Siegfried Goeschl

Gatling
Tales From A Journey
Jetzt wechseln:
mygeorge.at
Überweisung

An
Name, IBAN oder Kontonummer des Empfängers

Betrag
0,00 €

Verwendungszweck

Zahlungsreferenz

Auftraggeber-Referenz

Empfang bis
TT.MM.JJJJ

Speichern & neue Überweisung
Diese Überweisung freigeben
George International

- Turning George Online Banking into a multi-tenant and group-wide platform
  - Single code base
  - Currently targeting four tenants
- Server-side George consists of two parts
  - George API Server
  - Core Banking Interface
George International

- George API Server is a fancy transaction search engine based on Elastic Search
- Core banking interface consists of a set REST endpoints provided by each tenant
- Multiple geographical distributed development teams working together
Wrong End of Continuous Delivery?!!
Being At The Wrong End

- Automatic deployment in the early hours
- REST endpoints broken in the morning
- Front-end development tasks delayed
- Complaining became a form of art
What about having a REST client which tests the various REST endpoints every morning to see if they work?!
Functional versus Performance Test Tools
Which Tool To Choose

- Need to test various REST endpoints
- Various functional testing tools in use
  - SoapUI, Selenium, Tosca
- Performance testing got little love
  - Existing JMeter test suite outdated
  - Having a working performance test suite would be a huge bonus
That sounds neat. Our QA people are moving to Gatling, so we probably won’t change our JMeter approach now.

We use the JMeter Plugs CMDrunner, telling it to generate only CSV.

http://jmeter-plugins.org/wiki/JMeterPluginsCMD/

Walter Underwood
wunder@wunderwood.org
http://observer.wunderwood.org/  (my blog)
The Case For Gatling

- Gatling scripts are written in Scala
- Scala DSL works nicely with your IDE
  - Auto-completion
  - Refactoring
  - Version control
  - Debugging
Software is like sex, it’s better when it's free.

Linus Torvalds
Under The Hood

- Supports HTTP & JMS protocol
- Response validation
  - Regular expressions
  - XPath & JSONPath
  - CSS selectors
- Management-friendly HTML test reports
Under The Hood

- Gatling Recorder to capture user requests
- Based on Akka & Netty libraries
  - Non-blocking, asynchronous requests
  - One thread for many virtual users
Plans are only good intentions unless they immediately degenerate into hard work.

Peter Drucker
The First Steps

- Proof of concept done as side project
- Gatling tests executed every morning
- When the Gatling test failed
  - Contact backend developers
  - Inform George FE team
The First Steps

class Test extends ConfigurableSimulation {

    val users = scenario("Users")
        .repeat(userCsvFeeder.records.length) {
            feed(userCsvFeeder)
                .exec(
                    Token.get,
                    GeorgeConfiguration.get,
                    GeorgePreferences.get,
                    GeorgeAccounts.get,
                    GeorgeTransactions.get
                )
        }

    setUp(users.inject(atOnceUsers(1)))
        .protocols(httpConfServer)
        .assertions(global.failedRequests.count.is(0))
}
class Test extends ConfigurableSimulation {

  val users = scenario("Users")
  .repeat(userCsvFeeder.records.length) {
    feed(userCsvFeeder)
    .exec(
      Token.get,
      GeorgeConfiguration.get,
      GeorgePreferences.get,
      GeorgeAccounts.get,
      GeorgeTransactions.get
    )
  }

  setUp(users.inject(atOnceUsers(1)))
  .protocols(httpConfServer)
  .assertions(global.failedRequests.count.is(0))
}
class Test extends ConfigurableSimulation {

    val users = scenario("Users")
        .repeat(userCsvFeeder.records.length) {
            feed(userCsvFeeder)
            .exec(
                Token.get, 
                GeorgeConfiguration.get, 
                GeorgePreferences.get, 
                GeorgeAccounts.get,
                GeorgeTransactions.get
            )
        }

    setUp(users.inject(atOnceUsers(1)))
        .protocols(httpConfServer)
        .assertions(global.failedRequests.count.is(0))
}
class Test extends ConfigurableSimulation {

