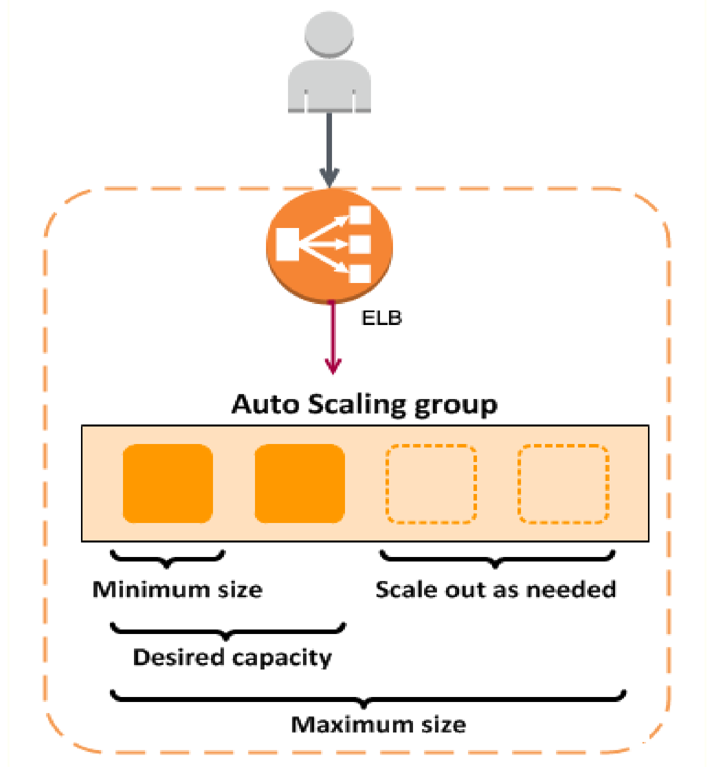
# AUTOSCALING GROUPS

AWS Auto Scaling is a service provided by Amazon Web Services (AWS) that automatically adjusts the number of computing resources (such as virtual servers) in your cloud environment based on the demands of your application. This helps ensure that you have the right number of resources running at any given time, preventing underuse (which can lead to unnecessary costs) or overuse (which can lead to performance issues).

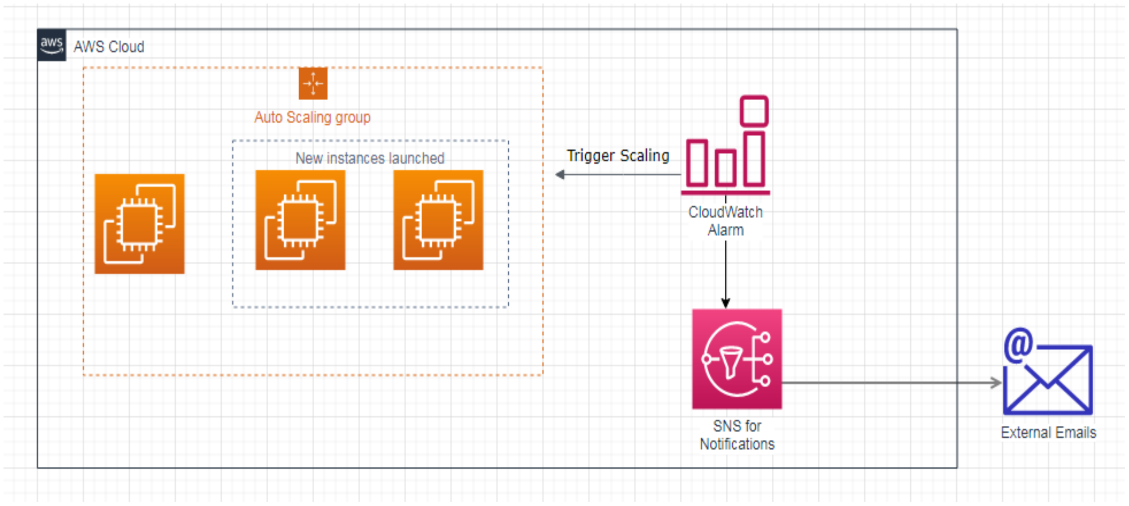


## Here's how AWS Auto Scaling works:

1. Scaling Up: When your application experiences increased traffic or workload (e.g., more users visiting your website), Auto Scaling automatically adds more instances (virtual servers) to handle the extra load. This ensures your app runs smoothly without any delays.

2. **Scaling Down**: When the demand decreases, Auto Scaling reduces the number of instances, so you don't pay for unused resources. This helps to save costs by only using the required resources.

3. **Policies and Alarms**: You can set scaling policies based on metrics like CPU usage or network traffic. For example, you can create a policy that says, "If CPU usage goes above 70% for 10 minutes, add two more instances." AWS also uses **Amazon CloudWatch** to monitor these metrics and trigger scaling actions.



## Benefits of AWS Auto Scaling:

1. **Cost-Effective:** You only pay for the resources you actually need, helping to reduce unnecessary costs.

2. **Improved Performance:** Your application can handle changes in traffic smoothly, without downtime or lag.

3. **Flexibility:** You can set custom scaling policies based on your app's specific needs.

4. **Reliability and Availability**: Explain how autoscaling improves fault tolerance by automatically adjusting resources to handle failures or unexpected traffic surges.

