Universitat Politècnica de València

Departamento de Sistemas Informáticos y Computación

# Plataforma en la Nube para la Gestión y Evaluación de Portafolios Académicos

Trabajo Fin de Máster

**Máster Universitario en Computación Paralela y Distribuida**

**Autor**: Franz Robert Miranda Vasquez

**Directores**: J. Damián Segrelles

Germán Moltó

[Curso 2015 - 2016]

# Abstract

This Master's Thesis describes the deployment of Virtual Environments on a Cloud in order to create and evaluate a digital portfolio of educational activities related to programming. The instructor declares the hardware, software and configuration requirements to use during an activity together with the repositories where the source-code is stored within a continuous integration environment. This introduces automated testing and the ability to extract quality indicators of the programming codes as well as to analyze the quality of the programs carried out by the students. The usage of a distributed computing infrastructure together with automatic configuration tools has resulted in the creation of a web application to deploy virtual infrastructures in the area of programming education.

**Keywords**: Virtual Environments, Educational Activities, Cloud, Resource Management.

# Resumen

Esta Tesis Final de Máster (TFM) describe el despliegue de Entornos Virtuales (EVs) sobre un Cloud, para facilitar la creación y evaluación de un portafolio digital de actividades educativas relacionadas con la programación. El instructor declara las necesidades hardware, software y de configuración a utilizar en una actividad así como los repositorios donde se almacenan los códigos fuente con un entorno de integración continua. Esto posibilita la verificación automática de código y la extracción de indicadores de calidad de los programas, así como el análisis de la calidad del código de las aplicaciones realizadas. El uso de infraestructuras de computación distribuida junto con herramientas de configuración automatizada ha resultado en la creación de una aplicación web para el despliegue de infraestructuras virtuales de soporte a la actividad docente en el contexto de la programación.

**Palabras clave**: Entorno Virtual, Actividades Educativas, Cloud, Gestión de Recursos.

# AGRADECIMIENTOS

Quisiera agradecer a mi familia por todo el cariño y comprensión en estos meses, y durante todos mis estudios. A los directores de mi proyecto, con los que las reuniones han sido productivas y enriquecedoras para mi desarrollo personal y profesional, guiándome en todo momento en la concepción del proyecto. Al GRyCAP (Grupo de Grid y Computación de Altas Prestaciones) por la cesión de la infraestructura para la ejecución de Entornos Virtuales sobre AWS que se describen en este documento, en el contexto del programa AWS Educate.

# Tabla de contenidos

## CONTENIDO

# Tabla de ilustraciones

# 1 INTRODUCCIÓN

Hoy en día, que los estudiantes en las Universidades adquieran experiencia en el uso de las herramientas que se emplean en el mundo empresarial es de vital importancia, tanto para su desarrollo personal como profesional, debido a que acerca a los alumnos a escenarios laborales más realistas desde las aulas, donde se les forma para su integración en el mercado laboral. Este hecho, lleva al instructor a la necesidad de proporcionar acceso a los alumnos de recursos específicos (hardware y software), los cuales se encargan de emular estos escenarios a través de simuladores, laboratorios virtuales o herramientas software específicas [1].

El alto coste de adquisición y mantenimiento de dichos recursos hace que los docentes de países no desarrollados no puedan dar a sus estudiantes la posibilidad de acceder a este tipo de entornos "on premises" [2], siendo esta una de las barreras que dificulta en muchas ocasiones una formación de calidad y cercana a las necesidades del mercado. Sin embargo, los avances tecnológicos en los últimos tiempos en el áreas de Información y de la Comunicación (TIC), ha posibilitado interactuar con Entornos Virtuales (EVs), que facilitan el acceso a estos recursos a través de la virtualización. Un EV es un conjunto de Recursos Computacionales (RCs) virtualizados, en el que los alumnos tienen la posibilidad de conectarse de forma remota, y están específicamente configurados para la realización de una actividad educativa en la que se emplea una metodología de aprendizaje (como por ejemplo la creación de un portafolio educativo). El uso de EVs lleva a un proceso de enseñanza-aprendizaje más eficiente al tener un entorno listo para su uso, y configurado ad-hoc para aplicar una determinada metodología en el aula, además de proporcionar los recursos software y hardware requeridos para la realización de la actividad, evitando así las complicaciones de instalación y configuración del entorno software y su coexistencia con otros entornos.

Actualmente, las tecnologías Cloud, a través de sus proveedores, proporcionan infraestructuras (e.g. recursos de almacén y cómputo) que dan soporte a los EV. Estos proveedores pueden ser privados (on-premise) o públicos (e.g. Microsoft Azure, Amazon Web Services (AWS)) bajo un modelo de pago por uso. Los proveedores de Cloud públicos evitan que el usuario incurra en el coste de mantenimiento, instalación y configuración de una infraestructura propia, además de la necesidad disponer de

personal técnico experto [3], dado que se aplica un modelo de pago por su uso donde se incluyen dichos costes en la factura.

Teniendo en cuenta la economía actual, utilizar plataformas Cloud que permiten pagar sólo por los recursos utilizados bajo un modelo de pago por uso, puede ayudar a fomentar el uso de EVs en muchas Universidades. El auge de las tecnologías Cloud para proporcionar infraestructura virtualizadas, abre la oportunidad para el acceso de grandes cantidades recursos de cómputo y almacenamiento bajo un modelo de pago por uso. Estas tecnologías se están utilizando a nivel empresarial y científico. Sin embargo, no es tan común su uso en el área educativa (ej. Universidades). Si bien es cierto, que los proveedores Cloud actuales están haciendo un gran esfuerzo en introducir estas tecnologías en las Universidades, ofreciendo sus herramientas de gestión de plataformas Cloud a través de convenios que permiten utilizar sus recursos Cloud de forma gratuita. Este es el caso del programa AWS Educate [44], que permite a docentes y a alumnos tener un acceso sufragado a la plataforma Cloud proporcionada por AWS y que ha proporcionado los recursos de cómputo necesarios para llevar a cabo este Trabajo Final de Máster (TFM).

El problema al que nos enfrentamos es el de configurar los EVs a través de una plataforma que automatice el despliegue y configuración de los mismos mediante un interfaz sencillo y usable. Por ello, las tecnologías Cloud parecen una buena alternativa, dado que estas permiten aprovisionar de una forma rápida recursos computacionales, permitiendo la configuración dinámica descrita en una receta en base a los EVs que se necesiten en un momento dado para el desarrollo de una actividad.

El TFM se centra en la creación de un EV desplegado en la nube que facilite la aplicación y evaluación de un portafolio digital para asignaturas relacionadas con la programación, en el que los alumnos realizan boletines de ejercicios utilizando un determinado lenguaje de programación (e.g. Java o C). Los códigos fuentes son guardados y evaluados automáticamente, de forma que conformen un portafolio. Para ello, se requieren diferentes herramientas software, tales como frameworks de programación y las herramientas que permiten evaluar a través de unos indicadores de calidad los códigos fuentes desarrollados por el alumno.

# 2 OBJETIVOS

Este TFM tiene como objetivo crear una plataforma Cloud para el despliegue de EVs que faciliten la creación y evaluación de un portafolio digital de actividades educativas relacionadas con la programación. Los objetivos que debe cumplir la plataforma son los siguientes:

- Proporcionar una forma sencilla y usable para el despliegue automático de EVs que permitan la gestión y evaluación de un portafolio digital dirigida a actividades de programación, sin necesidad de tener conocimientos técnicos sobre el Cloud.
- Tener la capacidad de desplegar los EVs tanto en un Cloud privado como en un Cloud público.
- Proporcionar al instructor de una herramienta que permita expresar los requerimientos (hardware, software y de configuración) de los EVs que dan soporte a un portafolio digital. Permitiendo personalizar la inclusión en los EVs de diferentes tipos de IDE para programación, compiladores, etc.
- Organizar y guardar el portafolio en repositorios que permitan gestionar las diferentes versiones de los programas que vaya desarrollando el alumno y así poder evaluar su progreso.
- Combinar los repositorios donde se almacenan los códigos fuente con un entorno de integración continua, posibilitando el auto-testeo y extracción de indicadores de calidad de los programas, y el análisis de la calidad del código de las aplicaciones realizadas.

