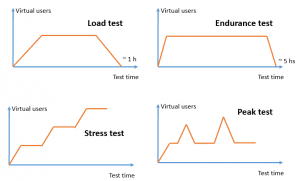
**#1. Objective:**

With Performance testing on **Solo Expenses**, the below questions are to be answered with the results from the tests.

* **Stress test**: What is the maximum number of concurrent users that the system supports with an acceptable user experience? What is the breaking point?
* **Load test**: Given the current system load scenario, how will the application behave? What opportunities for improvement do we see for that expected scenario?
* **Endurance test**: How will the system work after running for some time (say, after a full day)? (In some cases, this involves taking actions like restarting the server every night until finding a definitive solution for some leak present in the system. Sounds terrible, but it could happen!)
* **Peak test**: If my normal system works properly and there is a peak in stress (the casuistry makes it so that many more requests match at the same time than normal, meaning the response times may become worse than acceptable), then how fast does the system recover?



What are the main **bottlenecks**? And then, how do I solve these potential problems or limitations?

**#2) Scope [Web Application]**

**Performance Test Plan: How Many Concurrent Users Do I Run?**

Something that I have repeated many times (so many times that it must be true!): If we design/define our load scenario with X number of users, we can’t start by running a test that simulates that full number of concurrent users. If we do that, as experience tells us, surely several problems will appear all at the same time and we won’t know from where to start — hence, the idea of applying an incremental, iterative methodology for our performance test plan.

It’s iterative because different test iterations are executed and incremental because we start with a reduced number of concurrent users, solving the problems that arise and increasing the concurrency as we go.

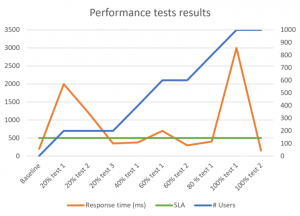
**Example: Load Test**

We’ll consider a load test in our first example where the objective of the test is to analyse if the system supports 1,000 concurrent users.

1. **First test: 1 user without concurrency**. This can serve as the baseline for later comparisons; it can be with 1, 5, 10, or more, but it has to be something extremely lower than what is actually expected in the system).
2. **Second test: 200 concurrent users** (or 20% of the expected load). Here, you can get a lot of information, especially about just how difficult it will be to complete the test on time and in what way.

When executing these initial tests, we will solve the heavier problems and the default configurations (connection pools or Java heap size for example), and we will have an idea of how to scale the system by comparing the response times to the baseline. Once the analysis and troubleshooting are completed, this test is run again and again until acceptable times are obtained.

Depending on how tight those results are, we will decide whether the third test will be 40% (to continue with increments of 20) or if we go with 50% of the load (thinking about going to 75 and 100). On the other hand, if the system responds very well, we may be encouraged to go straight to much more. In any case, what we want to have at the end is a graph that shows us the response times obtained with each test (with each percentage of the expected load), and thus we can see how the system evolved thanks to our work.

[](https://abstracta.us/wp-content/uploads/2017/03/plan-de-pruebas-de-performance-resultados-768x553.png)

In this example graph, we see how different tests were performed increasing the load by 20%. In addition, it is easy to observe that the tests were repeated until reaching the SLA expected in each case, and right after reaching it, the next step was taken.

**Example: Stress Test**

As a second example, imagine that we want to find the breaking point of the system with a stress test. For that, we want to execute different tests with different amounts of users, analyzing if increasing the concurrency continues to increase throughput. If increasing the concurrency does not increase transactions per second, that indicates that we reached the breakpoint since it is saturating the system at a certain point without scaling.

If we start running tests with random numbers of users, it’s going to be like a blindfold over our eyes and we’ll lose a lot of time. What we consider to be the best strategy is to run a test that we could call an exploratory performance test, since we would run it to get an initial idea of where that breakpoint is. For this, we run an incremental test from 0 to X, where X is a large number of users (say 1,000 to consider a single load generator) and we believe that breakage has to be in that range.

What can be done in any load simulation tool is run the test to establish a uniform ramp-up during the test time — that is, if we want that test to last one hour, we set up the test to start with zero concurrent users, and after one hour, it has 1,000. Here, we will be able to have a first approximation to see when the throughput of the system is degraded. If we observe that it is around 650 users, we can begin to refine, running specific tests for this aim.

For example, we could run a test with 500, another with 600, and another with 700. If the test of 700 users actually has less throughput than 600, we’ll have to refine and execute one with 650, and so we continue with the midpoint, to improve accuracy.

**Example: Endurance Test**

For an endurance test, I would say run a constant load that is between 50-70% of the load supported by the system under acceptable conditions. A smaller load could also serve us in this case too, it all depends on how complex it is to prepare the test data in order for it to run for many hours.

Generally, these tests are executed after the stress or load tests have been completed to try to identify other types of problems (memory leaks, hanging connections, etc.). If you have the time, data and what is necessary for this, you could increase the loads that are used for the tests, running them on a prolonged basis.

**Example: Peak Test**

For a peak test, as we said before, the idea is to see how much it taxes the system to recover after a peak occurs. If there is a peak then does the system respond properly or does it get hang? At 10 seconds, does it recover? In two hours? Or what? For this, it is necessary to know the breakpoint of the system, to be able to prepare a test that is below that threshold, and generate a peak by raising the load for a period of one minute, for example, and then lowering it back.

The incremental approach that can be applied here is in the peak itself. You could start by experimenting with small peaks (short duration or low load) and then study how the system reacts to larger peaks. In any case, this is something that should be modeled based on a study of the behaviors of the users, especially based on the access logs that you may have.

Contents of Performance Test Strategy Document

**#3) Test** **Approach:** Here we need to mention about the approach that we are going to follow for our Performance Tests like each script will be executed with a single user to create a baseline and then this baseline tests will be used as a reference for Benchmarking at a later point of time during Test Runs.

Also, each component will be tested individually before integrating them together and so on.

**#4) Test** **Types:** Here we mention the different types of tests to be covered, like Load Test, Stress Test, Endurance Test etc.

**#5) Test** **Deliverables:** Mention what all deliverables will be provided as a part of Performance Testing for the Project like Test Run Report, Executive Summary Report etc.

**#6) Environment:** Here we need to mention the details of the environment. Environment details are very important as it describes what operating systems will be used for Performance Testing.

If the environment will be a replica of production or will it be sized up or sized down from production and also the ratio of sizing up and sizing down i.e will it be half the size of the production or will it be double the size of the production?

Also, we need to clearly mention any Patches or security updates to be considered as a part of the environment set up and also during the Performance Test Run.

