Investigación y desarrollo de pruebas de Rendimiento automatizadas

Wildey Alejandro Gallego Durango

Informe de prácticas académicas para optar por título de Ingeniería de Sistemas

Asesores

Asesor interno: Daniel Esteban Yepes Palacio, MSc en Ingeniería.

Asesor externo: Juan Agustín Stepanenko

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería de Sistemas

Medellín, Antioquia, Colombia

2022

Cita	(Gallego Durango, 2022)
Referencia Estilo IEEE (2020)	[1] Gallego Durango, W.A, “Investigación y desarrollo de pruebas de Performance automatizadas”, [Trabajo de grado profesional]. Universidad de Antioquia, Medellín, Colombia, 2022.

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Decano/Director: Jesús Francisco Vargas Bonilla.

Jefe departamento: Diego José Luis Botia Valderrama.

Dedicatoria

A mi familia,
y a cada uno de los compañeros que en el paso por esta maravillosa Alma Máter se convirtieron en amigos para toda la vida.

Agradecimientos

Especialmente a mis padres: Wilson Gallego y Deysy Durango, que a pesar de las adversidades de la vida, lo han dado todo por mí. Además, a todos los que estuvieron presentes en mi proceso de formación, que me acompañaron, aconsejaron, enseñaron y gracias a cada granito que aportaron desde su quehacer, hicieron de mí la persona que soy hoy.

Tabla de contenido

LISTA DE TABLAS

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

QA	Quality Assurance.
RNF	Requisitos no funcionales.
DSL	Domain Specific Language.
API	Interfaz de programación de aplicaciones.
CLI	Interfaz de línea de comandos.

RESUMEN

El presente informe de práctica empresarial muestra los resultados obtenidos al realizar la investigación, desarrollo de pruebas y posterior comparativa de tres herramientas dedicadas a la automatización de pruebas de carga vía scripting. Dado que el departamento de Aseguramiento de la Calidad (QA) de la compañía Wolox Part of Accenture le apuesta a la realización de pruebas de software tanto funcionales como no funcionales, se vio en la necesidad de decidir qué herramienta le podría brindar mejores resultados a la hora de hacer pruebas de rendimiento, teniendo foco principal en las pruebas de carga.

Se desarrolla de manera satisfactoria la comparativa entre tres herramientas propuestas por los líderes del área de QA, tales herramientas fueron: Gatling, K6 y Artillery.io. Logrando tener una decisión basada en el análisis de diferentes factores clave a la hora de desarrollar pruebas automatizadas y, al final, se tomó la decisión de cuál podría ser la herramienta más conveniente para realizar pruebas de carga a un desarrollo de software.

Palabras clave **— Automatización de pruebas, pruebas de software, pruebas de rendimiento, pruebas de carga.**

ABSTRACT

This business practice report shows the results obtained from the research, test development and subsequent comparison of three tools dedicated to the automation of load testing via scripting. Since the Quality Assurance (QA) department of the company Wolox Part of Accenture is committed to the performance of both functional and non-functional software testing, it was necessary to decide which tool could provide better results when it comes to performance testing, with the main focus on load testing.

The comparison between three tools proposed by the leaders of the QA area is developed in a satisfactory way, such tools were: Gatling, K6 and Artillery.io. Achieving a decision based on the analysis of different key factors when developing automated tests and, at the end, a decision was made as to which could be the most convenient tool to perform load tests to a software development.

Keywords **— automation testing, software testing, performance testing, load testing.**

I. INTRODUCCIÓN

En la industria de *software* se hace necesario que los aplicativos estén disponibles gran parte del tiempo para que las personas que los usan día a día puedan cumplir con sus respectivas tareas. Puntualmente para esto, se describen los requisitos no funcionales o RNF enfocados en el rendimiento de los desarrollos de software [1]. Dentro de Calidad de Software se realizan diversas pruebas que se agrupan en un conjunto llamado pruebas de *performance* o rendimiento para chequear que se cumpla con estándares mínimos.

Cuándo se abordan las pruebas de *performance* una de las preguntas más recurrentes es, ¿qué herramienta utilizar que apoye el proceso de estas pruebas?, y la respuesta no es única debido a que se pueden abordar dependiendo de diversos factores. Por lo cual se hace necesario realizar un *benchmark* para ayudar a seleccionar las herramientas más adecuadas según las necesidades de los proyectos que se toman desde el departamento de QA de *Wolox part of Accenture*.

Actualmente en Wolox, no se cuenta con un servicio donde se ofrezca formalmente a los clientes realizar pruebas de rendimiento o *performance* a los desarrollos, ya sean implementados por Wolox, que existan en las plataformas del cliente, o que se encuentren en etapas de desarrollo por parte de otros proveedores. Sin embargo, dados los beneficios que tales pruebas aportan a la calidad general del *software*, se ve la necesidad que sean implementadas y ofrecidas a los potenciales clientes, por lo que podría nacer una oportunidad de negocio. Además, la empresa desea realizar pruebas de concepto a diferentes herramientas propias de *performance* para en un futuro implementarlas dentro de sus procesos del área de aseguramiento de la calidad.

En el proyecto, la labor del estudiante consistió en explorar qué son, cómo funcionan, qué hacen y en qué casos generan valor al ser realizadas las pruebas de *performance*. Para cumplir con los objetivos de la práctica académica se realizaron pruebas de concepto de las herramientas *K6*, *Gatling* y *Artillery*; intentando que las condiciones sean similares para que la comparativa sea

lo más real posible. Dentro de los aspectos a evaluar, se tuvieron en cuenta: facilidad de aprendizaje, tiempo que toma realizar el script de pruebas, comunidad existente sobre la herramienta, documentación, costo económico, consumo de recursos, visualización y detalle del reporte de las pruebas ejecutadas, tanto los aspectos a evaluar como la herramienta de evaluación, una matriz de decisión, fueron proporcionados al estudiante por la empresa. Para el desarrollo del proyecto se contará con el acompañamiento de los asesores de la empresa *Wolox part of Accenture* y se hará uso de un marco de trabajo ágil para el seguimiento de las tareas y monitorear los avances del proyecto.

II. OBJETIVOS

A. Objetivo general

Desarrollar pruebas de concepto de diversas herramientas de performance para la generación de una comparativa de rendimiento con el objetivo de conocer la adecuación de estas según casos específicos.

B. Objetivos específicos

- Investigar sobre qué son, cómo funcionan y en qué casos generan valor al ser implementadas las pruebas de performance.
- Analizar arquitecturas, patrones, anti-patrones y buenas prácticas de las pruebas de performance
- Conocer herramientas de pruebas de performance: K6, Gatling y Artillery.
- Calificar cada una de las herramientas de performance anteriormente mencionadas según unos criterios específicos.
- Generar *benchmark* entre K6, Gatling y Artillery.

III. MARCO TEÓRICO

Con el desarrollo tecnológico que se ha dado a lo largo de los últimos años, hay requisitos no funcionales que cada vez toman más importancia a la hora de desarrollar software, entre ellos se encuentran la Disponibilidad, definida como: acceso al software cuando sea requerido, y la Integridad, es decir, que la información y su procesamiento sean exactos y permanezcan completos [2].

No es un secreto que estos aspectos son más relevantes y lo ideal es cubrirlos al desarrollar software. Que el rendimiento de los aplicativos sea el esperado, que esté disponible cuando se necesite, que soporte la cantidad de usuarios esperada y que cuando se encuentre bajo condiciones de uso extremas responda sin ningún contratiempo, son las cualidades que todo cliente desea en su software, llegando al punto en el que esto se convierte en una necesidad que debe ser resuelta por el equipo encargado de desarrollar el software. Para esto, se cuenta con un conjunto de pruebas agrupadas dentro de la categoría llamada *performance* o rendimiento.

Las pruebas de *performance* pueden clasificarse en varios grupos, entre los cuales podemos encontrar: pruebas de carga, que sirven para evaluar el comportamiento de un aplicativo al simular un número de usuarios concurrentes según los esperados en la vida real durante cierto período de tiempo. Pruebas de capacidad, donde se busca el punto de quiebre del producto, determinando la cantidad máxima de usuarios que puede soportar el sistema sin que se vea afectado. Las pruebas de estrés, donde se chequea la robustez y confiabilidad del producto desarrollado saturando el aplicativo hasta puntos extremos para facilitar la configuración de alarmas cuando se llegue a algunos límites [3] y otros más.

Para llevar a cabo tal tipo de pruebas, existen diferentes formas de hacerlo, desde la manual hasta en la que se usan diversas herramientas especializadas para tal fin. En el mercado, pueden encontrarse muchas alternativas [4], siendo algunas de ellas más destacables que otras por diferentes razones como facilidad de uso, comunidad que las soporta e incluso el factor económico.

Entre las herramientas más destacables y que más interesa a la empresa evaluar, se encuentran *K6*, *Gatling* y *Artillery*. Tales herramientas han permitido hacer uso de las técnicas de pruebas específicas para que el aseguramiento de la calidad con respecto al rendimiento de los sistemas pueda ser óptimo.

IV. PLANTEAMIENTO DEL PROBLEMA

Actualmente, el mercado de desarrollo de software ve la necesidad de que los productos que se entregan al cliente final cumplan con requisitos funcionales y no funcionales. Comúnmente, las áreas de QA de las compañías de desarrollo de software se han encargado de asegurar que los productos cumplan con estándares mínimos que satisfagan las necesidades y requerimientos de los clientes. Dentro de la compañía Wolox Part of Accenture, no se cuenta con un proceso definido para la ejecución de pruebas de rendimiento, parte importante de los requisitos no funcionales de aplicativos de software, por lo que no se hace uso de las herramientas que se encuentran disponibles actualmente para tal fin.

Se ha visto la necesidad de que se inicie la exploración y se definan qué herramientas deberían usarse en caso de que en algún momento un cliente de la compañía requiera que, desde el equipo de QA, se inicie el proceso de aseguramiento de la calidad de un desarrollo de software que requiera pruebas de rendimiento, centrándose en las pruebas de carga.

