**POC(Proof of Concept)**

**Job posting:**

**AWS VPC Test Environment Setup and Realistic Data Generation**

We are seeking an adept professional with a focus on setting up AWS VPC test environments and expertise in generating realistic data. The project involves creating a genuine VPC environment, simulating network components, and implementing tools for meaningful data generation.

**Responsibilities:**

Design and implement a realistic AWS VPC, configuring subnets, route tables, and security groups.

Launch and configure EC2 instances to simulate various components within the network.

Enable VPC Flow Logs and direct them to a CloudWatch Logs group for centralized storage.

Develop scripts or use tools to generate realistic data within the VPC environment.

Ensure seamless connectivity between the test environment and a MongoDB instance for log storage and analysis.

**Requirements:**

Proven experience in setting up AWS VPCs, including configuring subnets, route tables, and security groups.

Strong background in launching and configuring EC2 instances for simulating realistic network components.

Expertise in enabling and configuring VPC Flow Logs and directing them to CloudWatch Logs.

Demonstrated ability to generate realistic data within a simulated environment.

Familiarity with MongoDB and the ability to forward and analyze logs within this database.

Scripting skills or experience using tools for data generation and processing.

**Preferred Qualifications:**

Previous experience in simulating realistic network traffic and generating meaningful data within AWS environments.

Knowledge of AWS security configurations, IAM roles, and best practices.

Capability to troubleshoot and optimize data generation scenarios for accurate analysis.

**Design and implement a realistic AWS VPC, configuring subnets, route tables, and security groups.**

Created a VPC named **dr-vpc** with public subnets, route-tables and IGW

Security groups are to be setupd on subnet level while creating resources inside the subnets.

**Additional -**

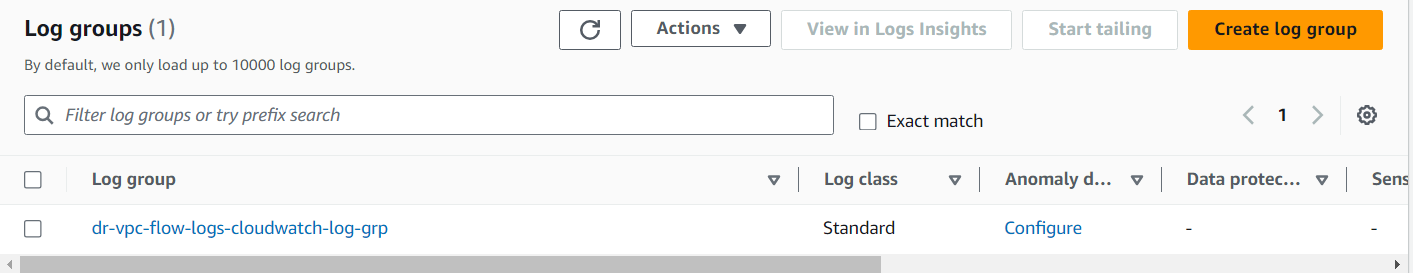
If required we can create private subnets also in which the applications present in the private subnets can access the internet but the internet cannot access the application present in the private subnet by configuring NAT gateway.



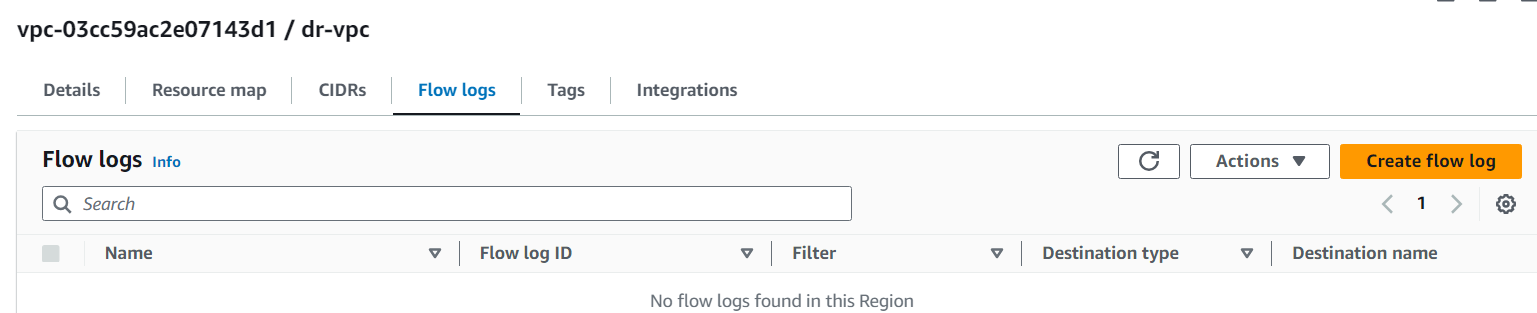
**Enable VPC Flow Logs and direct them to a CloudWatch Logs group for centralized storage.**

