

Problem Description

The project is to deploy and implement a Django application on the Amazon Web Services and the application should be auto scaled in such a way that it satisfies the requirements scalability with respect to computation and the high availability of the computation.

Design Description

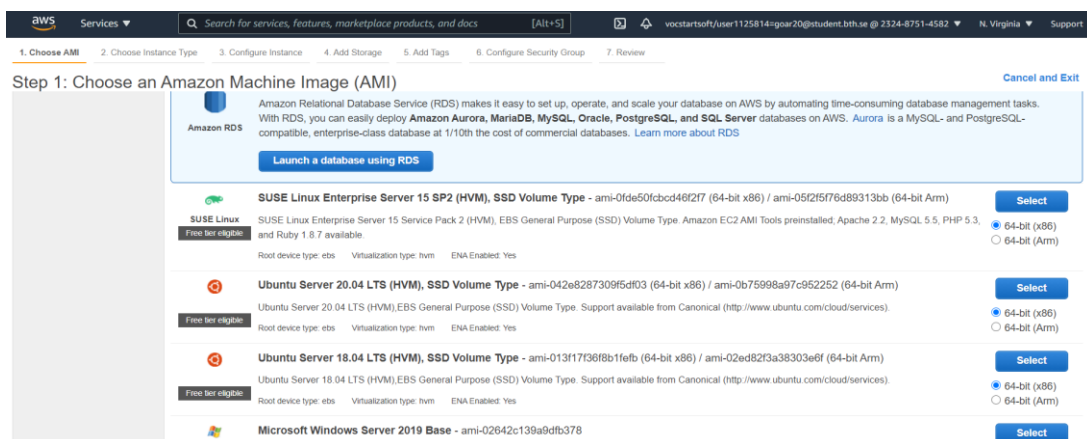
We have selected static django application from GitHub to deploy in the Amazon Web Services. For that, an EC2 instance of ubuntu AMI is launched and connected to the server. To deploy the application in ubuntu server, Nginx Web server and Gunicorn WSGI are configured. After deploying the Django application, it is auto scaled by adding the scaling policies that configures the metric average CPU utilization of the application and takes the action satisfying scaling policies.

Implementation

To deploy an application, at first an Ec2 instance should be launched on the Amazon web services.

To launch an Ec2 instance

1. Select the Ec2 services on Aws console and then launch an instance.
2. Select ubuntu server image from the Amazon Machine Images (AMI).



3. Select the default VPC and subnets.

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances [Launch into Auto Scaling Group](#)

Purchasing option ☐ Request Spot instances

Network [Create new VPC](#)

Subnet [Create new subnet](#)
249 IP Addresses available

Auto-assign Public IP

Placement group ☐ Add instance to placement group

Capacity Reservation

Domain join directory [Create new directory](#)

IAM role [Create new IAM role](#)

CPU options ☐ Specify CPU options

Shutdown behavior

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Add Storage](#)

- Set a security group for the instance with proper inbound rules.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more about Amazon EC2 security groups.](#)

Assign a security group: ☐ Create a new security group
☒ Select an existing security group

Security Group ID	Name	Description	Actions
sg-03fd82a51359820ec	abdef	launch-wizard-2 created 2021-02-23T10:18:06.426+01:00	Copy to new
sg-4b0edc7f	default	default VPC security group	Copy to new
sg-06cd43e55d69e846f	project sg	launch-wizard-2 created 2021-02-20T14:34:11.384+01:00	Copy to new
sg-0cb1eb5a4323750aa	secg	AutoScaling-Security-Group-1 (2021-02-19T11:03:40.123Z)	Copy to new
sg-08f0e79433871fefa	security_group_proj	AutoScaling-Security-Group-1 (2021-02-21T17:44:11.043Z)	Copy to new

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
HTTP	TCP	80	:::0	
SSH	TCP	22	0.0.0.0/0	
SSH	TCP	22	:::0	
Custom TCP Rule	TCP	0	0.0.0.0/0	

[Cancel](#) [Previous](#) [Review and Launch](#)

- Create a key pair and download the pem file.

Go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

Your security group ID is **sg-06cd43e55d69e846f**. You can add additional IP addresses to your security group.

SSD Volume
General Purpose SSD (gp2) 16 GB

1

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. [Learn more about removing existing key pairs from a public AMI.](#)

Select a key pair

☒ I acknowledge that I have access to the selected private key file (keypair16.pem), and that without this file, I won't be able to log into my instance.

[Cancel](#) [Launch Instances](#)

- Connect the EC2 instance to the server using putty.

