National College of Ireland

Configuration Manual

MSc Research Project

Programme Name

Sahil Das

Student ID: 22211446

School of Computing

National College of Ireland

Supervisor: Mark Monaghan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sahil Das……. ……
Student ID:	22211446……
Programme:	MSc in Cyber Security…… **Year:** 2024……
Module:	Practicum……
Lecturer:	Mark Monaghan……
Submission Due Date:	12/08/2024……
Project Title:	…… ……
Word Count:	…… **Page Count:** ……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sahil Das……
Date:	12/08/2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

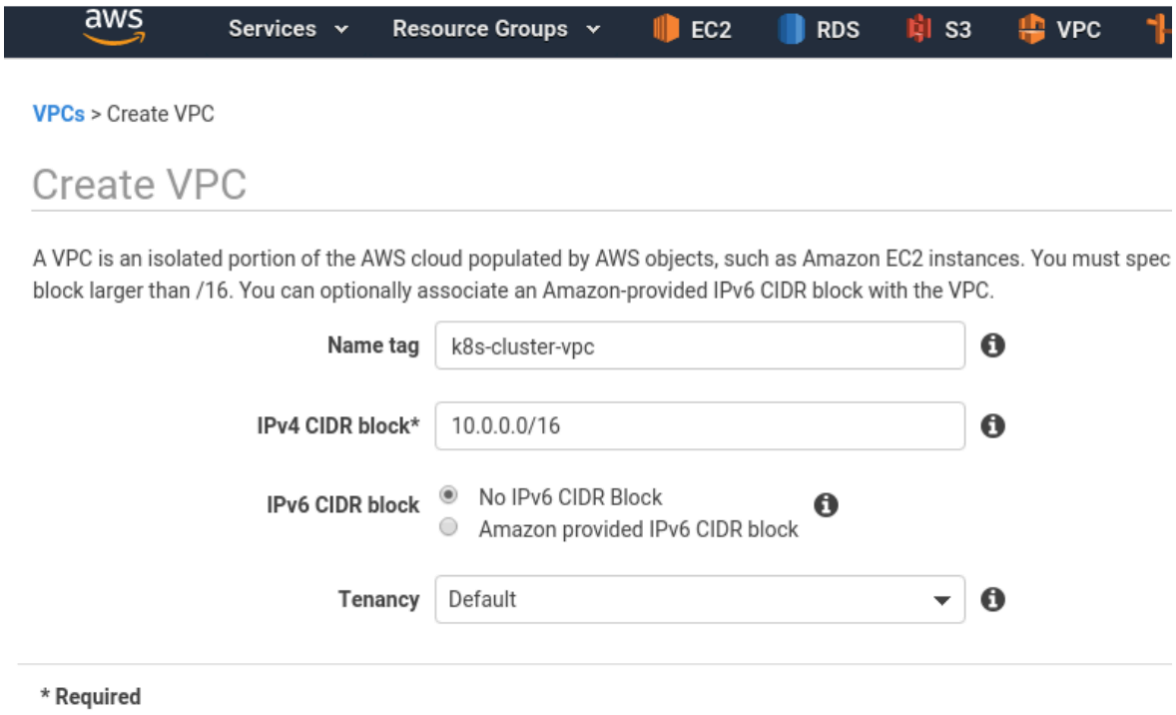
Sahil Das
Student ID: 22211446

1 Configuring AWS Cloud for Kubernetes Deployments

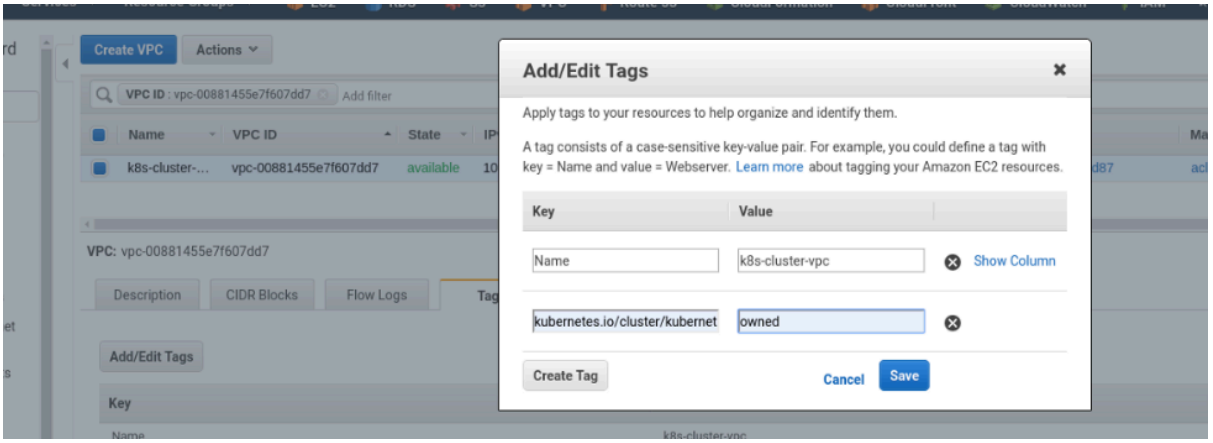
Configuring Secondary IP to EC2 instance

