UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC

Departament de Sistemes Informàtics i Computació

Universitat Politècnica de València

Integración de Herramientas en la nube para la Extracción Automatizada de Métricas para la Evaluación del Pensamiento Computacional.

Trabajo Fin de Master

Master Universitario en Computacion Paralela y Distribuida

Autor: Marcos Ramos Montes.

Tutor: José Damián Segrelles Quilis.

Directora Experimental: Amanda Calatrava Arroyo.

Curso: 2019 – 2020.

Tabla de Contenido

Índice de Figuras

RESUMEN.

En la actualidad, resulta de vital importancia una adecuada adquisición de competencias relacionadas con la programación y desarrollo de software, ya no sólo en el ámbito de carreras relacionadas con las TIC, sino de manera más general, en todas las titulaciones universitarias del ámbito STEM *(Scientific, Technology, Engineering and Mathematics).*

Con el advenimiento de los nuevos planes de estudio de Bolonia, el estudiante se ha convertido en el protagonista de su proceso de aprendizaje, siendo la tarea del docente, la de supervisar y ofrecer materiales de calidad que faciliten el trabajo autónomo de los estudiantes. Si bien es cierto, que la práctica y experimentación es de vital importancia en materias de ciencia e ingeniería, en general, este hecho, es especialmente relevante en materias relacionadas con la programación y el desarrollo del software, pues no hay mejor manera de adquirir los conocimientos y competencias marcadas por estas asignaturas, que llevándolos a la práctica y a escenarios lo más realistas posibles mediante el uso de instrumentos empleados en el mercado.

En este Trabajo Fin de Máster (TFM), se define una metodología de trabajo basada en un sistema de integración continua, para impartir materia relacionada con el diseño e implementación de software. Dicha metodología, pretende fomentar, tanto el trabajo individual, como en equipo de los estudiantes en materia de desarrollo y evaluación del pensamiento computacional. Además, este TFM presenta una serie de instrumentos que dotan al docente de una eficaz herramienta que permite orquestar toda la infraestructura de cómputo necesaria para la puesta en marcha de dicho método, así como, organizar las actividades de aprendizaje de carácter práctico de asignaturas relacionadas con la programación y desarrollo de software. Estos instrumentos, permitirán también al docente la posterior recogida de datos de las tareas desarrolladas, así como su posterior evaluación. Desde el punto de vista de los estudiantes, los instrumentos presentados en este TFM les permitirán autoevaluarse, obteniendo al instante retroalimentación sobre sus ejercicios de prácticas, todo ello, de forma ubicua y contando con una infraestructura altamente disponible, aprovechando los servicios ofrecidos por cualquier proveedor de *cloud* público actual. Mediante este enfoque, el estudiante no necesita esperar a la corrección del docente, agilizando de esta forma su trabajo y flexibilizando el mismo, ya que es posible acceder a la infraestructura necesaria para realizar las prácticas desde cualquier dispositivo conectado a internet.

Palabras clave: Jenkins, Infrastructure Manager, integración continua, Git, GitHub Classroom, Ansible, test unitarios, CUnit.

1. INTRODUCCIÓN Y BACKGROUND.

Actualmente, las competencias relacionadas con el diseño y programación de software son de vital importancia en los estudios universitarios, ya no sólo en el ámbito de las carreras relacionadas con las TIC, sino también en el ámbito de las enseñanzas STEM *(Science, Technology, Engineering and Mathematical),* dado que permiten desarrollar habilidades en los estudiantes relativas al pensamiento computacional *(Computational Thinking - CT)* [1] [2].

Por este motivo, todas las titulaciones STEM incorporan en sus planes de estudios asignaturas relacionadas con el diseño e implementación software, con el fin de introducir de forma práctica al alumnado en estas disciplinas. A la vista de este escenario, es esencial familiarizar al estudiante con instrumental específico del mundo real, tal como entornos de desarrollo integrado [3] *(Integrated Development Environment - IDE),* herramientas de *testing* de software, sistemas de integración continua o sistemas de control de versiones *(Version control System - SVN),* entre otros, favoreciendo el trabajo en equipo y facilitando la evaluación de las competencias transversales asociadas. Sin embargo, organizar e implementar la infraestructura para la impartición de estas materias mediante el uso de dicho instrumental, es una tarea ardua y retadora que acaba convirtiéndose en una carga de trabajo excesiva para el docente, dado que dichos entornos, por lo general, están más enfocados a producción de software y no para entornos docentes, provocando que su configuración y adecuación a dichos entornos no sea trivial. Además, una mala adecuación del entorno puede provocar rechazo por parte del alumnado si las herramientas no están bien configuradas y preparadas para las tareas encomendadas.

En este contexto, las tecnologías en la nube son un valioso recurso, ya que permiten automatizar el despliegue de entornos computacionales, ofreciendo gran flexibilidad para su orquestación. De esta forma, es posible preparar entornos personalizados para la realización de actividades dentro y fuera del aula, permitiendo acceso ubicuo y altamente disponible a las mismas, mediante proveedores de la nube.

Este TFM se centra en la incorporación a la práctica docente de Jenkins como sistema de integración continua. Jenkins deriva de un proyecto anterior denominado Hudson, el cual, es un servidor de integración continua de código abierto desarrollado por Kohsuke Kawaguchi, mientras trabajaba en Sun microsystems. Cuando Sun fue absorbida por Oracle, el proyecto Hudson fue renombrado como Jenkins. La integración continua es una metodología de desarrollo de software, en la cual, los desarrolladores integran su código con bastante frecuencia, obteniendo versiones ejecutables del mismo, que se someten a prueba inmediatamente, y, todo ello, de forma automatizada.

Esta metodología permite localizar y aislar errores rápidamente, asegurando que el software cumple con todas las métricas de calidad establecidas, evitando los catastróficos errores que podrían ocurrir si postpusiéramos las pruebas para la fase de producción.

Para trabajar con Jenkins, hemos de disponer del código a probar, así como de los test a realizar en repositorios. En este TFM, hemos utilizado GitHub como repositorio, el cual, se basa en Git[1] (un sistema de control de versiones distribuido [4]), facilitando y extendiendo su uso, ya que, en realidad, se trata de un servicio de hosting para repositorios Git, que mediante una interfaz web ofrece control de acceso y varias herramientas colaborativas.

Una de las labores que puede llevar a cabo Jenkins es la monitorización de sistemas de control de versiones, respondiendo a la ocurrencia de eventos específicos y avisando al desarrollador, por correo electrónico, entre otros medios.

Desde el propio Jenkins también es posible lanzar test de prueba para el software y observar el resultado, acceder a la documentación del proyecto e indicar qué métricas de calidad quieren utilizarse, entre otras muchas posibilidades. Dichas métricas pueden ser utilizadas para la evaluación de las competencias subyacentes de las actividades educativas realizadas mediante estos instrumentos.

Los trabajos desarrollados en este TFM, forman parte del proyecto AICO/2019/313 "Plataforma Colaborativa en la Nube para el Desarrollo y Evaluación de Competencias en las Enseñanzas STEM" financiado por la Conselleria D'innovació, Universitats, Ciència i Societat Digital de la Generalitat Valènciana, y que se está desarrollando actualmente en el grupo de Grid, Cloud y Computación de Altas Prestaciones (GRyCAP)[2] del Instituto de Instrumentación para Imagen Molecular (I3M) de la Universitat Politècnica de València (UPV). En el contexto del proyecto AICO/2019/313, se pretende implantar para la práctica docente el uso del sistema de integración continua Jenkins[3] [5], junto con GitHub[4], como herramienta de gestión de versiones de los desarrollos de los estudiantes y GitHub Classroom para la gestión de las tareas a encomendar a los estudiantes. Para la adecuación y configuración de todo este instrumental, se ha utilizado la herramienta Infrastructure Manager[5] (IM) [6] y desarrollado varios scripts bash [7], mediante los cuales, se realiza la configuración automatizada *ad-hoc* de Jenkins para la práctica docente. También se han implementado una serie de test unitarios que permiten la evaluación automática de los ejercicios de prácticas de los estudiantes y que se han integrado en el propio sistema de integración continua a través de una serie de scripts de configuración desarrollados.

[1] Git https://git-scm.com/

[2] Grupo de Grid, Cloud y Computación de Altas prestaciones perteneciente al Instituto de Instrumentación para Imagen Molecular https://www.i3m.upv.es/view.php/GRyCAP/Principal

[3] Jenkins User Documentation https://jenkins.io/doc/

[4] GitHub https://github.com/

[5] Infrastructure Manager https://www.grycap.upv.es/im/index.php

El presente TFM se estructura en los siguientes apartados:

- ✓ Objetivos generales y específicos, donde se marcarán los fines del proyecto, tanto generales, como aquellos específicos, que llevarán a la consecución de las metas generales del TFM.

- ✓ Escenario de referencia, donde describiremos el entorno docente a partir del cual se extraerán las necesidades de la infraestructura propuesta a desplegar y que será utilizada a partir del curso docente 2020/2021. De dicho escenario, también se extraerá la metodología docente a impartir en el aula.

- ✓ Estado del arte, donde realizaremos una breve introducción teórica de los principales conceptos tratados en este trabajo y sus herramientas asociadas. El fin de este apartado, es que sea un trabajo autocontenido que permita el entendimiento de todos los desarrollos realizados en el TFM.

- ✓ Material y métodos, donde realizaremos una descripción de alto nivel, incluyendo el instrumental específico necesario, de la infraestructura que pretendemos construir para dar soporte al escenario de referencia descrito, así como la metodología a emplear para su uso en las actividades educativas que se desarrollen sobre ésta.

- ✓ Resultados, donde mostraremos la receta de automatización en la nube que despliega y orquesta el sistema de integración continua y todo el instrumental necesario para la práctica docente según el escenario de referencia. También se presentarán los scripts de configuración desarrollados para la automatización de la configuración del sistema de integración continua, así como los test unitarios implementados e integrados en Jenkins y Github.

- ✓ Conclusiones y trabajos futuros, hablaremos de las conclusiones obtenidas y las dificultades encontradas en el presente trabajo, así como las líneas de trabajo futuro que pueden derivarse del mismo.

2. OBJETIVOS.

En este apartado vamos a describir los objetivos generales y específicos del presente TFM.

2.1. Objetivos generales.

En este TFM se marca como objetivo general desarrollar una herramienta que permita orquestar el despliegue automatizado de entornos en la nube con el fin de que doten al alumnado de la infraestructura necesaria (software y hardware) que dé soporte a instrumentos que posibiliten la autoevaluación de desarrollos

software, a la vez que defina un flujo de trabajo en las actividades educativas, mediante el cual, los estudiantes trabajen con instrumentos propios del ámbito profesional del desarrollo del software. Dichos instrumentos, deberán ser proporcionados por la propia herramienta a través de proveedores de la nube (públicos o privados).

Esta herramienta reducirá la carga de trabajo del docente, asociada al despliegue de la infraestructura necesaria para la impartición de las prácticas de laboratorio, facilitando, además, la obtención de métricas útiles para la evaluación de competencias del alumnado.

