Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Grado en Ingeniería Informática

Pruebas automáticas con Selenium sobre Docker Windows Containers

Ubicación del código fuente:
https://github.com/sglezcasco/sgc-tfgii

Trabajo Fin de Grado

presentado por: González Casco, Sergio

Director/a: Rouyet Ruiz, Néstor

Ciudad: Madrid
Fecha: 19 de marzo de 2018

Resumen

En el desarrollo de una página web es necesaria una fase de pruebas para validar la funcionalidad y el cumplimiento de los requisitos iniciales establecidos. Además, debe de realizarse pruebas en los diferentes sistemas operativos y navegadores sobre los que vaya dirigida la página web, bien de forma secuencial o en paralelo. Para la ejecución de forma paralela sobre un mismo equipo de escritorio es necesario apoyarse en sistemas de virtualización o contenedores.

Se ha desarrollado un proyecto de automatización de pruebas sobre una web de ejemplo, ejecutadas en paralelo por diferentes sistemas operativos y navegadores, embebidos en un computador de escritorio Windows. Para la automatización nos apoyamos en la librería Selenium, y para la virtualización en las tecnologías Docker e Hyper-V.

Palabras Clave: Pruebas Contenedores Docker Selenium Windows

Abstract

In the development of a web page, a testing phase is necessary to validate the functionality and compliance with the initial requirements established. In addition, tests must be carried out in the differents operating systems and browsers to which the web page is directed, either sequentially or in parallel. For parallel execution on the same desktop it is necessary to use on virtualization systems or containers.

A project of automation of tests has been developed on an example web, executed in parallel by different operating systems and browsers, embedded in a Windows desktop computer. For automation we will use the Selenium library, and for virtualization in Docker and Hyper-V technologies.

Keywords: Testing Containers Docker Selenium Windows

Índice de figuras

Índice de tablas

Índice de contenidos

1. Introducción

En la actualidad, desarrollar una página web y ponerla en funcionamiento, está al alcance de cualquier persona que tenga unos mínimos conocimientos de informática, gracias a la cantidad de herramientas y servicios que se ofrecen para tal fin.

Ahora bien, una vez la página web esté desarrollada, será necesario realizar unas pruebas funcionales para validar su correcto funcionamiento, para posteriormente ponerla en producción.

Esa validación y verificación de la funcionalidad, deberá realizarse desde los distintos dispositivos hacia los que esté dirigida esa página web. Si definimos que dicha web estará destinada a ser utilizada desde PCs de escritorio, al menos habrá que tener en cuenta los principales sistemas operativos: Windows, Linux, Mac…, así como los diferentes navegadores web que se puedan utilizar en ellos.

Si estuviera destinada al acceso desde dispositivos móviles, de igual modo habrá que tener en cuenta sus principales sistemas operativos: Android, iOS... y los navegadores o aplicaciones móviles que se vayan a utilizar para acceder a nuestra página web.

Para cubrir todos estos diferentes escenarios y poder ejecutar los tests, disponemos de varias opciones:

- Ejecutar los tests de forma local en nuestro computador; requerirá de configuración de diferentes máquinas virtuales y/o contenedores, que incluyan los diferentes sistemas operativos y navegadores.

- Ejecutar los tests de forma remota, contra diferentes servidores propios configurados con los diferentes sistemas operativos y navegadores.

- Ejecutar los tests de forma remota contra servicios ofrecidos por terceros, las llamadas granjas de dispositivos, que además conllevan un coste económico a considerar.

1.1 Objetivo y Alcance

El objetivo de este trabajo es centrar el foco en la ejecución en paralelo de los tests necesarios para validar la funcionalidad de una página web concreta sobre entornos de escritorio Windows, ahorro tiempo y dando un buen feedback al desarrollador.

Se ha tomado la decisión que la página web de ejemplo está dirigida para el acceso mediante PCs de escritorio con sistema operativo Windows. Dado que es actualmente es el sistema operativo de escritorio con mayor cuota de mercado, con mucha diferencia sobre el resto. El más utilizado concretamente es su versión Windows 10, como podemos a continuación en el siguiente gráfico:

OS Platform Statistics

2018	Win10	Win8	Win7	Vista	WinXP	Linux	Mac	Chrome OS	Mobile
January	41.6%	7.9%	26.3%	0.1%	0.5%	5.7%	10.0%	1.3%	7.7%

Figura 1. Estadísticas utilización sistemas operativos. IEEE. Enero 2018.

Además, nos interesa realizar nuestras pruebas sobre los navegadores con mayor cuota de uso en el mercado, siendo recomendable realizarlo sobre sus últimas versiones estables. A continuación, mostramos un gráfico con las estadísticas de uso de los principales navegadores web existentes en el mercado:

2018	Chrome	Edge/IE	Firefox	Safari	Opera
January	77.2 %	4.1 %	12.4 %	3.2 %	1.6 %

Figura 2. Estadísticas utilización navegadores web. IEEE. Enero 2018.

Posteriormente, nos centraremos en mostrar las ventajas y desventajas del uso de contenedores Docker sobre el uso de máquinas virtuales. Dado que la tecnología Docker nació para virtualizar contenedores con sistemas operativos tipo Linux, dejaremos Linux a un lado y pondremos el punto de mira en la virtualización de contenedores con sistemas operativos Windows, cuya funcionalidad ha sido liberada en las últimas versiones de la herramienta Docker.

Así pues, partiendo de una página web de ejemplo, diseñaremos un conjunto de tests sobre ella, apoyándonos en una librería para la automatización de pruebas web, llamada Selenium.

Sobre nuestro sistema operativo Windows crearemos múltiples nodos remotos de ejecución con diferentes configuraciones (definiremos una configuración por navegador web). Y por último ejecutaremos esos tests de forma paralela contra esos nodos virtuales, sobre nuestro sistema operativo Windows 10.

Por último, nos apoyaremos en ciertas herramientas para gestión de contenedores y máquinas virtuales y herramientas de gestión de tareas, como son Docker Compose, Microsoft Powershell y Jenkins.

Queda fuera del alcance de proyecto, la ejecución de tests sobre contenedores Linux desde un PC Windows, puesto que no es algo novedoso y existe bastante información al respecto.

También queda fuera del alcance de este proyecto, la automatización de tests sobre dispositivos móviles, implicaría más piezas para la integración con simuladores y emuladores de dispositivos móviles (smartphones, tablets, smartwatches…).

1.2 Estructura del documento

En este apartado se describe como está estructurado el presente documento:

Sección 1. Introducción

En esta primera sección se lleva a cabo una pequeña introducción, para definir el objetivo y el alcance del proyecto.

Sección 2. Contexto y estudio preliminar

En esta sección se sitúa el proyecto en un contexto, se analiza el dominio de aplicación, se identifican las herramientas y tecnologías existentes en el mercado, y su uso en el proyecto.

Sección 3. Identificación de requisitos

En esta sección se identifican y describen los requisitos tanto funcionales como no funcionales del proyecto. Además, se describen los casos de uso.

Sección 4. Diseño de la solución

En esta sección se expone la arquitectura de la solución a alto nivel.

Sección 5. Descripción técnica de la solución

En esta sección se describen con detalle las diferentes piezas del proyecto, así como su integración para llegar a la solución.

Sección 6. Evaluación de la solución

En esta sección se valida si la solución cumple con los requisitos iniciales.

Sección 7. Conclusiones y línea de trabajo futura

En esta sección recogen las conclusiones obtenidas y planes de mejora futuras.

Sección 8. Bibliografía

En esta sección se incluyen las fuentes consultadas para el desarrollo del proyecto.

2. Contexto y estudio preliminar

En esta sección se describe el dominio de aplicación, se identifican las herramientas y tecnologías existentes en el mercado, y su posible uso en el proyecto.

2.1 Dominio de aplicación

Este proyecto se enmarca en el ciclo de vida de construcción del Software, concretamente en la fase de evaluación y validación que se incluye en ciertos modelos del ciclo de vida del Software, como en:

- Modelo en cascada
- Modelo en V
- Modelo en espiral o evolutivo
- Modelo iterativo
- Modelo basado en prototipos
- Modelo incremental
- Programación extrema

Todos ellos coinciden en contener una fase de testing llamada bien; fase de pruebas, de validación o evaluación, en la cual se testea la aplicación objeto.

Se puede afinar más el dominio de aplicación, puesto que, si estos tests se ejecutan de forma automática y se ejecutan con cada cambio/evolución introducido en la aplicación, estaremos utilizando un modelo de integración continua, que puede evolucionar hasta un modelo de entrega continua, en el cual cada cambio introducido en la solución se incluye, se compila, se empaqueta, se redespliega la aplicación, se prueba y se actualiza en producción de manera automatizada, sin intervención humana.

2.2 Herramientas y tecnologías del dominio

En esta sección se describen los lenguajes de programación, herramientas y tecnologías existentes en el dominio de aplicación, que podrían ser utilizados en este proyecto.

2.2.1 Lenguajes de programación

En la actualidad existe un gran número de lenguajes de programación de alto nivel que podremos utilizar para codificar nuestros proyectos de desarrollo.

Dichos lenguajes pueden basarse en diferentes paradigmas de programación:

- Paradigma imperativo: Describe los pasos para resolver una tarea como una secuencia de instrucciones que altera el estado de un programa. De este modo se describe cómo debe solucionarse un problema mediante instrucciones en un determinado orden.

- Paradigma declarativo: Contrario al paradigma imperativo. Describe la solución del problema sin indicar explícitamente los pasos para poder alcanzarla. Dentro de esa categoría se encuentran:
  - Paradigma lógico: se apoya en la lógica de predicados para llegar a la solución del problema.
  - Paradigma funcional: se apoya en la definición de funciones y/o procedimientos, que en función de unos valores de entrada devuelve unos valores de salida.

- Paradigma orientado a objetos: La resolución del problema se apoya en la creación de objetos y su comunicación. Los objetos son entidades con atributos y comportamientos propios, que representan elementos del problema a resolver.
  Se basa también en conceptos como la abstracción de datos**,** la encapsulación**,** modularidad**,** la herencia y el polimorfismo**.**

Los lenguajes de programación pueden cumplir uno o varios paradigmas de programación.

A continuación, se muestra un estudio de una clasificación realizada por el instituto IEEE[1] sobre la cuota de utilización en el mercado de los lenguajes de programación para el desarrollo de aplicaciones web durante el año 2017:

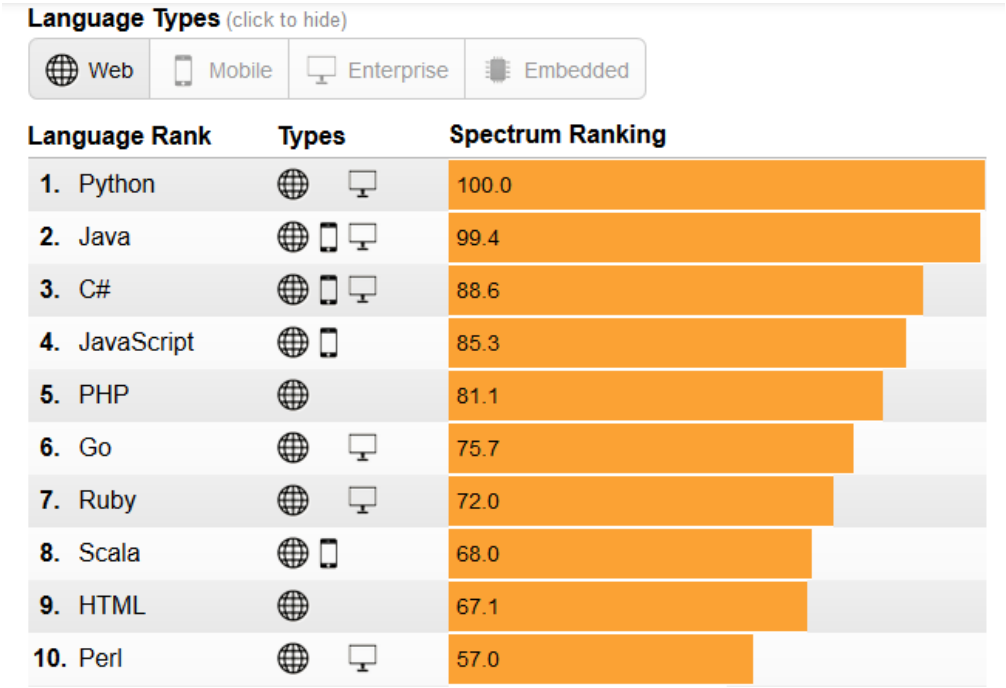

Figura 3. Clasificación uso lenguajes programación desarrollo web en año 2017

Describiremos de forma breve los lenguajes más utilizados en el mundo web:

- **Python**: Es un lenguaje interpretado (no necesita ser compilado), multiplataforma y multiparadigma, puesto que soporta los paradigmas de orientación a objetos, imperativo y funcional. Gracias a su sintaxis fácil es una buena opción para nuevos programadores. En gran auge en los últimos años.

- **Java**: Es uno de los lenguajes de programación más populares y utilizados. Creado por "Sun MicroSystems". Es un lenguaje multiplataforma, orientado a objetos, portable, concurrente y de propósito general.

[1] Instituto IEEE: https://www.ieee.org/standards/index.html

- **C#:** Llamado C Sharp, es un lenguaje de programación orientado a objetos desarrollado por "Microsoft" para ser empleado en la plataforma .NET. Es la evolución de los lenguajes de programación C y C++.

- **JavaScript**: Desarrollado por "Netscape Communications", es un lenguaje interpretativo, basado en prototipos. Históricamente ha sido usado para implementar funciones en el lado del cliente, pero también puede ser utilizado en el lado del servidor.

- **PHP**: Hypertext PreProccesor. Se trata de un lenguaje de script interpretado en el lado del servidor, utilizado para la generación de páginas web dinámicas, embebidas en páginas HTML y ejecutadas en un servidor.

- **Go**: Creado por "Google", es un lenguaje de programación compilado**,** concurrente**,** imperativo**,** estructurado y no orientado a objetos. Inspirado en C.

- **Ruby**: Es un lenguaje interpretado, dinámico para programación orientada a objetos y de código abierto, que está enfocado en la simplicidad y la productividad.

- **Scala**: Scala es un lenguaje de programación moderno multi-paradigma diseñado para expresar patrones de programación comunes de una forma concisa, elegante, y de tipado seguro. Integra fácilmente características de lenguajes orientados a objetos y funcionales.

- **HTML**: Acrónimo de "HyperText Markup Language," Es un lenguaje estático para el desarrollo de sitios web desarrollado por el World Wide Web Consortium (W3C[2]). Es el lenguaje "natural" sobre el que se construyen todas las páginas web e incluso muchas apps en dispositivos móviles.

- **Perl**: Acrónimo de "Practical Extraction and Report Language". Es un lenguaje de propósito general, multiparadigma, fácil de usar. Destaca por su poderoso sistema de procesamiento de texto.

[2] W3C: Comunidad internacional para el desarrollo de estándares web. https://www.w3c.es/

2.2.2 Software automatización de pruebas web

En la actualidad existe un gran número de herramientas y librerías en el mercado, para la automatización de pruebas sobre páginas webs. Vamos a realizar un pequeño análisis sobre las más populares:

- **Selenium**: es la herramienta de código libre más popular para automatización de pruebas sobre aplicaciones web. Permite programación avanzada y scripting.
  - Puede ser utilizada en multitud de lenguajes de programación (Java, Groovy, Python, C#, PHP, Ruby y Perl).
  - Es multiplataforma (Windows, Mac y Linux)
  - Incluye soporte para múltiples navegadores (Chrome, Firefox, IE, Edge, Safari, Opera, incluso modos de ejecución headless)
  - Incluye herramientas para grabación (Selenium IDE), para ejecución remota (Selenium Grid) e integración con otros frameworks (Appium)
  - Es la base de otras herramientas de automatización de pruebas web.

- **Watir**: es una herramienta de automatización de pruebas webs open source, creada sobre Ruby.
  - Es compatible con metodologías BDD (desarrollo dirigido por comportamiento)
  - Se integra con herramientas como RSpec, Cucumber y Test/Unit.
  - Incluye soporte para múltiples navegadores (Chrome, Firefox, IE, Edge, Safari, Opera, incluso modos de ejecución headless)

- **HP UFT**: HPE Unified Functional Testing. (Antiguo HP QTP: QuickTest Professional). Es una herramienta comercial, desarrollada por “Hewlett Packard” que ofrece automatización de pruebas funcionales y de regresión.
  - Utiliza Visual Basic como lenguajes de scripting.
  - Se integra con otras herramientas de la suite HP (como Quality Center)
  - Soporta pruebas sobre apps móviles, webs y aplicaciones de escritorio.

- **TestComplete**: Herramienta comercial, desarrollada por “SmartBear”.
  - Ofrece una interfaz gráfica de usuario para gestionar las pruebas.
  - Soporta multitud delenguajes de scripting como: JavaScript, Python, VBScript, JScript, DelphiScript, C++Script y C#Script.
  - Incluye herramientas de grabación, ejecución y visualización de tests.
  - Soporta pruebas sobre apps móviles, web y aplicaciones de escritorio.

- **IBM RFT**: IBM Rational Functional Tester. Es una herramienta comercial creada por IBM, para pruebas funcionales y de regresión.
  - Soporta aplicaciones realizadas en .Net, JAVA, SAP, Flex y AJAX.
  - Utiliza JAVA y VBScript como lenguaje de scripting.
  - Aporta herramientas de grabación, visualización y ejecución de pruebas.
  - Integración con otras herramientas de la suite de pruebas de IBM (como IBM Rational Quality Manager)
  - Es compatible con metodologías TDD (desarrollo dirigido por pruebas)
  - Soporta pruebas móviles, web y aplicaciones de escritorio.

- **SauceLabs**: Es una plataforma de pruebas en la nube, ofrecida por terceros, para ejecución de pruebas automatizas sobre aplicaciones web y móviles.
  - Soporta pruebas sobre todo tipo y versiones de navegadores web, sistemas operativos, emuladores y simuladores móviles.
  - Proporciona infraestructuras para pruebas con Selenium y Appium.
  - Pruebas sobre dispositivos móviles reales.
  - Permite pruebas en paralelo, con buena escalabilidad.

- **Cucumber**: es una herramienta open source para implementar metodologías BDD (desarrollo dirigido por comportamiento), que permite ejecutar pruebas de software automatizadas, basadas en descripciones en un texto plano.
  - Se escriben en un lenguaje específico de dominio, llamado Gherkin.
  - Está escrito en Ruby, pero puede ser utilizado por otros lenguajes, incluyendo Java, C#, Python, .NET, Flex.
  - Las pruebas se escriben antes de iniciar el desarrollo, por los analistas de negocio.

