LocalEGA/SDA stress testing

Release 1.0.0

Nov 05, 2019

Table of Contents

CHAPTER 1

Inbox Testing Specs

1.1 Testing environments

Testing environments below should test with both types of backend storage S3 and POSIX.

1. Isolated Inbox - A scenario where we focus exclusively on the inbox
   - For the test we mock the Inbox dependencies such as MQ and CEGA-users
   - Makes more sense in a scenario for Load Testing
2. Inbox as part of the Ingestion Stack:
   - We have all the ingestion components running.

1.2 Maximum expected load

Note: this would be the success criteria, by which we make sure that under such a load the system recovers easily.

In order to determine the expected user load, or more precisely the system limits we are going to perform an exploratory performance tests in this stage.

> e.g. x concurrent users uploading files without the services crashing or becoming unusable. e.g. 1000/10000 users that upload (or perform an operation with inbox) simultaneously. (we should also establish some file sizes e.g. 100Mb, 1Gb or 10Gb for the expected load)

1.3 Metrics

1. Average operation response time - successful/erroneous operations
2. Total number of transactions per second - total successful, total erroneous operations in a second

3. Response time under heavy load - do we experience any throttling under heavy load

1.4 Testing Scenarios

The scenarios will consistent of two stages: Exploratory Stage - where we try to identify the limits of the system, based on different setups - meaning try to first perform the test in local environment then move to a deployed environment, but without any scaling enabled. With this we will be able to figure out the limits Perform the tests but with different scaling strategies - vertical or horizontal .

1.4.1 Hypothesis

We should be able to do multiple types of operations, at the same time with multiple users. In the exploratory phase we will establish the limits of the system and to what degree of expected load we can stress the system, and still maintain functionality. For this purpose we must considers scenarios that overload the system such as:

1. Load testing - increase user load over a period of time
   - 10 users try to submit a file
   - 100 users try to submit a file
   - 1000 users try to submit a file
2. Test 100 users trying to upload 1Gb file at the same time
3. Test different operations in the inbox with e.g. 100 users
   - upload (might already be covered by the scenarios above)
   - remove (maybe try to delete the same file)
   - rename

1.4.2 Scenarios Template

1. User enters inbox
   - User selects an encrypted file
2. User does operation with a file or multiple files (upload, rename, remove)
3. Finishes operation, e.g. exit inbox
   - Enters the inbox again and repeats step 2 and 3.

Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Connect	Connect	Connect	Connect	Connect
Upload	Upload	Upload	Upload	Upload
Disconnect	Disconnect	Rename	Rename	Disconnect
	Reconnect	Disconnect	Remove	Reconnect
	Rename		Disconnect	Rename
	Disconnect			Disconnect
				Reconnect
				Remove
				Disconnect

1.5 Tools

1. Build our own: Custom build Locust http://locust.io/ files for each scenario. It seems we are able to use locust for this purpose from initial tests. As this will be used in Data API performance testing, make sense to use the same tool. To Be Establish what data we can extract, based on this custom tool. We will have to add functionality so that we can measure the speed. 2. Use JMeter - seems the most appropriate one

1.6 Further Reading

- http://tryqa.com/what-is-stress-testing-in-software/
- https://www.blazemeter.com/blog/load-testing-ftp-and-sftp-servers-using-jmeter

CHAPTER 2

Data API Testing Specs

2.1 Testing environments

Testing environments below should test with both types of backend storage S3 and POSIX. A JWT Token along with its corresponding Public Key will need to be set up for the testing environment,

1. DataEdge as part of the LocalEGA + Data API Stack:

- We have all the ingestion and outgestion components running.

2. DataEdge as part the Data API stack:

- in case we just want to test the performance with no hassle of setting up the full stack

Endpoints are described at: https://github.com/EGA-archive/ega-data-api/tree/master/mock-services/openapi/dataedge and we will be focusing on the /files endpoint with both encrypted and decrypted format.

2.2 Maximum expected load

Note: this would be the success criteria, by which we make sure that under such a load the system recovers easily.

In order to determine the expected user load, or more precisely the system limits we are going to perform an exploratory performance tests at this stage.

> e.g. x concurrent users downloading files without the service crashing or becoming unusable. e.g. 1000/10000 users that download simultaneously. (we should also establish some file sizes e.g. 100Mb, 1Gb or 10Gb for the expected load)

2.3 Metrics

1. Average operation response time - successful/erroneous operations

2. Total number of transactions per second - total successful, total erroneous operations in a second
3. Response time under heavy load - do we experience any throttling under heavy load

2.4 Testing Scenarios

The scenarios will consistent of two stages: Exploratory Stage - where we try to identify the limits of the system, based on different setups - meaning try to first perform the test in local environment then move to a deployed environment, but without any scaling enabled. With this we will be able to figure out the limits Perform the tests but with different scaling strategies - vertical or horizontal.

2.4.1 Hypothesis

We should be able to do multiple types of operations, at the same time with multiple users. For this purpose we must considers scenarios that overload the system such as:

1. Load testing - increase user load over a period of time
   - 10 users try to download a file
   - 100 users try to download a file
   - 1000 users try to download a file
2. Test 100 users trying to download 1Gb file at the same time
3. Test different scenarios with the DataEdge API endpoint with e.g. 100 users
   - download a file that exists
   - download a file that does not exist
   - try different parameters of the DataEdge API endpoints

2.4.2 Scenarios Template

1. User tries the endpoints
   - `/files/{file_id}?destinationFormat=plain`
   - `/files/{file_id}?destinationFormat={aes,crypt4gh}&destinationKey={key}&destinationI`
   - User provides appropriate headers with JWT token
2. User waits for download to finish.

Scenario 1	Scenario 2
Token with correct permissions	token with no data access
Download Decrypted file	Download a file user does not have access
Response should be 200 in given time interval	Should return the 403 error

Scenario 3	Scenario 4
Provide token with correct permissions	Multiple tokens with correct permissions
Download multiple decrypted files	Download multiple decrypted files
Response is 200 in time interval	Response is 200 in time interval

Note: For scenario 3 and scenario 4 response should be `200` for all files.

Scenario 5
Token with correct permissions
Download decrypted big file(s)
Response should be 200 in given time interval and the connection should be kept alive

2.5 Tools

Custom build Locust http://locust.io/ files for each scenario, or we might be able to group them together. We will have to add functionality so that we can measure the speed.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search