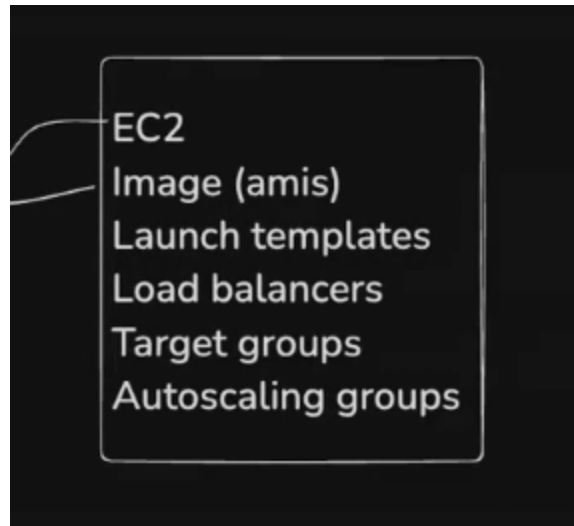


Practical (Steps to create and deploy your application to ASG)

Steps to create and deploy your application to ASG



Step 1: Create an EC2 instance to create an image on it.

- create an EC2 instance manually and clone your code over there.
- install all dependences and pm2.
- run the application using pm2

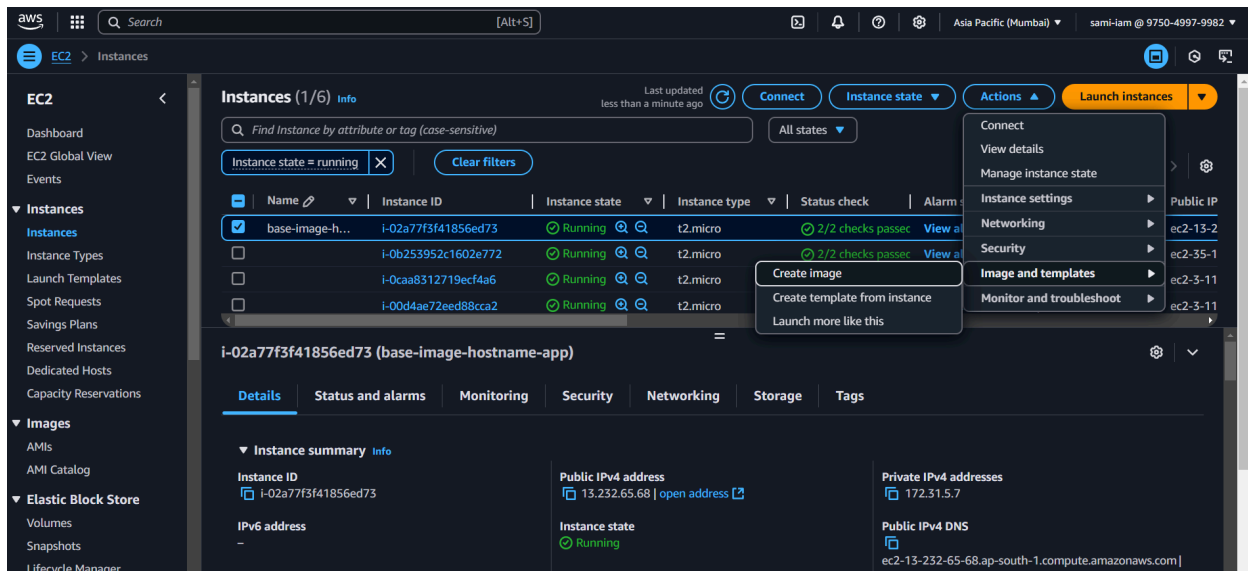
Create an instance which runs your app

- Start an AWS EC2 instance
- SSH into the machine
- Install docker in the machine - <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04>
- or Install node.js on the machine - [How To Install Node.js on Ubuntu | DigitalOcean](#), also install bun `npm install -g bun`
- Clone the repo - <https://github.com/100xdevs-cohort-3/ASG>
- bun install
- bun bin.ts
- Install pm2

```
ubuntu@ip-172-31-35-172:~/ASG$ bun bin.ts
Master 1776 is running
Worker 1785 started
^C
```

```
pm2 start --interpreter /home/ubuntu/.nvm/versions/node/v22.14.0/bin/bun bin.ts
```

