PtU Certified Performance Tester with JMeter (CPTJM)
Syllabus

Released
Version 1.0 2019

Performance Testing United

Copyright

Thank you to the main collaborators

Delvis Echeverria, Guillermo Skrilec, Rahul Verma.

Thank you to the review committee

Alexis Herrera, Alfonso Fernández, Almidena Vivanco, Ángel Rayo Acevedo, Antonio Jimenez, Arjan Brands, Arturo Eduardo Salguero Guiterrez, Boris Wrubel, Bruno Mareque Acosta, Daniel Garcia Castillo, Daniel Tolosa, Eduardo Gonzalez, Eduardo Ricardo Delgado Cortés, Emmanuel Espinoza Sales, Erik van Veenendaal, Guino Henostroza, Gustavo Marquez Sosa, Gustavo Terrera, Héctor Revalcaba, Jefferson Pereira Ortiz, Juan Pablo Rios Alvarez, Julie Gardiner, Leandro Melendez, Manuel Fischer, Marcela Mellado, Márcia Araújo Coelho, Marelis V. Pérez García, Michael Frontner, Miguel Angel De León Trejo, Nadia Soledad Cavalleri, Rafael Márquez Galicia, Richard Seidl, Samuel Ouko, Silvia Nane, Thomas Cagley, Vanessa Lslas Padilla & Wim Decoutere.

Revision history

Version	Date	Observations
PtU 2019	September 2019	First official release

Table of Contents

Purpose of this Document

This program provides the basis for Performance Testing United's Performance Tester with JMeter (CPTJM) certification. This document defines what you need to know to pass the Performance Tester with JMeter (CPTJM) certification exam. The certification exam will only cover the concepts and knowledge described in this document.

PtU Resources

An overview of PtU resources, as well as all the information relevant to the PtU certification and other types of PtU certifications are available on www.pt-united.com — the official website of Performance Testing United. The information available on www.pt-united.com includes:

- A complete list of recognized training providers and available courses. Please note that training is recommended, but not required to take the CPTJM PtU certification exam.
- The downloadable syllabus (this document).
- A complete sample set of 40 PtU CPTJM questions, with answers, for training purposes.

Our goal is to make the documents available in other languages as soon as possible. To see the language versions currently available, please visit www.pt-united.com.

What is Performance Testing?

Performance testing is a specific type of testing that determines a system's responsiveness to a particular workload in a particular context. It also allows to verify and validate software quality attributes such as scalability, reliability, availability, and resource usage, among other non-functional aspects.

There are different tools that can be used to execute performance testing. JMeter is one of the most widely used tools worldwide; it is free and open source.

About PtU Certified Performance Tester with JMeter (CPTJM)

PtU CPTJM is a practical course for testers who are already involved in performance testing, or for those who wish to start in that area. The course is focused on the use of the JMeter tool to carry out performance testing on web applications, thus bringing the use of the tool closer to real implementations through concepts such as recording the tests, building scripts that make it

possible to simulate the real use of the system, executing the tests, and monitoring the application during the tests, among other concepts. PtU CPTJM provides a solid technical base suitable for software testers, test engineers, performance analysts, test leaders, quality managers and operation support.

Business Objectives

(BOs for "Business Objectives"):

BO1	Understand the main concepts of performance testing and the methodology for conducting it.
BO2	Use the JMeter tool to create and run performance testing on web applications.
BO3	Analyze performance testing results and identify application performance improvements.
BO4	Identify application performance indicators and use tools for monitoring.

Learning Objectives / Cognitive Levels of Knowledge

Learning objectives (LOs) are short statements that describe what you are expected to know after studying each chapter. Learning objectives are defined according to the following levels:

- K1: Remember
- K2: Understand
- K3: Apply

The following table lists the main LOs for the PtU-CPTJM certification (LOs for "Learning Objectives"):

LO1	Understand what performance testing is and the different types of performance testing. (K2)
LO2	Understand the methodology for performance testing. (K2)
LO3	Identify the different types of tools used for performance testing. (K1)
LO4	Understand the basics of the HTTP(S) protocol. (K2)

LO5	Understand why JMeter is used for performance testing. (K2)
LO6	Install and run JMeter on Windows and Linux. (K3)
LO7	Understand and apply the main elements to build a test plan in JMeter. (K2)
LO8	Create basic scripts by recording a session and running it in JMeter. (K3)
LO9	Analyze the results of the tests through different reports. (K3)
LO10	Understand the concept of correlation and how to use regular expressions. (K2)
LO11	Apply the use of regular expressions to manage correlation in JMeter. (K3)
LO12	Understand the concept of parameterization. (K2)
LO13	Build and configure data sources for use in the test script. (K3)
LO14	Understand and apply timers, assertions and controllers in JMeter. (K3)
LO15	Debugging a script in JMeter. (K3)
LO16	Preparation of the scripts for the execution of the tests. (K3)
LO17	Run tests using the command line mode. (K3)
LO18	Execution of the tests through the distributed mode. (K1)
LO19	Understand how system resources are monitored during performance testing and their main indicators. (K2)
LO20	Apply basic monitoring tools during the execution of performance testing. (K3)
LO21	Performance testing documentation. (K2)
LO22	Understand best practices when using JMeter. (K2)

LO23	Execution of scripts for web services. (K3)

Prerequisites

- Basic knowledge of programming. Knowledge of the concepts of variable, function and control structures (conditionals and loops).
- Basic knowledge of web applications and their architecture. Knowledge of client-server architecture.
- Basic knowledge of the HTTP(S) protocol. Knowledge of the concepts of request, response, cookies, URL parameters, invocation methods (GET/POST), message headers and message body.

Chapter 1 - Introduction

LO1	Understand what performance testing is and their different types. (K2)
LO2	Understand the methodology for performance testing. (K2)
LO3	Identify the different types of tools used for performance testing. (K1)
LO4	Understand the basics of the HTTP(S) protocol. (K2)
LO5	Understand why JMeter is used for performance testing (K2)
LO6	Install and run JMeter on Windows and Linux. (K3)

1.1 Main Concepts of Performance Testing

1.1.1 Introduction to Performance Testing

Performance testing offers a means to study the performance of a system when faced with load and stress scenarios, similar to those that may occur in production.

These tests make it possible to know, among other things, the number of simultaneous users that the system supports, to obtain data to dimension the necessary infrastructure for a system, as well as to provide information in order to improve the response times of the system.

To carry out tests, the load scenarios must be previously defined and simulated through automated scripts, which represent the user's operations in the system.

Using load generation tools, we execute the defined scripts and monitor the different aspects of the application, such as response times, and resource consumption, among others.

As a result, hardware and software configuration problems, link problems, and even performance problems due to implementation, can be detected.

Performance testing is the most effective mechanism for minimizing risks in the production start-up, such as high response times, inadequate resource consumption, or service interruptions.

1.1.2 Types of Performance Tests

There are different types of performance tests, all of which pursue different objectives.

- Load tests
- Stress tests
- Spike tests
- Endurance tests
- Scalability tests

1.1.2.1 Load Tests

Load tests are performed to evaluate the system's behavior under an expected number of concurrent users who perform a specific number of transactions during a predefined time.

These tests make it possible to measure the response times of the transactions, as well as to evaluate if the system resources limit the system's performance.

1.1.2.2 Stress Tests

Stress tests are performed to determine the system's behavior when faced with extreme loads. They are performed by increasing the number of concurrent users and the number of transactions they execute, exceeding the expected load.

They give an insight into how the system's performance is, in case the actual load exceeds the expected load, determining which components and/or resources fail first and limit the system's performance.

1.1.2.3 Spike Tests

These types of tests are very similar to stress tests, but are performed over short periods of time, simulating significant load changes at a given time.

They allow us to know the behavior of the system when there is a spike in its use, and to evaluate whether the system is able to return to a stable state afterwards.

1.1.2.4 Endurance Tests

Endurance tests allows us to determine whether the system can support an expected load continuously during a period of time that corresponds to the context of the system use.

They help evaluate the performance of the different resources; for example, if there are memory leaks, or degradations due to bad management of the database connections, among others.

1.1.2.5 Scalability Tests

Scalability tests are performed to evaluate the system's ability to grow. Typically, the number of concurrent users, the number of transactions they can perform, the volume of information in the database, and other non-functional aspects of the system, are projected into the future.

1.1.3 Performance Testing Methodology

The methodology to carry out performance tests is as follows:

1.1.3.1 Test Planning

This phase consists of setting up the performance testing plan. During this phase, we define the test scenarios, in which we identify the different types of users who interact with the system and the flows they perform. We determine the number of users in each scenario and the number of transactions they will execute to simulate the use of the system.

As part of the performance testing plan, consider these as well:

1.1.3.1.1 Test Data

This is where we define both the data in the database and the data to be used as input for the transactions to be executed. The data must be similar to what is expected in production, both in quality and quantity; otherwise, we would not be adequately modeling reality and may not be able to observe many of the potential problems in the tests.

1.1.3.1.2 Test Infrastructure

This is where we define the infrastructure on which the tests will be run. In this regard, there are different options; two alternatives are presented below:

- The first one is to run tests on the production infrastructure, either because it is not being used or because it is possible to define a time window to use it.
- If this is not possible, we will have to set up an infrastructure identical to the production one to carry out the testing process.

The test environment should be monitored. We define primary indicators that allow us to observe the behavior of the different resources, and, if we need to obtain more information, we can implement second level indicators.