# 3 ESTADO DEL ARTE

En esta sección se va a proceder a detallar aquellas tecnologías y herramientas que se han empleado para el desarrollo de la plataforma. En primer lugar, se describe muy brevemente el concepto de Cloud Computing [4], así como el modelo y los proveedores empleados para las pruebas unitarias de la plataforma creada en este TFM. A continuación, se detalla el componente principal de la plataforma, el *Infrastructure Manager* (IM) [5], describiendo su procedimiento de uso, el lenguaje que utiliza para describir los EVs y la API REST que proporciona. En el siguiente punto,

se detalla el repositorio empleado para el control de versiones de los códigos, las herramientas de integración continua y evaluación de calidad del código empleada para la autoevaluación del portafolio que implementa la plataforma. Finalmente, se describe el escritorio remoto empleado para dar acceso a los alumnos a las MV desde donde tienen que trabajar los alumnos implementando los programas.

## 3.1 Cloud Computing

En términos generales, Cloud Computing es un modelo que proporciona y da acceso a un conjunto de recursos (e.g. redes, recurso de cómputo y almacenamiento), de forma que estos se pueden aprovisionar y liberar rápidamente con un esfuerzo mínimo de gestión. Los proveedores Cloud pueden proporcionar tres modelos de servicio, estos son Software as a Service (SaaS), Platform as a Service (PaaS) e Infrastructure as a Service (IaaS). Las infraestructuras que soportan estos modelos pueden estar gestionadas y dar servicio a nivel privado, es decir, en una determinada organización (on premises) o público en el caso que sean proporcionados por empresas (ej. AWS, Microsoft Azure, etc.) [4], donde se utiliza un modelo de pago por uso. También existen modelos híbridos que combinan recursos de Clouds privados y públicos.

Concretamente, en este TFM se utilizará el modelo IaaS, que proporciona al consumidor, en nuestro caso la plataforma, los recursos de procesamiento, almacenamiento, redes y otros recursos fundamentales para desplegar los EVs necesarios para la gestión y evaluación del portafolio. El consumidor tiene control sobre la infraestructura virtual y está limitado por la infraestructura física.

La plataforma a desarrollar en este TFM debe ser agnóstica y debe de poder desplegar los EVs sobre Clouds privados y públicos.

Los servicios Cloud ofrecidos por los proveedores proporcionan a las organizaciones: capacidad de almacenamiento, potencia de cómputo, herramientas de comunicación y colaboración además de software y redes, que pueden ser combinados de acuerdo a lo que la organización requiera y con una alta disponibilidad de recursos. Desde ciclos de CPU para proyectos HPC (High Performance Computing), muy importante en el desarrollo de simulaciones computacionales de problemas complejos mediante la computación paralela, hasta la capacidad de almacenamiento para respaldos

empresariales; aplicaciones empresariales CRM (Customer Relationship Management), gestión de relaciones con clientes una estrategia orientada a la satisfacción y fidelización del cliente; ERP (Enterprise Resource Planning), destinados a la administración de recursos en una organización; aplicaciones orientados a la integración continua para el desarrollo de software, entre otras.

Grandes corporaciones que operan en Internet como Google, Amazon, Microsoft proporcionan el acceso a sus infraestructuras con un modelo de despliegue de Cloud público y cobrando por los servicios mediante "pago por uso". De tal manera, los usuarios solo pagan por aquellos recursos que utilizan. Este tipo de plataformas proporcionan una gran cantidad de recursos y una funcionalidad base que ofrece muchas posibilidades tanto en el ámbito comercial como en el científico.

### 3.1.1 Amazon Web Services (AWS)

Amazon Web Services [6] (AWS) es el proveedor público empleado en este TFM y es uno de los pioneros y actual líder entre los servicio de Cloud públicos existentes. Inicialmente proporcionaba la funcionalidad basada en el modelo IaaS, mediante los servicios de cómputo EC2 (Elastic Compute Cloud) y de almacenamiento escalable, mediante el servicio S3 (Simple Storage Service). AWS proporciona una plataforma de infraestructura escalable, de confianza y de bajo costo en el Cloud.

En este TFM se utiliza principalmente el servicio Amazon EC2. Es un servicio web que proporciona capacidad de cómputo con tamaño modificable en la nube. Nos brinda la posibilidad de elegir entre varios tipos de instancia, sistemas operativos y paquetes de software. Amazon EC2 permite seleccionar una configuración de memoria, CPU y almacenamiento de la instancia, así como el tamaño de la partición de arranque óptimo para su sistema operativo y su aplicación.

## 3.2 Infrastructure Manager (IM)

Para el desarrollo de la plataforma propuesta en este TFM, se requiere un componente que facilite el despliegue de EV a través de recetas en proveedores tanto privados como públicos. Estas recetas deben indicar requerimientos software y hardware, así como la configuración necesaria. En este sentido el Infrastructure Manager [5] (IM) resulta un componente idóneo para este fin, dado que proporciona funciones para el

despliegue de MV (Máquinas Virtuales) conectadas entre sí por una red de interconexión, sobre diferentes proveedores Cloud. Nos permite modificar la infraestructura lanzada bajo demanda y de forma elástica tras su despliegue. El IM expone una API con un conjunto de métodos para facilitar y monitorizar el despliegue. Las funciones permiten la creación, consulta de estado, añadir y eliminar de MVs desplegadas. Para el despliegue de la infraestructura el IM utiliza el lenguaje de especificación RADL (Resource and Application Description Language) [45].

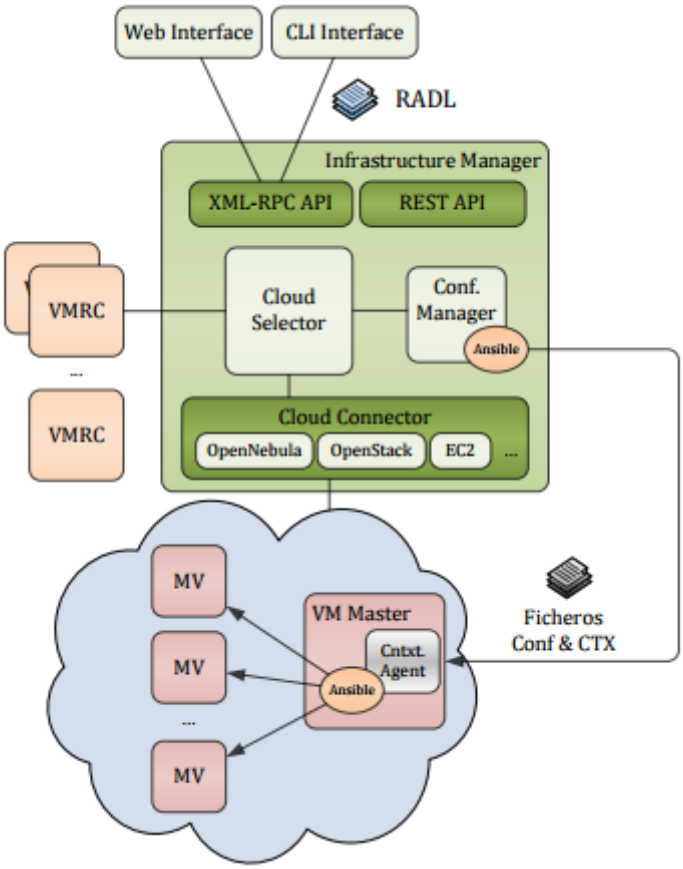



FIGURA 1. ARQUITECTURA DEL GESTOR DE LA INFRAESTRUCTURA

El IM cubre todos los aspectos necesarios para la creación y gestión de las infraestructuras tanto en el Cloud público como privado. Contempla los siguientes aspectos reflejados en la figura 1:

- **Expresar requerimientos:** Expresa los requerimientos, hardware, software y de configuración, sobre los recursos que va a necesitar para la ejecución de un EV.

- **Creación de la infraestructura:** En base a los requerimientos definidos por el usuario, el sistema realiza todos los pasos necesarios que incluyen:
  - **Selección de Imágenes de MV:** Búsqueda y selección de la mejor imagen de MV disponible. Para ello tiene un sistema de catalogación de dichas imágenes a través del sistema VMRC [7] que indexa y almacena Imágenes de Máquinas Virtuales (IMVs), junto con toda la información relevante de las mismas, incluyendo todo el software disponible.
- **Selección de Clouds**: Elección del proveedor Cloud, de todos los disponibles para el usuario. Hay que tener en cuenta que tanto la selección de la imagen como la del Cloud están relacionadas. Por tanto, se deberán elegir de forma combinada.
- **Despliegue en el proveedor Cloud:** Conexión con el proveedor Cloud elegido y lanzamiento efectivo. Esto se debe hacer de forma independiente del sistema Cloud elegido.
- **Contextualización:** Una vez seleccionada una IMV se deberá realizar un proceso denominado contextualización, en caso de ser necesario instalar software adicional requerido por el usuario, así como realizar todos los pasos necesarios para su configuración.

