Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

ÁREA DE LENGUAJES Y SISTEMAS INFORMÁTICOS

TRABAJO FIN DE MÁSTER

OPTIMIZACIÓN DE RECURSOS EN PRUEBAS DE SISTEMA

MEMORIA

D. AUGUSTO ALONSO, Cristian
TUTOR: D. DE LA RIVA, Claudio

FECHA: JULIO 2020

Tabla de contenido

Tabla de Ilustraciones

Tablas

Fragmentos de Código

1. Introducción

La ejecución de grandes conjuntos de casos de pruebas en productos software complejos es algo costoso que puede llegar a requerir prohibitivas cantidades de recursos como pudiera ser tiempo, dinero o infraestructuras lógicas/físicas [1]. Para resolver este tipo de problemáticas, en otros campos científico-técnicos suelen abordar estas referidas como uso eficiente u optimización de recursos. La problemática más recurrente suele ir relacionada con dados unos recursos finitos, optimizar su uso para maximizar los beneficios o resultados (realizar un estudio de la calidad del código con una efectividad similar). En el campo de las pruebas, el rápido crecimiento que ha experimentado el sector del desarrollo durante los últimos años ha dado de sí varias problemáticas en las que hay un amplio margen de mejora en cuanto a la optimización de recursos. Las últimas tendencias, acorde con el crecimiento y la popularidad de los entornos de desarrollo continuos y las infraestructuras en el *Cloud* [2], son el uso de los recursos contenerizados/ virtualizados con objeto de aprovechar los potenciales recursos flexibles e "ilimitados" bajo demanda que nos ofrece el *Cloud*.

A pesar de que la virtualización/contenerización ha supuesto diversos avances en cuanto a uso de los recursos, hay una serie de desafíos abiertos como la ejecución eficiente de pruebas de sistema (también denominadas *End to End*) contenerizadas. Durante la ejecución de las pruebas de sistema, son necesarios sistemas cuyo despliegue y/o preparación son costosos y complejos, debido a los cuales ejecutarlas tantas veces como sería necesario se convierte en prohibitivo [1][3]. Esta problemática se agrava cuando dichas pruebas son incorporadas en entornos de integración continua, donde serán re-ejecutadas por cada nueva versión desplegada, con el consiguiente gasto de recursos. Para optimizar estos recursos se podrían realizar menos pruebas, repercutiendo en que quedasen defectos encubiertos en el software y una vez desplegado, arreglarlos en producción pueda tener un coste hasta 100 superior [4]. En cambio, otra solución que si ejecutase todos los casos de prueba, podría pasar por optimizar los recursos orquestando las pruebas de manera que se pudiera reutilizar los recursos que ya hubiera desplegados.

Este trabajo de fin de máster presenta una técnica de orquestación denominada RETORCH (*Resource-aware E2E Test ORCHestration*) compuesta de una caracterización racional de los recursos empleados en las pruebas de sistema para posteriormente basándose en dicha identificación realizar un agrupamiento y ordenación de los casos de prueba acorde al uso que estos hacen de los recursos con el objetivo de reducir el coste

Las ideas previas fueron presentadas en QUATIC [5], artículo que más tarde sería aceptado y extendido para la revista *Software Quality Journal* [6]. Recientemente RETORCH ha sido presentado cómo póster en el *ACM Student Contest* de la 42th *International Conference on Software Engineering* [7].

El presente documento se estructura como se describe a continuación: en el apartado introductorio se resume de forma breve la contribución y se da una visión a alto nivel de los términos y tecnologías empleadas. En las Secciones 2. y 3. se describen los objetivos y la relación con el estado del arte actual, en la Sección 4. se presentan las hipótesis de partida y el alcance del trabajo. Las secciones 5. 6. tratan las metodologías empleadas y el trabajo realizado para finalmente tener la validación de RETORCH y las conclusiones en las Secciones 8. y 9.

2. Objetivos concretos

El marco de trabajo RETORCH busca optimizar el uso de los recursos para mejorar el coste de ejecución de las pruebas de sistema. Para ello, RETORCH orquesta las pruebas de sistema en diferentes máquinas basándose en los recursos que se requieren para ejecutarse. En este trabajo de fin de máster se propone realizar una prueba de concepto, con una serie de objetivos que representarían los requisitos de usuario:

1. Crear un modelo que pueda ser empleado para caracterizar los recursos que se requieren en las pruebas de sistema. Dicho modelo, permitirá hacer una definición de cada recurso respecto uso que pudiera hacer los casos de prueba de estos, independientemente de las tecnologías, plataformas o lenguajes que se empleasen.

2. Respaldándose en el anterior modelo, se busca crear un sistema de identificación de recursos que, vía metadatos, anotaciones u cualquier otro mecanismo, permita

aportar información del recurso y el uso que se le da. Este sistema deberá dar como salida una estructura con la información recogida en el modelo conceptual del punto 1.

3. A partir de la identificación anterior, se pretende crear otro sistema que de cómo salida un agrupamiento de casos de prueba que puedan ser ejecutados juntos, debido al uso que hacen de los recursos.

4. Basándose en la anterior salida, se propone la creación de otro sistema que permita organizar los casos de prueba con necesidades similares, para optimizar su ejecución acorde a varios objetivos.

5. Se propone estudiar la automatización del RETORCH y su integración con la plataforma *ElasTest* o un sistema de integración continua.

6. Diseño de una arquitectura compuesta de los componentes anteriormente descritos, con objeto de que el presente trabajo fin de máster sea una prueba de concepto de RETORCH.

3. Relación con el estado actual

Este trabajo parte de una problemática encontrada durante el desarrollo del proyecto *ElasTest* [8], enmarcado en el programa europeo *HORIZON 2020*. Este proyecto tiene como objetivo desarrollar una plataforma para facilitar la ejecución de las pruebas, pero se encontraron que al ejecutar y automatizar las pruebas del sistema se requerían grandes esfuerzos en términos de recursos. Estos recursos van más allá del tiempo o el coste monetario, por lo que al igual que habían planteado otros autores [9], la ejecución de las pruebas podría abordarse en diversas dimensiones (recursos, tiempo, dinero, restricciones entre otros) a considerar. Centrada en esta problemática, la técnica propuesta en este trabajo (RETORCH) propone una solución específica para el caso de las pruebas de sistema, considerando una serie de recursos y el tiempo empleado en la ejecución. Ortogonales a RETORCH existen una serie de líneas de trabajo que son parte del estado de arte actual referidas a: (1) técnicas de reducción, priorización y minimización (2) detección de

dependencias entre casos de prueba (3) técnicas de optimización de recursos y finalmente (4) técnicas de orquestación.

3.1 Técnicas de reducción, priorización y minimización de casos de prueba

A pesar de todos los avances que ha habido respecto al uso eficiente y efectivo de recursos en las pruebas, hay una serie de desafíos como la automatización completa de los conjuntos de pruebas [10] que deben de ser solucionados cuando se realizan técnicas de priorización, reducción o minimización de casos de prueba. La minimización, reducción y priorización de casos de prueba han sido ampliamente estudiadas en la literatura. Yoo y Harman [11], han estudiado varios trabajos sobre estas temáticas y han comparado los resultados de estas técnicas en términos de tasa de detección de fallos y efectividad. Además de esto, en el citado trabajo, han estudiado y discutido los problemas que permanecen abiertos y las líneas de trabajo futuro que surgen a raíz de estas tres técnicas. Varios autores [12]–[14] han estudiado diversos enfoques con objeto de optimizar estas técnicas, considerando tanto el coste de ejecución de las pruebas como su desempeño en cuanto a tasa de detección de fallos. Estas técnicas han sido también empleadas con éxito en grandes compañías como Google [3], en la cual plantean crear subconjuntos de casos de prueba de acuerdo con el número histórico de modificaciones y de probabilidades de detectar un fallo. Otras líneas de investigación, combinan estas técnicas con *Machine Learning* [15], para priorizar los casos de acuerdo con los requisitos de cobertura, coste de ejecución e histórico de fallos detectados por los casos de prueba.

Este trabajo comparte aspectos con los ordenamientos de casos de prueba provistos por las técnicas de priorización [11], teniendo en cuenta la información provista por la previa identificación de recursos.

3.2 Detección de dependencias entre pruebas

Durante la priorización, minimización o reducción de casos de prueba, una problemática a tener en cuenta son las dependencias entre casos de prueba. Algunos autores han propuesto direccionar ese problema con una serie de herramientas, utilidades o técnicas. Bell et al. [16] han desarrollado una herramienta para detectar dependencias denominada *Electrictest*, que

comparada con otras herramientas de referencia, ha demostrado empíricamente obtener una tasa de detección de fallos similar con una menor recarga del sistema. *Electrictest* fue mejorada por Gambi et al. [17], quienes han obtenido mejores resultados: detectando tanto dependencias ya conocidas como otras nuevas (no detectadas anteriormente por ninguna herramienta), en una batería de casos de prueba estándar de facto en el campo. Gyori et al. [18] han introducido el concepto de pruebas contaminantes (*test pollution*) y han implementado otra herramienta (*PolDet*) que detecta en tiempo de ejecución dichos test "contaminantes"

Este trabajo presenta un marco de trabajo para optimizar los recursos durante la ejecución de los casos de prueba, evitando que los redespliegues innecesarios vía una agrupación de aquellas pruebas que no tienen dependencias entre sí. Somos conscientes de que las dependencias entre pruebas juegan un papel clave a la hora de descubrir las relaciones entre los casos y los recursos que emplean. Es por ello, que el enfoque de RETOCH pretende introducir un mecanismo de detección de dependencias que mejore la eficiencia de las pruebas, agrupando aquellas que no se interfieren durante ejecución.

3.3 Optimización de recursos en las pruebas

La optimización de los recursos durante las pruebas ha sido ampliamente tratada por la literatura. Varios trabajos presentan sopesar distintos objetivos a optimizar como el tiempo, el coste de ejecución o una mezcla de ambos [19], u optimizar los recursos a la vez que se cumplen con ciertas restricciones de tiempo [20]. Otros autores [21] se han centrado en la optimización de costes vía diferentes procesos de partición, agrupamiento y redistribución de los casos de prueba con el objetivo de paralelizarlos. Estas agrupaciones o particiones de casos de prueba están también presentes en las técnicas de clustering aplicadas a resolver este problema de optimización [22], en las cuales los recursos son enlazados con los casos de prueba para descubrir las dependencias subyacentes y ejecutarlos.

A diferencia de los trabajos previos, este trabajo propone optimizar los recursos no solo centrándonos en el tiempo o el coste, sino también en los recursos empleados durante la fase de pruebas. García et al. [8], también han propuesto orquestar los casos de prueba a través de una selección y secuenciación basándose en la salida de su ejecución (veredict-driven) o en la salida producida (data-driven).

En el campo de los servicios en el *Cloud*, se han propuesto varios enfoques para enfrentarse a problemáticas relacionadas con la optimización de recursos. La plataforma de Microsoft, *CloudBuild* [23], ha experimentado problemáticas muy similares al extraer los grafos y derivar las dependencias de ellos automáticamente. Basándose en ese análisis de dependencias, Microsoft *CloudBuild* optimiza la ejecución de las pruebas a través del despliegue y la ejecución de los casos de prueba que cubren el código que se ha cambiado. RETORCH propone una línea de trabajo futuro que trate de lograr una detección automática similar dentro de los contenedores.

3.4 Orquestación

Dependiendo del campo al que nos refiramos, el término orquestación tiene diferentes significados y uso. En la jerga coloquial, la orquestación se refiere a la acción de coordinar y ordenar varios componentes mejorando su ejecución. En el campo de las Redes, Giotis et al. [24] han propuesto orquestar Redes Funcionales Virtualizadas (NFV) con el objetivo de administrar el tráfico de red basado en políticas en las mismas. En el campo del *Cloud*, hay un gran número de sistemas de orquestación, como por ejemplo *Kubernetes* [25], *Borg* [25], *Swarm* [26], *Fuxi* [27], *System Center–Orchestrator* [28] compartiendo arquitecturas cómo TOSCA (*Cloudify* and *Kubernetes*) [29].

Muy relacionado con los servicios en el *Cloud*, la denominada *Fog and Edge Computing* también direcciona varios desafíos relacionados con la optimización de recursos [30]. Estos desafíos suelen orientarlos vía diferentes arquitecturas que presentan en común una figura de un "orquestador" actuando como ente que asigna y organiza los recursos disponibles [31], [32]. Muchos de estos trabajos han sido analizados de acuerdo a sus técnicas de ordenación, asignación, compartición y optimización de recursos por Tozé et al. [33] proponiendo una taxonomía de recursos en el denominado "*Edge*".

Todos estos trabajos proponen técnicas de orquestación que optimizan los recursos atendiendo a sus dominios específicos. El enfoque buscado por RETORCH es optimizar los recursos empleados en las pruebas de sistema (denominadas también End to End) reduciendo el tiempo de ejecución al mismo tiempo que se logran ahorros en términos de recursos.

4. Hipótesis de partida y alcance

La investigación realizada en este trabajo nace del interés por lograr una mejor comprensión de los recursos empleados en las pruebas de sistema, para así realizar una caracterización racional de los mismos (*Resource Identification*), previa a un proceso de agrupamiento (*Grouping*) y ordenación de las pruebas optimizando su ejecución (*Scheduling*). El estado del arte actual junto con la problemática encontrada, sugieren una serie de hipótesis de partida que a continuación se relacionan:

Hipótesis 1: La ejecución de conjuntos de casos de prueba de Sistema (*End to End o E2E*) pueden requerir grandes cantidades de recursos físico-lógicos, con los que la efectividad de técnicas como la paralelización, minimización o reducción están limitadas, debido a las dependencias o a la distribución del consumo de recursos.

Hipótesis 2: La ejecución de pruebas de sistema puede ser optimizada analizando las dependencias entre pruebas buscando concurrencia en el acceso a los recursos (ej. variables, dispositivos hardware, archivos, etcétera). Una vez esas dependencias son detectadas, los casos de prueba que interfieren con otros pueden ser aislados o modificados para evitar efectos colaterales en la ejecución de las pruebas.

Hipótesis 3: Una caracterización racional de los casos de prueba acompañada de un agrupamiento y una ordenación atendiendo a los recursos que emplean, puede derivar en un ahorro de tiempo/coste en la ejecución del conjunto de casos de prueba.

5. Metodología de trabajo

Para acometer el presente proyecto se ha empleado una adaptación de la metodología de desarrollo SCRUM. SCRUM se basa organizar el desarrollo del proyecto en una serie de entregas parciales o incrementos denominados *Sprints*. Este tipo de metodologías al igual que muchas otras de las denominadas ágiles, se basa en ir entregando al cliente lo que necesita, adaptando de manera flexible el alcance de dicho proyecto modulándolo a lo que le proporcione valor al cliente. Referente a este proyecto podríamos destacar que:

- **Reuniones continuas**: Me he reunido con mi tutor al menos una vez a la semana con objeto de comentar los avances y marcar los pasos a seguir durante el tiempo que ha durado el proyecto

- **Roles desempeñados**: mi tutor ha ejercido de dueño del producto y los roles de equipo y *Scrum Master*, los he desempeñado yo

- **Sprints**: La duración de los *Sprints* ha sido variable, pero se ha intentado mantener los mismos en torno a las dos semanas de duración.

Durante el proyecto se ha empleado un tablero tipo *Kanban* usando Microsoft *Planner* en el cual se han tenido las historias de usuario de cada uno de los *Sprints* en los diferentes depósitos (backlog del sprint, en curso, finalizadas…). El proyecto comenzó el día 5 de marzo con una reunión inicial con el tutor, en él se acordó que la primera tarea seria desarrollar este apartado en la propia herramienta MS *Planner*.

El proyecto se planificó en seis *Sprints* poniéndonos como objetivo empezar el 13 de marzo y acabar el 5 de junio. En la siguiente ilustración (Ilustración 1), se presenta el *Story Mapping* del proyecto con objeto de representar el mismo en dos dimensiones: los *Sprint*s en el eje vertical y las funcionalidades en el eje horizontal.

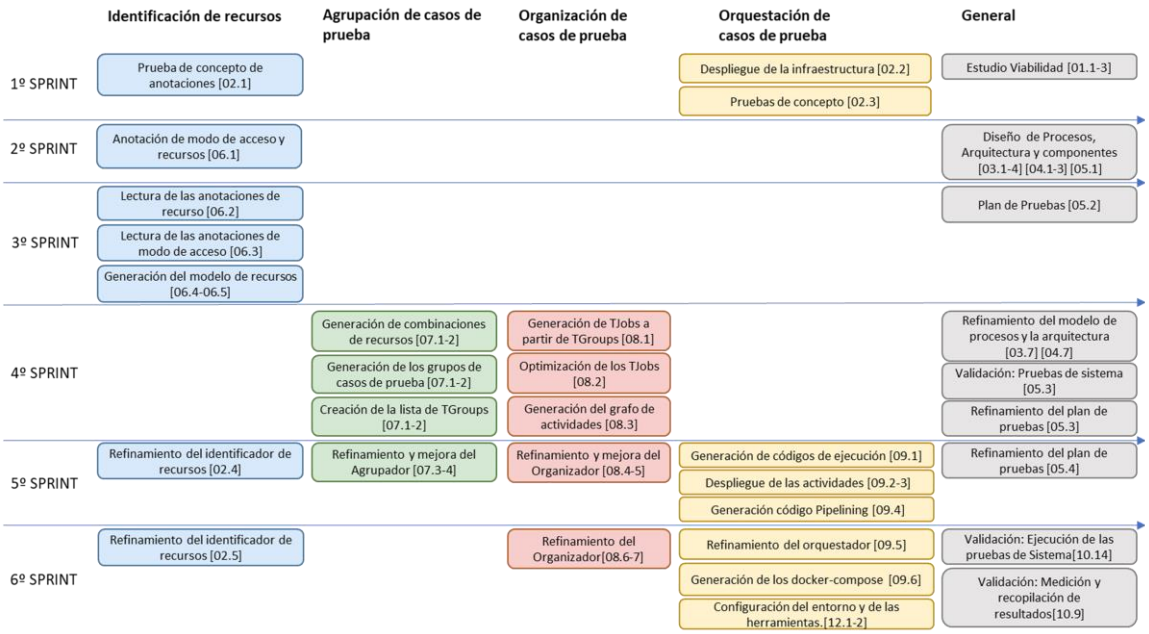

Ilustración 1 Planificación Story Mapping

El *Story Mapping*, presenta las historias de usuario, junto con su identificador entre corchetes al lado de estas. Las historias de usuario están numeradas de acuerdo con una serie de *EPICS* que se presentan en la Tabla 1:

Tabla 1 Tabla de EPICS

Descripción	Etiquetas
01. Estudio del estado del arte	Memoria
02. Pruebas de concepto iniciales	Implementación y Pruebas
03. Modelado de conceptos	Conceptualización
04. Modelado de la arquitectura	Conceptualización
05. Diseño	Diseño
06. Proceso de Identificación de Recursos	Implementación y Pruebas
07. Proceso de Agrupación	Implementación y Pruebas
08. Proceso de Ordenación	Implementación y Pruebas
09. Proceso de Despliegue	Implementación y Pruebas
10. Documentación	Memoria
11. Presentación	Presentación
12. Configuración de la infraestructura	Implementación y Pruebas

Las historias de usuario se presentan en la Tabla 2, organizadas por el sprint al que pertenecen con una etiqueta que busca clasificar a que “clase” o elemento pertenecen:

Tabla 2 Historias de usuario

Descripción	*Sprint*	Etiqueta
01.1. Estudio del funcionamiento de la plataforma *ElasTest*, así como que utilidades provee para realizar la orquestación	1	Memoria
01.2. Estudio de diferentes alternativas que existen para proveer de información vía anotaciones en los distintos lenguajes de programación	1	Memoria
01.3 Estudio de las diferentes propuestas que existe en cuanto a optimización de recursos, en pruebas y mecanismos de orquestación de estas	1	Memoria
02.1. Implementación y estudio de la viabilidad de proveer información sobre los recursos para cada caso de prueba	1	Implementación y Pruebas
02.2. Realización de diferentes pruebas de concepto con *Jenkins*, así como las diferentes extensiones/*addons* que esta pudiera tener	1	Implementación y Pruebas
02.3 Desarrollo y realización pruebas de concepto de la librería de orquestación provista en la plataforma *ElasTest*.	1	Implementación y Pruebas
10.1. Formato y adecuación de la memoria	1	Memoria
10.2. Redacción de la introducción	1	Memoria
10.3. Redacción del estado del arte.	1	Memoria
03.1 Modelado del proceso de Identificación de Recursos de forma que queden claras las entradas y salidas de los mismos	2	Conceptualización
03.2 Modelado del proceso de Agrupación, asentando las bases de los conceptos de *TGroups* y como se podrían sacar estos de cara a la salida que nos provee el proceso de Identificación de Recursos.	2	Conceptualización
03.3. Modelado del proceso de Ordenación atendiendo a como se podría organizar la ejecución de los casos de prueba:	2	Conceptualización
03.4. Modelado del proceso de Orquestación, una vez se tiene el plan de cómo van a ejecutarse los casos de prueba	2	Conceptualización
04.1. Modelado de la arquitectura del proceso de Identificación de Recursos	2	Conceptualización
04.2 Modelado de la arquitectura del sistema de Agrupación	2	Conceptualización
04.3. Modelado de la arquitectura con los componentes que se encargarían de realizar el proceso de Ordenación	2	Conceptualización
05.1. Diseño de la arquitectura y componentes que tendrá el sistema	2	Diseño

06.1. Implementación del sistema de anotaciones de RETORCH	2	Implementación y Pruebas
10.4. Formulación y redacción de las Hipótesis	2	Conceptualización; Memoria
10.5 Estudio y análisis de viabilidad de alternativas de implementación	2	Implementación y Pruebas; Memoria; Diseño
03.5. Refinado del modelo conceptual en proceso de Agrupación	3	Conceptualización
03.6 Refinado del modelo conceptual del proceso de Ordenación	3	Conceptualización
04.4. Refinado de la arquitectura en proceso de Identificación de Recursos	3	Conceptualización
04.5. Refinado de la arquitectura del proceso de Agrupación	3	Conceptualización
04.6 Refinado la arquitectura del proceso de Ordenación	3	Conceptualización
05.2. Plan de Pruebas	3	Diseño
06.2. Implementación de un componente que realice la lectura de las anotaciones de recurso junto con las pruebas de que se correspondan	3	Implementación y Pruebas
06.3. Implementación de un componente que realice la lectura de las anotaciones de modo de acceso junto con las pruebas que correspondan para asegurar el funcionamiento del mismo	3	Implementación y Pruebas
06.4. Integración de ambos módulos de Identificación de Recursos e identificación de modos de acceso, para dada una clase recopilar los casos de prueba de esta con sus recursos y modos de acceso	3	Implementación y Pruebas
10.6 Refinamiento y mejora de las hipótesis de partida y el alcance del proyecto	3	Conceptualización; Memoria
6.5. Integración del módulo 6.4 en otro módulo que analice un paquete y sea capaz de devolver el Sistema: casos de prueba, recursos y modos de acceso, que posteriormente se emplearán en el proceso de Agrupación	3	Implementación y Pruebas
03.7 Refinamiento del modelo conceptual	4	Conceptualización
04.7. Refinado de la arquitectura del sistema: Cambio en las entradas y las salidas del sistema	4	Conceptualización
05.3 Refinamiento del plan de pruebas: mejora de la validación e incorporación de los cambios del modelo conceptual	4	Diseño
07.1 Implementación un módulo que, dado un sistema, sea capaz de agruparme los casos de prueba de acuerdo con los recursos que emplean en *TGroups*	4	Implementación y Pruebas

07.2 Implementación de un módulo, que extienda el desarrollado en 7.2 y que sea capaz de además de segregar por recursos, hacerlo por modos de acceso.	4	Implementación y Pruebas
08.1. Implementación de un módulo que dados una serie de *TGroups*, sea capaz de realizar una agrupación básica de *TJobs*	4	Implementación y Pruebas
08.2. Implementación de un módulo que extienda el desarrollado en 8.1 que sea capaz de dados unos *TJobs* optimizarlos de acuerdo con los recursos disponibles	4	Implementación y Pruebas
08.3. Implementación de un módulo que extienda los módulos 8.2 y 8.3 que dados una serie de *TJobs* optimizados, sea capaz de generar las actividades que requieren.	4	Implementación y Pruebas
10.13. Redacción del Plan de Pruebas	4	Memoria; Planificación
10.7 Trabajo y Resultados: Descripción del sistema de anotaciones de RETORCH	4	Memoria
02.4 Modificación del Clasificador para que tome como entrada los ficheros de configuración nuevos	5	Implementación y Pruebas
05.4. Refinamiento y mejora de la validación del plan de Pruebas	5	Diseño
07.3 Refinamiento de los módulos implementados en 07.1-2, mejora en la forma de realizar los *TGroups*	5	Implementación y Pruebas
07.4 Mejora en el manejo de errores del Agrupador tras añadir los últimos casos de prueba y adaptación de las nuevas estructuras debidas a las entradas incorporadas	5	Implementación y Pruebas
08.4 Refinamiento del módulo desarrollado mejorando el algoritmo sencillo de ordenación seleccionando los *TJobs* atendiendo al coste de elasticidad	5	Implementación y Pruebas
08.5. Mejora y solución de problemas con el Organizador tras los resultados de la ejecución de las pruebas, para tener organizaciones deterministas con independencia de la máquina	5	Implementación y Pruebas
09.1. Implementación de un módulo que, dados unos casos de prueba de un *TJob*, sea capaz de generar el código de ejecución de las pruebas	5	Implementación y Pruebas
09.2. Implementación de un módulo que dados unos recursos de un *TJob* sea capaz de generar el código para su despliegue	5	Implementación y Pruebas
09.3. Implementación de un módulo, que dada una actividad sea capaz de generar el código de despliegue de su *TJob*	5	Implementación y Pruebas

09.4. Implementación de un módulo que, dadas una serie de actividades ordenadas, sea capaz de generar su código de despliegue en un entorno de CI con el Orquestador genérico (*Jenkins*)	5	Implementación y Pruebas
10.10. Descripción de como se ha implementado	5	Memoria
10.11. Descripción de la metodología	5	Memoria; Planificación
10.12. Descripción de como se ha dividido el proyecto en incrementos y de que tareas estarían compuestos dichos incrementos	5	Memoria; Planificación
11.0. Realización de la presentación	5	Presentación
11.0.1 Mejora y cambios de la presentación	5	Presentación
11.1. Realizar una primera presentación ante el grupo	5	Presentación
11.2 Realizar una segunda presentación ante el grupo	5	Presentación
02.5 Modificación del Clasificador para que, de un sistema determinista, en el que no se altere el orden de los casos de prueba y recursos dependiendo de la ejecución	6	Implementación y Pruebas
08.6 Modificación del Clasificador para que solucionar errores reportados por la validación (ejecución determinista para dar el sistema con el mismo orden)	6	Implementación y Pruebas
08.7 Adaptación del Organizador para dar soporte a un mayor número de configuraciones en los *docker-compose* de los recursos	6	Implementación y Pruebas
09.5 Adaptación y corrección del *Jenkinsfile* al entorno de integración continua	6	Implementación y Pruebas
09.6. Implementación de un módulo que genere los *docker-compose* para cada uno de los TJobs que se quieran ejecutar	6	Implementación y Pruebas
10.14. Ejecución de RETORCH y análisis de los resultados	6	Implementación y Pruebas; Memoria
10.15 Descripción del empaquetado de RETORCH	6	Memoria
10.16 Redacción de las conclusiones	6	Memoria
10.7. Validación del enfoque: Descripción del sujeto de experimentación, la infraestructura y los casos de prueba empleados	6	Memoria
10.9. Validación del enfoque: Validación de RETORCH y Resultados	6	Memoria
11.0.2 Mejora y cambios de la presentación	6	Presentación

11.3. Realizar una tercera presentación ante el grupo	6	Presentación
12.1. Configurar el entorno de virtualización	6	Implementación y Pruebas
12.2 Configuración de la infraestructura de contenedores	6	Implementación y Pruebas

Aunque la fecha de finalización del proyecto estaba marcada para el 5 de junio, debido a varios acontecimientos, retrasos en el desarrollo y refinamiento de los conceptos, el proyecto se alargó un mes más, acabando el mismo el 13 de julio de 2020.