Deploying the Django application on the Amazon Web Services Ec2 instance

Django provides the sever that facilitates to run the Django applications on the local host.one of the most likely used services to host Django applications is Amazon services.

Certain set of applications are required to host Django application.

1. Firstly, we need to update our server.

```
sudo apt-get update  
sudo apt-get upgrade
```

2. Make sure that python is installed on the connected server.
3. Create a virtual environment for the application.

```
sudo apt-get install python3-venv  
python3 -m venv env
```

4. Activate the created environment.

```
source env/bin/activate
```

5. Install Django on the sever using pip.

```
pip3 install django
```

6. Pulling our django application from the GitHub.

As our django application is static which is used for the deployment based, we prepared our django application ready for production and pushed it into the GitHub.

Our Django application in GitHub: https://github.com/gowthamsandaka/django_test.git

To clone the django application from GitHub:

```
git clone https://github.com/gowthamsandaka/django\_test.git
```

Configuration of Nginx and Gunicorn

1. For the faster performance we will use Nginx web server.
To install Nginx

```
sudo apt-get install -y nginx
```

2. We will use Gunicorn to link the django application and Nginx web server. For that install gunicorn using pip.

```
Pip3 install gunicorn
```

3. To configure gunicorn and Nginx and run the application always on background, supervisor should be installed.

```
sudo apt-get install supervisor
```

4. Create the configuration for the supervisor in a directory /etc/supervisor/conf.d/
Reach that directory.

```
cd /etc/supervisor/conf.d  
sudo touch gunicorn.conf
```

5. Open the created configuration file and write the below code in it.

```
sudo nano gunicorn.conf
```

```
[program:gunicorn]

directory=/home/ubuntu/django_test

command=/home/ubuntu/env/bin/gunicorn --workers 3 --bind
unix:/home/ubuntu/django_test/app.sock smarttuts.wsgi:application

autostart=true

autorestart=true

stderr_logfile=/var/log/gunicorn/gunicorn.err.log

stdout_logfile=/var/log/gunicorn/gunicorn.out.log

[group:guni]

programs:gunicorn
```

6. We need to create a log directory that we have given in the configuration file.

```
sudo mkdir /var/log/gunicorn
```

7. Make supervisor to read the created configuration file.

```
sudo supervisorctl reread
```

8. Make supervisor to start gunicorn.

```
sudo supervisorctl update
```

9. To check the status of the supervisor, run

```
sudo supervisorctl status
```

10. Now, configure the Nginx server. For that we need to reach the directory /etc/nginx/sites-available, where the entire Nginx configuration is saved.

```
cd /etc/nginx/sites-available
```

11. Create a configuration file for the django application and write in it.

```
sudo touch django.conf
sudo nano django.conf
```

```

server{

listen 80;

server_name ec2-3-84-35-84.compute-1.amazonaws.com;

location / {

include proxy_params;

proxy_pass http://unix:/home/ubuntu/django_test/app.sock;

}

}

```

12. To test the configuration

```
cd/etc/nginx/sites-available
```

13. Enable the created configuration, to make nginx read it.

```
sudo ln django.conf/etc/nginx/sites-enabled/
```

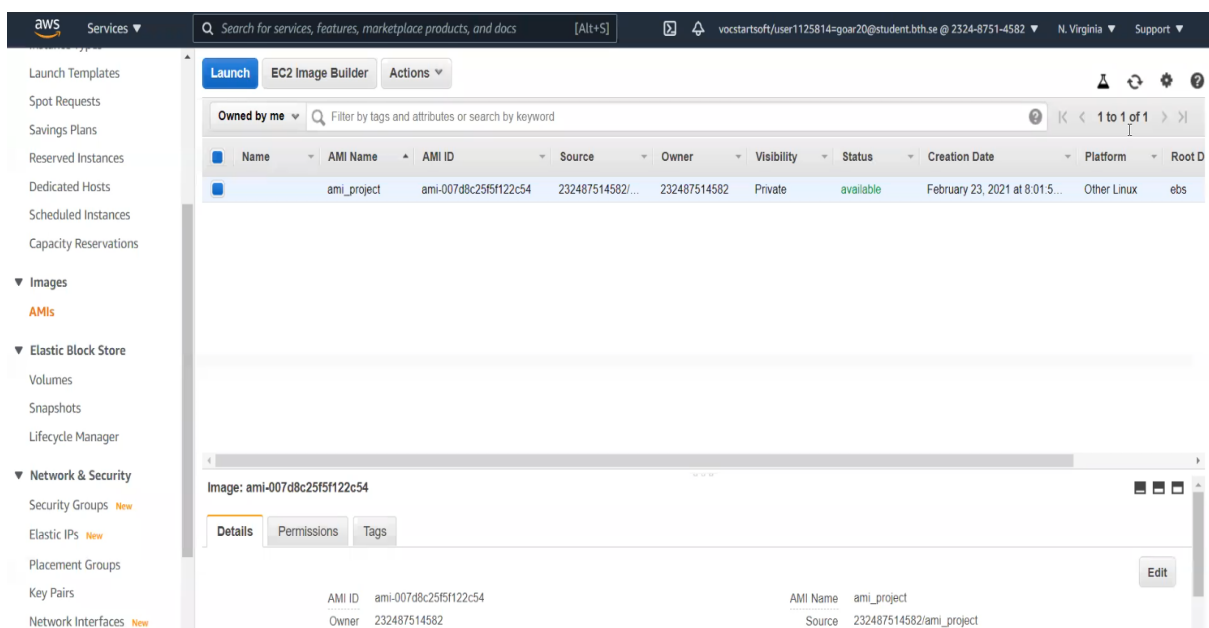
14. Finally, restart the Nginx server using the command:

```
sudo service nginx restart
```