1) Create a new network interface on the EC2 Console under the Networking and Security section with same security group and VPC as that of EC2 instance
2) Attach the elastic ip to the instance
3) Attach the network interface to the instance by selecting the instance and going to Actions and then Networking and Attach Network Interface

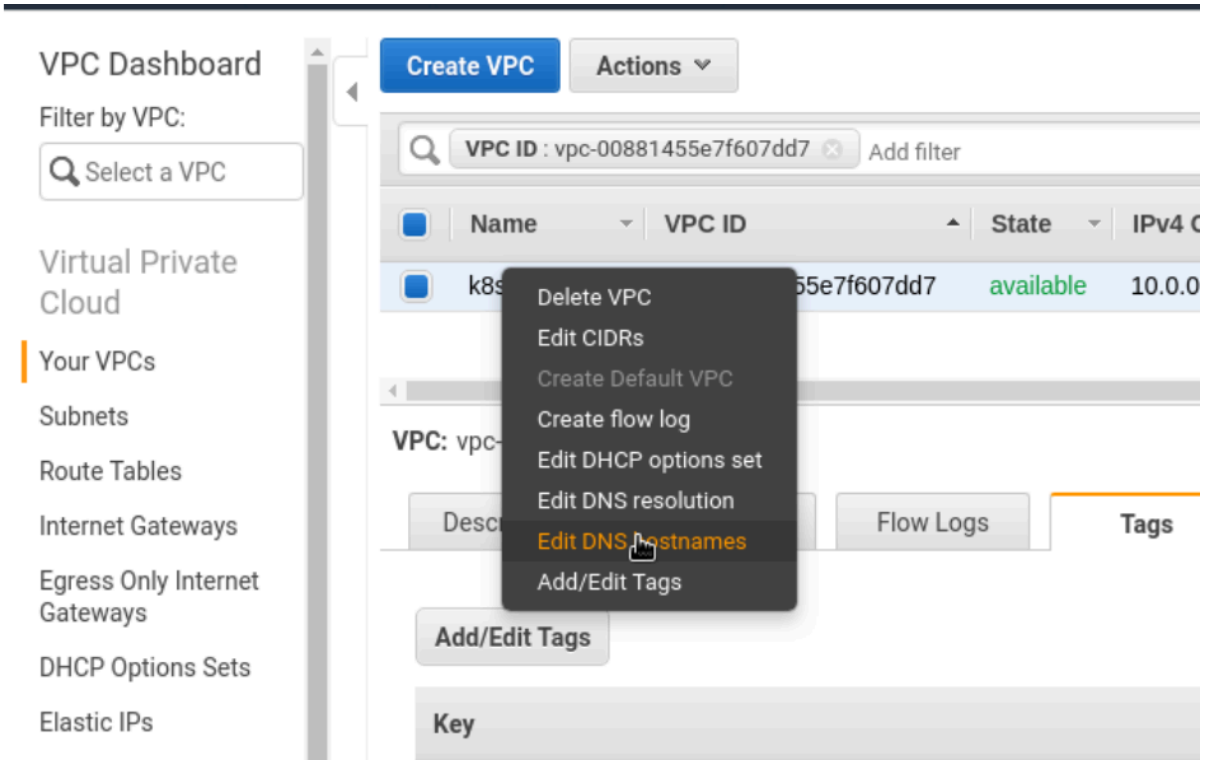
Create a VPC with the *10.0.0.0/16* CIDR:



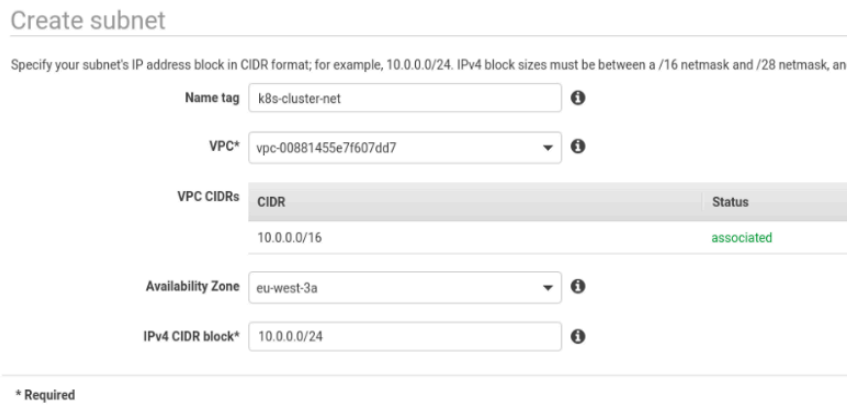
Add the tag which will have the name value as kubernetes.io/cluster/kubernetes with the owned value which will be helpful for the autodiscovery while setting up anything related to the Kubernetes Stack