2.2.3 Software virtualización de máquinas

Estas herramientas nos permiten instalar varios sistemas operativos sobre un mismo hardware, lo que comúnmente llamamos máquinas virtuales. En la actualidad, existen numerosas herramientas para virtualizar sistemas operativos.

- **VMWare Workstation**: la principal herramienta en sistemas de virtualización. Válida tanto para ordenadores de escritorio como sistemas de servidores. Aunque siempre ha sido de pago, ahora ofrece también una versión libre para uso doméstico. Sus principales características son:
  - Virtualización completa: sistemas operativos Windows, Linux, Mac.
  - Virtualización de hardware asistido
  - Migraciones en caliente
  - Conversión P2V: máquinas físicas hacia máquinas virtuales.
  - Medidas e informes de rendimiento.
  - Control de energía
  - Alertas en tiempo real
  - Almacenamiento fino
  - Restauración y copia de seguridad de las máquinas virtuales
  - Migraciones de máquinas virtuales

- **Citrix XenServer**: Está basado en software open source. Sus principales características son:
  - Virtualización de hardware asistido
  - Migración en caliente
  - Informes de rendimiento
  - Almacenamiento fino
  - Capacidad de realizar instantáneas.
  - Dos versiones: una de pago y otra libre.

- **Virtual Box**: equivalente gratuito de VMWare para virtualizar sistemas operativos Windows, Linux o Mac. Es propiedad de Oracle. El uso de algunas características requiere disponer de licencia de pago.
    - Virtualización completa: sistemas operativos Windows, Linux, Mac y Solaris.
    - Compatible con máquinas virtuales VMware.
    - Puede controlarse a través de símbolo de sistema.
    - Cuenta con herramientas especiales de compartición de archivos.
    - Permite crear instantáneas para restaurar el estado anterior de una máquina virtual fácilmente.
    - Soporte limitado para gráficos 3D.
    - Cuenta con una herramienta de captura de vídeo.
    - Cifrado de unidades y soporte para puertos USB 2.0 y 3.0 (con licencia).

- **Microsoft Hyper-V**: es el sistema de virtualización de Microsoft.
    - Capaz de virtualizar los sistemas Microsoft y algunos de los sistemas Linux más comunes: Ubuntu, Suse, RedHat, centOS y Fedora.
    - Funcionamiento bajo licencia.
    - Puede controlarse mediante scripts en Powershell.

- **Parallels**: es el sistema de virtualización más extendido para crear máquinas virtuales en Mac. Sus principales características son:
    - Al saltar a la máquina virtual, la máquina host renuncia automáticamente a la potencia de procesamiento.
    - Tiene portapapeles sincronizado, carpetas compartidas y soporte para impresoras y periféricos.

Debido a la constante evolución tecnológica surgida en los últimos años con la aparición de la computación en la nube, surgieron nuevas tecnologías que se consideran una evolución a las máquinas virtuales, son los llamados contenedores.

Para diferenciar máquinas virtuales de contenedores, vamos a describir internamente como funcionan ambos, para ver sus similitudes y diferencias.

Las máquinas virtuales consisten en un hipervisor que sobre un hardware físico permite emular varias máquinas virtuales cada una con su propio sistema operativo, librerías y aplicaciones, con esto en un servidor físico se pueden tener varios sistemas operativos y aplicaciones, reduciendo costes, con un provisionamiento más rápido y una mejor recuperación ante fallos, aunque con un consumo elevado en recursos de computación, procesador, memoria y almacenamiento.

Los contenedores son una nueva tecnología que busca lo mismo, pero con la gran diferencia que además del servidor físico, se tiene un solo sistema operativo host compartido para todos los contenedores. Gracias a esto el tamaño de los contenedores es mucho menor, haciéndolos más eficientes, más fácil de migrar, iniciar, recuperar y mover entre nube pública y privada.

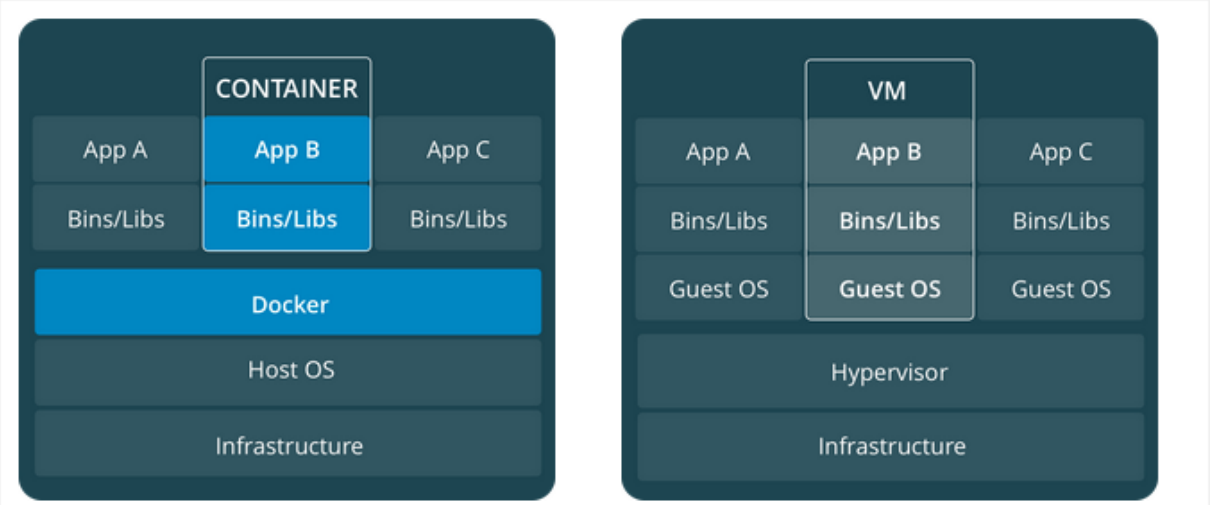

Figura 4. Diferencia Contenedores VS Máquinas Virtuales

Un ejemplo de esta tecnología es la herramienta Docker[3], la cual está revolucionando la forma de virtualizar sistemas y servicios, en comparación a los sistemas tradicionales como VMware, Virtual Box, etc.

[3] Docker: https://www.docker.com/

Docker es un proyecto de código abierto con el que fácilmente podremos crear "contenedores". Las principales características de estos contenedores son:

- *Rapidez*: Los contenedores son capaces de compartir un solo núcleo y bibliotecas de aplicaciones, esto ayuda a que los contenedores se carguen antes que las máquinas virtuales. Los procesos son mucho más ligeros y se reaprovecha el hardware. Además, no es necesario el sistema de archivos completo del sistema operativo invitado con lo que Docker únicamente usa una fracción de espacio de almacenamiento necesario en la virtualización.

- *Portabilidad*: Las imágenes Docker se pueden desplegar en diferentes sistemas, el único requisito es que la herramienta Docker esté instalada en el sistema huésped. Además, se pueden desplegar multitud de contenedores en un mismo equipo físico.

- *Seguridad*: Los contenedores carecen de seguridad. Lo que se ve como una desventaja frente a las máquinas virtuales.

- *Administración*: Existen dificultades en la gestión de contenedores Docker, existen soluciones tales como Docker Swarm o Kubernetes que hacen un poco más fácil la gestión de múltiples contenedores.

- *Autosuficiencia*: Un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga. Además, las aplicaciones desplegadas en un contenedor están libres de dependencias instaladas en el sistema anfitrión.

- *Multiplataforma*: Docker está disponible para Ubuntu, ArchLinux, Gentoo, Fedora, OpenSUSE y FrugalWare y permite desplegar contenedores bajo entornos Windows, Mac, Amazon EC2, Rackspace o Google Cloud**.**

Además, Docker también se compone de herramientas para construir imágenes, gestionarlas y un repositorio para compartirlas.

Existen diferentes alternativas a Docker, entre ellas las más populares son:

- **rkt**: es el entorno de ejecución rkt del distribuidor de Linux CoreOS, es el mayor competidor de Docker en el mercado de la virtualización basada en contenedores. A diferencia de Docker, rkt se ejecuta en el modelo de proceso clásico de Unix, en un entorno autónomo y aislado. Además, incluye otras características:

    - Implementa un formato de contenedor estándar abierto y puede ejecutar otras imágenes de contenedor, como las creadas con Docker.
    - Existen muchas menos integraciones de proveedores externos para rkt que para Docker.

- **LXC y LXD**: LXD es un hipervisor para contenedores Linux desarrollado por "Canonical Ltd". Es un complemento para los contenedores de Linux (LXC) que facilita su uso y añade nuevas posibilidades. Son la siguiente alternativa a los contenedores Docker.

    - LXC ofrece a los usuarios de Linux una forma sencilla de crear y administrar contenedores para aplicaciones y sistemas.
    - LXC permite ejecutar diferentes sistemas Linux (contenedores) aislados en una máquina Linux
    - LXC y LXD se desarrollan bajo la tutela de "LinuxContainers.org"
    - Un contenedor LXC es como un *servidor virtual* completo, con todos los servicios que esperarías encontrar en una máquina Linux.
    - LXC es una tecnología de Linux. No funciona ni en Windows ni en Mac.

2.2.4 Software gestión de tareas

Existen diferentes herramientas y tecnologías para la ejecución automática de tareas, en nuestro caso de ejecución automática de pruebas. Nos apoyaremos en herramientas de integración continua.

La integración continua es una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo frecuentemente, dando lugar a múltiples cambios en el software por día. Cada integración incluye los últimos cambios en el proyecto, es verificada mediante una compilación automatizada, y ejecuta automáticamente pruebas con el fin de detectar errores lo más pronto posible.

En la actualidad, existen numerosas herramientas para realizar integración continua:

- **Hudson**: es una herramienta de integración continua escrita en Java, que se ejecuta en un contenedor de servlets, como Apache Tomcat o el servidor de aplicaciones GlassFish. Es compatible con las herramientas SCM y se puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como también scripts de shell y comandos de Windows. Actualmente requiere licencia y es propiedad de “Oracle”.

- **Jenkins**: es una herramienta de integración continua open source escrita en Java, creada como un fork a partir de Hudson, tras su compra por Oracle. Jenkins proporciona servicios de integración continua para el desarrollo de software. Es un sistema basado en servidor que se ejecuta en un contenedor de servlets como Apache Tomcat. Es compatible con las herramientas de SCM y se puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y comandos de Windows. Contiene además gran cantidad de plugins que aumenta su funcionalidad y flexibilidad.

- **Go**: es una herramienta open source que se utiliza en el desarrollo de software para lograr la entrega continua de software, creada por la empresa “ThougthWorks”. Es compatible con la automatización de todo el proceso de compilación y prueba, desde el registro de código hasta la implementación. Ayuda a producir software en ciclos cortos y asegura que el software se puede lanzar confiablemente en cualquier momento. Soporte a la herramienta previo pago.

- **Travis CI**: es un servicio de integración continúa distribuida y alojada, open source, que se utiliza para construir y probar proyectos alojados en GitHub. Travis CI se configura añadiendo un archivo llamado ".travis.yml" (que es un archivo de texto en formato YAML) al directorio raíz del repositorio de GitHub. Necesita licencia bajo pago para extender la funcionalidad a partir de 100 compilaciones.

- **Bamboo**: es un servidor de integración continua de Atlassian, los creadores de JIRA, Bitbucket, Confluence y Crowd. Admite compilaciones en cualquier lenguaje de programación utilizando cualquier herramienta de compilación, incluidas Ant, Maven, make y cualquier herramienta de línea de comandos. Las notificaciones de compilación se pueden personalizar según el tipo de evento y se pueden recibir por correo electrónico, mensaje instantáneo, redes sociales, etc… Existe una versión libre con un período de evaluación de 30 días.

- **TeamCity**: es una herramienta de gestión de compilación basada en Java y un servidor de integración continua de JetBrains. Es una poderosa herramienta de integración continua. La versión libre es bastante limitada con un límite de 20 configuraciones.

- **GitLab CI**: forma parte de GitLab, una aplicación web con una API que almacena su estado en una base de datos. Gestiona proyectos y proporciona una interfaz de usuario agradable, además de todas las características de GitLab. GitLab Runner es una aplicación que procesa compilaciones. Se puede implementar por separado y funciona con GitLab CI a través de una API. Para ejecutar pruebas, necesita al menos una instancia de GitLab y un GitLab Runner.

- **Circle CI**: es un servidor de integración continua basado en la nube que admite Ruby on Rails, Sinatra, Node, Python, PHP, Java y Clojure. Es una solución de pago, alojada, diseñada para reducir la sobrecarga posible del proceso de prueba.

Existen muchas más herramientas de integración continua, como pueden ser: Semaphore, Continuum, CruiseControl, CodeShip, Microsoft Team Foundation Server, etc..

2.3 Conclusiones

Existen gran variedad en cuanto a elección de lenguajes de programación, herramientas de pruebas automáticas, herramientas de virtualización y de gestión de tareas.

En cuanto a los lenguajes de programación nos interesa su conocimiento, portabilidad y facilidad de codificación. Aquí destacamos **Java** y **Python**. Dado que vamos a trabajar sobre entornos Windows también se podría incluir C#, pero lo descartamos por si en el futuro queremos ampliar el alcance de este proyecto a plataformas Linux. Elegiremos Java como lenguaje de programación por tener de una mayor experiencia en la codificación de desarrollos software.

En cuanto a las herramientas de automatización de pruebas nos interesan herramientas libres sin costo, que se puedan integrar de forma fácil con los lenguajes de programación elegidos, en este aspecto, nos encajan tanto **Selenium** como **Cucumber**. Dado que nuestras pruebas están más dirigidas a pruebas funcionales que pruebas de aceptación, nos centraremos en el primero.

En cuanto a la virtualización, nos decantaremos por el uso de contenedores en vez de máquinas virtuales, para reutilizar el núcleo del sistema operativo host (en nuestro caso Windows), y por la capacidad de virtualización en sistemas Windows, elegiremos **Docker**, que nos permitiría en un futuro añadir la virtualización sobre plataformas Linux. Ahora bien, las imágenes Docker proporcionadas por Microsoft carecen de soporte de interfaz gráfica de usuario, por lo que para la ejecución sobre navegadores reales necesitaremos crear máquinas virtuales Windows, que elegiremos Hyper-V dado que viene por defecto en el sistema operativo Windows 10 y utiliza la misma tecnología de Virtualización que Docker sobre Windows. Por ello, iremos a un **modelo mixto** formado por contenedores **Docker** y máquinas virtuales administradas mediante la **Microsoft Hyper-V Manager**.

En cuanto a las herramientas de gestión de tareas o de integración continua, nos interesan herramientas libres o con una versión de prueba que sea suficiente para nuestros interesas, con lo cual destacamos **Jenkins**, **TeamCity, Gitlab CI y Go CD**. Habrá que tener en cuenta también que sea capaz de integrarse con Docker. En este aspecto nos decantaremos por la flexibilidad y la extensión de la funcionalidad que nos ofrece Jenkins a partir de amplio catálogo de plugins disponibles de forma gratuita.

3 Identificación de requisitos

En esta sección se describirán tanto los requisitos funcionales, como no funcionales del proyecto, que fueron definidos en una fase inicial del proyecto de captura, análisis y especificación de requisitos.

3.1 Especificación de requisitos

A continuación, se describirán tanto los requisitos funcionales como los requisitos no funcionales, que se han identificado inicialmente para poder llevar a cabo el desarrollo de la solución. También se va a describir la metodología seguida para la gestión del proyecto y las herramientas de apoyo utilizadas en el desarrollo del software.

3.1.1 Requisitos funcionales

A continuación, se muestra el conjunto de requisitos funcionales, estos van a ser organizados en forma de tabla con los siguientes campos:

- Código de identificación del requisito
- Nombre descriptivo del requisito
- Descripción detallada del requisito
- Autor
- Precondiciones necesarias del requisito
- Postcondiciones del requisito
- Prioridad
- Observaciones

Identificador	**RF-001**
Nombre	Creación de página web de ejemplo
Descripción	Se deberá desarrollar una página web como base para el proyecto que contenga diferentes secciones como login, página principal y logout. No es necesario dotar a la web de tecnología de servidor. Para la capa cliente nos apoyaremos en HTML, CSS y Javascript, y en la librería de Bootstrap 4.0. Es recomendable alojar la página web en un host para asegurar su disponibilidad.
Precondición	Ninguna.
Postcondición	La web debe ser accesible desde Internet.
Autor	Sergio Gonzalez.
Prioridad	Alta.
Observaciones	Se recomienda el desarrollo de una página web propia de pruebas para evitar posibles problemas de indisponibilidad web o tener que readaptar nuestra solución ante posibles cambios en la web.

Identificador	**RF-002**
Nombre	Creación de proyecto de automatización de pruebas
Descripción	Se deberá desarrollar un proyecto de automatización de pruebas en lenguaje Java, basado en Maven y apoyándose en las librerías de Selenium y TestNG. Así mismo debería seguir la metodología de PageObjetcs donde cada página es representada como un objeto con sus métodos y atributos. Disponer de una serie de tests para probar la funcionalidad de la página web en entorno local.
Precondición	Página web creada en el RF-001
Postcondición	Posibilidad de ejecutar los tests de forma local sobre algún navegador.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	Se recomienda el desarrollo utilizando las últimas versiones de Selenium y TestNG.

Identificador	**RF-003**
Nombre	Ejecución de tests sobre diferentes navegadores
Descripción	Se deberá ampliar la funcionalidad del proyecto, para que las pruebas sean ejecutadas sobre las últimas versiones de los siguientes navegadores: • Microsoft Edge • Google Chrome • Mozilla Firefox • PhantomJS Sobre entornos Windows 10.
Precondición	Proyecto de automatización de pruebas creado en el RF-002
Postcondición	Ejecución de pruebas sobre diferentes navegadores.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	Es necesario descargarse ciertos drivers para la integración del framework de Selenium con los navegadores citados anteriormente.

Identificador	**RF-004**
Nombre	Ejecución de tests de forma remota
Descripción	Adaptar la funcionalidad del proyecto para poder ejecutar los tests de forma remota, apoyándose en la tecnología Selenium Grid, donde habrá que configurar diferentes nodos con los diferentes sistemas operativos y navegadores sobre los que realizar las pruebas.
Precondición	Proyecto de automatización de pruebas ejecutado sobre varios navegadores, creado en el RF-003
Postcondición	La web debe ser accesible desde Internet.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	Se recomienda configurar un nodo independiente por cada navegador que vayamos a utilizar en pruebas.