**#7) Architecture:** Details of the Application Architecture should be mentioned here, like the total number of Application servers, Web servers, DB servers, Firewalls, 3rdd party application Load generator machines etc.

**#8) Dependencies:** All pre-performance testing actions should be mentioned here, like the components to be performance tested are functionally stable, environment is scaled to a production like one and is available or not, Test date is available or not, Performance Testing tools are available with licenses if any and so on.

**#7) Environment:** We need to mention all the details of the system like IP address, how many servers etc. We should also mention clearly as how the Environment should be set up like the prerequisites, any patches to be updated etc.

**#9) Test Scenarios:**  The list of scenarios to be tested are mentioned in this section.

**#10) Work Load Mix:** The work Load mix plays a vital role in the successful execution of the performance test and if the workload mix doesn’t predict the real-time end-user action, then all the test results go vain and we end up with poor performance in production when the application goes live.

Hence it is necessary to properly design the workload. Understand how the users are accessing the application in production and if the application is already available or else try to get more details from the business team to properly understand the application usage and define the workload.

**#11) Performance Execution cycles:** Details of the number of performance test runs will be described in this section. **For Example,** Base Line test, Cycle 1 50 user test etc.

**#12) Performance Test Metrics:** The details of the metrics collected will be described here, these metrics should be in [acceptance criteria](https://www.softwaretestinghelp.com/what-is-acceptance-testing/) with the agreed performance requirements.

**#13) Tools:** Here we need to mention all the Tools which will be used like Defect Tracking tools, [Management tools](https://www.softwaretestinghelp.com/15-best-test-management-tools-for-software-testers/), Performance Testing, and Monitoring Tools. Some **Examples** of tools for defect tracking is VSTS, for Performance Testing [Jmeter](https://www.softwaretestinghelp.com/jmeter-tutorials/) and for monitoring [Nagios](https://www.nagios.org/).

**#14) Resources:** Details of the Resources required for the Performance Testing Team are documented in this section.

**#15) Entry** **&** **Exit** **Criteria:** Entry and Exit criteria will be described in this section.

**For Example,**

**Entry Criteria** – Application should be functionally stable before deploying the build for Performance Testing.

**Exit Criteria** – All the Priority- 1 defects are closed and most of the SLAs are met.

**#16) Risk and Mitigation:** Any Risks which will affect the Performance Testing must be listed here along with the mitigation plan for the same. This will help any risks from occurring during Performance testing or at least a workaround for the Risk will be planned well in advance. This will help with completing the Performance Test Schedules on time without affecting the deliverables.

**#17) Abbreviations:** Used for Abbreviations. **For Example,** PT – Performance Test.

# 2 Stategy for Load Testing

It provides a basic methodology for testing the scalability and performance of Web applications throughout the life cycle. It outlines the process for selecting the appropriate tools and the recommended steps to perform effective scalability testing. This chapter is broadly divided into the following sections:

* Goals and Requirements of Scalability Testing: What should you aim to accomplish as a result of scalability testing for each phase of your Web application development.
* Methodology: The process and the steps that are required to ensure performance and scalability throughout the application life cycle.
* Test Planning and Execution: How you should plan and execute scalability testing during each phase of development.

## 2.1 Goals of Scalability Testing

The primary goals of a load test are as follows:

1. Determine the user limit for the Web application.
   * The user limit is the maximum number of concurrent users that the system can support while remaining stable and providing reasonable response time to users as they perform a variety of typical business transactions.
   * The user limit should be higher than the required number of concurrent users that the application must support when it is deployed.
2. Determine client-side degradation and end user experience under load.
   * Can users get to the Web application in a timely manner?
   * Are users able to conduct business or perform a transaction within an acceptable time?
   * How does the time of day, number of concurrent users, transactions and usage affect the performance of the Web application?
   * Is the degradation "graceful?" Under heavy loading conditions, does the application behave correctly in "slow motion," or do components crash or send erroneous/incomplete pages to the client?
   * What is the failure rate that users observe? Is it within acceptable limits? Under heavy loading conditions do most users continue to complete their business transactions or do a large number of users receive error messages?
3. Determine server-side robustness and degradation.
   * Does my Web server crash under heavy load?
   * Does my application server crash under heavy load?
   * Do other middle-tier servers crash or slow down under heavy load?
   * Does my database server crash under heavy load?
   * Does my system load require balancing, or if a load balancing system is in place, is it functioning correctly?
   * Can my current architecture be fine-tuned to extract better performance?
   * Should hardware changes be made for improved performance?
   * Are there any resource deadlocks in my system?

## 2.2 Phases of Scalability Testing

The following are the different phases of load and scalability testing for a Web application:

**Performance Benchmarking** - sets and creates the benchmark tests for the initial version of the application for all business transactions and gives the engineering and the quality assurance groups a set of metrics to quantify the scalability of the application. Based on the requirements specified, the development group will either maintain this scalability or improve upon it through the subsequent milestones.

**Performance Regression** - is the phase where the Web application is tested with the established benchmarks to ensure that the changes made to the application do not result in degradation of scalability. These tests are executed when key milestones have been reached or architectural modifications have been made during the development of the application. It is also common that the benchmark tests and the metrics originally set for the application be replaced or augmented with additional tests and newer metrics to reflect the improvements made to the application.

**Acceptance and Scalability Fine Tuning** - is the final load testing phase prior to the official launch of the Web application where all the different pieces of the Web application - including the hardware, load balancing components and all software components - are integrated and the scalability is validated. Different scenarios of real-life usage are emulated and the scalability of the final configuration is validated. These different scenarios are also used to configure the hardware and software components to yield optimal performance.

**24x7 Performance Monitoring** - after the application is deployed, it is essential to monitor the performance of the system under the real load generated by actual users so that crashes or slow-downs can be spotted before they become problematic. In this phase, data pertaining to real life usage can be collected to help refine future scalability tests for accurate emulation of load.

## 2.3 Criteria for Accurate Scalability Testing

In order to emulate a realistic load that will correlate with real-life usage of the application, a load-testing tool must:

* produce load that stresses all tiers of a multi-tier application;
* allow for the simulation of a realistic mix of different groups performing different types of business activities on the site during peak periods;
* emulate page and resource request patterns produced by popular browsers such as Internet Explorer and Netscape;
* validate the responses coming back from the Web server for each of the thousands of concurrent users to ensure that the correct pages are being returned by the Web application under stress;
* allow for easy maintenance of the scripts as the application changes so that scalability can be re-verified each time that the system is changed.

