JUNE 17, 2024

PERFORMANCE TEST PLAN FOR {JSON} Placeholder

Majed Alssaidi

THE ROAD TO A POSITIVE USER EXPERIENCE

OVERVIEW

An application is as fast as it's slowest component. Performance testing focuses on exposing system limits and weaknesses. With full understanding of the system's limits, users and product owners can manage the system and user expectations to provide the most pleasant experience to the end user.

For the {JSON} Place Holder api, this document will define the approach, assumptions, expectations, risks and involved resources in order to verify the system's performance. It will also provide an overview of the approach for the performance testing that we will be taking to measure the performance and identify the system limits.

SCOPE OF THE TEST

IN SCOPE:

BACKEND

The test will focus on validating the API's performance, the backend systems' health and database responsiveness.

PERFORMANCE TEST

These tests will focus on validating that the system is functioning as expected under normal load, high load, unusual spike and

BREAK TEST

This tests will focus on breaking the system. The test will keep on adding a user continuously until the system stops. This test will identify the breaking points of the system.

SOAK TEST

This tests will focus on saturating system resources. The test will keep running a load test for a workday maybe multiple workdays then compare system resources before and after the test. This test usually uncovers memory leaks, or less optimized caching and threading.

OUT OF SCOPE:

FRONT END

The performance tests will not verify the user interface. They will not verify the API's functional side, but there will be limited functional testing as a smoke test to ensure the application is available for testing.

THIRD PARTY COMPONENTS

The test will not verify any third party components that may used as part of this system.

ASSUMPTIONS

- The system will be released to support 20 users. The expectation is that it supports up to 20 concurrent users. So the maximum supported load is 20 concurrent users. If this number changes we will need to test for the new number.

- A load test uses the maximum number of users (20).

- A stress test executes the tests using the 150%-200% of the number of users (30-40).

- All 20 users will be located within the company private network. If we release this product to users outside of our network, we should consider adding test machines in each of the user locations to test with network latency related to VPN, network speed or other Internet related delays.

- The tester will have access to the servers under test or operations will export the following metrics to be correlated to the test results. Metrics to be collected are CPU usage, memory usage, disk usage, thread count and network usage.

- Performance Test Environment matches production environment in terms of processing power and hardware specifications.

EXPECTATIONS

- System should maintain request success rate of 95% or higher for an 8 hour day. This refers to the metric(http_req_failed)

- All requests must be completed in less than 3 seconds.

    - If a request takes more than 3 seconds but less than 5 seconds, it is a minor defect with severity 3.

    - If a request takes 5 seconds or more but less than 10 seconds, it is considered a major defect with severity 2.

    - If a request takes 10 seconds or more, it is considered a major defect with severity 1.

- The system should continue to work as expected while 20 users are running the same request.

- The system should continue to work as expected while 20 users are running different requests.

RISKS AND MITIGATIONS

RESOURCE AVAILABILITY:

- **Support team's availability:** Critical issues require developers or operations to spend time investigating the issues and the possible solutions. To mitigate this risk, we need to free up an operation support resource and a developer to be available as soon as needed. **This will impact the cost and schedule.** We will include a developer and an operation resource on stand by.

- **Environment availability:** Test environment needs to be available for the duration of the test and the time for investigating the issues. **This will impact the schedule.** The operations and dev resources will be available to make sure the environment is available.

TESTING SOFTWARE:

The testing software (K6 and other underlaying tools like NodeJS, Visual Studio Code and others) may break during the test due to changes that require rework. **This will impact the cost and schedule.** We will disable automatic updates and will not run any software update during the test. K6 will be used with the latest stable version. No tool upgrade will be conducted for the duration of the test.