6. Trabajo realizado y resultados obtenidos

En el presente trabajo de fin de máster se ha realizado un primer prototipo de RETORCH, que, mediante una identificación de los recursos con anotaciones provistas por el *Tester*, agrupa los casos de prueba con uso compatible de recursos y los despliega en instancias con el objetivo de lograr una ejecución eficiente de esos casos de prueba.

El trabajo realizado y los resultados obtenidos se presentan estructurados en las siguientes secciones. En las Secciones 6.1 y 6.2 se presentan el modelado de datos y procesos de RETORCH, en la Sección 6.3 se presenta la arquitectura, la Sección 6.4 presenta el análisis de las diferentes alternativas barajadas para realizar la implementación del sistema de etiquetado, describiéndose en la Sección 6.5 como se ha abordado el mismo. Finalmente, en las secciones 6.6 y 6.7 se tienen la descripción de la implementación, así como el empaquetado y uso de RETORCH.

6.1 Modelado de procesos

RETORCH está formado por cuatro procesos diferentes: **Identificación de Recursos *(Resource Identification)***, **Agrupamiento** (***Grouping***), **Ordenación** (***Scheduling***) y **Despliegue** (***Deployment***) los cuales tienen como objetivo optimizar el uso de recursos durante las pruebas de sistema. Estos procesos toman como entrada los casos de prueba,

caracterizan los recursos que se emplean, para posteriormente agruparlos para su ejecución o desplegarlos y ejecutarlos con el objetivo de optimizar tiempo/coste empleado. Estos procesos están representados en la Ilustración 2 y se describen a continuación:

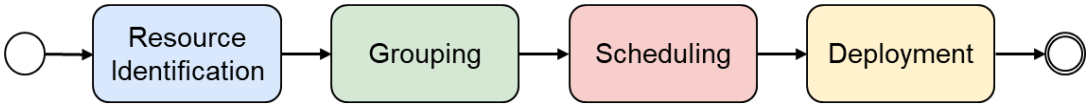

Ilustración 2 Modelado de Procesos

El proceso de **Identificación de Recursos** toma como entrada el conjunto de ***casos de prueba*** a optimizar y unos ficheros de configuración, encargándose de proveer información sobre cómo los recursos son requeridos por los casos de prueba y sus dependencias. Este proceso da como salida un ***modelo de datos (Resource Identification Model (RI Model))*** que permite realizar los siguientes procesos.

El proceso de **agrupación** toma el modelo dado por la Identificación de Recursos y agrega aquellos casos de prueba que pueden ser desplegados juntos para reutilizar los recursos evitando la infrautilización o el redespliegue innecesario de los mismos. Estos grupos de casos de prueba se denominan ***TGroups*** y contienen aquellos casos de prueba que pueden desplegarse sin interferirse durante su ejecución. El proceso de agrupación hace todas las posibles agrupaciones de recursos/casos de prueba para que luego en posteriores procesos se elija la mejor combinación de recursos y modos de acceso.

El proceso de **ordenación** genera un plan de ejecución que indica: (1) que casos de prueba se van a ejecutar juntos y (2) el orden de ejecución. Dados todas las posibles agrupaciones de recursos/casos de prueba, el proceso de ordenación genera subconjuntos de dichos TGroups denominados ***TJobs,*** que contienen el emplazamiento definitivo de los casos de prueba junto con los recursos necesarios para su ejecución. Cada caso de prueba que anteriormente podía estar en varios TGroups pasa a un único TJob con los recursos concretos que empleará durante la fase de pruebas. Estos ***TJobs*** son ordenados dando lugar a la salida: un ***plan de ejecución***, con objeto de optimizar recursos/tiempo empleado en las pruebas.

Por último, el proceso de **despliegue** ejecuta el plan de ejecución en el **entorno de integración continua** mediante una o varias **instancias** con objeto de lograr la ejecución

óptima del conjunto de pruebas. Para ello genera un Jenkinsfile, junto con los archivos docker-compose que necesita dicho script, con los que la tecnología de contenerización instancia los recursos requeridos para la prueba de forma aislada en contenedores.

6.2 Modelado de datos de RETORCH

Se presenta a continuación el modelo conceptual de RETORCH, refinado y extendido desde que fuera presentado en el artículo [6].

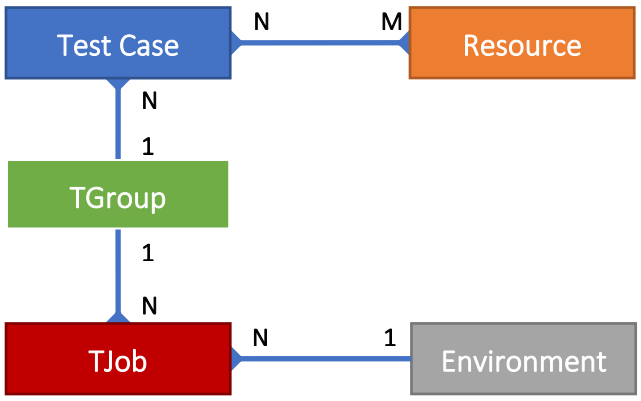

Ilustración 3 Modelado inicial del artículo

Representadas en dicho modelo se tienen las entidades con las siguientes relaciones: un caso de prueba (*Test Case*) puede usar varios recursos, al igual que un recurso puede ser accedido por varios casos de prueba. Un ***TGroup*** contiene varios casos de prueba, y este mismo ***TGroup*** se divide en varios ***TJobs*** que se despliegan en un mismo **entorno** (*Environment)***.*** Este modelo ha evolucionado y se ha refinado en los modelos que se presentan en las siguientes subsecciones.

6.2.1 Visión general

En la Ilustración 4 se representan las entidades y relaciones que componen el modelo de datos en RETORCH. Para cada entidad se representa con un color diferente el proceso donde son usadas, y se detallan en los siguientes apartados: (1) proceso de Identificación de Recursos (**Resource Identification**), (2) proceso de Agrupación (**Grouping**), (3) el proceso

de Ordenación (**Scheduling**) y (4) el proceso de Despliegue (**Deployment**).

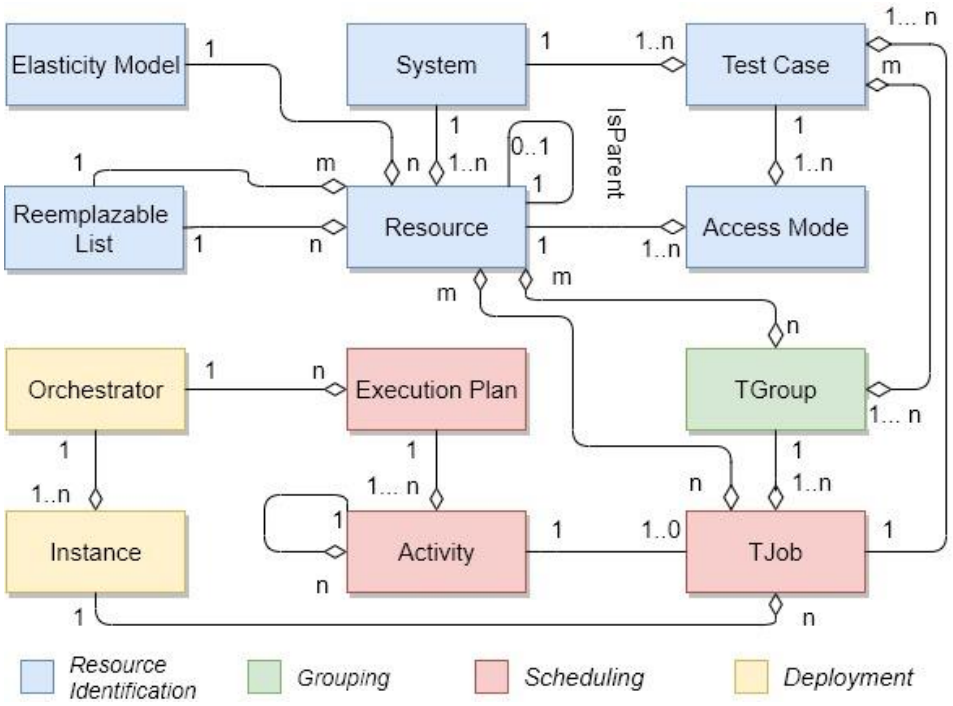

Ilustración 4 Visión General

6.2.2 **Identificación de recursos**

Un **recurso** es una entidad física, lógica o computacional que es requerida para la ejecución de uno o más casos de prueba (ej. una base de datos, una serie de sensores hardware, una serie de máquinas virtuales, un servidor web o una API externa como pudiera ser una pasarela de pagos). Los recursos son clasificados atendiendo a una serie de **categorías**, **modos de acceso** y unos **atributos**, que determina las particularidades que lo distinguen de los demás recursos.

Representado en la Ilustración 5 se tiene el modelado de entidades de la Identificación de Recursos. En él se puede observar como la **categoría** de los recursos se han modelado como un atributo (*Category*) que pueden ser: (1) físicos si son recursos tangibles (ej. un sensor o un teléfono móvil) (2) lógicos si son recursos no tangibles provistos de la forma "tradicional" (ej. una máquina virtual) o (3) computacionales si son recursos lógicos provistos o consumidos como un servicio (ej. una instancia en el *Cloud*, con su tiempo de cómputo, transferencias entre zonas, etc.)

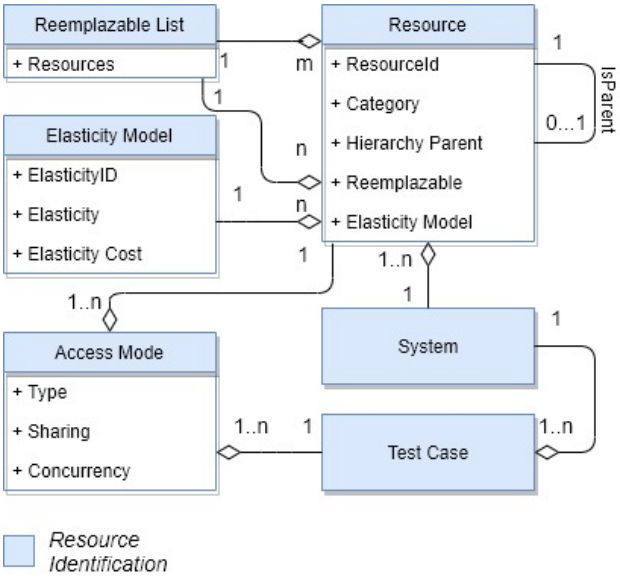

Ilustración 5 Modelado de datos de la Identificación de Recursos

Los **recursos** están caracterizados por una serie de *atributos estáticos* que no varían durante la ejecución, entre los que cuentan la **elasticidad (elasticity)**, la **jerarquía (Hierarchy Parent)**, **reemplazable** y la **compartición (sharing)**.

Un recurso es *elástico* si se puede instanciar y estar listo para su uso de forma inmediata (ej. una base de datos que se ejecuta en un contenedor o un simulador software). Por lo contrario, es *inelástico* si existe una cantidad fija de recursos (ej. una cantidad de sensores o un prototipo hardware). Para representar la **elasticidad** de los recursos, tenemos un **modelo de elasticidad**, el cual poseen todos y puede ser compartido por varios casos de prueba.

La **jerarquía** en los recursos representa las relaciones de herencia (recursos-subrecursos) que existen y para ello se ha incorporado un atributo *(hierarchy parent)*. Una jerarquía establece una orden o criterio de subordinación entre dos o más recursos, siendo el padre el recurso que tiene toda la funcionalidad y el hijo el que la tiene esa funcionalidad reducida. Por ejemplo, imaginemos una aplicación con dos funcionalidades, validar entradas y comprarlas. Si dividimos esa aplicación en dos subrecursos: recurso de validación y recurso de compra, ambos tendrán como padre jerárquico la propia aplicación.

El atributo **reemplazable** supone que un recurso puede ser intercambiado por otro durante la fase de pruebas sin perjuicio a la ejecución de estas. Para modelar dicho atributo se incorpora un atributo *(Reemplazable List)* que contiene las relaciones con los recursos que pueden reemplazar al mismo.

El atributo **compartición** representa la posibilidad que dicho recurso sea accedido de manera simultánea por dos o más casos de prueba sin que la ejecución de uno interfiera en el otro. La **compartición** se modela mediante los dos atributos (*Sharing* y *Concurrency*) de la entidad **modo de acceso** representando si permite y cuál es el paralelismo máximo soportado respectivamente.

Los recursos también son caracterizados según el uso que hacen de ellos los casos de prueba durante su ejecución. Para ello tienen gran importancia dos propiedades que caracterizan a las operaciones recurso-caso de prueba: la *seguridad* y la *idempotencia*. Una operación es segura si su ejecución no modifica el recurso (ej. una operación *JOIN* o *SELECT* en una base de datos), evitando que pueda interferir en otros casos de prueba. Una operación idempotente es aquella que puede ser ejecutada varias veces produciendo idéntico resultado.

Dependiendo de la seguridad-idempotencia de los patrones de uso, cada par caso de prueba-recurso es asociado a un **tipo** de **modo de acceso**, albergado en el atributo con el mismo nombre (*Type*) de dicha entidad tomando uno de los siguientes valores:

1. **Solo-Lectura:** las operaciones realizadas por el caso de prueba son seguras e idempotentes, este modo de acceso permite que varios casos usen el recurso al mismo tiempo.
2. **Lectura-Escritura**: el caso de prueba realiza operaciones que ni son seguras ni son idempotentes, por lo que otros casos no pueden usar el mismo recurso sin sufrir efectos colaterales.
3. **Solo-Escritura:** El caso de prueba realiza operaciones que no son seguras ni idempotentes, similares al anterior, pero permite la escritura concurrente de recursos restringiendo las lecturas a asertos de comprobación del estado esperado.
4. **Dinámico:** El caso de prueba realiza únicamente operaciones seguras, pero no idempotentes, pero como el recurso es particionado de forma inmediata, permite que

cada caso de prueba cree y acceda a una partición de forma independiente sin interferirse.

5. **Sin-acceso:** Este modo de acceso es banalmente seguro, ya que las operaciones que hace el caso de prueba no usan el recurso.

Durante la ejecución, existen una serie de atributos que caracterizan al recurso y cambian dinámicamente dependiendo de cómo se use. El *atributo dinámico* principal para el proceso de Identificación de Recursos se encuentra relacionado con el Modelo de Elasticidad. Este atributo es el **coste de elasticidad** y representa los gastos o expensas que supone el despliegue, ejecución de los casos de prueba sobre y liberación del recurso. El **coste de elasticidad** es una combinación de dinero, tiempo, poder de procesamiento, memoria, energía entre otros y para representarlo se tiene un atributo (*Elasticity Cost*) que alberga el coste numérico del mismo.

Finalmente se presentan las entidades del **caso de prueba**, que acceden a uno o varios recursos con sus respectivos modos de acceso modo de acceso y el **sistema,** el cual está compuesto por una serie de recursos y casos de pruebas con sus modos de acceso.

6.2.3 **Agrupamiento.**

El **Agrupamiento** (representado en la Ilustración 6), sería el encargado de agrupar los casos de prueba con uso de recursos compatibles tendría en su modelo como entidad principal el *TGroup*. Un *TGroup* es un conjunto de casos de prueba con sus recursos y modos de acceso que pueden ser ejecutados en un mismo entorno, ya que las operaciones que realizan sobre los recursos no interfieren entre sí. Los *TGroups* representan una combinación factible de caso-recurso-modo de acceso, cuyo recurso puede ser bien el que solicita el caso de prueba o bien uno de los que sus recursos reemplazables. Es por ello por lo que un mismo caso de prueba puede estar en uno o más *TGroups* diferentes si su modo de acceso fuese compatible con todos ellos.

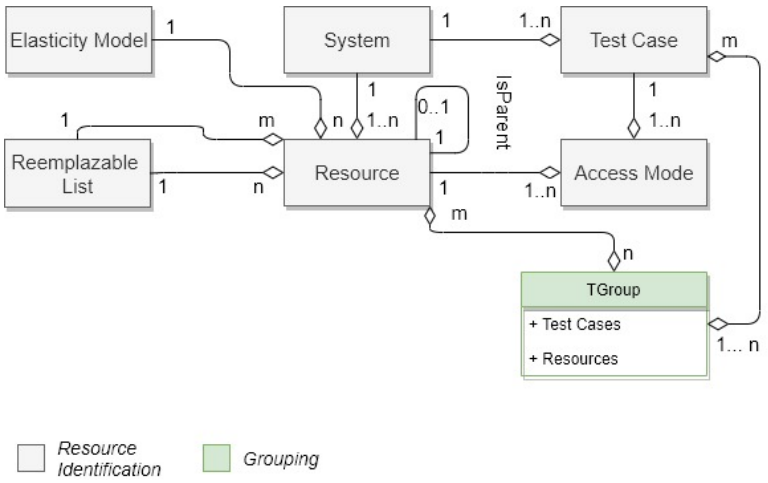

Ilustración 6 Modelo Conceptual del proceso de Agrupación

La salida del proceso de agrupamiento será el conjunto de *TGroups*, cada uno de ellos con sus recursos característicos.

6.2.4 **Ordenación**

El modelado del proceso de **ordenación** (Representado en la Ilustración 7) parte de los ***TGroups*** dados como salida en el proceso de Agrupamiento. Los *TGroups* representan todas

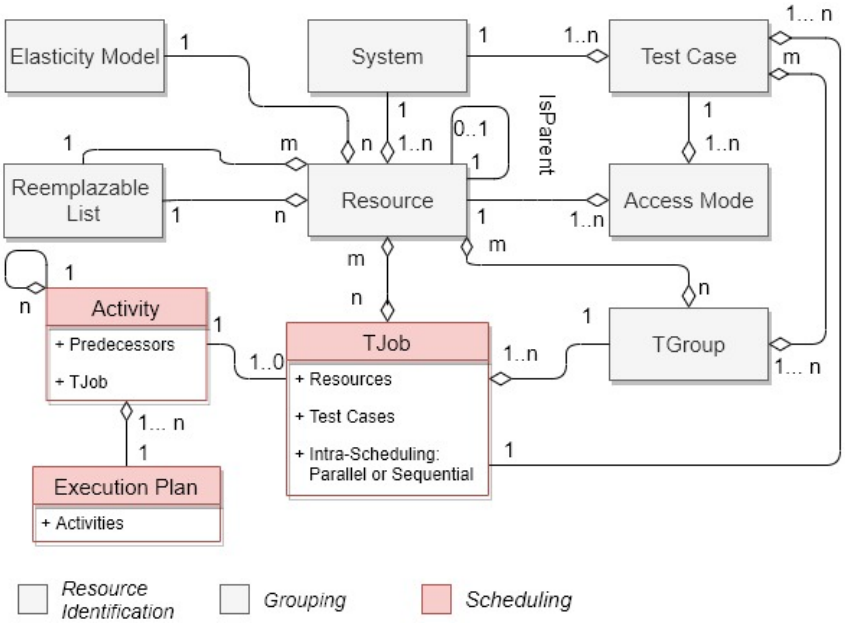

Ilustración 7 Modelo Conceptual del proceso de Ordenación

las posibles agrupaciones de casos de prueba con recursos, por lo que un mismo caso de prueba está incluido en uno o varios *TGroups*.

Como el objetivo es ejecutar la prueba una única vez, se crean otros conjuntos de casos de prueba llamados *TJobs*, en los cuales un único caso de prueba es asignado a un solo *TJob*, asegurándose que se ejecute una única vez con unos recursos determinados. Los *TJobs* son las mejores agrupaciones posibles de casos de prueba y recursos.

Un ***TJob*** está formado por un *TGroup* o un subconjunto de este y contiene de forma contenerizada varios casos de prueba cuyo uso de recursos es compatible, así como los recursos concretos que necesitarán para su ejecución. Dentro de un mismo *TJob*, los casos de prueba pueden ejecutarse de distinta forma, mediante un **intra-ordenamiento** que puede ser de dos tipos: paralelo o secuencial.

Cada uno de los *TJobs* tiene una única **actividad** asociada. Una actividad representa la ejecución un *TJob* con sus recursos y casos de prueba. Las actividades tienen una relación de ***precedencia*** con otras actividades que deben de ser ejecutadas antes. Esa relación de precedencia marca que actividades son consideradas **iniciales** en el plan de ejecución al no estar precedidas por ninguna otra, y **finales** cuando no son predecesoras de ninguna otra. Además de todo esto, una actividad puede no poseer *TJob* asociado siendo la entrada (*Gateway*) de otras actividades.

El conjunto de actividades con sus relaciones de precedencia da lugar al denominado **plan de ejecución** con la secuencia de todos los ordenamientos anteriormente descritos.

6.2.5 **Despliegue**

El modelo de datos del proceso de **Despliegue** (representado en la Ilustración 8), está basado en dos entidades: el Orquestador y las instancias. El Orquestador es el encargado de dado un plan de ejecución generado en el proceso de ordenamiento, interpretarlo y direccionar los recursos necesarios en el entorno para su ejecución.

