AirBnB

Performance Testing Strategy

1. Introduction

The purpose of this document is to specifically describe how the performance requirements for AirBnB application will be tested and verified in test environment (System Under Test, SUT). The document will outline the scenarios, tests, parameters, and data used in evaluating the capacity of the included features. The scope of tests described in current document is to verify how SUT behaves under load stress. All deliverables should be used for internal (in-house) usage only and should not be presented to public.

This document describes the strategy of performance testing for the AirBnB project. It consists of outlines for the following items:

* Scope of testing, test descriptions
* Non-functional requirements (NFR) related to performance
* Pass/fail criteria
* Approach
* Main scenarios
* Test cases
* Test data
* Requirements for test environment

1. Items to be tested

Mainly, Airbnb application will be tested via API, server side.

|  |  |
| --- | --- |
| # | Modules name |
|  | Search service |
|  | Booking service |
|  | Payment service |
|  | Booking management service |

Also, there should be performed integration performance tests with other services. The following components are to be tested:

|  |  |
| --- | --- |
| # | Modules name |
|  | Integration with selected project’s services |
|  | Integration with host service |
|  | End to End |

1. Items not to be tested

Functionality of the whole application

1. Approach
   1. Test types assumed for conducting

The following activities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Test type | Priority | Period | Notes |
|  | Smoke test | Highest | Regular |  |
|  | Capacity | Highest | After significant changes |  |
|  | Load test | High | Regular |  |
|  | Stress | High | Rare |  |
|  | Scalability | Optional | Once |  |
|  | Volume | Optional | Once |  |
|  | Durable | High | Rare |  |

* + 1. Smoke testing

Should be performed every time when functionality of the application and the script need to be checked. Also, if needed can be used as warming up test before main testing step.

* + 1. Capacity testing

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.

* + 1. Fixed load test

The load level should be defined vs capacity:

* + - * Low-load (~10% of capacity)
      * Mid-load (~45% of capacity)
      * High-load (~80% of capacity) (stress test)

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
  + 1. Scalability testing

Can be performed once or more to get multiplier(s) for different number of front-end/other servers.

* + 1. Volume testing

Is to run with small/planned/huge amount of data with regular load to get indicators on application’s responsiveness/metrics change. Should be performed at least once or by special request.

* + 1. Durable(Stability) testing

Supposed long time running the test with the load lower than average. Purpose: assess system stability during long-time load, to find memory leaks. Should be performed occasionally after significant code changes or by special request to make sure the application’s responsiveness and key performance indicators do not change significantly after long time running, and to check on memory leak as well.

1. Non-Functional requirements

* High availability of the system 99.99 %
* Low latency should be < 100 ms
* All transactions should be highly consistent to prevent parallel booking of the same room
* The system should be highly scalable to handle the increasing number of hotels and clients

1. Environmental needs (facilities, hardware, software, network, supplies, level of security, special tools)
   1. Test environment(s)

For performance testing a dedicated environment needed. The configurations of the servers should be as much as close to production’s ones.

Expected numbers of the servers:

* Front-End servers (search, booking, payment & booking management server) - 4
* Kafka Cluster – 1
* Redis Cluster-1
* Casandra Cluster -1
* MySQL

To perform partial and full end-to-end performance testing it’s recommended to have a separated database and other services, i.e. they should not be located on the same server where related services are being tested by another team.

* 1. Testing tools

For performance testing Automated Performance Testing Platform APTP platform usage is supposed.