Step 2: Create image from the selected instance



Step 3: Create a Launch Template

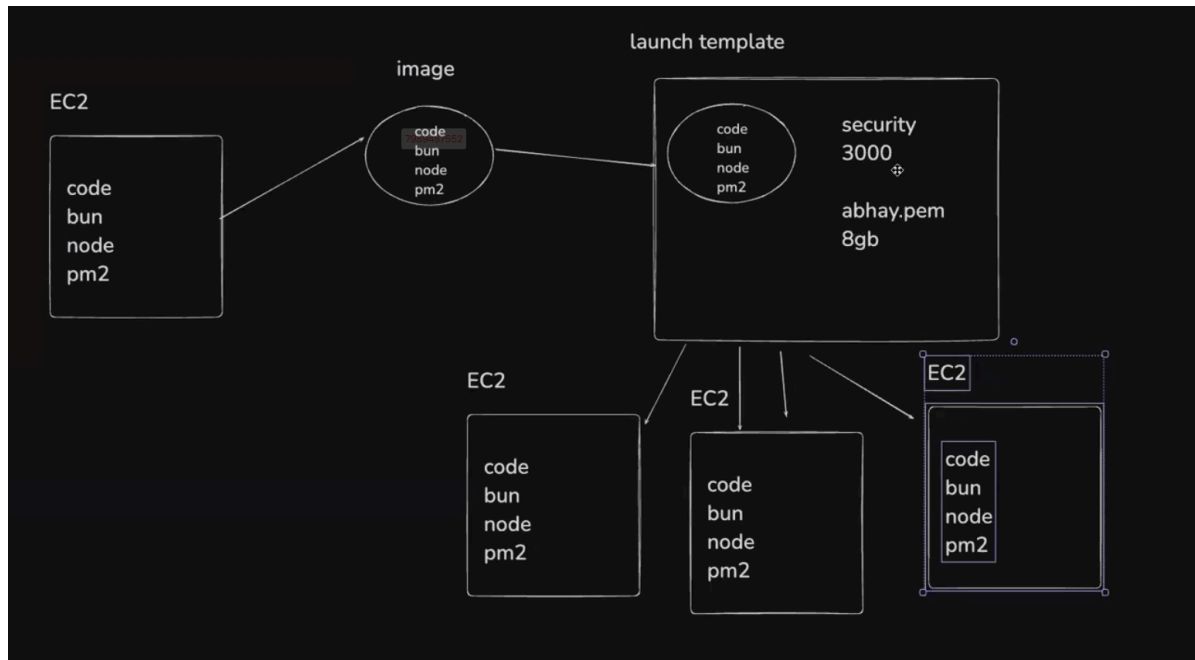
- Give name to the template
- Select the recently create Amazon Machine Image (AMI)
- Select instance type (t2.micro)
- Select a key-pair login (for future debugging purposes)
- Expose port 22 for ssh by creating security group from anywhere.
- Now, we need to expose the application's port (here, 3000) specifically to load balancer.
- But we would allow the inbound traffic to port 3000 from anywhere.
- In advanced details:
 - Add the following code to userdata:
 - It means, after the machine starts with the content from the image you selected, what all do you want running on the machine.

```
#!/bin/bash
cd ~/ASG
export PATH=$PATH:/home/ubuntu/.nvm/versions/node/v22.14.0/bin

# The above export command would add node.js to path
```

```
npm install -g pm2
```

```
pm2 start --interpreter /home/ubuntu/.nvm/versions/node/v22.14.0/bin/bun  
/home/ubuntu/ASG/bin.ts
```



Step 4: Create a Target Group

Step 1: Specify group details
Step 2: Register targets

Specify group details

Your load balancer routes requests to the targets in a target group and performs health checks on the targets.

Basic configuration
Settings in this section can't be changed after the target group is created.

Choose a target type

- ☒ **Instances**
 - Supports load balancing to instances within a specific VPC.
 - Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.
- ☐ **IP addresses**
 - Supports load balancing to VPC and on-premises resources.
 - Facilitates routing to multiple IP addresses and network interfaces on the same instance.
 - Offers flexibility with microservice based architectures, simplifying inter-application communication.
 - Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
- ☐ **Lambda function**
 - Facilitates routing to a single Lambda function.
 - Accessible to Application Load Balancers only.
- ☐ **Application Load Balancer**
 - Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
 - Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.