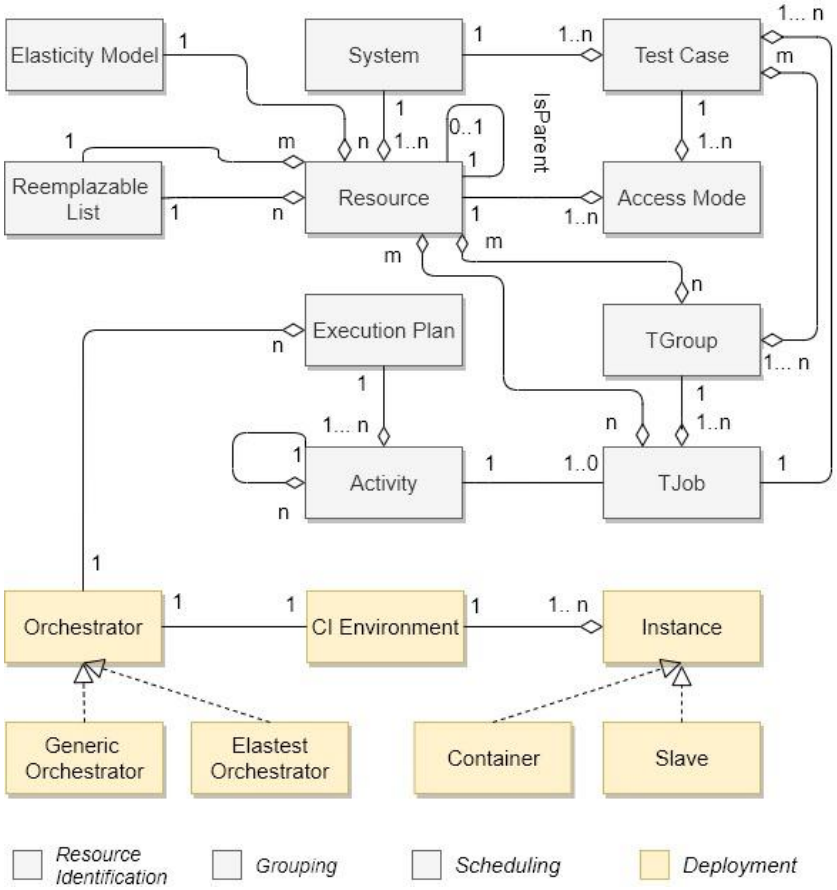

Ilustración 8 Modelo Conceptual del proceso de Despliegue

El Orquestador puede ser de dos tipos, bien es un **Orquestador genérico** basado en los pipelines de *Jenkins***,** o bien se integra con la **tecnología de orquestación de *ElasTest***. Ambos Orquestadores son los encargados de ordenar construir el artefacto software, desplegarlo y ejecutar las pruebas sobre el mismo. Para ello se sirven de una o varias instancias en las cuales se despliegan uno o varios *TJobs* junto con sus recursos. Estas **instancias** son **máquinas "esclavo"** o bien una serie de **contenedores** que ejecutan cada uno de los *TJobs* que se les asigne con el plan de ejecución.

6.3 Arquitectura del sistema

La arquitectura del sistema (representada en la Ilustración 9) viene definida por cuatro componentes, todos ellos ligados a los diferentes procesos de RETORCH.

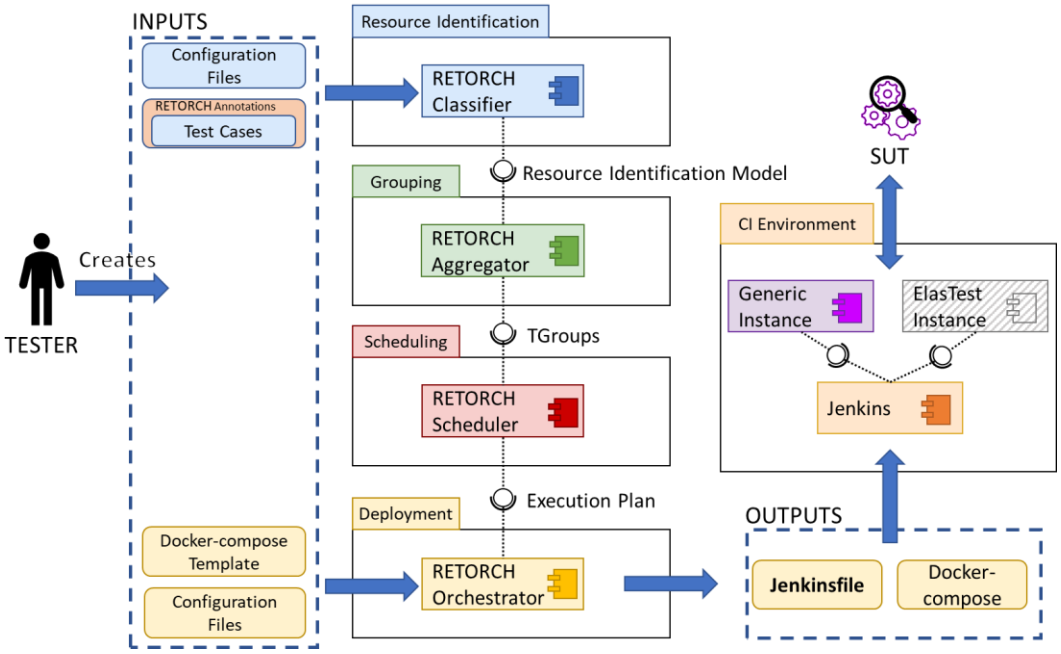

Ilustración 9 Modelo de la arquitectura del sistema

En el proceso de **Identificación de Recursos,** tenemos un componente denominado **etiquetador (*Classifier*)**, el cual permite identificar los recursos de cada caso de prueba. Atendiendo a los recursos identificados y a las incompatibilidades en el modo en el que los casos de prueba acceden a los mismos, varios de esos casos de prueba podrán ser ejecutados juntos mientras que otros no.

Las **entradas** de RETORCH son cinco: los casos de prueba, las anotaciones, la configuración de los recursos, los ficheros de configuración de *Docker* y la plantilla para la creación de los *docker-compose*. El *Tester* realiza la identificación de los recursos empleando un sistema de **anotaciones personalizadas** (*RETORCH Annotations*), con las que apunta en cada caso de prueba los recursos que necesita. El objetivo de esta identificación será encontrar aquellos casos de prueba que emplean los recursos de forma similar. El *Tester* proporciona como entrada unos ficheros de configuración en formato *JSON*, en los que especifica los diferentes atributos de los recursos y la configuración de los

docker-compose junto con su plantilla. La plantilla junto con la información provista en los ficheros se empleará para generar unos ficheros *docker-compose*, que contendrán la información de los *SUT (Software Under Test)* a desplegar en *Docker*.

En el proceso de **Agrupación** (*Grouping*), se encuentra el segundo componente, denominado **Agrupador** (***Aggregator***). El Agrupador se encarga de asociar los casos de prueba en *TGroups*, atendiendo a las (in)compatibilidades de ejecución que tengan. Los casos de prueba que no pertenezcan al mismo *TGroup* no pueden ser ejecutados juntos, ya que los recursos que emplean son diferentes.

En el tercer proceso, **Ordenación *(Scheduling),*** está compuesto por un **Organizador** (*Scheduler*) que se encarga de disponer la ejecución de los casos de prueba de forma que se optimice el tiempo, coste y/o recursos que necesitan para su ejecución. Para ello los *TGroups* son divididos en conjuntos de casos de prueba disjuntos más pequeños denominados *TJobs*. Cada uno de los *TJob*, contiene además de los casos de prueba el entorno con las dependencias aisladas en un contenedor, que facilita el despliegue de los casos de prueba en las instancias en el *Cloud*. En este componente se emplea una estrategia/algoritmo de ordenación que de una manera inteligente ordena la ejecución de los casos de prueba de una forma óptima. El **Organizador** da como salida un plan de ejecución que es la entrada del proceso de Despliegue.

El proceso de **Despliegue** está formado por el último componente, denominado **Orquestador** (***Orchestrator***). El Orquestador emplea una o varias instancias (*Instances*) en las cuales se despliegan los *TJobs*: los casos de prueba a ejecutar y los recursos que estos necesitan según se disponga en el plan de ejecución y las instancias que tenga disponibles. La salida para que se ejecute en *Jenkins*, es el *Jenkinsfile* y los ficheros *docker-compose* con los recursos.

En el presente trabajo de fin de máster se desarrolla una prueba de concepto de RETORCH, que tiene como objeto ser integrada en un entorno de pruebas continuas como pudiera ser *Jenkins* o *ElasTest* [8] empleando las librerías / componentes que esta plataforma pudiera proveer.

6.4 Mecanismos de anotación

A la hora de crear la información provista por el modelo de Identificación de Recursos, se requiere de algún mecanismo que permita incorporar junto al código de pruebas información adicional sobre el uso de recursos. La forma más habitual de proveer información en el código es el empleo de anotaciones personalizadas y comentarios. Por definición las anotaciones personalizadas permiten proveer metadatos a nivel de código, mientras los comentarios *Javadoc* se plantearon en un principio como una forma de documentar el código [34], aunque actualmente se tengan herramientas que permitan sacar la información que hay en los mismos. A continuación, se va a presentar dos alternativas diferentes de anotar/etiquetar la información sobre el propio código.

6.4.1 Anotaciones en *Javadoc*

Las anotaciones se introducen en el código encima de los métodos-clases con los caracteres /** */ pudiendo obtenerse utilizando *parseadores* de código como *JavaParser*. Este tipo de elementos, permiten procesar fuentes de texto con el código y extraer los comentarios de estos (así como otra información extra como pudiera ser el nombre de la clase o método, o el tipo como campo, interfaz o clase entre otros).

Como principal contrapartida tendríamos que el comentario no se puede extraer de forma estructurada, es decir, aprovechándose de los elementos propios del *Javadoc* como pudieran ser las anotaciones personalizadas *@author* o *@return*, con este método no podríamos sacar la información de estas a no ser que se realizase un post-procesado por el usuario.

6.4.2 Creación de anotaciones personalizadas

Las anotaciones personalizadas son típicamente utilizadas para especificar instrucciones de compilación o instrucciones de ejecución. Están disponibles desde la versión 5 del *JDK*, dando una alternativa al uso de descriptores *XML* e interfaces de marcadores. Pueden añadirse a paquetes, clases interfaces, métodos y atributos, no alterando la ejecución del programa por sí mismas. Se crean como cualquier otra clase, con la peculiaridad de que el tipo será *"@interface*". Junto con ellas se especifican el alcance (tiempo de ejecución,

compilación) y el objetivo (método, atributo, clase…) empleando meta-anotaciones (ej. *@Retention, @Target*).

Las anotaciones personalizadas pueden tener parámetros, los cuales son especificados entre paréntesis, junto con una asignación y su nombre. Estos parámetros pueden ser obtenidos de la propia anotación, en los *scopes* que se hayan especificado anteriormente.

```
@Override
@SuppressWarnings
public void methodExample () {}
```

Código 1 Ejemplo de anotacción

El procesado de las anotaciones personalizadas se realiza con la API *Reflection* de *Java*, comprobando si estas existen (es nulo el atributo), para después mediante la clase acceder a dicha anotación y obtener sus parámetros. En *Java*, se proveen tres anotaciones ya predefinidas: la anotación de sobreescritura *(@Override*), de supresión de advertencias (*@suppressWarnings*) y la anotación de obsoleta (*@deprecated*)

6.4.3 **Análisis de viabilidad de ambas alternativas.**

Ambas alternativas pueden ser empleadas para especificar información extra sobre los métodos/clases. La principal diferencia radica en que las anotaciones personalizadas van dentro del propio .class de *Java*, pudiendo ser obtenidos sus valores en tiempo de ejecución/compilación. Las anotaciones *JavaDoc*, pueden ser obtenidas de forma estática (analizando los ficheros de código fuente) pero al omitirse junto con los comentarios al generar el bytecode, no es posible obtenerlas en tiempo de ejecución. Además, las anotaciones personalizadas permiten al desarrollador valerse de las utilidades de autocompletado de los diferentes IDE, acelerando la curva de aprendizaje de estas y reduciendo la posibilidad de introducir errores. Finalmente, se debe destacar que ambos tipos de anotaciones tienen como nexo común la etiquetas *@documented*, la cual permite que dichas anotaciones personalizadas, autogeneren los *Javadoc* correspondientes (evitando que el usuario tenga que documentar dos veces el código)

6.5 Modelado de Identificación de Recursos vía anotaciones y etiquetas

El proceso de Identificación de Recursos se realiza con una serie de anotaciones de forma manual por el *Tester* que desarrolla los casos de prueba. Este, en cada uno de los métodos de prueba deberá incorporar dichas anotaciones que dependerán en formato del lenguaje adoptado, pero no en su semántica. En este documento se empleará el lenguaje orientado a objetos *Java* al tratarse del lenguaje empleado en el demostrador elegido.

Sintaxis general

El sistema de etiquetas de RETORCH es una transposición del modelo conceptual de Identificación de Recursos representando en la Ilustración 5. Para ello emplearemos una serie de anotaciones que permitirán dados unos casos de prueba anotados correctamente, crear dicho modelo y emplearlo en las sucesivas fases de agrupamiento y ordenación. Se presentan a continuación las etiquetas, tomándose como valores de primitivas base las del lenguaje de programación *Java*. Las etiquetas pueden ser anotadas con indiferencia de orden, tanto en el caso de sus atributos como en el orden del modo de acceso/recurso

Etiqueta @Resource:

La etiqueta de recurso tendrá como objetivo definir las características de los recursos requeridos por los casos de prueba. La etiqueta (ejemplificada en el extracto Código 2 tiene los siguientes parámetros que se deberán especificar entre paréntesis:

```
@Resource (resourceID = "OpenviduMedium", type= "LOGICAL",
hierarchyParent = "OpenVidu", reemplazable = {"OpenviduHeavy"},
elasticityModel = @ElasticityModel (…))
public void testOneH () {}
```

Código 2 Ejemplo de la anotación de recursos

- *<resource> ::= <resourceID> [<type>] [<hierarchyParent>] [<reemplazable>]*
- *<resourceID> ::= <String>*
- *<type> ::= "LOGICAL" || "PHYSICAL" || "COMPUTATIONAL"*
- *<hierarchyParent> ::= <String>*
- *<reemplazable> ::= {<resourceID>,<resourceID>,<resourceID> …}.*

Etiqueta @ElasticityModel:

El objetivo de la etiqueta de modelo de elasticidad será delimitar la cantidad de recursos que los *TJobs* desplegaran durante la ejecución de los casos de prueba. Esta etiqueta (representada en el extracto Código 3) deberá ser dispuesta junto con el recurso, pasando entre paréntesis los siguientes atributos:

```
@Resource (… elasticityModel = @ElasticityModel (elasticityID =
"elastlight", elasticity = 5, elasticityCost = 3000.0) …)
public void testOneH() {}
```

Código 3 Ejemplo de la anotación de Modelo de elasticidad

- *<elasticityModel>::= <elasticityID> [<elasticity>] [<elasticitycost>]*
- *<elasticityId>::= <String>*
- *<elasticity>::= <Integer>*
- *<elasticityCost>::= <double>*

Etiqueta @AccessMode:

El objetivo de la etiqueta de modo de acceso es especificar el uso que hacen los casos de prueba de los recursos. Esta etiqueta (representada en Código 4), se le pasará entre paréntesis los siguientes atributos:

```
@AccessMode (typeOfAccessMode = "READWRITE",sharing =
false,concurrency = 1)
public void testOneH() {}
```

Código 4 Ejemplo de la anotación de los modos de acceso

- *<accessMode> ::= [<typeOfAccessMode>] [<sharing>] [<concurrency>]*
- *<typeOfAccessMode> ::= ”READONLY” || “READWRITE” || “WRITEONLY” || “DYNAMIC” || ”NOACCESS”.*
- *<sharing> ::= <boolean>.*
- *<concurrency> ::= <Integer>*

6.6 Implementación

A continuación, se realiza una descripción de como se ha abordado la implementación de la herramienta, dividiendo está en: (1) herramientas de desarrollo y tecnologías empleadas, (2) estructura del proyecto y (3) descripción de las funcionalidades implementadas más relevantes.

6.6.1 Herramientas de desarrollo, tecnologías y lenguajes empleados

En este subapartado se presentan aquellas herramientas, tecnologías y lenguajes que se han empleado para el desarrollo del trabajo fin de máster, estas son:

Herramientas y Tecnologías

Las herramientas empleadas se presentan a continuación en formato de tabla divididas en herramientas para el desarrollador y para el *Tester*, proporcionando una breve descripción de para que se ha empleado junto como sus versiones y *plugins*. Varias de estas herramientas han venido marcadas por la infraestructura desplegada, entre ellas citar *GitLab*, *Jenkins* o *SonarQube* presentes en entorno de CI del grupo. Todas las herramientas y tecnologías aquí recogidas permiten ser usadas y descargadas de forma gratuita (con alguna restricción como la explotación comercial en algún caso).

Desarrollador

Para el desarrollo de RETORCH se han empleado las siguientes herramientas:

Producto	Descripción	Versión y *plugins*
IntelliJ	Entorno empleado para el desarrollo de RETORCH	v2020.1 *Plugins*: *SonarQube*
Visual Studio Code	Entorno multiplataforma empleado para manejar los ficheros de configuración y/realizar modificaciones rápidas en el código	v1.46.1 *Plugins*: *Jenkins* y *Docker*
Maven	Gestor de paquetes empleado en el desarrollo del proyecto	v3.3

Git	Sistema de control de versiones	Repositorio del Grupo (*GitLab Comunity* v12.10.12)
Jenkins	Entorno de CI en el que se ejecutan las pruebas y se despliegan los artefactos de RETORCH	Repositorio del Grupo (v2.235.1)
SonarQube	Análisis estático del código	Repositorio del Grupo (v7.9.2)

Tester

Las herramientas que como Tester se requieren para usar RETORCH son las siguientes:

Producto	Descripción	Versión y plugins
Maven	Gestor de paquetes	v3.3
Jenkins	Entorno de CI en el que se ejecutan los casos de prueba del SUT	*ElasTest* (v1.2.0) *Plugins*: *Docker*, *ElasTest* y *Maven*
Selenoid	Provisión dinámica de navegadores	v1.10
Docker	Despliegue de instancias contenerizadas	v19.03

Lenguajes y formatos de datos

Los lenguajes y formatos empleados para acometer el desarrollo del proyecto han sido principalmente cuatro:

Lenguaje	Uso	Versión
Java	Desarrollo de RETORCH	1.8
Pipeline DSL	Especificación de los pipelines de *Jenkins*, para crear el *Job SCM* que despliega el proyecto.	-
Bash	Ejecución de comandos *Unix* en los pipelines de *Jenkins*	-
YAML	Especificación de los recursos en los *docker-compose*	-

6.6.2 Estructura del Proyecto y diagrama de clases

El Proyecto se ha desarrollado siguiendo la estructura de paquetes típica de *Maven*: el código se encuentra en la carpeta *src/java* y las pruebas en la carpeta *src/test*. El único directorio

que difiere de esta estructura es el directorio *testdata*, en el cual se tienen los datos de prueba de RETORCH. En este directorio ubicado en la carpeta */test*, se tienen los paquetes de clases con casos de prueba sintéticos que se toman como entrada en el Clasificador para las pruebas unitarias y de integración. El resto de los directorios se presentan en la Tabla 3:

Tabla 3 Directorios del directorio src/java de RETORCH

Directorio	Contenido
retorch/data/structures	Estructuras de datos empleadas en los componentes
retorch/deployment	Lógica del Orquestador
retorch/entities	Entidades empleadas por RETORCH (ej. Modo de acceso, Recurso…)
retorch/exceptions	Excepciones del sistema
retorch/grouping	Lógica del Agrupador
retorch/main	-
retorch/resourceidentification	Lógica del Clasificador de recursos
retorch/scheduling	Lógica del Organizador
retorch/testanotations	Anotaciones personalizadas de RETORCH

La estructura de directorios de la carpeta *src/test*, se explicará detalladamente en la sección de verificación, situada más adelante en el documento.

6.6.3 Integración con ElasTest

En último lugar, la implementación de RETORCH se ha dejado preparada para ser integrada con la librería de orquestación de *ElasTest*. Este punto se discutirá más adelante en las conclusiones/trabajo futuro.

6.6.4 Descripción de las funcionalidades

La descripción de las funcionalidades de RETORCH se encuentran embebidos en el propio código, en el cual se dan detalles de cómo esta implementado cada uno de los métodos y cuáles son los parámetros que toma.

6.7 Empaquetado y uso de RETORCH

RETORCH se encuentra alojado en el repositorio del Grupo de Investigación en Ingeniería del Software (*GIIS*) de la Universidad de Oviedo [35]. Este repositorio se encuentra

integrado en un sistema de integración continua, en el que se ejecutan las pruebas para probar la calidad de RETORCH, se realiza el análisis estático y se publican los artefactos necesarios para su uso.

6.7.1 Configuración de RETORCH

En este apartado se presentan todas las acciones que son necesarias para realizar la correcta configuración de RETORCH. La mayor parte de los ficheros que el *Tester* debe configurar, deben de estar en formato *JSON*, en caso contrario se especifica el formato a seguir. Los ficheros deben ubicarse en la raíz del proyecto, con excepción de los referentes a la configuración de los recursos, que se situaran en un directorio denominado *"/resourcefiles"* y la plantilla para los *docker-compose*, que se situará en la carpeta recursos de prueba (*"test/resources"*). En los siguientes subapartados (1) incorporación del repositorio, (2) configuración de los recursos (3) configuración de los *docker-compose* los cuales describen los pasos a seguir para configurar RETORCH

6.7.1.1 Configuración del modelo de objetos del proyecto (pom.xml)

Partiendo de un proyecto en el que se tienen las pruebas de sistema a optimizar, el primer paso para emplear RETORCH y sus anotaciones, es añadir la dependencia y el repositorio en el modelo de objetos del proyecto (*pom.xml*). Este repositorio y dependencia se presentan a continuación en el fragmento de Código 5.

```
<repositories>
  <repository>
    <id>giis</id>
    <url>https://in4test.lsi.uniovi.es/xlib/maven/</url>
  </repository>
</repositories>

<dependency>
  <groupId> giis </groupId>
  <artifactId> retorch </artifactId>
  <version> [1.1,0) </version>
</dependency>
```

Código 5 Repositorio y dependencia a añadir al pom.xml del proyecto

En el fragmento, se dispone del código necesario para incorporar a RETORCH en el caso de que se emplee como gestor de dependencias *Apache Maven*. Se pueden emplear otros gestores (como *Ant*, *Grandle* o *Buck*) cuya forma de especificar las dependencias y repositorios difieren en formato a la propuesta. Al modificar el modelo de objetos como se indica, RETORCH se importa automáticamente en el proyecto, mediante un *jar* junto con todas las dependencias internas que requiere para su correcto funcionamiento.

6.7.1.2 Configuración de los recursos

Una vez añadida las dependencias e identificados los recursos, se debe de crear un **fichero con información de los recursos**, en el que se representen los recursos con sus atributos. Este fichero debe de seguir el convenio de nombrado: [Nombre del sistema] + "*SystemResources.json*" (ej. Sistema *Fullteaching* → *FullteachingSystemResources.json*). El fichero se estructura en forma de diccionario en el que sus pares tienen como clave el id del recurso y como valor una estructura de atributos. En el fragmento de Código 6, se muestra el aspecto de este fichero, junto con los campos que debe de tener.

```
{
  "Database": {
    "hierarchyParent": [],
    "replaceable": [],
    "elasticityModel": {
      "elasticityID": "DatabaseElasModel",
      "elasticity": 50,
      "elasticityCost": 0.0
    },
    "resourceType": "LOGICAL",
    "resourceID": "Database"},
  "Server": {
    "hierarchyParent": [],
    "replaceable": [],
    "elasticityModel": {
      "elasticityID": "ServerElasModel",
      "elasticity": 35,
      "elasticityCost": 15.0 }, }
```

Código 6 Extracto de un fichero con los atributos de los recursos

6.7.1.3 Configuración de los docker-compose

Para realizar la configuración de los *docker-compose*, se deben de crear tres ficheros: un fichero de información de los *docker-compose*, otro con la información del repositorio y un último que contendrá la plantilla para generarlos. Un *docker-compose* es un fichero que contiene en un formato denominado *YAML* la información necesaria para que *Docker* cree el contenedor.

En la raíz del proyecto, se debe de crear un fichero "*retorch-info.json*", estructurado en forma de diccionario en el que sus pares, presentan como clave el identificador del recurso, y como valor toda la información que se necesita para generar el *docker-compose*. Esta información comprendería la *URL* en la que se despliega el recurso, rango de puertos reservados para el mismo y ruta base en la que están los casos de prueba. En el fragmento de Código 7, se muestra uno de los recursos representados en *JSON*:

```
{
  "mockElasticResource": {
    "urlDeploymentResource": "localhost",
    "dockerImageResource": [
      {
        "placeHolderId": "mockedimage",
        "imageName": "mockedimage"
      }
    ],
    "basePorts": [
      {
        "name": "mainport",
        "lowerPort": "5001",
        "upperPort": "5100"
      },
      {
        "name": "otherport",
        "upperPort": "7000",
        "lowerPort": "6800"
      }
    ],
    "testsBasePath": "fullteaching/e2e-test/no-Elastest/"
  },... }
```

Código 7 Extracto de un fichero con la información de los recursos

En el directorio raíz, se debe de crear el fichero *"retorch-repo.json"* con la información referente al repositorio: su *URL*, los credenciales y la rama en la que se está trabajando. El contenido de este fichero se muestra en el fragmento de Código 8.

```
{
  "urlGitHub": "http://gitlabInvented.con/testSuiteProject",
  "credentialsGitHub": "GitHub",
  "branchGitHub": "develop"
}
```

Código 8 Fichero retorch-repo.json

Finalmente, en la carpeta recursos de prueba, se debe de crear el fichero *"docker-compose-template.yml"*, Este fichero, contiene la plantilla base para la creación de los *docker-compose* necesarios para desplegar los recursos y sigue una sintaxis propia. Un ejemplo sencillo de este fichero se muestra en el fragmento de Código 9:

```
version: '3'
services:
  'webServer-${tjobname}':
    image: 'httpd:lastest'
    ports:
      - '${webServerPort}:80'
    environment:
      - DATABASE_PASSWD=pass
      - 'DATABASE_URL=${databaseurl}'
  'database-${tjobname}':
    image: '${databaseimage}'
    ports:
      - '${databaseport}:80'
```

Código 9 Extracto de un fichero docker-compose-template.yml

Como se puede observar, el fichero contiene diversos *placeholders* (${…}), con los que se realiza una sustitución de los valores contenidos en *retorch-info.json*, dependiendo de los recursos que requiera el caso de prueba. Estos *placeholders*, permiten que para cada uno de los *TJob* se pueda realizar un *docker-compose* personalizado con sus recursos concretos.

