



Group 30 Members:

Bejjagam Lokesh
Chikkala Veera Sairam
Syed Saif Ahmed

Project Report

I - Description of the Problem Addressed

We were required to deploy a web app onto an online/virtual portal. This virtual platform should allow us to monitor our application and make updates if and when required. It should not allow our application to collapse due to an increase in load. We should also be able to keep an eye on our expenditure as we go.

II - Proposed Solution

Our proposal revolved around the **Deployment of a Scalable Web-Application on a Cloud Infrastructure**. The cloud technology in question would be AWS's **Elastic BeanStalk**. We chose this service because it would fulfil the following requirements.

>> **Deployment of a Web Application:** To deploy a web app we can either upload an already existing static website, or we can create a simple website using either **WordPress** or **Wix**.

>> **Self-Configuration:** When we upload our application, Elastic Beanstalk will, by default set up an environment suited for our application on its own. We can of course change the configurations whenever we want.

>> **AutoScaling Health Check:** By default, the Auto Scaling group created for your environment uses Amazon EC2 status checks. On AWS's official page it says, "If an instance in your environment fails an Amazon EC2 status check, Auto Scaling takes it down and replaces it." Using Auto-scaling within Elastic Beanstalk makes the application dynamically scalable.

>> **Load-Balancer:** When we enable load balancing, AWS Elastic Beanstalk creates an Elastic Load Balancing load balancer dedicated to our environment. Elastic Beanstalk fully manages this load balancer, taking care of security settings and of terminating the load balancer when we terminate our environment. This solves a lot of performance issues and in general, will reduce stress on the web app.

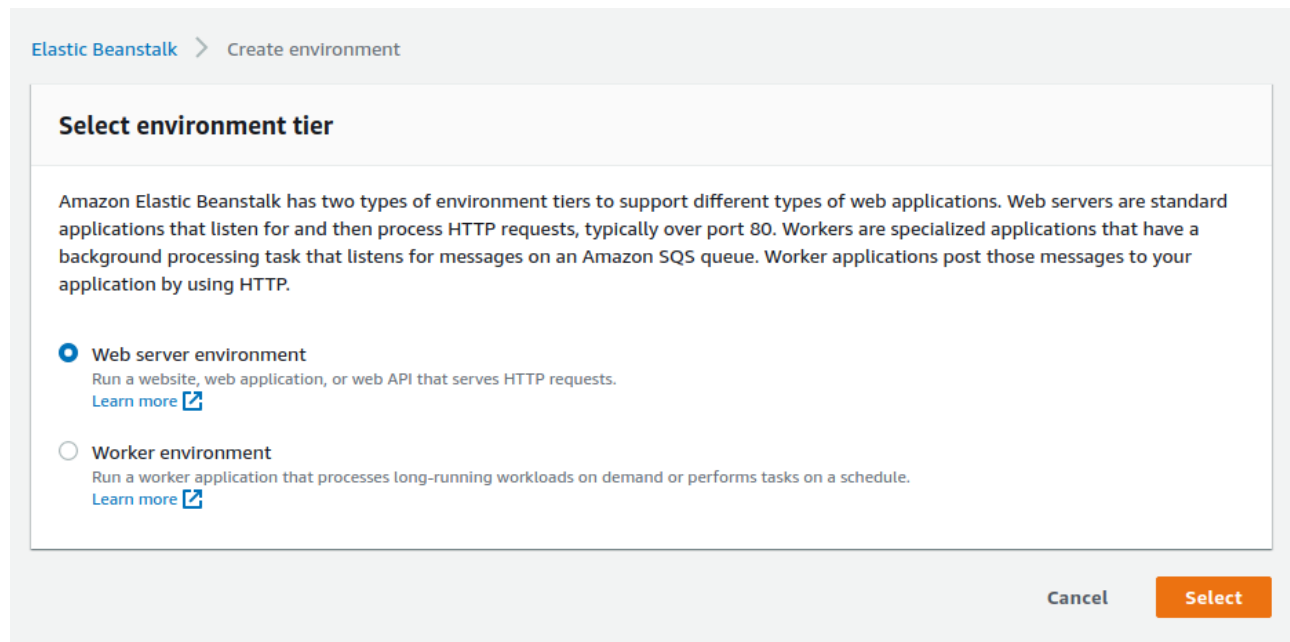
Using the above-mentioned services/functionalities and more we were able to complete the implementation part of our project.

III - Implementation of the Proposed Solution

To implement the solution we had to go through the following steps:

>> First of all we had to create an AWS account and make sure to stay as close to the “free tier” options as possible.

>> Once we’re on the console, we search for the Elastic BeanStalk service and select “Create new environment”. Here we select the Web server environment because that’s the best suited for our web app.



