Load Testing Made Easy with Gatling

Rafał Piotrowski

Rafał Piotrowski

ITERATORS

https://twitter.com/r_piotrow

https://github.com/rpiotrow

Load testing

“modeling the expected usage of a software program by simulating multiple users accessing the program concurrently”

Wescott, Bob (2013). The Every Computer Performance Book, Chapter 6: Load Testing. CreateSpace. ISBN 978-1482657753.

Use case

- Scala service
- GraphQL API (exposed using Grackle)
- PostgreSQL database

GraphQL

- Data query and manipulation language
- API standard
- Efficient, powerful and flexible alternative to REST
- With schema and strong type system
- Client choose what to fetch
- Server parse and serve result
- Client can modify data and subscribe to notifications (if schema contains that possibilities)

https://graphql.org/

Grackle

- GraphQL server implementation
- Powered by cats, cats-effect, and circe
- Integration with doobie
- Easy mapping GraphQL into SQL

https://github.com/gemini-hlsw/gsp-graphql

GraphQL Schema: Company

```
type Company {
 id: String!
 name: String!
 industry: String!
 location: Location!
 foundedYear: Int!
 website: String
 email: String
 phone: String
 socialMedia: SocialMedia!
 employees: [Employee!]!
}
```

GraphQL Schema: Location & SocialMedia

```
type Location {
 address: String!
 postCode: String!
 city: String!
 country: String!
}

type SocialMedia {
 facebook: String
 instagram: String
 twitter: String
 mastodon: String
 linkedIn: String
}
```

GraphQL Schema: Employee

```
type Employee {
 firstName: String!
 lastName: String!
 email: String!
 phone: String
 position: String!
 department: String!
 startDate: DateTime!
 projects: [Project!]!
}
```

GraphQL Schema: Project

```
type Project {
 name: String!
 description: String!
 startDate: DateTime!
 endDate: DateTime!
 status: ProjectStatus!
 budget: Float
}
```

GraphQL Schema: Query

```
type Query {
 company(id: String!): Company
 companies(
   pageNumber: Int = 1,
   itemsPerPage: ItemsPerPage = ItemsPerPage_10,
   orderBy: OrderBy = OrderByNameAscending
 ): [Company!]!
}
```

GraphQL: Query example

```
query {
 company(id: "63cd7961-cd2c-4107-8ec9-bdd947ab41a5" ) {
   name
   location {
     city
     country
   }
   employees {
     firstName
     lastName
     projects {
       startDate
       endDate
     }
   }
 }
}
```

GraphQL: Query example result


```
{
 "data": {
   "company": {
     "name": "Shufflester",
     "location": {
       "city": "Sherbrooke",
       "country": "Canada"
     },
     "employees": [
       {
         "firstName": "Lynn",
         "lastName": "Frankcom",
         "projects": [
           {
             "startDate": "1999-05-31T02:02:42Z",
             "endDate": "2001-01-27T02:02:42Z"
           }
         ]
       }
     ]
   }
 }
}
```

PostgreSQL (under the hood)

```
CREATE TABLE companies
(
   id                     VARCHAR PRIMARY KEY,
   name                   VARCHAR NOT NULL,
   industry               VARCHAR NOT NULL,
   location address       VARCHAR NOT NULL,
   location post code     VARCHAR NOT NULL,
   location city          VARCHAR NOT NULL,
   location country       VARCHAR NOT NULL,
   founded year           INTEGER NOT NULL,
   website                VARCHAR,
   email                  VARCHAR,
   phone                  VARCHAR,
   social media facebook  VARCHAR,
   social media instagram VARCHAR,
   social media twitter   VARCHAR,
   social media mastodon  VARCHAR,
   social_media_linked_in VARCHAR
);
```

PostgreSQL (under the hood)

```
CREATE TABLE employees
(
   id          VARCHAR PRIMARY KEY,
   company id  VARCHAR                  NOT NULL REFERENCES companies (id),
   first name  VARCHAR                  NOT NULL,
   last Name   VARCHAR                  NOT NULL,
   email       VARCHAR                  NOT NULL,
   phone       VARCHAR,
   position    VARCHAR                  NOT NULL,
   department  VARCHAR                  NOT NULL,
   start_date  TIMESTAMP WITH TIME ZONE NOT NULL
);
```

PostgreSQL (under the hood)

```
CREATE TABLE projects
(
   id          VARCHAR PRIMARY KEY,
   name        VARCHAR                   NOT NULL,
   description VARCHAR                   NOT NULL,
   start date  TIMESTAMP WITH TIME ZONE NOT NULL,
   end date    TIMESTAMP WITH TIME ZONE NOT NULL,
   status      VARCHAR                   NOT NULL,
   budget      NUMERIC(16, 2)
);
```

