Module 2: Security Management in AWS

Demo Document 7

Title: Configure WAF to Protect Website from Attacks

Use Case

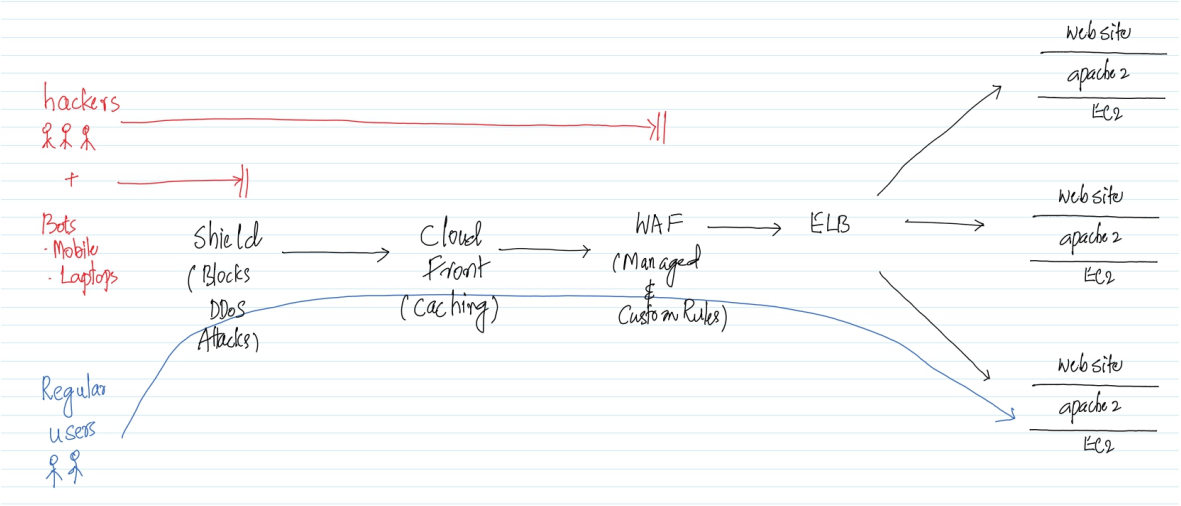
With the proliferation of the internet, many companies are trying to become global. For example, Amazon has its presence in many countries worldwide. But, with expansion comes a greater risk of hacking and data theft. The application stack has become so complex that I there is always some sort of vulnerability in the different layers (OS, web, DB, network, storage, balancing, etc.), and the hackers try to exploit them for financial and other benefits. Companies have to be on their toes to keep the hackers at bay while making sure only the legitimate users have access the website and other applications.

Protecting the publicly exposed services like websites and REST/Web Services based interfaces consumes a lot of effort, time, and money for the organizations. However, AWS provides a few managed services like AWS Shield and AWS WAF (Web Application Firewall) that help in mitigating the security risks to a greater degree. This feature is especially valuable to cash-strapped companies who have a variety of costs to over.

AWS WAF is a service for protecting the websites and the APIs from some of the common attacks like SQL Injection, XSS (Cross-Site Scripting), SSRF (Server Side Request Forgery) and including those published in OWASP (Open Web Application Security Project). However, it may sometimes call for the need to apply custom rules to meet the application requirements.

AWS Shield provides support against the DDOS (Distributed Denial of Service) attacks. These are the types of attacks when invalid requests are sent in bulk, while the legitimate user requests are throttled for resources. AWS Shield comes in two editions: Standard, which is free of cost, and Advanced, which costs around 3000 USD a month with a commitment of 1 year full-fledged support. By default, Shield Standard is turned on CloudFront and Route53, and there is nothing we have to do except using a CloudFront distribution, which automatically comes with Shield Standard.

Architectural diagram:

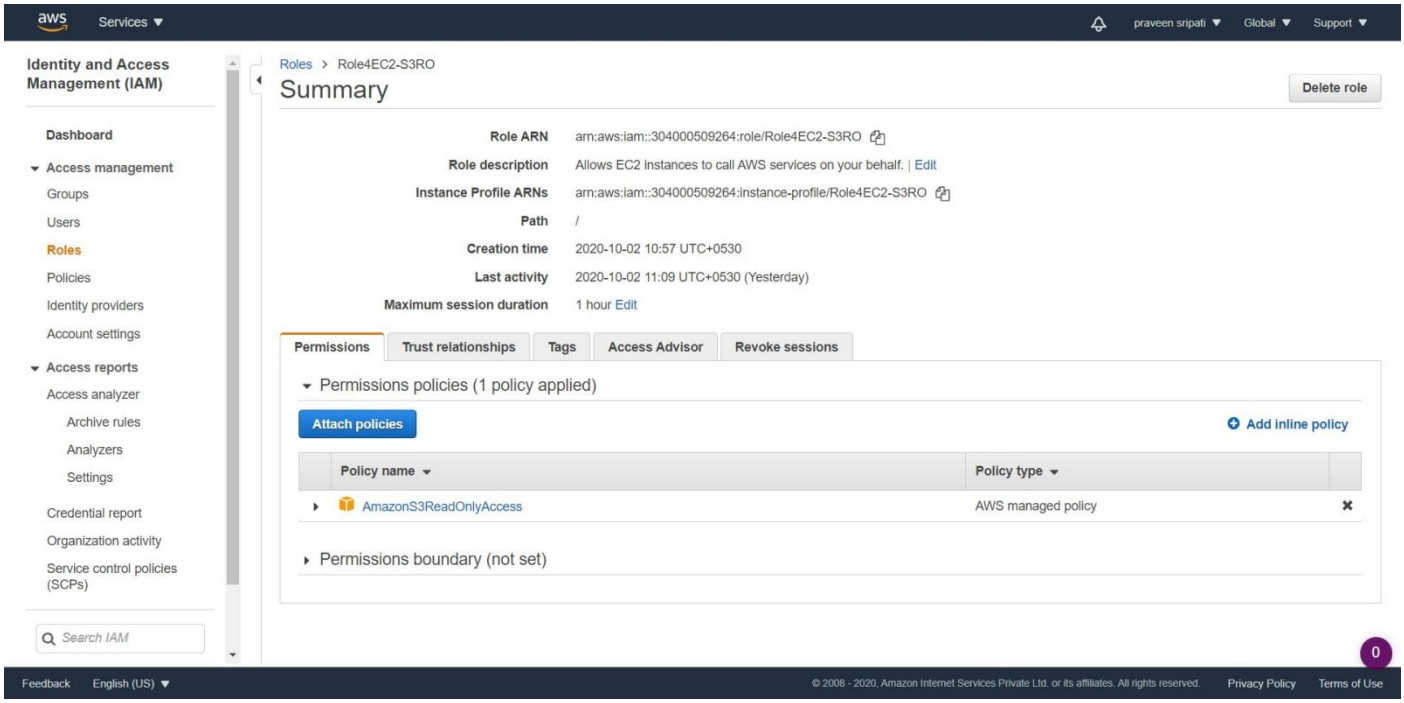


In this use case, we will create an EC2 with an SSRF vulnerability application deployed on it. Then, we will try to mitigate the vulnerability using WAF. For this, we need to create an Application Load Balancer and integrate it with WAF.

AWS Services: WAF, ELB, EC2

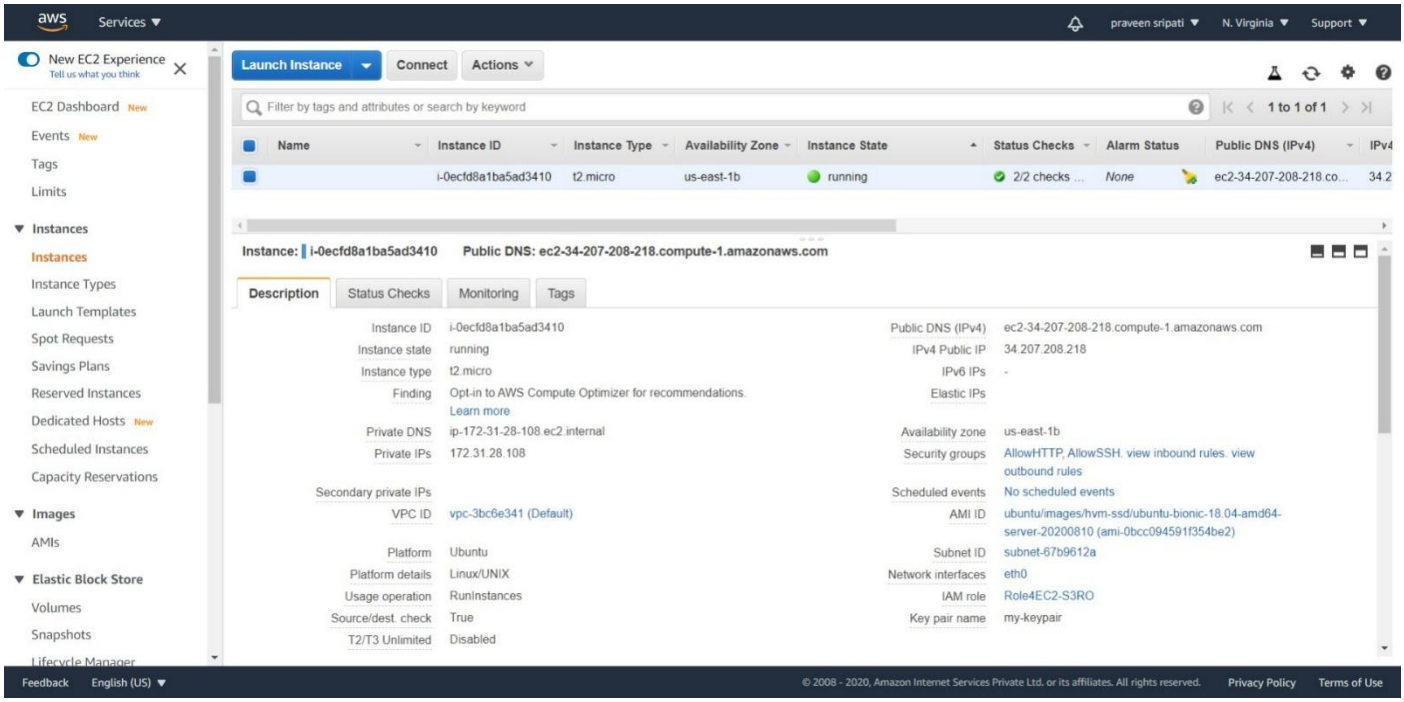
Solution Steps:

Step 1: Go to the IAM Management Console and create a Role for EC2 with AmazonS3ReadOnlyAccess Policy attached to it. Make sure the name of the role is **Role4EC2-S3RO**.



