

# Cloud Computing project report

## Group members:

Lorenzo Chicca, Patryk Choma, Nicolò d'Evangelista, Andrea Ghezzi, Rigels Hysaj

## Project title:

Reddit Frontpage



<b>Problem description</b>	<b>3</b>
<b>Solution Design</b>	<b>4</b>
Data collection	4
Webapp	5
<b>Implementation of the solution</b>	<b>6</b>
Data collection script	6
Running the script	8
Database choice and creation	8
Partition key and sort key	10
Web application	11
Docker container	14
<b>Description of the deployment of your solution</b>	<b>14</b>
<b>Test/validation design</b>	<b>16</b>
<b>Conclusions and Future developments</b>	<b>18</b>

## Problem description

Reddit is a social media platform that allows users to share posts within certain categories (called *subreddits*) so that other users *subscribed* to that particular *subreddit* can see them, comment on them and express whether they like the post or not by *upvoting* or *downvoting*.

Due to this system, most people are mainly going to see the most *upvoted* posts, in other words the most popular posts, which get pushed to the top of the page. This is a good system to keep in touch with the latest news, but older posts are often difficult to find.

While the regular website already offers a search option, it offers limited functionality especially with regards to filtering by date of posting, and it is unclear how the results offered are sorted, leading to not being able to find what is being searched. Specific sorting options offered, such as 'top' or 'hot' are also unhelpful or ambiguous. The unsatisfactory search engine within reddit is a well-known problem that our project tries to address.

We also implemented data analysis on the posts, allowing users to find trending topics in a specific community or category.

## Solution Design



Project architecture

The whole solution can be summed up in two different objectives that need to be achieved: **Data Collection** and **Serving the Webapp**.

In the following section we will illustrate the design of our solution for both of these objectives.

### Data collection

- **Overview:** we made use of a **python** script running inside an **Amazon EC2** instance that requests data from reddit by means of reddit's **REST API** and stores it inside an **Amazon DynamoDB** table.
- **Python**
  - Used to write the scripts that orchestrates the retrieval of reddit posts and writing them to our database
- **Amazon EC2**
  - An EC2 instance was used to keep the python script running to collect data.
- **Reddit API**
  - We used the **praw** Reddit API Wrapper for python to download all new post published on reddit
- **DynamoDB**
  - Amazon DynamoDB is a fully managed proprietary NoSQL database. We have stored there all the data taken from Reddit that we needed for our webapp.

## Webapp

- **Overview:** we once again made use of **python** to execute the logic of our webapp by leveraging the **Streamlit** framework to display the data to the user. The webapp runs within a **Docker container** inside an **Amazon EC2** instance. The data is obtained by making use of the **boto3** library, which allows us to interface with various AWS products. In our specific case we used the **DynamoDB** client to retrieve the data we previously collected.
- **Python**
  - Our choice for running the webapp logic and communicating with the database for data retrieval.
- **Streamlit**
  - It allows the creation of apps with simple Python scripts. We used it to create our single page web application that dynamically serves pages to the user.
- **Docker**
  - Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker automates the process of deploying applications within software containers, providing additional abstraction through Linux OS-level virtualization.
- **Amazon EC2**
  - Our webapp is deployed in a Docker Container running within an EC2 instance. This allows for easy auto scaling in case of need.
- **DynamoDB**
  - We retrieve the data we collected using the official boto3 library for python, which is then passed to Streamlit to be displayed to the user.

## Implementation of the solution

In this section we will talk about the implementation of our solution, sharing snippets of code and explaining their purpose. All the code can be found on github at the public repository <https://github.com/ch-pat/aws-reddit>.

