JMeter  
🡺JMeter also known as 'Apache JMeter' is an open source, 100% java based application with a graphical interface. It is designed to analyze and measure the performance and load functional behavior of web application and variety of services.

🡺jmeter is used in  like Performance, Load, Stress, Regression and Functional testing, in order to get accurate performance metrics against your web server.

Following is a list of protocols supported by JMeter:

* Web Services - SOAP / XML-RPC
* Web - HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
* Database via JDBC drivers
* Directory - LDAP
* Messaging oriented service via JMS
* Service - POP3, IMAP, SMTP

**JMeter Features:** Some of the most important features of JMeter are listed below:

## **Open source application, User-friendly GUI**

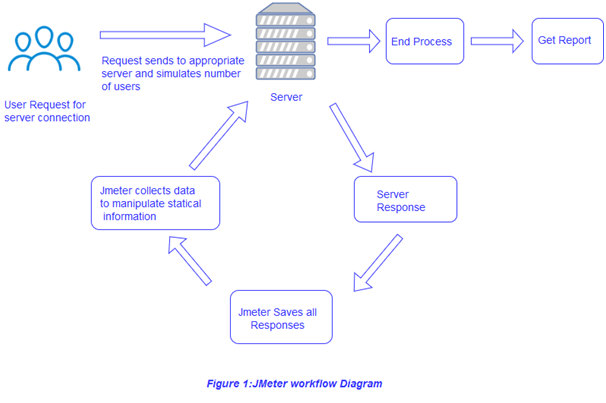
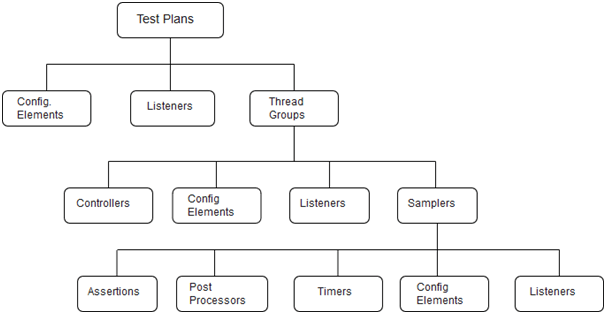
**Support various testing approach:** JMeter supports various testing approach like Load Testing, Distributed Testing, and Functional Testing, etc.

* Support various server types: JMeter is highly extensible and capable to load the performance test in different server types:
* Web: HTTP, HTTPS, SOAP,
* Database: JDBC, LDAP, JMS, and
* Mail: POP3.

**Support multi-protocol:** JMeter supports protocols such as HTTP, JDBC, LDAP, SOAP, JMS, and FTP.

* **Platform independent:** JMeter is written and developed using java, so it can run on any environment / workstation that accepts a Java virtual machine, for example - Windows, Linux, Mac, etc.
* **Support various server types:** JMeter is highly extensible and capable to load the performance test in different server types:

Web: HTTP, HTTPS, SOAP,  
Database: JDBC, LDAP, JMS, and  
Mail: POP3.

* **Support multi-protocol:** JMeter supports protocols such as HTTP, JDBC, LDAP, SOAP, JMS, and FTP.
* **Simulation:** JMeter can simulate multiple users by using virtual users or unique users in order to generate heavy load against web application under test.
* **Framework:** JMeter is a multi-threading framework which allows concurrent and simultaneous sampling of different functions by many or separate thread groups.
* **Remote distributed testing:** JMeter has Master-Slave concept for distributed testing where master will distribute tests among all slaves and slaves will execute scripts against your server.
* **Test result visualization:** Test results can be viewed in different formats like graph, table, tree, and report etc.**Distributed testing** is a method of performance testing where multiple machines (or nodes) are used to simulate a large number of users or requests to a system. This approach helps in assessing how an application performs under high load conditions by spreading the workload across several machines, rather than relying on a single machine.  
  **Slaves** in JMeter refer to the machines that execute test scripts during distributed testing. They work under the control of a **master** machine, which distributes the test load. Each slave simulates a specified number of users, runs the tests, and sends the results back to the master for analysis. This setup allows for testing with a larger number of users than a single machine can handle.  
    