A los líderes del área de calidad, les ha parecido atractivo que los automatizadores de pruebas con los que cuenta la compañía, aprovechen sus habilidades de desarrollo de *scripts* para el desarrollo de pruebas de rendimiento usando herramientas diferentes a las comunes, como lo es puntualmente JMeter y se exploren algunas que puedan utilizarse a través de lenguajes de programación o DSL propios de las herramientas, tales como *K6*, *Gatling* y *Artillery*. Por tal razón, se inició el proceso para compararlas por medio de una matriz de decisión que tiene en cuenta los aspectos básicos y deseables que debería suplir una herramienta para realizar pruebas de carga por medio de *scripts*.

V. METODOLOGÍA

El proyecto se desarrolló en cuatro fases para lograr los objetivos que se plantearon. Tales fueron: investigación, desarrollo, recopilación de datos y conclusiones. Estas fases o etapas se dieron de forma secuencial, a excepción de la de recopilación de datos que fue transversal a la etapa de desarrollo.

Durante la etapa de investigación, se buscó saber qué son, cómo funcionan y en qué casos generan valor al ser implementadas las pruebas de *performance*.

En la etapa de desarrollo, se aplicaron los conocimientos obtenidos en la investigación para realizar las pruebas de concepto de las herramientas de performance elegidas por *Wolox*: *K6, Gatling y Artillery*.

La etapa de recopilación de datos es transversal a la de desarrollo, ya que cada prueba de concepto arrojó información importante para la realización de la comparativa final de las herramientas.

En la última etapa, se realizaron las conclusiones y se resolvió la matriz de decisión aportada por la empresa para la elección de la herramienta que más se acomoda a las necesidades.

Se trabajó haciendo uso del marco de trabajo ágil Kanban [5], en el que se cuenta con un tablero de 3 columnas (por hacer, haciendo y hecho) y donde se mapean las tareas específicas para llevar a cabo durante cada fase.

VI. RESULTADOS

Para el caso de las pruebas de carga, su principal objetivo es detectar posibles cuellos de botella dentro de sus flujos críticos. Al hablar de cuellos de botella, se hace referencia a comprobar componentes del sistema que reciben muchas solicitudes a resolver y estas no están siendo solucionadas en el tiempo esperado, lo que causa que se tenga una gran entrada y una salida muy reducida. Este problema afecta directamente el rendimiento de los aplicativos y, por ende, la existencia de los usuarios.

Dado que en el proceso de prácticas del estudiante el objetivo principal era establecer qué herramienta de desarrollo de pruebas de carga se adecuaba mejor a los lineamientos de la compañía, se optó por elegir un escenario de prueba que permitiera evaluar las capacidades más básicas de cada una de las herramientas, por lo que las pruebas fueron hechas a una API pública y gratuita cuyo fin es el de la realización de diversas pruebas haciendo uso de ella y no la evaluación del sistema en sí mismo.

Tal API, funciona bajo una arquitectura REST. Definido por la comunidad *RedHat*:

> *"REST no es un protocolo ni un estándar, sino más bien un conjunto de límites de arquitectura. Los desarrolladores de las API pueden implementarlo de distintas maneras.*
>
> *Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario."* [7]

Definición de escenarios de prueba.

En primera instancia, se definió el escenario a probar dentro de la API {JSON} Placeholder [6]. Al ingresar al apartado “Guide”, se logra ver cómo puede ser consumida tal API y el proceso definido de prueba fue el siguiente:

A. Consultar todos los recursos de la API: en este caso, se consultará el servicio REST por medio de una petición GET a la ruta “/posts”. Esto retornará una lista de elementos en formato JSON, que contará con un total de 100 mensajes y los datos dentro de cada uno serán: id, título y cuerpo. La verificación a realizar, será que la lista de elementos sea 100 y que el código HTTP de la respuesta sea de OK (200). Al finalizar, se realiza una pausa de 5 segundos para simular el tiempo que un usuario tarda en leer los posts y elegir uno.

B. Consultar un recurso específico de la API: se generará un número aleatorio entre 1 y 100 y se procederá a obtener el mensaje que tenga como id el número generado. Para esto, se consumirá el servicio REST por medio de una petición GET a la ruta “/posts/¨{{idMensaje}})”, donde {{idMensaje}} será el equivalente al número aleatorio generado. La verificación a realizar, será que el código HTTP de la respuesta sea de OK (200). Al finalizar, se realiza una pausa aleatoria entre 1 y 20 segundos para simular el tiempo que un usuario tarda en leer los posts y elegir uno.

C. Consultar los comentarios de un mensaje en específico: se consultarán los comentarios asociados al mensaje obtenido en el paso anterior. A través del servicio REST por medio de una petición GET a la ruta “/posts/{{idMensaje}}/comments”, donde {{idMensaje}} será el equivalente al número aleatorio generado. La verificación a realizar, será que el código HTTP de la respuesta sea de OK (200). Al finalizar, se realiza una pausa de 2 segundos para simular el tiempo que un usuario tarda en leer los posts y elegir.

D. Agregar un nuevo comentario a un mensaje en específico: se hará una petición tipo POST al servicio REST por medio de la ruta “/posts/{{idMensaje}}/comments”, donde {{idMensaje}} será el equivalente al número aleatorio generado. Y cuyo cuerpo de petición será el comentario en formato JSON: “{“name”: “{{nombreHerramienta}}”,

“email”: “{{nombreHerramienta}}@test.com”, “body”: “{{This is a simple comment}}” }”. Donde {{nombreHerramienta}} será: *K6, Gatling* o *Artillery*, según sea el caso. La verificación a realizar, será que el código HTTP de la respuesta sea de Created (201).

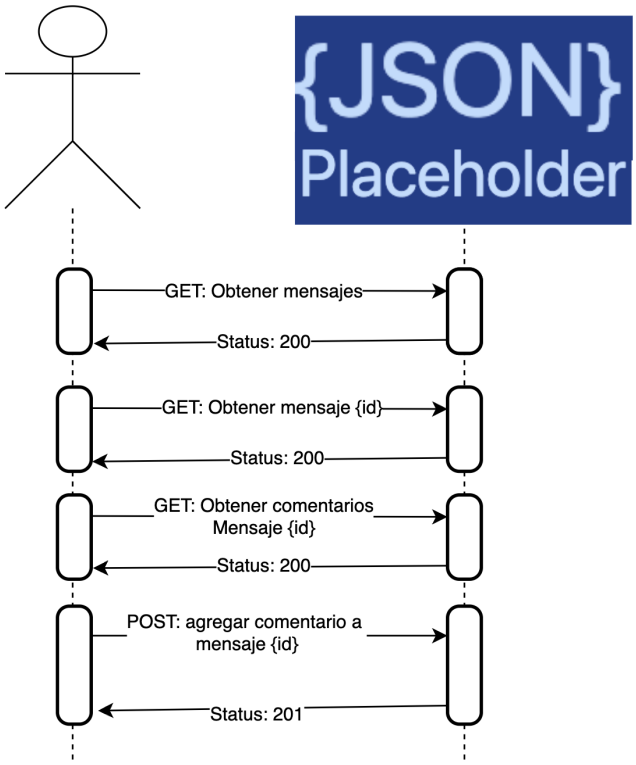


Fig. 1. Diagrama de secuencia de los escenarios de prueba.

Herramientas de pruebas de carga.

1. K6

K6 es una herramienta de pruebas de carga de código libre y centrada en el desarrollador que ha sido diseñada para hacer que la prueba de rendimiento sea un proceso productivo. Desarrollada por Grafana Labs. Es posible testear la fiabilidad y rendimiento de aplicaciones e identificar regresiones y errores más tempranamente. Esta herramienta, según su portal web, ayudará a construir aplicaciones rápidas y robustas que puedan escalar. El backend de este framework se desarrolla utilizando GO y los scripts deben escribirse en Javascript. Hay dos versiones diferentes disponibles para los usuarios: Código abierto y la versión en la nube.

Sus principales características son: una herramienta CLI con APIs amigables para el desarrollador. Scripting en JavaScript ES2015/ES6, con soporte para módulos locales y remotos. Checks y Thresholds - para pruebas de carga orientadas a objetivos y de fácil automatización.

Su reporte es compatible con: Amazon CloudWatch, Apache Kafka, CSV, Datadog, Grafana Cloud, InflixDB+Grafana, Json, Netdata, New Relic, Prometheus, TimescaleDB y StatsD. [7]

Escenario de pruebas

La forma cómo se ejecutan los scripts en K6, se muestra a continuación.

Se ejecuta la prueba anteriormente definida, realizando la codificación en Javascript, donde puede usarse la estructura de typescript si se desea tener orden a la hora de codificar:

```
group('Get all Posts - 1st Call',()=>{
  let URL = `${BASE_URL}/posts`
  const res = http.get(URL)
  check(res,{'response code was 200':(res)=>res.status==200})
});
sleep(5)
```

Fig. 2. Consultar todos los recursos de la API en K6.

```
group('Get Specific Post - 2nd Call',()=>{
  let URL = `${BASE_URL}/posts/${postIdNumber}`
  const res = http.get(URL)
  if(check(res,{'response code was 200':(res)=>res.status==200})){
    postIdNumber = res.json('id');
  }else{
    console.log(`Unable to get specific post number: ${postIdNumber}.
    res status: ${res.status} res body:${res.body}`);
      return;
  }
})
sleep(randomIntBetween(1,20))
```

Fig. 3. Consultar un recurso específico de la API en K6.

```
group('Get Comments for above post - 3rd call ',()=>{
  let URL = `${BASE_URL}/posts/${postIdNumber}/comments`
  const res = http.get(URL)
  check(res,{'response code was 200':(res)=>res.status==200})
})
sleep(2)
```

Fig. 4. Consultar los comentarios de un mensaje en específico en K6.

```
group('Add Comment for above post - 4th call ',()=>{
  let URL = `${BASE_URL}/posts/${postIdNumber}/comments`
  const payload ={
    name: "K6",
    email: "K6@test.com",
    body: "This is a simple comment"
  };
  const res = http.post(URL,payload)
  check(res,{'response code was 201':(res)=>res.status==201})
})
```

Fig. 5. Agregar un nuevo comentario a un mensaje en específico en K6.

