Project Report

by Group –18

Ananya Reddivari.

Harika Kosuru.

Monica Gattupalli.

Sri Sai Ganesh Satyadeva Naidu Totakura.

Syam Kumar Vemana.

Problem statement and Problem Description

This project is to deploy and implement a scalable Django application (one-tier architecture), deploying it in Amazon Web Services (AWS) in Ubuntu free tier version, also implemented auto scaling for the application in case of increased load. To measure the load, average CPU utilization is used. This auto scaled instances satisfies the requirements like scalability with respect to computation and high availability of the computation.

Design Description

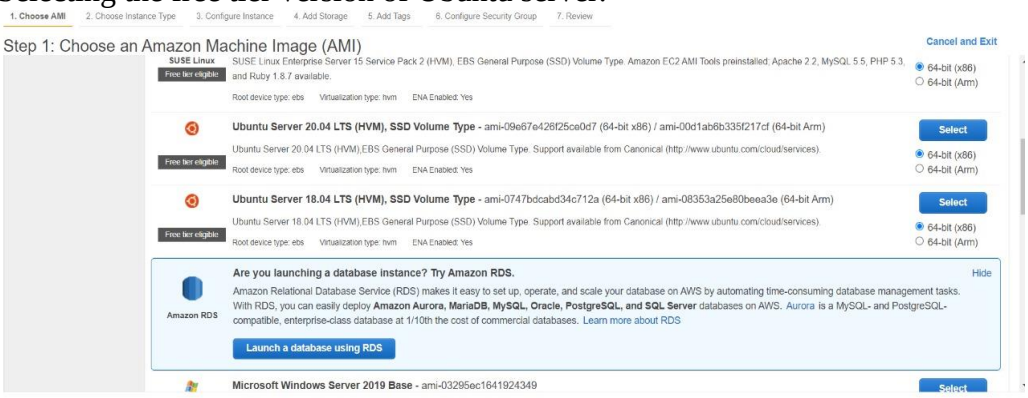
We have achieved the problem statement in 6 steps: -

1. Creation of **EC2** instance in AWS.
2. Deploying the scalable Django application into the created instance.
3. Creation of Amazon Machine Image (AMI) for the instance created in step 1.
4. Creation of launch template for the instance created in step 1.
5. Creation of target group for the instance created in step 1.
6. Configuring the auto scaling group for the instance created in step 1.

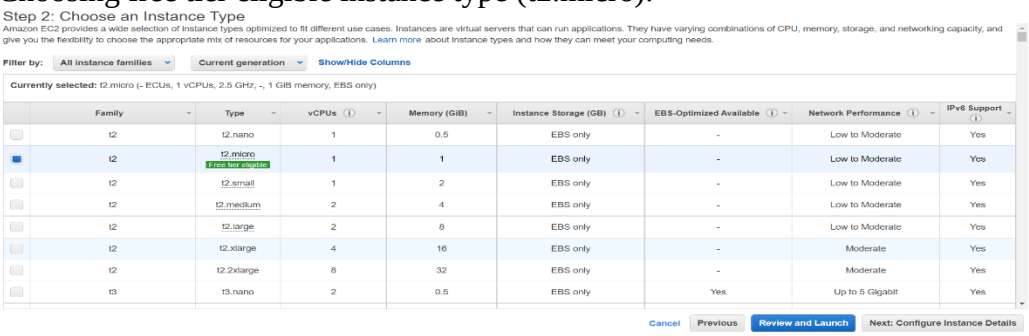
Implementation

1. **Creation of EC2 instance in AWS.**

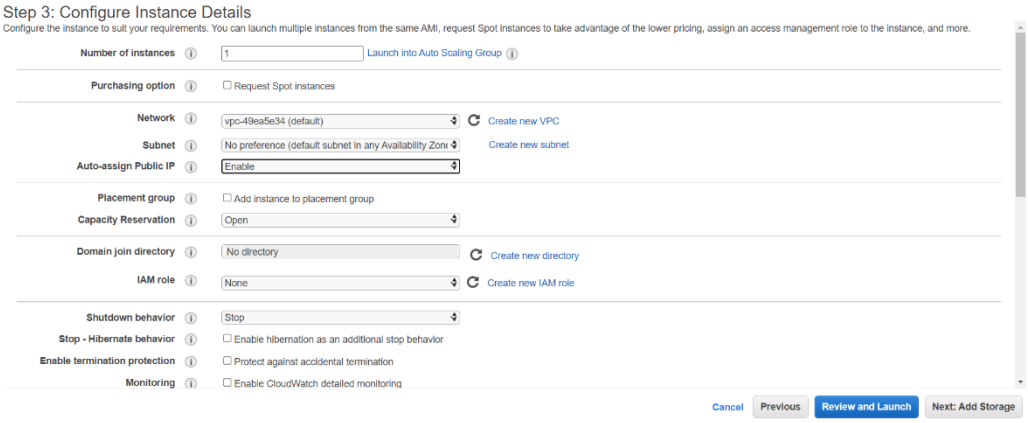
   - Selecting the free tier version of Ubuntu server.



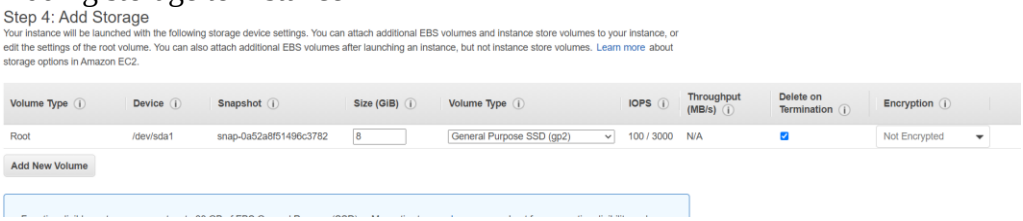
   - Choosing free tier eligible instance type (t2.micro).