>> Then we are taken to the next page where we choose a name and platform. The environment name is auto-assigned. Since our code is HTML based we choose PHP as our platform since it is compatible with it.

>> Our final step for deploying the web app is to upload the source code bundle and provide a relative version name.

>> We also have an option to configure the env ourselves. But there is no harm in letting Elastic BeanStalk do it for it. This is where it uses its **Self Configuration** property and automatically decides what environment would be best suited for our application. We can of course change the configurations whenever we want, after the deployment.

>> After the source file has been uploaded with/without further configurations we can click the “Create Environment” button.

The screenshot shows the 'Create environment' dialog box in the AWS Lambda console. It has two main sections: 'Existing version' and 'Upload your code'. The 'Existing version' section is inactive, showing a dropdown menu with the text '-- Choose a version --'. The 'Upload your code' section is active, indicated by a blue radio button. It contains a 'Version label' field with the text 'blackweb-v1.0', a 'Source code origin' section with 'Local file' selected, a 'Choose file' button, and a 'File name' field showing 'blackweb.zip'. Below this, there is a green checkmark icon and the text 'File successfully uploaded'. At the bottom of the dialog, there are three buttons: 'Cancel', 'Configure more options', and 'Create environment'.

☐ Existing version
Application versions that you have uploaded for **BlackWeb**.

-- Choose a version --

☒ Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

Version label
Unique name for this version of your application code.

