

|  |
| --- |
| PERFORMANCE TESTING STRATEGY |
| Folio Library Services Platform |

Version: **0.1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 0.1 | draft version | viacheslav\_vostrietsov@epam.com | 21-Nov-2019 |  |  |

Contents

[1. INTRODUCTION 5](#_Toc25587418)

[1.1 Description 5](#_Toc25587419)

[1.2 Purpose 6](#_Toc25587420)

[1.3 Scope 6](#_Toc25587421)

[2. PERFORMANCE ACCEPTANCE CRITERIA 6](#_Toc25587422)

[2.1 Introduction 6](#_Toc25587423)

[2.2 Identifying the Performance Testing Objectives 7](#_Toc25587424)

[2.3 Defining Performance Pass/Fail Criteria 8](#_Toc25587425)

[3. LOAD MODEL 8](#_Toc25587426)

[3.1 Introduction 8](#_Toc25587427)

[3.2 Workload Distribution 8](#_Toc25587428)

[4. WORKFLOW 10](#_Toc25587429)

[4.1 Introduction 10](#_Toc25587430)

[4.2 Iterative Testing Performance Activities 10](#_Toc25587431)

[5. PERFORMANCE TEST TYPES (WITH PRIORITIES) 11](#_Toc25587432)

[5.1 Server-side tests 11](#_Toc25587433)

[5.1.1 Pipe-clean (Baseline) test 11](#_Toc25587434)

[5.1.2 Capacity (Ramp up) test 11](#_Toc25587435)

[5.1.3 Fixed load (Benchmark) test 12](#_Toc25587436)

[5.1.4 Longevity (Endurance) tests 13](#_Toc25587437)

[5.1.5 Volume tests 13](#_Toc25587438)

[6. TOOL SELECTION APPROACH 13](#_Toc25587439)

[6.1 Performance Testing Tool Architecture 13](#_Toc25587440)

[6.1.1 Scripting module 13](#_Toc25587441)

[6.1.2 Load generator(s) 14](#_Toc25587442)

[6.1.3 Analysis module 14](#_Toc25587443)

[6.1.4 Optional modules 14](#_Toc25587444)

[6.2 Performance testing environment in continuous integration 14](#_Toc25587445)

[6.2.1 Proposed Solution 14](#_Toc25587446)

[6.2.2 Proposed Tools 15](#_Toc25587447)

[7. METRICS 17](#_Toc25587448)