REQUIREMENTS:

- **Incomplete or incorrect requirements:** This may delay the creation of the test scenarios. **This will impact the cost and schedule.**

- **Change of the requirement during testing:** Sometimes the initial tests reveal better performance than expected and that can trigger re-evaluation of the requirements. Other times a test might reveal a weakness that will require negotiation with the customer to adjust expectations for a limited time until the issue is resolved or indefinitely. **This will impact the schedule.** We will evaluate if a schedule change is needed based on the requirement change impact. If it is acceptable, we will update the schedule, otherwise we will keep the original plan and plan for another test cycle with the updated requirements.

TEST SCENARIOS:

- **Late test scenarios approval: This will impact the schedule.** We will make sure to get this task done on time in order to avoid delays, but if that is not possible, we will have to get stakeholders approval on the new timelines.

QUALITY CRITERIA

The system may not have any severity 1 or 2 defects by the end of the test.

All the tests are expected to meet the success criteria for each.

TOOLS

K6

Visual Studio Code

PLANNED TESTS

We should spend some time to define the most common business scenarios for using these services. Based on my analysis I think the following scenarios are the most common ones.

SCENARIOS

BASIC LOOKUPS

➡ Look for all the users.

➡ Look for all albums.

➡ Look for all photos.

➡ Look for all posts.

➡ Look for all comments.

➡ Look for all todos

COMPLEX LOOKUPS

➡ Look for albums for a specific user.

➡ Look for photos for a specific album.

➡ Look for posts for a specific user.

➡ Look for comments for a specific post.

➡ Look for todos for a specific user.

NEW DATA ENTRY

- Add a new user.
- Add a post for that user.
- Add a comment for that post.
- Add an album for that user.
- Add a photo for that album.
- Add a todo for that user.

UPDATER

- Modify an existing user
- Modify an existing post
- Modify an existing comment
- Modify an existing album
- Modify an existing photo
- Modify an existing todo

DELETE

- Delete an existing user
- Delete an existing post
- Delete an existing comment
- Delete an existing album
- Delete an existing photo
- Delete an existing todo

Each of the tests (except the smoke test), will be run three times using the following three test scenarios:

SCENARIO1

This scenario is meant to check each call individually in order to verify the most basic interaction between the user and the system.

100% Basic lookups

SCENARIO2

This scenario is to verify that concurrent updates and lookups do not lock system resources.

90% Complex lookups

10% Updater

SCENARIO3

This scenario is to simulate real environment where some users are looking up data, some are updating data , some are deleting and some are adding new data.

70% Complex lookup

10% New Data Entry

10% Updater

10% Delete

SMOKE TEST

This test is to communicate with every one of the routers successfully and verify that the system is working and able to execute the rest of the tests.

TEST DESCRIPTION

Send a GET request to each of the Routers and make sure it is up and running.