2.2. Objetivos específicos.

Para la consecución del objetivo general descrito, se han marcado los siguientes objetivos específicos:

1º. Desarrollar el software (scripts de configuración) necesario para facilitar el manejo de la herramienta de integración continua Jenkins, de forma que, su gestión en las prácticas docentes por parte del profesor sea sencilla y el acceso por parte del alumno sea restringido y de forma aislada, pudiendo acceder sólo a sus propios desarrollos software.

2º. Desarrollar una receta para automatizar el despliegue de la infraestructura (software y hardware) completa requerida para proveer en la nube de un sistema de integración continua enfocado a la práctica docente.

3º. Diseñar e implementar test unitarios, además de integrarlos en el sistema de integración continua Jenkins, para dar soporte a un caso real circunscrito al escenario de referencia que se describe en el TFM.

4º. Automatizar la configuración del sistema de integración continua Jenkins para dar soporte a la ejecución automática de test unitarios.

5º. Integrar en el flujo de trabajo del alumnado un sistema de control de versiones ampliamente utilizado en el mundo laboral, como es Git. También, se utilizará GitHub Classroom[6] [8] dado que facilita la gestión de múltiples repositorios, lo que es muy útil para el trabajo en el entorno docente.

6º. Proporcionar retroalimentación en los desarrollos de prácticas a través de Jenkins, de forma que, el alumnado pueda conocer al instante, si su código es correcto o no, sin tener que esperar la corrección del docente.

7º. Implementar la infraestructura propuesta en la nube, ya que ofrece acceso ubicuo y disponibilidad 24x7.

8º. Desarrollar un recurso docente que automatice la configuración y orquestación en la nube de todas estas herramientas y que facilite la labor del personal docente.

[6] GitHub Classroom https://Classroom.github.com/

3. ESCENARIO DE REFERENCIA.

En este apartado vamos a describir el entorno que se ha tomado como escenario de referencia para la realización del presente TFM.

El escenario de referencia tiene lugar en la Escuela Técnica Superior de Ingeniería del Diseño (ETSID) de la Universitat Politècnica de València (UPV), concretamente, en la asignatura Informática, la cual, es de carácter obligatorio y se imparte durante el primer cuatrimestre del primer curso del Grado en Ingeniería Electrónica Industrial y Automática (GEIA). La asignatura se divide en dos bloques, por una parte, 3 créditos a impartir mediante clases teóricas (2 horas semanales), y, por otra parte, 3 créditos a impartir mediante prácticas de laboratorio (2 horas semanales) a lo largo de 16 semanas.

Las sesiones de prácticas de laboratorio son de vital importancia en el proceso de aprendizaje del alumnado, pues permiten la adquisición de habilidades y destrezas de programación, permitiendo a los estudiantes poner en práctica los conocimientos adquiridos durante las clases de teoría mediante software específico. Actualmente, los estudiantes resuelven los problemas planteados en una serie de boletines, implementando programas en lenguaje C, que pueden desarrollar de forma presencial o autónoma fuera del aula, utilizando para ello, un entorno de desarrollo integrado orientado al trabajo con los lenguajes C/C++ como son los IDE Dev C++[7] o Code::Blocks[8], entre otros. Estos entornos son compatibles con una amplia variedad de compiladores. En la asignatura se hace uso del compilador gcc, el cual, está disponible para los sistemas operativos más comunes.

La metodología de trabajo actual está lejos de un entorno de desarrollo profesional, dado que los estudiantes no hacen uso de instrumentos ampliamente utilizados en el ámbito profesional del desarrollo de software, como son los sistemas de control de versiones o las herramientas de testeo e integración continua. Debido a ello, surge la necesidad de modificar la metodología de trabajo actual, mediante nuevos mecanismos que permitan familiarizar al alumnado con el trabajo profesional en el ámbito del desarrollo software mediante instrumentos específicos.

Una dificultad añadida en este escenario, es que algunos estudiantes pueden encontrar dificultades a la hora de preparar el entorno de trabajo necesario para realizar las prácticas de la asignatura fuera del laboratorio, bien por que requiera llevar a cabo tareas de configuración complejas, por incompatibilidades de software e incluso por los requerimientos hardware del software a utilizar. Una solución a este problema es proporcionar todas estas herramientas a través de la nube, aprovisionando los recursos hardware y software necesarios sobre proveedores Cloud para que sean accesibles desde cualquier dispositivo.

[7] DEV-C++ IDE https://www.bloodshed.net/devcpp.html
[8] Code..Blocks IDE http://www.codeblocks.org/

4. ESTADO DEL ARTE.

En este apartado vamos a presentar aquellas tecnologías empleadas en la realización del TFM. Para cada una de ellas, realizaremos una breve introducción teórica con el fin de que sea lo más autocontenido posible, de manera que resulte más sencillo explicar los conceptos que se desarrollarán en los siguientes apartados. Concretamente, abordaremos los siguientes aspectos:

- ✓ Fundamentos y herramientas de integración continua.
- ✓ Gestión organizativa de repositorios Git con GitHub y GitHub Classroom.
- ✓ Test unitarios: concepto y alternativas disponibles.
- ✓ Orquestación y despliegue de infraestructuras en la nube.

4.1. Fundamentos de integración continua.

Este concepto apareció por primera vez en 2004 [9] y describe una metodología que consiste en realizar integraciones y compilaciones automáticas del código de un proyecto, o de partes del mismo, con mucha frecuencia, con el objetivo de poder detectar y corregir errores en las primeras etapas del ciclo de desarrollo de software. El principal motivo que llevó a la aparición de esta nueva metodología, es que uno de los problemas de las metodologías de desarrollo de software clásicas, es precisamente, que el software se prueba en etapas avanzadas del ciclo de desarrollo, con lo que la aparición de errores en el código puede requerir de una vuelta atrás impracticable en muchos casos. En la Figura 1, extraída del trabajo recogido en [10], podemos observar cuál es el flujo de trabajo que se sigue en un sistema de integración continua.

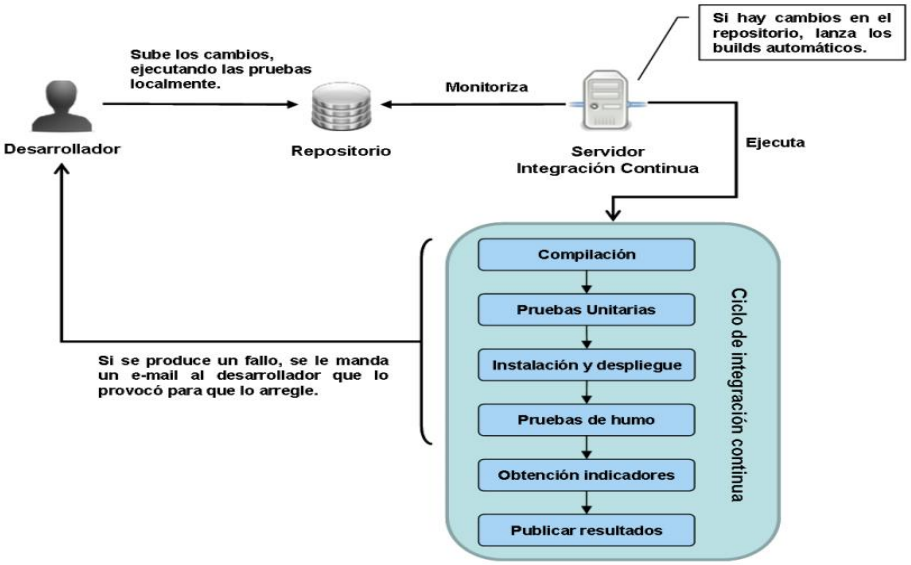


Figura 1. Flujo de trabajo en sistemas de integración continua [10].

Normalmente, el trabajo con un sistema de integración continua, se divide en las siguientes etapas:

a) El desarrollador prueba el software en su equipo local y cuando está libre de errores sube el código fuente a un repositorio gestionado por un sistema de control de versiones, como, por ejemplo, Git, Subversion[9], etc.

b) El servidor de integración continua comprueba si ha habido cambios en el repositorio de código fuente.

c) En caso de detectar cambios en el repositorio, se vuelve a compilar el código del proyecto de forma automática.

d) Se lanzan los test unitarios sobre el código compilado, informando de si se han ejecutado correctamente o no.

e) Se realiza el despliegue del software.

f) Tanto el código fuente, como los binarios, las bibliotecas generadas, junto con los resultados de los test unitarios, etc., quedan a disposición del equipo de desarrollo.

g) Se realizan pruebas de humo que básicamente sirven para comprobar que el software en su conjunto lleva a cabo su funcionalidad básica.

h) En caso de que alguno de estos pasos falle, se notificará de ello al desarrollador responsable de ese código por correo electrónico.

i) En su caso, se obtienen indicadores mediante las métricas de calidad que se hayan definido.

j) Se publican los resultados.

Esta metodología de trabajo aporta los siguientes beneficios:

- ✓ Detección temprana de errores en el desarrollo de software.
- ✓ Disminuye el impacto que supone añadir nueva funcionalidad.
- ✓ Cada vez que hay un cambio en el código se ejecutan test unitarios que los verifican.
- ✓ Se pueden obtener, de forma inmediata, métricas de calidad que nos permitan estimar el impacto de las modificaciones realizadas.

[9] Apache Subversion https://subversion.apache.org/

Como desventajas, podemos citar las siguientes:

- ✓ El temor de que el código sea erróneo puede hacer que los desarrolladores no integren el código con la periodicidad deseable.
- ✓ Es necesaria una adecuada asignación de roles para evitar problemas organizativos y de gestión.
- ✓ Es necesario definir adecuadamente el ciclo de integración continua.

En este TFM, se va a definir un nuevo flujo de trabajo para las sesiones prácticas del escenario de referencia descrito anteriormente, basándose en un sistema de integración continua. De esta forma, los estudiantes pueden crear programas e ir testándolos de forma continua y obtener métricas que les permitan valorar lo que están haciendo, obteniendo una retroalimentación continua de su trabajo.

4.1.1. Herramientas de integración continua.

Como ya se ha indicado, las herramientas de integración continua permiten compilar el código de un proyecto software y someterlo a prueba con bastante frecuencia con el objetivo de detectar errores en las primeras etapas del ciclo de desarrollo del software.

En la literatura, existe una gran variedad de este tipo de herramientas. Las más relevantes quedan recogidas en la Tabla 1.

Producto	Entrega continua	Alojamiento en nube	Licencia	Características	URL
Jenkins	Sí	Sí	MIT	-Gran cantidad de plugins -Dispone de API REST, interfaz web y CLI	https://www.jenkins.io/
Travis CI	No	Sí	MIT	-Multiplataforma -Compatible con GitHub	https://travis-ci.org/
Bamboo	Sí	Sí	Propietaria	-Gran cantidad de características -Gratuito sólo para proyectos de código abierto	https://www.atlassian.com/es/software/bamboo
GitLab CI	Sí	Sí	MIT/EE	-Compatible con otros productos Atlassian	https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/
Circle CI	Sí	Sí	Propietaria	Fácil utilización	https://circleci.com/
Cruise Control	No	No	BSD	Totalmente gratis	http://cruisecontrol.sourceforge.net/
Codeship	Sí	Sí	Propietaria	Dispone de versión básica y profesional	https://codeship.com/

Tabla 1. Comparativa sistemas de integración continua.

