Polytechnique Montréal

LOG8371E

Software Quality Engineering

Assignment 2 - Performance efficiency assurance for Weka

Students :
Yuri Azar - 1780762
Jérôme Désilets - 1684912

Teaching Assistant:
Mahsa Hadian

Saturday November 9th 2019

Contents

1 Introduction

After planning for the Functionality, Maintainability and Performance Criteria in a previously handed-in report, it is required to further the quality assurance for Weka, which is done by more analysis as well as implementation of deployment and scaling.

More specifically, We are now to add into the already existing plan the quality assurance planning required to validate that the Performance efficiency criteria is correct for this software. Also, we will detail our implementation of a REST version of Weka using Docker, profile its performance, run load tests, and configure monitoring on the docker container.

According to the previously mentioned tasks, this current report is split in 8 sections. First, we are making notes on accessing previously stated details about the report. Secondly, the additions to the quality plan that need to be made to account for the performance efficiency criteria will be explained. Then, the required tests will be described and planified. After the new criteria's addition is dealt with, we will delve deeper into this report's more technical aspects. The fourth section will describe a deployment of a REST version of weka and its usage manual. The fifth section will describe our profiling of the deployed Weka version and the reports that emerge from it. The sixth section will concern load tests and how to execute them. The seventh section will be about our configuration for the monitoring for Docker and its scaling according to the load. The final section is simply a link to our demonstration on how we deployed and scaled our Docker containers

2 Introductory Notes

Since a complete report is already available surrounding many of the details of our Quality Plan, it is important to note that all the previously mentioned information is still valid.

Therefore, you can refer to our first report for more information on these following aspects:

- The team, including its members and their assigned responsibilities
- The selected features (Naive Bayes, Multinomial naive Bayes, Expectation-Maximization, Farthest-first, Cobweb, Apriori)
- The Functionality, Reliability and Maintainability criteria and their sub-criteria.
- Discussion on our criteria's validity, the tests themselves, the testing schedule and the test reports for our quality criteria.

For everything else related to this report's goal, it will be contained in the following pages.

3 Quality Plan Update for the Performance efficiency Criteria

To be able to ensure that Weka's deployment with its REST version is satisfyingly performant, we have to validate that, as well as the criteria mentioned in the previous report, the Performance efficiency criterion is correctly covered. But before we are able to validate, it is needed to define the criteria, its sub-criteria, their objectives, and the validation metrics and methods.[1]

The performance efficiency criteria is related to the amount of resources used under stated conditions.

ESTCE QU'ON EN EVALUE 2 OU 3?? It contains 3 sub-criterias, which we will evaluated. There is Time behavior, which is the degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements. Also, there is Resource utilization, which is the degree

to which the amounts and types of resources used by the system meet requirements. There is also capacity, which is the degree to which the maximum limits of a product or system parameter meet requirements.[1]

The following table details each sub-criteria's validation objective, metrics and methods.

Quality Objectives for Performance efficiency Criteria			
Sub-Criteria	**Objectives**	**Validation Metrics**	**Validation Methods**
Time behavior	Average response time for a request is within 0.5 seconds for an average load test	Response time	Monitoring, profiling, load tests
Resource utilization	Resources (CPU, RAM) reach a peak usage of 40% during an average load test	Peak percentage of resource utilization	Load tests, profiling, monitoring
Capacity	Software can handle 200 requests a minute without crashing	Maximum number of requests handled by the system	Load tests, capacity testing

4 Tests

In this section, we will add to the pre-existing plan the tests that need to be made in order to validate the performance efficiency criteria. Before that, we will establish a preliminary analysis of the sub-criteria's validity and then establish which members of our team will be responsible to realize the required tests and analysis.

4.1 Preliminary Sub-Criteria Validity

Since our newly established sub-criteria are all related to performance efficiency, it is practically impossible to judge of their validity at a first glance, without monitoring, profiling and testing. The tools that will be used in future tests are necessary to

establish the behavior.

We can only establish that the quality objectives are basically met since the software proves usable when a user uses the program. Which means that the time behavior, the resource utilization, and the capacity are satisfying for a single user.