### Data collection script

The first step of our project was the data collection. We needed to download reddit posts in order to be able to display them on our web application. To do this we decided to use Python, our programming language of choice, and take advantage of [PRAW: The Python Reddit API Wrapper](#) for ease of use.

```
table = dynamo.get_posts_table()
start_time = time.time()

for i in range(TOTAL_POSTS // LIMIT):
    posts = get_posts()
    count = 0
    with table.batch_writer(overwrite_by_pkeys=['name', 'timestamp']) as batch:
        for post in posts:
            time.sleep(0.2)
            count += 1
            batch.put_item(
                Item={
                    'name': str(post[0]),
                    'title': str(post[1]),
                    'url': str(post[2]),
                    'num_comments': int(post[3]),
                    'timestamp': int(post[4]),
                    'subreddit': str(post[5])
                }
            )
    time.sleep(5)
    print(f"Currently at iteration {i} of {TOTAL_POSTS // LIMIT}")
    print(f"Elapsed time: {time.time() - start_time}")
    print(f"Added {count} posts")
```

Code for uploading posts to the database

The script can theoretically be run continuously to gather data ad infinitum, but due to the nature of this project we applied some restrictions to the code that we will now explain and describe.

```
def get_posts() -> str:
    reddit = praw.Reddit(
        client_id="*****",
        client_secret="*****",
        password="*****",
        user_agent="*****",
        username="*****",
    )
    lista = reddit.subreddit('all').new(limit=LIMIT)

    items = []
    for s in lista:
        items += [(s.name, s.title, s.url, s.num_comments, s.created_utc, s.subreddit)]
    return items
```

Code for downloading the posts from Reddit

## Restrictions

- **Reddit API request limit**

- Reddit limits the number of requests to their API to 30 per minute, where a request can contain up to 100 posts. This is somewhat limiting but it would still allow us to download 100 new posts every 2 seconds, for a total of 3000 new posts per minute. We also noticed a slowdown in the downloads if this limit is not respected, but anyway the bottleneck for data collection lies elsewhere, which brings us to implementing a limit of 500 posts every ~100 seconds, which corresponds to 5 requests per minute, well below the API limit of 30 per minute.

- **DynamoDB write capacity**

- In order to stay within the boundaries of the free plan, we had to limit our writes to the database to be below 5 write capacity units, which correspond to 5 writes per second. Of course there is some leeway with burst capacity, but we cannot rely on it since this is supposed to be a continuously running task. Therefore we implemented static waits inside our code when batch uploading posts to the database to be exactly 5 per second at most, which results in storing 500 posts every 100 seconds as mentioned above. To be safe and avoid huge batches, we also added a 5 second wait after uploading 500 posts.

- **DynamoDB storage limit**

- The above script could in theory run continuously to gather large amounts of data, but for our purposes we decided to stop after gathering what we thought was enough posts. The storage limit for a free plan corresponds to 25GB and since we are only storing text data this is plenty, but we still needed to choose when to stop. Therefore we decided to stop at 1 million posts, which is more than enough to run some data analysis and as a proof of concept.

### **Running the script**

In order to collect the desired amount of data we calculated that the script needed to run for ~58 hours ( $1'000'000 \text{ posts} / (500 \text{ posts} / 105 \text{ seconds}) = 210'000 \text{ seconds} = 58.33 \text{ hours}$ ), and we needed a solution that would allow the script to run for an indefinite amount of time. To do this we decided to use an Amazon EC2 instance and we ran the script in a detached session by using the **screen** command.

We added some print statements to monitor the execution to make sure everything went smoothly and the execution completed as we expected around the 58 hours mark.