6.7.2 Anotación de los casos de prueba

El *Tester*, tras haber identificado los recursos y realizado el *set-up* de RETORCH, puede comenzar a anotar los casos de prueba con los modos de acceso y recursos. Las anotaciones las realiza de forma individualizada para cada caso de prueba, siguiendo las pautas marcadas en la sección 6.5, Sintaxis de las anotaciones. El *Tester* debe especificar como mínimo el id de recurso y sus recursos reemplazables, y en caso de que algún atributo sea diferente al especificado, el nuevo valor de ese atributo. Las anotaciones (fragmento de Código 10) se realizan en la parte superior del caso de prueba, junto con otras anotaciones que pudieran ya existir (ej. anotación *@Test* o *@ParameterizedTest*).

```
@AccessMode (resID = "heavyInElasRest", accessMode = "READWRITE",
concurrency = 1, sharing = false)
@AccessMode (resID = "lightElasticResource", accessMode = "READWRITE",
concurrency = 4, sharing = true)
@Resource (resID = "heavyInElasRest", elasModel = @ElasticityModel
(elasID = "elasModelHeavyInElasRest", elasCost = 50.0, elasticity =
1), parent = "parentAllInelastic", replaceable = {}, rType =
"LOGICAL")
@Resource (elasModel = @ElasticityModel (elasID =
"elasModelLightElasticResource", elasCost = 15.0, elasticity = 35),
resID = "lightElasticResource", replaceable = {}, parent = {}, rType =
"LOGICAL")
@Test
public void testOneH () {…}


@Resource (parent = {"parentAllInelastic"}, rType = "LOGICAL",
replaceable = {}, resID = "heavyInElasRest", elasModel =
@ElasticityModel (elasCost = 5.0, elasID = "elasModelHeavyInElasRest",
elasticity = 1))
@Resource (resID = "lightElasticResource", elasModel =
@ElasticityModel (elasID = "elasModelLightElasticResource", elasCost =
1.0, elasticity = 3), rType = "LOGICAL", parent = {}, replaceable= {})
@AccessMode (resID = "heavyInElasRest", accessMode = "READWRITE",
concurrency = 1, sharing = false)
@AccessMode (sharing = true, concurrency = 4, accessMode =
"READWRITE", resID = "lightElasticResource")
@Test
public void testTwoH() {…}
```

Código 10 Anotaciones de los casos de prueba

El *Tester* debe anotar todos los casos de prueba que quiera ejecutar, ya que en caso contrario

RETORCH no considerará los casos de prueba durante la Identificación de Recursos y por tanto no los ejecutará.

6.7.3 Configuración final de RETORCH

Una vez se han anotado los casos de prueba, el *Tester* debe de crear una clase en la que realice una llamada a los métodos de RETORCH. Para ello debe de crear un método *main* en el que instancie un objeto del tipo *OrchestrationGenericToolBox*, del cual se invoca su método *generateJenkinsfile* (Código 11). A este método, se le debe de pasar dos parámetros: (1) el paquete en el que están los casos de prueba o en caso de que se tengan varios paquetes, el paquete jerárquicamente superior que englobe a todos ellos y (2) el nombre del sistema que se está analizando

```
public class RetorchMain {
  public static void main (String [] args) (…){
OrchestrationGenericToolBox toolBox = new
OrchestrationGenericToolBox();
toolBox.generateJenkinsfile("some.package.of.tests","NameSystem");}}
```

Código 11 Invocación de RETORCH

6.7.4 Ejecución de RETORCH

La salida de RETORCH, como bien se ha descrito en la Arquitectura, será un fichero *Jenkinsfile* en la raíz (fragmento de Código 12) junto con los *docker-compose* de cada recurso ubicados en el subdirectorio */retorchdockercomposes*. Tanto los *docker-compose* como el *Jenkinsfile*, deberán subirse al repositorio del proyecto para poder emplearlos en el despliegue.

```
pipeline {
agent any
 tools {
    maven 'M3.3.9'}//EndTools
stages {
  stage ('Clone Repository') {
    steps {
        dir('fullteaching') {
            git branch: 'someBranch',
            credentialsId: 'GitHub',
            url:'https://github.com/githubuser/namerepo'}//endDir}//EndStep}
stage ('Time 0') {
steps {
parallel (
"TJob 2 IdResource: HeavyDatabase LightWebServer ": {
  echo 'Deploying Resources Required'
  sh 'cd base/path/retorchdockercomposes/TJobN; ls . ; docker-compose --no-
ansi -p TJobN up -d'
  echo 'Executing the test cases'
  sh ('cd path/to/tests/; ls .;mvn -Dapp.url = "https://localhost:5001" -
Dtest = testClass#tcOne test')
```

Código 12 Fragmento del JenkinsFile de salida

6.7.5 Despliegue de los casos de prueba

Una vez RETORCH genera los docker-compose de los recursos y el *Jenkinsfile*, se debe de configurar el Job para que despliegue los *TJobs* en *Jenkins*. Para ello se emplea un *Pipeline* (Ilustración 10), previo al cual el entorno debe tener: (1) Algún mecanismo para instanciar y desplegar contenedores. (2) Acceso al repositorio donde se han subido los casos de prueba anotados, *Jenkinsfile* y ficheros con los *docker-compose*.

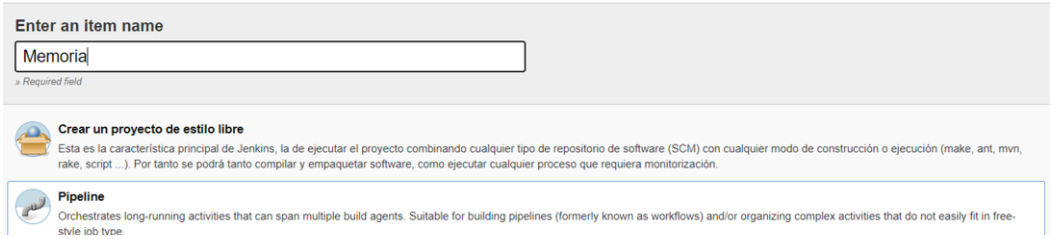

Ilustración 10 Opción en Jenkins para crear un Pipeline

El *Pipeline* se debe de configurar para que busque *Jenkinsfile* del proyecto en el repositorio (Ilustración 11). Para esto en la sección Pipeline se deberá seleccionar "*Pipeline Script de SCM*" y añadir el repositorio junto con la ruta al *Jenkinsfile*. Si el sistema de integración continua está conectado al repositorio, con cada cambio en el mismo ejecutará de forma automática los casos de prueba según se especifica en el fichero. En caso contrario, siempre se podrá ordenar que se haga la ejecución de forma manual.

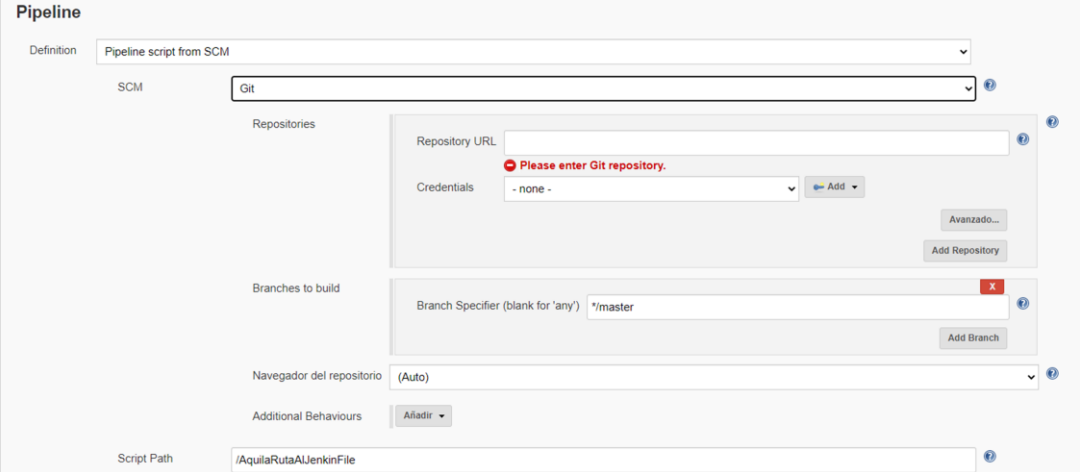

Ilustración 11 Opción en Jenkins para crear un Pipeline desde SCM

7. Plan de pruebas

Con objeto de mejorar la calidad de RETORCH, se realiza un plan de pruebas que permita verificar y validar cada una de las funcionalidades. Este plan de pruebas está compuesto de una serie de pruebas funcionales y estáticas de RETORCH, así como su automatización. Los siguientes subapartados tienen como objeto (1) hacer una descripción general del plan de pruebas, (2) presentar el diseño y (3) describir su implementación.

7.1 Descripción general

El presente plan está compuesto por una serie de pruebas de adecuación funcional y calidad, parte de las cuales, se encuentran automatizadas empleando la herramienta *JUnit*. Estas pruebas están ubicadas en el repositorio del proyecto e integradas con *Jenkins*, un entorno de integración continúa (CI) encargado de desplegarlas y ejecutarlas. El grueso de la ejecución de estas pruebas es realizado por el entorno de CI de forma automática, no

obstante, el nivel de pruebas de sistema requiere de cierta intervención manual para completar su ejecución. Enumeradas a continuación se describen las pruebas a realizar:

- Pruebas funcionales: se desea comprobar la adecuación funcional de RETORCH a los procesos modelados en el apartado 6.1. Para ello se realizarán las siguientes pruebas:
  - Pruebas unitarias de los componentes de la arquitectura, se realizará la verificación del funcionamiento de cada uno de los componentes: Clasificador, Agregador, Organizador y Orquestador. Cada uno de estos componentes toma una parte del modelo de datos y lo transforma en otra parte de dicho modelo. Las pruebas verificarán que esa transformación es correcta tomando unos objetos creados como entrada o generándolos con Java *Reflection* de un paquete o clase. Las salidas de cada componente se verifican con objetos creados o con otra serie de salidas esperadas guardadas en ficheros.
  - Pruebas de integración de los distintos componentes de la arquitectura en que se emplearán datos de entrada **sintéticos**: archivos *Java* con casos de prueba anotados, para permitir verificar los diferentes componentes de la arquitectura de forma integrada. En estas pruebas se verifica cada uno los objetos del modelo, empezando por los sistemas creados por el Clasificador y acabando por los scripts de orquestación generados por el Orquestador para ese conjunto de datos de entrada sintéticos.
  - Pruebas de sistema que constituyen la validación de RETORCH en un escenario real con un número mayor de recursos y modos de acceso. Para realizar esta validación, se emplea el demostrador de *ElasTest FullTeaching* desplegado de forma contenerizada. Como **datos de entrada**, se toma un conjunto archivos *Java* **reales,** con un recopilatorio de casos de prueba ***E2E*** sobre esa aplicación extraídos de los repositorios del proyecto. Estos casos de prueba son anotados identificando modos de acceso, recursos y modelos de elasticidad. La salida del Orquestador se traslada a un *pipeline* de *Jenkins* de forma manual, donde se ejecuta y se valida que las pruebas son desplegadas y ejecutadas sobre el sistema contenerizado.
- Pruebas estáticas del código: Se analizarán los resultados provistos por la herramienta de análisis estático *SonarQube*, así como el cumplimiento de los diferentes objetivos de calidad. Para ello se medirán indicadores como el porcentaje de duplicidades y

cobertura o la cantidad de *code-smells*, vulnerabilidades y defectos, marcándose un objetivo de "calidad" para poder desplegar el sistema.

Los scripts de prueba se almacenan en:

Tabla 4 Ubicación de los scripts

Paquete	Contenido
retorch.architecture.components	Pruebas de nivel unitario de los componentes de arquitectura
retorch.architecture.integration	Pruebas a nivel de integración de los componentes de arquitectura
retorch.testdata	Datos de entrada (archivos *Java*) para las pruebas de nivel unitario, integración y sistema
retorch.e2e	Pruebas de nivel de sistema
retorch.utils	Utilidades necesarias para crear las entidades de I/O
resources	Salidas para realizar la verificación del sistema

El diseño de pruebas se encuentra estructurado de acuerdo con los niveles de prueba descritos anteriormente, dividiéndose el nivel unitario en componentes de prueba a la que pertenecen. Para cada uno de estos niveles se realiza una identificación de las condiciones de prueba para posteriormente con ellas identificar las situaciones a cubrir obtenidas analizando las clases de equivalencia y los valores límite. En la implementación se presenta la derivación de los casos de prueba y la elaboración de scripts de prueba junto con su automatización. Los casos de prueba siguen un convenio de nombrado en el que se precede el nombre por "*test"*, seguido del nivel de prueba junto con la abreviatura de que se está probando (componente y objeto). Finalmente, se especificará "*valid*" en el caso de que ejercite una situación válida e "*invalid*" en el contrario, incorporando al final del nombre algún elemento diferenciador.

7.2 Pruebas unitarias

Para la verificación de los componentes de forma unitaria, se presenta el diseño (condiciones de entrada y situaciones a cubrir) y la implementación (derivación de casos de prueba, scripts y automatización) que se ha realizado. En cada componente, el diseño se hace con las clases de equivalencia de las entradas, para posteriormente en la implementación, seguir una estrategia de combinación mínima (*minimized approach*) con el objetivo de cubrir todas las situaciones con unas pocas clases/objetos. Los datos de prueba del Clasificador son casos de prueba contenidos en un paquete compuesto por una o varias clases, mientras que el resto de

los componentes emplean como datos de entrada objetos como listas de *TJobs*, *TGroups* o actividades. Para representar la trazabilidad entre las situaciones a cubrir y los casos de prueba se ha anotado en rojo al lado de cada una que casos de prueba la ejercitan, detallándose en el siguiente apartado. Estas anotaciones siguen el convenio de nombrado TC + Subapartado (1-Clasificador,2-Agrupador,3-Organizador y 4-Orquestador) + número de caso de prueba. En las siguientes subsecciones (1) Clasificador de recursos, (2) Agrupador (3) Organizador y finalmente (4) Orquestador, se presenta el diseño e implementación de las pruebas unitarias de los componentes:

7.2.1 Clasificador de recursos

Diseño

En las pruebas referentes al Clasificador de recursos, la entrada es un paquete de clases o clases individuales en las que se tienen casos de prueba anotados, siendo las situaciones a cubrir:

- Ubicación de los casos de prueba que toma como datos de entrada el Clasificador de recursos:
  - En una clase (TC1.1).
  - En un paquete con las diferentes clases (TC1.2).
- Cantidad de casos se encuentran anotados en los datos de entrada:
  - Varios casos (TC1.1).
  - Ningún caso (Inválida) (TC1.3).
- Existencia de anotaciones previas a las de RETORCH:
  - No hay ninguna anotación previa (Inválida) (TC1.3).
  - Hay una o varias anotaciones previas (TC1.1).
- Cómo es la sintaxis de las anotaciones:
  - Cambia el orden de las anotaciones:
    - Primero se especifican los modos de acceso y después los recursos (TC1.1).
    - Primero se especifica los recursos y después los modos de acceso (TC1.1).
    - Se especifican modos de acceso y recursos intercalados (TC1.1).
  - Se especifican más atributos en las anotaciones de recurso y modos de acceso

        - Al inicio:
            - Un atributo más (Inválida) (Irrealizable).
            - Dos atributos más (Inválida) (Irrealizable).
        - Al final
            - Un atributo más (Inválida) (Irrealizable).
            - Dos atributos más (Inválida) (Irrealizable).
        - En el medio
            - Un atributo más (Inválida) (Irrealizable).
    - Cambia el tipo de los atributos (Inválida) (TC1.4).
- Cómo son las anotaciones de recurso
    - Número de anotaciones de recurso en el caso de prueba:
        - Una anotación especificada (TC1.1).
        - Varias anotaciones especificadas (TC1.1).
    - Atributos del recurso:
        - Orden de los atributos e inclusión de parámetros nominales:
            - Identificador-Modelo-reemplazabilidad-jerarquía:
                - Con parámetros nominales (TC1.1).
                - Sin parámetros nominales (Irrealizable).
            - Jerarquía-reemplazabilidad-identificador-modelo:
                - Con parámetros nominales. (TC1.1).
                - Sin parámetros nominales (Irrealizable).
        - Identificador del recurso:
            - Se especifica una cadena de caracteres (TC1.1).
            - Se omite:
                - Recurso serializado (TC1.1).
                - Recurso no serializado en *JSON* (Inválida) (TC1.4).
        - Modelo de elasticidad:
            - Orden de los atributos e inclusión de parámetros nominales:
                - IdElasticidad-coste-elasticidad:
                    - Con parámetros (TC1.1).
                    - Sin parámetros (Irrealizable).
                - Coste-elasticidad-IdElasticidad:

        - Con parámetros (TC1.1).
        - Sin parámetros (Irrealizable).
    - Elasticidad-IdElasticidad-coste:
        - Con parámetros (TC1.1).
        - Sin parámetros (Irrealizable).
- Id Elasticidad:
    - Cadena de caracteres única (TC1.1).
    - Hay modelos con el mismo id con atributos diferentes (Inválida) (TC1.4).
- Coste de elasticidad:
    - Mayor o igual a cero (TC1.1).
    - Negativo (Inválida) (TC1.4).
    - Se omite el campo
        - Recurso serializado (TC1.1).
        - Recurso no serializado en *JSON* (Inválida) (TC1.4).
- Elasticidad:
    - Recurso elástico con un valor mayor que uno de elasticidad (TC1.1).
    - Recurso inelástico con un valor igual a uno de elasticidad (TC1.1).
    - Se omite el campo.
        - Recurso serializado (TC1.1).
        - Recurso no serializado en *JSON* (Inválida) (TC1.4).
    - Se especifica un valor menor o igual a cero (Inválida) (TC1.4).

- Recurso jerárquico:
    - Número de recursos:
        - Sin especificar:
            - Recurso serializado (TC1.1).
            - Recurso no serializado en *JSON* (TC1.4).
        - Uno especificado (TC1.1).
        - Más de uno (Inválida) (TC1.4).
    - Relación con otros recursos:

- Se especifica un id de recurso jerárquico que no existe (Inválida) (TC1.4).
- Se especifica dos recursos con idéntico identificador y distintos recursos jerárquicos (Inválida) (TC1.4).

- Se omite el campo (Inválida) (TC1.4).

- Reemplazable:
  - Número y recursos apuntados:
    - Ningún recurso especificado (TC1.1).
    - Varios recursos especificados:
      - Identificador de recurso existente (TC1.1).
      - Identificador de recurso que no existe (Inválida) (TC1.4).
      - Dos recursos con mismo identificador y distintos recursos reemplazables (TC1.1).
    - Se omite el campo (Inválida):
  - Relaciones entre el modelo de elasticidad y el recurso:
    - Se especifican dos recursos con idéntico identificador con distinto modelo de elasticidad (Inválida) (TC1.4).

- Cómo son las anotaciones de modo de acceso:
  - Número de anotaciones especificadas:
    - Una anotación (TC1.1).
    - Varias anotaciones (TC1.1).
  - Atributos:
    - Orden e inclusión de parámetros nominales:
      - IdRecurso-tipo-concurrencia-compartición:
        - Con parámetros (TC1.1).
        - Sin parámetros (Irrealizable).
      - Compartición-concurrencia-tipo-IdRecurso:
        - Con parámetros (TC1.1).
        - Sin parámetros (Irrealizable).
      - Concurrencia-compartición-IdRecurso-tipo:
        - Con parámetros (TC1.1).

        - Sin parámetros (Irrealizable).
    - Id de recurso:
        - Se proporciona un Id de recurso existente (TC1.1).
        - Se proporciona un Id de recurso que no existe (Inválida) (TC1.4).
        - Se omite (Inválida) (TC1.4).
    - Tipo de acceso
        - Incluido en la enumeración (lectura, lectura-escritura, escritura, sin acceso y dinámico) (TC1.1).
        - Otro (Inválida) (TC1.4).
        - Se omite (Inválida) (TC1.4).
    - Compartición y concurrencia:
        - Compartible:
            - Concurrencia mayor o igual a dos (TC1.1).
            - Concurrencia menor de dos (Inválida) (TC1.4).
            - Se omite la concurrencia (Inválida) (TC1.4).
        - No compartible:
            - Con una concurrencia de uno (TC1.1).
            - Con concurrencia mayor que uno (Inválida) (TC1.4).
            - Se omite la concurrencia (Inválida) (TC1.4).
        - No se especifica compartición:
            - Con una concurrencia de uno (Inválida) (TC1.4).
            - Con concurrencia mayor que uno (Inválida) (TC1.4).
            - Se omite la concurrencia (Inválida) (TC1.4).
- Relaciones entre las anotaciones de recurso y modo de acceso:
    - Dos recursos iguales con distinto modo de acceso (TC1.1).
    - Distinto número de anotaciones de modo de acceso y recurso en el caso de prueba (Inválida) (TC1.4).
    - Se especifica en el modo de acceso un identificador de recurso:
        - No existente (Inválida) (TC1.4).
        - Existente, pero no corresponde con la anotación de recurso del caso de prueba (Inválida) (TC1.4).

- Se especifica una anotación de recurso sin su modo de acceso (Inválida) (TC1.4).

Implementación

En el caso del Clasificador, la combinación ha dado lugar a cuatro casos de prueba, ubicados en el paquete *retorch.architecture.components* en la clase *TestClassifier.java*. Los casos de prueba serían los siguientes:

- TC1.1: *testUnitArchIResValidPackage*.
- TC1.2: *testUnitArchIResValidClass*.
- TC1.3: *testUnitArchIResInvalidEmptyPackage*.
- TC1.4: *testUnitArchIResInvalidClass*.

Especificando con esas abreviaturas las situaciones que ejercitan los casos de prueba en el apartado anterior (TC1.X). Los datos de entrada para estas pruebas se pueden encontrar en el paquete *retorch.testdata.architecture.aggregator*, entre los que tenemos:

- El subpaquete "*synteticpackage*"
- El subpaquete "*emptypackage*"
- La clase "*synteticClassOne.java*"
- La clase "*synteticInvalidclass.java*"

Las salidas son verificadas contra una serie de sistemas instanciados con los métodos del paquete *utils.architecture*, clase *ClassifierUtils.java*.

7.2.2 Agrupador

Diseño

La entrada del Agrupador son objetos *Java* sintéticos, con sistemas en los que se tienen las listas de recursos y casos de prueba, siendo las situaciones a cubrir:

- Cómo son las listas de recursos, modos de acceso y casos que componen el sistema:
  - Número de elementos de la lista de recursos:
    - Tienen varios elementos (TC2.1).
    - Tienen elementos repetidos (Inválida) (TC2.2).

        - No tienen elementos (Inválida) (TC2.3).
    - Número de elementos de la lista de casos de prueba
        - La lista tiene varios casos de prueba (TC2.1).
        - Tiene casos de prueba repetidos (Inválida) (TC2.2).
        - No tienen elementos (Inválida) (TC2.3).
    - Relaciones entre modos de acceso, recursos y casos de prueba:
        - Modos de acceso:
            - Dos modos de acceso diferentes tienen el mismo id de recurso:
                - Pueden ejecutarse sobre el mismo recurso (operaciones seguras e idempotentes) (TC2.1).
                - No se pueden ejecutar sobre el mismo recurso (operaciones no seguras y no idempotentes) (TC2.1).
        - Reemplazabilidad:
            - Se tienen varios recursos reemplazables (TC2.1).
            - Se tiene un recurso que no puede ser reemplazado (TC2.1).
        - Recursos requeridos por los casos de prueba:
            - Un recurso requerido por varios casos (TC2.1).
            - Se tiene un recurso requerido por un único caso de prueba (TC2.1).
        - Existen elementos sin relación (casos de prueba sin recursos, modos de acceso sin casos de prueba…) (Inválida) (TC2.2).

Implementación

La implementación se realiza con tres casos de prueba ubicados en el paquete *retorch.architecture.components* en la clase *TestAggregator*. Se sigue análogo convenio de nombrado, siendo estos:

- TC2.1: *TestUnitArchIAggValidTGroup*
- TC2.2: *TestUnitArchAggInvalidTGroup*
- TC2.3: *TestUnitArchAggInvalidEmptyTGroup*

Estando anotados al lado de las situaciones que cubren con su abreviatura (TC2.X). Las salidas son verificadas contra una serie de *TGroups*, que se instancian empleando los

métodos ubicados en el paquete *utils.architecture* clase *AggregatorUtils.java*. Con estos métodos se generan en el propio método de prueba, varias listas de objetos con varios *TGroups*, una cada caso de prueba. Esos objetos son usados para verificar la salida del Agrupador, o bien se controla que en los casos de prueba que cubren situaciones inválidas, se genere el error en el log correspondiente.

