Design Document – URL Shortener

High-Level Design

We create a mapping of a long URL to a short URL consisting of 5 characters. Each character lies from [a-z][A-Z][0-9] giving us 62 possibilities. Thus, we can store up to 62^5 which is about 916 M possibilities of URLs in our database. The md5 hash of the Base62 encoding of the user's URL input is taken.

All the functionality is present on the Web GUI application We have added a CURL API call service as well to generate a short URL via the terminal/command line if necessary. This functionality is limited to only creating new short URLs and not for checking user history.

Four 4 major aspects were covered:

Freemium model implemented

Free-Tier

- Just gives a mapping between the long and short URL of length 5

Premium Customer

- The user submits his email which is followed by OTP verification which will come into the client's mailbox (Do check spam/promotions etc) after which he is directed to a new page.
- The user is given the option to enter his long URL followed by a custom string that gives the short URL as the same as his custom string (Let us say he wants to keep it IITBhilai, so we give him IITBhilai) which allows him to use it.
- The user can specify a time for which the short URL will work as well. If the custom string is given a dialog says that the user needs to give another custom string.

Website Analytics (Front-Page)

Website usage is stored to give an idea of popularity to its old and new users. Website analytics is displayed on the front page which gives a history of the count of reads and count of writes count over the past 2 weeks to see the increase and decrease in the usage of service.

User Analytics (Premium Users)

Custom data associated with the custom string and database entry of a particular premium user is provided. Once the authentication is completed, we provide the user with a history of the service he has used. A table is given that gives him statistics of the number of redirections when each URL was created and each long URL that was inserted.

Tech Stack

Backend - Node.js, EJS

Integrates the database queries and information from the front end. API Keys for curl requests are covered using this. It also schedules the running of python scripts for daily database updates to render on the webpage effectively.

EJS – Generated JavaScript templates to render the real-time user data for better UX.

Frontend – HTML/CSS

To display the graph on the home page and user history tables for premium users. Adding color features to the pages as well as making the page interactive graphically is performed here.

Analytics – Python

Written scripts to effectively trigger SQL queries on user email authentication to display history in a tabular format and auto-update the appropriate database tables daily (runs at 12:05 AM every day), values of redirection count, etc. It was also used to generate a curl script to check the application performance to the scaling specifications.

Database Schema

We use a relational database (MySQL server) as a large hash table, mapping the generated URL to a file server and path containing the paste file. It stores extra things such as redirection count, creation time, email ID for premium users.

As we have built an OTP authentications system, we even store the OTPs for a particular email for the first 5 minutes as we need to deal with multiple users dealing with the authentication process at the same time.

We store a table for the website usage to render it on the home page as well.

The reason for a SQL database is the vertical scaling as the numbers will grow bigger by the number of entries. We will rarely need to add a new row (such as a situation where we require new analytics on the system) but for the functional requirements, it won’t be necessary.

The MySQL schemas are as follows:

```
mysql> describe urlMap;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| url                | varchar(500) | NO   |     | NULL    |       |
| shortenedurl       | varchar(20)  | NO   | PRI | NULL    |       |
| creation_timestamp | timestamp    | NO   |     | NULL    |       |
| ttl                | bigint       | NO   |     | 30      |       |
| num_of_redirections | bigint      | NO   |     | 0       |       |
| email              | varchar(50)  | YES  |     | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

```
mysql> describe readWriteCount;
+------------+--------+------+-----+---------+-------+
| Field      | Type   | Null | Key | Default | Extra |
+------------+--------+------+-----+---------+-------+
| currDate   | date   | NO   | PRI | NULL    |       |
| readCount  | bigint | NO   |     | 0       |       |
| writeCount | bigint | NO   |     | 0       |       |
+------------+--------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

```
mysql> describe emailOTP;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| email | varchar(100) | NO   | PRI | NULL    |       |
| otp   | bigint       | NO   |     | NULL    |       |
| time  | timestamp    | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

```
mysql> describe emailAPIKeys;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| email  | varchar(200) | NO   |     | NULL    |       |
| apikey | varchar(100) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

CURL Functionality:

We use a Public REST API Service:

4 API calls will be created.

In all the calls, ":apiKey" is a variable that a premium user will have to provide to run his curl requests. This key will be provided to him the moment his first authentication is complete, after which the key enables him to use the service without worrying about future authentication.