15. visit the website with the IP address of the instance.

Autoscaling of the deployed django application in AWS:

Step1: create an image for the instance in which application is deployed.



The screenshot shows the AWS Management Console interface for the EC2 Image Builder service. The 'Launch' tab is selected, and a table lists the available AMIs. The table has columns for Name, AMI Name, AMI ID, Source, Owner, Visibility, Status, Creation Date, Platform, and Root Device. One AMI is listed: 'ami_project' with AMI ID 'ami-007d8c25f5f122c54'. Below the table, the 'Details' tab is selected, showing the AMI ID, Name, and Source.

Name	AMI Name	AMI ID	Source	Owner	Visibility	Status	Creation Date	Platform	Root D
ami_project	ami-007d8c25f5f122c54	232487514582/...	232487514582	Private	available	February 23, 2021 at 8:01:5...	Other Linux	ebs	

Image: ami-007d8c25f5f122c54

Details Permissions Tags

AMI ID: ami-007d8c25f5f122c54
Owner: 232487514582

AMI Name: ami_project
Source: 232487514582/ami_project

Step2: create an application load balancer for the application:

- In the configuration settings, select default VPC and subnets for that VPC.

1. Configure Load Balancer 2. Configure Security Settings 3. Configure Security Groups 4. Configure Routing 5. Register Targets 6. Review

Step 1: Configure Load Balancer

Availability Zones to increase the availability of your load balancer.

VPC

Availability Zones ☒ us-east-1a

IPv4 address

☒ us-east-1b

IPv4 address

Add-on services

Additional AWS services can be integrated with this load balancer at launch when you enable them below. You can also add these and other services after your load balancer is created by reviewing the "Integrated Services" tab for the selected load balancer.

AWS Global Accelerator ☐ Create an accelerator to get static IP addresses and improve the performance and availability of your application. [Learn more](#)
[Additional charges apply](#)

Your Accelerator will be created with the following name that you can customize. Once your Accelerator is created you can manage it from the Global Accelerator console.

Accelerator name

[Cancel](#) [Next: Configure Security Settings](#)

- Configure the security group with HTTP, SSH inbound rules.

Step 3: Configure Security Groups

A security group is a set of firewall rules that control the traffic to your load balancer. On this page, you can add rules to allow specific traffic to reach your load balancer. First, decide whether to create a new security group or select an existing one.

Assign a security group ☐ Create a new security group ☒ Select an existing security group

Filter

Security Group ID	Name	Description	Actions
<input checked="" type="checkbox"/> sg-03f982a51359820ec	abcdef	launch-wizard-2 created 2021-02-23T10:18:06.426+01:00	Copy to new
<input type="checkbox"/> sg-4b0edc7f	default	default VPC security group	Copy to new
<input type="checkbox"/> sg-06cd43e55d69e846f	project sg	launch-wizard-2 created 2021-02-20T14:34:11.384+01:00	Copy to new
<input type="checkbox"/> sg-0cb1eb5a4323750aa	secg	AutoScaling-Security-Group-1 (2021-02-19T11:03:40.123Z)	Copy to new
<input type="checkbox"/> sg-08f0e79433871fefa	security group_proj	AutoScaling-Security-Group-1 (2021-02-21T17:44:11.043Z)	Copy to new

[Cancel](#) [Previous](#) [Next: Configure Routing](#)

- Create a target group for the load balancer and then launch the application load balancer.