To use VPC flow logs need to setup a cloudwatch log grp

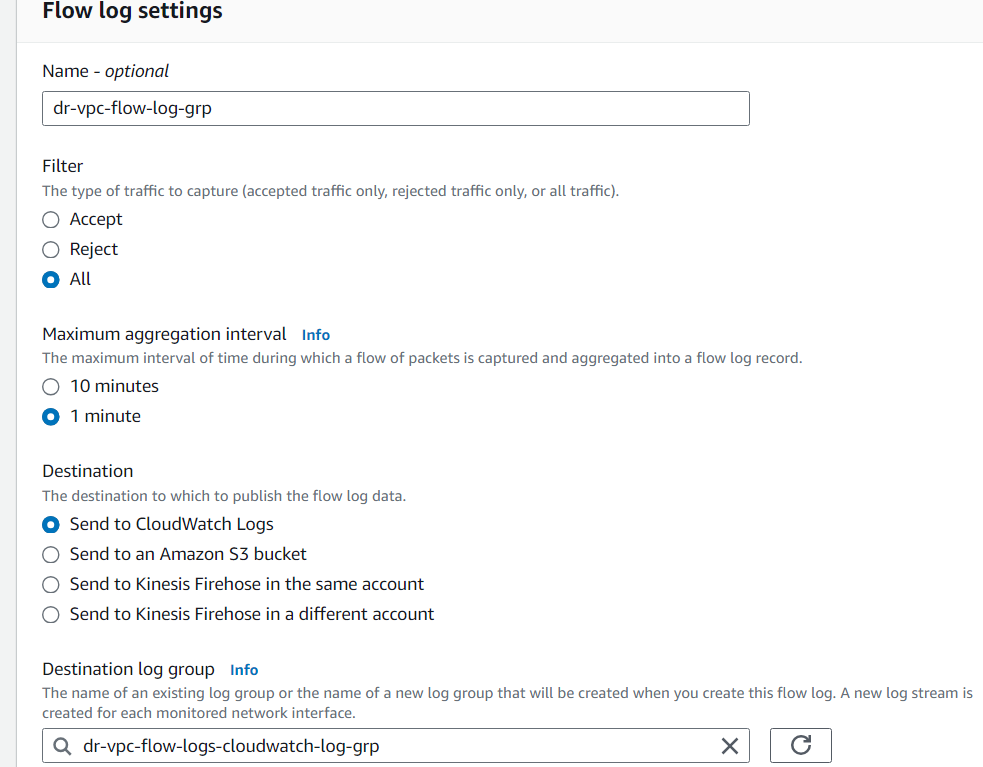
Setupd a cloudwatch log grp to which the vpc flow logs will be forwarded.

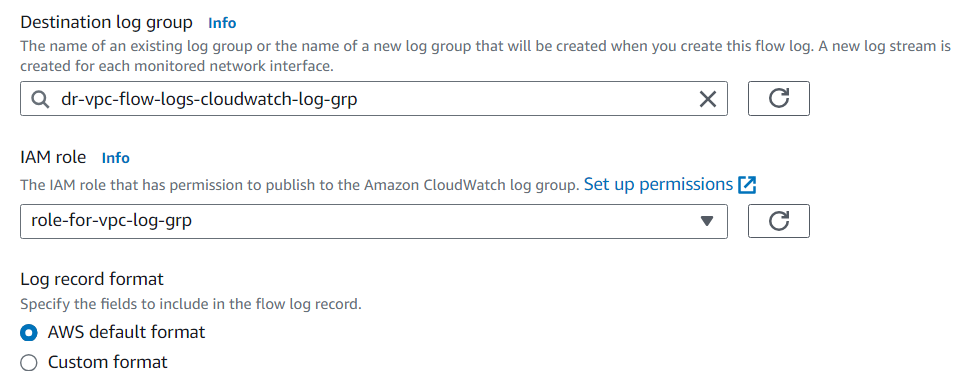


Creating VPC flow logs

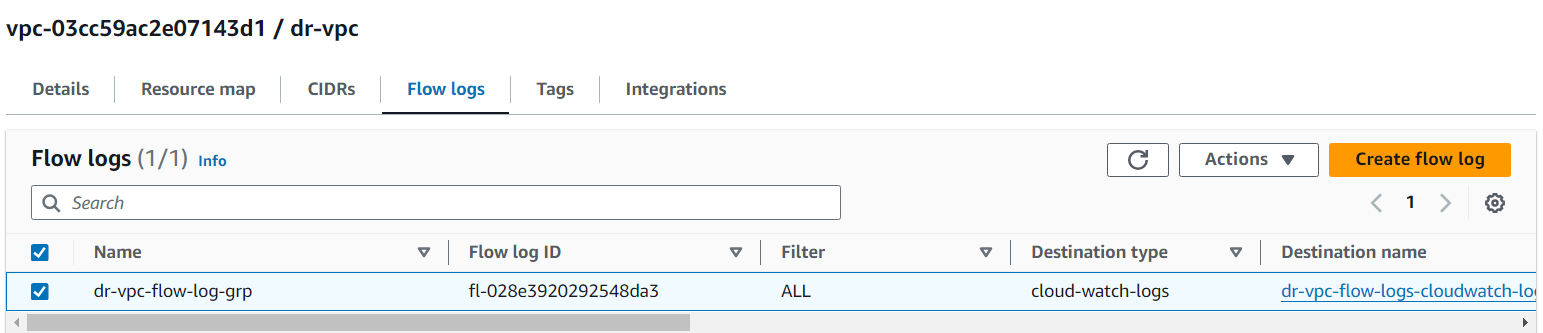


Configured the vpc-logs to transfer the logs to cloudwatch and along with that provided the vpc-logs with necessary IAM role to publish the logs to cloudwatch





Finally, the VPC flow logs is created



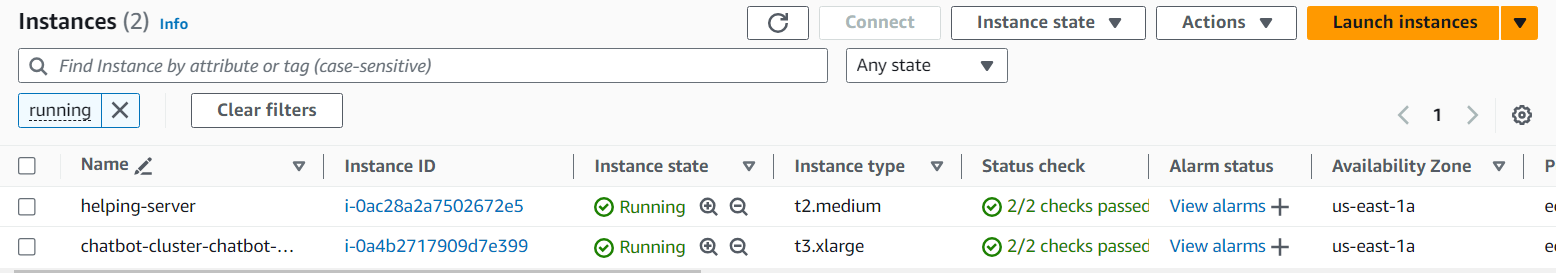
**Develop scripts or use tools to generate realistic data within the VPC environment.**

Using JMeter tool for load Testing in the VPC

**Launch and configure EC2 instances to simulate various components within the network.**

Configured my test-application inside a Linux EC2 to be accessible by external world.