1. To Read:

route:- '/api/readshorten/:apiKey'

data:- {shorturl: "short LINK of which we want full long URL "}

2. To Delete:

route:- '/api/deleteurl/:apiKey'

data:- {shorturl: "short LINK of which we want to delete DB entry and have access of that link "}

3. To Shorten:

route:- '/api/createshorten/:apiKey'

data:- {urldata: "LINK_TO shorten"}

4. To Create Custom

route:- '/api/createcustomshorten/:apiKey'

data:- {customurldataa: "custom short LINK of which we want link",

longurldata: "LINK_TO shorten",

ttl: "Time to live in number of days"}

Refer to the performance.py file for the execution of more details in CURL and the ReadMe file.

Load Balancer

We have used an Application Elastic Load Balancer directly provided to us by Beanstalk configuration. This helps us manage multiple virtual machines at the same time.

```
  RootVolumeIOPS: null
  RootVolumeSize: null
  RootVolumeThroughput: null
  RootVolumeType: null
  SecurityGroups: awseb-e-zhefxpsztx-stack-AWSEBSecurityGroup-15N198VTSNG6E
AWSEBEC2LaunchTemplate.aws:ec2:vpc:
  AssociatePublicIpAddress: null
AWSEBLoadBalancerSecurityGroup.aws:ec2:vpc:
  VPCId: null
AWSEBV2LoadBalancer.aws:elbv2:loadbalancer:
  AccessLogsS3Bucket: null
  AccessLogsS3Enabled: 'false'
  AccessLogsS3Prefix: null
  IdleTimeout: null
  SecurityGroups: sg-0786b70b02a1505e9
AWSEBV2LoadBalancerListener.aws:elbv2:listener:default:
  DefaultProcess: default
  ListenerEnabled: 'true'
  Protocol: HTTP
  Rules: null
  SSLCertificateArns: null
  SSLPolicy: null
aws:autoscaling:asg:
  Availability Zones: Any
  Cooldown: '360'
```