Identificador	**RF-005**
Nombre	Creación de imágenes en Docker
Descripción	Se deberá crear imágenes Docker propias para el nodo servidor de Selenium Grid y para las imágenes de los navegadores headless, a partir de imágenes base proporcionadas por fabricantes En ellas será necesario configurar la instalación de Java, el tipo de instalación de Selenium Grid, el navegador correspondiente junto a su driver.
Precondición	Tener Docker para Windows, instalado en el pc de escritorio.
Postcondición	Disponer de un conjunto predefinido de imágenes Docker para el proyecto.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	Construir las nuevas imágenes Docker a partir de la imagen base liberada por Microsoft para Windows 10: Microsoft windoesservercore.

Identificador	**RF-006**
Nombre	Creación de 3 máquinas virtuales con Hyper-V Manager
Descripción	Se deberá crear máquinas virtuales Windows 10 para los nodos de ejecución de los navegadores reales. En ellas será necesario configurar la instalación de Java, el tipo de instalación de Selenium Grid, el navegador correspondiente junto a su driver.
Precondición	Proyecto de automatización de pruebas ejecutado sobre varios navegadores de forma remota, creado en el RF-004 y creación de imágenes Docker creadas en el RF-005.
Postcondición	Las pruebas automáticas son ejecutadas de forma remota contra los diferentes contenedores, y además en paralelo.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	La dirección url del servidor central del Selenium Grid, debe ser estática, para poder apuntar las pruebas siempre contra esa misma dirección IP:puerto.

Identificador	**RF-007**
Nombre	Integración de pruebas con contenedores Docker y máquinas virtuales
Descripción	Se deberá de realizar la integración para que las pruebas automáticas sobre la página se ejecuten de forma remota y en paralelo, a través de Selenium Grid en los diferentes contenedores iniciados a partir de las imágenes Docker creadas en el requisito RF-005 y diferentes máquinas virtuales creadas en el requisito RF-006.
Precondición	Proyecto de automatización de pruebas ejecutado sobre varios navegadores de forma remota, creado en el RF-004 y creación de imágenes Docker y máquinas virtuales creadas en el RF-005 y RF-006.
Postcondición	Las pruebas automáticas son ejecutadas de forma remota contra los diferentes contenedores y máquinas virtuales, y además en paralelo.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	La dirección url del servidor central del Selenium Grid, debe ser estática, para poder apuntar las pruebas siempre contra esa misma dirección IP.

Identificador	**RF-008**
Nombre	Gestión automática de contenedores y máquinas virtuales
Descripción	Se deberá desarrollar una tarea automática que, a partir de las imágenes y máquinas virtuales creadas, sea capaz mediante un script de iniciarlas Y una vez finalizadas las pruebas se puedan parar todos los contenedores. Se ha de realizar dicha tarea apoyándose en Docker, concretamente en Docker Compose y en scripts con Powershell.
Precondición	Deben de existir las imágenes Docker creadas en el RF-005.
Postcondición	Debemos ser capaces de iniciar y parar todos los contenedores y máquinas virtuales existentes a la vez.
Autor	Sergio Gonzalez.
Prioridad	Media
Observaciones	Ninguna

Identificador	**RF-009**
Nombre	Gestor de Tareas
Descripción	Se deberá configurar una herramienta de gestión de tares que permita iniciar los contenedores, ejecutar las pruebas automáticas de esos contenedores y una vez finalizadas pare los contenedores, para ello será necesario utilizar Jenkins y apoyarse en los plugins de Maven y la funcionalidad de ejecución por líneas de comandos que ofrece
Precondición	Tener realizada la integración de pruebas automáticas sobre contenedores Docker recogido en el RF-006 y disponer de scripts para la gestión de contenedores recogido en el RF-007.
Postcondición	La web debe ser accesible desde Internet.
Autor	Sergio Gonzalez.
Prioridad	Media
Observaciones	También se puede configurar que recoja el código fuente de algún sistema de configuración del software, como por ejemplo puede ser GitHub.

3.1.2 Requisitos no funcionales

De forma análoga, se muestran el conjunto de requisitos no funcionales, organizados en forma de tabla con los siguientes campos:

- Código de identificación del requisito
- Nombre descriptivo del requisito
- Descripción detallada del requisito
- Autor
- Precondiciones necesarias del requisito
- Postcondiciones del requisito
- Prioridad
- Observaciones

Identificador	**RNF-001**
Nombre	Manual de instalación y configuración
Descripción	Se deberá proporcionar un manual para la instalación y configuración de la herramienta Docker. Deberá detallar desde cómo crear/importar las imágenes, como iniciar ó parar los contenedores de forma manual. De igual modo, habrá que detallar como instalar y configurar Jenkins como herramienta de gestión de tareas. Por último, deberá de añadirse un manual de usuario para detallar como lanzar las pruebas automáticas contra nuestra página web desde dicha herramienta y de forma automática.
Precondición	Ninguna.
Postcondición	Manual disponible para usuarios.
Autor	Sergio Gonzalez.
Prioridad	Media
Observaciones	Se recomienda que el manual se entregue en formato PDF.

Identificador	**RNF-002**
Nombre	Compatibilidad con plataformas Windows 10
Descripción	La solución deberá ser compatible con sistemas operativos de escritorio Windows 10 Pro / Enterprise / Education, sobre arquitecturas de 64 bits, en las siguientes distribuciones: • Versión 1709, OS Build: 16299, Fecha liberación: 17/10/2017 • Versión 1703, OS Build: 15063, Fecha liberación: 11/04/2017 • Versión 1607, OS Build: 14393, Fecha liberación: 02/08/2016 Sobre las siguientes distribuciones **no** será compatible: • Versión 1511, OS Build: 10586, Fecha liberación: 12/11/2015 • Versión 1507, OS Build: 10240, Fecha liberación: 29/07/2015
Precondición	Ninguna.
Postcondición	Ninguna.
Autor	Sergio Gonzalez.
Prioridad	Alta
Observaciones	La mayor limitación viene impuesta por Docker, por su funcionalidad: Windows Docker Containers.

Identificador	**RNF-03**
Nombre	Ciclo de vida del software: Modelo iterativo e incremental
Descripción	El proyecto deberá seguir el modelo de desarrollo iterativo e incremental como ciclo de vida en la construcción de software, basado en la generación de prototipos e integración en la solución, por cada iteración.
Precondición	Ninguna.
Postcondición	Ninguna.
Autor	Sergio Gonzalez.
Prioridad	Media.
Observaciones	Ninguna.

3.1.3 Diagrama de casos de uso

A continuación, mediante el siguiente diagrama de casos de uso, se describe la interacción entre los diferentes actores con el conjunto de acciones realizadas en el sistema.

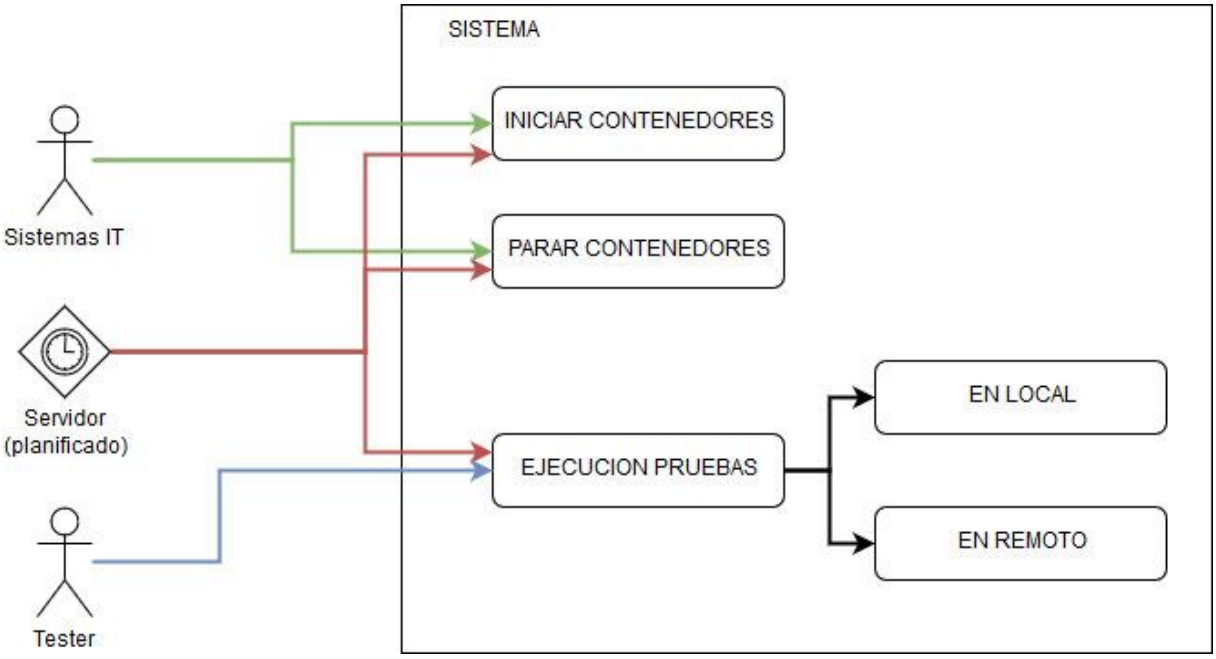

Figura 5. Diagrama de casos de uso del sistema

Desde el departamento de sistemas se invocaría manual o automaticamente las tareas de iniciar y parar los contenedores Docker involucrados.

Desde el departamento de testing, se lanzaría el proceso para ejecutar las pruebas automáticas, que bien podrían ejecutarse en local o de forma remota.

Por último, el propio servidor donde esté alojado el sistema de gestión de tareas podría invocar de forma programada las tareas de iniciar contenedores y máquinas virtuales, ejecución de pruebas automáticas y una vez finalizadas las pruebas, invocar la tarea de parar los contenedores y máquinas virtuales.

3.2 Gestión y planificación del proyecto

En esta sección se va a describir la metodología de desarrollo software a emplear durante el ciclo de vida del software, de este proyecto. Además, se mostrará la planificación inicial, así como las herramientas de apoyo utilizadas durante el desarrollo de este proyecto.

3.2.1 Metodología utilizada

Durante el desarrollo de este proyecto nos hemos apoyado en la metodología de desarrollo iterativo e incremental, que es una metodología ágil en la gestión y ejecución de proyectos.

Un modelo de desarrollo iterativo e incremental está basado en la ejecución en forma de iteraciones de un modelo en cascada: formado por las fases de análisis, diseño, codificación, pruebas e integración, hasta completar la funcionalidad del proyecto.

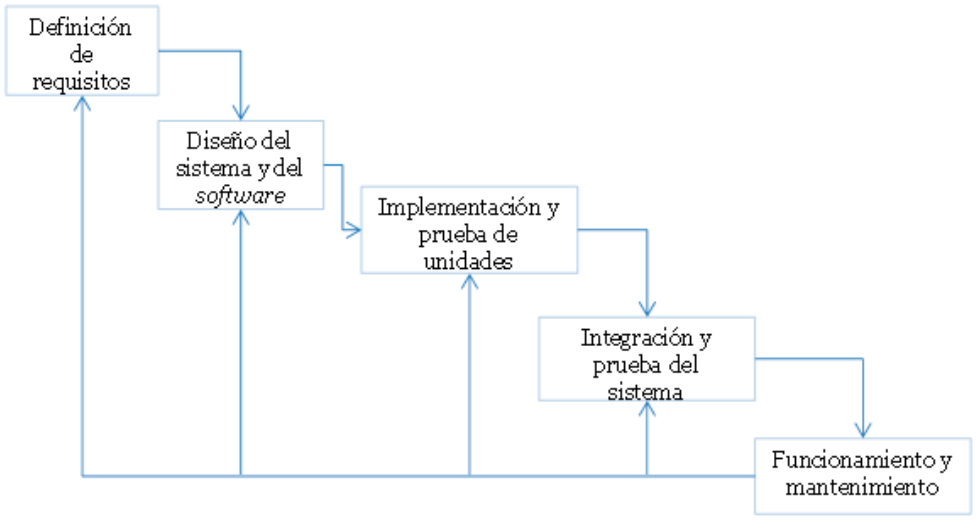

Figura 1.1. Modelo en cascada. (Adaptado de Sommerville, 2005).

Figura 6. Ciclo de vida en cascada[4]

[4] Figura 6 sacada del libro: Ingeniería del software, Ian Sommerville. Pearson Educación, 2005

En un desarrollo iterativo e incremental el proyecto se planifica en diversos bloques temporales, llamados iteraciones o ciclos de desarrollo.

En cada iteración, se construye una parte del proyecto y se reproduce un ciclo de vida en cascada completo. Al final de cada iteración se entrega un bloque del proyecto.

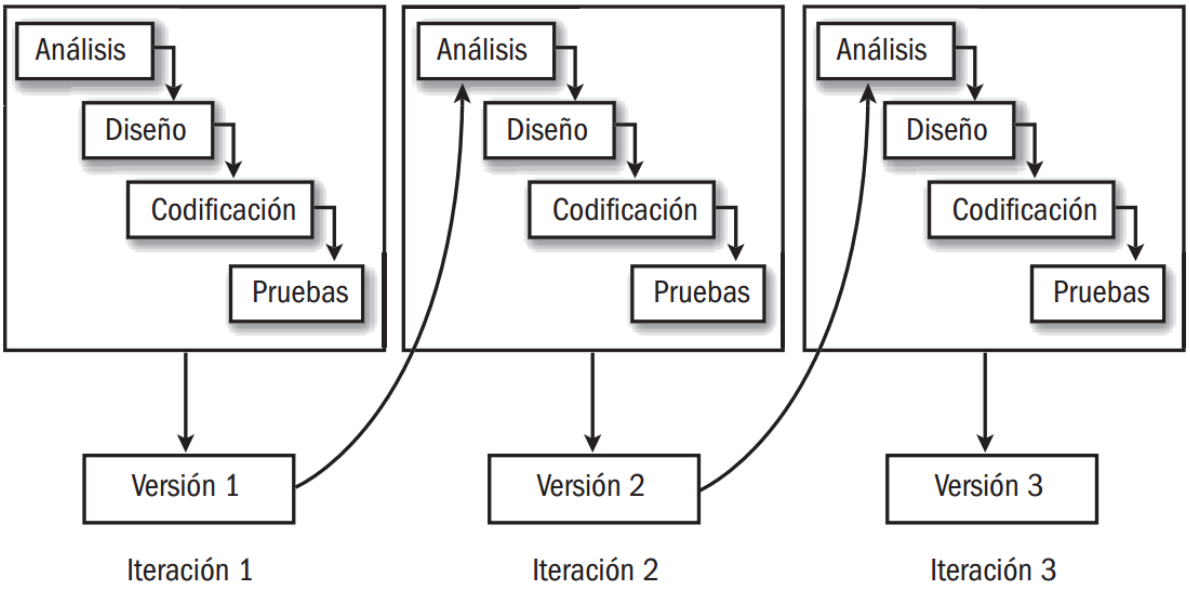

Figura 7. Ciclo de vida iterativo e incremental

En sucesivas iteraciones, el equipo evoluciona el proyecto entregado al final de la iteración anterior, bien añadiendo nuevas funcionalidades o mejorando las que ya fueron completadas.

Un aspecto fundamental para guiar el desarrollo iterativo e incremental es la priorización de los objetivos o requisitos en función del valor que aportan al cliente.

3.2.2 Planificación inicial y entregas

Cumpliendo la metodología de desarrollo iterativo e incremental, se muestra a continuación la planificación realizada y el desglose de funcionalidad entregada al final de cada iteración.

Desarrollo página web objeto	A	D	C	P	I					M					M				M					M					M				M
Desarrollo proyecto automatización de pruebas en local						A	D	C	P	I					M				M					M					M				M
Ampliación funcionalidad: Ejecución en remoto											A	D	C	P	I				M					M					M				M
Creación de imágenes Docker																A	D	C	P	I					M					M				M
Ampliación funcionalidad: Ejecución en remoto contra contenedores Docker																					A	D	C	P	I					M				M
Gestion de contenedores con Docker Compose																										A	D	C	P	I				M
Gestión de Tareas con Jenkins																															A	D	C	P	I

Figura 8. Hitos de entrega iterativa e incremental

Como se puede observar en el gráfico anterior, en ciertas iteraciones se entrega funcionalidad nueva que se va integrando al proyecto, en otras se amplían y mejoran ciertas funcionalidades.

A continuación, mostramos las fechas de entrega planificadas para cada iteración, las cuales definiremos con un ancho de 2 semanas:

- **Iteración 1**: Desarrollo página web objeto → 03/12/2017
- **Iteración 2:** Desarrollo proyecto pruebas automáticas en local →17/12/2017
- **Iteración 3**: Ampliación funcionalidad: Ejecución en remoto →31/12/2017
- **Iteración 4:** Creación de imágenes Docker →14/01/2018
- **Iteración 5:** Ampliación funcionalidad: Ejecución en remoto contra contenedores Docker → 28/01/2018
- **Iteración 6:** Gestión de contenedores con Docker Compose → 11/02/2018
- **Iteración 7:** Gestión de Tareas con Jenkins → 25/02/2018

3.2.3 Herramientas de ingeniería del software

Durante el desarrollo de este proyecto nos hemos apoyado en las siguientes herramientas:

- **Git**: es un sistema de control de versiones open source y gratuito diseñado por "Linus Torvalds", para manejar desde proyectos pequeños a muy grandes, con velocidad y eficiencia. Nos ayudará a gestionar los cambios que se realizarán sobre los elementos de nuestro proyecto. Se caracteriza por su rapidez, y su gestión eficiente y distribuida

- **GitHub**: GitHub aloja tu repositorio de código y te brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto. Se apoya en el sistema de control de versiones Git.

- **Eclipse**: es una plataforma de desarrollo. No tiene en mente un lenguaje específico, sino que es un IDE [5] genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java. Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones.

- **Notepad++**: es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación. De soporte nativo a Microsoft Windows. Se parece al Bloc de notas en cuanto al hecho de que puede editar texto sin formato y de forma simple, pero con más funcionalidades.

- **Bootstrap**: es un framework desarrollado y liberado por "Twitter", que tiene como objetivo facilitar el diseño web. Permite crear de forma sencilla webs de diseño adaptable, es decir, que se ajusten a cualquier dispositivo y tamaño de pantalla. Se apoya en HTML, CSS, Javascript y JQuery.

[5] IDE: Entorno de desarrollo integrado

4 Diseño de la solución

En esta sección, se describe la funcionalidad aportada por el proyecto de desarrollo que hemos realizado, haciendo hincapié en lo que añade y puede hacer.

4.1 Página web objeto de estudio

El primer paso de este proyecto es el desarrollo y puesta en producción de una página web, sobre la cual se realizarán las pruebas automáticas utilizando la librería de automatización Selenium y se ejecutarán en servidores virtuales Windows facilitados por las herramientas Docker e Hyper-V, sobre un pc de escritorio con sistema operativo Windows 10.