### 3.2.1 Resource and Application Description Language (RADL)

El propósito principal del Resource and Application Description Language (RADL) [45] es especificar los requisitos de los recursos donde se ejecutarán las aplicaciones científicas o aplicaciones de propósito general. Debe incluir todos los detalles de configuración necesarios para conseguir una MV completamente funcional y configurado. Se utiliza un esquema declarativo basado en el lenguaje de contextualización ofrecido por Ansible [46]. También describe las capacidades de red subyacentes necesarias. Básicamente RADL permite:

- Especificar los requerimientos de las aplicaciones a ser instaladas en las máquinas virtuales.
- Especificar los requisitos hardware (número de procesadores, RAM, puertos, red privada y pública, usuario de la máquina virtual). También, requisitos software (paquetes de instalación, bases de datos, librerías, etc).

- RADL sirve como lenguaje de especificación de requisitos de las aplicaciones y recursos de la MV a desplegar. Es un lenguaje común para distintos proveedores facilitando el proceso de despliegue a los usuarios. Los requisitos de las aplicaciones comprenden una serie de características:
  - Característica asociada a la imagen de la máquina virtual y el repositorio de imágenes que cumplen las características de sistema operativo o las aplicaciones especificadas en la receta.
- Características de las máquinas virtuales como ser: cantidad de memoria, número de CPUs, adecuados a las recetas.

En la figura 2 podemos observar un ejemplo de un documento RADL.

```
# Definición de cero o más redes.
# Una instrucción "network" por cada red que a definir
network <id> [(<características>)]

# Definición de los sistemas o MVs.
# Una instrucción system por cada tipo de MV a definir.
system <id> [(<características>)]

# Definición de las recetas de configuración ansible
configure <id> (<recetas ansible>)

# Despliegue de un número de instancias de las MVs
deploy <id> <num_instancias> [<cloud_id>]

# Instrucciones de contextualización
contextualize [max_time] (
    system <system_id> configure <configure_id> [step <num>]
    …
)
```

FIGURA 2. ESQUEMA BÁSICO RADL

### 3.2.2 SERVICIO REST

Las peticiones al IM [8] se realizan mediante el protocolo HTTP o HTTPS. Toda petición que se realice al IM tienen que estar acompañadas de una AUTORIZACIÓN. En la figura 3 vemos un ejemplo de AUTORIZACIÓN para el caso de lanzar en el proveedor Cloud AWS.

FIGURA 3. ESQUEMA AUTORIZACIÓN

```
id => nombre al despliegue ( "mi despliegue" ) ( string )
type => servicio a utilizar ( "EC2" ) ( string )
username => Acces Key ID ( "AKIAJ56DDB3T5D6NY5IA" ) ( string )
password => Secret Access Key ( "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" ) (
string )
```

Esta AUTORIZACIÓN se envía en cada petición que se realice al IM. Si el contenido no se puede analizar correctamente o el usuario y la contraseña no son válidos, se devuelve el código de error HTTP 401.

Los métodos disponibles se muestran en la figura 4:

| HTTP method | /infrastructures | /infrastructures/<infid> | /infrastructures/<infid>/vms/<vmid> |
|---|---|---|---|
| GET | **List** the infrastructure IDs. | **List** the virtual machines in the infrastructure `infId` | **Get** information associated to the virtual machine `vmId` in `infId`. |
| POST | **Create** a new infrastructure based on the RADL posted | **Create** a new virtual machine based on the RADL posted. | |
| PUT | | | **Modify** the virtual machine based on the RADL posted. |
| DELETE | | **Undeploy** all the virtual machines in the infrastructure. | **Undeploy** the virtual machine. |

| HTTP method | /infrastructures/<infid>/stop | /infrastructures/<infid>/start | /infrastructures/<infid>/reconfigure |
|---|---|---|---|
| PUT | **Stop** the infrastructure. | **Start** the infrastructure. | **Reconfigure** the infrastructure. |

| HTTP method | /infrastructures/<infid>/vms/<vmid>/<property_name> | /infrastructures/<infid>/<property_name> |
|---|---|---|
| GET | **Get** the specified property `property_name` associated to the machine `vmId` in `infId`. It has one special property: `contmsg`. | **Get** the specified property `property_name` associated to the infrastructure `infId`. It has three properties: `contmsg`, `radl`, `state`. |

| HTTP method | /infrastructures/<infid>/vms/<vmid>/stop | /infrastructures/<infid>/start |
|---|---|---|
| PUT | **Stop** the machine `vmId` in `infId`. | **Start** the machine `vmId` in `infId`. |

FIGURA 4. MÉTODO REST IM

### 3.2.3 CONTEXTUALIZACIÓN

Como se ha comentado anteriormente, el IM utiliza Ansible [9] para la contextualización de los EVs. El IM emplea esta herramienta para adaptar las IMVs a los requerimientos específicos de las aplicaciones, instalando y configurando todo el software necesario.

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*, por la cual desde una máquina se ejecutan de forma remota a través de comandos SSH sobre una o varias máquinas para configurarlas. Por tanto, necesita que el puerto 22 esté habilitado en las instancias desplegadas. Ansible, no instala ningún agente de configuración en las instancias remotas, por lo que tan solo es necesario poder realizar conexiones SSH en la máquina remota.

## 3.3 Control de Versiones

La plataforma que se plantea en este TFM, requiere un repositorio que gestione los cambios de los códigos fuentes generados por los alumnos. Para ello, las herramientas más apropiadas son las herramientas de control de versiones, las cuales son sistemas que registran los cambios en un archivo o conjunto de archivos con el tiempo, con el objeto de recordar y recuperar versiones específicas en un momento dado [10].

Estos sistemas, posibilitan revertir cambios en los archivos y volver a un estado anterior. También posibilitan comparar los cambios con el tiempo con el objeto de analizar las causas de un determinado problema.

Existen numerosos sistemas de control de versiones, los cuales son por lo general sistemas centralizados en los que se dispone de un único servidor que contiene todos los archivos versionados, y un número de clientes que se conecta a los archivos de ese lugar central. Durante muchos años, este ha sido el estándar para el control de versiones (ver figura 5). Esta configuración tiene ventajas e inconvenientes. La desventaja más obvia, es el punto único de fallo que representa el servidor centralizado. Si ese servidor cae nadie puede conectarse y guardar los cambios versionados en los que está trabajando. En cambio, las versiones también se guardan en los equipos locales y permiten volver a reconstruir el servidor central.

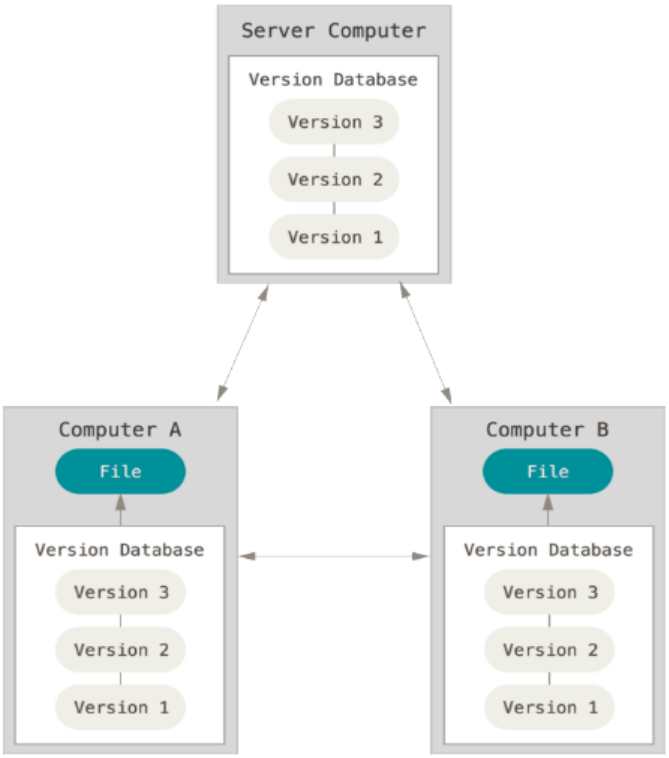



FIGURA 5. ESQUEMA DE UN SISTEMA DE CONTROL DE VERSIONES CENTRALIZADO

A continuación, se describen las características más destacables de algunas herramientas existentes para el control de versiones:

- **Subversión:** La parte principal de Subversión [11] es el almacén central de datos y versiones de un número indeterminado de proyectos. Contiene árboles de ficheros y directorios. Cada árbol es una instantánea de cómo eran los ficheros y directorios versionados en el repositorio en un momento determinado. Recuerda todos los cambios que se han hecho sobre él. Si trabajan varias personas en un proyecto, pueden modificar con sus cambios el repositorio central. Los clientes, pueden conectarse al repositorio y leer o escribir en esos archivos.