1.1.3.1.3 Acceptance Criteria

It is important to set one or more acceptance criteria or objectives to be achieved. These can be defined based on the number of transactions to be executed in a time window, the consumption of system resources, or the response time of each transaction, any of which is measured in the system load situation defined by the corresponding scenario. Acceptance criteria must be defined early on, and it is a good practice for them to be part of the system requirements. When the acceptance criteria are met, then performance tests are completed.

1.1.3.2 Test Scenario Specification

We define the tests that make up each of the scenarios in detail. We use a template that details each step and the final check that must be performed to ensure that the test has been successfully completed. These scenarios are validated with members of the technical team and the business team to ensure that the behavior expected in reality has been modeled.

1.1.3.2.1 Test Scenarios

The scenarios specify the different usage conditions that the system must support when used in production. Generally, a scenario is associated with a certain time of the day. For example, we can define a “daytime scenario” as the operation that determines the load on the system from 1:00 PM to 2:00 PM each day. This expected load is taken as a reference and is called a 100% scenario, different percentage loads are then executed through different transactions.

In addition, you must specify the number of users that execute each transaction, the number of iterations for each user, how users log on to the system, and which processes they can execute in parallel in that scenario.

1.1.3.2.2 Test Scripts

Test scripts are similar to test cases, for performance testing. An accurate test script must be defined for each transaction, including every action the user performs on the system, how it is performed, and the corresponding system response.

1.1.3.3 Test Automation

The automation stage aims to build the scripts that will carry out the transactions, following the script defined in the previous stage.

For the construction of the scripts, there are several tools that can be used to create them, and then to efficiently execute hundreds of virtual users. These tools provide the ability to enter the different parameters for the defined scenario, such as the number of users that execute each transaction, the number of times that each user executes them, the way in which the users enter the system (cadence), the time in which the scenario is executed, etc.

This stage of the methodology is the focus of Performance Testing United, and the tool to be used is Apache JMeter.

1.1.3.4 Test Environment Setup

The test environment setup consists of different activities that enable the preparation of the execution environment.

Some of the main activities include:

- Preparation and configuration of the load generating machines.
- Installation of the monitoring tools in the different system components, and configuration of the selected indicators.
- Preparation of the environment where the system is deployed, with its data and necessary configurations.
- Implementation of decisions related to the system's interoperability.

1.1.3.5 Test Execution

We run different tests (individual functionalities, baseline, load scenarios) in which we analyze the system behavior, resource usage and the response times obtained.

1.1.3.5.1 Baseline

Baseline execution consists of executing each transaction with a single user and all the infrastructure dedicated to it. The objective is to know the best times that can be obtained in each transaction and thus have a reference point to analyze the impact of the different concurrency scenarios.

1.1.3.5.2 Scenario Execution

The execution of the defined scenarios is carried out gradually. We must define how the load will be increased according to the reality of the project, although a 20% increase is generally applied in each test. For this reason, we execute 20% of the load, then 40%, 60%, 80% until we reach 100%, which is the target scenario.

1.1.3.5.3 System Monitoring

During the course of the tests, the system performance indicators should be monitored using the corresponding tools.

All the information collected at this stage should be saved for later analysis.

1.1.3.6 Results Analysis

Based on the data obtained in the previous stage, the corresponding analysis of the performance of the system and the components of its architecture during the tests must be carried out.

We suggest supporting the analysis by using graphic tools that simplify the data interpretation.

1.1.3.6.1 Re-execution of the Tests

From the monitoring of the indicators and the analysis of the results obtained in the execution of the test scenarios, opportunities for improvement can be expected. These improvement opportunities are applied in the system through adjustments.

After the adjustments are applied, we should re-run the performance tests to assess the impact of introduced adjustments.

1.1.3.6.2 Test Report

Once our objectives have been achieved, we proceed with the creation of a test report, which details the results we have obtained.

1.1.4 Performance Testing Tools

To carry out performance tests, different types of tools are available. These are detailed below.

1.1.4.1 Script Construction

The construction of the scripts is performed using the JMeter tool.

This tool offers the ability to record a flow in the system and generate a script that can be used as a starting point for the tests.

1.1.4.2 Load Generation

To execute the tests simulating a concurrent user load, we use load generation tools that take the defined script as input. JMeter includes this feature and, therefore, will be used for this purpose as well.

1.1.4.3 Resource Monitoring

While the tests are running, we monitor the system components. To this end, we use monitoring tools that take information from different indicators of the components at a hardware and a software level.

1.1.5 Introduction to the HTTP(S) Protocol

Billions of JPEG images, HTML pages, text files, video, audio, and more, surf the Internet every day using the HTTP(S) protocol. The Hypertext Transfer Protocol (HTTP) or its version with built-in security, Hypertext Transfer Protocol Secure (HTTPS), are the most widely used application protocols on the Internet to date.

It is important to have a thorough understanding of the HTTP(S) protocol before we start creating the scripts. By understanding this protocol, we will be able to assimilate how scripting works in performance testing more quickly.

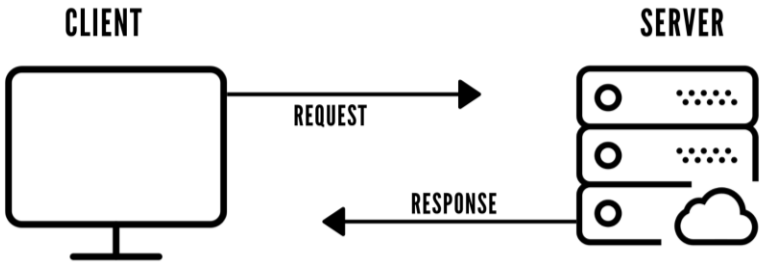
The protocol makes it possible for a client to communicate with a server. The client initiates a request that is sent to the server and the server returns a response associated with the request sent from the client.

Like most Internet protocols, this is a text-based request and response protocol, which uses a client-server communications model.

The HTTP(S) protocol is a stateless protocol, which means that the server is not required to store session information and that each request is independent from each other.

When a script is recorded using the JMeter tool, the tool captures all communication between the browser and the web server. The captured information becomes part of the script.

When the script is executed through JMeter, it interacts with the web application, simulating a real user's behavior.



[The image shows the user (client) on the left side, who interacts through a request with the server. It returns a response to the request made.]

1.2 JMeter Basic Concepts

1.2.1 Introduction to JMeter

JMeter is a free and open source tool created by the Apache Software Foundation to carry out performance testing. The tool is a Java-based application and was originally designed to test web applications.

It allows for load simulations to be carried out through different types of applications and protocols:

- Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, etc.)

- SOAP/REST web services
- FTP, FTPS
- Database via JDBC
- LDAP
- Message-oriented middleware (MOM) via JMS
- Mail - SMTP, POP3 e IMAP
- Native commands or shell scripts
- TCP
- Java objects

JMeter features an integrated development environment (IDE), which allows for tests to be created quickly using its script-recording function from a web browser and to be subsequently executed.

It includes a command line execution mode, which makes it possible to run the tests from any Java-compatible operating system (Windows, Linux, Mac OS, among others).

It provides the means to export the test results through a complete HTML report.

JMeter allows the extension of its functionalities through add-ons. Several are available through the Apache community, but it is also possible to build new ones when needed.

It also allows an integration with tools and libraries in continuous integration schemes, such as Maven, Gradle and Jenkins.

Understanding that JMeter is not a web browser is essential; the tool works on a protocol level. More specifically, JMeter does not execute embedded code on the pages as a web browser would; therefore, it does not execute JavaScript and does not render HTML pages.

1.2.2 Installation of JMeter

To run JMeter, you must have Java 8 or higher installed.

To start using JMeter, download it from the following address:
http://jmeter.apache.org/download_jmeter.cgi

JMeter is downloaded as a zip file; it must be unzipped in order to run.

To run JMeter, you must execute the file corresponding to the operating system in the "bin" directory: the "jmeter.bat" file in Windows and "jmeter.sh" in Linux and Mac OS.

JMeter will be executed in "GUI" mode, which means that it will show its graphic interface to create the test scripts.

Chapter 2 - Basic Scripts

LO7	Understand and apply the main elements to build a test plan in JMeter. (K2)
LO8	Create basic scripts by recording a session and running it in JMeter. (K3)
LO9	Analyze test results through various reports. (K3)

2.1 Test plan

2.1.1 Main Concepts of the Test Plan

A test plan in JMeter defines a tree structure of how, when and what to test, providing for the execution of a sequence of actions.

Given the tree structure that JMeter manages, it is important to understand that the elements have a hierarchy where the ones above apply to the elements below.

These elements are grouped into categories, which makes it easier to find each of them when creating a test script.