Target group name
A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Protocol : Port
Choose a protocol for your target group that corresponds to the Load Balancer type that will route traffic to it. Some protocols also include secondary destination for the requests and you can set utilization notified once your target group is...

- Give Target group name
- add the application port (in this case, 3000)
- add health check. This is really important in ASGs since the ASG would hit this health check endpoint to determine if the EC2 instance is up and running properly or not.
 - if the ASG receives a 200 status code, it means the EC2 server is running fine. else, it would terminate that EC2 instance and bring up another.

```
index.ts > ...
import express from "express";
import os from "os";

export const app = express();

app.get("/healthchecks", (req, res) => {
  res.send("Hello World");
});

app.get("/cpu", (req, res) => {
  for (let i = 0; i < 1000000000; i++) {
    Math.random();
  }
  res.send("Hello World");
});

app.get("/host", (req, res) => {
  res.send(os.hostname());
});
```

While registering targets on the next step, you should not manually select any machine since it will be managed by ASG.

- Now create the target group.

Step 5: Create a ASG

- Give your ASG a name
- select the recently created launch template
- In networks section, select as many availability zones as possible (all 3).

The screenshot shows the 'Create Auto Scaling group' wizard in the AWS Management Console, specifically the 'Review' step. The 'Network' section is expanded, showing the following configuration:

- Launch template:** hostname-app-launch-template (lt-0dc2c4bccc3be1c55)
- Instance type:** t2.small
- VPC:** vpc-0f95a0b9b6d0062 (172.31.0.0/16)
- Availability Zones and subnets:**
 - ap-south-1a | subnet-027d18538448c6293 (172.31.32.0/20)
 - ap-south-1b | subnet-0344f06dc909a243f (172.31.0.0/20)
- Availability Zone distribution - new:**
 - ☒ **Balanced best effort**: If launches fail in one Availability Zone, Auto Scaling will attempt to launch in another healthy Availability Zone.
 - ☐ **Balanced only**: If launches fail in one Availability Zone, Auto Scaling will continue to attempt to launch in the unhealthy Availability Zone to preserve balanced distribution.

Navigation buttons at the bottom: Cancel, Skip to review, Previous, Next.

- Now, in the **Integrate with other services**, Attach a new load balancer.
- Select Application Load Balance
- Give your load balancer a name
- the load balancer scheme is **internet-facing**
- the load balancer should listen on port 80 for http.

In **Configure group size and scaling** section, select desired, minimum and maximum instances.

EC2 > Auto Scaling groups > Create Auto Scaling group

Step 1: Choose launch template
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

Configure group size and scaling - optional

Define your group's desired capacity and scaling limits. You can optionally add automatic scaling to adjust the size of your group.

Group size
Set the initial size of the Auto Scaling group. After creating the group, you can change its size to meet demand, either manually or by using automatic scaling.

Desired capacity type
Choose the unit of measurement for the desired capacity value. vCPUs and Memory(GiB) are only supported for mixed instances groups configured with a set of instance attributes.
Units (number of instances)

Desired capacity
Specify your group size.
2

Scaling
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
5
Equal or greater than desired capacity

Automatic scaling - optional
Choose whether to use a target tracking policy.

☒ 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.

Instance maintenance policy
Control your Auto Scaling group's availability during instance replacement events. This includes health checks, instance refreshes, maximum instance lifetime features and events that happen

Now, finally, create the auto scaling group.