blackweb-v1.0

Source code origin
Maximum size 512 MB

☒ Local file

☐ Public S3 URL

Choose file

File name : **blackweb.zip**

File successfully uploaded

Application code tags

Cancel Configure more options Create environment

>> Once the creation of the env has started, it may take a while before the environment is up and running. We get a terminal looking screen that shows us the progress and logs reports of the env during creation. It looks something like this

```

9:47pm Instance deployment completed successfully.

9:47pm Instance deployment: You didn't include a 'composer.json' file in your source bundle. The deployment didn't install Composer dependencies.

9:47pm Created CloudWatch alarm named:
awseb-e-ptsfgkn4vf-stack-AWSEBCloudwatchAlarmLow-GLD5VJPU7VAU

9:47pm Created CloudWatch alarm named:
awseb-e-ptsfgkn4vf-stack-AWSEBCloudwatchAlarmHigh-F16HC7FKXHCN

9:47pm Created Auto Scaling group policy named:
arn:aws:autoscaling:us-east-2:997422096465:scalingPolicy:f7839bc0-da1f-4f96-b5a7-34d3a06f50d4:autoScalingGroupName/awseb-e-ptsfgkn4vf-stack-AWSEBAutoScalingGroup-QFYL31ZQ201L:policyName/awseb-e-ptsfgkn4vf-stack-AWSEBAutoScalingScaleDownPolicy-1SJDJI45N7HZM

9:47pm Created Auto Scaling group policy named:
arn:aws:autoscaling:us-east-2:997422096465:scalingPolicy:c3cb8334-398a-4f21-bfba-960886eaac58:autoScalingGroupName/awseb-e-ptsfgkn4vf-stack-AWSEBAutoScalingGroup-QFYL31ZQ201L:policyName/awseb-e-ptsfgkn4vf-stack-AWSEBAutoScalingScaleUpPolicy-4H18987VTH8H

9:47pm Created Load Balancer listener named:
arn:aws:elasticloadbalancing:us-east-2:997422096465:listener/app/awseb-AWSEB-1BGZRLO2QSSVA/73e84cab4b32f25/bed0aabd1811e1ba

9:47pm Waiting for EC2 instances to launch. This may take a few minutes.

9:47pm Created Auto Scaling group named:
awseb-e-ptsfgkn4vf-stack-AWSEBAutoScalingGroup-QFYL31ZQ201L

9:47pm Created Load Balancer listener named:
arn:aws:elasticloadbalancing:us-east-2:997422096465:listener/app/awseb-AWSEB-1BGZRLO2QSSVA/73e84cab4b32f25/bed0aabd1811e1ba

9:47pm Waiting for EC2 instances to launch. This may take a few minutes.

9:47pm Created Auto Scaling group named:
awseb-e-ptsfgkn4vf-stack-AWSEBAutoScalingGroup-QFYL31ZQ201L

9:47pm Created load balancer named:
arn:aws:elasticloadbalancing:us-east-2:997422096465:loadbalancer/app/awseb-AWSEB-1BGZRLO2QSSVA/73e84cab4b32f25

9:45pm Environment health has transitioned to Pending. Initialization in progress (running for 13 seconds). There are no instances.

9:45pm Created security group named:
awseb-e-ptsfgkn4vf-stack-AWSEBSecurityGroup-1F146F8IQS1SI

9:45pm Created security group named:
sg-0e61f480b4bf00061

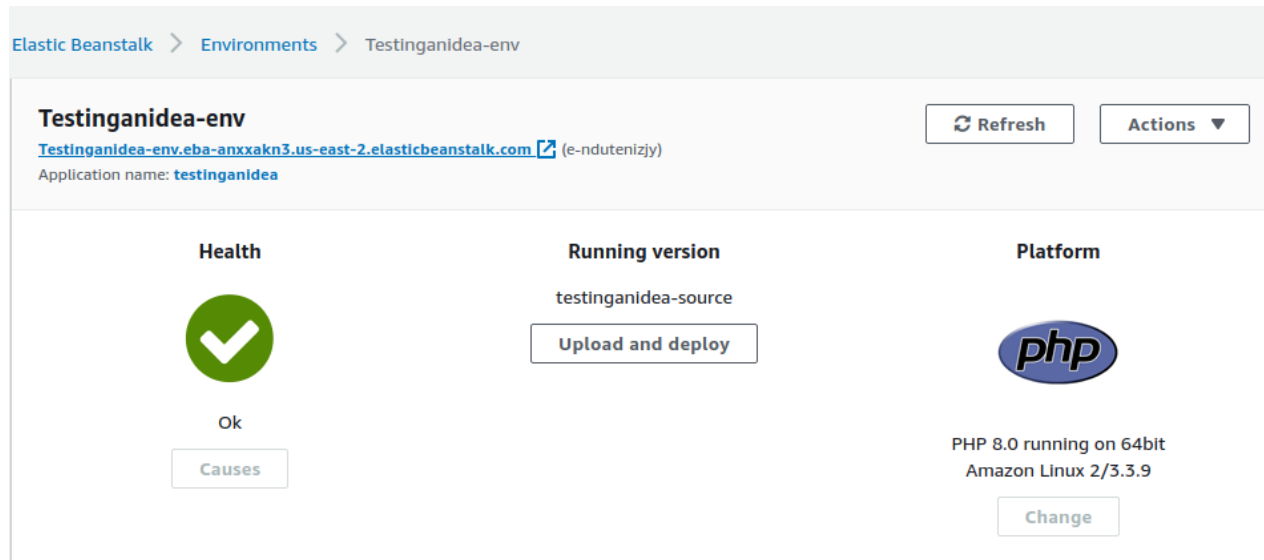
9:45pm Created target group named:
arn:aws:elasticloadbalancing:us-east-2:997422096465:targetgroup/awseb-AWSEB-1O8Q131PNM9GK/1571a2a9b722ef0e

9:45pm Using elasticbeanstalk-us-east-2-997422096465 as Amazon S3 storage bucket for environment data.

9:45pm createEnvironment is starting.

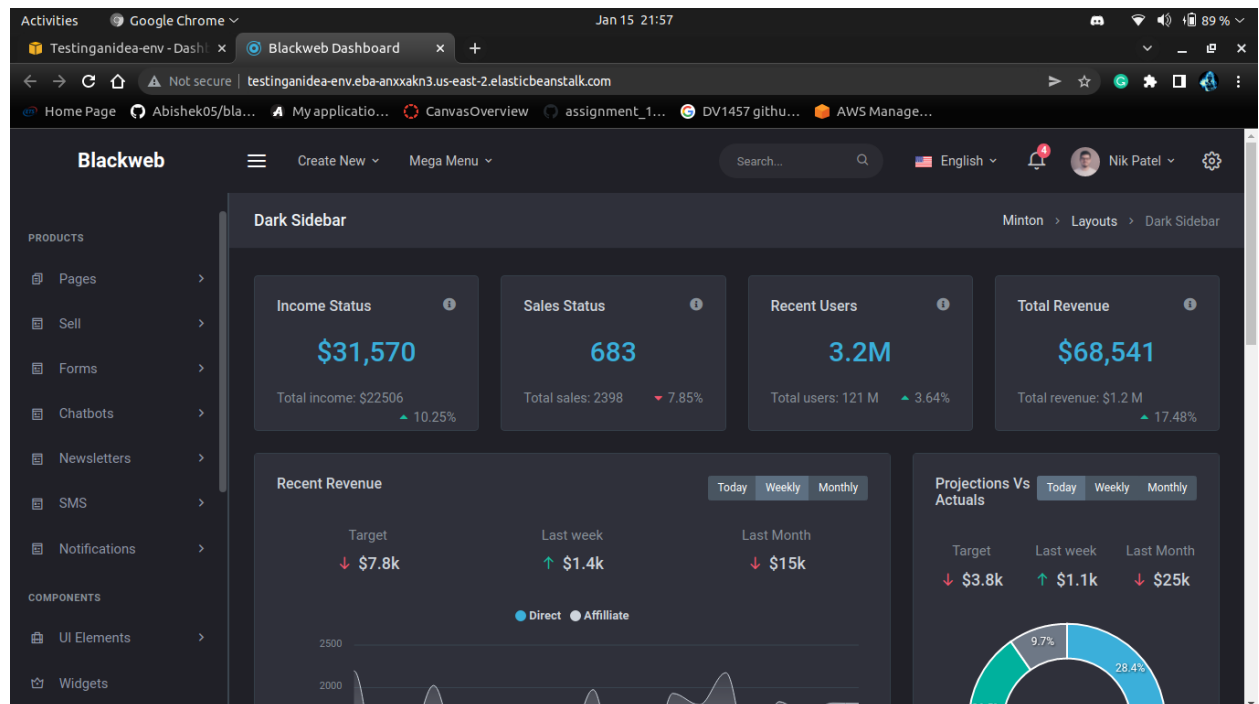
```