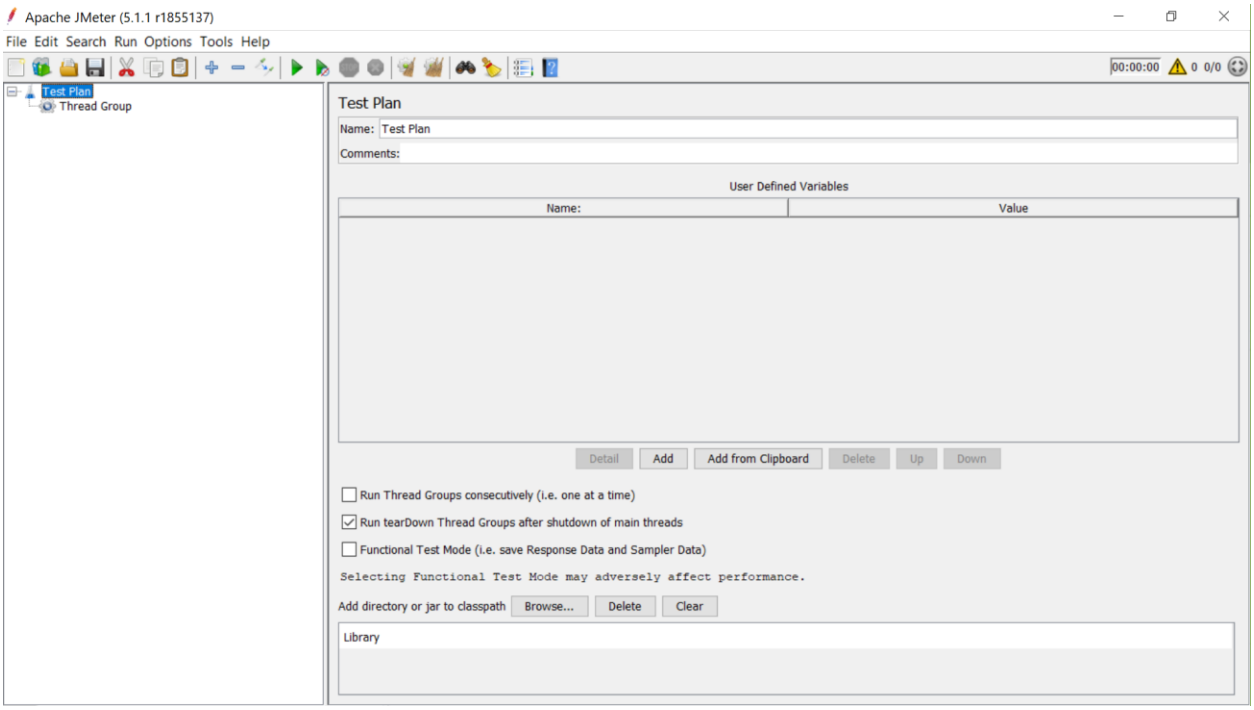
The element categories that JMeter handles are:

- Requests (Sampler): the different types of requests for different protocols can be found here.
- Logic Controllers: the logic controllers that allow you to define the flow of the tests are located here.
- Pre Processors: this category comprises all the elements that can be executed before making a request.
- Post Processors: this category comprises all the elements that can be executed after making a request.
- Assertions: all the elements used to carry out verifications and validations during the execution of the tests are grouped here.
- Timer: the elements related to waiting times are located here.
- Script fragments (Test Fragment): this category contains a single element with the same name. This element allows us to define script fragments to

be reused in different sections of the script.

- Config Element: this category contains all the configuration elements for the script.
- Listener: the elements related to the results listeners and test execution reports can be found here.

A test plan can include elements from the categories mentioned above, which are located in the tree shown in the image below on the left side. Depending on the desired behavior, the elements can be nested or located in parallel with others.

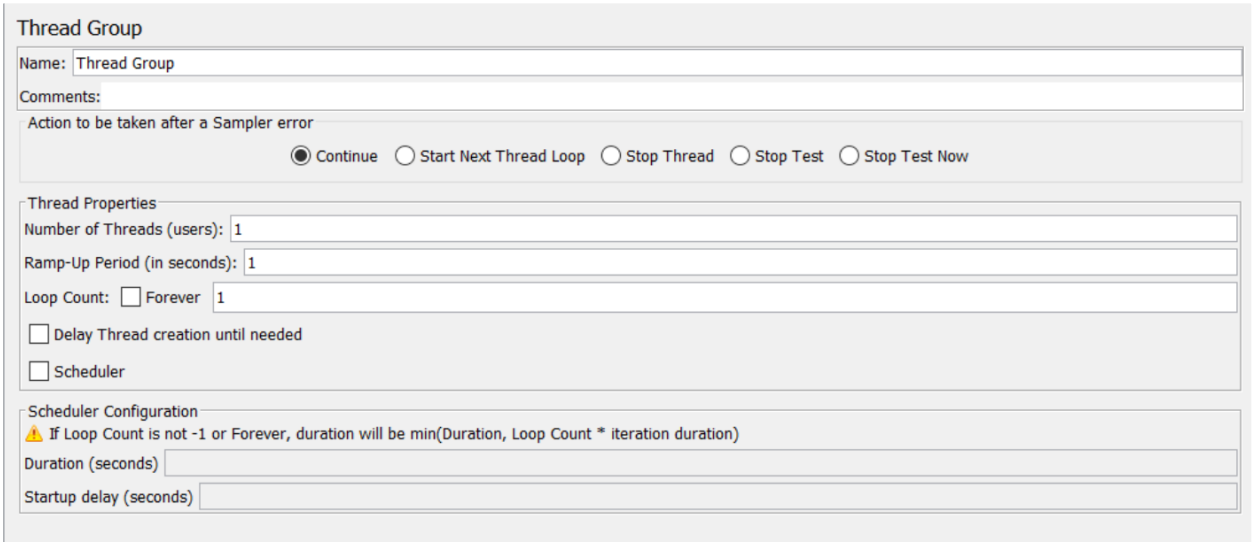


2.1.2 Threads/Users

The threads or users are represented in JMeter through groups of threads and constitute the initial element to start building the test plan.

Each group of threads represents a set of system users, and they are used to execute the performance tests to simulate the concurrence of users on the application.

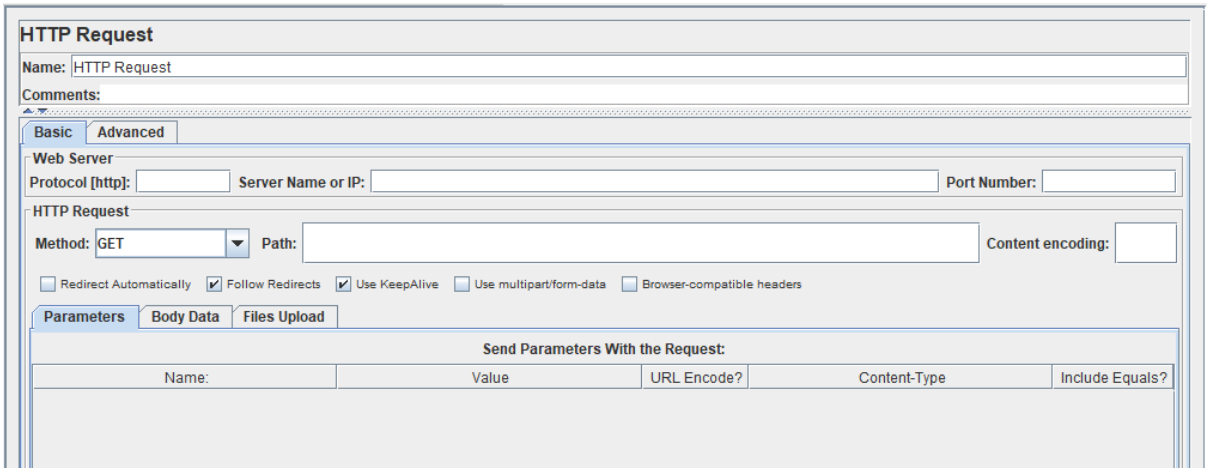
Other elements, such as HTTP(S) requests, controllers, etc., depend on the thread groups. We will present this later.
The thread group in JMeter is displayed as follows:



2.1.3 HTTP(S) Requests

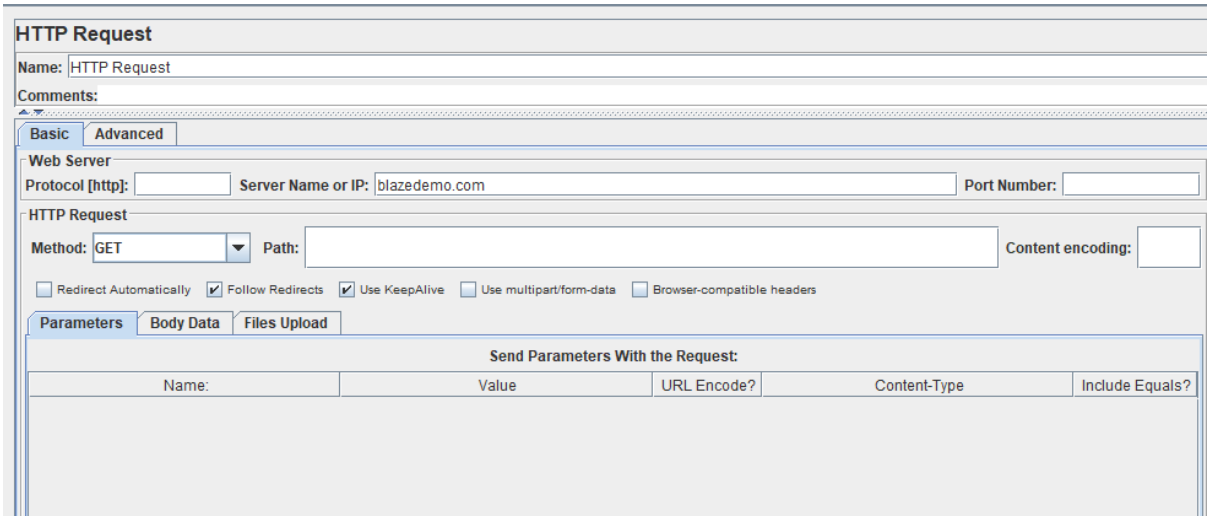
HTTP(S) requests are simulated in JMeter through a specific element. Each request is handled independently and is set up to call a specific URL.