In addition, the following criteria are also important:

* Dynamic Dial-up of users - this capability allows you to add new users to the load test without stopping the current test. For example, if you are running a 100 user load test and you will dynamically add another 100 users, you don't have to stop the load test and restart a new test with 200 users;
* Real-Time Virtual User Debugger - the load testing tool should have some capability to allow you to visually monitor the progress of a user at any given point in time when the load test is in progress;
* Real-Time Graphs that allow you to understand the scalability characteristics of the application as the load test is in progress;
* allow for distributing load tests from a number of machines on the LAN/WAN with a central point of control;
* allow the load tests to be executed with recorded think times, random think times (following some kind of statistical distribution), and with no think times;
* measure response times of entire business transactions in addition to individual objects on pages such as sub-frames and images;
* allow for simulation of different types of caching behaviors;
* run data-driven tests to allow for unique concurrent users on the system;
* allow for complex scheduling to allow for different scenarios of starting, stopping, and ramp-up;
* provide reports and a performance database to allow for post-run analysis and comparison with previously established benchmarks.

## 2.5 Determining the Hardware Needed to Execute the Tests

To execute a scalability test effectively, the appropriate hardware needed to run the test tools must be procured and configured.

In order to generate the load on the Web application using thousands of concurrent users you must consider the following:

**Load Distribution Capability** - Does the load test tool allow for load to be generated from multiple machines and controlled from a central point?

**Operating System** - What operating systems does the load test master and the load generation agents run under?

**Processor** - What type of CPU is required for the master and virtual user agent?

**Memory** - How much memory is required for the master and virtual user agent?

To insure that you have the appropriate hardware to execute scalability tests, ask your load testing tool vendor to provide you with the hardware requirements for the load test master and agent machines.

**General Rules of Thumb:**

* Windows 10- stable operating systems.
* If the CPU utilization of any workstation in the load test - be it the load master or the load agents running concurrent users - is higher than 70-80%, or the memory consumption is over 85%, the processes running on that workstation will experience operating system resource conflicts and the performance results from that test station will be skewed.
* You should consider running the load master on a separate machine if possible. The machine that serves as the master typically needs a high performance CPU. The virtual users can run on one or more machines. These machines need to have sufficient memory to run a large number of virtual users.
* To determine the number of virtual users that can be run on a machine, you can estimate based on the amount of memory each virtual user would consume. If the virtual users are running as threads within a process, on an average they consume 300-500 KB of memory. If the virtual users are running as separate processes within a process, on an average they consume 1024 - 2048 KB of memory.
* A hardware configuration document is usually available from each test vendor that explains the hardware requirements for a particular load testing setup.

## 2.6 Who Should be Responsible for Load Testing?

The following groups of QA and Dev team should have active participation in the load test:

## 2.8 Performing Scalability Testing

The general process for performing scalability testing on a Web application is as follows:

1. Define a process that is repeatable for executing scalability tests throughout the application life-cycle.
2. Define the criteria for scalability.
3. Determine the software tools required to run the load test.
4. Determine and configure the hardware and environment needed to execute the scalability tests.
5. Plan the scalability tests.
6. Plan the test scenarios.
7. Create and verify the scripts.
8. Create and verify the load test scenarios.
9. Execute the tests.
10. Evaluate the results against the defined criteria.
11. Generate required reports.

The details for the above steps are explained in the following sections.

### 2.8.1 Define the Process

Once the requirements for a load testing effort are defined, test planners need to define the process. In defining the process, test planners should consider the following issues and questions:

**Required Applications** - What application(s) will the load testing be performed against?

**Scheduling** - When will the testing be performed? What are the dates, times, build availability, and testing milestones that need to be met?

**Personnel** - Who will perform the analysis, planning, test development, test execution, and evaluation? Which internal department personnel (for example, business analysts, network specialists, quality assurance engineers, and developers) will be involved? Will any third-party personnel (for example, tools vendor, Internet Service Provider, or testing lab) be required?

**Location** - Where will the testing be performed? Will testing be performed internally or at an external location such as at an Internet Service Provider or testing lab?

**Testing Environment**- What SW/HW environment will the load tests be run against? When specifying the testing environment, you should look for and avoid the following common pitfalls:

* **Application stability** - Make sure that the application is not being changed even as the load test is being undertaken. Quite frequently the entire application or parts of it are changed even as the application is being load tested.
* **Deployment environment** - Make sure that the environment under which the application is operating when the load test is being performed is very close to the real deployment environment, if not exactly the same. For example, if your requirement states that the load test has to be performed against an HTTPS server that is configured with enough horsepower to sustain heavy loads, you should not run it against a smaller server that is used by the development group.
* **Acceptance environment** - As part of the acceptance tests that are run prior to shipping the product, you must make sure that the environment used to perform the load test is exactly the same as the live production environment (as defined in a specification document).

**Hardware Allocation** - Is the required hardware (network, master load-test computer, agent computers, etc.) allocated and available for use? Testing vendors should be able to help determine the necessary hardware based on some of the following information:

* number of virtual users or the desired throughput for the application as a whole (Transactions per second);
* maximum or acceptable duration for each business transaction;
* maximum or acceptable duration for delay between business transactions.

### 2.8.2 Define the Criteria

Before you can begin planning for load testing, you need to define the criteria that indicate whether or not the application will be accepted and ready for live deployment. When defining the criteria, you should specify the following:

**Load to be Simulated** - What number of virtual users need to be emulated? This indicates the number of concurrent users on the Web server.

**Number of Business Transactions to Simulate** - How many business transactions are to be simulated for the load test? This is determined by the analysis of the application during requirements planning and may be specified as transactions-per-second (TPS), transactions-per-hour, or simultaneous user sessions.

**Types of Business Transactions to be Simulated** - What are the business transactions that need to be simulated (for example, read an account balance, make an account transaction, check account details, check contributions, etc.)?

**Criteria for Each Business Transactions** - For each business transaction you should determine the following:

* **Acceptable response time under various loads** - What is an acceptable response time under various conditions of load. For example, what is the acceptable response time when running 100 virtual users? 200 virtual users? Also, what is the acceptable response time when running the maximum limit of virtual users.
* **Acceptable failure rate** - What is the acceptable failure rate for all of the transactions and for each business transactions when under load? For example, zero failures allowed for up to 100 virtual users, 5% failures for 200 virtual users, etc.
* **Categories of users** - What are the categories of users to simulate in the various transactions? Are they first time users or are they repeat users? First time users have a higher overhead on the Web server since all the images must be downloaded. You can design and develop tests for both types of users and run combinations of load tests under differing conditions.
* **SSL****and HTTP** - Does testing require a combination of SSL and plain HTTP, only SSL, or only HTTP?
* **Browsers to simulate** - What browsers will be simulated in the load test? Will testing simulate Internet Explorer or Netscape (or both)?
* **Pacing mode** - What is the virtual user pacing that will be used for the load test? Will the testing be performed using recorded "think times" (that is, running with delays between pages that correspond to the same natural pauses that occurred while recording the script)? Should you try a worst-case stress test with no delays between pages? Or alternatively, should you use a random distribution of delays representing a range of user-speeds from expert users on T3 connections to novice users on slow modems?
* **Delay Between Business Transaction Runs** - What delay time will be included between business transaction tests, if any?
* **With or without images** - Will virtual users run with images or without images? Images constitute an additional load on the Web server. In many cases, you may want to perform load testing both with and without images for comparison.