7.2.3 Organizador

Diseño

Aquí la entrada son objetos *Java* generados sintéticamente con listas de *TGroups*, compuestos por los casos de prueba y recursos que requieren, siendo las situaciones a cubrir:

- Cómo son las listas que toma por entrada el Organizador:
  - No tienen ningún elemento (Inválida) (TC3.3).
  - Tienen elementos:
    - Alguno de esos elementos esta repetido (TC3.1).
    - No tiene elementos repetidos (TC3.2).
- Lista con los *TGroups* que toma el Organizador como entrada:
  - Los recursos contenidos en los *TGroups* de las listas:
    - Elementos repetidos:
      - El *TGroup* no tiene recursos repetidos (TC3.1).
      - El *TGroup* tiene recursos repetidos (Inválida) (TC3.4).
    - Elasticidad:
      - Se tiene un recurso totalmente inelástico (elasticidad con valor uno) (TC3.1).
      - Se tienen recursos elásticos (elasticidad con valor mayor o igual a dos) (TC3.1).
    - Relaciones entre los recursos del *TGroup*:
      - Hay algún recurso con una elasticidad menor que el resto (TC3.1).
      - Alguno de los recursos no permite el acceso concurrente (TC3.1).
      - Recursos inelásticos:
        - Hay un recurso inelástico repetido en más de un *TGroup* (TC3.1).

- Un recurso inelástico es requerido por dos recursos con modos de acceso incompatible (TC3.1).

- Los casos de prueba de un *TGroup*:
  - Un caso de prueba esta repetido en varios *TGroups* (TC3.1).
  - Un caso de prueba está en un único *TGroup* (TC3.1).

Implementación

La implementación se lleva a cabo cuatro casos de prueba (dos para las situaciones válidas y dos para las inválidas). Los cuatro casos de prueba se encuentran ubicados en el paquete *retorch.architecture.components* en la clase *TestScheduler*, siguen el mismo convenio de nombrado:

- TC3.1: *testUnitArchSchValidTGroupRepeated*
- TC3.2: *testUnitArchSchValidTGroupNoRepeated*
- TC3.3: *testUnitArchSchInvalidEmptyTGroup*
- TC3.4: *testUnitArchSchInvalidDuplicatedResources*

La salida es verificada contra una serie de objetos *Java* que se generan en el propio caso de prueba. Estos objetos contienen listas con actividades, instanciadas con la clase *SchedulerUtils*, una vez más dentro del paquete *utils.architec*ture. La salida del Agrupador deberá corresponder con esos objetos, o bien aquellos casos de prueba que cubran situaciones inválidas, mostrar los mensajes en el log correspondientes.

7.2.4 Orquestador

Diseño

Aquí la entrada es la lista de actividades con *TJobs* asociados y precedencias entre ellas, las situaciones a cubrir son:

- Cómo son las listas de actividades:
  - Tienen elementos (TC4.1).
  - No tienen ningún elemento (Inválida) (TC4.3).

- Grafo de actividades
  - Cantidad y precedencia de actividades:
    - Tienen actividades precedentes (TC4.1).
    - Ninguna actividad precedente (nodo inicial) (TC4.1).
    - No es precedida por ninguna actividad (nodo final) (TC4.1).
  - Cantidad de actividades finales e iniciales:
    - Hay actividades finales e iniciales (TC4.1).
    - No hay actividades iniciales (Inválida) (TC4.4)
    - No hay actividades finales (Inválida) (TC4.4)
  - Combinación de actividades finales e iniciales:
    - Todas las actividades son iniciales y finales (ejecución en paralelo sin precedencias) (TC4.2)
    - Hay varias actividades iniciales, pero todas tienen que estar completadas para comenzar la actividad final (TC4.1).
    - Hay una actividad inicial y varias finales (TC4.1).
  - Concatenación de actividades:
    - Concatenación de actividades en paralelo con actividades en secuencial (TC4.1).
    - Concatenación de varias actividades en secuencial (TC4.1).
    - Concatenación de varias actividades en paralelo (TC4.1).
  - *TJob* asociado:
    - La actividad no tiene *TJob* asociado, actuando como *gateway* con el inicio de ejecución de otras actividades (TC4.1).
    - La actividad tiene un *TJob* asociado (TC4.1).
  - Rango de puertos del recurso para el *docker-compose*:
    - La cantidad de recursos provistos no exceden el rango de puertos reservados (TC.4.1)
    - Los recursos exceden el rango de puertos que se reserva (Inválida) (TC4.5)

Implementación

La implementación se realiza con cuatro casos de prueba (dos para las situaciones válidas y dos para las inválidas). Los casos están ubicados en el paquete

retorch.architecture.components en la clase *TestGenericOrchestrator*, siguen el mismo convenio de nombrado:

- T4.1: *TestUnitArchOrchValidComplexGraph*
- T4.2: *TestUnitArchOrchValidSimpleGraph*
- TC4.3: *TestUnitArchOrchInvalidNoActivities*
- TC4.4: *TestUnitArchOrchInvalidNoInitialAndFinalActivities*

Todos los casos de prueba válidos verifican su salida con una serie de ficheros en los que se encuentra el código de *pipelining* esperado. Estos ficheros se encuentran ubicados en el paquete *resources/expected_outputs/unitary*, siguiendo el convenio de nomenclatura de añadir "*output*" al nombre del caso de prueba:

- *outputTestUnitArchOrchValidComplexGraph*
- *outputTestUnitArchOrchValidSimpleGraph*

7.3 Pruebas de Integración

El objetivo de las pruebas de integración es verificar la cadena que forman los diferentes componentes de la arquitectura, es decir, probar la interacción entre Clasificador, Agrupador, Organizador y Orquestador. Para realizar la prueba, se proporciona un paquete con clases en las que hay casos de prueba anotados como entrada, y se comprobará a la salida de cada uno de los componentes los objetos generados.

Diseño

Las situaciones a cubrir son las siguientes:

- Los casos de prueba:
  - Ubicación en varios paquetes, clasificados según la funcionalidad del sistema que estén probando
  - Varios casos de prueba tienen anotaciones previas.
- Los recursos:
  - Los casos se encuentran anotados con varios recursos
  - Las anotaciones:

        - Tienen especificados sus atributos en el archivo de configuración.
        - Se especifica en la anotación el menor número de atributos posibles (reemplazable e id).
    - Jerarquía:
        - Se tienen recursos con padre jerárquico.
        - Se tienen recursos sin padre jerárquico.
    - Modelo de elasticidad:
        - Se tienen recursos totalmente inelásticos (no pueden ser instanciados más de una vez).
        - Se tienen recursos elásticos (posibilidad de instanciarlos una o varias veces).
    - Reemplazabilidad:
        - Se proveen de recursos reemplazables.
        - No se proveen de recursos reemplazables.
- Modos de acceso:
    - Tipo de operaciones:
        - Hacen un uso seguro e idempotente de los recursos.
        - Realizan operaciones no seguras y no idempotentes sobre los recursos.
    - Compartición:
        - Recursos compartibles con concurrencia >1.
        - Recursos no compartibles con concurrencia 1.
- Relaciones entre los casos de prueba y los recursos:
    - Respecto a cómo acceden los casos de prueba a los recursos:
        - Diferentes modos de acceso.
        - El mismo modo de acceso
    - Respecto a cuantos casos requieren un recurso:
        - Requerido por un único caso de prueba.
        - Requerido por varios casos de prueba.

Implementación

La implementación de las pruebas de integración se ha realizado en el paquete *retorch.architecture.integration*, con un único caso de prueba. Este caso de prueba se encuentra en la clase *TestIntegrationArchComp.java*, y toma como datos de prueba una serie

de ficheros *Java*, en los que se tienen casos de prueba sintéticos con las anotaciones. Estos ficheros están ubicados en el paquete *retorch\testdata\integration* y son los siguientes:

- *IntegrationClassOne.java*
- *IntegrationClassTwo.java*

Se han encadenado las salidas de los distintos componentes, verificando a la salida de estos que el sistema, los *TGroups* y las actividades sean las esperadas, así como el código del *Jenkinsfile* generado. La verificación del Orquestador vuelve a realizarse una vez más con un fichero de texto ubicado en el directorio *resources\expected_outputs\integration*, denominado *outputTestIntegration*, y las salidas de cada componente intermedio con objetos instanciados mediante los métodos ubicados en la clase *IntegrationUtils.java*, del paquete *utils.integration*.

7.4 Pruebas de Sistema

Nota: Las pruebas de sistema que se describen aquí, se presentan en un apartado aparte (Sección 8.). Esto se debe a que las pruebas de sistema en este proyecto corresponden con la validación, en la cual se aplica RETORCH a un sistema real con un número mayor de recursos.

7.5 Pruebas de calidad del código

Las pruebas de calidad de código tienen como objetivo verificar cómo de adecuado, seguro y fiable es el código fuente. Para ello se emplean herramientas de análisis estático de código como *SonarQube* en el sistema de integración continua o *SonarLint* y el propio análisis de *JetBrains* en el IDE de desarrollo. Estas herramientas presentan unos objetivos de calidad (*gates*), que marcan los criterios de aceptación.

Diseño

Los datos de prueba para las pruebas de calidad son las carpetas de código fuente del proyecto, con excepción de los directorios de recursos y *testdata*. Como salida estas pruebas dan un informe de la calidad del código en el que se analizarán: duplicidades, cobertura, *code-smells*, vulnerabilidades y defectos.

Las pruebas de análisis de calidad del código se encuentran automatizadas ejecutándose: (1) Antes de una nueva aportación al repositorio (*Push*) en el propio IDE y (2) al desplegar en el entorno de integración continua al final de la ejecución de las pruebas unitarias y de integración. Los criterios de aceptación para las pruebas de análisis estático es la no existencia de duplicidades, *code-smells*, deuda técnica, *bugs* y fallos de seguridad en el código. Se requiere también que existan al menos un 20% de comentarios en el código fuente.

7.6 Resultados

Para analizar los resultados obtenidos de la ejecución de las pruebas unitarias y de integración, nos centraremos en el informe provisto por la herramienta de análisis estático del código y en la información provista por *Jenkins*. Estos resultados se estructuran en tres apartados: (1) Ejecución de las pruebas (2) Análisis del código y (3) Análisis de las pruebas.

Ejecución de las pruebas

Se han ejecutado los 16 casos de prueba y todas han pasado con éxito. El entorno de integración continua ha ejecutado el conjunto de pruebas en 132 ocasiones de las cuales en un 40% (54 ejecuciones) las pruebas han encontrado algún defecto (datos de la rama de desarrollo). El promedio de tiempo de ejecución de las 20 últimas ejecuciones es de 43,45 segundos en ejecutar todos los casos de prueba (unitarios y de integración).

Análisis del código

La evolución de las pruebas estáticas en el sistema de integración continua se presenta en la Ilustración 12. Como se puede observar en la gráfica, la cantidad de problemas (*issues*) durante las últimas etapas del proyecto, se ha mantenido cercano a cero en cuanto a vulnerabilidades, *bugs* y *code-smells*. La primera aportación presenta con gran diferencia el peor estado, con 5 *bugs* 50 *code-smells* y 24 vulnerabilidades, teniendo repuntes durante principios y finales del mes de abril y tendiendo a la baja desde entonces.

En esta misma grafica (Ilustración 12.) se puede observar el estado del repositorio a 6 de julio de 2020

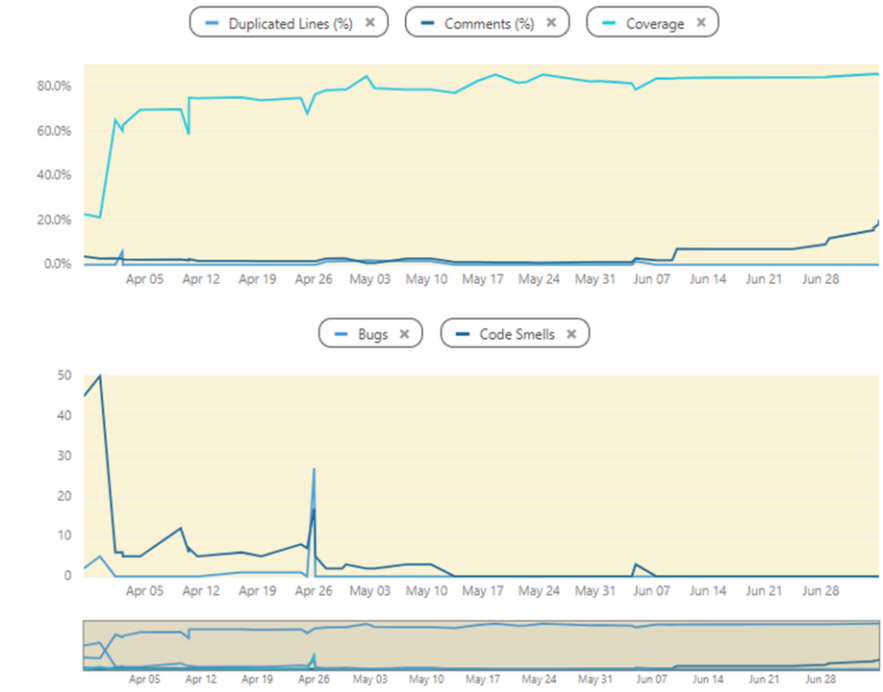

Ilustración 12 Estado actual del repositorio

Análisis de la cobertura

La cobertura, mostrada en la Ilustración 13, presenta también una trayectoria ascendente a lo largo del proyecto, comenzando con un 22% en las primeras aportaciones, hasta el actual 84% de cobertura.

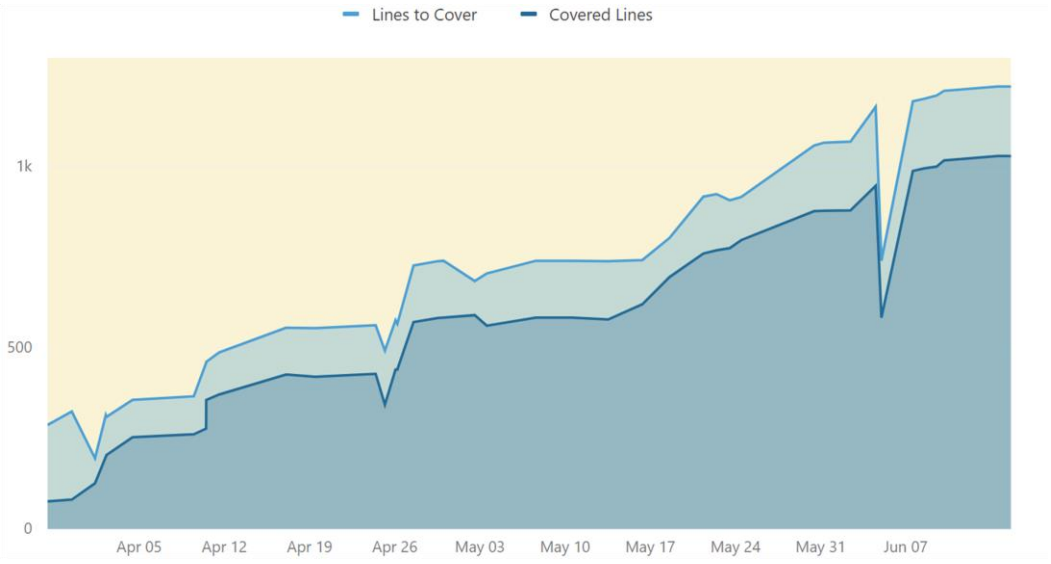

Ilustración 13 Evolución de la Cobertura

8. Validación

Para la validación de RETORCH se emplea un conjunto de pruebas de sistema sobre la aplicación *Fullteaching* en las que:

- La entrada son los casos de prueba adaptados a la última versión de *Fullteaching*.
- Se identifican los recursos, así como sus atributos (elasticidad, reemplazabilidad…).
- Se realiza la configuración de RETORCH: ficheros de configuración de los recursos y la configuración y plantilla de los *docker-compose*.
- En la ejecución de RETORCH mostrar tanto las salidas finales (*Jenkinsfile* y los *docker-compose*) como los resultados intermedios de los componentes (salida del Agrupador, Organizador, Orquestador…).

La continuación, se presenta el sistema bajo prueba empleado (Sección 8.1), los casos de prueba empleados junto con los cambios y problemáticas encontradas (Sección 8.2), el diseño (Sección 8.3), la implementación (Sección 8.4) y finalmente la infraestructura y los resultados (Secciones 8.5 y 8.6).

8.1 Sistema bajo prueba: *Fullteaching*

FullTeaching [36] es una plataforma educativa que nace cómo aplicación para validar el proyecto *ElasTest*, un proyecto europeo que busca facilitar la realización de las pruebas de sistema, también denominadas *E2E* [37]. Para ello, *ElasTest* dispone de herramientas que permiten instrumentar los sistemas bajo prueba, realizar recomendaciones, grabar la interacción del usuario mientras realiza las pruebas, monitorizar la cantidad de recursos, estimar el coste de ejecución y proveer de herramientas de análisis de logs.

El demostrador de *ElasTest Fullteaching*, provee de herramientas para gestionar e impartir la docencia de forma online con herramientas como pudieran ser: Aulas virtuales, muros de contenidos, calendarios y servicios de videoconferencia. Estos últimos son desplegados en la aplicación gracias a un servidor de *Streaming* desarrollado por la misma Universidad Rey Juan Carlos llamado *OpenVidu* [38], dando la posibilidad de realizar

videollamadas impartiendo la docencia a través de las mismas. Todos estos servicios son soportados por una capa de persistencia provista por un servidor de bases de datos (*MySQL*) y un servidor de contenidos web (*NodeJS*)

8.2 Casos de prueba empleados

Los casos de prueba empleados en este proyecto de fin de máster son todos los casos de prueba *E2E* de la aplicación de *Fullteaching* provistos por la plataforma *ElasTest* y desarrollos paralelos. Todos estos casos de prueba pueden encontrarse en los diferentes repositorios y ramas del proyecto [36], [39]. En total este conjunto cuenta con 21 pruebas de sistema, que abarcan desde que la interfaz se cargue correctamente hasta el correcto funcionamiento de los chats con su video. En la Tabla 5 se muestra el número de líneas de código de cada uno de los casos de prueba. Como se puede observar, los casos de prueba tienen cientos de líneas de código, contando con un promedio de más de 243 líneas de código.

Tabla 5 Casos de prueba con su extensión

Caso de Prueba	Extensión
studentCourseMainTest	*193*
teacherCourseMainTest	*160*
teacherCreateAndDeleteCourseTest	*192*
teacherEditCourseValues	462
teacherDeleteCourseTest	*206*
oneToOneChatInSessionChrome	*197*
courseRestOperations	161
courseInfoRestOperations	*153*
sessionRestOperations	*268*
forumRestOperations	*251*
filesRestOperations	*325*
attendersRestOperations	*199*
oneToOneVideoAudioSessionChrome	*564*
forumLoadEntriesTest	203
forumNewEntryTest	*193*
forumNewCommentTest	*252*
forumNewReply2CommentTest	*242*
spiderLoggedTest	*181*
spiderUnloggedTest	166
sessionTest	*426*
loginTest	148

8.2.1 Cambios realizados en los casos de prueba

Una vez unificados los conjuntos de pruebas en un único proyecto, se han tenido que realizar diversos cambios para adaptar todos los casos de prueba a la última versión de *Fullteaching*. Algunos de estos cambios fueron triviales como etiquetas de identificación que habían cambiado en las nuevas versiones o elementos de la interfaz que habían cambiado su posición/aspecto. Sin embargo, varios casos de prueba presentaban problemáticas que por su complejidad se ha decidido tratarlas de forma individual en la Sección 8.2.2.

Durante el desarrollo de la validación de RETORCH, se probaron también diversas alternativas con las que proveer de navegador a los casos de pruebas. Entre otras destacar la gestión automática de los navegadores web realizada por la Liberia *WebDriverManager* [40] de *Selenium Jupiter* [41]. Esta alternativa permite adquirir los drivers de forma dinámica y por tanto permitir en un entorno de integración continua adquirir el *driver* de forma automática, ejecutando las pruebas con la última versión y detectando los problemas de manera temprana. También se probaron los navegadores contenerizados de esta librería, así como las diferentes variantes y tipos de navegador (*Headless, Firefox, Chrome, Edge*..).

Debido a la naturaleza del despliegue que se realiza, finalmente se ha decidido implementar una infraestructura de navegadores basada en *Selenoid*. *Selenoid* es una implementación del *Hub* original de *Selenium* que usa *Docker* para lanzar los navegadores. Esta alternativa permite aislar los casos de prueba, cada ejecución en un navegador individual evitando los problemas que pudieran derivar de la concurrencia o las dependencias entre casos de prueba. Los contenedores de *Selenoid* se han desplegado en la máquina virtual en la cual instancian el *driver* remoto, y así puedan tomar control del navegador y ejecutar las acciones sobre la interfaz web.

8.2.2 Problemáticas encontradas

Como se ha comentado, durante la adaptación del conjunto de pruebas se han encontrado tres problemáticas, la primera se ha podido resuelto efectivamente, en la segunda se ha tenido que desactivar la prueba y en la tercera se ha dejado documentada al no poder reproducir ese comportamiento.

La primera problemática residía en el comportamiento *"flaky"* que algunos de los casos de prueba presentaban al ser ejecutados en uno de los equipos de desarrollo. Una prueba *flaky* es aquella que ante una misma configuración de ejecución puede ejecutarse correctamente unas veces y fallar en otras. En este caso la raíz del problema residía en las resoluciones de pantalla: el navegador sobre el que se hacía la prueba, aparecía ocupando la mitad del espacio disponible haciendo que con ciertas resoluciones pasase y en otras no. Uno de los ordenadores que se usaron en desarrollo, tenía una pantalla cuya resolución era un 33% mayor que los equipos del laboratorio y esa diferencia hacía que los elementos "responsive" de la interfaz apareciesen/desapareciesen. Se consiguió resolver el problema de dos formas diferentes: (1) Maximizando el navegador por defecto y (2) Configurando los navegadores contenerizados a una resolución fija.

La segunda problemática tiene como causa raíz el servidor multimedia y su función de videollamadas, la cual no funciona correctamente al lanzarlo sobre una máquina virtual. Tras la última actualización *ElasTest*, el equipo de desarrollo proporcionó la solución a parte de esos errores que se presentan al desplegar *OpenVidu* en una infraestructura virtual, pero permanece la problemática con el servidor de video, el cual no funciona correctamente (no muestra el *stream* en ambos extremos de la comunicación) incluso desplegándolo sobre un sistema operativo *Linux* instalado directamente sin virtualización. Esta problemática repercute directamente en uno de los casos de prueba (*oneToOneVideoAudioSessionChrome*), que a pesar de que se ha tenido en cuenta para identificar los recursos y en el ordenamiento se ha desactivado su ejecución documentando en el código que ocurre (además de representarlo en color gris en los gráficos). Esta problemática ha sido reportada al equipo de desarrollo [42] y se sospecha que la raíz del problema está en la tecnología que soporta el servidor la cual presenta problemas al desplegarse con *Docker* [43].

La tercera problemática se ha presentado en alguna de las ejecuciones de los casos de prueba de la aplicación. Al ejecutar los casos de prueba, en alguna ocasión el plugin de *Surefire* de *Maven* ha reportado un error en el que dispone que "No hay test que ejecutar". Este error ya ha sido reportado en el pasado [44] y se ha descartado las problemáticas referentes a la incorporación de versiones antiguas o dependencias externas del *classpath*. El error no ha podido ser reproducido en otra ocasión, por lo que se ha dejado documentada la existencia de este problema.

8.3 Diseño

El objetivo del diseño es realizar una identificación de los recursos que requieren los casos de prueba de la aplicación *FullTeaching*. *Fullteaching* está formada por tres componentes, consta de un servidor web, una base de datos y un servidor multimedia (*OpenVidu*). Analizando como los casos de prueba emplean esos componentes, se identifican los siguientes recursos:

- ***OpenVidu***: Recurso lógico totalmente inelástico, debido a que requiere una máquina dedicada en exclusiva para él. Este recurso lo emplearán los casos de prueba que ejerciten las funcionalidades de videollamada o chat de la aplicación.
- ***Mock-OpenVidu*:** Se trata de un recurso lógico que reemplaza al recurso *OpenVidu* en aquellos casos de prueba que no emplean su funcionalidad. Es un recurso elástico, que permite realizar varias instanciaciones de este y acceder concurrentemente.
- **Servicio de *Login*:** recurso lógico empleado por todos los casos de prueba del conjunto al realizar el inicio en sesión del sistema. Permite un acceso concurrente al mismo el cual vendrá limitado por el número de transacciones simultáneas al componente de base de datos.
- **Curso**: recurso lógico particionado en una serie de subrecursos maestros que usan o en los que crearan detalles los casos de prueba. Estos subrecursos permitirán el acceso concurrente o no, atendiendo al modo de acceso que se realice sobre el mismo:
    - **Configuración**: compuesto por la configuración interna del curso, como su nombre, su icono o la activación/desactivación del borrado.
    - **Información**: página principal del curso, en la que se puede añadir o eliminar contenidos de esta
    - **Sesión**: conjunto de todas las sesiones del curso con sus parámetros: fecha, nombre, descripción y enlace a la sesión de videoconferencia.
    - **Foro**: foros de discusión del propio curso, en estos foros encontramos entradas creadas por los usuarios que pueden ser comentadas, y esos comentarios a su vez ser respondidos.
    - **Archivos**: espacio de colaboración en el que se pueden subir/eliminar archivos.
    - **Participantes**: listado con los alumnos/profesores que tienen acceso al curso online

Con estos recursos, se procede a analizar el uso que los casos de prueba hacer de los mismos. Sus modos de acceso se encuentran detallados en la Tabla 6, especificando que recursos emplea cada caso de prueba, y en el caso del recurso Curso, que subrecursos son empleados como maestros y donde se crean los detalles (en el caso de que los cree).