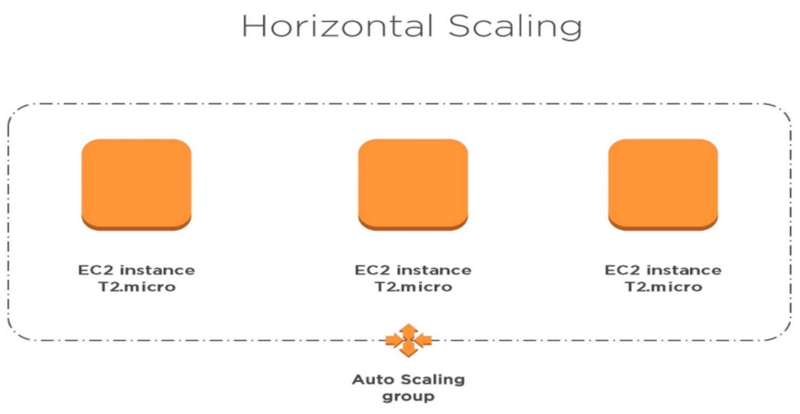
5. **Automation:** Once set up, Auto Scaling adjusts resources automatically, saving you time and effort in managing them.

## Key Concepts of AWS Auto Scaling:

### 1. Horizontal Scaling (Scaling Out/In)

- **Horizontal Scaling** refers to the process of adding or removing **instances** (virtual servers) of a resource, rather than changing the size of an individual instance. This is often referred to as **Scaling Out** (adding more instances) or **Scaling In** (removing instances).
    - **Scaling Out**: When your application experiences an increase in demand (like more users or traffic), you may need additional instances to handle the extra load. AWS Auto Scaling automatically adds more EC2 instances to your Auto Scaling Group.
    - **Scaling In**: When demand decreases (fewer users or less traffic), Auto Scaling will remove some instances to reduce costs and avoid overprovisioning.

**Example**: Imagine you run a website that gets more visitors during the holiday season. With horizontal scaling, you can automatically add more instances (scale out) to handle the increased traffic, and later remove them (scale in) when traffic goes back to normal.

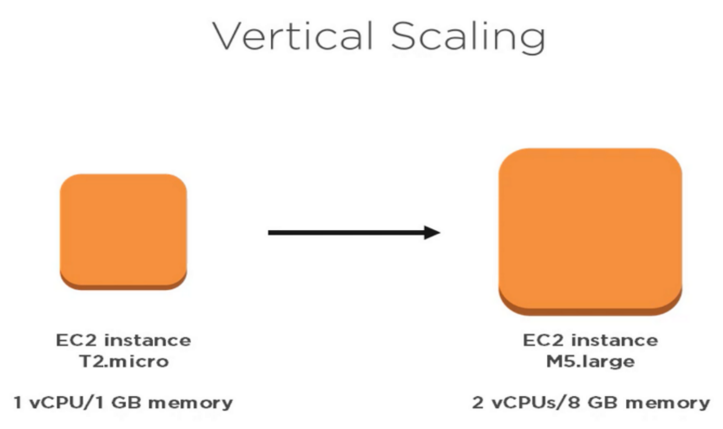


## 2. Vertical Scaling (Scaling Up/Down)

- **Vertical Scaling** involves changing the size of an individual resource, such as increasing or decreasing the **CPU**, **RAM**, or other hardware capabilities of a single instance, rather than adding more instances.
  - **Scaling Up**: This means upgrading the instance to a more powerful one with more CPU or RAM. You would do this if your application requires more power, but you want to keep the same number of instances.
  - **Scaling Down**: This means downgrading the instance to a less powerful one, reducing the CPU or RAM when your application needs less computing power.

**Example**: If you have a virtual server that is underperforming because it doesn't have enough memory (RAM), you could scale it up by changing to a more powerful instance type. Conversely, if the server is underutilized, you might scale it down to save costs.

**Note**: AWS Auto Scaling primarily focuses on **horizontal scaling**. Vertical scaling usually requires manual intervention or using features like **Elastic Load Balancer** for managing traffic across instances.



**3. Scaling Policies**

- **Scaling Policies** define the **rules or conditions** that trigger scaling actions (either scaling out or scaling in) based on specific **metrics** like CPU usage, memory usage, or network traffic.
  - **Scale-Up Policy**: This rule tells Auto Scaling to add more instances when specific conditions are met. For example, if CPU usage goes above 70% for 5 minutes, a scale-up policy would trigger the creation of more instances.
  - **Scale-Down Policy**: This rule tells Auto Scaling to reduce the number of instances when certain conditions are met. For example, if CPU usage drops below 30% for 5 minutes, Auto Scaling would remove instances to save costs.

**Example**: If you set a scaling policy that states, "When CPU usage exceeds 80% for more than 5 minutes, add 2 instances," Auto Scaling will monitor your EC2 instances and automatically add more resources when the workload increases. Similarly, if the load decreases, it will remove unnecessary instances.

Scaling policies help ensure that your application runs efficiently by adjusting the number of resources based on real-time demand.

**4. Cooldown Periods**

- **Cooldown Periods** are a feature of AWS Auto Scaling that helps prevent **rapid scaling actions**. After a scaling event (either scale-up or scale-down), AWS waits for a specified **cooldown period** before initiating another scaling action.
  - The cooldown period helps avoid the scenario where Auto Scaling keeps launching or terminating instances too quickly. This could happen if the scaling policy triggers actions multiple times in a short period, which could lead to instability or unnecessary costs.
  - **Example**: If your scale-up policy triggers the addition of 2 instances, Auto Scaling will wait for a cooldown period (like 5 minutes) before

evaluating whether it needs to scale further. This allows time for the system to stabilize and avoid adding too many instances too quickly.