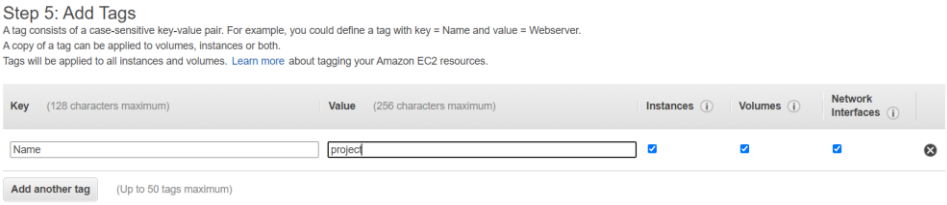
   - Configuring the instance details



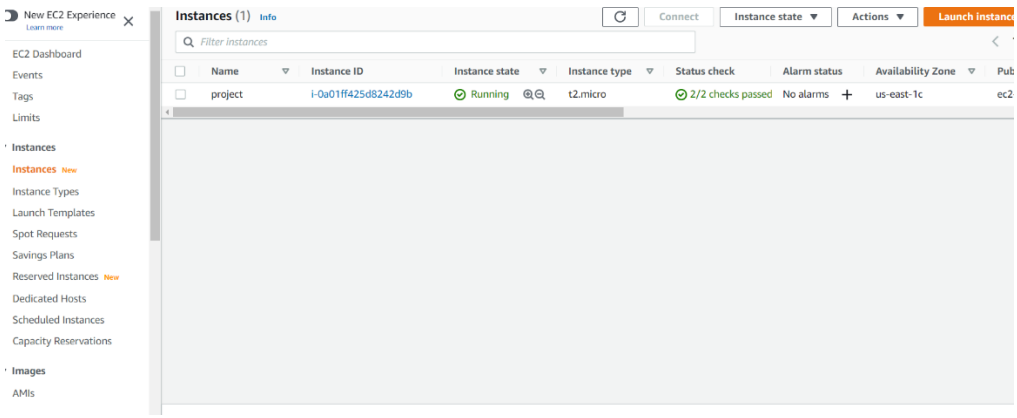
- Adding storage to instance



- Adding tags to instance



- Successful creation of Ubuntu instance



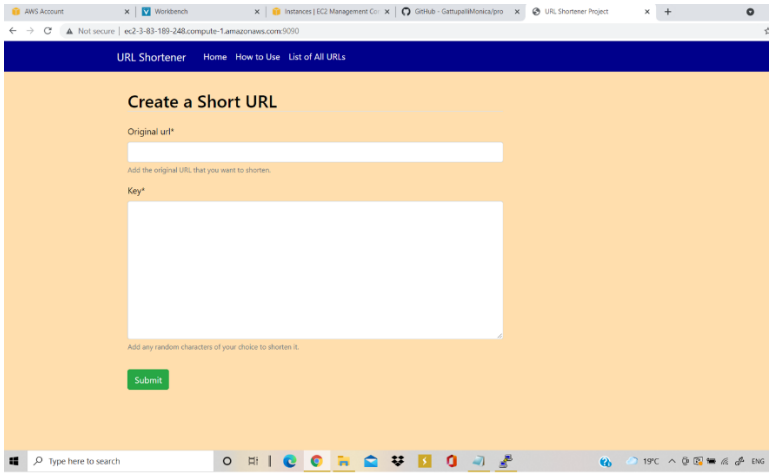
2. Deploying the scalable Django application into the created instance.

URL shortener using Django is selected as the scalable Django application for this project. The chosen application is used for customizing the URL's given by the user, and it simply replaces the URL with the key provided by the users. The List of all URLs which are customized is stored in the application. This URL shortener is cloned from our GitHub repository – **https://github.com/GattupalliMonica/pro**

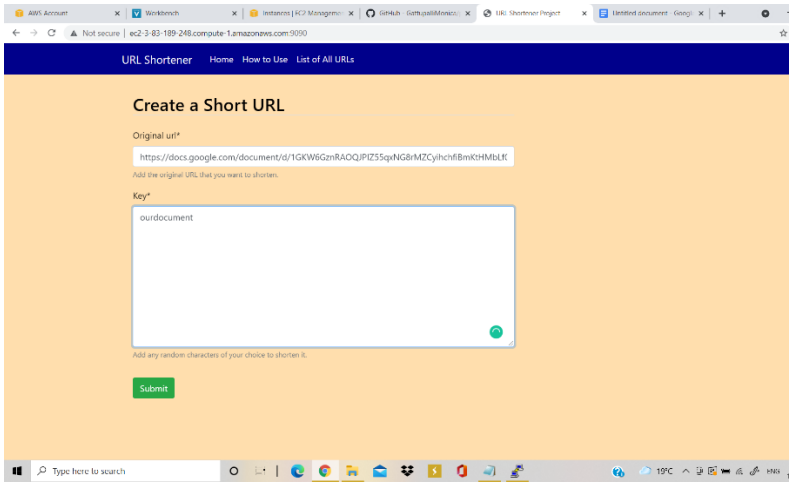
Commands for deploying the Django application into EC2 instance are as follows:

- **sudo apt update –** Updating the Ubuntu server to restore the new packages.
- **sudo apt install nginx** – To install nginx web server for faster performance.
- **cd /etc/nginx** – Changing the directory to nginx.
- **sudo systemctl start nginx –** Used to start the nginx web server.
- **sudo apt install python3-pip** – Used to install the python3 in the instance.
- **cd –** Changing the directory from nginx.
- **sudo apt install python3-virtualenv –** Installing a virtual environment with corresponding to python3.
- **ls –** Displays list of files in virtual environments.
- **mkdir project –** Creating a new directory with name "project".
- **cd project –** Changing the directory to "project" directory.
- **git clone https://github.com/GattupalliMonica/pro** - Cloning the git repository to access the application files.
- **virtualenv venv** – Creating a virtual environment with name "venv".
- **ls –** Displays list of files in virtual environments.
- **source venv/bin/activate** – Activating the virtual environment from the source.
- **pip install -r pro/requirements.txt** – Installing the required modules for the URL shortener application from "requirements.txt" file
- **pip install gunicorn** – Installing the gunicorn which is used to connect the Django application with nginx web server.
- **cd URL_Shortener_Project_using_Django –** Changing the directory to our main project directory.
- **gunicorn --bind 0.0.0.0:9090 URL_Shortener_Project_using_Django.wsgi** – Accessing the application from 9090 port number through "wsgi" interface.

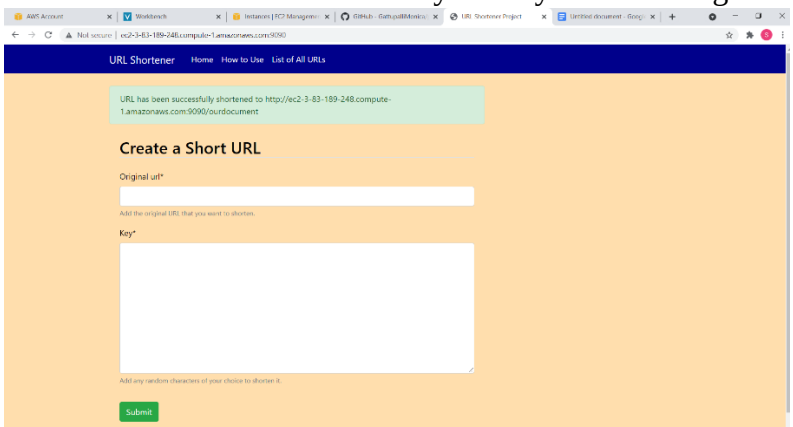
Accessing the URL shortener application using Django from the public DNS of the Ubuntu instance.



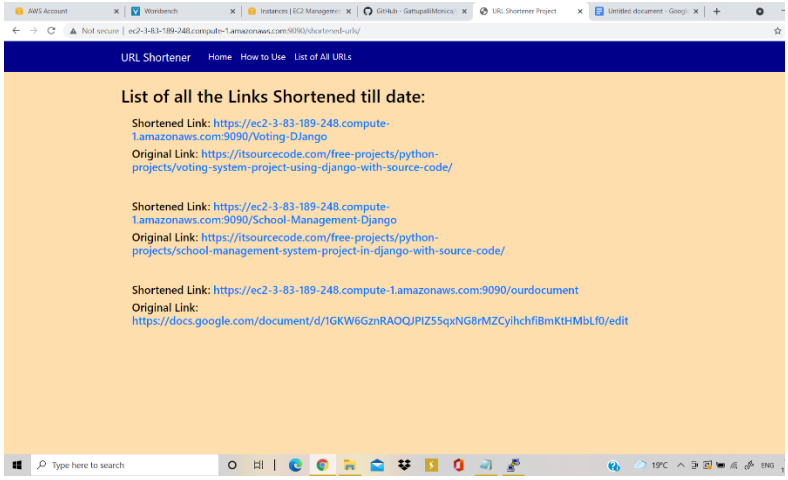
A rondom URl and key is entered by the user to check the application.



The entered URL is customised by the key after clicking submit button.