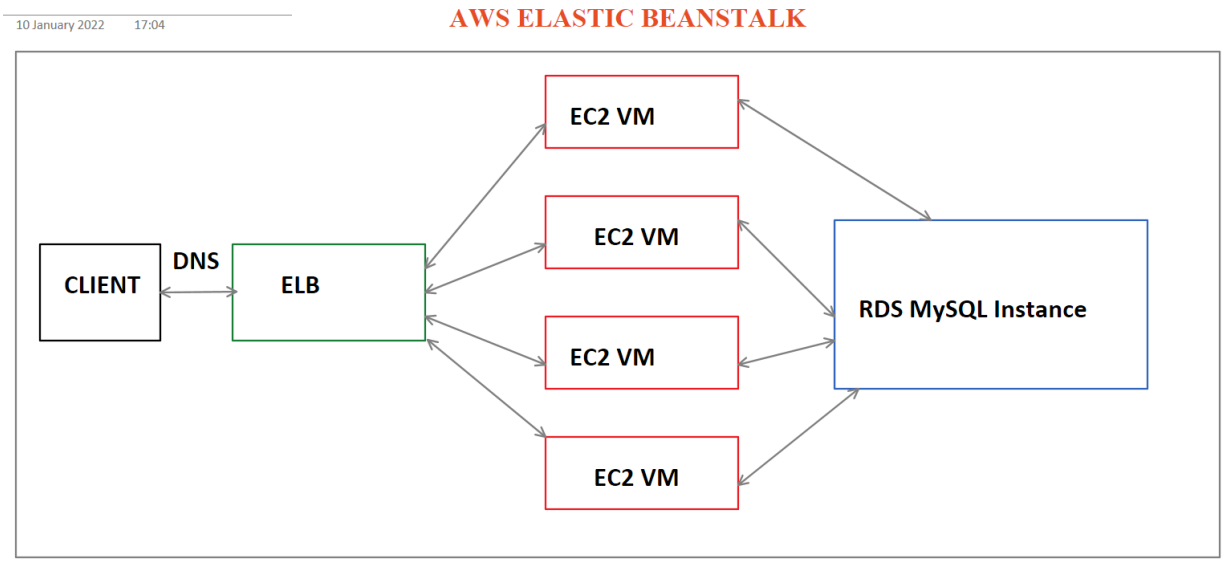
Fault-Tolerance

To make the application Fault Tolerant, we have used AWS Elastic Beanstalk that helps us to Auto-Scale applications directly by linking multiple EC2 (Elastic Compute Cloud) Instances to an Application Elastic Load Balancer (ELB/ALB). This load balancer will help redirect data to running instances in a situation where they are down or the network is slow.

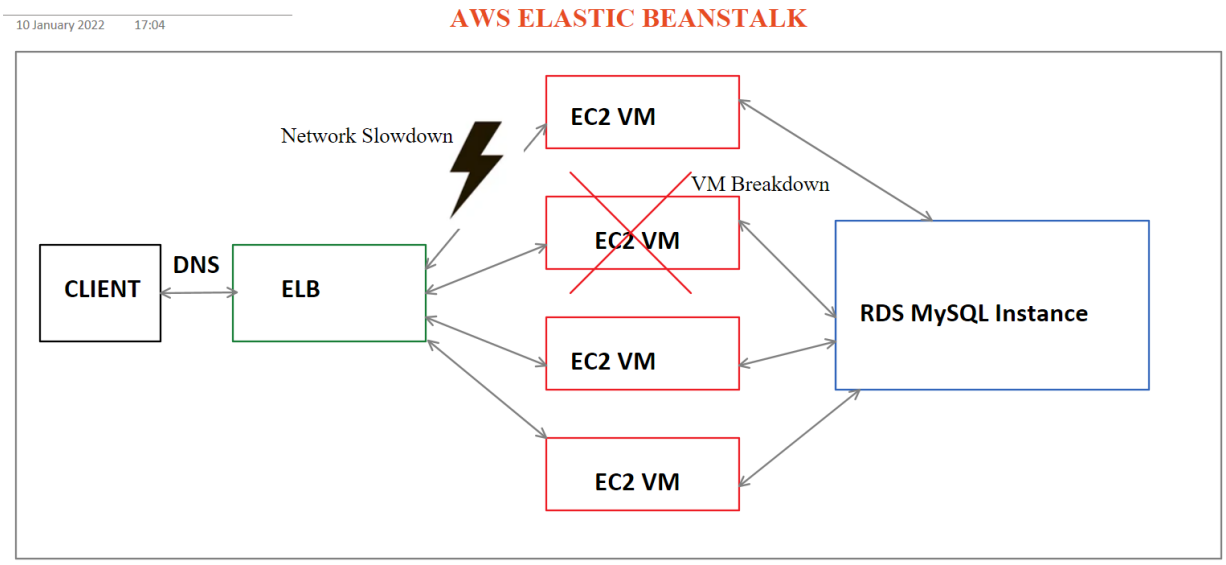
At the backend, we have linked an RDS MySQL Instance that stores all the data coming from each VM directly. We can continue to access data from there, once a VM has crashed, as the other VMs have been connected to it.

Diagrammatically:

This is the environment created by us on Elastic Beanstalk.



If the system is fault-tolerant, then the faults below are handles. As we have extra VMs to take care of the CURL and URL Requests are application will be fault-tolerant.