1. Configure Load Balancer 2. Configure Security Settings 3. Configure Security Groups 4. Configure Routing 5. Register Targets 6. Review

Step 4: Configure Routing

Your load balancer routes requests to the targets in this target group using the protocol and port that you specify, and performs health checks on the targets using these health check settings. The target group you specify in this step will apply to all of the listeners configured on this load balancer; you can edit the listeners and add listeners after the load balancer is created.

Target group

Target group

Name

Target type ☒ Instance ☐ IP ☐ Lambda function

Protocol

Port

Protocol version ☒ HTTP1
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.

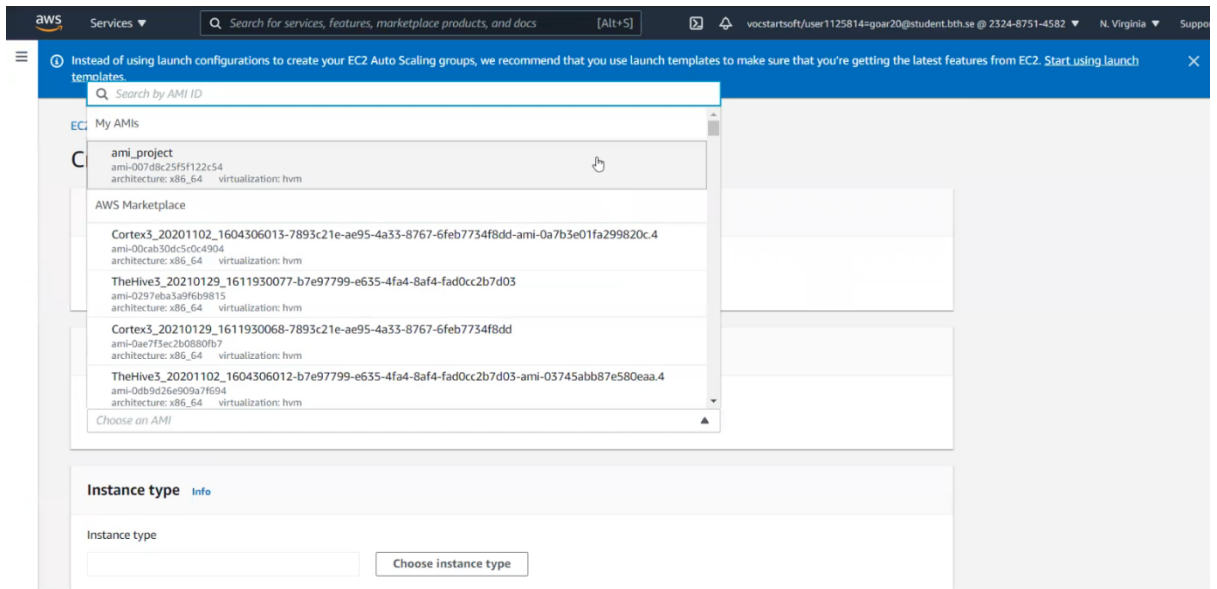
☐ HTTP2
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.

☐ gRPC
Send requests to targets using gRPC. Supported when the request protocol is gRPC.

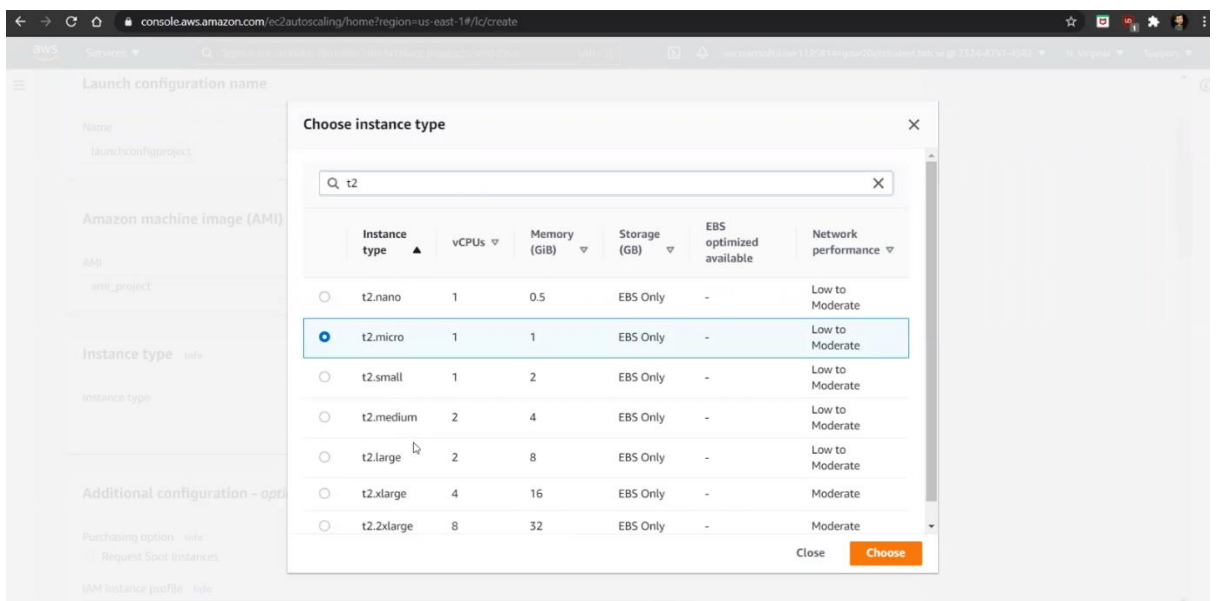
[Cancel](#) [Previous](#) [Next: Register Targets](#)