En este TFM hemos utilizado Jenkins como herramienta de integración continua, debido a que es altamente personalizable mediante plugins, siendo, además, la herramienta de trabajo utilizada en el grupo de investigación GRyCAP, donde se ha realizado el trabajo.

Como ya dijimos anteriormente, una de las principales características de los sistemas de integración continua, es que permiten monitorizar el estado de los repositorios de código realizando ciertas acciones en respuesta a la ocurrencia de ciertos eventos. Concretamente, en este trabajo hemos hecho uso del plugin "Gitlab Hook Plugin"[10] que permite que Jenkins lance una tarea[11] cada vez que un estudiante modifica el contenido de su repositorio GIT y realiza un *"pull"* del repositorio, pudiendo también lanzar la ejecución de los test unitarios manualmente, cada vez que el estudiante lo considere.

4.2. Gestión organizativa de repositorios Git con GitHub y GitHub Classroom.

Como ya hemos indicado, en este TFM hacemos uso de Github como repositorio de código.

Además de las cuentas de usuario, GitHub ofrece cuentas para organizaciones, las cuales, representan un grupo de usuarios que comparten proyectos y permite gestionar a estos usuarios como equipos de trabajo. Así, el docente puede crear una organización específica para la asignatura en GitHub que servirá para agrupar los repositorios de prácticas de todos los estudiantes de forma centralizada.

Para facilitar al docente la gestión centralizada de los repositorios de los estudiantes creados dentro de la organización, en este TFM hacemos también uso de la herramienta GitHub Classroom. Esta herramienta está diseñada para facilitar el uso de Github en el aula, permite la gestión simplificada del reparto de las tareas y generación de los repositorios de cada estudiante asociados a estas tareas. Para guiar al alumnado en cada práctica, el docente puede crear un repositorio base que servirá como plantilla de la tarea a resolver y que contendrá el código de partida para la realización de los ejercicios planteados en las prácticas. Este repositorio plantilla, se asocia a las tareas de GitHub Classroom, de modo que, cuando los estudiantes aceptan la tarea, se clona automáticamente un repositorio privado del repositorio plantilla para el estudiante, con el fin de que éstos, puedan realizar las actividades encomendadas en la tarea.

Otra ventaja de utilizar GitHub Classroom es que proporciona un mecanismo denominado *roster,* el cual, permite establecer una asociación entre los repositorios GitHub de los estudiantes y sus direcciones de correo electrónico. Esto permite establecer fácilmente la autoría del contenido de cada repositorio. Con lo mencionado en este apartado, si utilizamos estas herramientas, estaríamos dando cumplimiento a lo indicado en el quinto objetivo específico de este trabajo, relativo a la integración en el flujo de trabajo del alumnado del sistema de control de versiones GitHub y GitHub Classroom.

[10] Gitlab Hook plugin https://plugins.jenkins.io/gitlab-hook/

[11] En este caso, tarea se refiere a la ejecución de un test unitario.

4.3. Test unitarios: concepto y alternativas disponibles.

Las pruebas unitarias, también conocidas como test unitarios, las lleva a cabo el desarrollador y su objetivo no es otro que garantizar que el código se comporta de acuerdo con las especificaciones del programa.

Debemos huir de un enfoque tradicional en el que las pruebas se hacen "a mano" y automatizar el proceso, realizando las pruebas dentro de un entorno de integración continua.

La implementación de pruebas unitarias no es en absoluto trivial, tanto es así, que Feathers [11], en el año 2004, definió una serie de criterios que nos permiten saber si una prueba unitaria es buena o no. Así pues, diremos que una prueba unitaria es mala sí:

- ✓ Requiere una configuración específica del entorno.
- ✓ Accede a discos, bases de datos o redes.
- ✓ No permite la ejecución concurrente de otros test unitarios.

Es habitual confundir las pruebas unitarias con las pruebas de integración. En el primer caso, se prueba cada componente del sistema de forma aislada, mientras que, en el segundo, se prueba el sistema completo.

Las pruebas unitarias permiten detectar errores más rápidamente, aún a costa de añadir más código al proyecto, que, por supuesto, también deberá mantenerse.

4.3.1. Herramientas para realizar test unitarios.

Si bien es cierto que en la actualidad existen gran cantidad de herramientas para llevar a cabo test unitarios[12], que muchas de ellas están orientadas al trabajo con C++ y Java, no ocurre lo mismo, si lo que queremos es trabajar con el lenguaje C, como es nuestro caso.

En este TFM se han analizado dos herramientas para el desarrollo de los test unitarios en C: CUnit[13] y CPPUnit[14]. Pero, descartamos el uso de CPPUnit, debido a que obliga a encapsular el código que se desea probar dentro de una clase[15], cosa que no sucede, en el caso de CUnit.

A continuación, presentamos un ejemplo sencillo del uso de CUnit. Supongamos que queremos generar un test unitario, partiendo de un código suministrado por el estudiante, como el que se observa en la Figura 2.

[12] Herramientas para unit testing https://es.wikipedia.org/wiki/Prueba_unitaria#Herramientas
[13] Herramienta para unit testing CUnit http://CUnit.sourceforge.net/
[14] Herramienta para unit testing CPPUnit https://freedesktop.org/wiki/Software/cppunit/
[15] Las prácticas se desarrollan en ANSI C.

```c
#include <stdio.h>

int main(void)
{
   int n;

   printf("Dame un numero: ");
   scanf("%d", &n);

   if(n % 2)
     printf("%d es impar\n", n);
   else
     printf("%d es par\n", n);

   return 0;

}
```

Figura 2. Código fuente suministrado por el estudiante.

Para generar los test unitarios, tenemos que preparar un fichero de código fuente en el que:

- ✓ Encapsular el código a testear dentro de una función (en este caso, hemos encapsulado el código a probar dentro de la función “esPar”).
- ✓ Generar un banco de pruebas.
- ✓ Añadirlo al registro de CUnit.
- ✓ Generar las aserciones deseadas.
- ✓ Indicar cómo queremos obtener o visualizar los resultados de los test.
- ✓ Eliminar las estructuras de datos asociadas a CUnit.

En nuestro ejemplo, todas estas tareas las lleva a cabo el código que podemos observar en las Figuras 3 y 4, que, como vemos, no es tan trivial como cabría esperar.

```c
#include <CUnit/CUnit.h>

int esPar(int);
int init_suite(void);
int clean_suite(void);
void test_es_par(void);

int main (void)
{
   CU_pSuite pSuite = NULL;

   /* Iniciar el registro de cunit */
   if (CUE_SUCCESS != CU_initialize_registry())
      return CU_get_error();

   /* Agregar banco de pruebas */
   pSuite = CU_add_suite("Basic_Test_Suite", init_suite, clean_suite);
   if (NULL == pSuite) {
      CU_cleanup_registry();
      return CU_get_error();
      }

   /* Agregar test al banco */
   if ((NULL == CU_add_test(pSuite, "test_funcion_esPar", test_es_par)))
   {
      CU_cleanup_registry();
      return CU_get_error();
   }

   /* Ejecutar test en modo texto */
   CU_basic_run_tests();

   /* Limpiar el registro de cunit y acabar */
   CU_cleanup_registry();
   return CU_get_error();
}
```

Figura 3. Test unitario con CUnit. Parte I.

```c
int init_suite(void)  { return 0; }
int clean_suite(void) { return 0; }


int esPar(int x)
{
  return (x % 2 == 0);
}

void test_es_par(void)
{
  CU_ASSERT(esPar(1)  == 0);
  CU_ASSERT(esPar(2)  == 1);
  CU_ASSERT(esPar(3)  == 1);
}
```

Figura 4. Test unitario con CUnit. Parte II.

A continuación, en la Figura 5, podemos ver la salida de la ejecución del test generado con CUnit. En este caso, hemos optado por una salida en modo texto, pero, la herramienta nos permite generar la salida en otros formatos, como, por ejemplo, XML, entre otros.

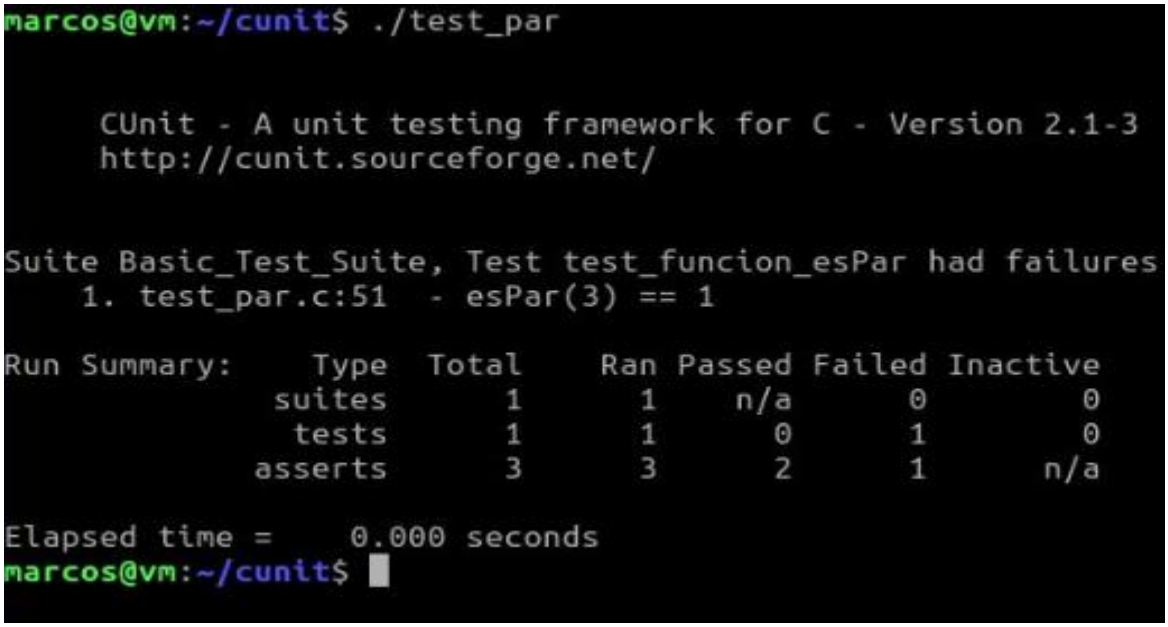


Figura 5. Test unitario creado con CUnit.

Como podemos observar, se ha ejecutado un test con tres aserciones (líneas 47 a 52), de las cuales, dos son ciertas y una es falsa, ya que, si nos fijamos en el código, veremos que, de las tres aserciones que hay, las dos primeras son ciertas, mientras que, la última es falsa.

Las dos herramientas analizadas, hacen necesario modificar el código fuente que se desea probar, lo cual, unido a la sencillez de los ejercicios de prácticas utilizados para realizar las pruebas, nos hizo adoptar la decisión de postponer el uso de CUnit para una fase más avanzada de este proyecto, concretamente, cuando abordemos la corrección automática de ejercicios que hagan uso de un diseño modular, ya que, los ejercicios utilizados en esta primera fase del proyecto son todos de carácter introductorio, por lo que, todo el código de los mismos se encuentra en la función *main*, razón por la cual, los test unitarios se han desarrollado en forma de shell scripts.