Step 2: Create an Ubuntu EC2 with the below details and connect to it:

- t2.micro
- Security Group (22/SSH and 80/HTTP) inbound
- The above role attached
- And, Keypair attached for authentication


aws
Services
praveen sripati
N. Virginia
Support
New EC2 Experience
Tell us what you think
EC2 Dashboard New
Events New
Tags
Limits
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts New
Scheduled Instances
Capacity Reservations
Images
AMIs
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 1 of 1
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
Alarm Status
Public DNS (IPv4)
i-0ecfd8a1ba5ad3410
t2.micro
us-east-1b
running
2/2 checks ...
None
ec2-34-207-208-218.co...
Instance: i-0ecfd8a1ba5ad3410
Public DNS: ec2-34-207-208-218.compute-1.amazonaws.com
Description
Status Checks
Monitoring
Tags
Instance ID i-0ecfd8a1ba5ad3410
Instance state running
Instance type t2.micro
Finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
Private DNS ip-172-31-28-108.ec2.internal
Private IPs 172.31.28.108
Secondary private IPs
VPC ID vpc-3bc6e341 (Default)
Platform Ubuntu
Platform details Linux/UNIX
Usage operation RunInstances
Source/dest. check True
T2/T3 Unlimited Disabled
Public DNS (IPv4) ec2-34-207-208-218.compute-1.amazonaws.com
IPv4 Public IP 34.207.208.218
IPv6 IPs -
Elastic IPs
Availability zone us-east-1b
Security groups AllowHTTP, AllowSSH. view inbound rules. view outbound rules
Scheduled events No scheduled events
AMI ID ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200810 (ami-0bcc094591f354be2)
Subnet ID subnet-67b9612a
Network interfaces eth0
IAM role Role4EC2-S3RO
Key pair name my-keypair
Feedback
English (US)
© 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Step 3: Execute the below command on the EC2 and notice the Access Keys are displayed as part of the **AccessKeyId** and **SecretAccessKey** fields. This is displayed because it is part of the EC2 instance metadata and can be retrieved, as mentioned here.

curl http://169.254.169.254/latest/meta-data/iam/security-credentials/Role4EC2-S3RO

Step 4: Execute the below commands in the EC2 to install Ruby and Sinatra, a DSL (Domain Specific Language) to create web applications in Ruby. Installation of Sinatra will take a few minutes to complete.

```
sudo apt-get update
sudo apt-get install ruby -y
sudo gem install sinatra
```

Step 5: On the EC2, create a file **server.rb** with the below content. This is what the program does

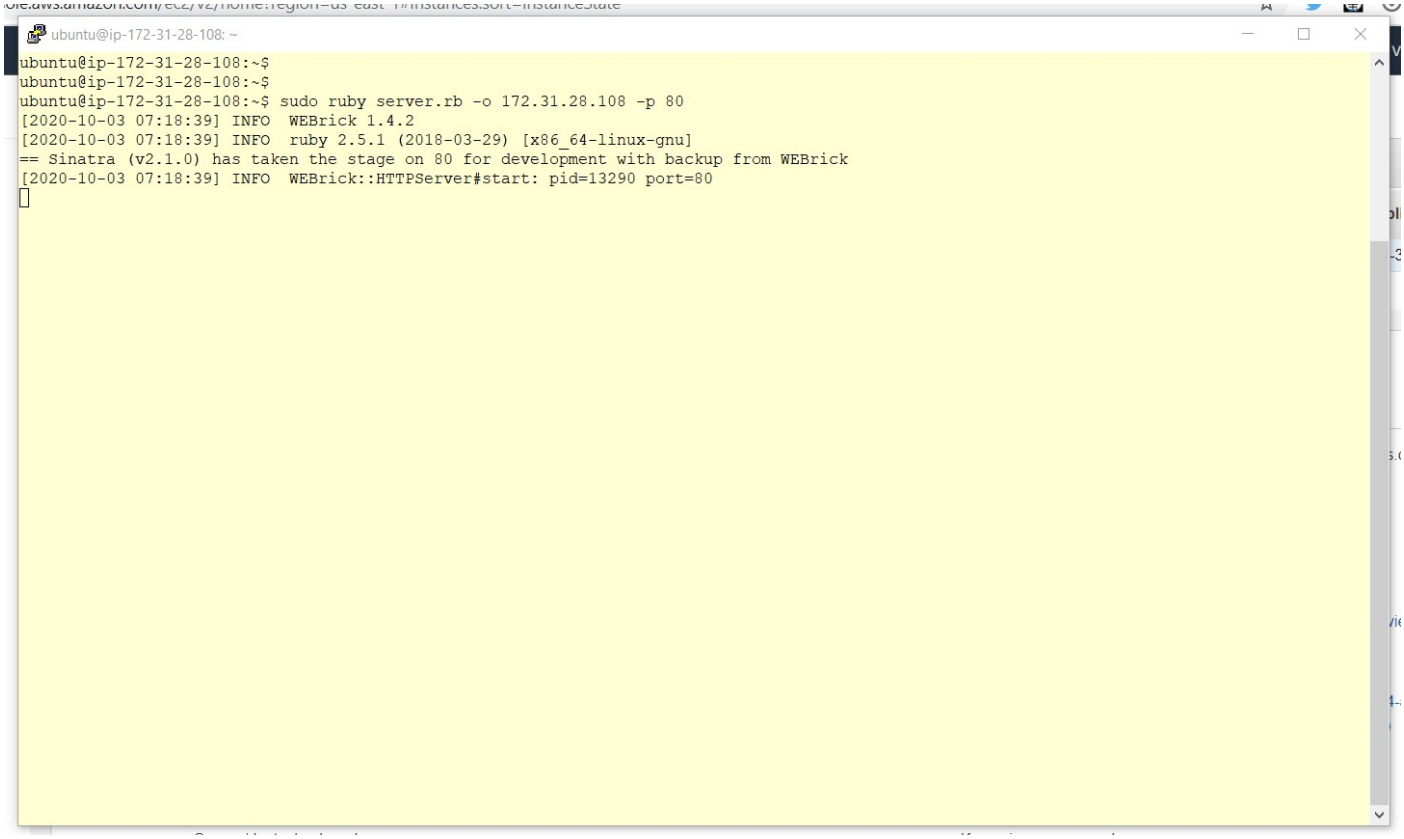
- Creates an HTTP endpoint (webserver) on port 80
- Takes a URL as an input (query parameter) via HTTP
- Invokes the same URL via the open function
- Responds back to the response from the URL

```
require 'sinatra'
require 'open-uri'

get '/' do
  format 'RESPONSE: %s', open(params[:url]).read
end
```

Step 6: Run the program using the below command. Make sure to replace the IP address with the private IP address of the EC2. The program will go in an infinite loop, and if any other command has to be run on the EC2, then a new session has to be opened.

sudo ruby server.rb -o **172.31.28.108** -p 80

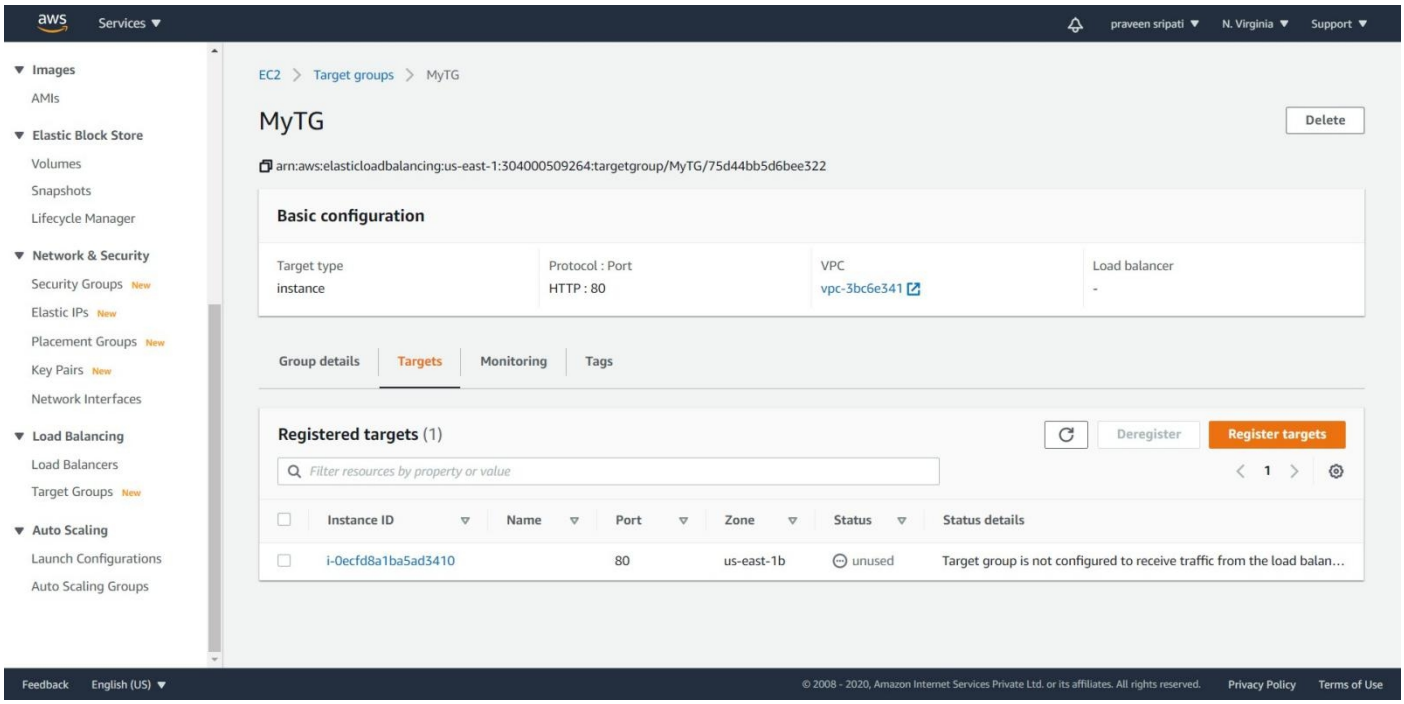


Step 7: Open the below URL in a browser and note that the AccessKeyId and the SecretAccessKey are displayed, as shown below. This happens because the program executed above will take a URL, invoke it, and send it back. There is no check on what the URL is and the internal network can be probed. Make sure to replace 34.207.208.218 with the Public IP of the EC2 instance.