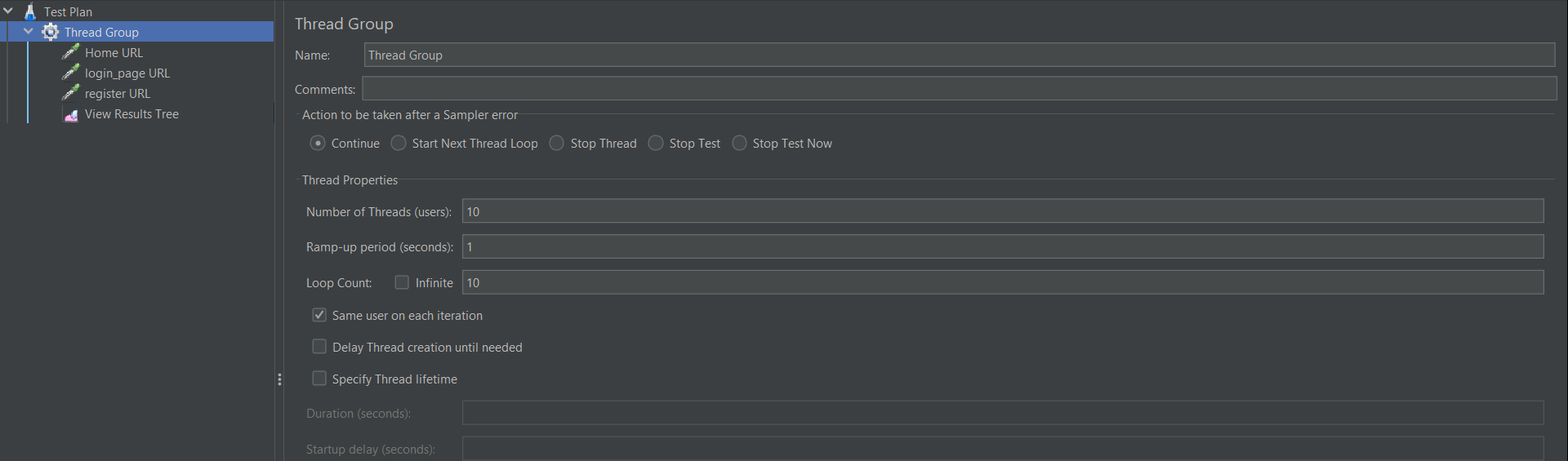
To setup that launched various EC2 instances with different configurations.



**JMeter**

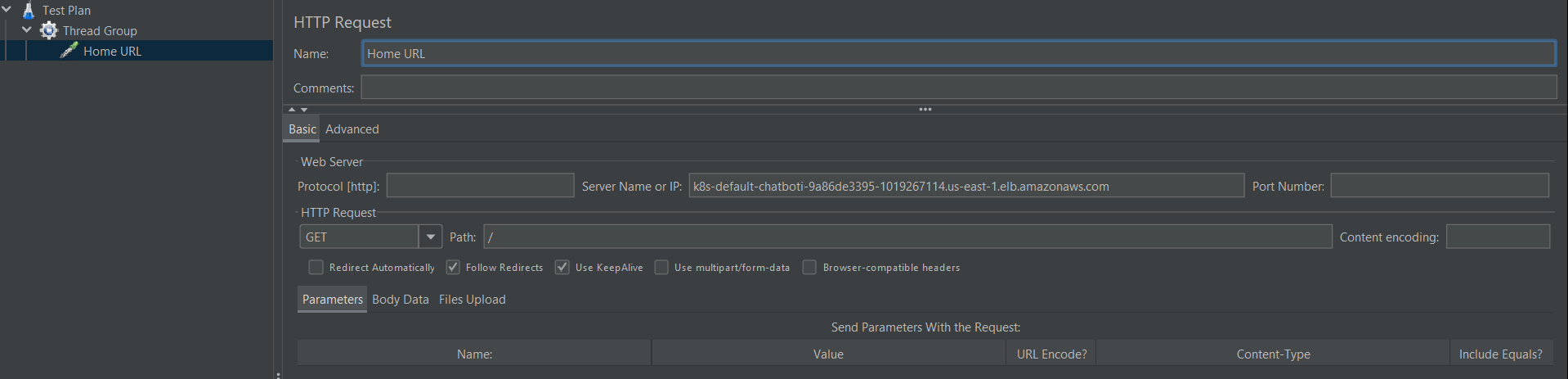
Configured thread grp which is used to mention the no of users which are going to hit the URL and other configurations are done based on the needs.

Thread Grp configurations:

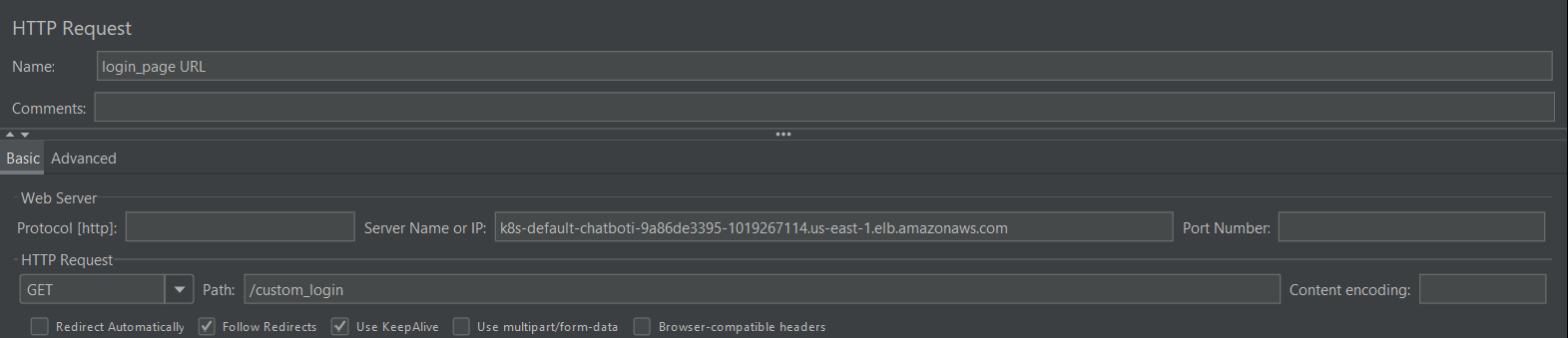


Added the **HTTP Request** section to the Thread Grp (path: **/** )