- **Perforce:** Esta herramienta [12] es un control de versiones propietario desarrollado por Perforce Software y permite almacenar cualquier archivo sin ningún límite en el tamaño del archivo. Sus características son el control de acceso de grano fino, apoyar DevOps distribuidos escalables, la eficiencia del desarrollo, arquitectura de servidor optimizado, instalaciones de colaboración

de código incorporadas y una solución completa de gestión de Git para la empresa.

Para el desarrollo de la plataforma planteada en este TFM se ha optado por un sistema de control de versiones Git, el cual es un sistema distribuido y que se detalla en la siguiente subsección.

### 3.3.1 GIT

Los objetivos de Git [10] se resumen a continuación:

- ∇ Velocidad.
- ∇ Diseño simple.
- ∇ Un fuerte apoyo para el desarrollo no lineal (ramas paralelas).
- ∇ Totalmente distribuida.

Git es capaz de manejar grandes proyectos, tales como el núcleo de Linux, de manera eficiente. Git permite configurar e inicializar un repositorio, comenzar y detener el seguimiento de archivos, y preparar (*stage*) y confirmar (*commit*) cambios. También permite que se ignoren ciertos archivos y patrones, deshacer errores rápida y fácilmente, navegar por la historia de un proyecto, ver cambios entre confirmaciones, y enviar (*push*) y recibir (*pull*) de repositorios remotos.

La versión que se va a utilizar en el proyecto es **GitLab** [13] GitLab proporciona una herramienta centralizada para alojar repositorios de código. Incluye un interfaz web que posibilitará al docente revisar de forma sencilla, a través de un navegador web, el contenido de los repositorios de código creados por los alumnos.

## 3.4 HERRAMIENTAS DE PRUEBAS UNITARIAS

Uno de los objetivos de la plataforma de este TFM es el auto-testeo del portafolio, de forma que podamos saber de forma automática si los programas desarrollados por los alumnos se han realizado satisfactoriamente o no.

Por ello, necesitamos herramientas que permitan gestionar las pruebas unitarias [14]. Con estas herramientas probamos las unidades del software, normalmente métodos.

Para ello se describen pruebas unitarias que testean los métodos y ver si funcionan correctamente de forma aislada.

Las ventajas de las pruebas unitarias es que tardan menos tiempo en ejecutarse, por lo que se tiende a lanzarlos más a menudo. Además, las pruebas unitarias fuerzan al programador a escribir clases menos acopladas (con menos dependencias entre ellas), lo que mejora el diseño del software.

Si una prueba unitaria falla, es posible determinar que existe un problema en el código. Las pruebas aseguran que el código sigue cumpliendo los requisitos funcionales tras haber realizado modificaciones en el código.

Para automatizar y realizar este tipo de pruebas unitarias en este TFM, se han empleado los JUnit Test Framework en el caso de Java y Cutest para el caso de C.

### 3.4.1 CUTEST

**Cutest** [15] es una biblioteca de pruebas unitarias para el lenguaje C. Sus características son:

- ∇ Posibilita una depuración más rápida en los programas. Las pruebas indican qué subprograma no funciona correctamente.
- ∇ Velocidad en el desarrollo.
- ∇ Fácil de implementar.
- ∇ Altamente portátil. Funciona con todos los principales compiladores en Windows (Microsoft, Borland), Linux, Unix, PalmOS.
- ∇ Fuente abierta. Se puede extender para añadir más funcionalidad.
- ∇ Los métodos disponibles para realizar pruebas unitarias con esta librería son:

```
void CuAssert(CuTest* tc, char* message, int condition);
void CuAssertTrue(CuTest* tc, int condition);
void CuAssertStrEquals(CuTest* tc, char* expected, char* actual);
void CuAssertIntEquals(CuTest* tc, int expected, int actual);
void CuAssertPtrEquals(CuTest* tc, void* expected, void* actual);
void CuAssertPtrNotNull(CuTest* tc, void* pointer);
```

En los repositorios GitLab del instructor se tiene la documentación de la librería como también ejemplos prácticos del uso de la misma.

### 3.4.2 JUnit

**JUnit** [16] define afirmaciones. Una afirmación es una forma de especificar el resultado deseado y compararlo con el resultado real. Si los resultados deseados y los reales son idénticos, la afirmación tiene éxito; de lo contrario, se produce un error. Para ello JUnit brinda métodos como:

```
void assertEquals(boolean expected, boolean actual)
void assertTrue(boolean expected, boolean actual)
void assertFalse(boolean condition)
void assertNotNull(Object object)
void assertNull(Object object)
void assertSame(boolean condition)
void assertNotSame(boolean condition)
void assertArrayEquals(expectedArray, resultArray);
```

Una guía práctica indispensable para explorar el mundo de JUnit a tener en cuenta está disponible en [17]. En la implementación se explicará con ejemplos las pruebas unitarias tanto en Java como C.

## 3.5 Integración Continua

La Integración Continua [18] propone la realización de integraciones automáticas de un proyecto continuamente. En este TFM, entendemos como integración la compilación y ejecución de pruebas unitarias de todo un proyecto (ejercicios prácticos de programación), incluyendo en dicha integración la descarga de repositorios, generación de informes, etc. Todas estas tareas se conforman de forma cíclica para formar este proceso llamado Integración Continua.

Una herramienta de Integración Continua tiene la capacidad de monitorizar un sistema de control de versiones. De forma automática compila y ejecuta las pruebas unitarias de la aplicación pertinente, notificando a los desarrolladores la necesidad de revisar el código si se detecta un fallo, ayudando a ubicar los elementos que intervienen durante este proceso, manteniendo un histórico de artefactos y reportes entre otros.

Este tipo de herramientas permiten realizar test automáticos básicos, en la que el servidor realiza *builds* por cada *commit* realizado, por lo que se tiene acceso a los

resultados al momento. Los *builds* incluyen compilación y ejecución y reporte de pruebas unitarias por lo que los errores pueden ser corregidos al momento.

Existen diversas herramientas de integración continua, como las descritas a continuación:

- **Drone:** Esta herramienta [19] es un sistema alojado. Aunque se puede descargar una imagen de contenedor Docker [47] para generar entornos aislados en los nodos.
- **BuildBot:** Esta herramienta [20] de código abierto automatiza la construcción del software así como el testeo del mismo.
- **Travis:** Esta herramienta [21] es quizás la mejor opción online. Es libre, pero puede ser complejo instalarlo localmente.

La plataforma creada en este TFM debe proporcionar al instructor y a los estudiantes un proceso de integración continua del portafolio que permita la compilación, ejecución y pruebas unitarias para que los alumnos puedan corregir los errores y los profesores evaluar los resultados. Por ello, se ha optado por la herramienta Jenkins que se describe en el siguiente apartado.

### 3.5.1 JENKINS

Jenkins [18] es una herramienta de Integración Continua de código abierto escrita en Java. Surgió inicialmente como bifurcación de un proyecto llamado Hudson. Ofrece soporte para muchos lenguajes de desarrollo y tecnologías, ya sea por soporte propio o mediante plugins.

Jenkins promueve la sencillez de uso con una interfaz web simple pero que ofrece grandes posibilidades abarcando obtención de código de diversos tipos de repositorios, compilación, ejecución de pruebas unitarias, presentación de reportes, automatización de comandos, avisos mediante correos electrónicos, etc. Permite desplegarse en cualquier máquina o servidor.

Para la realización de todas estas tareas, Jenkins permite crear y configurar los llamados 'Jobs', conjuntos de tareas parametrizables que definirán el proceso de integración continua que queremos realizar en un proyecto concreto. Se tratan de

flujos de trabajo configurados para un determinado objetivo basado en uno o varios proyectos concretos.

### 3.5.2 SonarQube

SonarQube [22] en una herramienta que permite evaluar la calidad del software y se puede integrar en la herramienta Jenkins como complemento, con lo que permitirá a la plataforma de este TFM extraer más indicadores de calidad del portafolio que los propios de Jenkins.