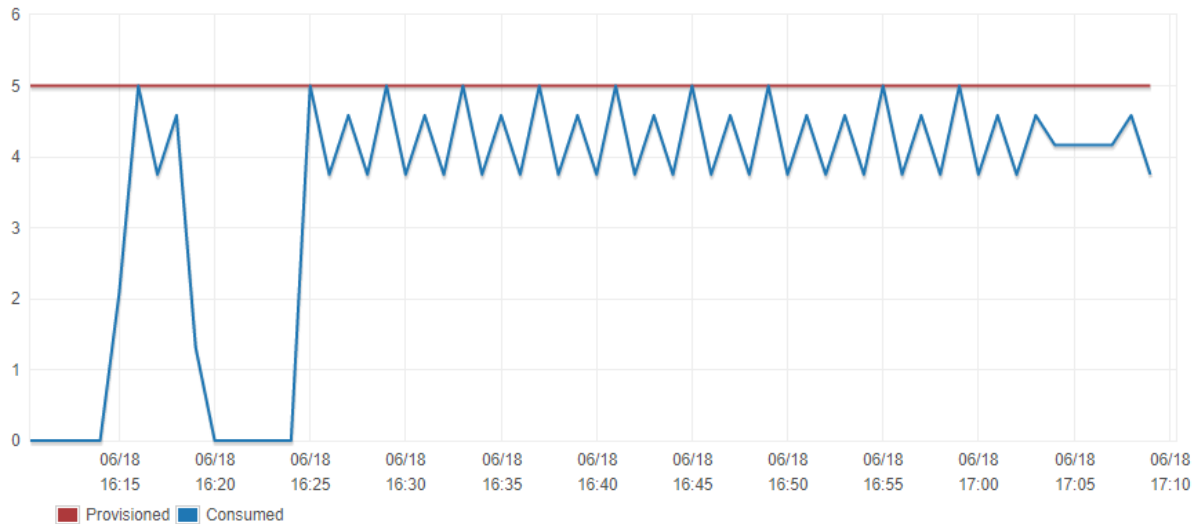
### **Database choice and creation**

For storing the posts we downloaded from reddit, we decided to use **DynamoDB**. The reasoning for this choice is that we needed a database that would be highly scalable, due to the massive influx of new posts. We did not need complex queries, therefore we are fine with using a noSQL database.

With our previously described setup we were able to limit the write load to be at exactly the 5 write capacity units, and if we wanted we could theoretically increase our write capacity limit and download more posts from reddit while maintaining a predictable write load.

On the other hand, as this is a new application, the burst read capacity offered by DynamoDB would be helpful in handling unpredictable traffic to our web application.





Write capacity metric constantly below 5 units during data collection

## Database table structure

<input type="checkbox"/>	name	timestamp	num_comments	subreddit	title	url
<input type="checkbox"/>	t3_nivo04	1621728583	0	ACVillager	[LF] Pompom [...]	https://www.reddit.com/r/ACVillager/comments/nivo04/lf_pompom_ft_nmmts/
<input type="checkbox"/>	t3_nio04h	1621705689	0	Advice	Unfortunate Cir...	https://www.reddit.com/r/Advice/comments/nio04h/unfortunate_circumstan...
<input type="checkbox"/>	t3_njei4k	1621796422	0	Alopa_Al...	14_41	https://www.reddit.com/r/Alopa_Alopa_Alopa/comments/njei4k/14_41/
<input type="checkbox"/>	t3_ni5uy7	1621640030	0	AskReddit	What is your fa...	https://www.reddit.com/r/AskReddit/comments/ni5uy7/what_is_your_favorit.
<input type="checkbox"/>	t3_nia350	1621655444	0	AskReddit	what is your m...	https://www.reddit.com/r/AskReddit/comments/nia350/what_is_your_most_...
<input type="checkbox"/>	t3_niu116	1621723368	0	Barber	Looking for bar...	https://www.youtube.com/watch?v=CRk7Df3HL8U
<input type="checkbox"/>	t3_nhyoi9	1621619794	0	BattlefieldV	The Mastery A...	https://www.reddit.com/r/BattlefieldV/comments/nhyoi9/the_mastery_assig...

## Sneak peek of our *posts* table

Since we are only effectively interested in the new *posts* made by users, we only needed to have a *posts* table.

Inside this table we save the attributes that we are interested in and that will allow us to perform the opportune analysis and queries to implement the functionality of our web application. We will now illustrate the choices we made and what we learned from our mistakes.

## Partition key and sort key

When we first created the table, we had a misconception about the *Primary partition key* that we needed to define for our table. We set our table to have the *name* of the post, which is a unique identifier for the reddit post, as the partition key, because we thought this value needed to be unique within the table as well.

