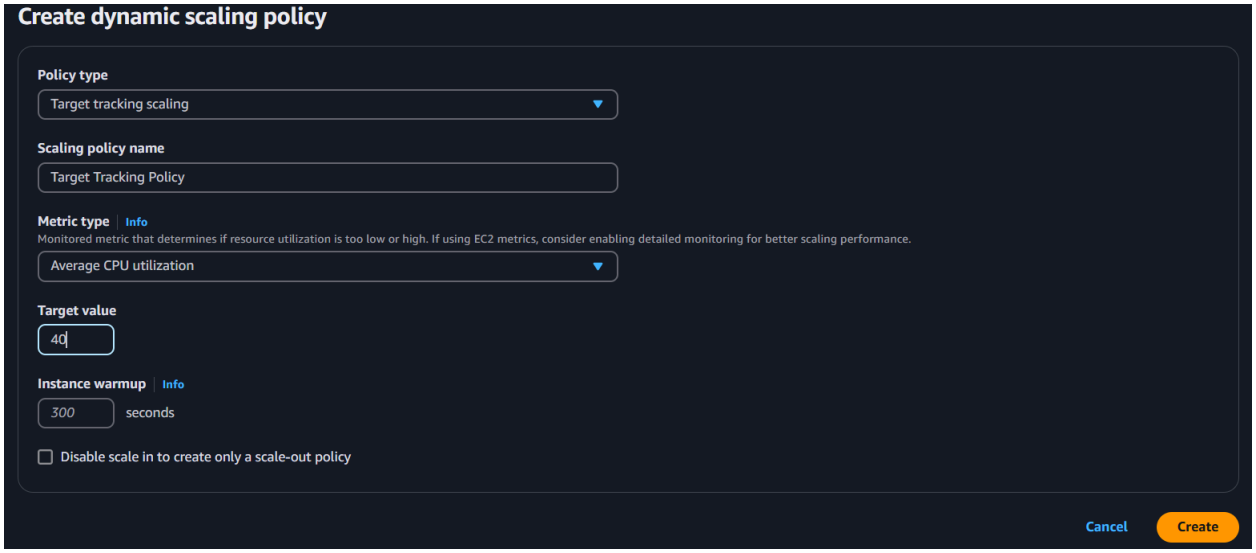


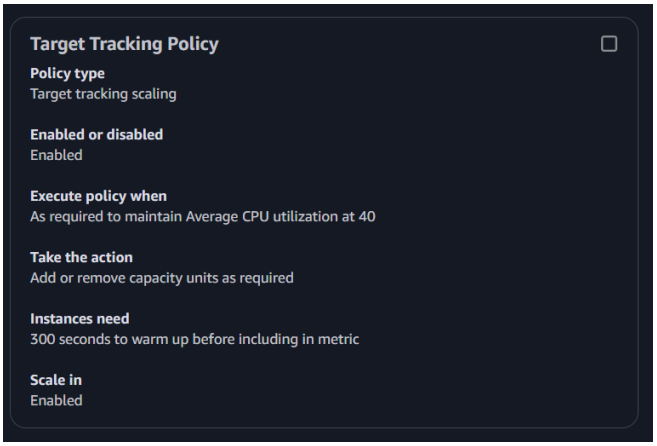
Step 1:
Go to **ASG ➡️ Automatic Scaling ➡️ Dynamic Scaling ➡️ Target Tracking**

Set the following:

- 📑 **Policy Type**
- 🏷️ **Policy Name**
- 📊 **Metric Type** (e.g., CPU Utilization)
- 🎯 **Target Value** (e.g., 50%)
- 🔥 **Instance Warm-up** – time ASG waits before new instance affects scaling

When a new instance is launched, it takes time to **start up and stabilize**.
Warm-up time tells ASG to **pause scaling decisions** during this period.
Prevents **false triggers** and **unnecessary scaling** due to new instance not yet reporting metrics.





🔧 Step 2: Stress EC2 to Increase CPU Utilization

To test **Auto Scaling**, we need to **manually increase CPU usage** on an EC2 instance.

