**The purpose of this document is to support the development of well designed and useful performance tests.**

## Types Of Performance Test

|  |  |  |
| --- | --- | --- |
| **Type of test** | **Why do we need it** | **How do we run it** |
| **Load** | Will I meet my SLA under peak prod load | Run typical user journeys in CI, with stubs for dependent services (having SLA latency) under peak prod load. |
| **Stress** | Will I continue to meet my SLA over unexpected peaks.  Can I recover from the peaks.  Can also be used to verify scalability and concurrency. | Run the Load test, with extreme load, ramping up from normal load, to extreme load causing errors, then back to normal load |
| **Soak** | Will I continue to meet my SLA over extended time, taking note of potentially slowly depleting resources | Run the Load test over a long period. |

# 

## 

## Environment Setup

We need to ensure we have the infrastructure in place to support the testing:

* production like data volumes (individual tables/documents reflecting prod like sizes)
* sufficient test user accounts to meet needs
* production like load balanced app servers (cpus, memory, cores) … multiple nodes
* stubbed third party services with known latency (as per SLA) which might be graded
* well understood caching

## 

## Test Design

We need to create test scenarios that are representative of real flows and journey mixes:

* Each test user will perform a typical journey with representative service calls and pauses
* The journey should take the common code paths  
    
  E.g. In a booking system, 80% are availability requests, and 20% are booking requests, with 5 sec gap between them so we create a journey as follows:  
    
  avail + sleep + avail + sleep + avail + sleep + avail + sleep + book + sleep

## 

## Acceptance Criteria

We need a clear understanding of what the requirements are for the test:

* Requests Per Minute
* Response times for a percentile E.g. check that 99% response time is be less than 'n' seconds overall as well as by each service

Nb. The business/architects should feed into this in the form of an SLA

Further ACs to check scalability, concurrency, resource depletion:

* error rates
* cpu usage
* thread count
* memory usage
* database locks
* thread contention

If you don’t have production like environments (so you can’t get production like performance metrics), you can take the approach of ‘relative testing’ where you measure the difference between performance tests over time, alerting if performance has degraded sufficiently.

## Test Tool

Gatling

<https://gatling.io/> <https://gatling.io/docs/current/quickstart/>

WebPageTest

<https://www.webpagetest.org>

## Further Considerations

* Create a data set-up job that cleans and rebuilds data and users, perhaps using existing CRUD APIs
* Design you application so that it is a independent as possible, e.g. users can be created within your application rather than requiring a third party
* Get a DBA involved if needed
* Network impact may need to be factored-in, e.g. if there will be a significant network lag in production, this needs to be reflected in the results
* Need visualisations of the server and database to surface issues