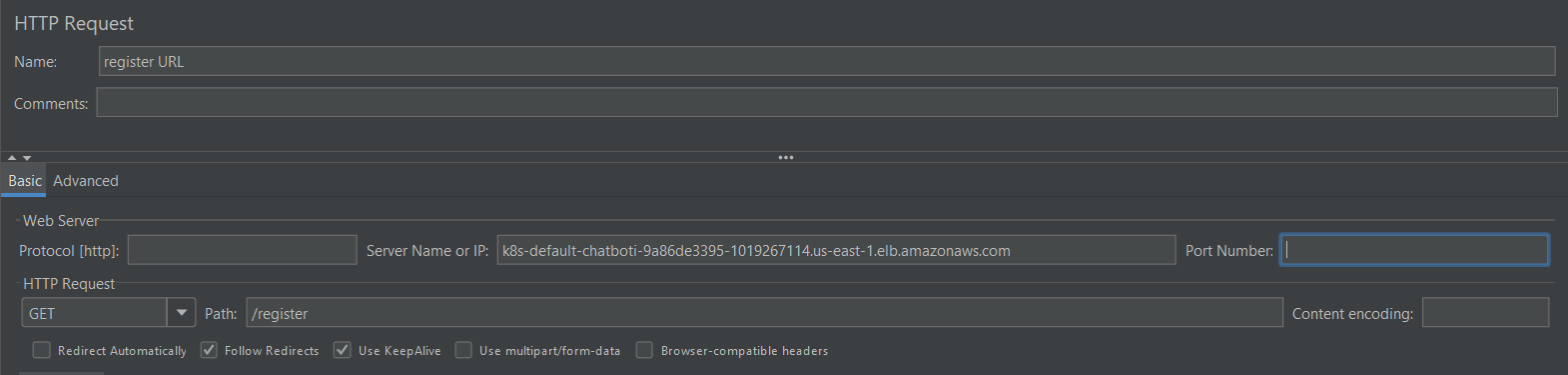
Entered the URL and passed the path on which the load testing is to be done.

****

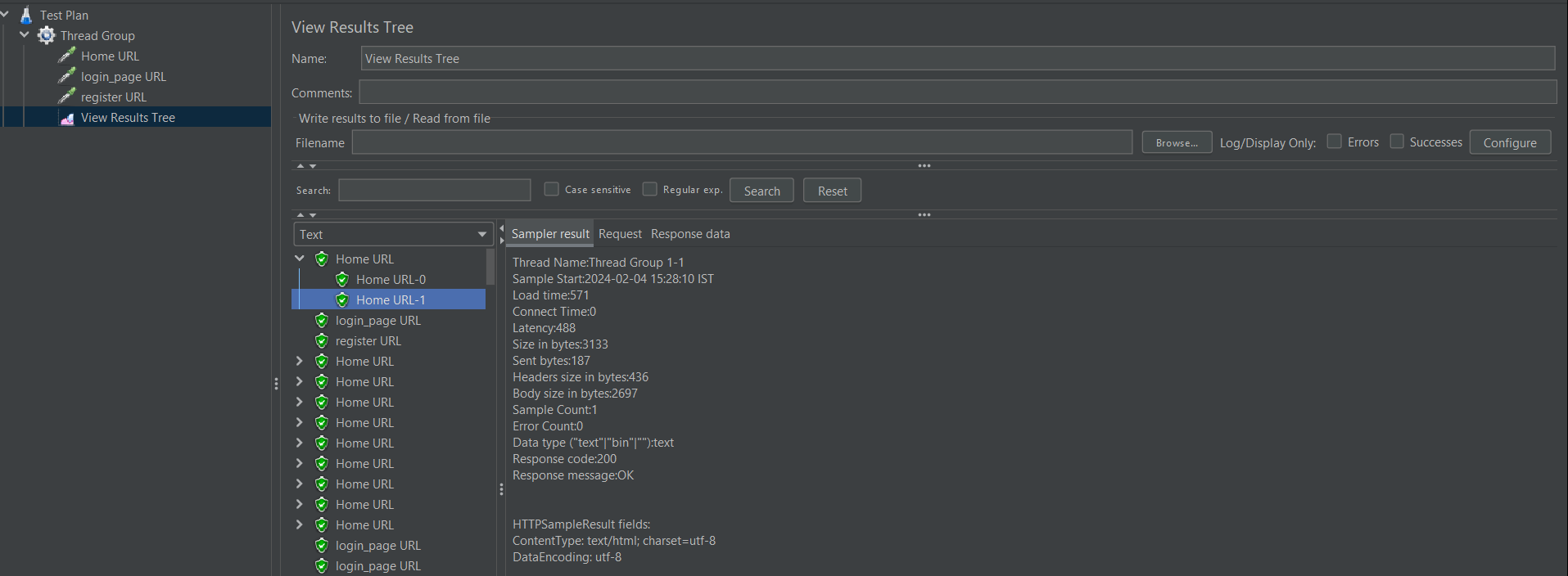
Added another HTTP Request for path /custom\_login