4.4. Orquestación y despliegue de infraestructuras en la nube mediante IM y Ansible.

La configuración automática de infraestructuras se basa principalmente en la definición de recetas que definen el estado que se desea que alcance una determinada instancia (o un grupo de ellas). Este concepto es de principal relevancia en el ámbito de la computación en la nube, o Cloud Computing [12], donde las infraestructuras se componen de máquinas virtuales configurables que se ejecutan sobre los recursos físicos de proveedores *cloud*.

Esto permite, por un lado, realizar el aprovisionamiento de recursos, y, por otro lado, la configuración automática de los mismos, por lo que se desacopla el hardware y el sistema operativo sobre el que se ejecutan las aplicaciones y la configuración deseada. Esta aproximación facilita el mantenimiento y la replicación de las infraestructuras, que pueden ser virtualizadas o físicas.

Ansible es una herramienta surgida en 2012 que permite la orquestación y administración de sistemas mediante la creación de recetas de configuración que permiten configurar recursos computacionales de forma determinista. Utiliza una aproximación de tipo *push,* lo que quiere decir que la máquina que ejecuta Ansible realiza conexiones SSH a las máquinas remotas para ejecutar los comandos necesarios para realizar la configuración especificada por el usuario. En la Figura 6, podemos observar el esquema de funcionamiento de Ansible.

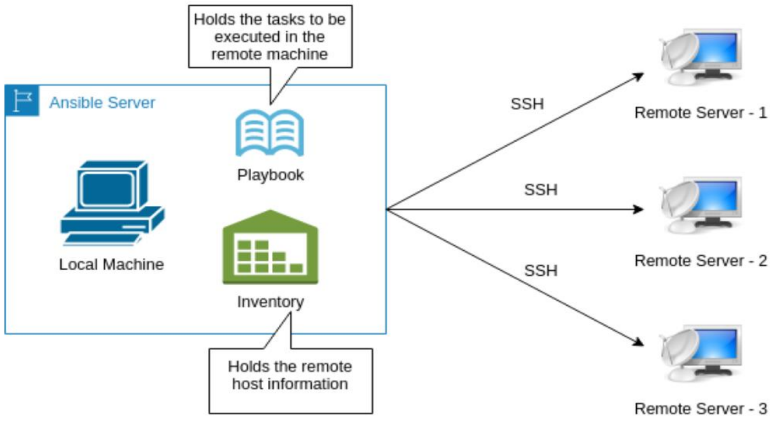


Figura 6. Arquitectura de Ansible[16].

Con el objeto de dar cumplimiento al segundo y séptimo objetivo específico de este TFM, relativo al despliegue y orquestación de infraestructuras de cómputo, utilizaremos la herramienta IM. Esta herramienta permite el despliegue y configuración de infraestructuras virtuales sobre recursos *cloud* y es compatible con la mayoría de los proveedores *cloud* actuales, tanto públicos (como Amazon Web Services, Microsoft Azure o Google Cloud Engine), como privados (OpenNebula y OpenStack, entre otros). Además, proporciona un lenguaje de descripción propio, conocido como Resource and Application Description Language (RADL), que facilita que los usuarios creen “recetas” compatibles con Ansible con el objeto de describir la infraestructura y la configuración requerida de la misma.

IM proporciona un conjunto de funciones que permiten a los usuarios orquestar, desplegar infraestructuras y obtener información sobre éstas. Además, su cliente permite añadir, eliminar y modificar las características de los componentes existentes, tanto software como hardware, y, todo ello, en tiempo de ejecución.

[16] Arquitectura de Ansible https://medium.com/swlh/automating-the-server-management-with-ansible-8fa464379fa9

En la Figura 7, podemos observar los distintos niveles de la arquitectura de IM.

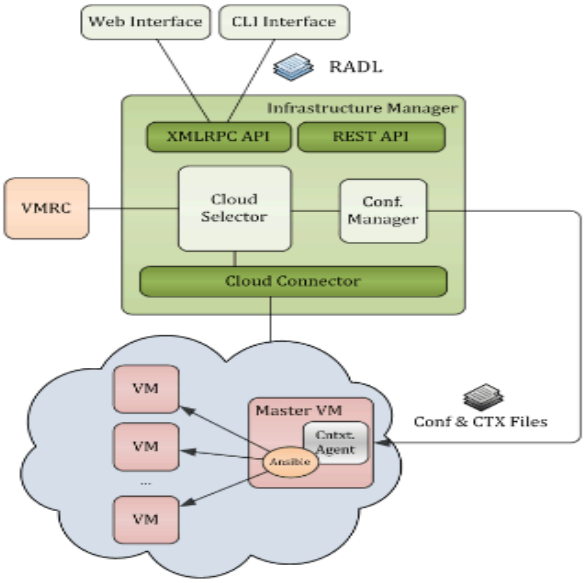


Figura 7. Arquitectura de IM[17].

5. MATERIAL Y MÉTODOS.

En este apartado vamos a realizar una descripción de alto nivel de la infraestructura de cómputo que se pretende construir y desplegar de forma automática para cumplir con el objetivo general de este TFM. A continuación, explicaremos la utilidad y el proceso de configuración de las herramientas utilizadas en la implementación de la infraestructura propuesta, teniendo en cuenta las características del escenario de referencia, descrito anteriormente.

La Figura 8, muestra un resumen del flujo de trabajo propuesto, donde en naranja se han etiquetado las acciones que realiza el docente y en verde las acciones que realiza el alumnado [13].

[17] Arquitectura de IM https://www.grycap.upv.es/im/overview.php

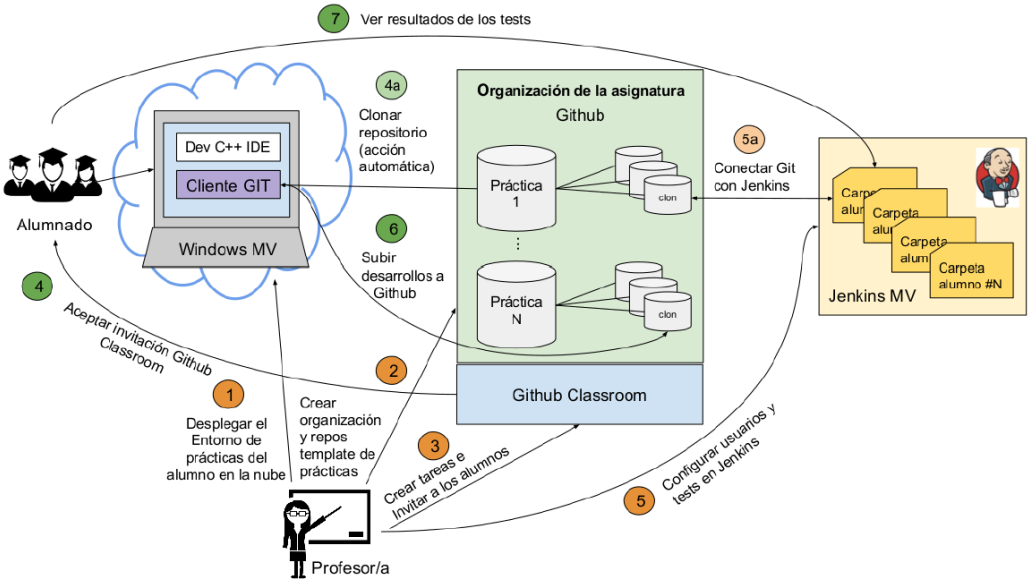


Figura 8. Metodología de trabajo [13].

En primer lugar, el docente tiene que desplegar en la nube el software necesario para llevar a cabo las prácticas, esto incluye el despliegue de una Máquina Virtual (MV) para cada estudiante configurada con todo el software necesario para ello. En concreto, el entorno de prácticas de nuestro escenario de referencia está compuesto por máquinas Windows configuradas con el IDE DevC++, así como por un cliente Git que facilita la interacción con GitHub desde un entorno gráfico, que resulta más sencillo e intuitivo para los estudiantes que la línea de comandos. Por otra parte, se despliega una MV con Jenkins adecuadamente configurado que servirá para llevar a cabo la corrección automática de los ejercicios de prácticas de cada estudiante. Las MV de los estudiantes y la MV que aloja Jenkins se encuentran en la misma subred.

El siguiente paso, consiste en configurar el espacio de trabajo en GitHub, creando por parte del docente la organización correspondiente para albergar los repositorios de las tareas de cada estudiante. También en este paso el docente crea los repositorios base de cada práctica propuesta que servirán como punto de partida para los estudiantes en cada tarea propuesta.

Seguidamente, el docente creará las tareas[18] para cada una de las prácticas en GitHub Classroom. Como hemos comentado anteriormente, esta plataforma permite automatizar tanto la creación de repositorios, como el control de acceso a los mismos, facilitando tanto la distribución del código, del que deben partir los estudiantes para la realización de las prácticas, como su posterior entrega. En la creación de las tareas, Github Classroom permite indicar el repositorio donde reside el código que se suministra a los estudiantes para la realización de la práctica. El docente obtendrá un enlace que deberá enviar a los estudiantes por correo electrónico para que acepten la invitación a la tarea. Esta tarea puede ser

[18] Assingment en GitHub Classroom.

individual o grupal. En este punto, lo único que necesita el docente es una lista con los correos de sus estudiantes.

Cuando el estudiante acepta la tarea, haciendo simplemente click en el enlace que le proporciona el docente en el correo, se clona automáticamente y de forma privada, el repositorio donde residen los recursos para la realización de la práctica correspondiente a la tarea en la cuenta del estudiante, dentro de la organización inicialmente creada por el docente.

Una vez creados los repositorios de cada estudiante a través de GitHub Classroom, el docente tendrá que configurar Jenkins para que cada vez que un estudiante suba código de una práctica a su repositorio, sea posible ejecutar automáticamente un test unitario que indique si el código del estudiante es correcto o no. Para ello, en primer lugar, el docente creará una carpeta[19] dentro de Jenkins por cada estudiante, la cual, contendrá una subcarpeta por cada ejercicio de la práctica que definirá una tarea Jenkins con el test unitario correspondiente, estando toda esta configuración asociada al repositorio del estudiante.

Estos test servirán como herramienta de autoevaluación al estudiante, a la vez que facilitarán la labor del docente, tanto en el proceso de corrección, como en la extracción de datos estadísticos útiles. Si bien, no debemos olvidar, que un escenario como el descrito aquí, requiere de una configuración de seguridad adecuada que impida que unos estudiantes puedan ver el código de otros.

Finalmente, los estudiantes podrán desarrollar sus prácticas de laboratorio integrando en su flujo de trabajo la utilización de GitHub, clonando localmente su repositorio, desarrollando los ejercicios de cada práctica, subiendo los cambios al mismo y accediendo a Jenkins para comprobar los resultados de los test. Estos test serán configurados para que se ejecuten cada vez que un estudiante haga cambios en su repositorio mediante las acciones *"push"*, aunque también pueden ser ejecutados bajo demanda, para lo cual, el estudiante debe conectarse a la interfaz web de Jenkins y lanzar manualmente el test unitario correspondiente al ejercicio que quiera evaluar.

