UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática – Ingeniería del Software

Trabajo Fin de Grado

Una propuesta para la estimación de costes de computación en la nube en etapas tempranas de diseño

Rubén Martín Sánchez

Junio, 2017

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática – Ingeniería del Software

Trabajo Fin de Grado

Una propuesta para la estimación de costes de computación en la nube en etapas tempranas de diseño

Alumno: Rubén Martín Sánchez
Director: Juan Carlos Preciado Rodríguez

Tribunal Calificador
Presidente: Fernando Sánchez Figueroa
Secretario: José María Conejero Manzano
Vocal: Roberto Rodríguez Echeverría

Agradecimientos

En primer lugar, quiero agradecer a mi familia su apoyo todos estos años, que ha contribuido a que sea quien soy y haya llegado a donde estoy, inculcándome valores como el esfuerzo, perseverancia y responsabilidad, claves en mi desarrollo personal.

También quiero expresar mi gratitud a quienes han hecho posible vivir esta experiencia de cursar mi último año en la Universidad de Extremadura, a los responsables de Relaciones Internacionales de la Universidad de Sevilla, de la Universidad de Extremadura y al personal de Secretaría en ambos centros. Ha sido un curso increíble que para mí ha marcado un antes y después tanto a nivel personal como profesional. Sin su ayuda y apoyo no habría sido posible.

Agradecer también al equipo de Homeria Open Solutions. Mi estancia con ellos me ha aportado visiones muy útiles que he podido emplear en la realización de este trabajo.

Por último y no menos importante, a mi tutor del TFG, Juan Carlos, por haber confiado en mí y haberme dado la oportunidad de realizar este trabajo con él y haberme guiado y apoyado durante todo el proceso. He aprendido muchísimo gracias a él y le estaré siempre muy agradecido.

A todos vosotros, gracias de corazón.

Contenido

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Una de las soluciones más habituales en el despliegue de aplicaciones web es utilizar una infraestructura de computación en la nube o *cloud computing*. Este tipo de infraestructuras suelen proporcionar un alto nivel de disponibilidad que además junto a la escalabilidad de estos sistemas favorece el uso intensivo de grandes volúmenes de datos y un gran número de usuarios concurrentes. Estos factores son variables clave a considerar que determinarán el rendimiento de la aplicación y supondrán necesitar una infraestructura más o menos potente.

Sin embargo, no suele ser hasta la fase de pruebas cuando se tiene una estimación de los costes computacionales de las operaciones que realiza la aplicación. Estos costes frecuentemente no satisfacen los requisitos iniciales del proyecto, o incluso conducen a un incumplimiento del dinero previsto para su mantenimiento en una infraestructura más potente, que sea capaz de satisfacer las necesidades del sistema en unos tiempos de respuesta razonables.

Si en la etapa de diseño del sistema se conociese una estimación de estos costes operacionales, sería posible reorientar la arquitectura del sistema en otra más eficiente, que evitase diseños que son computacionalmente muy costosos. Al mismo tiempo se podría prever cuáles serán los cuellos de botella de la aplicación y qué configuraciones son las más adecuadas en el modelo de dominio para que estos costes al desplegar la aplicación en una plataforma en la nube sean lo más bajos posible.

Ese es el objetivo principal de este trabajo, elaborar un marco teórico a partir del análisis de los costes computacionales en diversos diseños arquitectónicos que permita anticipar dichos costes para otros proyectos software y también determinar posibles buenas prácticas en etapas tempranas de desarrollo para una plataforma determinada.

1. Introducción

Cuando desde la perspectiva del cliente y del desarrollador se habla del desarrollo de un proyecto software, como por ejemplo puede ser una aplicación web, a menudo se hace referencia a los numerosos requisitos que ésta debe cumplir y con ello, por relación directa, a aquellas funcionalidades que se deben implementar.

En la mayoría de proyectos estos requisitos no siempre están claros desde el principio. Y aún en el caso que parezca que sí lo están, probablemente a lo largo del desarrollo del proyecto sufrirán cambios, o bien se incorporarán nuevos requisitos o se eliminarán algunos que existieran previamente.

No existe evidencia de ningún proyecto software de envergadura en el que el conjunto inicial de requisitos se haya mantenido intacto sin modificaciones ni ampliaciones hasta el final del desarrollo.

Y que estos cambios se produzcan no tiene por qué vaticinar el fracaso de un proyecto, más bien suele ocurrir al contrario, si no se efectúan los cambios precisos en aquellos requisitos que lo requieran a lo largo del desarrollo del proyecto, es probable que se comprometa su viabilidad a corto o medio plazo.

Esto sucede porque lo habitual es que surjan imprevistos o contratiempos que no se pueden prever en las etapas iniciales de planificación del proyecto y definición de requisitos. Factores como los plazos de entrega o algún aspecto de las tecnologías empleadas pueden influir considerablemente en la consecución de estos objetivos iniciales.

Si en lugar de una metodología clásica de desarrollo se sigue una metodología ágil, los cambios y ampliaciones en el conjunto de requisitos serán mucho más frecuentes.

No obstante, es importante tener en cuenta que los posibles cambios que pudieran surgir no tienen el mismo coste de aplicarlos en una fase del proyecto u en otra. La siguiente imagen ilustra muy bien esta idea:

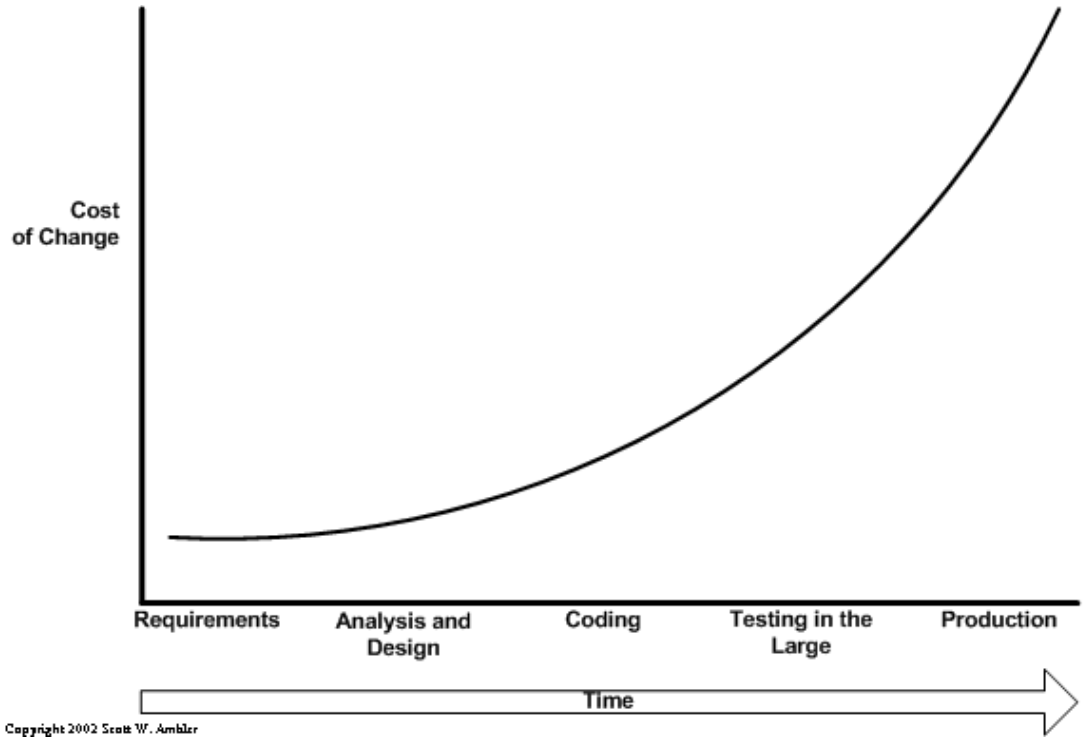


Ilustración 1. Introducción. Coste de los cambios en las fases del desarrollo de un proyecto software.

Tal y como se puede observar en esta gráfica los cambios son mucho menos costosos de aplicar en etapas tempranas de desarrollo. Sin embargo, no siempre esto es posible porque hay información que no se conoce hasta que la aplicación se prueba en un entorno de pre-producción, que suele ser una copia del entorno de producción en el que se desplegará la versión definitiva.

¿Y cuál es esta información desconocida? Normalmente aquella que tiene que ver con algunos requisitos no funcionales.

Durante el desarrollo de un proyecto es habitual priorizar los requisitos funcionales para después en la etapa de pruebas ocuparse de otros no funcionales. Hay algunos de ellos, como por ejemplo que la aplicación esté en varios idiomas, que deben tenerse en cuenta desde la etapa de codificación o implementación, pero hay otros requisitos no funcionales relacionados con el rendimiento de la aplicación que hasta la fase de pruebas no se suelen considerar, como por ejemplo los tiempos de respuesta de una operación o la carga simultánea de usuarios.

Con frecuencia para enfrentarse a unos rendimientos insatisfactorios en esos parámetros se suele optar por comprar o contratar máquinas más potentes, o en su

defecto aumentar la cantidad de máquinas aprovechando su condición de escalabilidad, es decir, se aplicaría una escalabilidad vertical u horizontal para subsanar el problema de rendimiento.

El principal inconveniente de esta solución es que los costes de mantener una infraestructura más potente y compleja pueden ser superiores a los costes de mantenimiento que se habían presupuestado inicialmente.

Estos costes a veces no pueden ser asumidos por la empresa y pueden desembocar en que la aplicación no satisfaga ni la velocidad ni la carga de usuarios reflejadas en los requisitos.

¿Qué es lo que provoca este mal rendimiento? Habría que analizar los cuellos de botella en la aplicación, ver qué funcionalidades son más costosas o rinden peor en situaciones de concurrencia de usuarios.

Estos datos suelen averiguarse en la fase de pruebas, sin embargo, para entonces suele ser demasiado tarde para intentar corregirlos ya que pueden implicar rediseñar de nuevo la arquitectura de la aplicación y eso en la práctica podría suponer comenzar de nuevo la etapa de codificación o implementación.

Una amplia proporción de las funcionalidades que implementa una aplicación se reduce a 4 operaciones, estas operaciones por su relevancia y frecuencia suelen conocerse en la jerga como operaciones *CRUD*. Utilizaré a menudo este término para referirme a ellas a lo largo del documento. Las siglas vienen de la unión de las palabras en inglés **C**reate **R**ead **U**pdate y **D**elete, y hacen referencia a las operaciones más comunes de acceso a datos. Casi todas las funcionalidades de una aplicación van a requerir en algún momento crear algún dato, consultarlo de alguna *fuente,* actualizarlo o eliminarlo. Por algo las aplicaciones web se denominan también sistemas de información web, porque trabajan y manipulan datos, y esas son las operaciones básicas y fundamentales para ello.

A menudo el almacenamiento de estos datos se asocia principalmente a bases de datos relacionales o incluso bases de datos NoSQL, no obstante, no siempre tiene por qué ser así, hay otros tipos de persistencias volátiles que hacen uso de otras fuentes

de datos, como la memoria caché, por ejemplo. En el trabajo se tratarán 2 persistencias volátiles: de sesión y aplicación, que serán explicadas más adelante.

Pues bien, teniendo en cuenta que la mayoría de las operaciones de una aplicación se corresponden con operaciones CRUD, cabe pensar que podría ser muy útil conocer antes de iniciar la etapa de diseño del proyecto cómo se comportan estas operaciones en distintos contextos. Así gracias a diversos análisis se podrían tomar decisiones en una etapa temprana de desarrollo que repercutieran positivamente más adelante en la etapa de pruebas de la aplicación, consiguiendo un mejor rendimiento y ahorrando costes en la fase de producción.

La creación de un marco teórico para analizar el comportamiento de estas operaciones permitiría definir un procedimiento válido para muchos proyectos software.

En una primera fase de dicho marco lo que se hace es aislar las operaciones CRUD y haciendo uso de temporizadores se calculan los tiempos que suponen realizar dichas operaciones. Con el entorno de desarrollo WebRatio y algunos de sus componentes se desarrollarán un conjunto de proyectos para conseguir este propósito. Estos costes temporales se almacenarán en unas hojas de cálculo que, ya en una segunda fase, serán procesadas mediante un algoritmo de clusterización y dará lugar a unos tiempos medios que se utilizarán para extraer conclusiones. En esta última fase se hará uso del software RStudio y del algoritmo K-means que implementa en una de sus librerías.

Con esos tiempos medios también se realizarán gráficas comparativas que permitirán analizar tendencias y extraer patrones de comportamiento.

Ésta es la motivación de este Trabajo Fin de Grado, y que en los siguientes apartados se desarrollará más detalladamente. Al respecto se han publicado dos artículos que se citan en la bibliografía: *A First Step to Cloud Infrastructure Cost Estimation in Early Stages of Web Development* y *Hacia una propuesta de estimación de costes de producción desde etapas tempranas del desarrollo Web*.

1.1. Origen del trabajo

El origen de la idea para este trabajo se remonta al mes de septiembre, cuando llegué nuevo a Cáceres y poco después de comenzar el curso y presentarme, coincidí con Juan Carlos. Entonces en una breve conversación informal estuvimos hablando sobre los motivos que me habían llevado a participar en el Sistema de Intercambio de Centros Universitarios Españoles (SICUE) y escoger la Universidad de Extremadura como universidad de destino en la que terminar la carrera. Le comenté que estaba interesado en realizar el TFG este curso y aún no tenía muy claro la temática sobre la que realizarlo.

Unos días más tarde recibí un correo para reunirnos en su despacho y analizar posibles temas en los que pudiera estar interesado. Así fue como él me propuso la idea de desarrollar un marco teórico para anticipar los costes computacionales de un proyecto software desplegado en la nube. Como contexto me presentó la herramienta WebRatio, por aquel entonces desconocida para mí, y que me dio muy buenas impresiones ya que no había trabajado antes con una herramienta parecida y tan potente.

Aquella propuesta me pareció muy interesante, si bien entonces no comprendía demasiado bien su alcance y posibilidades que ofrecería el estudio y desarrollo de este marco teórico. Acepté su propuesta y decidí realizar este trabajo sobre aquella idea. He de reconocer que en un principio estaba un poco intranquilo porque si bien entendía lo que se quería conseguir, aún no sabía muy bien el cómo. Gracias a su ayuda muchas de estas incertidumbres poco a poco se fueron despejando, fui entendiendo mejor los objetivos y se fue avanzando en el desarrollo del proyecto.

A partir de aquel día comenzaría un largo proceso de investigación cuyos resultados, y caminos y alternativas que se han seguido para llegar hasta ellos, quedan reflejados en este trabajo.

2. Objetivos

El objetivo de este trabajo es el de elaborar un marco teórico base que permita obtener una primera aproximación de los costes computacionales de las operaciones más comunes en una plataforma determinada.

De esta forma se desarrolla una posible metodología para intentar dar respuestas a preguntas como:

- ¿Qué tipo de persistencia me conviene más para esta cantidad “x” de atributos: volátil o en base de datos?
- ¿Cuánto tiempo aproximadamente llevará realizar esta operación “y”?
- ¿Cuál es la operación CRUD más costosa del sistema?
- ¿Conviene dividir esta entidad en varias entidades con menos atributos?

Y ser capaz de responderlas sin haber previamente realizado la fase de desarrollo de la aplicación. Estas son solo algunas de las preguntas que permitiría responder un análisis previo de la puesta en práctica de este marco teórico. Conocer una estimación de los costes operacionales también permite poder tomar mejores decisiones en una etapa temprana de desarrollo.

Es posible prever qué cuellos de botella puede haber en la aplicación e incluso detectar si serán necesarios más recursos para que la aplicación funcione correctamente y cumpla requisitos no funcionales como tiempos de respuesta, cargas de usuarios, entre otros.

Este marco teórico debe ser lo suficientemente genérico como para que no dependa de ninguna tecnología particular, habida cuenta que de ser así podría quedar obsoleto en un periodo de tiempo no muy largo. No obstante, es importante tener en cuenta que habrá tecnologías más favorables que otras para aplicar este marco. Algunos de los pasos que se seguirán en este trabajo con el fin de obtener mediciones pueden ser más difíciles de poner en práctica en algunas tecnologías que en otras.

En este caso concreto se hará uso de la plataforma WebRatio para el desarrollo del sistema y de Amazon Web Services para su despliegue.

Se ha elegido WebRatio y no otros frameworks populares de desarrollo por la sencillez que ofrece al dividir en componentes muchas de las operaciones que interesan. Es una ventaja en este caso su arquitectura dirigida por modelos o Model-driven. Otras plataformas con una arquitectura diferente requerirán otros métodos para aislar y encapsular las operaciones que serán objeto de estudio en este trabajo.

Por otra parta la plataforma AWS de Amazon es una de las PaaS (*Platform as a Service*) más populares, empleadas y robustas en la actualizad, lo que hace que sea una buena alternativa para realizar las pruebas de rendimiento.

3. Antecedentes

Existe la percepción de que en el desarrollo de un proyecto software es la fase de implementación la etapa más costosa y larga de todo el proyecto. Sin embargo, la realidad es que, siempre que el proyecto concluya de forma exitosa, sea el mantenimiento del producto software lo que suponga a medio y largo plazo la mayor parte de los costes. Este mantenimiento a menudo requerirá correcciones de errores, ampliaciones, etc. Pero, además, y no menos importante, también incluye la infraestructura sobre la que se aloja la aplicación.

Ante la realidad de que cada año hay más y más usuarios y dispositivos con acceso a Internet, se encuentra la necesidad cada vez más imperante de que los sistemas web sean escalables y soporten una carga considerable y variable de usuarios.

Las empresas son conscientes de este hecho, y cada vez dedican más recursos a desarrollar modelos con los que construir sistemas más eficientes que a medio y largo plazo supongan un desembolso mucho menor en la infraestructura sobre la que se despliegan estos sistemas. Cuantas más prestaciones tenga esa infraestructura más caro será, y por ende se persigue obtener buenos rendimientos y tiempos de respuesta en infraestructuras que puedan ser asumibles por la empresa y sus costes afecten lo menos posible al margen de beneficios.

Hay una gran competencia entre las numerosas tecnologías de desarrollo que pueden utilizarse para crear estos sistemas, y gran parte de las mejoras de rendimiento provienen de ahí, de que se haya utilizado una tecnología que sea más eficiente que otras. Pero con ello no se está garantizando que esa tecnología se esté utilizando de la forma más eficiente posible, porque también el diseño y la arquitectura de la aplicación, que ya no dependen exclusivamente de la tecnología, sino que involucran al desarrollador, afectarán al rendimiento de ésta.

Este trabajo se enfocará en esa segunda parte. La metodología que se propone para conseguirlo se explica en el siguiente apartado.

4. Metodología

El desarrollo del trabajo ha sido un proceso bastante laborioso que no ha estado exento de contratiempos y altibajos. Es un aspecto que no debe resultar extraño teniendo en cuenta que se trata de un proyecto de investigación, en cuya naturaleza es habitual escoger alternativas que finalmente pueden descubrirse como no válidas o no aptas para el propósito buscado.

En este apartado se pretende establecer una cronología de los principales acontecimientos y fases en las que se ha dividido el proyecto, así como exponer los imprevistos que han ido sucediendo y han obligado a cambiar el rumbo que se estaba siguiendo hasta el momento.

- FASE 1: Mediados de septiembre – mediados de octubre

Esta primera fase se enmarca en el contexto de familiarización con la plataforma WebRatio. Principalmente estas semanas se dedicaron a instalar y configurar el entorno de desarrollo, explorar los distintos componentes que ofrecía, realizar proyectos de ejemplo y planificar los próximos pasos. En el apartado 5.1. se ofrece más información sobre esta fase.

- FASE 2: Mediados de octubre – mediados de noviembre

En este periodo se investiga una tecnología para realizar las pruebas de rendimiento: Apache JMeter. Se realizan en WebRatio los 4 proyectos correspondientes a las 4 operaciones CRUD. Los resultados obtenidos con esta tecnología no son del todo convincentes y por ello se descarta su uso para realizar las mediciones. Más información en el apartado 6.1.

- FASE 3: Finales de noviembre

En esta etapa se contempla la alternativa de modificar los componentes de WebRatio para que estos hagan además de contadores temporales y así no sea necesario el uso de tecnologías externas para medir el rendimiento. Se consigue la modificación mostrando los tiempos por consola, pero no se halla la vía para extraer y exportar

automáticamente esos datos a una base de datos u hoja de cálculo. En el apartado 6.2. se expone más detalladamente esta idea.

- FASE 4: diciembre de 2016

De este modo a comienzos de mes se prueba el uso de la Time Unit (apartado 5.2.), un componente propio de WebRatio cuya máxima precisión es en milisegundos, y se hace uso de la Excel Unit (apartado 5.3.) para exportar los resultados a una hoja de cálculo.

A finales de mes se desarrolla un script en RStudio que permite el análisis de los datos de una hoja de cálculo aplicando el algoritmo de K-means.

- FASE 5: enero de 2017

Se realiza el análisis en RStudio de todos los archivos Excel exportados por las aplicaciones realizadas en WebRatio y se elaboran las primeras gráficas comparativas tomando los tiempos medios y haciendo uso de Excel. Más información en el apartado 5.4.

- FASE 6: febrero – mediados de marzo

Hasta este momento las pruebas y las hojas de cálculo exportadas eran de la aplicación desplegada en local, durante este periodo se repiten las pruebas con las distintas versiones de la aplicación desplegadas en la nube de Amazon Web Services. Se hacen pruebas de un único usuario y también pruebas de 5 y 10 usuarios concurrentes. Posteriormente se analizan los datos con RStudio y se representan en Excel.

En esta fase hay numerosos imprevistos que vienen detallados en los anexos IV, V y VI de este trabajo.

- FASE 7: segunda mitad de marzo – abril

Algunas incoherencias en las gráficas obligan a repetir el análisis para ciertas configuraciones en RStudio, se modifican algunos parámetros del algoritmo y los resultados devueltos comienzan a ser más coherentes. Sigue habiendo incoherencias en la comparación de 1 usuario frente a 5 usuarios concurrentes, tal vez debido al estado de la máquina virtual de Amazon en el momento de hacer las pruebas de concurrencia.

- FASE 8: mayo

Comienza la recopilación y ensamblado de la información del proyecto de todos los meses anteriores. Se inicia la redacción del TFG según el formato requerido en la normativa.

Un diagrama de todas estas fases y sus tareas más relevantes es el siguiente:

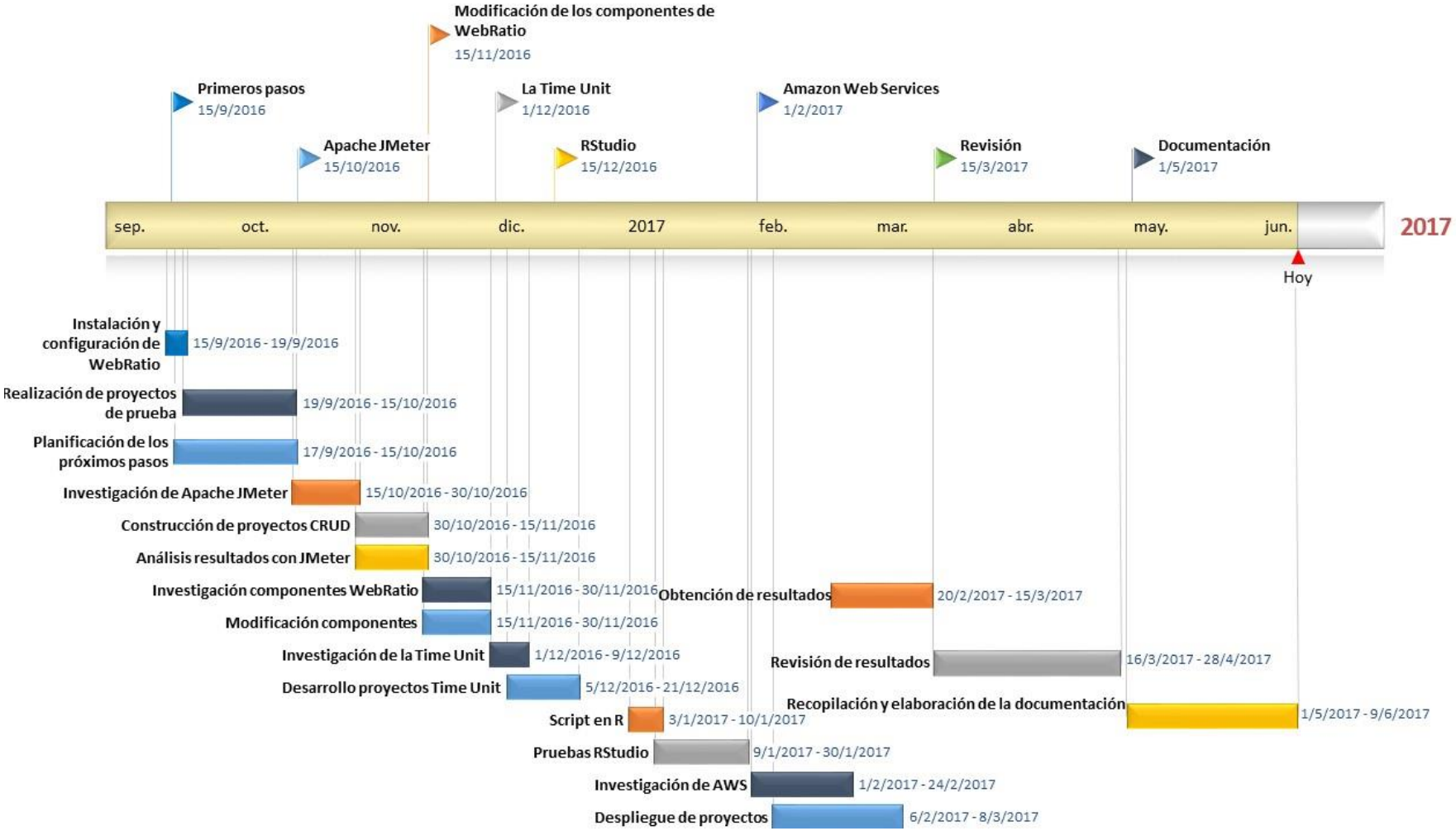


Ilustración 2. Diagrama de las fases del proyecto

4.1. Marco teórico

Este apartado intenta profundizar en ese marco teórico presentado anteriormente con el objetivo de anticipar los costes computacionales en etapas tempranas de desarrollo.

En primer lugar, es necesario identificar las operaciones básicas que van a constituir la aplicación, en la mayoría de casos estas operaciones serán las operaciones CRUD.

A continuación, será necesario utilizar una serie de mecanismos para hacer que estas operaciones estén aisladas, lo más atómicas posible, con el fin de establecer contadores temporales antes y después de la operación y averiguar así cuánto tiempo ha llevado realizarla.

El tiempo de la medida será el resultado de restar ambos contadores, y ese valor será necesario almacenarlo en una base de datos con tal de poder ser objeto de estudio al final del proceso. En este trabajo la magnitud está en milisegundos, no obstante, sería posible aumentar la precisión a otra magnitud, como los nanosegundos.

Junto a dicho valor habrá que almacenar otros conjuntos de valores que identifiquen a esa medida. Entre ellos puede estar un identificador numérico que contabilice esa muestra entre todas las muestras realizadas, pero además también deben estar otros valores relacionados con las características específicas, estas son el tipo de persistencia y la operación concreta del CRUD.

Se pueden distinguir tres tipos de persistencia generales en una aplicación web: la persistencia en base de datos, la persistencia volátil de sesión y la persistencia volátil de aplicación. En este trabajo se tratan todas ellas y se establecen gráficas comparativas para estudiar su comportamiento en distintas configuraciones de máquinas virtuales.

Es muy importante tener un conjunto lo suficientemente amplio de muestra. Esto quiere decir que debe ser representativo, por ejemplo, en este trabajo se han considerado 2000 repeticiones para los casos de un único usuario y 250 para los casos concurrentes. De esta forma en el estudio analítico posterior se podrán deducir comportamientos comunes y similares en forma de patrones.

Estas pruebas de rendimiento convienen que se hagan en entornos lo más parecidos posible al entorno final de producción, quiere decir, si se pretende conseguir una estimación de cómo funcionaría la aplicación en una plataforma determinada con unas características concretas, lo mejor es que las pruebas se hagan sobre dicha plataforma. Incluso los resultados pueden ser reutilizados para otros proyectos que se desplieguen en esa misma plataforma.

Se debe procurar testar el máximo número de variables posible, es decir, hay que probar cada una de las operaciones CRUD con los 3 tipos de persistencia, tomando entidades de prueba que tengan un número variable de atributos. Un ejemplo de prueba de rendimiento sería ver el comportamiento de la operación CREATE con tipo de persistencia volátil de aplicación para una entidad con 40 atributos.

Al final los datos que se obtendrían almacenados vendrían a responder a un modelo de datos muy similar a este:

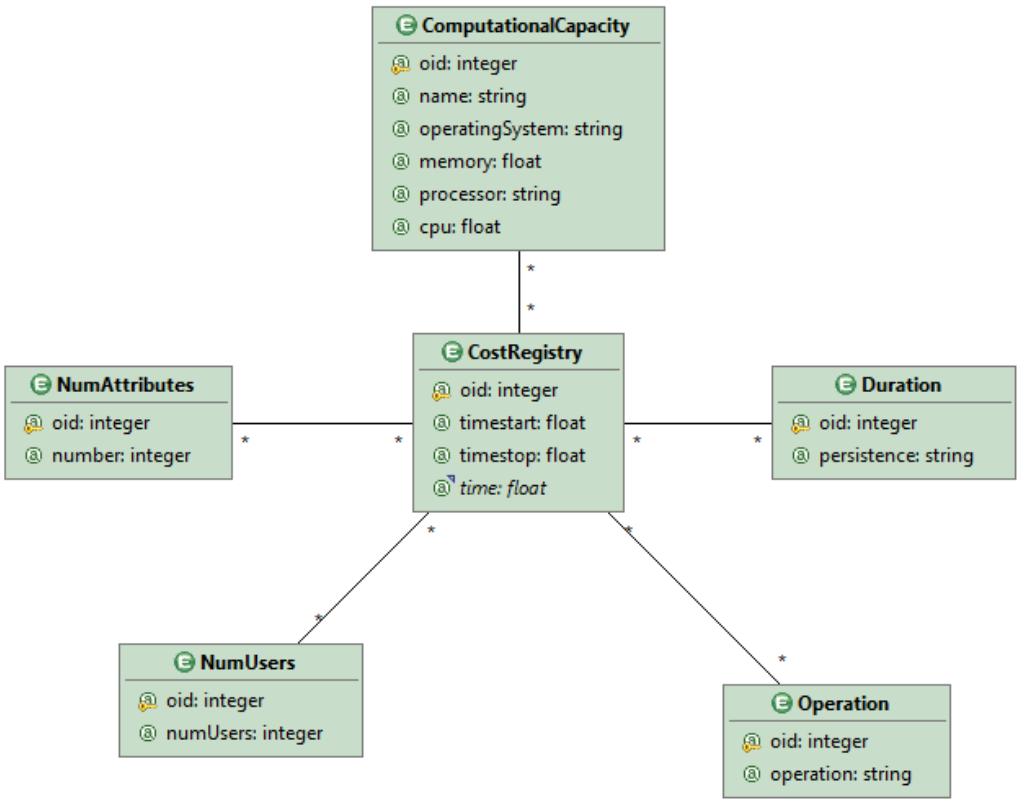


Ilustración 3. Metodología. Modelo de datos de estudio

Las variables serían el número de atributos, el tipo de operación, el tipo de persistencia, la medida del tiempo *time* como derivada de la diferencia de *timestop* y

y *timestart,* el número de usuarios concurrentes y los detalles técnicos de la máquina sobre la que se ejecutarán las pruebas (memoria, CPU…). Habría cinco dimensiones a modo de un cubo ROLAP.

Los resultados de los tiempos que se almacenen en la base de datos deben ser exportados a documentos en formato de hoja de cálculo para su posterior estudio. En este trabajo para cada configuración de atributos (1, 10, 20, 30, 40 y 50) se ha generado un documento Excel en la que aparecen los atributos *oid*, *time*, *operation* y *persistence*.

Las pruebas no deben limitarse a un único usuario, también deben contemplar usuarios concurrentes. En el trabajo se han contemplado 5 y 10 usuarios concurrentes.

Hasta aquí llegaría una primera fase del marco teórico. La segunda fase tiene que ver con el estudio de todos los datos que se han generado.

Para ello será necesario procesar las distintas hojas de cálculo y aplicar un algoritmo de clusterización, con dos objetivos principales:

- Agrupar los valores y descartar aquellos que sean excepcionales (*outliers*) y no sean representativos.
- Obtener un valor medio del coste computacional que ha significado dicha operación con ese tipo de persistencia para esa cantidad de atributos.

Con esos valores medios obtenidos se podrán representar en una última fase gráficas comparativas que permitan analizar y extraer conclusiones sobre el comportamiento de estas operaciones en la plataforma desplegada.

Todo ello permitirá tomar mejores decisiones en etapas de diseño de futuras aplicaciones para esa plataforma. En el trabajo se han realizado las pruebas con dos máquinas virtuales de Amazon Web Services, cuyas especificaciones se expondrán más adelante.

5. Implementación y desarrollo

5.1. Introducción a la plataforma de desarrollo WebRatio

Ilustración 4. Introducción a la plataforma de desarrollo WebRatio. Logo

El framework WebRatio Web Platform es un entorno de desarrollo que persigue una alta productividad y un mejor mantenimiento de sistemas de información web, basándose en una arquitectura dirigida por modelos (Model-driven architecture o MDA).

Esto hace que el desarrollo web utilizando esta plataforma sea muy visual, con un diseño basado en prototipos es más sencillo realizar modificaciones.

El entorno está basado en el IDE de Eclipse lo cual hace que tenga retro compatibilidad con todos sus plugins. El código generado es Java estándar, luego las aplicaciones pueden desplegarse en cualquier servidor de aplicaciones Java.

5.1.1. Estándares: IFML

WebRatio introdujo en el desarrollo de sus aplicaciones un nuevo estándar denominado IFML (Interaction Flow Modeling Language), respaldado por la OMG (Object Management Group) desde 2013.

El resultado es la evolución del estándar WebML. Entre los objetivos de este estándar están los de aislar los aspectos de la implementación de los aspectos de las interfaces gráficas de usuario y hacer una especificación formal de los componentes que constituyen el front-end de una aplicación.

De esta forma se podrán distinguir componentes, eventos, acciones, flujos de navegación y flujos de datos.

5.1.2. Realización de proyectos en WebRatio

Ejecutando la herramienta la vista principal del entorno luce así:

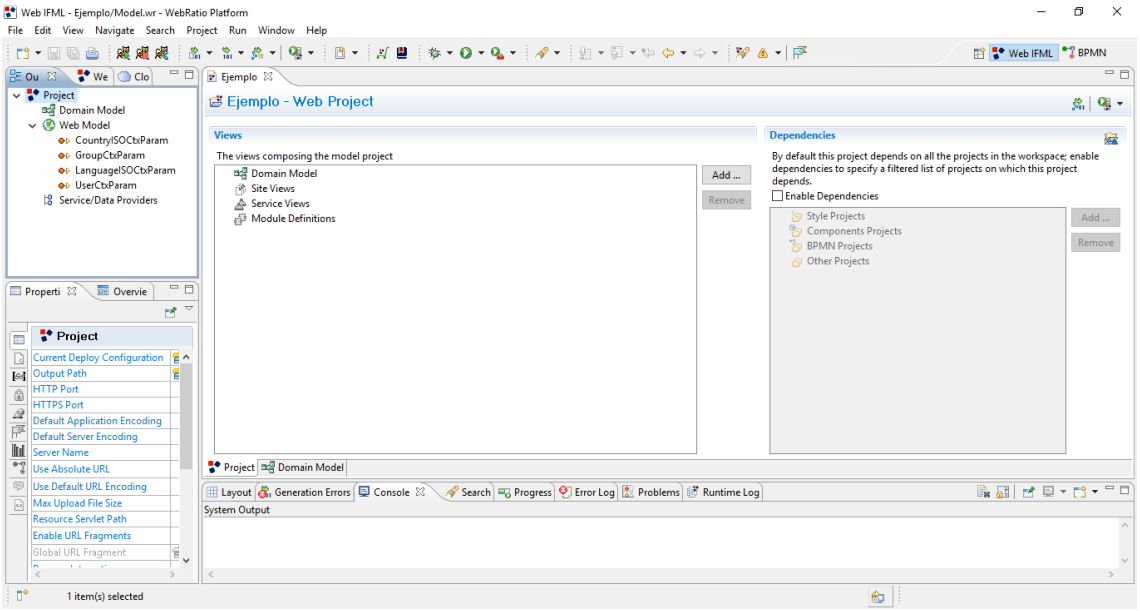


Ilustración 5. Introducción a la plataforma de desarrollo WebRatio.Ventana principal del entorno WebRatio

Para crear un proyecto habrá que ir a File > New > Web Project y establecer un nombre del proyecto.

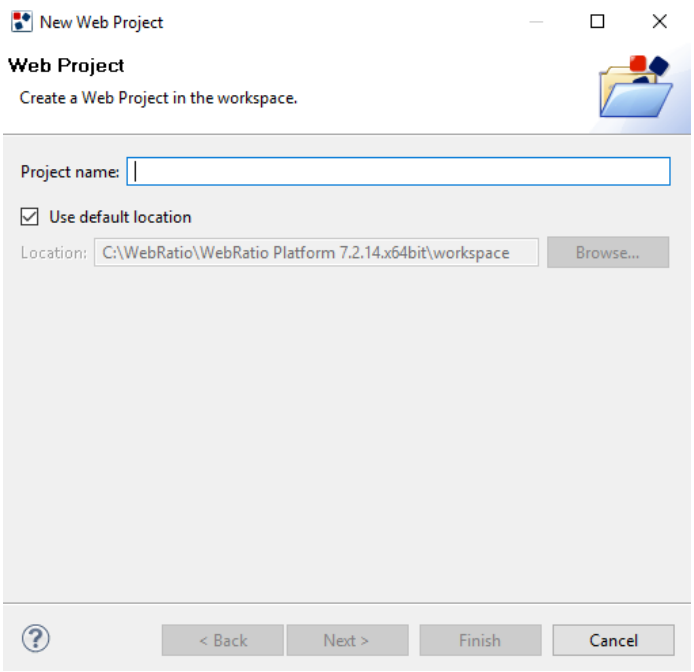


Ilustración 6. Introducción a la plataforma de desarrollo WebRatio. Creación de un nuevo proyecto

También el proyecto tiene una serie de vistas:

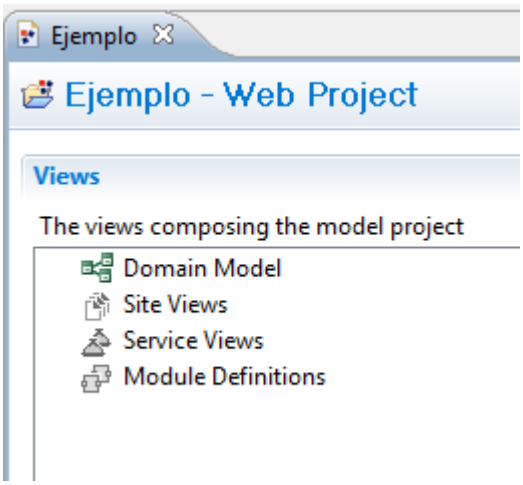


Ilustración 7. Introducción a la plataforma de desarrollo WebRatio. Vistas de un proyecto

Por defecto, el modelo de dominio incluido en el proyecto base tiene 3 entidades: User, Group y Module imprescindibles para el acceso de usuarios por roles a zonas restringidas por autenticación. Estas clases no se pueden eliminar del modelo de datos:

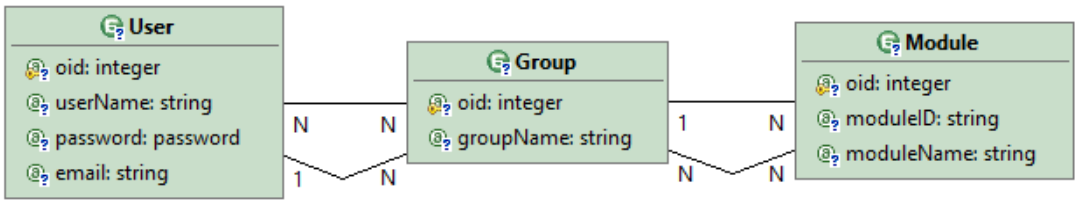


Ilustración 8. Introducción a la plataforma de desarrollo WebRatio. Entidades del modelo de dominio por defecto

Las distintas vistas se definirán como Site Views, también es posible definir servicios programables desde Service Views o encapsular en Module Definitions flujos de operaciones que puedan ser reutilizadas.

5.1.3. Componentes principales de WebRatio

En la vista Site Views aparecen los componentes y flujos que se pueden utilizar para el modelo IFML de la aplicación.

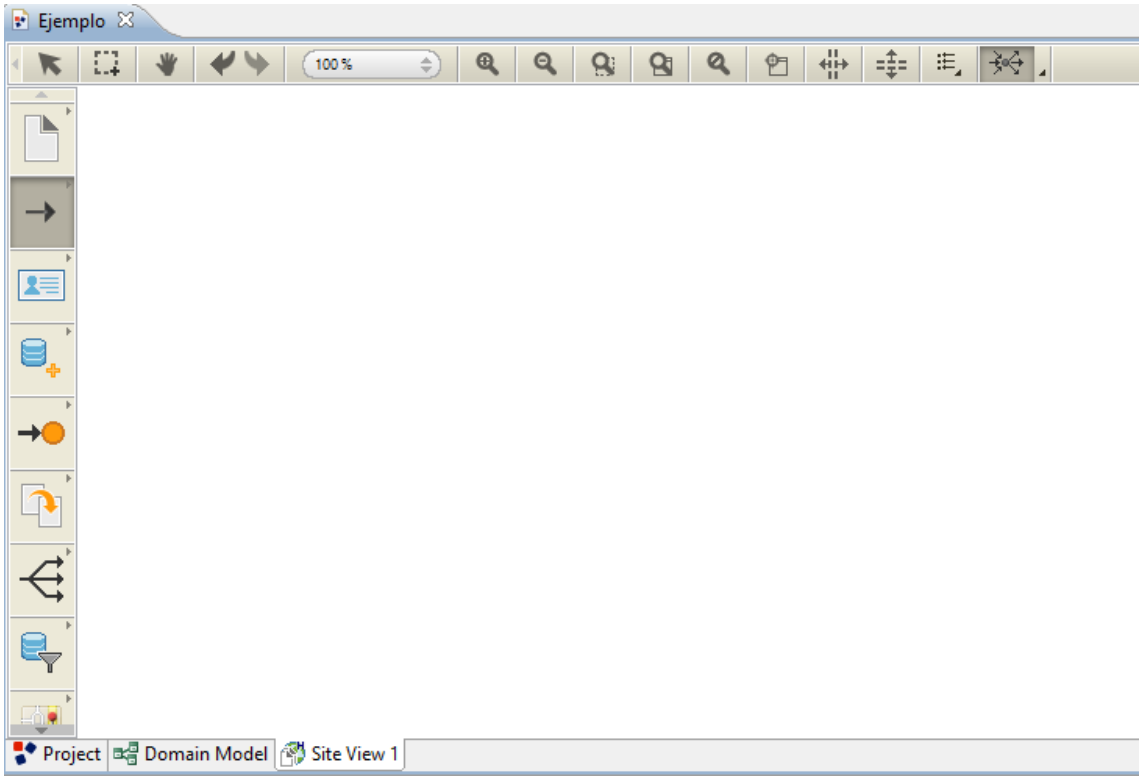


Ilustración 9. Introducción a la plataforma de desarrollo WebRatio. Lienzo de representación IFML

Se pueden encontrar los siguientes componentes:

Ilustración 10. Introducción a la plataforma de desarrollo WebRatio. Containers

Para definir las páginas que compondrán la aplicación web. Permite la agrupación de ellas y la creación de páginas maestras o master page.

