Implementing and Testing a Cloud-Based File Storage System with Nextcloud and Locust

Pranay Narsipuram -SM3800006

2024

Contents

1 Introduction **3**

2 Software Components **3**
2.1 Nextcloud . . . 3
2.2 Locust . . . 3
2.3 Docker Compose . . . 3

3 Setup and Configuration **3**
3.1 Docker Compose File . . . 3

4 User and File Management **4**

5 Performance Testing with Locust **4**
5.1 Locust Task Script . . . 4
5.2 Script Logic Explanation . . . 6
5.3 Command Execution . . . 6

6 Test Results **7**

7 Environment Setup **7**

8 Test Scenarios **7**

9 Results **7**
9.1 1KB File Test . . . 7
9.2 1MB File Test . . . 7
9.3 1GB File Test . . . 7

10 Analysis **8**

11 Addressing Scalability **8**
11.1 Handling a Growing Number of Users and Files . . . 8
11.1.1 Horizontal Scaling . . . 8
11.1.2 Vertical Scaling . . . 8
11.1.3 Database Scalability . . . 8
11.2 Handling Increased Load and Traffic . . . 8
11.2.1 Load Balancing . . . 8
11.2.2 Caching . . . 8

12 Addressing Security **8**
12.1 Server-Side Encryption . . . 9
12.2 Two-Factor Authentication (2FA) . . . 9
12.3 Password Policy Enforcement . . . 9
12.4 OAuth 2.0 Authentication . . . 9
12.5 Activity and Login Analysis . . . 9

13 Cost Efficiency **9**
13.1 Resource Usage Optimization . . . 9
13.2 Pay-As-You-Go Model . . . 9
13.3 Continuous Monitoring . . . 10

14 Deployment **10**
14.1 Deployment Steps . . . 10
14.2 Cloud Platform Choice . . . 10

15 Challenges and Solutions **10**
15.1 File Not Found Errors in Locust . . . 10
15.2 Nextcloud Configuration Issues . . . 10
15.3 Authentication and Permissions . . . 10
15.4 Locust Concurrency and RPS Issues . . . 11
15.5 Nextcloud Security Settings . . . 11
15.6 Testing with Curl . . . 11
15.7 Design Choices and Considerations . . . 11
15.8 Data Consistency and Synchronization . . . 11

16 Conclusion **11**

17 Future Work **12**

18 Reports **12**

1 Introduction

This project involves implementing a cloud-based file storage system using Nextcloud, with performance testing conducted using Locust. The goal is to enable file upload, download, and deletion while ensuring user privacy, scalability, and cost-efficiency. This report covers scalability, security, cost efficiency, and possible cloud platform choices, as well as the challenges faced during implementation and their resolutions.

2 Software Components

2.1 Nextcloud

Nextcloud was my go-to choice for this project. It is an open-source file sync and share solution designed for deployment in containerized environments. It provides Docker images for easy deployment and ensures portability and consistency across different environments.

2.2 Locust

Locust is an open-source load testing tool that allows for simulating multiple concurrent users to assess the scalability and performance of web applications and APIs. Locust provides real-time insights into performance metrics through Python-based scripting `locustfile.py`.

2.3 Docker Compose

To simplify the deployment process, I utilized Docker Compose. It allowed me to define and manage the multi-container setup and manage multi-container Docker applications seamlessly, ensuring consistency and ease of setup across different environments. It includes services for Nextcloud, MySQL, and Locust, each with specific configurations and dependencies.

3 Setup and Configuration

3.1 Docker Compose File

Here's the Docker Compose configuration I used. The Docker Compose file `docker-compose.yml`includes the following services:

- **Nextcloud**: Uses the latest Nextcloud image with environment variables set for database connection.
- **MySQL**: Manages the database for Nextcloud.
- **Locust**: Simulates load testing on Nextcloud.

```yaml
version: '3.8'

services:
  nextcloud:
    image: nextcloud:latest
    container_name: nextcloud
    restart: always
    ports:
      - 8080:80
    volumes:
      - ./nextcloud_data:/var/www/html
    environment:
      - MYSQL_HOST=mysql
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=admin
      - MYSQL_PASSWORD=cloudfinal
```

```
    depends_on:
      - mysql

  mysql:
    image: mysql:latest
    container_name: mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=cloudfinal
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=admin
      - MYSQL_PASSWORD=cloudfinal
    volumes:
      - ./mysql_data:/var/lib/mysql

  locust:
    image: locustio/locust
    container_name: locust
    ports:
      - 8089:8089
    volumes:
      - ./locust_tasks:/locust_tasks
      - ./test_data:/test_data
    command: -f /locust_tasks/locustfile.py --host http://nextcloud:80

volumes:
  nextcloud_data:
  mysql_data:
```