Nótese el elevado número de tareas que el docente tiene que realizar "a mano" para poner en marcha un entorno de desarrollo con integración continua en el aula. Es por ello, que este TFM ha desarrollado una serie de utilidades que, alineadas con los objetivos específicos del mismo, ayudarán al docente a la puesta en marcha de la metodología descrita en el aula. Estas utilidades explican con detalle en el siguiente apartado de resultados.

[19] Vista en la terminología de Jenkins.

6. RESULTADOS.

En este apartado presentaremos los resultados que han hecho posible llevar a cabo un despliegue de una infraestructura (software, hardware y configuración) para dar soporte al escenario de referencia descrito.

En primer lugar, hemos creado una receta de Ansible para IM que despliega y configura una máquina con Jenkins instalado, junto con los scripts de integración y los test unitarios necesarios, para la corrección automática de las prácticas desde Jenkins. Para ello, se han desarrollado dos scripts que automatizan la configuración de Jenkins, para albergar las cuentas de los alumnos y las vistas personalizadas para cada uno de ellos, ofreciendo una eficaz herramienta, que sirve, por una parte, para la retroalimentación y autoevaluación del alumnado, y, por otra parte, facilita la labor del docente en lo que se refiere a la adopción de la metodología propuesta en el aula.

6.1. Receta IM.

A continuación, vamos a abordar el segundo y octavo objetivo específico propuesto en este TFM, para lo cual, vamos a explicar cómo hemos construido la receta IM que permite automatizar el despliegue de Jenkins. La receta está estructurada en dos partes, primero, se define la máquina, y, después, se configuran todas las aplicaciones requeridas, incluyendo Jenkins.

Veamos, en primer lugar, en la Figura 9, cómo se definen las características principales de la MV que constituirá el servidor Jenkins.

```
network publica (outbound = 'yes')
network privada ()

system front (
cpu.count>=1 and
memory.size>=4g and
net_interface.1.connection = 'publica' and
net_interface.0.connection = 'privada' and
net_interface.0.dns_name = 'jenkinserver' and
disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.image.url = 'one://ramses.i3m.upv.es/1262' and
disk.0.os.credentials.username = 'ubuntu' and
disk.0.os.credentials.password = 'yoY0&yo' and
disk.0.os.credentials.new.username = 'ubuntu' and
disk.0.os.credentials.new.password = 'yoY0&yo'
)

configure front (
@begin
- tasks:
  #Install java 8
  - name: Add java repository to ub14
    apt_repository:
      repo: 'ppa:openjdk-r/ppa'
    when: ansible_os_family == "Debian" and ansible_distribution_major_version == "14"

  - name: Ensure Java is installed
    apt: name=openjdk-8-jre-headless state=present update_cache=yes

  - name: Select java version 8 (Problems with java 9,10 and 12 in jenkins)
    alternatives:
      name: java
      path: /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

  #Install Jenkins (https://jenkins.io/doc/book/installing/)
```

Figura 9. Receta IM para Jenkins. Parte I.

En primer lugar, se definen dos interfaces de red virtuales, una denominada “publica” que conecta la máquina a Internet, y, otra denominada “privada”, que conecta la máquina a la red local, como es habitual en los servidores.

En la sección System front (líneas 4 a 17) se especifican las características de la máquina y las credenciales de acceso a la misma, como, por ejemplo, que la MV tendrá como mínimo una CPU (línea 5) y un mínimo de 4GB de memoria RAM (línea 6), que se instalará un sistema Linux, concretamente, una distribución Ubuntu cuya imagen se descargará de la URL “one://ramses.i3m.upv.es/1262” (línea 12). Por último, especifica que las credenciales de acceso a la máquina, una vez desplegada, serán usuario “ubuntu” y contraseña “yoY0&yo” (líneas 13 a 16).

A continuación, pasamos a ver la parte de la receta en la que se realiza la configuración del software (líneas 19 a 79). Un aspecto a destacar sobre este punto, es que Jenkins se actualiza con mucha frecuencia, por lo que, para evitar problemas de compatibilidad, lo que hemos hecho, es configurar la receta, de modo que, si partimos de una imagen que ya tiene Jenkins instalado, conservamos dicha instalación (línea 63), para lo que usamos la opción **`--no-upgrade`** del comando apt, como se puede ver, seguidamente, en la Figura 10, en la instalación de paquetes.

```
37  #wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
38  - name: Add Jenkins apt repository key.
39    apt_key:
40      url: "https://pkg.jenkins.io/debian/jenkins.io.key"
41      state: present
42
43  # sudo sh -c 'echo deb deb https://pkg.jenkins.io/debian binary/ > /etc/apt/sources.list.d/jenkins.list'
44  - name: Add Jenkins apt repository.
45    apt_repository:
46      repo: "deb https://pkg.jenkins.io/debian-stable binary/"
47      state: present
48      update_cache: true
49    when: ansible_os_family == "Debian"
50
51  - name: apt update
52    shell: apt-get update
53    args:
54      warn: false
55    when: ansible_os_family == "Debian"
56
57
58  - name: solve error installing Jenkins
59    shell: add-apt-repository universe
60    when: ansible_os_family == "Debian"
61
62  - name: Install Jenkins
63    shell: apt-get install -y --allow-unauthenticated --no-upgrade jenkins
64    args:
65      warn: false
66    when: ansible_os_family == "Debian"
67
68
69  - name: Install gcc
70    shell: apt-get install -y build-essential
71    args:
72      warn: false
```

Figura 10. Receta IM para Jenkins. Parte II.

Veamos ahora en la Figura 11 como se clona el repositorio (líneas 82 a 87), donde residen los test de integración, las pruebas unitarias, etc., y como se configuran adecuadamente los permisos para éstas (líneas 89 a 102).

```
73      when: ansible_os_family == "Debian"
74
75    - name: Install bc calculator
76      shell: apt-get install -y bc
77      args:
78        warn: false
79      when: ansible_os_family == "Debian"
80
81
82    # Install Autoasssesment scrips from Github
83    - name: Install Autoasssesment scrips from Github
84      git:
85        repo: https://github.com/grycap/ACTaaS
86        dest: /var/tmp/education/ACTaaS
87        version: master
88
89    - name: Set permisions for unittest scripts
90      file:
91        path: /var/tmp/education/ACTaaS/practices/P1/Face/unittests
92        state: directory
93        recurse: yes
94        owner: root
95        group: root
96        mode: '0777'
97
98    - name: Restore permisions for unittest directory
99      file:
100       path: /var/tmp/education/ACTaaS/practices/P1/Face/unittests
101       state: directory
102       mode: '0755'
103
104 @end
105 )
106
107 deploy front 1
108
```

Figura 11. Receta IM para Jenkins. Parte III.

Para clonar el repositorio público[20] ACTaaS[21] (donde residen los scripts de integración, los test unitarios y la receta IM que se han desarrollado en este TFM), utilizamos el módulo GIT de Ansible, para lo que basta con indicar la URL del repositorio, el directorio local donde se descargará (que debe estar vacío) y la rama, que, en este caso, es la principal o master.

Para configurar adecuadamente los permisos de las pruebas unitarias, una vez clonado el repositorio, hemos definido dos reglas con el módulo *file* de Ansible.

Con la primera (líneas 89 a 96), establecemos los permisos del directorio **`/var/tmp/education/ACTaaS/practices/P1/Face/unittests`** y todo su contenido en 777. La razón de utilizar esta máscara de permisos se debe a que el usuario efectivo que lanza los test desde Jenkins no tiene permisos fuera del espacio de trabajo *(workspace)* de Jenkins. La última regla (líneas 98 a 102) restablece los permisos del directorio que contiene los test unitarios a 755, de manera que, tenemos los permisos de los test unitarios debidamente configurados, sin modificar los permisos del directorio que los contiene.

6.2. Scripts de integración.

En este apartado vamos a abordar el desarrollo de dos scripts que nos facilitarán el trabajo de configuración con la herramienta de integración continua Jenkins, dando cumplimiento al primer objetivo específico del TFM.

[20] También es posible clonar un repositorio privado, pero, el procedimiento es distinto del mostrado aquí.

[21] Repositorio del grupo de Grid y Computación de Altas Prestaciones https://github.com/grycap/ACTaaS

El primer script, registerStudents.sh, cuyo código se muestra en las Figuras 12 y 13, crea los usuarios, la estructura de carpetas y establece los permisos de éstas dentro del entorno de Jenkins, bien leyendo de un fichero la lista de cuentas GitHub de los estudiantes o permitiendo la creación de una única carpeta, especificando una cuenta de estudiante, desde la línea de comandos. La sintaxis para la invocación de este script es la siguiente:

```
registerStudents.sh [-f <students_names_file>] [-j<Jenkins_URL>]
[-a <number_of_asssingments>] [-u <Jenkins_user>] [-p
<Jenkins_password>] [-s <single_student_account>]
```

Como se observa, el script también necesita conocer el número de prácticas que tendrá la asignatura, para crear una estructura de carpetas que albergará un test, por cada ejercicio de cada práctica.

Así, el script automatiza la creación de los usuarios en Jenkins y configura adecuadamente los permisos del entorno, generando un fichero de texto con sus credenciales de acceso que podrá ser consultado por el docente y que le servirá para dar a conocer a cada estudiante su cuenta de usuario en Jenkins.

La configuración del plugin Role-based Authorization Strategy (líneas 43 a 46) nos permite definir y asignar roles a cada usuario y carpeta.

Seguidamente, en las Figuras 12 y 13, se muestra el código más relevante de este script.

```
17 createStudent() {
18
19     #Create user
20     #NEW_PASS=$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1)
21     NEW_PASS="vamosacambiarla"
22     #devolver al profesor la lista de alumnos y su contraseña de jenkins creada de forma aleatoria
23     echo "${STUDENT},${NEW_PASS}" >> students_jenkins_accounts.txt
24     CMD="jenkins.model.Jenkins.instance.securityRealm.createAccount(\"${STUDENT}\",\"${NEW_PASS}\")"
25     echo $CMD | java -jar jenkins-cli.jar -auth ${USER}:${PASS} -s ${JENKINS_URL} groovy =
26     #>/dev/null
27
28     #Create the main folder
29     curl -s -X POST "${JENKINS_URL}/createItem?name=${STUDENT}
   &mode=com.cloudbees.hudson.plugins.folder.Folder&from=&json={"name":"$
   {STUDENT}","mode":"com.cloudbees.hudson.plugins.folder.Folder","from":"","Submit":"OK"}&Submit=OK" --user $
   {USER}:${PASS} -H "Content-Type:application/x-www-form-urlencoded"
```

Figura 12. Fragmento de código de registerStudents.sh. Parte I.

```
30
31 #Create the subfolders (We need to know how many practices are in the subject)
32     i="0"
33     while [ $i -lt $PRACTICES ]
34     do
35        curl -s -X POST "${JENKINS_URL}/job/${STUDENT}/createItem?name=practice${i}_${STUDENT}
   &mode=com.cloudbees.hudson.plugins.folder.Folder&Submit=OK" -H "Content-Type:application/x-www-form-urlencoded" --user $
   {USER}:${PASS}
36        i=$[$i+1]
37     done
38
39        #Create project roles for each student, to restrict the view of the folders
40        curl -s --user ${USER}:${PASS} ${JENKINS_URL}/role-strategy/strategy/addRole --data "type=projectRoles&roleName=$
   {STUDENT}&pattern=${STUDENT}&permissionIds=hudson.model.Item.Read,hudson.model.Item.Discover&overwrite=true"
41        curl -s --user ${USER}:${PASS} ${JENKINS_URL}/role-strategy/strategy/addRole --data "type=projectRoles&roleName=$
   {STUDENT}sub&pattern=${STUDENT}/.*&permissionIds=hudson.model.Item.Read,hudson.model.Item.Discover&overwrite=true"
42
43        #Assing global and project roles to the student
44        curl -s --user ${USER}:${PASS} ${JENKINS_URL}/role-strategy/strategy/assignRole --data
   "type=globalRoles&roleName=${ROLE_NAME}&sid=${STUDENT}"
45        curl -s --user ${USER}:${PASS} ${JENKINS_URL}/role-strategy/strategy/assignRole --data
   "type=projectRoles&roleName=${STUDENT}&sid=${STUDENT}"
46        curl -s --user ${USER}:${PASS} ${JENKINS_URL}/role-strategy/strategy/assignRole --data
   "type=projectRoles&roleName=${STUDENT}sub&sid=${STUDENT}"
47 }
```

Figura 13. Fragmento de código de registerStudents.sh. Parte II.