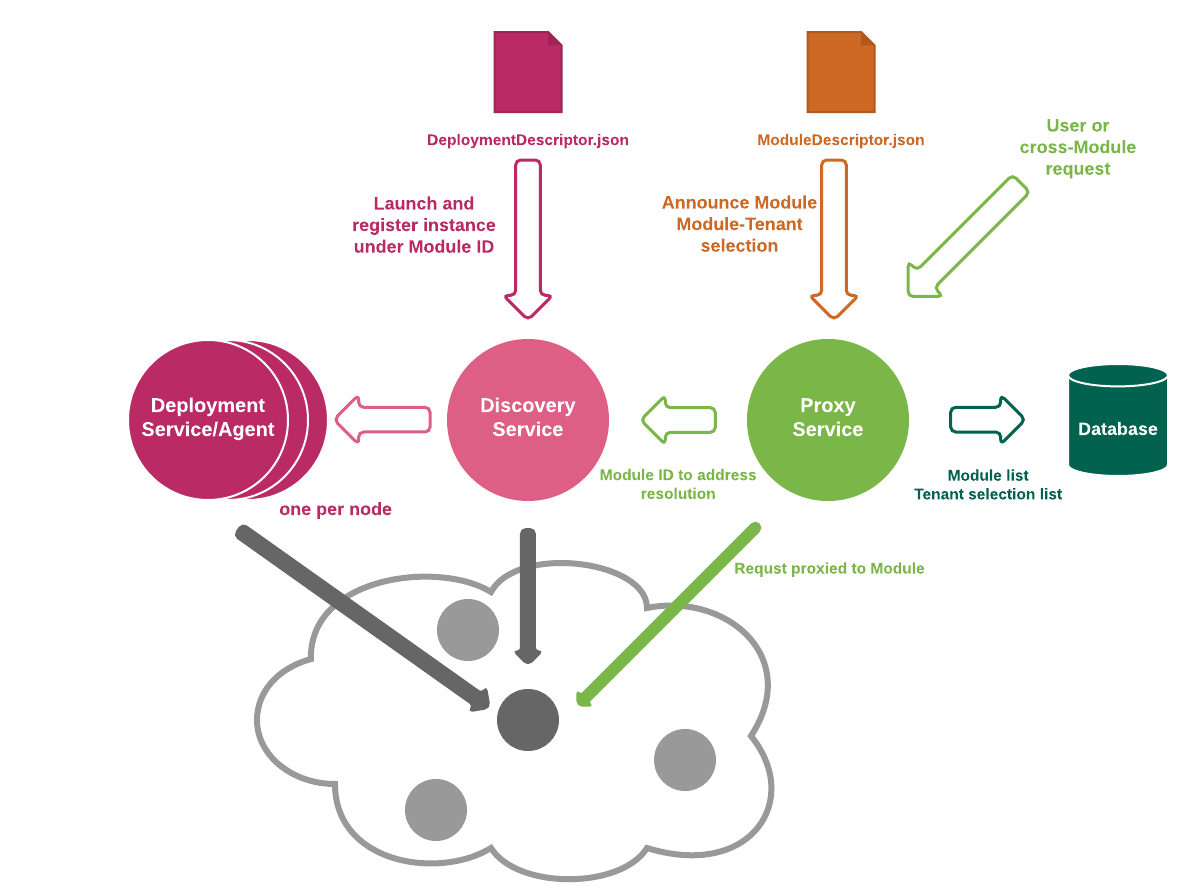
[8. TEAM LINE-UP 18](#_Toc25587449)

# 1. INTRODUCTION

## 1.1 Description

This document defines testing approach for FOLIO Library Services Platform. It briefly describes

methods and tools to be used to evaluate and tune performance of the platform.



## 1.2 Purpose

The purpose of this document is to outline the approach for Performance Optimization team to follow to assure Performance Acceptance Criteria to be defined and met. Specifically, this document details:

* Performance Acceptance Criteria;
* Performance Testing workflow;
* Load Model;
* Test types to be performed;
* Tools and infrastructure.

## 1.3 Scope

This document provides strategy to carry out all performance analysis activities for the platform. It briefly describes resources required, including toolset to accomplish test executions, results analysis, and performance tuning. It covers Performance Acceptance Criteria, explains system interaction models to be tested, and describes scripts to be developed. This strategy doesn’t include functional testing, nor does it guarantee any specific performance results. The approach, defined in this document, may be used for both - scheduled and interim releases, however, interim releases may not allow for complete adherence to this test approach. The approach used for interim releases will be dependent upon both the criticality of the release and the completeness of the functionality included in the release. The primary objectives for this testing effort are to:

* Validate that the Performance Acceptance Criteria are met by the system AND/OR;
* Identify and ensure that performance related defects are addressed prior to deployment.

# 2. PERFORMANCE ACCEPTANCE CRITERIA

## 2.1 Introduction

Acceptance criteria provide the essentials to be satisfied to accept the platform by the end user. Performance efforts associated with two sets of acceptance criteria:

* requirements and objectives;
* engagement criteria;

In the sections below, both types are explained in general and in specific detail for the application performance optimization effort. The efforts will be deemed complete when either all performance criteria are met, or any one of the engagement completion criteria is met.

## 2.2 Identifying the Performance Testing Objectives

The primary goal is to test server-side performance of the FOLIO Library Services Platform.