Ilustración 11. Introducción a la plataforma de desarrollo WebRatio. Flows

Para definir la interacción entre los componentes con flujos de datos o de navegación, y flujos OK y KO para controlar el funcionamiento del componente.

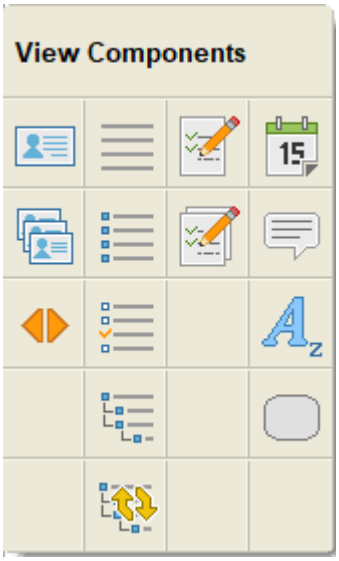


Ilustración 12. Introducción a la plataforma de desarrollo WebRatio. View Components

Para la creación de vistas, listados y formularios de entidades presentes en el modelo de dominio.

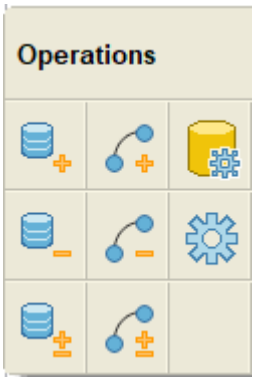


Ilustración 13. Introducción a la plataforma de desarrollo WebRatio. Operations

Para realizar operaciones de inserción, borrado y actualización de entidades presentes en el modelo de datos.

Ilustración 14. Introducción a la plataforma de desarrollo WebRatio. Session Components

Para controlar variables de sesión como el usuario logueado en ese momento, por ejemplo, y definir enlaces de login/logout.

Ilustración 15. Introducción a la plataforma de desarrollo WebRatio. Service Components

Estos componentes se utilizan para añadir servicios web de terceros que devuelven resultados en XML o JSON. También permiten la creación de tareas programables o *Jobs*.

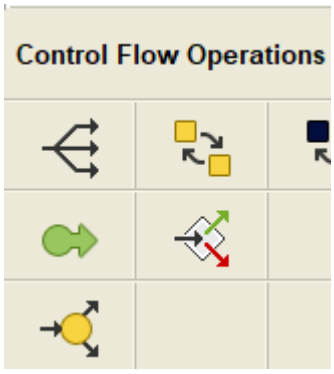


Ilustración 16. Introducción a la plataforma de desarrollo WebRatio. Control Flow Operations

Para controlar flujos en forma de switch, bucles o condicionado al valor de una variable.

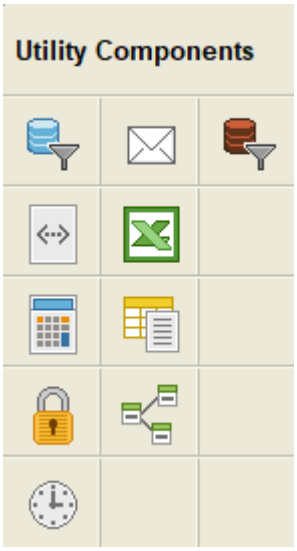


Ilustración 17. Introducción a la plataforma de desarrollo WebRatio. Utility Components

Para realizar consultas sobre una entidad, enviar un correo electrónico usando un servidor SMTP, ejecutar código a partir de un script escrito en Groovy, importar o exportar datos desde un Excel, realizar alguna operación matemática, una consulta en SQL a la base de datos, un generador de contraseñas u obtener el tiempo del sistema en distintos formatos.

5.2. Proyectos de rendimiento con Time Units en WebRatio

A continuación, se va a exponer el proceso seguido para obtener las mediciones que se han utilizado para realizar el estudio experimental.

5.2.1. Justificación

Para el objeto del estudio desde un principio se ha perseguido obtener las mediciones temporales de cada operación para posteriormente analizar sus resultados en conjunto. Se barajó, aunque sin éxito, el uso de herramientas como Apache JMeter y también la modificación de los componentes CRUD de WebRatio.

De este modo se buscó una nueva alternativa, que consiste en utilizar un componente propio de WebRatio, la Time Unit, como un contador de los milisegundos que tarda en realizarse la operación analizada.

5.2.2. Creación de los 4 proyectos CRUD

Para cada operación CRUD se desarrolló un proyecto que calculara dichos tiempos y los almacenase en una base de datos.

En el caso de la operación CREATE se diseñó un modelo IFML en el que la operación era lanzada desde una *No Operation Unit* y de este modo mediante una *Loop Unit* y los temporizadores de la *Time Unit* se calculan los tiempos de varias iteraciones de la misma operación. Por un lado, se creaba una instancia de la clase de prueba *Cost* y por otro se creaba un dato de registro temporal de la entidad *CostRegistry* que se almacenaba en la base de datos:

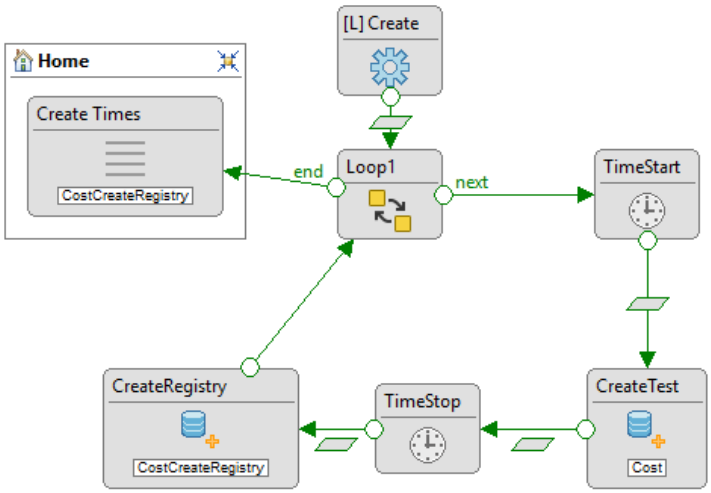


Ilustración 18. Proyectos de rendimiento con Time Units en WebRatio. Create

El modelo de dominio para este diagrama es el siguiente:

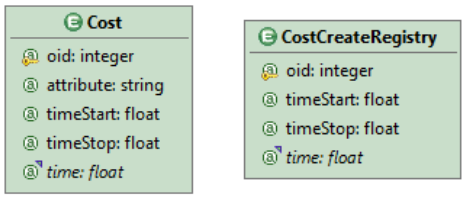


Ilustración 19. Proyectos de rendimiento con Time Units en WebRatio. Create. Modelo de datos

La entidad *Cost* aparece como entidad persistente en base de datos, pero este tipo de persistencia iba variando según se quería probar una configuración volátil de sesión o de aplicación.

Para la operación READ el diagrama IFML es muy similar al anterior, pero en vez de tener una segunda *Create Unit* tiene una *Selector Unit* para simular el comportamiento de la operación de lectura:

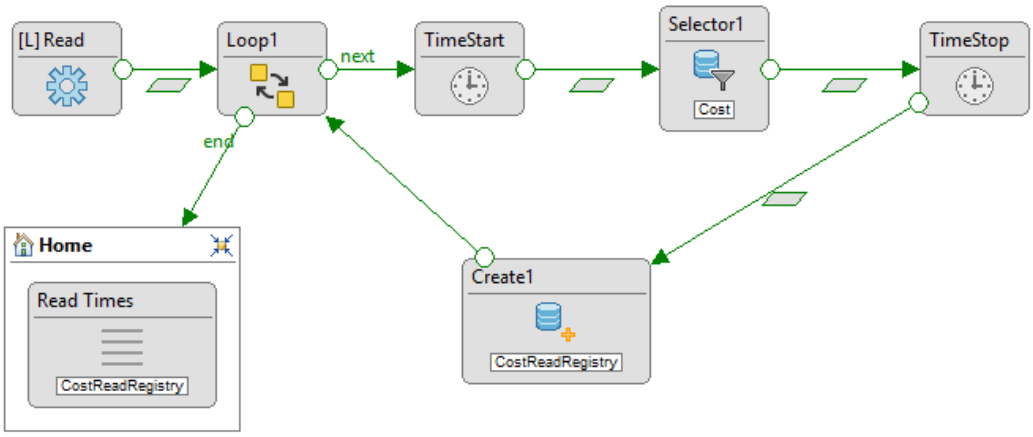


Ilustración 20. Proyectos de rendimiento con Time Units en WebRatio. Read

Para este proyecto el modelo de dominio es el siguiente, idéntico a la operación anterior:

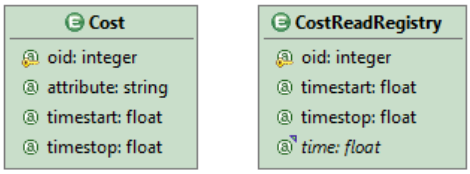


Ilustración 21. Proyectos de rendimiento con Time Units en WebRatio. Read. Modelo de datos

En el caso de la operación UPDATE, el diagrama IFML tendrá como elemento diferente a los diagramas anteriores una *Update Unit* de la entidad de prueba *Cost*. El resto de elementos son comunes a las operaciones anteriores, y también viene ejecutada por una *No Operation Unit*:

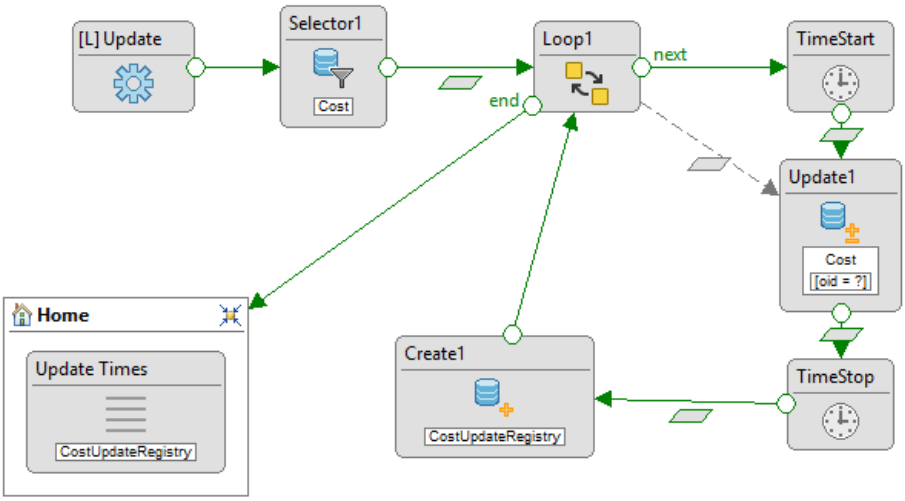


Ilustración 22. Proyectos de rendimiento con Time Units en WebRatio. Update

A continuación, el modelo de dominio, idéntico a los casos anteriores:

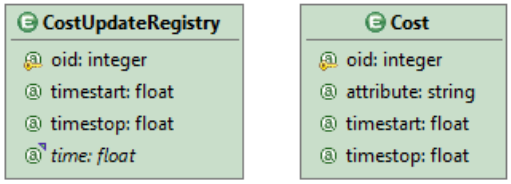


Ilustración 23. Proyectos de rendimiento con Time Units en WebRatio. Update. Modelo de datos

Finalmente, para la operación DELETE la novedad que incorpora el diagrama IFML es una *Delete Unit*, el resto de elementos sigue siendo invariable:

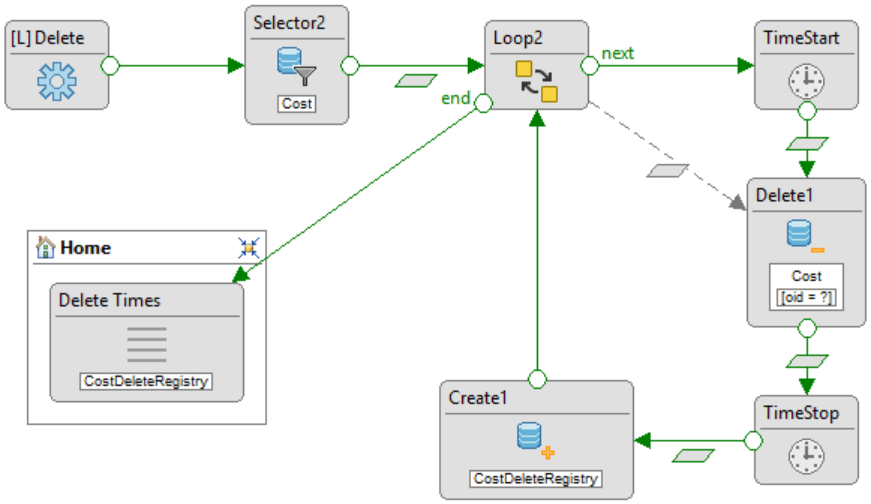


Ilustración 24. Proyectos de rendimiento con Time Units en WebRatio. Delete

El modelo de dominio es idéntico al de las otras tres operaciones:

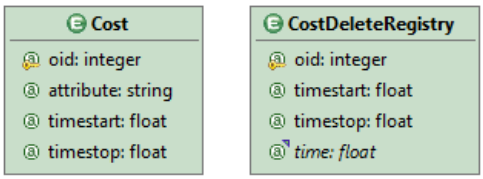


Ilustración 25. Proyectos de rendimiento con Time Units en WebRatio. Delete. Modelo de datos

5.2.3. Unificación en el circuito CRUD

Se ha podido observar que los 4 diagramas anteriores compartían muchos elementos en común, el más evidente era el modelo de dominio, idéntico para las 4 operaciones CRUD. Adicionalmente tendría en cuenta otros parámetros como el tipo de operación y persistencia, para automatizar más el proceso. Por ello, se propuso unir los 4 proyectos y montar un circuito CRUD. El sería el siguiente:

CRUDProject_Home

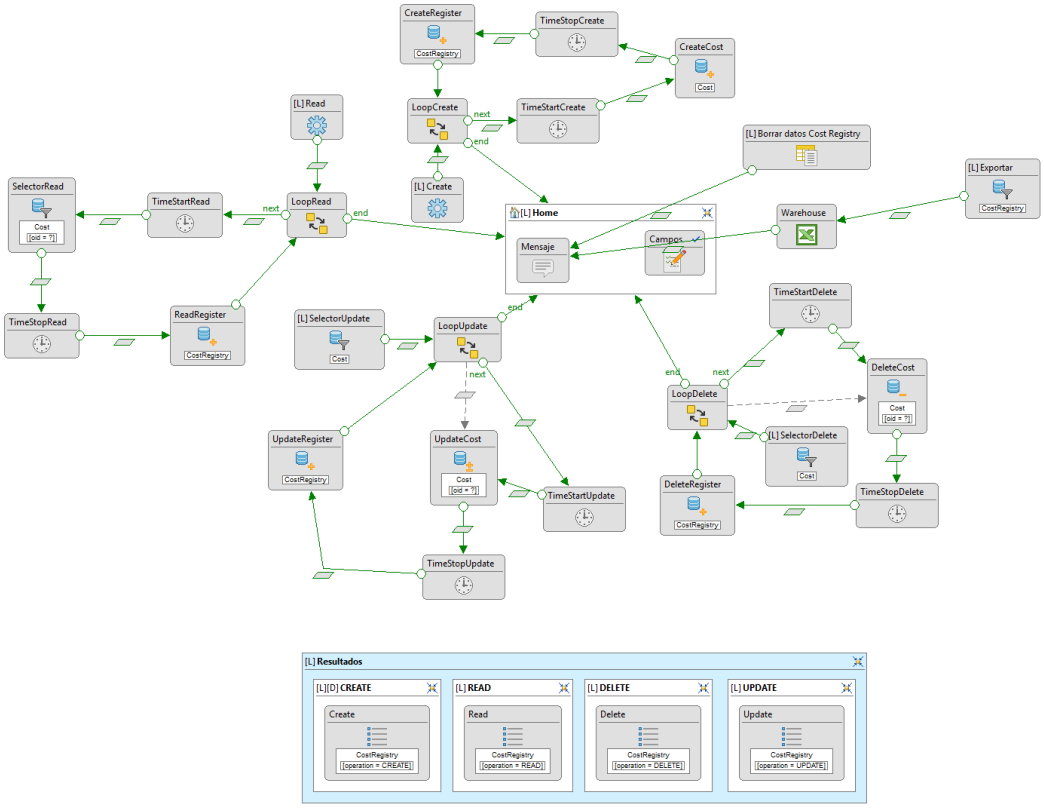


Ilustración 26. Proyectos de rendimiento con Time Units en WebRatio. Circuito completo (ampliar para más detalles)

Y su modelo de dominio se vería ligeramente modificado, en la entidad que mide los costes temporales tendríamos atributos adicionales como el tipo de operación CRUD, el número de atributos de la entidad de prueba *Cost*, y el tipo de persistencia (*duration*):

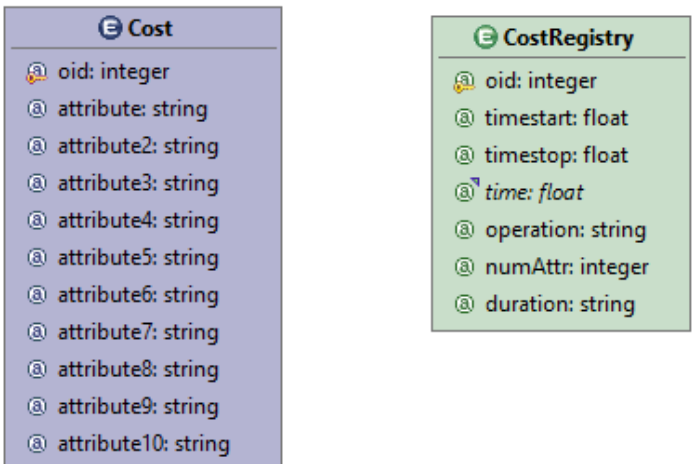


Ilustración 27. Proyectos de rendimiento con Time Units en WebRatio. Modelo de datos

5.2.4. Justificación

Unificar las 4 operaciones en un mismo proyecto y de forma consecutiva presenta varias ventajas. La cantidad de proyectos que hay que desplegar a la nube de AWS es mucho menor, ahorrando así un tiempo considerable. Las 4 operaciones por separado supondrían un total de 72 proyectos, resultado de multiplicar 4 operaciones x 6 cantidades distintas de atributos x 3 tipos de persistencia. No hay que olvidar que para cambiar el número de atributos a probar o el tipo de persistencia hay que cambiar el modelo de dominio y generar un .WAR completamente distinto.

Al unificar todas las operaciones en un único circuito la cantidad de .WAR a generar se reducen a 18, 6 cantidades distintas de atributos x 3 tipos de persistencia.

Por otro lado, además se garantiza de forma fácil en el caso de las pruebas de persistencia en base de datos que el estado al final de la ejecución de las pruebas es idéntico al de inicio, esto simplemente es por el orden en que se ejecutan las

operaciones, primero crear, luego leer, actualizar y finalmente borrar. Así el estado de la base de datos siempre es consistente al inicio y final de las pruebas.

5.3. Exportación de los datos

Para exportar los datos almacenados en la base de datos se ha recurrido a la Excel Unit en modo escritura:

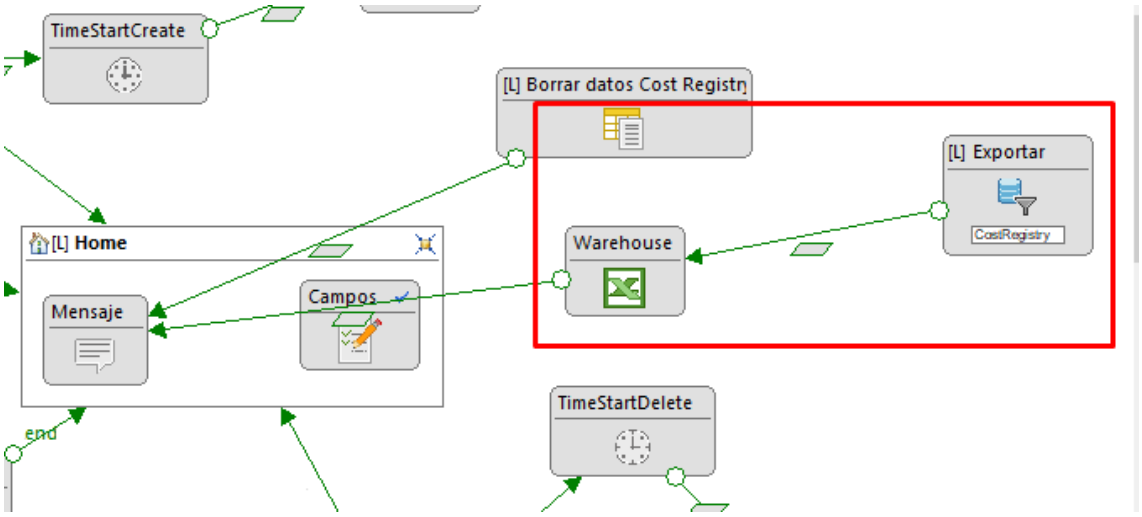


Ilustración 28. Exportación de los datos. Excel Unit

De esta forma se obtiene un Excel con la siguiente estructura:

A1 | oid

	A	B	C	D
1	oid	time	operation	duration
2	1	313,0	CREATE	PERSISTENT
3	2	39,0	CREATE	PERSISTENT
4	3	15,0	CREATE	PERSISTENT
5	4	9,0	CREATE	PERSISTENT
6	5	10,0	CREATE	PERSISTENT
7	6	9,0	CREATE	PERSISTENT
8	7	7,0	CREATE	PERSISTENT
9	8	12,0	CREATE	PERSISTENT
10	9	9,0	CREATE	PERSISTENT
11	10	12,0	CREATE	PERSISTENT

Ilustración 29. Exportación de los datos. Resultado

Hay 4 columnas: *oid, time, operation y duration*.

Para importar los datos de una tabla EXCEL en R, lo primero que hay que hacer será convertirlo a .csv en formato UTF-8.

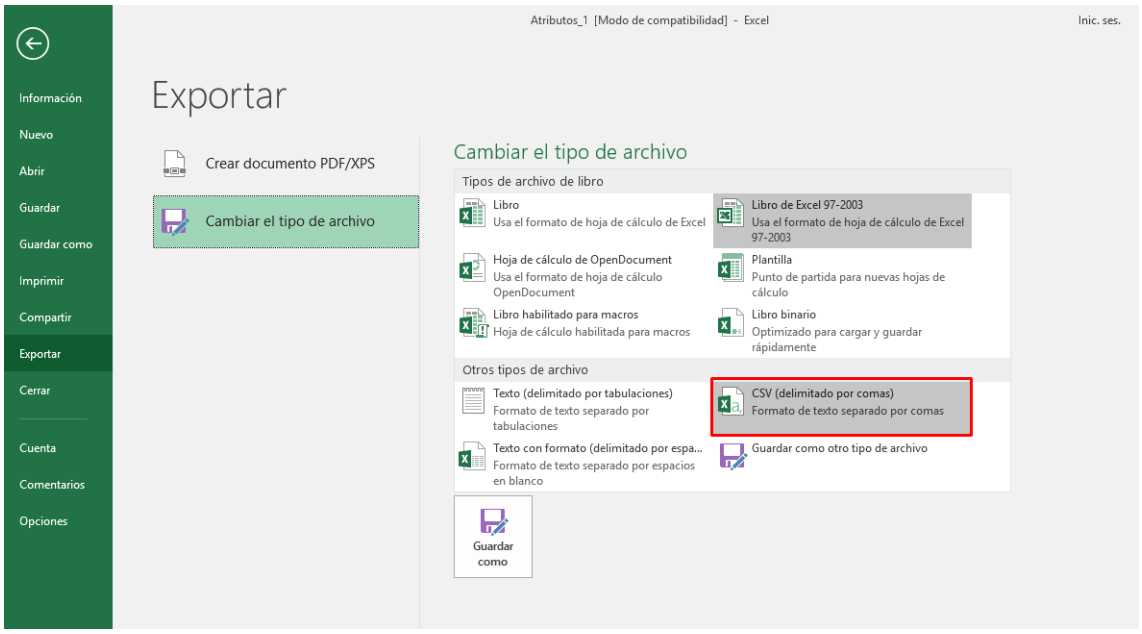


Ilustración 30. Exportación de los datos. Transformar a CSV

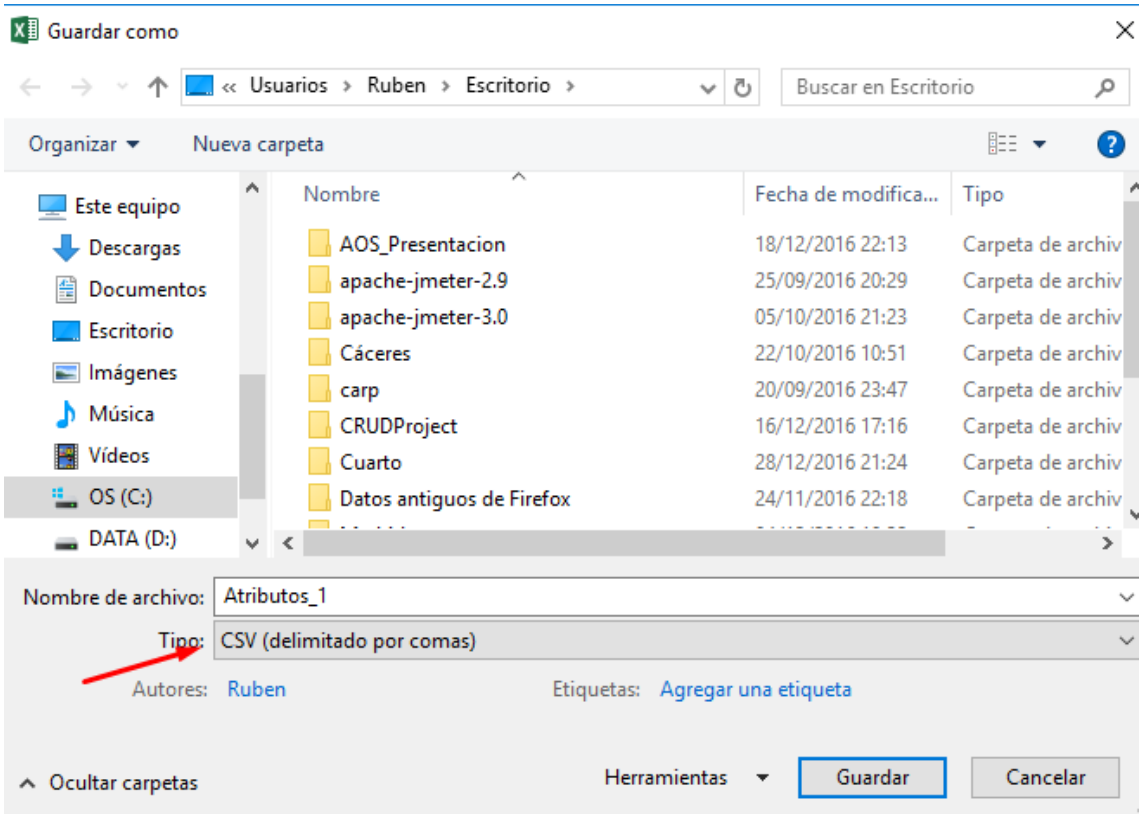


Ilustración 31. Exportación de los datos. Formato del CSV

5.4. Análisis de los datos

5.4.1. Introducción a RStudio

Ilustración 32. Introducción a RStudio. Logo

RStudio es un entorno de desarrollo integrado multiplataforma muy popular para el lenguaje estadístico y matemático R. Se trata de un lenguaje ampliamente utilizado en estudios científicos por la gran biblioteca de librerías disponibles que incorporan multitud de algoritmos y variantes.

En el campo de la minería de datos, la bioinformática y la biomedicina es bastante conocido.

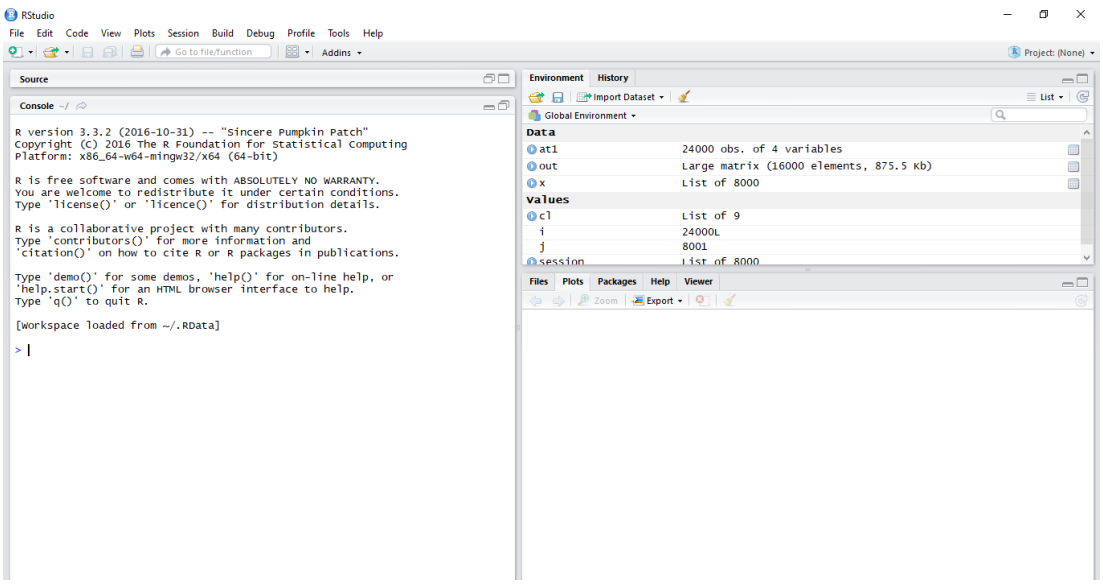


Ilustración 33. Introducción a RStudio. Ventana principal

En este proyecto el uso de RStudio se enmarca en la necesidad de analizar los datos de las mediciones que se han exportado a una hoja de cálculo. En dicho análisis se hará uso de un algoritmo ya implementado en las librerías de RStudio y también de librerías gráficas para representar los resultados de la aplicación de dicho algoritmo.

En los siguientes apartados se darán más detalles al respecto.

5.4.2. Pre-procesado de datos

Una vez obtenidos los ficheros Excel con las mediciones de los tiempos, el siguiente paso es analizar dichos valores en RStudio.

En primer lugar, será necesario leer los datos de cada archivo Excel y almacenarlos en listas según su tipo de persistencia y operación. En el siguiente código queda reflejado este proceso:

```
at1 <-
read.csv2("C:\\Users\\Ruben\\Desktop\\50_attributes.csv",
header=TRUE)
# Columnas de los tiempos
at1[,2]
# Columnas de las operaciones
at1[,3]
# Columnas de las persistencias
at1[,4]
#plot(at1[,4])

session_create<-list()
session_read<-list()
session_update<-list()
session_delete<-list()
persistent_create<-list()
persistent_read<-list()
persistent_update<-list()
```

```
persistent_delete<-list()
application_create<-list()
application_read<-list()
application_update<-list()
application_delete<-list()

sc<-1
sr<-1
su<-1
sd<-1

pc<-1
pr<-1
pu<-1
pd<-1

ac<-1
ar<-1
au<-1
ad<-1

for(i in 1:24000){

  if(at1[i,4]=='SESSION' &&
at1[i,3]=='CREATE'){session_create[sc]<-at1[i,2]; sc<-sc+1}
  if(at1[i,4]=='SESSION' &&
at1[i,3]=='READ'){session_read[sr]<-at1[i,2]; sr<-sr+1}
  if(at1[i,4]=='SESSION' &&
at1[i,3]=='UPDATE'){session_update[su]<-at1[i,2]; su<-su+1}
  if(at1[i,4]=='SESSION' &&
at1[i,3]=='DELETE'){session_delete[sd]<-at1[i,2]; sd<-sd+1}
```

```
  if(at1[i,4]=='APPLICATION' &&
at1[i,3]=='CREATE'){application_create[ac]<-at1[i,2]; ac<-
ac+1}
  if(at1[i,4]=='APPLICATION' &&
at1[i,3]=='READ'){application_read[ar]<-at1[i,2]; ar<-ar+1}
  if(at1[i,4]=='APPLICATION' &&
at1[i,3]=='UPDATE'){application_update[au]<-at1[i,2]; au<-
au+1}
  if(at1[i,4]=='APPLICATION' &&
at1[i,3]=='DELETE'){application_delete[ad]<-at1[i,2]; ad<-
ad+1}

  if(at1[i,4]=='PERSISTENT' &&
at1[i,3]=='CREATE'){persistent_create[pc]<-at1[i,2]; pc<-pc+1}
  if(at1[i,4]=='PERSISTENT' &&
at1[i,3]=='READ'){persistent_read[pr]<-at1[i,2]; pr<-pr+1}
  if(at1[i,4]=='PERSISTENT' &&
at1[i,3]=='UPDATE'){persistent_update[pu]<-at1[i,2]; pu<-pu+1}
  if(at1[i,4]=='PERSISTENT' &&
at1[i,3]=='DELETE'){persistent_delete[pd]<-at1[i,2]; pd<-pd+1}

    }
```

Se crean primero las listas y luego unos índices para recorrerlas y actualizar sus valores. En función del valor de la tercera y cuarta columna los datos se almacenan en una lista u otra. Esto facilitará posteriormente el procesado de esos datos según su clasificación.

Mediante un algoritmo de aprendizaje no supervisado, concretamente uno conocido como k-means o k-medias, se procesarán los datos por tipo de persistencia y operación, con el objetivo de aislar aquellas mediciones que hayan podido ser excepcionales y se encuentren fuera del rango habitual de valores. De este modo se

podrá obtener un valor medio para cada caso de estudio mucho más preciso, y en el que se minimiza el impacto de valores que no son representativos.

El propio entorno RStudio incorpora una implementación de dicho algoritmo en R, no obstante, antes de pasar a los detalles en el código, se va a explicar más detalladamente en qué consiste este algoritmo.

5.4.3. El algoritmo K-Means

El algoritmo del K-means o k-medias es un algoritmo de clusterización. Permite agrupar un conjunto de datos numéricos en grupos o *clústeres* según sus similitudes.

Aunque se trata de un algoritmo de aprendizaje no supervisado, requiere que inicialmente se establezcan ciertos valores para la ejecución del algoritmo. Uno de ellos es el número de clústeres en que se van a agrupar los datos. Si este número (habitualmente conocido como *k*) vale 4, el algoritmo devolverá como resultado 4 agrupaciones de datos que han ido constituyéndose durante varias iteraciones del algoritmo.

La secuencia de pasos del algoritmo es la siguiente:

1. Se cogen *k* valores aleatorios de entre los valores de entrada, constituyendo cada valor escogido un clúster con un valor medio *m* que en el primer paso de forma excepcional coincide con *k.*

2. En forma de bucle, a cada uno de los demás valores que no han sido escogidos en el paso 1. se va comparando con el valor medio *m* de cada uno de los clústeres, y aquel con el que la diferencia de valores sea menor pues será el clúster al que se asigne dicho valor.

   Al añadir un nuevo valor a un clúster, la media *m* de ese clúster es recalculada, y esto provoca haya que volver a comparar cada uno de los valores de todos los clústeres con los nuevos valores *m* resultantes de haber clasificado un nuevo valor. Es un proceso complejo y según el número de valores de entrada

puede ser computacionalmente muy costoso, puede haber valores que previamente estaban en un clúster, pero como consecuencia de un reajuste de los valores medios *m* dicho valor pase a pertenecer a otro clúster diferente al que había sido asignado previamente.

3. La condición de parada del algoritmo puede ser o bien el número de iteraciones (se establece que el algoritmo no haga más de *x* iteraciones) o bien se deja terminar el bucle anterior en el que como estado final las medias *m* de cada clúster ya no sufrirían variaciones. Este sería el indicador de que se ha completado el proceso de clusterización.

Una de las dificultades presentes en el uso de este algoritmo es saber cuál es el valor adecuado *k* para su ejecución. Hay algunos algoritmos como el de Análisis de Componentes Principales (PCA en inglés) que dan una estimación de cuál puede ser un buen número para el parámetro *k*. No obstante, dependiendo del conjunto de datos a veces el resultado no está claro, y requiere de una representación bidimensional de los datos para escoger dicho valor.

Un buen indicador de si se ha escogido un valor acertado para *k* es la medida del error cuadrático o también conocida como suma de cuadrados cuya fórmula es la siguiente:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2$$

Ilustración 34. Análisis de los datos. Expresión del error

P es uno de los valores del clúster y *m* es su media. Tomando módulo y elevando al cuadrado se evitan números negativos en la expresión. Así lo que se hace es sumar todas las diferencias elevadas al cuadrado de cada uno de los puntos de un clúster y su media, y ese resultado a su vez se suma con el resultado del mismo proceso en el

resto de clústeres. Esa suma de sumas es lo que en la expresión viene como *E*, o error cuadrático.

¿Cuál es el objetivo? Minimizar el valor de *E.* ¿Por qué? Porque eso significará que los valores que pertenecen a un clúster son muy similares entre sí. El objetivo final de aplicar el algoritmo del k-means es el de obtener agrupaciones de datos suficientemente grandes heterogéneas entre sí, pero lo más homogéneas posibles a nivel de los datos contenidos.

La implementación del algoritmo en RStudio devuelve el valor de dicho error, y eso es lo que va a permitir saber si el valor inicial de *k* que se ha escogido es el más apropiado o no. Se harán varias pruebas hasta llegar a un error E mínimo. Con un valor *k* demasiado bajo es posible que el error cuadrático no sea muy pequeño debido a la posible heterogeneidad de valores, pero por otro lado tampoco es apropiado un valor de *k* demasiado alto, ya que provocaría que la aplicación del algoritmo no tuviera sentido.

Para ilustrar esta idea valdría el siguiente ejemplo. Suponiendo que se dispone de un conjunto de 10 valores numéricos diferentes y se quiere aplicar el algoritmo del k-means para buscar clasificaciones según un criterio que a priori se desconocen, un valor inicial razonable para *k* podría ser 3, y posteriormente se probaría para k=2 y k=4 con tal de detectar ese valor más apropiado para k donde el error cuadrático sea bajo.

Evidentemente no tiene sentido aplicar el algoritmo para el valor k=1, no haría ninguna clasificación ya que el conjunto devuelto en la salida del algoritmo sería idéntico al de entrada. Únicamente se obtendría la información adicional de cuál es la media de todos los valores, pero para ello no es necesario aplicar un procedimiento tan complejo.

No obstante, en este caso de ejemplo, podría ser posible cometer el error de darle a k un número demasiado elevado como por ejemplo k=9. Previsiblemente el error cuadrático fuese pequeño, pero se obtendrían casi tantos conjuntos de salida como valores distintos de entrada. Eso no resulta muy útil.

Si el valor de k coincidiese con la cantidad de datos distintos de entrada, es decir, k=10, el error sería 0, porque a cada valor le asociaría un grupo, y la media de valores de cada uno de los grupos sería idéntica al valor que contuviese cada uno. Se estaría en una situación parecida a la de k=1.

Tampoco tiene sentido aplicar un valor de k > 10. Directamente el algoritmo no podría avanzar del paso 1 porque no podría escoger los suficientes valores aleatorios para establecer los *k* conjuntos de datos iniciales.

En resumen, hay que buscar un valor de *k* razonablemente pequeño en comparación con el tamaño del conjunto de datos de entrada y a partir del cual, incrementando *k*, el error cuadrático no disminuye sustancialmente. Esto implica aceptar que el error cuadrático puede que no sea el menor error posible, pero sí estará muy cerca de ese mínimo. A cambio de esa flexibilidad en el error se obtendrán conjuntos de datos más grandes y significativos para clasificar los datos de entrada.

5.4.4. Aplicación del algoritmo K-Means en RStudio

Volviendo al aspecto del código como continuación de lo anterior, en la implementación del algoritmo proporcionada se establece como uno de los parámetros de entrada un array con los datos a *clusterizar*. Anteriormente se definieron listas en las que guardar los datos del fichero ya de forma clasificada, por tanto, a continuación, se crearán un conjunto de arrays a partir de esas listas:

```
array_sc <- array(session_create,
dim=c(length(session_create),1))

array_sr <- array(session_read, dim=c(length(session_read),1))

array_su <- array(session_update,
dim=c(length(session_update),1))

array_sd <- array(session_delete,
dim=c(length(session_delete),1))
```

```
array_ac <- array(application_create,
dim=c(length(application_create),1))

array_ar <- array(application_read,
dim=c(length(application_read),1))

array_au <- array(application_update,
dim=c(length(application_update),1))

array_ad <- array(application_delete,
dim=c(length(application_delete),1))



array_pc <- array(persistent_create,
dim=c(length(persistent_create),1))

array_pr <- array(persistent_read,
dim=c(length(persistent_read),1))

array_pu <- array(persistent_update,
dim=c(length(persistent_update),1))

array_pd <- array(persistent_delete,
dim=c(length(persistent_delete),1))
```

Los arrays creados tienen una única dimensión. Una vez hecho esto, será posible invocar al algoritmo.

```
require(graphics)

(cl1 <- kmeans(array_sc, 4))

plot(array_sc, col = cl1$cluster)

points(cl1$centers, col = 1:3, pch = 20, cex = 2)



(cl2 <- kmeans(array_sr, 6))
```

```
plot(array_sr, col = cl2$cluster)
points(cl2$centers, col = 1:3, pch = 20, cex = 2)

(cl3 <- kmeans(array_su, 5))
plot(array_su, col = cl3$cluster)
points(cl3$centers, col = 1:3, pch = 20, cex = 2)

(cl4 <- kmeans(array_sd, 10))
plot(array_sd, col = cl4$cluster)
points(cl4$centers, col = 1:3, pch = 20, cex = 2)

(cl5 <- kmeans(array_ac, 6))
plot(array_ac, col = cl5$cluster)
points(cl5$centers, col = 1:3, pch = 20, cex = 2)

(cl6 <- kmeans(array_ar, 7))
plot(array_ar, col = cl6$cluster)
points(cl6$centers, col = 1:3, pch = 20, cex = 2)

(cl7 <- kmeans(array_au, 4))
plot(array_au, col = cl7$cluster)
points(cl7$centers, col = 1:3, pch = 20, cex = 2)

(cl8 <- kmeans(array_ad, 5))
```

```
plot(array_ad, col = cl8$cluster)
points(cl8$centers, col = 1:3, pch = 20, cex = 2)

(cl9 <- kmeans(array_pc, 9))
plot(array_pc, col = cl9$cluster)
points(cl9$centers, col = 1:3, pch = 20, cex = 2)

(cl10 <- kmeans(array_pr, 7))
plot(array_pr, col = cl10$cluster)
points(cl10$centers, col = 1:3, pch = 20, cex = 2)

(cl11 <- kmeans(array_pu, 6))
plot(array_pu, col = cl11$cluster)
points(cl11$centers, col = 1:3, pch = 20, cex = 2)

(cl12 <- kmeans(array_pd, 9))
plot(array_pd, col = cl12$cluster)
points(cl12$centers, col = 1:3, pch = 20, cex = 2)
```

La invocación al algoritmo tiene lugar en las líneas con la siguiente estructura:

Cl_# <- kmeans(array_ #, k)

Eso devuelve una estructura con varias columnas en las que en una de ellas aparecen todos los valores numéricos de entrada y en otra el identificador numérico del clúster al que pertenece dicho valor, y que variará desde 1 hasta *k.*

Por cada array se repite la misma sentencia que viene seguida de una invocación a la función *plot* para representar gráficamente esos datos. Con la función *points* se modifican algunos de los parámetros por defecto de *plot* para visualizar mejor los puntos representados, por ejemplo, el número de píxeles por cada punto, y los puntos que hacen de centros geométricos de cada clúster, denominados *centroides*.