The basic view of this element is as follows:

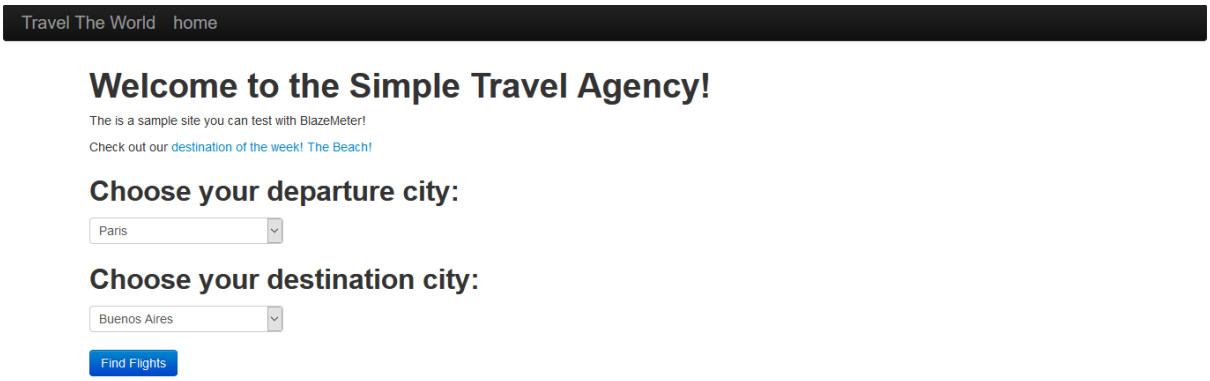


Two examples of different HTTP(S) requests are presented below.

The first example is a GET request to a home page located at the URL http://www.blazedemo.com.

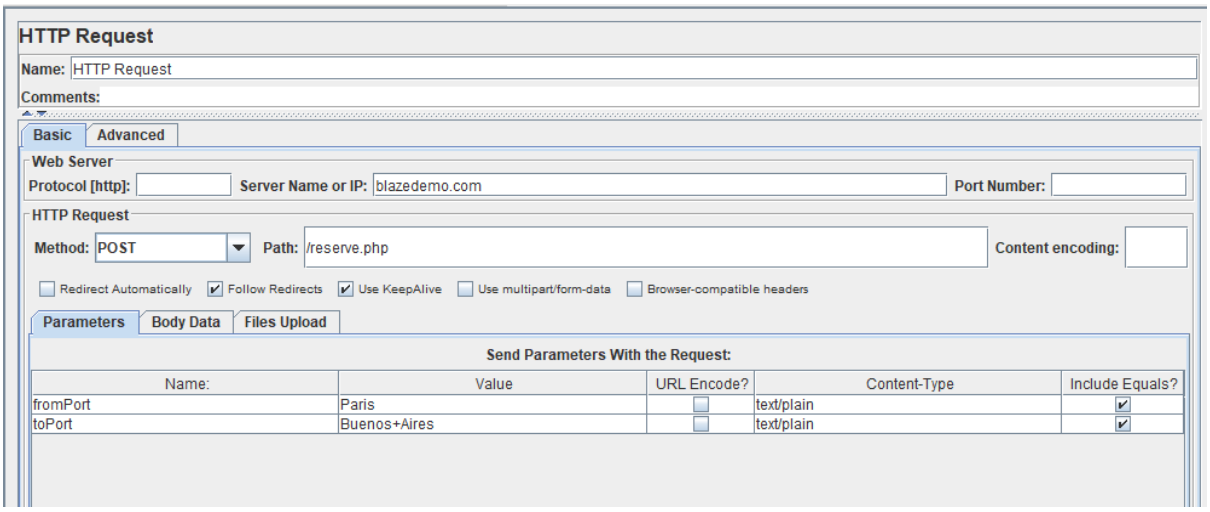


This request simulates the loading of the next page:

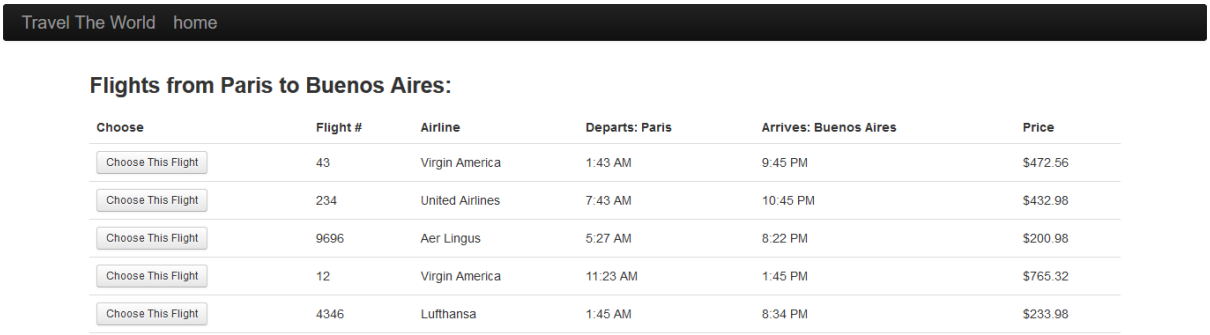


Later on, we will present a way to verify in JMeter if the petition was correctly uploaded. In this case, we can see that it is possible to visually verify this if the text “Welcome to the Simple Travel Agency” appears on the screen.

The second example is a POST request that is generated by selecting the departure and arrival airport by pressing the “Find Flights” button. In this case, the object that is invoked has the name “reserve.php”.



This request should result in a page as shown below:



2.1.4 Cookie Management

Cookies are small files that are stored on the client and contain user-related information in the context of the web application you are using. It is common for cookies to store information about the session of the user running the application, as well as other data that allows the user to have a personalized experience when using the application. In JMeter, cookies are handled through a specific element that stores and sends cookies in the same way a web browser does.

When we make an HTTP(S) request, a new cookie may come in the response, which is automatically stored in this element in JMeter, and is sent in each of the following test script requests.

2.1.5 Cache Management

The cache is an important element that is part of all current web browsers and makes it possible to store some resources locally without constantly needing to go to the server to find them. These resources are typically static, such as images, CSS, JavaScript, etc.

A good management of the cache has a positive impact on the performance of web applications. Caching is primarily used to reduce the latency and download times of resources, and to reduce network traffic between the client and the server.

JMeter allows cache management through a specific element. It is important to remember at this point that JMeter is not a web browser, so it emulates the behavior of the web browser; however, the implementation of the cache may be different in each case.

2.1.6 Handling of HTTP(S) Headers

HTTP(S) headers are sent to the server from the client with the additional information required to meet specific server requirements and thus respond appropriately to the client's request.

JMeter provides a specific element to attach that additional information to the request, or in some cases override headers.

2.1.7 HTTP(S) Request Defaults

It is common for HTTP(S) requests within a test plan to be made using the same values; for example, in the case of server name and port.

There is an element in JMeter where these values are configured and taken over by the default HTTP(S) requests.

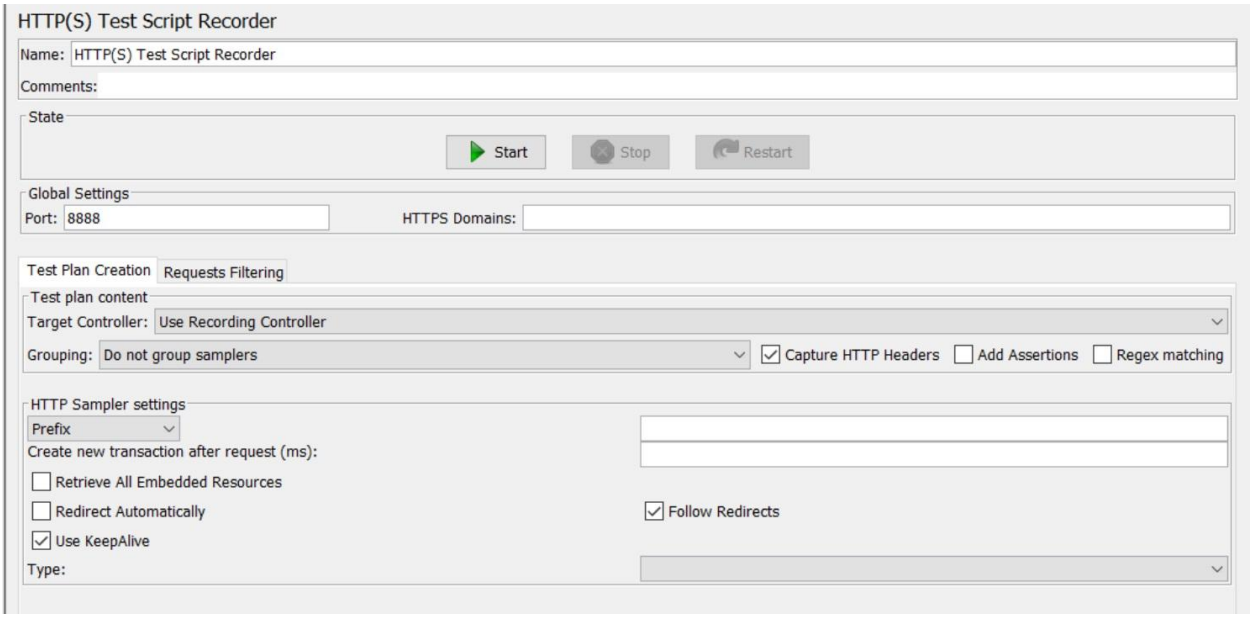
It is a good practice to use this element; we may need to restructure the test scripts and run them on another server. If you have this element and it is used correctly, making these changes is a very simple job.