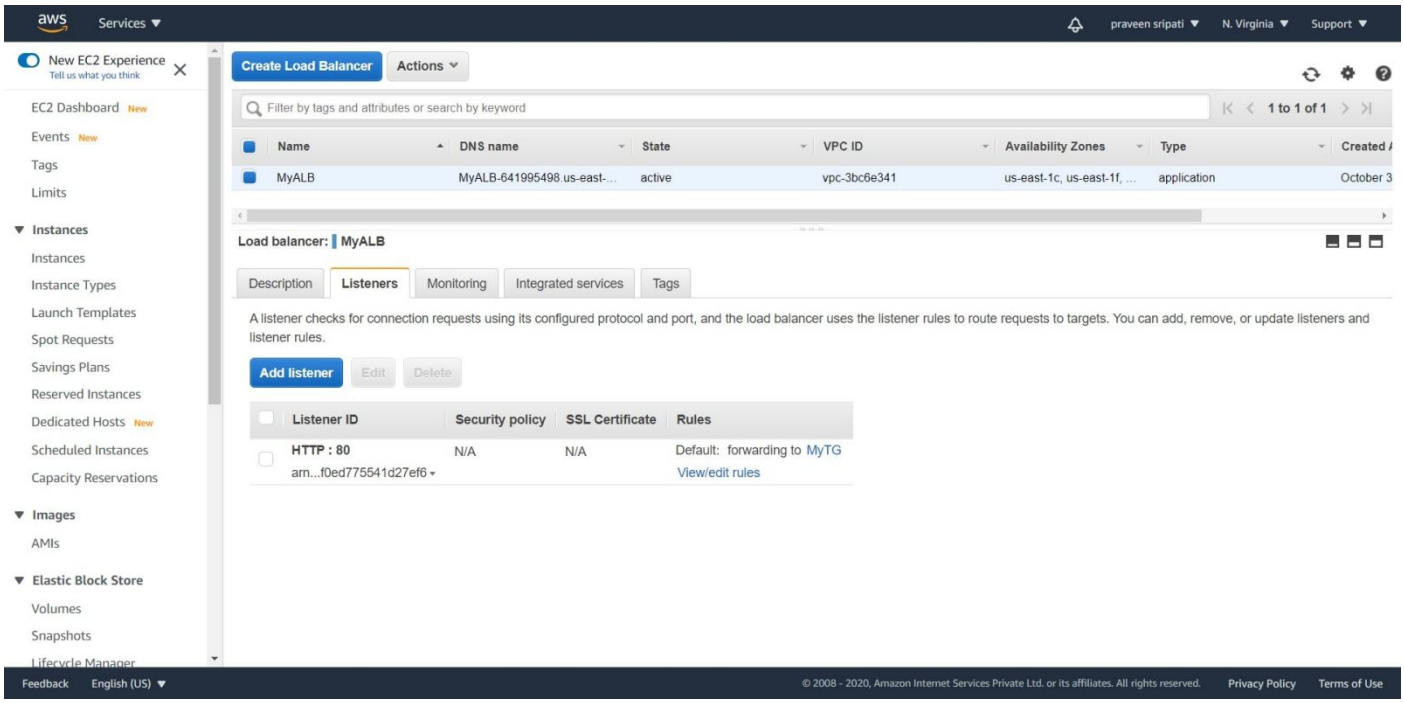
http://**34.207.208.218**?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/Role4EC2-S3RO

RESPONSE: { "Code" : "Success", "LastUpdated" : "2020-10-03T07:05:13Z", "Type" : "AWS-HMAC", "AccessKeyId" : "ASIAUNR6XUVIMD4J4AM4", "SecretAccessKey" : "K3CCCXb+htgkS0rGKu5qMwoWEt+jGQF+3wlVKVw5", "Token" : "IQoJb3JpZ2luX2VjEHgaCXVzLWVhc3QtMSJHMEUCIQDshOE1TjPPbperL5Upip+ePA0ZTW+zDyf0/re2cUeteQIgYQnNz0VeZ3PC7sFWEsKvtUXmNjXZRnFPcsgYReyulgcqvQMIoP/////////ARABGgwzMDQwMDA1MDkyNjQiDI
"Expiration" : "2020-10-03T13:41:08Z" }

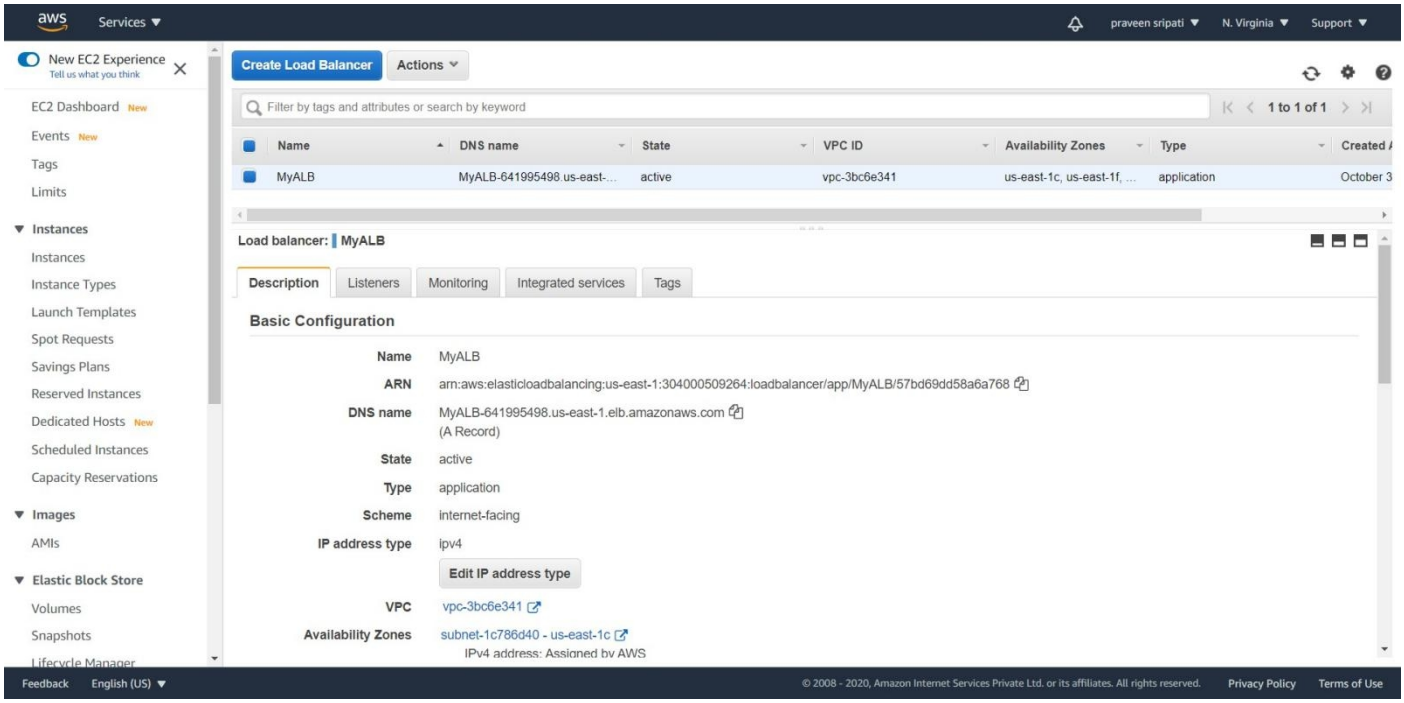
Step 8: Go to the EC2 Management Console and create a Target Group called MyTG and attach the above EC2.



Step 9: In the same EC2 Management Console, create an Application Load Balancer and specify the Target Group as the default forwarding destination, as shown below. Under the listeners Tab, it should appear, as shown below.



Step 10: Note down the DNS name of the Load Balancer.

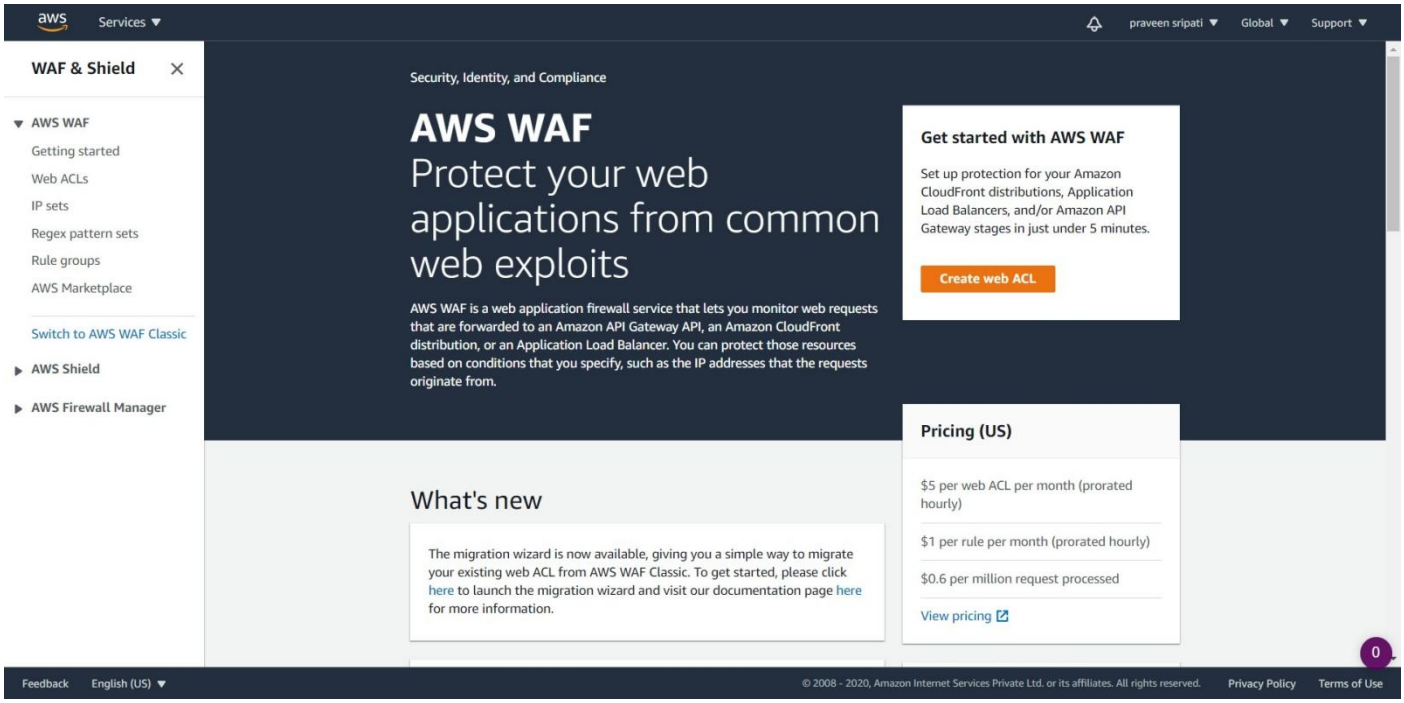


Step 11: Open the below URL in a browser. The Access Keys are displayed in the browser as there is still a vulnerability in the application. These Access Keys can be used to read the data from S3 as we have given the Read-Only permissions to the EC2 via the IAM Role. Make sure to replace elb.com with the DNS name of the Load Balancer.