1. GET /posts
2. GET /comments
3. GET /albums
4. GET /photos
5. GET /todos
6. GET /users

SUCCESS CRITERIA

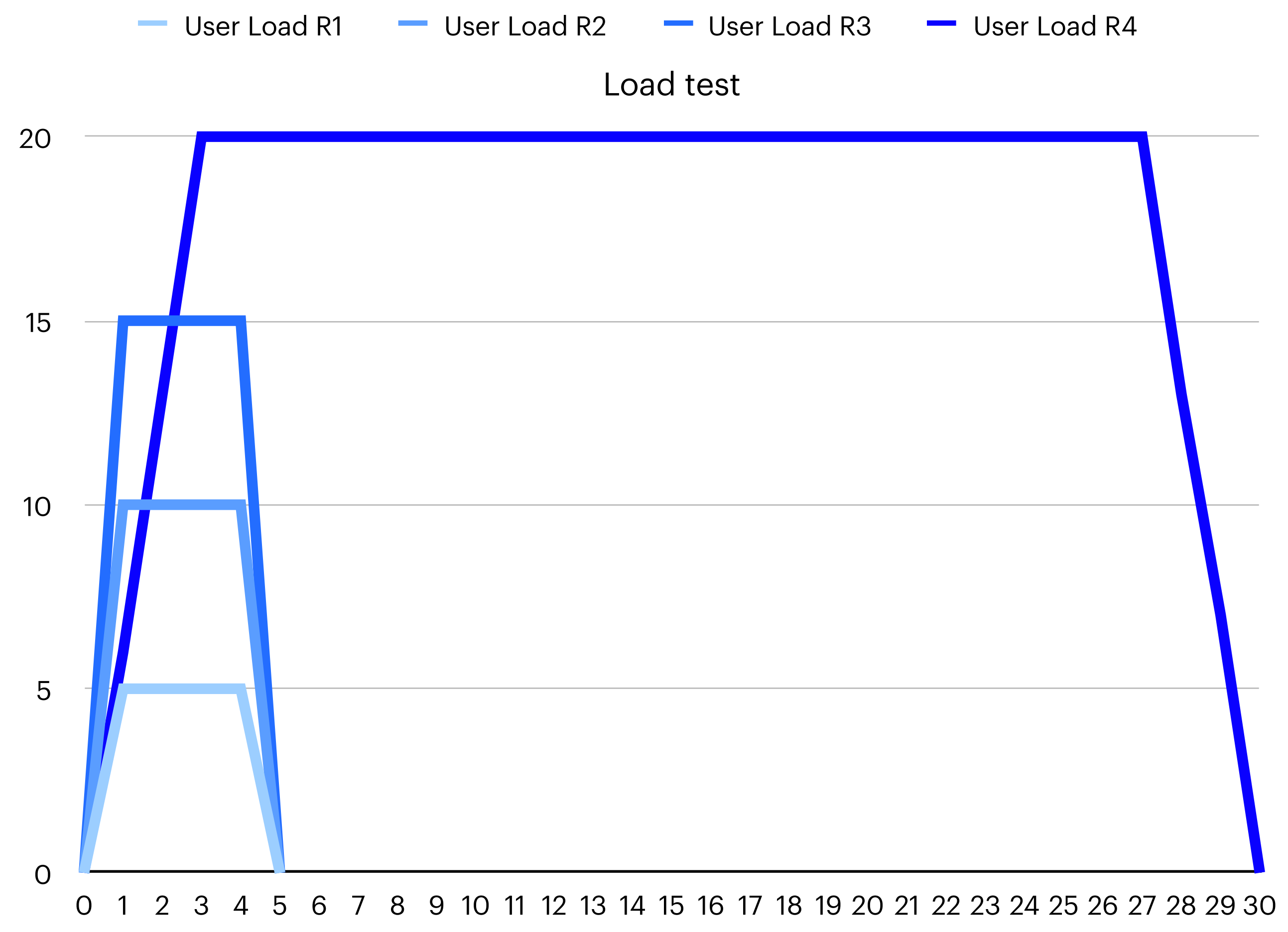
All validations run successfully with 200 response code.

LOAD TEST

This test is to verify that the system can handle the load of 80-100% of the users.

TEST DESCRIPTION

Record Server vitals(CPU, Memory, Disk and #of threads) before and after the test and verify the resources are released 15 minutes after the test. This test will run in 4 rounds. 3 Rounds will run while increasing the load by 25% each time. Each test runs for 5 minutes. This approach is designed to gradually increase the load to identify the breaking points (if any.) The final round runs with full load (20 users) for 40 minutes. If the system is stable under all of the rounds subsequent tests can run only round 4.



Each test round have three stages:

1. Stage 1: Ramp up. This stage takes 10% of the time to reach the target load.

2. Stage 2: load. This stage takes 80% of the time.

3. Stage 3: Ramp down. This stage takes 10% of the time to get to 0.

Test runs:

A. Round 1: test for 5 users for 5 minutes. Ramp up for 30 seconds. Run for 4 minutes then ramp down for 30 seconds.

B. Round 2: test for 10 users for 5 minutes Ramp up for 30 seconds. Run for 4 minutes then ramp down for 30 seconds.

C. Round 3: test for 15 users for 5 minutes Ramp up for 30 seconds. Run for 4 minutes then ramp down for 30 seconds.

D. Round 4: test for 20 users for 30 minutes Ramp up for 3 minutes. Run for 30 minutes then ramp down for 3 minutes.

SUCCESS CRITERIA

For each round, no thresholds are violated.