4.2 Tests description

Sub-Criteria	**Scope**	**Test process**	**Action in case of failure**
Time behavior	Software	Orchestrate load tests, measure their average response time while monitoring and profiling	Find what the cause is, optimize or increase scale
Resource utilization	Software	Orchestrate load tests while profiling, monitoring and measuring the resource utilization	Find what the cause is, optimize or increase scale
Capacity	Software	Orchestrate capacity tests while profiling and monitoring the resource utilization	Find what the cause is, optimize or increase scale

4.3 Testing schedule

To test the new criteria, we are planning a new sprint of a week. Since the previous tests were scheduled over a week, we can suppose that this new week is the sprint that follows the tests made for the Functionality, Reliability, and Maintainability criteria.

As for the previously planned schedule, we are also ready plan for making the system more compliant to our quality criteria in the next sprint (if it is needed), and the sprint after that if the tasks proves too big.

Monday, November 11th (Day 1):

Morning:

- Jérôme and Bobby will set up the monitoring and profiling for Weka
- Dandelion and Percival will validate and correct the sub-criteria objectives according to their knowledge of Weka's needs

Afternoon:

- Jérôme and Bobby will configure the system to make it ready for capacity and load tests.
- Dandelion and Percival will design the load tests profiles

Tuesday, November 12th (Day 2):

Morning:

- Jérôme and Bobby will orchestrate the load tests profiles
- Dandelion and Percival will analyze the monitoring and the profiling during the capacity tests and determine whether or not the capacity criteria is valid

Afternoon:

- Jérôme and Bobby will keep orchestrating the load tests according to the load profiles
- Dandelion and Percival will analyze the monitoring and the profiling and determine whether or not the time behavior and resource utilization criteria are valid

Wednesday, November 13th (Day 3)

Morning:

- Jérôme and Bobby will orchestrate the capacity test
- Dandelion and Percival will analyze the monitoring and the profiling during the capacity tests and determine whether or not the capacity criteria is valid

Morning:

- Jérôme and Bobby will keep orchestrating the capacity test
- Dandelion and Percival will analyze the monitoring and the profiling during the capacity tests and determine whether or not the capacity criteria is valid

Thursday, October 14th to to Friday, October 15th (Day 5): Any fault encountered in the previous days' tests are to be fixed by Jérôme and Bobby so as to increase the software's performance efficiency metrics to their apex.

4.4 Test reports

Due to the scope of this document, results for capacity tests will not be provided, and so will have to be provided in the future.

However, preliminary load tests will be made and their implementation and output will be provided further in the document.

5 REST Deployment

5.1 Prerequisites

In order for the REST deployment of weka to work, you should have the following softwares already installed on your machine :

- Docker Desktop [2]
- MongoDB
- Maven
- A REST version of WEKA [3]

You will also need your Docker Hub ID that you used to download Docker Desktop.

5.2 Deployment steps

1. Open a terminal window and navigate to the jguwekarest directory using the **cd** command

2. Compile the rest version of weka using the **mvn clean package** command

3. Create a Docker image of the rest version of weka using the **docker build -t {yourDockerHubID}/restweka .** command
   You might get an error on this step that says you don't have the a certificate installed on your machine. In order to fix that error, open the Dockerfile in the directory and delete the lines that start with RUN openssl and RUN keytool (lines 16-17)

4. Pull the mongo image using the command **docker pull mongo**

5. Create a local docker container that will run mongodb using the command **docker run --name mongodb -d mongo**

6. In order to create a local docker container that will run the REST version of weka, we will need to link it to the mongodb container that runs on port 8080. That can be done by using the command **docker run -p 8080:8080 --link mongodb:mongodb {yourDockerHubID}/restweka**
   Note that the last argument of the previous command should match exactly with the docker image name created at step 3

7. The rest version of weka should now be deployed ! In order to confirm the deployment, open a web browser and go to **localhost:8080**. You should be greated with the weka web UI.