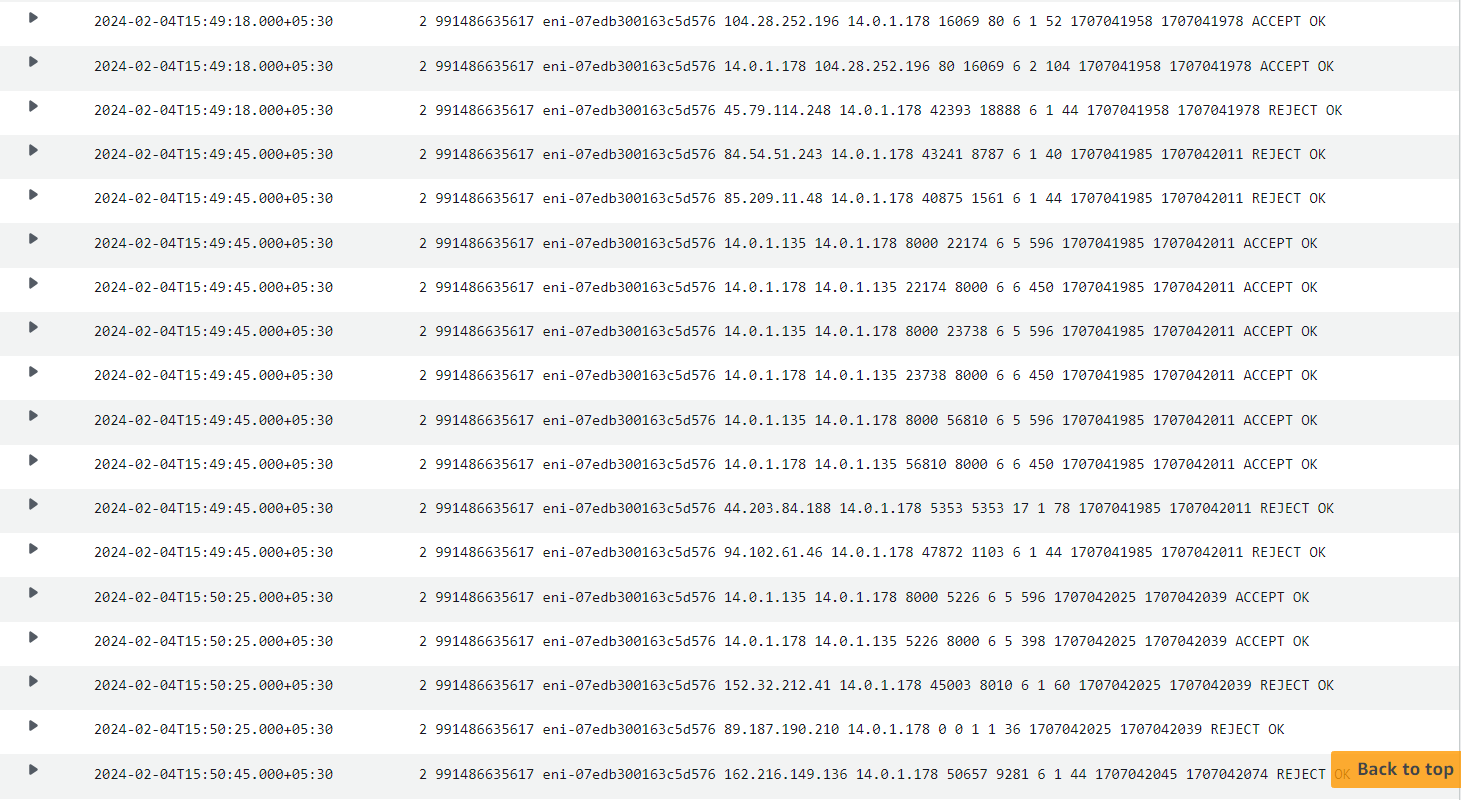
Added another HTTP request for path /register



Added **view Results Tree** in order to view the result of the traffic sent i.e reponse data, response code Latency etc.

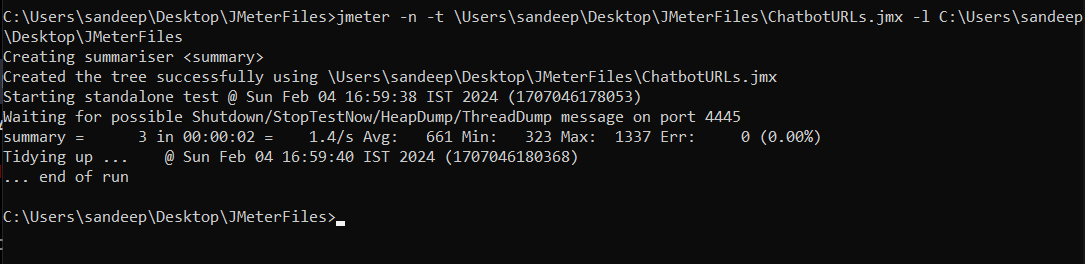


All the traffic that has been flown in and out of the traffic and is forwarded to cloudwatch log grp successfully.

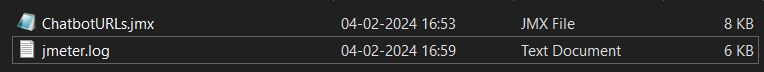


**Note**: Here, the NACL and sg of the ec2 in which application is hosted and the sg of the LB is kept all traffic anywhere and this can be tailored as per the need.

The JMeter is also tested using terminal.

Command: **jmeter -n -t \Users\sandeep\Desktop\JMeterFiles\ChatbotURLs.jmx -l C:\Users\sandeep\Desktop\JMeterFiles**  