PostgreSQL (under the hood)

```
CREATE TABLE employee_project
(
   employee id VARCHAR NOT NULL REFERENCES employees (id),
   project id  VARCHAR NOT NULL REFERENCES projects (id),
   CONSTRAINT employee_project_primary_key PRIMARY KEY (employee_id, project_id)
);
```

Grackle mapping

```
trait CompaniesMapping[F[_]] extends DoobieMapping[F]:

 object companies extends TableDef("companies"): [...]
 object employees extends TableDef("employees"): [...]
 object projects extends TableDef("projects"): [...]
 object employeeProject extends TableDef("employee_project"): [...]

 val schema = [...]

 val typeMappings = [...]

 override val selectElaborator: SelectElaborator = [...]
```

GraphQL Service

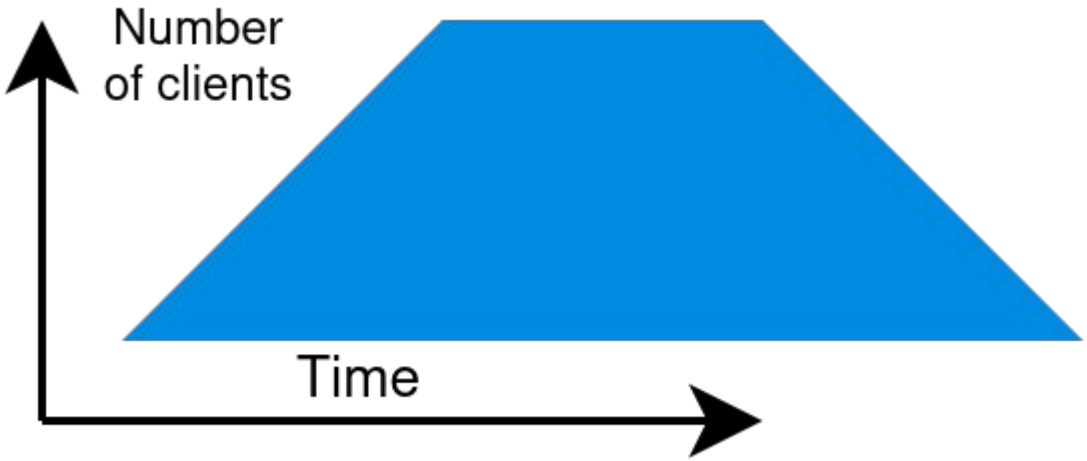
```
trait GraphQLService[F[ ]]:
 def runQuery(op: Option[String], vars: Option[Json], query: String):
   F[Json]

object GraphQLService:

 def fromMapping[F[ ]: Concurrent](mapping: Mapping[F]):
   GraphQLService[F] =
     (op: Option[String], vars: Option[Json], query: String) =>
       mapping.compileAndRun(query, op, vars)

 def routes[F[ ]: Concurrent](
   prefix: String,
   service: GraphQLService[F]): HttpRoutes[F] = [...]
```

Start the server

```
object HttpServer:
 def run[F[_]: Async: Network: LoggerFactory](graphQLRoutes: HttpRoutes[F]): F[Unit] =
   for
     // Routes for static resources, i.e. GraphQL Playground
     assetRoutes <- resourceServiceBuilder[F]("/assets").toRoutes
     // adding GraphQL routes
     routes = (assetRoutes <+> graphQLRoutes).orNotFound
     // request and response logger
     httpApp = Logger.httpApp[F](true, false)(routes)
     // Spin up the server ...
       <- EmberServerBuilder
       .default[F]
       .withHost(ipv4"0.0.0.0")
       .withPort(port"8080")
       .withHttpApp(httpApp)
       .build
       .useForever
   yield ()
```

Test approach

- Ramp up
- Constant traffic
- Ramp down



Gatling

- open-source load testing solution
- load testing as code
- scenarios readable for everyone
- colorful reports

Gatling: Simulation

```
class CompanyGraphQLQuerySimulation extends Simulation {

 private val httpConf = http
   .baseUrl(Settings.baseUrl)
   .shareConnections

 setUp(
   Scenarios.companyGraphQLQuery.inject(
     (rampUsersPerSec(0) to 20).during(30.seconds),
     constantUsersPerSec(20).during(30.seconds),
     (rampUsersPerSec(20) to 0).during(30.seconds)
   )
 )
   .protocols(httpConf)
   .assertions(
     global.responseTime.percentile3.lt(3000),
     global.responseTime.max.lt(5000),
     global.failedRequests.percent.lt(5)
   )
}
```

Gatling: Simulation

```
setUp(
  Scenarios.companyGraphQLQuery.inject(
    (rampUsersPerSec(0) to 20).during(30.seconds),
    constantUsersPerSec(20).during(30.seconds),
    (rampUsersPerSec(20) to 0).during(30.seconds)
  )
)
```