We realized this mistake when we noticed 2 major performance flaws:

- Incomplete results
  - At first we used **Scan** to retrieve data, which allowed us to filter results arbitrarily, making it easy to obtain, for example, all posts of a specific subreddit. But in practice we only obtained a few posts from these Scans. We then learned that a Scan *only retrieves up to 1MB of data and THEN applies the specified filter to the data.*
  - To circumvent this issue, we made use of **sequential Scans** to scan the entire database, but this only led us to our next performance flaw.
- Slow data retrieval
  - Scanning the entire database is an incredibly slow operation and costs a lot of read capacity units. We realized at this point that something must be wrong with how our table was set up, and we learned the purpose of the primary partition key and of the **Global Secondary Index**.

In the end we figured out that if we wanted to retrieve all posts of a specific subreddit, we needed to set the subreddit as the primary partition key. To remedy our initial mistake, we set up the correct partition key within a **Global Secondary Index**. It was a costly operation because it required re-indexing the entire table, but this also allowed us to perform **Queries** rather than Scan operations, which are tremendously **fast and efficient**.

```
def query_table_by_title(query_string: str, subreddit: str):
    table = get_posts_table()
    response = table.query(
        IndexName='subreddit-timestamp-index',
        KeyConditionExpression=Key('subreddit').eq(subreddit),
        FilterExpression=Attr('title').contains(query_string)
    )
    results = []
    items = response['Items']
    for item in items:
        row = [item['title'], item['subreddit'], item['url'], item['timestamp'], item['name']]
        results += [row]
    return results
```

Code for querying the global secondary index

To sum up what we learned, in order to retrieve data efficiently for our application when using DynamoDB, we first need to know what kind of queries are going to be needed, and set up partition keys and sort keys accordingly, and create secondary indexes as needed for different queries.

## Web application

The interface to perform queries is presented to the users by means of a **web application**. The application was developed using **Streamlit**, a python library that aims to provide an intuitive way to present data to the users with an easy to use and clear interface with a responsive design, usable both from mobile and desktop web browsers.

Users can search for subreddits and enter keywords for the title of a post to find all matching posts from the specified subreddit. If the user does not know the exact name of the subreddit, a number of different options matching the entered string will be shown to facilitate the use of the application.

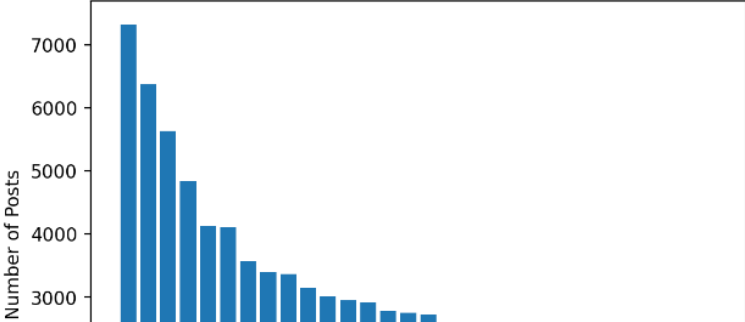
A graph is also shown featuring a ranking of the most popular subreddits by number of posts submitted by users.

Search Subreddit

AskReddit

Search Title

People of reddit



## Landing page of the application

19 results found in subreddit AskReddit

	Title	Subreddit	url	timestamp
0	People of reddit, in w...	AskReddit	https://www.reddit.com...	2021-05-21 18:20:01
1	People of reddit, what...	AskReddit	https://www.reddit.com...	2021-05-21 18:36:01
2	People of reddit who d...	AskReddit	https://www.reddit.com...	2021-05-21 18:50:01
3	People of reddit that ...	AskReddit	https://www.reddit.com...	2021-05-21 19:09:01
4	People of reddit who l...	AskReddit	https://www.reddit.com...	2021-05-21 22:14:01
5	People of reddit, what...	AskReddit	https://www.reddit.com...	2021-05-21 22:26:01
6	People of reddit, what...	AskReddit	https://www.reddit.com...	2021-05-22 01:00:01
7	People of reddit who m...	AskReddit	https://www.reddit.com...	2021-05-22 03:42:01
8	People of reddit what ...	AskReddit	https://www.reddit.com...	2021-05-22 05:46:01
9	People of reddit. Who ...	AskReddit	https://www.reddit.com...	2021-05-22 05:48:01
10	People of reddit, have...	AskReddit	https://www.reddit.com...	2021-05-22 06:09:01