Reporte de pruebas

Al finalizar la prueba, se obtienen los siguientes reportes. Cabe mencionar que la salida del reporte se da en formato JSON y puede especificarse qué herramienta tomará tal salida para realizar el reporte. En este caso puntual, se hace uso de una librería que genera un reporte simple en formato HTML donde se resumen los principales resultados y métricas del test.

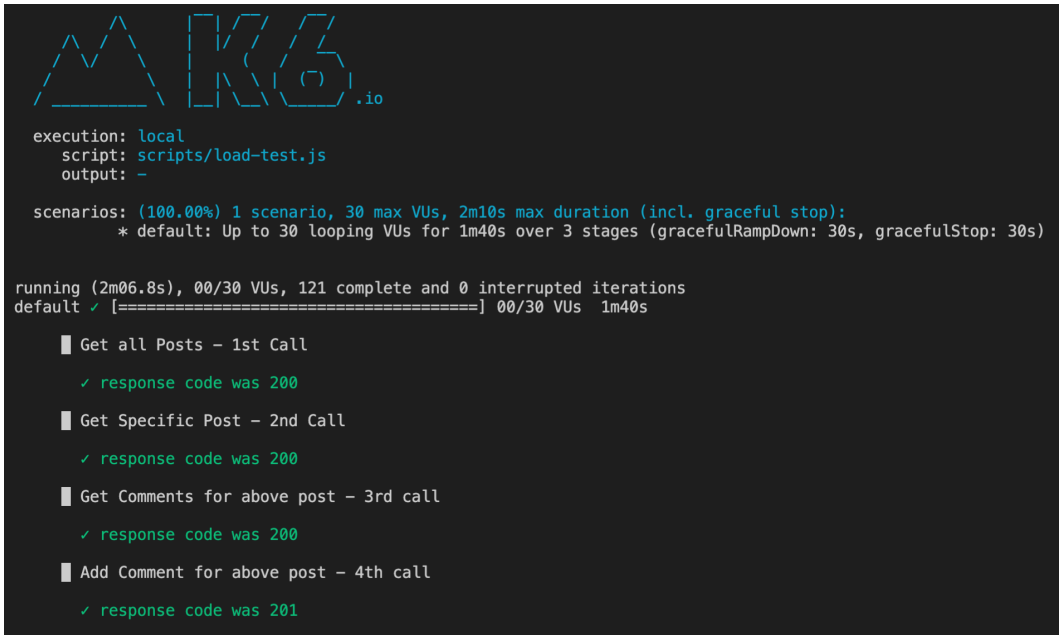
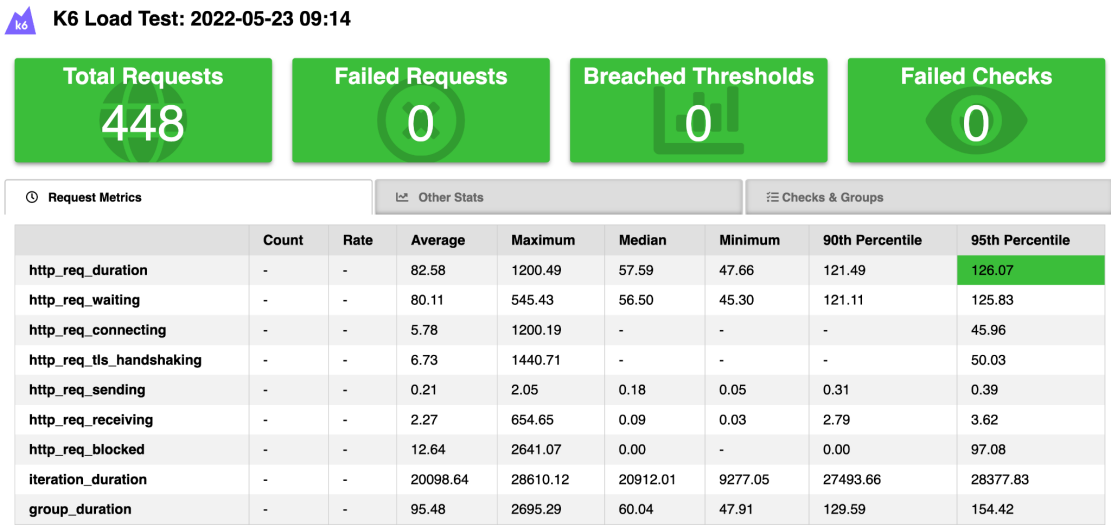


Fig. 6. Reporte de pruebas en consola K6.



	Count	Rate	Average	Maximum	Median	Minimum	90th Percentile	95th Percentile
http_req_duration	-	-	82.58	1200.49	57.59	47.66	121.49	126.07
http_req_waiting	-	-	80.11	545.43	56.50	45.30	121.11	125.83
http_req_connecting	-	-	5.78	1200.19	-	-	-	45.96
http_req_tls_handshaking	-	-	6.73	1440.71	-	-	-	50.03
http_req_sending	-	-	0.21	2.05	0.18	0.05	0.31	0.39
http_req_receiving	-	-	2.27	654.65	0.09	0.03	2.79	3.62
http_req_blocked	-	-	12.64	2641.07	0.00	-	0.00	97.08
iteration_duration	-	-	20098.64	28610.12	20912.01	9277.05	27493.66	28377.83
group_duration	-	-	95.48	2695.29	60.04	47.91	129.59	154.42

Fig. 7. Resumen reporte en formato html en K6.

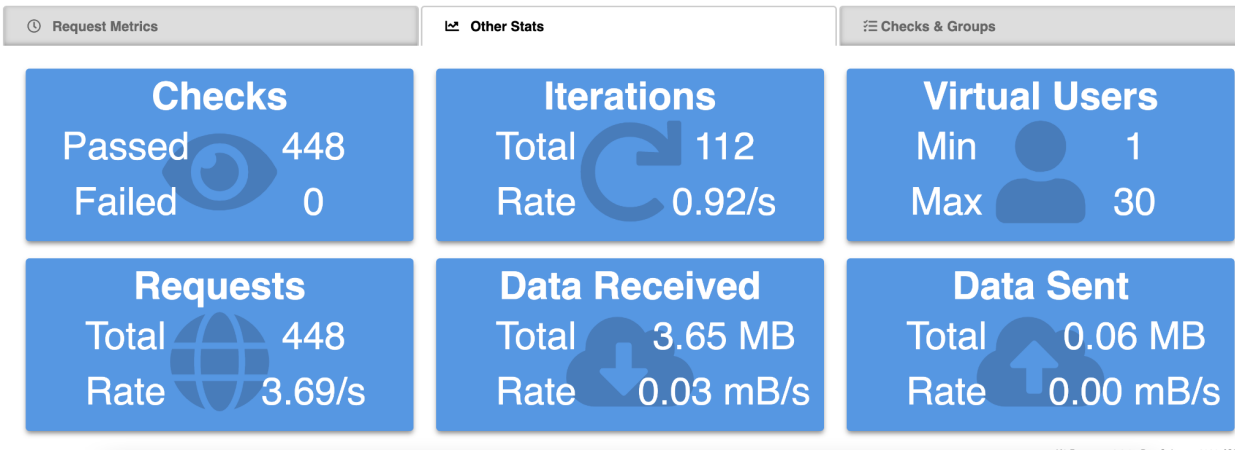
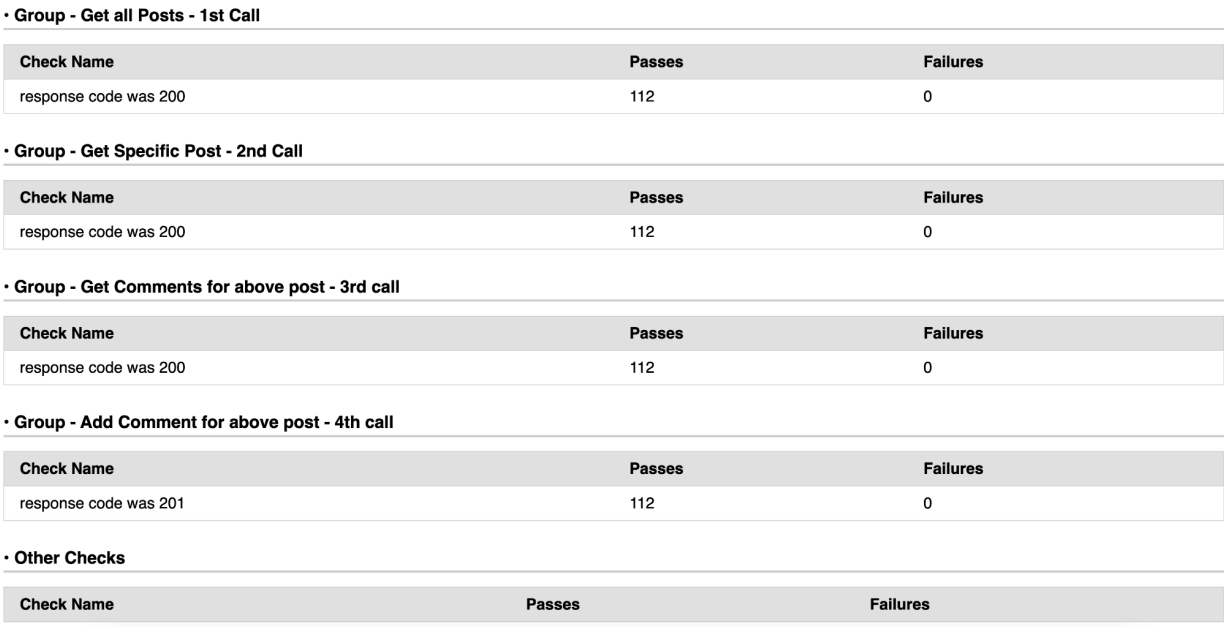


Fig. 8. Estadísticas varias del test realizado por K6 en formato html.