6 Profiling

6.1 Prerequisites

In order to profile the rest weka app using JProfiler, you should have followed the steps to deploy the rest weka app on a docker container like explained in the section above. You should also have the JProfiler software installed on your machine. [4]

6.2 Profiling steps

1. In the jguwekarest directory, open the Dockerfile and add these two lines at the end of the file

   (a) RUN wget
   https://download-gcdn.ej-technologies.com/jprofiler/jprofiler_linux_11_0_2.tar.gz

   (b) EXPOSE 8849

2. Create a new Docker image of the rest weka using the command **docker build -t {yourDockerHubID}/restweka .**

3. Recreate the container that will run the rest version of weka by exposing the port 8849 using the command **docker run --name weka -p 8080:8080 -p 8849:8849 --link mongodb:mongodb {yourDockerHubID}/restweka**

4. Go inside the container using the command **docker exec -ti weka bash**
   The first argument after -ti in the previous command should be the same name you used when you ran the rest weka container

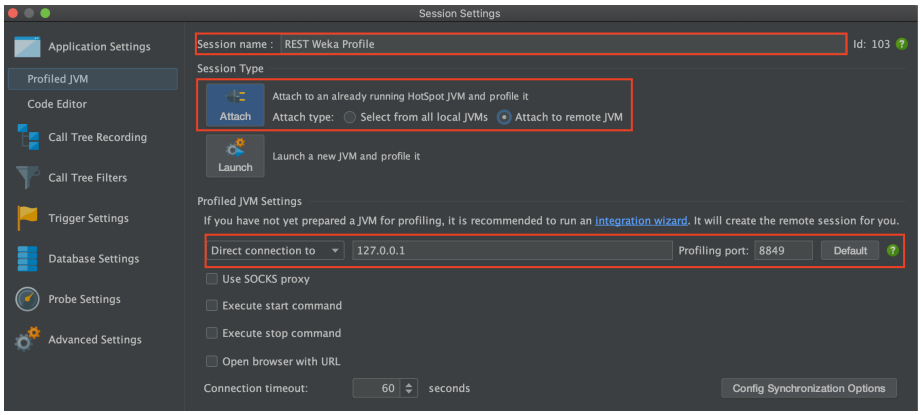
5. Once inside the container, you should be able to find the file jprofiler_linux_11_0_2.tar.gz. Use the following command to extract the contents of the tarball : **tar -xzf jprofiler_linux_11_0_2.tar.gz**

6. Once the extracting is done, use the following commands to attach JProfiler to the container :
   cd jprofiler11.0.2/
   ./bin/jpenable

7. The profiler will then require you to entrer two numbers. Just enter 1 for the profiling mode followed by 8849 for the profiling port just like in the following example.

8. On your machine, open the JProfiler GUI and create a new session.

9. Enter a session name, choose **Attach to remote JVM**, enter **127.0.0.1** next to direct connection to and choose the port **8849**, just like the following example.



10. You can click OK, follow the recommanded settings and you're ready to use JProfiler with weka.

6.3 Load tests scenarios

We have designed load tests so that we can validate both resource utilization and time behavior.

Considering that the software we are profiling is a small one and is mostly used by academics, we have designed the load scenarios, even the exceptionally increased load, to not represent huge exchanges of data. At most, 16 users will be doing 20 executions of 20 requests each over 20 seconds, which is not much compared to well known website's load-bearing abilities.

6.3.1 Scenario 1: Reduced load

Number of users: 1
Total number of requests: 1
Data: 587 Bytes file (weather.nominal_587B.arff)

6.3.2 Scenario 2: Average load

Number of users: 4
Total number of requests: 4
Data: 5500 Bytes file (cpu_5500B.arff)