Most used words Word Cloud



### Results of the above query

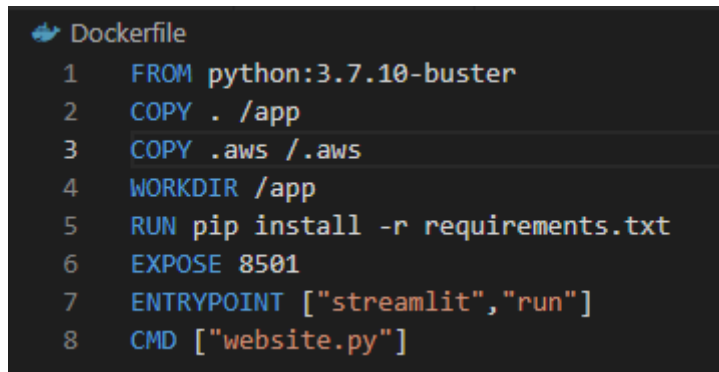
Along with the results of the query, we also decided to show off a *word cloud* that represents the most used words used in posts.

Results are displayed to the users with a table representing a pandas dataframe, due to the integration of **Streamlit** with most popular python data science libraries.

```
website.py > ...
55 if submit_button:
56     subreddits = find_subreddit(form_subreddit)
57     if len(subreddits) == 0:
58         message = "No matching subreddit found"
59     elif len(subreddits) == 1:
60         table_contents = dynamo.query_table_by_title(form_search, subreddits[0])
61         if len(table_contents) == 0:
62             message = f"No posts with that title found in subreddit {subreddits[0]}"
63         else:
64             message = f"{len(table_contents)} results found in subreddit {subreddits[0]}"
65             # Prepare table
66             headings = ['Title', 'Subreddit', 'url', 'timestamp', 'name']
67             table = pd.DataFrame(table_contents)
68             table.columns = headings
69             titles = extract_string_from_titles(table['Title'])
70             wordcloud = WordCloud().generate(titles)
71             # Apply aesthetic changes to table
72             table = table.drop(labels='name', axis=1)
73             table['timestamp'] = table['timestamp'].apply(convert_timestamp)
74     else:
75         message = "Multiple matching subreddits found!\nDid you mean one of these: \n"
76         for s in subreddits:
77             message += f"{s} \n"
78         message = message[:-1]
79
80     st.write(message)
81
82     if type(table) is pd.DataFrame:
83         st.write(table)
84         # Display wordcloud
85         st.markdown("<center>Most used words Word Cloud</center>", unsafe_allow_html=True)
86         fig, ax = plt.subplots(1, 1)
87         plt.imshow(wordcloud)
88         plt.axis("off")
89         st.pyplot(fig)
90
91 else:
92     # Main page before search
93     sub_counts = dynamo.get_subreddit_counts()
```

Code implementation, single page web application design

## Docker container

A screenshot of a code editor showing a Dockerfile. The file is titled 'Dockerfile' with a small icon. It contains 8 lines of code, numbered 1 through 8. The code is as follows:

```
1 FROM python:3.7.10-buster
2 COPY . /app
3 COPY .aws /.aws
4 WORKDIR /app
5 RUN pip install -r requirements.txt
6 EXPOSE 8501
7 ENTRYPOINT ["streamlit","run"]
8 CMD ["website.py"]
```

Dockerfile for the container

In order to easily deploy our solution and to make scaling easier, we decided to deploy our application in a docker container running inside an Amazon EC2 instance. In order to run our application we just need to have python running within the container. We copy the application files and the aws configuration files inside the container. We install the required dependencies and expose the port 8501 to make the website reachable and finally execute the command 'streamlit run website.py' to run the application.

## Description of the deployment of your solution

For the deployment of our solution, we launch 2 Amazon EC2 instances with the default free tier eligible options. In particular, they both run an Amazon Linux 2 AMI volume of type t2.micro. This is more than enough for our tasks that are not particularly demanding. One of the instances will run the *script* tasked with downloading the reddit posts and uploading them to our DynamoDB table, while the other will serve the *web application*.