Gatling: Scenario

```
lazy val companyGraphQLQuery: ScenarioBuilder =
 scenario("Company GraphQL query")
   .feed(companyGraphQLQueryFeeder.random)
   .exec {
     http("Company GraphQL query")
       .post("/api")
       .body(StringBody("#{companyquery}"))
       .headers(testHeaders)
       .check(status.is(200))
       .check(jmesPath("errors").notExists)
       .check(jmesPath("data.company").exists)
   }
```

Gatling: Feeder

```
protected lazy val companyGraphQLQueryFeeder =
 val host = "localhost"
 val port = 5432
 val databaseName = "companies"
 jdbcFeeder(
   url = s"jdbc:postgresql://$host:$port/$databaseName",
   username = "postgres",
   password = "postgres",
   sql = "SELECT id as companyquery FROM companies"
 ).transform { case (_, companyId: String) =>
   val query = companyQuery(companyId)
     .pureApply(Gen.Parameters.default, Seed.random())
   GraphQLQueries.from(query).asJson.noSpaces
 }
```

Digression: ScalaCheck generators

- responsible for generating test data
- many data types supported by default
- represented by the `org.scalacheck.Gen`
- easily composable

https://github.com/typelevel/scalacheck/blob/main/doc/UserGuide.md#generators

Query generator

```
def companyQuery(id: String): Gen[CompanyQuery] =
 for
   fields <- companyFields
 yield CompanyQuery(id, fields)

private def companyFields: Gen[Seq[CompanyField]] =
 flattenSequence(
   Seq(
     option(CompanyField.IdF),
     option(CompanyField.NameF),
     option(CompanyField.IndustryF),
     location,
     option(CompanyField.FoundedYearF),
     option(CompanyField.WebsiteF),
     option(CompanyField.EmailF),
     option(CompanyField.PhoneF),
     socialMedia,
     employees
   )
 )
```

Query model

```
sealed trait Query:
 def fields: Seq[Field]

case class CompanyQuery(id: String, fields: Seq[CompanyField])
  extends Query
```

Query model

```
sealed trait Field:
  def name: String

sealed trait Leaf extends Field

sealed trait NonLeaf extends Field:
  def fields: Seq[Field]
```

Query model

```
enum CompanyField(val name: String) extends Field:
  case IdF extends CompanyField("id") with Leaf
  case NameF extends CompanyField("name") with Leaf
  case IndustryF extends CompanyField("industry") with Leaf
  case LocationF(val fields: Seq[LocationField])
    extends CompanyField("location") with NonLeaf
  case FoundedYearF extends CompanyField("foundedYear") with Leaf
  case WebsiteF extends CompanyField("website") with Leaf
  case EmailF extends CompanyField("email") with Leaf
  case PhoneF extends CompanyField("phone") with Leaf
  case SocialMediaF(val fields: Seq[SocialMediaField])
    extends CompanyField("socialMedia") with NonLeaf
  case EmployeeF(val fields: Seq[EmployeeField])
    extends CompanyField("employees") with NonLeaf
```

Query serialization

```
case class GraphQLQuery(query: String) extends AnyVal

object GraphQLQueries:
 def from(query: Query): GraphQLQuery =
   GraphQLQuery(
     query match
       case companiesQuery: CompaniesQuery => build(companiesQuery)
       case companyQuery: CompanyQuery => build(companyQuery)
   )

 private def build(query: CompanyQuery): String =
   s"""{ company(id: "${query.id}") { ${build(query.fields)} } }"""
```

Invoke Gatling

```
rpiotrow@rpiotrow:~/git/github/rpiotrow/load-testing-graphql-api$ sbt "load-tests/Gatling/te
stOnly io.github.rpiotrow.simulations.CompanyGraphQLQuerySimulation"
```

Invoke Gatling

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                       1200 (OK=63     KO=1137  )
> min response time                                     72 (OK=72     KO=10000 )
> max response time                                 60011 (OK=59996  KO=60011 )
> mean response time                                56697 (OK=31198  KO=58110 )
> std deviation                                     11775 (OK=17686  KO=9538  )
> response time 50th percentile                     60000 (OK=31674  KO=60000 )
> response time 75th percentile                     60001 (OK=46362  KO=60001 )
> response time 95th percentile                     60003 (OK=58901  KO=60003 )
> response time 99th percentile                     60005 (OK=59652  KO=60006 )
> mean requests/sec                                 8.108 (OK=0.426  KO=7.682 )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                            1 (  0%)
> 800 ms <= t < 1200 ms                                 0 (  0%)
> t >= 1200 ms                                         62 (  5%)
> failed                                             1137 ( 95%)
---- Errors --------------------------------------------------------------------
> Request timeout to localhost/127.0.0.1:8080 after 60000 ms      1094 (96.22%)
> i.n.c.ConnectTimeoutException: connection timed out: localhost  43   ( 3.78%)
/127.0.0.1:8080
================================================================================
```

Fix