    val users = scenario("Users")
        .repeat(userCsvFeeder.records.length) {
            feed(userCsvFeeder)
                .exec(
                    Token.get,
                    GeorgeConfiguration.get,
                    GeorgePreferences.get,
                    GeorgeAccounts.get,
                    GeorgeTransactions.get
                )
        }

    setUp(users.inject(atOnceUsers(1)))
        .protocols(httpConfServer)
        .assertions(global.failedRequests.count.is(0))
}
class Test extends ConfigurableSimulation {

    val users = scenario("Users")
        .repeat(userCsvFeeder.records.length) {
            feed(userCsvFeeder)
                .exec(
                    Token.get,
                    GeorgeConfiguration.get,
                    GeorgePreferences.get,
                    GeorgeAccounts.get,
                    GeorgeTransactions.get
                )
        }

    setUp(users.inject(atOnceUsers(1)))
        .protocols(httpConfServer)
        .assertions(global.failedRequests.count.is(0))
}
class Test extends ConfigurableSimulation {

val users = scenario("Users")
    .repeat(userCsvFeeder.records.length) {
        feed(userCsvFeeder)
        .exec(
            Token.get,
            GeorgeConfiguration.get,
            GeorgePreferences.get,
            GeorgeAccounts.get,
            GeorgeTransactions.get
        )
    }

setUp(users.inject(atOnceUsers(1)))
    .protocols(httpConfServer)
    .assertions(global.failedRequests.count.is(0))
}
The First Steps

```java
object GeorgeAccounts {

    val get = exec(http("my/accounts")
        .get(ConfigurationTool.getUrl("georgeapi", "my/accounts"))
        .header("Authorization", "bearer ${token}")
        .check(
            jsonPath("$.collection[*].id")
                .ofType[String].findAll().optional().saveAs("accountIds"))
    }
```
object GeorgeAccounts {
    val get = exec(http("my/accounts")
        .get(ConfigurationTool.getUrl("georgeapi", "/my/accounts"))
        .header("Authorization", "bearer \${token}"))
    .check{
        jsonPath("$.collection[\*].id")
            .ofType[String].findAll.optional.saveAs("accountIds"))
    }
}
object GeorgeAccounts {

    val get = exec(http("my/accounts")
        .get(ConfigurationTool.getUrl("georgeapi", "/my/accounts"))
        .header("Authorization", "bearer ${token}"))
    .check(
        jsonPath("$.collection[*].id")
            .ofType[String].findAll.optional.saveAs("accountIds"))
}

Set OAuth Token
object GeorgeAccounts {
    val get = exec(http("my/accounts")
        .get(ConfigurationTool.getUrl("georgeapi", "/my/accounts"))
        .header("Authorization", "bearer ${token}"))
        .check(
            jsonPath("$.collection[*].id")
                .ofType[String].findAll().optional().saveAs("accountIds")))
}
Project gatling-george-at-dev-functional

Functional Gatling test for the George Deployment on george.beeone.lan

Project disk usage information + trend graph

- Disk Usage: Workspace 501 MB (On slaves 189 MB, Non slave workspaces 312 MB), Builds 2 MB (Locked -), Job directory 2 MB

Disk Usage Trend

- Disk usage (MB)

Workspace

Recent Changes

Permalinks

- Last build (#336), 14 min ago
- Last stable build (#334), 1 day 2 hr ago
- Last successful build (#334), 1 day 2 hr ago
- Last failed build (#336), 14 min ago
- Last unsuccessful build (#336), 14 min ago
- Last completed build (#336), 14 min ago
Ant Or Not To Ant

- Apache Ant to start Gatling
  - Simplify command line invocation
  - Providing standard Maven targets
- Add convenience function
  - Configuration information
  - Delete temporary data
  - Archiving test reports
Ant Or Not To Ant

gatling.sh --simulation
at.beeone.george.test.gatling.
simulation.george.at.functional.Test

ant test
What about running Gatling for other tenants and/or environments?!
Welcome To The Matrix
## Configuration Dimensions