Tabla 6 Relación casos de prueba recursos-modo de acceso

Caso de Prueba	Recurso	Maestro	Detalles	Modo de Acceso
studentCourseMainTest	*Login* Curso *MockOpenvidu*	Configuración, Información, Sesión, Foro, Archivos y Participantes	-	*Read-Only (RO)* *RO* *No-Access (NA)*
teacherCourseMainTest	*Login* Curso *MockOpenvidu*	Configuración, Información, Sesión, Foro, Archivos, Participantes	-	*RO* *RO* *NA*
teacherCreateAndDeleteCourseTest	*Login* Curso *MockOpenvidu*	Configuración	Curso: Configuración, Información, Sesión, Foro, Archivos, Participantes	*RO* *Dynamic* *NA*
teacherEditCourseValues	*Login* Curso MockOpenvidu	Configuración	Configuración	*RO* *Dynamic* *NA*
teacherDeleteCourseTest	Login Curso *MockOpenvidu*	Curso	Curso	*RO* *RW* *NA*
oneToOneChatInSessionChrome	*Login* Curso *Openvidu*	Sesión	-	*RO* *RO* *RW*
courseRestOperations	Login Curso MockOpenvidu	Configuración	Configuración	*RO* *RW* *NA*
courseInfoRestOperations	*Login* Curso *MockOpenvidu*	Información	Información	*RO* *RW* *NA*
sessionRestOperations	*Login* Curso *MockOpenvidu*	Sesión	Sesión	*RO* *RW* *NA*
forumRestOperations	*Login* Curso *MockOpenvidu*	Foro	Foro	*RO* *RW* *NA*
filesRestOperations	*Login* Curso *MockOpenvidu*	Ficheros	Ficheros	*RO* *RW* *NA*
attendersRestOperations	*Login* Curso *MockOpenvidu*	Participantes	Participantes	*RO* *RW* *NA*
oneToOneVideoAudioSessionChrome	*Login* Curso *Openvidu*	Sesión	-	*RO* *RO* *RW*
forumLoadEntriesTest	Login Curso MockOpenvidu	Foro	-	*RO* *RO* *NA*
forumNewEntryTest	*Login* Curso	Foro	Foro (entrada)	*RO* *RW*

	MockOpenvidu			*NA*
forumNewCommentTest	*Login* Curso *MockOpenvidu*	Foro	Foro (comentarios)	*Read-Write* *RW* *No-Access*
forumNewReply2CommentTest	*Login* Curso *MockOpenvidu*	Foro	Foro (entrada)	*Read-Write* *RO* *No-Access*
spiderLoggedTest	*Login* Curso *MockOpenvidu*		-	*RO* *RO* *No-Access*
spiderUnloggedTest	*Login* *MockOpenvidu*		-	*RO* *No-Access*
sessionTest	*Login* Curso *OpenVidu*	Sesión	-	*RO* *RO* *RW*
loginTest	*Login* *MockOpenvidu*		-	*RO* *NA*

Tras identificar los diferentes recursos, estos se representan en la Tabla 7 con sus atributos:

Tabla 7 Valor Atributos de los recursos

Id Recurso	Coste de Elasticidad	Elasticidad	Tipo	Padre jerárquico
Course	100	5	*"Logical"*	*"MySQL"*
LoginService	30	5	*"Logical"*	*"MySQL"*
Configuration	15	40	*"Logical"*	*"MySQL"*
Information	15	20	*"Logical"*	*"MySQL"*
Session	15	20	*"Logical"*	*"MySQL"*
Forum	15	20	*"Logical"*	*"MySQL"*
Files	15	20	*"Logical"*	*"MySQL"*
Attenders	15	20	*"Logical"*	*"MySQL"*
OpenVidu	300	1	*"Logical"*	*"MySQL"*
OpenViduMock	10	30	*"Logical"*	*"MySQL"*

El coste de elasticidad y elasticidad, se han estimado teniendo en cuenta los recursos que disponemos y las restricciones de estos.

Respecto a la elasticidad, como el servidor *OpenVidu* es un servicio externo (a pesar de que en los casos de prueba se instancie de forma contenerizada) se trata de un servicio totalmente inelástico del cual no podemos tener más de una instancia. Se ha acotado la elasticidad de los recursos que tienen como padre la base de datos, para evitar que se ejecutasen más de 5 instancias simultáneas.

Para la estimación de costes se ha decidido partir de un máximo de 300, el cual se asigna al recurso más costoso (*OpenVidu*). La diferencia de costes entre los dos servidores

multimedia (*OpenVidu* y *OpenVidu Mock*), viene dada por consumir 15 veces más memoria RAM y más del doble de almacenamiento, por lo que se ha supuesto un valor 30 veces inferior.

Como el servicio de *Login* y el Curso parten de la misma base de datos y esta consume aproximadamente un tercio de los recursos que consume el servidor web, se ha decidido repartir esos recursos entre las distintos subrecursos del mismo. Para los subrecursos del servidor de base de datos se ha asignado un coste total sume en torno al coste del recurso jerárquico padre.

8.4 Implementación

El objetivo de la implementación es tener los casos de prueba anotados, de acuerdo con los recursos identificados en el diseño, además de una configuración con los ficheros necesarios. La implementación se divide en cinco apartados (1) configuración previa, (2) anotación de los recursos, (3) Creación de los ficheros de configuración, (4) configuración final de RETORCH y (5) ejecución de RETORCH.

8.4.1 Configuración previa de los recursos y docker-compose

Para realizar las pruebas de sistema de RETORCH, es necesario representar la información de los recursos disponibles. La representación se realiza mediante un fichero *JSON* según se describe en la sección 6.7.1. El contenido del fichero *FullTeachingSystemResources.json*, con los atributos de los recursos, se presenta en el fragmento de Código 13. Cada recurso es puesto en el conjunto, empleando como clave su id, seguido de los atributos con sus valores.

```
{"Configuration":
{"hierarchyParent": ["Course"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelConfiguration", "elasticity": 40, "elasticityCost":
15.0}, "resourceType": "LOGICAL", "resourceID": "Configuration"},
 "OpenViduMock":
{"hierarchyParent": ["OpenVidu"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelOpenViduMock", "elasticity": 30, "elasticityCost":
10.0}, "resourceType": "LOGICAL", "resourceID": "OpenViduMock"},
 "OpenVidu":
{"hierarchyParent": [], "replaceable": [], "elasticityModel": {"elasticityID":
"elasModelOpenVidu", "elasticity": 1, "elasticityCost": 300.0},
"resourceType": "LOGICAL", "resourceID": "OpenVidu"},
 "Course":
{"hierarchyParent": ["MySQL"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelCourse", "elasticity": 5, "elasticityCost": 100.0},
"resourceType": "LOGICAL", "resourceID": "Course"},
 "Information":
{"hierarchyParent": ["Course"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelInformation", "elasticity": 20, "elasticityCost":
15.0}, "resourceType": "LOGICAL", "resourceID": "Information"},
 "Files":
{"hierarchyParent": ["Course"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelFiles", "elasticity": 20, "elasticityCost": 15.0},
"resourceType": "LOGICAL", "resourceID": "Files"},
 "Attenders":
{"hierarchyParent": ["Course"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelAttenders", "elasticity": 20, "elasticityCost":
15.0}, "resourceType": "LOGICAL", "resourceID": "Attenders"},
 "LoginService":
{"hierarchyParent": ["MySQL"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelLoginService", "elasticity": 5, "elasticityCost":
30.0}, "resourceType": "LOGICAL", "resourceID": "LoginService"},
 "Session":
{"hierarchyParent": ["Course"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelSession", "elasticity": 20, "elasticityCost":
15.0}, "resourceType": "LOGICAL", "resourceID": "Session"},
 "Forum":
{"hierarchyParent": ["Course"], "replaceable": [], "elasticityModel":
{"elasticityID": "elasModelForum", "elasticity": 20, "elasticityCost": 15.0},
"resourceType": "LOGICAL", "resourceID": "Forum"}
}
```


Código 13 Fichero FullTeachingSystemResources.json

Para realizar las anotaciones, se incorpora la dependencia y el repositorio de RETORCH, según se especifica en la sección 6.7.1., tras lo cual se comienzan a anotar los casos de prueba.

8.4.2 Anotaciones de los casos de prueba

Cada uno de los casos de prueba, es anotado con el identificador del recurso padre del que necesita, además de incorporar los recursos que pueden reemplazarlo. Junto con cada anotación de recurso, se tiene su anotación de modo de acceso, con la concurrencia, la posibilidad de compartición y el tipo de acceso. A continuación, se presentan las anotaciones realizadas sobre los casos de prueba, agrupadas por la clase *Java* a la que pertenecen:

```
@Resource (resID="LoginService", replaceable = {})
@AccessMode (resID="LoginService", concurrency=10, sharing=true,
accessMode="READONLY")
@Resource (resID="OpenVidu", replaceable = {})
@AccessMode (resID="OpenVidu", concurrency=10, sharing=true,
accessMode="READWRITE")
@Resource (resID="Course", replaceable={"Configuration"})
@AccessMode (resID="Course", concurrency=1, sharing=false,
accessMode="READONLY")
@Test
void oneToOneChatInSessionChrome () throws InterruptedException {…}
```

Código 14 Anotaciones de los casos de prueba FullTeachingE2EChat

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Configuration"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void courseRestOperations () throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Information"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void courseInfoRestOperations () throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Session"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void sessionRestOperations () throws InterruptedException {…}
```

Código 15 Anotaciones de los casos de prueba FullTeachingE2EREST I

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Forum"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void forumRestOperations () throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Files"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void filesRestOperations () throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource(resID = "Course", replaceable = {"Attenders"})
@AccessMode(resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void attendersRestOperations() throws InterruptedException {…}
```

Código 16 Anotaciones de los casos de prueba FullTeachingE2EREST II

```
@Resource (resID = "Course", replaceable = {})
@AccessMode (resID = "Course", concurrency = 15, sharing = true, accessMode =
"READONLY")
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@ParameterizedTest
@MethodSource("data")
public void studentCourseMainTest(String usermail, String password, String
role) throws InterruptedException {…}
```

Código 17 Anotaciones de los casos de prueba CourseStudentTest

```
@Disabled
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "READWRITE")
@Resource (resID = "Course", replaceable = {"Session"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Test
void oneToOneVideoAudioSessionChrome () throws InterruptedException {…}
```

Código 18 Anotaciones de los casos de prueba FullTeachingE2EVideoSession

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "READWRITE")
@Resource (resID = "Course", replaceable = {"Session"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READONLY")
@ParameterizedTest
@MethodSource("data")
public void sessionTest () throws InterruptedException {…}
```

Código 19 Anotaciones de los casos de prueba LoggedVideoSession

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {})
@AccessMode (resID = "Course", concurrency = 15, sharing = true, accessMode =
"READONLY")
@ParameterizedTest
@MethodSource ("data")
public void teacherCourseMainTest(String usermail, String password, String
role) throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {})
@AccessMode (resID = "Course", concurrency = 15, sharing = true, accessMode =
"DYNAMIC")
@ParameterizedTest
@MethodSource ("data")
public void teacherCreateAndDeleteCourseTest (String usermail, String
password, String role) throws InterruptedException {…}
```

Código 20 Anotaciones de los casos de prueba CourseTeacherTest I

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Configuration"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@ParameterizedTest
@MethodSource("data")
public void teacherEditCourseValues (String usermail, String password, String
role) throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@Disabled
@ParameterizedTest
@MethodSource ("data")
public void teacherDeleteCourseTest (String usermail, String password, String
role) throws InterruptedException {…}
```

Código 21 Anotaciones de los casos de prueba CourseTeacherTest II

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Forum"})
@AccessMode (resID = "Course", concurrency = 10, sharing = true, accessMode =
"READONLY")
@ParameterizedTest
@MethodSource ("data")
public void forumLoadEntriesTest (String usermail, String password, String
role) throws InterruptedException {…}

@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Forum"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@ParameterizedTest
@MethodSource ("data")
public void forumNewEntryTest (String usermail, String password, String role)
throws InterruptedException {…}
```

Código 22 Anotaciones de los casos de prueba LoggedForumTest I

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Forum"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
@ParameterizedTest
@MethodSource ("data")
public void forumNewCommentTest (String usermail, String password, String
role) throws InterruptedException {…}

@ParameterizedTest
@MethodSource ("data")
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {"Forum"})
@AccessMode (resID = "Course", concurrency = 1, sharing = false, accessMode =
"READWRITE")
public void forumNewReply2CommentTest (String usermail, String password,
String role) throws InterruptedException {…}
```

Código 23 Anotaciones de los casos de prueba LoggedForumTest II

```
@ParameterizedTest
@MethodSource ("data")
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
public void loginTest (String usermail, String password, String role) throws
InterruptedException {…}
```

Código 24 Anotaciones de los casos de prueba UserTest

```
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {})
@AccessMode (resID = "Course", concurrency = 15, sharing = true, accessMode =
"READWRITE")
@Test
public void spiderUnloggedTest () throws InterruptedException {…}
```

Código 25 Anotaciones de los casos de prueba UnLoggedLinkTests

```
@ParameterizedTest
@MethodSource ("data")
@Resource (resID = "LoginService", replaceable = {})
@AccessMode (resID = "LoginService", concurrency = 10, sharing = true,
accessMode = "READONLY")
@Resource (resID = "OpenVidu", replaceable = {"OpenViduMock"})
@AccessMode (resID = "OpenVidu", concurrency = 10, sharing = true, accessMode
= "NOACCESS")
@Resource (resID = "Course", replaceable = {})
@AccessMode (resID = "Course", concurrency = 15, sharing = true, accessMode =
"READWRITE")
public void spiderLoggedTest (String usermail, String password, String role)
throws InterruptedException {…}
```

Código 26 Anotaciones de los casos de prueba LoggedLinkTests

8.4.3 Creación de ficheros de configuración y plantilla de docker-compose

Una vez anotados los casos de prueba, se crean los ficheros de configuración con la información y plantilla para generar los *docker-compose*, así como la información del repositorio. Los ficheros denominados *retorch-info.json* y *retorch-repo.json* se deben de situar en la raíz del proyecto y el fichero *docker-compose-template.yml* en la carpeta recursos de test. A continuación, se presenta el contenido de estos en los fragmentos Código 27 Código 28 y Código 29 :

```
{
  "OpenViduMock": {
    "urlDeploymentResource": "localhost",
    "dockerImageResource": [
      {
        "placeHolderId": "openviduimageresource",
        "imageName": "eexit/mirror-http-server"
      },
      {"placeHolderId": "mysqlimageresource","imageName": "mysql:5.7.22"}
    ],
    "basePorts": [
      {"name": "mainport","lowerPort": "5001", "upperPort": "5015"},
      {"name": "openviduport", "lowerPort": "10000", "upperPort": "10010"}
    ],"testsBasePath": "fullteaching/e2e-test/no-Elastest/"
  },
  "generic": {
    "urlDeploymentResource": "localhost",
    "dockerImageResource": [
      {"placeHolderId": "openviduimageresource","imageName":
"openvidu/openvidu-server-kms:2.3.0"},
      {"placeHolderId": "mysqlimageresource","imageName": "mysql:5.7.22"}
    ],
    "basePorts": [
      {"name": "mainport", "lowerPort": "5001", "upperPort": "5015"},
      {"name": "openviduport","lowerPort": "10000","upperPort": "10010"}
    ],"testsBasePath": "fullteaching/e2e-test/no-Elastest/"}
}
```

Código 27 Fichero retorch-info.json

```
{
 "urlGitHub": "https://github.com/augustocristian/full-teaching-tunon-tests",
 "credentialsGiHub": "GitHub",
 "branchGitHub": "emptyTestCasesBranch"
}
```

Código 28 Fichero retorch-repo.json

```
version: '3'
services:
  full-teaching-mysql-${tjobname}:
    image: mysql:5.7.21
    environment:
      - MYSQL_ROOT_PASSWORD=pass
      - MYSQL_DATABASE=full_teaching
      - MYSQL_USER=ft-root
      - MYSQL_PASSWORD=pass
  full-teaching-${tjobname}-openvidu-server-kms:
    image:  ${openviduimageresource}
    expose:
      - 8443
    ports:
      - ${openviduport}:8443
    environment:
      - KMS_STUN_IP=stun.l.google.com
      - KMS_STUN_PORT=19302
      - openvidu.secret=MY_SECRET
      - openvidu.publicurl=https://${openviduurl}:${openviduport}
  full-teaching-${tjobname}:
    image: codeurjc/full-teaching:demo
    depends_on:
      - full-teaching-mysql-${tjobname}
      - full-teaching-${tjobname}-openvidu-server-kms
    ports:
      - ${mainport}:5000
    expose:
      - 5000
    environment:
      - WAIT_HOSTS=full-teaching-mysql-${tjobname}:3306
      - WAIT_HOSTS_TIMEOUT=120
      - MYSQL_PORT_3306_TCP_ADDR=full-teaching-mysql-${tjobname}
      - MYSQL_PORT_3306_TCP_PORT=3306
      - MYSQL_ENV_MYSQL_DATABASE=full_teaching
      - MYSQL_ENV_MYSQL_USER=ft-root
      - MYSQL_ENV_MYSQL_PASSWORD=pass
      - server.port=5000
      - openvidu.url=https://${openviduurl}:${openviduport}
      - openvidu.secret=MY_SECRET
networks:
  fullteaching-${tjobname}:
    driver: bridge
```

Código 29 Contenido del fichero docker-compose-template.yml

En este último fichero se puede observar que los *placeholders* en este caso son cuatro ubicaciones sobre las que podremos configurar distintos valores, estas son:

- Rango de puertos de despliegue de la aplicación (*mainport*)
- Rango de puertos de despliegue del servidor multimedia (*openviduport*)
- Sufijo de los contenedores imagen empleada para el servidor multimedia (*tjobname*).
- Imágenes empleadas para levantar los contenedores (en nuestro caso *openviduimageresource* y *mysqlimageresource*)

Los rangos de puertos deben de calcularse teniendo en cuenta el recurso más la elasticidad de los recursos del sistema. Este rango debe de como mínimo tener suficiente tamaño como para desplegar esa cantidad de instancias que acota la elasticidad del recurso.

8.4.4 Configuración final de RETORCH

Una vez se completa los ficheros de configuración necesarios para ejecutar RETORCH, resta únicamente crear el método *main* desde el que se ejecuta. Este método debe de crearse en el paquete test del proyecto e invocar a RETORCH con la ruta y el nombre del proyecto ("*Fullteaching*"), según se indica en la Sección 8.1.2. Dentro del método principal se invoca a *generateJenkinsfile()* que desencadena la interacción de todos los componentes.

```
public class RetorchMain {
 public static void main (String [] args) {
  OrchestrationGenericToolBox toolBox = new OrchestrationGenericToolBox ();
  toolBox.generateJenkinsfile
("com.fullteaching.e2e.no_elastest.functional.test", "FullTeaching");}}
```

Código 30 Main con la invocación de RETORCH

8.4.5 Ejecución de RETORCH

La ejecución de *generateJenkinsfile ()* marca el inicio de la interacción entre los diferentes componentes de la arquitectura de RETORCH para lograr la optimización de los casos de prueba:

- El Clasificador toma los casos de prueba del paquete proporcionado y crea el sistema. El sistema que da como salida el Clasificador, tiene 21 casos de prueba y 10 recursos (presentados anteriormente).
- El Agrupador, realiza las combinaciones teniendo en cuenta la reemplazabilidad de los recursos, y agrupa los casos de prueba acorde al uso que hacen de esos recursos. RETORCH provee de los siguientes 20 *TGroups*:

```
0 = [ Resources [Attenders LoginService OpenVidu], TestCases
[attendersRestOperations]]

1 = [ Resources [Attenders LoginService OpenViduMock], TestCases
[attendersRestOperations]]

2 = [ Resources [Configuration LoginService OpenVidu], TestCases
[courseRestOperations oneToOneChatInSessionChrome teacherEditCourseValues]]

3 = [ Resources [Configuration LoginService OpenViduMock], TestCases
[courseRestOperations teacherEditCourseValues]]

4 = [ Resources [Course LoginService OpenVidu], TestCases
[attendersRestOperations courseInfoRestOperations courseRestOperations
filesRestOperations forumNewCommentTest forumNewEntryTest
forumNewReply2CommentTest forumRestOperations oneToOneChatInSessionChrome
oneToOneVideoAudioSessionChrome sessionRestOperations sessionTest
spiderLoggedTest spiderUnloggedTest teacherDeleteCourseTest
teacherEditCourseValues]]

5 = [ Resources [Course LoginService OpenVidu], TestCases
[forumLoadEntriesTest studentCourseMainTest teacherCourseMainTest
teacherCreateAndDeleteCourseTest]]

6 = [ Resources [Course LoginService OpenViduMock], TestCases
[attendersRestOperations courseInfoRestOperations courseRestOperations
filesRestOperations forumNewCommentTest forumNewEntryTest
forumNewReply2CommentTest forumRestOperations sessionRestOperations
spiderLoggedTest spiderUnloggedTest teacherDeleteCourseTest
teacherEditCourseValues]]

7 = [ Resources [Course LoginService OpenViduMock], TestCases
[forumLoadEntriesTest studentCourseMainTest teacherCourseMainTest
teacherCreateAndDeleteCourseTest]]

8 = [ Resources [Files LoginService OpenVidu], TestCases
[filesRestOperations]]

9 = [ Resources [Files LoginService OpenViduMock], TestCases
[filesRestOperations]]

10 = [ Resources [Forum LoginService OpenVidu], TestCases
[forumLoadEntriesTest]]

11 = [ Resources [Forum LoginService OpenVidu], TestCases
[forumNewCommentTest forumNewEntryTest forumNewReply2CommentTest
forumRestOperations]]

12 = [ Resources [Forum LoginService OpenViduMock], TestCases
[forumLoadEntriesTest]]

13 = [ Resources [Forum LoginService OpenViduMock], TestCases
[forumNewCommentTest forumNewEntryTest forumNewReply2CommentTest
forumRestOperations]]
```

Código 31 Lista de TGroups I

```
14 = [ Resources [Information LoginService OpenVidu], TestCases
[courseInfoRestOperations]]

15 = [ Resources [Information LoginService OpenViduMock], TestCases
[courseInfoRestOperations]]

16 = [ Resources [LoginService OpenVidu], TestCases [loginTest]]

17 = [ Resources [LoginService OpenViduMock], TestCases [loginTest]]

18 = [ Resources [Session LoginService OpenVidu], TestCases
[oneToOneVideoAudioSessionChrome sessionRestOperations sessionTest]]

19 = [ Resources [Session LoginService OpenViduMock], TestCases
[sessionRestOperations]]
```

Código 32 Lista de TGroups II

- El Organizador, partiendo de esos *TGroups*, y teniendo en cuenta el coste de los recursos da como salida los siguientes *TJobs*

```
0 = [listTc [attendersRestOperations], listRes [Attenders, LoginService,
OpenViduMock], intraSchedule 'SequentialScheduling', con 1]

1 = [listTc [oneToOneChatInSessionChrome], listRes [Configuration, LoginService,
OpenVidu], intraSchedule 'SequentialScheduling', con 1]

2 = [listTc [courseRestOperations, teacherEditCourseValues], listRes [Configuration,
LoginService, OpenViduMock], intraSchedule 'SequentialScheduling', con 1]

3 = [listTc [spiderLoggedTest, spiderUnloggedTest, teacherDeleteCourseTest], listRes
[Course, LoginService, OpenViduMock], intraSchedule 'SequentialScheduling', con 1]

4 = [listTc [studentCourseMainTest, teacherCourseMainTest,
teacherCreateAndDeleteCourseTest], listRes [Course, LoginService, OpenViduMock],
intraSchedule 'ParallelScheduling', con 10]

5 = [listTc [filesRestOperations], listRes [Files, LoginService, OpenViduMock],
intraSchedule 'SequentialScheduling', con 1]
```

Código 33 Lista de TJobs I

```
6 = [listTc [forumLoadEntriesTest], listRes [Forum, LoginService, OpenViduMock],
intraSchedule 'ParallelScheduling', con 10]

7 = [listTc [forumNewCommentTest, forumNewEntryTest, forumNewReply2CommentTest,
forumRestOperations], listRes [Forum, LoginService, OpenViduMock], intraSchedule
'SequentialScheduling', con 1]

8 = [listTc [courseInfoRestOperations], listRes [Information, LoginService,
OpenViduMock], intraSchedule 'SequentialScheduling', con 1]

9 = [listTc [loginTest], listRes [LoginService, OpenViduMock], intraSchedule
'ParallelScheduling', con 10]

10 = [listTc [oneToOneVideoAudioSessionChrome, sessionTest], listRes [Session,
LoginService, OpenVidu], intraSchedule 'SequentialScheduling', con 1]