Step3: create a launch configuration for the AMI created from the instance.

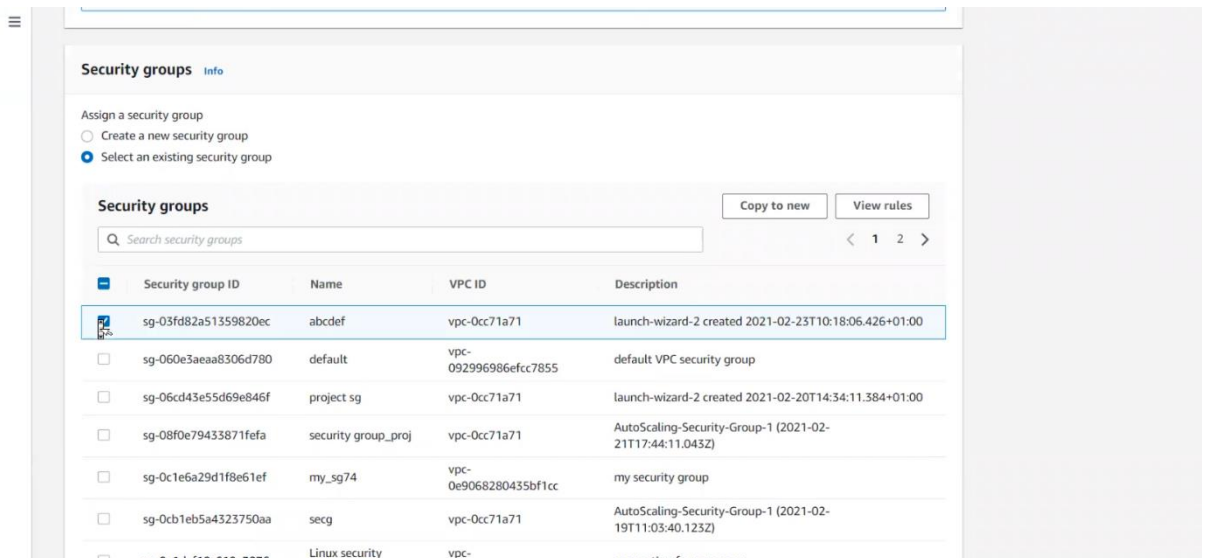
- Choose an AMI we created before from the AMIs list.



- Select the t2.micro instance type.



- Select the security group from the list of security groups created before or create a new one.



- Select the key pair and create the launch configuration.

Step4: create an autoscaling group using the created launch configuration.

- Select the launch configuration we created before.

- Select the default VPC and the subnets created for that default VPC.

Configure settings info

Configure the settings below. Depending on whether you chose a launch template, these settings may include options to help you make optimal use of EC2 resources.

Network info

For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.

VPC

vpc-0cc71a71
172.31.0.0/16 Default

[Create a VPC](#)

Subnets

Select subnets

us-east-1a | subnet-02614bc790af9e1cf (subnet1)
172.31.0.0/24

us-east-1b | subnet-096bfc8989f400b28 (subnet02)
172.31.1.0/24

[Create a subnet](#)

Cancel Previous Skip to review **Next**

- Attach the load balancer we created before to the auto scaling group.

Attach to an existing load balancer

Select the load balancers that you want to attach to your Auto Scaling group.

☒ Choose from your load balancer target groups
This option allows you to attach Application, Network, or Gateway Load Balancers.