  **Working of JMeter** sends requests to a target server by simulating a group of users. Subsequently, data is collected to calculate statistics and display performance metrics of the target server through various formats.  
    
    
   **Performance Testing:** Software performance testing is a type of non-functional testing in which the performance of the application is evaluated under expected or higher load. Performance testing is carried out to measure different performance attributes of the system like - **response time (speed), reliability, resource usage, scalability, Stability under variety of load conditions etc.  
  Why do we need Performance Testing?** Before launching the final software product in the market, the product should be tested against the speed, scalability and stability under variety of load conditions.

**Load Testing**is a non-functional software testing process in which the performance of software application is tested under a specific expected load. It determines how the software application behaves while being accessed by multiple users simultaneously**.  
This testing usually identifies –**The maximum operating capacity of an applicationDetermine whether the current infrastructure is sufficient to run the applicationSustainability of application with respect to peak user load**.**Number of concurrent users that an application can support, and scalability to allow more users to access it.

## Why Load Testing?

* Load testing gives confidence in the system & its reliability and performance.
* Load Testing helps identify the bottlenecks in the system under heavy user stress scenarios before they happen in a production environment.
* Load testing gives excellent protection against poor performance and accommodates complementary strategies for performance management and monitoring of a production environment  
  **Load Testing**
* Response time for each transaction
* Performance of System components under various loads
* Performance of Database components under different loads
* Network delay between the client and the server
* Software design issues
* Server configuration issues like a Web server, application server, database server etc.
* Hardware limitation issues like CPU maximization, memory limitations, network bottleneck, etc.

## Prerequisites of load testing

* Whether the response time is already measured and compared – Quantitative
* Whether the response time is applicable to the business process – Relevant
* Whether the response time is justifiable – Realistic
* Whether the response time is achievable – Achievable
* Whether the response time is measurable using a tool or stopwatch – Measurable

**An environment needs to be set up before starting the load testing:**

| **Hardware Platform** | **Software Configuration** |
| --- | --- |
| * Server Machines * Processors * Memory * Disk Storage * Load Machines configuration * Network configuration | * Operating System * Server Software |

## **Strategies of load Testing**

There are many numbers of ways to perform load testing. Following are a few load testing strategies-

[](https://www.guru99.com/images/L3.png)

* **Manual Load Testing**: This is one of the strategies to execute load testing, but it does not produce repeatable results, cannot provide measurable levels of stress on an application and is an impossible process to coordinate.
* **In house developed load testing tools**: An organization, which realizes the importance of load testing, may build their own tools to execute load tests.
* **Open source load testing tools**: There are several load testing tools available as open source that are free of charge. They may not be as sophisticated as their paid counterparts, but if you are on a budget, they are the best choice.
* **Enterprise-class load testing tools**: They usually come with capture/playback facility. They support a large number of protocols. They can simulate an exceptionally large number of users.

**How to do Load Testing**

### Load Testing Process

1. **Set Up Test Environment**
   * Create an environment that mirrors production.
2. **Identify Test Scenarios**
   * Determine key user journeys and transactions.
3. **Prepare Test Data**
   * Generate a large pool of unique data for different scenarios.
4. **Define Load Conditions**
   * Specify the number of users and load patterns (steady, spike, etc.).
5. **Select Tools**
   * Choose load testing tools (e.g., JMeter, LoadRunner).
6. **Execute Load Tests**
   * Run tests while monitoring performance metrics (response time, throughput).
7. **Analyze Results**
   * Review the results to identify bottlenecks and performance issues.
8. **Report Findings**
   * Prepare a report summarizing the test results and recommendations.
9. **Optimize and Retest**
   * Make necessary optimizations based on findings and repeat testing.

**Guidelines for load testing**

1. Load testing should be planned once the application becomes functionally stable.
2. A large number of unique data should be ready in the data pool
3. Number of users should be decided for each scenario or scripts
4. Avoid creation of detailed logs to conserve the disk IO space
5. Try to avoid downloading of images in the site

In the process of executing load testing test cases, the consistency of response time over the elapsed period should be logged and the same should be compared with various test runs.  
  