```
create index employees company_id_idx
    on employees(company id);
create index employee project employee_id_idx
    on employee project (employee id);
create index employee project project_id_idx
    on employee_project (project_id);
```

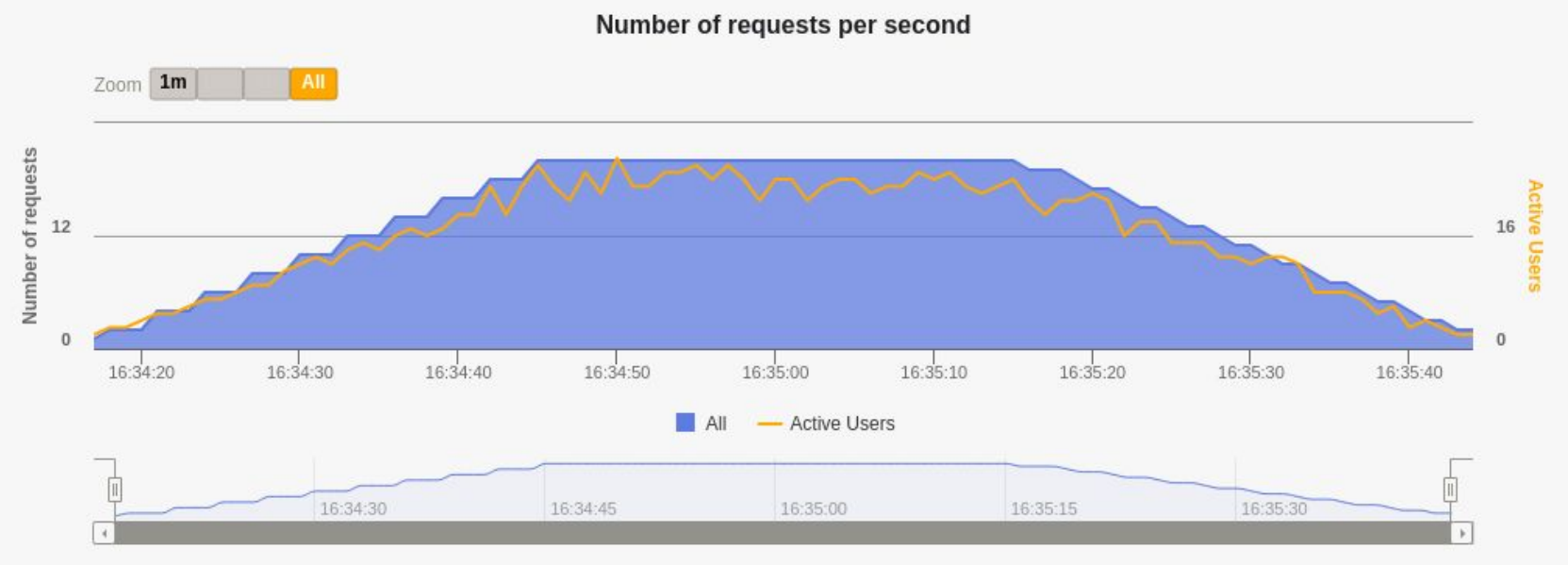
Invoke Gatling