**Overall Transactions-Per-Second Throughput****Required** - What is the overall transactions per second (TPS) throughput required for the load test? This can be computed based on the number of simultaneous business transactions and the duration of typical transactions.

**Type of Error Handling** - What type of error handling is required when executing the load test? Does the load test need to be stopped on encountering certain types of error or just log the error and continue? What types of error logging do we need to enable for each concurrent user and for the different components in the application architecture?

**Type of Transaction and Performance****Data Logging** - What type of transaction and performance data needs to be logged for the various scripts?

### 2.8.3 Planning the Scalability Tests

Developing detailed test plans before you actually create the tests is an important step in making sure the tests conform to the business analysis of the application and the defined criteria.

For each test that will perform a business transaction you need to plan and define the following information:

**Steps for Scripts** - Each script should have a detailed sequence of steps that define the exact actions a user would perform. Multiple scripts can be used. For example, you can define a specific script that performs user login, several scripts that perform specific business transactions, and another script that logs users off. For each script, you should define the expected results. Oracle OpenScript lets you quickly and easily record scripts that emulate a user's actions.

**Run-Time Data** - The test plan should specify any run-time data that is needed to interact with the application, for example, login user IDs, passwords, and other run-time data specific to the application.

**Data Driven Tests** - If the scripts require varying data at run-time, you'll need to have an understanding of all the fields that require this data. You also need to define the data sources and any tool(s) needed to either create fictitious data or extract real data from existing databases.

### 2.8.4 Planning the Load Test Scenarios

In addition to the business transaction details for each script, the test plan should also specify the different user groups and test scenarios that will be required for load testing. For each test scenario you need to plan and define the following information:

**Type of****User** - Is this user a first-time user of the application or a repeat user? This is important if the application responds differently for a first-time user than it does for a repeat user and places more stress on the server.

**Transactions to Perform** - Which business transaction(s) will this user perform? In what sequence? If the application requires a first-time user to perform some type of registration, then the user profile for first-time users should include a registration script.

**Number of****Users** - How many virtual users with this user profile will run over the same time interval?

**Which System** - Which specific computer(s) will be used to generate the load for this user group? Oracle Load Testing can run virtual users on a single system or on multiple, distributed systems running.

**Which Browser** - Which browser will this user group emulate?

**Pacing mode** - What pacing mode will be used for the user group? Will the testing be performed using recorded think times, a range of times, or as fast as possible?

**Delay Between Business Transaction Runs** - What delay time will be included between business transactions, if any?

**With or Without Images** - Will the user group run with images or without images? You may want to create different user groups that perform load testing both with and without images for comparison.

### 2.8.5 Create and Verify the Test Scripts

After planning the scripts, you will use Oracle OpenScript to create and verify each script.

**Create the Scripts** - This process is defined by Oracle OpenScript (recording user actions) and the individual test plans for each script. When creating the script, you specify the following information as defined in the test plan:

* user actions to perform
* timers
* tests to perform
* data sources

**Verify the Scripts** - Once each script is created, you should verify that the script performs as expected and produces the desired result. Each script should be verified independent of any other scripts and in a controlled manner to simplify script debugging.

### 2.8.6 Create and Verify the Load Test Scenarios

Once the individual scripts have been created and verified, you can create and verify the load test scenarios. It will save you a lot of time and aggravation if you perform a number of simple verification steps before your full-blown load test.

**Verify scripts with Multiple Virtual****Users** - Before combining multiple scripts into a single load test scenario, you should verify that you can successfully run a single script as multiple virtual users. Each script should perform as expected as defined by the criteria for the application.

Oracle Load Testing Autopilot lets you run multiple scenarios with different virtual user characteristics.

**Verify distributed test execution****on multiple machines** - You should verify the load test tool's ability to execute the individual scripts properly in a distributed environment if you plan to use multiple CPU's for load generation. This usually involves a master system controlling the virtual user execution on multiple workstations on the network. This can help you isolate any installation or networking-related issues.

**Verify real-life scenarios that include one of each user group** - Before executing the full load test you should create and verify a scenario that includes one virtual user of each user group you wish to run at the same time. That is, before you run a test with 20 VU's of group A, and 20 VU's of group B, and 60 VU's of group C, you should first run one VU of group A, one VU of group B, and one VU of group C, and make sure that the results are as expected.

**Create real-life scenarios** - This process should be defined in the test plans for each scenario. When creating the individual scenarios, you specify the following information as defined in the test plan:

* type of user
* pacing mode
* navigation/transactions to perform
* delay between transaction runs
* number of users of each type
* with or without images
* system used for load generation
* error log settings
* browser emulation

### 2.8.7 Execute the Tests

Once you have created and verified the basic load test scenarios above, you can begin to run the load test scenarios with many virtual users, and expect that the test results will be valid.

**Run basic tests to ensure scaling** - run tests with a minimal amount of virtual users to ensure that the system scales up correctly.

* **Run individual business transactions** - Run each of the different business transactions starting with 10 virtual users scaling up to 25 - 50 virtual users.
* **Run combinations of business transactions** - Run a combination of different business transaction scripts starting with 5 virtual users scaling up to 25 virtual users.

If the above two scenarios execute without any problems, the next step is to execute the full load test with the full number of virtual users of each user-group type.

**Run the real-life scenarios** - Run each of the real scenarios as outlined in the previous steps:

* Increase the scenarios up to the required number of simultaneous virtual users;
* Monitor for any errors in the system.

**Re-Run these scenarios with a real user** - While the load test is running, a real person should access the system through a standard browser and report performance observations:

* Observe the degradation times for a real user;
* Observe any errors if they are being reported back in the browser.

### 2.8.8 Evaluate the Results

For each of the load test scenarios, examine the following performance data and validate the results against the expected criteria:

* Response times for groups of users at the different numbers of virtual users;
* System throughput at various numbers of virtual users;
* Any errors that may have occurred.

View Run graph options let you evaluate performance in real-time.

Save the erroneous HTML when problems occur to help the development group debug the errors.