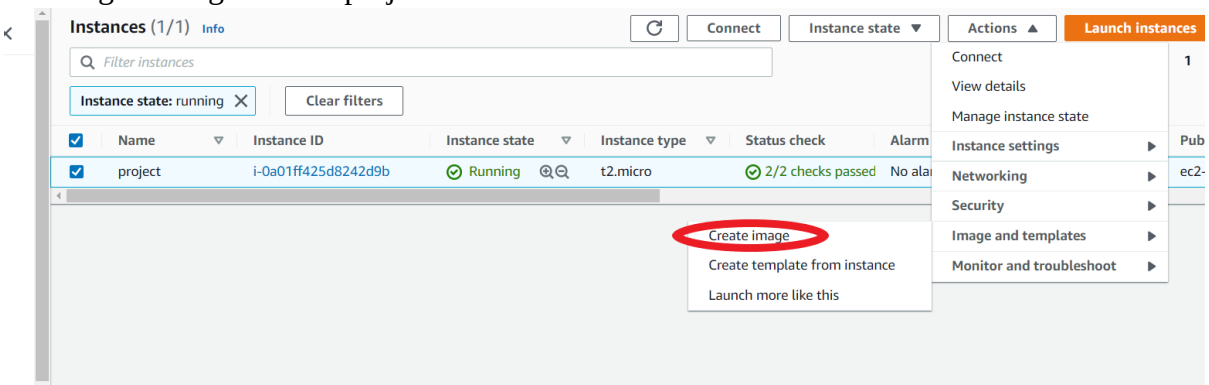
List of all customised URLs



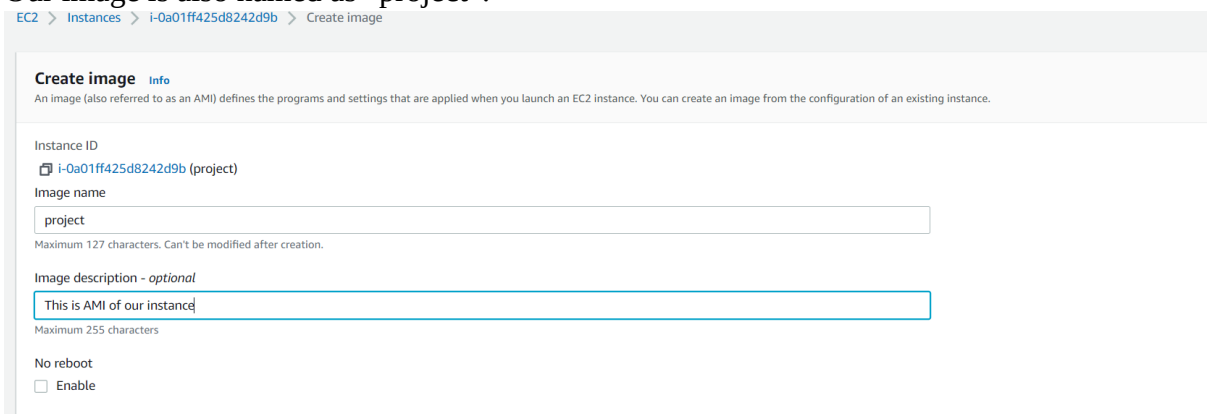
3. Creation of Amazon Machine Image (AMI)

AMI provides required information to launch an instance and used to deploy multiple instances with same configuration.

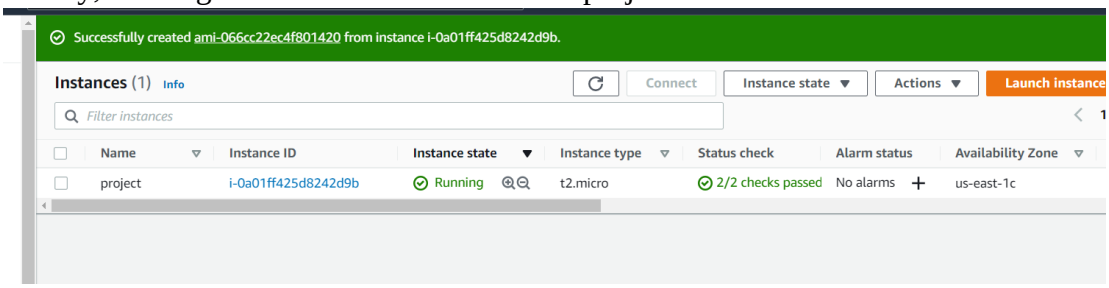
Creating an image to our "project" instance.



Our image is also named as "project".



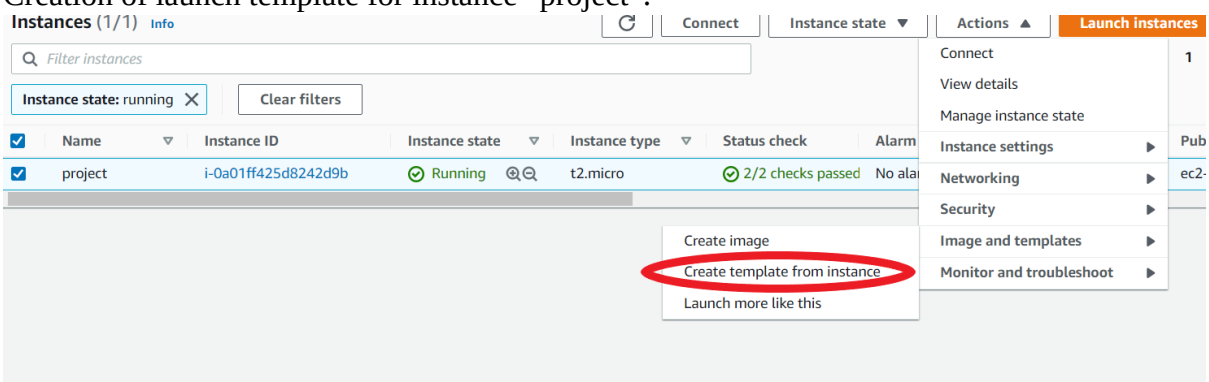
Finally, an image is created for the instance “project”



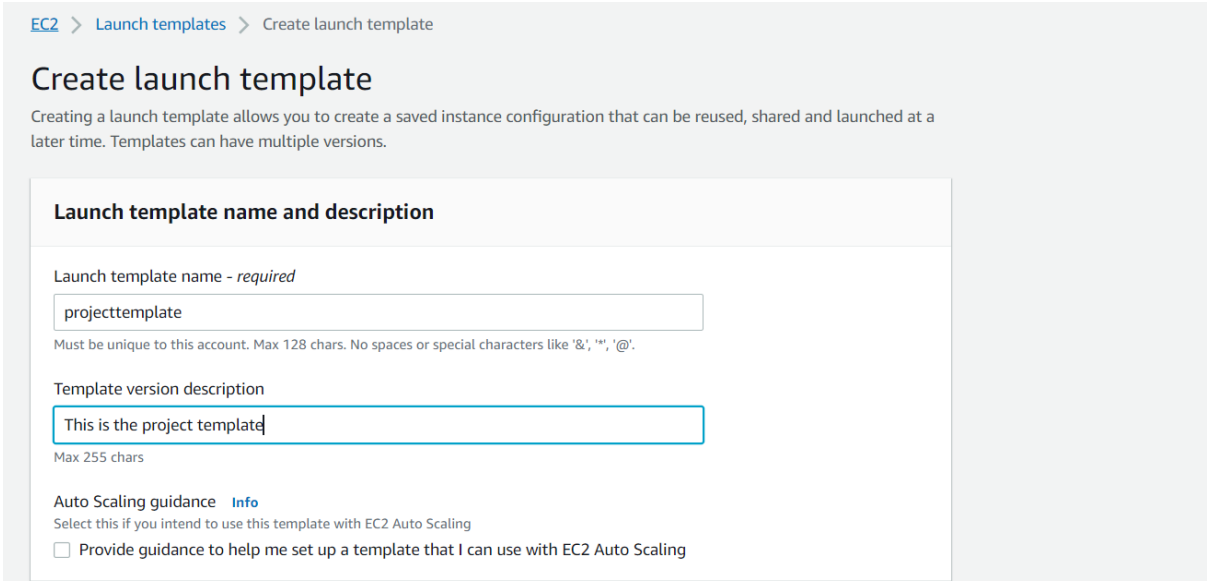
4. Creation of launch template

Launch template specifies the instance configuration details and enables us to use the auto scaling groups.