☐ Choose from Classic Load Balancers

Existing load balancer target groups
Only Instance target groups that belong to the same VPC as your Auto Scaling group are available for selection.

Select target groups

abcdeftg | HTTP
Load balancer: Not associated with any load balancer

mytg | HTTP
Application Load Balancer: loadbalancerproject

targetgroupproject | HTTP
Load balancer: Not associated with any load balancer

tgproject | HTTP
Load balancer: Not associated with any load balancer

Health check grace period
The amount of time until EC2 Auto Scaling performs the first health check on new instances after they are put into service.

300 seconds

- Configure the group size and add the scaling policies.
- In this we can mention the desired, minimum, maximum capacities of the group.

Configure group size and scaling policies [Info](#)

Set the desired, minimum, and maximum capacity of your Auto Scaling group. You can optionally add a scaling policy to dynamically scale the number of instances in the group.

Group size - optional [Info](#)

Specify the size of the Auto Scaling group by changing the desired capacity. You can also specify minimum and maximum capacity limits. Your desired capacity must be within the limit range.

Desired capacity:

Minimum capacity:

Maximum capacity:

Scaling policies - optional

Choose whether to use a scaling policy to dynamically resize your Auto Scaling group to meet changes in demand. [Info](#)

- After successful creation of auto scaling group, the stated number of desired instances get launched and the application is deployed in them, that appears in the activity history of the auto scaling group.

Auto Scaling groups (1/1)

<input checked="" type="checkbox"/>	Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max	Availability zones
<input checked="" type="checkbox"/>	autoscaleproject.	launchconfigproject	2	-	2	1	3	us-east-1

Status	Description	Cause	Start time	End time
Successful	Launching a new EC2 instance: i-07c303486ceb26a96	At 2021-02-23T19:12:36Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2021-02-23T19:12:40Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2.	2021 February 23, 08:12:43 PM +01:00	2021 February 23, 08:12:43 PM +01:00
Successful	Launching a new EC2 instance: i-0597fda3cd5c47f6d	At 2021-02-23T19:12:36Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2021-02-23T19:12:40Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2.	2021 February 23, 08:12:43 PM +01:00	2021 February 23, 08:12:43 PM +01:00

- In the step scaling policy, select the metric CPU utilization and set the threshold value for the average CPU utilization, add the alarm for the metric, add the action to take and create the step scaling policy.

EC2 > Auto Scaling groups > project_autoScale

Create scaling policy

Policy type
Step scaling

Scaling policy name
Autoscaling for Django

CloudWatch alarm
Choose an alarm that can scale capacity whenever:
Average

Create a CloudWatch alarm

Take the action
Add

0 capacity units

Add step

Instances need
300 seconds warm up before including in metric

- Create an alarm that triggers when CPU utilization reaches the threshold value.

Statistic
Average

Period
5 minutes

Conditions

Threshold type
Static

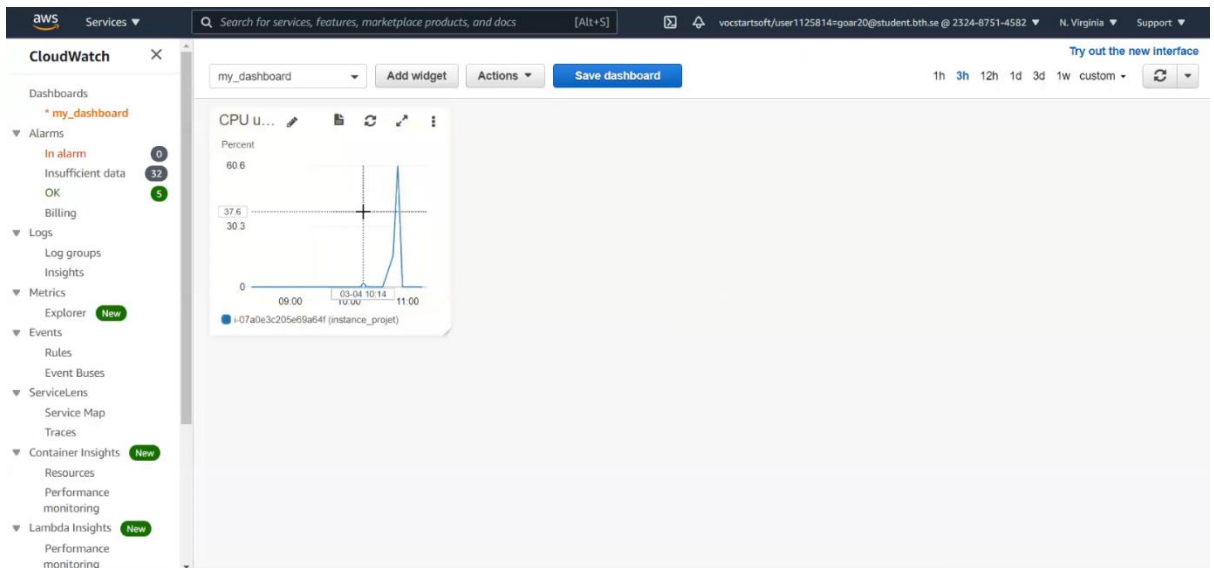
Whenever CPUUtilization is...
Define the alarm condition.
Greater