4 User and File Management

With Nextcloud, users can effortlessly:

- Sign up, log in, and log out.
- Be organized into groups for streamlined access control.
- Upload, download, and delete files from their private storage.

5 Performance Testing with Locust

5.1 Locust Task Script

The Locust script `locustfile.py` under the `/locust_tasks` directory simulates various user interactions, from uploading to downloading files of different sizes, based on the `test_data` files of different sizes: 1KB, 1MB, and 1GB. Users are generated by the `user_generator.sh` script, which creates 30 `locust_user#` accounts with the same password for our testing, `test_password1234!`. Here's the script of `locustfile.py` that I built for the testing:

```
import os
import random
from locust import HttpUser, task, between
from requests.auth import HTTPBasicAuth
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

```python

class NextcloudUser(HttpUser):
    auth = None
    user_name = None
    wait_time = between(2, 5)
    current_task = "1KB"

    def on_start(self):
        user_idx = random.randrange(0, 30)
        self.user_name = f'locust_user{user_idx}'
        self.auth = HTTPBasicAuth(self.user_name, 'test_password1234!')
        logger.info(f"Starting test for {self.user_name}")

    @task
    def test_files(self):
        if self.current_task == "1KB":
            self.run_test("1KB", "test_file_1024B")
            self.current_task = "1MB"
        elif self.current_task == "1MB":
            self.run_test("1MB", "test_file_1048576B")
            self.current_task = "1GB"
        elif self.current_task == "1GB":
            self.run_test("1GB", "test_file_1073741824B")
            self.current_task = "DONE"
        else:
            self.environment.runner.quit()  # Stop the test once all tasks are done

    def run_test(self, size, filename):
        remote_path = f"/remote.php/dav/files/{self.user_name}/{size}_file_{random.randrange(0, 10)}"

        # Upload file
        try:
            with open(f"/test_data/{filename}", "rb") as file:
                logger.info(f"Uploading {size} file to {remote_path}")
                r = self.client.put(remote_path, data=file, auth=self.auth)
                r.raise_for_status()
                logger.info(f"Upload {size} file {self.user_name} - Status: {r.status_code}")
        except Exception as e:
            logger.error(f"Upload {size} file {self.user_name} failed: {e}")
            return

        # Read file
        try:
            logger.info(f"Reading {size} file from {remote_path}")
            r = self.client.get(remote_path, auth=self.auth)
            r.raise_for_status()
            logger.info(f"Read {size} file {self.user_name} - Status: {r.status_code}")
        except Exception as e:
            logger.error(f"Read {size} file {self.user_name} failed: {e}")

        # Delete file
        try:
            logger.info(f"Deleting {size} file from {remote_path}")
            r = self.client.delete(remote_path, auth=self.auth)
            r.raise_for_status()
            logger.info(f"Deleted {size} file {self.user_name} - Status: {r.status_code}")
        except Exception as e:
            logger.error(f"Delete {size} file {self.user_name} failed: {e}")