Cada uno de los clústeres tiene un valor que está más próximo al valor medio *m* que el resto de los puntos de ese clúster. A ese valor único se le denomina centroide, y no tiene por qué coincidir con el valor medio de todos los puntos, aunque a veces así suceda.

A continuación, se realizará la ejecución del algoritmo para los datos de *sesión* y *delete,* con un único usuario y para 50 atributos. La salida gráfica de la ejecución es la siguiente:

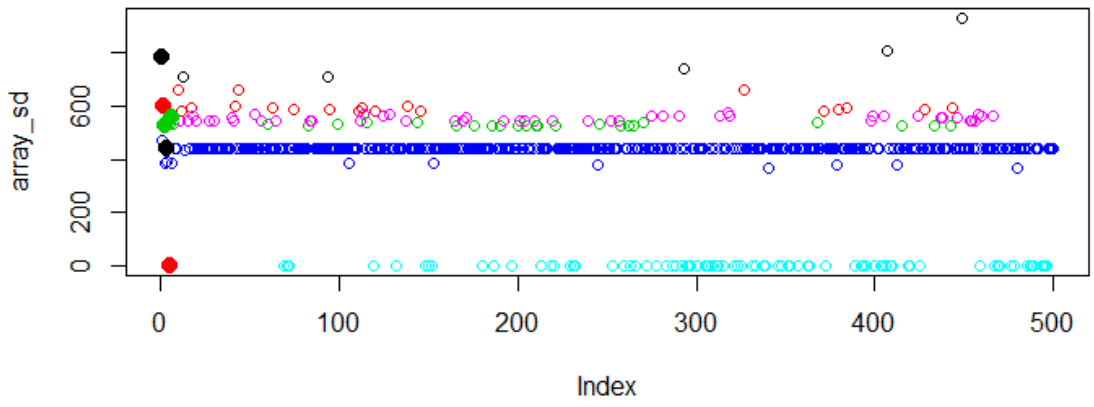


Ilustración 35. Análisis de los datos. Gráfica de ejemplo

Se puede apreciar que la mayoría de puntos se concentra por encima del valor 400 en el eje Y, y estos han sido agrupados en un clúster de color azul oscuro. Hay unos puntos que aparecen más grandes que el resto y con un color sólido de relleno. Son los centroides de los distintos clústeres, y su color no siempre coincide con el del resto de puntos de su clúster. Por ejemplo, el centroide del clúster azul claro, situado en la parte inferior de la gráfica, es de color rojo.

Por otro lado, por la ventana de consola han salido nuevos datos numéricos, como el tamaño de cada uno de los clústeres y sus medias:

```
> plot(array_sd, col = cl4$cluster)
> points(cl4$centers, col = 1:3, pch = 20, cex = 2)
> (cl4 <- kmeans(array_sd, 6))
K-means clustering with 6 clusters of sizes 5, 19, 26, 322, 80, 48

Cluster means:
      [,1]
1 778.8000
2 599.8947
3 528.0769
4 438.2329
5   4.0000
6 553.0000

Clustering vector:
  [1] 4 3 4 4 3 4 3 4 4 2 6 2 1 4 6 4 2 6 4 6 4 4 4 4 4 4 6 4 4 6 4 4 4 4
 [35] 4 4 4 4 4 6 6 2 4 2 4 4 4 4 4 4 4 4 6 4 4 6 4 4 4 3 4 4 2 4 6 4 4 4
 [69] 5 4 5 5 4 4 2 4 4 4 4 4 4 4 3 6 6 4 4 4 4 4 4 4 4 1 2 4 4 4 3 4 4 4
[103] 4 4 4 4 4 4 4 4 2 6 2 6 4 3 4 4 5 2 4 4 4 4 6 4 4 6 4 4 4 5 4 4 4 4
[137] 6 2 4 4 4 4 4 3 4 2 4 5 4 5 4 5 4 4 4 4 4 4 4 4 4 4 4 4 4 6 3 4 4 6 4
[171] 6 4 4 4 4 3 4 4 4 5 4 4 4 4 4 3 5 4 4 3 4 6 4 4 4 4 5 4 4 3 6 4 4 6
[205] 3 4 4 4 6 3 3 4 5 4 4 4 4 5 6 5 3 4 4 4 4 4 4 4 5 4 5 5 4 4 4 4 4 4
[239] 6 4 4 4 4 4 4 3 4 4 4 4 4 6 5 4 4 4 6 3 5 4 4 3 5 4 3 5 4 4 4 3 4 5
[273] 4 4 6 4 4 5 4 4 6 4 5 4 4 4 5 4 5 6 4 5 1 4 5 5 5 4 4 5 4 4 4 4 5 5
[307] 5 4 5 4 5 5 6 4 5 4 4 6 6 4 5 4 5 4 5 4 2 4 4 4 5 4 5 4 4 4 4 5 5 4
[341] 4 4 4 4 4 5 4 5 4 4 5 5 4 4 4 4 5 4 4 4 4 5 5 4 4 4 4 3 4 4 2 5 4 4
[375] 4 4 4 4 4 2 4 4 4 2 4 4 4 4 5 4 4 5 5 4 5 4 4 6 6 5 4 4 5 5 6 5 1 4
[409] 5 5 4 4 4 4 3 4 4 4 5 5 4 4 4 6 5 4 4 2 4 4 4 4 3 4 4 4 6 6 4 4 4 3
[443] 2 4 4 6 4 4 1 4 4 4 6 6 4 6 4 6 5 6 4 4 4 4 4 6 5 4 5 5 4 4 4 4 4 5
[477] 4 5 4 4 4 4 4 4 5 4 5 5 5 4 4 4 5 5 5 5 4 4 4 4

Within cluster sum of squares by cluster:
[1] 32976.8000 12671.7895   749.8462 36293.5311     0.0000  4128.0000
 (between_SS / total_SS =  99.5 %)
```

Ilustración 36. Análisis de los datos. Resultado devuelto por consola

El número *k* escogido ha sido 6, de ahí que en la gráfica aparezcan 6 clústeres con 1 color diferente cada uno, y el valor medio del clúster más numeroso (que contiene 322 puntos sobre un total de 500 mediciones) es de 438,2329. Coincide con lo que se observa en la gráfica anterior de que la mayoría de los puntos estaban por encima del valor 400 en el eje Y.

¿Qué significado tiene este resultado? Pues que en la mayoría de los casos para la operación *delete* con el tipo de persistencia volátil *session* y 50 atributos el coste medio estará en unos 438 milisegundos, que es la magnitud del eje Y.

Un valor muy interesante es el devuelto casi al final de la ejecución, es un porcentaje que ha sido calculado como *between_ss / total_ss*, y que en este caso vale 99,5%. Este es un valor que está directamente relacionado con el error cuadrático o suma de cuadrados (*sum of squares o ss*). Sin embargo, aquí lo muestra en un porcentaje para poder interpretarlo mejor.

Ese 99,5% indica que los valores que constituyen cada uno de los clústeres son casi idénticos, y por tanto se ha hecho una buena clusterización. Para ello haber escogido el valor de *k*=6 ha sido clave. Si se hacen variaciones en el valor de *k* se podrá comprobar aquello que se dijo anteriormente sobre los riesgos de coger un número demasiado bajo o demasiado alto.

Para el mismo conjunto de datos se va a coger el valor k=2:

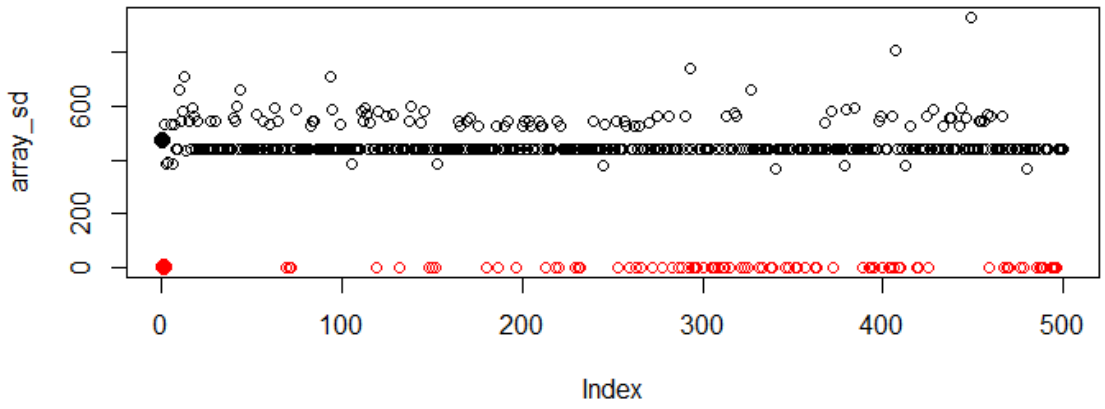


Ilustración 37. Análisis de los datos. Gráfica de ejemplo mal clusterizado

Ahora únicamente se distinguen dos agrupaciones de puntos en dos colores: rojo y negro. El resultado por consola ha sido el siguiente:

```
> plot(array_sd, col = cl4$cluster)
> points(cl4$centers, col = 1:3, pch = 20, cex = 2)
> (cl4 <- kmeans(array_sd, 2))
K-means clustering with 2 clusters of sizes 420, 80

Cluster means:
      [,1]
1 468.2786
2   4.0000

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [69] 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[103] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
[137] 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[171] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1
[205] 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1
[239] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1
[273] 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 2 1 1 2 1 1 2 2 2 1 1 2 1 1 1 1
[307] 2 1 2 1 2 2 1 1 2 1 1 1 1 1 2 1 2 1 2 1 1 1 1 1 2 1 2 1 1 1 1 2
[341] 1 1 1 1 1 2 1 2 1 1 2 2 1 1 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1
[375] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 1 1 1 1 2 1 1 2 2 1 2
[409] 2 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[443] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1
[477] 1 2 1 1 1 1 1 1 2 1 2 2 2 1 1 1 2 2 2 2 1 1 1 1

Within cluster sum of squares by cluster:
[1] 1626256       0
 (between_SS / total_SS =  89.9 %)
```

Ilustración 38. Análisis de los datos. Resultado por consola de mala clusterización

Los 500 valores que constituyen el conjunto de datos ahora han sido distribuidos en 2 clústeres con un tamaño de 420 y 80 valores respectivamente. La media del clúster más grande es de 468,2786 milisegundos, ligeramente superior al resultado para *k=6*. El motivo de este valor es que al incluir en ese clúster valores más altos que antes pertenecían a otro clúster la media del conjunto de puntos se ha incrementado.

Si se presta atención en el cociente de la suma de cuadrados, el valor devuelto en este caso es de 89,9%, es decir, casi 10 puntos porcentuales menos que en la ejecución anterior. Sigue siendo un valor aceptable, sin embargo, no es tan preciso e indica que el grado de heterogeneidad en los conjuntos de datos que constituyen los clústeres es mayor. Claramente conviene probar con un valor superior de *k* para obtener una mejor precisión, además que a simple vista de la representación gráfica es fácil observar que hay un conjunto de puntos que no deberían pertenecer a un único clúster por ser valores tan distintos:

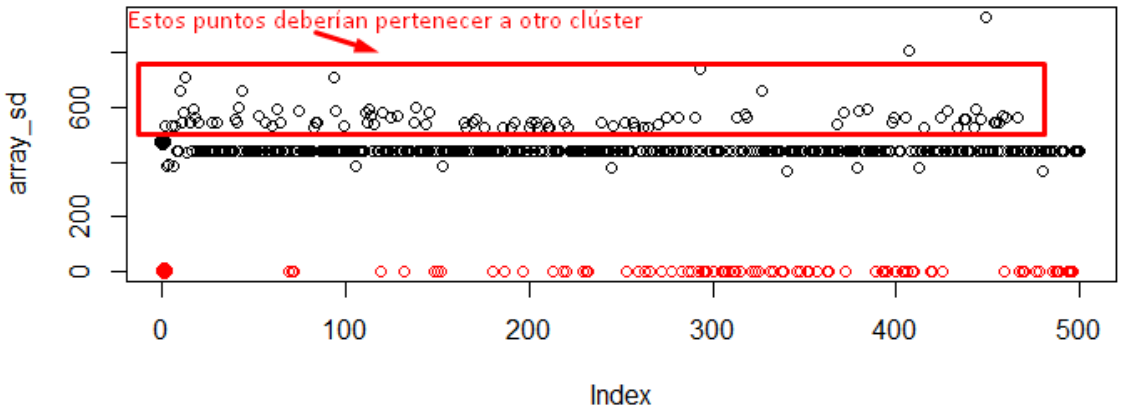


Ilustración 39. Análisis de los datos. Explicación gráfica de clusterización

A continuación, se repetirá para el valor *k*=12:

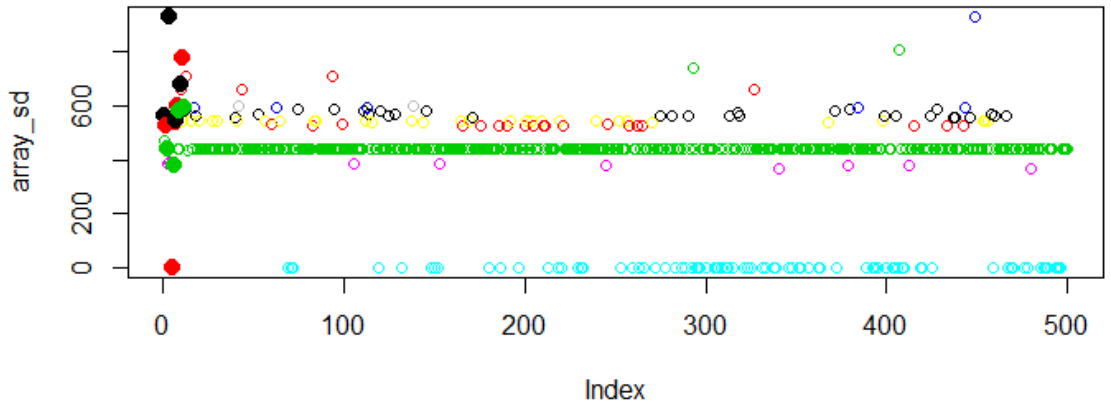


Ilustración 40. Análisis de los datos. Clusterización para k con un valor alto

Se puede apreciar que la agrupación con más puntos esta vez ha sido coloreada de verde. La variedad de tonos en la representación gráfica es limitada, por ello a veces en un número alto para *k* a veces varios clústeres diferentes comparten color, como es el caso de algunos puntos aislados que se observan más arriba.

El valor devuelto por consola es el siguiente:

```
> plot(array_sd, col = cl4$cluster)
> points(cl4$centers, col = 1:3, pch = 20, cex = 2)
> (cl4 <- kmeans(array_sd, 12))
K-means clustering with 12 clusters of sizes 26, 4, 10, 17, 13, 2, 5, 16, 8
0, 312, 6, 9

Cluster means:
       [,1]
1  545.0000
2  527.5000
3  380.3000
4  560.5882
5  523.6154
6  865.5000
7  568.8000
8  589.0000
9    4.0000
10 440.0897
11 689.5000
12 534.7778


Clustering vector:
  [1] 10 12  3  3  2  3 12 10 10 11  1  8 11 10  1 10  8  4 10  1 10 10
 [23] 10 10 10 10  1 10 10  1 10 10 10 10 10 10 10 10 10  4  1  8 10 11
 [45] 10 10 10 10 10 10 10 10  7 10 10  1 10 10 10 12 10 10  8 10  1 10
 [67] 10 10  9 10  9  9 10 10  8 10 10 10 10 10 10 10  5  1  1 10 10 10
 [89] 10 10 10 10 10 11  8 10 10 10 12 10 10 10 10 10 10  3 10 10 10 10
[111]  8  1  8  7 10 12 10 10  9  8 10 10 10 10  4 10 10  7 10 10 10  9
[133] 10 10 10 10  1  8 10 10 10 10 10 12 10  8 10  9 10  9 10  9  3 10
[155] 10 10 10 10 10 10 10 10 10 10  1  5 10 10  1 10  4 10 10 10 10  5
[177] 10 10 10  9 10 10 10 10 10  5  9 10 10  5 10  1 10 10 10 10  9 10
[199] 10  2  1 10 10  1  5 10 10 10  1  5  5 10  9 10 10 10 10  9  1  9
[221]  5 10 10 10 10 10 10 10  9 10  9  9 10 10 10 10 10 10  1 10 10 10
[243] 10 10  3 12 10 10 10 10 10  1  9 10 10 10  1  5  9 10 10  5  9 10
[265]  5  9 10 10 10 12 10  9 10 10  4 10 10  9 10 10  4 10  9 10 10 10
[287]  9 10  9  4 10  9 11 10  9  9  9 10 10  9 10 10 10 10  9  9  9 10
[309]  9 10  9  9  4 10  9 10 10  7  4 10  9 10  9 10  9 10 11 10 10 10
[331]  9 10  9 10 10 10 10  9  9  3 10 10 10 10 10  9 10  9 10 10  9  9
[353] 10 10 10 10  9 10 10 10 10  9  9 10 10 10 10 12 10 10  8  9 10 10
[375] 10 10 10 10  3  8 10 10 10  8 10 10 10 10  9 10 10  9  9 10  9 10
[397] 10  1  4  9 10 10  9  9  4  9  6 10  9  9 10  3 10 10  5 10 10 10
[419]  9  9 10 10 10  4  9 10 10  8 10 10 10 10  2 10 10 10  4  4 10 10
[441] 10  2  8 10 10  4 10 10  6 10 10 10  1  1 10  1 10  7  9  4 10 10
[463] 10 10 10  4  9 10  9  9 10 10 10 10 10  9 10  9 10  3 10 10 10 10
[485]  9 10  9  9  9 10 10 10  9  9  9  9 10 10 10 10

Within cluster sum of squares by cluster:
 [1]    0.000000    5.000000  552.100000  182.117647    5.076923
 [6] 7320.500000   54.800000  644.000000    0.000000 1103.487179
[11] 6553.500000   75.555556
 (between_SS / total_SS =  99.9 %)
```

Ilustración 41. Análisis de los datos. Resultado devuelto por consola para k con valor alto

Se puede comprobar que en este caso el conjunto más grande es el número 10, con 312 valores y una media de 440,0897 milisegundos. El cociente de la suma de cuadrados es del 99,9 % apenas un 0,4% más que en el caso de *k*=6. Pese a haber duplicado el número de clústeres devueltos, la mejora en el proceso de clusterización

ha sido muy residual, ya que de partida ese valor era bastante bueno (99,5 % para *k*=6). Es por este motivo que no sería demasiado apropiado escoger un valor tan alto para *k*, ya que la mejora apenas va a ser significativa.

En el análisis de los datos siempre se ha perseguido conseguir un porcentaje elevado de precisión o *accuracy*, no obstante hay algunos casos excepcionales en los que la dispersión de los puntos era tal que no era posible conseguir tal precisión y había que bajar hacia un umbral del 65% para que los resultados fueran coherentes.

6. Técnicas alternativas estudiadas y descartadas

6.1. Introducción a Apache JMeter

Ilustración 42. Introducción a Apache JMeter. Logo

Apache JMeter es una aplicación JAVA de software libre diseñada para realizar pruebas funcionales y de rendimiento a aplicaciones web.

Entre sus muchas funcionalidades una de las más importantes es la de simular varios hilos.

Es una herramienta que permite diseñar fácilmente casos de pruebas y exportar informes estadísticos con los resultados devueltos de las mediciones realizadas.

6.1.1. Justificación

Medir el rendimiento de nuestras operaciones y obtener datos estadísticos de las mediciones es uno de los objetivos de este proyecto. Es por ello que esta herramienta podría ser la solución apropiada para conseguirlo.

Es una herramienta para mediciones de código abierto y respaldada por una amplia comunidad de desarrolladores, por tanto, es una opción bastante interesante a tener en cuenta.

6.1.2. Puesta a punto de la herramienta

Lo primero que hay que hacer es descargar la última versión del software desde la dirección http://jmeter.apache.org/download_jmeter.cgi

Cuando estén descargados los archivos binarios comprimidos, se procede a descompromirlos. La estructura de las carpetas es la siguiente:

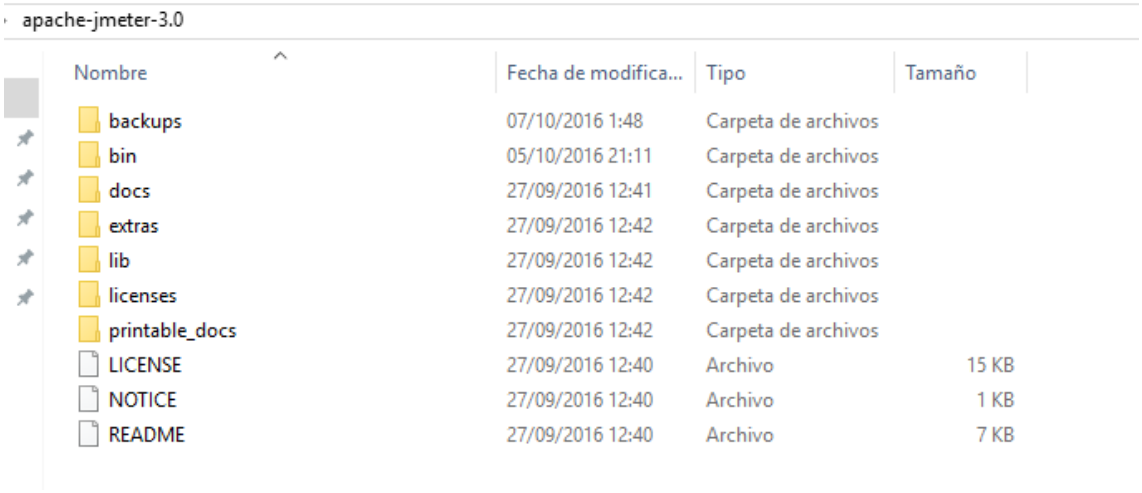


Ilustración 43. Introducción a Apache JMeter. Directorio del programa.

Dentro de la carpeta *bin* está el ejecutable de la aplicación. Será necesario tener instalada una versión de Java 7 o superior para poder ejecutarla. Hay varios ejecutables, por ejemplo, se puede escoger el siguiente:

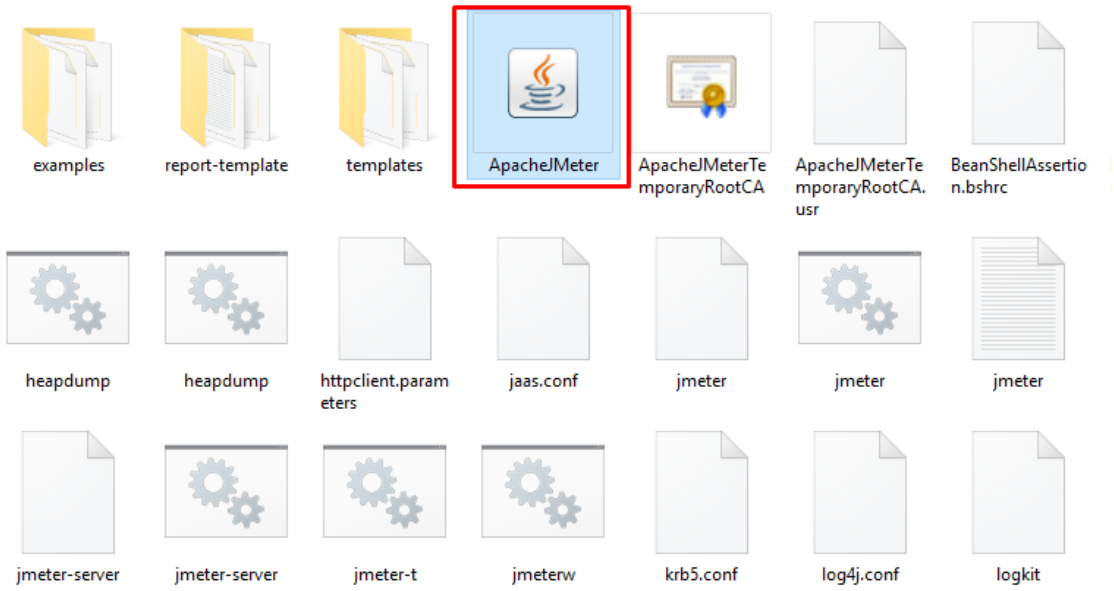


Ilustración 44. Introducción a Apache JMeter. Carpeta bin

Cuando cargue la aplicación aparecerá la siguiente ventana:

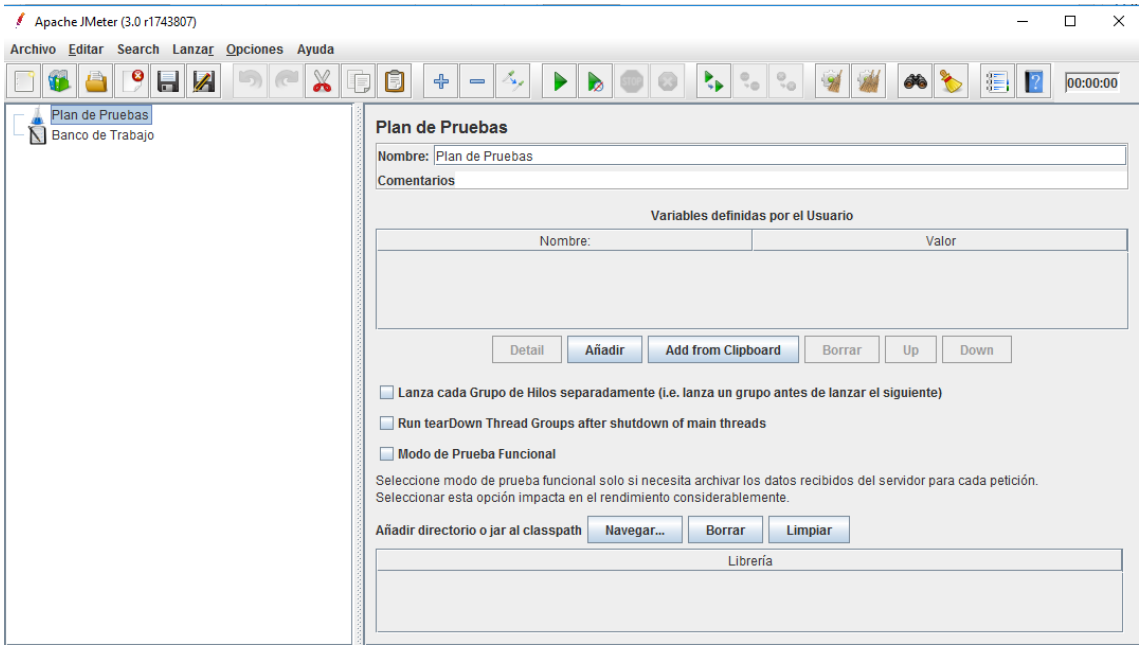


Ilustración 45. Introducción a Apache JMeter. Ventana principal de la aplicación

6.1.3. Realización de los casos de prueba

A continuación, se va a iniciar la grabación de los casos de prueba. Hay que partir de del siguiente proyecto base creado con este esquema:

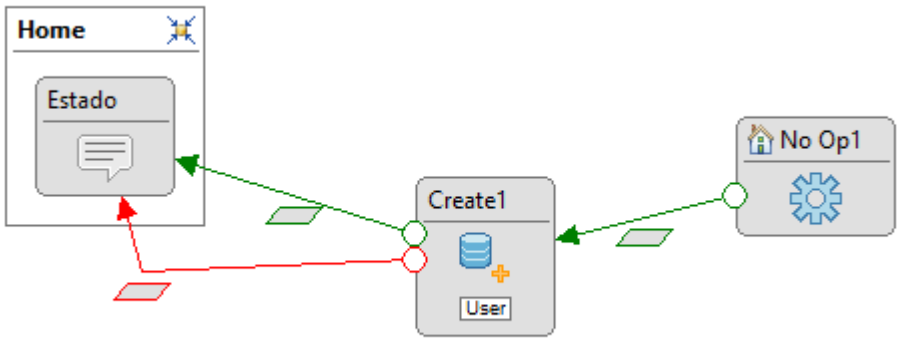


Ilustración 46. Introducción a Apache JMeter. Ejemplo proyecto Create

Es un esquema bastante sencillo, que en este caso hay que seguirlo de derecha a izquierda. El comienzo está definido por una *No Operation Unit que* está marcada como *Home* en sus propiedades y que será lo primero que se ejecute una vez esté desplegada la aplicación web. Este componente lo que hará será pasarle los parámetros necesarios a la *Create Unit* diseñada para que así pueda persistir una tupla en la base de datos de la entidad *User.*

De esta forma, en el *OK Flow que va desde No Op1* hasta *Create1* están los siguientes valores:

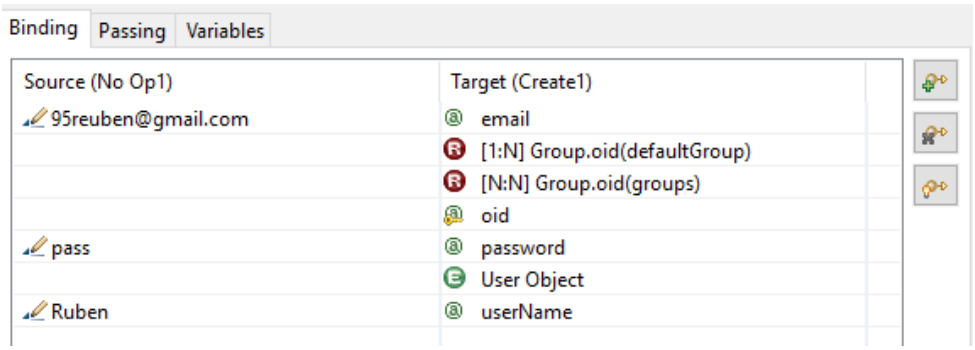


Ilustración 47. Introducción a Apache JMeter. Binding Create Unit

Como resultado de la operación se mostrará en una página *Home* un mensaje. Si el resultado ha sido positivo aparecerá el mensaje “Usuario creado correctamente” y en caso de producirse algún error “Error al crear el usuario”.

Si se arranca el servidor *Tomcat* y se despliega la aplicación el resultado después de lanzar la ejecución es el siguiente:

Ilustración 48. Introducción a Apache JMeter. Ejecución proyecto prueba

Volviendo a Apache JMeter, se procederá a la configuración del *Plan de Pruebas:*

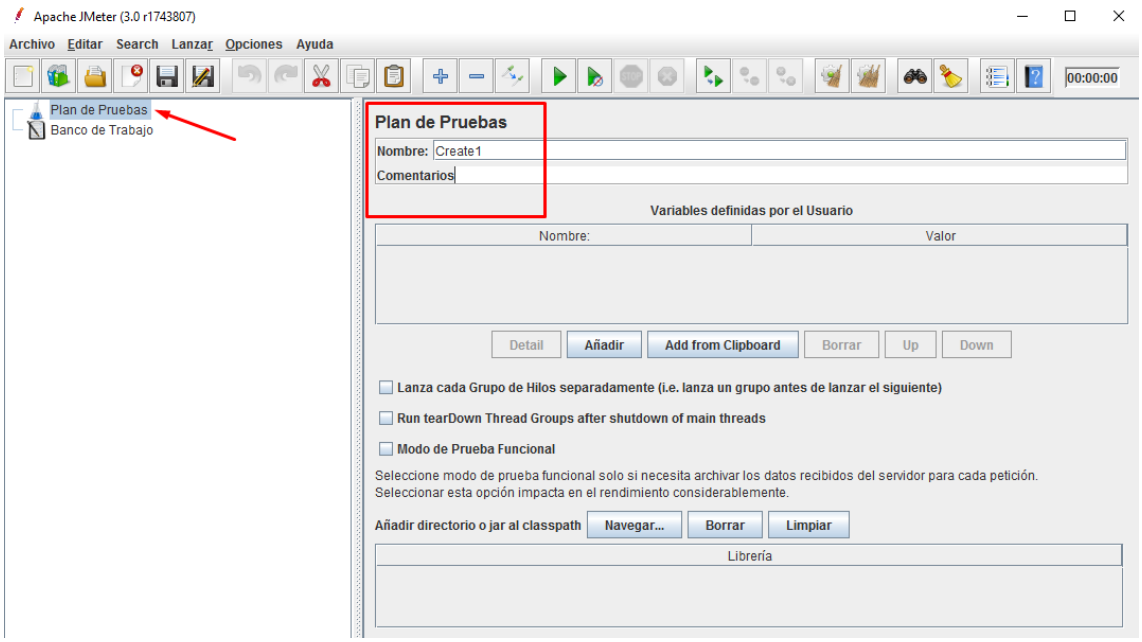


Ilustración 49. Introducción a Apache JMeter. Nuevo Plan de Pruebas

En primer lugar, habrá que añadir algunos componentes para poder grabar las pruebas y posteriormente ejecutarlas.

El primero de esos componentes será el de *Grupo de hilos.* Para ello se hace clic con el botón derecho > Añadir > Hilos (Usuarios) > Grupo de Hilos.

Ilustración 50. Introducción a Apache JMeter. Grupo de Hilos

Este componente permite configurar cuántos usuarios van a solicitar un determinado recurso alojado en una aplicación web.

El *Número de Hilos* indica el número de usuarios que va a haber durante la ejecución de la prueba de rendimiento. Por defecto este valor viene a 1.

El *Periodo de Subida* es una magnitud que se mide en segundos y que indica cuánto tiempo tardará en que se alcance el número de hilos establecido. Por ejemplo, si el número de hilos fuese 300 y el periodo de subida 10 segundos, durante la ejecución de las pruebas tardaría 10 segundos en ponerse la aplicación con 300 hilos simultáneos, mientras tanto la cantidad de hilos activos iría subiendo progresivamente desde 1 hasta 300. También por defecto este valor viene a 1.

El *Contador del bucle* indica cuántas veces va a realizar cada hilo la operación. Si por ejemplo lo que se quiere es que cada hilo realice 10 veces esa operación, el valor de esta variable será 10. Si hay 30 hilos, en total serán 300 operaciones las que se realizarán. Por defecto el valor es 1.

Ahora hay que añadir otro elemento a las pruebas. Con el botón derecho > Añadir > Elemento de configuración > Gestor de Cookies HTTP:

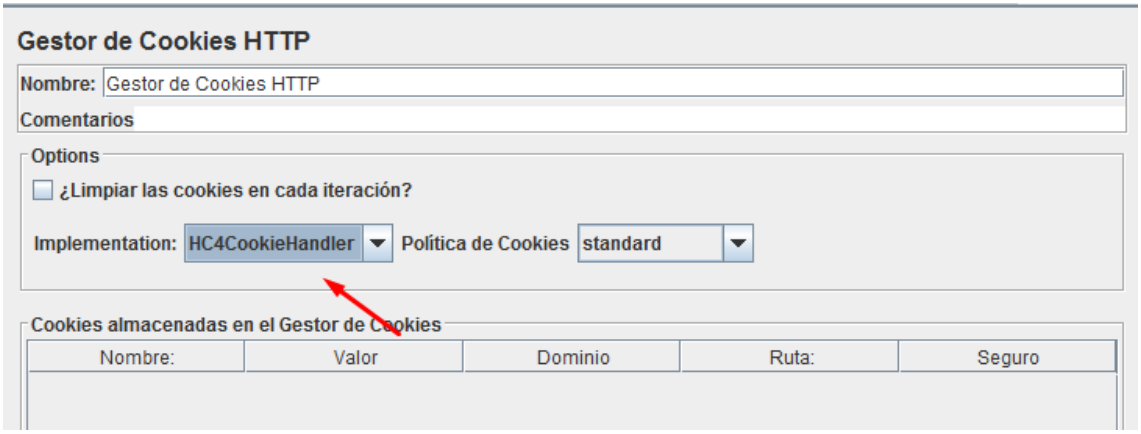


Ilustración 51. Introducción a Apache JMeter. Gestor de Cookies HTTP

Se debe la última versión HC4CookieHandler. La utilidad de este componente es la de gestionar automáticamente las cookies que pudieran generarse durante la ejecución de las pruebas.

A continuación, se añadirá otros componentes relacionados esta vez con los gráficos de las pruebas de rendimiento. Para ello con clic con el botón derecho > Añadir > Receptor > Informe Agregado / Gráfico de Resultados. Estos dos componentes mostrarán información durante la ejecución de las pruebas.

El Informe Agregado muestra datos estadísticos como las siguientes columnas:

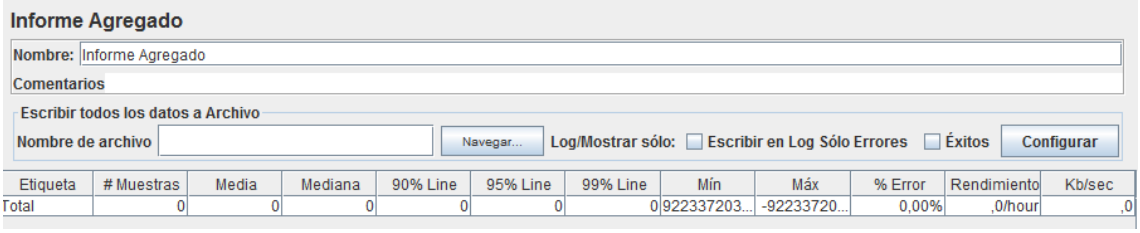


Ilustración 52. Introducción a Apache JMeter. Informe Agregado

Entre los campos más importantes está la Media, el percentir 90%, el Mín, el Máx, el rendimiento y los Kb/sec trasmitidos.

Durante la ejecución de las pruebas serán datos dinámicos que no serán fijos hasta que se completen. Mientras tanto será posible seguir el progreso de algunas de esas variables de forma gráfica con el componente de Gráfico de Resultados:

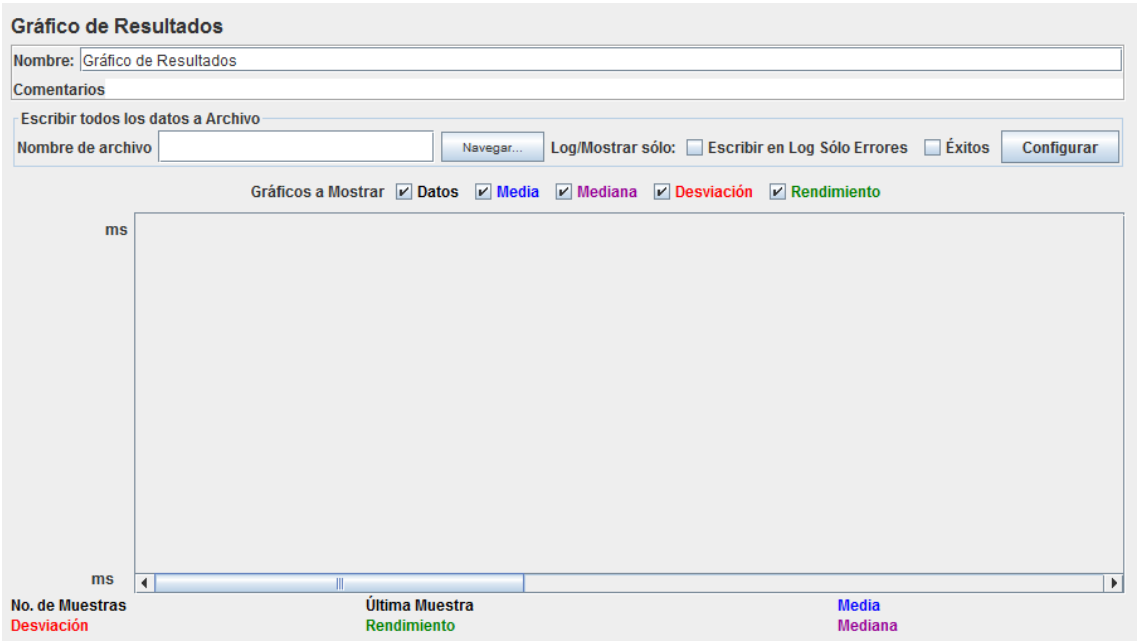


Ilustración 53. Introducción a Apache JMeter. Gráfico de Resultados

Ahora mismo no muestra ningún dato representado, solamente lo hace durante la ejecución, no almacena luego las gráficas resultantes.

Por último, será necesario añadir un componente que se utilizará para grabar nuestras pruebas, en esta ocasión hay que hacer clic con el botón derecho sobre "Banco de Trabajo"> Añadir > Elementos No De Prueba > Servidor Proxy HTTP:

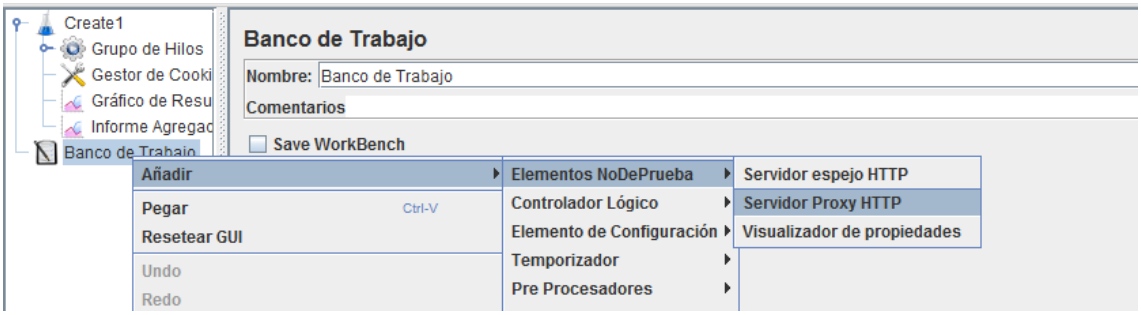


Ilustración 54. Introducción a Apache JMeter. Banco de Trabajo

Este componente será el encargado de construir el plan de pruebas que posteriormente se ejecutará.