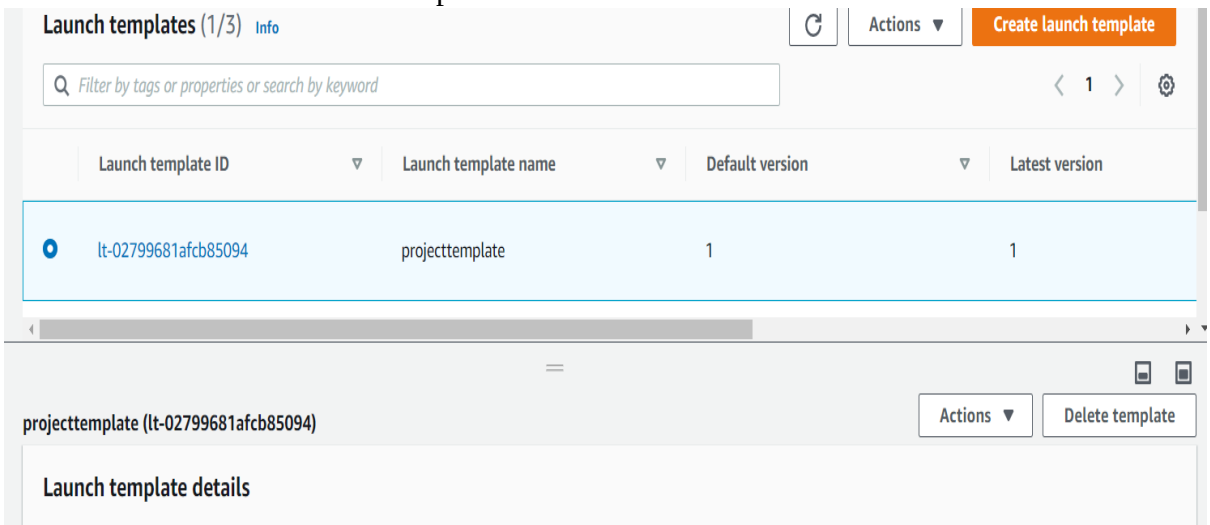
Creation of launch template for instance “project”.



Configuring the launch template with template name and the template version description.



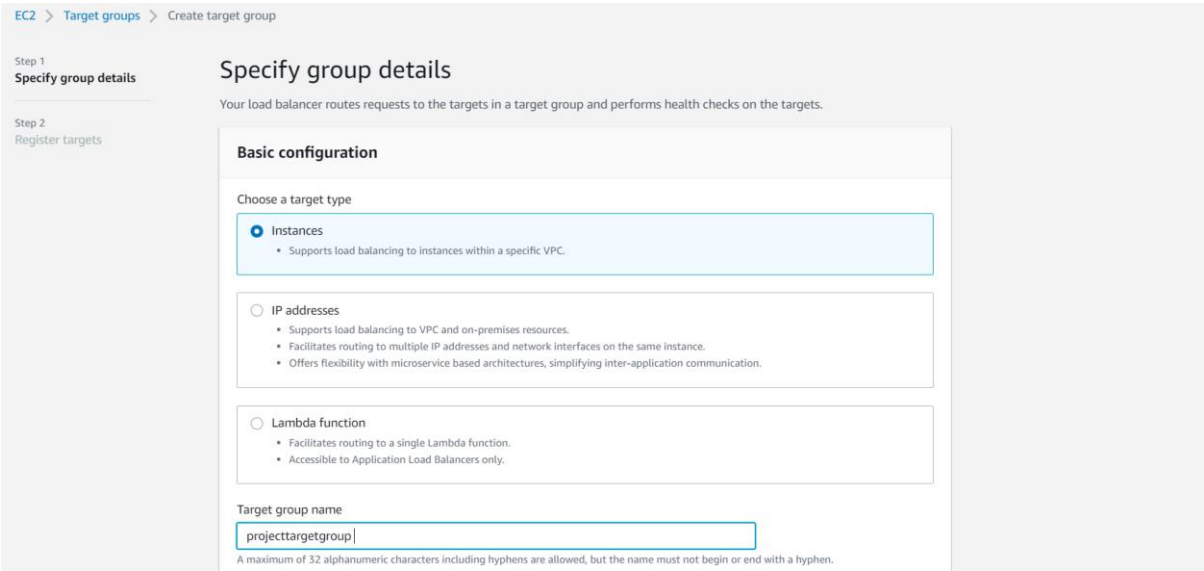
Successful creation of launch template.



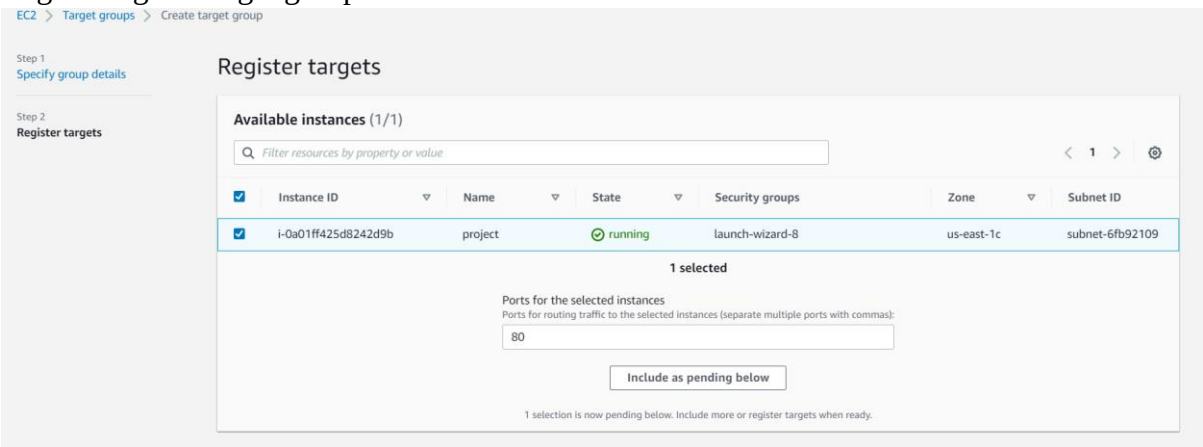
5. **Creation of target group.**

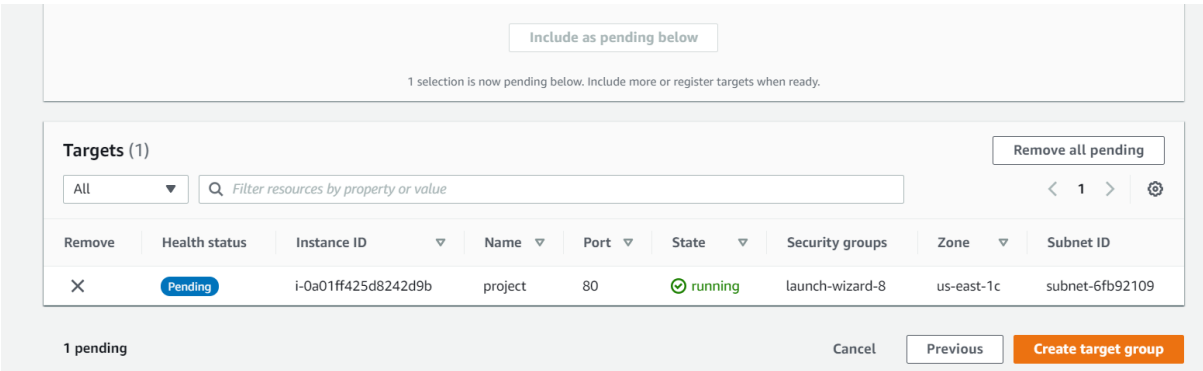
Target group is used to control the network topic under the specified protocols and ports to specified instance. This ensures the "high availability of computation" by avoiding the network traffic of the specified instance. This target group is attached to the auto scaling group to manage the increase and decrease of instances due to load variations.