```
rpiotrow@rpiotrow:~/git/github/rpiotrow/load-testing-graphql-api$ sbt "load-tests/Gatling/test
Only io.github.rpiotrow.simulations.CompanyGraphQLQuerySimulation"
```

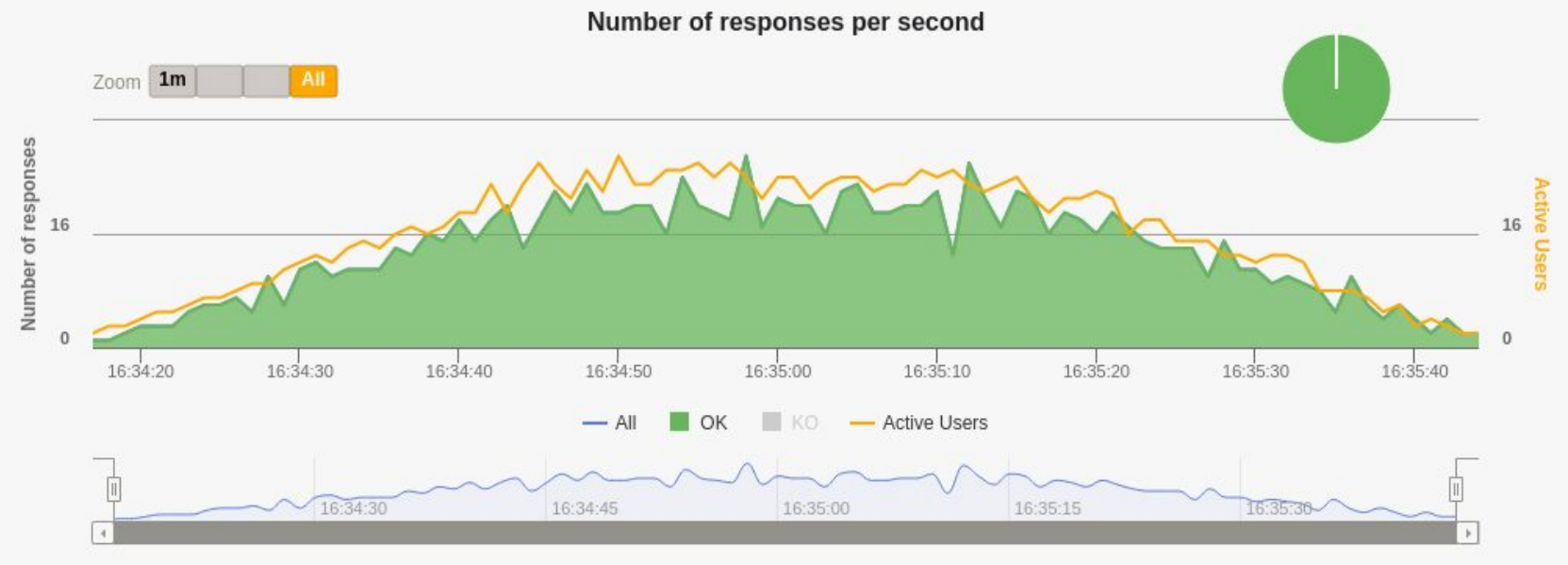
Invoke Gatling

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                       1200   (OK=1200   KO=0   )
> min response time                                   3      (OK=3      KO=-   )
> max response time                                   1222   (OK=1222   KO=-   )
> mean response time                                  156    (OK=156    KO=-   )
> std deviation                                       196    (OK=196    KO=-   )
> response time 50th percentile                       73     (OK=73     KO=-   )
> response time 75th percentile                       241    (OK=241    KO=-   )
> response time 95th percentile                       556    (OK=556    KO=-   )
> response time 99th percentile                       862    (OK=862    KO=-   )
> mean requests/sec                                13.636 (OK=13.636 KO=-   )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                          1179 ( 98%)
> 800 ms <= t < 1200 ms                                 20 (  2%)
> t >= 1200 ms                                           1 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

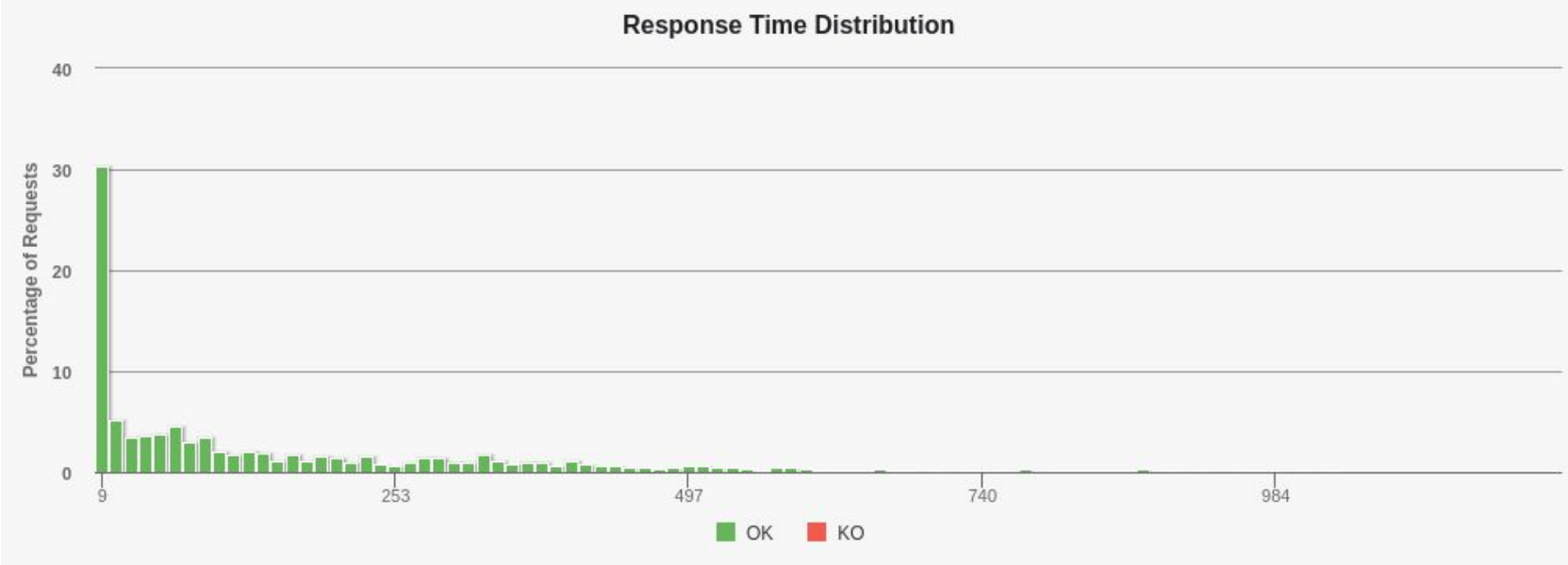
Gatling test summary



Gatling test summary



Gatling test summary



Companies query: simulation