Se ha tomado esta decisión de crear una página web propia en vez de utilizar una existente por varias razones:

1. Utilizar una web externa puede suponer que nuestro proyecto no se pueda ejecutar con éxito, si dicha página web está indisponible, bien porque se haya dado de baja, se encuentre en período de actualización o en tareas de mantenimiento. Disponer una página web propia implica conocer cuándo van a ser dichos períodos de actualización y tareas de mantenimiento en la web.

2. Utilizar una web externa puede suponer que esté expuesta a cambios y que nuestro proyecto no se pueda ejecutar con éxito, si ha habido un cambio funcional en dicha web. Dichos cambios pueden ser muy variados, desde cambios en la url, en los campos de un formulario, en los elementos visuales, en el modo de ejecución de ciertos eventos o incluso la actualización de la web por una completamente distinta.

De hecho, alojaremos la página web en una plataforma de hosting de terceros, para obtener una mejora de la disponibilidad, puesto que estos servicios, realizan balanceo entre servidores en caso de indisponibilidad de un servidor web concreto.

En cuanto a la elección del servicio de hosting, la decisión será bastante sencilla, puesto que no vamos a utilizar tecnologías en la capa servidor, y nuestra web se tratará de una web “tonta” que realice navegaciones básicas, login, rellenar formulario, aceptar, navegar por alguna sección y desconexión. Para ello sólo necesitaremos un dominio y un host con acceso ftp para subir el código.

La implementación de la aplicación utilizará la librería Bootstrap 4.0 para crear contenido web adaptable a distintos tamaños de dispositivos (estilo responsive). Dicha librería se apoya en lenguaje HTML5, hojas de estilo CSS (concretamente CSS3) y lenguaje Javascript.

Se ha elegido cómo hosting la plataforma de servicios ofertada en “Arsys”, que contiene tanto servicio de asignación de dominio como servicio de hosting.

4.2 Descripción funcional del proyecto

Una vez tenemos desarrollada la página web que usaremos como base para nuestro proyecto, detallaremos qué se quiere obtener con este proyecto y realizaremos una descripción de sus funcionalidades.

Lo primero que tenemos que desarrollar es un proyecto de automatización de pruebas funcionales sobre dicha página web. Describiremos un conjunto de pruebas, que habrá que automatizar, y que sean configurables para ejecutar sobre diferentes navegadores en sus últimas versiones estables:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- PhantomJS (navegador headless)

Se ha descartado la ejecución en Microsoft Internet Explorer 11, puesto que ya se trata de un producto discontinuado. De igual modo también se ha descartado la ejecución sobre navegadores Opera y Safari, por su bajo índice de utilización en sistemas operativos Windows, que es el sistema operativo sobre el que se va a centrar y realizar el proyecto.

Una vez tengamos configurada la ejecución de las pruebas de forma automática en nuestro proyecto, podremos ejecutar dichas pruebas en nuestro equipo de forma secuencial. Si lo ejecutamos en paralelo, las pruebas se solapan y es muy probable que induzcan a errores.

Para poder ejecutar en paralelo todas las suites de pruebas y reducir los tiempos de pruebas, deberemos configurar nuestro proyecto para ejecución remota, aquí la primera solución sería apoyarse en máquinas virtuales y la tecnología Selenium Grid.

En principio vamos a definir los siguientes escenarios máquinas virtuales

- Sistema operativo Windows 10 y navegador Google Chrome
- Sistema operativo Windows 10 y navegador Mozilla Firefox
- Sistema operativo Windows 10 y navegador Microsoft Edge
- Sistema operativo Windows 10 y navegador headless PhantomJS

Se podrían definir máquinas virtuales con una configuración más compleja, pero nos centraremos en este escenario por simplicidad.

Además, se deberá montar un servidor que atienda las peticiones de nuestras pruebas y las dirija contra el nodo correspondiente. Esto se hace configurado un servidor mediante Selenium Grid y enlazando como nodos de ejecución las diferentes máquinas virtuales que hemos configurado. Ahora bien, falta un último cambio, apuntar nuestro proyecto para que se ejecute de forma remota contra la url donde está expuesta el servidor de Selenium Grid.

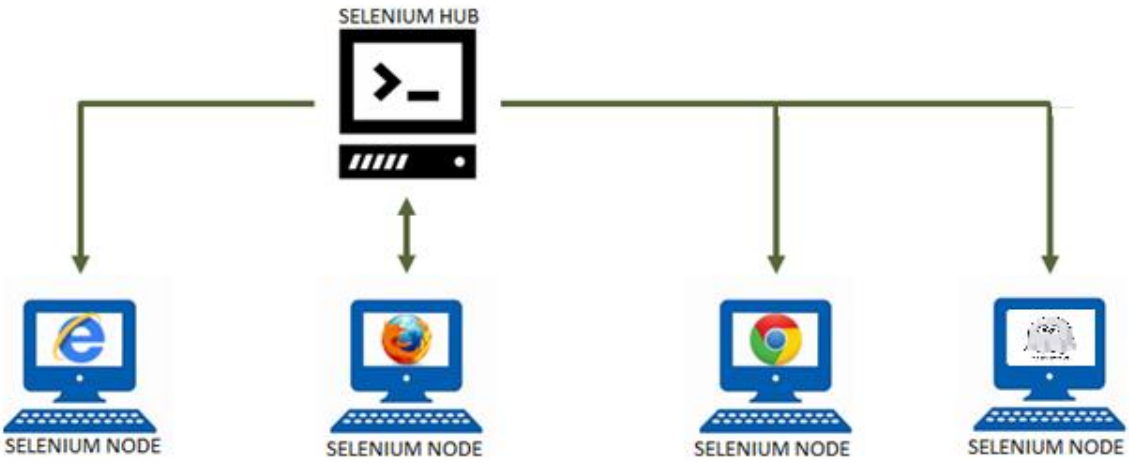

Figura 9. Esquema Selenium Grid

Ahora bien, esta solución no es la más óptima, ya que requiere tener que construir muchas máquinas virtuales en nuestro pc, junto con el usuario del sistema operativo y bastante espacio en disco, si a esto le sumamos más procesos corriendo como puede ser el servidor de Selenium Grid, puede conllevar a problemas de rendimiento o que incluso nuestra máquina no permita realizar está configuración por limitaciones en procesamiento o en espacio en disco.

Otra alternativa mejor en rendimiento es usar los contenedores Windows que aporta el uso de la tecnología Docker, que reutiliza el núcleo del sistema operativo host en los contenedores que se inicien en nuestro pc y sean del mismo sistema operativo, conllevando un ahorro en potencia de procesamiento y espacio en disco.

Además, Docker en su última versión, añade soporte para contenedores sobre sistemas operativos Windows. Por ello esta solución es la que se desea abordar en este proyecto.

A continuación, describiremos como montar la nueva estructura para la ejecución paralela de pruebas, basada en contenedores Windows gestionados por la herramienta Docker. Así en este nuevo escenario, crearemos los siguientes contenedores:

- Sistema operativo Windows 10 y servidor de Selenium Grid
- Sistema operativo Windows 10 y navegador Google Chrome
- Sistema operativo Windows 10 y navegador Mozilla Firefox
- Sistema operativo Windows 10 y navegador Microsoft Edge
- Sistema operativo Windows 10 y navegador PhantomJS

Sobre el contenedor Docker en el que está montado el servidor de Selenium Grid, ahora tendremos que configurar como nodos de ejecución, el resto de los contendores que incluyen los navegadores.

Por último, será necesario apuntar nuestro proyecto para que se ejecute de forma remota contra la url donde está expuesta el servidor de Selenium Grid del contenedor Docker. Que coincidirá con la IP de nuestro pc y el puerto 4444.

La estructura resultante sería la siguiente:

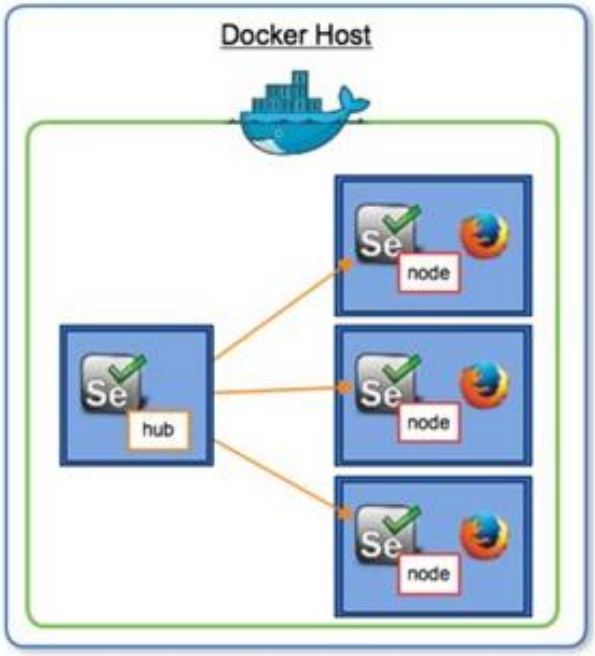

Figura 10. Estructura Selenium Grid con contenedores Docker

Para crear estos contenedores, nos basaremos en las imágenes Docker base proporcionadas por Microsoft: nanoserver o windowsservercore.[6]

La imagen Docker *microsoft/nanoserver* es una imagen más pequeña en tamaño diseñada para administración de sitios remotos, que consume menos recursos, carece de interfaz gráfica y está orientado a comportarse como un servidor web, un servidor DNS o un sistema de archivos.

La imagen Docker *microsoft/windowsservercore* es una imagen de mayor tamaño (unos 10 Gb), que incluye soporte a 32 bits y determinadas APIs de Windows que no incluye la imagen anterior. También carece de interfaz gráfica.

Esta carencia de soporte a aplicaciones con interfaz gráfica de usuario nos afecta de forma considerable en nuestro proyecto, puesto que nos impide ejecutar los tests sobre navegadores web reales como Google Chrome, Mozilla Firefox y Microsoft Edge.

[6] Ambas disponibles en https://hub.docker.com/r/microsoft/

Por ello utilizaremos máquinas virtuales creadas con Hyper-V Manager para ejecutar los tests sobre estos navegadores y Docker containers para el resto de escenarios.

En las máquinas virtuales instalaremos Java y los navegadores web citados anteriormente (excepto PhantomJS) y los integraremos como nodos en Selenium Grid.

En cuanto al resto de escenarios nos basaremos en la imagen de Microsoft windowsservercore. Sobre esa imagen base y apoyándonos en un fichero “Dockerfile”, personalizaremos nuestras imágenes a nuestro antojo, instalando Java, Selenium Grid, nuestro navegador headless PhantomJS y realizando la integración de comunicaciones entre el servidor y el nodo de ejecución del navegador PhantomJS.

Una vez tengamos montada la infraestructura de la solución, nos centraremos en la automatización de procesos.

Primero automatizando la forma de iniciar y parar los diferentes nodos de la infraestructura: imágenes Docker y máquinas virtuales creadas y segundo automatizando todo el proceso de ejecución de pruebas mediante un gestor de tareas.

Para la gestión de contenedores, nos apoyaremos en otra tecnología que aporta Docker para la gestión de contenedores, se llama Docker Compose y mediante un script ejecutado por Docker, permite iniciar y parar los contenedores incluidos.

Para la gestión de máquinas virtuales, nos apoyaremos en la herramienta de administración Hyper-V incluida en Windows 10 y en el uso de scripts con Windows Powershell, que nos permitirá arrancar y parar dichas máquinas virtuales.

Para la gestión de tareas, nos apoyaremos en herramientas de integración continua, concretamente en Jenkins, levantaremos Jenkins en nuestro pc, de modo local y apoyando en líneas de comandos y/o plugins como Powershell Jenkins plugin, podremos ejecutar las pruebas automáticas e iniciar / parar los contenedores y máquinas virtuales.

De hecho, el proceso completo sería: descargarse el código del proyecto, iniciar los contenedores Docker y máquinas virtuales, ejecutar la batería de pruebas automáticas de forma remota contra el servidor de Selenium Grid, que está montado en un contenedor Docker y que redirigirá las pruebas para ejecutarse contra los contenedores Docker y máquinas virtuales que contengan el navegador sobre el que se solicite ejecutar dicha batería de pruebas, una vez finalizadas todas las baterías de pruebas, se pararan los contenedores y máquinas virtuales y se finalizará el proceso.

4.3 Arquitectura funcional

Desde el gestor de tareas, se ejecutarán bien de forma manual o automática las tareas de arrancar y parar contenedores Docker y máquinas virtuales y ejecutar los tests de forma local/remota, secuencial/paralela. La arquitectura final resultante es la siguiente:

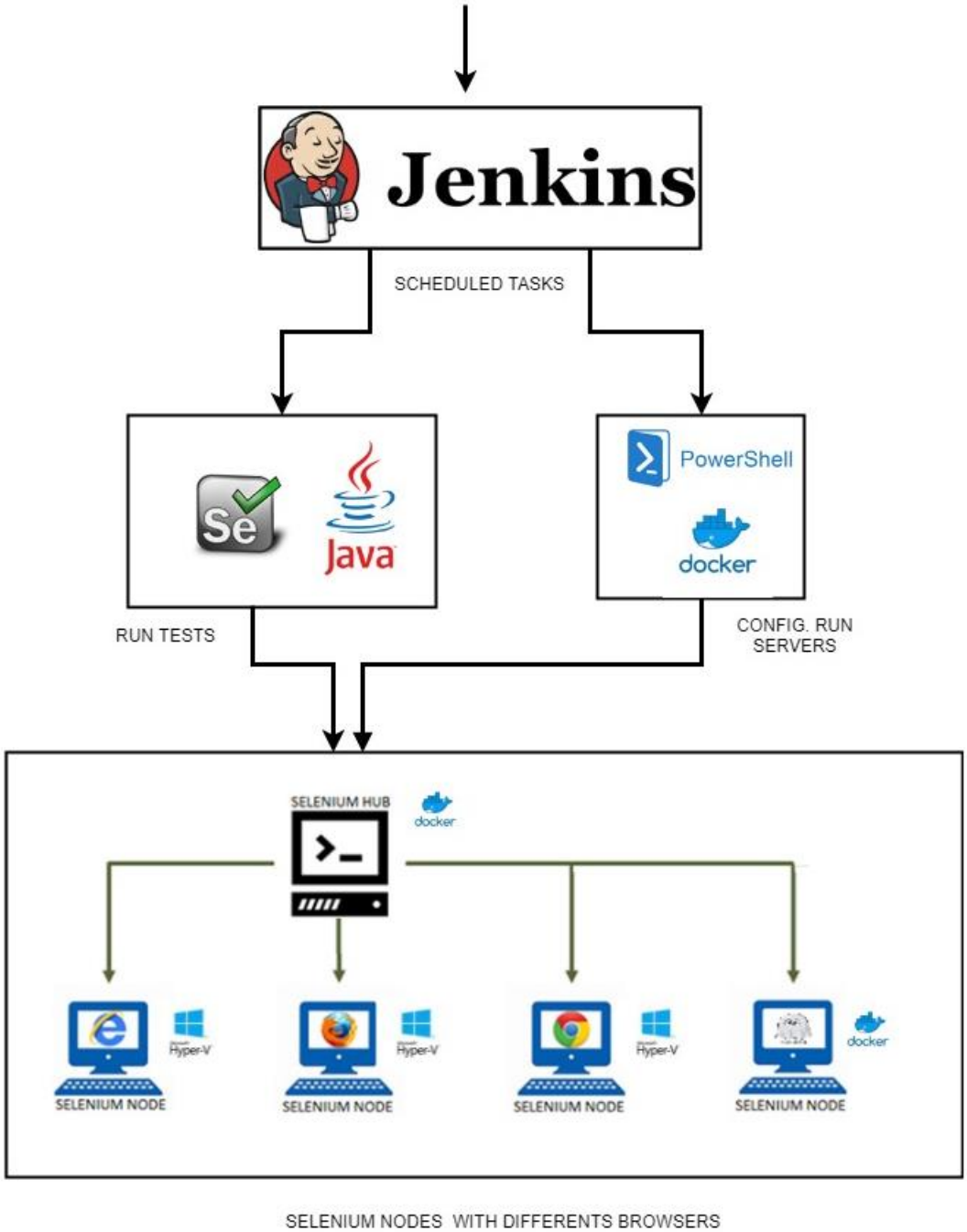

Figura 11: Arquitectura funcional

5. Descripción técnica de la solución

En esta sección describiremos los aspectos técnicos a considerar en el desarrollo e implementación de nuestro proyecto, realizando una separación entre los diferentes módulos identificados en el proyecto:

1. Desarrollo de página web objeto
2. Desarrollo proyecto automatización de pruebas web
3. Construcción de imágenes Docker
4. Implementación infraestructura del proyecto
5. Gestión automática del proyecto

El orden de definición de cada módulo es relevante, puesto que aparte de describir de forma técnica cada módulo, también se detallará como se integra con los módulos anteriormente desarrollados, si procede.

5.1 Módulo de desarrollo de página web objeto

Se ha desarrollado una página web básica, que será utilizada como objeto base del proyecto planteado.

Dicha página web se ha desarrollado utilizando la tecnología BootStrap 4.0.0.

Boostrap es un framework open source desarrollado y liberado por Twitter cuyo objetivo es facilitar el diseño web de forma que las webs sean responsive; esto es que sean adaptables a cualquier dispositivo y tamaño de pantalla.

Al apoyarnos en Bootstrap, nos hemos centrado en un diseño basado en el sistema de rejilla, para asegurar la ordenación automática de componentes en diferentes tamaños y configuraciones de pantalla, y que la web se siga mostrando de forma correcta y elegante.

También hemos utilizado las facilidades de integración que incluye Bootstrap, para trabajar con HTML5, librerías CSS3 y Javascript, haciendo uso de las versiones minificadas de CSS y JQuery.

En cuanto a funcionalidad, la página web dispondrá de una página principal que contendrá un formulario de login con usuario y contraseña.

Dicho login validará mediante Javascript que el usuario exista en el sistema y su contraseña sea la correcta. Si el login es correcto, redirigirá al usuario contra la página principal de la web.

Además, dispondrá de un enlace por si un usuario nuevo quiere registrarse.