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenant</td>
<td>The tenants (aka countries) provide additional features you need to test</td>
</tr>
<tr>
<td>application</td>
<td>The application to simulate, e.g. the web client might provide a different functionality than an iOS app</td>
</tr>
<tr>
<td>site</td>
<td>Different sites (staging environments) require different REST endpoint URLs</td>
</tr>
<tr>
<td>scope</td>
<td>Scope of tests, e.g. smoke test versus functional test</td>
</tr>
</tbody>
</table>
## Configuration Items

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment.properties</td>
<td>Base URLs for REST endpoints and other data</td>
</tr>
<tr>
<td>user.csv</td>
<td>User credentials consisting of username and password</td>
</tr>
</tbody>
</table>
- 'at/fat/george/smoketest'
- 'environment.properties' are merged upwards where first value wins
- 'user.csv' are searched upwards where first file wins
ant
-Dtenant=at
-Dsite=fat
-Dapplication=george
-Dscope=smoketest

clean info test
Server-side Refactoring
Server Side Refactoring

• Large refactoring to expose DTOs only
• Mostly a mechanical task aided by the IDE
• There is still room for errors
  ▶ Web UI can be quickly fixed
  ▶ Breaking mobile apps?!
What about having a REST client which tests the REST endpoints every morning to see if the JSON response has changed?!
Gatling Going Functional

- Pretty-print the server’s JSON response
- Skip moving parts, e.g. “lastLoginDate”
- Save final JSON with a meaningful name
- Ant script compares current and expected JSON responses
object GeorgeTransactions {

  val RELATIVE_URL = "my/transactions"

  val getAll = exec(http("my/trx"))
    .get(ConfigurationTool.getUrl("georgeapi", RELATIVE_URL))
    .header("Authorization", "bearer ${token}")
    .QueryParam("pageSize", 50)
    .check(
      jsonPath("$").ofType[Any].find.saveAs("lastResponse")
    )
    .exec(session => {
      val userId: String = session.get("user").as[String]
      JsonResponseTool.saveToFile(session,
                                 "lastResponse",
                                 "george",
                                 RELATIVE_URL,
                                 userId)
      session
    })
}

Gatling Going Functional

- JSON path expression to extract response
- Stored as “lastResponse” in user session
- Extract current user id from user session
- The “JSONResponseTool” pretty-print and stores JSON content in file system
- Creates meaningful name “george-my-transactions-{$userId}.json”
ant clean info record

ant clean info verify
Did Not Work For Us

- Test users shared with QA
- Regular import of new transactions
- Useful for comparing JSON responses before/after a server deployment
Can we use this Gatling tool for performance tests?

Anonymous Manager
YES, WE CAN.
IT’S A BANK, STUPID.
The Challenges

• Working with highly sensitive data
• Running within a secure data center
• No access to load injector boxes
• Minimal effort for rollout
• Time window from 22:00 to 03:00
• Pre-configured test scenarios
Desaster Recovery Tests

- Test failover to backup data center
- Nightly session with multiple teams
- Using Gatling seriously for the first time
  - VMs with 4 CPUs and 8 GB RAM
  - Simulation of 1,000 concurrent users
- Gatling is fast and scales well
<table>
<thead>
<tr>
<th>S</th>
<th>W</th>
<th>Name</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>☀</td>
<td>☀</td>
<td>23 hr - #3</td>
<td>23 hr - #3</td>
<td>N/A</td>
<td>7 min 6 sec</td>
</tr>
<tr>
<td>☀</td>
<td>☀</td>
<td>23 hr - #4</td>
<td>23 hr - #4</td>
<td>N/A</td>
<td>5 min 34 sec</td>
</tr>
<tr>
<td>☀</td>
<td>☀</td>
<td>23 hr - #3</td>
<td>23 hr - #3</td>
<td>N/A</td>
<td>7 min 6 sec</td>
</tr>
<tr>
<td>☀</td>
<td>☀</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>☀</td>
<td>☀</td>
<td>12 days - #7</td>
<td>12 days - #7</td>
<td>12 days - #4</td>
<td>7 min 17 sec</td>
</tr>
<tr>
<td>☀</td>
<td>☀</td>
<td>2 days 12 hr - #11</td>
<td>1 day 0 hr - #14</td>
<td>2 min 10 sec</td>
<td></td>
</tr>
<tr>
<td>☀</td>
<td>☀</td>
<td>12 days - #5</td>
<td>N/A</td>
<td>0.54 sec</td>
<td></td>
</tr>
</tbody>
</table>
READ THE SOURCE
LUKE
package at.beeone.george.test.gatling.simulation.george.at.rampupload