• **Group - Get all Posts - 1st Call**

Check Name	Passes	Failures
response code was 200	112	0

• **Group - Get Specific Post - 2nd Call**

Check Name	Passes	Failures
response code was 200	112	0

• **Group - Get Comments for above post - 3rd call**

Check Name	Passes	Failures
response code was 200	112	0

• **Group - Add Comment for above post - 4th call**

Check Name	Passes	Failures
response code was 201	112	0

• **Other Checks**

Check Name	Passes	Failures

Fig. 9. Total de casos exitosos y fallidos por grupo en K6.

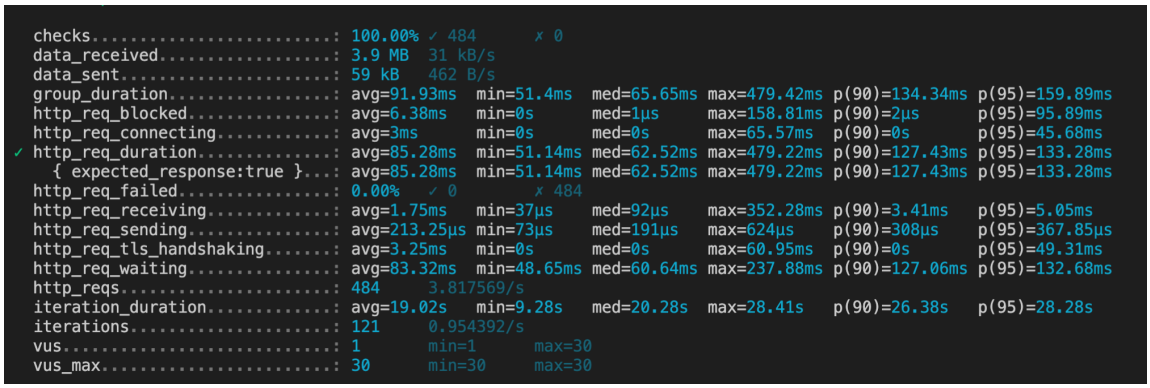


Fig. 10. Reporte de métricas de pruebas en K6.

2. Artillery.io

Es una herramienta de código libre, escrita en Node.js con un CLI básico que permite realizar pruebas de rendimiento. Según su propia documentación, es potente pero bastante simple de navegar. Además, se adapta a diferentes tipos de pruebas de rendimiento, en máquinas locales y como parte de CI. Puede realizar pruebas de carga haciendo uso de: HTTP, Socket.io, WebSocket. Además, permite realizar pruebas en: Apache Kafka, Amazon Kinesis gracias al soporte que tiene con HTTP Live Streaming (HLS), HTTP Live Streaming (también conocido como HLS) es un protocolo de comunicaciones de transmisión de medios basado en HTTP implementado por Apple Inc. como parte de su software QuickTime, Safari, OS X e iOS. Las implementaciones de clientes también están disponibles en Microsoft Edge, Firefox y algunas versiones de Google Chrome.

Para escribir las pruebas en Artillery, se hace uso de un formato YAML, también admite JSON, esto permite que los scripts de prueba sean fáciles de leer, como se puede notar a continuación.

Cuenta con integración a herramientas de CI/CD como: Azure DevOps, CircleCI, GitHub Actions, Gitlab CI/CD, Jenkins, AWS CodeBuild y Keptn.

A la hora de realizar los reportes cuenta con dos opciones: uno en formato JSON, donde se muestran los datos relevantes de la ejecución y otro formato HTML para ser visualizado a través de un navegador, a continuación se muestra el reporte generado en HTML.

En su versión PRO permite ejecutar millones de pruebas por segundo en 13 regiones geográficas, funciona con los sistemas de seguridad existentes, sin cargos repetidos ni mantenimiento pagado, servicios de pruebas internas de VPC. [8]

Escenario de pruebas

La forma cómo se ejecutan los scripts en Artillery, se muestra a continuación.

Se ejecuta la prueba anteriormente definida, realizando la codificación en YAML para Artillery.io:

```yaml
- name: "Get all Posts - 1st Call"
  flow:
    - get:
        url: "/posts"
        capture:
          - json: "$[-1:].id"
            as: totalPosts
        expect:
          - statusCode: 200
          - equals:
            - "{{ totalPosts }}"
            - "100"
    - think: 5
```

Fig. 11. Consultar todos los recursos de la API en Artillery.io.

```yaml
- name: "Get Specific Post - 2nd Call"
  flow:
    - get:
        url: "/posts/{{$randomNumber(1,100)}}"
        capture:
          - json: "$.id"
            as: postId
        expect:
          - statusCode: 200
    - think: {{ $randomNumber(1,20) }}
```

Fig. 12. Consultar un recurso específico de la API en Artillery.io.

```yaml
- name: "Get Comments for above post - 3rd call"
  flow:
    - get:
        url: "/posts/{{ postId }}/comments"
        expect:
          - statusCode: 200
    - think: 2
```

Fig. 13. Consultar todos los comentarios de un recurso en Artillery.io.

```yaml
- name: "Add Comment to the Post - 4th Call"
  flow:
    - post:
        url: "/posts/{{ postId }}/comments"
        json:
          name: "Artillery.io"
          email: "Artillery.io@test.com"
          body: "This is a simple comment"
        expect:
          - statusCode: 201
```

Fig. 14. Añadir un comentario a un recurso en específico en Artillery.io.

Reporte de pruebas

Al finalizar la prueba, se obtienen los siguientes reportes. Cabe mencionar que aquí solo se muestran los gráficos que pueden ser vistos en el reporte HTML. También se cuenta con la opción de tener el reporte en formato JSON e importarlo a una herramienta de reportes que tenga como entrada tal formato.

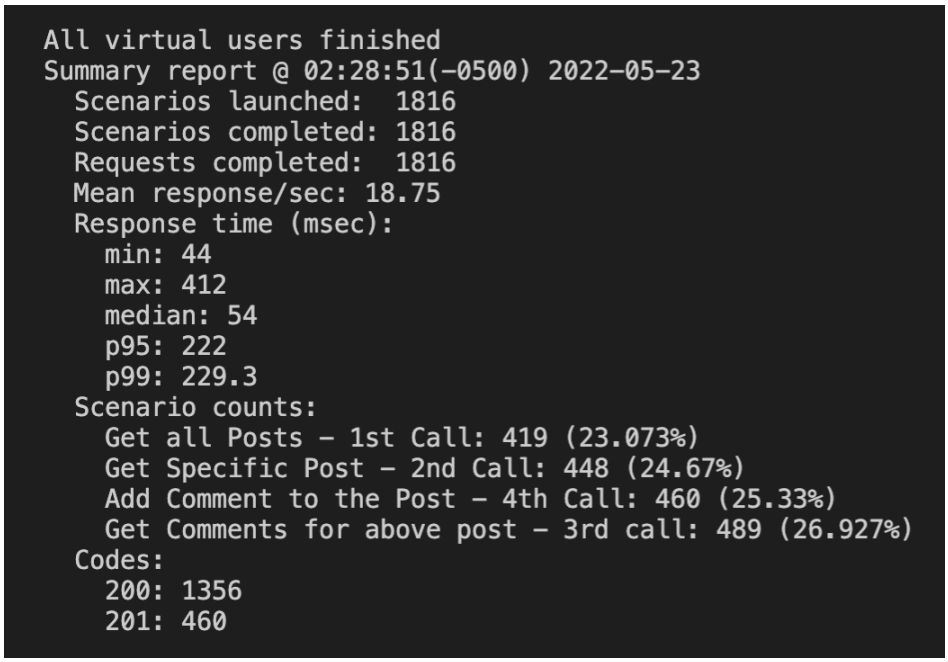
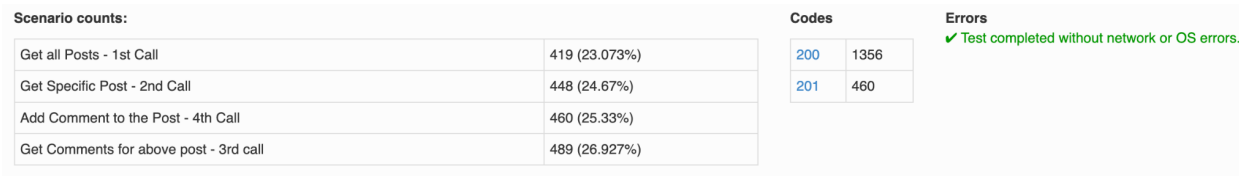


Fig. 15. Reporte en consola de Artillery.io.

Summary

Test duration	110 sec
Scenarios created	1816
Scenarios completed	1816

Fig 16. Resumen de reporte .html generado por Artillery.io.



Scenario counts:

Get all Posts - 1st Call	419 (23.073%)
Get Specific Post - 2nd Call	448 (24.67%)
Add Comment to the Post - 4th Call	460 (25.33%)
Get Comments for above post - 3rd call	489 (26.927%)

Codes

200	1356
201	460

Errors

✔ Test completed without network or OS errors.