>> Once the creation has been completed successfully we get a screen with all the environment details and link that leads to the web app.



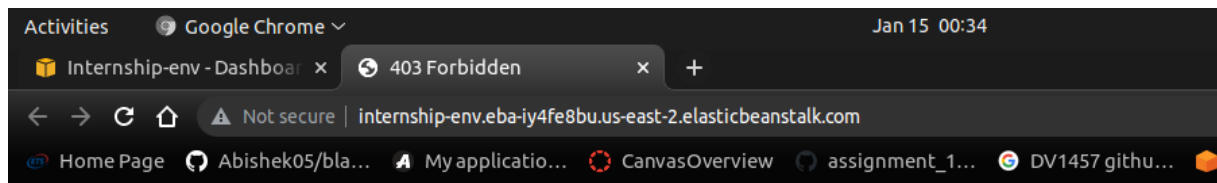
When we click on the link under “Testinganidea-env” we are led to our website which we wanted to host on AWS Elastic BeanStalk.

With this, we can conclude the Deployment part of our project as we have successfully deployed a Website (Static) on an AWS service called Elastic Beanstalk.



Few errors occurred during deployment.

In our initial deployments, we were getting multiple errors which were either “503 Bad Gateway” or “403 Forbidden”.



403 Forbidden

nginx/1.20.0

And we would also get sudden drops in health without any load put on it. And it would abort update operation.

Elastic Beanstalk > Environments > Blackweb-env

Blackweb-env
Blackweb-env.eba-cn7gkspz.us-east-2.elasticbeanstalk.com (e-mx4khgmud)
Application name: **BlackWeb**

RefreshActions ▾

Health

Severe
Causes

Running version
blackweb-source
Upload and deploy

Platform

PHP 8.0 running on 64bit
Amazon Linux 2/3.3.9
Change

Time	Type	Details
2022-01-14 18:44:39 UTC+0100	INFO	Environment health has transitioned from Warning to Ok.
2022-01-14 18:39:39 UTC+0100	WARN	Environment health has transitioned from Ok to Warning. Application update is aborting (running for 32 seconds).
2022-01-14 18:39:05 UTC+0100	ERROR	Failed to deploy application.
2022-01-14 18:39:05 UTC+0100	ERROR	Creating security group failed Reason: The vpc ID 'vpc-XXXXXXX' does not exist (Service: AmazonEC2; Status Code: 400; Error Code: InvalidVpcID.NotFound; Request ID: 33df59e2-ce65-4712-bc2f-6a9079392605; Proxy: null)
2022-01-14 18:39:05 UTC+0100	ERROR	Creating security group failed Reason: The vpc ID 'vpc-XXXXXXX' does not exist (Service: AmazonEC2; Status Code: 400; Error Code: InvalidVpcID.NotFound; Request ID: 12800652-a03f-4129-b98d-fc8bb313aeb; Proxy: null)

We went crazy trying to figure out what the issue was and why this was happening. As it turned out, the solution to this issue was very simple. There was nothing wrong with the configurations or the Elastic

Beanstalk. The problem was that our landing page on the website was named “Dashboard.html” and that caused chaos. Apparently, EBS requires that the first page of the website be called “Index.html”. As soon as we changed the name and tried again... Voila! It worked.