Figura 12: Página principal con formulario de login

Si el usuario accede a la página de registrar usuario, se visualizará un nuevo formulario con un mayor número de campos, que permitirá darse de alta para el acceso a la aplicación mediante la validación mediante Javascript de la validez de los campos de formulario. En caso de que no se cumplan los requisitos de validación se mostrarán mensajes de error. Las validaciones realizadas serán las siguientes:

- Es obligatorio rellenar/seleccionar todos los campos
- Id usuario no exista en el sistema
- Contraseña formada por al menos 6 dígitos, que contenga al menos un número, una letra mayúscula y una letra minúscula.
- Campos contraseña y confirmar contraseña deben coincidir
- Campos email contenga @ y dominio.
- Campos email y confirmar email deben coincidir

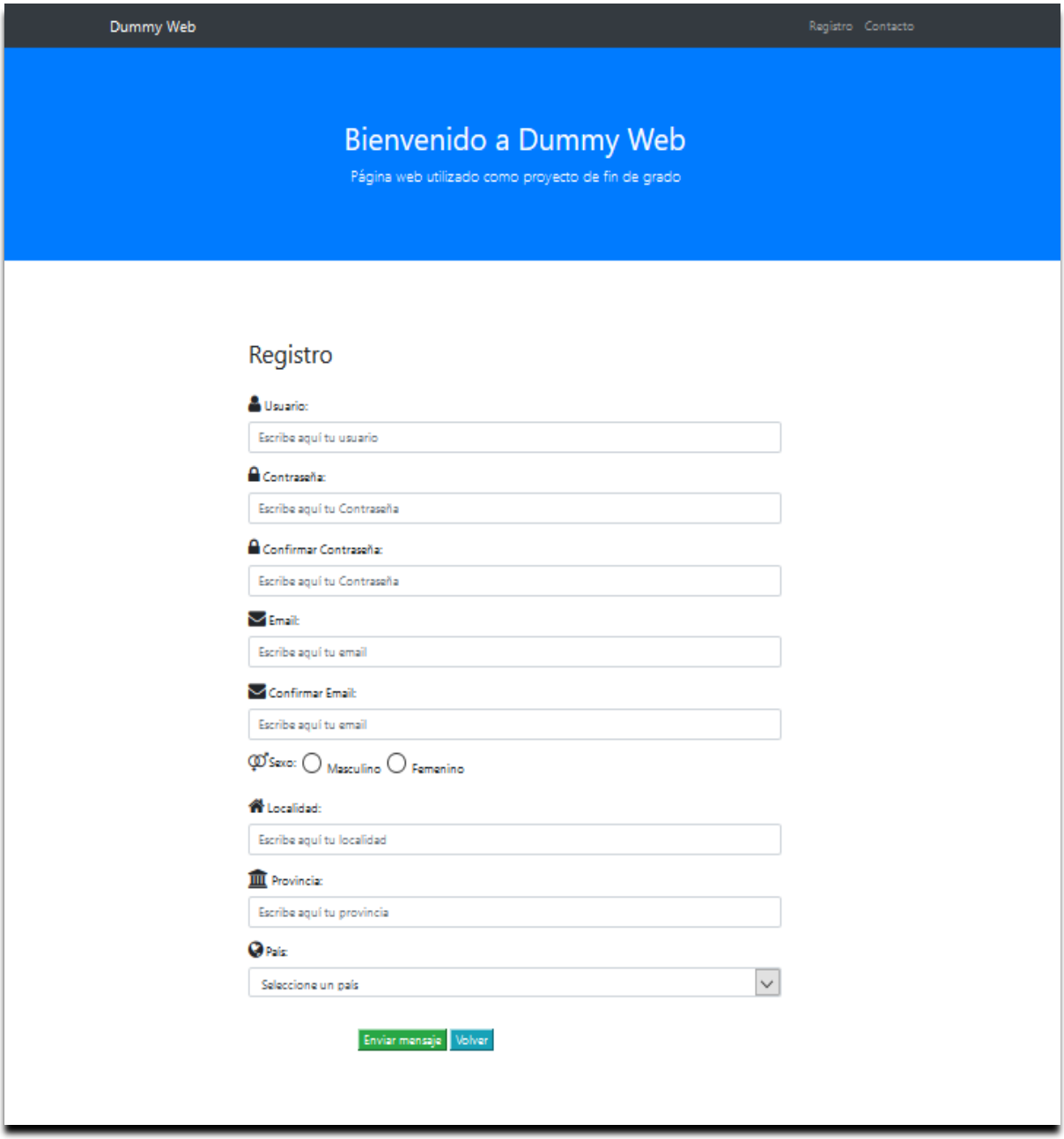

Figura 13: Página de registro de nuevo usuario

Una vez se haya realizado el login correctamente, accederemos a la página principal de la aplicación, en la cual se podrá navegar por las diferentes secciones, pestañas y listados.

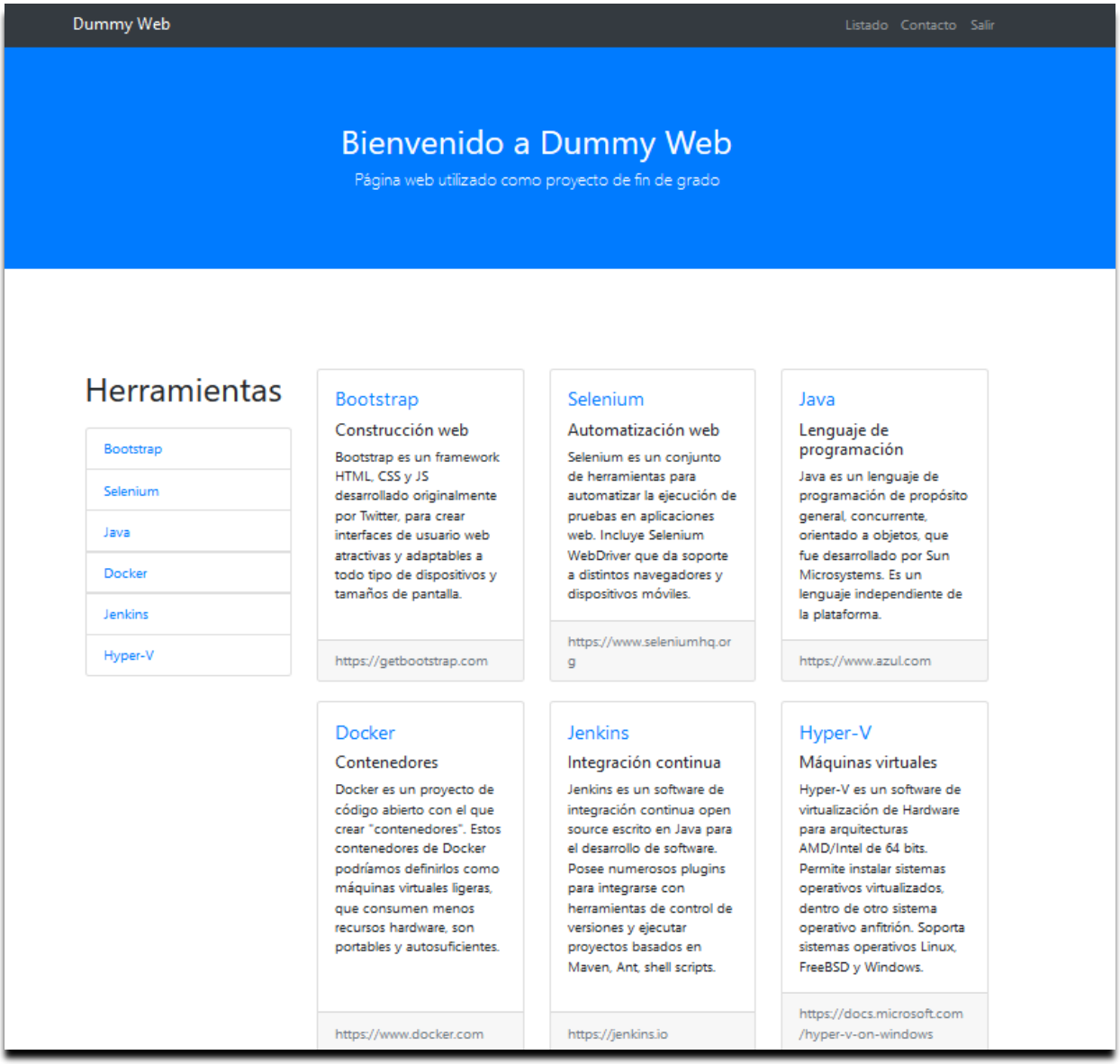

Figura 14: Página listado de información de usuario.

Además, desde esta página existirá un enlace de desconexión, que una vez pulsado cerrará la sesión y redirigirá a la página principal del formulario de login.

Hay que puntualizar que esta página web carece de scripts de ejecución en la capa del servidor, ni mecanismos de seguridad para la gestión de sesiones o cookies, no es objeto de este proyecto realizar una página web que implemente todas estas características.

Para garantizar la disponibilidad del servicio, la página web ha sido alojado en un servicio de terceros (Arsys) y es accesible desde la siguiente url:

http://www.sgc-tfgii.es/index.html

5.2 Módulo de desarrollo del proyecto de automatización de pruebas web

Este módulo se encarga de detallar la estructura y funcionamiento del proyecto de automatización de pruebas sobre la página web desarrollada en el punto anterior.

Las pruebas automáticas sobre dicha página web se realizarán bajo un proyecto en lenguaje de programación Java basado en Maven para la gestión y construcción del proyecto. Concretamente se utilizará Java 8 (versión 1.8.0_91) y Maven 3 (versión 3.3.9) y se apoyará en las siguientes librerías:

- Selenium como librería de apoyo a la automatización web, se utilizará sobre lla plataforma Java en su versión 3.7.1.
- TestNG como librería de ejecución de pruebas unitarias que trabaja sobre Java y está basado en JUnit. Se utiliza en su versión 6.8.
- Log4j: como librería de gestión y configuración de logs en Java, en su versión 1.2.15.

Estas versiones se incluyen en el archivo pom.xml que gobierna el proyecto mediante la Maven y se descargarán en fase de compilación del proyecto.

En cuanto al diseño de los tests, se utiliza la metodología PageObjects, en la cual cada página de la web se identifica como un objeto, en el cual se encapsula todas sus características y posibles acciones, en forme de atributos y métodos.

También es necesario definir los drivers que manejarán la navegación sobre los diferentes navegadores webs a utilizar, mediante la creación y configuración de objetos de la clase WebDriver de Selenium, que permitirán ejecutar las acciones pertinentes sobre el navegador web real.

Cuando se realice un evento que cambie de página en la web, será necesario realizar también un cambio de objeto y es sumamente importante que dicho objeto tenga acceso al driver, ya que es él que tendrá comunicación con el navegador.

Por último, se ha codificado un módulo de control de tratamiento de posibles excepciones que se produzcan en la ejecución del proyecto.

Este proyecto está dividido en los siguientes paquetes:

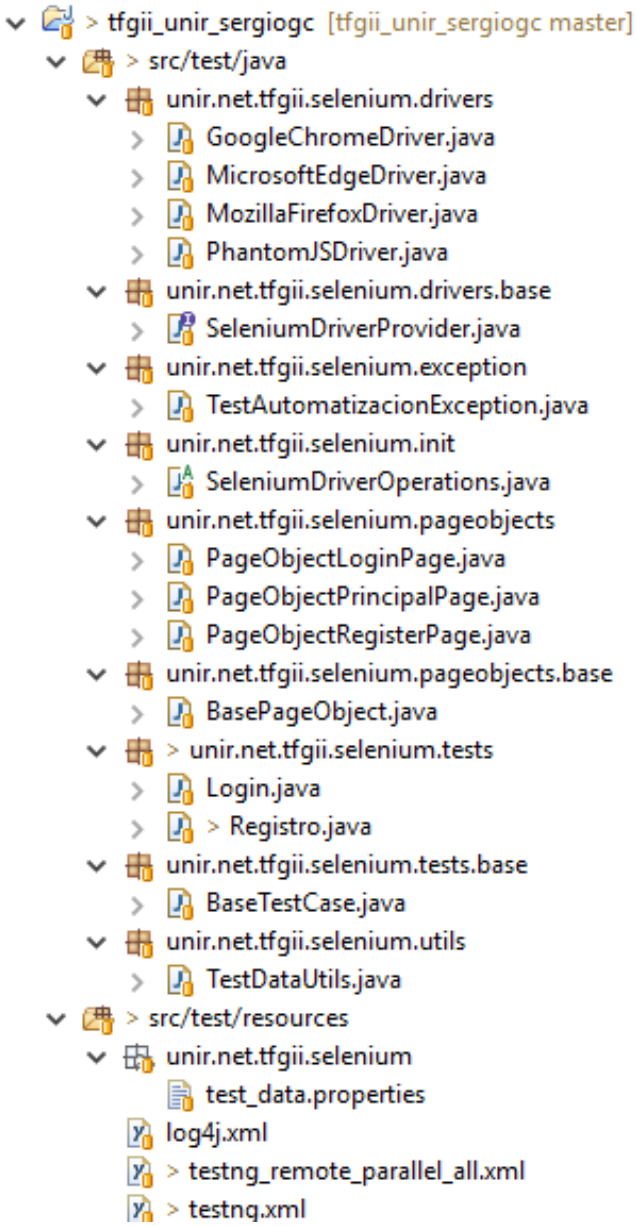

Figura 15: Estructura de proyecto Java

A continuación, describiremos como está organizado el proyecto:

- El código fuente se incluye en *src/test/java*
- Recursos de apoyo a los tests se incluye en *src/test/resources,*
- Fichero *pom.xml*: que incluye las dependencias Maven.

A nivel más detallado, la estructura del proyecto Java es la siguiente:

- Paquete **src/test/java**:
  - unir.net.tfgii.**selenium.drivers**
    - Incluye implementación drivers para la navegación sobre Mozilla Firefox, Microsoft Edge, Google Chrome y PhantomJS.
  - unir.net.tfgii.**selenium.drivers.base**
    - Clase padre de los drivers de Selenium.
  - unir.net.tfgii.**selenium.exception**
    - Incluye control de excepciones.
  - unir.net.tfgii.**selenium.init**
    - Clase base de los tests de automatización basados en TestNG. Incluye tareas previas y posteriores a los tests.
  - unir.net.tfgii.**selenium.pageobjects**
    - Cada página de navegación es tomada como un objeto con sus propios métodos y elementos con los que se puede interactuar.
  - unir.net.tfgii. **selenium.pageobjects.base**
    - Clase padre de los PageObjects. Métodos y atributos comunes a ellos.
  - unir.net.tfgii. **selenium.tests**
    - Definición de los tests. Navegación que realizar por los diferentes PageObjects. Incluye un test de login y otro deregistro en Dummy web.
  - unir.net.tfgii**. selenium.tests.base**
    - Clase padre de los tests. Métodos y atributos comunes a ellos.
  - unir.net.tfgii**. selenium.utils**
    - Utilidades: Integración con hojas de propiedades.
- Paquete **src/test/resources**:
  - unir.net.tfgii.**selenium**
    - Incluye hoja de propiedades (**test_data.properties**), con datos para utilizar en los tests. En esta hoja de propiedades están los paths a los drivers de navegadores.
  - **log4j.xml**: configuración de logs
  - **testng.xml**: archivo de tests de ejecución en modo secuencial.
  - **testng_remote_parallel_all .xml**: archivo de tests de ejecución en remoto.
- **pom.xml**
  - Archivo estructura proyecto, incluye dependencias maven, plugins, etc...

5.2.1 Configuración ejecución en local

Para poder realizar una ejecución local de los tests en nuestro equipo es necesaria realizar una configuración. Lo primero es tener instalados los diferentes navegadores sobre ls que queremos realizar las pruebas, en nuestro caso son Mozilla Firefox 57, PhantomJS 1.9.7, Google Chrome y Microsoft Edge, estos 2 últimos en sus últimas versiones.

Lo siguiente es haberse descargado los drivers compatibles de Selenium para dichos navegadores y descargarlos en disco. Una vez descargados es necesario actualizar su path en el ficheros "**test_data.properties**". En caso de que queramos que usen un perfil de usuario especifico en el navegador web, también habrá que indicarlo en este fichero.

Por último, se podrá ejecutar desde línea de comandos mediante Maven, utilizando el comando:

mvn test : por defecto ejecuta los tests definidos en el fichero **testng.xml**

Además, se han incluido parámetros para poder ejecutar con diferente configuración:

- En local, o en remoto (en este apartado obviamente elegiremos local)
- Selección fichero de test (en serie o en paralelo). Hay varios ficheros testng definidos, unos para ejecución secuencial y otros para ejecución en remoto (en este apartado no es necesario utilizarlo, ya que utilizar por defecto el fichero testng.xml)
- Selección de navegador, puede elegirse entre MozillaFirefoxDriver, ChromeDriver, PhantomJSDriver o MicrosoftEdgeDriver.

Así de este modo es posible la ejecución por parámetros:

mvn test –DdriverProvider=MozillaFirefoxDriver –Dexecution=local

mvn test –DdriverProvider=ChromeDriver –Dexecution=local –Ptestng_remote.xml

mvn test –DdriverProvider=MicrosoftEdgeDriver –Dexecution=local

mvn test –DdriverProvider=PhantomJSDriver –Dexecution=local

5.2.2 Configuración ejecución remota

Para la ejecución en remoto, será necesario una configuración diferente, Aquí nos apoyaremos en la funcionalidad de Selenium, llamada Selenium Grid; utilizada para la ejecución de tests de manera distribuida, sobre diferentes navegadores y plataformas.

Deberemos añadir una nueva configuración para que en el caso de que la ejecución sea en remoto, se ejecute el test contra el servidor de Selenium Grid (situado por defecto en la url http://IP_INTERNA:4444/wd/hub), y que esté sea quien redirija la ejecución sobre los distintos nodos que tenga configurados con las distintas plataformas y navegadores web.

El servidor de Selenium Grid, será el que balancee hacia los diferentes nodos. En nuestro proyecto esa dirección será la url que exponga un contenedor Docker para el servidor de Selenium Grid. Además, será necesario ejecutar contra el fichero testng para pruebas en paralelo llamado **tesng_remote_parallel_all.xml**

```
01. WebDriver driver ;
02.
03.         if(execution.equals("local")){
04.
05.             final String chromeServer = TestDataUtils.getData("ChromeServer");
06.             final String chromeProfile = TestDataUtils.getData("ChromeProfile");
07.
08.             LOG.debug("Inicializando driver Google Chrome.");
09.
10.             System.setProperty("webdriver.chrome.driver", chromeServer);
11.
12.             ChromeOptions options = new ChromeOptions();
13.             options.addArguments("user-data-dir="+chromeProfile);
14.             options.addArguments("--start-maximized");
15.             options.addArguments("--test-type");
16.             driver = new ChromeDriver(options);
17.         }else{ //is remote
18.             LOG.debug("Inicializando driver Chrome Remoto");
19.             DesiredCapabilities dc = DesiredCapabilities.chrome();
20.             dc.setPlatform(Platform.WINDOWS);
21.
22.             driver=new RemoteWebDriver(new URL("http://"+ip+":4444/wd/hub"), dc);
23.
24.         }
```

Figura 16: Fragmento de código inicialización del driver de Chrome en local o remoto

5.3 Módulo de construcción de imágenes

Para poder ejecutar el proyecto Java de forma remota, necesitamos tener previamente construidas las imágenes Docker necesarias para nuestro proyecto.