2.2 Recording and Execution

2.2.1 Script Recording

JMeter allows the recording of a user's navigation when interacting with a web application, which makes the creation of test scripts easier and more precise.

For this purpose, JMeter includes a specific element that is displayed as follows:



The following procedure is used to make a recording:

1. Set the port where the recording will be made.
2. Select which the main driver will be. When selecting one of them, all the recording remains inside this element.
3. Click the "Start" option to start the recording.
4. The application will be accessed through a web browser; therefore, it must be configured to use JMeter as a proxy.
5. It is now possible to start recording. To do so, access the web application through the URL and start running the test flows.
6. JMeter will start generating the requests corresponding to the interactions with the web application.
7. The requests will be recorded within the selected thread group to be used as part of the test script.

2.2.2 Embedded Resources

When running performance tests, we need to simulate the behavior of a real user interacting with the application. This means that when a request is sent to the

server, it returns the response and all the associated resources, usually different file types, such as images, CSS, JavaScript, etc.

JMeter provides the option to exclude embedded resources in order to make recordings clearer and more understandable. Excluding embedded resources in the recordings does not mean that they are excluded in the execution of the tests.

For tests to be as close as possible to the actual behavior of the application, it is possible to configure HTTP(S) requests to retrieve all embedded resources.

By doing so, the script execution behavior will be similar to the behavior of a user using a web browser, where all the embedded resources are downloaded.

2.2.3 Script Execution

Once the script is defined, we need to be aware of the different execution modes JMeter offers.

The script can be executed from the “Run” menu, using the “Start” option, or through the toolbar, pressing the  button.

Once the script execution has started, the following menu options are enabled:

- The “Stop” option immediately stops the execution of the script.
- The “Shutdown” option stops the script execution once all running elements have finished.

In addition to the menu options allowing you to stop the script being enabled, it is possible to see some information in the top right corner while the script is running. The elements present there are the following:

This icon  indicates that the script is not being executed.

If the icon is green  , it indicates that the script is running.

It is also possible to observe the time the script has been running, as well as the errors found during the execution of the script, and how many users are active over the total amount.

2.3 Results Listeners

2.3.1 Introduction to Listeners

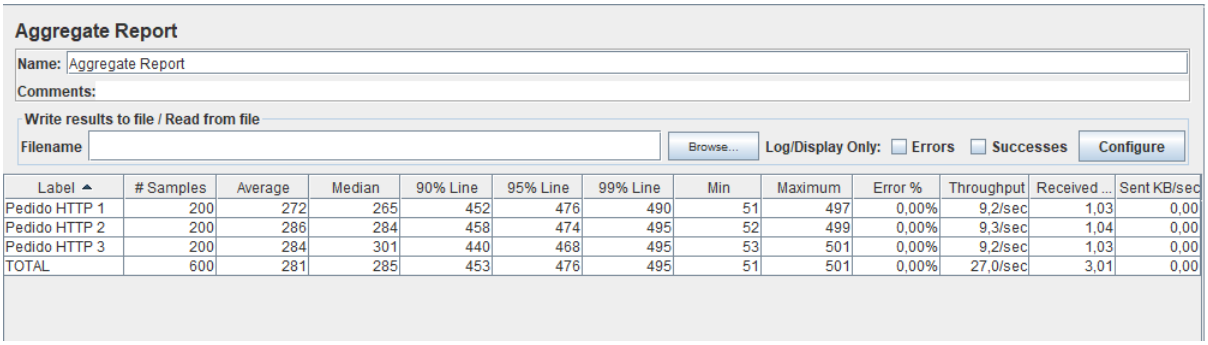
A results listener is a component of JMeter that displays test results. Results can be displayed in a tree, in tables, graphs, or simply written to a log file.

Most listeners allow for the storage of test results on the one hand, but they also have the ability to open a previously saved file and display the results in the listener's format.

2.3.2 Summary Report

This report is one of the simplest in JMeter, yet it is widely used to observe the results obtained at a high level.

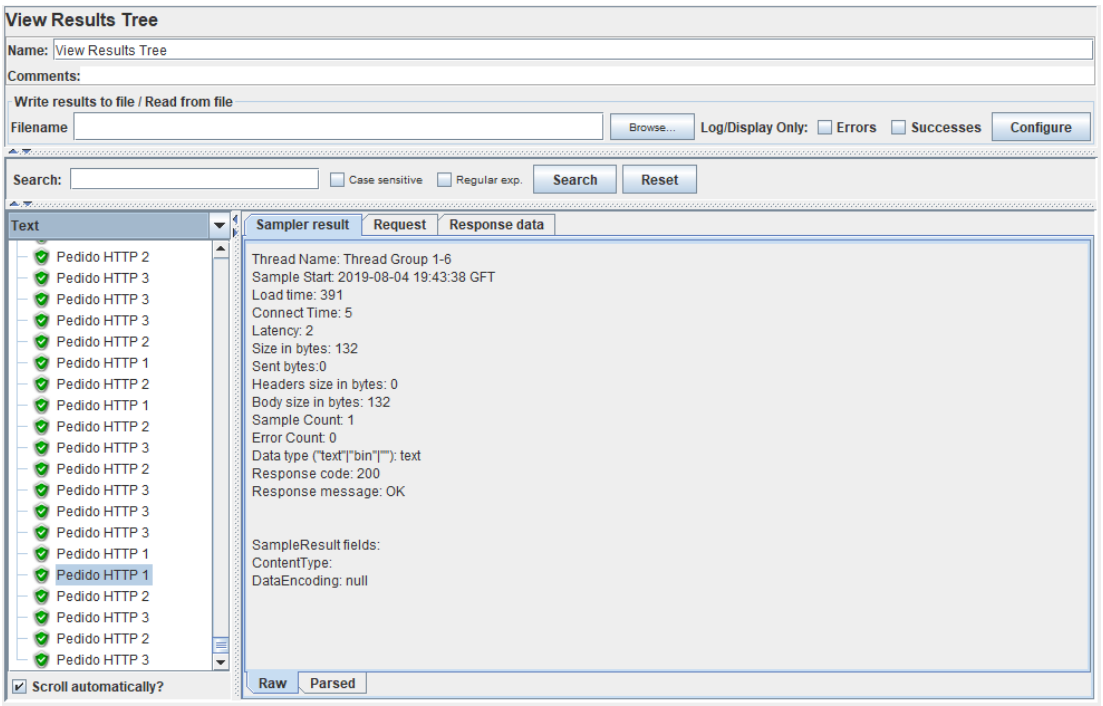
It generates a table where the test results are grouped by the name of the HTTP(S) requests.



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received ...	Sent KB/sec
Pedido HTTP 1	200	272	265	452	476	490	51	497	0,00%	9,2/sec	1,03	0,00
Pedido HTTP 2	200	286	284	458	474	495	52	499	0,00%	9,3/sec	1,04	0,00
Pedido HTTP 3	200	284	301	440	468	495	53	501	0,00%	9,2/sec	1,03	0,00
TOTAL	600	281	285	453	476	495	51	501	0,00%	27,0/sec	3,01	0,00

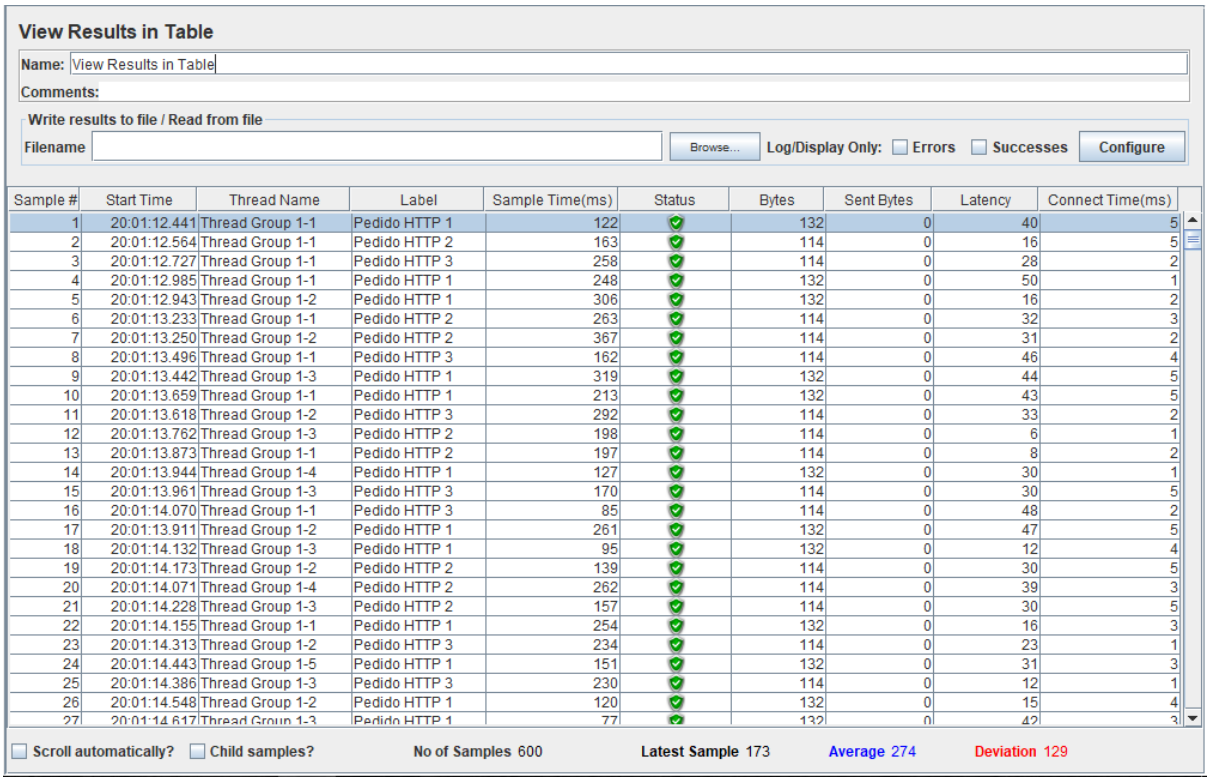
2.3.3 Results Tree

This report provides a comprehensive view of all requests and their responses in a tree structure. In addition to the HTTP(S) request and its response, it also displays general values such as response time, number of bytes sent and received, and other data.