### 2.8.9 Generate Analysis Reports

Document the performance by generating the various reports that may be required for acceptance and deployment of the application. The following are some examples of the types of reports that can be generated from a load test:

* Performance vs. Time
* Statistics vs. Time
* Users vs. Time
* Errors vs. Users
* Statistics vs. Users
* Errors vs. Time
* Any other error reports that may be required for the development group to debug and fix any problems that may have occurred.

Oracle Load Testing graphs in the Create Reports tab let you view performance and error data from the load test in multiple formats.

## 2.9 Summary

Load testing throughout the development cycle has become an essential part of the process of designing scalable, reliable Web applications. Developers and QA professionals now rely on load testing tools as a means to validate system architectures, tune applications for maximum performance, and assess the impact of hardware upgrades. Consequently, it is critical that the load test results can be used with confidence as the basis for key decisions about application readiness and potential changes to the system's hardware and software. Using the methodology embodied in this guide along with accurate load testing tools such as Oracle Load Testing, you now have a systematic approach to ensure the performance of your Web applications. With load testing established as a routine part of the application lifecycle you can be sure to avoid costly "scalability surprises" when your application goes live for the first time or after any subsequent release.

# QA : Performance Test Plan using JMeter

This document helps anyone who needs to create a performance test plan in Jmeter for the API’s. It describes the details of various performance testing types and how to add a thread group for each type. Detailed plugin information such as how to generate reports , monitoring and installation details are also available.

1. [JMeter Demo Service Test Plan](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#test-plan-in-jmeter)
2. [Test plan in JMeter](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#test-plan-in-jmeter)
3. [Thread Group](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#thread-group)
4. [JMeter Plugins Manager](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#jmeter-plugins-manager)
5. [Backend Listener](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#backend-listener)
6. [Dashboard Report](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#dashboard-report)
7. [Performance Plugin](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#performance-plugin)
8. [Email Notification in Jenkins](https://github.com/department-of-veterans-affairs/ascent-sample/wiki/QA-:-Performance-Test-Plan-using-JMeter#email-notification-in-jenkins)

### ****JMeter Demo Service Test Plan****

Created a test plan in JMeter with Ascent demo service end points. This document provides the details of the thread group, config element, sampler, assertions, and listener used in the test plan. Also, provides the details of external plugins used in the test plan.

### ****Test plan in JMeter****

A Test Plan can be viewed as a container for running tests. It defines what to test and how to go about it. A complete test plan consists of one or more elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements. A test plan must have at least one thread group.

Created a test plan “Demo Services” with number of thread groups. There are different thread group available in JMeter.

### ****Thread Group:****

Thread Group elements are the beginning points of test plan. The thread group elements control the number of threads JMeter will use during the test. We can also control the following with the Thread Group −

* Setting the number of threads
* Setting the ramp-up time
* Setting the number of test iterations

We created few performance test scenarios using the available thread group for the demo services.

**Load Testing:**

Load testing is performed to determine a system’s behavior under both normal and at peak conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. For e.g. If the number of users are increased then how much CPU, memory will be consumed, what is the network and bandwidth response time.

We have created a thread group for load testing using “Thread Group”.

**Stress Testing:**

Stress testing evaluates the behavior of the application beyond peak load and normal conditions. The goal of the stress testing is to analyze post-crash reports to define the behavior of application after failure. In a successful stress testing, the system will come back to normality along with all its components even after the most terrible breakdown.

We have created a thread group for stress testing using “stepping thread group” and “Concurrency Thread”.

**Spike Testing:**

Spike Testing is a form of testing process in which an application is tested with unusual increment and decrements in the load. The system is unexpectedly loaded and unloaded. It is done to notice how actually the system reacts with unexpected rise and decline of users.

We have created a thread group for spike testing using “Ultimate thread group”.

**Endurance Testing:**

In Endurance testing a system is tested with a load extended over a significant amount of time to analyze the behavior of the system under sustained use. This type of testing is performed at the last stage of performance run cycle. It ensures that the application is capable enough to handle the extended load without any deterioration of response time.

Primary goal of Endurance testing is to check for memory leaks, discover how the system performs under sustained usage, to ensure that after a long period, the system response time will remain the same or better than the start of the test.

We have created a thread group for endurance testing using “thread group”.

**Other Thread Group:**

There are other thread group available in JMeter. Based on the scenario we can select the thread group.

### ****JMeter Plugins Manager:****

Instead of installing different plugins manually, the plugin manager will do it in UI. This plugin manages to inuclude usual plugins from JMeter-Plugins.org, various third-party plugins, and even core JMeter plugins.

Following plugins are used in Ascent demo service:

Few plugins are used in the POC work. Currently, we are not using these plugins Servers Performance Monitoring, Graphs Generator and Backend Listener.

**Servers Performance Monitoring:**

This plugin helps to check the health of the servers loaded. Using this plugin, we can monitor the CPU, Memory, Swap, Disks I/O and Networks I/O.

More information about this plugin [Performance Monitoring](https://jmeter-plugins.org/wiki/PerfMon/?utm_source=jmeter&utm_medium=helplink&utm_campaign=PerfMon)

**Graphs Generator:**

This plugin creates a different report at the end of test execution. It generates the following report.

* Active Threads Over Time
* Response Times Over Time
* Transactions per Second
* Server Hits per Seconds
* Response Codes per Second
* Response Latencies Over Time
* Bytes Throughput Over Time
* Response Times vs Threads
* Transaction Throughput vs Threads
* Response Times Distribution
* Response Times Percentiles

More information about this plugin [Graph Generator](https://jmeter-plugins.org/wiki/GraphsGeneratorListener/)

### ****Backend Listener:****

Backend listener create real time graph of JMeter test. Real time result sent to a backend thru backend listener using any backend service by providing a class which implements abstract backend listener client. JMeter ships with a GraphiteBackendListenerClient which allows you to send metrics to a Graphite Backend.

This feature provides:

* Live results
* Nice graphs for metrics
* Ability to compare 2 or more load tests
* Storing monitoring data as long as JMeter results in the same backend

JMeter has a listener to send the results to a time-series database (influxdb/graphite) while the test is running. By configuring Grafana (an open source metrics dashboard) to connect to the influxdb/graphite, we can create nice graphs while the JMeter is running the test.

**Influxdb**

InfluxDB is an open-source, distributed, time-series database that allows to easily store metrics.