6.3.3 Scenario 3: Increased load

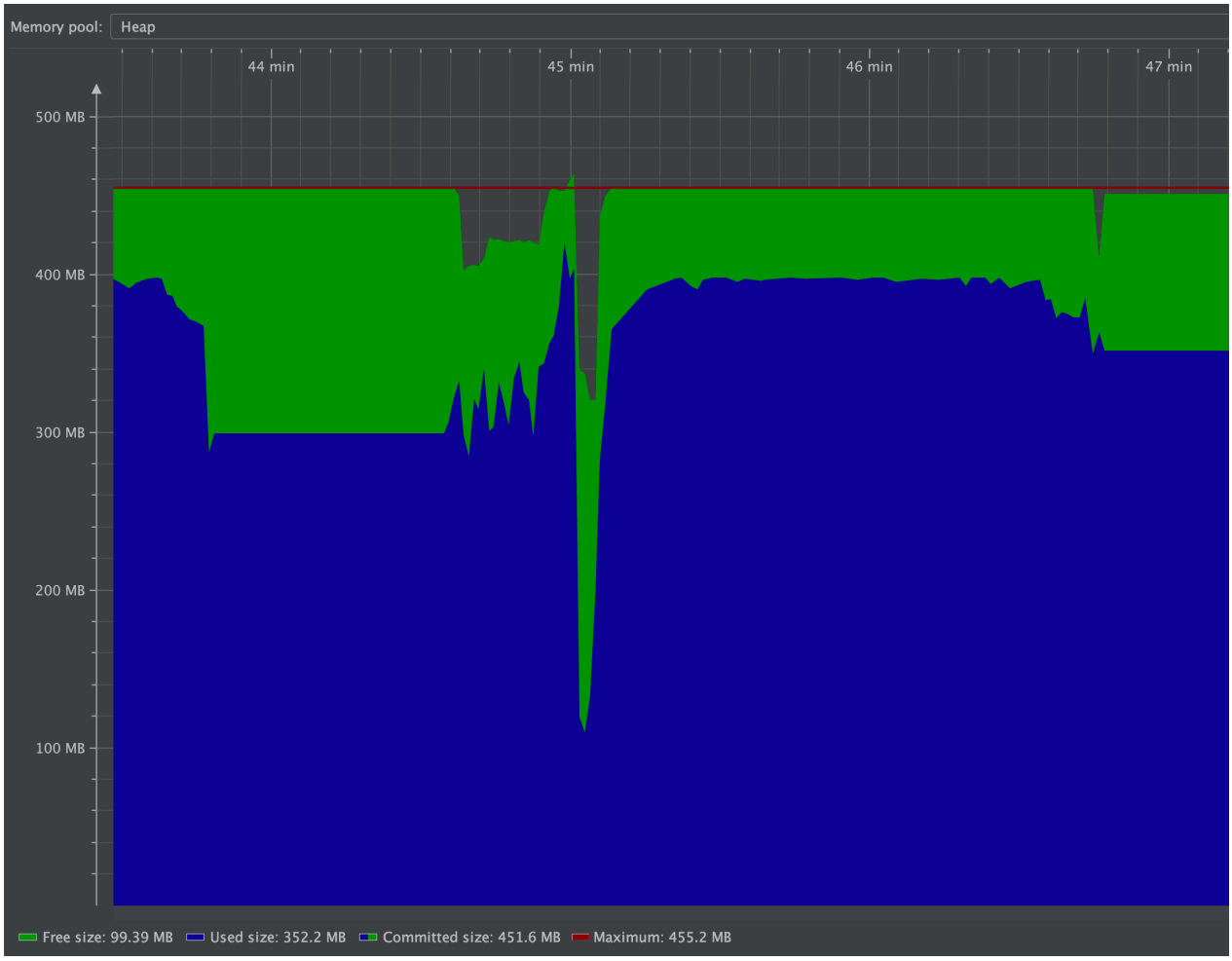
Number of users: 8
Total number of requests: 8
Data: 202 Kilobytes file (soybean_202kb.arff)

6.3.4 Scenario 4: Exceptionally increased load

Number of users: 16
Total number of requests: 16
Data: 2 Megabytes file (supermarket_2mb.arff)

