Program4 Report

Name: Pooja Nadagouda

Website URL:
http://program4-env.eba-kwvmv3zp.us-west-2.elasticbeanstalk.com/index

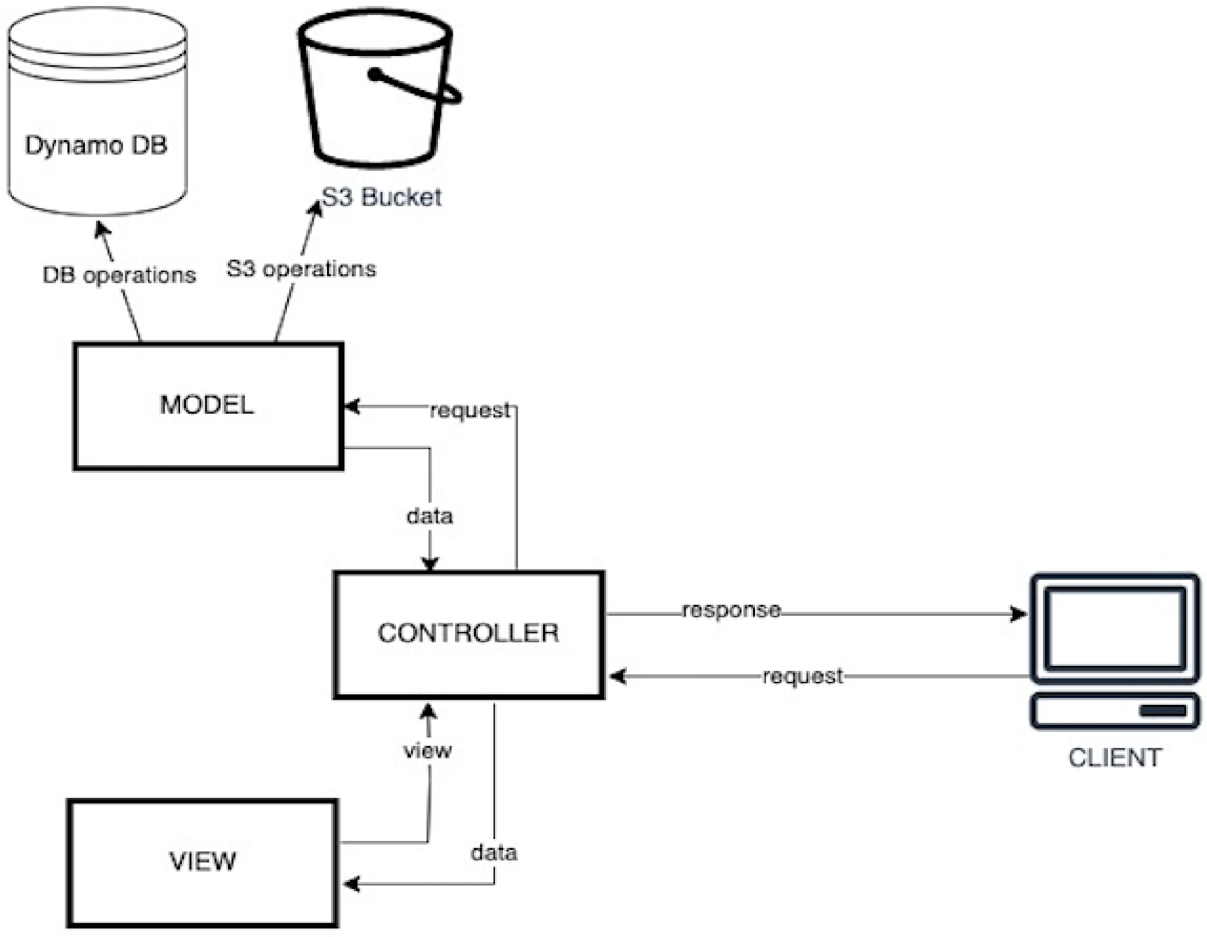
S3 URL to verify data loaded:
https://poojancss436program4inputfiledetails.s3.us-west-2.amazonaws.com/input.txt

Design:

Specification:

Cloud Provider used: AWS
Cloud Services used: AWS S3, AWS Dynamo DB, AWS Elastic Beanstalk
Framework used: Spring-Boot
Coding language for backend: Java
Coding language for frontend: JSP, JavaScript

Design diagram:



This web application follows MVC architectural pattern

Controller: The controller receives the request from client and directs it to the model. After the model sends in the processed data, controller requests the view for preferred presentation, the controller sends this view back to the client, which is displayed on client's browser.
Also, when the request is being processed, client/user should wait, the controller requests status messages e.g.: please wait or loading etc. from the view which intimate the user about ongoing process.
Controller classes: CloudController.java
This class sends the request to model for load, clear and query operations. It takes the processed data from model and sends it to the view to render the data in readable format for the client. It returns this view to the client's browser.