**Influxdb setup:**

* Download InfluxDB.
* Check this link for InfluxDB setup [InfluxDB Download](https://docs.influxdata.com/influxdb/v0.9/introduction/getting_started/)
* Refer to this page for more information regarding the setup process for InfluxDB [InfluxDB Setup](http://www.testautomationguru.com/jmeter-real-time-results-influxdb-grafana/)

**Grafana**

Grafana is an open-source platform for time series analytics, which allows to create real-time graphs based on time series data. Grafana is a pretty common solution for overall application monitoring.  Grafana allows us to customize dashboards in the way we want.

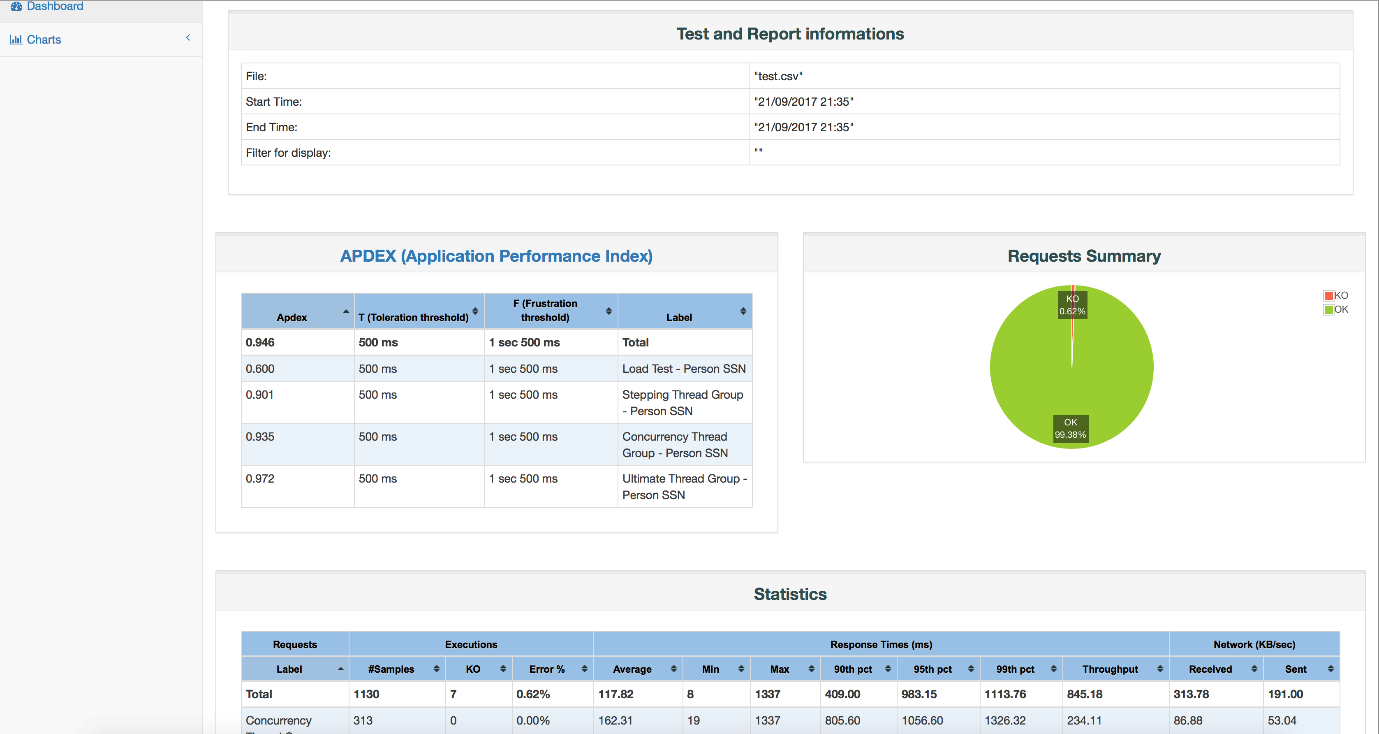
**Grafana Setup**

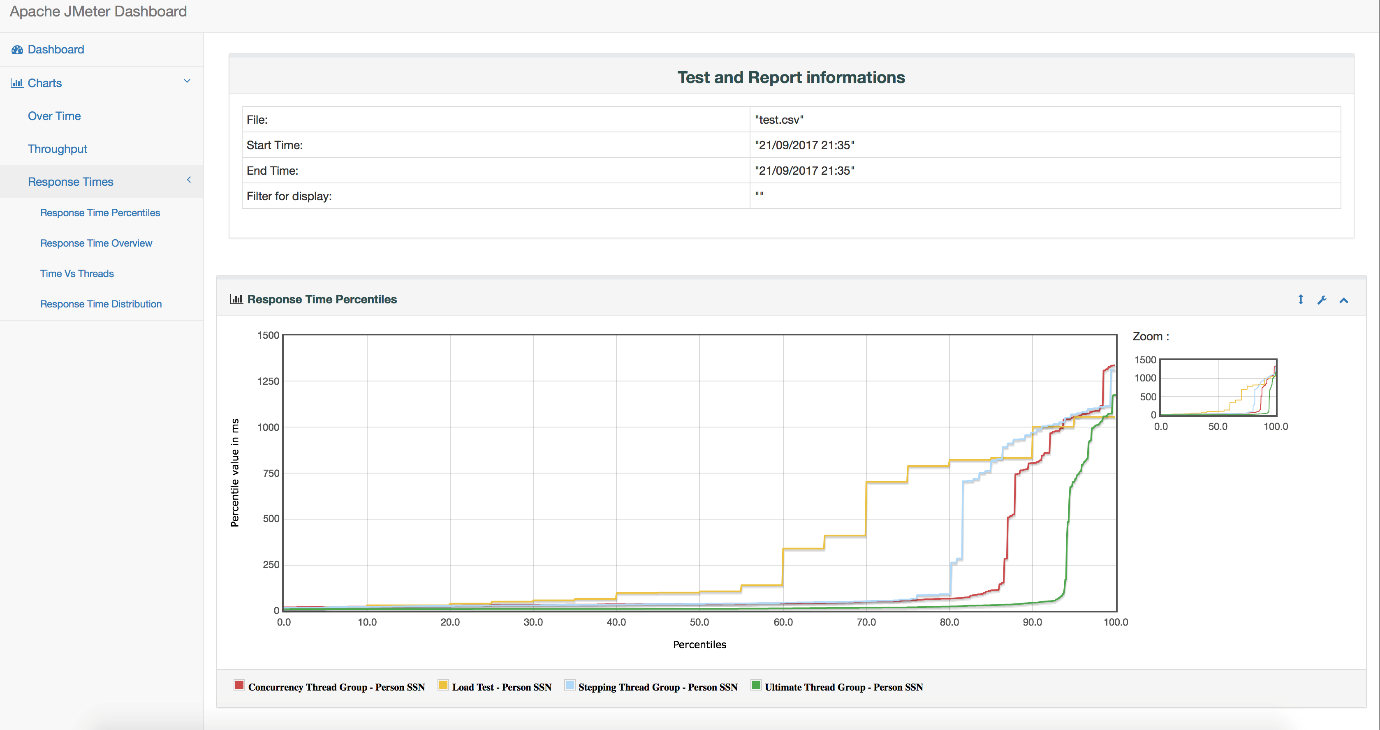
* [Download Grafana](http://docs.grafana.org/installation/)
* Grafana should be available on [http://localhost:3000](http://localhost:3000/). Use ‘admin’ as a default username and password to log in.
* Refer to this page for grafana and Influxdb integration [InfluxDB and Grafana](http://www.testautomationguru.com/jmeter-real-time-results-influxdb-grafana/)