than...
Define the threshold value.
30

Additional configuration

Cancel Next

- Create a dashboard in the amazon cloud watch and add CPU utilization metric to the dashboard for monitoring.



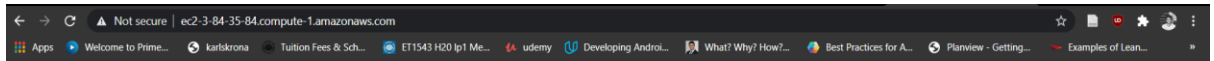
- Stress the CPU using apache bench below command which increases the CPU utilization:

ab -n 500000 -c 5 ip-172-31-1-208.ec2.internal/index.html

```
ubuntu@ip-172-31-1-208: ~  
ubuntu@ip-172-31-1-208:~$ ab -n 500000 -c 5 ip-172-31-1-208.ec2.internal/index.html  
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>  
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/  
Licensed to The Apache Software Foundation, http://www.apache.org/  
  
Benchmarking ip-172-31-1-208.ec2.internal (be patient)  
Completed 50000 requests  
Completed 100000 requests  
Completed 150000 requests  
Completed 200000 requests  
Completed 250000 requests  
[redacted]
```

Validation

- Static Django application is deployed in the ubuntu instance server and the application is accessed from the public IPV4 address of ubuntu instance (ec2-3-84-35-84.compute-1.amazonaws.com)



Deploying django app to aws Ec2

testing

- Auto scaling group with desired capacity 2 is launched with target tracking policy that tracks Average CPU utilization metric and deployed the application in that 2 instances using AMI created from the ubuntu instance.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
-	i-0eff8a529385b2171	Terminated	t2.micro	-	No alarms	us-east-1a	-
-	i-0893ae1502493bcd7	Terminated	t2.micro	-	No alarms	us-east-1a	-
auto Scaling in...	i-0dfbdbc424e64f14	Running	t2.micro	2/2 checks passed	1/1 has 1 alarm	us-east-1a	ec2-34-205-155-
instance_projet	i-07a0e3c205e69a64f	Running	t2.micro	2/2 checks passed	2 alarms	us-east-1b	ec2-3-84-35-84-
-	i-0e47c1dacec679de8	Terminated	t2.micro	-	No alarms	us-east-1b	-
-	i-0b12c93b6eda5b1dd	Terminated	t2.micro	-	1 alarm	us-east-1b	-
auto Scaling in...	i-075dc8334de3da95b	Running	t2.micro	2/2 checks passed	1/1 has 1 alarm	us-east-1b	ec2-107-22-132-

Instance ID	Public IPv4 address	Private IPv4 addresses
i-075dc8334de3da95b (auto Scaling instance2)	107.22.132.15 open address	172.31.1.246