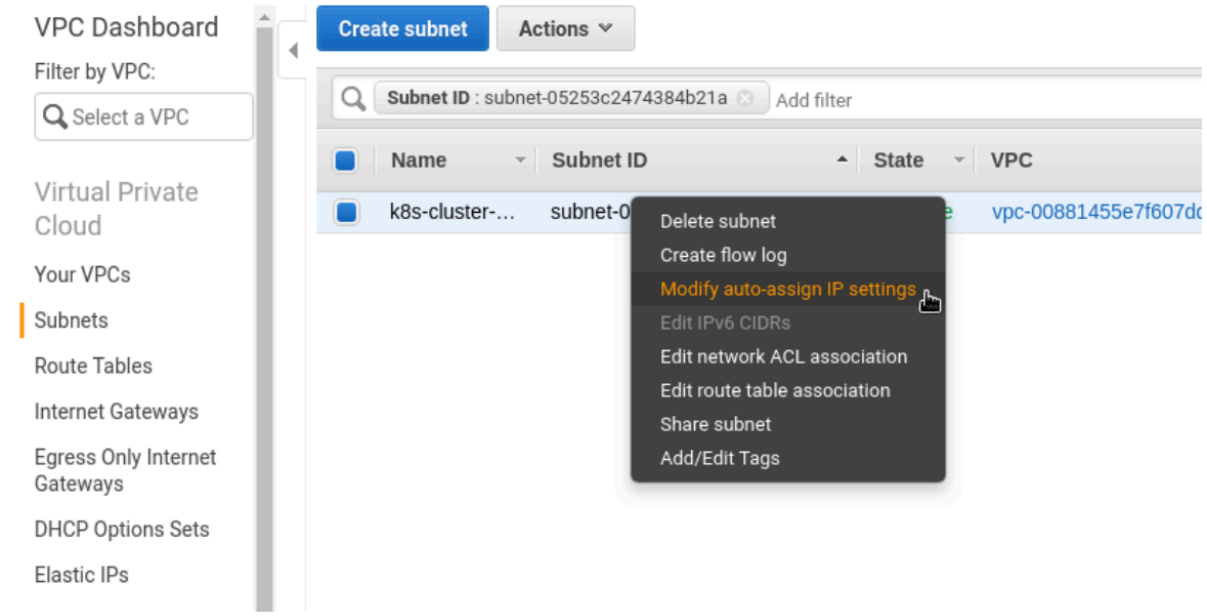
The DNS Hostname needs to be activated:



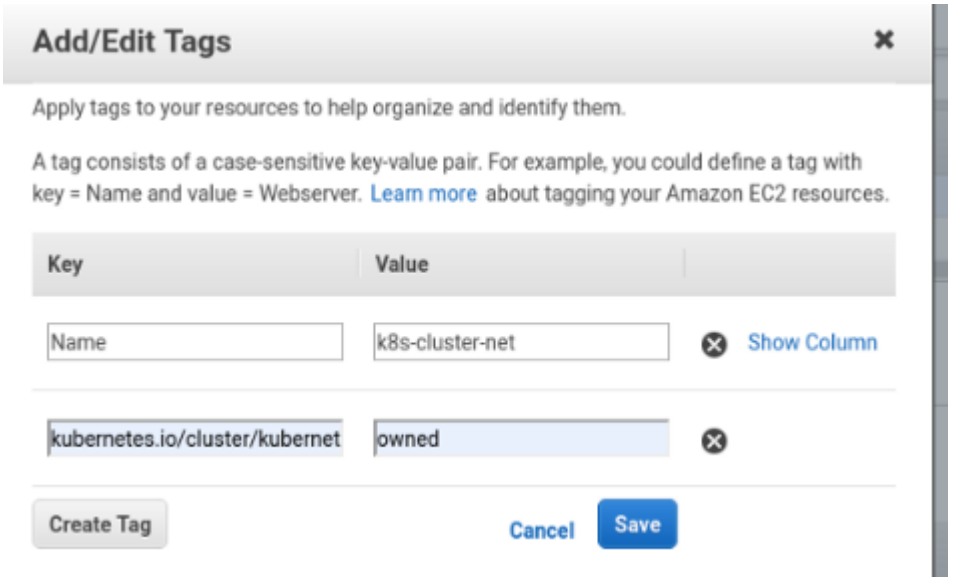
A new subnet will also be made in the process



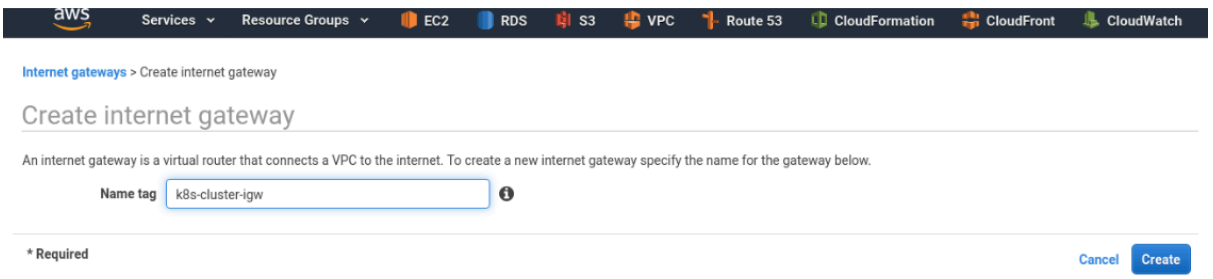
Public IPs will be enabled for the instances that will be present in this Subnet