El segundo script, createTests.sh, es el encargado de crear, para cada práctica, los test que evaluarán cada ejercicio de la misma.

Este script puede, o bien, recibir un fichero que contiene una lista con la cuenta de GitHub de cada estudiante junto con la URL del repositorio de la práctica para la que se van a crear los test, o bien, se le puede pedir que cree los test para un único estudiante desde línea de comandos. A continuación, se muestra la sintaxis para invocar al script:

```
createTests.sh [-f <students_names_file>] [-j <Jenkins_URL>] [-a
<number_of_the_correspondin
g_asssingment>] [-n <name_of_the_exercise>] [-t
<template_of_the_exercise.xml>] [-u <Jenkins_user>] [
-p <Jenkins_password>] -s <student account> -g <github url> -i
<credentials Id>
```

Como se observa, ambos scripts, registerStudents.sh y createTests.sh, están totalmente parametrizados, por lo que, para gestionar el elevado número de opciones, en ambos casos, hemos utilizado la herramienta getops[22], que facilita el trabajo con argumentos de línea de comandos.

[22] Página de manual de getopts https://man7.org/linux/man-pages/man1/getopts.1p.html

Seguidamente, en la Figura 14, se muestra el código más relevante del script createTests.sh.

```
17 addstudent() {
18             EX="exercise"
19             EX="$EX$JOB_NAME"
20             COM="sh /var/tmp/education/ACTaaS/practices/P${PRACTICA}/Face/unittests/run_test_exercise${JOB_NAME}.sh"
21
22            # Modify the job template of Jenkins to include the repo of each student
23             cat ${JOB_TEMPLATE} | sed "s/#URL#/$(echo $GIT_URL | sed -e 's/\\/\\\\/g; s/\//\\\//g; s/&/\\\&/g')/g" > /tmp/
   ${JOB_NAME}_${STUDENT}
24             cat /tmp/${JOB_NAME}_${STUDENT} | sed "s/#EX#/$(echo $EX | sed -e 's/\\/\\\\/g; s/\//\\\//g; s/&/\\\&/g')/g" >
   /tmp/${JOB_NAME}_${STUDENT}
25             cat /tmp/${JOB_NAME}_${STUDENT} | sed "s/#COM#/$(echo $COM | sed -e 's/\\/\\\\/g; s/\//\\\//g; s/&/\\\&/g')/g"
   > /tmp/${JOB_NAME}_${STUDENT}
26             cat /tmp/${JOB_NAME}_${STUDENT} | sed "s/#ASIG#/$(echo $PRACTICA | sed -e 's/\\/\\\\/g; s/\//\\\//g; s/&/\\\&/
   g')/g" > /tmp/${JOB_NAME}_${STUDENT}
27             cat /tmp/${JOB_NAME}_${STUDENT} | sed "s/#CID#/$(echo $CID | sed -e 's/\\/\\\\/g; s/\//\\\//g; s/&/\\\&/g')/g"
   > /tmp/${JOB_NAME}_${STUDENT}
28
29             # Create job in Jenkins for each student
30             curl -s -X POST "${JENKINS_URL}/job/${STUDENT}/job/practice${PRACTICA}_${STUDENT}/createItem?name=${JOB_NAME}"
   --user ${USER}:${PASS} --data-binary "@/tmp/${JOB_NAME}_${STUDENT}" -H "Content-Type:text/xml"
31  }
```

Figura 14. Fragmento de código del script createTest.sh.

Las tareas o *Jobs* en Jenkins se traducen internamente a ficheros XML, por lo que este script, utilizando expresiones regulares, genera una tarea Jenkins personalizada a partir de una plantilla XML, sustituyendo cada etiqueta encerrada entre "##" por el valor correspondiente (valor adecuado para las etiquetas que se muestran en la Tabla 2), de modo que, no es necesario crear las tareas desde la interfaz de Jenkins por parte del docente. Este script facilita así al docente la creación con un solo comando de los test de una tarea para todos los estudiantes.

A continuación, podemos observar el contenido de la plantilla básica utilizada para crear los test, en la Figura 15, prestando especial atención a sus argumentos parametrizados. Esta plantilla evitará que tengamos que crear tareas desde la interfaz de Jenkins y proporciona los argumentos necesarios para la creación de las tareas, que básicamente son, las credenciales de GitHub a utilizar (línea 12), el comando a ejecutar en respuesta a un evento de tipo *pull* (en nuestro caso, el lanzamiento del test unitario que corresponda, línea 38), así como la URL del repositorio de la práctica (línea 11). Por último, la variable EX almacena la ruta del ejercicio que se va a testear (línea 37).

```xml
<?xml version='1.1' encoding='UTF-8'?>
<project>
  <actions/>
  <description></description>
  <keepDependencies>false</keepDependencies>
  <properties/>
  <scm class="hudson.plugins.git.GitSCM" plugin="git@4.2.2">
    <configVersion>2</configVersion>
    <userRemoteConfigs>
      <hudson.plugins.git.UserRemoteConfig>
        <url>#URL#</url>
        <credentialsId>#CID#</credentialsId>
      </hudson.plugins.git.UserRemoteConfig>
    </userRemoteConfigs>
    <branches>
      <hudson.plugins.git.BranchSpec>
        <name>*/master</name>
      </hudson.plugins.git.BranchSpec>
    </branches>
    <doGenerateSubmoduleConfigurations>false</doGenerateSubmoduleConfigurations>
    <submoduleCfg class="list"/>
    <extensions/>
  </scm>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
  <triggers>
    <com.cloudbees.jenkins.GitHubPushTrigger plugin="github@1.29.5">
      <spec></spec>
    </com.cloudbees.jenkins.GitHubPushTrigger>
  </triggers>
  <concurrentBuild>false</concurrentBuild>
  <builders>
    <hudson.tasks.Shell>
      <command>
gcc ./#EX#/#EX#.c -Wall -o #EX#_bin
#COM#
</command>
    </hudson.tasks.Shell>
  </builders>
  <publishers/>
  <buildWrappers/>
</project>

```

Figura 15. Plantilla parametrizada para Jenkins.

Esta tarea está configurada para ejecutarse cada vez que el estudiante hace un *push* en su repositorio GitHub, aunque también es posible lanzar la tarea a mano desde la interfaz web de Jenkins, dando de este modo cumplimiento, al cuarto y sexto objetivo específico de este TFM.

La Tabla 2 recoge el significado de los campos parametrizados en la plantilla.

Campo	Descripción
URL	URL del repositorio de la práctica
CID	Credenciales compartidas de GitHub
EX	Ejercicio que se va a testear
COM	Comando para lanzar el test

Tabla 2. Campos parametrizables en la tarea Jenkins.

Para generar la plantilla, fue necesario crear desde la interfaz web de Jenkins la tarea correspondiente a un ejercicio de una práctica y extraer el fichero XML correspondiente mediante el comando:

```
java -jar jenkins-cli.jar -s http://158.42.105.24:8080 -auth
@jenkins_secret get-job
unittests_P01_Basic_Secuencial_Programs/test_exercicie1 >
test_template.xml
```

6.3. Test unitarios desarrollados.

En este apartado vamos a abordar la implementación de los test unitarios, para lo cual, se han desarrollado una serie de shell scripts que generan la salida esperada del ejercicio y la comparan con la salida del código C entregado por el estudiante, dando de este modo, cumplimiento al tercer objetivo específico de este TFM.

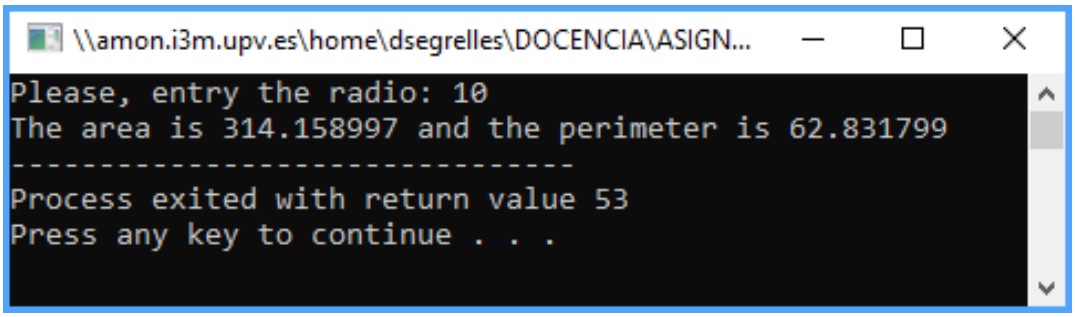
Estos scripts se han parametrizado para recibir la misma entrada que el ejercicio correspondiente, lo que permite reutilizarlos aún cuando se modifiquen los boletines de prácticas.

Además de los propios test, se han generado unos scripts “lanzadera”, de manera que, si deseamos cambiar los parámetros de entrada al test, sólo hay que modificar estos últimos. A continuación, se muestran un par de ejemplos extraídos de un boletín de prácticas de la asignatura:

Exercise 3. Design and implement a C program that compute the area and perimeter of a circle.

Note: PI number should be considered as the fixed value 3.14159

Example of Execution:



En este caso, el test unitario es el siguiente, que se observa en la Figura 16.

```
#!/bin/sh

if [ $# -ne 1 ]; then
    echo "test3 <radio>"
    exit
fi

radio=$1
pi=3.14159
area=$(echo "$pi*$radio*$radio"|bc -l)
perimeter=$(echo "2*$pi*$radio"|bc -l)

ent=$(printf 'The area is %f and the perimeter is %f' $area $perimeter)

echo "$@" > ent.txt
./exercise3_bin < ent.txt > sal.txt
sal=$(grep "The area is" sal.txt)

rm ent.txt sal.txt

areab=$(echo $ent | awk -F " " '{print $4}')
areac=$(echo $sal | awk -F " " '{print $4}')
error=$(echo "$areac - $areab"|bc -l)

res=$(awk -v cc="$areac" -v cb="$areab" 'BEGIN{ if(cc - cb < 0.5)  { print "OK!!" } else { print "ERROR!!" } }')
echo $res

```

Figura 16. Test unitario del ejercicio 3.