To Check for Network Slowdown:

Every system (our EC2 instance also) has a scheduler that enqueues the network packets, before sending them to a network interface. We can control the network traffic of our system by manipulating the queueing characteristic of packets in the Linux kernel network stack of our EC2 instance.

tc command

tc (https://man7.org/linux/man-pages/man8/tc.8.html) is a traffic control command-line tool in Linux. Specifically, it is through the tc command that we configure the traffic control settings in the Linux kernel network stack. A queueing discipline (or qdisc for short) is a scheduler that manages the scheduling of packets queue. It is through the qdisc that we are controlling the network traffic.

Adding delay

- We are using the delay option on netem to add the delay of 2s on our network queue, by which every incoming request and the outgoing response would be delayed by 2s.

- We are checking the RTT of our instance using the ping command to google.com

    $ ping -c 5 google.com

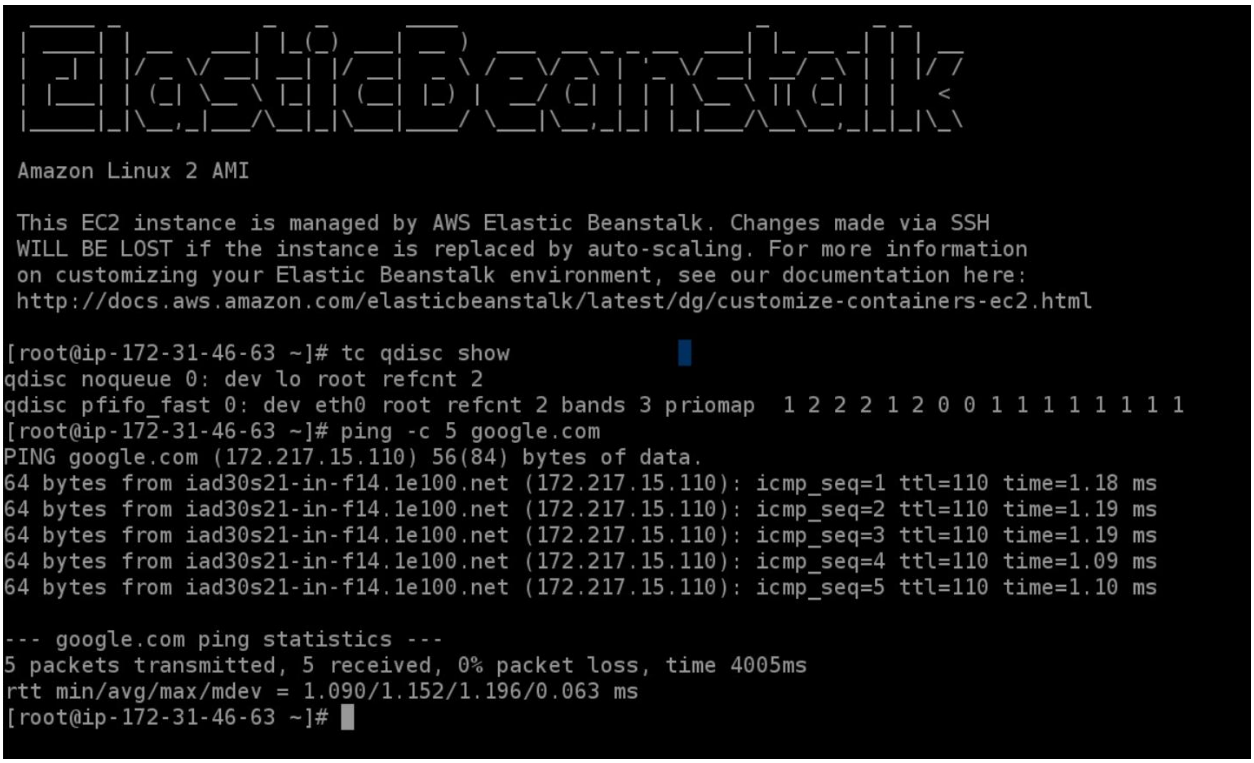
  The result is shown below screenshot.