```
class CompaniesGraphQLQuerySimulation extends Simulation {

 private val httpConf = http
   .baseUrl(Settings.baseUrl)
   .shareConnections

 setUp(
   Scenarios.companiesGraphQLQuery.inject(
     (rampUsersPerSec(0) to 2).during(30.seconds),
     constantUsersPerSec(2).during(30.seconds),
     (rampUsersPerSec(2) to 0).during(30.seconds)
   )
 )
   .protocols(httpConf)
   .assertions(
     global.responseTime.percentile2.lt(5000),
     global.responseTime.max.lt(10000),
     global.failedRequests.percent.lt(5)
   )
}
```

Companies query: scenario

```
lazy val companiesGraphQLQuery: ScenarioBuilder =
 scenario("Companies list GraphQL query")
   .feed(companiesGraphQLQueryAsString)
   .exec {
     http("Companies list GraphQL query")
       .post("/api")
       .body(StringBody("#{queryJson}"))
       .headers(testHeaders)
       .check(status.is(200))
       .check(jmesPath("errors").notExists)
       .check(jmesPath("data.companies").exists)
   }
```

Companies query: feeder

```
protected lazy val companiesGraphQLQueryAsString =
 Iterator.continually {
   val query = companiesQuery
     .pureApply(Gen.Parameters.default, Seed.random())
   Map("queryJson" -> GraphQLQueries.from(query).asJson.noSpaces)
 }
```

Companies query: generator

```
def companiesQuery: Gen[CompaniesQuery] =
 for
   itemsPerPage <- Gen.oneOf(ItemsPerPage_5, ItemsPerPage_10)
   //assuming there is at least 500 companies to query
   pageNumber <- Gen.chooseNum(1, 500 / itemsPerPage.value)
   orderBy <- Gen.oneOf(OrderBy.values.toSeq)
   fields <- companyFields
 yield CompaniesQuery(pageNumber, itemsPerPage, orderBy, fields)
```

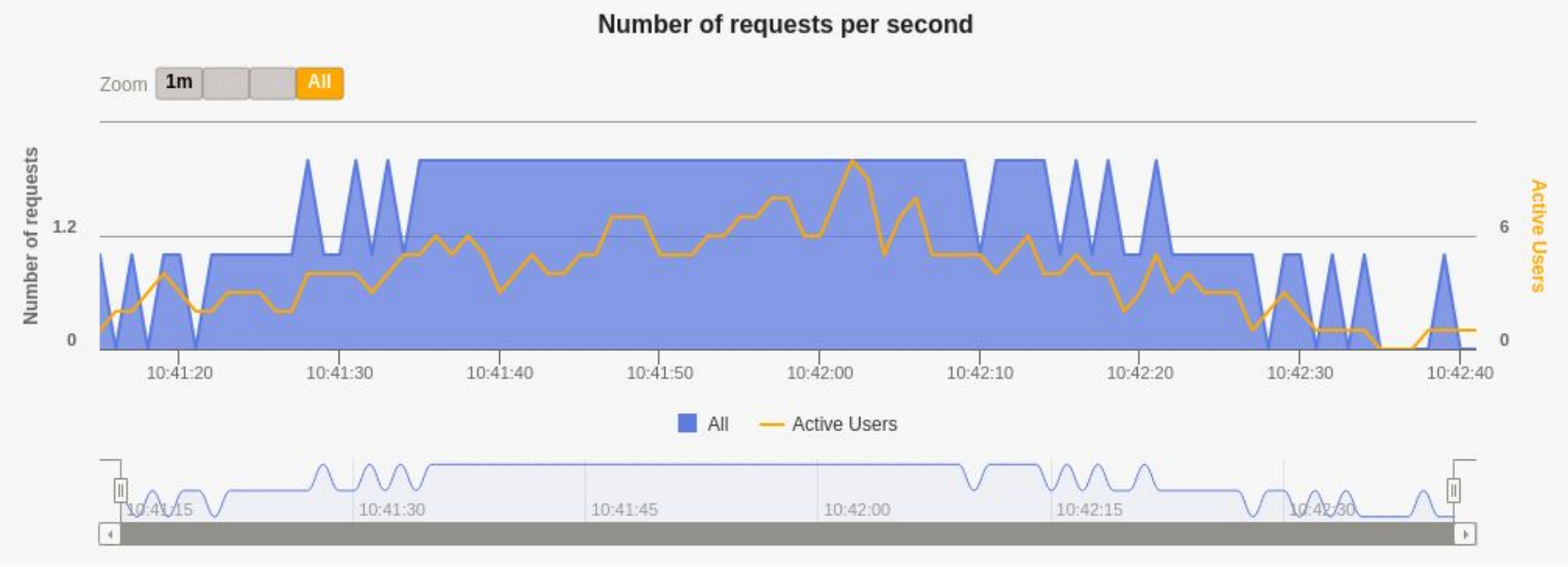
Invoke Gatling