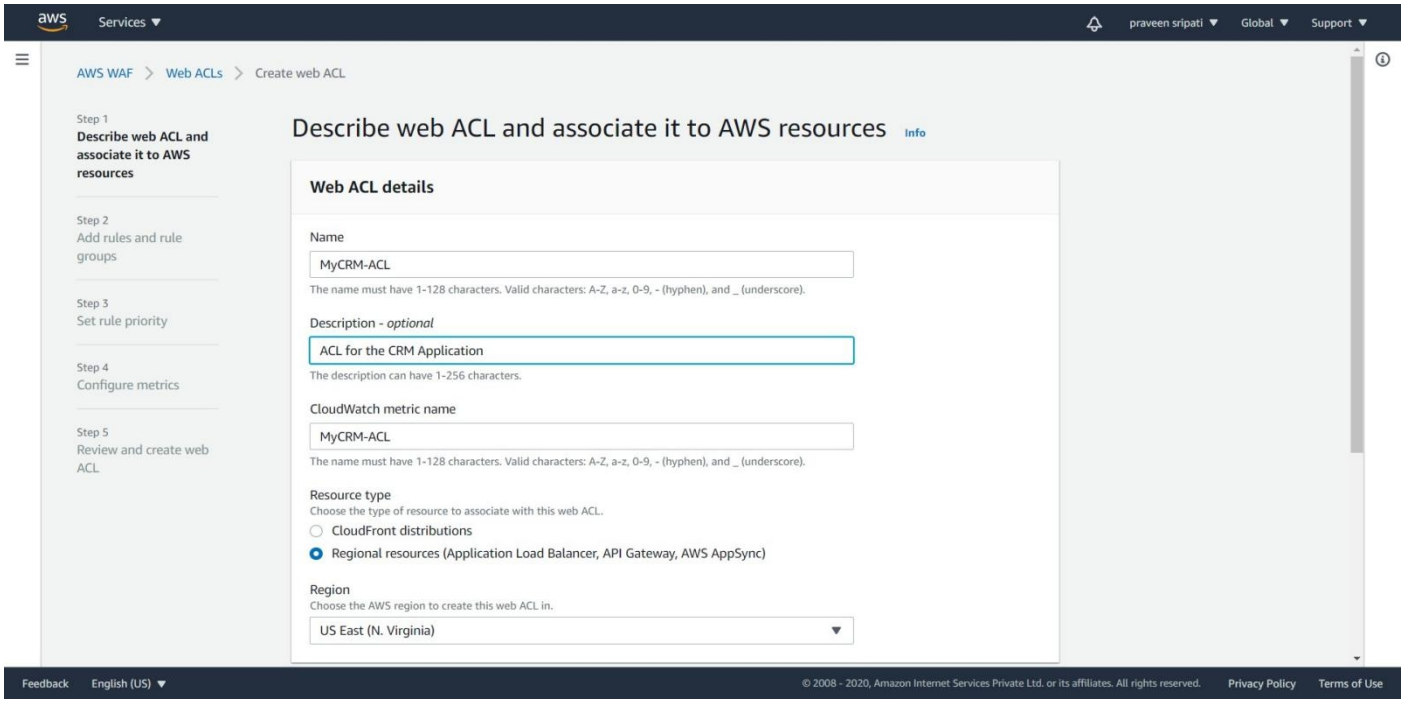
http://**elb.com**?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/Role4EC2-S3RO

RESPONSE: { "Code" : "Success", "LastUpdated" : "2020-10-03T07:25:48Z", "Type" : "AWS-HMAC", "AccessKeyId" : "ASIAUNR6XUVIOFACRCM5", "SecretAccessKey" : "IiNOrVtq2eJ4JdaqhjYGM5A7cQJxRV/VgHgH18k5",
"Token" :
"IQoJb3JpZ2luX2VjEHgaCXVzLWVhc3QtMSJHMEUCIQCgwk7axrkA7fl7msB1wr9kar2Ki95bIdchYz462qiMwwIgbfxdvbm+/MuAzIrjZTBxHfW/pm7MgiXtPpZidL+IrJ0qvQMIof/////////ARABGgwzMDQwMDA1MDkyNjQiDBPzL'
"Expiration" : "2020-10-03T13:45:48Z" }

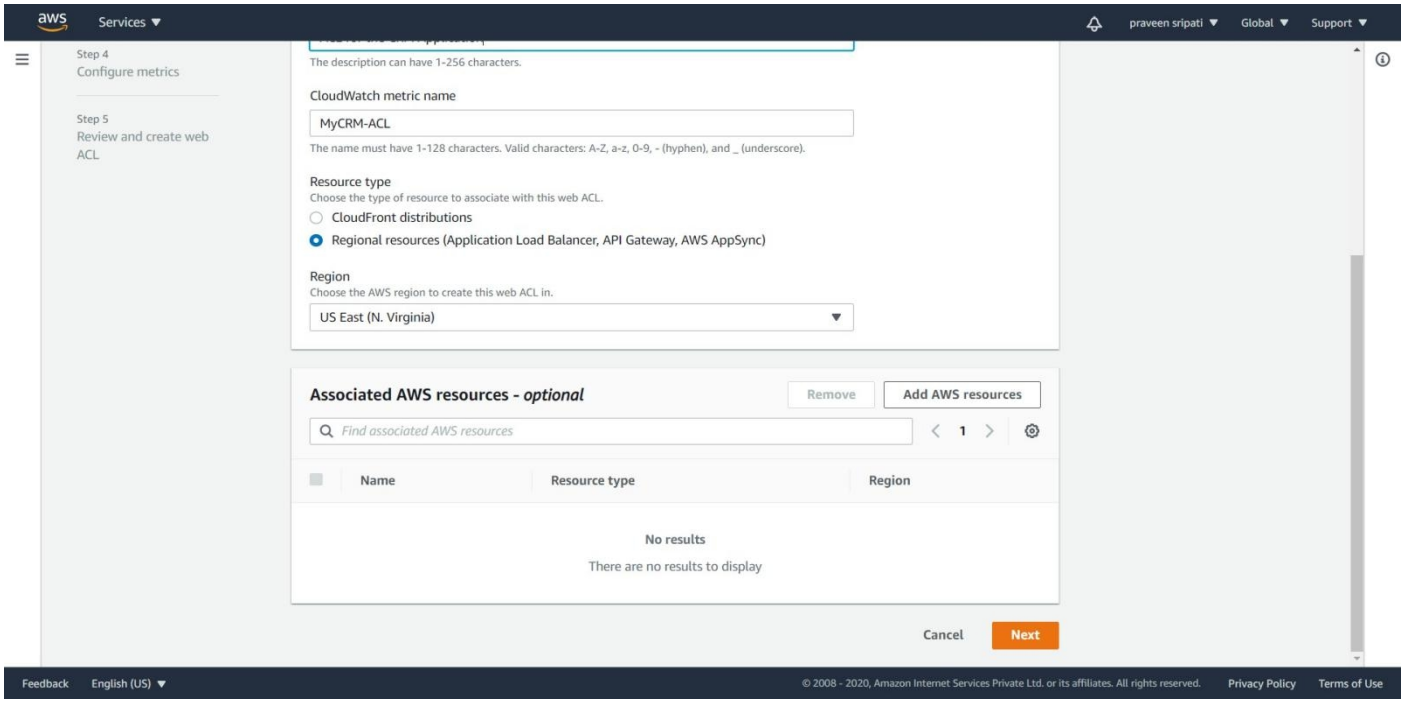
Step 12: Now, we will try to mitigate the vulnerability in the application using a WAF. Go to the WAF Management Console and click on **Create web ACL**.



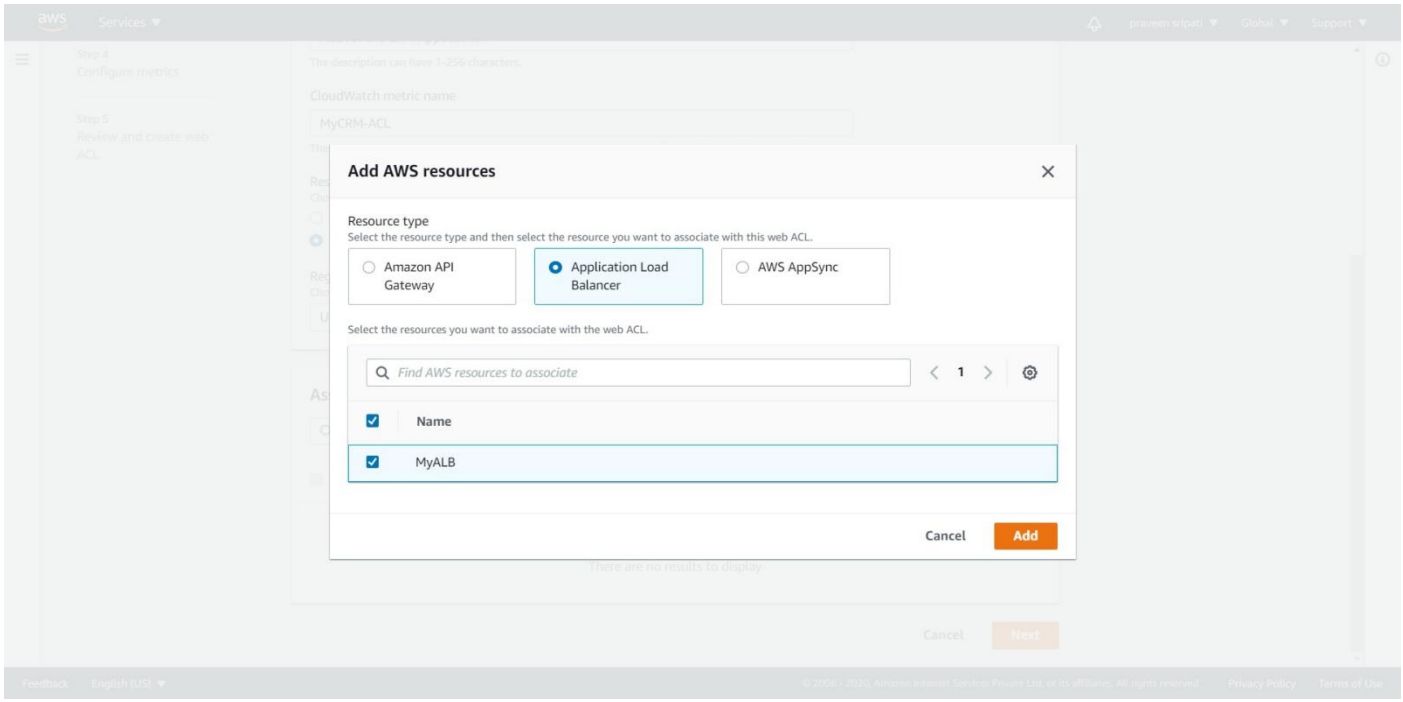
Step 13: Give a Name and provide some description, as shown below.