Since we will be making use of **git** to manage our code, we install it on both our instances by running 'sudo yum install git' and we then clone our code repository to both instances by running 'git clone https://github.com/ch-pat/aws-reddit.git' on each of them. After this is done we need to install the required additional libraries. All dependencies are listed

inside the 'requirements.txt' file, therefore we simply run the 'pip install -r requirements.txt' command on both instances.

In order to make use of the boto3 library to communicate with DynamoDB, we also need to specify configuration and credential files. We store them manually in the default directory where the library expects to find them. Another option was to utilize AMI roles for our instances, but we wanted a solution that was also easy to test on our local machines. We 'cd' into our working directory and proceed with the next steps.

For the *script* instance, we just need to run the reddit.py script and keep it running in a detached session. To do so, we run the 'screen' command followed by 'python reddit.py' to start the script. At this point we can detach the session and it will keep uploading new posts to DynamoDB in real time.

For the *web application* instance, we need a couple more steps, as we also need to use *docker*. First of all we install it by running 'sudo yum install docker' and then we start the service with 'sudo service start docker'. At this point we can build our docker container by simply running 'docker build . -t awsreddit/app' and finally we run the newly created container by running our simple 'run.sh' script after running 'screen' to detach our session.

```
run.sh
1  #!/usr/bin/bash
2  cp ~/.aws/config ~/aws-reddit/.aws/config
3  cp ~/.aws/credentials ~/aws-reddit/.aws/credentials
4  docker run -p 8501:8501 -e AWS_SHARED_CREDENTIALS_FILE="/.aws/credentials" -e AWS_CONFIG_FILE="/.aws/config" awsreddit/app
```

This little script was made in order to deal with the configuration files with a single command. We copy the files inside the scope available to the container and specify their path as an environment variable, that is automatically checked by boto3 if set.

With all this done, the web application is deployed while the DynamoDB table is being continuously populated.