Y la lanzadera tiene el siguiente aspecto:

```
sh /var/tmp/education/ACTaaS/practices/P1/Face/unittests/test_exercise3.sh 10
```

Veamos otro ejemplo:

Exercise 5. Design and implement a C program that solves the next problem definition:

In the ETSID, every year are brought agendas for its students. The provider sells boxes of different size or single agendas. The size of the boxes is the follow:

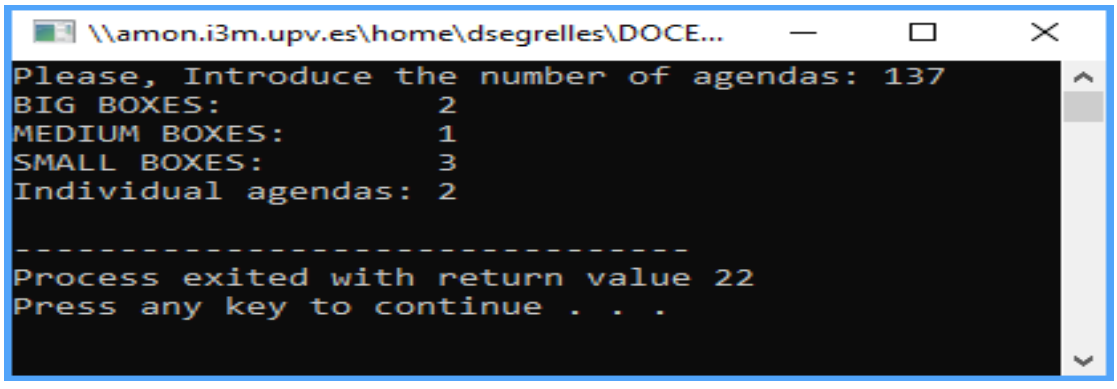
- Big box with 50 units.
- Medium box with 20 units.
- Small box with 5 units.
- Individual units.

When bigger is the box, this is cheaper.

The program has to request to the user the number of agendas that the ETSID wants to buy and compute number of boxes (big, medium, small) and individual agendas that supposes for the ETSID the minimum cost.

NOTE: The size of the boxes should be considered as a fixed value.

Example of execution:



Lanzadera y test unitario en la Figura 17:

```
sh /var/tmp/education/ACTaaS/practices/P1/Face/unittests/test_exercise5.sh 137
```

```
#!/bin/sh

if [ $# -ne 1 ]; then
   echo "test5 <number of agendas>"
   exit
fi

n_agendas=$1
BIG_BOX=50
MEDIUM_BOX=20
SMALL_BOX=5


n_big_boxes=$(echo "$n_agendas / $BIG_BOX"|bc)
aux=$(echo "$n_agendas % $BIG_BOX"|bc)
n_medium_boxes=$(echo "$aux / $MEDIUM_BOX"|bc)
aux=$(echo "$aux % $MEDIUM_BOX"|bc)
n_small_boxes=$(echo "$aux / $SMALL_BOX"|bc)
aux=$(echo "$aux % $SMALL_BOX" | bc)

ent=$(printf '%d BIG BOXES -- %d MEDIUM BOXES -- %d SMALL BOXES -- %d Individual agendas' $n_big_boxes $n_medium_boxes $n_small_boxes $aux)

echo "$@" > ent.txt
./exercise5_bin < ent.txt > sal.txt
sal=$(grep "BIG BOXES --" sal.txt)
rm ent.txt sal.txt
echo $ent
echo $sal
if [ "$ent" = "$sal" ]; then
   echo "Test OK!!"
else
  echo "Test ERROR!!"
fi

```

Figura 17. Test unitario del ejercicio 5.

En todos los casos, si se ejecuta el test correctamente, por la salida estándar se muestra un mensaje de conformidad, o un mensaje error, en caso de que el código del estudiante no genere el resultado esperado, como se observa en las Figuras 18 y 19.

Salida de consola

```
Lanzada por el usuario admin
Running as SYSTEM
Ejecutando.en el espacio de trabajo /var/lib/jenkins/workspace/marramon/practice1_marramon/3
using credential c541b288-fbec-4833-8cad-e34b60cb0af3
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/ARAGroupGIA/pr1-marramon # timeout=10
Fetching upstream changes from https://github.com/ARAGroupGIA/pr1-marramon
 > git --version # timeout=10
using GIT_ASKPASS to set credentials
 > git fetch --tags --progress https://github.com/ARAGroupGIA/pr1-marramon +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 510fc7c558cef1740fe7b2594e8862c2e50024d4 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 510fc7c558cef1740fe7b2594e8862c2e50024d4 # timeout=10
Commit message: "Add files via upload"
First time build. Skipping changelog.
[3] $ /bin/sh -xe /tmp/jenkins6391727806402269615.sh
+ gcc ./exercise3/exercise3.c -Wall -o exercise3_bin
./exercise3/exercise3.c:4:6: warning: return type of ‘main’ is not ‘int’ [-Wmain]
 void main(){
      ^
+ sh /var/tmp/education/ACTaaS/practices/P1/Face/unittests/run_test_exercise3.sh
OK!!
Finished: SUCCESS
```

Figura 18. Salida del test vista desde Jenkins. El código del estudiante es correcto.

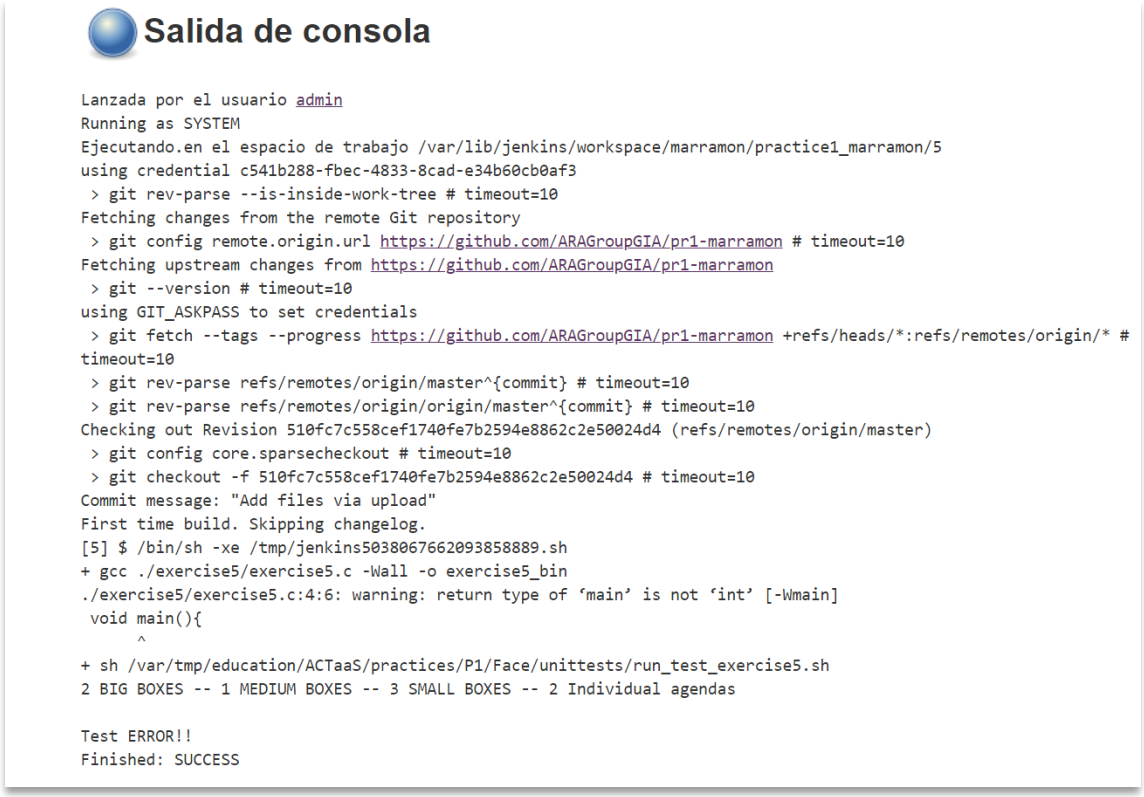


Figura 19. Salida del test vista desde Jenkins. El código del estudiante es incorrecto.

6.4. Acceso a Jenkins.

En este apartado ya tenemos la infraestructura desplegada, que tal y como podemos observar en las siguientes figuras, se basa en la definición de dos roles, el del docente/administrador y el del alumnado. Con ello, el objetivo es lograr que la interfaz de Jenkins ofrezca unas vistas que faciliten la rápida autoevaluación de los ejercicios de prácticas para los estudiantes y que permitan al docente obtener a golpe de vista el estado y la corrección de éstas.

En las Figuras 20 y 21 podemos observar la interfaz mostrada por Jenkins para los dos roles que hemos definido: el docente (Figura 20) y el alumnado (Figura 21), estos roles se han definido mediante el plugin Role-based Authorization Strategy, como ya se indicó anteriormente.

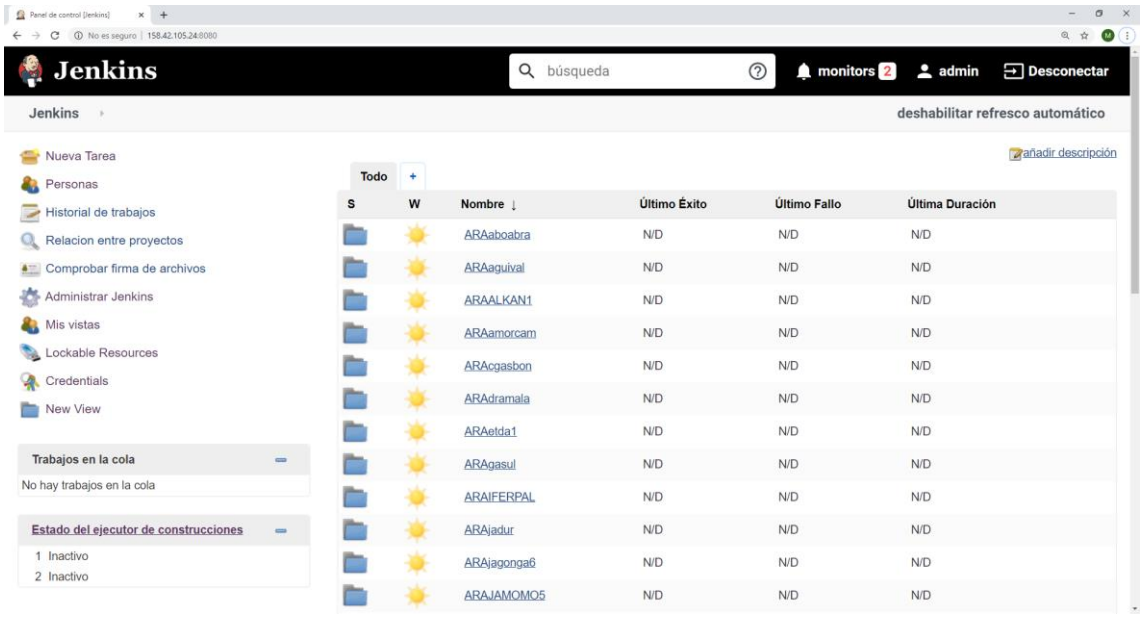


Figura 20. Panel de tareas de Jenkins visto desde la cuenta del docente/ administrador.