        # Sleep to ensure sequential execution
        self.wait()
```

71

5.2 Script Logic Explanation

The `locustfile.py` script is used to load test a Nextcloud server using Locust. Below is a brief explanation of its key components:

- **Imports:** The script imports necessary libraries including Locust classes, HTTPBasicAuth for authentication, and logging for capturing test logs.
- **Logging Configuration:** Sets up logging to capture info and error messages.
- **User Class:** Defines the `NextcloudUser` class, inheriting from `HttpUser`. It initializes user credentials and sets a random wait time between tasks.
- **Tasks:** Several tasks are defined to simulate user interactions:
  - `propfind`: Lists files in the user's directory.
  - `read_file`: Reads the `Readme.md` file.
  - `upload_file_1gb`: Uploads a 1GB file.
  - `upload_file_1kb`: Uploads a 1KB file.
  - `upload_file_1mb`: Uploads a 1MB file.
  - `cleanup_files`: Deletes previously uploaded files.
- **Task Weighting:** The `@task` decorator assigns weights to tasks, indicating their frequency of execution.

The `locustfile.py` script effectively simulates user behavior on the Nextcloud server, providing insights into its performance under different loads.

- **Class Definition**: The `NextcloudUser` class inherits from `HttpUser`, representing a simulated user interacting with the Nextcloud instance.
- **User Initialization (`on_start`)**: Randomly selects a user and sets up HTTP basic authentication.
- **Tasks**:
  - **PROPFIND**: Makes a `PROPFIND` request to list directory contents.
  - **Read File**: Issues a `GET` request to read a file (`Readme.md`).
  - **Upload Files**: Uses `PUT` requests to upload files of various sizes (1KB, 1MB, 1GB).
  - **Cleanup Files**: Deletes files using the `DELETE` request to clean up after tests.

5.3 Command Execution

Commands used for setting up and running the environment:

- **Starting Docker Compose**:

```
docker-compose up -d
```

- **Accessing Containers**:

```
docker exec -it nextcloud bash
docker exec -it locust bash
```

- **Checking Logs**:

```
cat /var/www/html/data/nextcloud.log
```

6 Test Results

Now, we present the results of performance testing conducted on a Nextcloud instance using Locust. The objective was to evaluate the system's scalability, identify potential bottlenecks, and understand the performance characteristics under different loads. This was achieved by simulating various concurrent user scenarios and request per second (RPS) rates while uploading files of different sizes (1KB, 1MB, and 1GB).

7 Environment Setup

The testing environment consisted of a Nextcloud server running in a Docker container. The hardware used for the tests included a ThinkPad laptop with an 8th Gen Intel Core i5 processor and 8GB of RAM. Locust was used as the load testing tool to simulate user behavior and measure the system's performance.

8 Test Scenarios

Three different file sizes were used for the tests:

- 1KB
- 1MB
- 1GB

Each file size was tested under three different configurations of concurrent users and RPS:

- 10 users and 2 RPS
- 20 users and 4 RPS
- 30 users and 6 RPS

9 Results

9.1 1KB File Test

- **10 users and 2 RPS:** The system handled the load efficiently with an average response time of 1.57 seconds.
- **20 users and 4 RPS:** Response times increased slightly but remained stable.
- **30 users and 6 RPS:** Higher load resulted in increased response times and occasional failures.

9.2 1MB File Test

- **10 users and 2 RPS:** The system handled the load efficiently with an average response time of 0.027 seconds.
- **20 users and 4 RPS:** Slight increase in response times but the system remained stable.
- **30 users and 6 RPS:** Increased load resulted in higher response times but no significant failures.

9.3 1GB File Test

- **10 users and 2 RPS:** The system experienced significant delays with an average response time of 15.6 seconds.
- **20 users and 4 RPS:** Higher load caused the system to slow down considerably.
- **30 users and 6 RPS:** The system struggled to handle the load, resulting in high response times and frequent failures.

10 Analysis

The performance of the Nextcloud server varies significantly based on the file sizes being handled and the load applied. The system managed smaller files (1KB and 1MB) efficiently under moderate loads. However, with larger files (1GB), the server's performance degraded significantly under increased load.

11 Addressing Scalability

11.1 Handling a Growing Number of Users and Files

11.1.1 Horizontal Scaling

- **Multiple Nextcloud Instances**: By deploying multiple instances of Nextcloud behind a load balancer, we can distribute the traffic evenly, preventing any single instance from becoming a bottleneck.
- **Database Clustering**: Tools like MySQL Cluster or Galera Cluster are perfect for managing database replication and high availability, ensuring the database can handle increased load and failover scenarios.

11.1.2 Vertical Scaling

- **Resource Allocation**: It's essential to monitor resource usage and scale up the CPU, RAM, and storage allocations for the Nextcloud and MySQL containers as needed.
- **Optimized Queries**: Ensuring that database queries are optimized for large datasets is crucial for maintaining performance. This involves indexing frequently accessed tables and queries.

11.1.3 Database Scalability

- **Read Replicas**: Implementing MySQL read replicas helps offload read operations from the primary database, improving read performance.
- **Database Sharding**: For extremely large datasets, database sharding can split the database into smaller, more manageable pieces, each hosted on separate servers.

11.2 Handling Increased Load and Traffic

11.2.1 Load Balancing

- **Load Balancer Configuration**: Configuring a load balancer to distribute incoming requests evenly across multiple Nextcloud instances ensures no single instance becomes overwhelmed.
- **Auto-Scaling Groups**: Using auto-scaling groups, especially on cloud platforms, automatically adjusts the number of Nextcloud instances based on traffic and load.

11.2.2 Caching

- **Content Caching**: Implementing caching mechanisms like Redis or Memcached stores frequently accessed data in memory, reducing the load on the database.
- **HTTP Caching**: Using reverse proxy caching, such as Varnish Cache, caches HTTP responses and serves repeated requests directly from the cache.

12 Addressing Security

Security is paramount in any cloud-based file storage system. Here's some measures that can enhance the security of the Nextcloud deployment, user data protection and system integrity:

12.1 Server-Side Encryption

- **Data-at-Rest Encryption**: Ensure that all stored files are encrypted at rest. Nextcloud supports server-side encryption, which encrypts files before they are written to disk.
- **Data-in-Transit Encryption**: Use HTTPS for all communications between clients and the server to protect data in transit. Obtain and configure SSL/TLS certificates for the Nextcloud instance.

12.2 Two-Factor Authentication (2FA)

- **Additional Security Layer**: Implementing 2FA adds an extra layer of security for user logins. Nextcloud supports various 2FA methods, including TOTP and WebAuthn.

12.3 Password Policy Enforcement