Elastic Beanstalk > Environments > Testinganidea-env

Testinganidea-env

[Testinganidea-env.eba-anxxakn3.us-east-2.elasticbeanstalk.com](#) (e-ndutenizjy)
Application name: **testinganidea**

[Refresh](#) [Actions](#)

Health



Ok


[Causes](#)

Running version

testinganidea-source

[Upload and deploy](#)

Platform



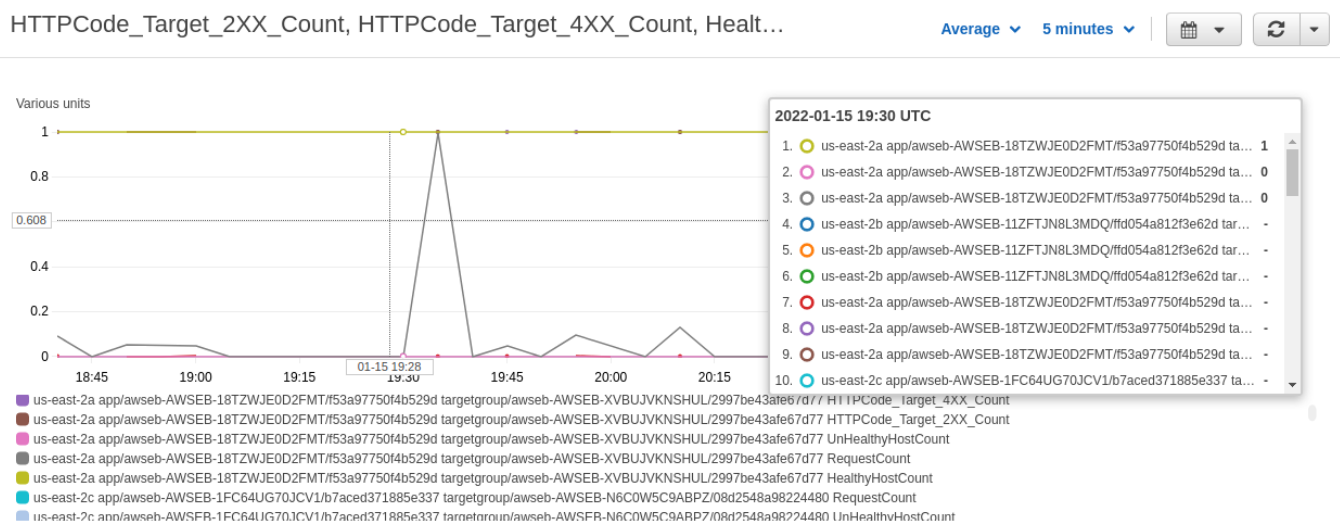
PHP 8.0 running on 64bit
Amazon Linux 2/3.3.9

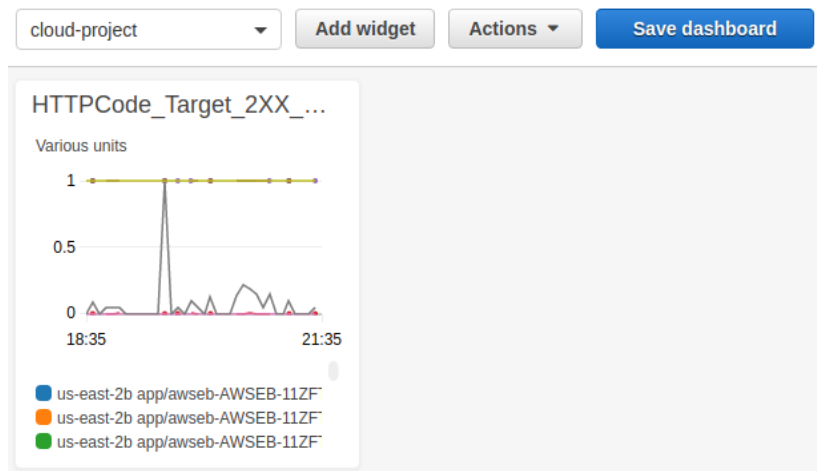
[Change](#)