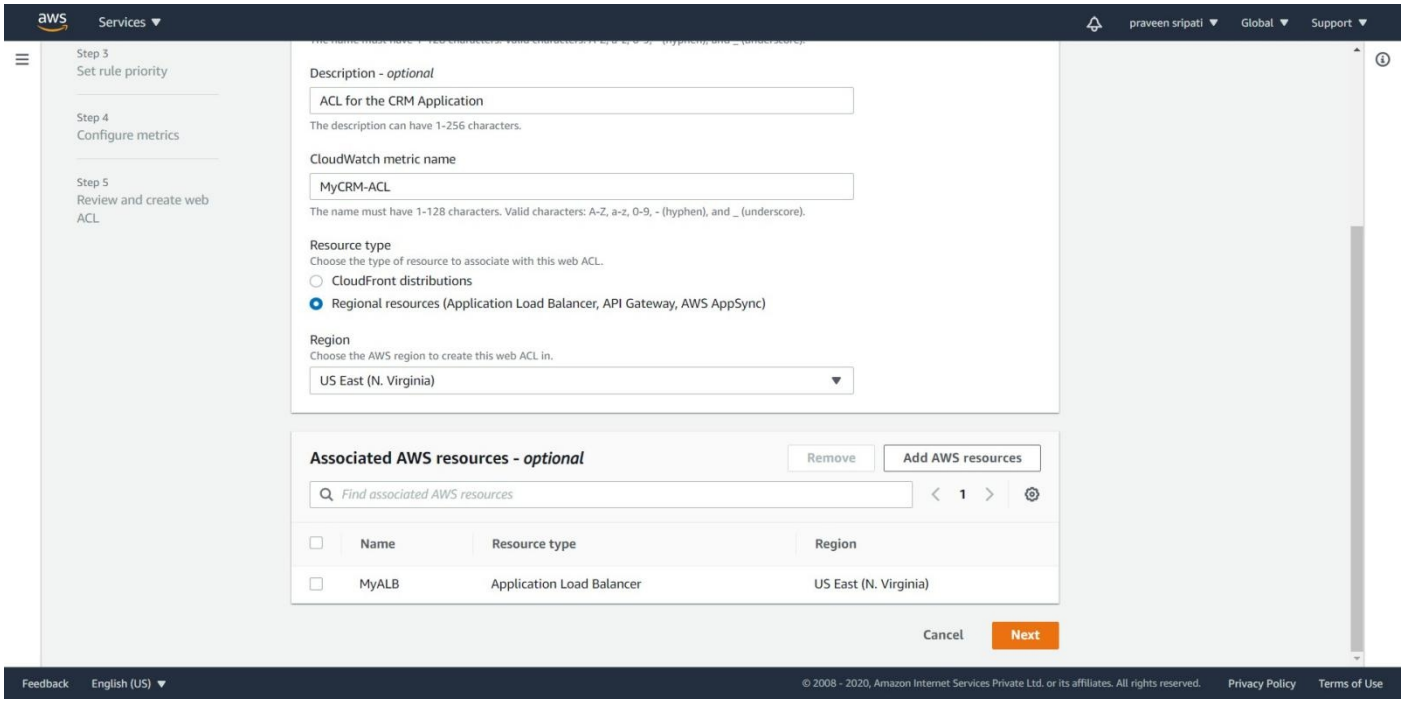
Step 14: In the same screen, click on **Add AWS resources**.



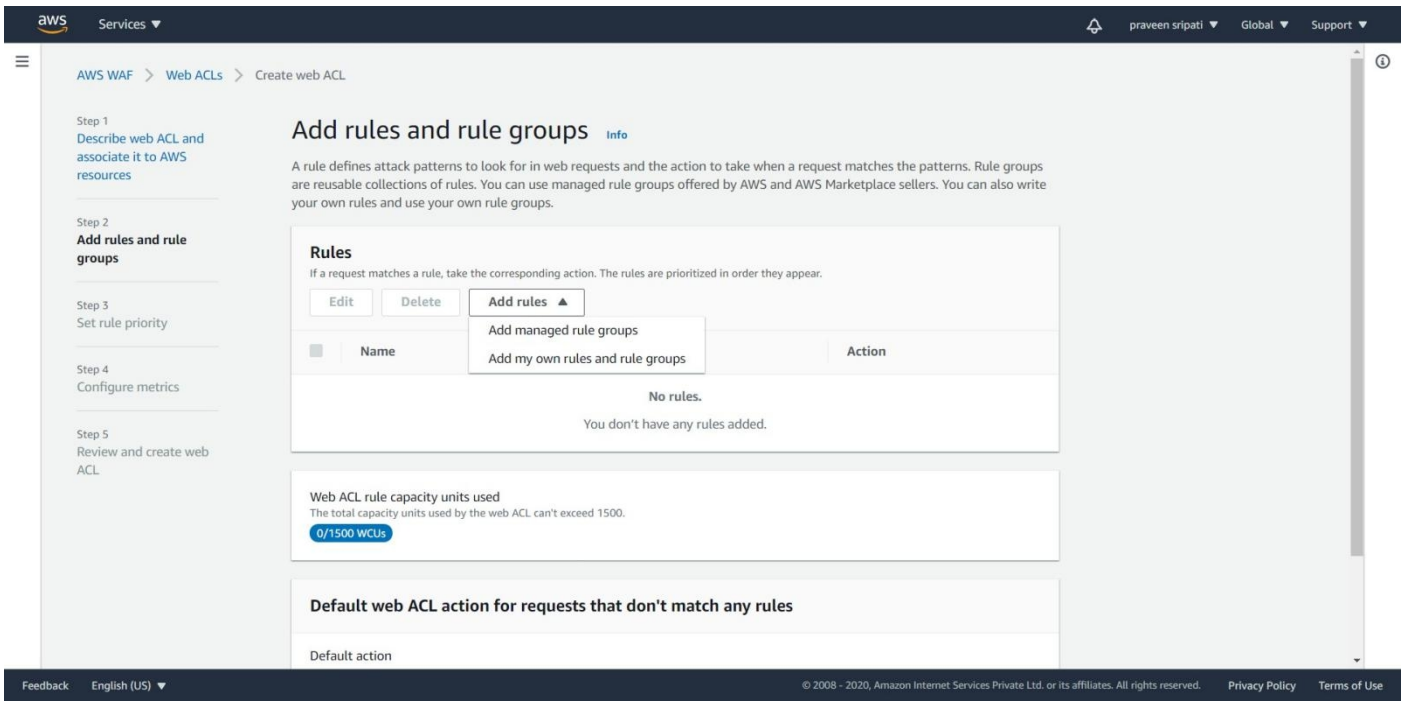
Step 15: Make sure to select **Application Load Balancer** and **MyALB** from the previously created Load Balancer from the list. Click on **Add**.



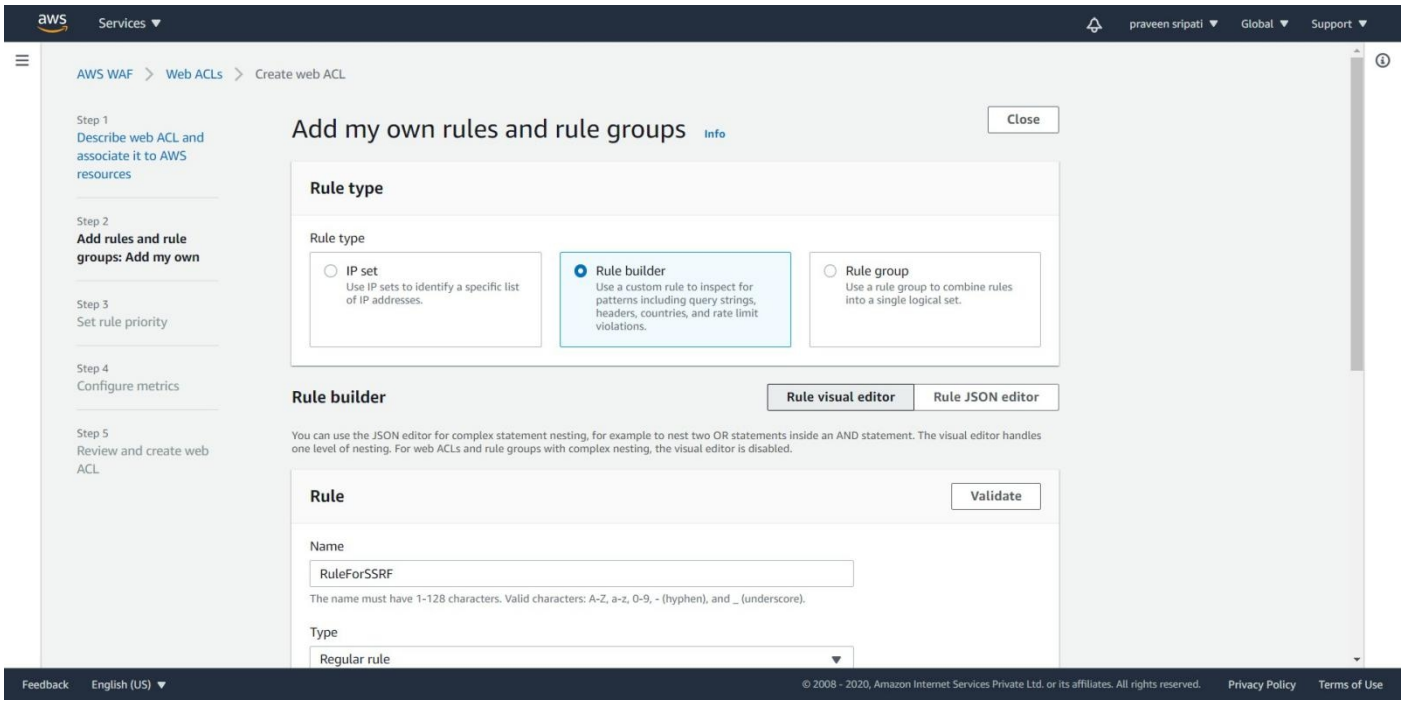
Step 16: Click on **Next**.



Step 17: Click on **Add rules** → Add my own rules and rule groups.