Creation of target group by selecting the instances as the basic configuration and the name of the target group is "projecttargetgroup".

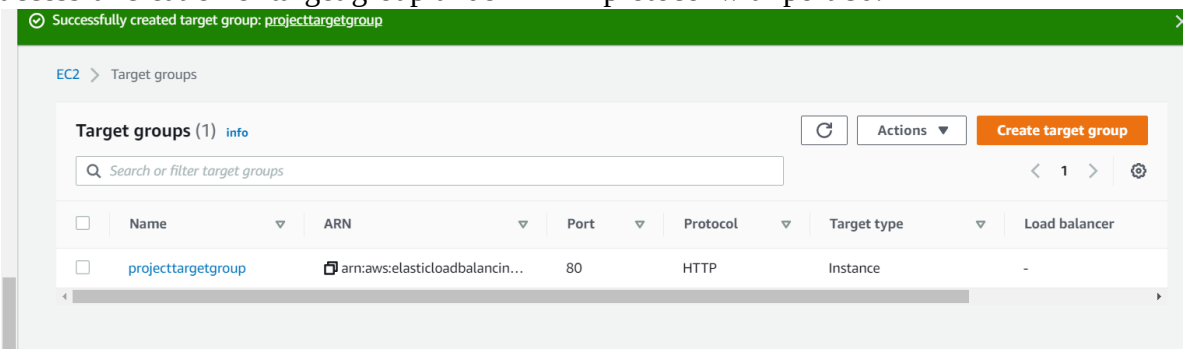


Registering the target group





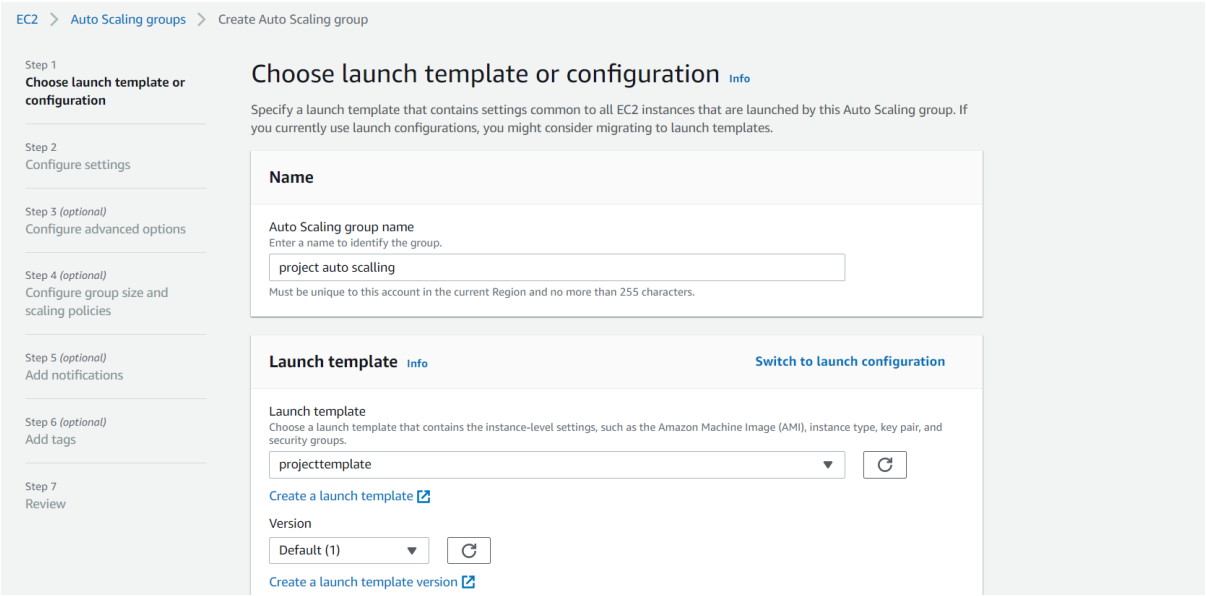
Successful creation of target group under HTTP protocol with port 80.



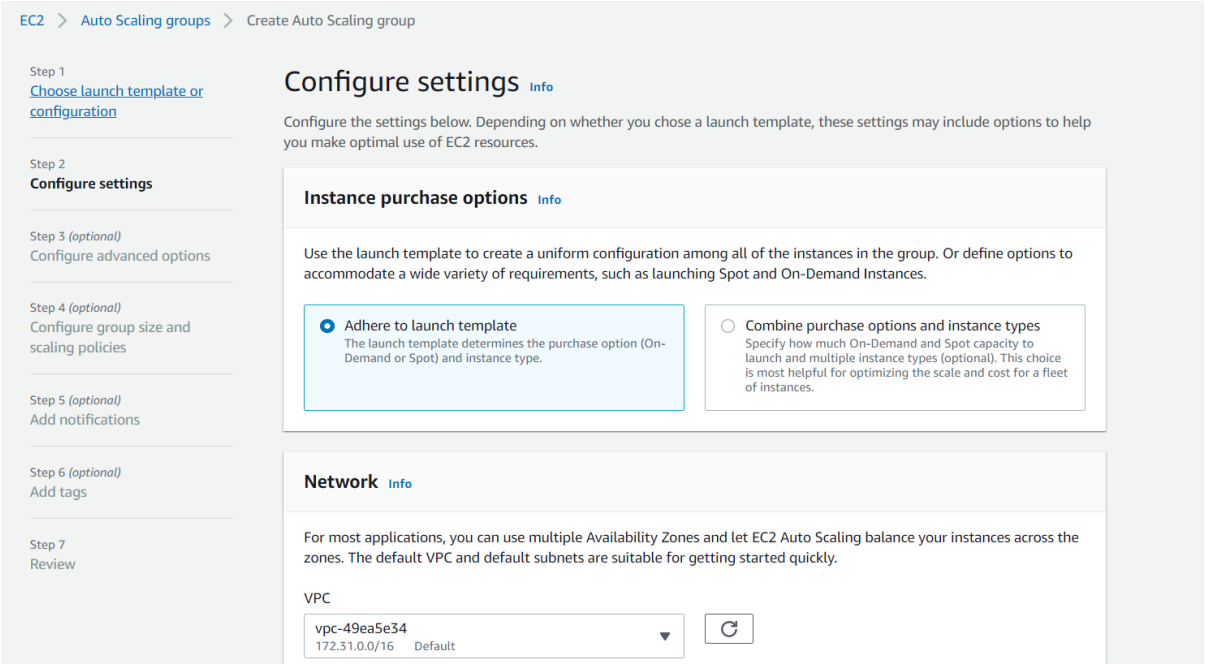
6. **Configuring the auto scaling group.**

Auto scaling is a technique for dynamically adding or removing the computational resources under the measure of average CPU utilisation or network in-bytes and out-bytes. In this project average CPU utilisation is taken as a scale to measure the load on the instance and the auto scaling is done.