Add security groups to the load balancer

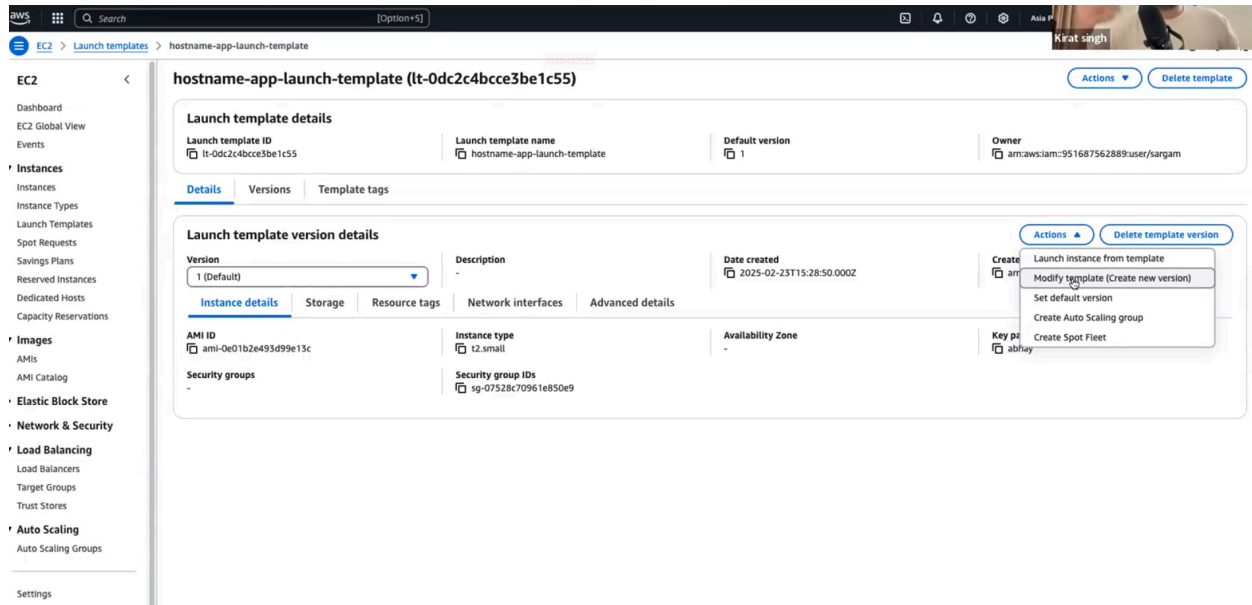
- open port 80 (HTTP) on the load balancer.

Debugging:

If the image does not run on your newly created EC2 instances via ASG, you can ssh into anyone of the EC2 instances spawned via ASG and check the logs by running the following command.

```
cat /var/log/cloud-init.log
cat /var/log/cloud-init-output.log
```

If there was an error in userdata in the launch template or you want to change security group or the instance type, you can edit it by modifying template (create new version)



- Now your template has 2 versions.
- You have to go to ASG, edit the version of the launch template to the latest version (version 2).

After Changing the version of launch template

- Now, you need to terminate all the instances spawned via ASG and start the new instances.
- you can do that by simply editing the ASG's desired, minimum and maximum instances values to 0.
- After all the EC2 instances are terminated, change the values of desired, minimum and maximum instances back to normal which will spawn new EC2 instances via ASG.

Debugging tip for Load balancer

- If you get **503 error page** when you hit the load balancer URL, make sure your ASG is properly connected to the load balancer and your load balancer is connected to the target group.
- add security groups to the load balancer to open port 80 (HTTP) on the load balancer.

setup Automatic scaling

- Go to ASG and go to Automatic scaling tab
- You can setup auto scaling based on CPU usage.
 - if the CPU usage hits an average of 50% combining all the running instances on average, then a new EC2 instance will be spawned to distribute the load and bring down the CPU utilization below 50% threshold.
 - You can also setup auto scaling based on no. of incoming requests

The screenshot shows the AWS Management Console interface for creating a dynamic scaling policy. The breadcrumb navigation at the top indicates the path: EC2 > Auto Scaling groups > hostname-application-prod > Dynamic scaling policy. The left-hand navigation pane lists various AWS services, with 'Auto Scaling' expanded. The main content area is titled 'Create dynamic scaling policy' and contains the following fields:

- Policy type:** A dropdown menu set to 'Target tracking scaling'.
- Scaling policy name:** A text input field containing 'Target Tracking Policy'.
- Metric type:** A dropdown menu set to 'Average CPU utilization'. Below this, a small note states: 'Monitored metric that determines if resource utilization is too low or high. If using EC2 metrics, consider enabling detailed monitoring for better scaling performance.'
- Target value:** A text input field containing '50'.
- Instance warmup:** A text input field containing '300' followed by the unit 'seconds'.
- Disable scale in to create only a scale-out policy:** An unchecked checkbox.

Test ASG

- you can test your asg by hitting the resource-intensive endpoint (in our case, it is the /cpu route which run a long loop till some big number 1000000000).
- If you run multiple node.js processes from your local machine and hit this endpoint multiple times, you can see the desired instances (the no. of instances currently running) would increase.