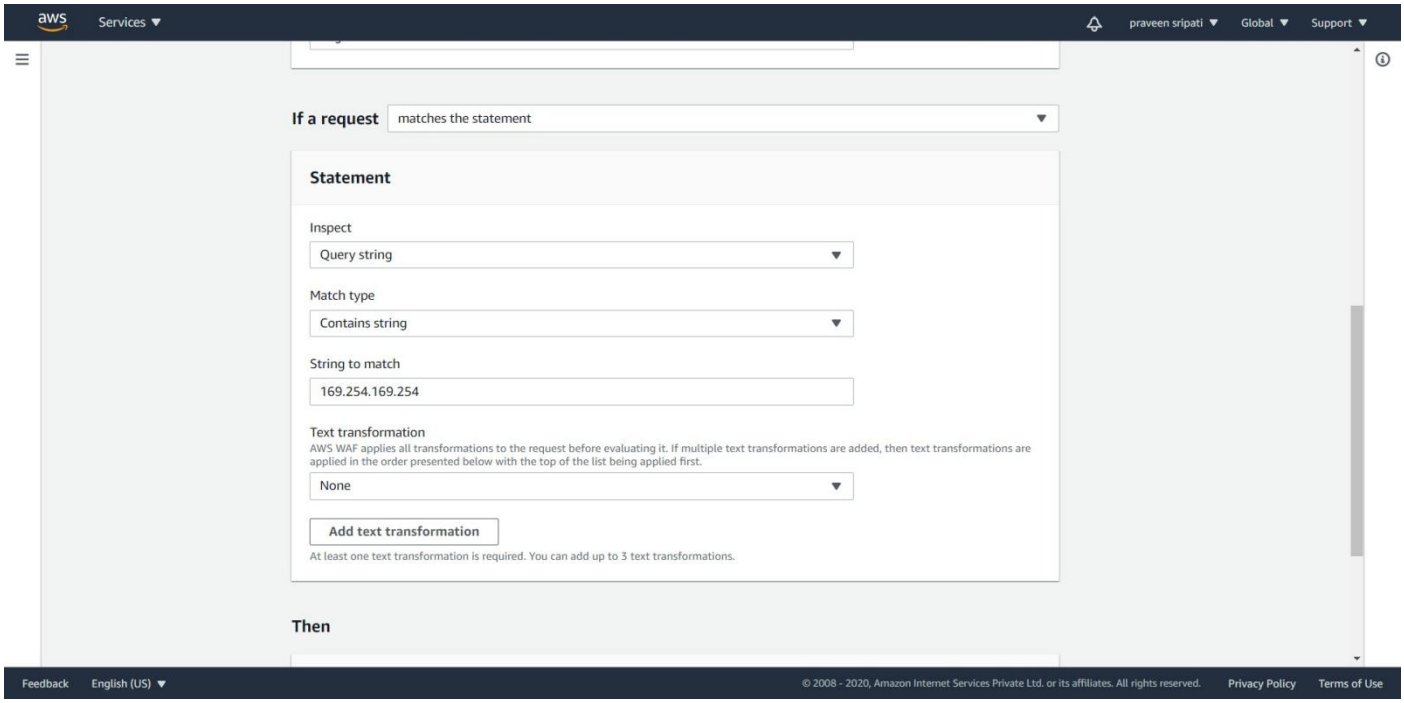


Step 18: Make sure to select the **Rule builder** and specify the rule name as **RuleForSSRF** and type of rule as **Regular rule**.

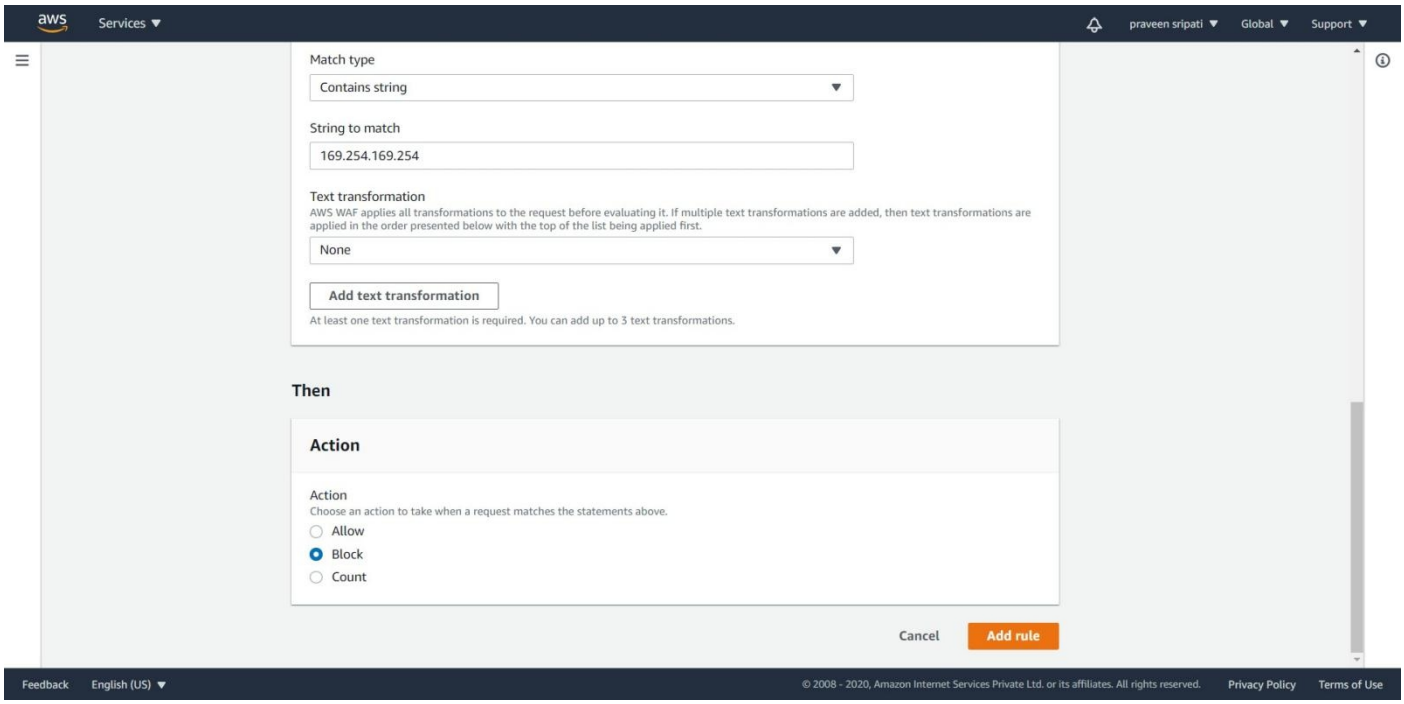


Step 19: In the **If a request**, select **matches the statement**and then select

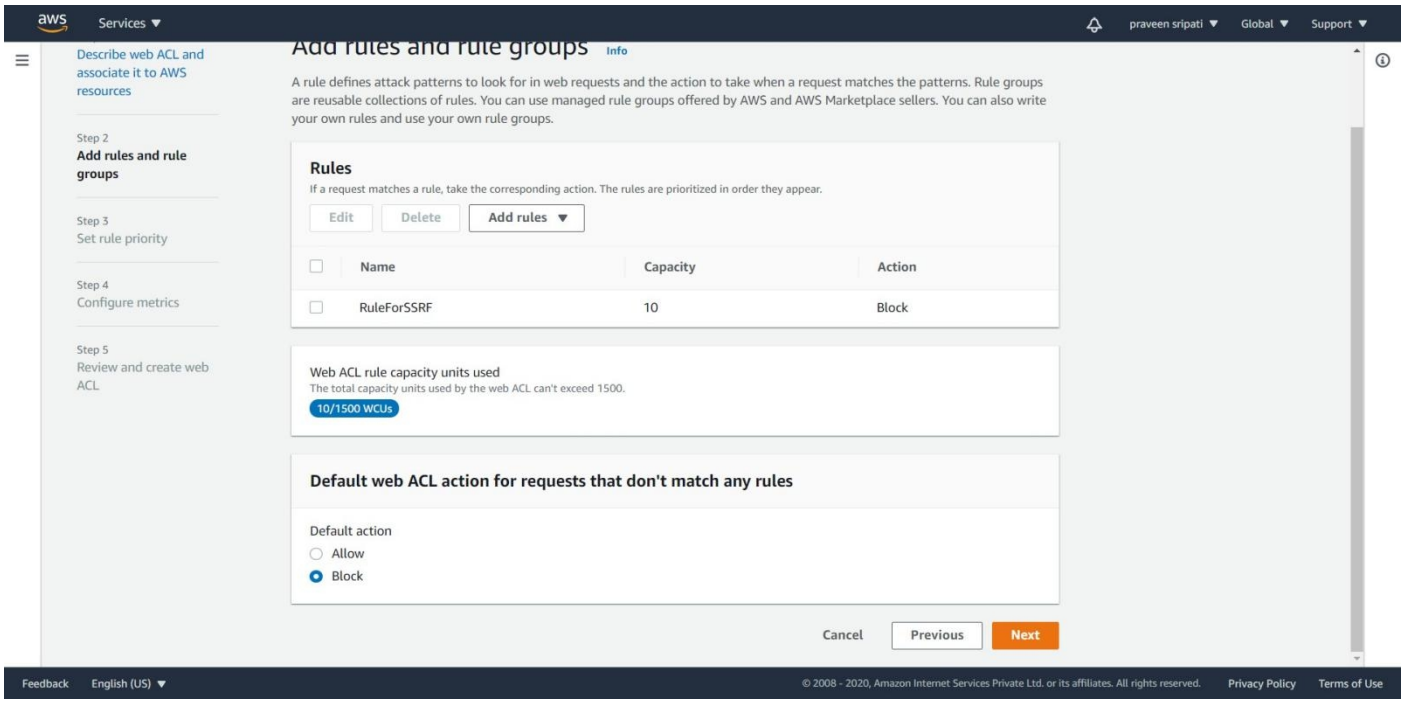
Query string for Inspect
Contains string for **Match type**
169.254.169.254 for **String to match**



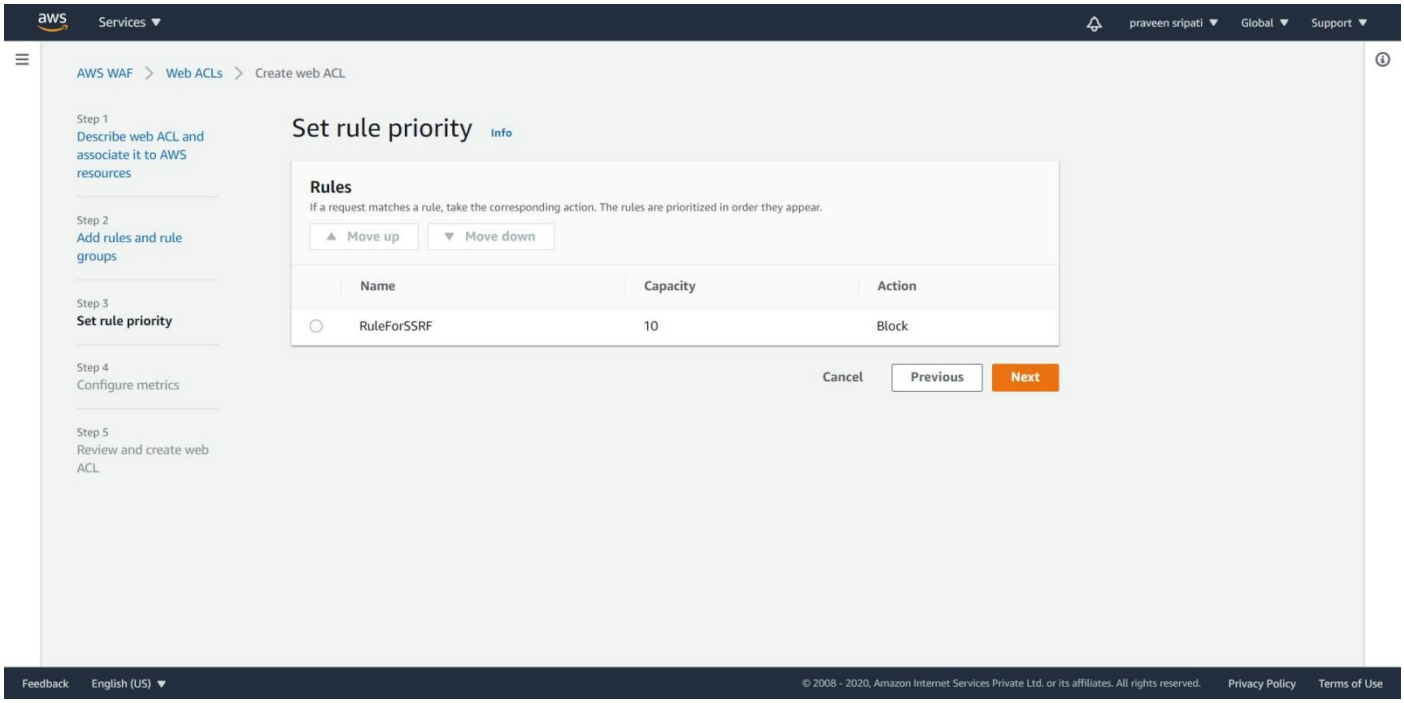
Step 20: Under the **Action**, select **Block**. Click on **Add rule**.