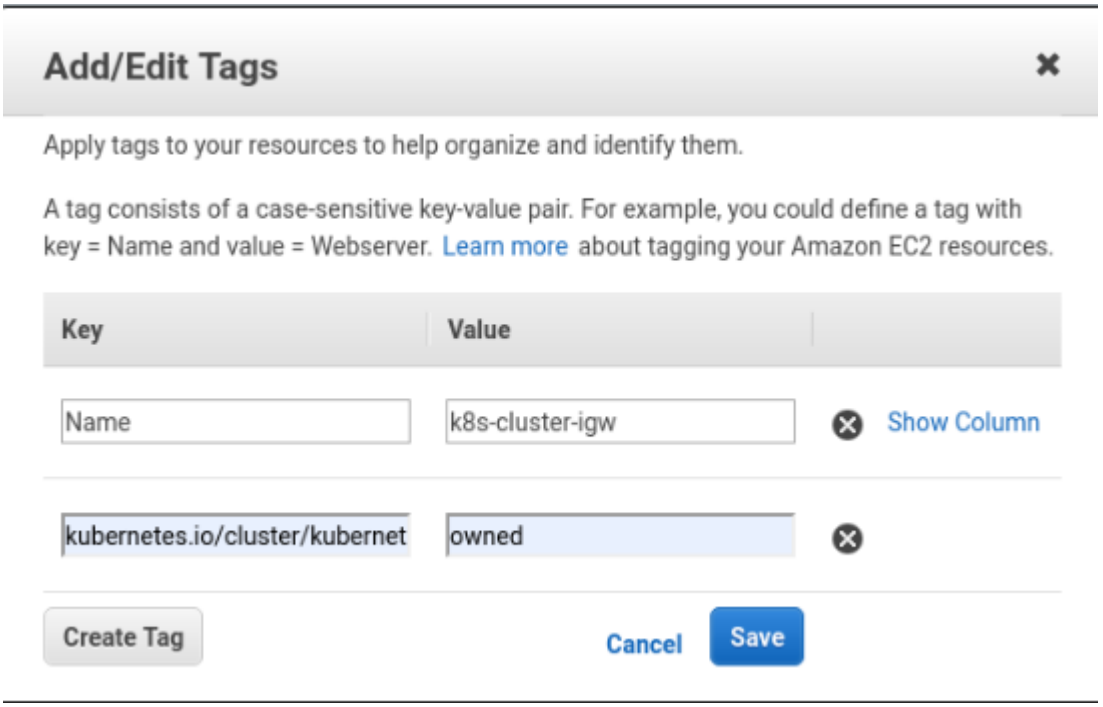
The tags created early on will be added which is kubernetes.io/cluster/kubernetes with value owned



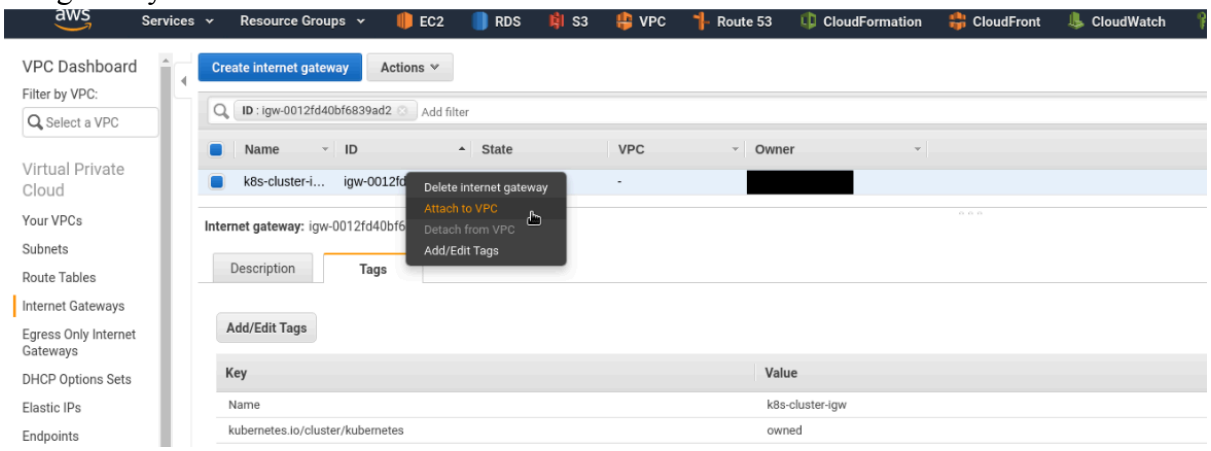
An internet gateway will also be created which will be attached to the subnet for providing external internet access