El plan de pruebas consiste en un conjunto de peticiones HTTP con sus cabeceras y datos en el cuerpo que sean necesarios. Estas peticiones serán las que se ejecutarán en bucle y de forma concurrente para simular una carga de usuarios concreta en nuestra aplicación.

Se podría introducir manualmente la petición HTTP que va a realizar la operación que se va a probar, sin embargo, hacerlo de esta forma resulta bastante complicado por todos los parámetros que habría que configurar para que la petición se realizase correctamente.

En su lugar lo que se va a hacer es *grabar* una de esas peticiones y después *reproducirla* tantas veces como se haya indicado en la configuración del Grupo de Hilos. Para grabar esa petición será necesario este componente, que hace de servidor proxy entre el usuario y la aplicación, de modo que captura la petición enviada desde el navegador web, la almacena en Apache JMeter y permite que la petición siga su curso hasta llegar al servidor de aplicaciones Tomcat donde está ejecutándose nuestra aplicación (normalmente el puerto será el 8080 de *localhost*).

La configuración de este componente debe ser la siguiente:

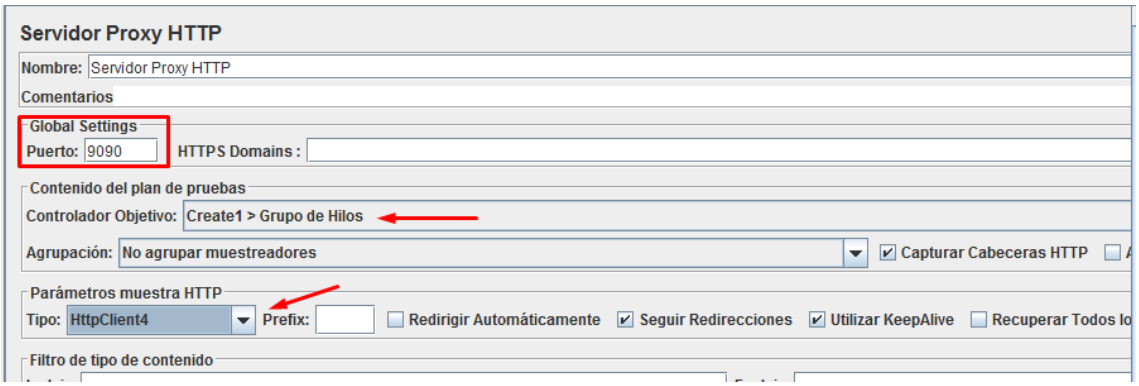


Ilustración 55. Introducción a Apache JMeter. Servidor Proxy HTTP

Se hará uso de un puerto que no sea el 8080, por ejemplo, el 9090, para prevenir posibles conflictos que pueda haber con la ejecución de nuestra aplicación. El Controlador Objetivo será el Grupo de Hilos del Plan de Pruebas, y también se utilizará el HttpClient4, que es la última versión.

Adicionalmente habrá que realizar una configuración en Windows. Desde Panel de Control > Opciones de Internet > Conexiones:

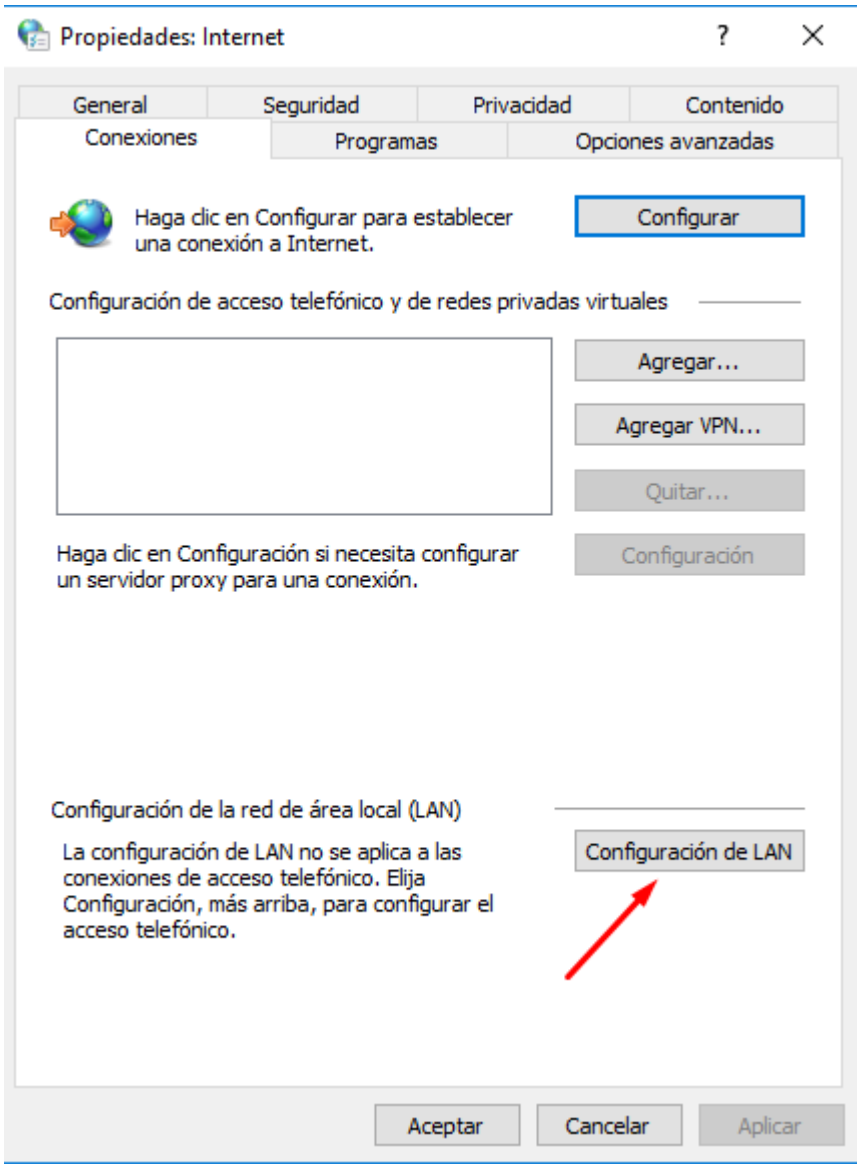


Ilustración 56. Introducción a Apache JMeter. Configuración de LAN

Haciendo clic en Configuración de LAN, se abrirá una nueva ventana:

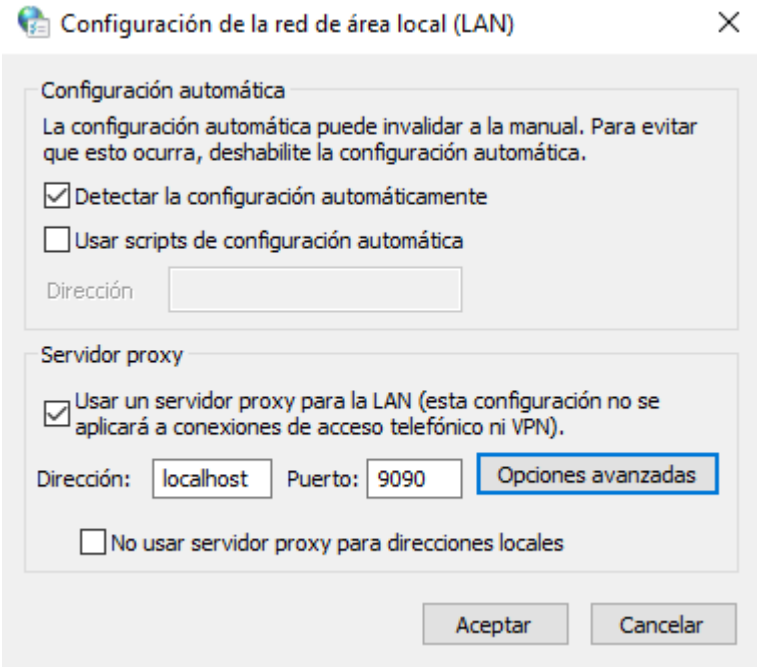


Ilustración 57. Introducción a Apache JMeter. Configuración de servidor proxy

Será necesario activar la opción de Servidor proxy y poner la dirección y puerto que ya se estableció en la configuración del elemento en Apache JMeter. Como se va a trabajar con direcciones locales es importante dejar desmarcada la opción de "No usar servidor proxy para direcciones locales".

Con Aceptar ya estará el Servidor Proxy totalmente configurado, y se podrá comenzar la grabación de la petición.

Conectando nuestra base de datos mediante el SGBD de pgAdmin, se desplegará nuestra aplicación y finalmente desde Apache JMeter se pulsará en el botón "Arrancar".

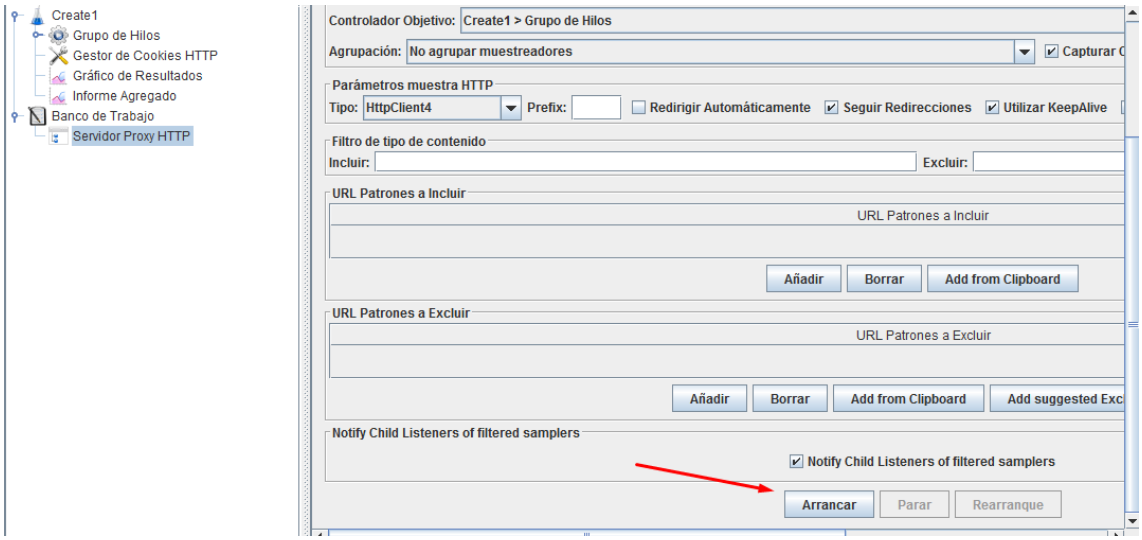


Ilustración 58. Introducción a Apache JMeter. Arranque del servidor proxy

Ahora Apache JMeter capturará todas las peticiones HTTP que se hagan desde el navegador y las almacenará en “Grupo de Hilos”.

Será necesario que muchas de esas peticiones sean borradas porque no son las que se están buscando, y en su lugar solamente se dejarán aquellas que participan en la ejecución de la operación:

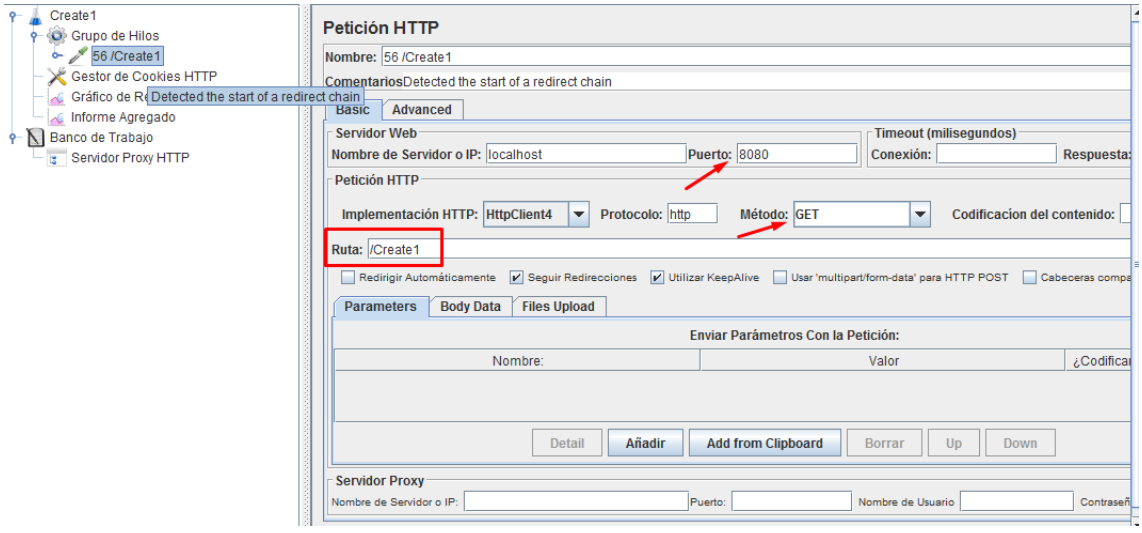


Ilustración 59. Introducción a Apache JMeter. Captura de tráfico

Con esto ya estará creado el Plan de Pruebas listo para ejecutar.

6.1.4. Decisión final sobre el uso de la tecnología

Apache JMeter es sin duda una herramienta bastante completa para la medición del rendimiento de las aplicaciones web. No obstante, hay algunos aspectos del software que no se ajustan bien a nuestras necesidades.

Uno de ellos es la sobrecarga que añade el programa a las propias mediciones, haciendo que éstas tengan una ligera desviación debido a que Apache JMeter actúa como *middleware* entre la aplicación web y el propio usuario o grupo de hilos de usuarios.

Para probar valores altos de concurrencia, es imprescindible subir los valores de la variable Número de Hilos de forma gradual, no se puede probar directamente con 300 hilos porque el software no responde del todo bien y durante las mediciones indica que se han producido errores. Esto provoca que sea bastante costoso llegar al límite de usuarios concurrentes que acepta el sistema.

A veces los valores son bastantes dispares a igualdad de Número de Hilos y contador del bucle, lo que impide tener cierta fiabilidad sobre los resultados devueltos por Apache JMeter.

También hay un inconveniente en cuanto al tratamiento de los datos, y es que el software no almacena todas las mediciones en un archivo o fichero, devuelve un conjunto de estadísticas calculadas, pero no es posible acceder a los datos que se han tenido en cuenta para obtener esos resultados. Para este caso ese aspecto es de vital importancia ya que en el marco del estudio se necesitan filtrar las mediciones realizadas descartando aquellas que sean excepcionales y estén fuera de lo común, es decir, aquellos valores que sean muy dispares de la media y entre ellos haya una varianza elevada no son de interés y no se pretende contemplarlos en el estudio porque distorsionan la información que puede obtenerse. No hay forma de omitirlos en Apache JMeter.

Por estos motivos, no es posible utilizar esta herramienta para el propósito buscado. Aunque podría utilizarse para hacer estimaciones de rendimiento para muchas aplicaciones web, el nivel de precisión necesario es tal que no se puede

sortear qué valores son los más cercanos a los costes temporales reales de realizar una determinada operación.

Es necesario obtener los datos de las mediciones para poder procesarlos y trabajar con ellos, y este software si bien ya procesa automáticamente los datos y ofrece información bastante completa, no permite configurar el cómo lo hace ni acceder a ellos.

6.2. Modificación y creación de nuevos componentes en WebRatio

A continuación, se explicará otra de las alternativas en las que se pensó para medir los tiempos de las operaciones CRUD. Esta alternativa consiste en la creación de nuevos componentes que incorporen un medidor de tiempo e indiquen así cuánto ha costado realizar esa operación. Para esta tarea no se partirá desde cero, se tomará como referencia los componentes ya existentes en WebRatio Platform.

6.2.1. Justificación

Utilizar unos componentes basados en otros componentes de WebRatio reduce la carga añadida de ejecutar simultáneamente otro software como era el caso de Apache JMeter. En este caso solamente se requerirá tener activas la plataforma de WebRatio, el sistema gestor de base de datos y el servidor de aplicaciones Tomcat, lo que hace que los resultados sean más fieles a la intención de aislar computacionalmente las operaciones CRUD, reduciendo el *ruido* de terceras aplicaciones. Ahora no hay intermediario para las mediciones y se trabaja directamente con la plataforma.

6.2.2. Estructura de los componentes

Antes que nada, lo primero que se necesitará será localizar dónde están los componentes de WebRatio. Para ello hay que acceder al directorio de instalación de WebRatio y navegar por el directorio de la plataforma de la versión de WebRatio instalada. Posteriormente se accede a la carpeta WebRatio y luego al directorio de *plugins.*

La estructura en carpetas debería ser parecida a la siguiente:

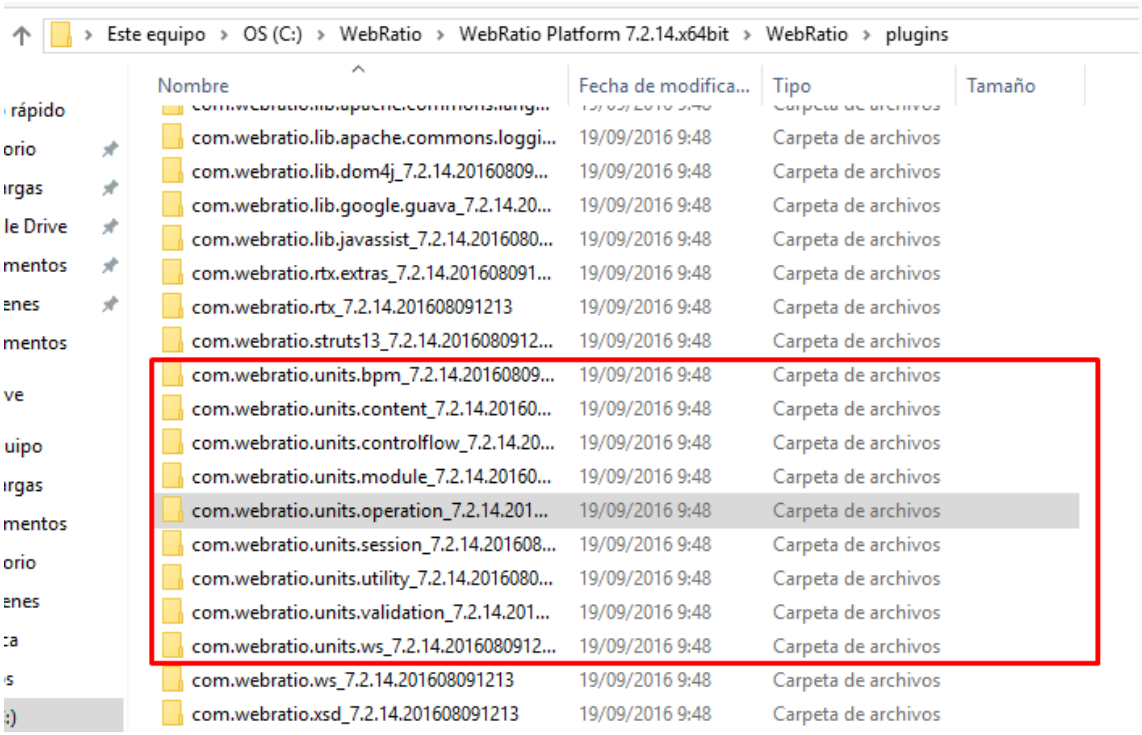


Ilustración 60. Modificación y creación de nuevos componentes en WebRatio. Directorio de plugins de WebRatio

Dentro de la carpeta de *com.webratio.units.operation_* se encontrará lo siguiente:

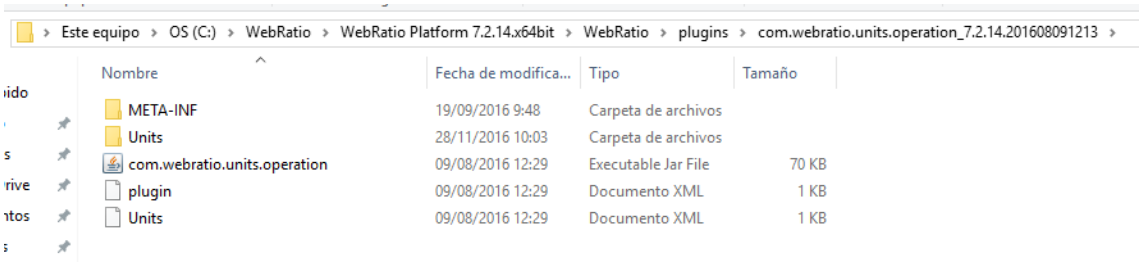


Ilustración 61. Modificación y creación de nuevos componentes en WebRatio. Directorio operation units

El archivo Units.xml es simplemente un archivo en el que se enumeran todos los componentes del tipo *operation*:

Units.xml

```
<Units priority="200" label="Operations">
  <Group>
    <Unit>CreateUnit</Unit>
    <Unit>DeleteUnit</Unit>
    <Unit>ModifyUnit</Unit>
  </Group>
  <Group>
    <Unit>ConnectUnit</Unit>
    <Unit>DisconnectUnit</Unit>
    <Unit>ReconnectUnit</Unit>
  </Group>
  <Group>
    <Unit>StoredProcedureUnit</Unit>
    <Unit>NoOpOperationUnit</Unit>
  </Group>
</Units>
```

Ilustración 62. Modificación y creación de nuevos componentes en WebRatio. Listado de todas las Operation Units en XML

Dentro en la carpeta Units existe una estructura de directorios, en la que cada directorio tendrá el nombre de un componente:

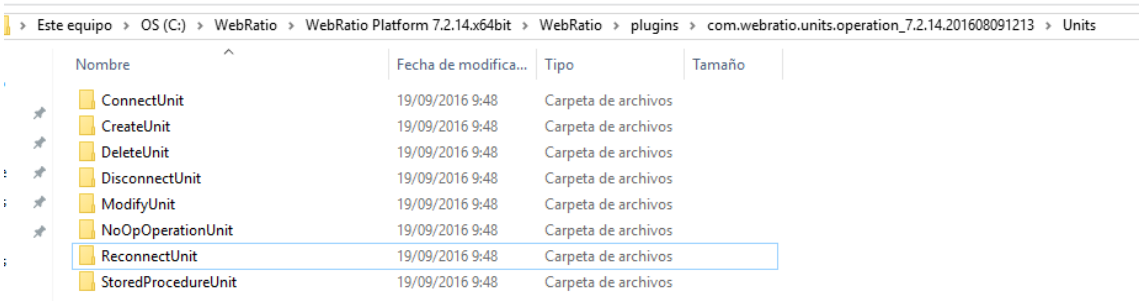


Ilustración 63. Modificación y creación de nuevos componentes en WebRatio. Directorios operation units

Cada uno de los componentes en WebRatio tendrá 3 carpetas y un archivo Unit.xml donde se define el nombre del componente y sus entradas y salidas.

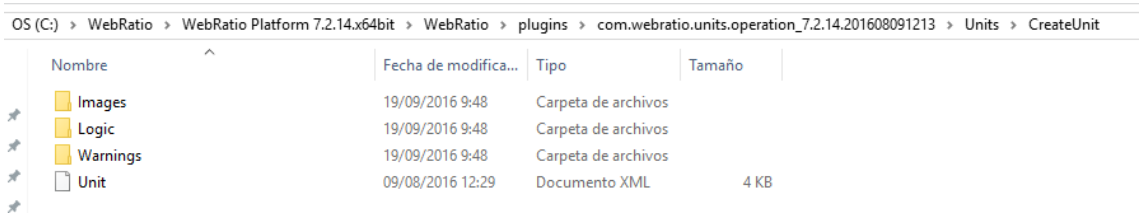


Ilustración 64. Modificación y creación de nuevos componentes en WebRatio. Directorio Create Unit

En Images están los .png que identifican al componente en la plataforma, en este caso:

Ilustración 65. Modificación y creación de nuevos componentes en WebRatio. Miniaturas Create Unit

En Logic se define en 3 archivos las entradas, salida y lógica del sistema:

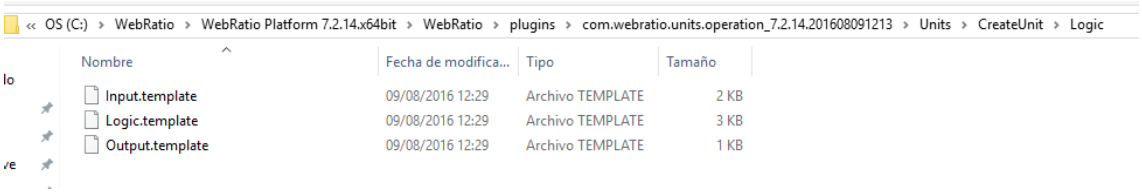


Ilustración 66. Modificación y creación de nuevos componentes en WebRatio. Create Unit .TEMPLATES

Estos archivos .TEMPLATE tienen una sintaxis basada en XML.

6.2.3. Modificación de los componentes

En total se necesitan un total de cuatro componentes nuevos, uno por cada operación CRUD. El procedimiento es el mismo para todos ellos, la única salvedad es que para la operación de Read/Retrieve el componente necesario no se encuentra en la carpeta de *operation*, sino en la de *utility:*

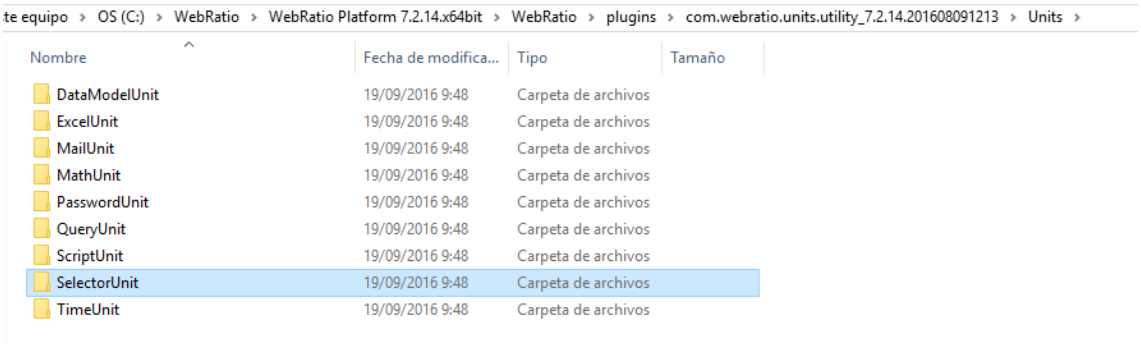


Ilustración 67. Modificación y creación de nuevos componentes en WebRatio. Directorios de Utility Units

Como ejemplo se va a explicar la creación del componente modificado con la operación de Create.

En primer lugar, hay que crear una copia de la carpeta CreateUnit, y se denominar CreateMod.

A continuación, se debe modificar los .png que identifican en el framework de WebRatio la unidad adicional que se va a crear. Con un editor gráfico como Gimp se puede, por ejemplo, invertir los colores de ambas imágenes, y así no habrá confusión entre el componente de *créate* nuevo y el original:

Ilustración 68. Modificación y creación de nuevos componentes en WebRatio. Miniaturas modificadas de la Create Unit

Luego es necesario abrir el archivo Unit.xml y cambiar el *label* del nuevo componente:

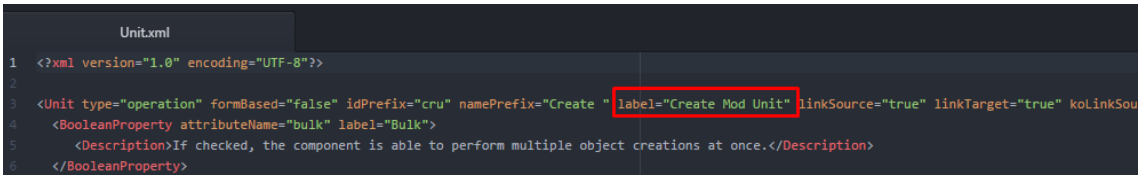


Ilustración 69. Modificación y creación de nuevos componentes en WebRatio. Modificación del label del componente

Hecho esto, solo quedará modificar el archivo *Logic.template* que se encuentra dentro de la carpeta *Logic*, y añadir los contadores de tiempo que se utilizarán para calcular el coste temporal de la operación. El lenguaje utilizado será Groovy, un lenguaje muy similar a Java, pero con un nivel mayor de abstracción.

Se mostrará al comienzo un mensaje por consola indicando el comiendo de la prueba y posteriormente se guardará en una variable *time_start* el tiempo en nanosegundos del sistema usando la función *nanoTime*(), que es la máxima precisión que se puede obtener:

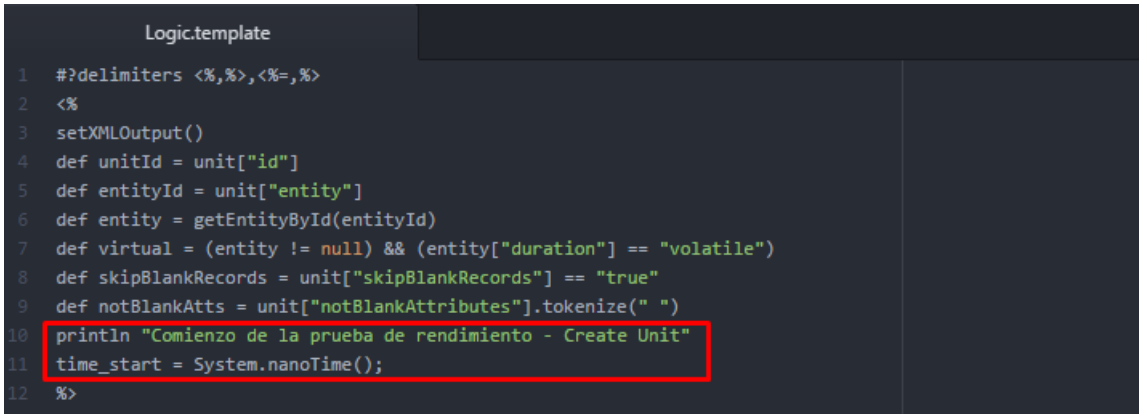


Ilustración 70. Modificación y creación de nuevos componentes en WebRatio. Primer temporizador

Al final se añade un segundo contador de tiempo *time_end* y se almacena la diferencia en una variable *time* que será la mostrada por consola. Así es como se obtiene el coste temporal de la operación.

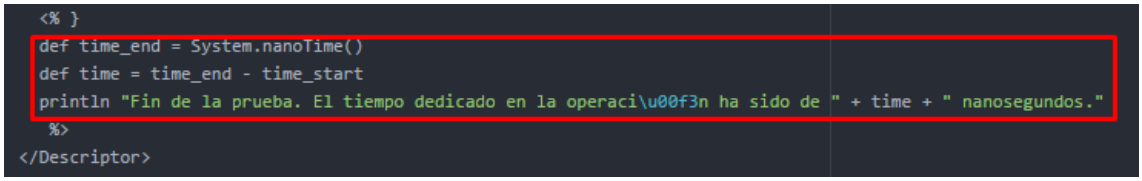


Ilustración 71. Modificación y creación de nuevos componentes en WebRatio. Segundo temporizador

Con esto ya está creada la nueva unidad de la operación *create.* La operación y utilidad es la misma que la de la unidad original, pero está ampliada con la información capturada y mostrada a la hora de realizar la operación. Con las otras 3 operaciones el procedimiento es idéntico, pero las unidades que habrá que replicar

serán otras. En el caso de *delete* se duplicará y modificará la DeleteUnit, en el caso de *update* será la ModifyUnit y en el caso de *read* será la SelectorUnit.

6.2.4. Pruebas de los nuevos componentes

Ahora se va a realizar una prueba el componente creado. Accediendo al entorno de WebRatio se puede observar que en los componentes *Operations* hay las tres nuevas unidades que correspondientes a las recién creadas.

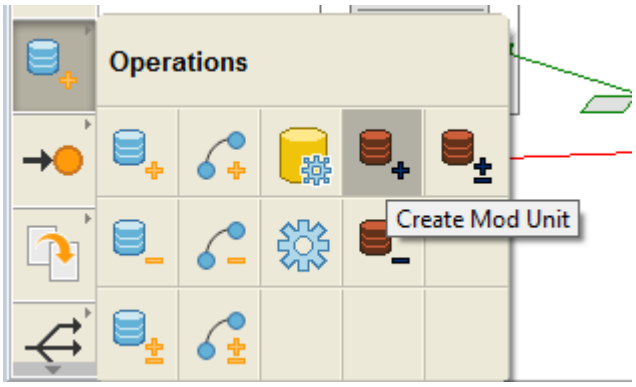


Ilustración 72.Modificación y creación de nuevos componentes en WebRatio. Componentes modificados

Lo mismo ocurre con los componentes *Utility*, aparecerá uno nuevo:

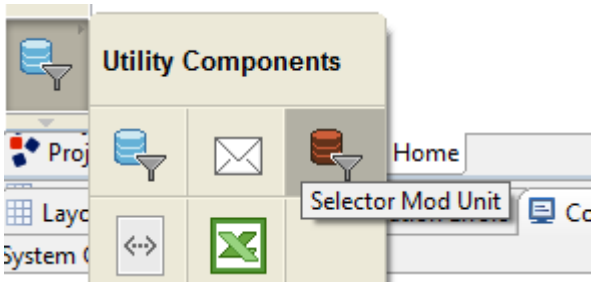


Ilustración 73. Modificación y creación de nuevos componentes en WebRatio. Componente modificado - 2

Una vez comprobado esto, se creará un proyecto base en WebRatio con el siguiente esquema IFML:

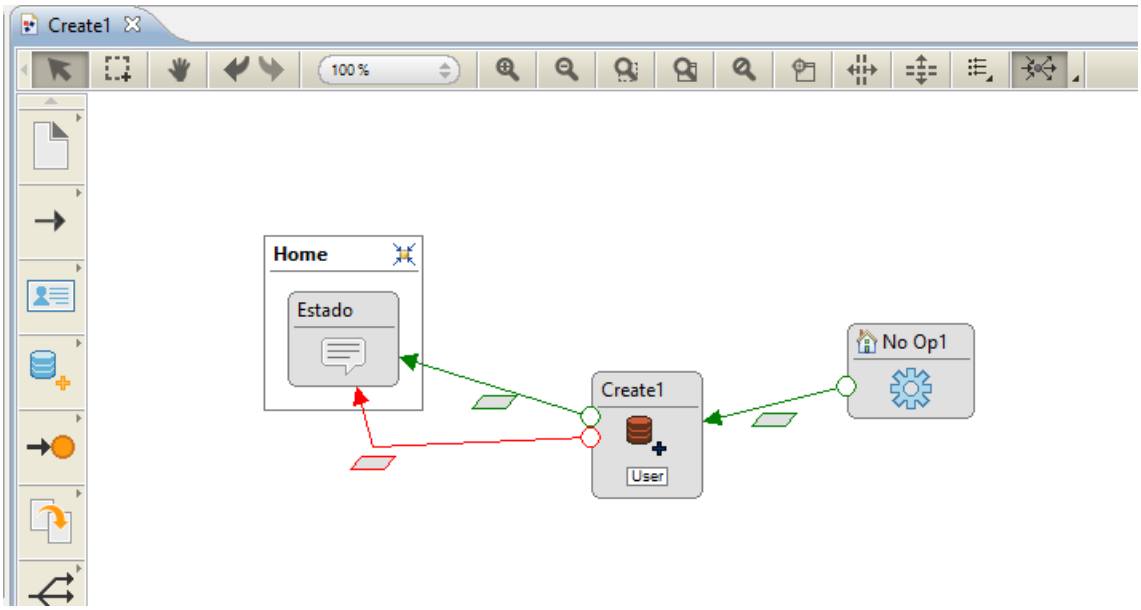


Ilustración 74. Modificación y creación de nuevos componentes en WebRatio. Uso de Create Unit modificada

Es un esquema bastante sencillo, que en este caso hay que seguirlo de derecha a izquierda. El comienzo está definido por una *No Operation Unit que* está marcada como *Home* en sus propiedades y que será lo primero que se ejecute una vez esté desplegada la aplicación web. Este componente lo que hará será pasarle los parámetros necesarios a la *Create Unit* diseñada para que así pueda persistir una tupla en la base de datos de la entidad *User.*

De esta forma, en el *OK Flow que va desde No Op1* hasta *Create1* tendrá los siguientes valores:

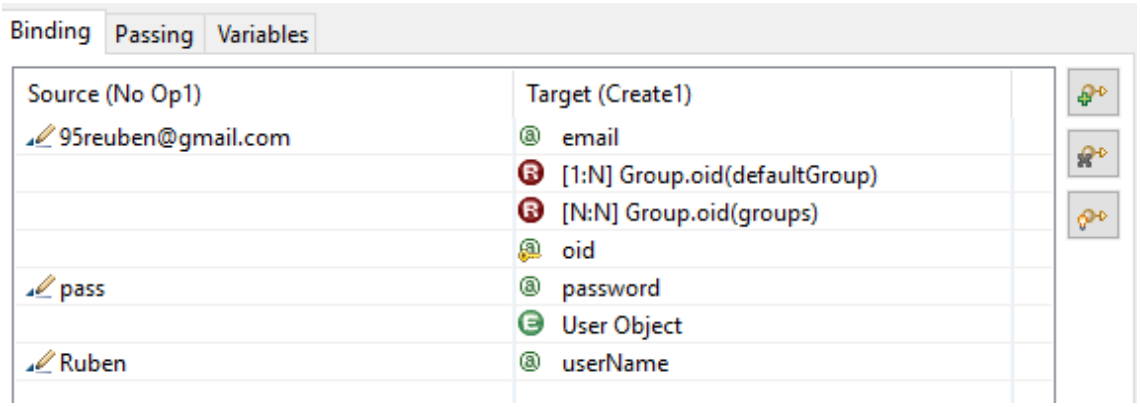


Ilustración 75. Modificación y creación de nuevos componentes en WebRatio. Campos Create Unit modificada

Como resultado de la operación se mostrará en una página *Home* un mensaje. Si el resultado ha sido positivo mostrará el mensaje “Usuario creado correctamente” y en caso de producirse algún error “Error al crear el usuario”.

Arrancando el servidor *Tomcat* y desplegando la aplicación el resultado después de lanzar la ejecución es el siguiente:

Ilustración 76. Modificación y creación de nuevos componentes en WebRatio. Prueba de la aplicación

Según el mensaje devuelto la operación se ha realizado correctamente. En la consola del entorno se puede ver el siguiente mensaje:

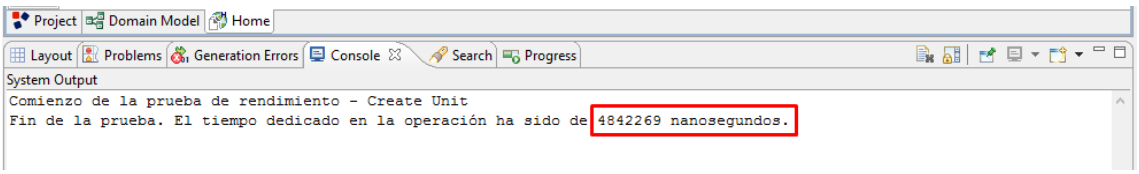


Ilustración 77. Modificación y creación de nuevos componentes en WebRatio. Tiempos en nanosegundos

El coste temporal ha sido de 4842269 nanosegundos. Si se toma el proyecto base que proporciona WebRatio y no se modifican las opciones de persistencia del modelo de datos (persistente en base de datos por defecto), se podrá comprobar también la correcta inserción en la base de datos desde el gestor *pgAdmin:*

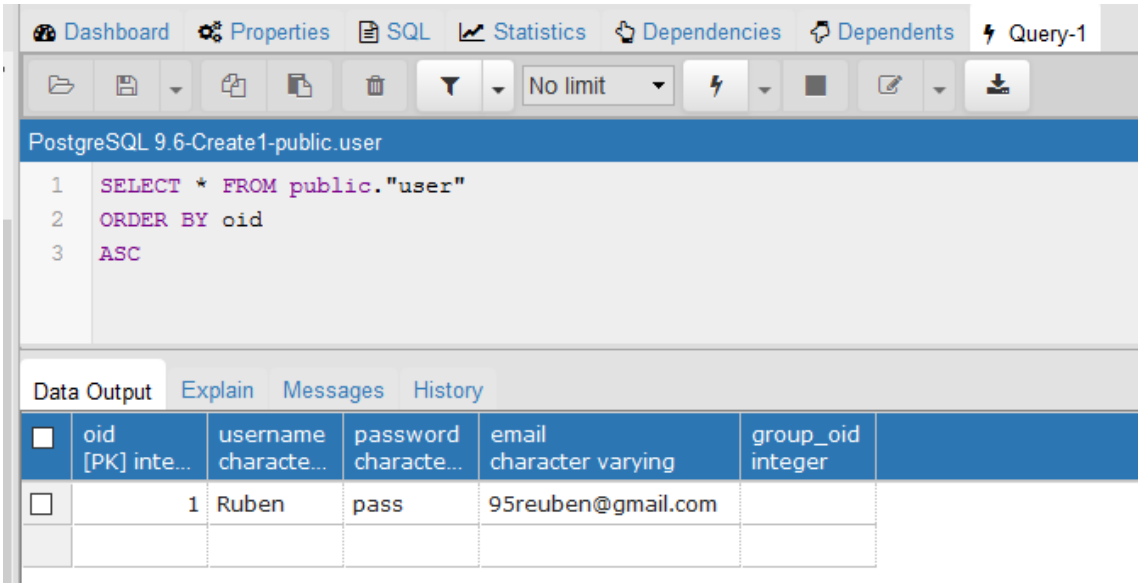


Ilustración 78. Modificación y creación de nuevos componentes en WebRatio. Modificación en la base de datos

6.2.5. Decisión final sobre esta técnica

Realizar las mediciones de tiempo utilizando nuevos componentes modificados para enfocarlos a tal fin es una opción bastante interesante, sin embargo, hay una serie de limitaciones que se verán a continuación y que pueden obligar a descartar esta vía.

La primera de ellas es la dificultad de hacer en una misma ejecución mediciones de tiempo no de una única inserción, sino de varias, por ejemplo 100, 200… y ver así cómo evoluciona el coste medio por inserción. Intentando modificar el archivo logic.TEMPLATE añadiendo un bucle *for* al cuerpo de su código no he conseguido que en la ejecución se realicen tantas inserciones como iteraciones haga el bucle. A día de hoy desconozco la razón de ello, una posible hipótesis puede ser que WebRatio prevenga esa situación de bucles de forma que la parte lógica de un componente sólo pueda ejecutarse una vez por ejecución de la aplicación web.