Fig. 17. Conteo del total de escenarios ejecutados en Artillery.io

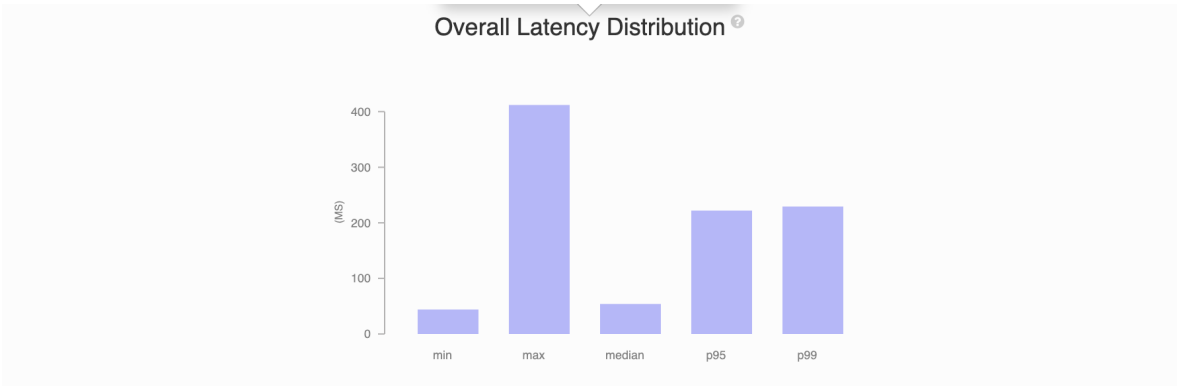


Fig. 18. Reporte de latencia general en Artillery.io

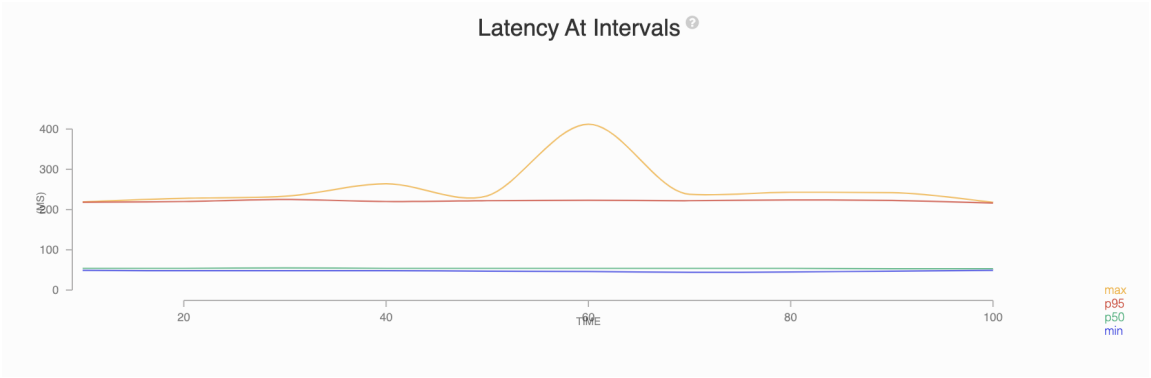


Fig 19. Reporte de latencia por intervalos de tiempo en Artillery.io.

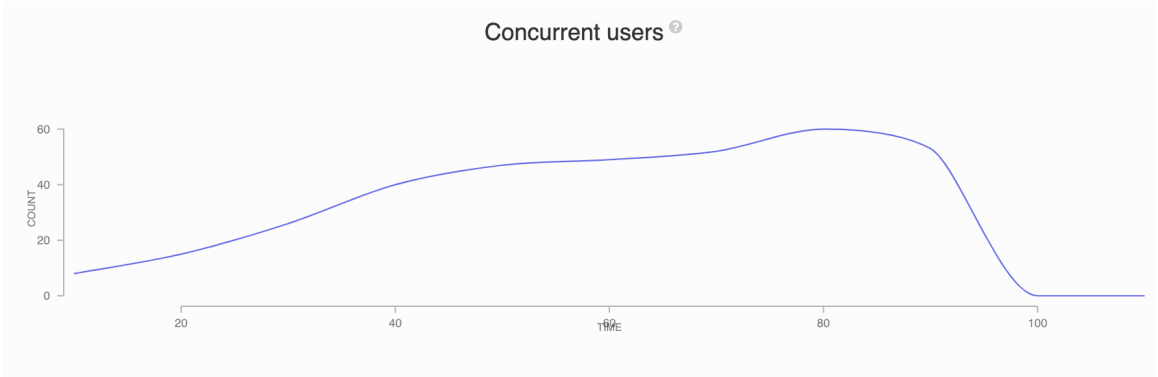


Fig. 20. Reporte de usuarios concurrentes en Artillery.io.

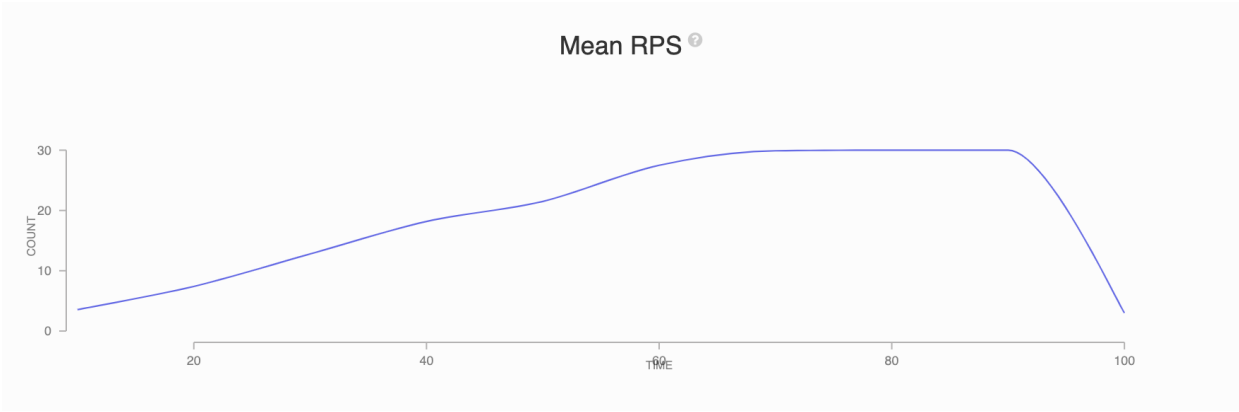


Fig. 21. Reporte de Media de peticiones por segundo (Request per second) en Artillery.io

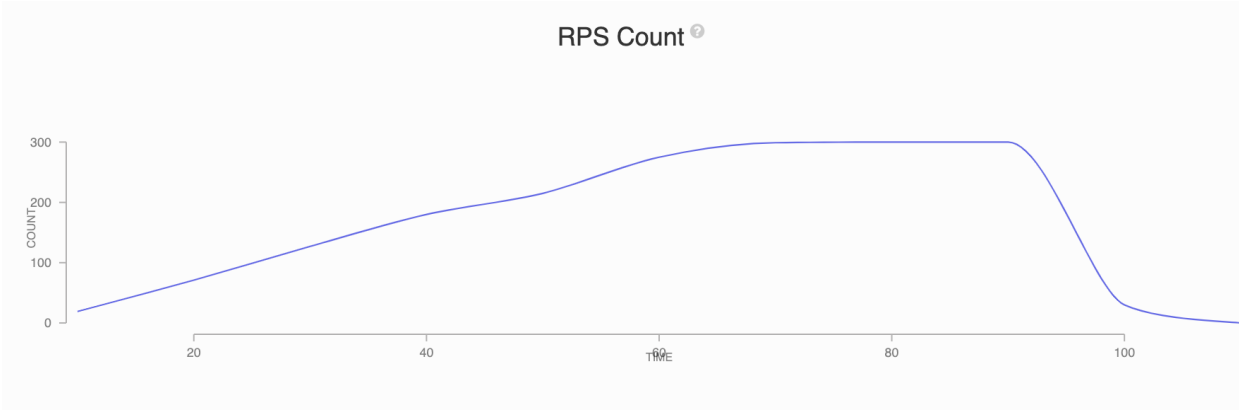


Fig. 22. Conteo total de peticiones por segundo en Artillery.io.

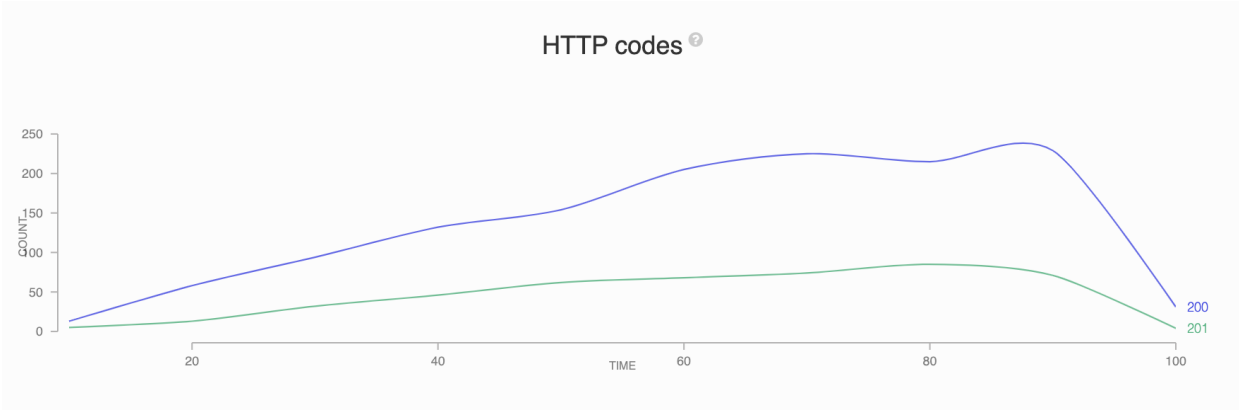


Fig. 23. Ocurrencia de los diferentes códigos http en Artillery.io.

3. Gatling

Gatling es un *framework* de pruebas de rendimiento de código abierto, que utiliza *Scala*, *Akka* y *Netty* como stack de tecnología y como su columna vertebral. El primer lanzamiento de *Gatling* fue en enero de 2012. Desde entonces, *Gatling* ha tenido un par de lanzamientos importantes casi todos los años, así como un crecimiento de popularidad bastante extenso en la comunidad de ingeniería de rendimiento.

Gatling es una herramienta de prueba de carga de código abierto escrita exclusivamente en código Scala. El DSL sencillo y expresivo que ofrece *Gatling* simplifica la escritura de *scripts*

de prueba de carga. Además, de ser compatible con Java, Scala y Kotlin. Es posible ejecutar una gran cantidad de tráfico desde una sola máquina, lo que elimina la necesidad de una infraestructura de prueba distribuida compleja. *Gatling* tiene tableros de métricas detalladas que se muestran a continuación.