**Why Cooldown Periods Matter**: Without cooldown periods, Auto Scaling might trigger multiple scaling actions in response to sudden traffic spikes or dips. This can cause unnecessary resource changes and disrupt application performance. The cooldown period ensures that resources are adjusted only when necessary and not constantly fluctuating.

## How AWS Auto Scaling Works:

### 1. Scaling Triggers

Scaling triggers are the conditions that activate Auto Scaling, typically based on resource demand:

- **CPU Usage**: High CPU usage (e.g., above 80%) triggers scaling actions to add more instances.
- **Memory Usage**: High memory usage (requires custom metrics) can trigger scaling.
- **Queue Length/Request Count**: A growing number of requests may trigger scaling to add more instances.
- **Disk I/O/Network Traffic**: High disk or network traffic can also prompt scaling actions.

### 2. Scaling Metrics

Metrics measure system performance and help decide when to scale:

- **CPU Utilization**: High CPU usage can trigger scaling actions.
- **Memory Utilization**: High memory usage can be a scaling trigger (via custom metrics).
- **Response Time**: Increased response time indicates resource strain, triggering scaling.

- **Request Count/Queue Length**: A high number of requests or queue length can trigger scaling actions.
- **Disk I/O/Network Traffic**: Heavy disk or network load can prompt scaling.

### 3. Scaling Actions

Scaling actions are the steps Auto Scaling takes when triggers are met:

- **Scaling Out**: Adding instances to handle increased demand.
- **Scaling In**: Removing instances when demand decreases.
- **Scaling Up**: Upgrading an instance to a more powerful type when needed.
- **Scaling Down**: Downgrading to a smaller instance type to save costs.
- **Load Balancing**: Ensures traffic is evenly distributed across instances during scaling actions.

## Auto Scaling Architecture and Components:

### 1. Auto Scaling Group (ASG)

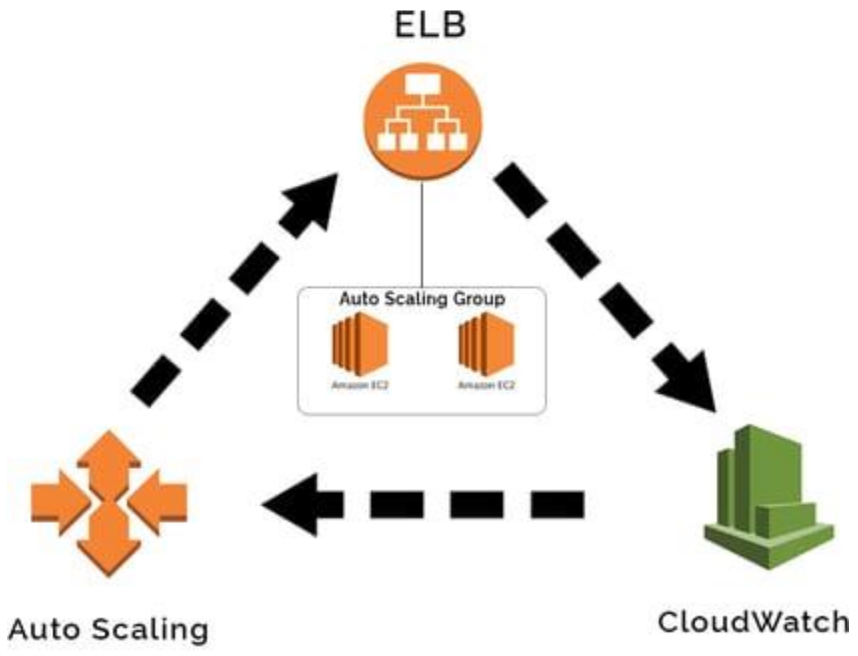
An **Auto Scaling Group (ASG)** manages a group of **EC2 instances**, automatically adjusting the number of instances based on demand. It ensures that the application has the right amount of resources by scaling up or down as needed.

### 2. Load Balancer

A **Load Balancer** distributes incoming traffic across multiple instances, ensuring an even load. During scaling, it automatically adds new instances to traffic distribution and removes instances when they are no longer needed.

### 3. Monitoring Tools

**Monitoring Tools** like **Amazon CloudWatch** track system performance, such as CPU and memory usage, and trigger scaling actions when thresholds are met. Other cloud platforms have similar tools, like Azure Monitor and Google Cloud Operations Suite.



## EXECUTION :

### Launch Configurations and Launch Templates in AWS Auto Scaling :

In AWS Auto Scaling, **Launch Configurations** and **Launch Templates** are used to define how new EC2 instances are created when scaling actions are triggered. Here's an overview of each –

### 1. Launch Configuration

A **Launch Configuration** is a template used to define the settings for new EC2 instances in an Auto Scaling Group. It specifies:

- **Amazon Machine Image (AMI)**: The image used to launch the instance.
- **Instance Type**: The type of instance (e.g., t2.micro, m5.large).
- **Security Groups**: The security groups associated with the instance.
- **Key Pair**: SSH key pairs for accessing the instance.