Model: The model interacts with AWS S3 storage and AWS dynamo DB either to load, clear or query the data. It sends the data back to controller.
Model classes: S3Operation.java, DynamoDBOperation.java
Util classes: DataParser.java, QueryParser.java, Person.java

- **Person.java** – Holds person information with FirstName, LastName as string variables and all other attributes as JSONObject.
- **DataParser.java** – Parses the data read from S3 file into Person Object
- **QueryParser.java** – Parses the request sent from controller for query request and calls appropriate query from DynamoDB class based on only FirstName sent or only LastName sent, or Both sent.

When controller sends load/clear/query request, the model processes the request and sends the success or error messages to the controller.

- **Load operation:**
  Here the S3Operation class creates the bucket if it doesn't exist and copies the object (input.txt file) from sender's bucket (professor hosted i.e., from css490 bucket) to receiver's bucket (here bucket in my aws account). The data is read from the object from receiver's bucket parsed by DataParser.java class. The parsed data is inserted into Dynamo DB. The DynamoDBOperation class creates a table if doesn't exist. The table has FirstName as partition key and LastName as sort key. Also, the table has a global secondary index LastName.

- **Clear operation:**
  Here the S3Operation class deletes the object (input.txt) from receiver's bucket (bucket in my account). DynamoDBOperation class deletes the table. In case the table or bucket object doesn't exist, appropriate error messages are sent to the controller.

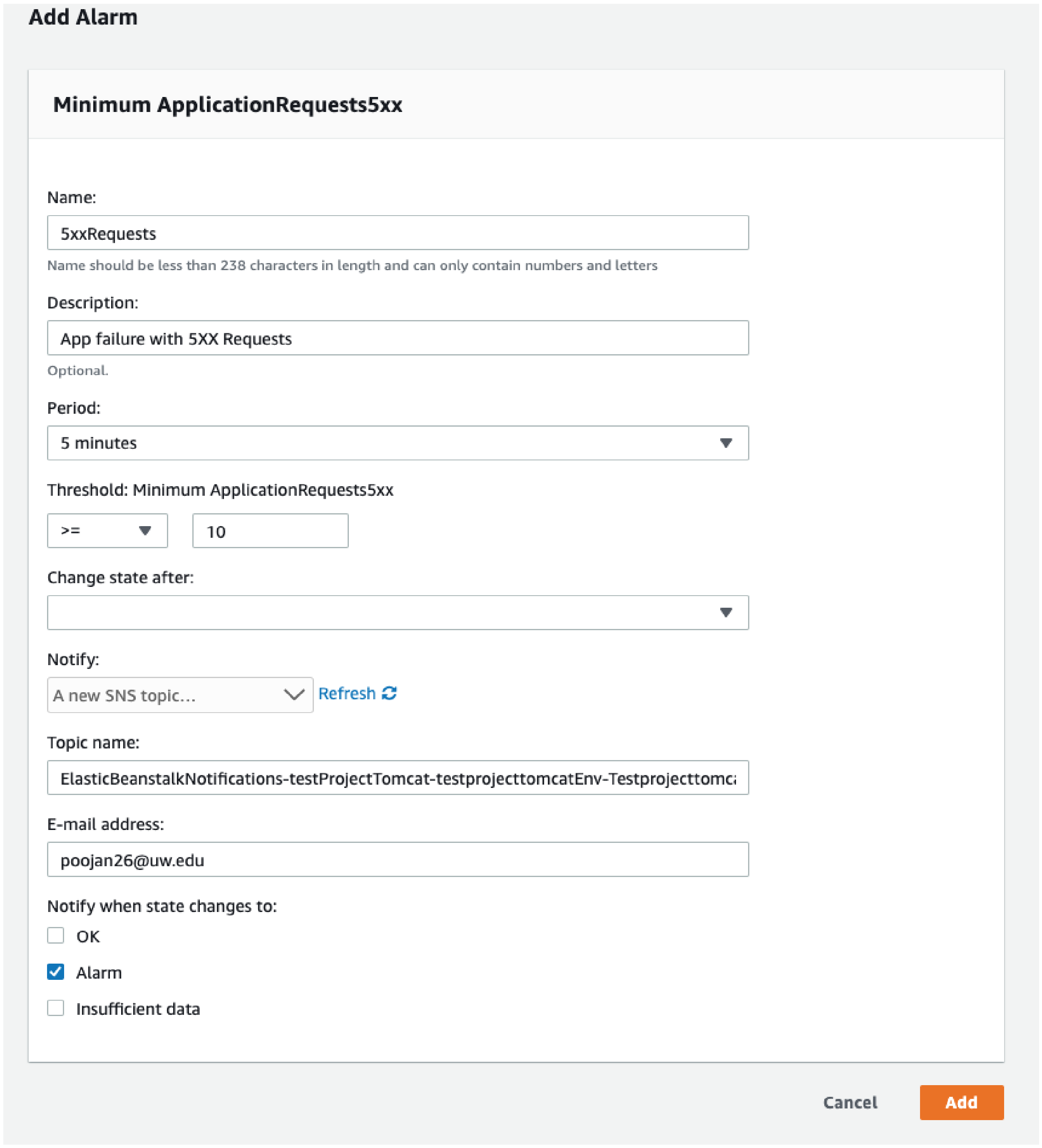
- **Query operation:**
  Here the QueryParser.java class receives the request from controller. Based on the client/user input (only FirstName sent or only LastName sent, or Both sent) the appropriate query method is called from DynamoDBOperation class. The DynamoDBOperation class processes the query and sends the result to DataParser.java class which parses the result to Person objects. These objects are sent to the controller.