The project goals are:

* implement performance analysis methodology that could be applied as a standard of Folio performance testing
* deploy and configure performance testing and analysis framework based on Carrier accelerator
* establish instrumentation that allow to get as deep as possible in analysis details (incl. code profiling and slow query identification)
* determine server-side robustness and degradation;
* ensure that the application will be able to handle the expected load;
* determine the capacity of the application;
* check for failures or errors that occur in the application due to high load;
* verify whether high volume data affects the speed of processing

The peak usage from production has been derived to determine the future load, however, there is a much higher load is expected so we can set received numbers as a minimum threshold



Figure 2.1 Production usage statistics

Acceptance criteria:

* Response times by 95 percentiles excluding network latency do not exceed 100ms with at least 210 requests per minute (one Checkout transaction consists of 36 requests and Checkin is around 14 requests. There were 4.5 checkouts and 3.4 checkins per minute in a peak which is around 8 transactions per minute);
* Error rate: 0%;

## 2.3 Defining Performance Pass/Fail Criteria

We need to define what constitutes passing the goal and what constitutes falling short of achieving it. The pass/fail criteria should be unambiguous so that they are measurable in absolute terms. Otherwise, the clients may challenge the pass/fail criteria later. The strategy for Folio Library Services Platform is to achieve step by step the goals set.

Step 1: Design workloads based on endpoints.

Step 2: Define Non-Functional Requirements.

Step 3: Define proper measurement criteria.

Step 4: Set the pass/fail criteria for each test run with each workload

# 3. LOAD MODEL

## 3.1 Introduction

Workload is an instrument simulating the real-world environment. It is an integrated part of the test execution process. Once system endpoints are known, a detailed workload can be planned. The workload provides in-depth knowledge of behaviors in proposed system. It explains how typical load will spread once the system goes live. In addition, the workload helps to understand the requirements in a structured way. Because the system performance capability viewed from different perspectives, like meeting the goals set by the users, impact on the architecture, and growth potential of the system and so on.

## 3.2 Workload Distribution

The section below describes the workload distributions to be used for testing the platform. The Workload Distribution percentages presented in workload profile that is based on multiple endpoint

A close up of text on a white background

Description automatically generated

Figure 3.1 Suggested and approved workload distribution model

# 4. WORKFLOW

## 4.1 Introduction

This section describes the concepts underlying the activities necessary to make performance testing successful within an iterative process, as well as specific, actionable items that can be immediately applied for the FOLIO Library Services platform project to gain a significant return on this investment. The key to working within an iteration-based work cycle is team coordination. For this reason, performance analysts must be able to adapt what he or she measures and analyzes per iteration cycle as circumstances change.

## 4.2 Iterative Testing Performance Activities

A picture containing chain, metalware

Description automatically generated

Figure 4.1 Iterative Testing Performance Activities

1. **Prepare testing scenarios**
   * Identify the main business flow of current project phase
   * Identify the vUsers think time to simulate real user usage
   * Prepare Load Model and user Flow diagrams
   * Run baseline test to identify the best-case results
2. **Preparation**
   * Prepare test data (user credentials, search data etc.)
   * Identify the test environment (check environment state, check deployed functionality)
3. **Test**
   * Run capacity test to identify the highest level of load without performance degradation
   * Run responsiveness tests to determine performance degradation on fix load performance tests (compare with previous iterations)
   * Run longevity (endurance) tests
   * Run volume tests with different FOLIO configurations
4. **Analyze**
   * Analyze derived data
   * Compare to previous test run
   * Correlate results
   * Identify the issues
5. **Report preparation**
   * Recommend the improvements

# 5. PERFORMANCE TEST TYPES (WITH PRIORITIES)

## 5.1 Server-side tests

### 5.1.1 Pipe-clean (Baseline) test

Priority: Highest

The pipe-clean test is a preparatory task that serves to validate each performance test script in the performance test environment. The test is normally executed for a single use case as a single virtual user for a set period or for a set number of iterations. This execution should ideally be carried out without any other activity on the system to provide a best-case measurement. You can then use the metrics obtained as a baseline to determine the amount of performance degradation that occurs in response to increasing numbers of users and to determine the server and network footprint for each scripted use case.

#### Purpose:

To check that system is ready for performance testing and scripts are OK

### 5.1.2 Capacity (Ramp up) test

Priority: Highest

Capacity of a system is the highest level of load it can take and handle without:

* Significant response times increase
* Stability decrease

In other words, capacity is the measure of how powerful the system is.