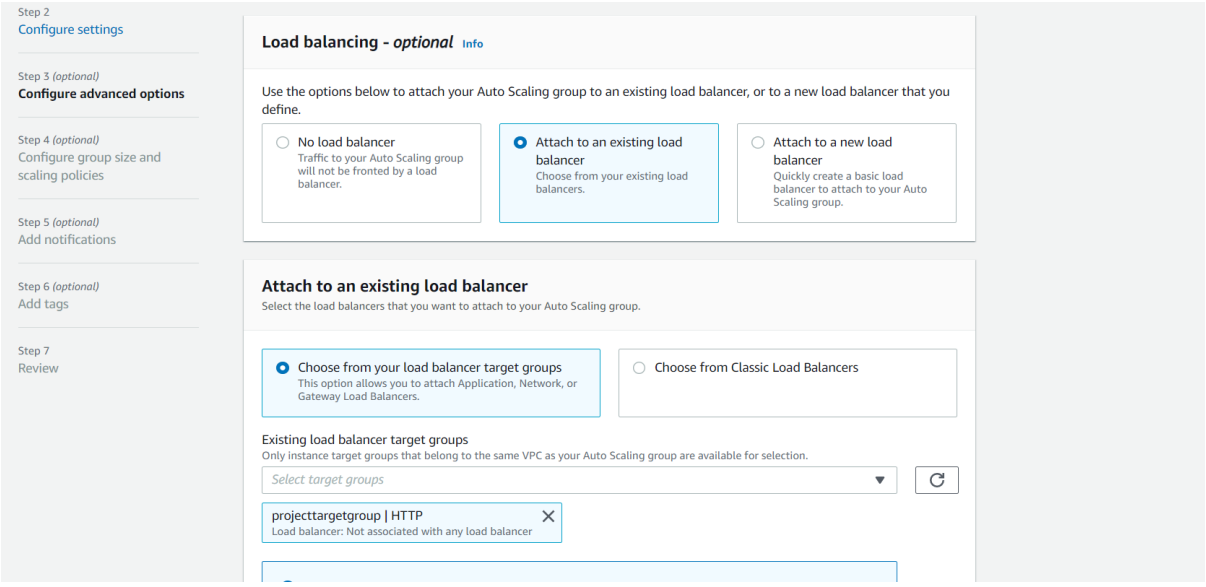
Choosing the launch template for creating the auto scaling group.



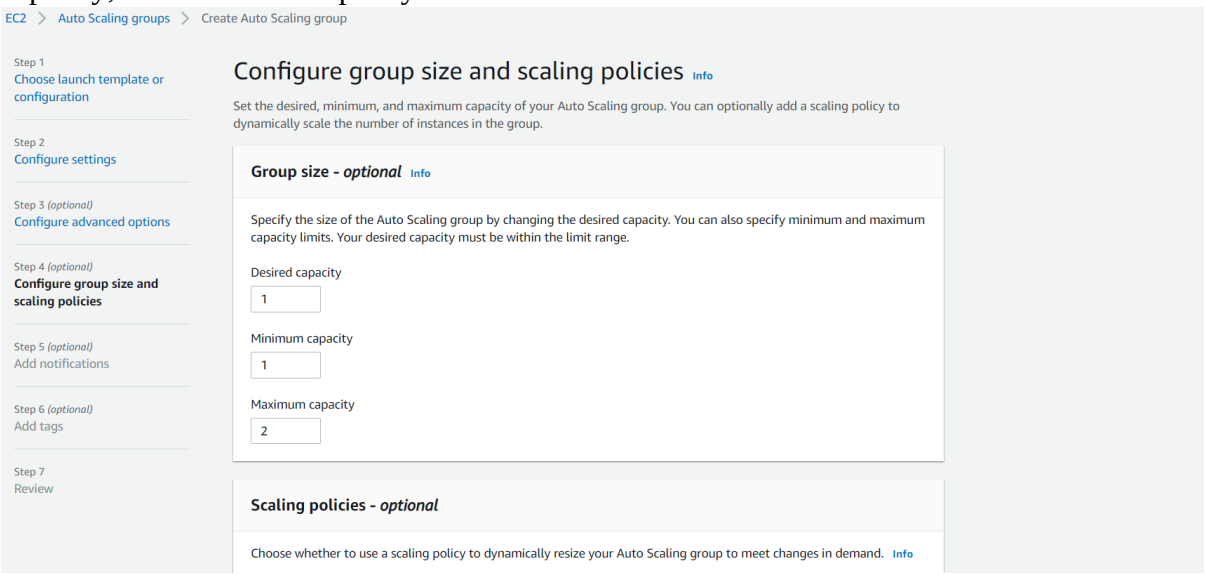
Adhere to launch template is used as configuration setting for the auto scaling group.



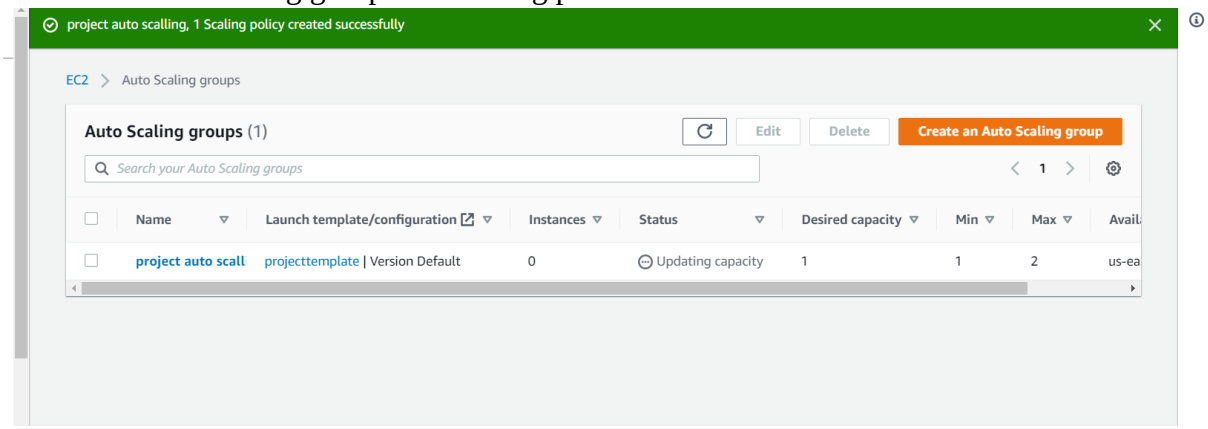
Attached to an existing load balancer by selecting the target groups of the instance “project”.