import at.beeone.george.test.gatling.common.ConfigurableSimulation
import at.beeone.george.test.gatling.simulation.george.at.TenantTestBuilder
import io.gatling.core.Predef._

class Test extends ConfigurableSimulation {

  val users = scenario("Rampup Load") {
    repeat(getSimulationLoops) {
      feed(userCsv.circular.random)
        .exec(
          TenantTestBuilder.create("performance")
        )
    }
  }

  setUp(users.inject(rampUsers(getSimulationUsers) over getSimulationUsersRampup))
    .maxDuration(getSimulationDuration)
    .protocols(httpConfServer)
}
object TenantTestBuilder {

val GEORGE_STATE_POLLING_DURATION = 120

def create(scope: String): List[ChainBuilder] = {

val result = scope match {
  case "functional" =>
    List(
      User.init,
      Token.get,
      GeorgeConfiguration.get,
      GeorgePersonalFinanceManager.post,
      GeorgeAccounts.get,
      GeorgeAccounts.stats,
      GeorgeTransactions.get,
      GeorgeTransactions.queryLastYear,
      GeorgeTransactions.queryAmount,
      GeorgePreferences.get,
      GeorgeCategorization.categorize,
      GeorgeImages.userImage,
      ...
    )
  case _ =>
    List()
}

result
}
Building Blocks
Five Years Ago
• JMeter tests run on the command line
• Ant scripts as cross-platform glue code
  ‣ Multi-dimensional configuration
  ‣ Archiving of test reports
• Hudson job to trigger test execution
  ‣ Scheduler, user management, notifications

That Was Five Years Ago
The Building Blocks

- Test execution over command line
  - Cater for SSH only access
- Ant script as cross-platform glue code
  - Human-friendly command line
- CI server to trigger test execution
  - Pre-configured test scenarios
BREAK THE CHAINS
The Case for CI Servers

- Everyone can start a performance test
- Test results are visible and archived
- CI servers have powerful features
  - User management, scheduler, emails
- Wired with version control system
  - Always using the latest version
Gatling @ Erste Bank
The Current State

- Gatling is well established for AT & CZ
  - Using a Gatling, Ant, Jenkins setup
  - Daily tests for DEV & FAT
- More things for CZ deployment
  - Performance test setup for CZ PROD
  - Competing with HP LoadRunner
Is Gatling For You?

• Gatling’s DSL is elegant & powerful
  ‣ Scala & DSL learning curve
  ‣ Requires solid development skills

• Developer-friendly tool
  ‣ Code only
  ‣ IDE support & refactoring
  ‣ Works on Windows, Linux & OS X
Things To Take Home

• Gatling is powerful but not for everyone
• Move performance tests to a CI server
  ‣ Ease of use & visibility
• Embrace the “configuration matrix”
  ‣ You will have DEV & PROD anyway
Things To Take Home

- Performance test tools should be agile
  - Move freely from your laptop to a newly provisioned VM in a data center
- Performance tests are an asset
  - Smoke tests for free
  - Re-used in unplanned & creative ways
Unplanned & Creative Ways
Questions & Answers
Resources

• http://gatling.io

• https://huddle.eurostarsoftwaretesting.com/gatling-tales-from-a-journey/

• http://automationrhapsody.com/performance-testing-with-gatling/

• https://github.com/sgoeschl

• http://people.apache.org/~sgoeschl