A capacity test complements load testing by determining server’s ultimate failure point. You perform capacity testing in conjunction with capacity planning, which you use to plan for future growth, such as an increased user base or increased volume of data. For example, to accommodate future loads, you need to know how many additional resources (such as processor capacity, memory usage, disk capacity, or network bandwidth) are necessary to support future usage levels. Capacity testing helps to identify a scaling strategy to determine whether system should scale up or scale down.

#### Purpose:

To determine how many users and/or transactions a given system will support and still meet performance goals, to find out server capacity, stability under incremental load and scalability of the system. Also, capacity testing results are key points to create performance tests of another types, for example fixed-load or stress.

### 5.1.3 Fixed load (Benchmark) test

Priority: High

Fixed load testing is conducted to verify that your application can meet your desired performance objectives; these performance objectives are often specified in a service level agreement (SLA). A load test enables you to measure response times, throughput rates, and resource-utilization levels, to verify application behavior under normal and peak load conditions.

Define the load level vs capacity:

* Low-load (~10% of capacity)
* Mid-load (~45% of capacity)
* High-load (~80% of capacity)
* Or some defined level of load (e.g. production-like, expected, etc.)

Duration should be:

* long enough to make results statistically meaningful
* short enough to avoid biased errors

#### Purpose:

* to get response times (and some other metrics) statistics under different levels of load and compare them against target/previous release (build, sprint, etc.) results
* to check that system under load is stable for a particular period
* (optional) system resources/application profiling for problematic transactions

The goal of checking system stability is correct for any performance test with fixed level of load.

### 5.1.4 Longevity (Endurance) tests

Priority: High

This is a long high load test to check if system can work with no issues for a long time. This is the most important stability test because it assumes no extreme cases, but rather a normal operation over time.

Basically, it`s the same as fixed load test, next parameters should be defined:

* Load
* Duration

The tricky part is how to balance load/duration:

* Less load -> more duration is needed: production/expected level of load for a week or so (ideal case)
* More load -> less duration is possible: higher load (e.g. 80% of capacity) for 24-48 hours (a more common case)

#### Purpose:

* to assess system stability during long-time load
* to find memory leaks
* to check if backend background activities affect system performance

### 5.1.5 Volume tests

Priority: High

Volume testing is testing of an application with large number of data in database is possible, it verifies if the system responds as expected for a certain volume of data. It may include increasing size of the file.

***Purpose***:

* Check system performance with increasing volumes of data in the database
* To identify the problem that are likely to occur with large amount of data
* To figure out the point at which the stability of the system degrades
* Volume Testing will help to identify the capacity of the system or application - normal and heavy volume

# 6. TOOL SELECTION APPROACH

## 6.1 Performance Testing Tool Architecture

### 6.1.1 Scripting module

Gives ability to script workloads and usage patterns. The term middleware refers to the primary protocol used by the application to communicate between the client applications and API tier (for web applications this is principally HTTP or HTTPS).

### 6.1.2 Load generator(s)

Generates the load—normally from multiple workstations or servers, depending on the amount of load required. Each load injector generates multiple “virtual” users, simulating a large amount of end-user activity from a relatively small number of physical or virtual machines. The application client memory and CPU footprint can have a significant impact on the number of virtual users that can run within a given injector platform, affecting the number of injectors required.

### 6.1.3 Analysis module

Provides the ability to analyze the data collected from each test execution. This data is typically a mixture of automatically generated reports and configurable graphical or tabular presentation. There may also be an expert capability that provides automated analysis of results and highlights areas of concern.

### 6.1.4 Optional modules

Complements the components to monitor server and network performance while a load test is running or allow integration with another vendor’s software.

## 6.2 Performance testing environment in continuous integration

### 6.2.1 Proposed Solution

The following architecture at concept is very flexible and isn’t tool dependent. The idea is to have a modular structure with variety of possible tooling options. Architecture of performance testing environment is shown in Figure 6.1.