IV - Test Validations & Results

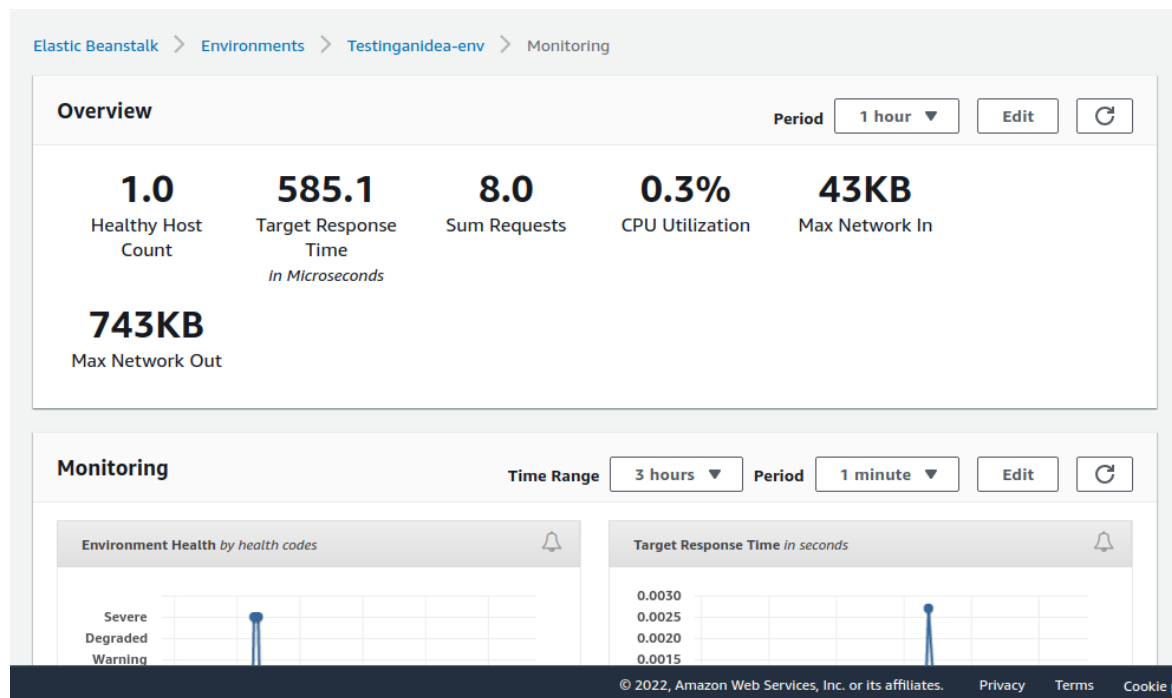
>> The main aim of a virtually hosted and deployed web server is that we should be able to monitor that environment whenever required. Fortunately in AWS, we are provided with 2 options to monitor the real-time performance of our website/environment.

One is via CloudWatch where we can monitor literally everything about our website and its performance. Here we can have a look at the detailed nerdy stats of everything or just keep a few relative metric stats under light by adding them to the dashboard as a widget.