### ****Dashboard Report:****

JMeter supports dashboard report generation to get graphs and statistics from a test plan. The dashboard generator is a modular extension of JMeter. Its default behavior is to read and process samples from CSV files to generate HTML files containing graph views. It can generate the report at end of a load test or on demand.

[Generating Report Dashboard](http://jmeter.apache.org/usermanual/generating-dashboard.html)



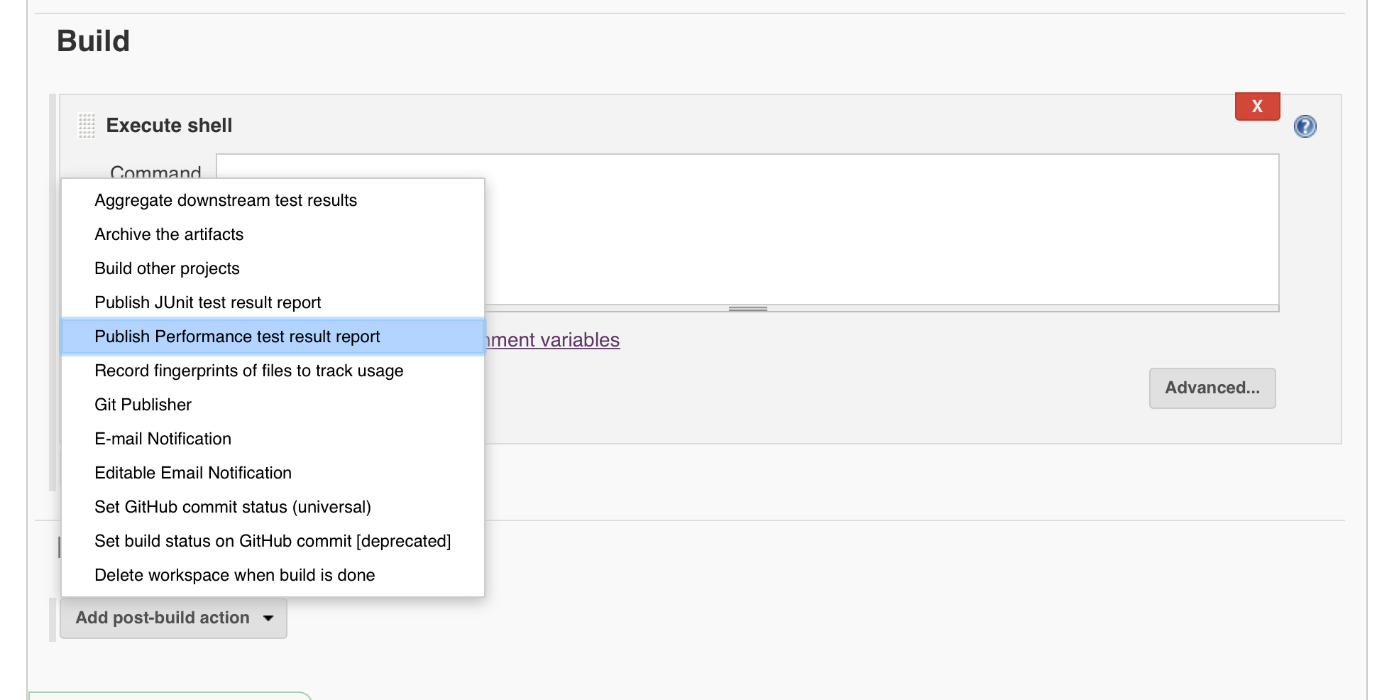


### ****Performance Plugin:****

This plugin allows to capture reports from JMeter tool. Jenkins will generate graphic charts with the trend report of performance and robustness. It includes the feature of setting the final build status as good, unstable or failed, based on the reported error percentage. Jenkins has a plugin for JMeter to parse the result files, create an aggregate report, create charts and to compare the current result with previous results.

**Steps for installing performance plugin:**

1. Download it from here [Performance Plugin](https://wiki.jenkins.io/display/JENKINS/Performance+Plugin)
2. Copy the performance.hpi file to the plugins folder of your Jenkins installation. If you run Jenkins from the .war file, copy the plugin to the. jenkins/plugins path under your home folder.
3. Restart Jenkins to detect and load the plugin.
4. If the installation is correct it will display the Publish Performance Test Result Report option under Jenkins -> Your Project -> Configure -> Add post-build action drop-down.



**Steps to run JMeter test in Jenkins:**

1. Add the jmeter.save.saveservice.output\_format=xml a line to the user.properties file in the /bin folder of your JMeter installation.
2. Create a new item in Jenkins and select freestyle project.
3. Enter all the details in the General and source code management tab.
4. Pass the JMeter command in the Execute shell command text box.
5. In post-build actions select publish performance test result report.