2.3.3 Results Table

This report makes it possible to analyze the results in a detailed table, where each line corresponds to each of the executed requests.



Chapter 3 - Advanced Scripting

LO10	Understand the concept of correlation and how to use regular expressions. (K2)
LO11	Apply the use of regular expressions to manage correlation in JMeter. (K3)
LO12	Understand the concept of parameterization. (K2)
LO13	Build and configure data sources to be used in the test script. (K3)
LO14	Understand and apply timers, assertions and controllers in JMeter. (K3)
LO15	Debugging a script in JMeter. (K3)

3.1 Correlation

3.1.1 What is Correlation?

Correlation is one of the most difficult tasks in script automation. It is the process of capturing and storing the dynamic values sent from the server and including them in the requests from the client.

A value is considered dynamic when it can return different data for each request in each iteration.

It is a critical process during the execution of the test scripts. If it is not managed correctly, the script will not be executed correctly.

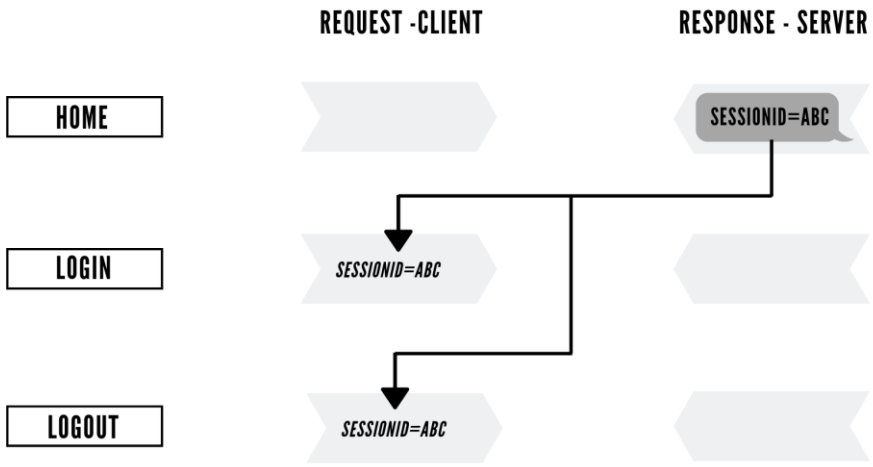
To carry out this process, we must first identify which values are dynamic, then find the response in which the server sends these values, extract these values through regular expressions, and finally substitute these values in the requests from where they are sent back to the server. In many cases, there are several dynamic values with different associated structures.

The following image represents a test execution that consists of going to the main page of the application, logging in and logging out.

Each time a run is performed, the value of the “SessionID” variable will change. In the example, the variable is sent from the server in the first request from the

main page, then the application will be using this value to execute the rest (login and logout). In this case, the variable will have the value “ABC”. If there is another execution of the same functionalities later on, the “SessionID” variable will take another value.

The correlation process should offer a solution to this situation.



[The image shows the interaction between the user (client) and the server (server), for three different requests (home, login, logout)]

When we make a recording with JMeter, the values for each of the requests are captured. JMeter does not have an automated correlation process; this must be done manually.

If this is not performed, the dynamic values will be sent as if they were static, and the tests will not be executed correctly.

3.1.2 Introduction to Regular Expressions

One of the most important elements to consider for the correlation process are regular expressions, also known as “Regex”.

A regular expression is a sequence of characters used to search for character strings or substitution operations patterns; these are widely used to extract dynamic values in responses sent from the server.

Certain characters (meta-characters) are used to indicate special functions within

the pattern, such as alternatives or repetitions, so they are no longer called literals; this means that they no longer represent their natural value. These characters are: \ . ^ $ [ ] ( ) | ? + *.

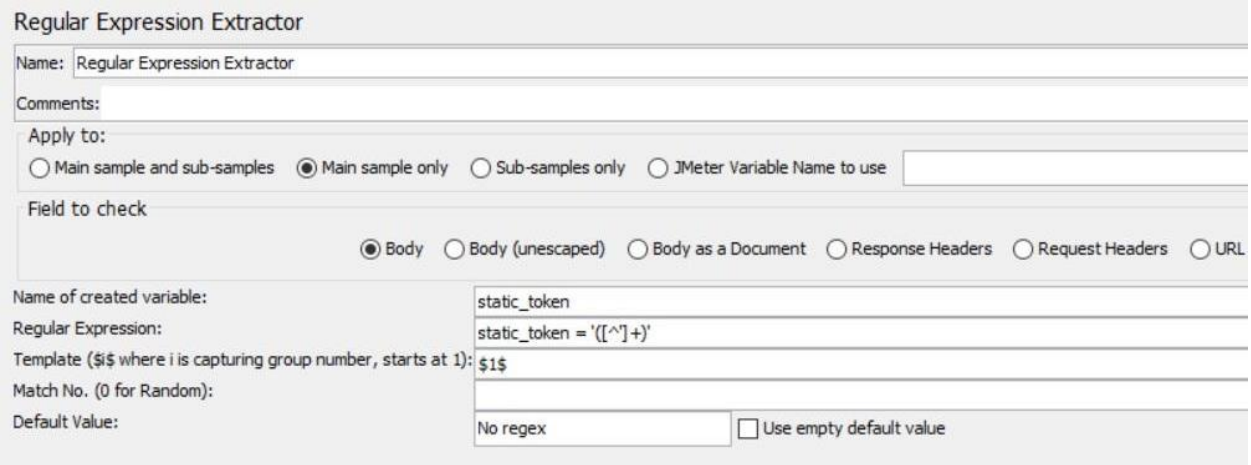
One of the most frequently used regular expressions in JMeter is (.+?) since there are many dynamic values with a similar structure that can be captured with this regular expression. However, dynamic values do not always display the same structure.

3.1.3 Regular Expression Extractor

To use regular expressions in JMeter, there is a regular expression extractor component. This element allows us to extract values from a server response using regular expressions.

This element will be executed after each request, applying the regular expression, extracting the requested values, and storing them in a specific variable.

In JMeter this element appears as follows:



3.2 Parameterization

3.2.1 Variables

In JMeter, as in different programming or scripting languages, we can define variables. The variables are local to each execution thread.

In contrast to programming languages, variables in JMeter are all of the same type: text type variables.

There are different ways to define variables in JMeter, the most common being the “User Defined Variables” element.

These variables can be used in the script. To do so, they should be placed between curly braces “{ }” braces preceded by a dollar sign.

Therefore, in order to use a variable called “name”, you must do as follows: “${name}”.

At runtime, that variable will take a value that will be assigned dynamically.

3.2.2 Functions

JMeter features several predefined functions, which are invoked using specific names.

The functions can be called as follows:
${__functionName(var1,var2,var3)}

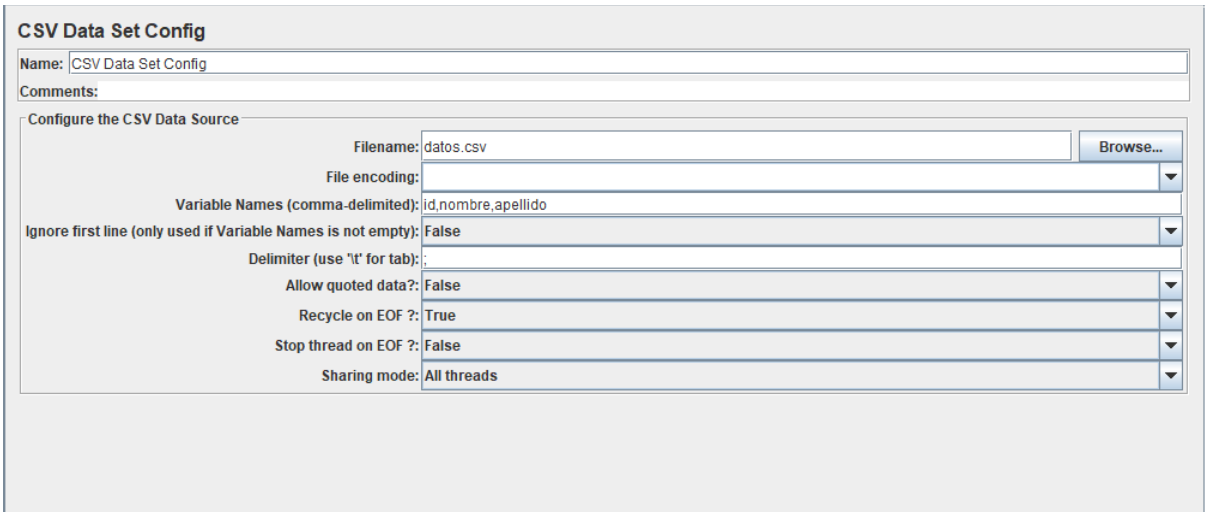
Some functions need parameters to work. In other cases, parameters are optional.