Otra dificultad de esta técnica es el hecho de trabajar con el dato resultante mostrado por consola. He probado sin éxito varias formas de intentar hacer visible a los componentes de la aplicación la variable *time* que devuelve los nanosegundos que ha costado realizarla. No sé si es posible hacer eso desde los .TEMPLATE de los componentes en WebRatio, pero al no haberlo podido conseguir, no podía guardar por ejemplo ese resultado en una tabla de la base de datos y posteriormente hacer

estadísticas con todos ellos. En su lugar sólo podía ir almacenando de forma manual los datos que aparecen por consola en sucesivas ejecuciones.

Al no conseguir realizar bucles de esta forma, no se puede probar bien el rendimiento en operaciones con varias repeticiones, es decir, únicamente se puede probar el rendimiento de una inserción, pero no el de 100 inserciones, por ejemplo, donde previsiblemente el resultado no sería proporcional.

Por estos motivos esta técnica ha sido descartada para obtener las mediciones de rendimiento. Es una técnica interesante para probar inserciones unitarias, pero no es cómoda para realizar un estudio de muchas de ellas ni factible para realizar mediciones de conjuntos de inserciones.

7. Resultados y discusión

En esta sección se van a presentar las conclusiones obtenidas a partir de los resultados de las ejecuciones de los proyectos y su posterior procesamiento en RStudio y su análisis.

7.1. Small vs Micro

A continuación, se exponen los resultados obtenidos para un único usuario al realizar la cadena de operaciones en dos máquinas virtuales con especificaciones distintas.

7.1.1. Especificaciones

	Amazon t2.small	Amazon t1.micro
RAM	2GB	1GB
Sistema Operativo	Linux	Linux
Procesador	Intel Xeon	Intel Xeon
CPUs	2,5 GHz	2,5 GHz

Tabla 1. Especificaciones máquinas Small vs Micro

7.1.2. Operaciones

PERSISTENT

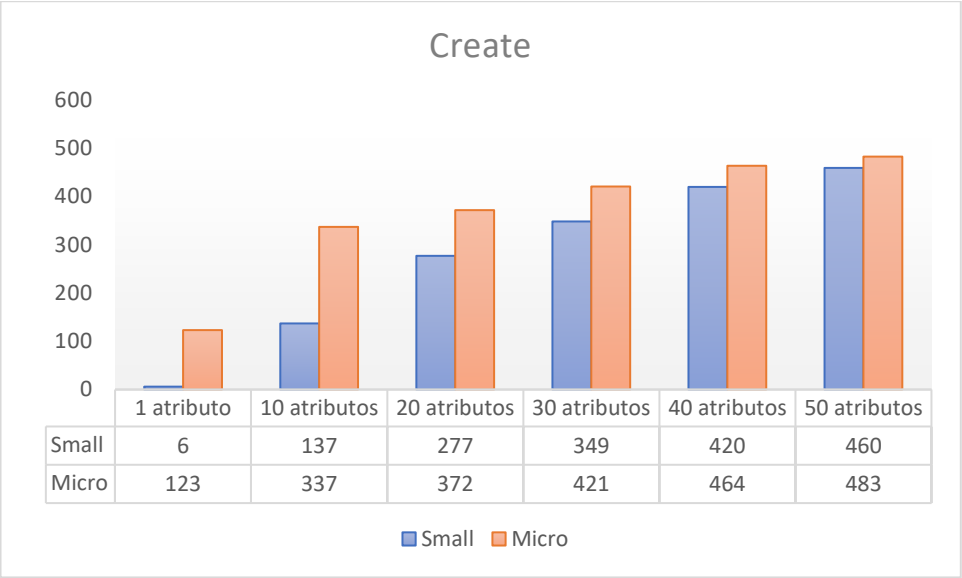


Ilustración 79. Small vs Micro. Persistencia. Create

En la gráfica se puede observar que los tiempos en ambas máquinas tienden a converger cuanto mayor sea el número de atributos.

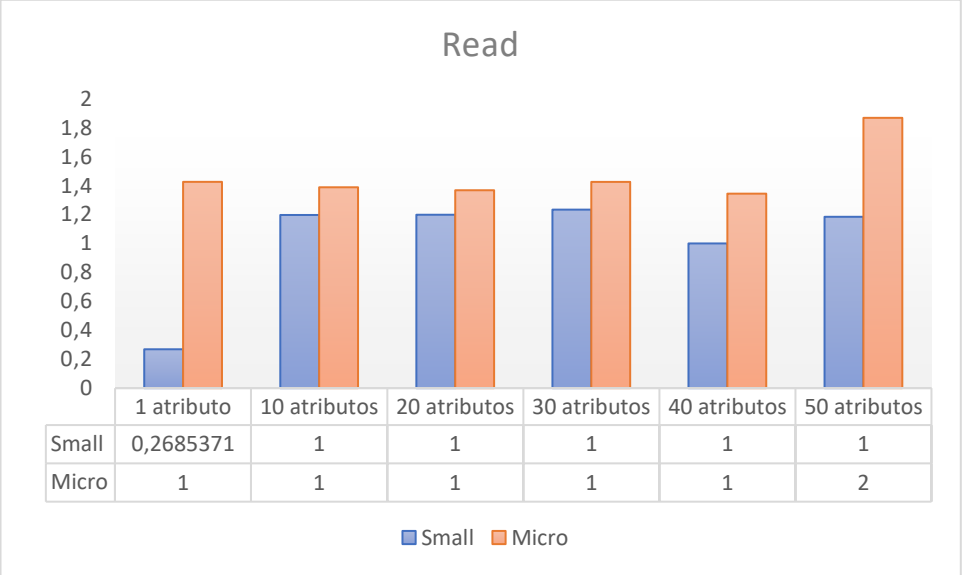
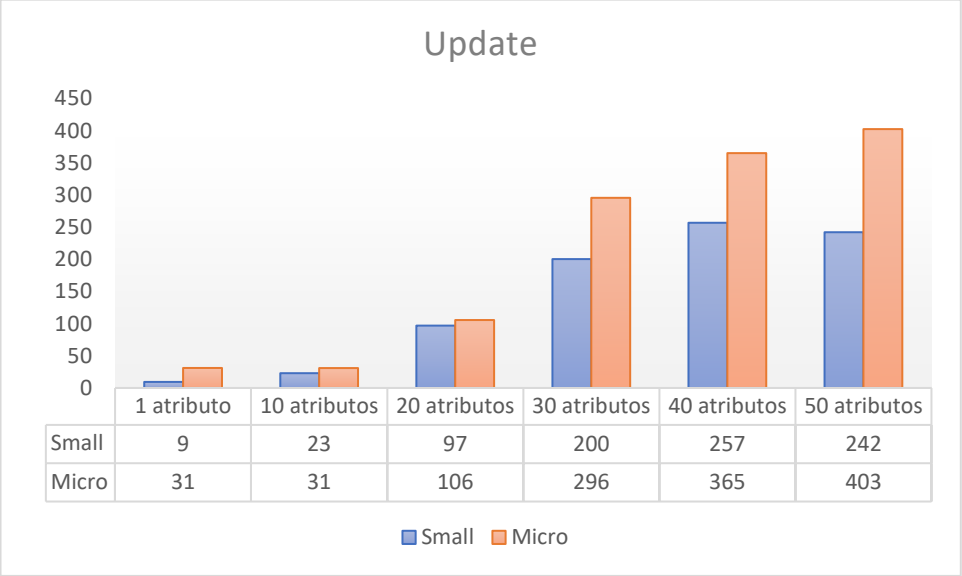


Ilustración 80. Small vs Micro. Persistencia. Read

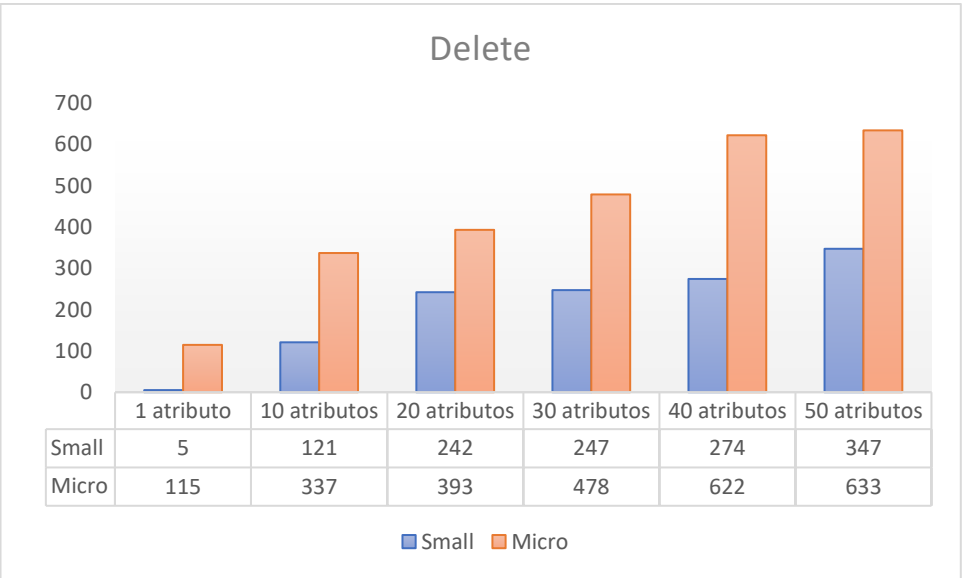
Aquí se puede observar que en el caso de las lecturas los tiempos son en general muy similares en ambas máquinas salvo para los casos de 1 y 50 atributos.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
Small	9	23	97	200	257	242
Micro	31	31	106	296	365	403

Ilustración 81. Small vs Micro. Persistencia. Update

Para esta operación no hay grandes diferencias entre ambas configuraciones, no obstante, parece ser que la variación de tiempos es más notoria cuanto mayor sea el número de atributos a actualizar.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
Small	5	121	242	247	274	347
Micro	115	337	393	478	622	633

Ilustración 82. Small vs Micro. Persistencia. Delete

En cuanto al borrado, la diferencia de tiempos se incrementa conforme aumenta el número de atributos.

SESSION

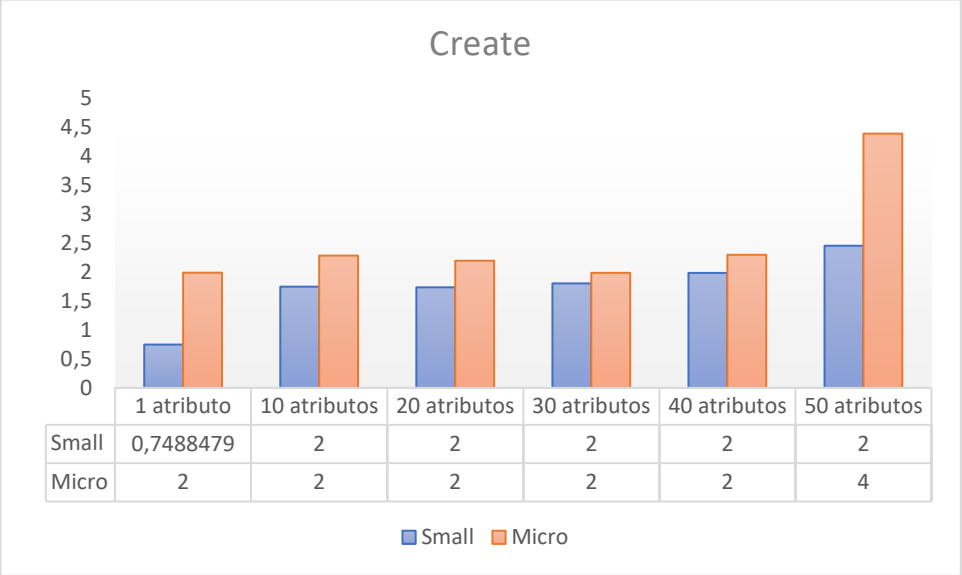
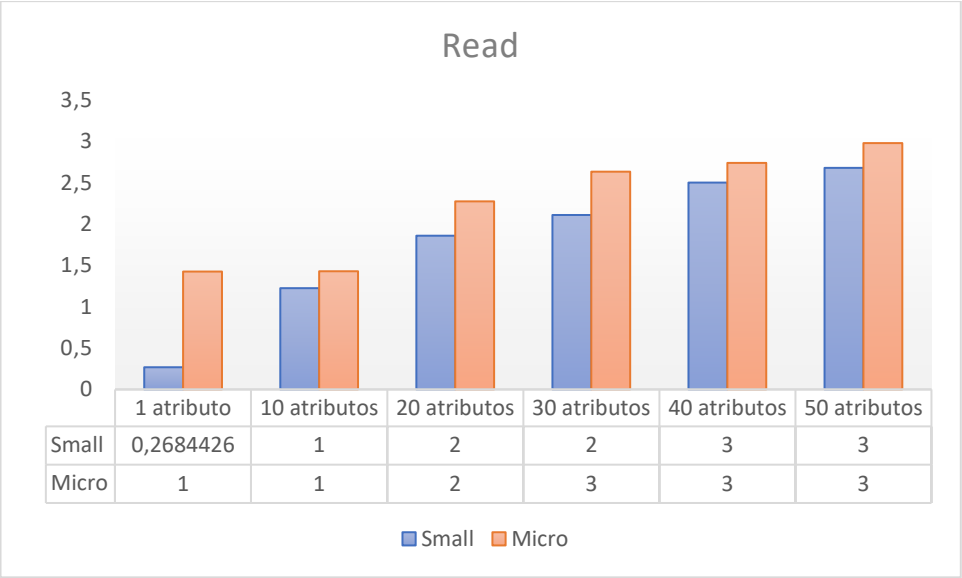


Ilustración 83. Small vs Micro. Session. Create

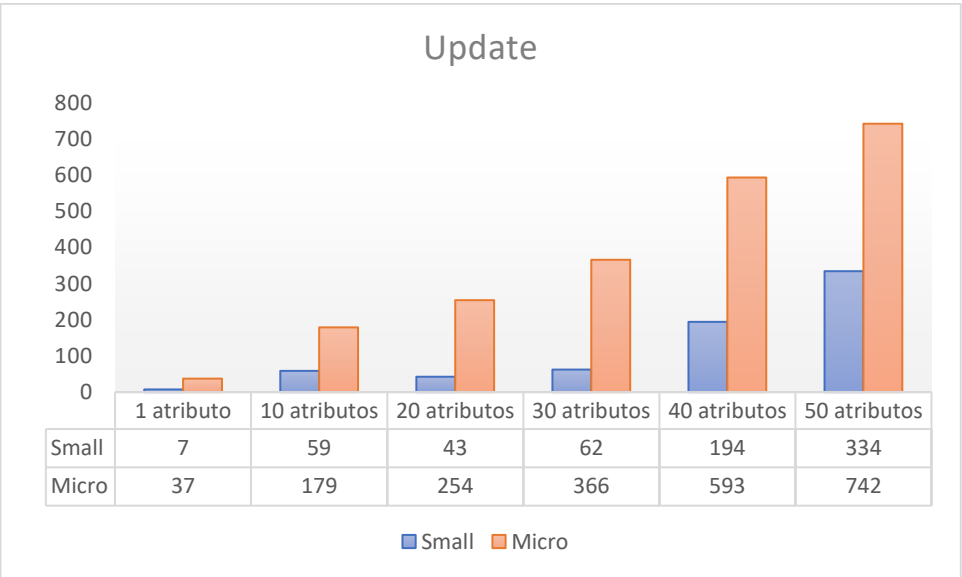
La operación de creación en este caso suele tener unos tiempos muy similares en ambas máquinas salvo para el caso de que el número de atributos sea 1 o 50.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
Small	0,2684426	1	2	2	3	3
Micro	1	1	2	3	3	3

Ilustración 84. Small vs Micro. Session. Read

En el caso de la lectura los tiempos son en general muy similares salvo en el caso de 1 atributo que es donde hay mayor divergencia a favor de la máquina más potente.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
Small	7	59	43	62	194	334
Micro	37	179	254	366	593	742

Ilustración 85. Small vs Micro. Session. Update

En esta operación la divergencia es cada vez mayor conforme aumenta el número de atributos, está claro que la máquina más potente es mucho más rápida y conveniente.

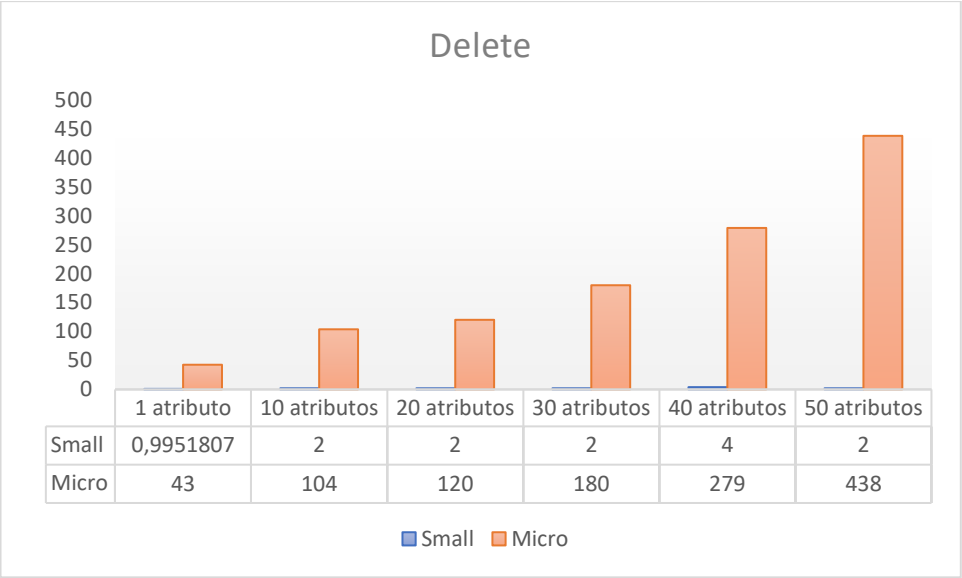


Ilustración 86. Small vs Micro. Session. Delete

En este caso los tiempos de la máquina small apenas varían según aumentan el número de atributos, sin embargo, en la micro la tendencia es alcista, lo cual hace mucho más adecuada la primera máquina para esta configuración en esta operación.

APPLICATION

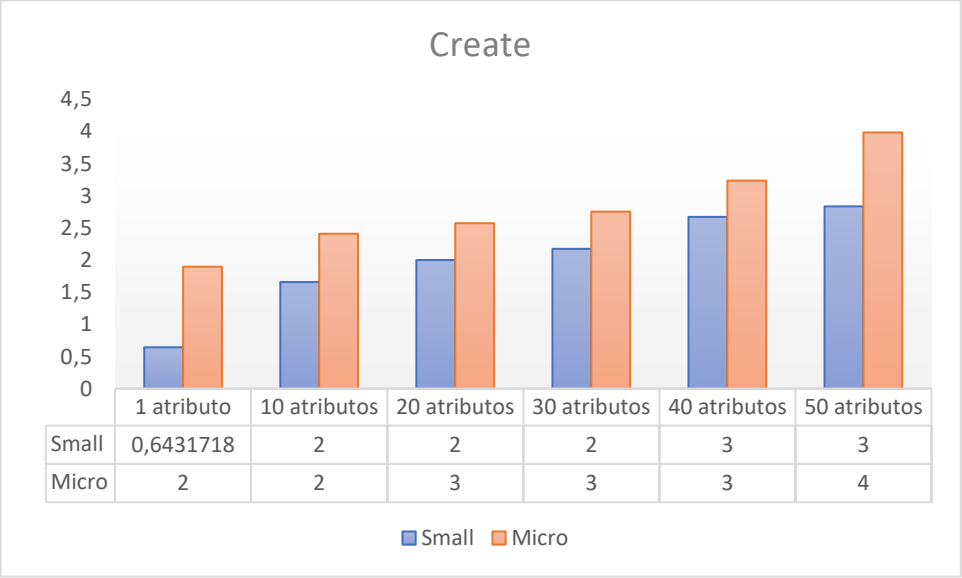
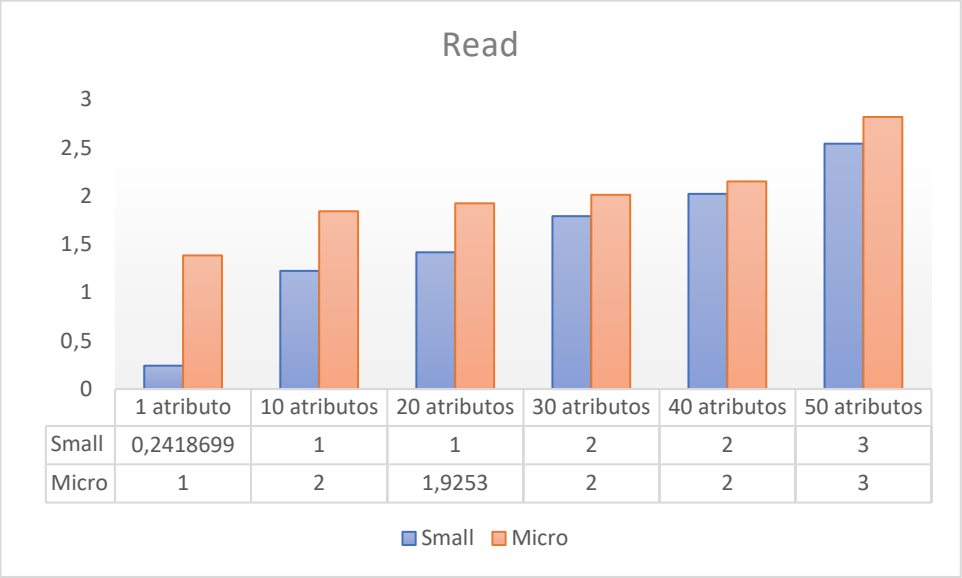


Ilustración 87. Small vs Micro. Application. Create

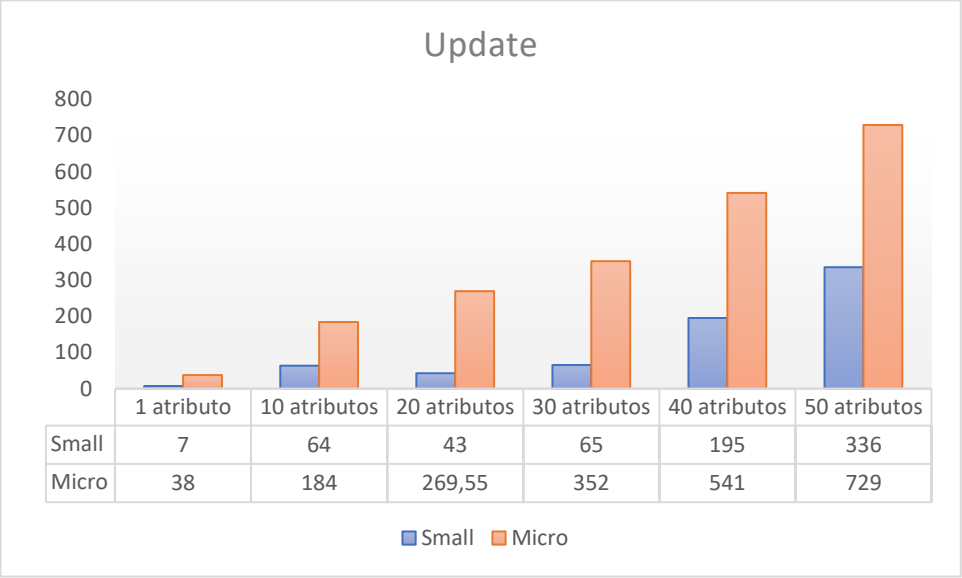
Los tiempos entre ambas máquinas no suelen variar demasiado y la tendencia final es más ventajosa para la máquina small, que es más potente.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
Small	0,2418699	1	1	2	2	3
Micro	1	2	1,9253	2	2	3

Ilustración 88. Small vs Micro. Application. Read

Los tiempos de lectura no suelen ser muy diferentes salvo en el comienzo de la gráfica, luego apenas hay diferencias entre ambas configuraciones.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
Small	7	64	43	65	195	336
Micro	38	184	269,55	352	541	729

Ilustración 89. Small vs Micro. Application. Update

En esta operación en esta configuración la divergencia es cada vez mayor según aumenta el número de atributos, la máquina small es por tanto una mejor elección en términos de rendimiento.

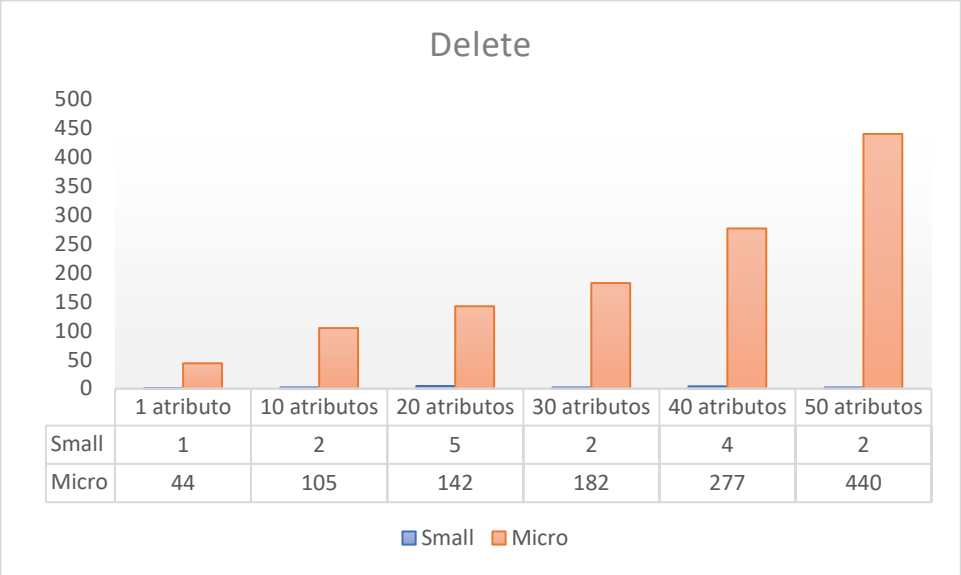


Ilustración 90. Small vs Micro. Application. Delete

Mientras los tiempos en la máquina small apenas varían y se mantienen en valores muy bajos, en la micro van aumentando considerablemente conforme el número de atributos es mayor.

CREATE

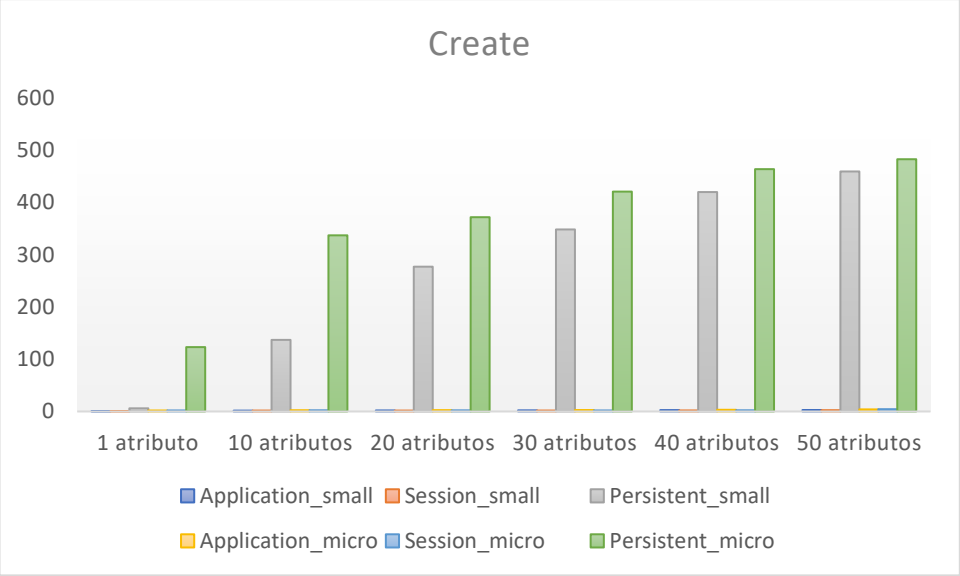


Ilustración 91. Small vs Micro. Create. Barras

Se puede apreciar que para la operación de creación los tiempos en los tipos de persistencia volátiles (session y application) son muy similares. Estos tiempos son mucho menores que los resultantes de la persistencia en base de datos, que tienden a converger según aumenta el número de atributos.

Podría entonces afirmarse que la persistencia volátil puede ser una opción bastante interesante y mucho más eficiente si el modelo de datos en cuestión es compatible con las reglas de negocio. Si es necesario persistir los datos en una base de datos esta ventaja no podrá ser aprovechada.

READ

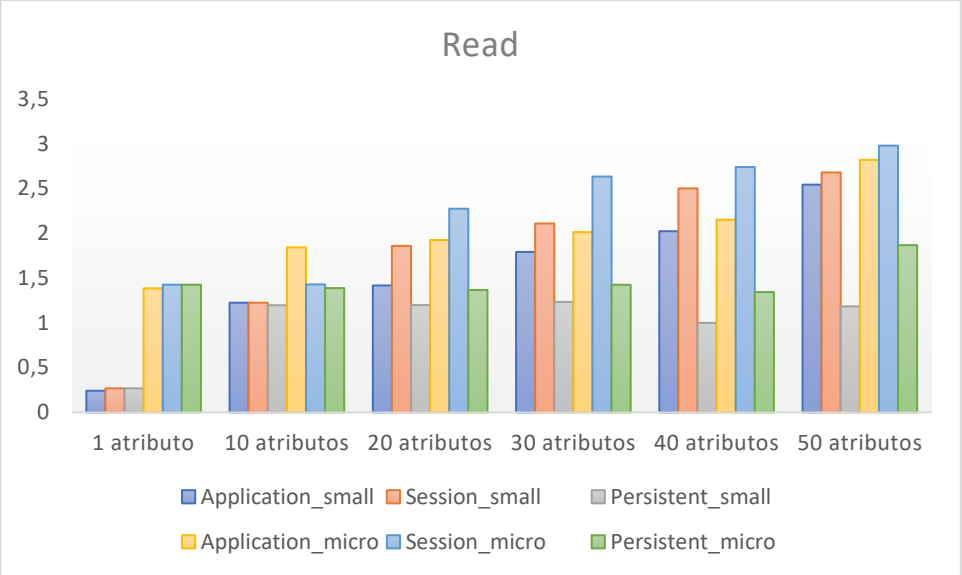


Ilustración 92. Small vs Micro. Read. Barras

En esta operación se puede observar que la configuración más costosa en tiempo es la de persistencia volátil de sesión, con una tendencia alcista. En el lado opuesto se encuentra la persistencia en base de datos, que en ambas máquinas no ha aumentado considerablemente conforme aumentaban los atributos y se encuentran en la parte baja de la gráfica en cuanto a coste temporal.

Por tanto, una conclusión que puede obtenerse al respecto es que los tiempos de lectura son sustancialmente menores en los casos de persistencia en la base de datos. Si es compatible con el modelo de datos, tener entidades persistidas con esta configuración será mucho más eficiente en términos de lectura.

UPDATE

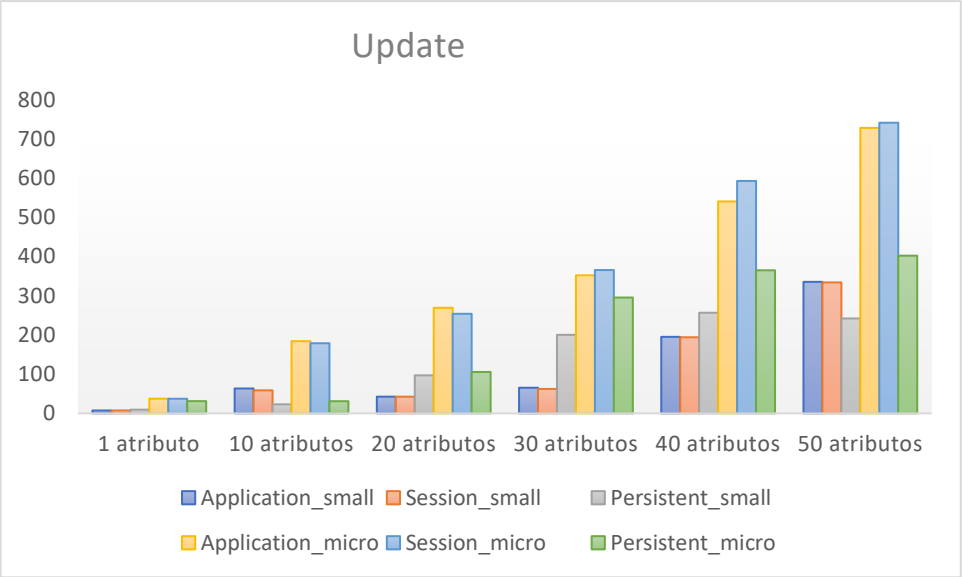


Ilustración 93. Small vs Micro. Barras

La configuración de la máquina small hace que la diferencia de tiempos entre la persistencia volátil y la persistencia de base de datos sea muy pequeña. Sin embargo, esta diferencia llega a ser notablemente mayor en el caso de la máquina micro, por lo que no sería recomendable esa configuración hardware.

DELETE

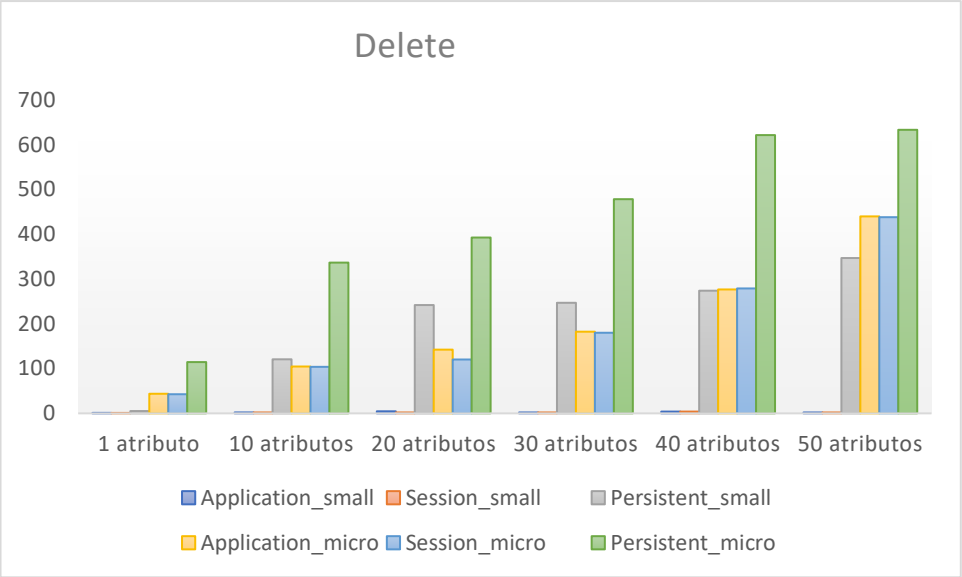


Ilustración 94. Small vs Micro. Delete. Barras

La configuración de persistencia volátil en la máquina small es bastante menos costosa que el resto de configuraciones para esta operación. La más costosa puede apreciarse que es la de persistencia en base de datos.

Así la configuración más recomendable es la de persistencia volátil para la máquina small, ya que sus tiempos apenas varían conforme aumenta el número de atributos.

7.2. 1 vs 5 usuarios

En este caso se compara el rendimiento en la misma máquina de la capa gratuita de Amazon entre un usuario y 5 usuarios concurrentes.

7.2.1. Especificaciones

	Amazon t1.micro
RAM	1GB
Sistema Operativo	Linux
Procesador	Intel Xeon
CPUs	2,5 GHz

Tabla 2. Especificaciones máquinas 1 vs 5 usuarios

7.2.2. Operaciones

PERSISTENT

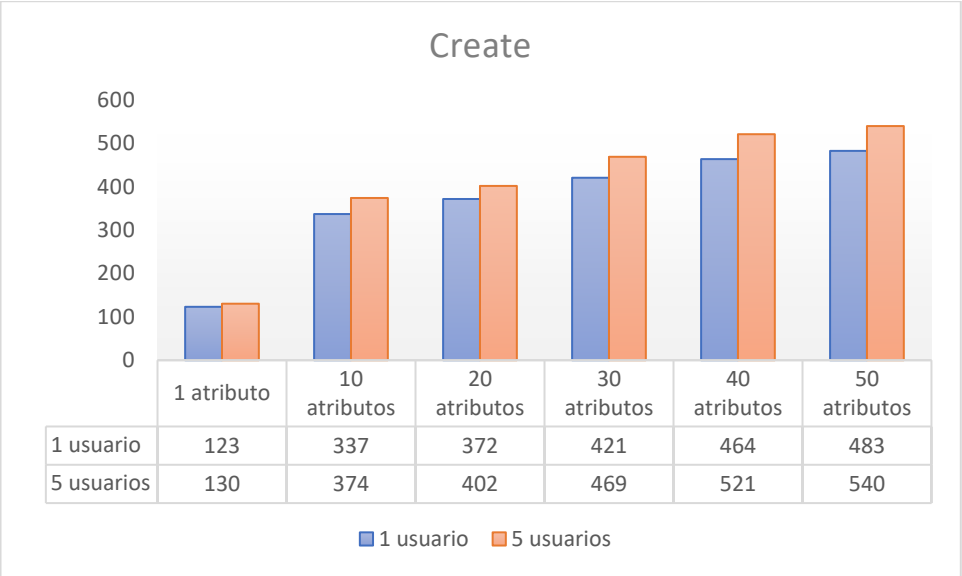


Ilustración 95. 1 vs 5 usuarios. Persistencia. Create

En este caso los tiempos medios son muy similares entre ambos casos, apenas se aprecia varianza por la concurrencia.

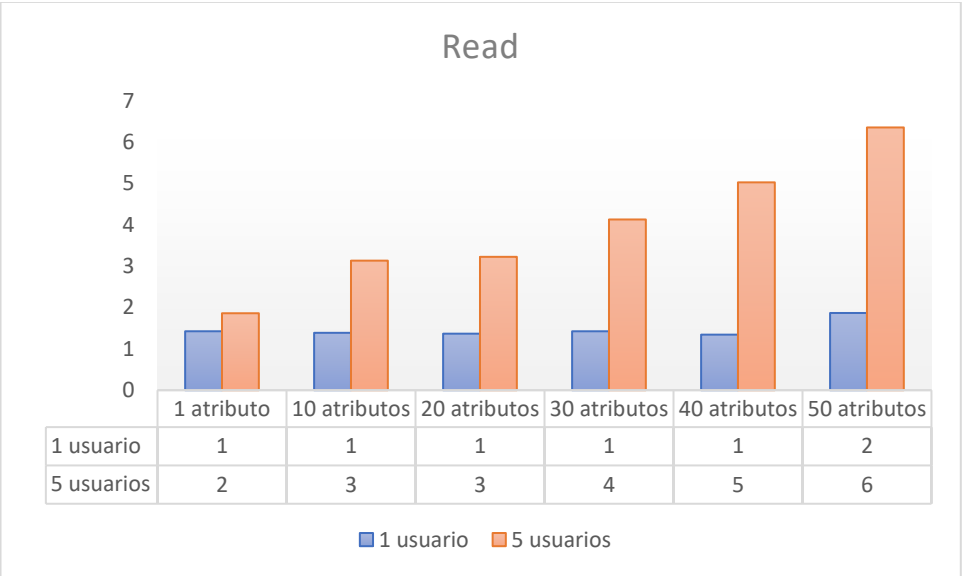
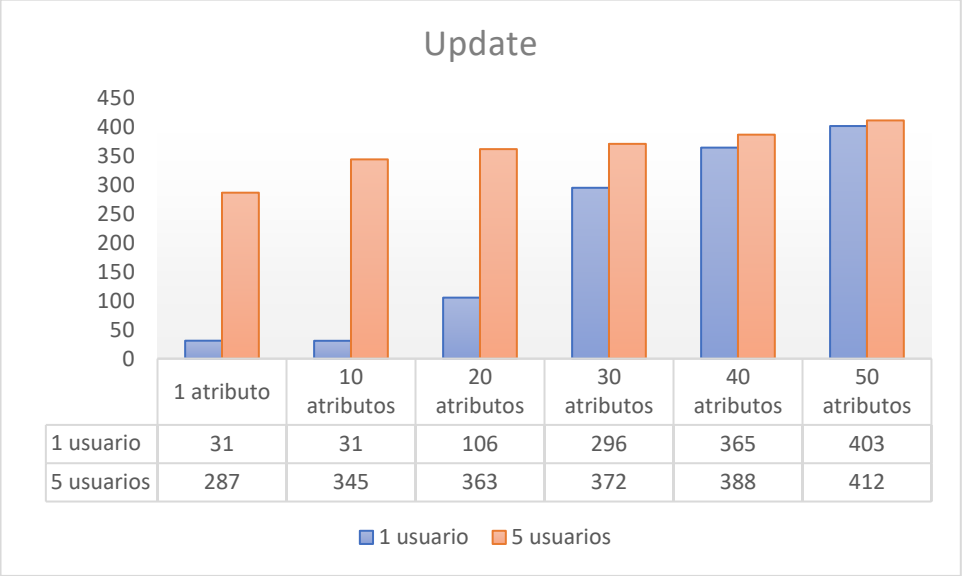


Ilustración 96. 1 vs 5 usuarios. Persistencia. Read

En el caso de la lectura, el acceso a datos por parte de varios usuarios concurrentes tiene un coste bastante mayor en general que un único usuario, con una tendencia

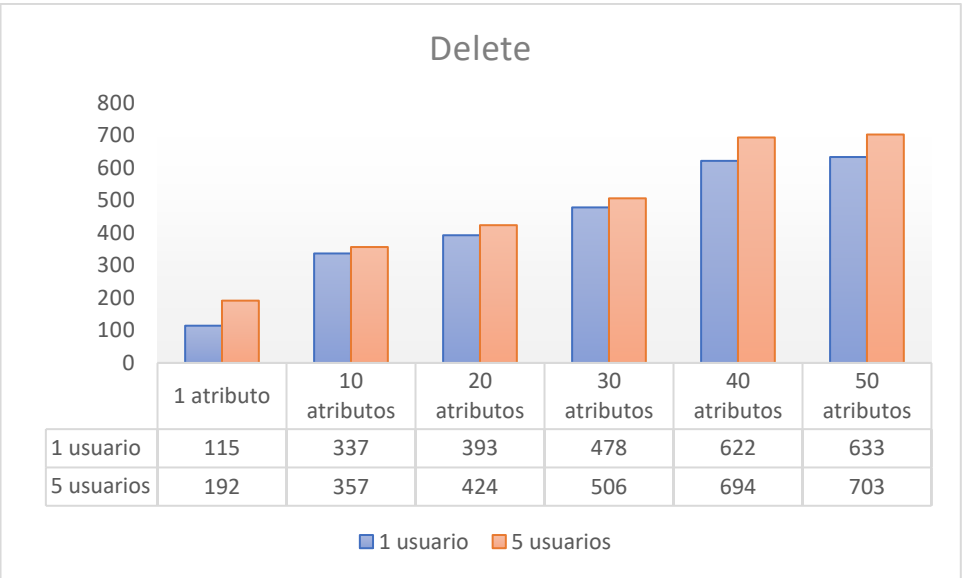
creciente conforme aumenta el número de atributos y una divergencia cada vez mayor. Con un único usuario los costes temporales apenas varían.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	31	31	106	296	365	403
5 usuarios	287	345	363	372	388	412

Ilustración 97. 1 vs 5 usuarios. Persistencia. Update

En este caso los costes temporales tienden a converger a partir de un número considerable de atributos. La divergencia por tanto es mucho mayor cuando el número de atributos está entre 1 y 20.