A close up of a logo

Description automatically generated

Figure 6.1 Performance testing environment in continuous integration perspective

### 6.2.2 Proposed Tools

**Scalable load generation cluster**

For deploying Performance testing infrastructure **Carrier.io** continuous testing toolkit will be used which is a set of tools and practices to integrate non-functional tests into delivery pipeline and organize an effective feedback loop.

*Suggested load tools:* The Apache JMeter application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

EPAM provides the custom Docker container with latest version of JMeter, which allows to add additional functionality, like performance test notifications, builds comparison and more.

**Monitoring engine** is responsible for resource utilization monitoring of target environment and load generation cluster.

*Suggested tools:* Telegraf is proposed to be used for monitoring of load. The data gathered by Telegraf will be streamed to InfluxDB.

**Profiling engine** is responsible for profiling of target environment and could be toggled when deep research is required.

*Suggested tools*: Epam provides the custom profiler for Java and Database which are sending all required metrics to InfluxDB

**Results visualization** allows variety of options for flexible graphing, unavailable out-of-the-box for load generation tools. It consists of Graph Engine and Time-Series DB. It collects data from target environment, load generation cluster and monitoring engine. Integration with profiling engine is possible, but optional

*Suggested tools:*  Grafana is visualization engine commonly used for visualization of time series data. Grafana features pluggable panels and data sources allowing easy extensibility and a variety of panels, including fully featured graph panels with rich visualization options. It supports real-time system management and alerting.

A close up of a screen

Description automatically generated

Figure 6.2 Grafana server-side performance dashboard example

A screen shot of a computer

Description automatically generated

Figure 6.3 Grafana resources monitoring dashboard example

InfluxDB is an open source database specifically to handle time series data with high availability and high-performance requirements.

**CI Engine** is responsible for building and deployment of environments and tests, running tests and observing test status.

*Suggested tools:* Jenkins

# 7. METRICS

Metrics provide information about how well or poorly application is performing as compared to performance objectives. In addition, metrics can help identify problem areas and bottlenecks within application.

Using the desired performance characteristics, identify metrics to be captured that focus on potential pitfalls for each scenario. The metrics can be related to both performance and throughput goals as well as providing information about potential problems; for example, custom performance counters that have been embedded in the application. Performance metrics that should be gathered are described below.

1. **Server-side metrics**
   * + 1. Processor:
          1. Processor utilization
       2. Process:
          1. Memory consumption
          2. Processor utilization
          3. Process recycles
          4. Garbage collection
       3. Memory:
          1. Memory available
          2. Memory utilization
       4. Disk:
          1. Disk utilization
          2. I/O operations rate
       5. Network:
          1. Network utilization
          2. Latency
2. **Business metrics**
   * + Requests:
       1. Requests/sec (throughput)
       2. Requests succeeded
       3. Requests failed
     + Response times:
       1. Requests response times
3. **Application metrics:**
   * + Deadlocks
     + Thread allocation
     + Heap memory usage

# 8. TEAM LINE-UP

Performance team described below

|  |  |
| --- | --- |
| **Team Member** | **Responsibility** |
| Viacheslav Vostrietsov  Role: Lead Performance analyst | * Local Folio environment configuration * Carrier elastic components configuration [Cloud] * Instrumentation of Folio java components with performance monitoring * Instrumentation of Folio DB components with performance monitoring * Enhancement of performance dashboard with monitoring data * Performance strategy * Performance methodology * Performance execution and analysis platform troubleshooting guide * Performance framework and tests development overview * End-to-end demo of performance execution and analysis platform * Recording of overview materials for performance analysis and execution platform |
| Mikhail Hunko  Role: Performance Analyst | * Carrier persistent components configuration [Cloud] * Development of instrumentation for Folio Java components * Development of instrumentation for Folio DB components * Test data generation tool development (based on backup) * Performance execution and analysis platform user guide |
| Vadym Savchenko  Role: Performance Analyst | * Folio check-in flow development * Folio check-out flow development * Folio check-in flow execution and analysis * Folio check-out flow execution and analysis * Performance framework development guide |