Cuenta con soporte para diferentes protocolos como lo son: HTTP, SSL, WebSockets, SSE, JMS, MQTT, lo que brinca robustez a la herramienta.[9]

Escenario de pruebas

La forma cómo se ejecutan los scripts en Gatling, se muestra a continuación.

Se ejecuta la prueba anteriormente definida, realizando la codificación en Java, Scala o Kotlin a través del DSL que provee Gatling. A continuación se muestra la codificación para cada escenario:

```
.exec(http( requestName = "Get all Posts - 1st Call")
  .get("/posts")
  .check(status.is( expected = 200))
  .check(jsonPath( path = "$[-1:].id").is( expected = "100"))
)
.pause( duration = 5)
```

Fig. 24. Consultar todos los recursos de la API en Gatling.

```
.exec (
  http( requestName = "Get Specific Post - 2nd Call")
    .get("/posts/" + postIdNumbers.nextInt(100))
    .check(status.is( expected = 200))
    .check(jsonPath( path = "$.id").saveAs( key = "postId"))
)
.pause(1, 20)
```

Fig. 25. Consultar un recurso específico de la API en Gatling.

```
.exec(http( requestName = "Get Comments for above post - 3rd call ")
  .get("/posts/${postId}/comments")
  .check(status.is( expected = 200))
)
.pause( duration = 2)
```

Fig. 26. Consultar todos los comentarios de un recurso en Gatling.

```
  .exec(http( requestName = "Add Comment to the Post - 4th Call")
    .post( url = "/posts/${postId}/comments")
    .body(StringBody(
      string = """{
          "name": "gatling",
          "email": "gatling@test.com",
          "body": "This is a simple comment"
        }"""))
    .check(status.is( expected = 201))
  )
```

Fig. 27. Añadir un comentario a un recurso en específico en Gatling.

Reporte de pruebas

El reporte generado por Gatling es un reporte en formato HTML, el cual es navegable y contiene mucha información útil para el interesado en realizar las pruebas de carga. Dentro del reporte se cuenta con datos generales de la ejecución, y se tiene además un apartado para revisar la información detallada de cada servicio.

Los datos arrojados por el reporte se muestran a continuación:

Nota: cabe mencionar que solo se muestran las figuras asociadas al reporte detallado de la ejecución del servicio de “Obtener todos los recursos”, dado que para el resto de servicios puede encontrarse algo similar en el reporte HTML.