Once created, launch configurations cannot be modified. If changes are needed, a new launch configuration must be created and associated with the Auto Scaling Group.

**2. Launch Template**

A **Launch Template** is a more flexible and advanced version of a launch configuration. It offers additional features such as:

- **Versioning**: Allows multiple versions of a template, making it easier to manage different configurations over time.
- **Additional Parameters**: Includes more configuration options like block device mappings, network interfaces, and tags.
- **Customizations**: Supports more advanced features like Elastic IP assignments, placement groups, and more.

Launch templates provide greater flexibility for Auto Scaling Groups and are recommended for most use cases over launch configurations.

## 1. SET UP LAUNCH TEMPLATE :

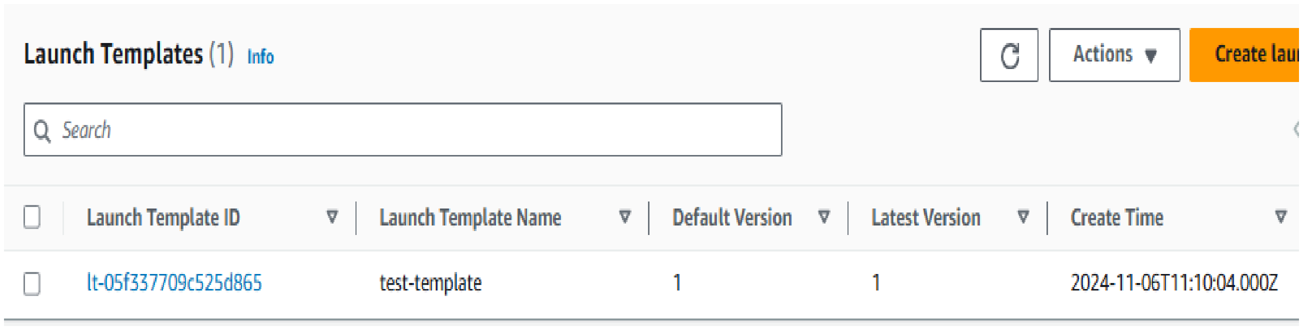
→Go to the EC2 Dashboard, On the left-hand panel, under Instances, select Launch Templates. Now click on **create launch template** and fill the required fields.

**Fill in the required fields:**

- **Launch template name:** A name for your template.
- **Version description**: Optional but useful for tracking versions.
- **AMI ID:** Select or provide the AMI ID for the EC2 instance you want to use.

- **Instance type:** Select the EC2 instance type (e.g., t2.micro, m5.large).
- **Key pair**: Select the key pair for SSH access.
- **Security groups:** Select the security group for the EC2 instance.
- **IAM role:** (Optional) If needed, select an IAM role for the EC2 instances.
- **User data:** (Optional) Provide user data (like bash scripts) to run at startup.

→After configuring the template, click **Create launch template**.



| | Launch Template ID | Launch Template Name | Default Version | Latest Version | Create Time |
|---|---|---|---|---|---|
| | lt-05f337709c525d865 | test-template | 1 | 1 | 2024-11-06T11:10:04.000Z |

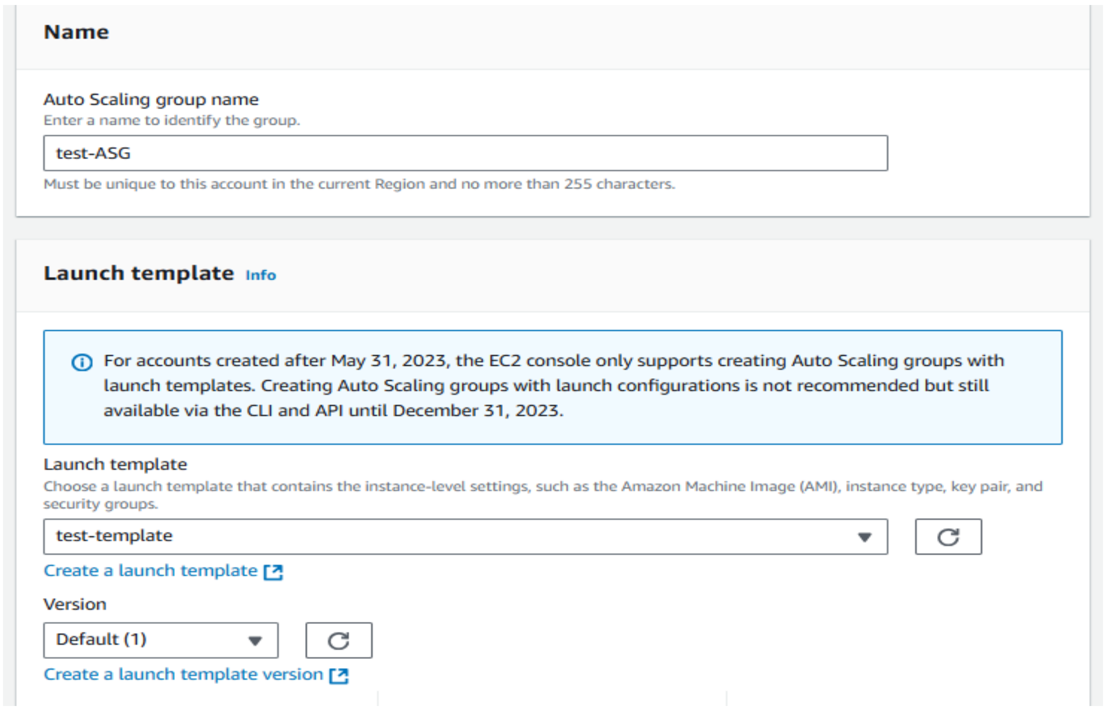
## 2. CREATE AUTOSCALING GROUP :

- Click **Create Auto Scaling group**.
- On the Create Auto Scaling group page, under the Launch template section, select Launch template.
- Choose the **Launch template name** (that we created above) and **Version** (if you have multiple versions of the template).
- Provide the name for the Auto Scaling group (e.g., my-auto-scaling-group).