A continuación, comentamos algunas de las características principales de la herramienta que pueden ser útiles en la evaluación de la calidad del portafolio planteada en este TFM [23]:

- Análisis de la calidad del código de las aplicaciones.
- Profiling de aplicaciones para detectar cuellos de botella y problemas de memoria (memory leaks).
- Ejecución de pruebas de carga/estrés que estudie la escalabilidad que presenta la infraestructura tanto hardware como software.
- Compatible con Maven 2 y Maven 3, con lo que puede generarse un informe complementario dentro del ciclo de construcción y mediante el uso de un servidor de integración continua.
- Integra las mejores herramientas de medición de la calidad de código: CPD, findbugs, PMD, checkstyle, agregando la información de dichos plugins y ofreciendo un resumen tipo cuadro de mando. Clasifica las incidencias en base a su severidad y a su naturaleza: Fiabilidad, usabilidad, eficiencia, mantenimiento, portabilidad (Matriz de radar del gráfico)
- La herramienta ayuda a disponer de un entorno de mejora continuo.
- Proporciona medidas de complejidad ciclomática [48], en definitiva, si nuestro código es complejo en la implementación (muchos bucles anidados, etc.).
- Proporciona medidas tales como LCOM4 (medida de cohesión de métodos) y RFC (Response for Class) que permiten conocer la cohesión de las clases

implementadas en el software (clases cuyo rol está claramente definido o por el contrario son dispersas y poco cohesionadas) y la capacidad de realizar pruebas unitarias sobre una clase (cantidad de objetos con los que colabora), respectivamente.

- Permite disponer de una matriz de dependencia entre paquetes, donde fácilmente se encuentran referencias cíclicas (falta de desacoplo en el diseño de las aplicaciones).

- Permite descubrir el volumen de comentarios de nuestra aplicación, así como el índice de duplicaciones (falta de refactorización ) de nuestro código.

## 3.6 Escritorio Virtual

En este TFM, a través de la plataforma se van a desplegar en un Cloud, entre otros recursos, un conjunto de MVs desde donde los alumnos desarrollarán los programas. Para ello, los alumnos se conectarán ellas a través de un escritorio remoto.

Si desea acceder a un escritorio de Linux a través de la red, o simplemente desea compartir los recursos de su computadora en la LAN dando cuentas a todos los usuarios en su computadora, puede configurar un servidor de escritorio remoto [24]. Este servicio es compatible con el cliente de escritorio remoto en Windows, gracias al software llamado XRDP que utiliza el protocolo RDP (Remote Desktop Protocol).

Además, también es compatible con otros entornos de escritorio como XFCE, LXDE o KDE.

En este TFM se ha optado por instalar en las MV LXDE, descrito en la siguiente subsección.

### 3.6.1 Lightweight X11 Desktop Environment (LXDE)

Lightweight X11 Desktop Environment [25] (LXDE) es un entorno de escritorio libre para Unix y otras plataformas POSIX, como Linux o BSD.

LXDE no es tan complejo como KDE o GNOME, pero es bastante usable y ligero, y mantiene una baja utilización de recursos y energía. LXDE usa Openbox como gestor de ventanas predeterminado y ofrece un escritorio ligero y rápido basado en

componentes independientes que pueden ser utilizados en otros entornos. Lo mantiene una comunidad internacional de desarrolladores.

LXDE no ha sido diseñado para ser poderoso ni sobrecargado, sino para ser útil y ergonómico. Uno de los objetivos principales de LXDE es el de mantener en un nivel mínimo la utilización de los recursos del sistema.

## 3.7 IDE DE DESARROLLO

En este TFM, se debe ofrecer a los alumnos una herramienta de desarrollo para que puedan realizar los boletines y generando los códigos fuentes que conformarán el portafolio. Para ello se ha optado por el IDE CodeBlocks que se detalla a continuación.

### 3.7.1 CODEBLOCKS

Es un entorno de desarrollo [26] integrado libre y multiplataforma para el desarrollo de programas en lenguaje C, C++ y Java, aunque no tenga de forma nativa una estructura de proyecto para Java. Está basado en la plataforma de interfaces gráficas WxWidgets, por lo que puede usarse libremente en diversos sistemas operativos, y tiene licencia pública general de GNU.

CodeBlocks es un IDE construido como un núcleo altamente expansible mediante complementos (plugins). Actualmente la mayor parte de la funcionalidad viene provista por los complementos incluidos de forma predeterminada. No es un IDE autónomo que acepta complementos, sino que es un núcleo abstracto donde los complementos se convierten en una parte vital del sistema. Esto lo convierte en una plataforma muy dinámica y potente, no solo por la facilidad con que puede incluirse nueva funcionalidad, sino por la capacidad de poder usarla para construir otras herramientas de desarrollo tan solo añadiendo complementos.

Algunos de los compiladores compatibles con este IDE son los siguientes:

- GCC, en sus versiones para Microsoft (ya sea MinGW o Cygwin) y GNU/Linux.
- Borland C++ Compiler
- Intel C++ Compiler

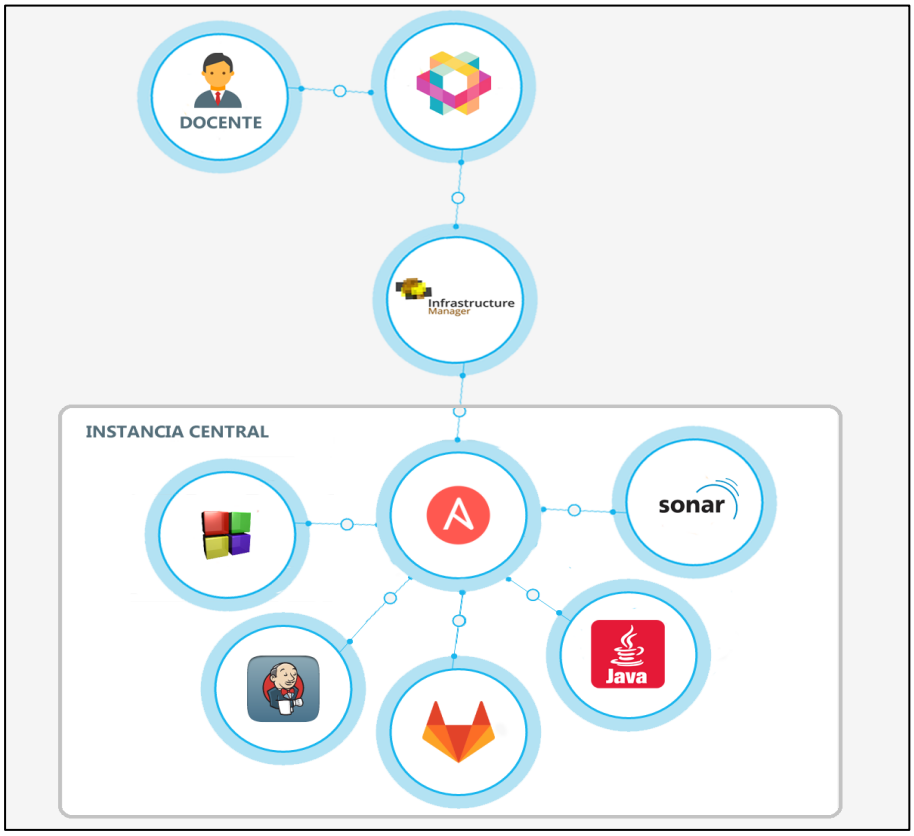
Todos estos compiladores pueden ser detectados automáticamente si están ya instalados al iniciar CodeBlocks.

# 4 Desarrollo de la Plataforma

Para los objetivos que se desean lograr en este proyecto, se llegó a la conclusión de dividir el proceso de lanzamiento de los entornos virtuales en una "instancia central" que aglutine los servicios para el análisis del portafolio y otras "instancias" para el uso de los alumnos. Por tanto, tendremos dos escenarios en el despliegue:

**1.- Instancia central:** Este permite desplegar los distintos servicios en una sola instancia, necesario en el caso de un portafolio de un lenguaje específico de programación, y las cuentas de acceso para los alumnos en esta misma instancia. Esta opción es la adecuada en el caso de que no se quiera incurrir en gastos adicionales.