The log file is generated successfully as **jmeter.log**

****

**Note:** The cloudwatch log grp is having **VPC flow logs** and this **jmeter.log** file is having different logs.

**Forwarding logs to MongoDB**

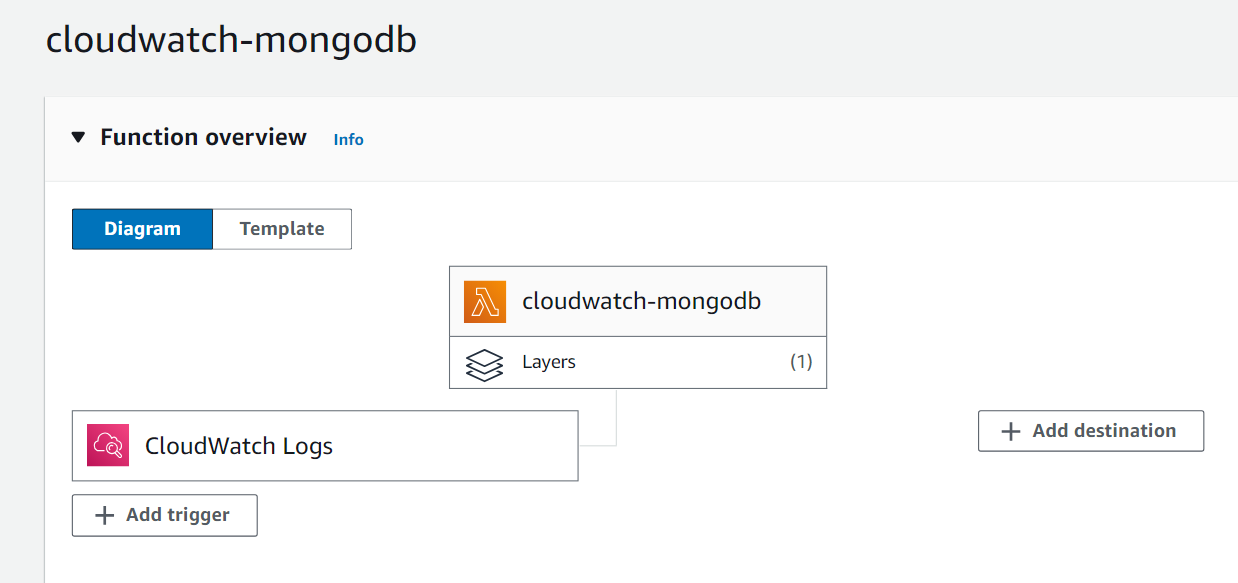
The cloudwatch logs can be forwarded to MongoDB via AWS Lambda by using boto3 python module

The log generated by the running of the JMeter can also be forwarded to MongoDB by running python script in the server where JMeter is running.

**After sending the logs to cloudwatch log grp**, the below lambda script is written to forward the logs to MongoDB in realtime.

CloudWatch log trigger has been setup which triggers the lambda to send the logs to MongoDB.

A pymongo module layer has been added to address the issue of **pymongo not found** while execution of the lambda function.



**For whitelisting scenarios**: https://www.mongodb.com/docs/atlas/manage-connections-aws-lambda/

**Code:**

import base64

import zlib

import json

from pymongo import MongoClient

# MongoDB settings

mongo\_connection\_string = "mongodb+srv://admin:TeleglobalPune2001@cluster0.vpx0ozf.mongodb.net/?retryWrites=true&w=majority"

mongo\_database\_name = "test-db"

mongo\_collection\_name = "vpc\_flow\_logs"

# Hardcoded log group name

log\_group = "dr-vpc-flow-logs-cloudwatch-log-grp"

def lambda\_handler(event, context):

try:

if not log\_group:

print("Invalid or missing log group name.")

return

# Extract the base64-encoded and compressed log data

log\_data = event['awslogs']['data']

decoded\_data = base64.b64decode(log\_data)

decompressed\_data = zlib.decompress(decoded\_data, zlib.MAX\_WBITS | 16)

# Convert the decompressed data to a string

log\_events\_str = decompressed\_data.decode('utf-8')

log\_events = log\_events\_str.splitlines()

# Process and upload logs to MongoDB

upload\_logs\_to\_mongodb(log\_events)

except Exception as e:

print(f"An error occurred: {str(e)}")

def upload\_logs\_to\_mongodb(logs):

client = MongoClient(mongo\_connection\_string)

db = client[mongo\_database\_name]

collection = db[mongo\_collection\_name]

for log in logs:

try:

log\_dict = json.loads(log)

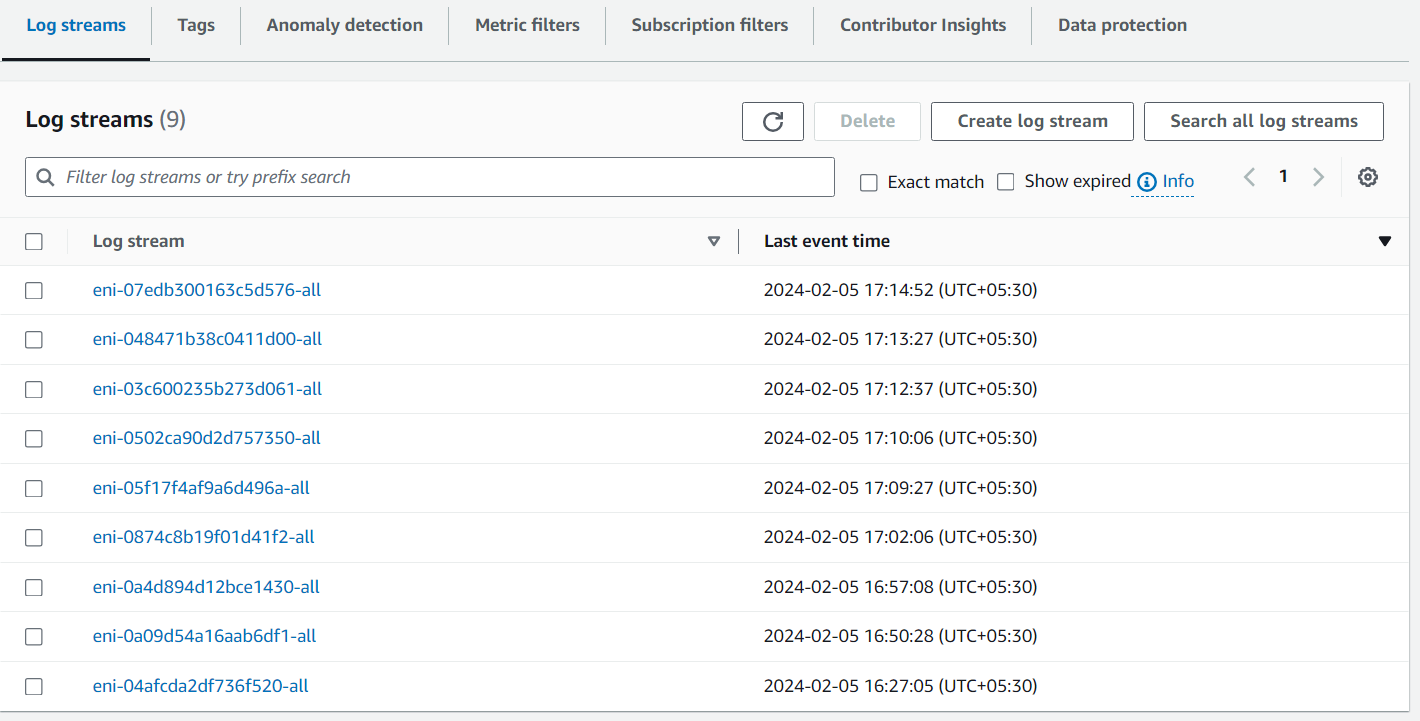
# Insert the log dictionary into MongoDB