→Auto Scaling Group Settings

- **VPC, AZ and Subnet**: Select the VPC and subnets in which the EC2 instances should be launched.
- **Load Balancer (Optional)**: If using a load balancer (e.g., Elastic Load Balancer), select the appropriate load balancer.
- **Desired Capacity**: Set the initial number of instances for the Auto Scaling Group. (should always be more than minimum capacity).

- **Min Size**: Set the minimum number of instances.(should always be greater than zero else it won't trigger ASG when no instance is available).
- **Max Size**: Set the maximum number of instances.



Name

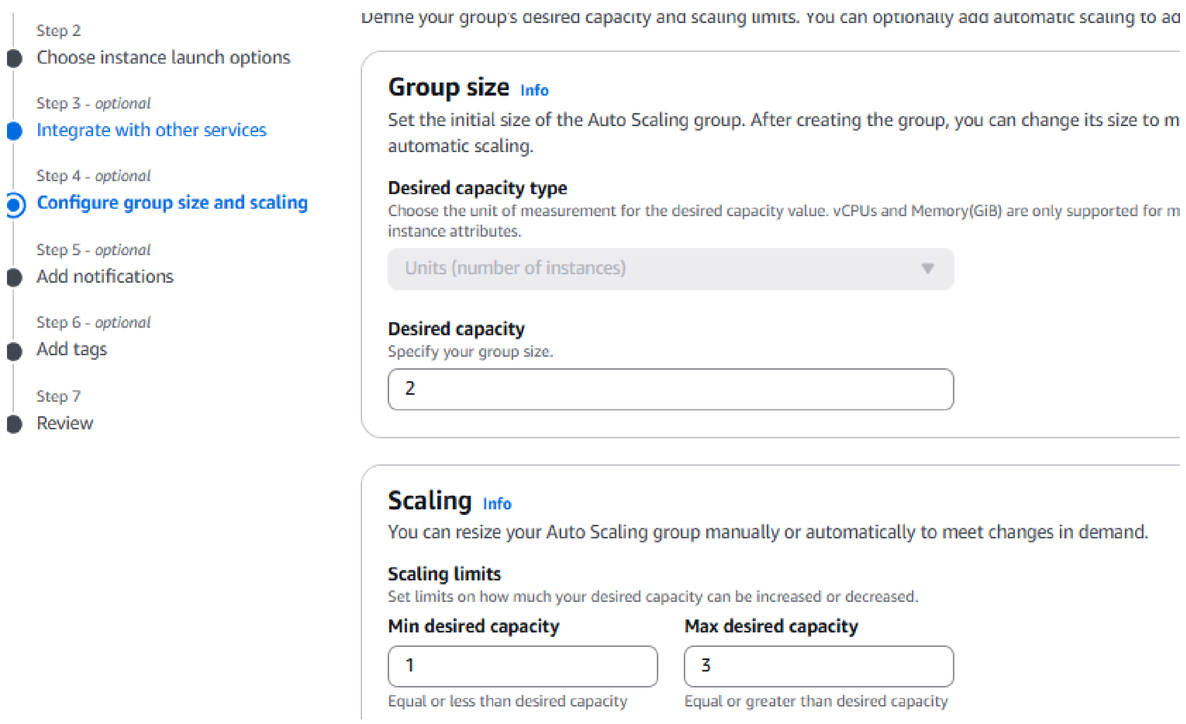
Auto Scaling group name
Enter a name to identify the group.
test-ASG
Must be unique to this account in the current Region and no more than 255 characters.

Launch template Info

For accounts created after May 31, 2023, the EC2 console only supports creating Auto Scaling groups with launch templates. Creating Auto Scaling groups with launch configurations is not recommended but still available via the CLI and API until December 31, 2023.

Launch template
Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.
test-template
Create a launch template

Version
Default (1)
Create a launch template version



Step 2
Choose instance launch options

Step 3 - optional
Integrate with other services

Step 4 - optional
Configure group size and scaling

Step 5 - optional
Add notifications

Step 6 - optional
Add tags

Step 7
Review

Define your group's desired capacity and scaling limits. You can optionally add automatic scaling to ad

Group size Info
Set the initial size of the Auto Scaling group. After creating the group, you can change its size to m automatic scaling.

Desired capacity type
Choose the unit of measurement for the desired capacity value. vCPUs and Memory(GiB) are only supported for m instance attributes.
Units (number of instances)

Desired capacity
Specify your group size.
2

Scaling Info
You can resize your Auto Scaling group manually or automatically to meet changes in demand.

Scaling limits
Set limits on how much your desired capacity can be increased or decreased.