FailureRate_Threshold: http_req_failed < .05%

ResponseTime_Threshold: 95% of the iteration_duration < 3s

STRESS TEST

Record Server vitals(CPU, Memory, Disk and #of threads) before and after the test and verify the resources are release 15 minutes after the test. This test is to validate that the system can handle 200% of the load. This test can only be run after a successful Load test.

TEST DESCRIPTION

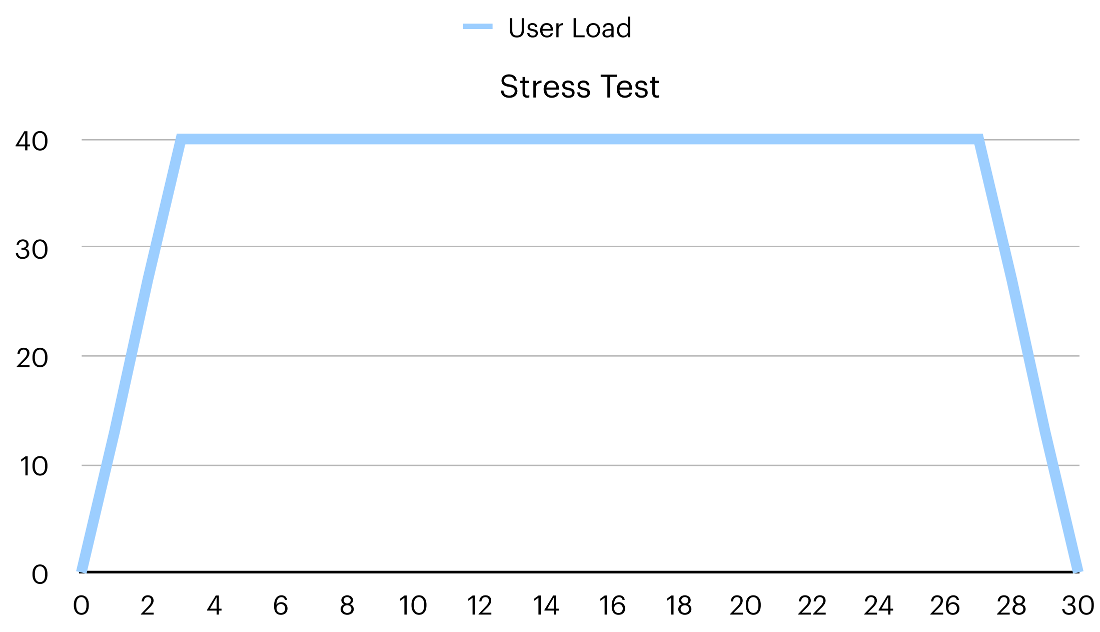
This test runs with double the full load (40 users) for 30 minutes.

The test have three stages:

1. Stage 1: Ramp up. This stage takes 10% of the time to reach the target load.
2. Stage 2: load. This stage takes 80% of the time.
3. Stage 3: Ramp down. This stage takes 10% of the time to get to 0.

Test runs:

A. Test for 40 users for 30 minutes Ramp up for 3 minutes. Run for 34 minutes then ramp down for 3 minutes.



SUCCESS CRITERIA

No thresholds are violated.

FailureRate_Threshold: http_req_failed < .05%

ResponseTime_Threshold: 95% of the iteration_duration < 3s

SPIKE TEST

Record Server vitals(CPU, Memory, Disk and #of threads) before and after the test and verify the resources are released 15 minutes after the test. This test is to ensure that the system can recover from a large sudden surge in the requests. This test is to validate that the system can handle 400% of the load for a short period. This test can only be run after a successful stress test.