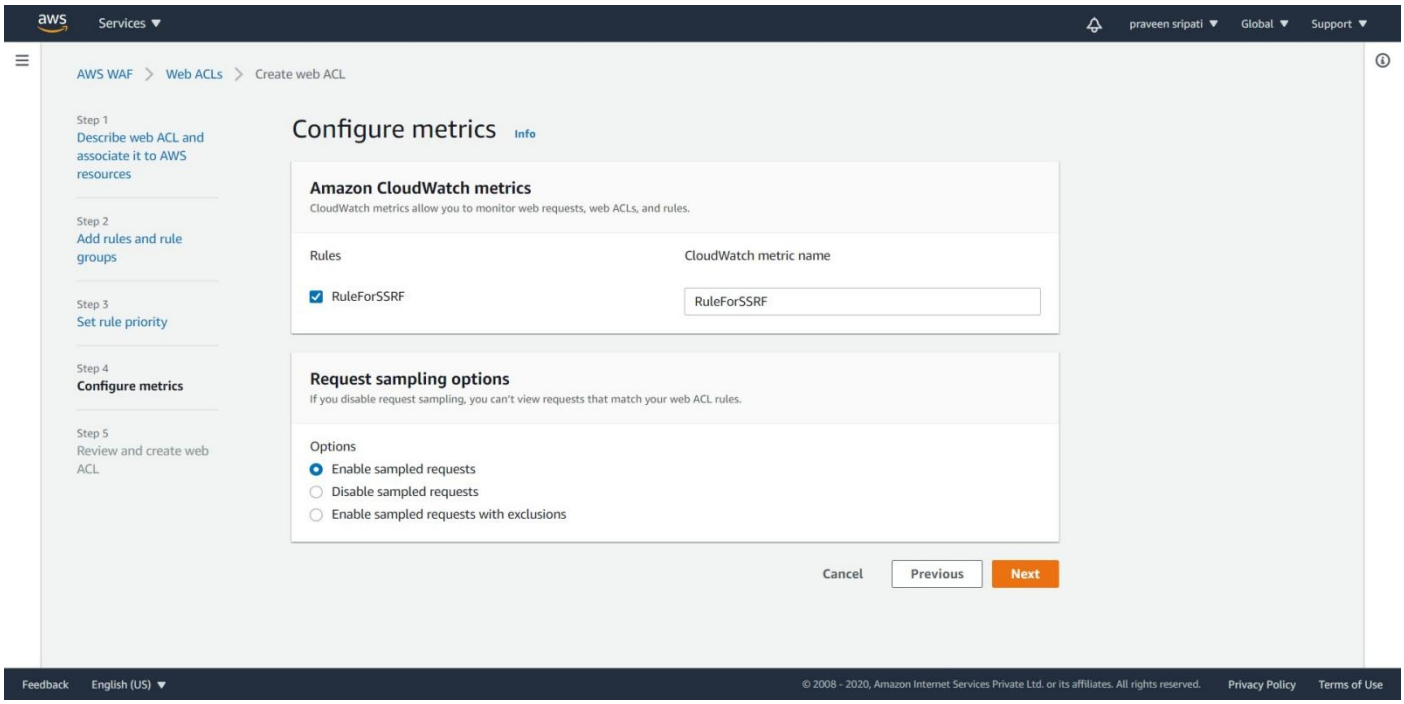
Step 21: The rule will be created, as shown below. Click on **Next**.



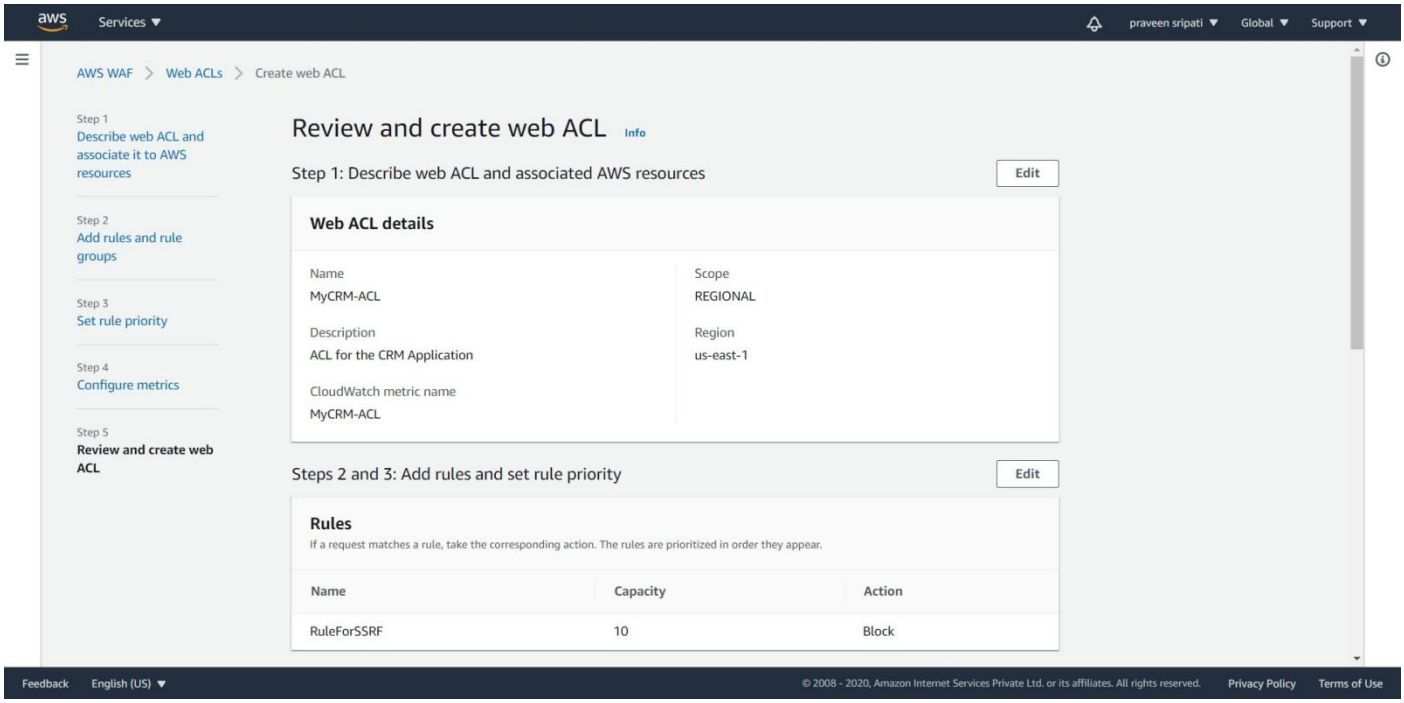
Step 22: There is only one rule that needs to be set on priority. Click on **Next**.

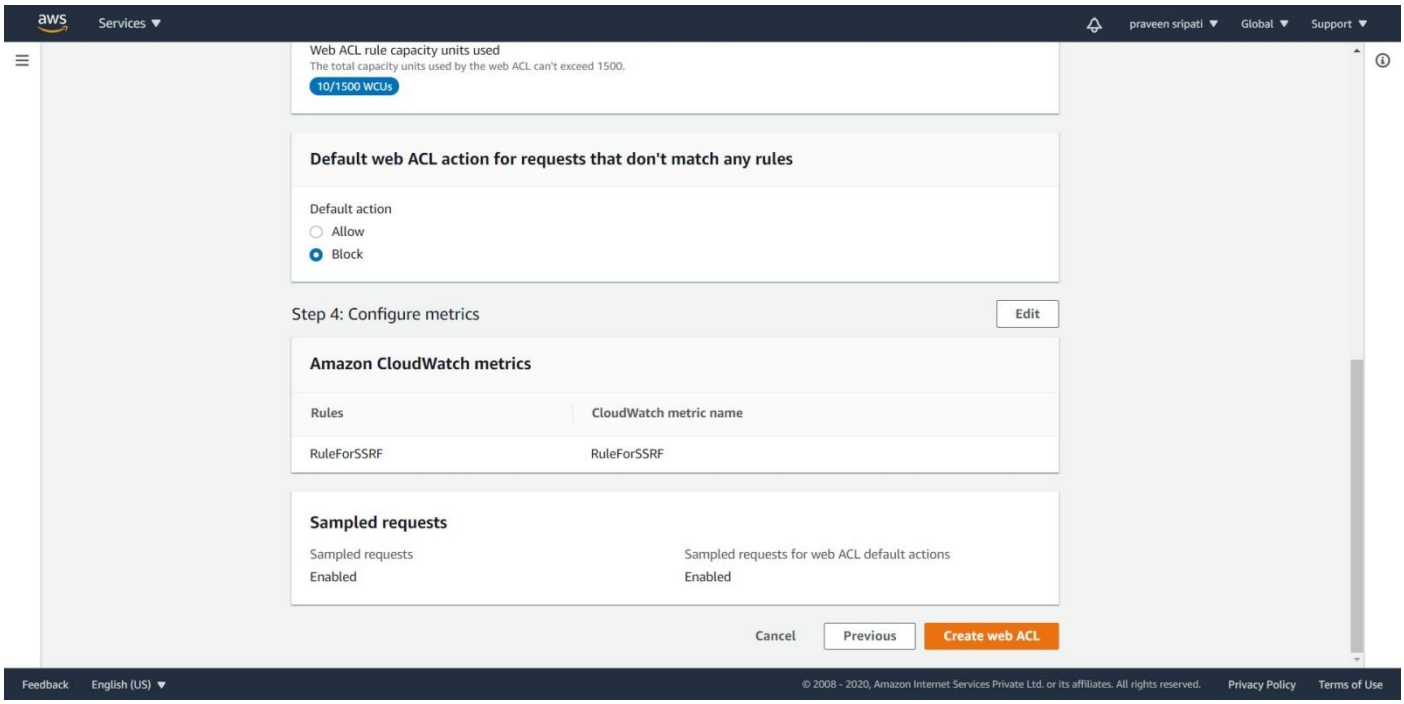


Step 23: Click on **Next**.