Functions are grouped in JMeter according to their type: information, input, calculation, formatting, scripting, properties, variables and text.

3.2.3 CSV Data Source

This element allows us to read data from a file, which will be used for HTTP(S) requests. The data is obtained from the file, line by line, and is stored in variables to be used in JMeter.

This element appears in JMeter as follows:



The “data.csv” file referenced in the previous item is in the following format:

```
1;Homero;Simpson
2;Marge;Simpson
3;Lisa;Simpson
4;Bart;Simpson
5;Montgomery;Burns
6;Apu;Nahasapeemapetilon
7;Milhouse;Van Houten
8;Ned;Flanders
9;Moe;Szyslak
10;Nelson;Muntz
```

Therefore, it is possible to use the “id”, “name” and “surname” variables in JMeter in script requests, and the data these variables will take will be those from the indicated file.

3.3 Think Times

3.3.1 What are Think Times?

When users use an application, they always make short pauses between the actions that they execute.

In performance testing, the time between user interactions is called think time.

Think times play a key role in scripting, as they are intended to simulate actual user behavior when interacting with the application.

In JMeter, the elements called timers are used to simulate think times.

Timers are processed before each request, according to the level where they are defined. If there is more than one timer within the same level, all timers will be processed before each request. A timer that is not on the same level as a request will not be processed.

3.3.2 Constant Timer

The constant timer is used to add a constant wait time (delay) between each request. In JMeter, this value is specified in milliseconds.

3.3.3 Other Types of Timers

Other timers are available to simulate different behaviors.

The uniform random timer delays each request for a random period of time, with a set maximum, each of the time intervals having the same chance to occur.

Another commonly used timer is the Gaussian Random Timer, which allows for the distribution of random response times in a Gaussian curve distribution.

3.4 Logic Controllers

3.4.1 What are Logic Controllers?

Logic controllers can be used to define different flows for requests and for their order.

The concepts handled by JMeter are similar to those used in any programming language, such as the conditional “If”, loop controller “Loop”, etc.

3.4.2 “If” Controller

The “If” controller allows us to control whether the test items under it (its children) are executed or not.

By default, the condition is evaluated only once at initial entry, but there is an option to have it evaluated for each item.

For example, if there are two types of users in an application, a basic user and an administrator user, and their authentication information is different, the script should be able to handle both types of users and execute the requests.

3.4.3 “Loop” Controller

The “Loop” controller allows us to execute requests a set number of times or

indeterminately.

For example, if you want to execute a specific request 5 times.

3.4.4 “Simple” Controller

We use the “Simple” controller to organize and store requests and other logic controllers. It offers no other functionality; it simply organizes the script to make it easier to understand and maintain.

3.4.5 “Transaction” Controller

The “Transaction” controller can be used to group a set of requests and other logic controllers in order to measure the transaction response time of the controller.

The controller generates an additional sample in the reports, where it shows the total time of the transaction.

3.5 Assertions

3.5.1 What are Assertions?

In JMeter, assertions are used to validate responses to requests that have been sent to the server.

Assertions should be placed as a secondary element within the script, either in the test plan, thread group, controllers, requests, etc.

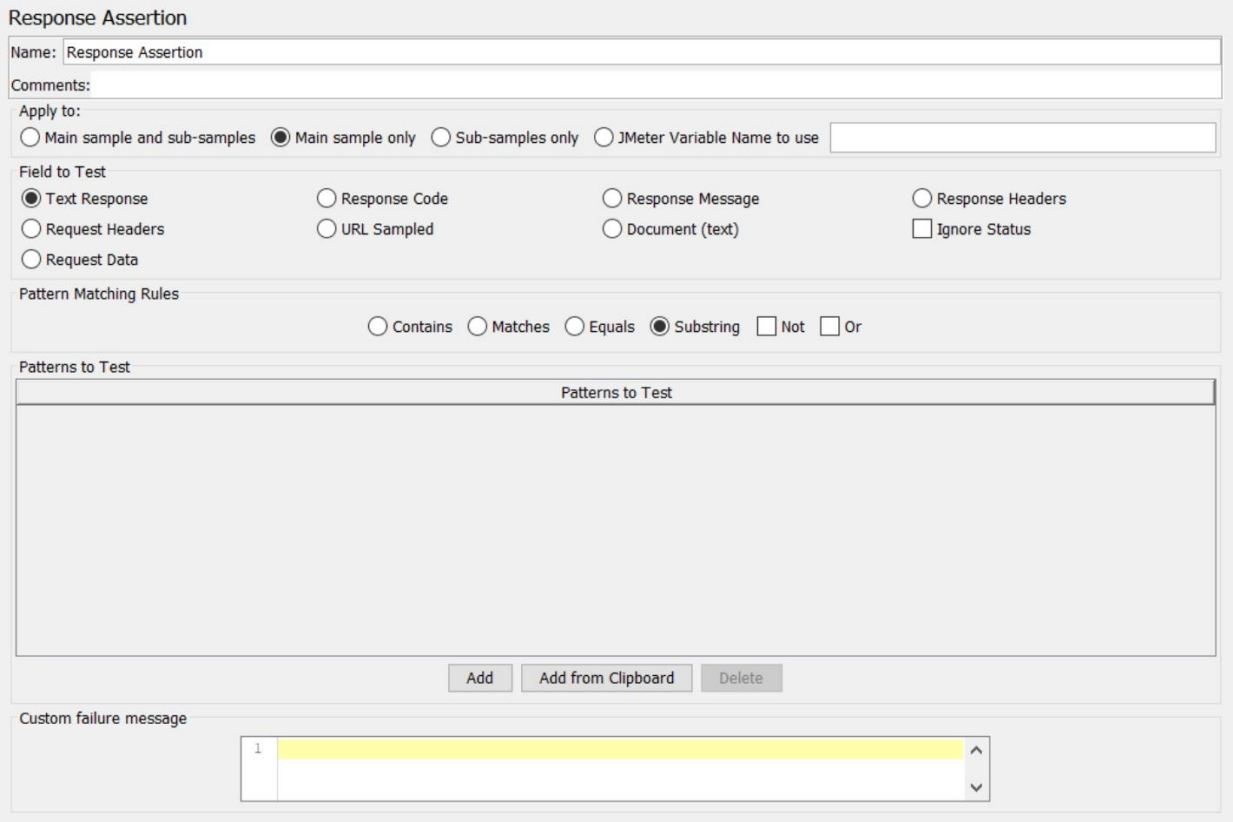
They help verify and ensure that our performance tests are being executed successfully and that the server under test is returning the expected results correctly. They can be either negative or positive.

For example, we can check that the response to a certain request contains a text string (positive), or that the response to a certain request does not contain the text string (negative).

3.5.2 Response Assertion

The response assertion allows us to compare strings with the response coming from the server.

In JMeter, this element appears as follows:

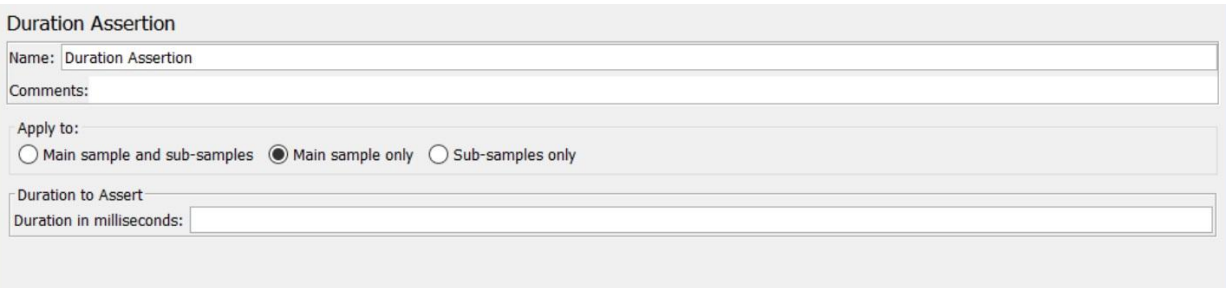


For example, when we log into the application, the text “Welcome" is displayed, so it is possible to define an assertion in JMeter that verifies this behavior. In the case where the answer does not contain this text, the assertion fails.

3.5.3 Duration Assertion

The duration assertion allows us to verify that the responses from the server are received within a given time frame. Any response that takes longer than the configured number of milliseconds is marked as a failed response.

In JMeter, this element is displayed as follows:



3.6 Debugging the Script

When building a script, we may find behaviors where it does not function properly, and the cause of the error may not be easy to detect. For these cases, in order to identify the error, we need to ensure that all the variables actually have the expected value at the right time.

It is possible to add a specific element in JMeter that allows to visualize what is happening when the script is executed and to therefore debug it.

Chapter 4 - Testing

LO16	Preparation of the scripts for the execution of the tests. (K3)
LO17	Run tests using the command line mode. (K3)
LO18	Execution of the tests through the distributed mode. (K1)
LO19	Understand how system resources are monitored during performance tests and their main indicators. (K2)
LO20	Apply basic monitoring tools during performance test execution. (K3)

4.1 Test Execution

4.1.1 Preparing the Script

There are several activities to be performed prior to the execution of the tests, some related to the script itself and others that have to do with system availability and monitoring configurations.

Regarding the script, we should consider removing elements used to debug it, results listeners used temporarily, configuring variable settings, enabling secondary requests, and including think times, among other aspects.

In addition to this, we should also verify that the script works correctly with different sets of data, executing with more than one user. This means that the script must be tested before starting to execute the tests. At this point, we should verify that the script executes from start to end, and that the volume of data to be used depends on the number of threads/users and iterations to be executed.