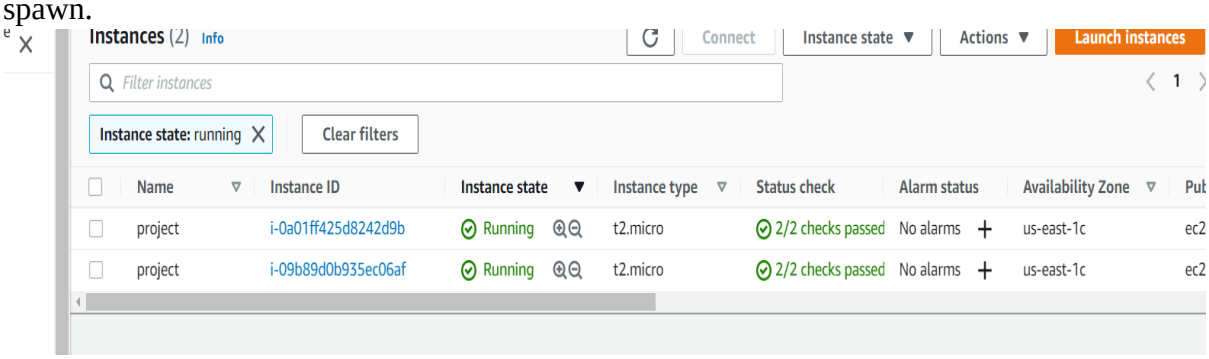
Configuring the group size and scaling policies like desired capacity, maximum capacity, and minimum capacity.



Successful auto scaling group with scaling policies.

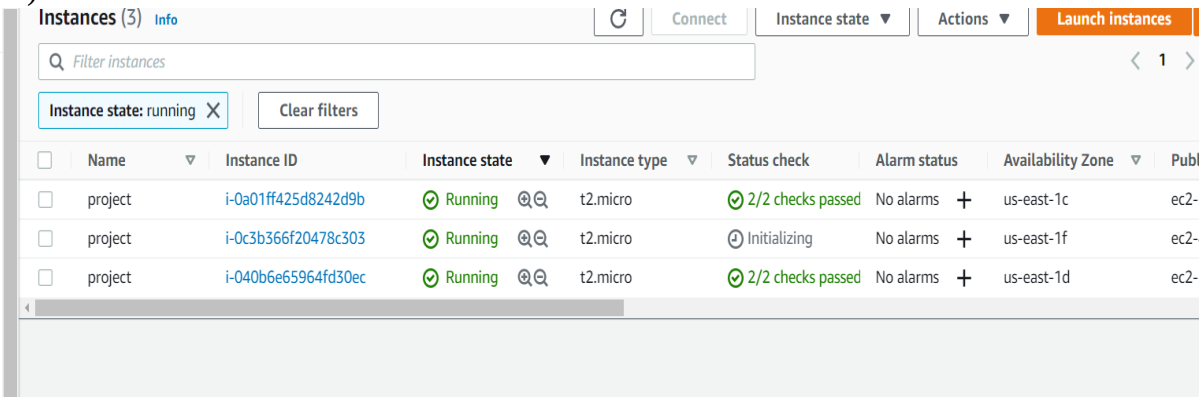


Now auto scaling group is attached to the instance “project”, as we have given desired capacity as 1 after attaching scaling group an instance with similar configuration is spawn.

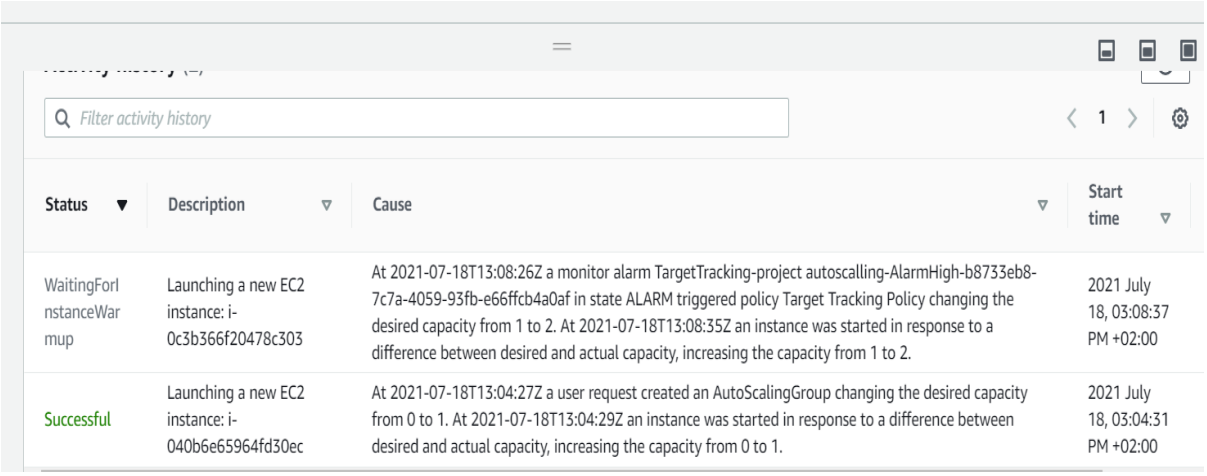


For our Django application multiple requests are taken from n number of users, this increases load on CPU. Another instance is created when the load threshold value is greater than 35.

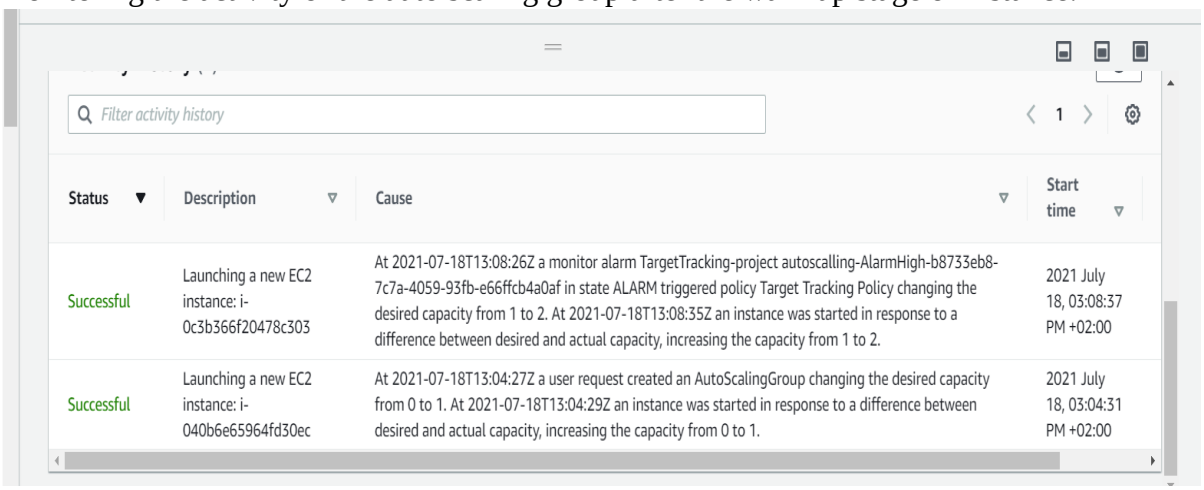
As the maximum capacity of the auto scaling group is 2, another resource is added to the instance project by satisfying the auto scaling policies (average CPU utilization > 35)



Monitoring the activity of the auto scaling group in warm up stage of instance.



Monitoring the activity of the auto scaling group after the warmup stage of instance.