11 = [listTc [sessionRestOperations], listRes [Session, LoginService, OpenViduMock],
```

Código 34 Lista de TJobs II

Teniendo en cuenta los *TJobs* y recursos disponibles, el Organizador genera el grafo de actividades (Ilustración 14). Esta figura ilustra como se ordenan las actividades para su ejecución, representando los recursos que emplean, y sus relaciones de precedencia.

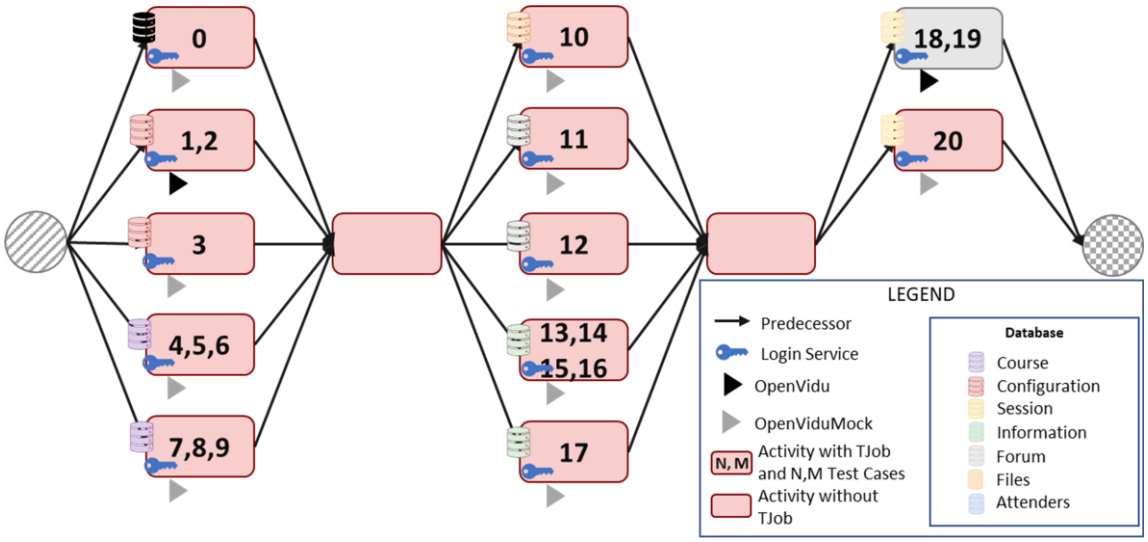

Ilustración 14 Grafo de Actividades

En este caso, actividades iniciales podrían ser indistintamente las que contienen los casos de prueba 0-1,2, mientras que los nodos finales serían las actividades que contienen a los casos de prueba 18-19. En color gris se dispone la ubicación del caso de prueba desactivado (18)

- En último lugar, partiendo del grafo anterior, el Orquestador genera el *Jenkinsfile* necesario para realizar el despliegue. Este fichero se muestra a continuación, en los fragmentos desde Código 35 al fragmento Código 42

```
pipeline {
agent any
 tools {
  maven 'M3.3.9'
  }//EndTools
stages{
  stage('Clone Repository') {
    steps{
       git branch: 'DevelopmentOpenViduTests',
           credentialsId: 'GitHub',
           url:'https://github.com/augustocristian/full-teaching-tunon-tests'
      }//EndStep}//EndStage
stage('Time 0'){
 steps{
parallel(
   "TJob 2 IdResource: Attenders LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobC; ls . ;docker-
compose --no-ansi -p TJobC up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5001 >
/dev/null; do echo waiting for TJobC; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5001" -
Dtest=FullTeachingTestE2EREST#attendersRestOperations test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobC; ls . ;docker-
compose --no-ansi -p TJobC down'
    }//EndParallelStep
   ,
```

Código 35 Jenkinsfile I

```
"TJob 3 IdResource: Configuration LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobD; ls . ;docker-
compose --no-ansi -p TJobD up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5002 >
/dev/null; do echo waiting for TJobD; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5002" -
Dtest=FullTeachingTestE2EREST#courseRestOperations,CourseTeacherTest#teacherEd
itCourseValues test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobD; ls . ;docker-
compose --no-ansi -p TJobD down'
   }//EndParallelStep
  ,
"TJob 4 IdResource: Configuration LoginService OpenVidu ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobE; ls . ;docker-
compose --no-ansi -p TJobE up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5003 >
/dev/null; do echo waiting for TJobE; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5003" -
Dtest=FullTeachingTestE2EChat#oneToOneChatInSessionChrome test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobE; ls . ;docker-
compose --no-ansi -p TJobE down'
  }//EndParallelStep
```

Código 36 Jenkinsfile II

```
"TJob 5 IdResource: Course LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobF; ls . ;docker-
compose --no-ansi -p TJobF up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5004 >
/dev/null; do echo waiting for TJobF; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5004" -
Dtest=LoggedLinksTests#spiderLoggedTest,UnLoggedLinksTests#spiderUnloggedTest,
CourseTeacherTest#teacherDeleteCourseTest test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobF; ls . ;docker-
compose --no-ansi -p TJobF down'
  }//EndParallelStep
,
"TJob 6 IdResource: Course LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobG; ls . ;docker-
compose --no-ansi -p TJobG up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5005 >
/dev/null; do echo waiting for TJobG; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -Dapp.url=https://192.168.0.103:5005
-Dtest=CourseStudentTest#studentCourseMainTest,
CourseTeacherTest#teacherCourseMainTest,
CourseTeacherTest#teacherCreateAndDeleteCourseTest test') //Can be
parallelized until 10 threads
echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobG; ls . ;docker-
compose --no-ansi -p TJobG down'

  }//EndParallelStep
 )//End Parallel
}//End Steps
}//End Stage
```

Código 37 Jenkinsfile III

```
stage('Time 1'){
 steps{
parallel(
"TJob 7 IdResource: Information LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobH; ls . ;docker-
compose --no-ansi -p TJobH up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5001 >
/dev/null; do echo waiting for TJobH; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5001" -
Dtest=FullTeachingTestE2EREST#courseInfoRestOperations test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobH; ls . ;docker-
compose --no-ansi -p TJobH down'

}//EndParallelStep
,
  "TJob 8 IdResource: Files LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobI; ls . ;docker-
compose --no-ansi -p TJobI up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5002 >
/dev/null; do echo waiting for TJobI; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5002" -
Dtest=FullTeachingTestE2EREST#filesRestOperations test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobI; ls . ;docker-
compose --no-ansi -p TJobI down'

  }//EndParallelStep
```

Código 38 Jenkinsfile IV

```
"TJob 9 IdResource: Forum LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobJ; ls . ;docker-
compose --no-ansi -p TJobJ up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5003 >
/dev/null; do echo waiting for TJobJ; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5003" -
Dtest=LoggedForumTest#forumLoadEntriesTest test')//Can be parallelized until
10 threads
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobJ; ls . ;docker-
compose --no-ansi -p TJobJ down'
  }//EndParallelStep
  ,
"TJob 10 IdResource: Forum LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobK; ls . ;docker-
compose --no-ansi -p TJobK up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5004 >
/dev/null; do echo waiting for TJobK; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5004" -
Dtest=LoggedForumTest#forumNewCommentTest,LoggedForumTest#forumNewEntryTest,L
oggedForumTest#forumNewReply2CommentTest,FullTeachingTestE2EREST#forumRestOpe
rations test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobK; ls . ;docker-
compose --no-ansi -p TJobK down'
    }//EndParallelStep
  ,
```

Código 39 Jenkinsfile V

```
"TJob 11 IdResource: LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobL; ls . ;docker-
compose --no-ansi -p TJobL up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5005 >
/dev/null; do echo waiting for TJobL; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5005" -Dtest=UserTest#loginTest test')//Can be
parallelized until 10 threads
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobL; ls . ;docker-
compose --no-ansi -p TJobL down'

    }//EndParallelStep
   )//End Parallel
}//End Steps
}//End Stage
stage('Time 2'){
 steps{
parallel(
  "TJob 12 IdResource: Session LoginService OpenVidu ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobM; ls . ;docker-
compose --no-ansi -p TJobM up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5001 >
/dev/null; do echo waiting for TJobM; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5001" -
Dtest=FullTeachingTestE2EVideoSession#oneToOneVideoAudioSessionChrome,LoggedVi
deoSession#sessionTest test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobM; ls . ;docker-
compose --no-ansi -p TJobM down'

    }//EndParallelStep
```

Código 40 Jenkinsfile VI

```
,
"TJob 13 IdResource: Session LoginService OpenViduMock ": {
  echo 'Deploying Resources Required'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobN; ls . ;docker-
compose --no-ansi -p TJobN up -d'
  echo 'Waiting for the SUT...'
  sh 'bash -c "while ! curl --insecure -s https://192.168.0.103:5002 >
/dev/null; do echo waiting for TJobN; sleep 0.3; done;"'
  echo 'SUT READY!'
  echo 'Executing the test cases'
  sh ('cd e2e-test/no-Elastest/;ls .;mvn -
Dapp.url="https://192.168.0.103:5002" -
Dtest=FullTeachingTestE2EREST#sessionRestOperations test')
  echo 'Disposing the resources'
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobN; ls . ;docker-
compose --no-ansi -p TJobN down'

    }//EndParallelStep
   )//End Parallel
}//End Steps
}//End Stage
}
 post {
  failure {
  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobC; ls . ;docker-
compose --no-ansi -p TJobC down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobD; ls . ;docker-
compose --no-ansi -p TJobD down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobE; ls . ;docker-
compose --no-ansi -p TJobE down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobF; ls . ;docker-
compose --no-ansi -p TJobF down'
```

Código 41 Jenkinsfile VII

```
sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobH; ls . ;docker-compose
--no-ansi -p TJobH down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobI; ls . ;docker-
compose --no-ansi -p TJobI down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobJ; ls . ;docker-
compose --no-ansi -p TJobJ down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobK; ls . ;docker-
compose --no-ansi -p TJobK down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobL; ls . ;docker-
compose --no-ansi -p TJobL down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobM; ls . ;docker-
compose --no-ansi -p TJobM down'

  sh 'cd e2e-test/no-Elastest/retorchdockercomposes/TJobN; ls . ;docker-
compose --no-ansi -p TJobN down'