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                       460 (OK=460    KO=0     )
> min response time                                    49 (OK=49     KO=-     )
> max response time                                   219 (OK=219    KO=-     )
> mean response time                                  100 (OK=100    KO=-     )
> std deviation                                        45 (OK=45     KO=-     )
> response time 50th percentile                       114 (OK=114    KO=-     )
> response time 75th percentile                       150 (OK=150    KO=-     )
> response time 95th percentile                       165 (OK=165    KO=-     )
> response time 99th percentile                       179 (OK=179    KO=-     )
> mean requests/sec                                 4.144 (OK=4.144  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                          460 (100%)
> 800 ms < t < 1200 ms                                  0 (  0%)
> t > 1200 ms                                           0 (  0%)
> failed                                                0 (  0%)
================================================================================
```

Fig. 28. Reporte en consola de Gatling.

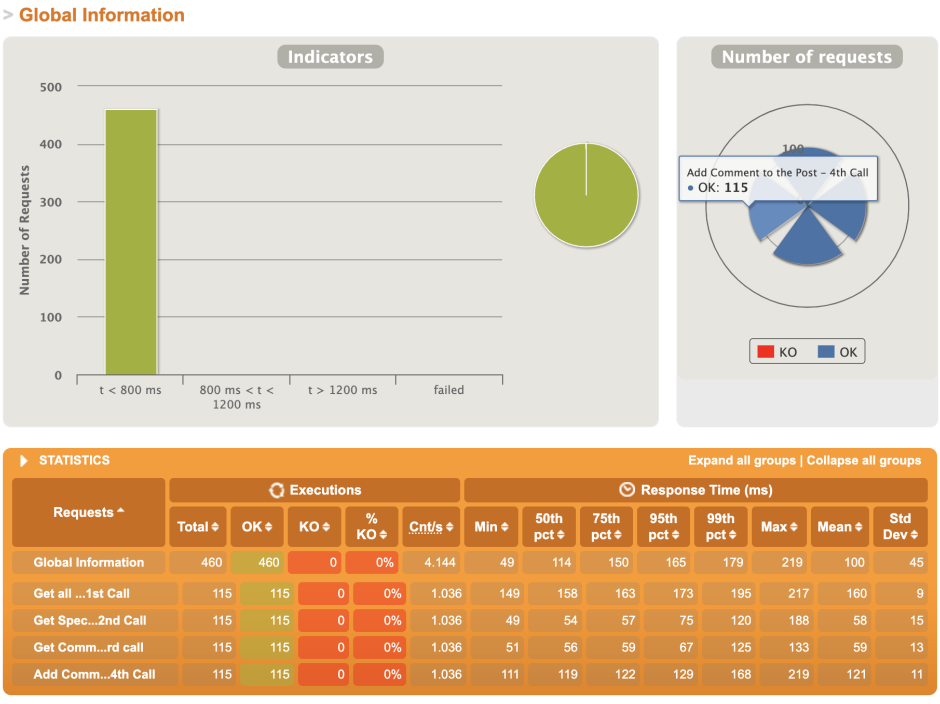


Fig. 29. Resumen general de reporte .hmtl de Gatling.

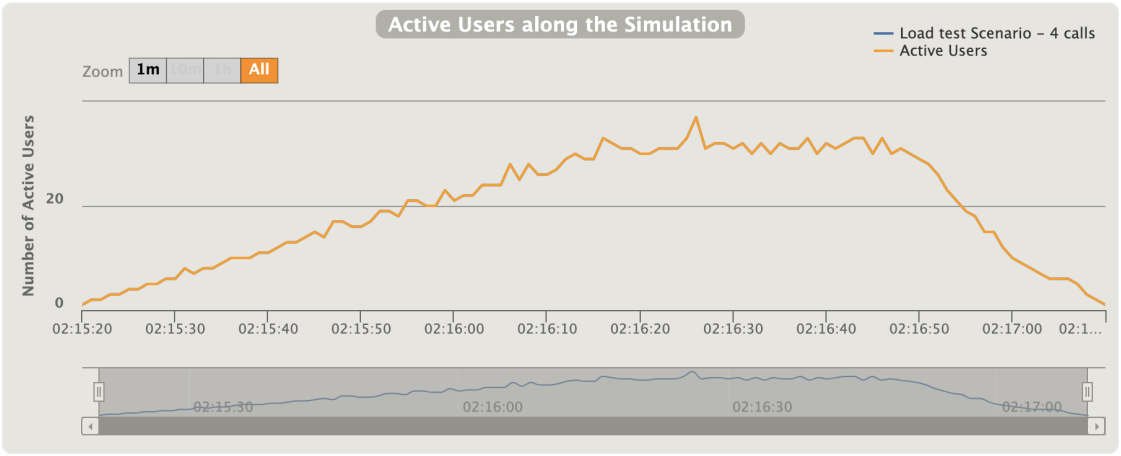


Fig. 30. Reporte de usuarios activos a lo largo de la simulación en Gatling.

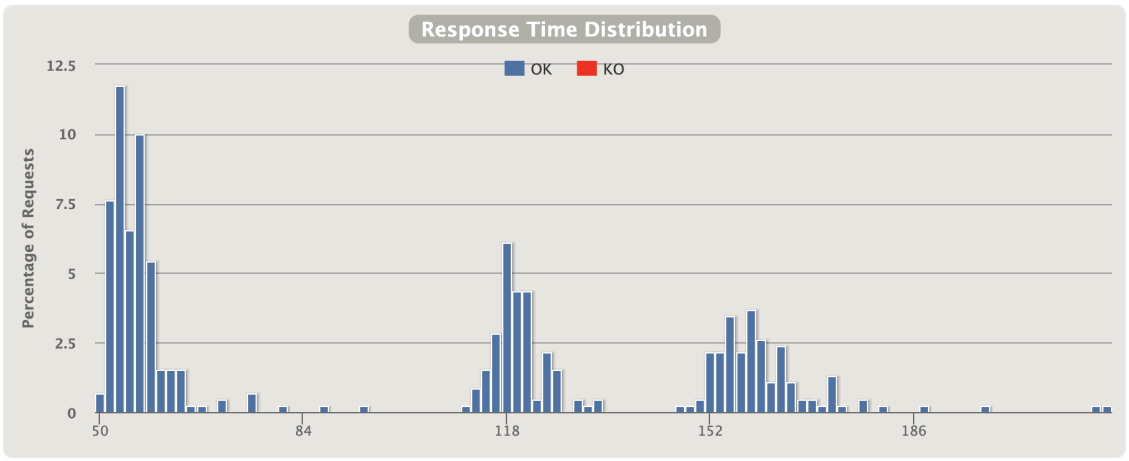


Fig. 31. Reporte de distribución del tiempo de respuesta de las llamadas a la API en Gatling.

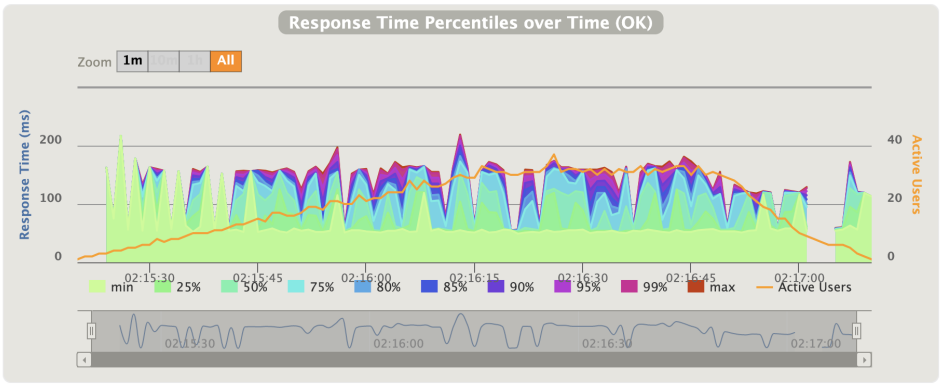


Fig. 32. Percentiles de tiempo de respuesta a lo largo del tiempo en Gatling.

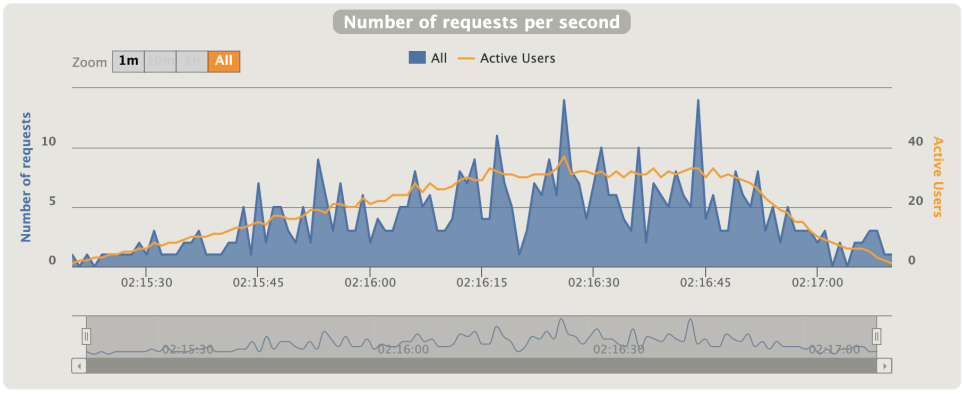


Fig. 33. Número de peticiones a la API por segundo en Gatling.

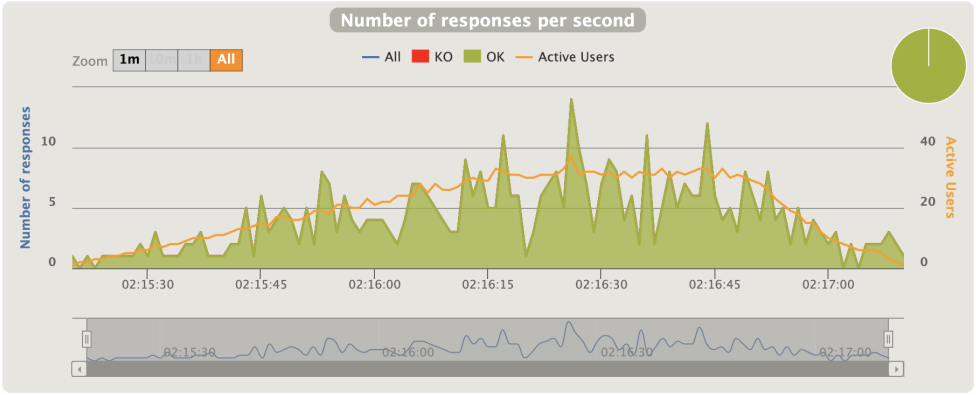


Fig. 34. Número de respuestas de la API por segundo en Gatling.

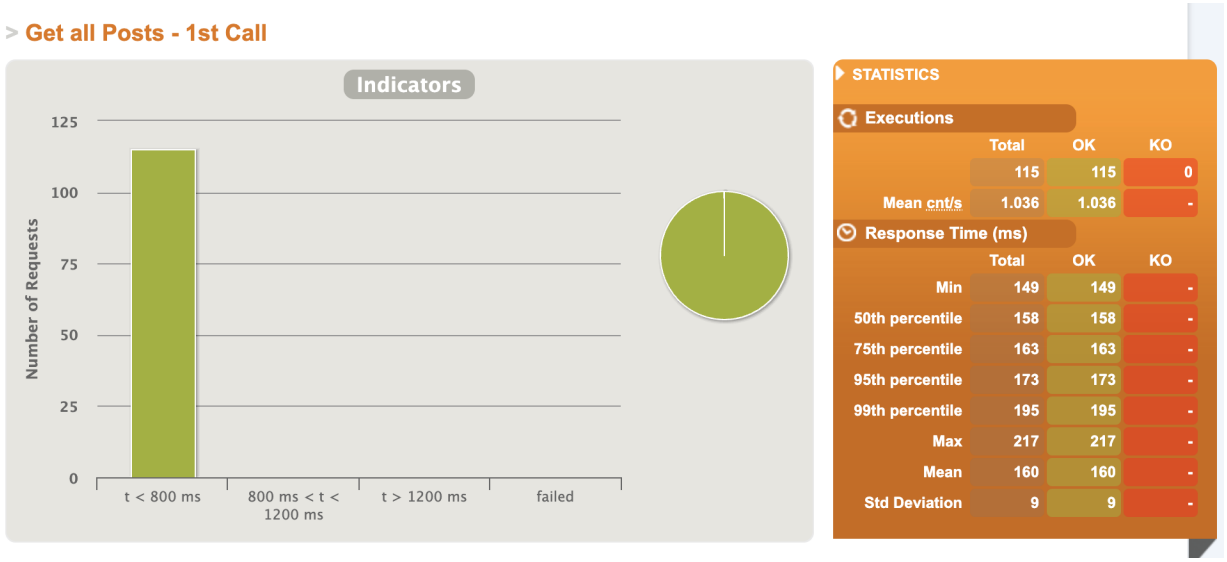


Fig. 35. Resumen de la ejecución del servicio de Obtener todos los mensajes en Gatling.

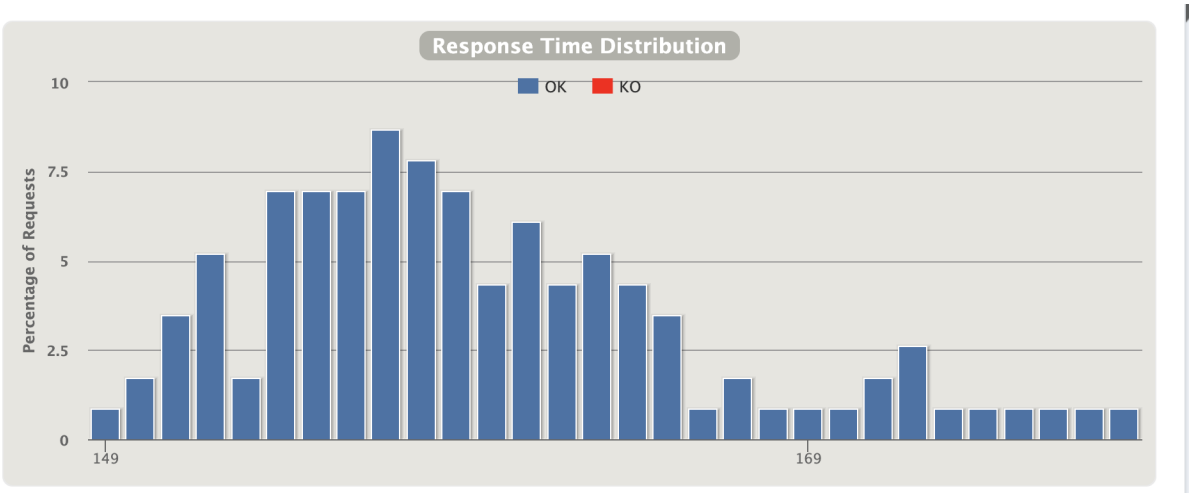


Fig. 36. Distribución de los tiempos de respuesta al ejecutar el servicio de Obtener todos los mensajes en Gatling.

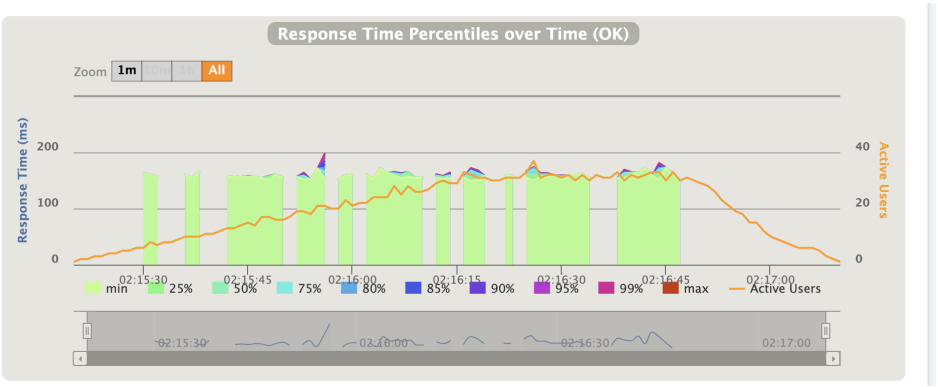


Fig. 37. Percentiles de tiempo de respuesta a lo largo del tiempo al ejecutar el servicio de Obtener todos los mensajes en Gatling.

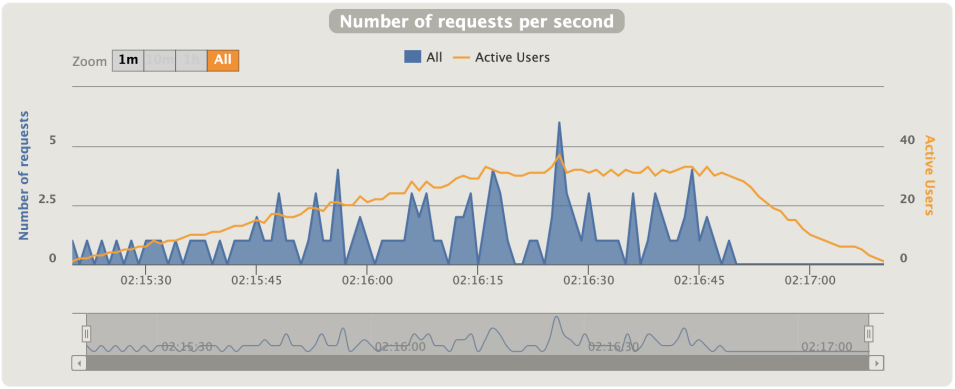


Fig. 38. Número de peticiones por segundo al ejecutar el servicio de Obtener todos los mensajes en Gatling.

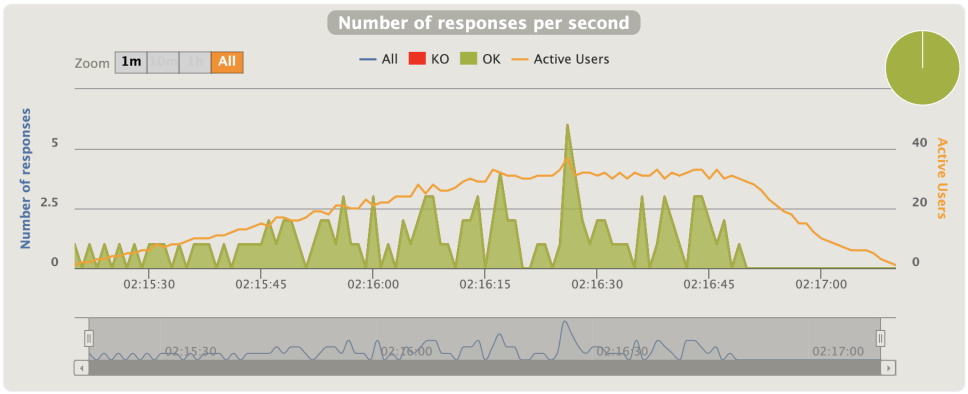


Fig. 39. Número de respuestas por segundo al ejecutar el servicio de Obtener todos los mensajes en Gatling.

Fig. 40. Tiempo de respuesta del servicio de Obtener todos los mensajes vs el global de tiempo de respuesta en Gatling

VI. COMPARATIVA DE HERRAMIENTAS

Junto a los referentes del área de Calidad de la compañía, se generó un instrumento para realizar la comparativa de las tres herramientas con el fin de tomar una decisión objetiva, donde no se evidenciaran preferencias que pudieran afectar la elección final de una u otra solución a la problemática que se tenía: ¿desde qué herramienta sería más factible realizar pruebas de carga haciendo uso de scripts? Los criterios que se muestran en la Tabla 1 fueron los seleccionados para tal comparación, se consideró que en estos se aborda gran parte de lo básico que debe tenerse en cuenta para realizar la elección de una u otra herramienta enfocada en pruebas de carga vía scripting.

Los atributos seleccionados a tener en cuenta son definidos o clarificados dentro de la misma tabla, donde en la primera columna se cuenta con el nombre del criterio a evaluar y en la segunda, una definición o descripción más detallada que permite al encargado de comparar las herramientas, tener mayor claridad a la hora de evaluar uno u otro criterio.

Criterios de evaluación	
Criterios	**Definición**
Facilidad de aprendizaje	*Tiempo/Dificultad en aprender a utilizar la herramienta*
Scripting	*Tiempo insumido en realizar el script necesario para una prueba*
Comunidad	*Comunidad existente sobre la herramienta*
Documentación	*Documentación sobre las distintas funcionalidades*
Costo económico	*Costo de insumos necesarios y/o licenciamiento*
Consumo de recursos	*Consumo de recursos de las máquinas generadoras de carga*
Reporting	*Visualización y detalle del reporte de las pruebas ejecutadas*

Tabla 1. Criterios de evaluación dados por Wolox Part of Accenture.

Para la calificación de cada una de las herramientas se le asigna un peso de manera subjetiva y con apoyo de los referentes del área de QA de la compañía. Se tomó en consideración la recomendación del Proceso Jerárquico Analítico (AHP, por sus siglas en inglés) [10], que es un enfoque genérico para la solución de problemas en toma de decisiones complejas multicriterio. El peso de cada característica se asigna con su intensidad de importancia, tomando como referencia los siguientes valores, que representan la relevancia de la característica para realizar una posible selección de las herramientas [10]:

- {1} Igual importancia: Dos actividades contribuyen igualmente al objetivo.
- {3} Importancia moderada: La experiencia y el juicio favorecen ligeramente una sobre otra.
- {5} Importancia fuerte: La experiencia y el juicio favorecen fuertemente una sobre otra.
- {7} Importancia muy fuerte: Se favorece fuertemente una actividad y su predominio se demuestra en la práctica.
- {9} Importancia absoluta: La importancia de una sobre otra se afirma en su máximo orden posible.

  Los valores {2}, {4}, {6} y {8} son valores intermedios y se usan para representar compromiso entre las prioridades listadas. Las características a evaluar se describen seguidamente. El número entre llaves es la intensidad de importancia que se asignará en cada caso.