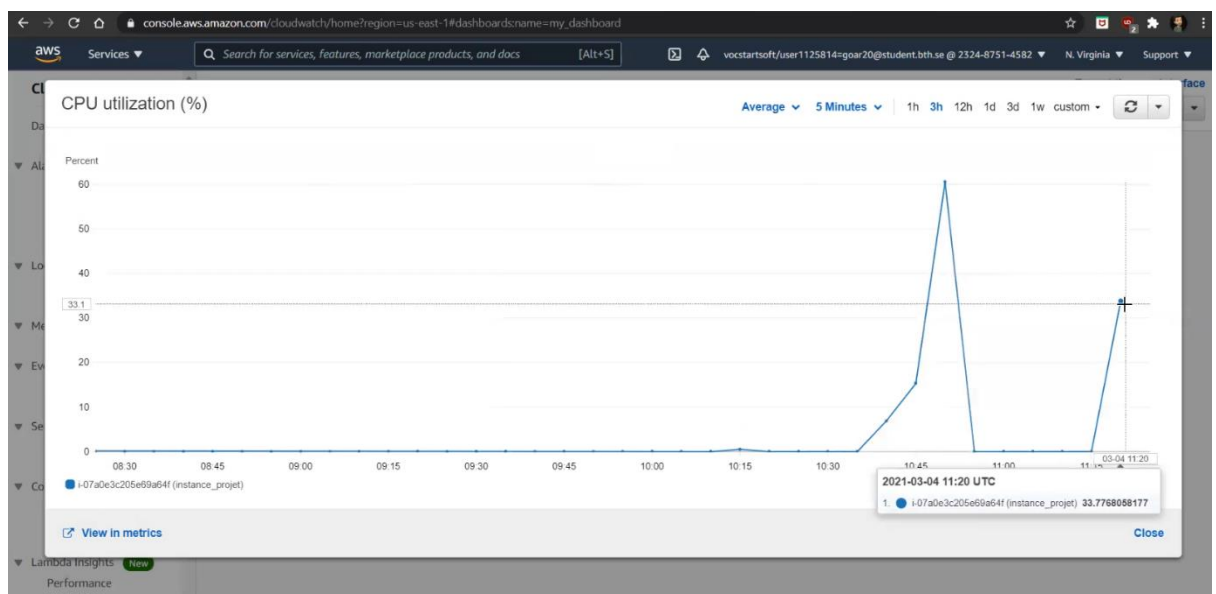
Instance state	Public IPv4 DNS	Private IPv4 DNS
Running	ec2-107-22-132-15.compute-1.amazonaws.com open address	ip-172-31-1-246.ec2.internal

Instance type	Elastic IP addresses	VPC ID
t2.micro		

Static Django application is deployed in the instances created and can be accessed from the public IPV4 address of the instance. (ec2-107-22-132-15.compute-1.amazonaws.com)



- The CPU utilization is increased with increased stress using apache bench and can be monitored in the amazon cloud watch dashboard.



- An instance is added after increasing the stress on CPU which reached the threshold value of CPU utilization 30, specified in the step scaling policy satisfying Scalability with respect to computation.

The screenshot shows the AWS Management Console interface. On the left is a navigation menu with options like 'New EC2 Experience', 'EC2 Dashboard', 'Events', 'Tags', 'Limits', 'Instances', 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Scheduled Instances', 'Capacity Reservations', 'Images', 'AMIs', 'Elastic Block Store', 'Volumes', and 'Snapshots'. The main content area is titled 'Instances (1/6)' and includes a search bar and a table of instances.

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
<input type="checkbox"/>	-	i-0bff8a529385b2171	Terminated	t2.micro	-	No alarms	us-east-1a	-
<input type="checkbox"/>	auto Scaling in...	i-0dfbdc424e64f14	Running	t2.micro	2/2 checks passed	1 alarms	us-east-1a	ec2-34-205-155-
<input checked="" type="checkbox"/>	instance_projct	i-07a0e3c205e69a64f	Running	t2.micro	2/2 checks passed	2 alarms	us-east-1b	ec2-3-84-35-84.c
<input type="checkbox"/>	-	i-0b12c93b6eda5b1dd	Terminated	t2.micro	-	1/1 has	us-east-1b	-
<input type="checkbox"/>	auto Scaling in...	i-075dc8334de3da95b	Running	t2.micro	2/2 checks passed	1 alarms	us-east-1b	ec2-107-22-132-
<input type="checkbox"/>	-	i-0ea60863608b356e1	Running	t2.micro	2/2 checks passed	1 alarms	us-east-1b	ec2-3-87-197-18

Below the table, the 'Instance: i-07a0e3c205e69a64f (instance_projct)' details are shown. The 'Details' tab is active, displaying the following information:

- Instance ID:** i-07a0e3c205e69a64f (instance_projct)
- Public IPv4 address:** 3.84.35.84 | [open address](#)
- Private IPv4 addresses:** 172.31.1.208
- Instance state:** Running
- Public IPv4 DNS:** ec2-3-84-35-84.c
- Private IPv4 DNS:** ec2-3-87-197-18

Results

- The static django application has pulled from the GitHub and deployed in the ubuntu instance server which is launched in the Amazon Web Services by configuring the Nginx web server, Gunicorn WSGI which increased the performance and made the application to run in the background as shown in the validation.
- The deployed django application is auto scaled by adding the scaling policies with 2 desired instances which configured with the threshold value of the average CPU utilization of the ubuntu server in which the static Django application is deployed.
- Increased stress also increased CPU utilization, which after reaching the threshold value triggered the alarm and acted satisfying the step scaling policy, added 1 instance in the dashboard.
- Thus, the Django application deployed in the Amazon Web Services satisfied the requirements scalability and high availability with respect to computation.