View: The view takes the processed data from controller and renders it in HTML format to the user.

View files: index.jsp
It takes data from controller, renders it in readable format. It displays the success messages, person details, error messages or status messages for all 3 operations i.e., load, clear and query.

Monitoring:
CloudWatch service offered by AWS monitors all the services being used.
We can also add alarm notification for monitoring, where we can receive email notifications when configured thresholds are reached. In case of this application, I have configured alarm email notification if more than 10 5XX requests are received in span of 15minutes.

Following the configuration screenshot:



Add Alarm

Minimum ApplicationRequests5xx

Name: 5xxRequests
Name should be less than 238 characters in length and can only contain numbers and letters

Description: App failure with 5XX Requests
Optional.

Period: 5 minutes

Threshold: Minimum ApplicationRequests5xx
>= 10

Change state after:

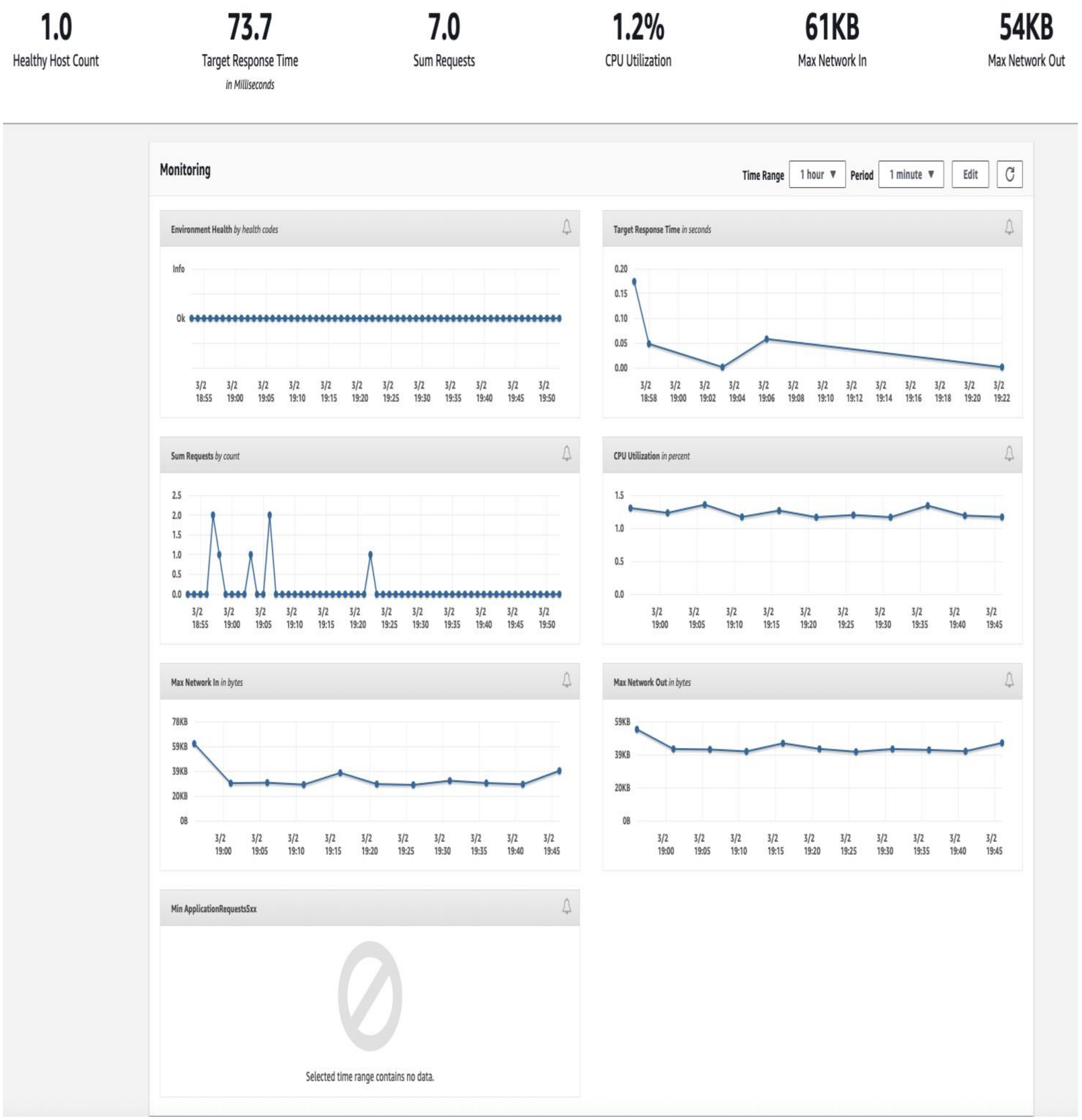
Notify: A new SNS topic... Refresh

Topic name: ElasticBeanstalkNotifications-testProjectTomcat-testprojecttomcatEnv-Testprojecttomc

E-mail address: poojan26@uw.edu

Notify when state changes to:
- [ ] OK
- [x] Alarm
- [ ] Insufficient data

Following screen shot shows overall monitoring enabled in elastic beanstalk along with monitoring for 5xx requests. It can be seen, there are zero 5xx requests.



1.0 Healthy Host Count

73.7 Target Response Time *in Milliseconds*

7.0 Sum Requests

1.2% CPU Utilization

61KB Max Network In

54KB Max Network Out

Monitoring

Time Range: 1 hour ▼ Period: 1 minute ▼

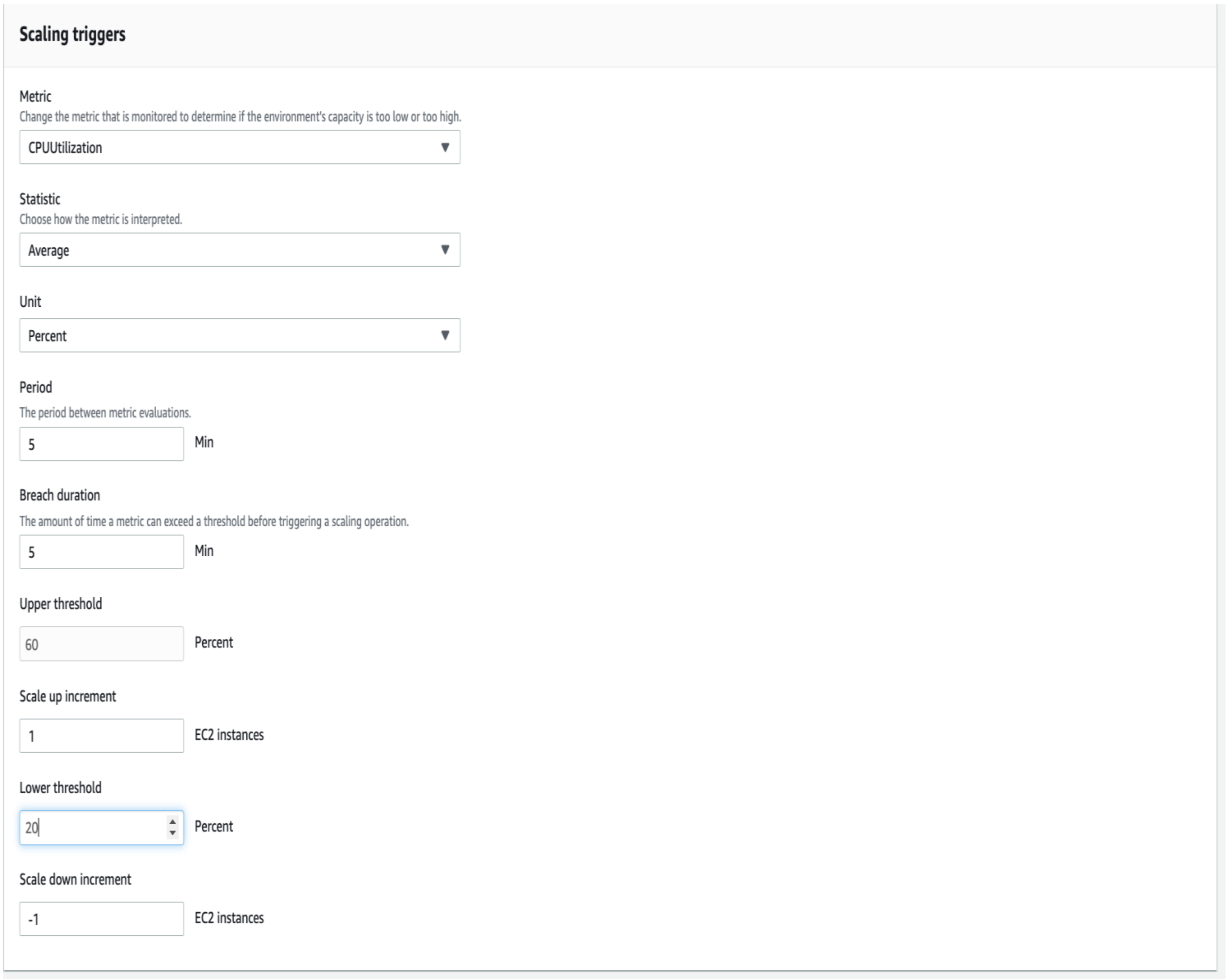
Min ApplicationRequests5xx: Selected time range contains no data.