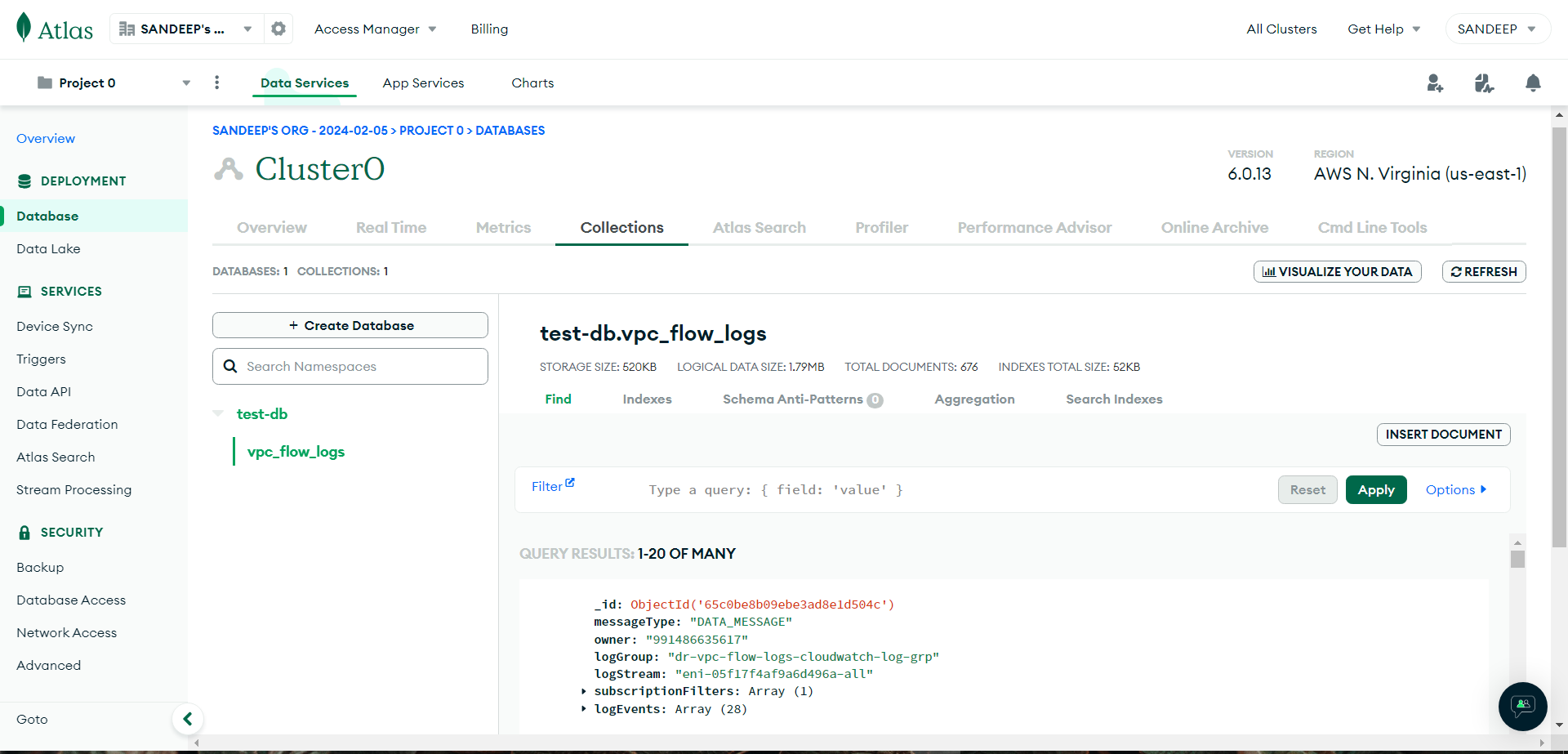
collection.insert\_one(log\_dict)

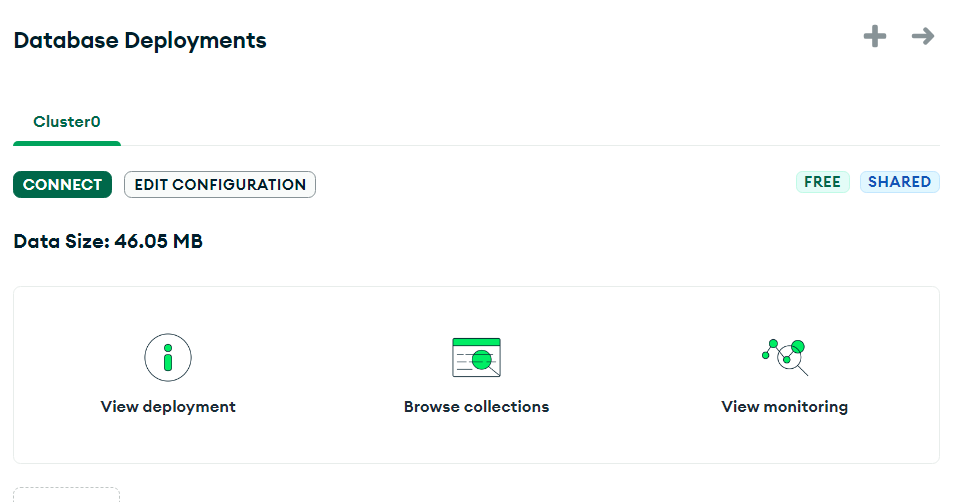
except json.JSONDecodeError:

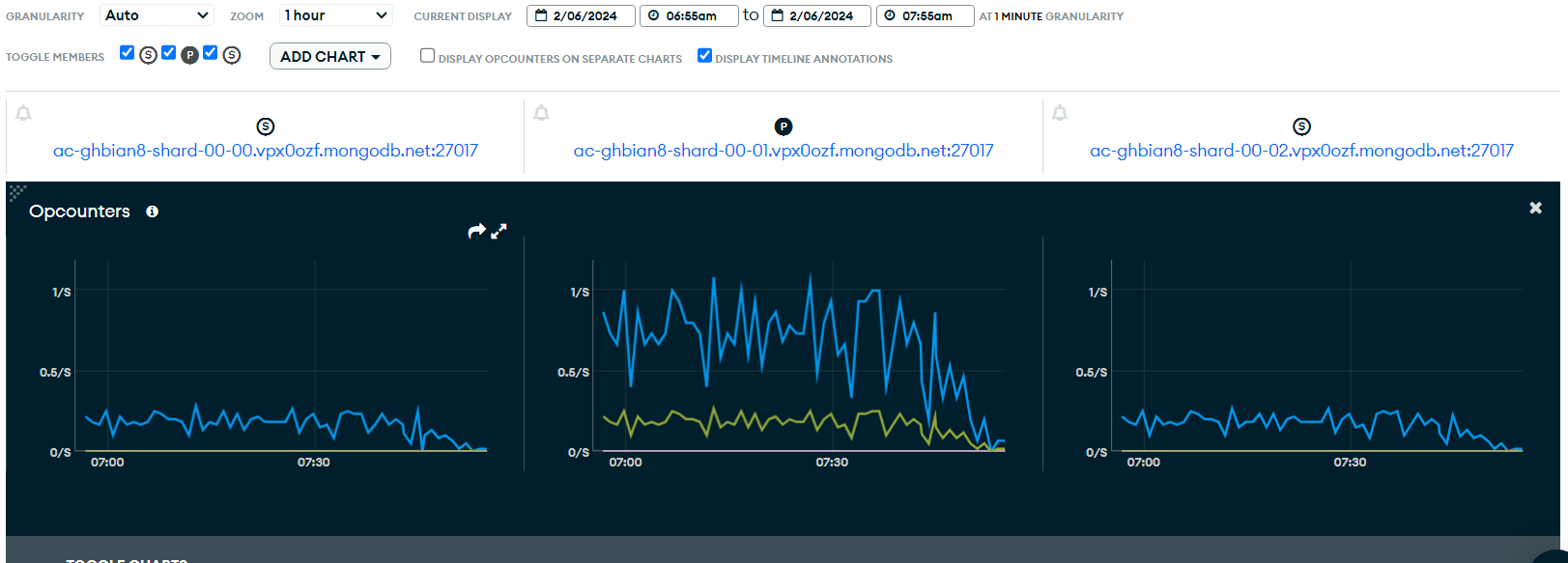
print(f"Invalid JSON log format: {log}")

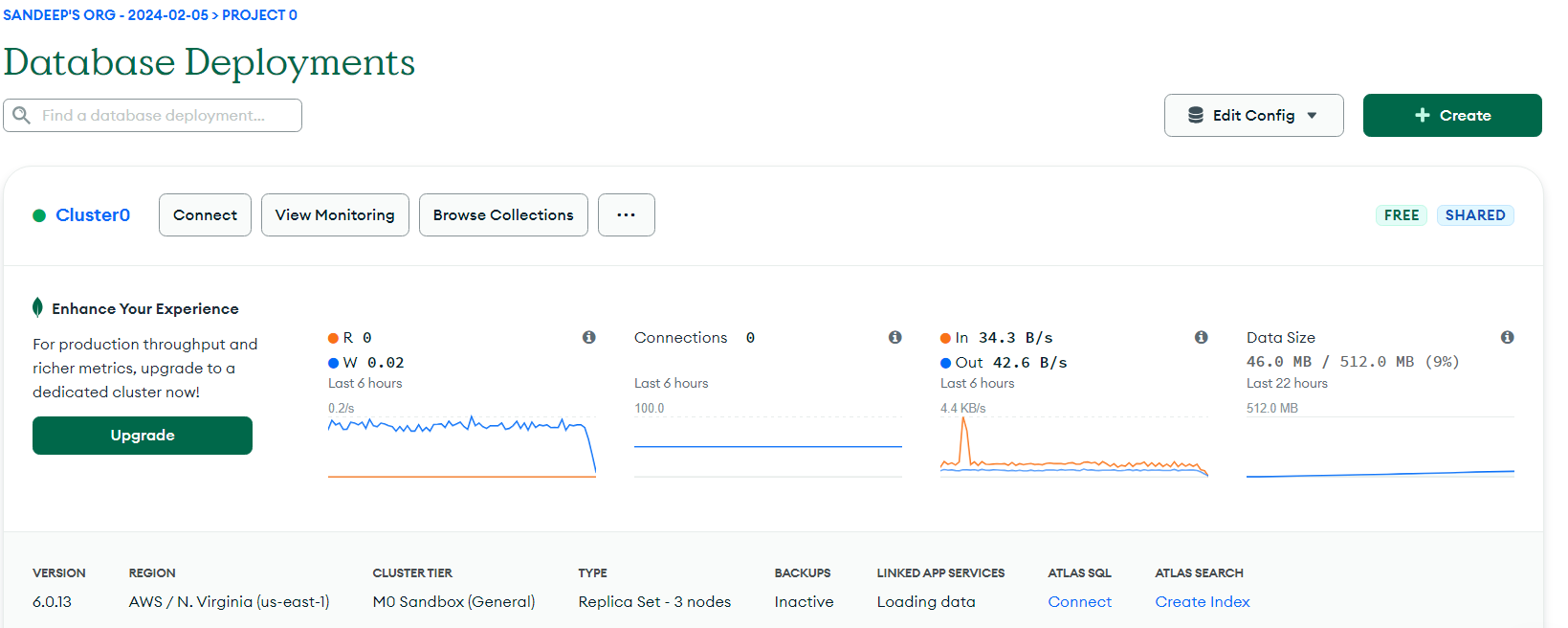
cloudwatch log streams



Used MongoDB Atlas for the POC   








The log can be seen clearly.



The network security of the MongoDB is set to **0.0.0.0/0** so that all the machines having the particular password of the user of the MongoDB (free tier Atlas restrictions, cannot use private endpoint or VPC peering, see the link mentioned above)

**SANDEEP NAYAK**

**Cloud Engineer**