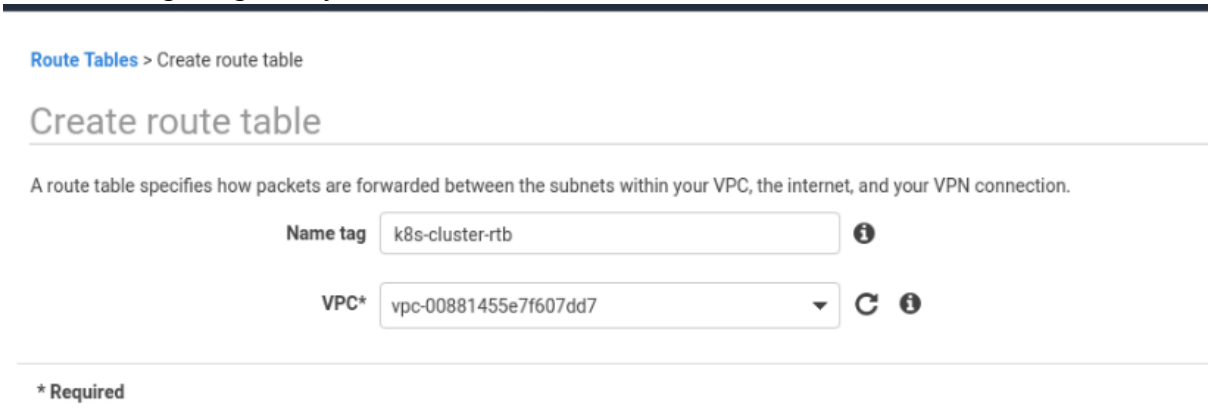
The same tags will be added again



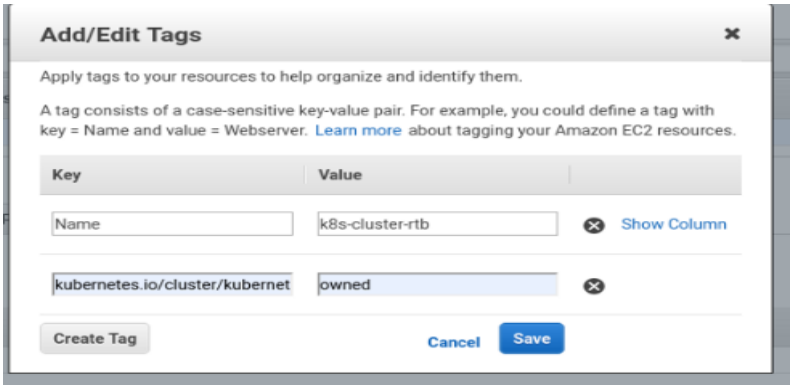
The gateway created above will be attached to the VPC



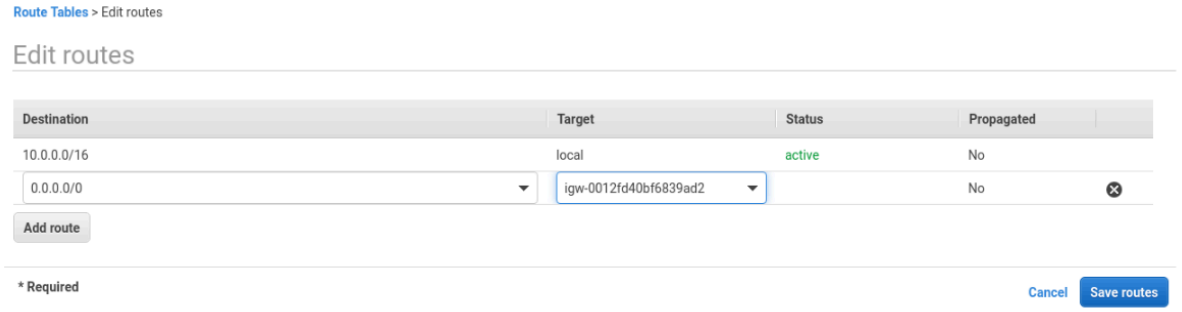
After creating the gateway a route table will be created for the flow of the traffic