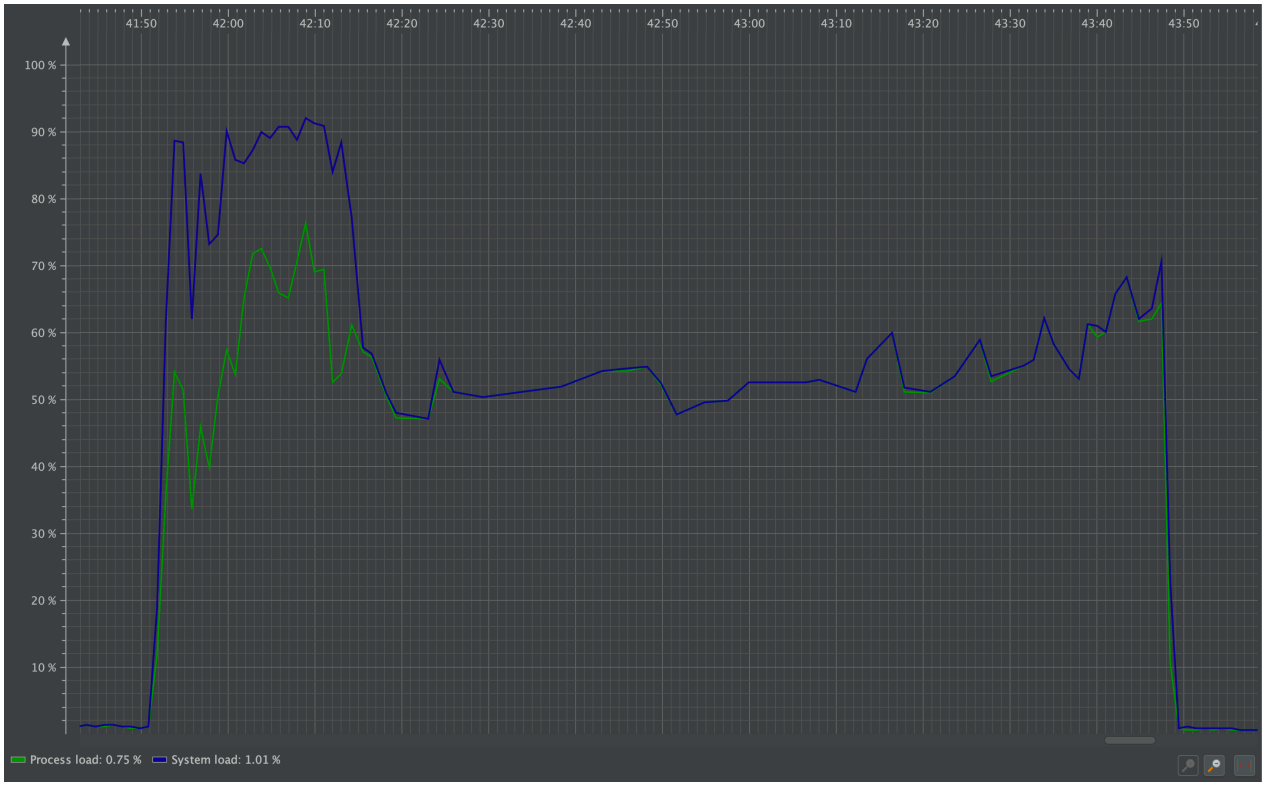
6.4 Consumption results according to load tests scenarios

After running the tests according to these 4 scenarios, we can now analyse the resource usage according to our two chosen resources: memory and CPU. The following figure concerns memory usage over the realization of the test.



The red line indicates the memory which the system has, and the blue colored surface indicates the memory usage. For most of the load tests, the memory usage is much higher than 40%, which is the maximum we have set for our resource utilization criteria. In fact, the average seems to be of about 75% of memory utilization. Since every scenario is executed consecutively, scenario 2 by itself is also higher than 40% memory usage, which is unacceptable.

We have also measured the CPU usage of the container for the load test, which is illustrated in the following figure.



Again, the lines, which are illustrating the cpu's usage percentage, is always over 50% when the process is running, no matter what the scenario is.

According to both the CPU and RAM usage results, changes have to be made by the team in Weka's implementation or deployment. One of the changes that will be made will be demonstrated in section 8 as well as explained in a video which is shown at the end of the report.

7 Load tests

7.1 Load tests steps

7.1.1 Running Jmeter

1. Download your prefered binary from: http://jmeter.apache.org/download_jmeter.cgi
2. Extract the binary

3. In /apache/jmeter-5.2/bin, run "jmeter.sh" on Linux, or "jmeter.bat" if you are using Windows. This should open the JMeter software.