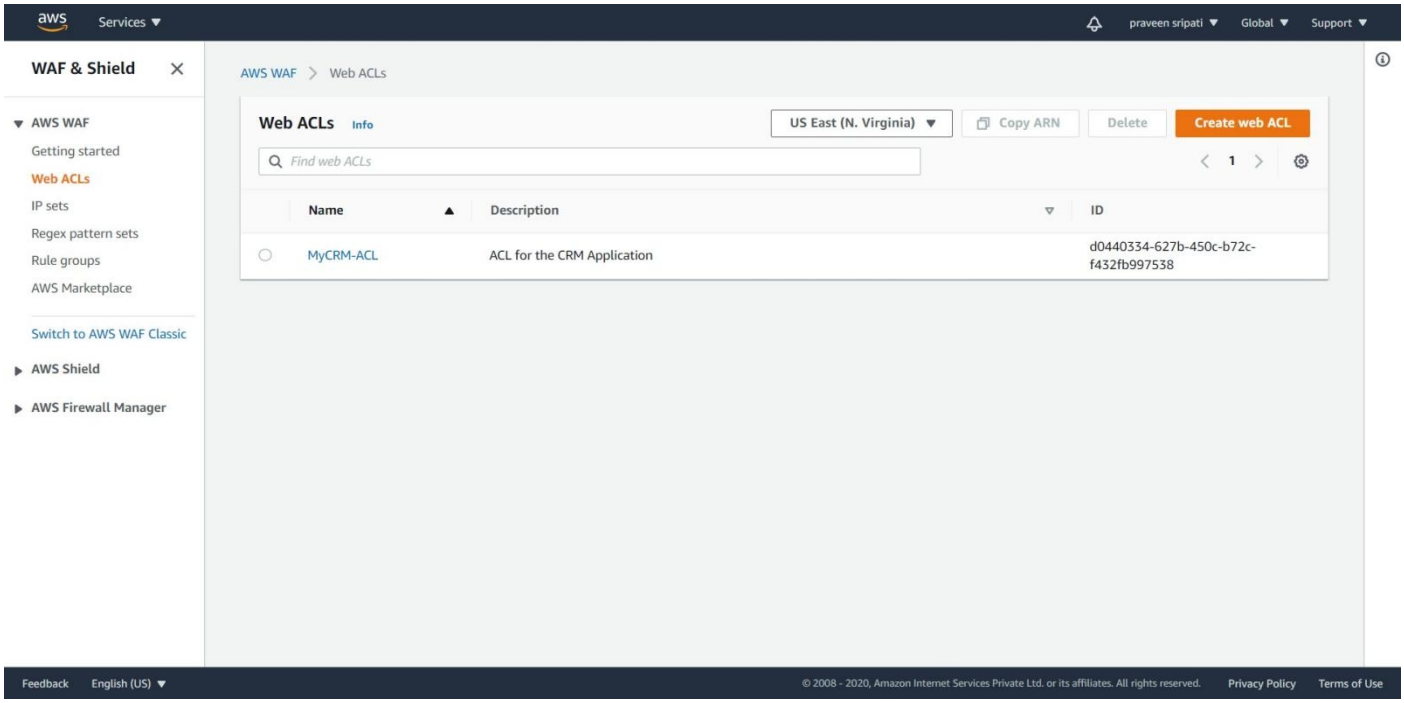


Step 24: Review the web ACL and click on **Create web ACL**.



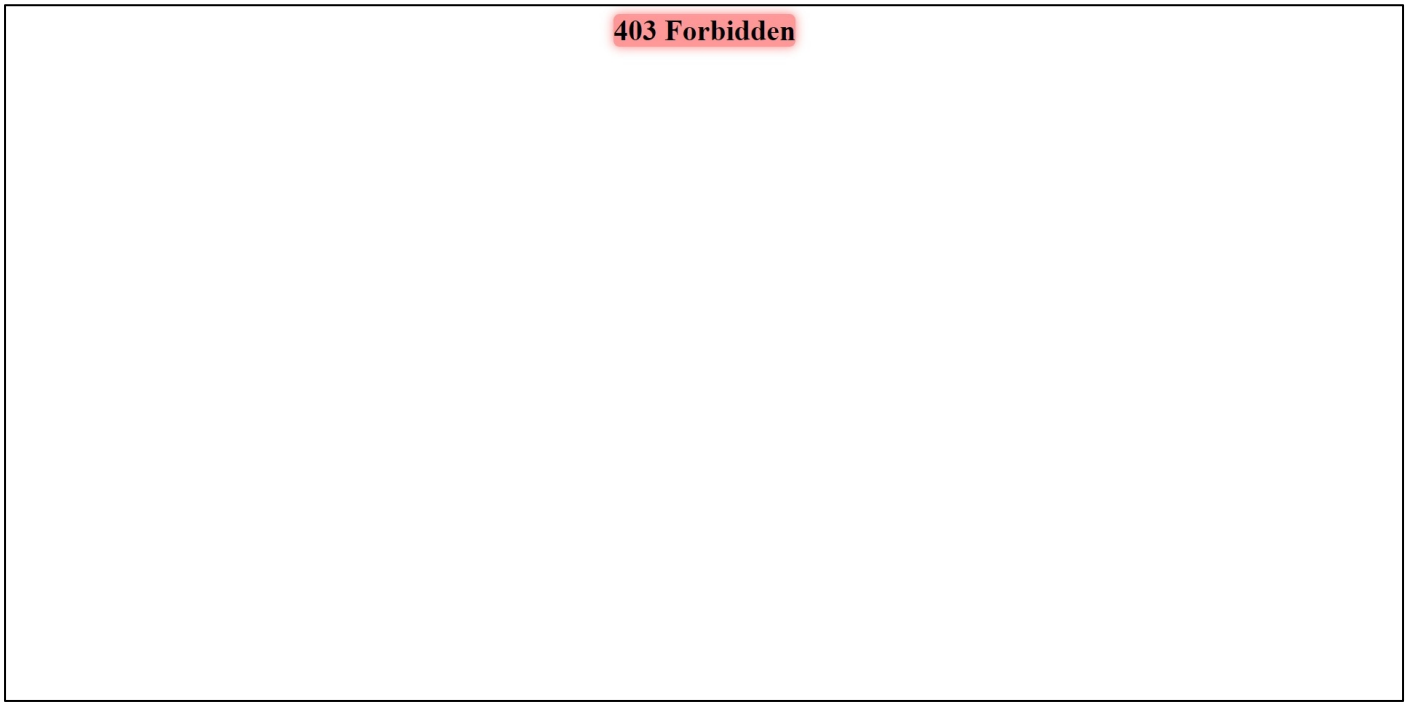


Step 25: The Web ACL would be created, as shown below.

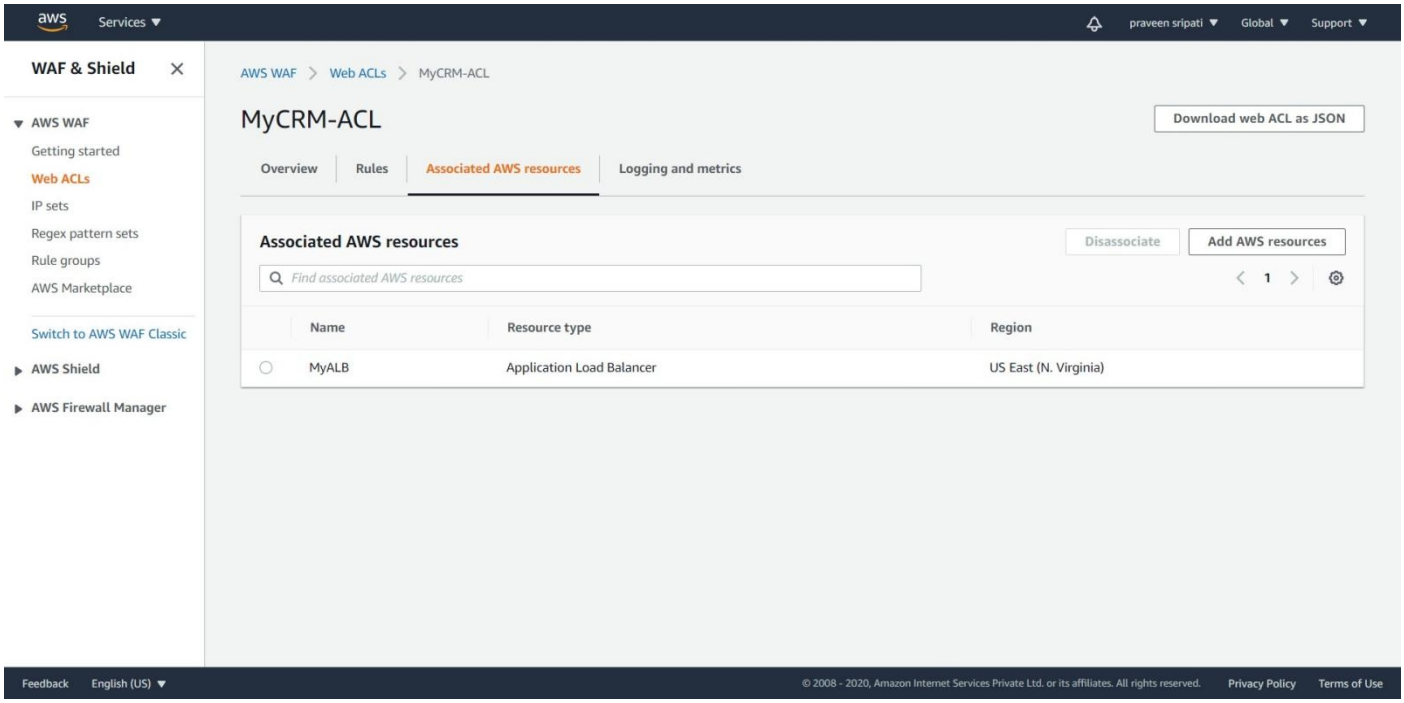


Step 26: Open the below URL in the browser, and **403 Forbidden** is displayed in the browser. This happens because 169.254.169.254 is there the URL and this is blocked by the WAF. Make sure to replace elb.com with the DNS name of the Load Balancer.

http://**elb.com**?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/Role4EC2-S3RO



Step 27: FYI, before deleting the ACL, we need to make sure that the **Associated AWS resources** are deleted, or else AWS won't allow the ACL to be deleted.



Conclusion:

We had a vulnerability in the application, and it was displaying the Access Keys in the browser, which can be exploited to get the data from S3. Although there is still a vulnerability in the application, we mitigated it using a WAF custom rule.