En la Figura 20, podemos observar una captura de pantalla del panel de tareas de Jenkins que encontraría el docente al acceder a la plataforma. En él, se observa una carpeta por cada estudiante, dentro de la cual, se alberga una estructura de carpetas por práctica, y, en cada carpeta de prácticas, se encuentran los test unitarios para cada ejercicio de la misma.

El docente debe tener acceso a las carpetas de todos los estudiantes, en cambio, cada estudiante sólo podrá ver el contenido de su carpeta personal y no la del resto de estudiantes. Por ello, el estudiante tendrá una vista muy similar a la recogida en la Figura 21, en la que verá una carpeta por cada práctica de la asignatura, que contiene en su interior un test para cada ejercicio de la misma, tal y como podemos observar, en la Figura 22.

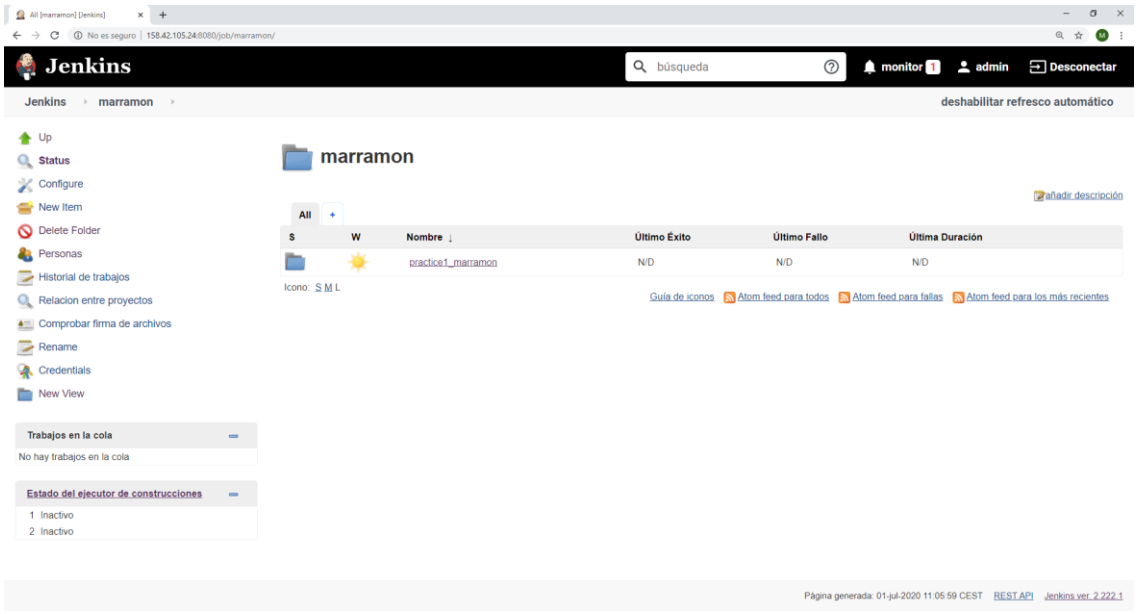


Figura 21. Panel de tareas de Jenkins visto desde la cuenta de un estudiante.

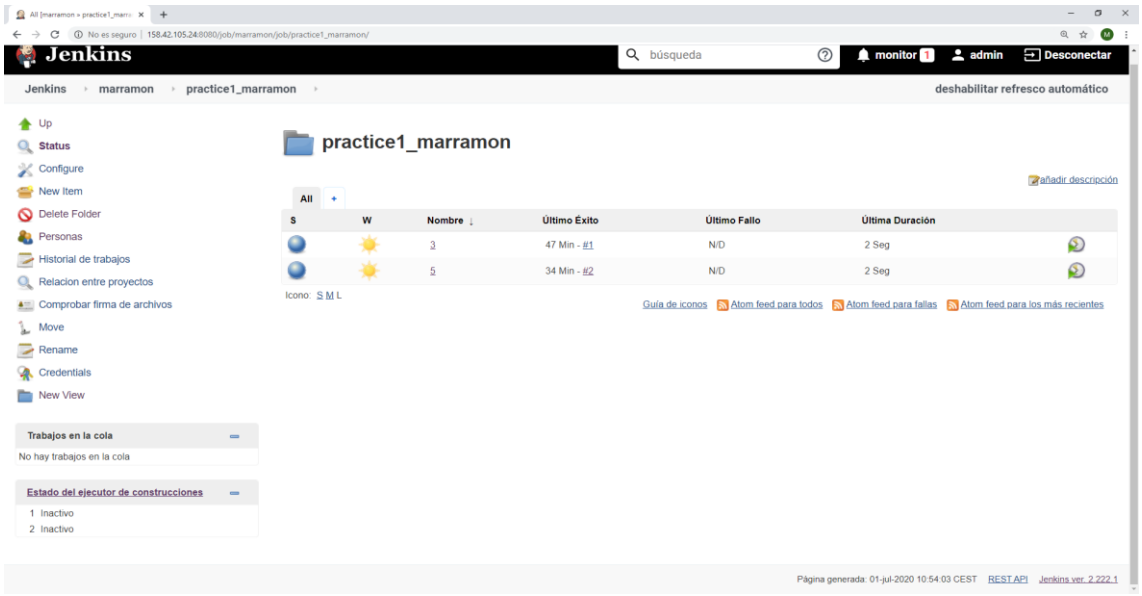


Figura 22. Vista de los test de cada ejercicio de la práctica 1 para el estudiante marramon.

Como se observa en la Figura 22, el estudiante puede ver la fecha del último test ejecutado correctamente y la fecha del último test fallido, así como el tiempo que ha tardado el último test en ejecutarse. Por último, hay que destacar que el botón de la flecha verde situado a la derecha de la Figura nos permite ejecutar el test manualmente tantas veces como queramos.

7. CONCLUSIONES.

En este TFM hemos presentado una metodología que permite desplegar en la nube y orquestar una infraestructura docente completa basada en un sistema de integración continua para una adecuada automatización de los procesos de enseñanza y aprendizaje en el ámbito de materias de introducción a la programación propias de primeros cursos universitarios de las enseñanzas STEM. La metodología basada en un sistema de integración continua, fuerza al estudiante a sumergirse en un entorno real de desarrollo software, utilizando instrumental específico ampliamente utilizado en el mercado laboral y científico.

Además, los scripts de configuración y recetas de despliegue permiten al profesorado realizar un despliegue rápido y sencillo a través de líneas de comando, haciendo viable su puesta en marcha.

Sin embargo, la consecución de los objetivos propuestos en este TFM no está exento de dificultades, como son, entre otras, las siguientes:

- ✓ El docente debe estar familiarizado con entornos de virtualización y con la computación en la nube para la creación de las máquinas virtuales necesarias para el despliegue del entorno de prácticas. Si bien es cierto, el uso del IM mitiga este hecho, dado que hace transparente el uso de diferentes proveedores *cloud*.

- ✓ Es necesaria experiencia con el entorno de integración continua Jenkins para la preparación de los test unitarios. Los scripts de configuración generados también han ayudado a mitigar los efectos de esta problemática.

- ✓ Es preciso tener conocimientos de Ansible para la creación y personalización de recetas para IM.

- ✓ Para implementar la infraestructura propuesta, es necesario el uso de varias herramientas aisladas, y, todo ello, desde líneas de comando, dificultando la adopción de este esquema para docentes que no tengan un perfil TIC.

- ✓ Es necesario que docentes y estudiantes tengan cuenta en GitHub, lo que puede dar problemas de suplantación de identidad, dificultad que se puede soslayar, mediante el uso de GitHub Classroom.

- ✓ Se precisa que los repositorios de los estudiantes sean privados, lo cual, podría suponer un sobrecoste para las entidades educativas, aunque existe la posibilidad de acceder a becas concedidas por GitHub.

8. TRABAJOS FUTUROS.

Como trabajo futuro se plantean varios objetivos. En primer lugar, la implementación de test unitarios de mayor complejidad que los descritos en este TFM. En segundo lugar, integrar toda la orquestación de la infraestructura propuesta en una aplicación web, lo que permitiría extender el uso de esta metodología a perfiles docentes distintos a las TIC. Sería posible dotar a esta aplicación de varias funcionalidades, como, por ejemplo, un editor de test unitarios que permitiría al docente generar éstos sin conocimientos de programación. Otras funcionalidades de la aplicación web, serían la generación asistida de tareas para el entorno de integración continua Jenkins (invocando de manera transparente a los scripts de integración desarrollados en este TFM, entre otros auxiliares), así como, un *front-end* que nos permitiera realizar toda la configuración administrativa de Jenkins (gestión de usuarios, asignación de roles, etc.) desde la aplicación web descrita, de manera que, el entorno de integración continua terminaría convirtiéndose en una caja negra para el usuario.

9. REFERENCIAS.

[1] KHINE, M.S. (ed.). Computational Thinking in the STEM Disciplines: Foundations and Research Highlights. Springer, 2018.

[2] VELÁZQUEZ-ITURBIED, J.A. “Towards an Analysis of Computational Thinking”, *International Symposium on Computers in Education (SIIE). IEEE, 2018,* pages 1- 6.

[3] PÉREZ LOZANO, M.D. Ingeniería del software y bases de datos: tendencias actuales. Universidad de Castilla La Mancha, 2000.

[4] CHACON, S. and STRAUB, B. Pro Git. New York, USA: Apress, 2019.

[5] SONI, M. and BERG A.M. Jenkins 2.x Continuous Integration Cookbook. Birmingham, UK: Packt Publishing, 2017.

[6] CABALLER, M.; BLANQUER, I.; MOLTÓ, G.; and DE ALFONSO, C. “Dynamic management of virtual infrastructures”, *Journal of Grid Computing, Volume 13, Issue 1, 2015,* pages 53 -70.

[7] MICHAEL, R.K. Mastering UNIX Shell Scripting. Indianapolis, USA: Wiley Publishing, Inc., 2008.

[8] ANGULO, M.A. and AKTUNC, O. “Using GitHub as a Teaching Tool for Programming Courses”, *ASEE Gulf-Southwest Section Annual Meeting, AT&T Executive Education and Conference Center, 2018.*

[9] BECK, K. and ANDRES, C. Extreme programming explained: Embrace change. Boston, USA: Addison - Wesley Professional, 2004.

[10] GARZÁS, J.; ENRÍQUEZ, J.A; IRRAZABAL, E. Gestión ágil de proyectos software. Madrid: Kybele Consulting, 2012.

[11] FEATHERS, M. Working effectively with legacy code. New Jersey, USA: Prentice Hall, 2004.

[12] MELL, P. and GRANCE, T. “The NIST definition of cloud computing”, *National Institute of Standards and Technology Special Publication 800-145, 2011*.

[13] CALATRAVA, A. y SEGRELLES QUILIS, J.D. “Metodología para el desarrollo y evaluación de competencias de programación software en la nube”, *Actas de la Jenui, vol. 4, 2019,* págs. 239 – 246.