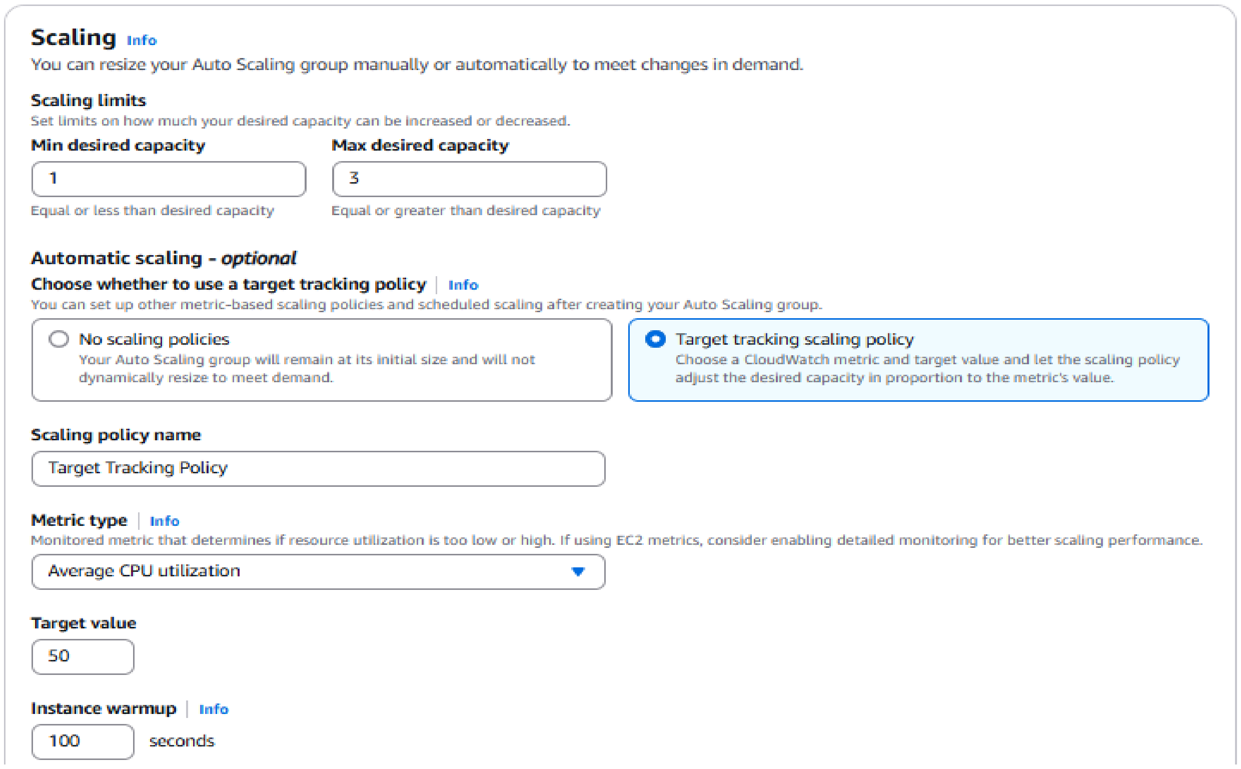
| Min desired capacity | Max desired capacity |
|---|---|
| 1 | 3 |
| Equal or less than desired capacity | Equal or greater than desired capacity |

→Configure Scaling Policies:

Set the rules for how the Auto Scaling group should scale:

- **Scaling Policies**: Define scaling policies (e.g., scale out when CPU usage is high or scale in when CPU usage is low).
- **Target Tracking**: (Optional) Set up target tracking policies, for example, maintaining an average CPU utilization of 50%.

→Here we configured CPU utilization as scaling policy to trigger ASG.



Scaling Info

You can resize your Auto Scaling group manually or automatically to meet changes in demand.

Scaling limits

Set limits on how much your desired capacity can be increased or decreased.

Min desired capacity: 1 (Equal or less than desired capacity)

Max desired capacity: 3 (Equal or greater than desired capacity)

Automatic scaling - *optional*

Choose whether to use a target tracking policy | Info

You can set up other metric-based scaling policies and scheduled scaling after creating your Auto Scaling group.

○ No scaling policies — Your Auto Scaling group will remain at its initial size and will not dynamically resize to meet demand.

● Target tracking scaling policy — Choose a CloudWatch metric and target value and let the scaling policy adjust the desired capacity in proportion to the metric's value.

Scaling policy name: Target Tracking Policy

Metric type | Info

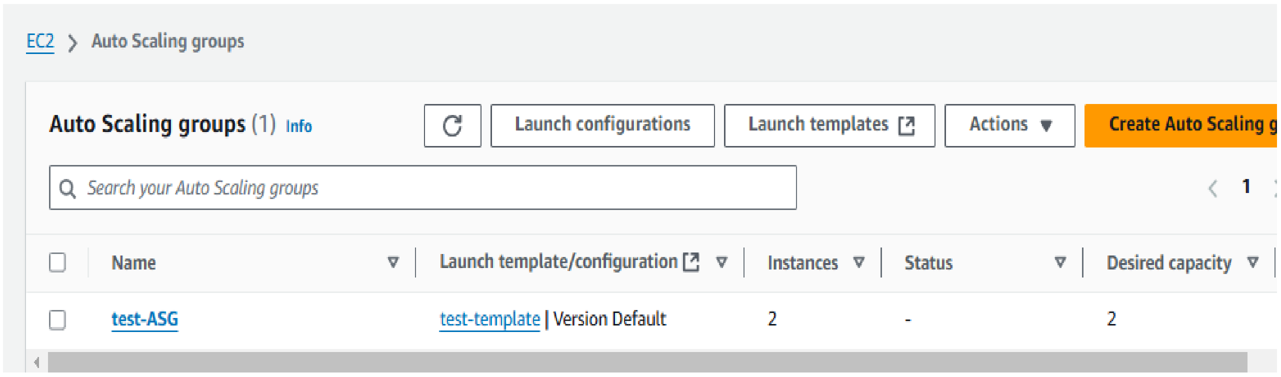
Monitored metric that determines if resource utilization is too low or high. If using EC2 metrics, consider enabling detailed monitoring for better scaling performance.