- Then, we are adding the delay to our network queue using the following command

    $ sudo tc qdisc add dev eth0 root netem delay 2s

- Then, again running the ping command to google.com would give the following result

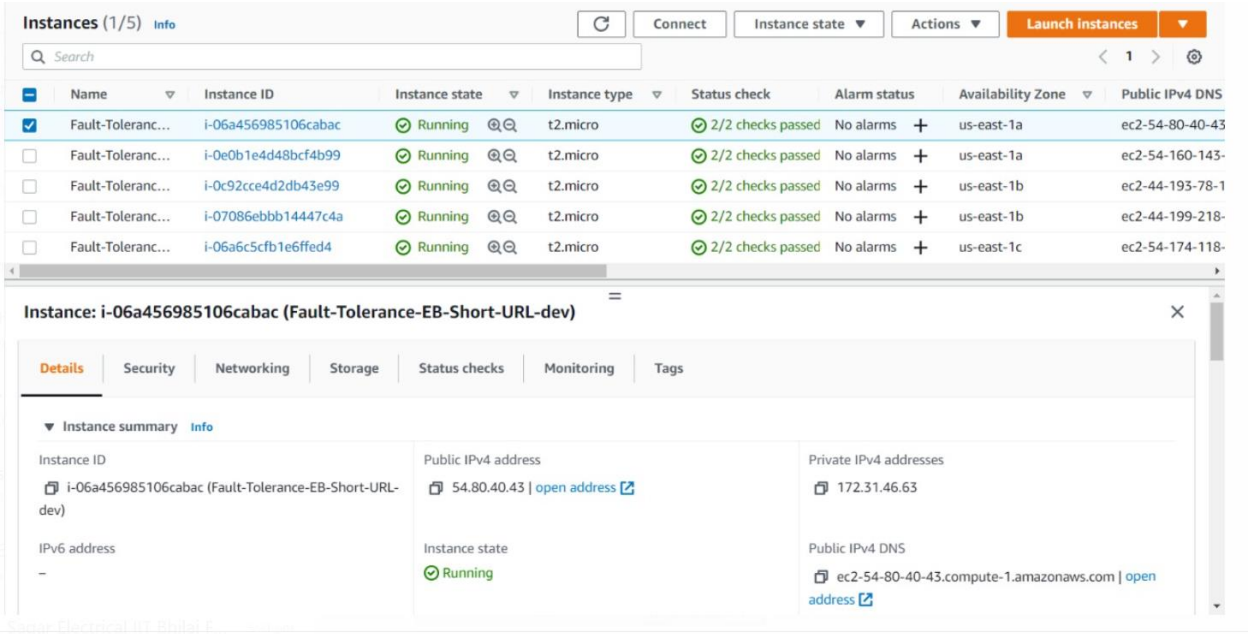
- We can run a similar procedure on other instances.