Figura 6. Instancia Central

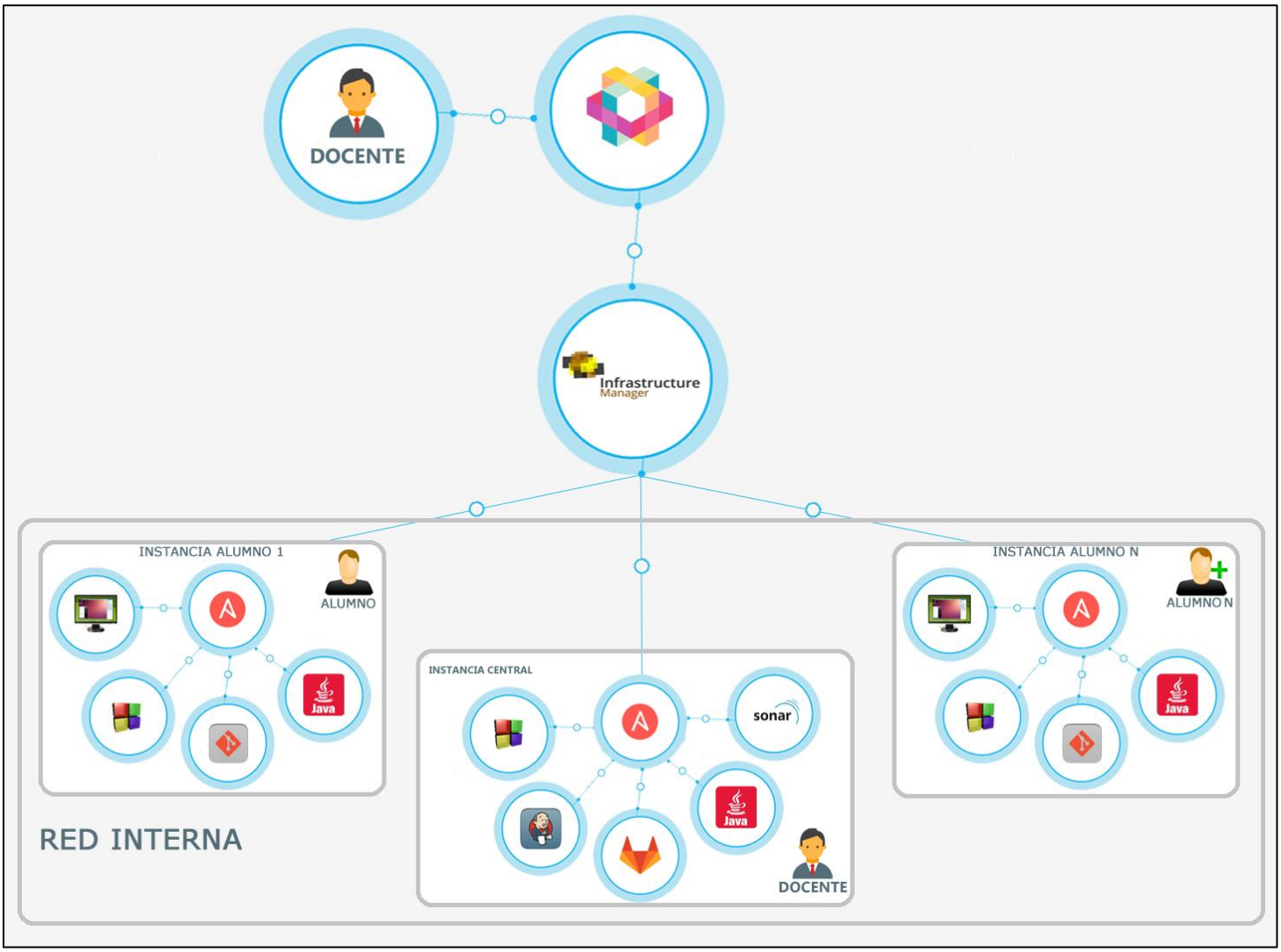



Esta configuración es ideal para que el instructor tenga un servidor central donde sus alumnos puedan guardar los trabajos de su portafolio en los repositorios de GitLab [27] y las herramientas para el análisis del código fuente de cada práctica mediante

Jenkins y, en el caso del lenguaje de programación Java, el análisis de las métricas en SonarQube.

**2.- Instancia central e instancias para alumnos:** Esta opción está habilitada cuando el usuario selecciona que el número de "Instancia por Alumno" en la interfaz web sea mayor o igual a 1, teniendo en cuenta que en la "instancia central" se instalarán los paquetes necesarios para brindar el servicio de portafolio, y en las "instancias de alumnos" solo se instalará el compilador de lenguajes de programación, escritorio virtual, Git y sus respectivos proyecto en cada cuenta del alumno y el IDE de desarrollo CodeBlocks necesario para realizar las prácticas.

FIGURA 7. INSTANCIA CENTRAL E INSTANCIA PARA ALUMNOS



Listado de servicio a desplegar en la instancia central:

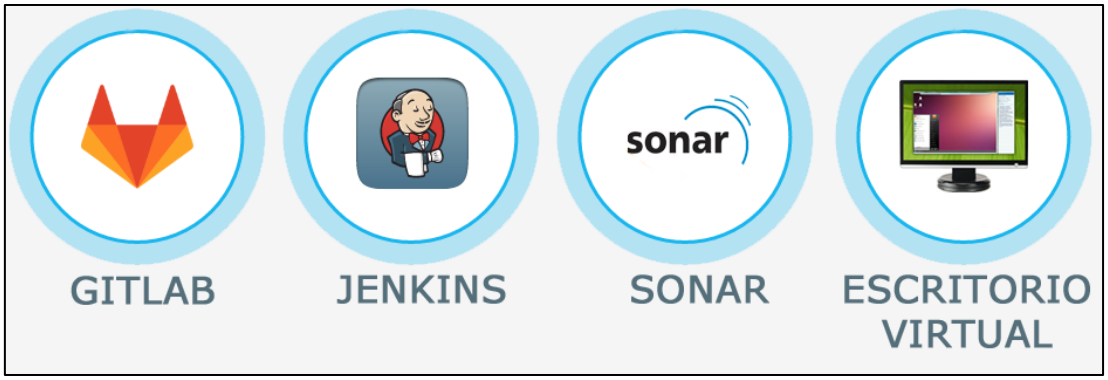



FIGURA 8. SERVICIO A DESPLEGAR

Herramienta para el desarrollo del portafolio:

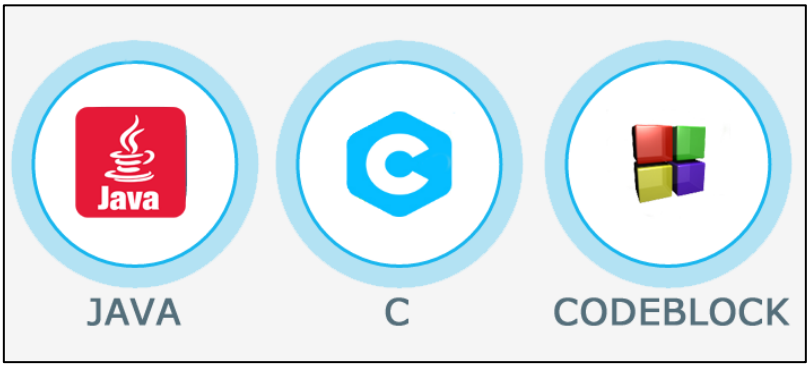



FIGURA 9. HERRAMIENTA DE DESARROLLO Y LENGUAJE DE PROGRAMACIÓN

Como se comentó anteriormente la "instancia central" tiene todos los servicios y la configuración necesaria, y es la opción por defecto habilitada en la interfaz web. Por tanto, el instructor no tiene que definir esta instancia. Los beneficios para el alumno consisten en darle la facilidad a través de GitLab de poder clonar su portafolio de forma externa a un ordenador personal u ordenador del aula pudiendo acceder en cualquier momento a su portafolio, evitando la necesidad de llevar sus trabajos en dispositivos como pendrive, además de permitir tener un control de versiones del código realizado en clases. Los alumnos y el instructor tienen sus cuentas configuradas y con los permisos necesarios para poder acceder a sus espacios de trabajo configurados con el lenguaje de programación elegido.

## 4.1 Definición del modelo de transición

Para el desarrollo de la plataforma dividiremos en diferentes etapas para explicar el proceso de desarrollo de la aplicación de gestión de portafolios en actividades de programación.

### 4.1.1 Etapa 1. Captura de datos.

El proceso de captura permitirá al usuario de manera sencilla a través de una interfaz web, utilizando un navegador web convencional, especificar el número de instancias a desplegar, las cuentas de alumnos de la asignatura y la cantidad de prácticas que pertenecen a su portafolio. Se crearán repositorios especificando el nombre de cada práctica.