Average CPU utilization

Target value: 50

Instance warmup | Info: 100 seconds

Created Auto Scaling Group.

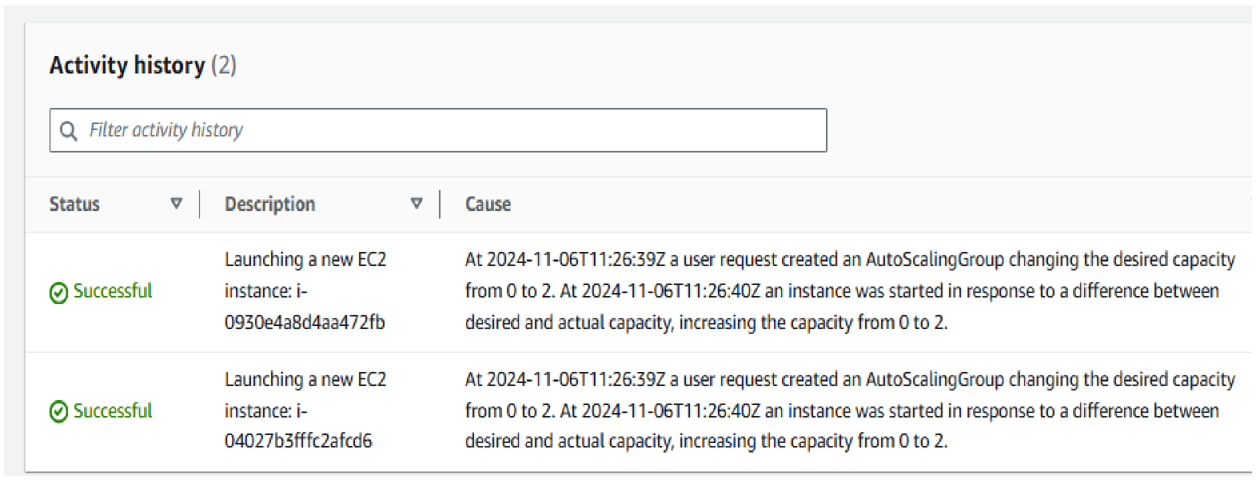


EC2 > Auto Scaling groups

Auto Scaling groups (1) Info

| Name | Launch template/configuration | Instances | Status | Desired capacity |
|---|---|---|---|---|
| test-ASG | test-template \| Version Default | 2 | - | 2 |

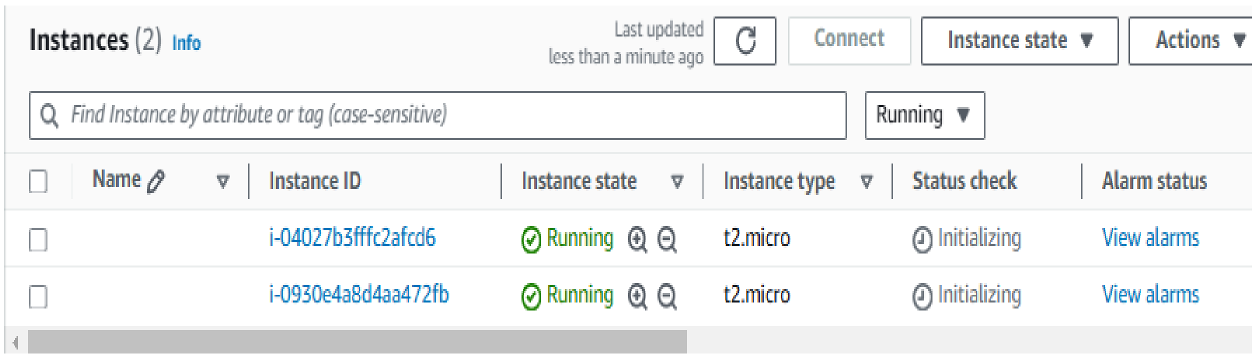
Two instances were launched (desired capacity)



**Activity history** (2)

Filter activity history

| Status | Description | Cause |
|---|---|---|
| ✓ Successful | Launching a new EC2 instance: i-0930e4a8d4aa472fb | At 2024-11-06T11:26:39Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2024-11-06T11:26:40Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2. |
| ✓ Successful | Launching a new EC2 instance: i-04027b3fffc2afcd6 | At 2024-11-06T11:26:39Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2024-11-06T11:26:40Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2. |