```
rpiotrow@rpiotrow:~/git/github/rpiotrow/load-testing-graphql-api$ sbt "load-tests/Gatling/te
stOnly io.github.rpiotrow.simulations.CompaniesGraphQLQuerySimulation"
```

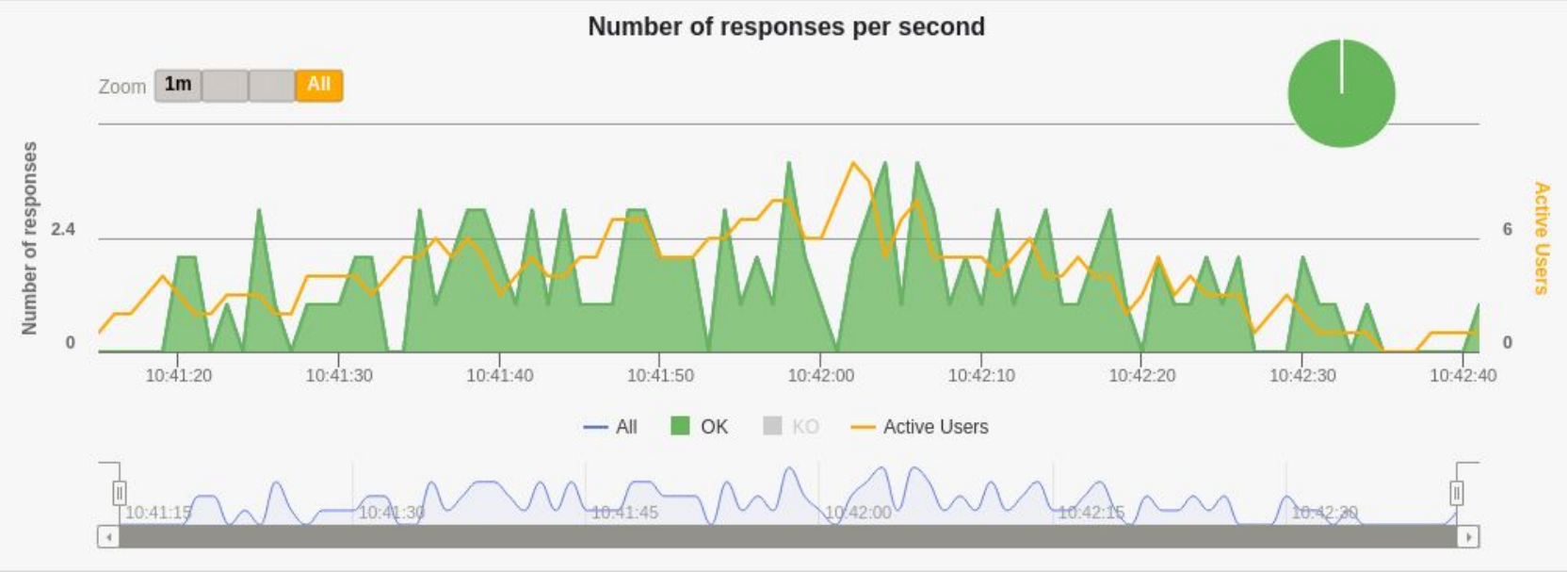
Invoke Gatling

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                        120   (OK=120    KO=0    )
> min response time                                     11   (OK=11     KO=-    )
> max response time                                   5289   (OK=5289   KO=-    )
> mean response time                                  1552   (OK=1552   KO=-    )
> std deviation                                       1266   (OK=1266   KO=-    )
> response time 50th percentile                       1305   (OK=1305   KO=-    )
> response time 75th percentile                       2348   (OK=2348   KO=-    )
> response time 95th percentile                       3816   (OK=3816   KO=-    )
> response time 99th percentile                       4792   (OK=4792   KO=-    )
> mean requests/sec                                  1.379 (OK=1.379  KO=-    )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                           38 ( 32%)
> 800 ms <= t < 1200 ms                                15 ( 13%)
> t >= 1200 ms                                         67 ( 56%)
> failed                                                0 (  0%)
================================================================================
```