Se tiene la posibilidad de escoger el tipo de instancia a desplegar, tanto para la "instancia central" como para la "instancia de alumnos". En este proyecto se ha escogido los servicio Cloud de AWS para desplegar el portafolio virtual. Concretamente, para el despliegue de la "instancia central" se utilizará una instancia *m1.small* con un coste mensual de $ 32.21 en la región de Virginia, y $ 2.20 adicionales en la región de Irlanda [28], con las siguientes características:

| Familia de instancias | Tipo de instancia | Arco de procesador | CPU virtual | Memoria (GiB) | Almacenamiento de instancias (GB) | Disponibilidad de instancia optimizada para EBS | Rendimiento de la red |
|---|---|---|---|---|---|---|---|
| Uso general | m1.small | 32 bits o 64 bits | 1 | 1,7 | 1 x 160 | – | Bajo |

**Figura 10. Instancia mínima recomendada**

No se ha optado por escoger una instancia más pequeña como t2.nano, con 1 CPU virtual y 500 Mb de memoria RAM, o una instancia t2.micro, con 1 CPU virtual y 1 Gb de memoria RAM, puesto que este tipo de instancias no son recomendadas para el despliegue de la aplicación de portafolio virtual debido a que los servicios de Jenkins, SonarQube y GitLab son servicios que necesitan requisitos mínimos de memoria y CPU para su correcto su funcionamiento. De hecho, se han realizado despliegues en instancias más pequeñas que la instancia m1.small, mínima recomendada y se ha tenido problemas con los servicios de GitLab y SonarQube, debido a que, al utilizar

instancias pequeñas, los servicios no se iniciaban y en algunos caso si se lograban iniciar estos servicios, al utilizarlos, llegan a picos de uso del CPU y de memoria RAM que provocan que los servicios se detengan o reinicien.

Las recomendaciones de cada aplicación para su funcionamiento son los siguientes:

**Jenkins:** La cantidad de memoria RAM asignada puede variar desde 200 MB para una pequeña instalación, a más de 70 GB para proyectos de gran envergadura. También tendrá que tener en cuenta la sobrecarga de la CPU que genera Jenkins al tener una gran cantidad de usuarios que vayan a acceder a la interfaz de usuario de la aplicación.

**GitLab:** El espacio necesario en el disco depende en gran medida del tamaño de los repositorios que se desee almacenar en GitLab. Además, también tiene requisitos de CPU y memoria RAM:

**Requisitos de CPU:**

- **1 núcleo** soporta hasta 100 usuarios, pero la aplicación puede resultar un poco más lenta debido a que los *worker* están trabajando en segundo plano que se ejecuta en el mismo núcleo.
- **2 núcleos** es la cantidad recomendada, soportando hasta 500 usuarios.
- **4 núcleos** soporta hasta 2.000 usuarios.
- **8 núcleos** soporta hasta 5.000 usuarios.
- **16 núcleos** soporta hasta 10.000 usuarios.

Con menos **memoria RAM** GitLab dará error de tipo *500 Internal server error* (Error interno del servidor) durante el uso.

- **1 GB** de RAM + 3 GB de swap es el mínimo absoluto para su funcionamiento, pero esta cantidad de memoria está desaconsejada.
- **2 GB** de RAM 2 GB soporta hasta 100 usuarios, pero será muy lento
- **4 GB** de RAM es el tamaño recomendado de memoria para todas las instalaciones y soporta hasta 100 usuarios.
- **8 GB** de RAM soporta hasta 1.000 usuarios.
- **16 GB** de RAM soporta hasta 2.000 usuarios.

**SonarQube:** El servidor SonarQube requiere al menos 2 GB de RAM para ejecutarse de manera eficiente y 1 GB de memoria RAM libre para el sistema operativo, aunque lo recomendado es 2 GB para proyectos de mediano a gran tamaño. La cantidad de espacio en disco que necesita dependerá de la cantidad de código que se analizará con SonarQube de igual manera GitLab.

Para dar un acceso intuitivo y sencillo al usuario se ha creado una interfaz web para definir el número instancias a desplegar, la cantidad de prácticas que tendrá el portafolio y las cuentas de los alumnos que pertenecen a la asignatura de programación.

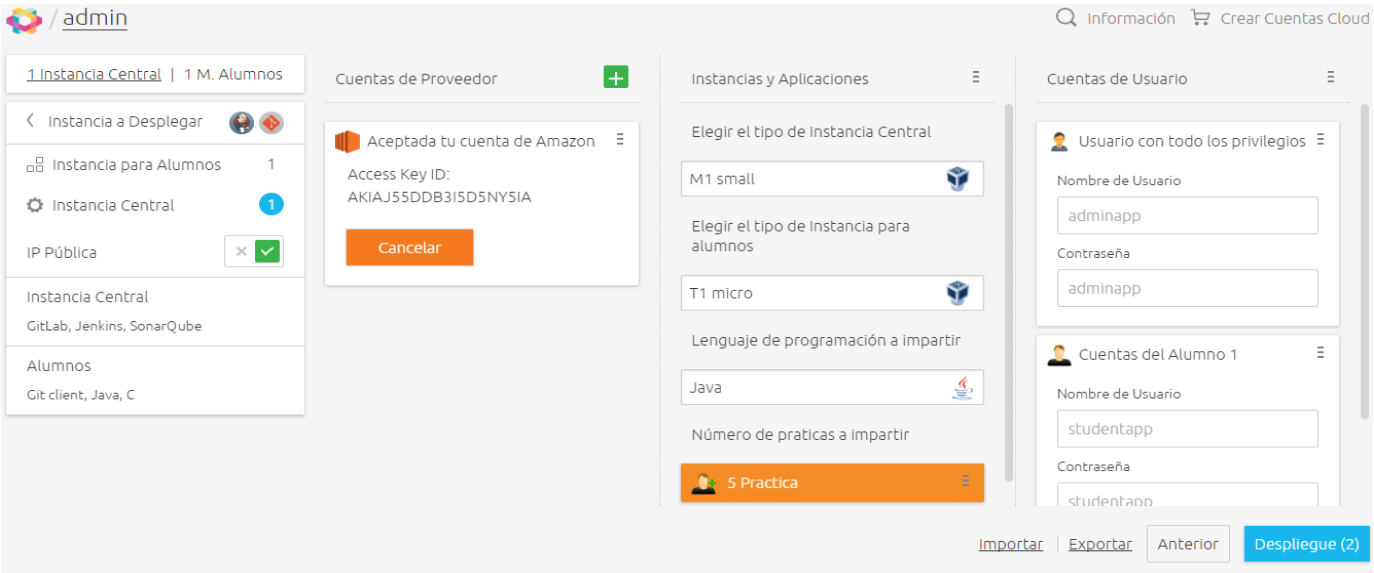



FIGURA 11. CAPTURA DE DATOS

### 4.1.2 ETAPA 2. ADECUACIÓN Y NORMALIZACIÓN.

Es evidente que existen diferentes tipos de configuración que puede escoger el instructor, y con estos datos que introduzca se deberá crear una receta en el formato RADL para lanzar las instancias necesarias requeridas.

Como primer paso de configuración debemos definir los puertos que necesitamos abrir al público. En el caso específico de AWS tendríamos que modificar el grupo de seguridad y definir reglas inbound para nuestra instancia desplegada, definir el puerto y el protocolo que determina el tráfico que puede llegar a nuestra instancia. Necesitaríamos especificar una dirección IP o un intervalo de direcciones IP en la

notación CIDR (por ejemplo, 203.0.113.5/32). También se puede especificar el nombre o el ID de otro grupo de seguridad en la misma región para que tenga acceso.

El IM facilita especificar estos puertos en la receta RADL de la siguiente manera:

```
network publica (
 outbound = 'yes' and outports = '22,80,3389,8089,9000'
)
```

El outbound = yes, proporciona una IP pública (accesible desde cualquier red externa), si el parámetro outbound = 'no', se obtendrá una IP privada. El valor predeterminado es no, AWS permite 2 formas de acceso público a una instancia desplegadas:

- **IP Pública:** Identifica de manera lógica y jerárquica a una interfaz de un dispositivo (habitualmente un ordenador) dentro de una red.
- **DNS AWS:** Los usuarios finales podrán acceder a las aplicaciones en Internet convirtiendo nombres legibles para las personas como www.example.com en direcciones IP numéricas como 192.0.2.1, El nombre de host público de la instancia, que resuelve la dirección IP pública o la dirección IP de Elastic de la instancia y tiene el siguiente esquema:

```
ec2-52-206-222-117.compute-1.amazonaws.com
```

Los puertos a abrir son los que especifican en el parámetro outports:

```
outports = '22,80,3389,8089,9000'
```

Los siguientes puertos se utilizan para acceder a los servicios desplegados:

- **22 SSH**: Permite acceder a máquinas remotas a través de una red, pudiendo manejar por completo la computadora mediante un intérprete de comandos.
- **80 GitLab:** El servicio de GitLab escuchará en este puerto y nos permite tener repositorios privados y públicos, sin costo.
- **3389 Escritorio virtual**: Para permitir que los múltiples usuarios de la red mantengan sus escritorios individuales en un único servidor u ordenador central.