TEST DESCRIPTION

In a 10 minute test, a spike runs for 2 minutes with 4x the full load (80 users) for 10 minutes.

This test has five stages:

1. Stage 1: Initial ramp up. This stage takes 10% of the time to reach 16 users (80% of 20).

2. Stage 2: Plateau. This stage takes 30% of the time. Users stay at 20.

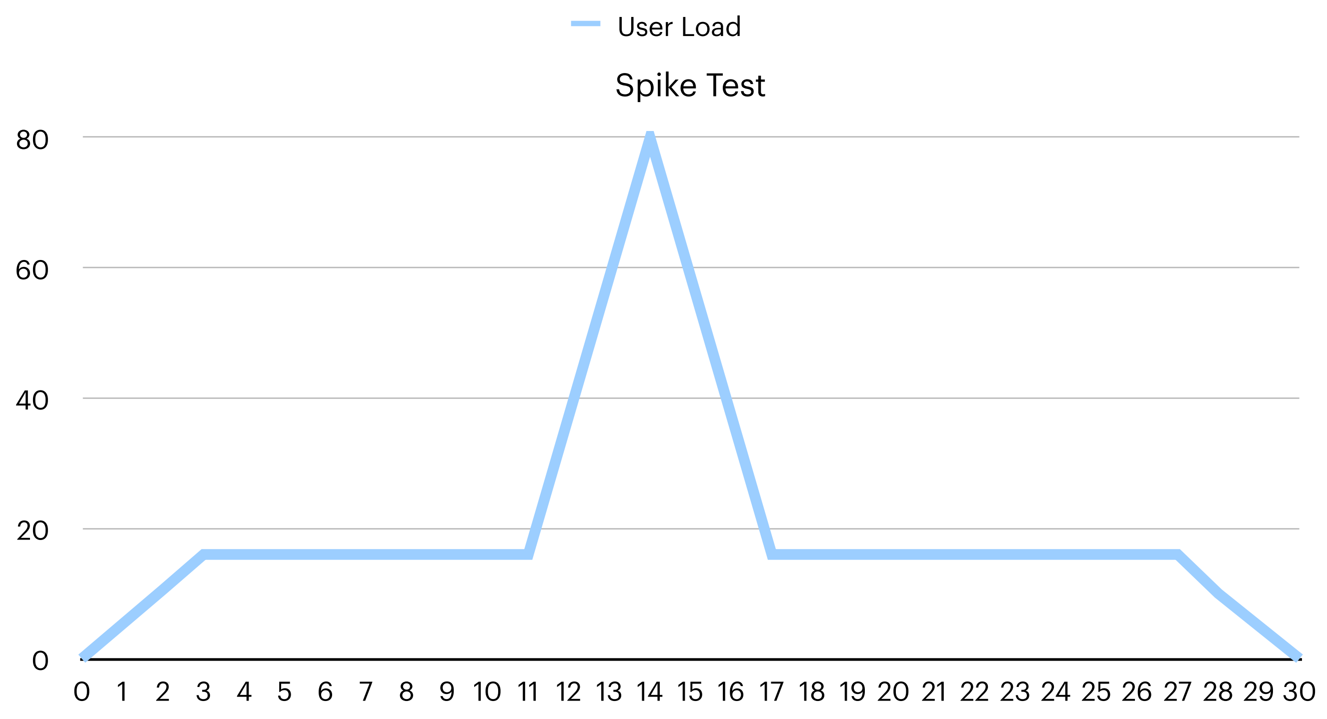
3. Stage 3: Spike up. This stage takes 10% of the time. The load is 80 users.

4. Stage 4: Spike down. This stage takes 10% of the time. The load is back to 20 users.

5. Stage 5: Ramp down. This stage take 10% of the time and it goes down to 0 users.

Test runs:

A. Test for 40 users for 30 minutes Ramp up for 3 minutes. Run for 34 minutes then ramp down for 3 minutes.



SUCCESS CRITERIA

No thresholds are violated.