Para ello partiremos primero de una imagen Docker basada en Windows 10 proporcionada por Microsoft: **microsoft/windowsservercore**, que podemos encontrar en:

https://hub.docker.com/r/microsoft/windowsservercore/

Posteriormente creamos nuestra propia imagen Docker base, que incluya el core de Windows 10 y alguna configuración propia, como por ejemplo la instalación de Java 8 y su configuración en el path.

A partir de esa imagen base, construiremos las siguientes nuevas imágenes Docker:

- Hub: Servidor que haga de balanceador (instalación selenium Grid y exponer la url que hemos definido como url de nuestro servidor de Selenium Grid)
- Nodo PhantomJS: Servidor de navegación headless basado en Javascript (instalación de PhantomJS, más configuración driver GhostDriver y enlace con el servidor Hub)

A la hora de construir las imágenes necesitamos tener instalado Docker en nuestro PC. En nuestro proyecto utilizaremos la versión de Docker 17.09.0-ce.

Además, puesto que vamos a construir contenedores Docker bajo sistema operativo Windows 10, nos apoyaremos en la consola de Windows y de Powershell.

Para poder construir cada imagen tendremos que crear un fichero “Dockerfile”, que contendrá los comandos necesarios para configurar la imagen apoyándonos en las herramientas anteriormente mencionadas.

A continuación, explicaremos como construir un fichero Dockerfile para nuestra imagen base del proyecto. A partir de una imagen predefinida (en nuestro caso microsoft/nanoserver) se descarga una copia y entonces se realizan modificaciones mediante comandos en PowerShell.

Posteriormente, definimos powershell como Shell por defecto, definimos también la versión de la distribución de Java que nos vamos a bajar y el directorio de instalación, se procede a descargar y almacenar la distribución Java, para posteriormente instalarla y configurarla en el path y en el registro de Windows.

```
01. FROM microsoft/windowsservercore:latest
02.
03. SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPreference
04.
05. ENV JAVA_VERSION 8.27.0.7-jdk8.0.162
06. ENV JAVA_HOME C:\\zulu${JAVA_VERSION}-win_x64
07.
08. RUN Invoke-WebRequest $('https://cdn.azul.com/zulu/bin/zulu{0}-
    win_x64.zip' -f $env:JAVA_VERSION) -OutFile 'openjdk.zip' -UseBasicParsing ; \
09.     Expand-Archive C:\openjdk.zip  -DestinationPath C:\ ; \
10.     $env:PATH = '{0}\bin;{1}' -f $env:JAVA_HOME, $env:PATH ; \
11. Set-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager
    \Environment\' -Name Path -Value $env:PATH ;
```

Figura 17: Dockerfile: Construcción imagen base de nuestros contenedores

Como nota, indicar que nos hemos basado en una versión de libre distribución de Java, puesto que tanto en Oracle como en RedHat es necesario disponer de una cuenta para poder descargarla. La distribución elegida es Azul y la versión 1.8u162.

Una vez tengamos finalizado nuestro primer Dockerfile, tenemos que crear nuestra imagen Docker y almacenarla en nuestro repositorio local, esto se hace situándonos en el directorio donde este ubicado el fichero Dockerfile y ejecutando el siguiente comando:

docker build -t [nombre_imagen] .

```
C:\SoftDesarrollo\SeleniumDockerWindows\base>docker build -t base_server .
Sending build context to Docker daemon   2.56kB
Step 1/5 : FROM microsoft/windowsservercore:latest
 ---> 8d57c0984795
Step 2/5 : SHELL powershell -Command $ErrorActionPreference = 'Stop'; $ProgressPreference = 'SilentlyContinue';
 ---> Using cache
 ---> 06198abe163f
Step 3/5 : ENV JAVA_VERSION 8.27.0.7-jdk8.0.162
 ---> Using cache
 ---> dd95414a43d6
Step 4/5 : ENV JAVA_HOME C:\\zulu${JAVA_VERSION}-win_x64
 ---> Using cache
 ---> 2b39a67a9bbf
Step 5/5 : RUN Invoke-WebRequest $('https://cdn.azul.com/zulu/bin/zulu{0}-win_x64.zip' -f $env:JAVA_VERSION) -Ou
tFile 'openjdk.zip' -UseBasicParsing ;     Expand-Archive C:\openjdk.zip  -DestinationPath C:\ ;     $env:PATH =
 '{0}\bin;{1}' -f $env:JAVA_HOME, $env:PATH ;     Set-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control
\Session Manager\Environment\' -Name Path -Value $env:PATH ;
 ---> Using cache
 ---> aa30122e19a6
Successfully built aa30122e19a6
Successfully tagged base_server:latest

C:\SoftDesarrollo\SeleniumDockerWindows\base>
```

Figura 18: Dockerfile: Construcción imagen base de nuestros contenedores

Para comprobar que se ha creado correctamente y se encuenta en nuestro repositorio local de imágenes Docker mediante el comando: **docker images**

```
C:\Users\sergiog>docker images
REPOSITORY                    TAG                 IMAGE ID
phantomjs_servernode          latest              8fcfced9b93f
hub_server                    latest              1b57c8a04c41
base_server                   latest              aa30122e19a6
microsoft/windowsservercore   latest              8d57c0984795
```

Figura 19: Consulta imágenes Docker en nuestro repositorio local

Una vez tenemos construida la imagen base para nuestro del proyecto, a partir de ella, podremos ir construyendo el resto de las imágenes Docker.

```
01. FROM base_server:latest
02.
03. SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPreference = 'SilentlyContinue';"]
04.
05. ADD https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-1.9.7-windows.zip C:/SoftDesarrollo
    /SeleniumWebdriver/phantomjs-1_9_7-windows.zip
06.
07. ADD https://selenium-release.storage.googleapis.com/3.7/selenium-server-standalone-3.7.1.jar C:/selenium-server-
    standalone-3_7_1.jar
08.
09. RUN java -version
10.
11. RUN Expand-Archive C:/SoftDesarrollo/SeleniumWebdriver/phantomjs-1_9_7-
    windows.zip  -DestinationPath C:/SoftDesarrollo/SeleniumWebdriver ;
12.
13. RUN Get-Package
14.
15. RUN Set-Location -Path C:/SoftDesarrollo/SeleniumWebdriver/phantomjs-1.9.7-windows/
16.
17. EXPOSE 8083
18.
19. ENTRYPOINT $hostnamehostname =  hostname ; \
20.     $ipV4 = Test-Connection -ComputerName $hostname -Count 1 | Select -ExpandProperty IPv4Address ; \
21.     $ip= $ipv4.IPAddressToString ; \
22.     $ip= $ip+":8083" ; \
23. C:/SoftDesarrollo/SeleniumWebdriver/phantomjs-1.9.7-windows/phantomjs --webdriver=$ip --webdriver-selenium-
    grid-hub=http://192.168.43.247:4444
```

Figura 20: Dockerfile construcción imagen PhantomJS a partir de imagen Base

Como podemos ver en el ejemplo anterior, a partir de nuestra imagen base con Java instalado, descargamos e instalamos el navegador PhantomJS, configurados el driver del navegador GhostDriver en dicha imagen y comunicamos este nodo con el servidor de Selenium Grid.

Una vez hayamos construido todos nuestros ficheros de definición, construiremos las imágenes cada una desde la ruta donde se encuentre sus correspondientes ficheros Dockerfile, mediante los siguientes comandos:

docker build -t hub .

docker build -t phantomjs_node .

Una vez construidas, serán visibles desde el repositorio local:

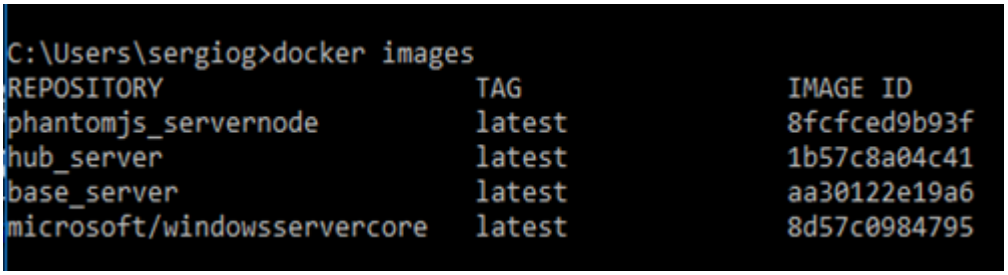

Figura 21: Imágenes Docker empleadas en el proyecto

Por último, una vez construidas, es necesario iniciarlas, tal tarea se realiza con los siguientes comandos:

docker run -d -p 4444:4444 --name hub hub_server:latest

(expone el puerto 4444 del contenedor con el puerto 4444 del sistema huésped)

docker run -d --name selenium-phantomjs phantomjs_servernode:latest

Como observamos es necesario definir el puerto que exponemos en la imagen que construye el servidor de Selenium Grid con el mismo puerto expuesto en nuestro pc. En el resto es necesario que se configuren como un nodo de ese servidor, para el posible balanceo de peticiones.

A partir de este momento, dejamos de hablar de imágenes y empezamos a hablar de contenedores, puesto que ya se están ejecutando.

Además, mientras se ejecutan estos contenedores, podremos ver su estado con los siguientes comandos que nos ofrece la herramienta Docker:

docker ps

Cuando se requiera finalizar la ejecución de estos contenedores se podrán parar mediante el comando:

docker stop [ID_CONTAINER]

Aunque estén parados se podrán observar como están inactivos, si deseamos que desparezcan del listado de contenedores habrá que ejecutar el comando:

docker rm [ID_CONTAINER]

Si además se quiere eliminar la imagen que da como lugar ese contenedor, habrá que ejecutar el siguiente comando:

docker rmi [ID_IMAGE]

5.4 Módulo de implementación de la infraestructura del proyecto

Una vez tenemos creadas las diferentes imágenes Docker y máquinas virtuales necesarias para nuestro proyecto, es necesario arrancarlas y comprobar la comunicación entre ellas, en nuestro caso, además, comprobar la comunicación entre la imagen Docker que contiene el servidor de Selenium Grid y el resto de las imágenes y máquinas virtuales que son las que contienen los navegadores.

Otra comprobación que realizar; es si el proyecto Java se comunica correctamente con la url donde está expuesto el servidor de Selenium Grid de la misma imagen.

5.5 Módulo de gestión automatica de contenedores

Toda esta gestión de contenedores puede ser centralizada, por una herramienta que incluye Docker llamada Docker Compose; la cual se centra en los contenedores Docker y sus relaciones, mediante la definición de dichas relaciones en un fichero plano en formato “*.yml” llamado “docker-compose.yml”.

Sobre este fichero de configuración se puede realizar diferentes comandos para arrancar y parar los contenedores definidos en dicho fichero, mediante los comandos:

docker-compose up -d

docker-compose down

Además, también se puede listar mediante el comando:

docker-compose ps

```
1. version: "3"
2. services:
3.   grid:
4.     image: hub_server
5.     container_name: selenium-hub
6.     ports:
7.       - "4444:4444"
8.   node_phantomjs:
9.     image: phantomjs_servernode
10.    Container_name: selenium-phantomjs
```

Figura 22: Docker-Compose.yml

5.6 Módulo de control de la ejecución del proyecto

Por último, nos apoyaremos en Jenkins como gestor y planificador de tareas, para las diferentes acciones de lanzar pruebas, arrancar/parar contenedores, etc…

Se trata de una WAR que puede ser desplegado de la siguiente forma:

java –jar jenkins.war

O bien configurado como servicio Windows. (Elegiremos la primera opción)

Una vez desplegado, se trata de entrar y crear tareas, que ejecutaran las acciones comentadas, por líneas de comandos.

Se proporcionará un manual detallado tanto de instalación, como de usuario en los siguientes anexos de este documento:

- **Anexo A.1: Manual de instalación**
- **Anexo A.2: Manual de usuario.**

Esta herramienta permite la configuración y ejecución de tareas de varias formas, siendo totalmente configurable, permitiendo programar las ejecuciones de las tareas o incluso crear un pipeline de tareas.

Además, permite la ejecución por línea de comandos, acciones condicionales, ejecución parametrizada, consulta por logs.

6 Validación técnica de la solución

En este apartado se centra en comentar las validaciones que se han realizado sobre la solución aportada.

6.1 Pruebas funcionales

Se han realizado pruebas funcionales de la solución, tanto a nivel de funcionalidad de la página web como del proyecto de pruebas automáticas.

En el proyecto de pruebas automáticas se han realizado pruebas, sobre los diferentes tests definidos en los requisitos, y sobre los diferentes navegadores, tanto en navegación visual como en navegación headless.

Se han ejecutado pruebas de concurrencia para ver el funcionamiento de los tests en paralelo y de forma secuencial.

6.2 Pruebas de integración de componentes

De igual modo se han realizado numerosas pruebas de integración.

Desde el proyecto de pruebas automáticas se han hecho integraciones contra contenedores y máquinas virtuales, con diferentes configuraciones de adaptadores de red. Pruebas de comunicación de las diferentes máquinas virtuales y contenedores con la parrilla de Selenium Grid.

También se han realizado pruebas de integración entre la herramienta de gestión de tareas (Jenkins) contra el Selenium Grid, los contenedores Docker y las máquinas virtuales en Hyper-V.

7. Conclusiones y líneas de trabajo futuras

En este apartado se muestran las conclusiones obtenidas en la realización de este trabajo y las líneas de trabajo futuras posibles a realizar a partir de este proyecto.

7.1 Conclusiones

Esta solución es óptima para poder disponer de un entorno multidispositivo a coste cero y con bastante potencial para tener trabajando multitud de máquinas virtuales en paralelo. Ahora bien, conlleva un alto consumo de procesador, memoria y almacenamiento.

Como hemos observado la solución con contenedores Docker es mejor en rendimiento y escalabilidad, sin embargo; aún está lejos de la solución con contenedores Linux, primero por su incapacidad para trabajar en Windows Docker Containers con aplicaciones que requieran soporte de interfaz gráfica de usuario.

Tal como hemos visto no es posible ejecutar navegadores Google Chrome y Mozilla Firefox, dentro de los contenedores Docker proporcionados por Microsoft para operar con sistemas operativos Windows. En la imagen Docker que incluye mayor funcionalidad: microsoft/windowsservercore permite instalar dichos navegadores web pero no permite la ejecución en modo headless, ya que necesita de soporte GUI.

En el segundo punto, penaliza bastante el tamaño de las imágenes proporcionadas por Microsoft, si bien aporta una imagen más ligera como es microsoft/nanoserver, es bastante limitada en cuanto a configuración y está destinada para usos muy básicos y especificados.

7.2 Líneas de trabajo futuras

Hay varias líneas de trabajo a fututo a partir de este proyecto:

La primera es, si Microsoft ofrece nuevas imágenes Docker basadas en Windows Containers con soporte para aplicaciones con interfaz gráfica de usuario, sería cambiar las máquinas virtuales por contenedores Docker sobre Wndows.

La segunda, sería integrar este trabajo, con el hómologo sobre Linux Docker Containers. Esta solución está bastante definida. La problemática vendría en que actualmente la solución depende del modo en que se encuente Docker en el sistema host: puede ser en modo Windows Containers o Linux Containers, y no es posible trabajar con contenedores de distinto tipo al modo en el que se encuentre ejecutando Docker en el sistema host. Por el momento habría que idear una forma de poder realizar ese cambio (switch) en el Docker del sistema host, levantar la infraestructura según corresponda, ejecutar las pruebas, pararla, cambiar el switch y repetir el proceso para el siguiente modo de ejecución.

La tercera línea de trabajo futuro sería aumentar las capacidades de la infraestructura de Linux Docker Containers para realizar pruebas sobre aplicaciones móviles, Integrar Selenium con el framework Appium para automatización de pruebas móviles y levantar emuladores/simuladores en Android para ejecutar las pruebas. Este escenario requiriria de disponer de una app Android para las pruebas.

8. Bibliografía

[1] **Selenium** Project. Last updated on Mar 20, 2018.

https://www.seleniumhq.org/docs/

https://www.seleniumhq.org/projects/grid/

[2] Book: Next Generation Java Testing: **TestNG** and Advanced Concepts 1st Edition

Pearson Education, 15 oct. 2007 , Cédric Beust, Hani Suleiman

http://testng.org/doc/

[3] **Docker** Inc oficial web site -2018

https://www.docker.com/

https://docs.docker.com/

[4] **Jenkins** Official web Site 2018

https://jenkins.io/download/

[5] Book: *Foundations of Software Testing **ISTQB** Certification* (2012)

Rex Black, Dorothy Graham and Erik Van Veenendaal

[6] eBook: Test Maturity Model integration (**TMMi**) Edición 1.0 (2015)

Producida por la TMMi Foundation, Erik van Veenendaal

www.tmmifoundation.org

[7] **Microsoft** Official Documentation Web Site (2018)

https://docs.microsoft.com/es-es/powershell/

https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/quick-start/

[8] **CMMI**® para Desarrollo, Versión 1.3 CMMI-DEV, V1.3 Noviembre 2010

Equipo del Producto CMMI .*Mejora de los procesos para el desarrollo de mejores productos y servicios*

Software Engineering Process Management Program

http://www.sei.cmu.edu

[9] PhantomJS oficial web site, 2018 Ariya Hidayat

http://phantomjs.org/documentation/

[10] Ingeniería del software, Ian Sommerville. Pearson Educación, 2005

A Anexos

A continuación, se detalla un manual de instalación para preparar el entorno de ejecución del proyecto y un manual de usuario de gestión de tareas desde la herramienta Jenkins.

A.1 Manual de instalación

Para poder ejecutar esta solución software se requiere de un computador de escritorio con los siguientes requisitos mínimos:

- Procesador de al menos 2 núcleos de 2Ghz
- Memoria RAM: 8 Gb.
- Disco duro con al menos 120 Gb de almacenamiento.
- Sistema operativo Windows 10, de 64 bits, cuya release sea la versión 1607, OS Build: 14393 o superior (requisito para Docker for Windows)
- Distribución de Java 8 instalada y configurada en el path.
- Distribución de Maven 3 instalada y configurada en el path.
- Disponer de 3 máquinas virtuales creadas y administradas con la herramienta Hyper-V Manager, con las siguientes características:
    - Sistema operativo Windows 10 y release superior a la versión 1607.
    - Disco duro de al menos 30 Gb de almacenamiento.
    - Memoria RAM asignada de máximo 2 Gb.
    - Adaptadores de red, con acceso a internet y red local interna.
    - Configuradas con inicio de sesión (usuario y contraseña)

En nuestro equipo disponemos de un procesador Intel Core 2 Duo a 2,5 Ghz, 8 Gb de memoria RAM y un sistema operativo Windows 10 Education, con arquitectura de 64 bits, cuya release es la versión 1709, OS Build: 16299, Fecha liberación: 17/10/2017.