🖥️ Connect to EC2 Instance

Use **EC2 Instance Connect** from AWS Console
➡️ No need for SSH keys!

🧪 Run Stress Commands

Once connected, run the following commands to install and run stress tool:

bash

CopyEdit

```
sudo amazon-linux-extras install epel -y

sudo yum install stress -y

clear

stress -c 4
```

```
Installed:
  stress.x86_64 0:1.0.4-16.el7

Complete!
[ec2-user@ip-172-31-3-142 ~]$ stress -c 4
stress: info: [4148] dispatching hogs: 4 cpu, 0 io, 0 vm, 0 hdd
```

✅ These commands:

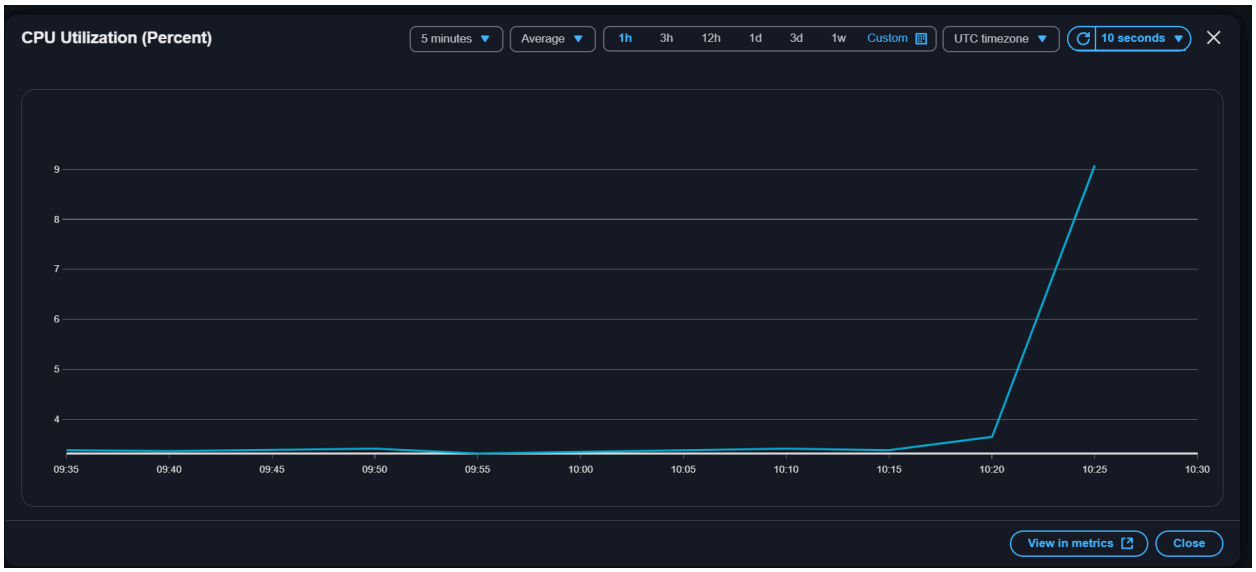
- Install EPEL repo
- Install **stress** tool to simulate high CPU load
- Leverages 4 CPU Units.

📊 Step 3: Monitor CPU & Auto Scaling in Action

After stressing the EC2 instance, let's **monitor the CPU usage** and watch ASG respond.