7.1.2 Executing pre-configured Jmeter load tests

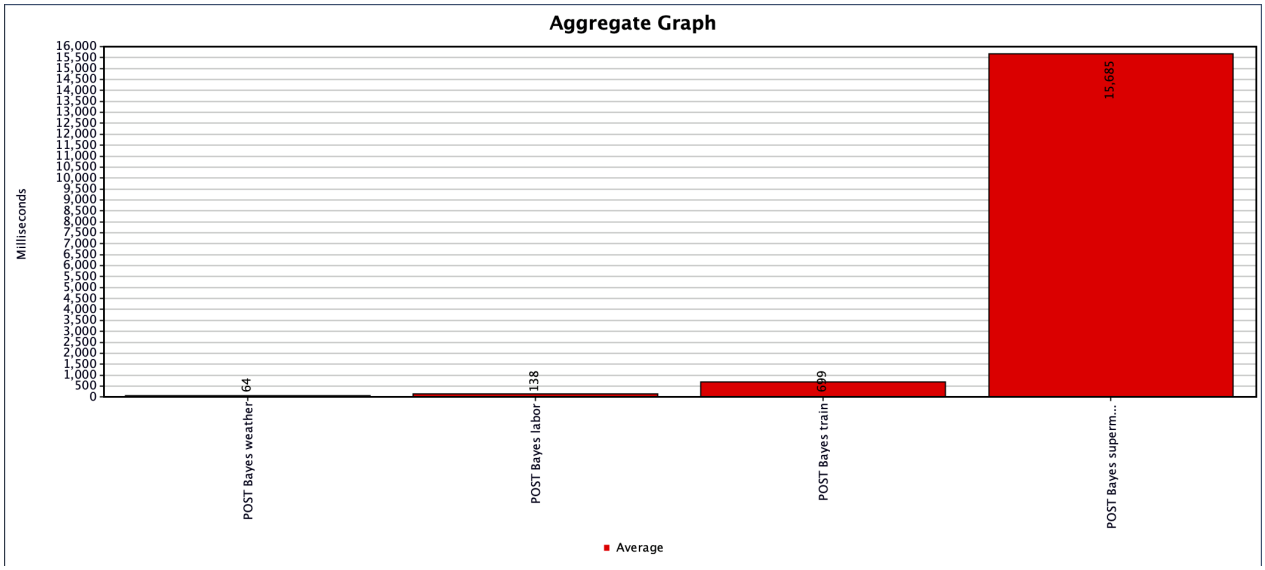
1. In your Jmeter window, click File -> Open -> Select WekaLoadTestProfiles.jmx
2. Press the green arrow named "Start"
3. Wait for the results to appear

7.1.3 Configuring your own scenario

1. In your JMeter window, select a thread group
2. To change the number of users, in a thread group's window, change the input field next to the number of threads according to the number of users you want
3. In the same window, feel free to adjust any of the parameters to your taste, namely the loop count for easier analysis
4. To change the used algorithm, select an HTTP Request sampler under a thread group that is configured to your taste, then adjust the path to fit the algorithm that you want to test.
5. To change the data set used for testing, in the same window, select the Files Upload tab, press the browse button and choose your dataset file.

7.2 Load tests results

The following image describes the average response time depending on each previously mentioned scenario, scenario 1 being to the left and scenario 4 being to the right.



For clearness purposes, the results will be mentioned here. Scenario 1 has an average of 64ms of response time. Scenario 2 has an average of 138ms of response time. Scenario 3 has an average of 699 ms of response time. Scenario 4 has an average of 15,685 ms of response time.

Since scenario 2 is for an average load and its result is an average of under 500ms of response time, the time behavior scenario is definitely validated. The Exceptionally increased load scenario, which is scenario 4, has an average response of 15,685. It is overly high, but since the software can still process the files without crashing, it is still bearable. Considering the previous results, no changes have to be made to Weka's implementation and deployment to fit its time behavior objective. However, that is according to our arbitrarily defined goals, so a more methodical approach could lead to a different result. Also, the response time seems to grow exponentially with the file's size, so if it happens that bigger and bigger datasets become the norm, the system's time behavior will have to be improved.

8 Monitoring

As it has been previously mentioned, our 4 scenarios' load tests lead to an invalidation of the resource utilization sub-criteria. Therefore, we have to scale our docker deployment so that more containers can take on the requests that are made during the load tests. In order to balance the load between different containers, we use the following command

docker-compose up --scale **service** = **number of containers needed**

We can use the JPorfiler GUI to monitor the CPU consumption, as well as the RAM usage, with a refresh rate of 2s. Therefore, we can know in real-time how many containers we need to add if the usage is high, or remove if the usage is too low. At this early stage of the project, the scaling process can be done manually, but eventually, once the load will be way higher, we would install our docker containers on a server, like Nginx, that would automatically scale the deployment based on the needs.

9 Deployment and Scaling demonstration of Weka

You can watch the demonstration at this address:

`https://youtu.be/VEmov-L3LSo`

References

[1] ISO 2500(n.d.), Retrieved November 09 2019, from
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[2] Docker Desktop can be found for free after creating an account on
https://hub.docker.com

[3] You can get a REST version of weka by cloning the repository on
https://github.com/jguwekarest/jguwekarest

[4] You can download a 10 day trial of JProfiler on
https://www.ej-technologies.com/products/jprofiler/overview.html