**Stress Testing:** The goal of Stress testing is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn’t crash under crunch situations.   
A most prominent use **of stress testing is to determine the limit, at which the system or software or hardware breaks**. It also checks whether the system demonstrates effective error management under extreme conditions.  
Stress testing is also extremely valuable for the following reasons:

* To check whether the system works under abnormal conditions.
* Displaying appropriate error message when the system is under stress.
* System failure under extreme conditions could result in enormous revenue loss
* It is better to be prepared for extreme conditions by executing Stress Testing.  
  **Goals of Stress Testing**
* The goal of stress testing is to analyze the behavior of the system after a failure. For stress testing to be successful, a system should display an appropriate error message while it is under extreme conditions.
* To conduct Stress Testing, sometimes, massive data sets may be used which may get lost during Stress Testing. Testers should not lose this security-related data while doing stress testing.
* The main purpose of stress testing is to make sure that the system recovers after failure which is called as **recoverability**.

**Aspect**

**Functional Testing**

**Load Testing**

**Predictability of**

**Results**

Predictable results based on defined

test cases.

Unpredictable results due to varying

system behavior.

**Variation in Results**

Slight variations in results.

Drastic variations based on load

conditions.

**Frequency of**

**Execution**

Run frequently during development.

Run less frequently, typically before

releases.

**Dependence on**

**Data**

Dependent on specific test data.

Dependent on the number of concurrent

users.

**Objective**

Validates that the application functions

correctly.

Assesses performance under load.

**Focus**

Focuses on individual features and

functions.

Focuses on overall system performance

under stress.

**Tools Used**

Tools like Selenium and QTP.

Tools like JMeter and LoadRunner.

**Success Criteria**

Pass/Fail based on functionality.

Performance metrics meet thresholds.

**Types of Tests**

Includes unit, integration, and

acceptance tests.

Includes stress, spike, and endurance tests.

**Aspect**

**Load Testing**

**Stress Testing**

**Objective**

To determine how a system performs under

expected load conditions.

To evaluate how a system behaves under

extreme or unexpected load conditions.

**Purpose**

Assess system performance, responsiveness,

and stability with a specified number of users.

Identify the breaking point of the system

and its recovery capabilities after failure.

**Load Level**

Tests with normal and peak load conditions.

Tests with beyond normal load conditions,

often until failure.

**Typical**

**Scenarios**

Simulating expected user traffic during regular

operation.

Simulating scenarios like sudden spikes in

traffic or sustained high traffic.

**Outcome**

Helps in identifying performance bottlenecks

and ensuring the system can handle expected

user load.

Reveals how the system fails and how it can

recover from such failures.

**Tools Used**

Tools like JMeter, LoadRunner, and Gatling are

commonly used for both.

Similar tools can be used, but scenarios are

specifically designed to push the system

beyond its limits.

**Results**

Focuses on response times, throughput, and

Resource utilization under normal load.

Focuses on stability, error rates, and

recovery time under extreme conditions.

**Throughput**

Definition: The amount of data processed by the system in a given time frame, usually measured in transactions per second (TPS) or bytes per second (Bps).

Importance: Reflects the system's capacity to handle load. Higher throughput indicates better performance.

These **metrics** provide valuable insights into an application's performance under stress. By analyzing response times, throughput, error rates, resource utilization, and other key indicators, organizations can identify weaknesses, optimize performance, and ensure that their applications are resilient in real-world scenarios

# Build JMeter Test Plan

Test plan can be visualized as your JMeter script for running tests. A test plan consists of test elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements.

There should be at least one thread group in every test plan. We can add or remove elements as per our requirement.

**Step 3: Load and save test plan elements.**

* To load elements to JMeter test plan tree, select and right click on any **Tree Element** on which you want to add the loaded element.
* Select "Merge" option.
* Choose the *.jmx*file where you save the elements.
* Elements will be merged into the JMeter test plan tree.