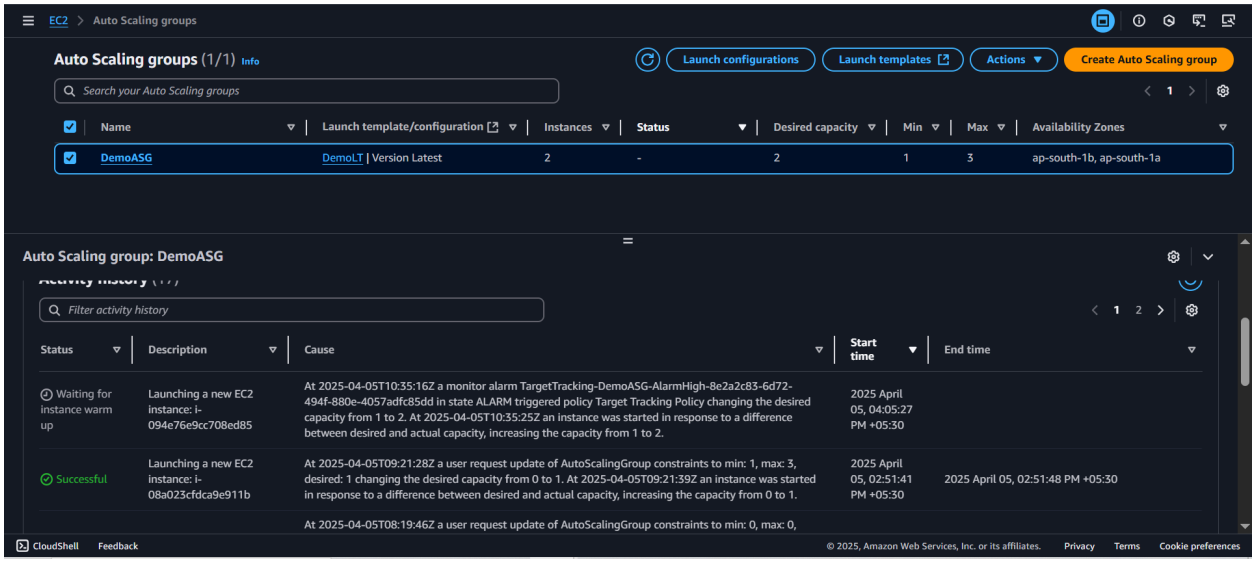
🖥️ Go to Monitoring Tab

1. Open your EC2 instance in AWS Console
2. Click on **Monitoring**
3. Check **CPU Utilization graph**

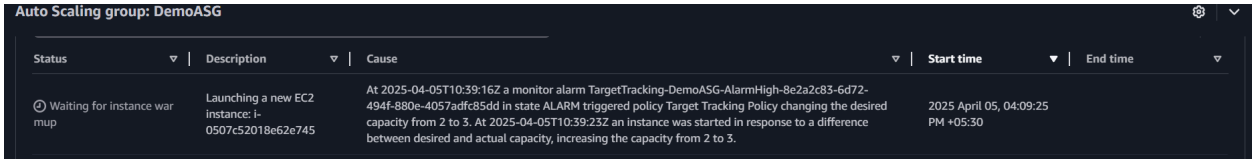


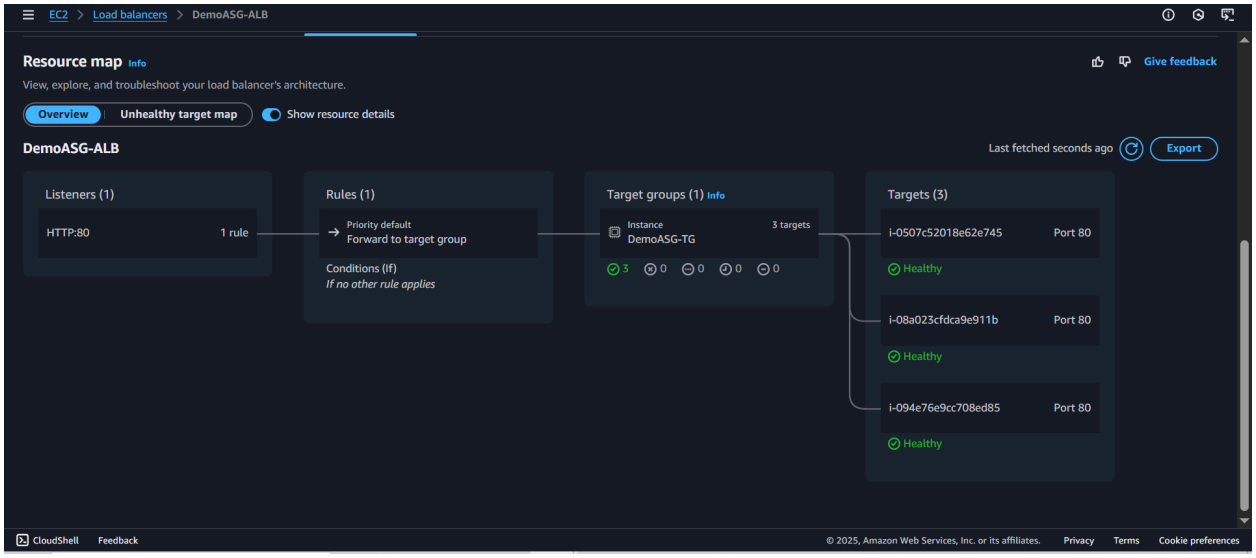
🚀 Auto Scaling Trigger

- When **CPU Utilization crosses 40%** 📈
- ASG automatically **launches a new EC2 instance** 🆕



You can see this in the **ASG Activity Tab** and EC2 instance list or Waiting for instance warmup.





Here 2 More instances are launched because we increased CPU Utilizationa and our Maximum Capacity is 3.

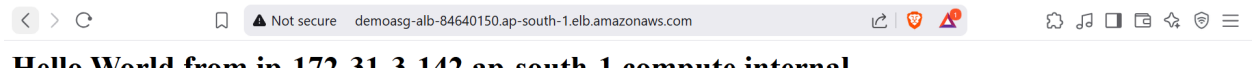
✅ Conclusion: Auto Scaling in Action

By applying a **Target Tracking Scaling Policy** and stressing the EC2 instance:

- 📈 **CPU Utilization** crossed the 40% threshold
- ⚙️ **ASG launched 2 more instances** (total 3) to handle the increased load
- This happened because our **Max Capacity was set to 3**
- 📊 Monitoring and Activity Tabs confirmed the scaling events in real time.

🎯 Result: **ASG scaled up efficiently**, maintaining performance automatically based on demand.

By Checking the Web Application access using our Load Balancer DNS You Can Traffic routing to 3 different instances.



Hello World from ip-172-31-3-142.ap-south-1.compute.internal

Hello World from ip-172-31-34-130.ap-south-1.compute.internal

Hello World from ip-172-31-47-119.ap-south-1.compute.internal

📘 CloudWatch Alarms – Target Tracking in ASG

When you set up a **Target Tracking Scaling Policy**, AWS automatically creates two CloudWatch alarms to monitor your metric (CPUUtilization).

🔔 1. High CPU Alarm

Alarm Name:
`TargetTracking-DemoASG-AlarmHigh-8e2a2c83-6d72-494f-880e-4057adfc85dd`

📅 **Triggered At:**
2025-04-05 16:18:16

📈 **Condition:**
`CPUUtilization > 40%` for **3 datapoints within 3 minutes**

⚙️ **Status:** `In Alarm`
✅ **Actions:** Enabled
➡️ This alarm triggers **scale out** (add instances)