It includes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Software | Where to install | Supported OS | Description |
| Module | **Software** | **Where to install** | **Supported OS** | **Description** |
| CI/CD agent | Jenkins | Load Generator host | Linux/ Windows | Module responsible for building and deployment of environments and tests, running tests and observing test status |
| Monitoring agent | Telegraf | All involved in testing hosts | Linux/ Windows | Module that get all performance and health metrics from host where it installed (including Load Generator) to get more accurate picture of your performance tests impact |
| Load Generator (LG) | JMeter | Dedicated host | Linux/ Windows | Module that generates load on your application, supported latest JMeter 5.5 |
| Collector | Influx DB | Dedicated/Load Generator host | Linux/ Windows | Module that collects performance metrics of each execution from monitoring agents. |
| Results visualization | Grafana | Dedicated/Load Generator host | Linux/ Windows | Module that provides the ability to visualize the data collected from each test execution |

* 1. Profiling tools

Profiling tools will be selected according to the development language of application itself.

1. Test data

To have whole cycle of performance testing test data for AirBnB should be:

1. Reusable
2. Generated in necessary amount for different stage of testing at any time:
   1. Hotels
   2. Bookings
   3. Messages
   4. Some history and log records for volume testing
   5. Other
3. Cleanable (for example history, log records)
4. Performance Entry, Exit, and Suspension Criteria
   1. Entry Criteria

* Test plan is complete and approved by the client.
* Correct version is installed in performance testing environment, i.e. the version previously functionally tested and fixed if needed
* Test data is complete and in the performance testing environment in sufficient time to allow test scripts to be completed.
* Test accounts have been created in the performance testing environment in sufficient time to allow test scripts to be completed.
* Test scripts complete.
* All assigned resources are available to monitor the test.
* All parameter sets used in the script are generated based on the Database values.
  1. Exit Criteria
* All test scripts completed successfully
* No critical problems encountered
* All non-critical problems are logged
* All test logs are captured
* All post-test notifications sent
  1. Suspension Criteria
* Not all test scripts will complete
* Critical problems are encountered and logged
* Hardware errors prevent the completion of the test

1. Responsibilities

Performance Analysts 2:

* Engineer working on Server-side performance and integration
* Engineer working on test scripts creation

System engineer - DevOps specialist who will work on environment setup

1. Schedule (test milestones and item transmittal events)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Stage | Period | Notes |
|  | Test strategy design | 1 week |  |
|  | Test plan creation | 1.5 weeks |  |
|  | Draft of NFR definition | 1 week |  |
|  | Setting up test environment | 3 weeks |  |
|  | Test data preparation | 1 week |  |
|  | Script/Scenarios development | 2 weeks |  |
|  | Automation test running from CI/CD | 2 weeks |  |
|  | Level 1 Basic tests round  Smoke, Capacity | 1 week |  |
|  | Level 1 test results analysis | 1 week |  |
|  | Level 1 test reporting | 1 week |  |
|  | Levels 2 Basic tests round on  Smoke, Volume | 1 week |  |
|  | Level 2 test results analysis | 1 week |  |
|  | Level 2 test reporting | 1 week |  |
|  | Levels 3 Smoke, Stress | 1 week |  |
|  | Levels 3 test results analysis | 1 week |  |
|  | Level 3 test reporting | 1 week |  |
|  | NFR update, load level definition | 1 week |  |
|  | Updating scripts/scenarios | 2 weeks |  |
|  | Starting end-to-end regular load test running (Level 3) | 2 weeks |  |
|  |  |  |  |
|  |  |  |  |
|  | Running special tests (Scalability, Durable) | By request |  |
|  |  |  |  |
|  |  |  |  |

1. Risks and contingencies

* A significant difference in configuration from the production environment
* Performance testing results can be essentially different even in case of minor difference in think times, arrival rate and test duration
* During the execution of the tests, some major performance or functional problems that may require code changes, creation of a new build may be discovered and in that case it may be necessary to repeat the load test from the beginning
* Load test should be performed against a build that is solid enough, and that has been functionally tested, after code is complete. Failure to follow this rule may result on rework to update test scripts for every new build, plus the load test may need to be repeated from the beginning. This will affect the schedule
* Performance testing tool is not capable of identically reproducing real life scenarios - so results could only be trusted as having limited reliability level
* Network/systems latency issues
* Environment’s unavailability