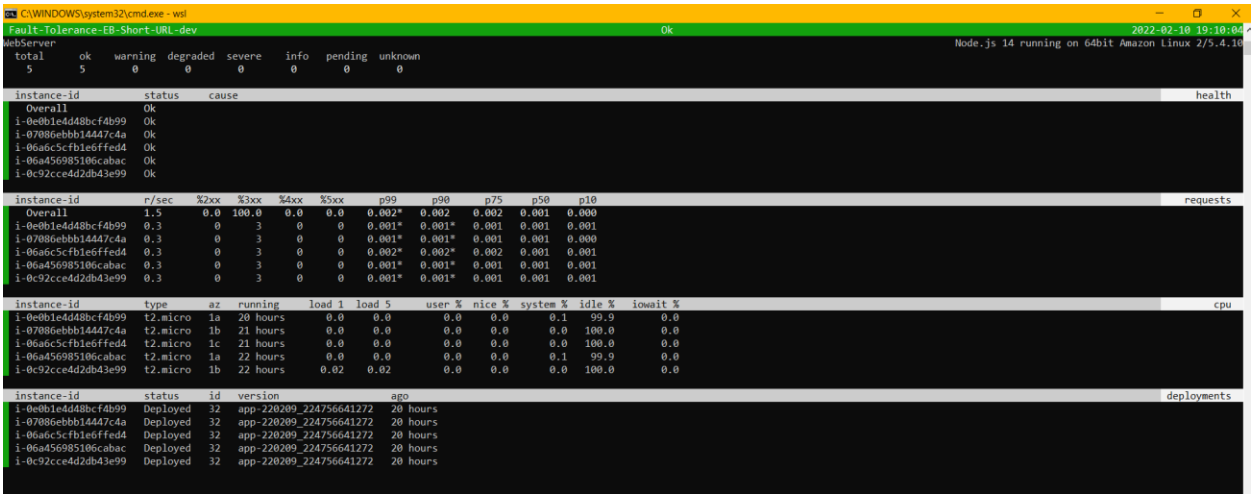


To check for a Hard System Crash Failure or EC2 VM Termination:

We have 5 instances out of which any VM can be terminated directly/manually and we can continue to see that the termination of any particular 1-2 instances will not create a problem and the system will continue to perform.

We can at a maximum allow 4 VMs to crash or support network outages on all 5 but beyond which are system will fail.