Además, partimos de la base de tener instalada y configurada, una distribución Java JSK en su versión 8, hemos optado por la distribución Java proporcionada por Azul Systems:

```
C:\SoftDesarrollo>java -version
openjdk version "1.8.0_112"
OpenJDK Runtime Environment (Zulu 8.19.0.1-win64) (build 1.8.0_112-b16)
OpenJDK 64-Bit Server VM (Zulu 8.19.0.1-win64) (build 25.112-b16, mixed mode)
```

Figura 23: Versión de Java

También necesitaremos tener instalada y configurada en el path la herramienta Apache Maven en su versión 3.0 o superior:

```
C:\SoftDesarrollo>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:47+01:00)
Maven home: C:\SoftDesarrollo\apache-maven-3.3.9\bin\..
Java version: 1.8.0_112, vendor: Azul Systems, Inc.
Java home: C:\SoftDesarrollo\Java\zulu8.19.0.1-jdk8.0.112-win_x64\jre
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
```

Figura 24: Versión de Maven

A partir de estos requisitos iniciales, necesitaremos también realizar una serie de acciones para configurar nuestro entorno, tanto el sistema operativo huésped, como las máquinas virtuales y los contenedores Docker.

En cuanto a nuestro pc de escritorio, es necesario además instalar los navegadores sobre los cuales vamos a realizar las pruebas automáticas sobre la web de ejemplo. Estos navegadores son Mozilla Firefox, Google Chrome, Microsoft Edge y PhantomJS. Para ello, se incluyen una serie de **scripts basados en Windows Powershell, que descargarán e instalarán dichos navegadores** en unas rutas específicas. Estos scripts se han subido al repositorio de código del proyecto y se encuentran en:

https://github.com/sglezcasco/sgc-tfgii/tree/master/powershell_scripts

Será necesario ejecutarlos tanto en el pc host, como en las máquinas virtuales.

Una vez tenemos configurados los navegadores, es necesario instalar la herramienta Docker para Windows, desde el siguiente enlace.

https://download.docker.com/win/stable/Docker%20for%20Windows%20Installer.exe

Descargaremos e instalaremos dicha herramienta, con las opciones por defecto. Si bien llegará un momento que nos pedirá un reinicio para habilitar el modo de virtualización Hyper-V (si es que aún no lo tenemos activado).

Una vez reiniciado, se iniciará la herramienta Docker, y veremos un icono en la parte inferior derecha iniciándose “starting”, despúes de unos segundos cambiará a estado en ejecución “running”, cuando ya la tendremos disponible.

Podemos verificar la instalación de Docker por línea de comandos:

```
C:\SoftDesarrollo>docker -v
Docker version 17.09.0-ce, build afdb6d4

C:\SoftDesarrollo>docker-compose -v
docker-compose version 1.16.1, build 6d1ac219
```

Figura 25: versión Docker

Ahora bien, Docker se instala por defecto para trabajar con contenedores Linux, como nuestro proyecto trabajará con contenedores Windows es necesario cambiar mediante la opción “Switch to Windows containers…”:

Figura 26: Docker switch to Window Containers

Una vez tenemos Docker instalado y configurado, el siguiente paso es revisar la configuración de las máquinas virtuales creadas, utilizaremos cada máquina virtual como un servidor para ejecutar los navegadores web con interfaz gráfica.

- **W10_firefox_node**
- **W10_chrome_node**
- **W10_edge_node**

Así hemos definido 3 máquinas virtuales como puede observarse, en la herramienta Hyper-V Manager incluida en Windows 10. Como puede verse en el siguiente Gráfico, la instalación de Docker también ha creado una propia máquina virtualizada Hyper-V "MobyLinuxVm" destinada para el uso de contenedores Linux con Docker.

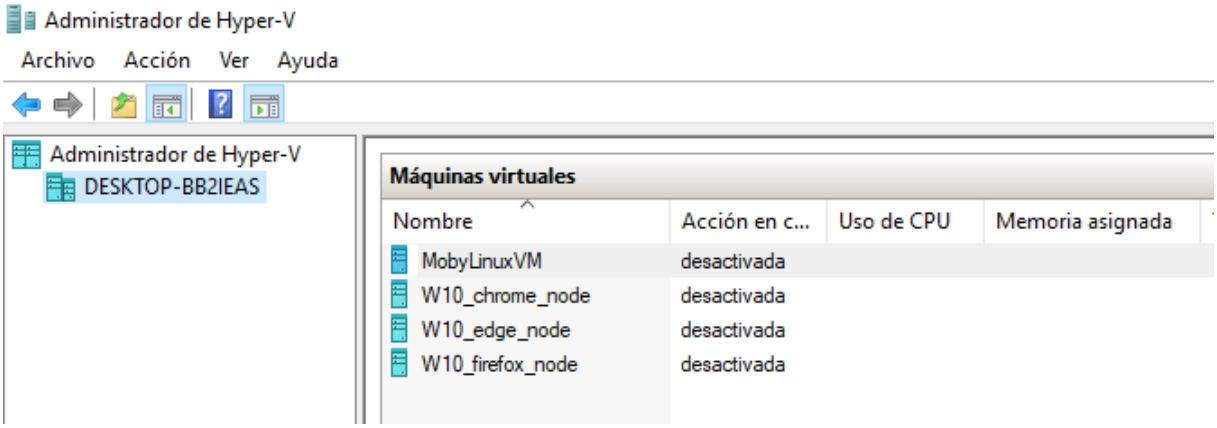

Figura 27: Hyper-V estado VM

Por último, es necesario asegurarse que las máquinas virtuales tengan acceso por usuario y contraseña, y que tengan las políticas de ejecución no restringidas, para poder ejecutar comandos de forma remota, para ello mediante consola Powershell con permisos de administrador ejecutaremos:

```
C:\SoftDesarrollo>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\SoftDesarrollo> Set-ExecutionPolicy Unrestricted
PS C:\SoftDesarrollo> Get-ExecutionPolicy
Unrestricted
```

Figura 28: Powershell políticas de seguridad

Este último paso, es muy probable que también tengamos que ejecutarlo en nuestro sistema operativo huésped.

Una vez tenemos ya montada toda la infraestructura montada, sólo nos falta configurar la herramienta que utilizaremos como interfaz gráfica de nuestra solución: Jenkins.

Para ello utilizamos el script de descarga de powershell situado en:

https://github.com/sglezcasco/sgc-tfgii/blob/master/powershell_scripts/install_jenkins.ps1

Situándonos en el directorio de descarga mediante la consola de Windows, con permisos de ejecución, ejecutaremos el siguente comando para proceder a su instalación y arranque: **java -jar jenkins.war**

Esto iniciará el proceso de instalación que podremos seguir visualmente en la dirección web por defecto: **localhost:8080**

En primer lugar, nos pedirá introducir el código de instalación, localizado en un fichero dentro de un directorio especificado:

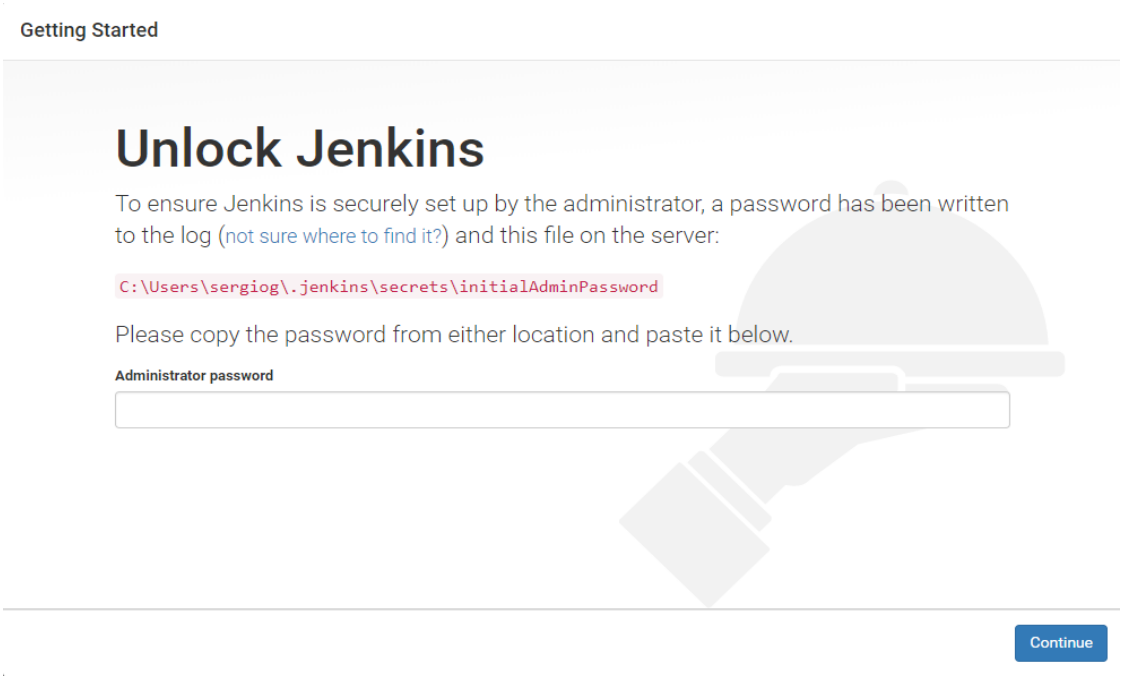

Figura 29: Jenkins instalación paso 1

Introduciremos el código y continuaremos, seleccionando la instalación estándar:

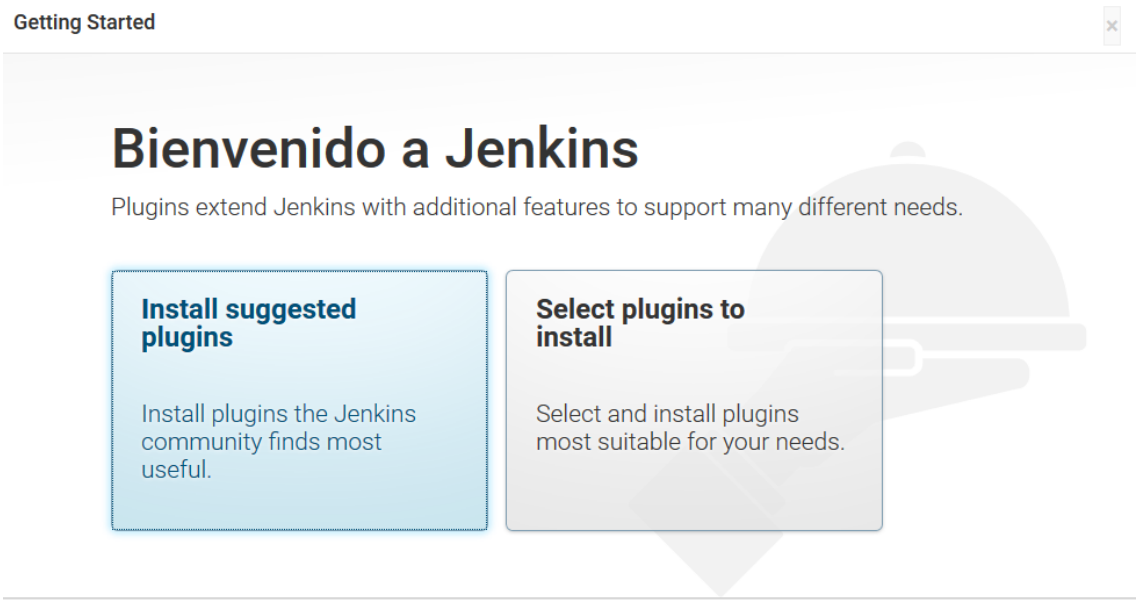

Figura 30: Jenkins instalación paso 2

Se irán instalando los plugins por defecto:

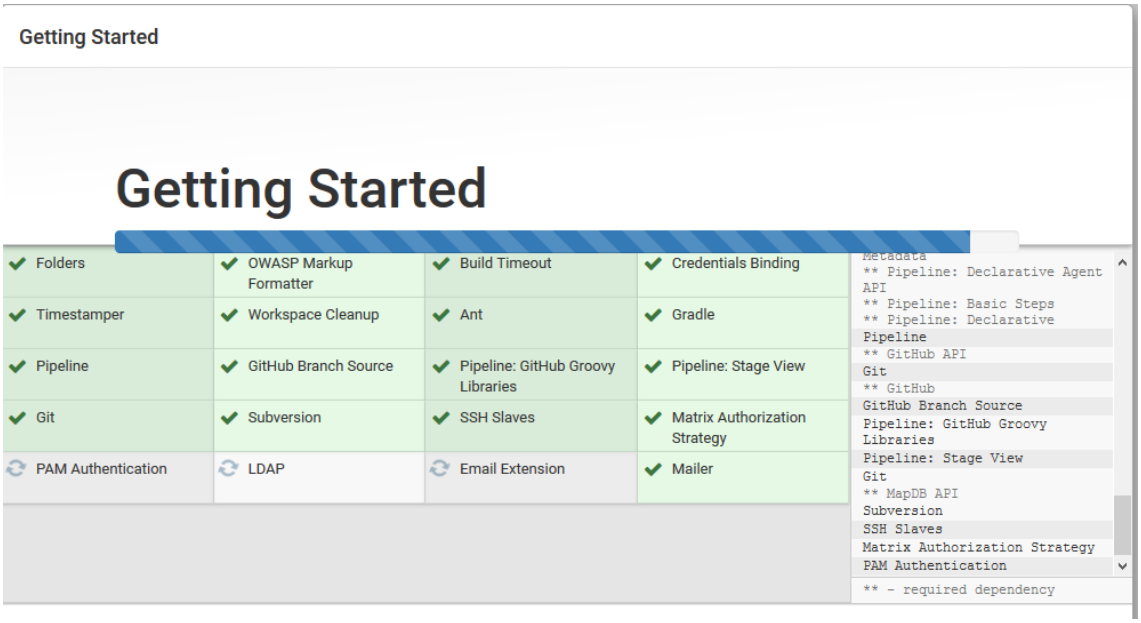

Figura 31: Jenkins instalación paso 3

Gestionaremos usuario administrador:

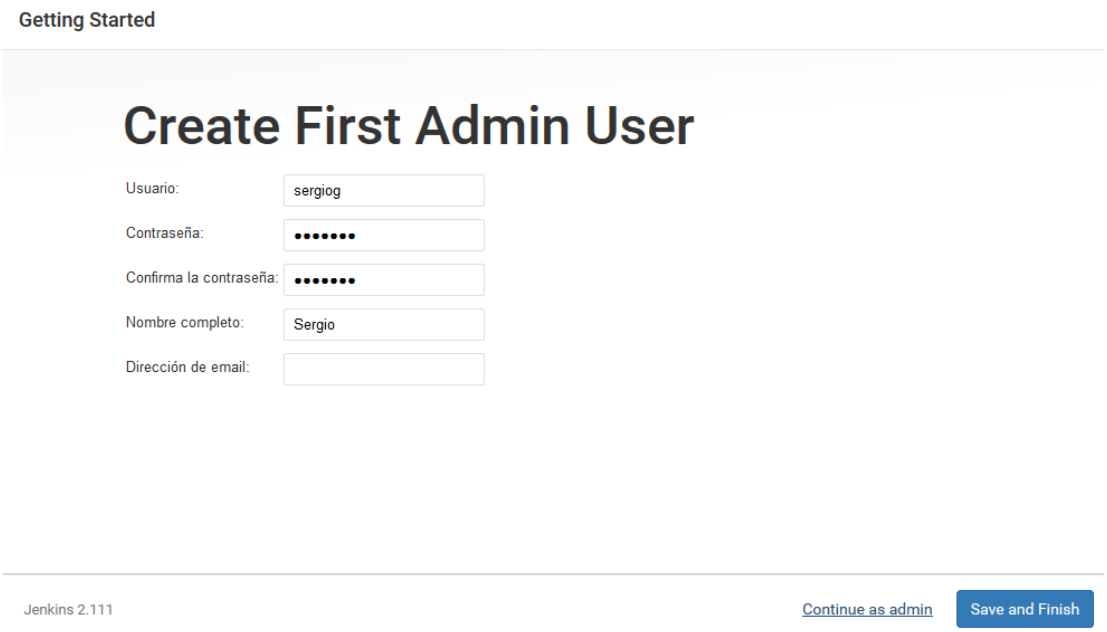

Figura 32: Jenkins instalación paso 4

Hasta que la instalación se complete:

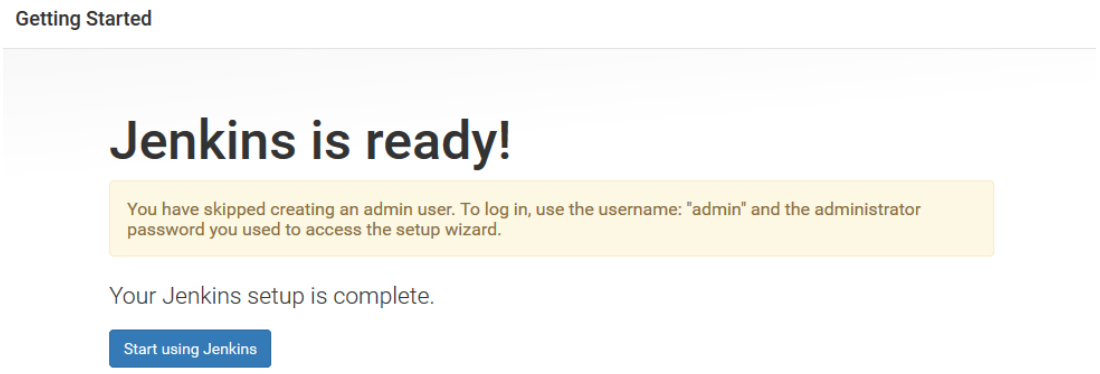

Figura 33: Jenkins instalación paso 5

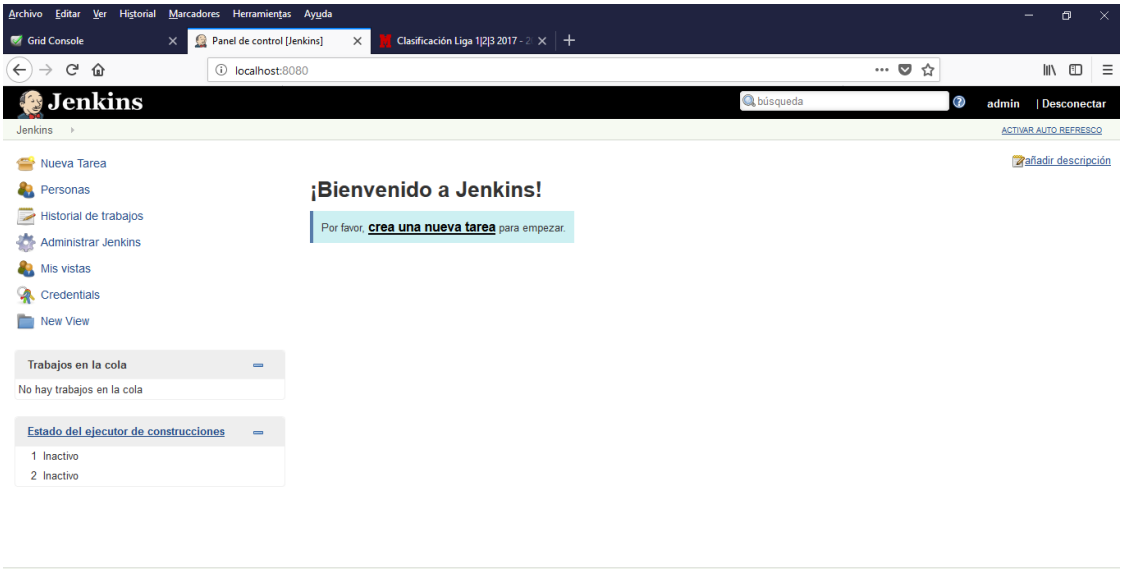

Figura 34: Jenkins página de Login

Una vez dentro es necesario realizar un par de acciones:

- Cambiar la contraseña de la cuenta administrador (admin), en el menú:

  Administrar Jenkins→Gestión de usuarios→ Opciones de la cuenta (admin)

  Aquí cambiaremos la pwd del user admin (dejaremos user:admin/pwd:admin) y Guardaremos.