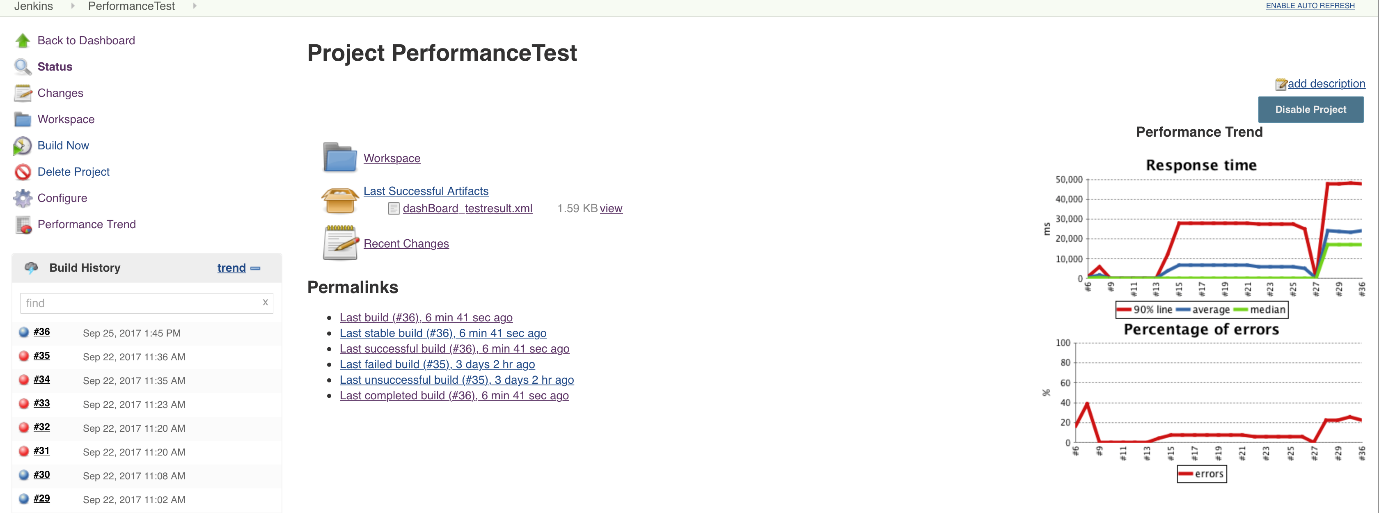
To verify the execution was successful click on the console output tab of the project.

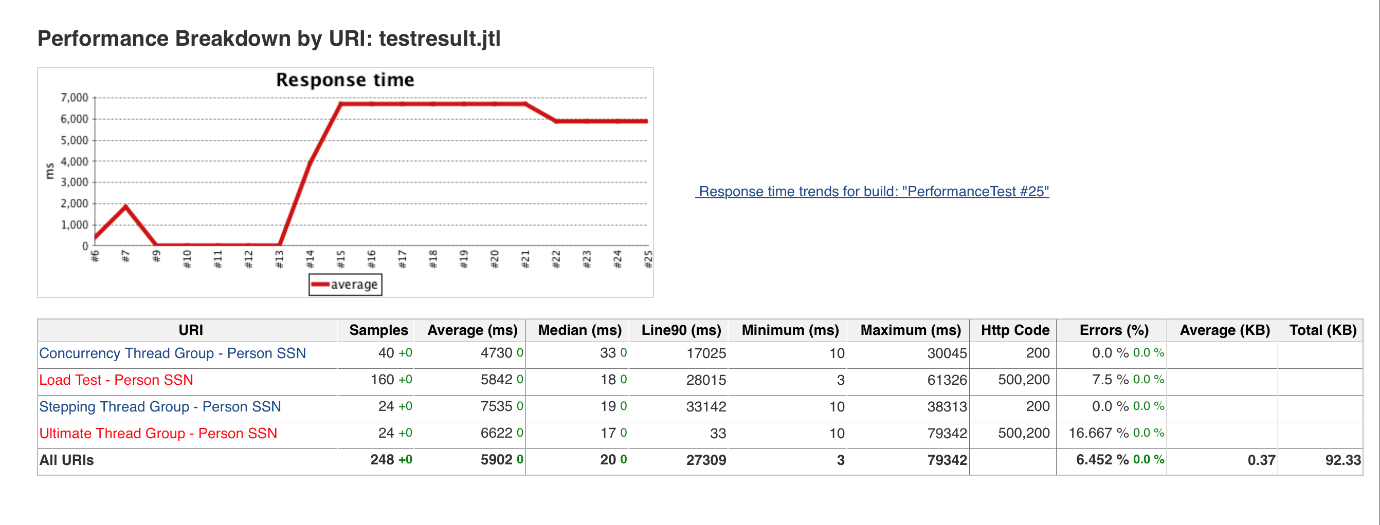
**View the result in JMeter:** The Performance Plugin displays statistics, trends and can be used to mark builds as failed based on results.

1. Select the project and click on the performance trend report.
2. It displays the result.

After the first build with the plugin, you will see empty charts "Performance Trend". These charts change as more builds are created. It will display Median, average and values with 90 percentile along with the errors count. To get more details on each sampler that is executed, click the charts.

[Setting up performance plugin with Jenkins](http://jenkinsci.github.io/performance-plugin/RunTests.html)





### ****Email Notification in Jenkins:****

Jenkins has a nice plugin for Emailing the results. Please check this [Link](https://wiki.jenkins.io/display/JENKINS/Email-ext+plugin) for more details.

Steps to configure Email notification in Jenkins:

1. Open Jenkins on any browser.
2. Click the ‘Manage Jenkins’ menu option displayed at the right side of the screen. You will be redirected to the ‘Manage Jenkins’ page, where you need to select the ‘Manage Plugin’ option. Click the ‘Available’ tab present at the top of the ‘Manage Plugin’ page.
3. Start typing ‘Notification’ in the ‘Filter’ field displayed at the top-right side of the ‘Manage Plugin’ page. Click the checkbox next to the ‘Email-ext plugin’ option. Click the ‘Install without restart’ button.
4. Now, click the checkbox next to the ‘Email-ext Template Plugin’ option. Click the ‘Install without restart’ button.
5. Go to the Jenkins home page and click the ‘Manage Jenkins’ menu option. Then, select the ‘Configure System’ option.
6. Enter the SMTP server name under ‘Email Notification’. Click the ‘Advanced’ button and then click the checkbox next to the ‘Use SMTP Authentication’ option. Now, set the following fields.

* SMTP server name : Server name
* User name: User Name
* Password: Password
* Use SSL : Checked
* SMTP Port: Port No

1. Check the email notification functionality by clicking the checkbox next to the ‘Test configuration by sending Test e-mail recipient’ option. Enter a valid email id and click the ‘Test configuration’ button to check whether the email id is valid or not.
2. Go to the home page and click on a created job. Then, click the ‘Configure’ option.
3. Click the ‘Add post-build action’ drop-down.
4. Select the ‘E-mail Notification’ value.
5. Enter the recipient email id in the ‘E-mail Notification’ box and select the checkbox next to the ‘Send e-mail for every unstable build’ option.
6. Click the ‘Add post-build action’ drop-down and select the ‘Editable Email Notification’ value.
7. Fill the ‘Editable Email Notification’ fields. Project Recipient List: [email\_id@mail.com](mailto:email_id@mail.com)
8. Click the ‘Advance Settings…’ button in the ‘Editable Email Notification’ box.
9. Click the ‘Add Trigger’ drop-down and select the ‘Always’ option.
10. Click the ‘Save’ button.
11. Go to the home page and click on the job.
12. Click the ‘Build now’ link and check the email id after the job execution.