### **Step 4 : Configuring Tree Elements in JMeter**

You can configure elements in your JMeter test plan using the controls on the right side. Here’s how to set up a Thread Group:

1. **Name**: Assign a meaningful name (e.g., "User Login Tests").
2. **Number of Threads (Users)**: Set the number of virtual users (e.g., 3).
3. **Ramp-Up Time**: Specify how long to start all users (e.g., 10 seconds for 3 threads).
4. **Loop Count**: Define how many times to repeat the test (e.g., 5 times).

Thread Group **As the name implies, thread group represents the group of threads JMeter will use during the test. Thread group elements are the beginning points of any test plan.**The Thread Group Control Panel includes:

* Itsname.
* Number of threads (the number of users you are testing).
* Ramp-up time (how much time you want to allow the Thread Group to go from 0 to 3 users).
* Loop count (How many times the test should be looped).
* Scheduler checkbox (The checkbox at the bottom of the Thread Group panel is used to enable/disable extra fields in which you can enter the duration of test, the startup delay, the start and end times of the run).

The "Loop Count" determines how many times a test will be repeated during execution. You can set this in the Thread Group settings, which controls the number of threads (users) and how they behave.

The "Scheduler" checkbox enables additional fields in the Thread Group. When activated, it allows you to specify:

* **Duration of Test**: How long the test should run.
* **Startup Delay**: Time to wait before starting the test.
* **Start and End Times**: Specific times for when the test should begin and end.  
  **Controllers** can be divided into two broad categories:
* Samplers
* Logical Controllers

## **Samplers**

Samplers are the components which allow JMeter to send specific types of requests to a server. It simulates a user's request for a page to the target server.

Samplers are a must to add component to a test plan as only it can let JMeter know what type of request need to go to a server. Requests could be HTTP, HTTP(s), FTP, TCP, SMTP, SOAP etc.

Absolutely! Here’s a straightforward breakdown of \*\*Logic Controllers\*\* in JMeter based on your description:

### What Are Logic Controllers?

\*\*Logic Controllers\*\* are elements in a JMeter Test Plan that help you control the order in which Samplers (requests) and other test elements are executed. They allow you to customize how your tests run, making them more flexible and powerful.

### Key Functions of Logic Controllers:

1. \*\*Alter Order of Processing\*\*:

- Change the sequence in which requests are sent.

- For example, you can group requests to run in a specific order.

2. \*\*Group Requests\*\*:

- Combine multiple requests into a single transaction.

- This helps you measure the performance of a group of requests as one unit.

**Loop Requests**

- Repeat requests a certain number of times.

- Useful for stress testing by simulating multiple users or repeated actions.

Common Types of Logic Controllers:

**If Controller**: Runs child requests based on a condition.

**While Controller** continues executing child requests as long as a condition is true.  
**Loop Controller:** Repeats child requests a defined number of times.

**Transaction Controller** Groups requests together to measure their total execution time.

**Interleave Controller** Executes child requests one at a time in a round-robin fashion.

**Random Controller** Runs child requests in a random order each time.

1. **Throughput Controller** Controls the execution of requests based on specified throughput percentages.  
   **Why Use Logic Controllers?** Using Logic Controllers allows you to create more realistic and complex test scenarios. You can simulate user behavior more accurately, manage how requests are processed, and analyze performance effectively. This customization is crucial for thorough performance testing.

Listeners are components that collect and display the results of your test executions View **Results Tree**: Displays detailed request and response data for each sampler, useful for debugging.

 View **Results in Table**: Presents results in a tabular format, showing essential metrics like response time and status.

 Aggregate **Report**: Summarizes results across multiple requests, providing averages, min/max times, and error percentages.

 Summary **Report**: A simplified version of the aggregate report with key statistics for quick overview.

 Graph **Results**: Visualizes the response time and throughput over the duration of the test in a graph.

 Response **Time Graph**: Plots response times for requests over time, helping identify performance trends.

 Transaction **Controller**: Groups requests and displays results for the entire transaction, showing cumulative metrics.

 Simple **Data Writer**: Saves test results to a file in a simple format for later analysis.

 Backend **Listener**: Sends metrics to an external system for monitoring, like InfluxDB or Graphite.

#  JDBC **Request**: Can be used as a listener for capturing database response data.