More realistic load testing

# Zu berücksichtigende Aspekte

## Erhebung von Parametern im Vorhinein

Um Load Tests realistisch zu gestalten, müssen bestimmte Parameter im Vorhinein erhoben werden. Dazu gehören die durchschnittliche Anzahl der Benutzer pro Stunde/Tag/Woche, welche Aktionen werden am häufigsten ausgeführt, wie viele Nutzer arbeiten mit der Applikation zu Höchstzeiten, wie verhalten sich unterschiedliche Nutzerrollen und wie sind diese aufgeteilt auf die Gesamtanzahl an Kunden.

## Realistische Testdaten

Damit die Tests auch aussagekräftig sind, müssen realistische Testdaten bereitgestellt werden. Je nach Anzahl und Ausführung dieser Daten kann es sein, dass die Load Tests stehen oder fallen. Mit nur einer geringen Anzahl kann es vermeintlich wirken, dass die Performanz sehr gut ist, dies jedoch in der Produktivumgebung nicht der Fall ist.

## Uhrzeit und Auslastungsrate

Es gibt bestimmte Zeiten am Tag, wo eine Website normalerweise mehr ausgelastet und andere Zeitpunkte wo nicht so viele Nutzer darauf zugreifen. Dies impliziert unterschiedliche Load Tests mit einer unterschiedlichen Anzahl an Nutzern. Weiters passiert es realistisch gesehen nicht, dass von einem Moment auf den anderen 1000 neue Benutzer auf die Website zugreifen. Dies passiert eher stetig in einer Rate, welche sich über einen Zeitraum erhöht. Diese Rate kann stark fluktuieren, je nach Uhrzeit/Tag/Feiertag, etc.

## Geo-Location

Die Distanz zu einem Data Center und die Anzahl an Hops zwischen dem Server und dem Client spielen eine wichtige Rolle, um Load und Performance Tests realistischer zu gestalten. Wenige Clients verhalten sich exakt gleich und haben dieselbe Distanz.

## Geräte und Browser

Unterschiedliche Geräte und unterschiedliche Browser laden die angefragte Website mit unterschiedlichen Zeiten. Dieser Aspekt ist nicht unbedingt maßgeblich, trägt aber dazu bei zu verstehen, wie lange ein Nutzer auf das Laden einer Website warten muss.

## Nutzerverhalten/Parametrisierung

Benutzer einer Website verhalten sich grundsätzlich nicht gleich. Durch Caching werden große Teile zwischengespeichert und somit die Performanz erhöht. Die Tests sollten randomisiert ablaufen, sodass nicht immer die gecachten Daten aufgerufen werden, sondern der Server dazu gezwungen wird die Daten neuzuladen. Unterschiedliche Nutzer haben unterschiedliche Parameter wie Authentifizierungs-Daten, Suchverhalten, etc. Dazu zählt ebenso das Warteverhalten der Benutzer: Während ein Kunde eine Website länger besucht, verweilt ein anderer Nutzer nur wenige Sekunden auf dieser Site.

## Nutzerprofile

Zusätzlich zum Benutzerverhalten sollten darauf basierend auch unterschiedliche Profile angelegt werden, welche die unterschiedlichen Rollen in dem System darstellen. Ein Administrator verhält sich anders als ein nicht authentifizierter Gast-Nutzer.

## Nutzerpfade

Es werden nicht immer dieselben Wege benutzt, um zu einem bestimmten Ziel zu gelangen. Beispielsweise beginnt ein Benutzer mit der Suche nach einem spezifischen Artikel, während andere Benutzer ihre Favoriten aufrufen und so letztendlich beim selben Artikel landen, aber einen komplett unterschiedlichen Pfad gewählt haben.

## Netzwerkverhalten

Unterschiede in Bandbreiten wie 5G oder Edge sind ebenfalls zu beachten. Es kann zu Unterbrechungen kommen oder grundsätzlich einfach nur langsame Verbindungen von Nutzern.

## Zeitgleiche Anfragen

Je größer die Nutzerbase, desto häufiger wird es vorkommen, dass eine Ressource mehrmals zur exakt selben Zeit aufgerufen und geladen wird. Diese Art von Anfragen sollen ebenfalls simuliert und in den Tests berücksichtigt werden.

# Tools

## Online-Services

### Loadium

Subscription-based Service, welches kompatibel mit JMeter und Gatling ist und verschiedene Arten von Load Tests anbietet: Performance, Stress, Spike Tests.

Bietet unterschiedliche Geopositionen und Bandbreiten für Load Tests an.

Monatliche/Jährliche Kosten.

### LoadNinja

Subscription-based Service. Führt Tests in realen Browsern aus, anstatt diese zu emulieren. Randomisierte sogenannte „think time“ wie lange ein Nutzer auf einer Website verweilt.

Enthält ein Tool, um Requests/Tests basierend auf einer manuell ausgeführten User-Session zu generieren und so den Aufwand für das Erstellen von Test-Skripts zu minimieren.

Monatliche/Jährliche Kosten

## Offline-Tools

### k6

Standalone-Tool, welches ebenso eine Cloud-Version in einem Subscription-Modell anbietet. Tests werden in JavaScript geschrieben und das Tool hat zahlreiche Integrationen für andere Applikationen wie etwa Postman, Apache Kafka, Grafana oder Gitlab. Einfach skalierbar in der k6 Cloud oder anderen distributed environments.

### Gatling

Standalone-Tool, welches auch eine kostenpflichtige Enterprise Version basierend auf einem Subscription-Modell anbietet. Diese kann aber auch selbst gehostet und muss nicht in deren Cloud ausgelagert werden. Tests werden in Java, Kotlin oder Scala erstellt und sind leicht durch ein Maven-Plugin in eine CI/CD Pipeline zu integrieren. Generiert Reports in Form von HTML-Dokumenten. Skalierbar mit der Enterprise Version oder manuell via Skripts.

# Quellen

<https://loadium.com/blog/5-steps-to-create-a-realistic-load-test>

<https://loadninja.com/articles/must-do-strategies-for-realistic-load-tests/>

<https://www.a1qa.com/blog/6-rules-to-perform-realistic-load-testing/>

<https://k6.io/>

<https://gatling.io/>