- Instalar ciertos plugins extras necesarios:

  Administrar Jenkins→Administrar plugins→ Todos los plugins

  Aquí buscaremos los siguientes plugins e instalaremos sin iniciar:

  - Conditional Build Step
  - Powershell Plugin
  - Build With Parameters
  - Parameterized Trigger

- Aumentar el número de ejecutores en Jenkins

  Administrar Jenkins→Configurar el sistema→Número ejecutores: subir a 10

- Añadir Java y Maven en la configuración de Jenkins

  Administrar Jenkins→Global Tool Configuration:

  Aquí rellenar instalaciones de JDK y MAVEN con la info del path del sistema

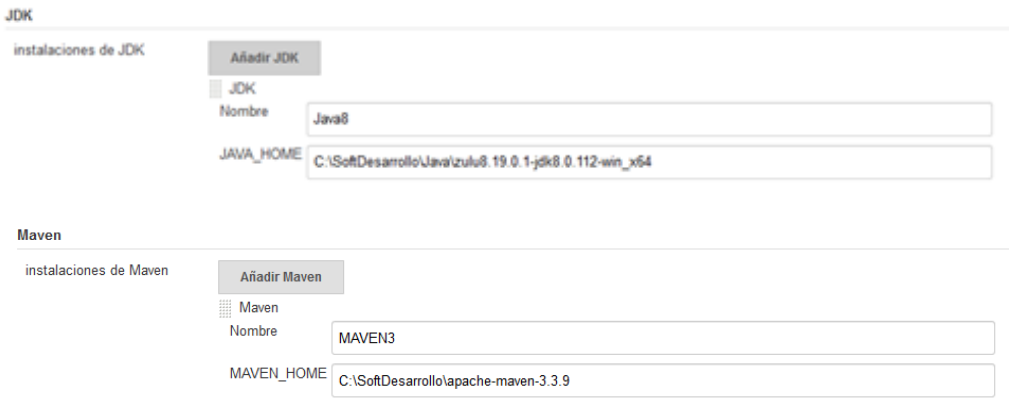

Figura 35: Configurar paths Java y Maven en Jenkins

Por último, habrá que situarse en el directorio “Jobs” de la instalación y copiar las carpetas adjuntadas en el repositorio: https://github.com/sglezcasco/sgc-tfgii/blob/master/jenkins_jobs

Reiniciaremos Jenkins (el proceso java -jar jenkins.war) y al logarnos con admin/admin obtendremos el siguiente panel principal con los Jobs por defecto ya cargados:

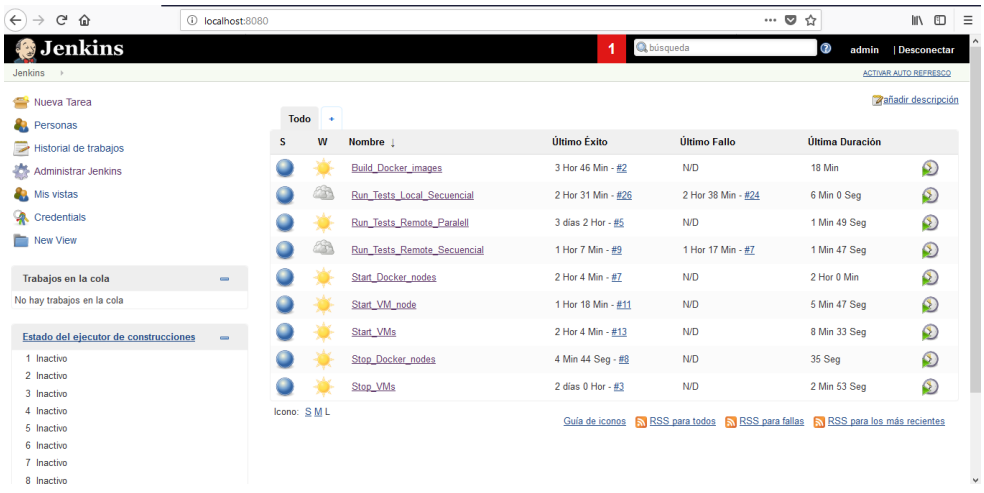

Figura 36: Jenkins. Panel de control principal

A.2 Manual de usuario

Una vez instalado y configurado Jenkins, accederemos a la herramienta vía web mediante la url: **localhost:8080** y nuestras credenciales de acceso: admin/admin

Figura 37: Jenkins. Panel de control principal

A continuación, explicaremos el flujo de ejecución y los jobs existentes en el panel:

1. **Build_Docker_Images**

Este job al ejecutarlo, construye las imágenes necesarias para nuestros contenedores Docker sobre Windows, en primer lugar, construye nuestra imagen **"base_server"** a partir de la imagen **"microsoft/windowsservercore"** proporcionada por Microsoft.

Posteriormente a partir de esa imagen base, crearemos el contenedor de selenium Grid, que levantará la parrilla donde se integren el resto de los nodos, y que llamaremos "**hub_server**".

Por último, también a partir de la imagen base, crearemos el nodo del navegador headless PhantomJS que llamaremos "**phantomjs_server",** que además deberá integrarse en la parrilla del "hub_server"

Todos los contenedores se crean sobre el directorio donde se encuentra el archivo de definición del contenedor [fichero Dockerfile] mediante el comando Docker:

docker build -t [name_image] .

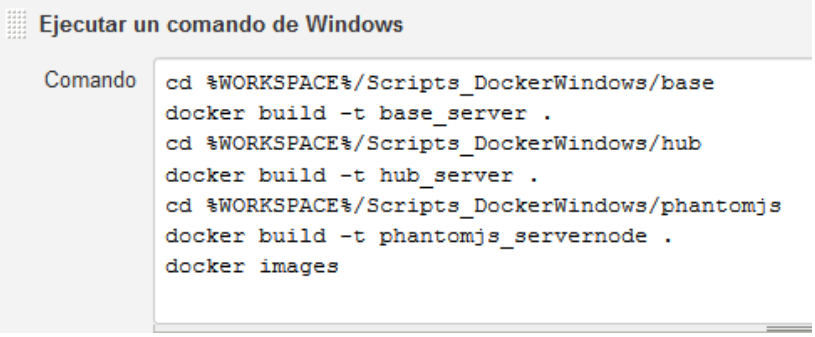

Figura 38: Script de construcción de imágenes Docker

Una vez tengamos todos los contenedores creados el resultado se podrá observar mediante línea de comandos:

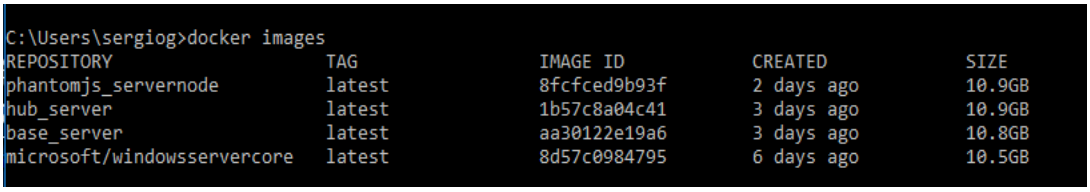

Figura 39: Docker. Imágenes creadas

Es importante tener en cuenta que este job sólo será necesario ejecutarlo la primera vez, puesto que no es necesario eliminar las imágenes una vez creadas, tan sólo será necesario arrancarlas o pararlas.

2. Start_Docker_nodes

Este job al ejecutarse, levanta el contenedor donde se ejecuta selenium grid en la ip por defecto del equipo y el puerto 4444, y además levanta otro contenedor que contiene el navegador headless phantomJS y que se integra en la parrilla del selenium Grid.

Habrá **que incluir nuestra IP local en el atributo hub del dockerfile del PhantomJS.**

Para arrancar ambos contenedores se apoya en Docker Compose, existe un fichero de configuración llamado “docker-compose.yml” el cual arranca los contenedores mediante el comando: **docker-compose up**

Así pues, tras ejecutarse este job, se puede observar la ejecución de ambos contenedores:

```
C:\SoftDesarrollo\SeleniumDockerWindows>docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
13d129392741        phantomjs_servernode   "powershell -Comma..."   About a minute ago   Up 26 seconds       8083/tcp                 selenium-phantomjs
f7ca2f830a7b        hub_server             "powershell -Comma..."   About a minute ago   Up 25 seconds       0.0.0.0:4444->4444/tcp   selenium-hub
```

Figura 40: Docker images resumen

Y también que se habrá creado la parrilla y se ha integrado un nodo remoto con navegador PhantomJS, en nuestro caso en: http://192.168.43.247:4444/grid/console

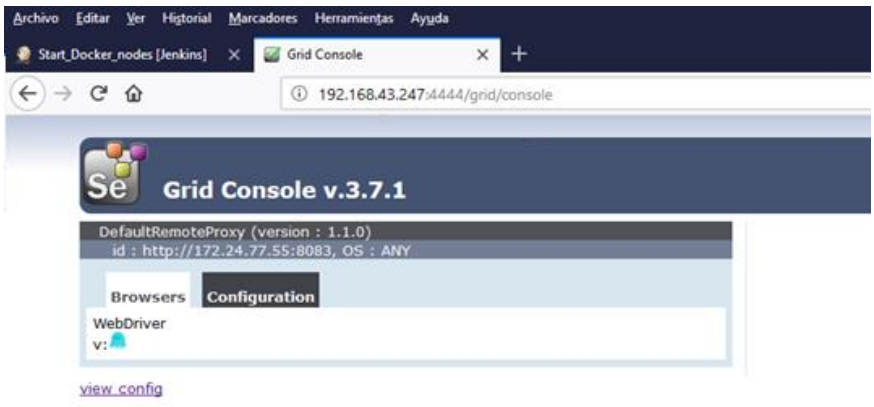

Figura 41: Selenium Grid con nodo PhantomJS

Este job se quedará en ejecución en Jenkins mientras los contenedores se estén ejecutando. Y finalizará una vez se ejecute el job “Stop_Docker_nodes”

3. Start_VMs

Al ejecutar este job, se iniciarán las 3 máquinas virtuales. Desde la herramienta Hyper-V Manager se observará que se han iniciado y también se podrá acceder al escritorio remoto.

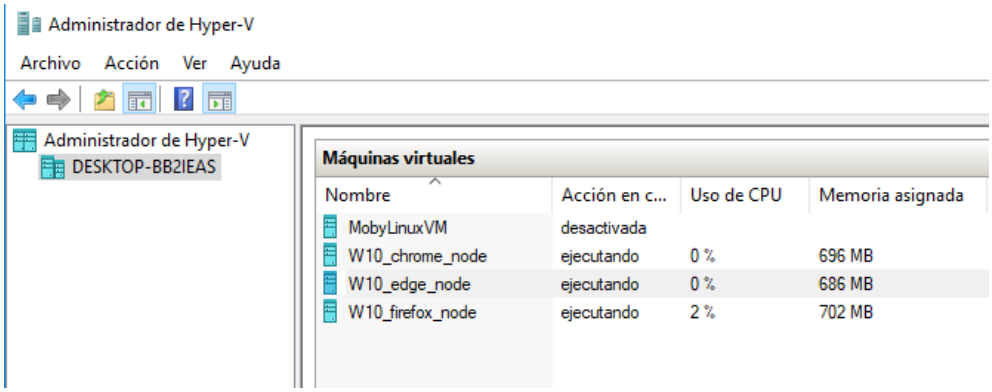

Figura 42: Hyper-V estado en ejecución

De igual modo, también es posible ver su estado mediante comandos powershell:

Start-VM -Name W10_firefox_node
Start-VM -Name W10_chrome_node
Start-VM -Name W10_edge_node

```
C:\WINDOWS\system32>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\WINDOWS\system32> Get-VM

Name             State   CPUUsage(%) MemoryAssigned(M) Uptime           Status                Version
----             -----   ----------- ----------------- ------           ------                -------
MobyLinuxVM      Off     0           0                 00:00:00         Funcionamiento normal 8.2
W10_chrome_node  Running 3           1536              00:01:36.8950000 Funcionamiento normal 8.2
W10_edge_node    Off     0           0                 00:00:00         Funcionamiento normal 8.2
W10_firefox_node Running 0           732               00:04:42.9170000 Funcionamiento normal 8.2
```

Figura 43: Powershell estado en ejecución

4. Start_VM_node

Posteriormente a la iniciación de máquinas, realizaremos la 2º fase de arranque en la cual, para cada nodo se levantará un nodo, que enganche con la parrilla de Selenium Grid y se integre con el resto de navegadores.

Para ello nos apoyaremos en scripts de powershell, que establecerán una sesión con la máquina virtual y se integrarán como nodo en Selenium Grid.

Habrá **que incluir nuestra IP local en el atributo hub de los scripts de start nodes en las maqunas remotas.**

Se establecerá una sesión entre la máquina huésped y las máquinas virtuales y posteriomente se integrará un nodo en selenium grid del modo:

Invoke-Command -Session $session_ff -FilePath . \start_firefox_node.ps1

A continuación, podemos ver el resultado de integración en la parrilla de Selenium Grid:

Figura 44: Selenium Grid con todos los nodos

Estos serán 3 procesos paralelos, uno por cada máquina virtual, que se quedarán en ejecución, mientras esos nodos esten activos. Estos Jobs finalizarán cuando se ejecute el job “Stop_VM_nodes”

5. Run_Tests_Local_Secuencial

Este job, se utiliza en la fase de desarrollo para ejecutar los tests de forma secuencial en el sistema operativo huésped. Levantado los navegadores y visualizando la navegación. Puede ser configurado para ejecutar por todos los navegadores de forma secuencial o sólo algunos navedores específicos.

Se ejecuta basándose en comandos Maven y ficheros TestNG, que apoyándose en la librería Selenium sobre Java y un driver especifico, levantará el navegador relacionado y ejecutará las pruebas sobre él.

mvn -Dtest=unir.net.tfgii.selenium.tests.Login,unir.net.tfgii.selenium.tests.Registro -DdriverProvider=MozillaFirefoxDriver -Dexecution=local -Dhost=localhost test

6. Run_Tests_Remote_Secuencial

Este job, se utiliza en la fase de desarrollo para ejecutar los tests de forma remota contra las máquinas virtuales o contenedores previamente arrancados. Puede ser configurado para ejecutar por todos los navegadores de forma secuencial o sólo algunos navedores específicos, eso sí, siempre en máquinas virtualizadas y remotas, y no permite la ejecución de forma paralela.

Se ejecuta basándose en comandos Maven y ficheros TestNG, que apoyándose en la librería Selenium sobre Java y un driver especifico, redirigirá a una parrilla de Selenium Grid que redirigirá el test contra la máquina virtual/contenedor correspondiente donde debe de ejecutar las pruebas.

mvn -Dtest=unir.net.tfgii.selenium.tests.Login,unir.net.tfgii.selenium.tests.Registro -DdriverProvider=MozillaFirefoxDriver -Dexecution=remote -Dhost=192.168.43.247 test

7. Run_Tests_Remote_Parallel

Este job, este job es el objetivo principal del proyecto para ejecutar los tests de forma remota y paralela contra las máquinas virtuales y contenedores previamente arrancados. No permite configuración, lanza todas las pruebas de forma paralela contra los diferentes navegadores alojados en máquinas virtuales y contenedores.

Se ejecuta basándose en comandos Maven y ficheros TestNG, que apoyándose en la librería Selenium sobre Java y un driver especifico, redirigirá a una parrilla de Selenium Grid que redirigirá el test contra la máquina virtual/contenedor correspondiente donde debe de ejecutar las pruebas.

Ejecutará un fichero testng específico donde se ha especificado las pruebas, mediante un perfil Maven:

mvn -PRemotoParalelo test

8. Stop_VMs

Este job para las 3 máquinas virtuales, finalizando los procesos Jenkins de arranque y finalizando los nodos de navegadores con interfaz gráfica de usuario de la parrilla de Selenium Grid, mediante la ejecución de:

Stop-VM -Name W10_firefox_node
Stop-VM -Name W10_chrome_node
Stop-VM -Name W10_edge_node

```
PS C:\WINDOWS\system32> Get-VM

Name             State CPUUsage(%) MemoryAssigned(M) Uptime   Status              Version
----             ----- ----------- ----------------- ------   ------              -------
MobyLinuxVM      Off   0           0                 00:00:00 Funcionamiento normal 8.2
W10_chrome_node  Off   0           0                 00:00:00 Funcionamiento normal 8.2
W10_edge_node    Off   0           0                 00:00:00 Funcionamiento normal 8.2
W10_firefox_node Off   0           0                 00:00:00 Funcionamiento normal 8.2
```

Figura 45: Powershell estado paradas

9. Stop_Docker_nodes

Este job para los 2 contenedores Docker, finalizando los procesos Jenkins de arranque del contenedor phantomjs_node y finalizando la parrilla de Selenium Grid, ejecutando el siguiente comando, en la ruta donde se encuentra el fichero Docker-compose.yml:

docker-compose down

```
PS C:\WINDOWS\system32> docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
NAMES
PS C:\WINDOWS\system32>
```

Figura 46: Docker procesos parados