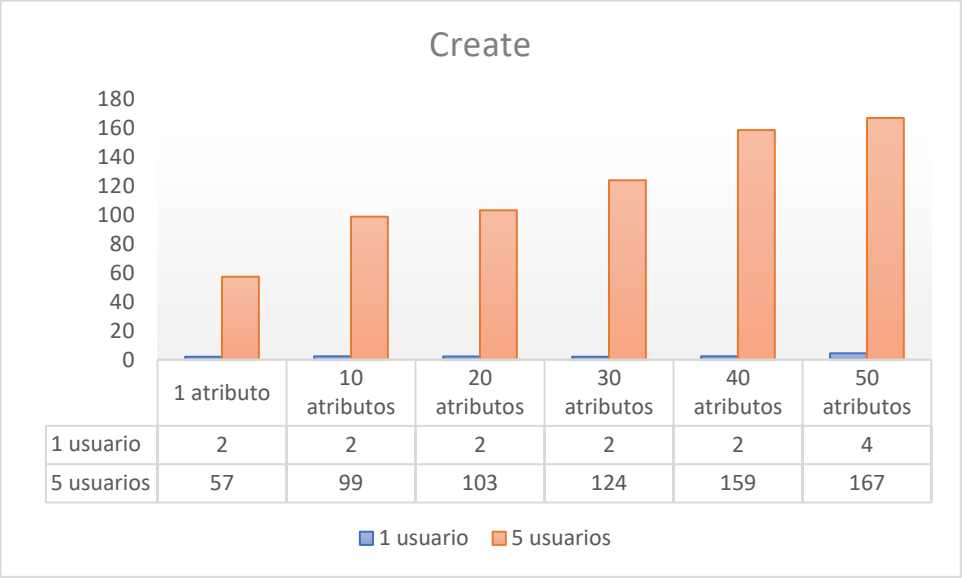


	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	115	337	393	478	622	633
5 usuarios	192	357	424	506	694	703

Ilustración 98. 1 vs 5 usuarios. Persistencia. Delete

En esta operación no hay demasiada diferencia en tiempos entre ambos casos. La concurrencia no afecta apenas a los costes temporales en esta operación.

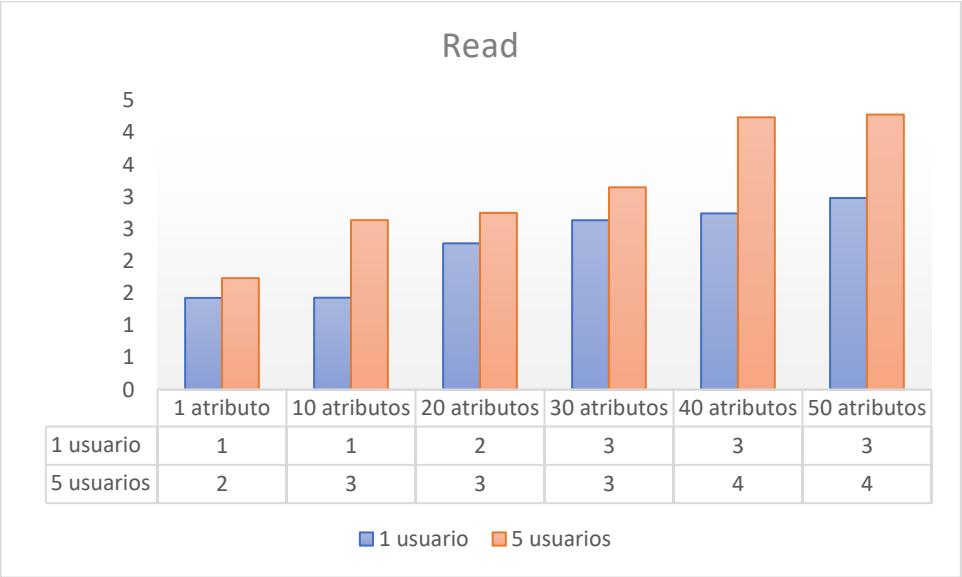
SESSION



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	2	2	2	2	2	4
5 usuarios	57	99	103	124	159	167

Ilustración 99. 1 vs 5 usuarios. Session. Create

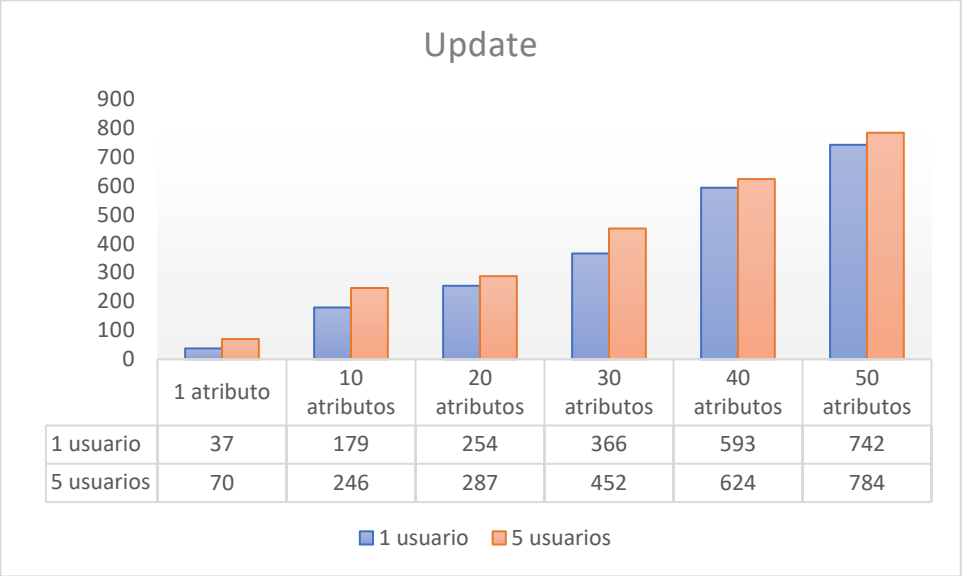
En este caso se puede apreciar que los costes son mucho mayores en el caso de varios usuarios concurrentes y apenas relevantes en el caso de un único usuario.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	1	1	2	3	3	3
5 usuarios	2	3	3	3	4	4

Ilustración 100. 1 vs 5 usuarios. Session. Read

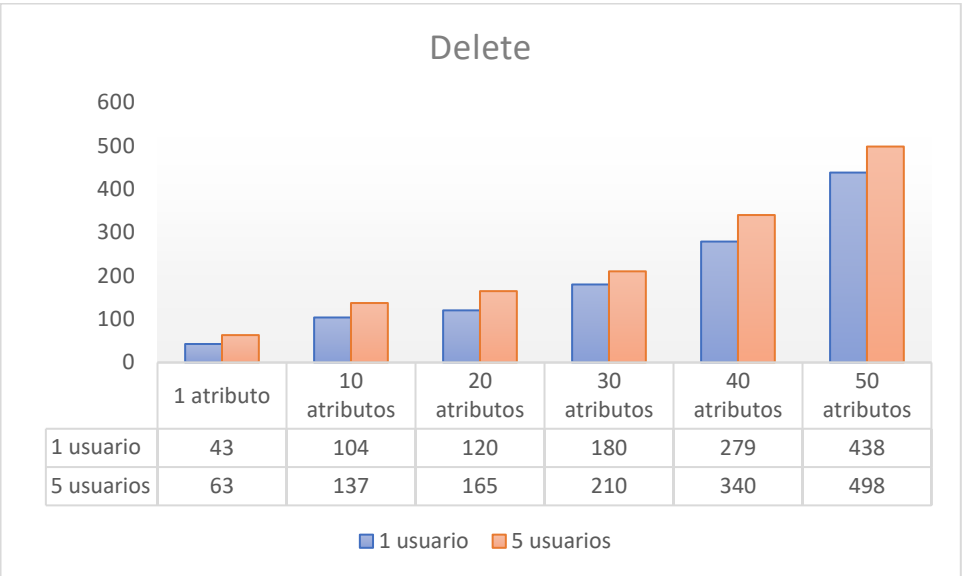
En promedio la lectura por parte de varios usuarios concurrentes es ligeramente más costosa, frente a la de un único usuario. La tendencia es alcista para los costes temporales en ambos casos, y la divergencia comienza a acentuarse para un número elevado de atributos (a partir de 40).



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	37	179	254	366	593	742
5 usuarios	70	246	287	452	624	784

Ilustración 101. 1 vs 5 usuarios. Session. Update

En esta operación los costes en ambos casos siguen una tendencia alcista, sin embargo, apenas existen diferencias entre un único usuario y algunos concurrentes.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	43	104	120	180	279	438
5 usuarios	63	137	165	210	340	498

Ilustración 102. 1 vs 5 usuarios. Session. Delete

En este caso los costes temporales de varios usuarios concurrentes para esta operación son ligeramente mayores que para los de un único usuario, pero las diferencias entre ambos no son sustanciales y siguen por igual una tendencia alcista conforme aumenta el número de atributos.

APPLICATION

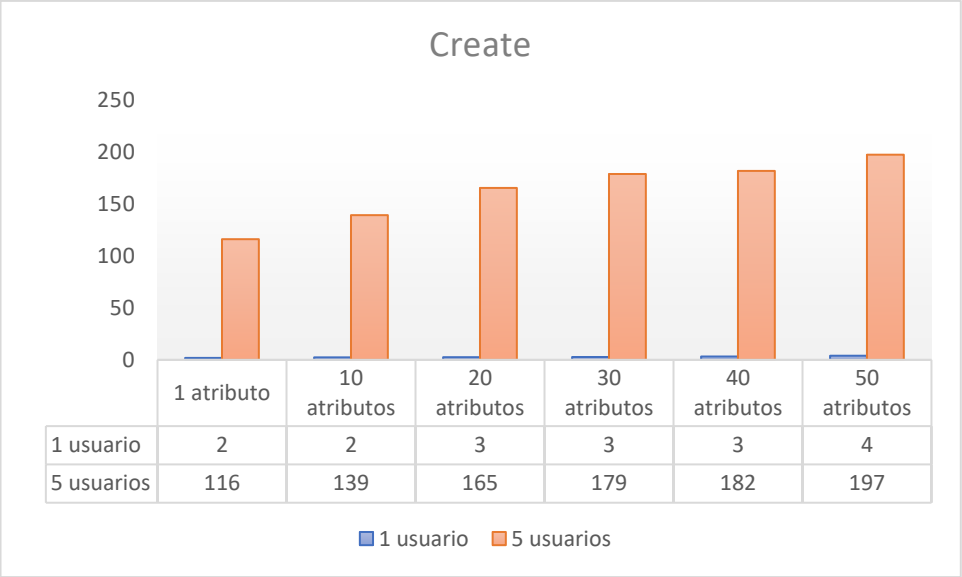
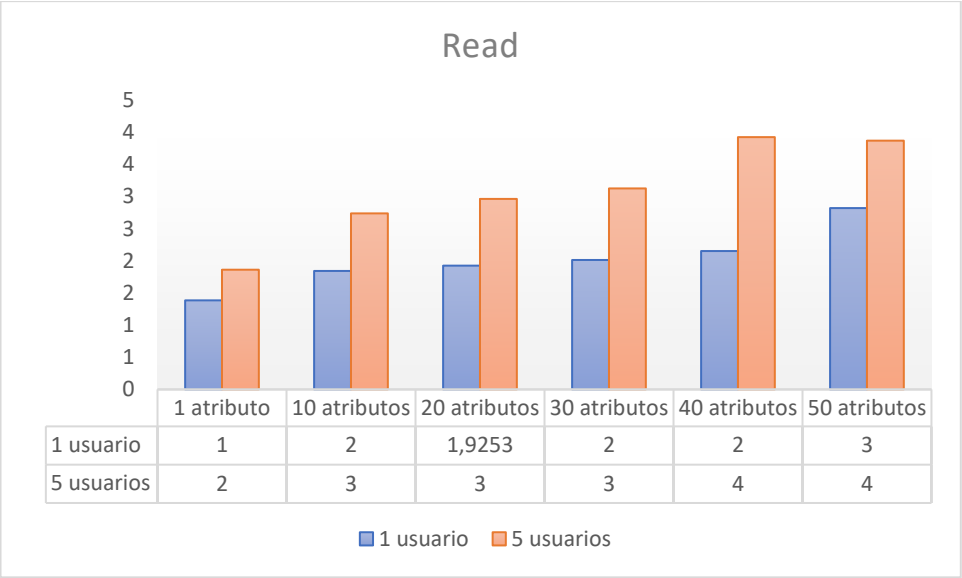


Ilustración 103. 1 vs 5 usuarios. Application. Create

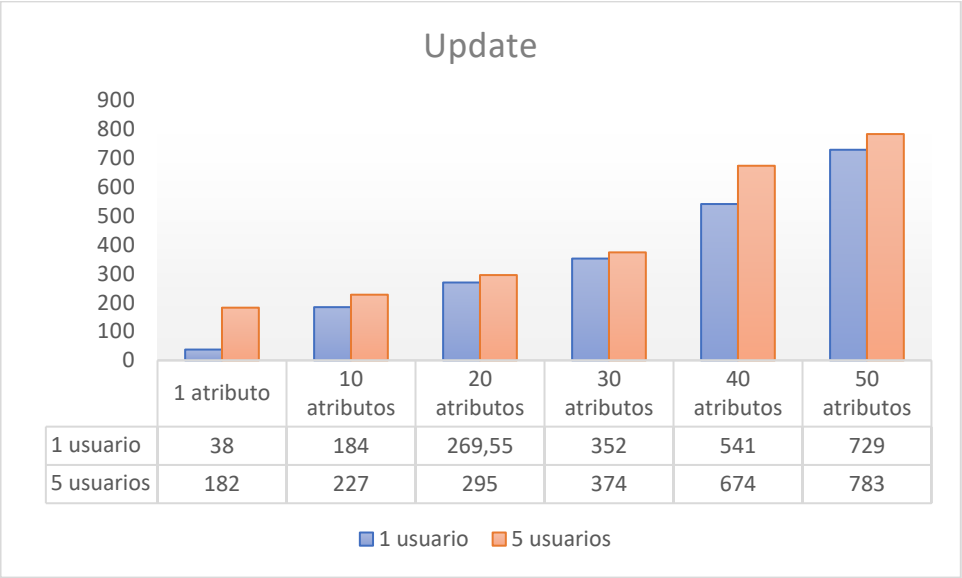
El coste de un único usuario es muy inferior y apenas varía con el número de atributos, en oposición al caso de tener 5 usuarios concurrentes que es mucho más elevado y tiene una tendencia ligeramente alcista.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	1	2	1,9253	2	2	3
5 usuarios	2	3	3	3	4	4

Ilustración 104. 1 vs 5 usuarios. Application. Read

Para el caso de un único usuario los tiempos son ligeramente inferiores frente al caso de 5 usuarios concurrentes. En ambos la tendencia es alcista pero no muy pronunciada.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
1 usuario	38	184	269,55	352	541	729
5 usuarios	182	227	295	374	674	783

Ilustración 105. 1 vs 5 usuarios. Application. Update

En este caso se muestra una tendencia alcista ambos casos. La divergencia en los costes temporales apenas es apreciable salvo en el caso de un único atributo.

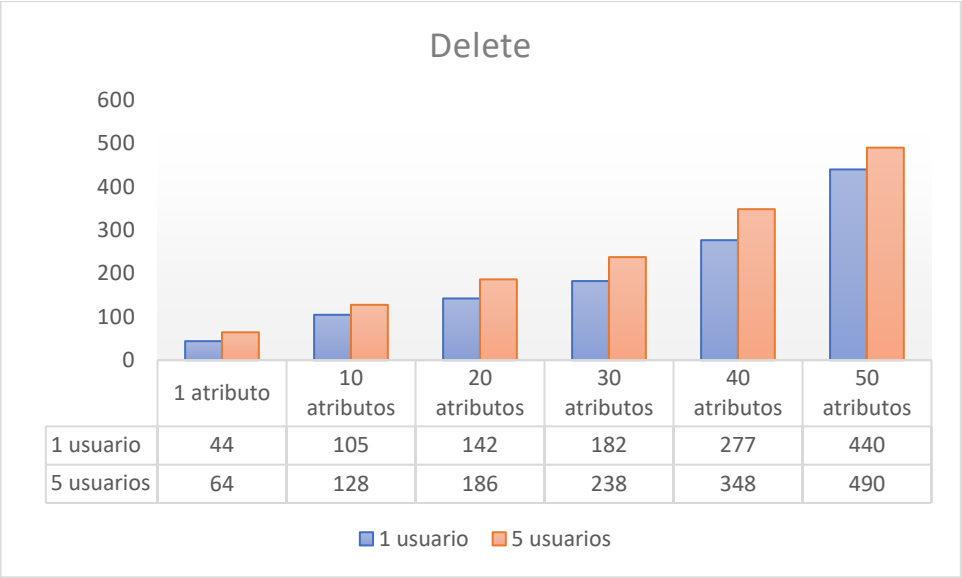


Ilustración 106. 1 vs 5 usuarios. Application. Delete

En este caso los tiempos apenas varían entre ambas opciones, la divergencia es muy pequeña, sin embargo, hay una tendencia alcista conforme aumenta el número de atributos.

CREATE

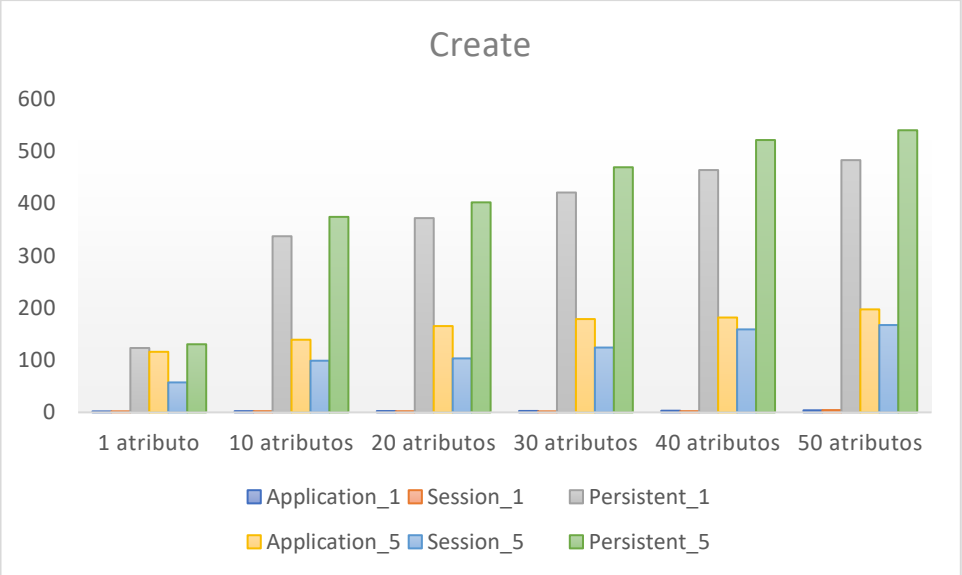


Ilustración 107. 1 vs 5 usuarios. Create. Barras

En la configuración de persistencia en base de datos apenas es apreciable la diferencia entre que haya 1 o 5 usuarios, se trata de la configuración más costosa frente a la persistencia volátil, en la que sí es notable la diferencia de un único usuario frente a 5 usuarios concurrentes.

READ

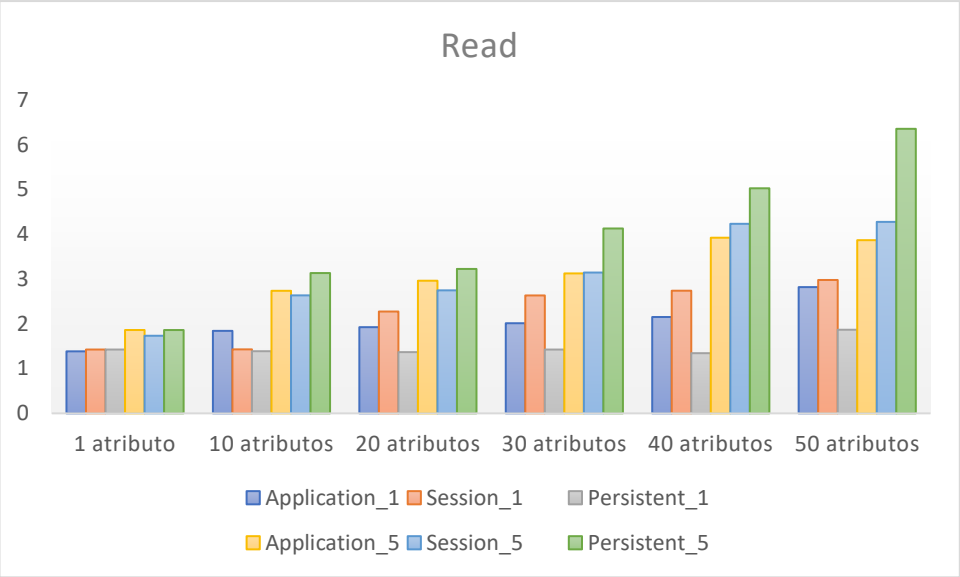


Ilustración 108. 1 vs 5 usuarios. Read. Barras

De esta gráfica se puede extraer la conclusión de que conforme aumenta el número de atributos las diferencias de costes temporales entre un único usuario y 5 usuarios concurrentes cada vez son más pronunciadas, a favor de un solo usuario.

UPDATE

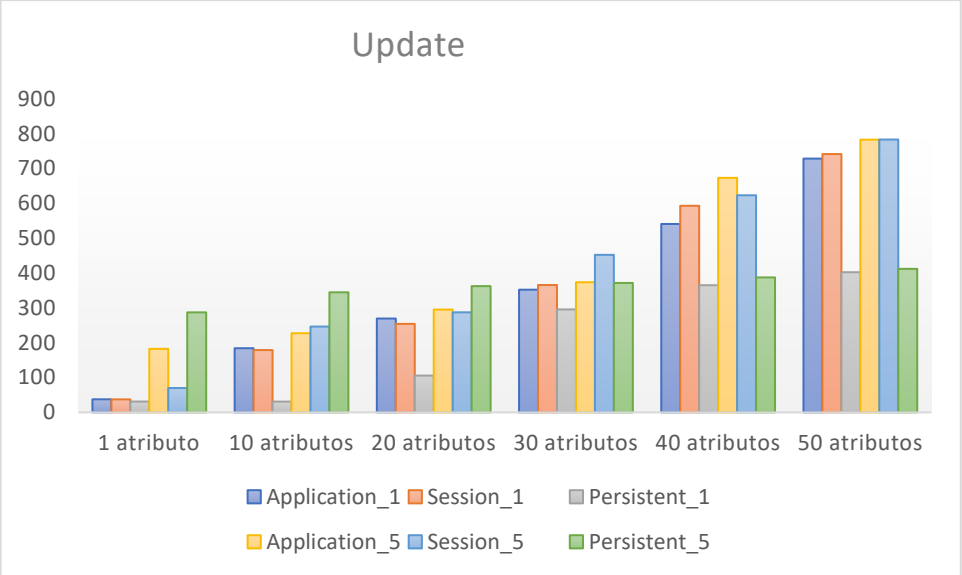


Ilustración 109. 1 vs 5 usuarios. Update. Barras

En este caso los tiempos siguen una tendencia alcista. El mejor comportamiento se da en la persistencia en bases de datos, las configuraciones de persistencia volátil tienen un peor rendimiento y por tanto no serían recomendables para esta operación.

DELETE

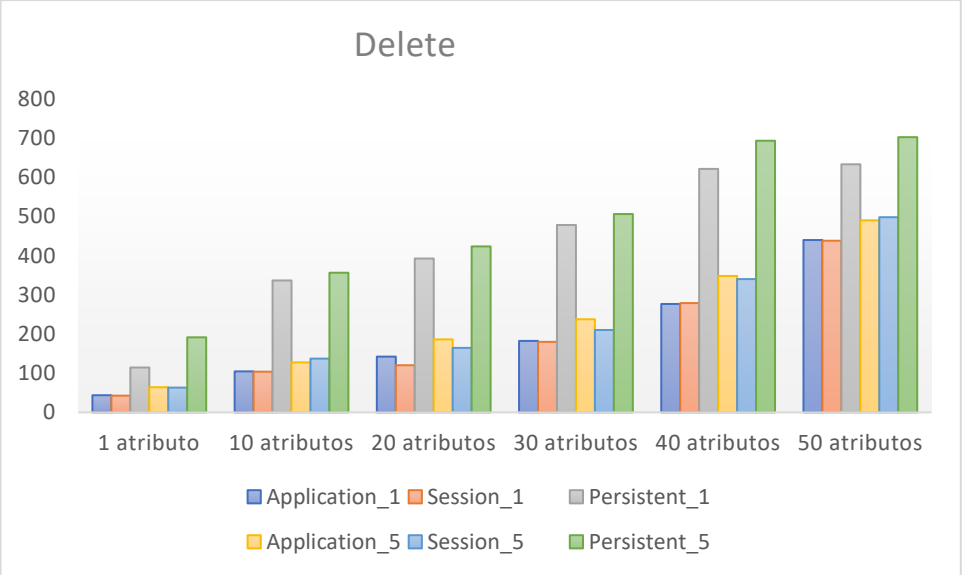


Ilustración 110. 1 vs 5 usuarios. Delete. Barras

Los tiempos de borrado en este caso son ligeramente superiores en la configuración de persistencia en bases de datos, por ello cualquier configuración de persistencia volátil es más recomendable en esta operación concreta. No hay apenas diferencia entre las configuraciones volátiles, cualquiera de ellas que se ajustase a las reglas de negocio y al dominio de datos de la aplicación sería válida.

7.3. 5 vs 10 usuarios

A continuación, se presentan los resultados comparativos entre 5 usuarios y 10 usuarios concurrentes para la misma máquina de la capa gratuita de Amazon.

7.3.1. Especificaciones

	Amazon t1.micro
RAM	1GB
Sistema Operativo	Linux
Procesador	Intel Xeon
CPUs	2,5 GHz

Tabla 3. Especificaciones máquinas 5 vs 10 usuarios

7.3.2. Operaciones

PERSISTENT

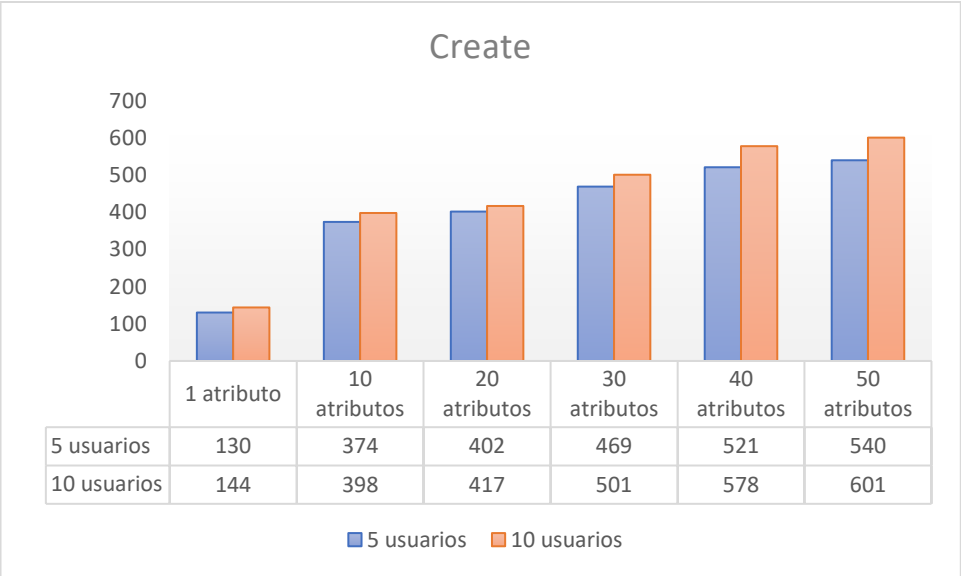


Ilustración 111. 5 vs 10 usuarios. Persistent. Create

En la gráfica se puede apreciar que para esta operación los tiempos aumentan conforme el número de atributos es cada vez mayor, sin embargo, no se aprecian diferencias notables entre 5 y 10 usuarios concurrentes.

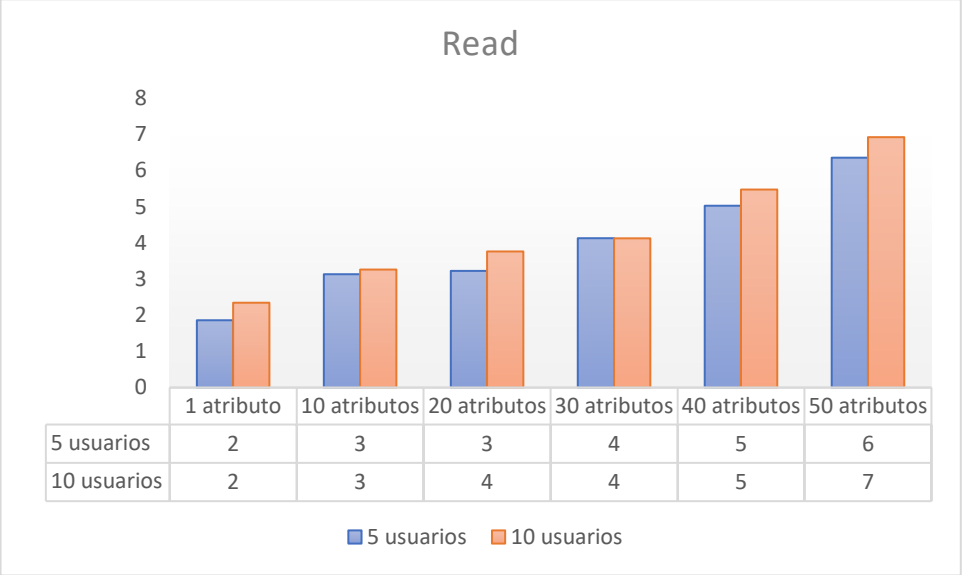


Ilustración 112. 5 vs 10 usuarios. Persistent. Read

En el caso de esta operación tampoco hay apenas diferencia entre 5 y 10 usuarios concurrentes. La tendencia es ligeramente alcista.

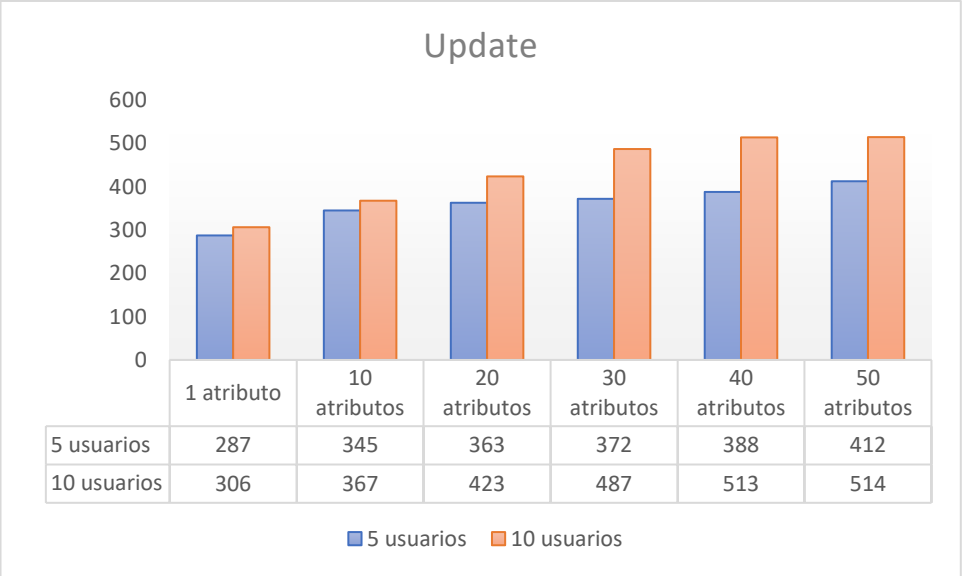


Ilustración 113. 5 vs 10 usuarios. Persistent. Update

Para esta operación existe una tendencia alcista en ambos casos, no obstante, la divergencia de costes temporales se acentúa conforme aumenta el número de atributos.

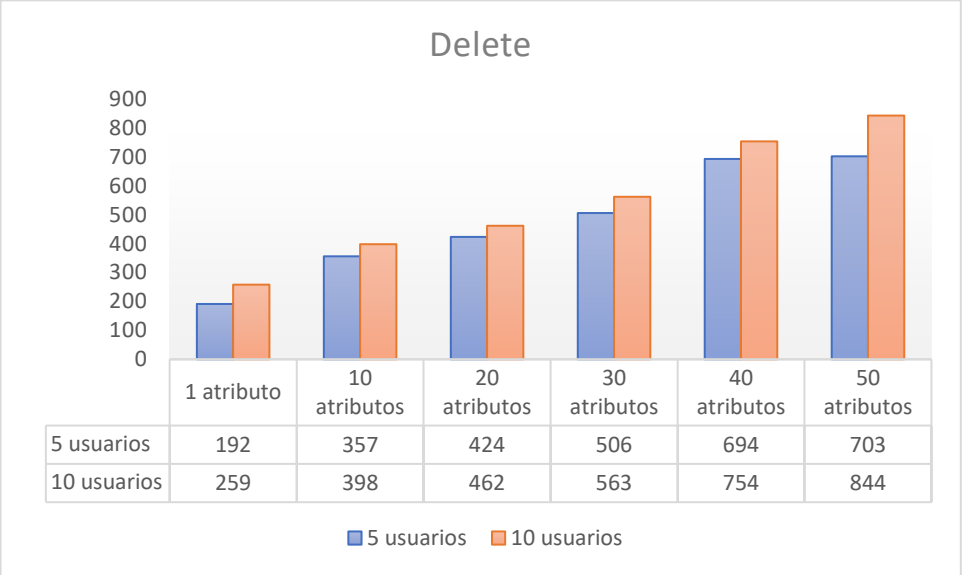


Ilustración 114. 5 vs 10 usuarios. Persistent. Delete

En esta operación los tiempos son muy similares y siguen una tendencia alcista en ambos casos.

SESSION

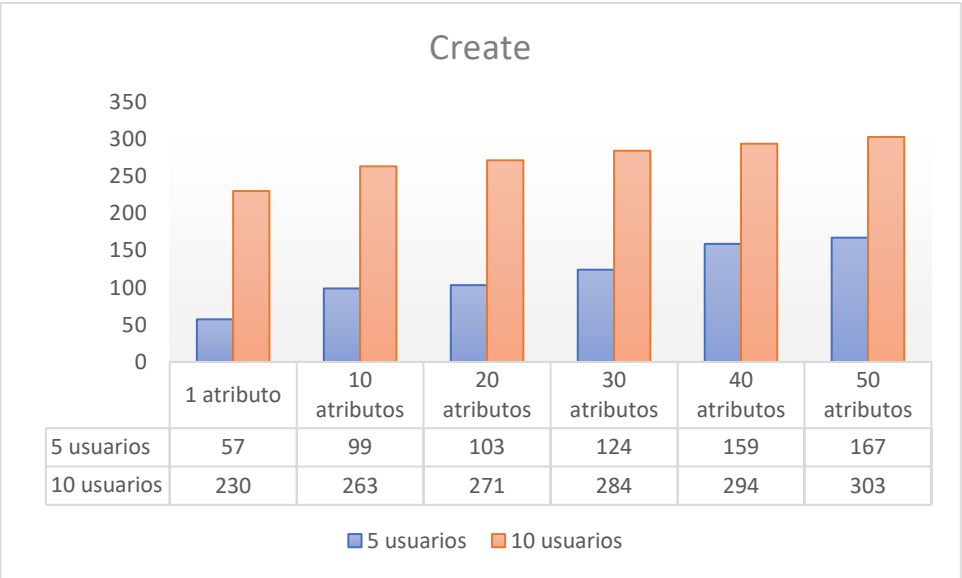


Ilustración 115. 5 vs 10 usuarios. Session. Create

Para esta configuración las diferencias sí son bastante considerables entre 5 y 10 usuarios concurrentes, y dicha divergencia no tiende a disminuir conforme aumenta el número de usuarios.

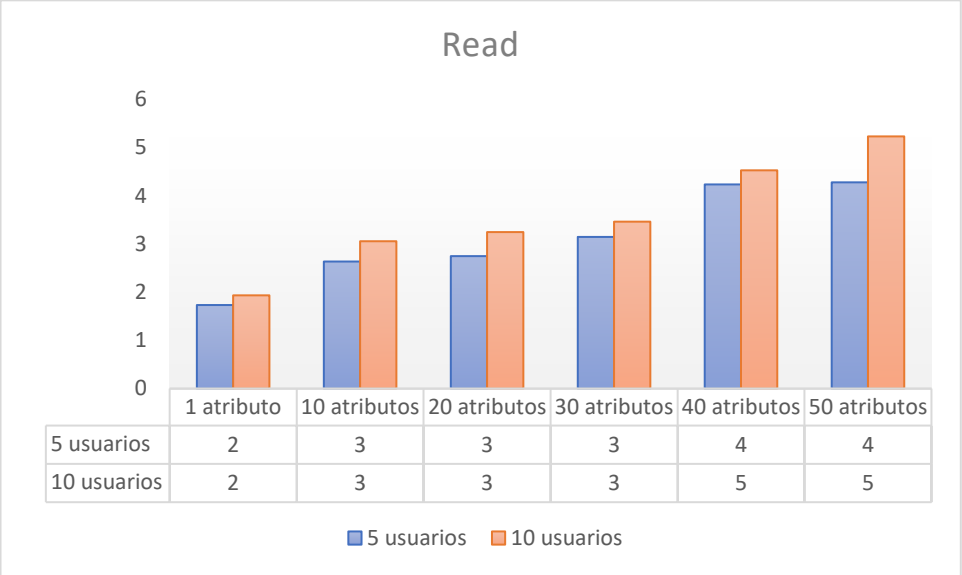


Ilustración 116. 5 vs 10 usuarios. Session. Read

Tal y como muestra la gráfica, apenas existe variación entre los tiempos de ambos casos, la tendencia es ligeramente alcista para esta operación.

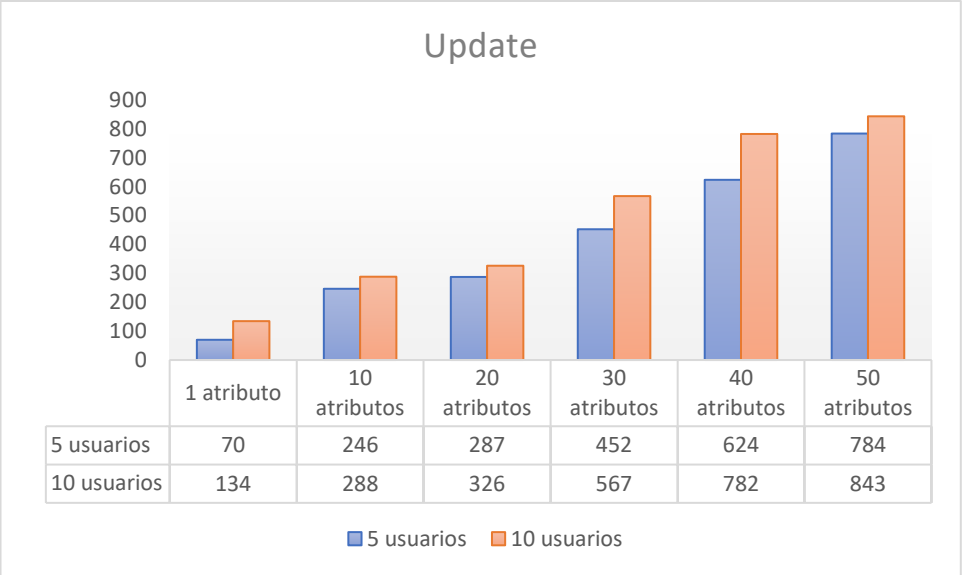


Ilustración 117. 5 vs 10 usuarios. Session. Update

En esta operación los costes temporales son muy similares y hay una tendencia alcista clara cuando el número de atributos aumenta.

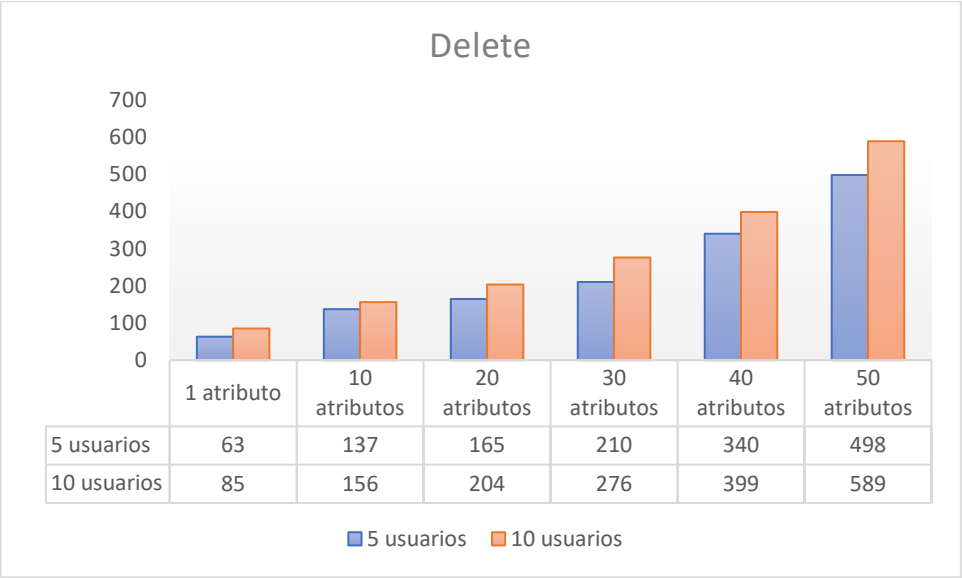


Ilustración 118. 5 vs 10 usuarios. Session. Delete

Los valores temporales en este caso también son muy similares para 5 y 10 usuarios concurrentes, y la tendencia es alcista también.