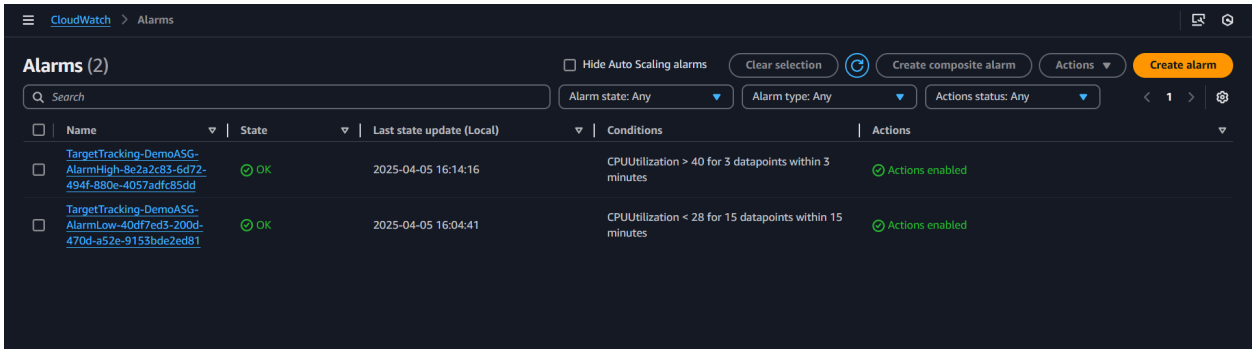
💤 2. Low CPU Alarm

Alarm Name:
`TargetTracking-DemoASG-AlarmLow-40df7ed3-200d-470d-a52e-9153bde2ed81`

📅 **Last Status Update:**
2025-04-05 16:04:41

📉 **Condition:**
`CPUUtilization < 28%` for **15 datapoints within 15 minutes**

🟢 **Status:** `OK`
✅ **Actions:** Enabled
➡️ This alarm triggers **scale in** (remove instances)



🧯 Step 4: Stop Stress & Observe Scale-In

After testing the scale-out, let's now stop the CPU stress and watch **ASG scale in** automatically.

🛑 Stop the Stress Command

In your EC2 terminal, press:

bash

```
Ctrl + C
```

⛔ This will stop:

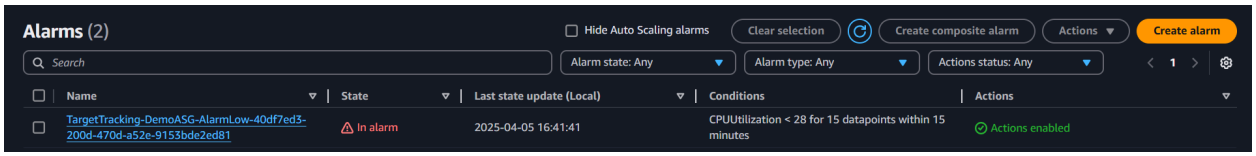
bash

```
stress -c 4
```

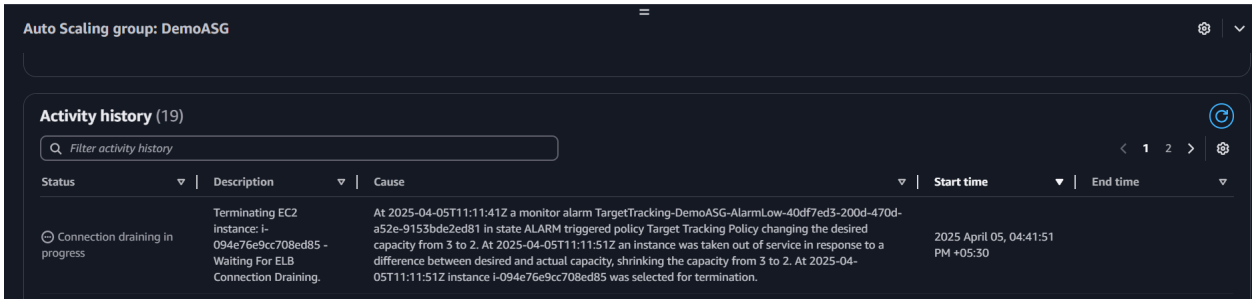
📉 Check CPU Utilization

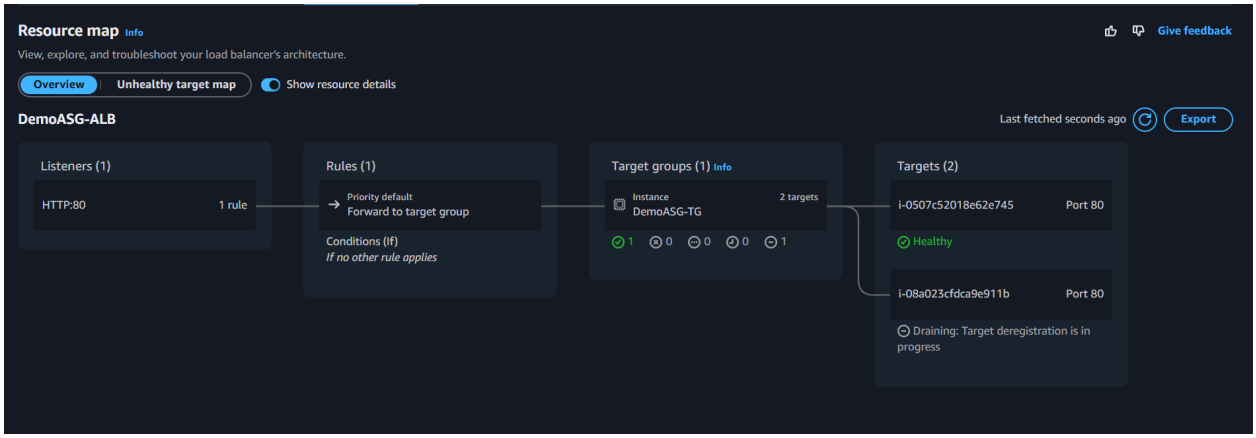
1. Go to **EC2** ➡️ **Monitoring Tab**
2. Watch **CPUUtilization** drop below 28%

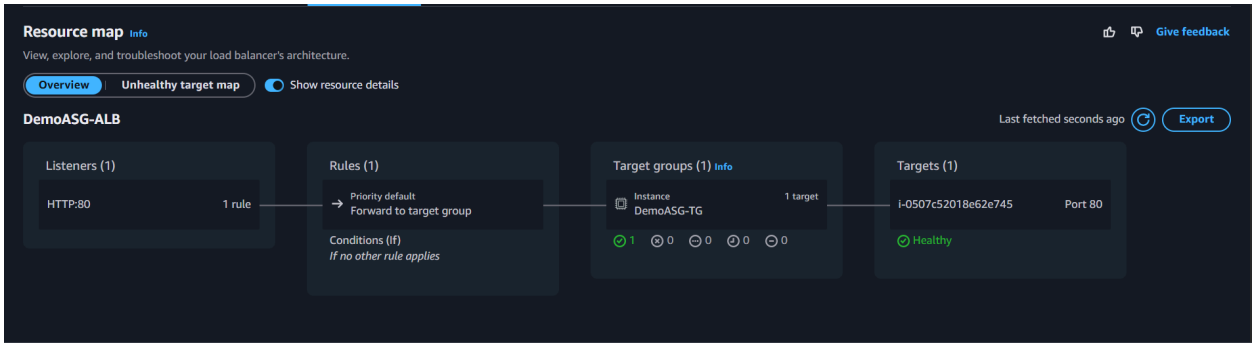
3. Go to **CloudWatch Alarms** – Low CPU alarm should change to `ALARM`



4. Then, head to **ASG** ➡️ **Activity Tab**
   ➡️ You'll see a **Scale-In activity** (instances terminating)







✅ Result

🎯 ASG automatically reduces instance count as load decreases — saving cost while maintaining efficiency.

✅ Final Conclusion: Auto Scaling with Target Tracking in AWS

In this demo, we successfully implemented and tested Automatic Scaling using a Target Tracking Scaling Policy in an Auto Scaling Group (ASG).

🔄 What We Did:

1. **🎯 Configured Target Tracking to monitor CPUUtilization**
2. **🧪 Stressed EC2 instance to trigger high CPU usage**
3. **📈 Observed scale-out as ASG added instances (up to max capacity)**

4. 🧯 **Stopped stress, and watched CPU drop**
5. 📉 **Confirmed scale-in as ASG removed extra instances when CPU stayed below 28% for 15 minutes**
6. 🔔 **Verified CloudWatch alarms automatically created and triggered actions**

🌟 **Key Takeaway:**

✅ **Target Tracking Scaling Policy is an easy and efficient way to maintain performance and optimize cost — scaling EC2 instances up or down automatically based on real-time metrics.**