FailureRate_Threshold: http_req_failed < .05%

ResponseTime_Threshold: 95% of the iteration_duration < 3s

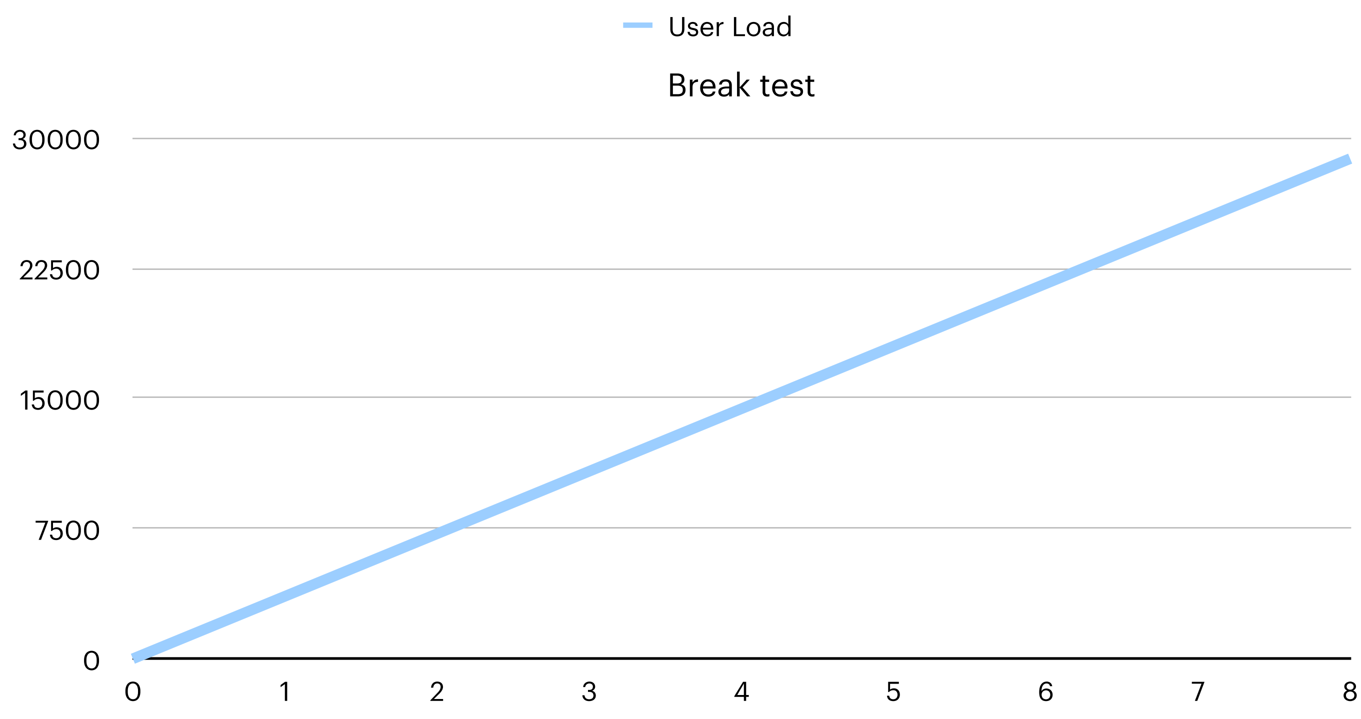
BREAK TEST

TEST DESCRIPTION

Record Server vitals(CPU, Memory, Disk and #of threads) before and after the test and verify the resources are released 15 minutes after the test. This test is to find the system's breaking point. In this test we increase the number of users by 1 every second and run the test for 8 hours or as long as it takes to break the system.

SUCCESS CRITERIA

If the system fails after reaching 200% of the planned number of users, it is considered a success.