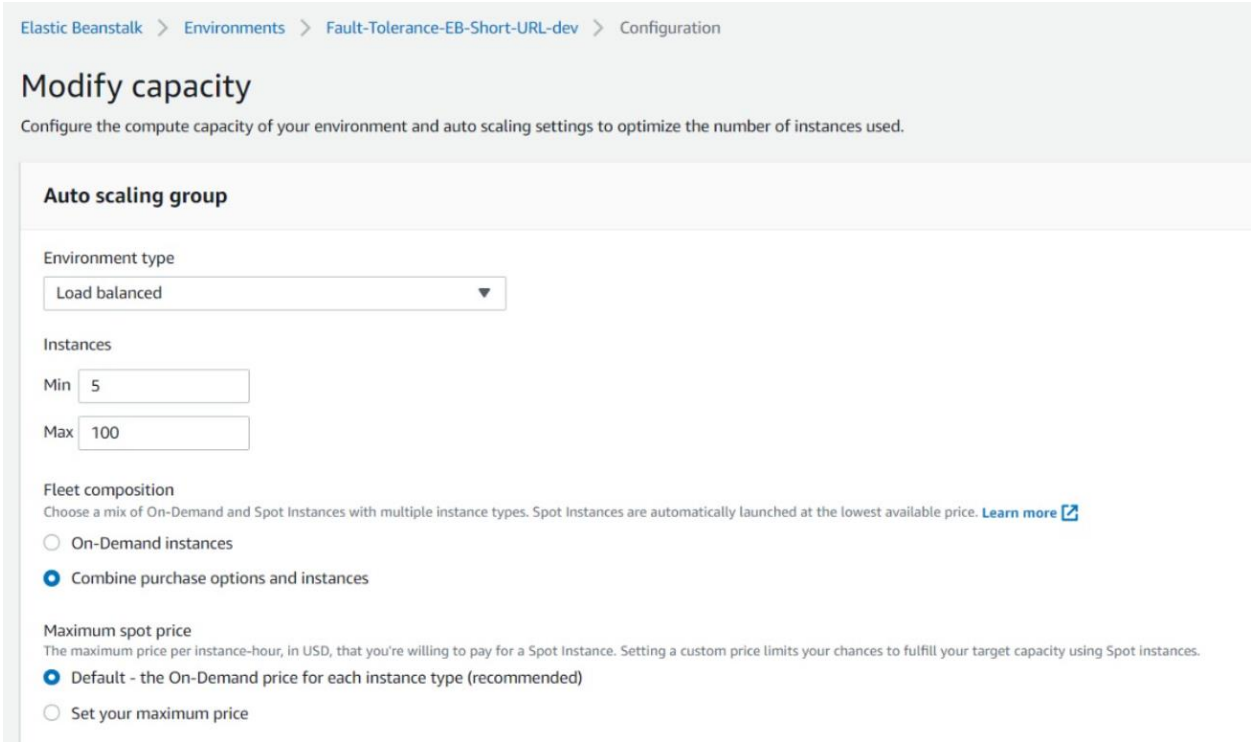




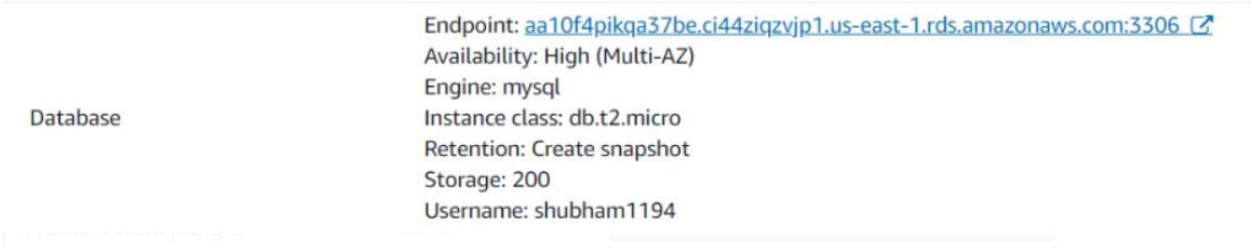
Our configurations on AWS are as follows:

The current status of our EB is:

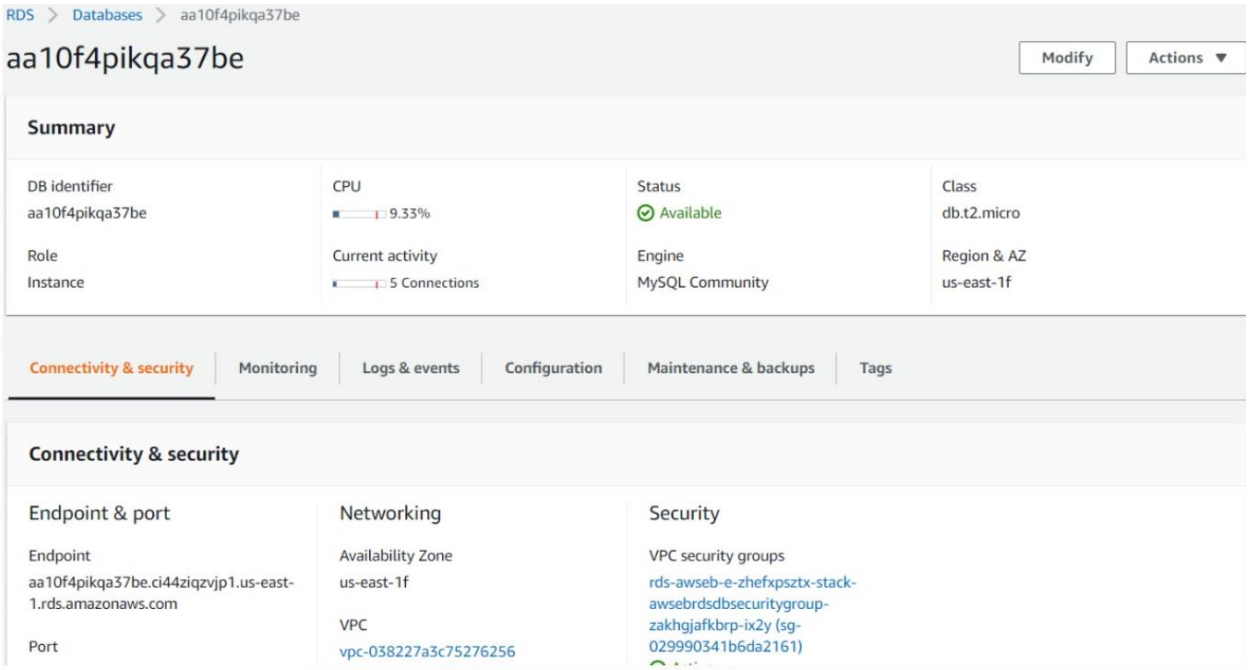
```
mastershubham@LAPTOP-8Q15SHE6:/mnt/d/songtest/final$ eb status
Environment details for: Fault-Tolerance-EB-Short-URL-dev
  Application name: Fault_Tolerance_EB_Short_URL
  Region: us-east-1
  Deployed Version: app-220209_224756641272
  Environment ID: e-zhefxpsztx
  Platform: arn:aws:elasticbeanstalk:us-east-1::platform/Node.js 14 running on 64bit Amazon Linux 2/5.4.10
  Tier: WebServer-Standard-1.0
  CNAME: Fault-Tolerance-EB-Short-URL-dev.us-east-1.elasticbeanstalk.com
  Updated: 2022-02-10 07:47:13.039000+00:00
  Status: Ready
  Health: Green
mastershubham@LAPTOP-8Q15SHE6:/mnt/d/songtest/final$
```