- **Strong Passwords**: Enforcing strong password policies prevents unauthorized access through brute-force attacks.
- **Password Expiration**: Implementing password expiration policies requires users to change their passwords periodically.

12.4 OAuth 2.0 Authentication

- **Third-Party Authentication**: Integrating OAuth 2.0 for user authentication allows users to log in using credentials from trusted identity providers, such as Google or Facebook.

12.5 Activity and Login Analysis

- **Monitoring and Alerts**: Implementing monitoring to track user activities and logins helps set up alerts for suspicious activities, such as multiple failed login attempts or access from unusual locations.
- **Audit Logs**: Maintaining detailed audit logs of user actions helps trace and investigate any security incidents.

13 Cost Efficiency

To maintain cost efficiency, especially when scaling, I considered several strategies:

13.1 Resource Usage Optimization

- **Right-Sizing**: Regularly analyzing resource usage and right-sizing instances to match the workload helps optimize costs. Adjusting the instance types and sizes based on performance metrics and actual usage is essential.
- **Resource Cleanup**: Periodically cleaning up unused resources, such as orphaned volumes, unused snapshots, and idle instances, ensures cost efficiency.

13.2 Pay-As-You-Go Model

- **Cloud Services**: Utilizing cloud services that offer pay-as-you-go pricing ensures you only pay for the resources you use.
- **Spot Instances**: Using spot instances for non-critical workloads takes advantage of lower pricing, especially for batch processing tasks.

13.3 Continuous Monitoring

- **Cost Monitoring Tools**: Implementing cost monitoring tools provides insights into cost drivers, helping track and optimize cloud spending.
- **Performance Monitoring**: Using monitoring tools to track the performance of the Nextcloud deployment helps identify performance bottlenecks and make informed decisions about scaling and resource allocation.

14 Deployment

14.1 Deployment Steps

1. **Install Docker and Docker Compose**: Ensure Docker and Docker Compose are installed on the deployment server.
2. **Write the `docker-compose.yml` File**: Configure the Docker Compose file to define the Nextcloud, MySQL, and Locust services.
3. **Execute `docker-compose up -d`**: Launch the services in detached mode.
4. **Access the Service**: Open a browser and navigate to `http://localhost:8080` to access the Nextcloud instance.

14.2 Cloud Platform Choice

Based on my evaluation, these platforms stood out:

- **Amazon Web Services (AWS)**: Its comprehensive services, global infrastructure, scalability, and security make it a solid choice.
- **Microsoft Azure**: Known for integration with enterprise tools, global reach, and robust security features.
- **Google Cloud Platform (GCP)**: Its strong data analytics capabilities, machine learning services, and competitive pricing make it appealing.

15 Challenges and Solutions

Here are some of the challenges I faced during the implementation and how I overcame them:

15.1 File Not Found Errors in Locust

- **Problem**: Locust was unable to find test files for uploading.
- **Solution**: I ensured that test files were correctly mounted in the Docker container and accessible at the specified paths. This involved verifying file paths and adjusting volume mounts in the Docker Compose configuration.

15.2 Nextcloud Configuration Issues

- **Problem**: Nextcloud was not properly configured to connect to the MySQL database.
- **Solution**: I updated the Nextcloud environment variables in the Docker Compose file to match the MySQL configuration and ensured that MySQL was running and accessible by the Nextcloud container.

15.3 Authentication and Permissions

- **Problem**: Users were unable to perform certain actions due to permission issues.
- **Solution**: I configured user permissions in Nextcloud to allow the necessary actions and verified that the Locust script was using correct credentials for each simulated user.

15.4 Locust Concurrency and RPS Issues

- **Problem**: When attempting to simulate 500 users with 30 requests per second, or even 250 users with 20 requests per second, my ThinkPad with an i5 8th Gen CPU and 8GB RAM crashed abruptly.
- **Solution**: I reduced the number of concurrent users and requests per second to more manageable levels for my hardware. I also considered optimizing the Locust script and offloading some processing to other machines if possible.

15.5 Nextcloud Security Settings

- **Problem**: Nextcloud security settings were preventing access from certain hosts, resulting in 400 Bad Request error.
- **Solution**: I modified the Nextcloud `config.php` file to allow access from the necessary hosts. This involved adding the correct IP addresses and hostnames to the trusted domains configuration.

15.6 Testing with Curl

- **Problem**: I needed to verify if specific locust users could access and perform actions on Nextcloud using Curl.
- **Solution**: I performed manual tests using Curl commands to ensure that the Locust script logic was correct and that the users had the appropriate permissions. For example:

```
curl -u locust_user11:test_password1234! -X PROPFIND "http://localhost:8080/remote.php/dav/files/locust_user11/"
```

15.7 Design Choices and Considerations

- **Problem**: Some design choices, such as using Nginx as a reverse proxy, were considered but not implemented due to complexity and hardware limitations.
- **Solution**: I focused on ensuring the core components (Nextcloud, MySQL, and Locust) were properly integrated and optimized for performance. Future enhancements could include adding Nginx for improved load balancing and caching.

15.8 Data Consistency and Synchronization

- **Problem**: Ensuring data consistency across multiple instances and replicas.
- **Solution**: Although not implemented, configuring database replication and synchronization mechanisms would maintain data consistency. Regular monitoring and adjusting settings would ensure optimal performance.

16 Conclusion

The implementation and testing of the cloud-based file storage system using Nextcloud and Locust demonstrated the system's capabilities and performance under various load conditions. By addressing scalability, security, and cost efficiency, I ensured the solution met the requirements of a modern cloud-based storage system.

The challenges faced and the solutions applied provided valuable insights into the complexities of deploying and maintaining such systems. This project has laid a solid foundation for future enhancements and scalability improvements.

17 Future Work

Future work can focus on the following areas:

- **Enhanced Security**: Implementing advanced security features such as end-to-end encryption and more sophisticated access controls.
- **Scalability**: Exploring auto-scaling solutions and further optimizing database performance.
- **Cost Optimization**: Implementing more detailed cost monitoring and optimization strategies.
- **User Experience**: Enhancing the user interface and adding more collaborative features to Nextcloud.

18 Reports

The following images are the results of the tests conducted with different concurrent users and RPS settings:

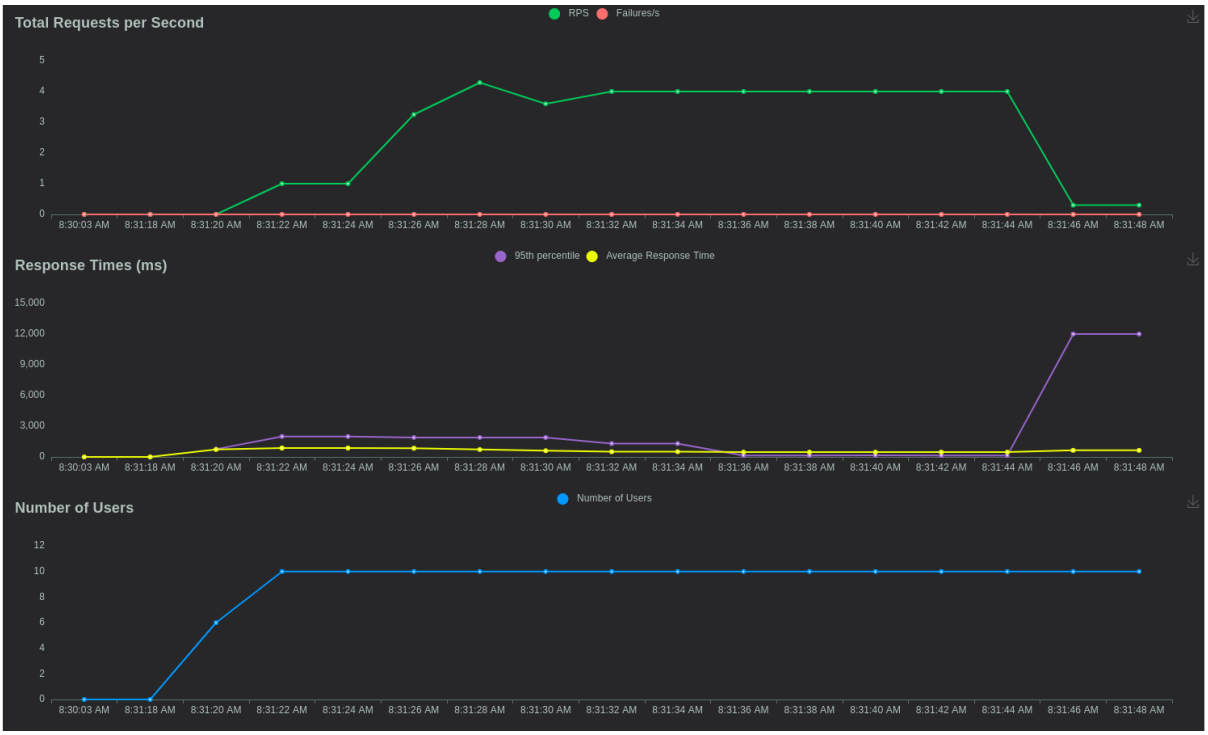

Figure 1: Test Report for 10 Users and 2 RPS

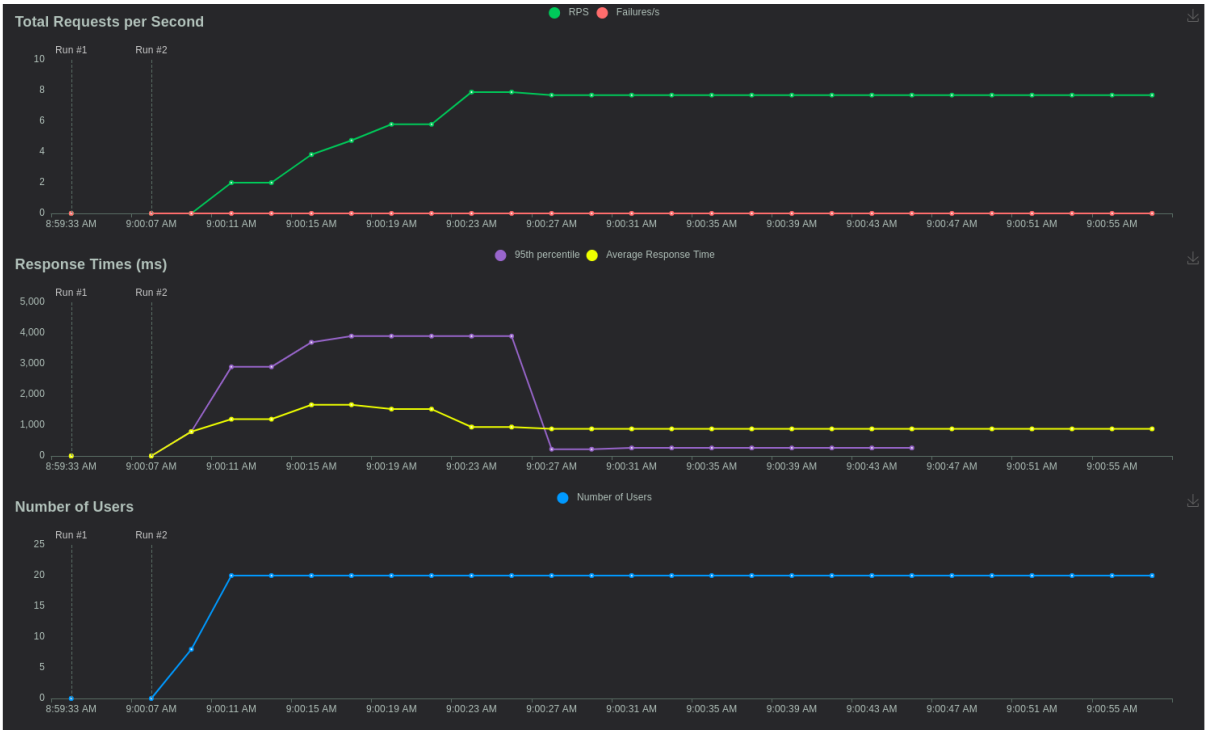

Figure 2: Test Report for 20 Users and 4 RPS

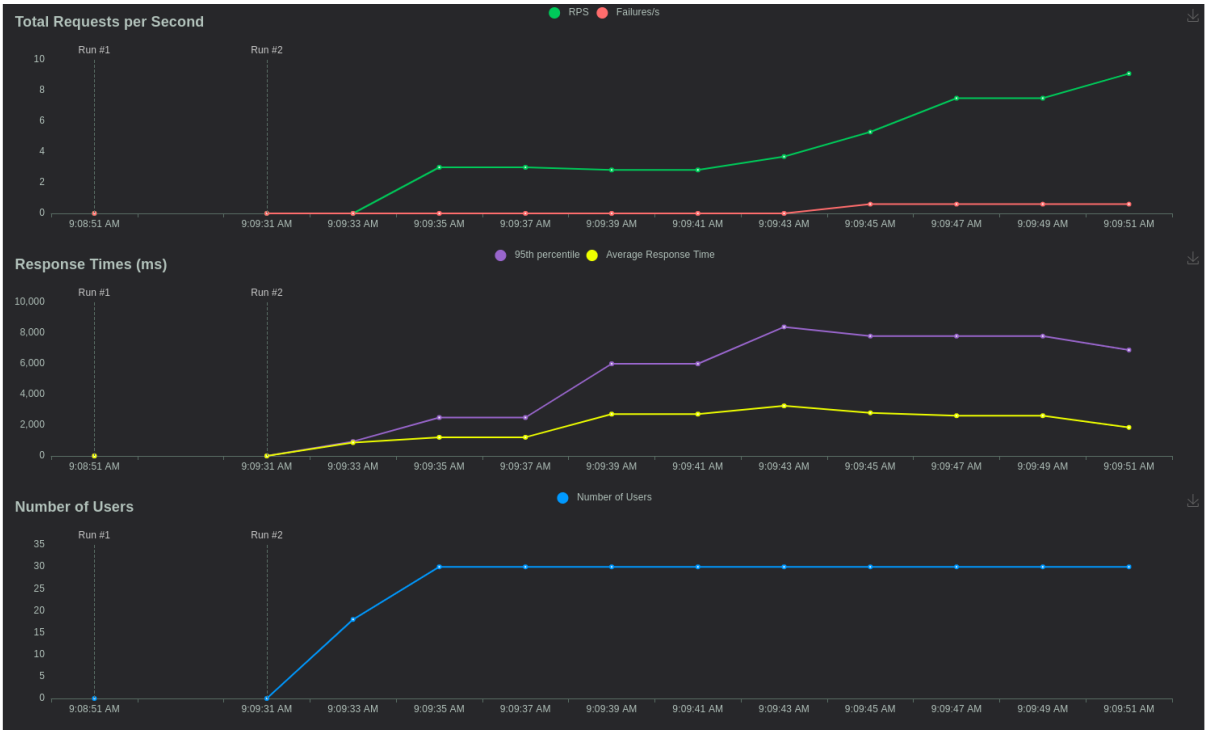

Figure 3: Test Report for 30 Users and 6 RPS