}
  cleanup {
  sh 'echo "y" | docker volume prune'}
}
}
```

Código 42 Jenkinsfile VIII

El *Jenkinsfile* y los *docker-compose* son añadidos al repositorio, y este al sistema de integración continua. Se crea un *Job* de *Jenkins* de tipo *SCM* y se indica en que ruta está el *Jenkinsfile*, en este caso: "*e2e-test/no-Elastest/Jenkinsfile*".

8.5 Infraestructura empleada

La validación de RETORCH se ha realizado desplegando la infraestructura en un ordenador de usuario, también denominado "*commodity pc*". Este ordenador posee las características que se citan a continuación:

- **Memoria y Almacenamiento**: El ordenador tiene 32 Gigabytes de memoria RAM de tipo *DDR4* con una frecuencia de 3000Mhz, esta memoria viene acompañada de un *SSD* M2 de 500 GB cuyas velocidades de escritura y lectura ascienden a más de 500Mb/s.

- **Procesador y placa base:** La placa base tiene un *chipset* 450-B y un procesador de la familia *Ryzen* con 8 núcleos físicos y 16 hilos lógicos, esta placa y procesador cuentan con el último juego de instrucciones AVX y soporte para virtualización.
- **Sistema operativo**: tiene instalado *Windows 10 Pro Edition*, con las últimas actualizaciones y parches de seguridad.
- **Software de virtualización:** se emplea *Hyper-V*, ya que viene de forma nativa con esta versión de *Windows*.

En el ordenador se ha virtualizado una máquina sobre las que se han desplegado los distintos contenedores de la infraestructura.

- **Máquina virtual *Ubuntu***: posee asignados 60 GB, 10 núcleos lógicos y 28,5 Gigabytes de RAM. En cuanto al software tiene instalado *Ubuntu* 18.04, y *Docker* en sus versiones 19.03 y 1.17 de *docker-compose*

Sobre esta máquina se desplegaron los distintos contenedores que usa la infraestructura (descritos en la Tabla 8):

Tabla 8 Contenedores desplegados en las máquinas

Máquina	Contenedor	Versión
Ubuntu	*Selenoid*	*lastest-release* (v1.10)
Ubuntu	*Selenoid UI*	*lastest-release* (v1.10)
Ubuntu	*Jenkins*	v1.2.0
Ubuntu	*Fullteaching*	v2.3
Ubuntu	*MySQL*	v5.7
Ubuntu	*OpenVidu*	v2.3.0
Ubuntu	*Mirror-http-server*	*lastest-release* (v1.1.2)
Ubuntu	*Portainer*	*lastest-release*(windows1903-amd64)
Ubuntu	*cAdvisor*	*lastest-release* (v0.33.0)
Ubuntu	*Prometheus*	*lastest-release* (v2.19.2)

Además de todo esto, han expuesto al exterior una serie de puertos para facilitar el acceso, especificados a continuación con una descripción de la funcionalidad aportada por el contenedor:

- *Jenkins*: Accesible mediante [IPUbuntu]:37092, se trata del sistema de integración continua. Cuenta con dos esclavos, *Maven* 3.3 instalado y acceso al *Daemon* de *Docker* de su anfitrión (*Ubuntu*)
- *cAdvisor*: Accesible mediante [IPUbuntu]:9000, provee de mecanismos de monitorización del uso de recursos de los contenedores.
- *Prometheus*. Accesible en [IPUbuntu]:9080 provee de mecanismos de recopilación y almacenamiento de datos de monitorización de servicios. Con este servicio se ha realizado la monitorización de los recursos del sistema, recopilando las métricas que saca *cAdvisor* en tiempo real.
- *Selenoid y SelenoidUI*: Su interfaz web se encuentra en [IPAnfitrion]:8080, estos dos contenedores son los encargados de proveer de navegadores contenerizados a las pruebas, además de mecanismos de monitorización como la grabación de la ejecución de los casos de prueba. Con estos contenedores se evita los problemas con las dependencias que podría acarrear usar el mismo navegador.
- *Portainer*: Interfaz web para el manejo de los contenedores, accesible mediante [IPAnfitrion]:9000.
- *Fullteaching, MySQL y http-mirror-server y OpenVidu*: Accesibles mediante los rangos de puertos especificados en los ficheros de configuración en la IP de la máquina *Ubuntu*. Componen el sistema bajo prueba (en este caso *Fullteaching*)

8.6 Resultados

Realizada la ejecución, se procede a analizar los resultados. Se compara si RETORCH es capaz de reducir el uso de recursos al orquestar casos de prueba. Para ello, se compara con una ejecución secuencial donde los casos de prueba despliegan y liberan los recursos según los van necesitando. En las siguientes secciones (1) Tiempo de ejecución, (2) Almacenamiento y cantidad de instancias contenerizadas, (3) Memoria empleada se describen los ahorros en términos de recursos logrados por RETORCH para finalmente, en (4) Discusión y conclusiones debatirse los mismos.

8.6.1 Tiempo de ejecución

Para medir el tiempo de ejecución de los casos de prueba, se ha empleado el tiempo reportado por el sistema de integración continua (*Jenkins*) en la ejecución del *Job*, comparándose la ejecución secuencial con el tiempo de ejecución de los casos de prueba orquestados por RETORCH. Ambas ejecuciones tienen en común el clonado del repositorio, el cual tarda 5-8 segundos en caso de traerse las últimas contribuciones. Para la medición del tiempo de ejecución se han realizado 5 ejecuciones con RETORCH y 5 ejecuciones en secuencial, tomando como tiempo el promedio de todas ellas (Representados en la Tabla 9)

Tabla 9 Tiempos ejecución secuencial en segundos

	Secuencial		RETORCH	
Nº de ejecución	Clonado Rep.	Ejecución casos de prueba	Clonado Rep.	Ejecución casos de prueba
1	6	1.495	6	520
2	5	1.555	6	474
3	6	1.435	6	495
4	5	1.435	5	526
5	5	1.435	6	600
Promedio	5,4	1.471	5,8	523

En la Tabla 9 se representa el tiempo de ejecución empleando RETORCH con el tiempo de ejecución secuencial. La ejecución secuencial se compone de tres pasos: (1) preparar los recursos necesarios, (2) ejecutar el caso de prueba y (3) liberar los recursos para cada uno de los casos de prueba del conjunto. La ejecución de los casos de prueba de forma secuencial ha llevado un total de 1.476,4 segundos, de los cuales 1.471 segundos corresponden a los propios casos de prueba.

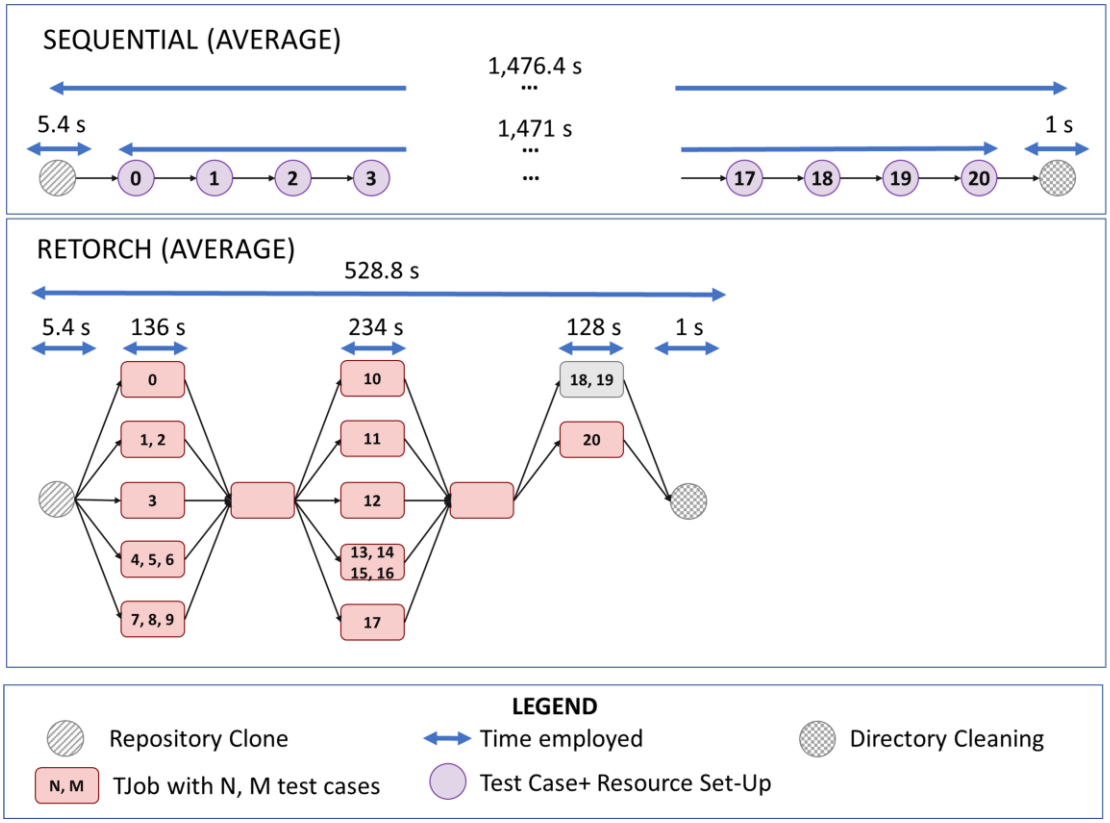

Ilustración 15 Tiempo de ejecución Secuencial y con RETORCH

Empleando RETORCH, la ejecución ha tardado 528,8 segundos en promedio, siendo 523 segundos el tiempo empleado en ejecutar los casos de prueba. Los promedios del tiempo de ejecución de los *TJobs* en paralelo son 135,6 segundos el primer grupo, 233,8 segundos el segundo grupo y finalmente 127,2 segundos el tercer grupo. Comparando los tiempos de ambas alternativas, se observa que **empleando RETORCH se ha conseguido reducir el tiempo de ejecución de los casos de prueba un 64%**

En la Tabla 10 se presenta el tiempo de ejecución con RETORCH de cada uno de los casos de prueba de forma individual con objeto de compararlo con la ejecución Secuencial. Entre paréntesis se anota que casos de prueba son ejecutados en un mismo *TJob*, y se presentan desglosados los tiempos de *Set-up* y espera por el *SUT*, el tiempo de ejecución y de liberación de recursos de RETORCH. En la última columna se dispone el porcentaje de tiempo que ha tardado RETORCH respecto a la ejecución secuencial. Notar que en caso de que un *TJob* ejecute varios casos de prueba, se sumarán los tiempos de ejecución de esos casos de prueba.

Tabla 10 Desglose del tiempo de ejecución de los casos de prueba

Nombre del caso de prueba	Número y Grupo	Secuencial	RETORCH					
			Set-Up	Wait	Exec	Dispose	Total	Porcentaje
attendersRestOperations	1-0	64	12	45	76	13	146	-22%
courseRestOperations	1-1	56	12	41	96	13	162	-13%
teacherEditCourseValues	1-2	66	12	41	96	13	162	-13%
oneToOneChatInSessionChrome	1-3	85	15	41	94	13	163	92%
spiderLoggedTest	1-4	68	15	43	101	12	171	3%
spiderUnloggedTest	1-5	51	15	43	101	12	171	3%
teacherDeleteCourseTest	1-6	47	15	43	101	12	171	3%
studentCourseMainTest	1-7	55	8	51	97	13	169	2%
teacherCourseMainTest	1-8	53	8	51	97	13	169	2%
teacherCreateAndDeleteCourseTest	1-9	57	8	51	97	13	169	2%
courseInfoRestOperations	2-10	62	8	31	41	13	93	50%
filesRestOperations	2-11	64	9	29	50	13	101	58%
forumLoadEntriesTest	2-12	193	13	32	163	12	220	12%
forumNewCommentTest	2-13	72	17	32	119	13	181	-35%
forumNewEntryTest	2-14	74	17	32	119	13	181	-35%
forumNewReply2CommentTest	2-15	72	17	32	119	13	181	-35%
forumRestOperations	2-16	62	17	32	119	13	181	-36%
loginTest	2-17	69	18	34	35	13	100	45%
oneToOneVideoAudioSessionChrome	3-18	-	-	-	-	-	113	-
sessionTest	3-19	73	7	26	67	13	113	-2%
sessionRestOperations	3-20	62	7	25	25	13	70	13%
Promedio casos singulares ((-22+92+50+58+12+45+13) /6)								45%
Promedio general ((-22-13+92+3+2+50+58+12-35+45-2+13) /12)								21%

Con los distintos colores se ha querido representar las agrupaciones de casos de prueba en los diferentes *TJobs*. Se puede observar que incluso en aquellas ejecuciones en las que se

ejecuta un solo caso de prueba (ej. *loginTest* o *forumLoadEntriesTest* en la Tabla 10) el tiempo de ejecución es un tanto mayor.

En promedio poniendo el foco en esos casos de prueba cuya ejecución se asemeja a la Secuencial (filas blancas), **tardan un 45% más** que la alternativa secuencial. Sin embargo, si se comparan los casos de prueba que se ejecutan dentro de un mismo *TJob* (tomando como referencia un solo tiempo por cada grupo), vemos que este valor mejora **tardando solamente un 21% más que la alternativa secuencial.** La diferencia entre porcentajes al tener en cuenta los casos similares al secuencial o incluirlos todos se puede explicar con el número de despliegues que RETORCH necesita contra la alternativa Secuencial: mientras que por ejemplo para *courseRestOperations* y *teacherEditCourseValues* la alternativa secuencial debe desplegar dos veces el *SUT*, con RETORCH únicamente es necesario un redespliegue.

8.6.2 Almacenamiento y número de contenedores desplegados.

Para medir el almacenamiento y número de contenedores desplegados, se han empleado las métricas de disco extraídas por *Prometheus*, recopiladas de la monitorización hecha por *cAdvisor*, así como la información provista en la liberación de recursos de *Docker*. Para cada alternativa, se ha comentado el *stage* de *Jenkins*, en el que se liberan los recursos tras la ejecución. Tras cada ejecución se ha realizado la eliminación de los volúmenes de los contenedores empleados, contabilizándose de forma manual el almacenamiento empleado. Cada conjunto de base de datos, servidor web y servidor multimedia ocupan 208MB de disco duro, por lo que cada *set-up* de los recursos reservará ese almacenamiento. Analizando el consumo de recursos desde el punto de vista de los contenedores, tenemos también que por cada *set-up* se requieren 3 contenedores (base de datos, servidor web y multimedia).

La ejecución secuencial realiza el *set-up* en 21 ocasiones, una vez por cada caso de prueba que se ejecuta, empleando 208MB de almacenamiento y 3 contenedores en cada ejecución. En almacenamiento consumido, esta ejecución requiere 4.368MB mientras que en número de contenedores requiere 63 instancias contenerizadas.

En contraposición, la ejecución con RETORCH requiere hacer el *set-up* en 12 ocasiones, lo que se traduce en 2.496 MB de almacenamiento y 36 instancias contenerizadas

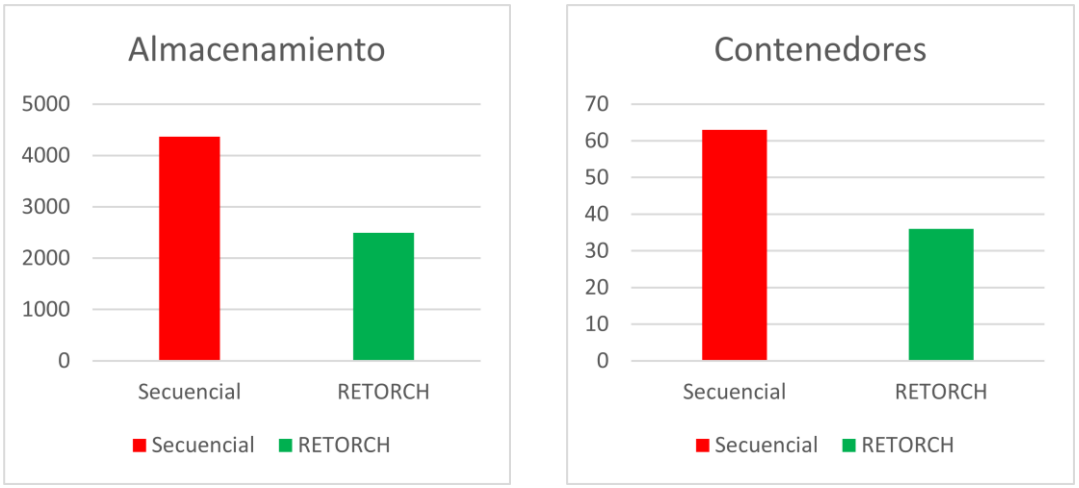

Ilustración 16 Almacenamiento y cantidad de contendores empleados

La diferencia de almacenamiento y contenedores se puede observar en la Ilustración 16, el ahorro conseguido con RETORCH, es de un 43% en términos de almacenamiento y contenedores desplegados, porcentaje calculado a partir de la cantidad de memoria y contenedores empleados.

8.6.3 **Uso de memoria**

Para la medición del uso de memoria del sistema, se han vuelto a emplear *Prometheus* y *cAdvisor*, esta vez monitorizando la métrica “bytes de memoria usados” durante la ejecución de los casos de prueba. En ambas ejecuciones se compara la cantidad de memoria necesaria para lograr la ejecución de estas (midiendo los máximos requeridos durante la ejecución). Las instancias virtualizadas en *Docker* (*Prometheus*, *CAdvisor y Jenkins*) sin ejecutar ningún caso de prueba (*idle*) presentan un uso de memoria de 1,14, lo que supone un 4 % de la memoria RAM máxima (28.5 GB). He de destacar que las escalas de las gráficas no representan Gigabytes (2^{10} bytes) si no Gigas (10^9 bytes)

En la ejecución secuencial (Ilustración 17) se observa de forma clara los 21 valles correspondientes con el *set-up* y la disposición de los recursos por cada caso de prueba. En esos valles entre casos de prueba, el uso de la memoria cae hasta los 2.01GB, valor próximo

al de la máquina en *idle*. Durante la ejecución de los casos de prueba, con el recurso levantado, el uso de memoria de la máquina virtual se mantiene en torno a los 4.4 GB, superándose en alguna ocasión los 7 GB de memoria (7,19 GB).

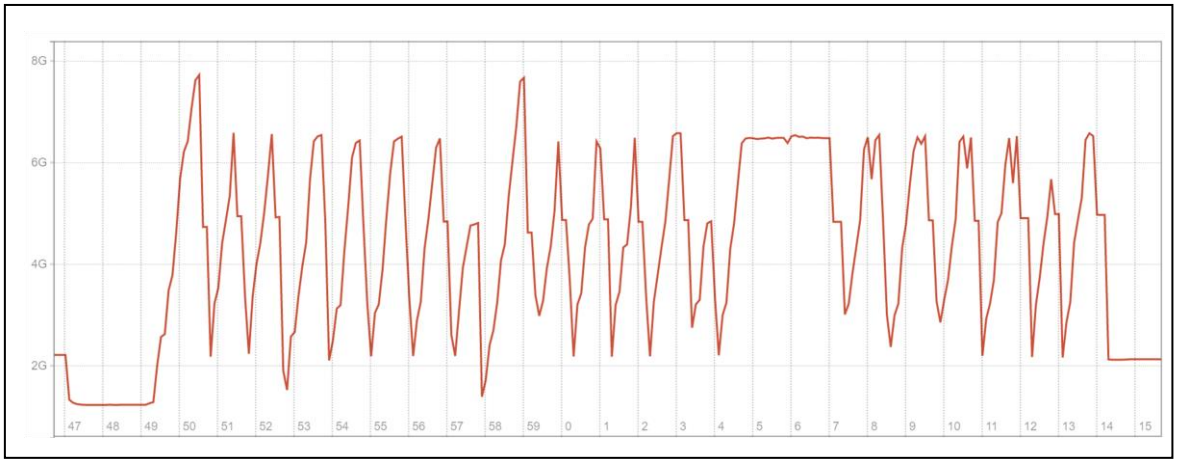

Ilustración 17 Uso de memoria con Secuencial en Gigabits

En la ejecución con RETORCH (Ilustración 18) se vuelven a apreciar los valles correspondientes a los tres grupos de *TJobs* desplegados en paralelo. El consumo de la máquina en *Idle* se mantiene, pero se ve como las cotas máximas de memoria cambian. Durante la ejecución del primer grupo de cinco *TJobs* en paralelo, el valor máximo de memoria es 19,6 GB, el segundo grupo de cinco *TJobs* consume una cantidad similar, llegando a los 19,55 GB. Finalmente, el tercer grupo formado únicamente por dos *TJobs* en paralelo llega a los 12,36 GB de memora RAM:

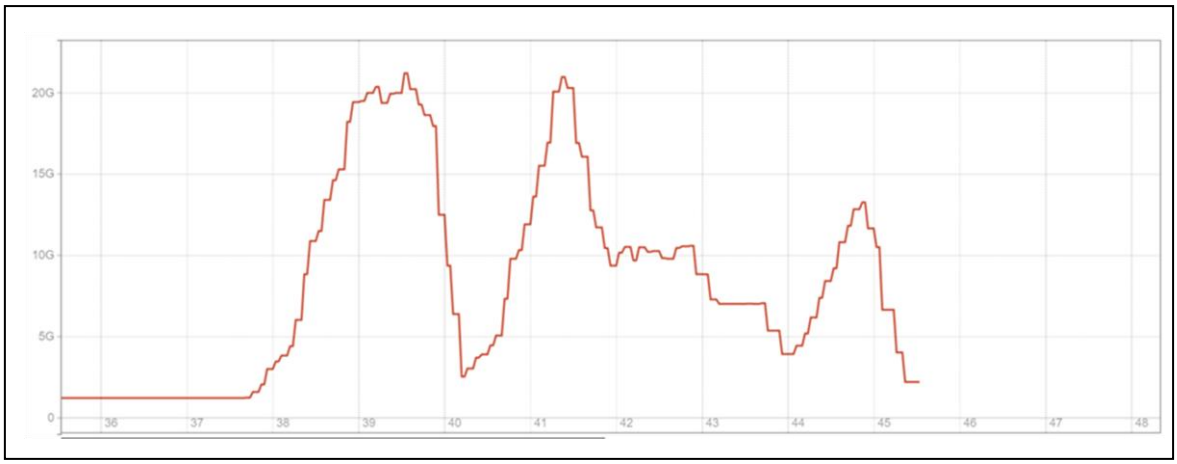

Ilustración 18 Uso de memoria con RETORCH en Gigabits

Comparando la ejecución en Secuencial con RETORCH, podemos concluir que en este caso la ejecución en secuencial alcanza máximos un 60% inferiores en cuanto a memoria física, logrando una mejor distribución de los recursos (menos picos y valles).

8.6.4 **Discusión**

Según lo expuesto en los apartados anteriores, se ha comprobado como RETORCH es capaz de lograr ahorros de recursos en términos de tiempo, almacenamiento y número de instancias empleadas.

Si observamos los resultados obtenidos con el despliegue en paralelo de los casos de prueba hecho por RETORCH, se ve de forma clara como la aceleración obtenida no es lineal. En nuestro caso, pasar de una ejecución secuencial, a ejecutar en cinco pipelines de ejecución, no divide entre cinco el tiempo de ejecución total. Algunas de las causas por las que no se puede llegar a esta aceleración son: (1) la forma en la que se han organizado los *TJobs* en grupos desplegándose en paralelo, deriva en que si existe un *TJob* que dure más que el resto, los demás hilos permanecerán ociosos hasta que acabe (los denominados caminos críticos, representados para una ejecución de RETORCH en la Ilustración 19). Como se puede observar, aunque en las ejecuciones en paralelo hay *TJobs* que se ejecutan en menos tiempo que el total, tienen que esperar que se ejecute el más costoso en términos de tiempo para finalizar

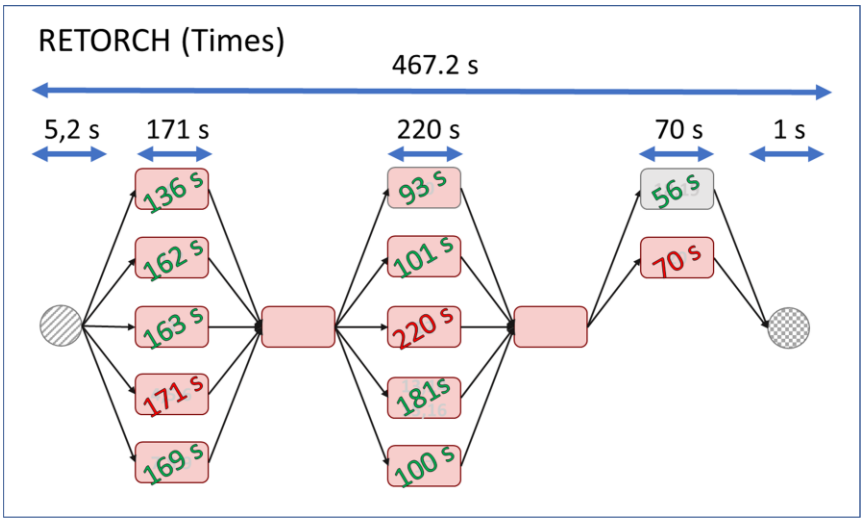

Ilustración 19 Caminos críticos RETORCH

(2) las dependencias entre los casos de prueba: (ej. emplear una misma instancia de *Selenoid*, partir todos del mismo proyecto clonado o emplear la misma instancia de *Docker*) y finalmente (3) limitaciones propias del hardware (acceso concurrente a memoria, núcleos "lógicos" de la MV o ineficiencias derivadas de virtualizar el hardware).

Respecto al uso de memoria, se ha visto que los ahorros en tiempo de ejecución han repercutido directamente en una mayor cantidad de memoria requerida para la ejecución (pico máximo de memoria) (60%) por parte de RETORCH comparada con la ejecución secuencial.

También se han ahorrado recursos al ejecutar los casos de prueba compatibles con el mock del servidor *OpenVidu* en vez de emplear el recurso original, como se describe en la Ilustración 20, en la que la cantidad de memoria máxima requerida para cada caso de prueba por con el *mock* del servidor *OpenVidu* (color rojo) es un tanto inferior.

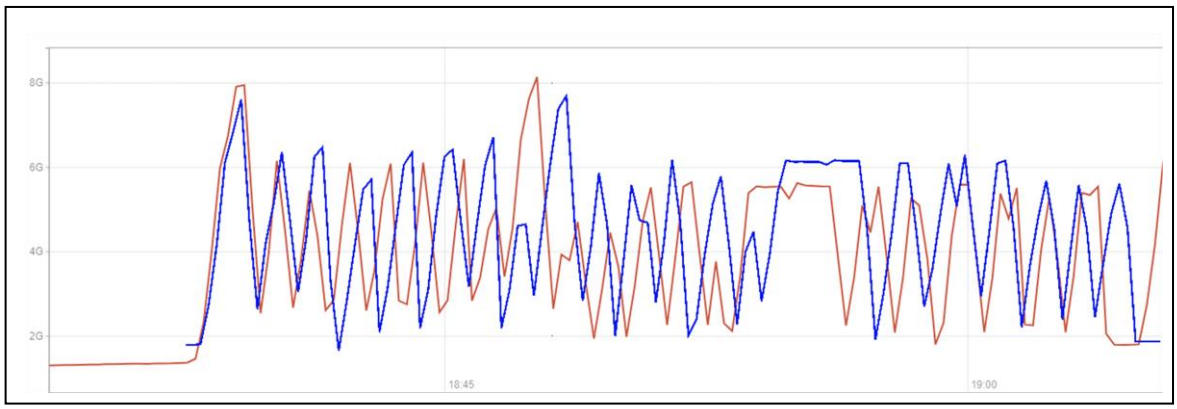

Ilustración 20 Comparación Secuencial con el mock de OpenVidu

Para finalizar, aunque se ha decidido dejar fuera del alcance de este trabajo fin de máster, debido a que la ejecución concurrente de casos de prueba de *Selenium* es una funcionalidad experimental, el ordenamiento *Intra-TJob* (Sección 6.2.4) podría derivar en ahorros todavía mayores en aquellos *TJobs* que permitiesen ejecutar de forma paralela sus casos de prueba, reduciendo aún más el tiempo de ejecución permitiendo un uso más eficiente de los recursos, que serán empleados durante más tiempo y su capacidad será aprovechada por un mayor número de casos de prueba concurrentemente.

9. Conclusiones y trabajos futuros

Tras la realización de este trabajo, se ha comprobado que la optimización de recursos en las pruebas de sistema no puede verse como un problema mono-objetivo, ya que son varios los factores a tener en cuenta, muchos de ellos antagonistas, aumentando uno al reducir el otro y viceversa. En el presente trabajo fin de máster se ha desarrollado un primer prototipo de RETORCH, el cual se ha validado en una aplicación real llamada *FullTeaching*. Al comienzo de este trabajo de fin de máster partíamos de tres Hipótesis iniciales (en adelante H1, H2 y H3), alrededor de las cuales se presentan las conclusiones.

Hipótesis 1: *"La ejecución de conjuntos de casos de prueba de Sistema (End to End o E2E) pueden requerir grandes cantidades de recursos físico-lógicos, con los que la efectividad de técnicas como la paralelización, minimización o reducción están limitadas, debido a las dependencias o a la distribución del consumo de recursos"*

H1: Tras la ejecución de los casos de prueba de la aplicación, se ha podido comprobar que la ejecución de las pruebas de sistema requería grandes cantidades de recursos como memoria, navegadores o contenedores. Tras intentar paralelizar los casos de prueba sin aplicar RETORCH, se ha comprobado que muchos de los casos de prueba fallaban debido a las dependencias existentes entre ellos. Además, debido a la naturaleza de los recursos de las pruebas de sistema, lograr encontrar un conjunto reducido no solucionaba el problema, ya que se siguen necesitando instanciar los recursos requeridos (aunque sea para ese conjunto mínimo).

Hipótesis 2: "*La ejecución de pruebas de sistema puede ser optimizada analizando las dependencias entre pruebas buscando concurrencia en el acceso a los recursos (ej. Variables, dispositivos hardware, archivos, etcétera). Una vez esas dependencias son detectadas, los casos de prueba que interfieren con otros pueden ser aislados o modificados para evitar efectos colaterales en la ejecución de las pruebas."*

Hipótesis 3: *"Una caracterización racional de los casos de prueba acompañada de un agrupamiento y una ordenación atendiendo a los recursos que emplean, puede derivar en un ahorro de tiempo/coste en la ejecución del conjunto de casos de prueba."*

H2 y H3: Se ha comprobado que, con una caracterización racional de los recursos empleados por los casos de prueba, estos pueden ser agrupados acorde a su uso similar de recursos. Estos grupos se pueden optimizar, dividiéndolos con objeto de optimizar el uso de recursos y tiempo de ejecución. Finalmente se pueden desplegar en el entorno de integración continúa obteniendo ahorros den términos de recursos y tiempo de ejecución. El prototipo implementado de RETORCH ha obtenido importantes ahorros para el conjunto de prueba empleado en la validación (*Fullteaching*). Estos ahorros se traducen en un menor tiempo de ejecución, que se ha visto reducido en un 64 %, así como un menor uso de almacenamiento e instancias contenerizadas a la hora de ejecutar los casos de prueba (ahorrando un 43%). Estos ahorros extrapolados a los grandes conjuntos de casos de prueba que existen en la industria permitirían ahorrar grandes ahorros, los cuales se podrían trasladar directamente a términos monetarios.

Son muchos los desafíos y líneas de trabajo futuro abiertas que ha dejado este trabajo de fin de máster tras de sí, algunas de las cuales se presentan a continuación:

La línea de trabajo futuro más prominente incide en la caracterización e identificación de los recursos empleados en las pruebas. Se propone seguir refinando la taxonomía de recursos buscando nuevos atributos que puedan aportar más información de como son los recursos empleados por los casos de prueba. Para esto planeamos emplear nuevos sujetos de estudio, con interesantes alternativas como el conjunto de casos de prueba [45] publicado por el equipo de desarrolladores de *ElasTest* en su último trabajo. Además, se plantea modelar e incluir el concepto de coste, con objeto de comparar los diferentes modelos que podemos encontrar hoy en día en la industria: la denominada computación *on-premise* que tradicionalmente se realiza en las instalaciones propias, con los nuevos paradigmas como el *Cloud Computing*.

Se tiene planeado integrar RETORCH con la plataforma *ElasTest* para orquestar de forma eficiente los casos de prueba del sistema. *ElasTest* provee de una librería de orquestación (*Orchestration ToolBox*) que permite ejecutar varios *Jobs* de *Jenkins*, ya bien sea de forma secuencial o paralela, teniendo en cuenta sus resultados de ejecución y pudiendo tomar como entrada la salida de otro.

Otra línea de trabajo futuro sería referente a la forma a la que se está agrupando los casos de prueba. En la actualidad el Agrupador está realizando un producto cartesiano de los recursos-recursos reemplazables posibles. En la aplicación empleada solo tenemos 10 recursos por lo que el producto de estos es manejable en términos de complejidad computacional. Se propone investigar cómo se podría realizar dicho agrupamiento de una forma mucho más eficiente para lograr que RETORCH pueda escalar de forma razonable junto con el tamaño del problema.

En este trabajo se han estudiado el impacto de RETORCH en términos de 3 factores: memoria, almacenamiento y tiempo de ejecución, como trabajo futuro se propone realizar un estudio más profundo para lograr un mayor entendimiento de las expensas en términos de recursos en las que incurrimos con RETORCH.

El proceso de ordenación es complejo y plantea nuevos desafíos por lo que en este trabajo fin de máster se ha abordado de forma sencilla con un algoritmo ad-hoc. Como otra línea de trabajo futuro, se propone investigar que algoritmos o técnicas se podrían emplear para lograr una ordenación óptima de los *TJobs*, como por ejemplo *Job Shop Scheduling*.

10. Bibliografía

[1] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy, “The art of testing less without sacrificing quality,” in *Proceedings - International Conference on Software Engineering*, 2015, vol. 1, pp. 483–493.

[2] A. Bertolino *et al.*, “A systematic review on cloud testing,” *ACM Comput. Surv.*, vol. 52, no. 5, 2019.

[3] A. Memon *et al.*, “Taming google-scale continuous testing,” in *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017*, 2017, pp. 233–242.

[4] F. Shull *et al.*, “What we have learned about fighting defects,” in *Proceedings - International Software Metrics Symposium*, 2002, vol. 2002-Janua, pp. 249–258.

[5] C. Augusto, J. Morán, A. Bertolino, C. de la Riva, and J. Tuya, “RETORCH:

Resource-Aware End-to-End Test Orchestration," in *Communications in Computer and Information Science*, 2019, vol. 1010, pp. 297–310.

[6] C. Augusto, J. Morán, A. Bertolino, C. de la Riva, and J. Tuya, "RETORCH: an approach for resource-aware orchestration of end-to-end test cases," *Softw. Qual. J.*, no. SQJO-D-19-00197R1, 2020.

[7] C. Augusto, "Efficient test execution in End to End testing (ICSE 2020 - ACM Student Research Competition) - ICSE 2020," *2020*. [Online]. Available: https://2020.icse-conferences.org/details/icse-2020-ACM-Student-Research-Competition/20/Efficient-test-execution-in-End-to-End-testing. [Accessed: 28-Jun-2020].

[8] B. Garcia *et al.*, "A proposal to orchestrate test cases," in *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*, 2018, pp. 38–46.

[9] M. Harman, "Making the case for MORTO: Multi objective regression test optimization," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 2011, pp. 111–114.

[10] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *FoSE 2007: Future of Software Engineering*, 2007, pp. 85–103.

[11] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing Verification and Reliability*, vol. 22, no. 2. John Wiley and Sons Ltd., pp. 67–120, Mar-2012.

[12] G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Softw. Test. Verif. Reliab.*, vol. 12, no. 4, pp. 219–249, 2002.

[13] W. E. Wong, J. R. Morgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," *Softw. - Pract. Exp.*, vol. 28, no. 4, pp. 347–369, 1998.

[14] E. Engström, M. Skoglund, and P. Runeson, "Empirical evaluations of regression test selection techniques: A systematic review," in *ESEM'08: Proceedings of the 2008*

ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 22–31.

[15] R. Lachmann, M. Nieke, C. Seidl, I. Schaefer, and S. Schulze, "System-level test case prioritization using machine learning," in *Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016*, 2017, pp. 361–368.

[16] J. Bell, G. Kaiser, E. Melski, and M. Dattatreya, "Efficient dependency detection for safe Java test acceleration," in *2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings*, Bergamo, Italy, 2015, pp. 770–781.

[17] A. Gambi, J. Bell, and A. Zeller, "Practical Test Dependency Detection," in *Proceedings - 2018 IEEE 11th International Conference on Software Testing, Verification and Validation, ICST 2018*, 2018, pp. 1–11.

[18] A. Gyori, A. Shi, F. Hariri, and D. Marinov, "Reliable testing: Detecting state-polluting tests to prevent test dependency," in *2015 International Symposium on Software Testing and Analysis, ISSTA 2015 - Proceedings*, 2015, pp. 223–233.

[19] A. Gambi, A. Gorla, and A. Zeller, "O!Snap: Cost-Efficient Testing in the Cloud," in *Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017*, 2017, pp. 454–459.

[20] C. H. Liu, S. L. Chen, and W. K. Chen, "Cost-benefit evaluation on parallel execution for improving test efficiency over cloud," in *Proceedings of the 2017 IEEE International Conference on Applied System Innovation: Applied System Innovation for Modern Technology, ICASI 2017*, 2017, pp. 199–202.

[21] S. S. Chakraborty and V. Shah, "Towards an approach and framework for test-execution plan derivation," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*, 2011, pp. 488–491.

[22] L. Yu, Y. Su, and Q. Wang, "Scheduling test execution of WBEM applications," in

Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2009, pp. 323–330.

[23] H. Esfahani *et al.*, "CloudBuild: Microsoft's distributed and caching build service," in *Proceedings - International Conference on Software Engineering*, Austin, Texas, 2016, pp. 11–20.

[24] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of NFV services in Software-Defined Networks," in *1st IEEE Conference on Network Softwarization: Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*, 2015, pp. 1–5.

[25] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[26] Docker Inc., "Swarm mode overview - Docker Documentation," 2019. [Online]. Available: https://docs.docker.com/engine/swarm/. [Accessed: 15-Oct-2019].

[27] Z. Zhang, C. Li, Y. Tao, R. Yangy, H. Tang, and J. Xu, "Fuxi: A fault-tolerant resource management and job scheduling system at internet scale," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1393–1404, Aug. 2014.

[28] Microsoft, "Orchestrator overview | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/system-center/orchestrator/learn-about-orchestrator?view=sc-orch-2019. [Accessed: 15-Oct-2019].

[29] W. Draft, "TOSCA Simple Profile in YAML Version 1.0," no. March, pp. 1–83, 2014.

[30] K. Velasquez *et al.*, "Fog orchestration for the Internet of Everything: state-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 9, no. 1, p. 14, Dec. 2018.

[31] K. Velasquez *et al.*, "Service orchestration in fog environments," in *Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017*, 2017, vol. 2017-Janua, pp. 329–336.

[32] M. S. De Brito *et al.*, "A service orchestration architecture for Fog-enabled

infrastructures," in *2017 2nd International Conference on Fog and Mobile Edge Computing, FMEC 2017*, 2017, pp. 127–132.

[33] K. Toczé and S. Nadjm-Tehrani, "A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing," *Wireless Communications and Mobile Computing*, vol. 2018. pp. 1–23, 04-Jun-2018.

[34] J. Jenkov, "Jenkov.com - Java Annotations." [Online]. Available: http://tutorials.jenkov.com/java/annotations.html#java-annotation-purposes. [Accessed: 29-Jan-2020].

[35] C. Augusto, J. Morán, C. De La Riva, and J. Tuya, "RETORCH Repository." 2020.

[36] P. F. Pérez, "Fullteaching: A web application to make teaching online easy." Universidad Rey Juan Carlos, 2017.

[37] "ElasTest Home." [Online]. Available: https://elastest.eu/. [Accessed: 27-Jun-2020].

[38] R. J. C. University, "OpenVidu." 2017.

[39] URJC and P. F. Pérez, "Repositorio FullTeaching - ElasTest." 2019.

[40] B. García, "WebDriverManager- Repository." 2015.

[41] B. García, "Selenium Jupiter - Repository," 2017.

[42] C. Augusto, "OpenVidu Issue Fullteaching," 2019.

[43] K. Technologies, "Kurento." 2014.

[44] J. Manak, "Bug SurefirePlugin." [Online]. Available: https://stackoverflow.com/questions/4646014/running-a-single-test-in-maven-no-tests-were-executed. [Accessed: 07-Jul-2020].

[45] ElasTest Developers Team, "E2E Test Dataset." 2020.