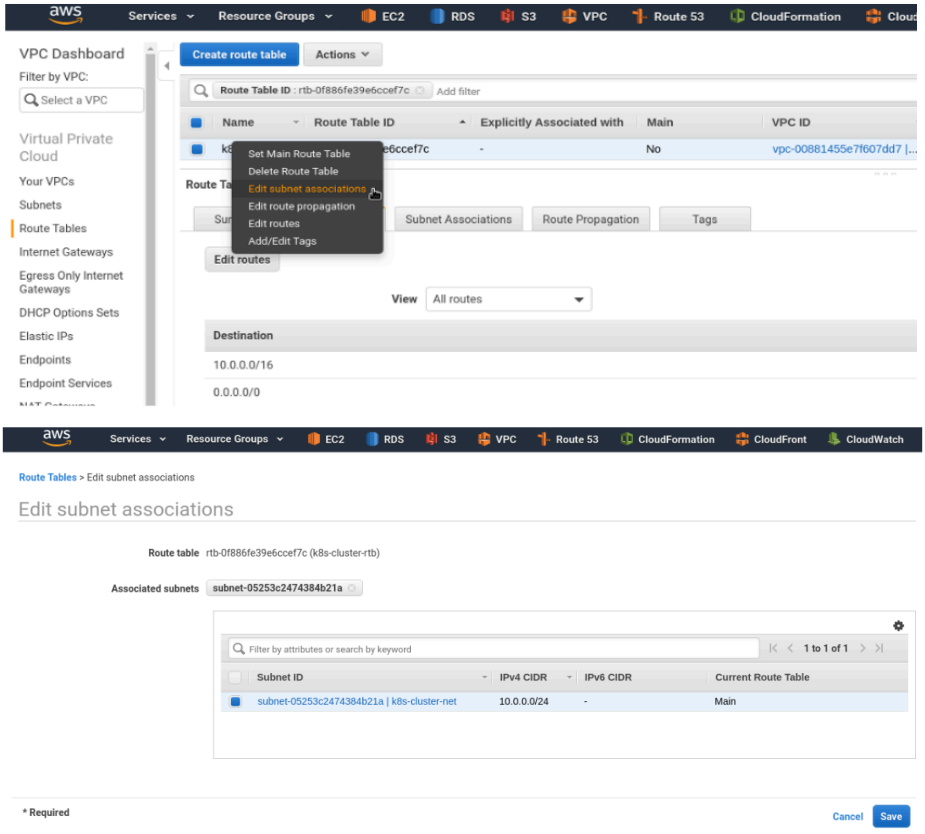
Those tags will be added here as well



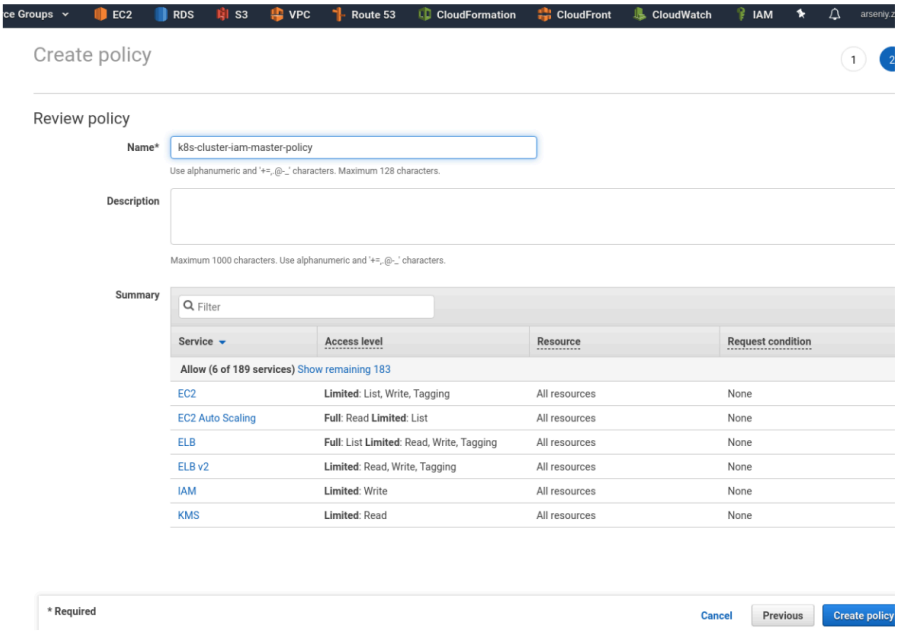
A new route will be added which will have a destination 0.0.0.0/0 that means any system can connect to the cluster



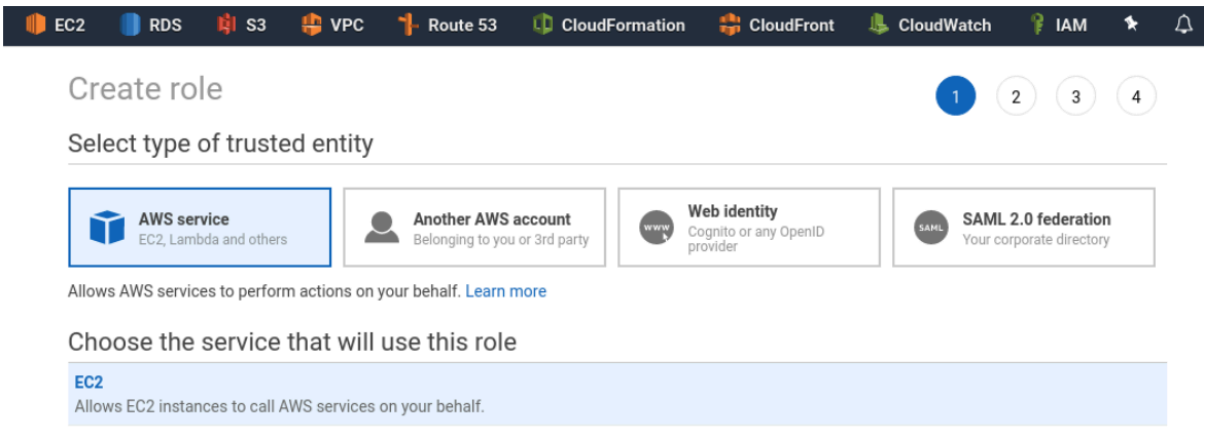
Editing the subnet association will help to attach the route table to this and the subnet created earlier is chosen



IAM role will be created for both master and worker nodes which will be the EC2 instances. In order to do so we go to the IAM and then Policies and then click on create Policies also cloud-provider-aws option would be there to generate the policies