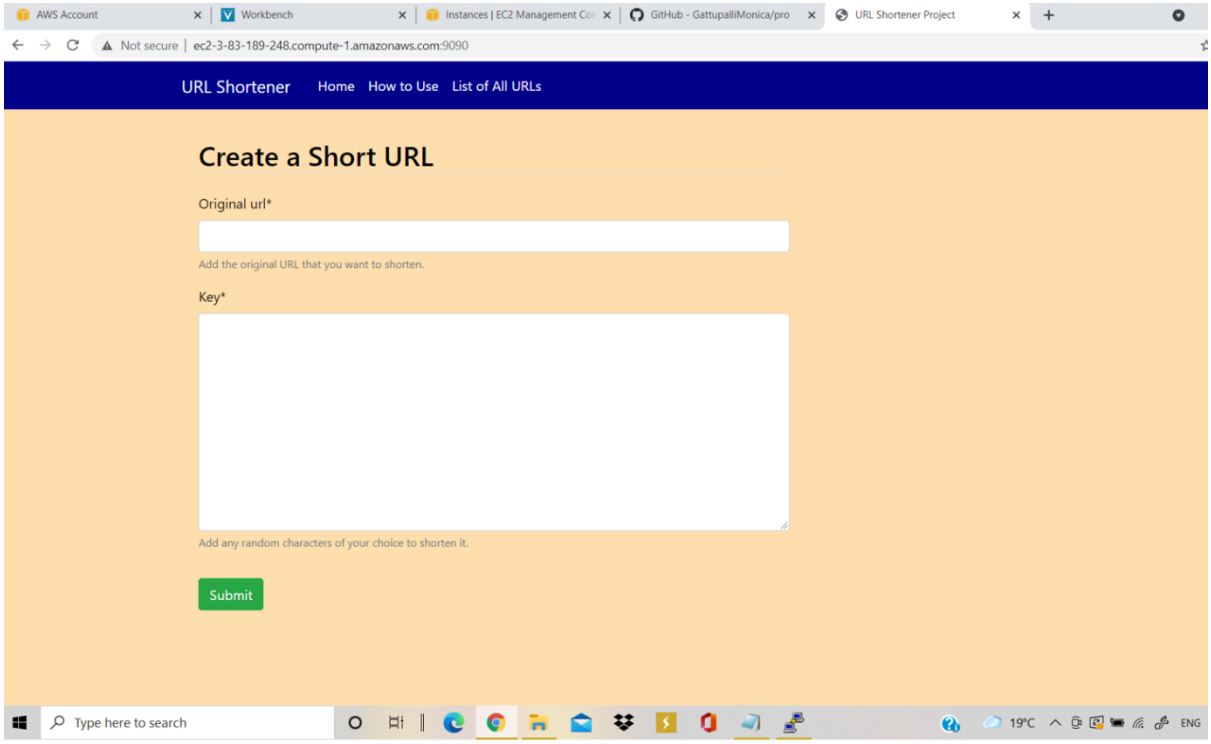


Filter activity history

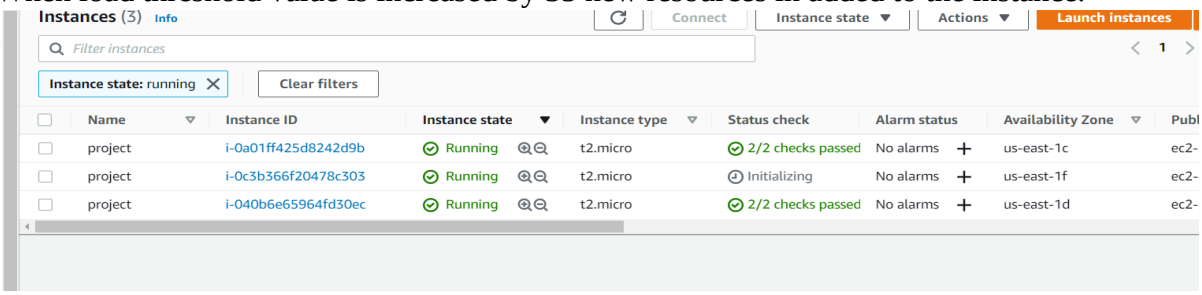
Status	Description	Cause	Start time
Successful	Launching a new EC2 instance: i-0c3b366f20478c303	At 2021-07-18T13:08:26Z a monitor alarm TargetTracking-project autoscalling-AlarmHigh-b8733eb8-7c7a-4059-93fb-e66ffcb4a0af in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 1 to 2. At 2021-07-18T13:08:35Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2021 July 18, 03:08:37 PM +02:00
Successful	Launching a new EC2 instance: i-040b6e65964fd30ec	At 2021-07-18T13:04:27Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 1. At 2021-07-18T13:04:29Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1.	2021 July 18, 03:04:31 PM +02:00

Validation

- Deploying the URL shortener project into the instance, executed it using public DNS of the instance.



- When load threshold value is increased by 35 new resources in added to the instance.



Instances (3) Info

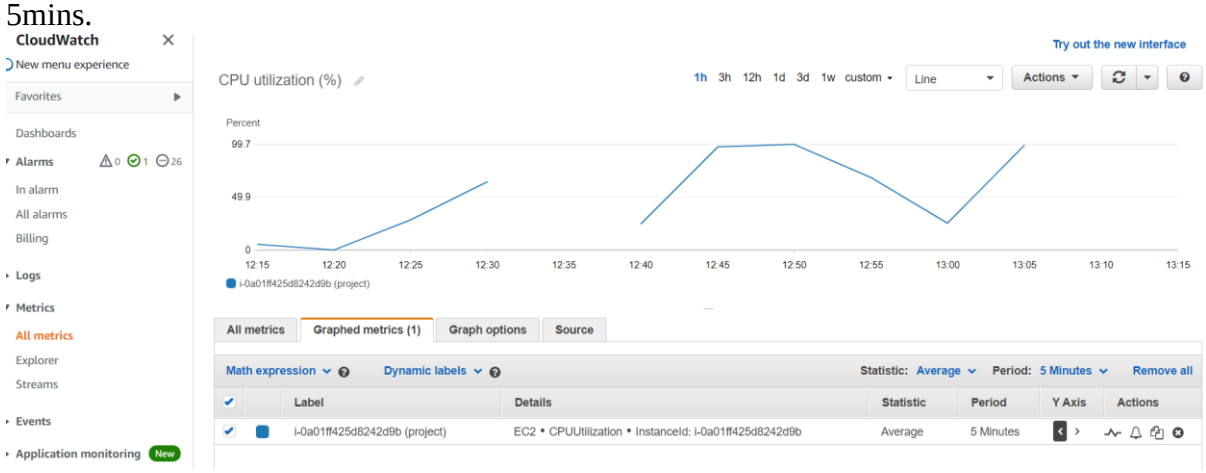
Connect | Instance state | Actions | Launch instances

Filter instances

Instance state: running | Clear filters

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Publ
project	i-0a01ff425d8242d9b	Running	t2.micro	2/2 checks passed	No alarms	us-east-1c	ec2-
project	i-0c3b366f20478c303	Running	t2.micro	Initializing	No alarms	us-east-1f	ec2-
project	i-040b6e65964fd30ec	Running	t2.micro	2/2 checks passed	No alarms	us-east-1d	ec2-

- Monitoring the CPU utilisation through the cloud watch within the warmup period of 5mins.



Results

- **Scalability with respect to computation**
  It is a property of a system to handle the amount of work by adding or removing resource to the system.
  In this project scalability with respect to computation is achieved through auto scaling where a resource is added to the instance when threshold value is greater than 35.
- **High availability of computation**
  High availability means access to the data, services, and tools whenever the user requires them.
  In this project high availability of computation is achieved with two different features such as EC2 instance and the target group. Where the EC2 instance holds the Django application and make it available to the users. The target group avoids the network traffic, increases the ease of access to application and the users.
- **One tier architecture**
  All the functions and the elements of the application like storage, database and server should be accessed from a single system.
  In this project we achieved one tier architecture by cloning the application from Git repository.
- Thus, Django application selected for this project satisfies scalability with respect to computation, high availability of computation and one tier architecture.