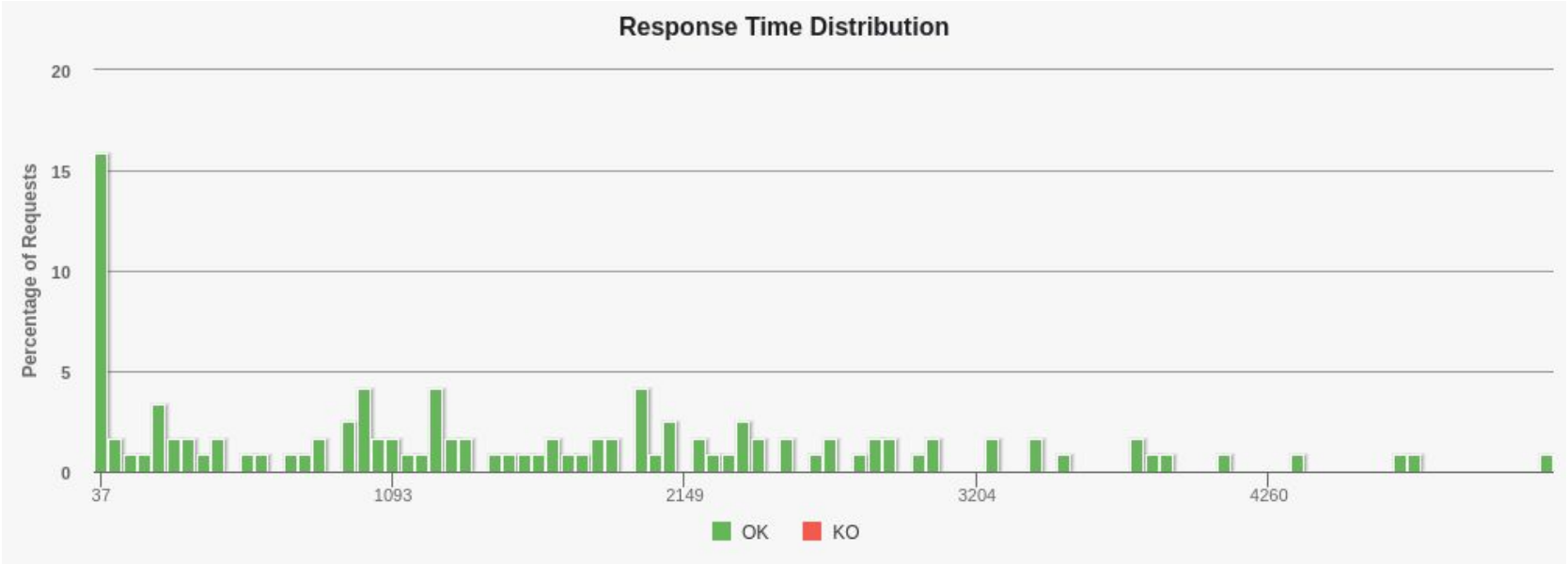
Gatling test summary



Gatling test summary



Gatling test summary



Difference

Simulation 1

```
(rampUsersPerSec(0) to 20)
constantUsersPerSec(20)
(rampUsersPerSec(20) to 0)
```

Simulation 2

```
(rampUsersPerSec(0) to 2)
constantUsersPerSec(2)
(rampUsersPerSec(2) to 0)
```

Bonus: Open vs Closed Workload Models in Gatling

Open model

- you control the arrival rate of users
- e.g. web service

```
(rampUsersPerSec(0) to 5)
constantUsersPerSec(5)
(rampUsersPerSec(5) to 0)
```

Closed model

- you control the concurrent number of users
- e.g. call center
- (should not be used for web service)

```
(rampConcurrentUsers(0) to 5)
constantConcurrentUsers(5)
(rampConcurrentUsers(5) to 0)
```

Summary

- load tests made with Gatling
- using scala 3
- ScalaCheck generators used as inputs to tests
- readable simulations
- nice charts
- open and closed workload models

Thank you!

https://github.com/rpiotrow/load-testing-graphql-api

https://github.com/rpiotrow/load-testing-graphql-api