In the roles section a new role will be created for the EC2 instance

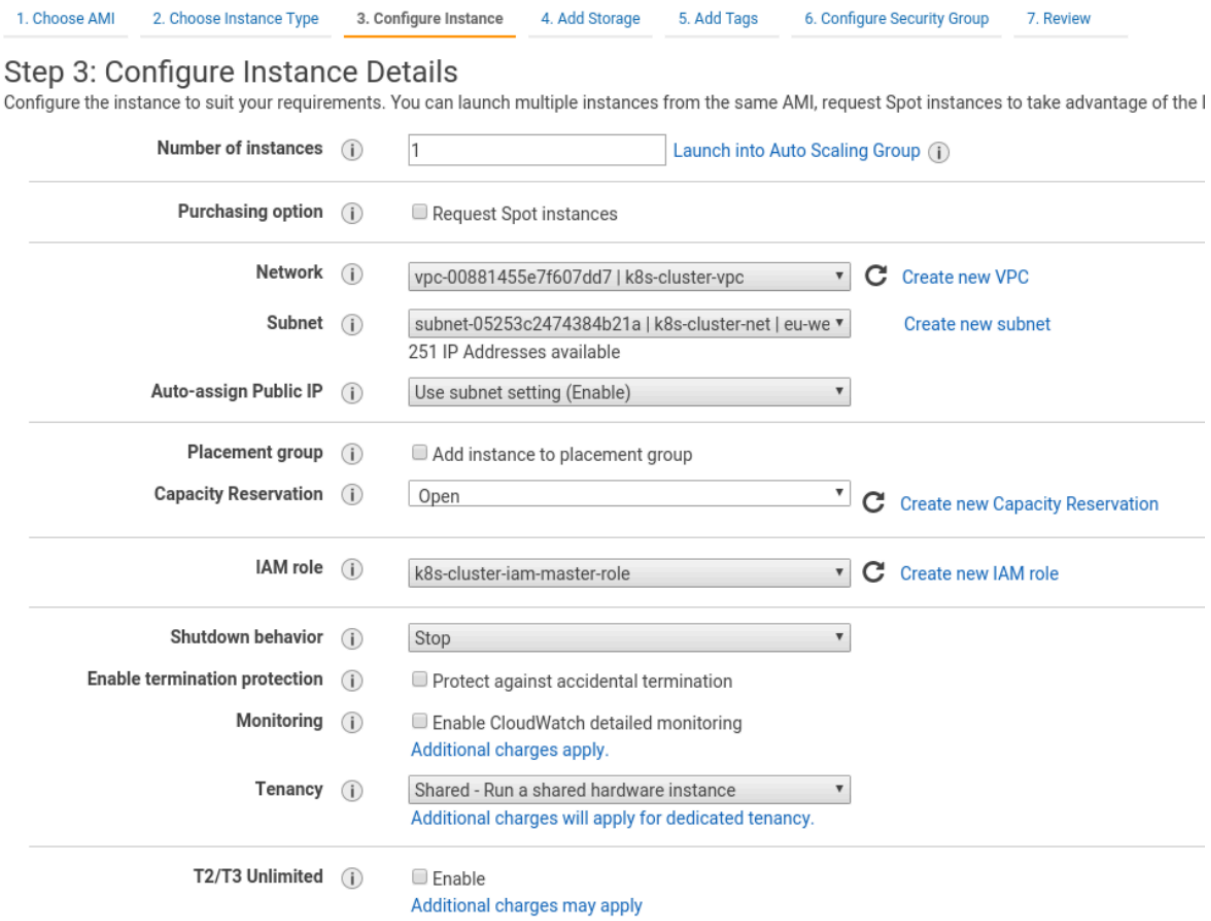


In the permissions tab we will find the policy created earlier and attach the same.

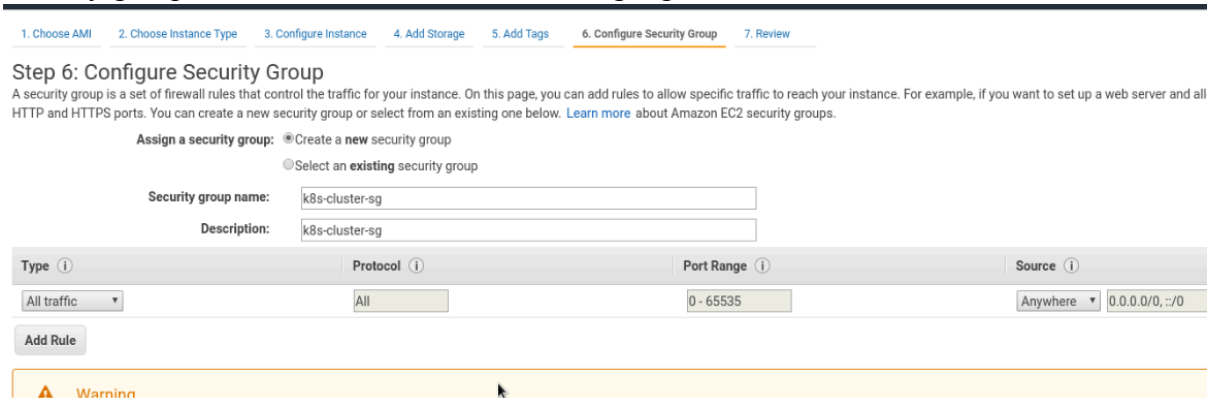
IAM Worker position
Likewise, develop an additional policy for worker nodes.
Save it under the name you choose, k8s-cluster-iam-worker-policy.