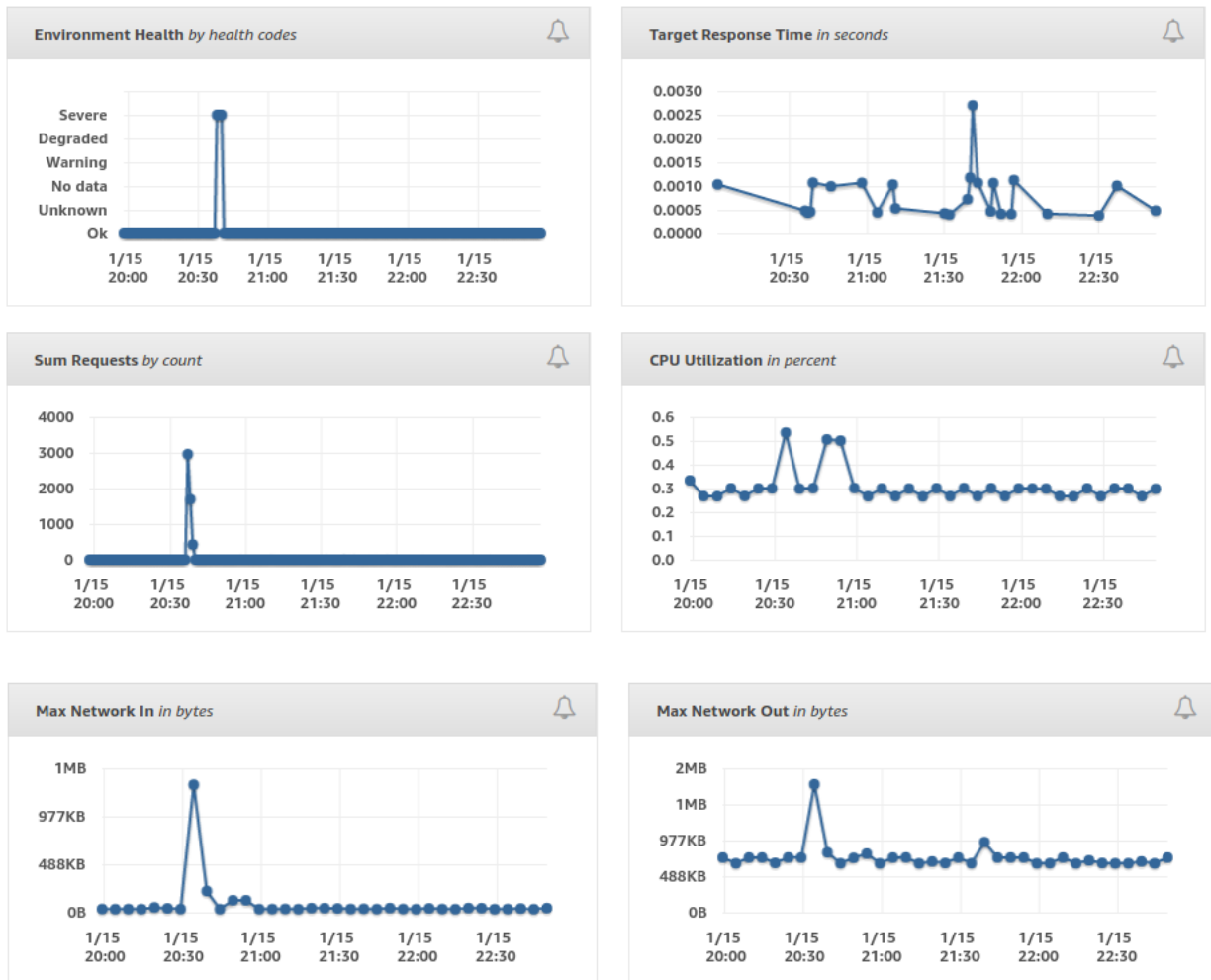




The other way to monitor our immediate environment data is to simply go to our application and in the side nav bar, select “Monitoring”. Once we are there we can easily see the stats our application is sending to our environment.



We are also provided with stats such as Environment Health, Target Response Time, Sum Request, CPU Utilizations, Max Network In and Max Network Out. We can set alarms on these parameters by clicking on the Bell icon at each statistic. We can also change the time range [from 1 hr to 2 weeks].

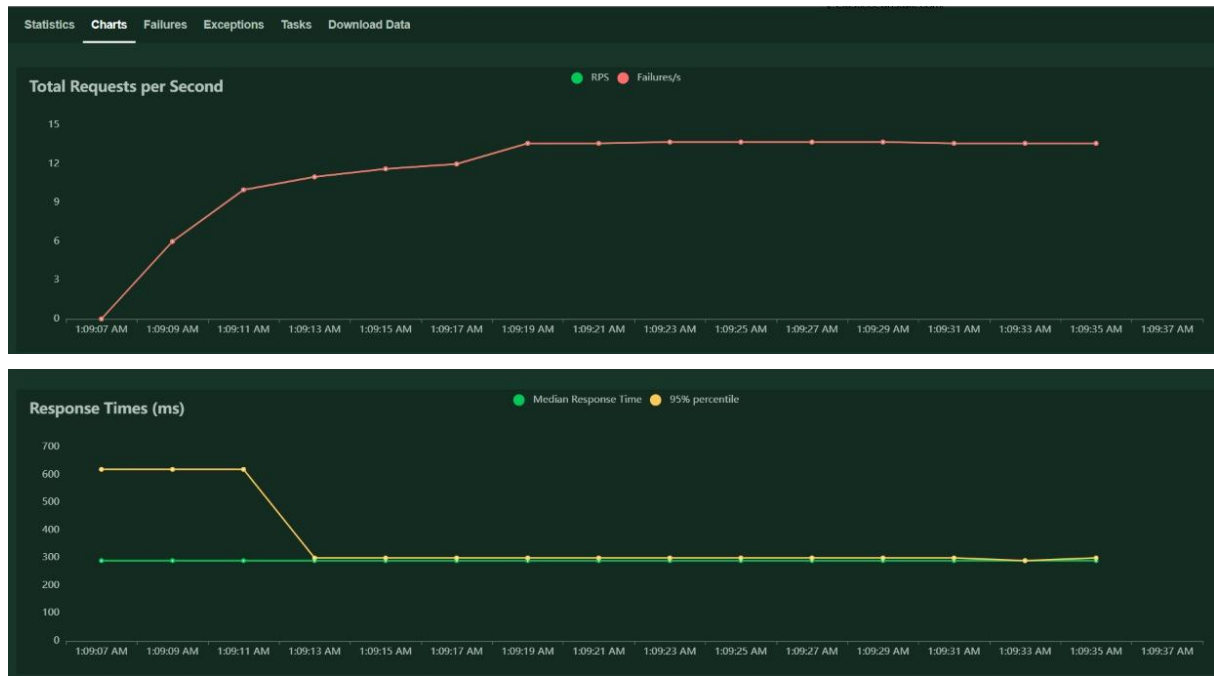


>> Now if you notice those graphs carefully, you will notice that almost all of them have had a spike or a reaction during the 20:30 and 21:00. This is because it was at this time we had put a load on it. We stressed the system using software called Locust. This helped us in creating artificial load and allows us to look at the performance of our website and more importantly, our environment which is hosted on Elastic Beanstalk. A Locust test is basically running a Python file called `locustfile.py` whose content is:

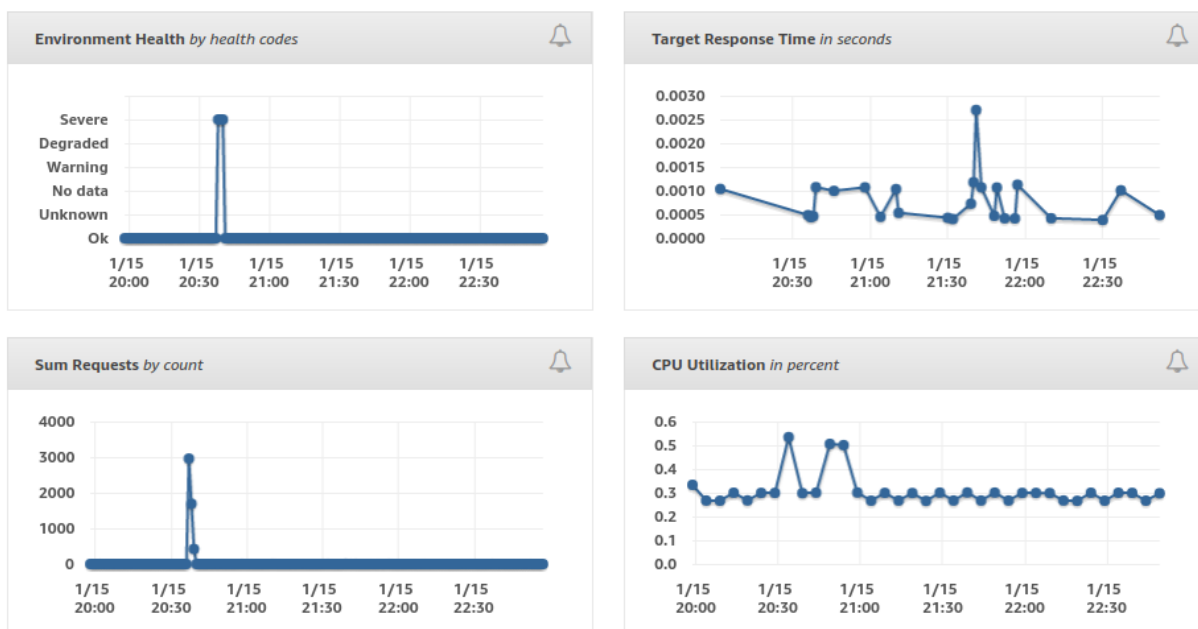
```
from locust import HttpUser, task

class HelloWorldUser(HttpUser):
    @task
    def hello_world(self):
        self.client.get("/hello")
        self.client.get("/world")
```

After running this program we go to our browser and type <http://localhost:8089/> into the URL section and arrive at the browser interface of Locust service. Here we can choose the number of users and their spawn rate which will be tested on the host website. In our case the host was our website hosted on AWS.



After the test we can get these visual charts which tells us how the test went. Durin this test our website behaved like this:



Now you might notice that our Env Health went to “Severe” during the test. That is because it was heavily stressed upon within a short period of time. But it is also important to notice the fact that even though the website was under pressure, Elastic Beanstalk did not allow it to collapse. This proves that it is capable of doing what is required to stay alive including load balancing and health checks.

>> One more test we did was to see if deploying our website on AWS's Elastic Beanstalk would yield better results than if we hosted it on a private server.

We tested the performances of both situations using GTmetrix. It is a website that displays performances of other websites.

The first one is our website hosted on a private server called "airafique.com". And the results aren't that impressive.



And now we did the same test using the same website but now it was from AWS Elastic BeanStalk, and the results were pretty impressive.



Latest Performance Report for:

<http://testinganidea-env.eba-anxxakn3.us-east-2.elasticbeanstalk.com/>

Report generated: Sat, Jan 15, 2022 9:23 AM -0800

Test Server Location:  Vancouver, Canada

Using:  Chrome (Desktop) 90.0.4430.212, Lighthouse 8.3.0

GTmetrix Grade ?

B

Performance ?
95%

Structure ?
73%

Web Vitals ?

LCP ?
1.2s

TBT ?
108ms

CLS ?
0.04

Summary

Performance

Structure

Waterfall

Video

History

Performance Metrics

The following metrics are generated using Lighthouse Performance data.

Metric details ☐ OFF

First Contentful Paint ?

Good - Nothing to do here

883ms

Time to Interactive ?

Good - Nothing to do here

1.2s

Speed Index ?

Good - Nothing to do here

1.1s

Total Blocking Time ?

Good - Nothing to do here

108ms

Largest Contentful Paint ?

Good - Nothing to do here

1.2s

Cumulative Layout Shift ?

Good - Nothing to do here

0.04

V Conclusion

After performing a battery of tests and getting amazing results that deliver the performance we expect. We as a team came to the conclusion that selecting **AWS Elastic BeanStalk** for our project was the correct idea.

This Conclusion marks the end of our Project Report