APPLICATION

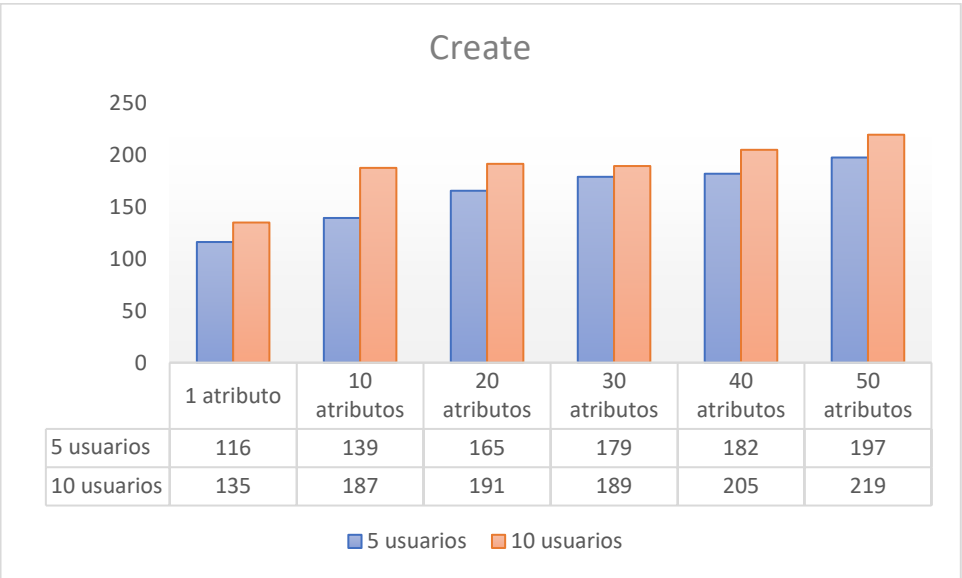
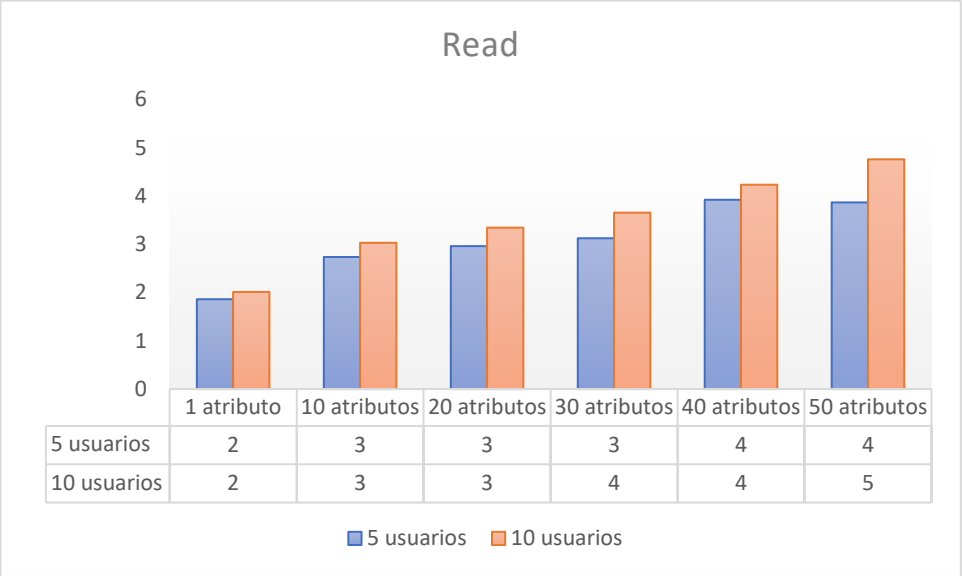


Ilustración 119. 5 vs 10 usuarios. Application. Create

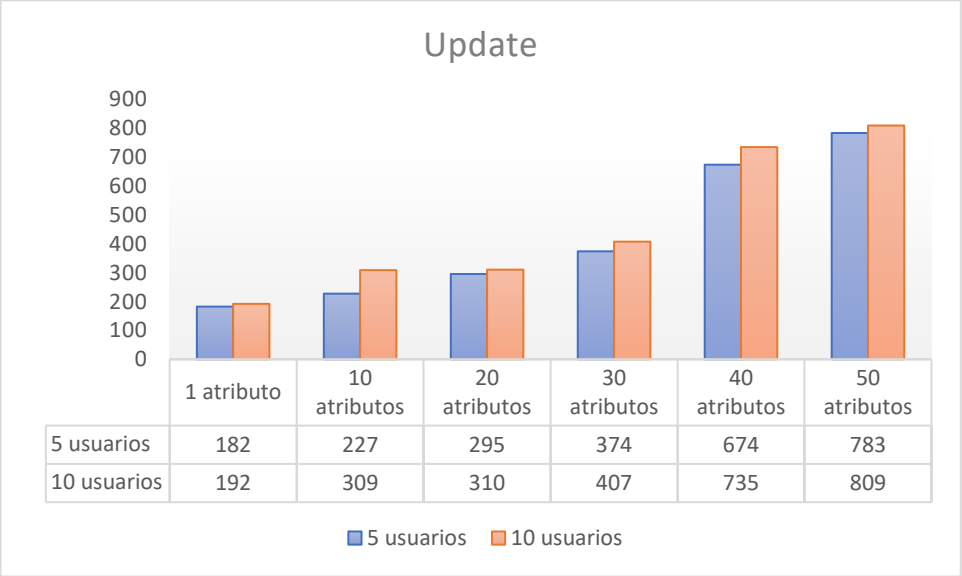
Los tiempos no aumentan mucho conforme se produce un incremento de los atributos, del mismo modo la diferencia entre los 5 y 10 usuarios no es sustancial.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
5 usuarios	2	3	3	3	4	4
10 usuarios	2	3	3	4	4	5

Ilustración 120. 5 vs 10 usuarios. Application. Read

Para esta operación los tiempos son muy similares y la tendencia es ligeramente alcista.



	1 atributo	10 atributos	20 atributos	30 atributos	40 atributos	50 atributos
5 usuarios	182	227	295	374	674	783
10 usuarios	192	309	310	407	735	809

Ilustración 121. 5 vs 10 usuarios. Application. Update

No existe una gran diferencia entre los tiempos de ambos casos, pero hay un salto bastante significativo a partir de los 30 atributos.

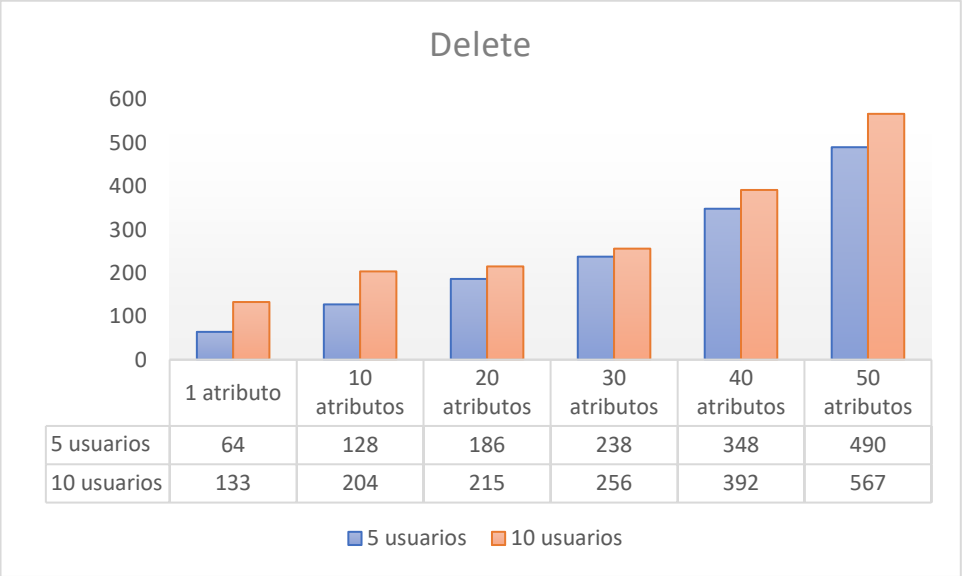


Ilustración 122. 5 vs 10 usuarios. Application. Delete

Tampoco existe una diferencia notable entre los tiempos de ambos casos, la tendencia que siguen es alcista.

CREATE

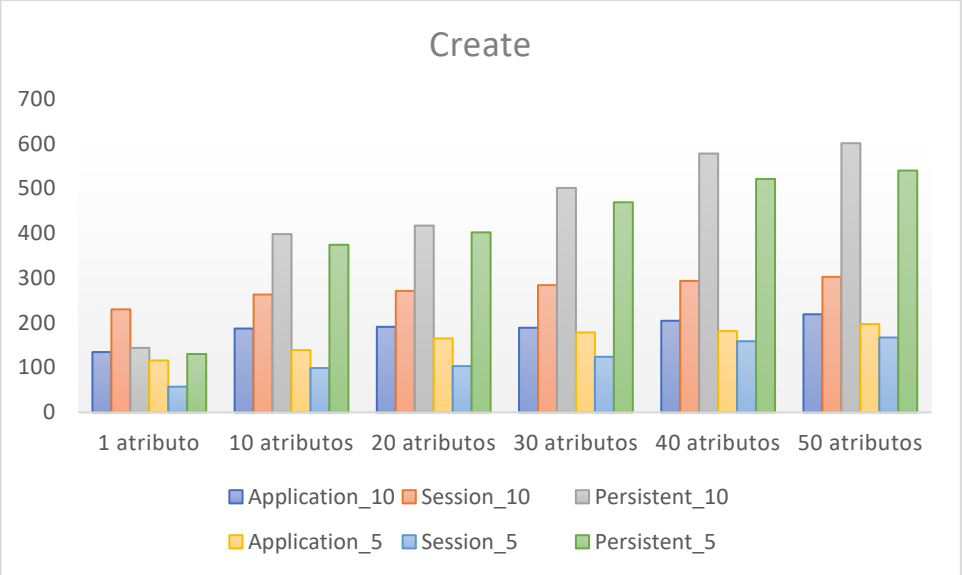


Ilustración 123. 5 vs 10 usuarios. Create. Barras

Los valores más bajos se corresponden generalmente con la persistencia volátil. Para 1 atributo apenas existe diferencia, pero conforme la cantidad de atributos aumenta cada vez es mayor la diferencia de tiempos. Por ello es más eficiente en este caso una configuración de persistencia volátil.

READ

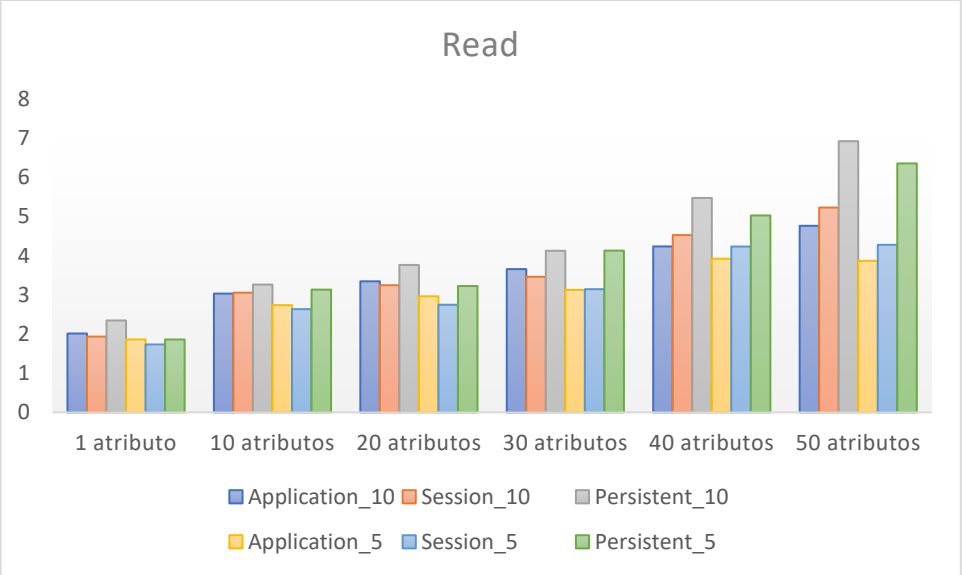


Ilustración 124. 5 vs 10 usuarios. Read. Barras

Independientemente del tipo de persistencia o del número de usuarios concurrentes, casi todos los valores tienden a converger y siguen una tendencia más o menos estable. No hay por tanto una gran diferencia de tiempos entre las distintas configuraciones y no podría decantarse por una en concreto.

UPDATE

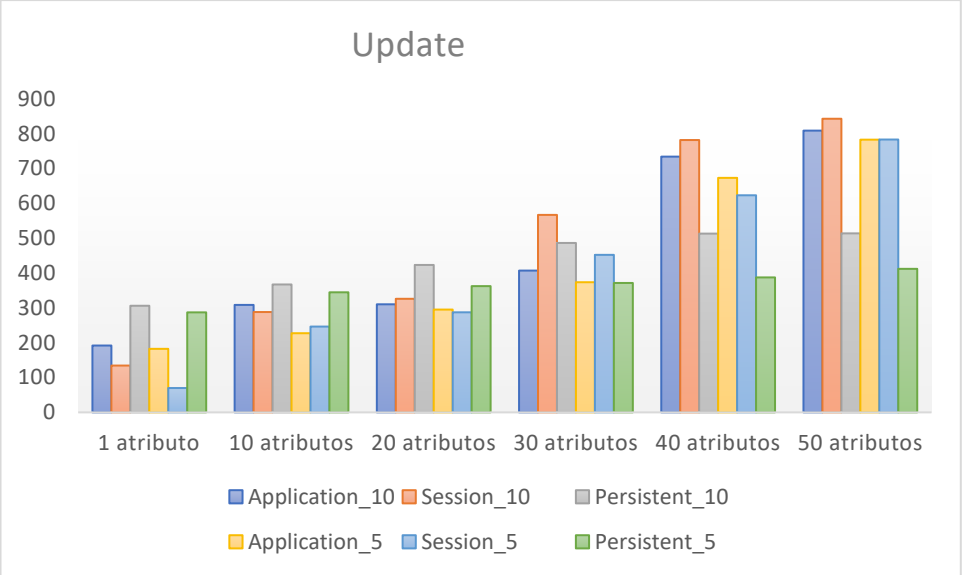


Ilustración 125. 5 vs 10 usuarios. Update. Barras

Para esta operación a partir de 40 atributos empieza a producirse una considerable divergencia de tiempos entre las opciones de persistencia volátiles y la persistencia de bases de datos. Esta última tiene un mejor comportamiento y no sigue una tendencia alcista tan pronunciada como los otros tipos de persistencia, lo que la hace más recomendable para este caso.

DELETE

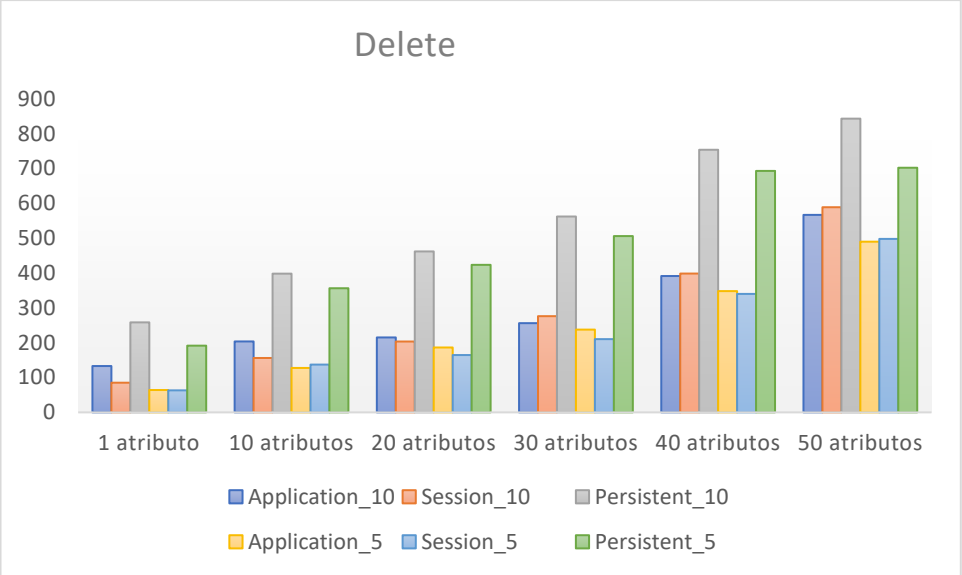


Ilustración 126. 5 vs 10 usuarios. Update. Barras

Los valores temporales para la persistencia en base de datos son los más altos de entre las 3 configuraciones. Así los valores más bajos se corresponden con la persistencia volátil, no habiendo apenas diferencias entre las dos configuraciones que se han estudiado. La tendencia es alcista conforme aumenta el número de atributos.

8. Conclusiones

La obtención y posterior análisis de las mediciones del apartado anterior han permitido descubrir cuáles son las configuraciones más adecuadas para un tipo de operación y un tipo de persistencia con un cierto número de atributos.

En ocasiones las diferencias no han sido significativas, pero al menos dan un argumento más para decidir si es relevante optar por una vía o por otra a la hora de diseñar la aplicación. También es posible anticipar qué número de atributos puede ser aceptable tener por una entidad, si conviene tenerlo todo en una o por el contrario sería mejor crear varias entidades.

Por otro lado, se han ha obtenido unas estimaciones de los costes de cada operación, que podrán dar una idea de cómo se espera que responda la aplicación en cada caso, tratando de enfocar las principales funcionalidades con las operaciones que sean menos costosas.

Las conclusiones extraídas del estudio pueden ser además reutilizadas para el desarrollo de otros proyectos que vayan a desplegarse en máquinas con esas especificaciones, luego podría afirmarse que dedicar recursos a este estudio puede ser una buena inversión a medio y largo plazo. Permitirá desarrollar aplicaciones de una forma más eficiente y aprovechando mejor la tecnología que se está empleando para su realización.

8.1. Ideas futuras de este trabajo

Los resultados obtenidos pueden tener varias proyecciones en futuros trabajos. Algunas de las posibilidades son:

- Modificar la Time Unit para que en lugar de milisegundos dé valores en nanosegundos, y permita resultados más precisos en operaciones que tardan muy pocos milisegundos.
- Monitorizar la memoria RAM, la capacidad de almacenamiento y el procesador en distintos puntos de ejecución de la cadena.
- Modificar la cadena de operaciones con distintas combinaciones de operaciones en distinto orden.
- Evaluar el impacto del tipo de atributo almacenado. En este trabajo se ha probado con una cadena de 255 caracteres, pero sería interesante evaluar otros tipos como int, float, date…
- Desarrollar un plugin para WebRatio que en tiempo de diseño fuese capaz de anticipar una estimación de los costes computacionales asociados a una infraestructura concreta.
- Estudiar la influencia de otras variables como la escalabilidad, el tipo de tecnología de base de datos, etc.

REFERENCIAS BIBLIOGRÁFICAS

No hay muchas referencias de libros o páginas en Internet porque la mayor parte de los conocimientos que he empleado en la realización de este trabajo los he adquirido en las tutorías con el director del trabajo, en la asignatura de Ingeniería Web y las Prácticas Externas. Las páginas que he consultado han sido las siguientes (y sus apartados):

- *Eduardo F. Morales*. Instituto Nacional de Astrofísica, Óptica y Electrónica de México. [Última consulta: 15 mayo 2017]

  https://ccc.inaoep.mx/~emorales/Cursos/NvoAprend/node78.html

- *IFML*. Object Management Group. [Última consulta: 17 mayo 2017]

  http://www.ifml.org/

- *WebRatio*. [Última consulta: 10 mayo 2017]

  http://www.webratio.com/site/content/es/web-application-development

- *Amazon Docs*. [Última consulta: 20 mayo 2017]

  https://aws.amazon.com/es/documentation/elastic-beanstalk/

Por otra parte, también se han tomado como referencia dos artículos publicados sobre este trabajo:

- *PRECIADO RODRÍGUEZ*, Juan Carlos; *RODRIGUEZ-ECHEVERRIA* Roberto; *CONEJERO MANZANO*, José María; *SÁNCHEZ-FIGUEROA*, Fernando and *MARTÍN SÁNCHEZ*, Rubén, (2017). **A First Step to Cloud Infrastructure Cost Estimation in Early Stages of Web Development**.In - *APMDWE,* pages 1-8. DOI: 10.5220/0006393904370443

- *PRECIADO RODRÍGUEZ*, Juan Carlos; *RODRIGUEZ-ECHEVERRIA* Roberto; *CONEJERO MANZANO*, José María; *SÁNCHEZ-FIGUEROA*, Fernando and *MARTÍN SÁNCHEZ*, Rubén, (2017). **Hacia una propuesta de estimación de costes de producción desde etapas tempranas del desarrollo Web**. In – JISBD, pages 1-4.

ANEXOS

I. Generar el WAR en WebRatio

Una de las primeras tareas a realizar para poder desplegar proyectos a la nube es obtener un archivo comprimido WAR que incluya el código fuente y sea válido para la plataforma en la que se quiere llevar a cabo el despliegue. En este trabajo la plataforma es Amazon Web Services, sin embargo, existen muchas otras que permiten cargar una aplicación web a partir de un archivo WAR.

Los pasos a seguir para obtener ese archivo a partir de la plataforma WebRatio son los siguientes:

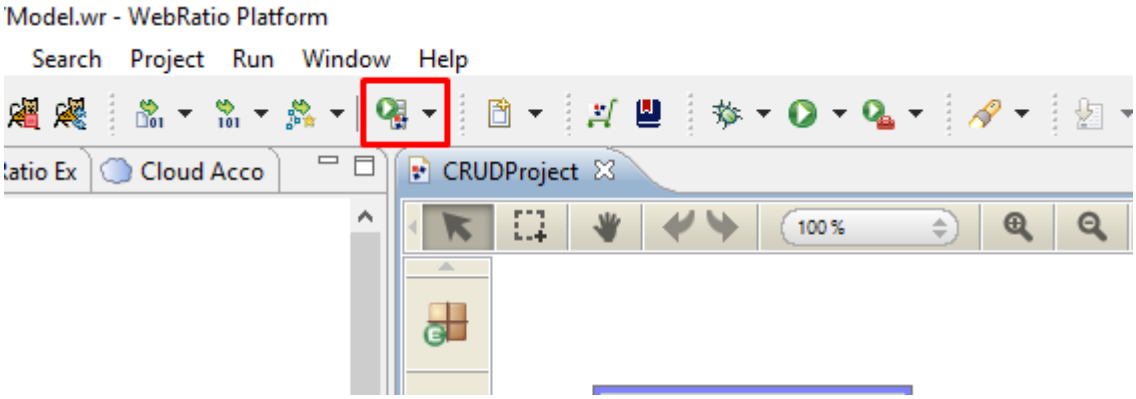


Ilustración 127. Generar el WAR en WebRatio. Opciones de despliegue

Hay que buscar un icono con una miniatura de lo que parece un servidor. Una vez se ha hecho clic en ese icono, luego se escoge la opción *Deploy Configurations…*:

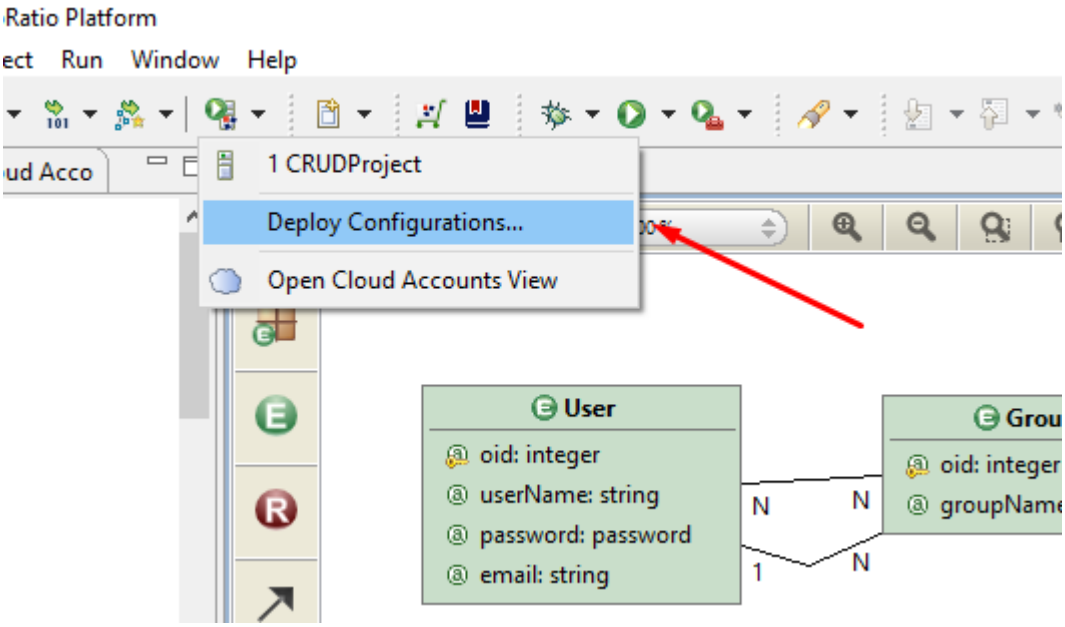


Ilustración 128. Generar el WAR en WebRatio. Deploy configurations

Se abrirá una nueva ventana en la que habrá que crear una nueva configuración de despliegue con el botón derecho sobre *Local(Web)*:

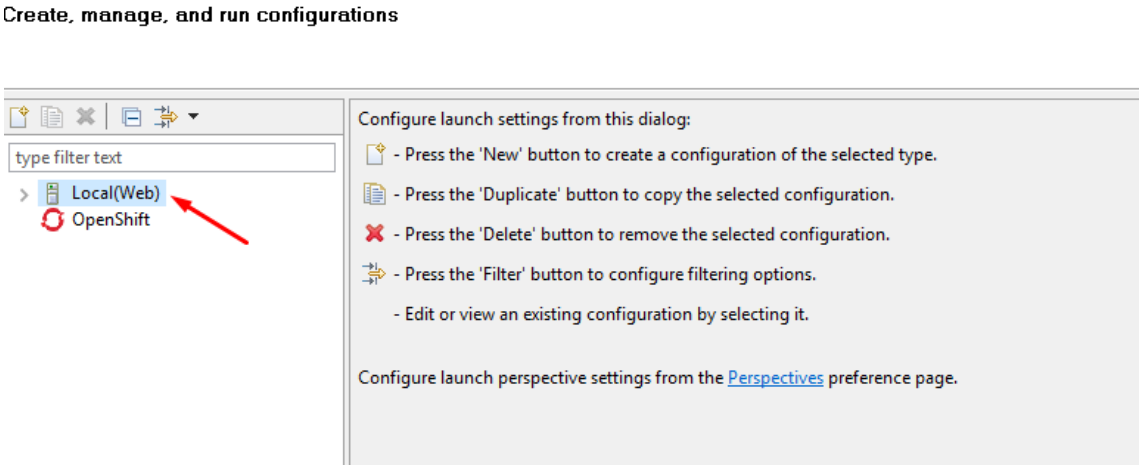


Ilustración 129. Generar el WAR en WebRatio. Configuración local de despliegue

Justo después hay que ir a la pestaña *Tasks*:

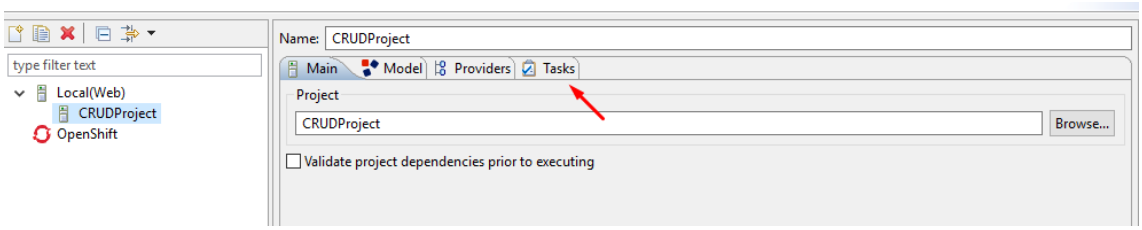


Ilustración 130. Generar el WAR en WebRatio. Configuración creada de despliegue

Y en esa pestaña añadir una nueva tarea en el botón *Add*:

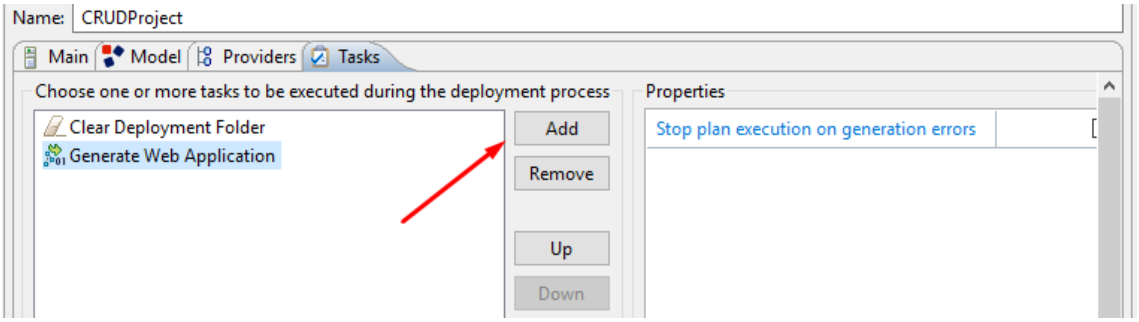


Ilustración 131. Generar el WAR en WebRatio. Tareas por defecto en el despliegue

Se abrirá la siguiente ventana:

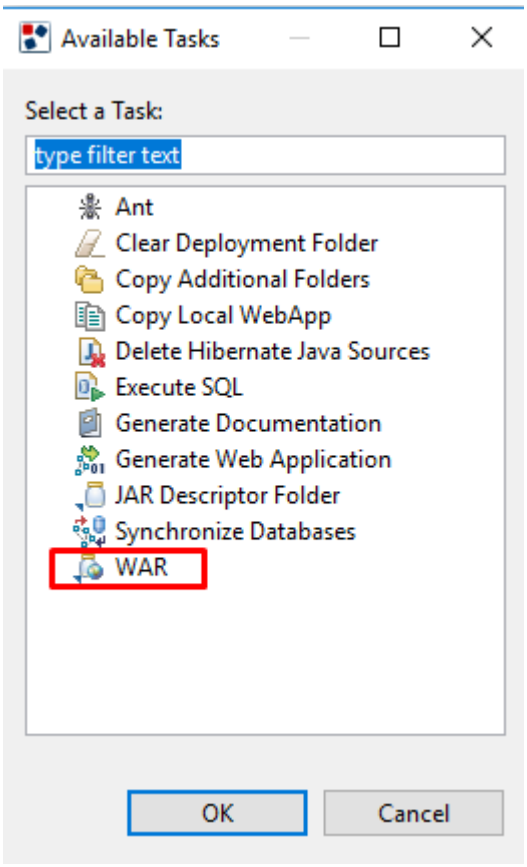
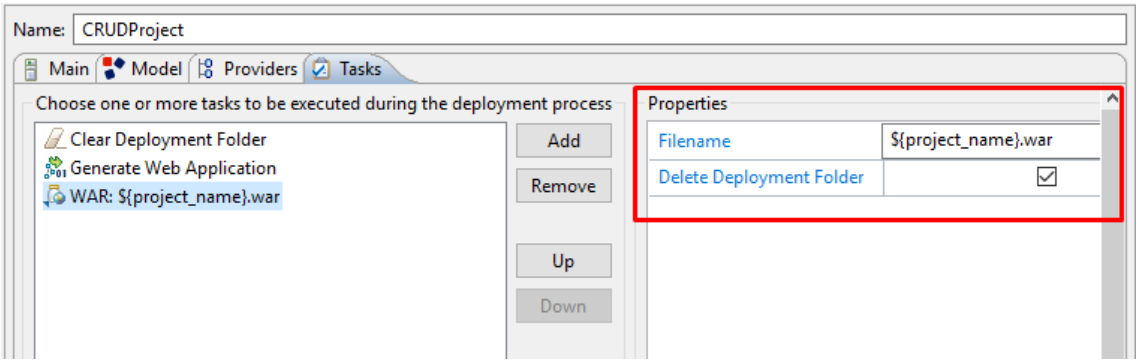


Ilustración 132. Generar el WAR en WebRatio. Tipos de tareas de despliegue

De entre todas las opciones hay que escoger la última, correspondiente al WAR, y así después de darle al botón OK aparecerá en la lista de tareas que se ejecutarán durante el proceso de despliegue:



En esa ventana misma ventana será necesario darle al botón *Apply* y luego a *Deploy* para que se apliquen los cambios y comience el proceso de generación del WAR. Con eso se generará el fichero de despliegue del proyecto en la carpeta Tomcat de WebRatio.

El resultado final será:

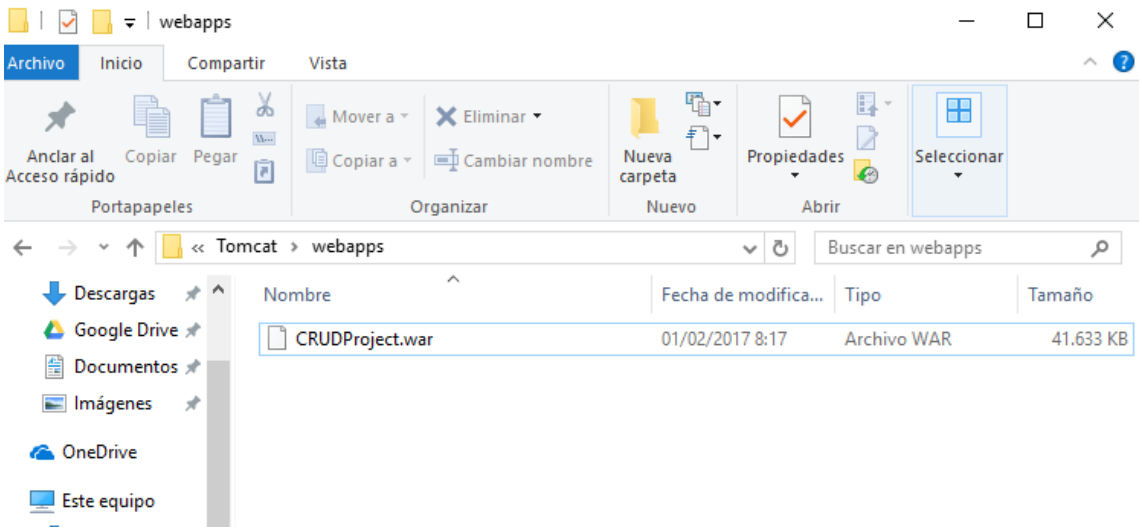


Ilustración 133. Generar el WAR en WebRatio. Fichero WAR creado en el directorio

II. Conectar una base de datos al proyecto

En este apartado se va a explicar cómo conectar una base de datos RDS de Amazon con una aplicación en WebRatio, y cómo hacer dicha base de datos accesible desde el exterior.

Partiendo de que ya hay creada una instancia de RDS, habrá que ir al *Dashboard* de RDS, más concretamente a la opción de *Security Groups*:

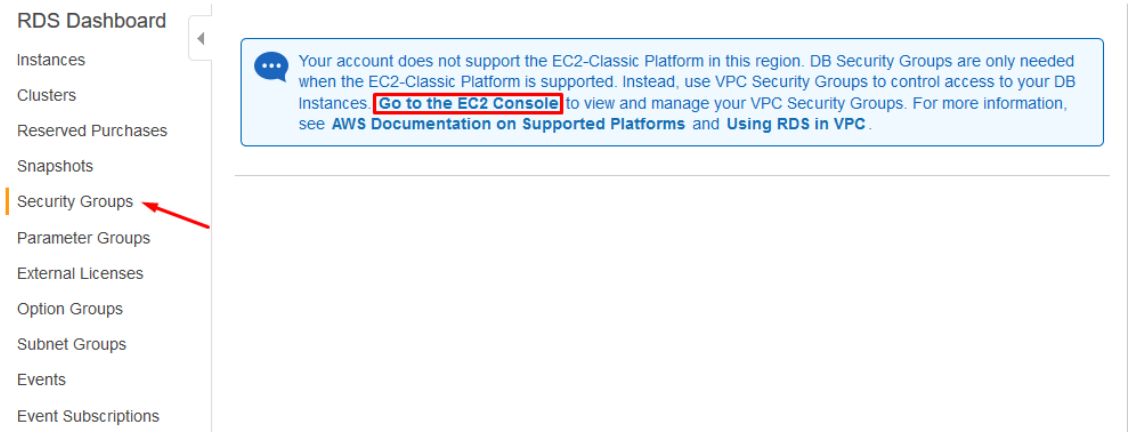


Ilustración 134. Conectar una base de datos al proyecto. Acceder a la Consola EC2 desde el RDS Dashboard

Luego, para que la base de datos sea accesible desde la aplicación hay que crear un grupo de seguridad específico y asignarlo a esa base de datos RDS.

Luego se hace clic en el siguiente botón:

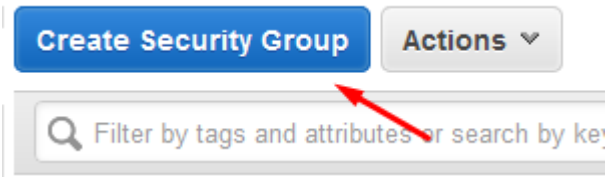


Ilustración 135. Conectar una base de datos al proyecto. Creación del grupo de seguridad

Y se abrirá una ventana emergente:

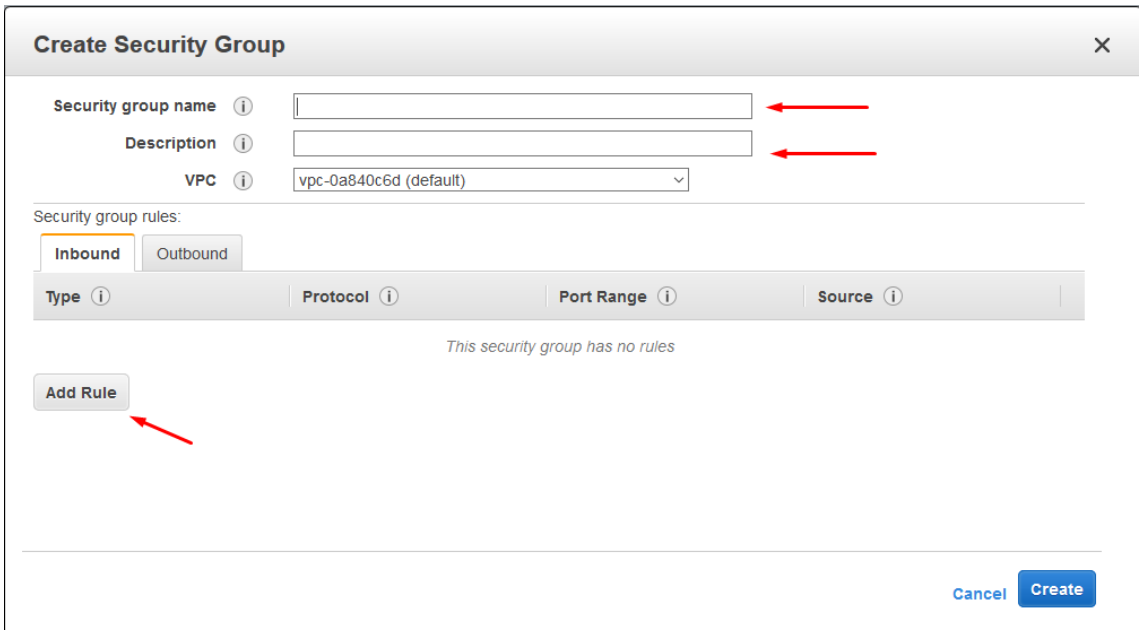


Ilustración 136. Conectar una base de datos al proyecto. Regla en el grupo de seguridad

Se mostrará un mensaje diciendo que ese grupo de seguridad no dispone de ninguna regla. Para crear una hay que pulsar el botón *Add Rule*:

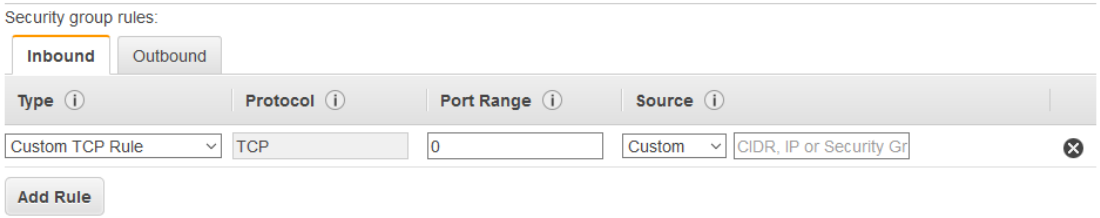


Ilustración 137. Conectar una base de datos al proyecto. Creación de la regla de seguridad

En el primer campo aparecerá un desplegable como el siguiente, con una lista bastante amplia de protocolos de acceso:

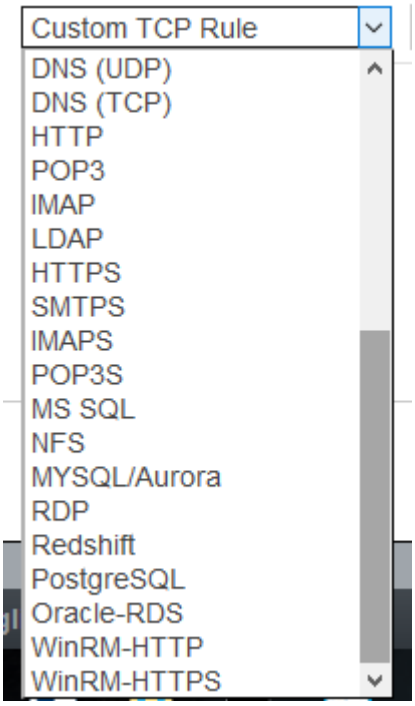


Ilustración 138. Conectar una base de datos al proyecto. Listado de protocolos de acceso

En este caso se escogerá PostgresSQL por ser la tecnología escogida para la base de datos RDS creada, aunque podría haberse escogido otra. Una vez seleccionada la base de datos, se autocompletará el campo relativo al puerto predeterminado de esa tecnología, y que suele variar según la base de datos:

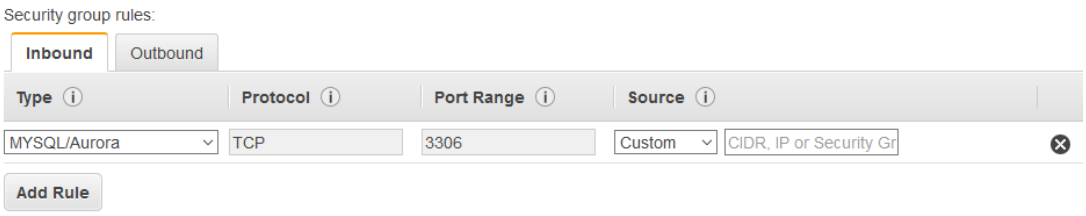


Ilustración 139. Conectar una base de datos al proyecto. Regla para MySQL

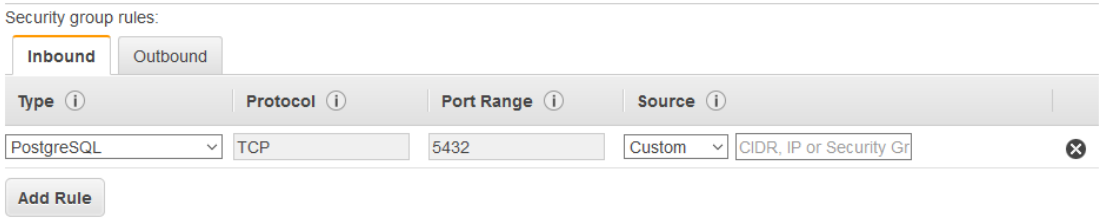


Ilustración 140. Conectar una base de datos al proyecto. Regla para PostgreSQL

Después de eso es imprescindible asignarle un bloque CIDR o IP desde las que se permitirá el acceso. En este caso se pretende un acceso desde cualquier dirección, luego el bloque será 0.0.0.0/0.