**Instances** (2) Info

Last updated less than a minute ago

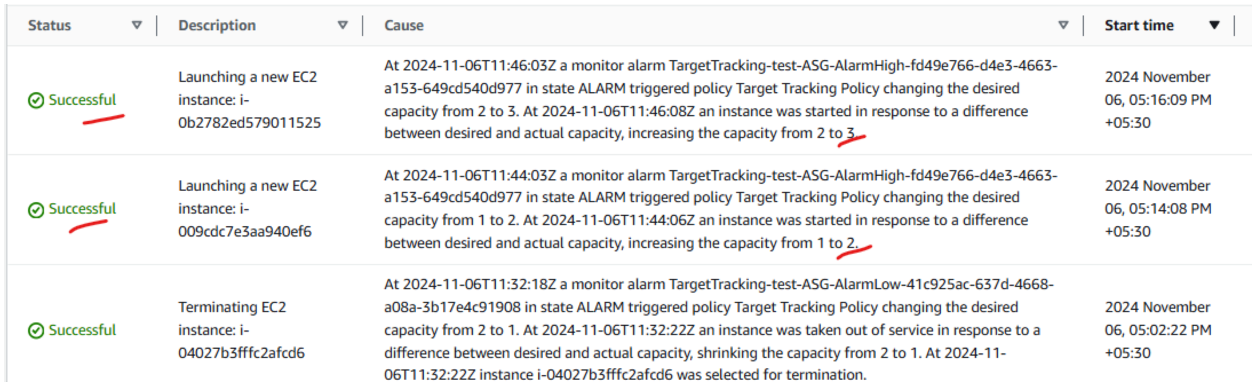
Find Instance by attribute or tag (case-sensitive)

Running ▼

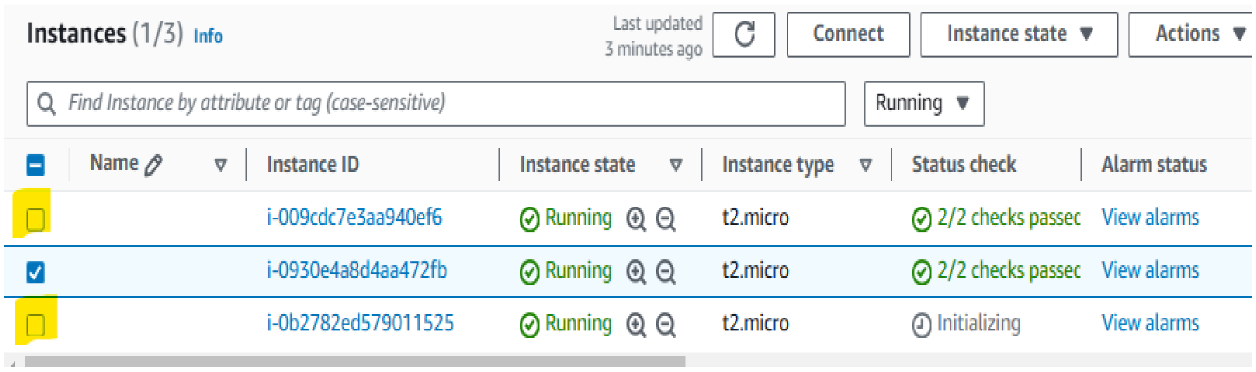
| Name | Instance ID | Instance state | Instance type | Status check | Alarm status |
|---|---|---|---|---|---|
| | i-04027b3fffc2afcd6 | ✓ Running | t2.micro | Initializing | View alarms |
| | i-0930e4a8d4aa472fb | ✓ Running | t2.micro | Initializing | View alarms |

After Increasing the CPU load.

New instances were launched as CPU utilization increased.



| Status | Description | Cause | Start time |
|---|---|---|---|
| ✓ Successful | Launching a new EC2 instance: i-0b2782ed579011525 | At 2024-11-06T11:46:03Z a monitor alarm TargetTracking-test-ASG-AlarmHigh-fd49e766-d4e3-4663-a153-649cd540d977 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 3. At 2024-11-06T11:46:08Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3. | 2024 November 06, 05:16:09 PM +05:30 |
| ✓ Successful | Launching a new EC2 instance: i-009cdc7e3aa940ef6 | At 2024-11-06T11:44:03Z a monitor alarm TargetTracking-test-ASG-AlarmHigh-fd49e766-d4e3-4663-a153-649cd540d977 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 1 to 2. At 2024-11-06T11:44:06Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2. | 2024 November 06, 05:14:08 PM +05:30 |
| ✓ Successful | Terminating EC2 instance: i-04027b3fffc2afcd6 | At 2024-11-06T11:32:18Z a monitor alarm TargetTracking-test-ASG-AlarmLow-41c925ac-637d-4668-a08a-3b17e4c91908 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 1. At 2024-11-06T11:32:22Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2024-11-06T11:32:22Z instance i-04027b3fffc2afcd6 was selected for termination. | 2024 November 06, 05:02:22 PM +05:30 |



Instances (1/3) Info

Last updated 3 minutes ago

Connect | Instance state ▼ | Actions ▼

Find Instance by attribute or tag (case-sensitive) | Running ▼

| | Name | Instance ID | Instance state | Instance type | Status check | Alarm status |
|---|---|---|---|---|---|---|
| ☐ | | i-009cdc7e3aa940ef6 | ✓ Running | t2.micro | ✓ 2/2 checks passec | View alarms |
| ☑ | | i-0930e4a8d4aa472fb | ✓ Running | t2.micro | ✓ 2/2 checks passec | View alarms |
| ☐ | | i-0b2782ed579011525 | ✓ Running | t2.micro | Initializing | View alarms |