Además, para este proceso de definición de relevancia de criterios y calificación de los mismos en las herramientas que se compararon, se toma como ejemplo base el proceso realizado en el artículo *Comparación de las características de algunas herramientas de software para pruebas de carga* [11]. Para obtener el total de puntos de la comparación, se realizará la sumatoria de la multiplicación del peso del criterio por el valor de este mismo asignado a la herramienta. Luego de haber realizado tal proceso se tiene que:

Evaluación		Puntaje		
Criterios	Peso	K6	Gatling	Artillery
Facilidad de aprendizaje	9	6	5	7
Scripting	7	7	6	8
Comunidad	6	7	7	5
Documentación	5	5	6	4
Costo económico	2	5	4	3
Consumo de recursos	1	1	1	1
Reporting	7	4	8	6
Puntajes Ponderados		**209**	**224**	**218**

Instrucciones: Seleccionar e insertar un puntaje de 1 a 9 para cada criterio (0=menor valor; 9=mejor valor). El puntaje es multiplicado por el peso para obtener el puntaje ponderado.
No modificar la primer columna (criterios) y calificar las opciones según cada criterio.

Tabla 2. Evaluación con pesos de las herramientas de pruebas de carga.

Como se evidencia en la Tabla 2, donde se muestra la evaluación junto a los pesos de los criterios para la calificación de las herramientas, la que obtuvo mayor puntaje fue *Gatling*, principalmente por su robustez, comunidad, documentación y sobretodo, por lo detallado de su reporte sin la necesidad de tener que realizar integraciones con otras herramientas.

VII. CONCLUSIONES

Para lograr la satisfacción de unos altos estándares de fiabilidad y eficiencia dentro de un desarrollo de *software*, este debería contar con la realización de un proceso de aseguramiento de la calidad que valide y verifique que, tanto los requisitos funcionales como los no funcionales, cumplan unos estándares de calidad que generen confianza y completen las expectativas del cliente. Para el caso puntual de las pruebas de carga, que se encuentran dentro del conjunto de pruebas de rendimiento y a su vez, el rendimiento de un aplicativo es parte importante de los requisitos no funcionales de un sistema, se concluye que estas pruebas permiten la detección de posibles defectos presentes en el producto o solución que se realice que afecten la competitividad de los mismos en el mercado.

Uno de los factores más importantes en un aplicativo o solución, es el índice de respuesta efectiva del mismo, por lo que las pruebas de carga permiten conocer, de manera aproximada, cómo será el comportamiento de este índice de respuesta a través de escenarios simulados y controlados en los que, por medio de peticiones de usuarios virtuales, se dé respuesta a si el desarrollo estará preparado para soportar la carga a la que estará sometido una vez sea lanzado al mercado y en caso tal de que no, se puedan tomar las acciones pertinentes antes de que sea tarde y se deban realizar reprocesos para asegurar el correcto funcionamiento. En este orden de ideas, la aplicación de este tipo de pruebas de manera temprana puede llevar a la disminución de reprocesos, lo que en términos de proyectos se traduce en ahorro de dinero.

La correcta y adecuada selección de una herramienta de carga parte del conocimiento que se tenga de las herramientas disponibles. Por ello, un trabajo de comparación como el que se abordó durante el proceso de prácticas es de suma importancia para evaluar las características del entorno y la aplicación que se dé en un futuro. Cabe mencionar que no hay una herramienta mágica que funcione en todos los escenarios posibles, pero este acercamiento a diferentes herramientas permite conocer en qué casos sería más provechoso hacer uso de una u otra, según el contexto.

Algo que vale la pena destacar, es la facilidad que proveen las herramientas con que pueden desarrollarse los scripts de prueba y que fueron abordadas en este proceso. Esto permite que la creación de una consciencia y una cultura de desarrollo de pruebas no funcionales para el cumpliento de los estándares de calidad de un desarrollo, esté al alcance de todos.

Dado que existen herramientas que permiten que la realización de pruebas esté al alcance de todos los interesados de un proyecto de desarrollo de *software*, se abre un mundo de posibilidades, entre los que están las evaluaciones de rendimiento de un aplicativo de manera temprana, haciendo uso de pruebas de carga a componentes granulares del sistema.

REFERENCIAS

[1] Glinz, M. (2007). On Non-Functional Requirements. 15th IEEE International Requirements Engineering Conference (RE 2007). doi:10.1109/re.2007.45

[2] PRESSMAN. Roger. Ingeniería del Software. Un enfoque práctico. 7ta edición. España: Ed: McGraw-Hill Interamericana. 2010.

[3] ISTQB, International Software Testing Qualifications Board. Level Specialist Syllabus Performance Testing, Versión 2018.

[4] Pathak, A., 2022. Las 27 mejores herramientas de pruebas de rendimiento para usar en 2022. [En línea] Disponible en: <https://kinsta.com/es/blog/herramientas-pruebas-rendimiento/>

[5] Typicode. {JSON} Placeholder, Free Fake API para pruebas y prototipado. [En línea]. Disponible en: <https://jsonplaceholder.typicode.com/>

[6] RedHat. ¿Qué es una API de REST?. [En línea]. Disponible en: <https://www.redhat.com/es/topics/api/what-is-a-rest-api>

[7] Artillery.io, Load & Smoke Testing. [En línea]. Disponible en: <https://www.artillery.io/>

[8] K6, Load Testing for Engineering Teams | Grafana K6. [En Línea]. Disponible en: <https://k6.io/>

[9] Gatling - Professional Load Testing Tool. [En línea]. Disponible en: <https://gatling.io/>

[10] T.L. Saaty. “The Analytic Hierarchy Process”. McGraw-Hill. New York, Estados Unidos. 1980.