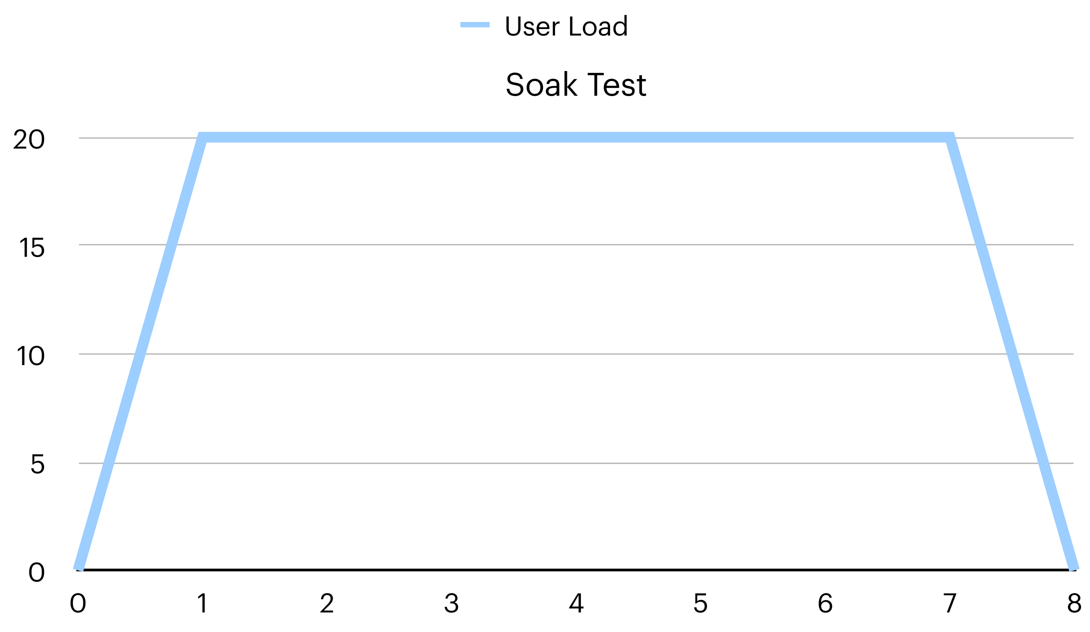
SOAK TEST

TEST DESCRIPTION

Record Server vitals(CPU, Memory, Disk and #of threads) before and after the test and verify the resources are released 15 minutes after the test. This test is to find the system's weak points. In this test we ramp up the users to the load of 80-100% of the expected load then run the test for 8 hours. Then check the system's resources 15 minutes after the test is done. Memory leaks show up as memory that is not released after the test is done. High CPU and Disk IO are indicators of rogue processes or threads.

SUCCESS CRITERIA

After the test is done, if the system goes back to the original state of the vitals, it is considered a success.



REFERENCES:

- **CPU usage:** Monitoring CPU usage includes insights into the server's processing capacity and helps identify bottlenecks. For example, high CPU usage can lead to slow response times, increased latency, and reduced overall performance.

- **Memory usage:** Monitoring memory usage allows you to understand how effectively the server is utilizing its available memory. It helps identify potential memory leaks or inadequate memory allocation. Higher memory usage may cause the server to rely on disk swapping, which significantly impacts performance.

- **Disk I/O:** Tracking disk I/O metrics, including read and write speeds, helps evaluate the performance of the server's storage subsystem. High disk I/O wait times can impact application responsiveness.

- **Network latency:** The latency measures the delay between a request sent from the server to another system or vice versa. It helps identify network-related performance issues and optimize network configurations. High network latency may slow data transfers, increase response times, and result in poor user experience.

- **Response time:** Monitoring response time measures the time it takes for the server to respond to requests. It is a critical metric for assessing the overall performance of the server. High response times can indicate performance bottlenecks, such as inefficient code, database issues, or resource constraints.