Edit
Environment Health *by health codes*
Info
Ok
Target Response Time *in seconds*
Sum Requests *by count*
CPU Utilization *in percent*
Max Network In *in bytes*
Max Network Out *in bytes*
Min ApplicationRequests5xx

Application Auto Scaling:

EC2 autoscaling:
Amazon EC2 Auto Scaling monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. Using the elastic beanstalk console we can configure the instances, i.e., upscale, or downscale EC2 instances based on CPU utilizations, disk read/write operations, request count, network usage etc. The scaling can be done based on timing as well depending on when (what time of day) the usage of service is high we can increase the number of EC2 instances.

Following screenshot shows the auto scaling configuration done for this website (program4)
Here it upscales when CPU utilization crosses 60% and down scales when CPU utilization falls below 20%.


Scaling triggers

Metric
Change the metric that is monitored to determine if the environment's capacity is too low or too high.
CPUUtilization

Statistic
Choose how the metric is interpreted.
Average

Unit
Percent

Period
The period between metric evaluations.
5 Min

Breach duration
The amount of time a metric can exceed a threshold before triggering a scaling operation.
5 Min

Upper threshold
60 Percent

Scale up increment
1 EC2 instances

Lower threshold
20 Percent

Scale down increment
-1 EC2 instances

Dynamo DB autoscaling:
In case of Dynamo DB, we can enable auto scaling using the AWS auto scaling service which dynamically enables the provisioned throughput capacity based on increased/decreased traffic patterns. When a DynamoDB table is created, auto scaling is the default capacity setting, but we can enable auto scaling on any table that does not have it active using a scaling policy in Application Auto Scaling. To configure auto scaling in DynamoDB, we set the minimum and maximum levels of read and write capacity in addition to the target utilization percentage. Auto scaling uses Amazon CloudWatch to monitor a table's read and write capacity metrics. To do so, it creates CloudWatch alarms that track consumed capacity. For this application I have not enabled auto scale due to charges applicability.

S3 autoscaling:
Applications can easily achieve thousands of transactions per second in request performance when uploading and retrieving storage from Amazon S3. Amazon S3 automatically scales to high request rates. For example, an application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per object in a bucket.
Hence this application (Program4) supports autoscaling handling any number of COPY/POST/GET requests. (Charges will apply on crossing free tier range)

SLA:
This web application's uptime depends on following services

1. Elastic bean stalk:
   Here the application is hosted on EC2 and uses ELB, hence taking SLA of EC2 instance and ELB which is 99.99%. Since load balancer and auto scaling is used there is no downtime of web application (for maintenance or upgradation), during upgradation or maintenance the ELB takes care of running the application on another instance.
2. Dynamo DB:
   AWS offers 99.99% SLA
3. S3 storage for uploading the input.txt file:
   AWS offers 99.9% SLA for S3 Standard storage
4. Service uptime for website from where input data is read:
   Since the input.txt file is placed in AWS S3, we can take same SLA as above i.e., 99.99% availability

Thus, total SLA = (0.9999*0.9999*0.9999*0.999*0.999) = 99.77