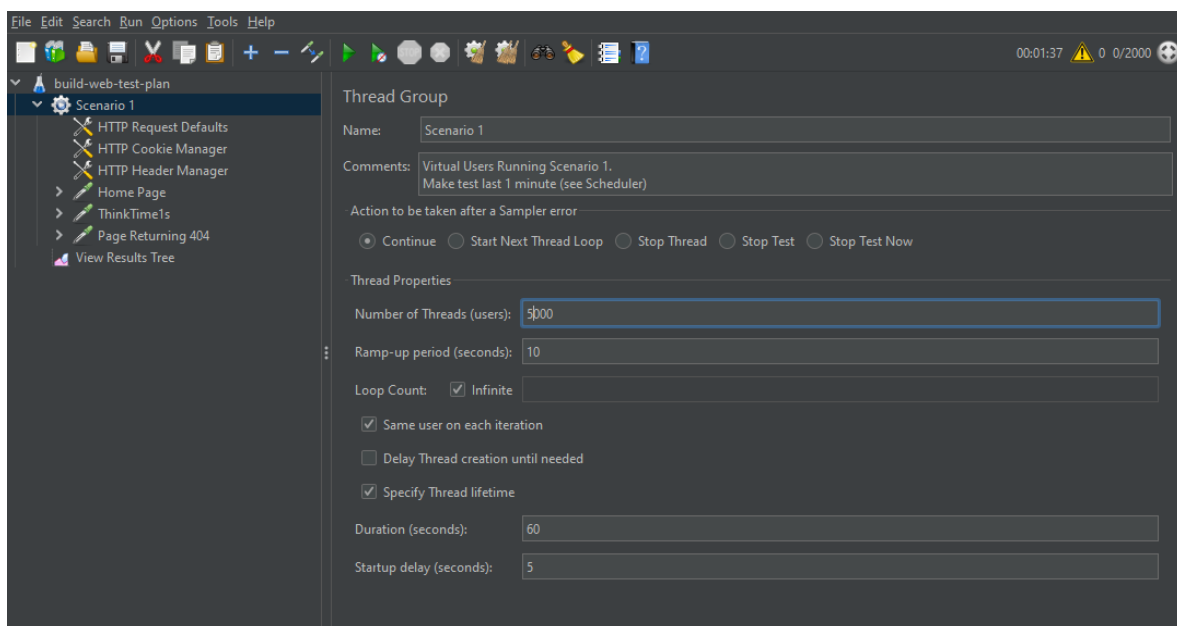
## Auto Scaling and Load Balancing

With the instances set up, we now define the service autoscaling to increase and decrease the amount of running instances depending on the load on the CPU utilization. We opted for a minimum number of 1 task, a desired number of 1 task and a maximum of 3 tasks. We will later re-evaluate this setting if this will not be enough in the testing phase.

## Test/validation design

Now that the web application is properly deployed, we must stress test it to figure out if it will be capable of handling a high number of concurrent users. In order to do so, we first tried with a simple single threaded script that would send as many requests as possible to the web application, but this resulted in, on average, one request every  $\sim 0.28$  seconds, which barely impacted on the CPU utilization of our EC2 instance.

To have a real effect on the CPU utilization, we decided to use a tool made specifically to stress test applications called **JMeter**. We have tried different configurations and eventually settled with 5000 concurrent requests in the span of 10 seconds.



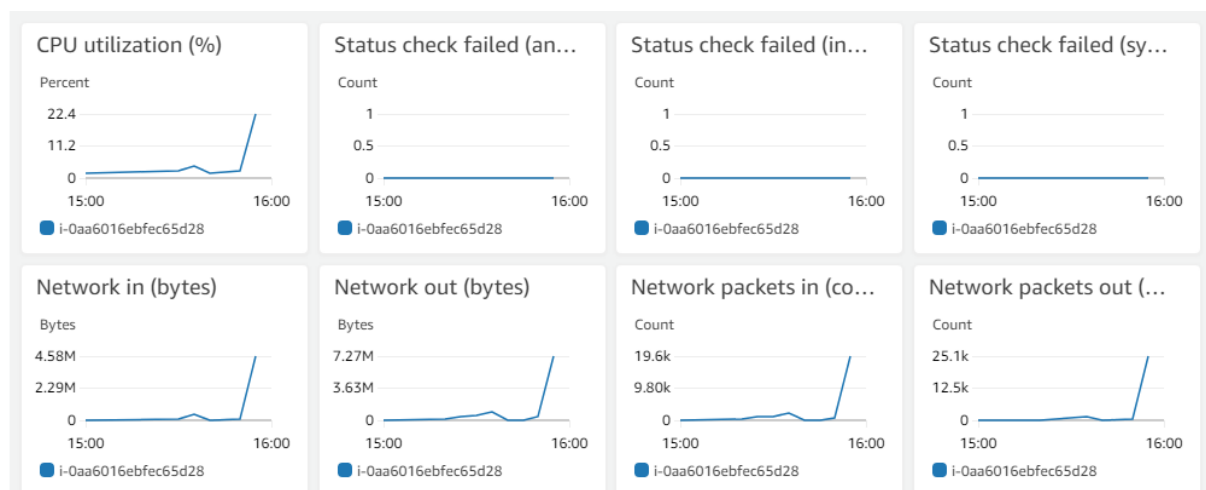
JMeter user interface



Even with such a load, a single EC2 instance had no problem dealing with that many requests, reaching a peak of 22.4% CPU utilization without triggering the autoscaling, which was set to duplicate the instance when average CPU utilization is greater than 30%



Peak CPU utilization under stress



Monitoring details showing spikes in CPU utilization and network usage

Therefore these results suggest that we can keep the previously described auto scaling settings reasonably safely, which allow for many concurrent users to visit the web application.

## **Conclusions and Future developments**

With this project we had the opportunity to try many modern technologies for development at scale, such as virtualization, containers and data analytics. We had to stay within the boundaries of the free tier aws services, which forced us to reason and develop around a 'budget'.

Without such limitations, the application could be expanded to provide more features with more varied queries to DynamoDB and the data collection script could be kept running 24/7 to accumulate all new posts made to reddit to perform more useful analysis.