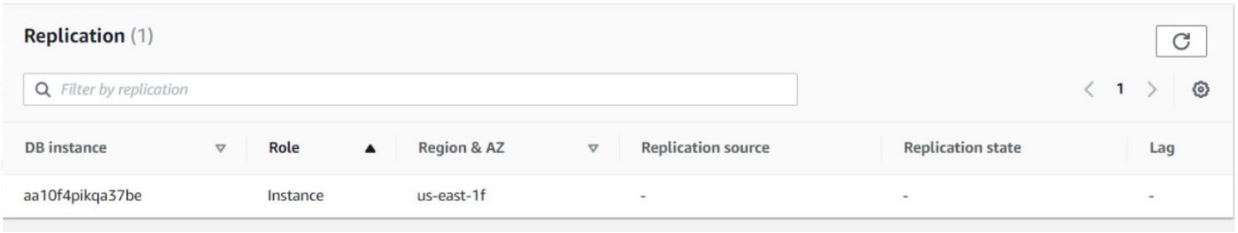
The RDS Instance is configured to:




```
aws:rds:dbinstance:
  DBAllocatedStorage: '200'
  DBDeletionPolicy: Snapshot
  DBInstanceClass: db.t2.micro
  DBSnapshotIdentifier: null
  HasCoupledDatabase: 'true'
  MultiAZDatabase: 'true'
```



We have created one replicated instance of RDS as well.



We have specified the various functionalities, features, configurations of the entire application above. The usage of scalable application with RDS at backend and NodeJS as our tech stack completed the URL Shortening application.

The Load Balancing with Fault Tolerance has been specified at the end of this design document.

References:

Website: *http://fault-tolerance-eb-short-url-dev.us-east-1.elasticbeanstalk.com/home*

GitHub: *https://github.com/shubham11941140/short_url*

Refer to the repository ReadMe file for further instructions on how to proceed with the document work and performance.py file to submit batch jobs on testing the system as previously mentioned.

Submitted To:

Course Instructors:

1. Dr. Gagan Raj Gupta – gagan@iitbhilai.ac.in
2. Dr. Subhajit Sidhanta – subhajit@iitbhilai.ac.in

Teaching Assistant:

1. Sudev Kumar Padhi – sudevp@iitbhilai.ac.in
2. Muttareddygari Sreechakra – muttareddygaris@iitbhilai.ac.in

Dated: 10th February 2022

Authored By:

1. Shubham Gupta – 11941140 – shubhamgupta@iitbhilai.ac.in
2. Abdur Rahman Khan – 11940020 – abdurrahman@iitbhilai.ac.in
3. Lavish – 11940640 – lavishg@iitbhilai.ac.in