Finalmente, hay que asignarle a la base de datos RDS ese grupo de seguridad creado, desde la opción *Instances*.

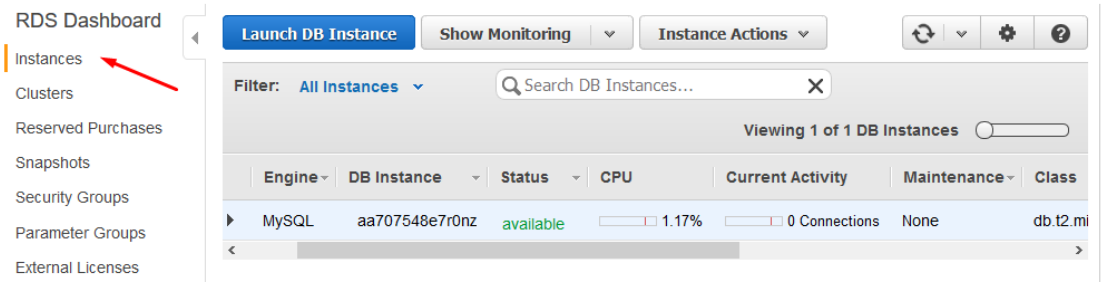


Ilustración 141. Conectar una base de datos al proyecto. RDS Dashboard

Se hace clic con el botón derecho en la instancia y luego a *Modify*:

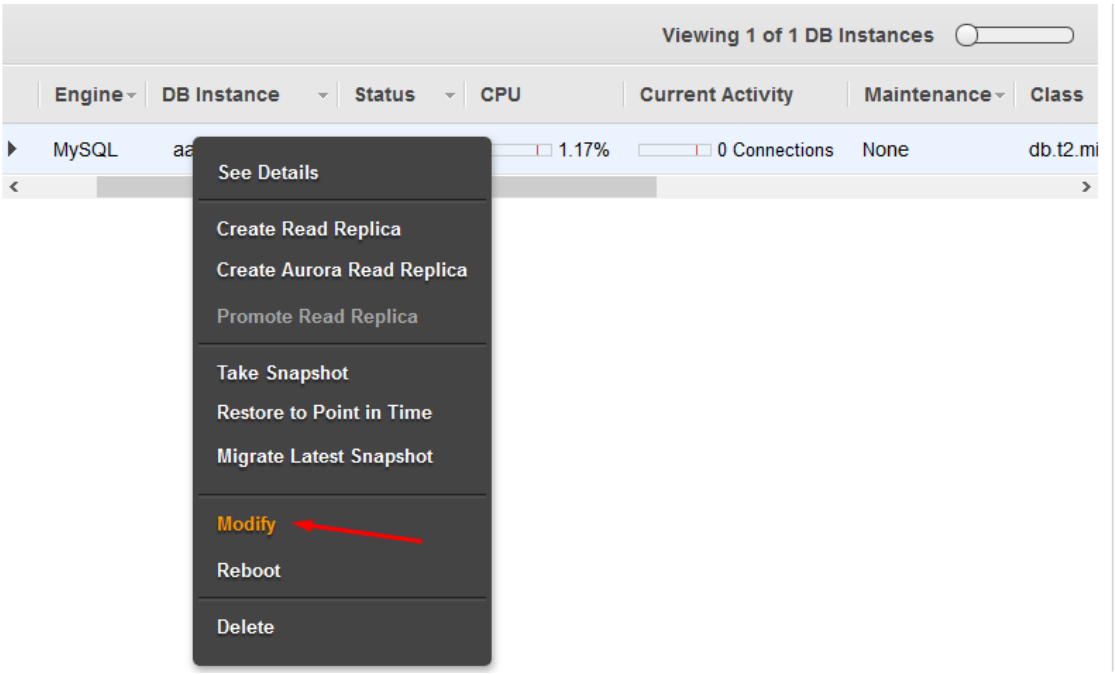


Ilustración 142. Conectar una base de datos al proyecto. Modificar la instancia

En la sección *Network & Security* se escogerá el grupo de seguridad que se haya creado:

Ilustración 143. Conectar una base de datos al proyecto. Selección del grupo de seguridad creado

En la siguiente ventana aparecerán los cambios que se van a aplicar a la base de datos con el nombre del grupo que se ha escogido:

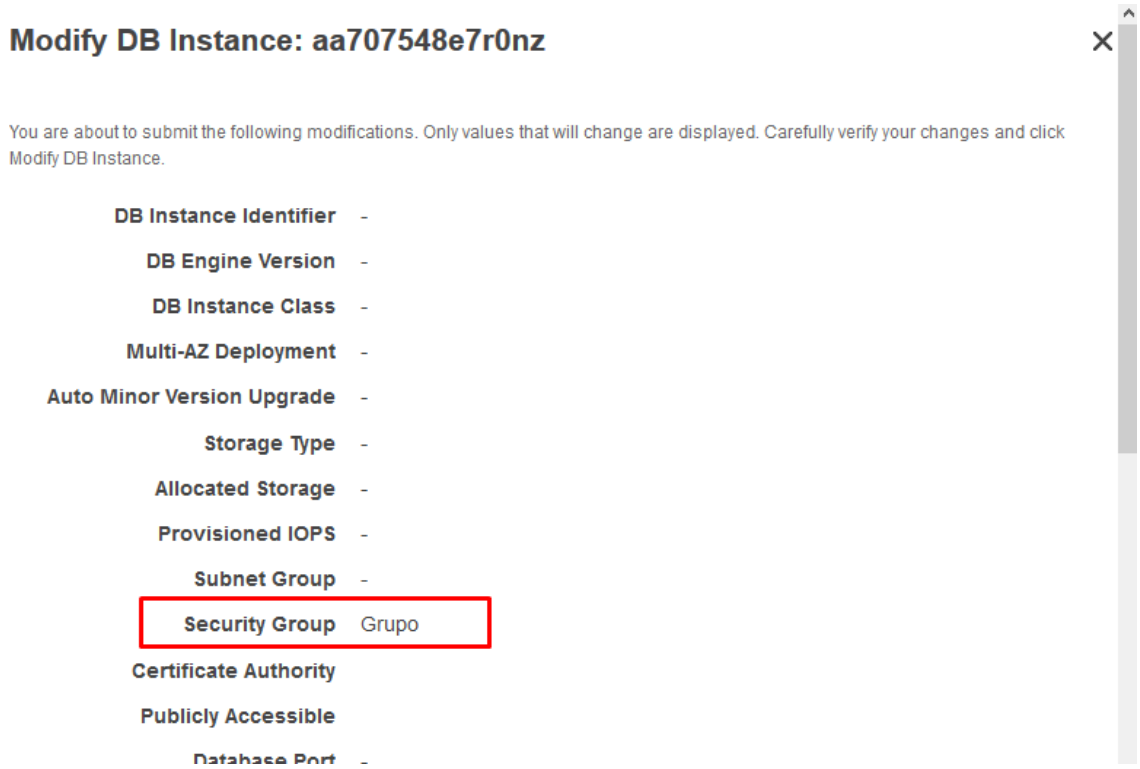


Ilustración 144. Conectar una base de datos al proyecto. Lista de cambios solicitados

Y modificamos la instancia:

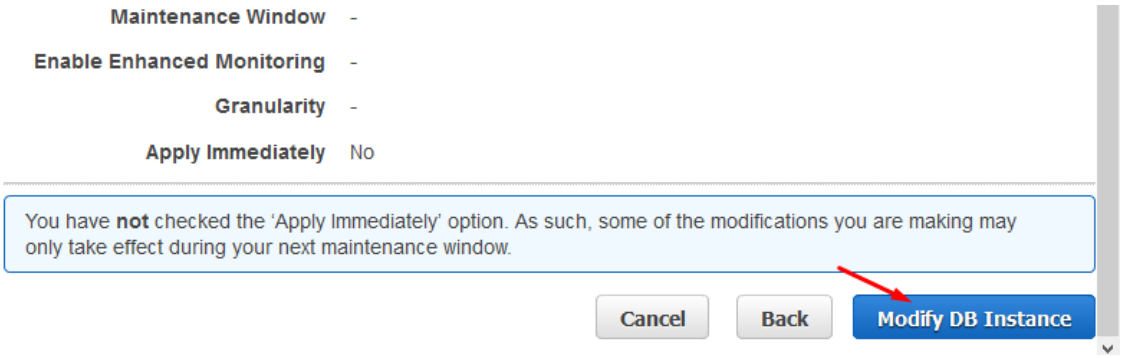


Ilustración 145. Conectar una base de datos al proyecto. Aplicar los cambios

Haciendo clic en el botón *Modify DB Instance*, la base de datos ya es accesible desde el exterior y puede utilizarse como fuente de datos en una aplicación de WebRatio.

III. Conexión por SSH

La instancia que se ha creado en la nube de Amazon se trata más concretamente de una instancia del servicio *Amazon BeanStalk*. Previamente a realizar la conexión por SSH es necesario asociarle a esa instancia un conjunto de claves o *key-pair* que sea compatible con el servicio de Amazon.

En la pestaña *Configuration* de la instancia hay que ir al apartado *Server*:

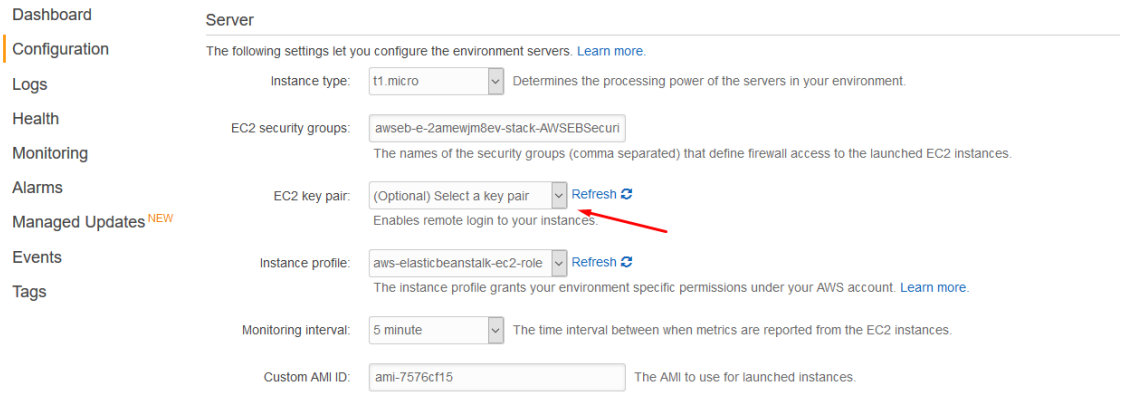


Ilustración 146. Conexión por SSH. Configuración del servidor

Se abrirá una ventana emergente en la que se darán indicaciones de los comandos para realizar la conexión desde la consola.

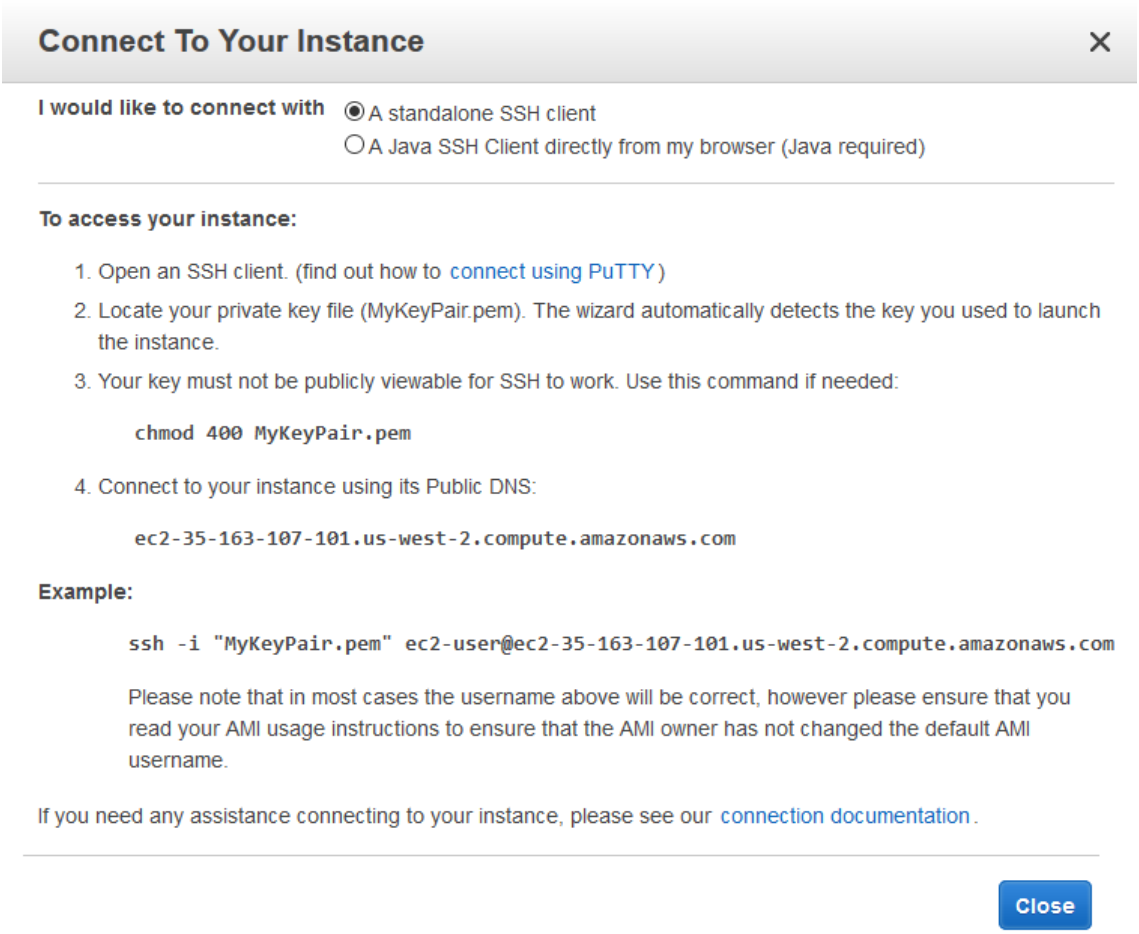


Ilustración 147. Conexión por SSH. Comandos de consola

Y ahora en una ventana de consola de comandos, se introducen las siguientes instrucciones para modificar los permisos del fichero y establecer la conexión.

Ilustración 148. Conexión por SSH. Ejemplo por consola

Al realizar los cambios en los permisos es posible que se lance un aviso de Warning advirtiendo que los cambios aplicados pueden afectar a la seguridad de la conexión. Este mensaje no tiene más relevancia, después se procede al resto de comandos y preguntará si se está seguro de seguir con la conexión. Cuando se haya hecho efectiva aparecerán las letras de Elastic Beanstalk, que es el servicio de Amazon al que se quiere acceder.

IV. Cambiar la configuración del proxy

Al ejecutar las pruebas sobre la aplicación desplegada a veces ocurría que el tiempo de la operación superaba el del *timeout* de Apache, y entonces el navegador devuelve el siguiente error:

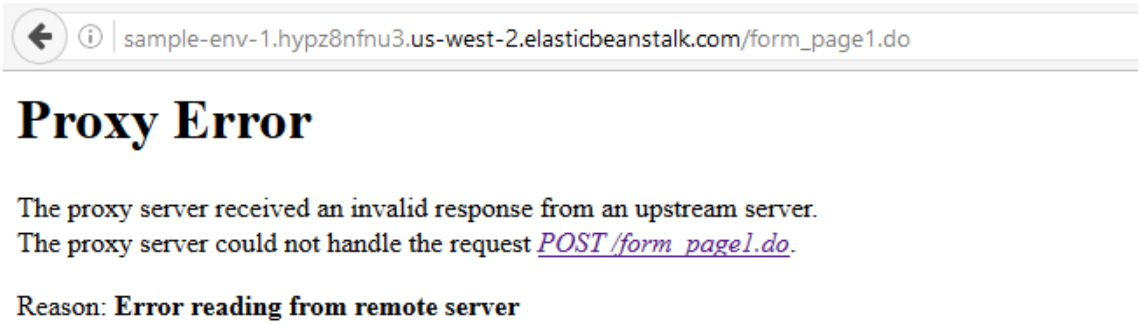


Ilustración 149. Cambiar la configuración del proxy. Mensaje de error

A priori el mensaje devuelto no da muchas pistas sobre cuál puede ser el motivo de ese error. Después de investigar, se llegó a la conclusión que era necesario modificar un archivo de configuración del servidor Apache del servicio de Amazon. Para ello es imprescindible establecer una conexión SSH con el servidor.

El archivo clave en cuestión que hay que modificar se encuentra en el directorio /etc/httpd.conf.

En ese archivo hay que prestar especial atención a dos valores: Timeout y KeepAliveTimeout. Por defecto su valor viene definido en 60 segundos, que era precisamente el tiempo que tardaba en devolver el error de la ilustración anterior.

Por tanto, sería necesario aumentar ese tiempo para que así el servidor tuviera más tiempo de procesar la petición y realizar todas las operaciones que se persigue medir.

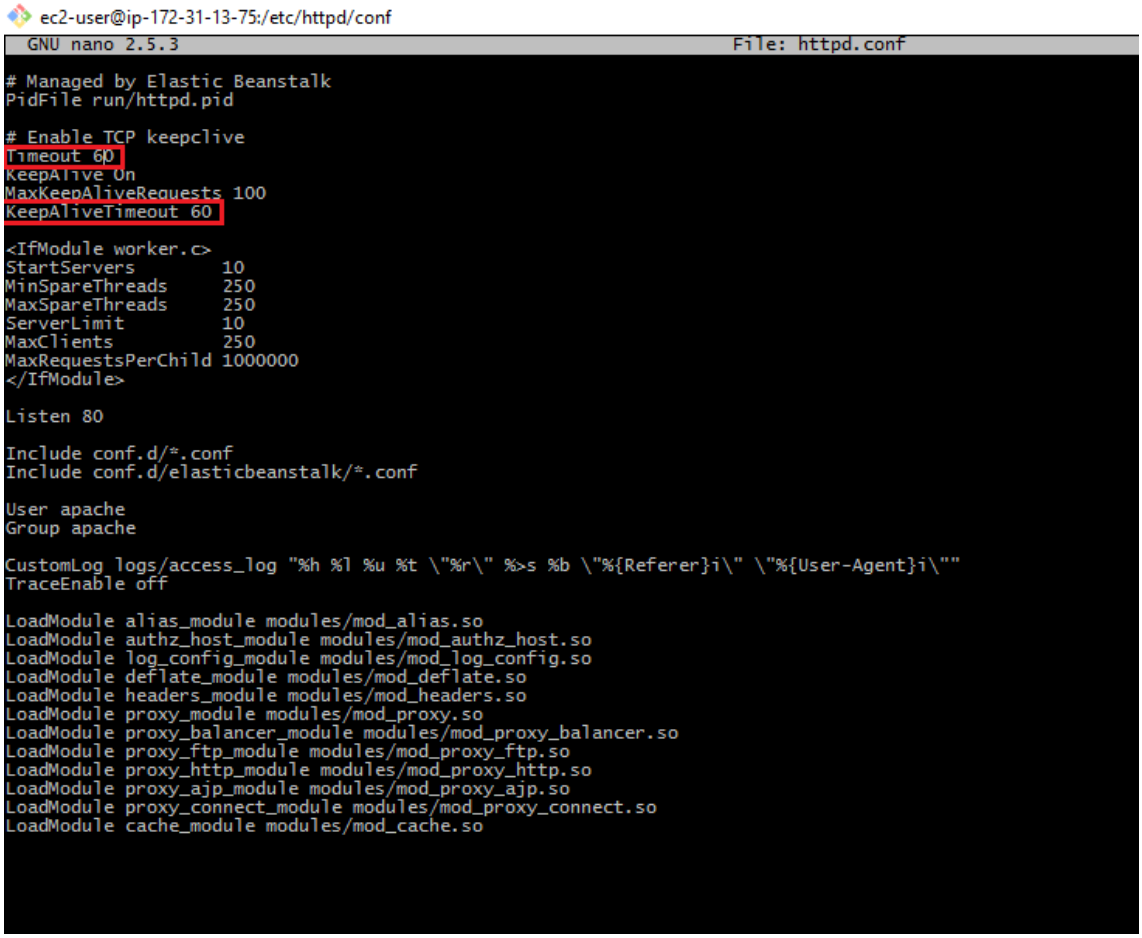


Ilustración 150. Cambiar la configuración del proxy. Modificación del archivo httpd.conf

Se optó por aumentar el valor a 3600, que equivale a una hora, y se guardaban los cambios. Con esta configuración ya se pudieron realizar todas las pruebas correctamente y de forma completa.

V. Problemas de la base de datos RDS Amazon

Durante la ejecución de las pruebas hubo un aspecto bastante llamativo de las bases de datos RDS de Amazon y es que no siempre disponen de la capacidad que se contrata. Por ejemplo, en el siguiente caso para esta configuración:

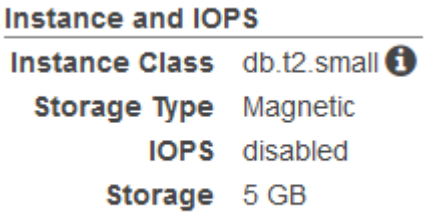


Ilustración 151. Problemas de la base de datos RDS Amazon. Especificaciones de la base de datos

En el monitor de la base de datos, en la sección de *Memory*, ponía 1110 MB, en lugar de 5GB:

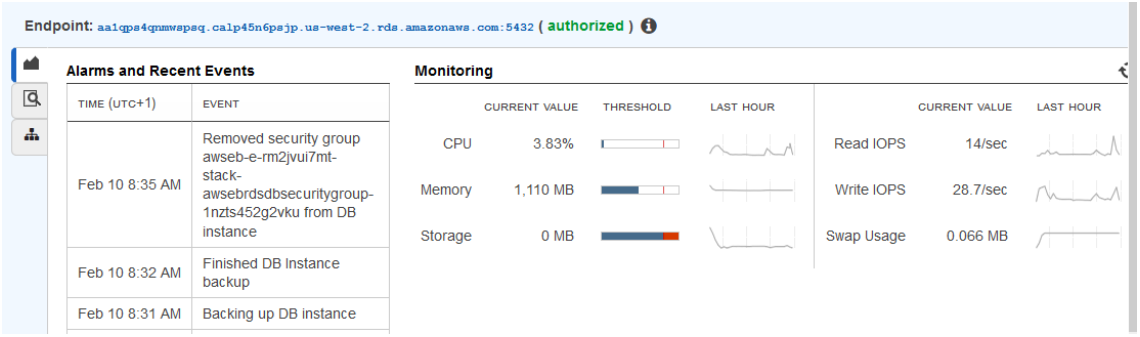


Ilustración 152. Problemas de la base de datos RDS Amazon. Monitor de la base de datos

Se ha podido comprobar que al realizar las pruebas hay picos en los que el rendimiento disminuye considerablemente y entonces parte del espacio contratado no se encuentra disponible.

Este hecho puede comprobarse en la siguiente gráfica:

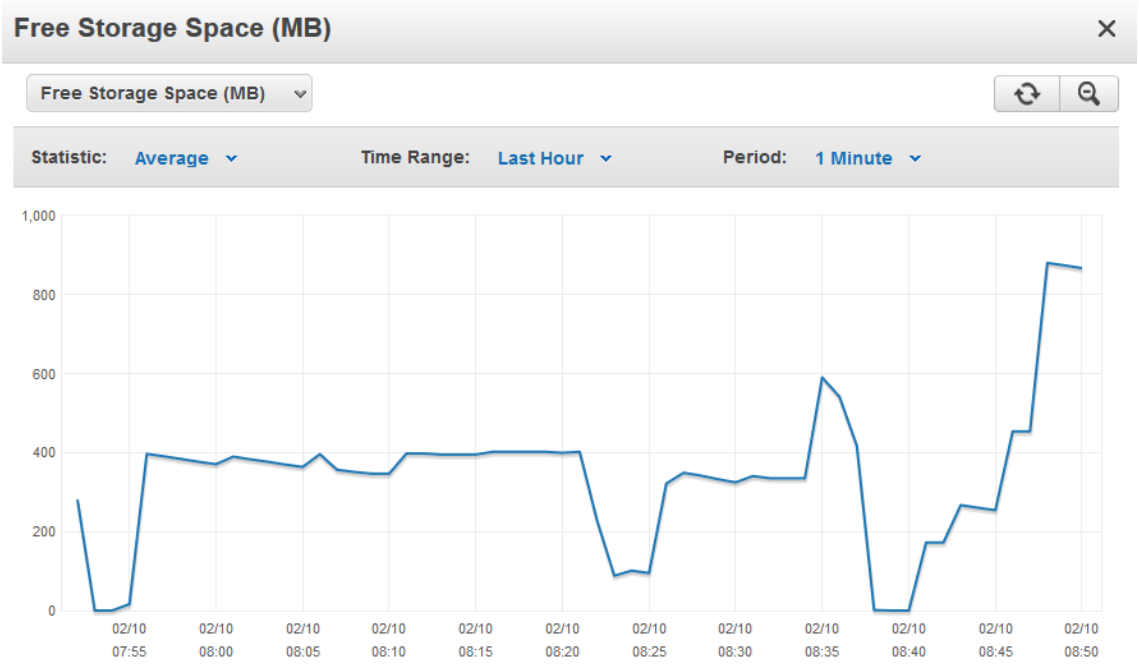


Ilustración 153. Problemas de la base de datos RDS Amazon. Gráfica de espacio libre de almacenamiento

Cuando los valores coincidían con 0, ocurría que en el monitor de la base de datos mostraba el mensaje de *storage-full*:

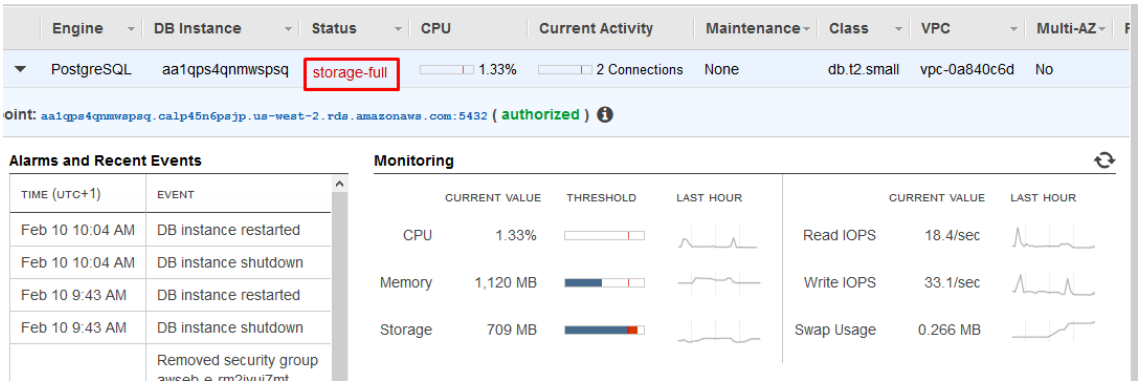


Ilustración 154. Problemas de la base de datos RDS Amazon. Storage-full

Debido a este problema, a menudo las pruebas no se terminaban de ejecutar completamente y era necesario repetirlas. Y no se debía a un asunto de la capacidad de almacenamiento como mostraba el monitor, sino al procesado de peticiones, eran muchas peticiones por segundo y la base de datos se colapsaba.

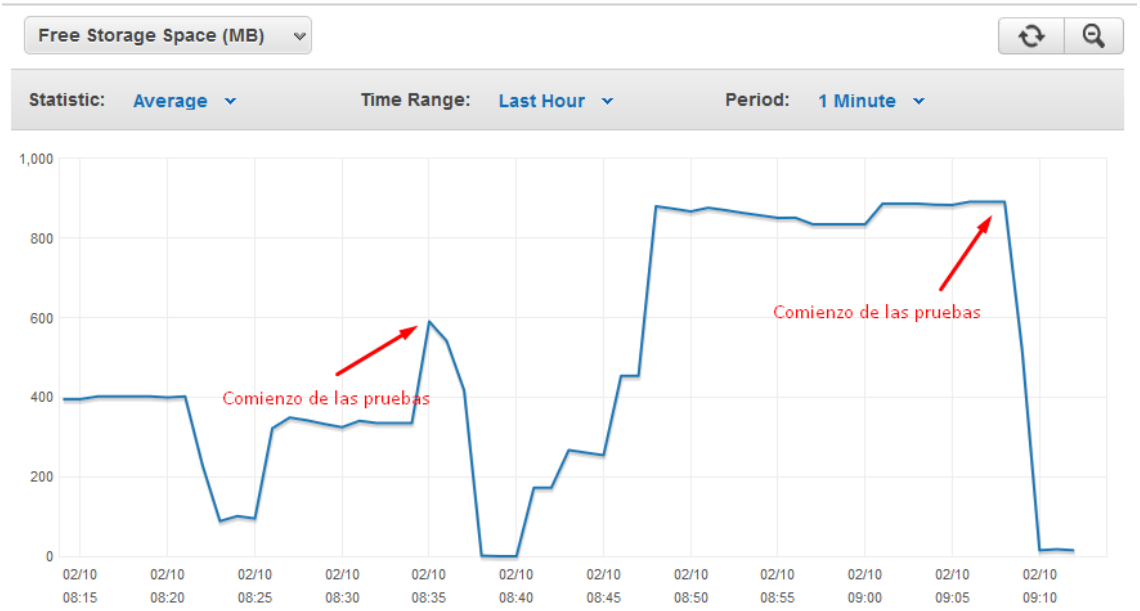


Ilustración 155. Problemas de la base de datos RDS Amazon. Gráfica de espacio libre de almacenamiento durante la ejecución de las pruebas

El número de operaciones por segundo era tan elevado que había una caída en picado de la capacidad que obligaba a esperar varios minutos hasta que se reestableciese la normalidad y volver a comenzar las pruebas. En ocasiones con esto no era suficiente y había que crear una nueva base de datos RDS.

VI. Apache Benchmark

Apache Benchmark es una herramienta orientada a realizar pruebas de carga desde la consola. Es mucho más ligera que Apache JMeter, no tiene interfaz gráfica de usuario y por tanto no añade esa sobrecarga al sistema a la hora de realizar las pruebas. Para utilizarla hay que usar el comando *ab* que tiene el siguiente conjunto de opciones:

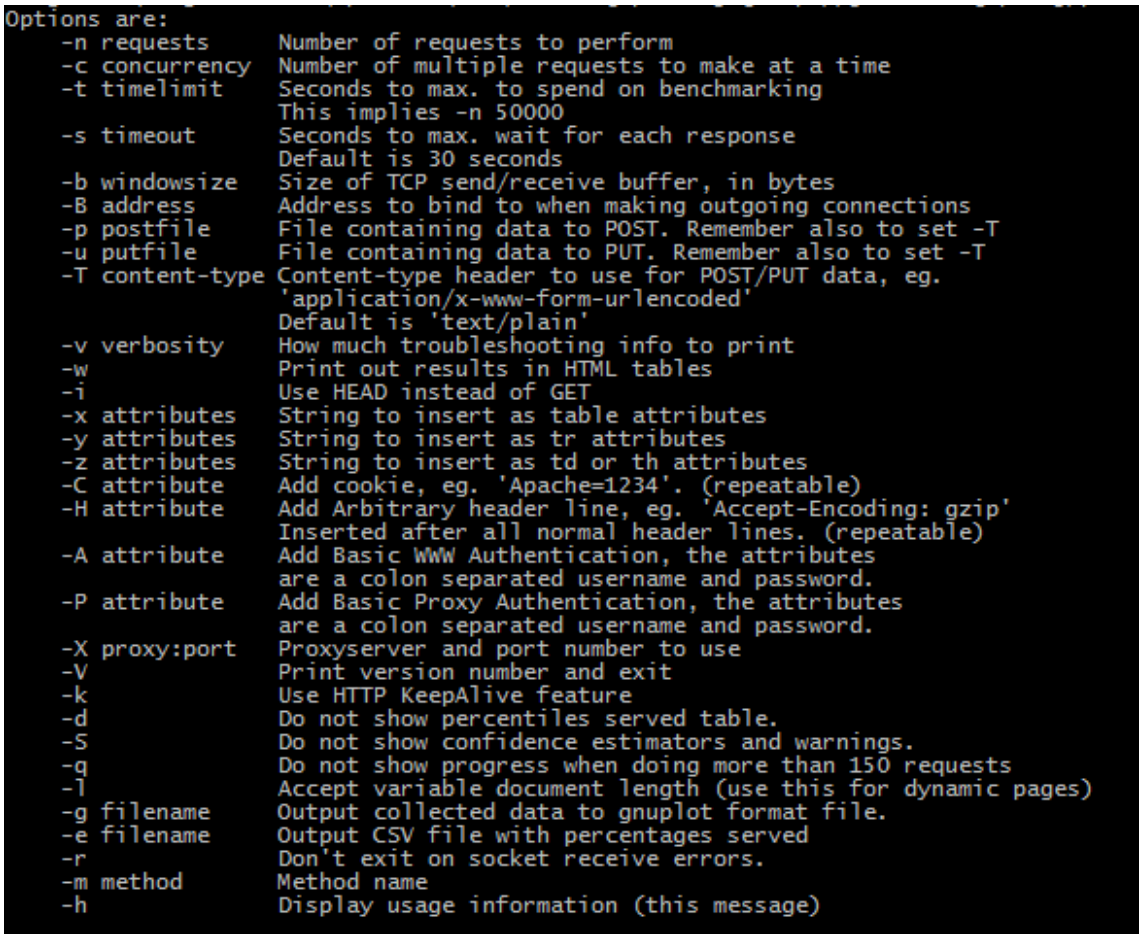


Ilustración 156. Apache Benchmark. Opciones Apache Benchmark

Un ejemplo de ejecución es el comando con la siguiente línea:

```
ab -c 5 -n 5 -k http://sample-env.hypz8nfnu3.us-west-2.elasticbeanstalk.com/
```

Lo que hace es realizar 1 petición por cada uno de los 5 usuarios concurrentes (haciendo un total de n = 5 peticiones).

```
$ ab -c 5 -n 5 http://sample-env.hypz8nfnu3.us-west-2.elasticbeanstalk.com/
This is ApacheBench, Version 2.3 <$Revision: 1757674 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking sample-env.hypz8nfnu3.us-west-2.elasticbeanstalk.com (be patient).....done


Server Software:        Apache-Coyote/1.1
Server Hostname:        sample-env.hypz8nfnu3.us-west-2.elasticbeanstalk.com
Server Port:            80

Document Path:          /
Document Length:        0 bytes

Concurrency Level:      5
Time taken for tests:   2.222 seconds
Complete requests:      5
Failed requests:        0
Non-2xx responses:      5
Total transferred:      1530 bytes
HTML transferred:       0 bytes
Requests per second:    2.25 [#/sec] (mean)
Time per request:       2221.913 [ms] (mean)
Time per request:       444.383 [ms] (mean, across all concurrent requests)
Transfer rate:          0.67 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      193  197   3.0    198     200
Processing:  1234 1628 312.2   1727    2023
Waiting:     1225 1624 313.2   1722    2021
Total:       1430 1825 312.9   1927    2218

Percentage of the requests served within a certain time (ms)
  50%   1826
  66%   2027
  75%   2027
  80%   2218
  90%   2218
  95%   2218
  98%   2218
  99%   2218
 100%   2218 (longest request)
```

Ilustración 157. Apache Benchmark. Resultado por consola

Al ejecutarlo, no obstante, en un principio el resultado devuelto no es el esperado. De acuerdo con la ilustración anterior, resulta que no se han recibido bytes y además los tiempos son muy bajos. Comprobando el estado de la base de datos, no se han realizado inserciones en la tabla, por tanto, el proceso no se ha ejecutado correctamente.

Después de investigar, lo que estaba ocurriendo es que el comando *ab* no sigue las redirecciones, y lo que estaba ejecutando era el index.jsp de la página sin redireccionar a la *NoOperationUnit* que ejecutaba todo el proceso. Para que se ejecutase había que poner la dirección específica de la unidad:

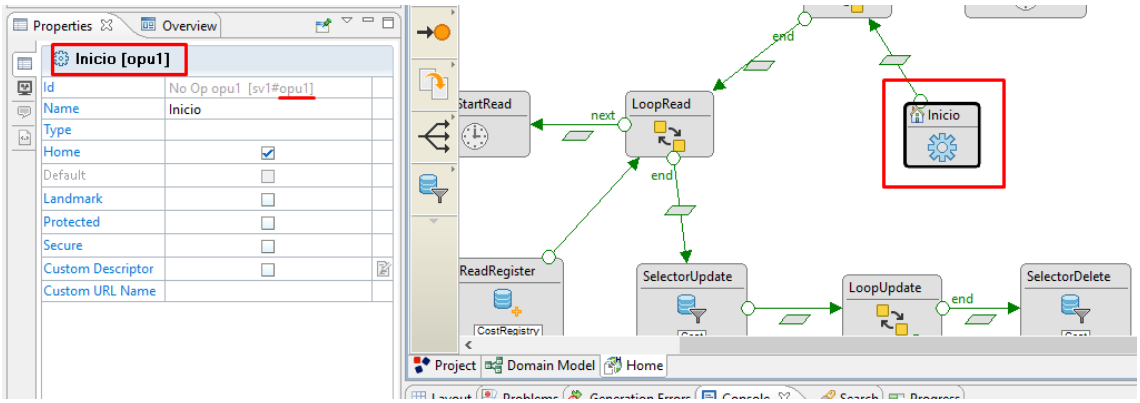


Ilustración 158. Apache Benchmark. Código id de la unidad

El comando entonces quedaría:

```
ab -c 5 -n 5 http://sample-env.hypz8nfnu3.us-west-2.elasticbeanstalk.com/opu1.do
```

Así, de esta forma ya sí se ejecuta correctamente el proceso y se realizan los cambios en la base de datos.

Por otro lado, el comando ab tiene un *timeout* por defecto de 30 segundos, como el proceso se lleva más tiempo hay que añadirle un modificador adicional:

```
ab -c 5 -n 5 -s 3600 http://sample-env.hypz8nfnu3.us-west-2.elasticbeanstalk.com/opu1.do
```

Para las configuraciones de Persistencia en base de datos y Aplicación no había problemas con el uso de Apache Benchmark. Sin embargo, no ocurre lo mismo con Sesión, ya que al hacer UPDATE y DELETE, Apache Benchmark lanza una nueva petición que genera un SESSION ID completamente nuevo y en el que no hay ningún objeto creado, por tanto, esas 2 operaciones no pueden probarse con Apache Benchmark ya que dependen de la sesión de otras peticiones, y cada petición es independiente en variable de sesión. Con Aplicación no ocurre lo mismo porque es una variable global.

Por este motivo, para conservar la sesión entre las operaciones se recurrirá a Apache JMeter para lanzar las peticiones (pero no para tomar los tiempos). Un ejemplo de plan de pruebas es el siguiente:

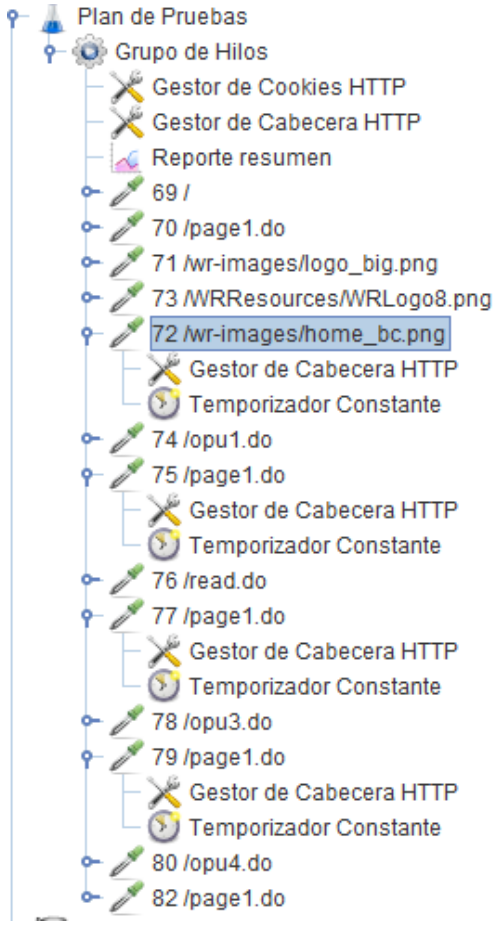


Ilustración 159. Apache Benchmark. Estructura del Plan de Pruebas con JMeter

Se han añadido temporizadores constantes para que al procesador tenga tiempo para liberar recursos.

Un ejemplo de ejecución del script de JMeter por consola es el siguiente:

```
Ruben@DESKTOP-K6COALO MINGW64 ~/Desktop/apache-jmeter-3.0/bin
$ jmeter -n -t Reporte\ resumen.jmx
Writing log file to: C:\Users\Ruben\Desktop\apache-jmeter-3.0\bin\jmeter.log
Creating summariser <summary>
Created the tree successfully using Reporte resumen.jmx
Starting the test @ Tue Feb 21 10:20:59 CET 2017 (1487668859177)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +      1 in 00:00:01 =    0,9/s Avg:   930 Min:   930 Max:   930 Err:
  0 (0,00%) Active: 1 Started: 1 Finished: 0
summary +      4 in 00:01:01 =    0,1/s Avg:   272 Min:   197 Max:   458 Err:     0 (0,00%) Active: 1 Started: 1 Finished: 0
summary =      5 in 00:01:02 =    0,1/s Avg:   403 Min:   197 Max:   930 Err:     0 (0,00%)
summary +      8 in 00:13:48 =    0,0/s Avg: 81013 Min:   211 Max: 620453 Err:     0 (0,00%) Active: 0 Started: 1 Finished: 1
summary =     13 in 00:14:50 =    0,0/s Avg: 50009 Min:   197 Max: 620453 Err:     0 (0,00%)
Tidying up ...    @ Tue Feb 21 10:35:49 CET 2017 (1487669749650)
... end of run

Ruben@DESKTOP-K6COALO MINGW64 ~/Desktop/apache-jmeter-3.0/bin
$
```

Ilustración 160. Apache Benchmark. Ejecución Apache JMeter por consola

VII. Índice de términos