Utilising EC2

Using your VPC, create an EC2 with the t2.medium type (minimum type since Kubernetes master requires at least 2 CPU cores), and set the IAM role to k8s-cluster-iam-master-role.



Security group will be created with the following tags



Use the k8s-cluster-iam-worker-role to create a Worker Node in the same manner as the Master is spinning up.

The master node must be deployed using these commands.(Note that you must execute these commands as the root.

Kubeadm join command
kubeadm token create --print-join-command
Note down the credentials of the join command and add them in the commands for the worker nodes
Commands Required to deploy the worker node using the Kubeadm method.(Please note that these commands
must be run as the root user)

To setup Ingress on Kubernetes
Apply this command to the kubernetes cluster
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.1/deploy/static/provider/aws/deploy.yaml

Default Install of Cert-Manager
kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.7.1/cert-manager.yaml

2 Testing the websites using Python and JMeter

The first testing is done using JMeter, For that we will be installing JMeter in our local system. JMeter is available online which we download and install.

Once installed, we will configure the JMeter to perform the tests accordingly.
For that first we have to create a thread group. In order to do so we have to do the following steps:

1) Launch the JMeter window.
2) The window is split into two sections: the right side has all of the element configurations, and the left side has the additional elements.
3) Save the test plan with a new name.
4) After performing a right-click on the test plan, select Thread(Users) and then Thread(Groups).

Once we click on the thread group there are three things that needs to be configured before starting the test:

1) The number of threads or users the tool will stimulate, in our case we will set it to 300 users
2) Ramp up period which will ensure the time gap in seconds for each thread in our case we will be doing it after every 10 seconds
3) Loop count means how many times the tests will be executed and this will be 3 in our case

The next step would be adding an HTTP Requests Default which will send multiple HTTP/HTTPS requests to the users, and this can be done by following these steps:

1. On the Threads Group, do a right-click.
2. After selecting the add config element, select HTTP Request Defaults.
3. Enter the server name or IP address that you wish to test in the window that pops up.
4. The next action item is to add an HTTP Cookie Manager. To do this, right-click on the add element, then select the config element, and finally choose HTTP Cookie Manager.

In order to try the concurrent requests on each directory of the website, we can add a HTTP Request Sampler by following these steps:
Under the HTTP Request gives the path that the user will request. In our case we will add /, /api, /robots.txt, /admin.php and some other directories to test out the race condition.

Then the test plan is saved and run.

In order to prevent the race condition that happened, certain changes are done in the ingress and deployment .yaml parameters of the website which in this case was travstack.tech. The parameter changes are mentioned below:

Updated parameters for deployment.yaml. This will create 3 more containers with a given memory and cpu configuration which will help the pod or the virtual system to handle more requests efficiently

replicas: 3

```
 resources:
    limits:
       memory: “200Mi”
       cpu: “700Mi”
    Requests:
       memory: “200Mi”
       cpu: “700Mi”
```

Once changes are done type the command, kubectl apply -f deployment.yaml

For the ingress, we will add the following parameter to the file:
nginx.ingress.kubernetes.io/limit-rps: “10” and then we use the command kubectl apply -f ingress.yaml

Also another thing will be added in the directory as an additional protection we will configure a rate limiting policy which will be protecting the entire namespace in which the website is operating, this will be done as follows:

```
apiVersion: specs.ami.nginx.com/valpha2
kind: RateLimit
metadata:
     name: ratelimit-v1
     namespace:  website
spec:
   destination:
  kind: Service
  name: newservice
  namespace: website
sources:
      -    kind: Deployment
           name: travstack
           namespace: website
name: 40rm
rate: 40r/m
```

Save this as ratelimit.yaml and use the command kubectl apply -f ratelimit.yaml

So this is one of the ways to prevent the race conditions that can lead to the DoS attack

Testing using Python Scripts and adding protection mechanisms by changing the logic of the code

Now apart from testing the website of the travel organisation, we will also test another website first which is a simple shopping website which is deployed on docker.
The docker image can be started as follows:

1) cd faster_shop
2) docker build . -t local/faster_shop
3) docker run –it –rm -p 1002:1002 local/faster_shop

Now we will try to perform the attack manually by modifying the requests going through. We will be intercepting the requests via BurpSuite. In order to configure BurpSuite with firefox browser we need to do the following steps:

1) Go to settings
2) Scroll down and look for Network Settings.
3) In the network settings select the Manual Proxy Settings and the IP Address and Port 8080

Once we perform the buy and sell requests we can see it in the burpsuite window. We will write a script in python which will buy one milk and sell N number of milk. The logic of the code can be something like this:

```
import threading
import time
class Buyer(threading.Thread):
  def run(self):
    while True:
      print("Buying milk...")
      buy(token, "1")
      time.sleep(500)
class Seller(threading.Thread):
  def run(self):
    while True:
      Some code that find the id
      id = 1
      sell(token, id)
```

We will be adding this snippet to a bigger aspect of the code and save it as racecondition.py and then execute it by the command python3 raceondition.py
And in this case we will fix the sell endpoint alongside the buy endpoint and we will do this by ensuring the following logic steps are implemented

1) Check if you are logged in
2) Check if the purchase exists
3) Delete the item
4) Update the balance