There are also some environment configurations and data checks that should be performed prior to the execution of the tests. For example, the environment should be available with all its components, according to the setting established in the test plan. If there are components that are not part of the tests, it is important to check that they are isolated and to apply the agreed mechanism.

As part of the final checks, it is important to perform a monitoring test to ensure that the tools are working and are configured to take data from the indicators defined in the test plan.

4.1.2 Execution in Non GUI Mode

Once the script has been completed, it is more convenient to run JMeter in command line mode, since it consumes less resources in the generating machine from where the tests are launched.

This is especially recommended when the scripts are complex and handle large volumes of data.

By running in this mode, integration with other tools, such as Jenkins, is very easy to implement.

4.1.3 Distributed Execution

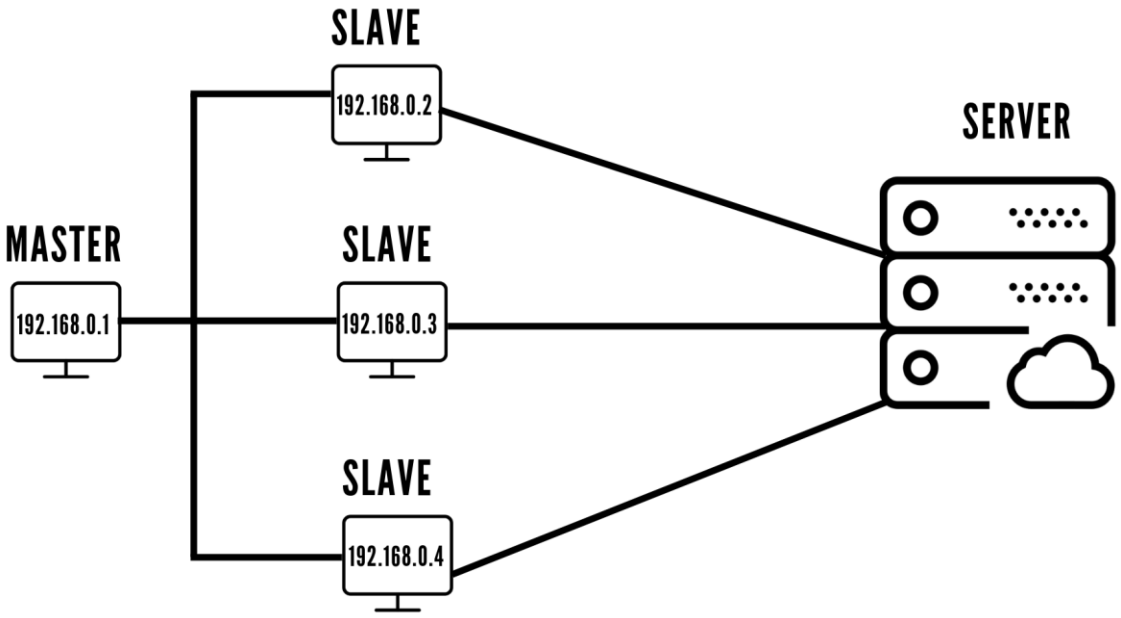
When the workload to be simulated is very large, we recommend using JMeter's distributed execution mode.

Since it is not possible to generate large amounts of threads/users from a single load generating machine, we can use multiple load generating machines for that purpose.

Before delving into the execution of distributed tests, we will introduce some terms:

- Master: the instance or machine from which we will be controlling the distributed tests, which are executed from other load generating machines.
- Slave: load-generating instances or machines.

The other Slave instances will be controlled from a machine holding the Master role, in order to achieve the expected load on the application.



[The image shows a Master instance that controls three Slave instances, which are the ones that execute the tests against the server]

4.2 Application Monitoring

4.2.1 Introduction to Monitoring

Monitoring is a fundamental activity when executing performance tests, in order to obtain relevant information to be used to analyze improvements in the application's performance.

JMeter provides information through results listeners. Each one of them presents the data through different reports. In some cases, the data obtained is consolidated and in other cases it is disaggregated.

It is important to note that, in JMeter, this information is obtained on the customer's side, as this is where the tool is located to perform the load simulation.

Although this data is very important, it is also required to perform monitoring on the server side. This section details how this is done.

When simulating load on a web application, it is important to understand what is happening at the server level and its infrastructure. It is necessary to define some indicators to understand whether the resources are working properly or whether

there is a problem.

For instance, if stress tests are run on the application and the HTTP(S) requests are not answered, it is critical to understand the cause of this problem.

When monitoring, it is common to detect situations in which one of the resources is saturated, and therefore it is likely to be the “bottleneck” in the application's performance.

Monitoring is a complex task and we recommend working together with the infrastructure and operations team in order to define the indicators that will be monitored, the collection of information during the execution of the tests and its subsequent analysis.

It is important to understand that, ideally, monitoring should be done at different infrastructure and software levels.

We need to understand that monitoring should ideally take place at different levels of infrastructure and software.

4.2.2 Primary Indicators

Indicators are used to evaluate the status of the various resources being monitored. The primary indicators are used for first level monitoring.

These are usually defined for different components of the solution, including:

- Servers (at operating system level)
- Application servers (such as web servers)
- Database servers
- Networks
- Application (application-specific indicators)

The resources that are commonly monitored through primary indicators for servers are:

CPU: as primary indicators for the CPU resource, they show its utilization percentage, as well as the queue length of the jobs to be processed.

Memory: for memory monitoring, it is important to know the server capacity and observe the memory available during the tests.

Disk: for disk monitoring, it is important to observe the reading and writing activity on the disk. It is also recommended to observe the length of the job queue for the disk and the available space on it.

Network: at a network interface level, observing the activity of sending and receiving data is relevant. It is recommended to know the capacity of the links used for the tests, between the different components of the solution.

4.2.3 Basic Monitoring Tools

While there is a very large variety of tools available to carry out monitoring during testing, there are two tools that are very useful for the two most widely used operating systems.

4.2.3.1 Windows Performance Monitor

In the case of solutions where the application server runs on a Windows operating system, it is possible to use the “Windows Performance Monitor” tool, also known as “perfmon”.

Use this tool to set up different indicators, which are grouped by category.

4.2.3.2 NMon

Alternatively, for solutions running application servers on Linux operating systems, we can use the “NMon” tool.

This is a command line type tool and it has two execution modes.

We use the first mode to observe the activity of the indicators in real time and can be activated by just executing the “nmon” command in the terminal. After starting up, it displays the different indicator options for viewing.

The second mode is to store indicator activity in a file that can then be analyzed.

Chapter 5 - Documentation

LO21	Documentation about performance tests. (K2)

5.1 Introduction

In the process of performance testing, it is important to define some documents, which support the implementation of activities and the recording of the results obtained.

The main documents are:

- Performance Test Plan
- Test script
- Results report

5.2 Performance Test Plan

The performance testing plan is the document that guides the testing process, where the main decisions and agreements are made.

It is composed of the following sections:

- Introduction
- Framework
    - Scope
    - Activity schedule
    - Team organization
    - Milestones and deliverables
- Test plan
    - System under test
        - System architecture
        - Interfaces with other systems
    - Risk/criticality analysis
    - Test environment and infrastructure
        - Integration with external systems
    - Acceptance criteria
    - Test scenarios
    - Test data
    - Load generation tools

    - Monitoring
        - Monitoring tools
        - Monitoring indicators

5.3 Test Script

The test script contains detailed information for each of the tests and their steps, which will be executed in the system.

Each test script contains the following information:

- Objective
- User who executes it
- Steps for execution (presented in table format)
    - Details of each step
    - Validations in each step
    - Think time
- Expected result at a global level

5.4 Results Report

The test result report contains the following information:

- Summary of the tests that were run
- Test results
    - Execution log
    - Base line
    - Details of the executed scenarios
    - Monitoring information for each scenario
- Improvements applied during testing
- Limitations
- Conclusions
- Recommendations

Chapter 6 - Extra

LO22	Understand best practices when using JMeter. (K2)
LO23	Running scripts for web services. (K3)

6.1 JMeter Best Practices

The following set of good practices can be used when using JMeter:

1. Use the latest version of JMeter.
2. Identify the correct number of threads/users.
3. Apply the appropriate JMeter elements for each specific function.
4. Start recording the script in JMeter.
5. Use proxy tools for the correlation process.
6. Centralize and standardize repeated settings.
7. Use timers to achieve realistic scenarios.
8. Verify responses with assertions.
9. Reduce resource consumption in the load generator.

6.2 Testing Web Services

A web service is a set of protocols and standards used to exchange data between software applications. These allow organizations to exchange data without them needing to know the details of their respective applications.

In short, web services can be operated in JMeter as HTTP(S) requests, including the headers and body format relevant to the service being invoked.

References

- The Apache Software Foundation. (2020). *Apache JMeter™.* Retrieved from https://jmeter.apache.org
- The Apache Software Foundation. (2020). *Apache JMeter™ Wiki.* Retrieved from https://wiki.apache.org/jmeter
- Microsoft Docs (2017). *Regular Expression Language — Quick Reference.* Retrieved from https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference
- MDN web docs moz://a (2020) *HTTP Protocol*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP
- NMon for Linux (2019). *nMon*. Retrieved from http://nmon.sourceforge.net/
- Microsoft Tech Community (2014). *Windows Performance Monitor Overview*. Retrieved from https://techcommunity.microsoft.com/t5/Ask-The-Performance-Team/Windows-Performance-Monitor-Overview/ba-p/375481