- ∇ **8089 Jenkins**: Herramienta de Integración Continua de aplicaciones con capacidad de monitorizar un sistema de control de versiones para controlar cambios.
- ∇ **9000 SonarQube**: Análisis de la calidad del código de las aplicaciones.

Por defecto, al no colocar el tipo de protocolo después del número de puerto se asigna el TCP. Si queremos especificar el protocolo UDP lo tendremos que definir de la siguiente manera:

```
outports = '80/tcp,22/udp'
```

Para tener comunicación entre las instancias desplegadas definiremos la red privada en el RADL de la siguiente manera:

```
network privada ()
```

Esto nos servirá para comunicarnos con las distintas máquinas desplegadas en la misma red privada.

Para especificar la característica de la instancia a desplegar en RADL definimos la etiqueta "system" seguido del nombre "central" para nuestro caso la instancia central. Este nombre es asignado por nosotros:

```
system central (
 cpu.arch='x86_64' and
 cpu.count>=1 and
 memory.size>=1740m and
 net_interface.0.connection = 'publica' and
 net_interface.1.connection = 'privada' and
 net_interface.0.dns_name = 'central-mv' and
 net_interface.1.dns_name = 'central-priv' and
 disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
 disk.0.os.name ='linux' and
 disk.0.os.credentials.username ='ubuntu' and
 disk.0.os.version ='14.04' and
 disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)
```

**cpu.arch:** En el primer campo definimos el tipo de arquitectura a utilizar en nuestro caso. Utilizaremos la arquitectura 'x86_64'.

**cpu.count>=1** El número de CPUs virtuales que tendrá nuestra máquina virtual.

**memory.size>=1740m** El tamaño de memoria RAM, podemos definirla de 2 maneras en Megabytes "m" o en Gigabytes "g".

```
net_interface.0.connection = 'publica' and
net_interface.1.connection = 'privada' and
net_interface.0.dns_name = 'central-mv' and
net_interface.1.dns_name = 'central-priv' and
```

Habilita las 2 interfaces de red: una pública y otra privada. Posteriormente definimos el nombre DNS para las 2 interfaces de red.

El acceso de las "instancias de los alumnos" a la "instancia central" a la hora del despliegue será a través del DNS de la "instancia central", usando la interfaz privada "central-priv". El IM configura toda la red interna privada y con esta utilidad no necesitamos saber la IP pública de la "instancia central" lo cual nos facilita al momento de realizar la receta RADL.

Especificamos la imagen que utilizaremos en el despliegue. La imagen que utilizaremos es una imagen del sistema operativo Linux con la distribución Ubuntu:

```
disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
disk.0.os.name ='linux' and
disk.0.os.credentials.username ='ubuntu' and
disk.0.os.version ='14.04' and
```

Posteriormente, se configura la instancia con una cuenta de usuario. Para este proyecto usaremos "ubuntu" como usuario. Utilizaremos los directorios de este usuario para descargar el software necesario para el portafolio virtual.

Para el caso que se desea desplegar instancias de alumno la configuración del RADL será:

```
system mv (
 cpu.arch='x86_64' and
 cpu.count>=1 and
 memory.size>=512m and
 net_interface.0.connection = 'publica' and
 net_interface.1.connection = 'privada' and
 net_interface.0.dns_name = 'student-mv-#N#' and
 net_interface.1.dns_name = 'student-priv-#N#' and
 disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
 disk.0.os.name = 'linux' and
 disk.0.os.credentials.username = 'ubuntu' and
 disk.0.os.version = '14.04' and
 disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
```

```
)
```

La configuración es similar al de la instancia central. La diferencia proviene en el nombre del DNS agregamos -#N# para que el nombre que se le asigne a cada instancia sea distinto, el IM asigna un número correlativo del 0 - N para las instancias desplegadas.

```
deploy central 1
deploy mv 1
contextualize (
 system central configure central step 1
 system mv configure mv step 2
)
```

La instrucción deploy "central" 1, indica que necesitamos 1 instancia de tipo central a desplegar. En el caso de las instancias de los alumnos lo realizamos de la misma manera deploy "mv" N, donde N es el número de instancias de alumnos que deseamos.

Para que primero se ejecute la configuración de la instancia central utilizaremos "step" asignando el "step 1" a la instancia central y el "step 2" para las instancias de los alumnos.

### 4.1.3 Etapa 3. Generación de receta.

En esta etapa definiremos las cuentas de alumnos, la cuenta del instructor, los repositorios de código donde se encuentra los scripts necesarios para la configuración de Jenkins, SonarQube y GitLab.

Para especificar las cuentas de usuarios se definirán listas de variables. Para ello se tiene que tener en cuenta que los tipos de elementos que se van a iterar sobre la lista indicada en 'with_items' es una lista de hashes. Se puede hacer referencia a estas subclaves al definir la instrucción Ansible que explicaremos más adelante. Tendremos dos listas, una para las cuentas de alumnos y otra para la cuenta del instructor, y tendrán la siguiente estructura:

```
accounts:
 - { name: "studentapp", pw: "stT7aa1qRryZQ", pass: "studentapp"}
 - { name: "pedrolorenzo", pw: "peiRiuC.dTd82", pass: "pedrolorenzo"}
admin:
 - { name: "adminapp", pw: " adJjgm2H1J1oc", pass: "adminapp"}
```

La clave "name" indica el nombre del usuario.

La clave "pw" es la contraseña encriptado necesario para crear las cuentas de usuario.

La clave "pass" es el password del usuario sin encriptar. Para encriptar el password usamos python la función crypt.

```
python -c 'import crypt; print crypt.crypt(password)'
```

Para llamar a estas listas y a sus valores usaremos el with_items para indicarle sobre qué lista iterar y {{ item.name }} o {{ item.pw }} para identificar el valor que debe obtener.

```
 - name: Create user accounts student
 user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
 with_items: accounts
```

Esto permite crear las cuentas de los nuevos usuarios. De la misma manera lo realizaremos para el instructor. Los permisos de las cuentas creadas lo asignamos con la siguiente instrucción Ansible:

```
 - name: Modify $HOME permissions student
 file: path=/home/{{ item.name }} state=directory owner={{ item.name }} group={{
item.name }} mode=0700
 with_items: accounts
```

El acceso a través de SSH a las nuevas cuentas creadas lo realizaremos mediante la instrucción:

```
 - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
 - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
 - service: name=ssh state=restarted
```

Agregar permisos de administrador al instructor se realizará agregando la cuenta del instructor a lista de "sudoer"

```
- name: Add sudo admin
shell: 'adduser {{ item.name }} sudo'
with_items: admin
```

Las URL (*Uniform Resource Locator [29]*) de los repositorios serán definidas en constantes que nos permite declarar Ansible, para que al momento de clonar los repositorios no sea necesario colocar la contraseña. Se especificará en la URL luego del protocolo http:// o https:// se colocará el nombre del usuario seguido de ":" el password y "@" seguido de la URL del repositorio a clonar. De esta manera no será necesario autentificarse al momento de clonar los repositorios.

```
ipmaster: localhost
baseurl: https://master-class:master-password@bitbucket.org/phantro/base-java.git
jenkinsnodeurl: https://master-class:master-password@bitbucket.org/phantro/node-
jenkins.git
jenkinsserver : https://master-class:master-password@bitbucket.org/phantro/jenkins-
server.git
jenkins: /home/ubuntu/jenkins-server
jenkinsnode: /home/ubuntu/node-jenkins
userproyect : proyect-user.sh
```

La clonación de un repositorio en Ansible lo realizaremos así:

```
- name: Cloning repos + submodules + jenkins 0f452458482
git: repo={{jenkinsserver}}
dest={{ item.dest }}
accept_hostkey=yes
force=yes
recursive=no
with_items:
-
dest: "{{ jenkins }}"
repo: PrimaryRepo
```

Usaremos las constantes {{ jenkinsserver }} que nos indica que repositorio vamos a clonar y {{ jenkins }} el directorio de destino.

Para la configuración de los repositorios en GitLab y la creación de *Jobs* en Jenkins necesitamos tener en la instancia central una lista de los alumnos y la lista de los repositorios a crear. Utilizaremos Node.js [30] para gestionar la creación de cuentas de usuarios en GitLab con sus respectivos repositorios y los Jobs en Jenkins. Para ello utilizaremos JSON (JavaScript Object Notation), un formato para el intercambio de datos. Utilizaremos tres archivos JSON:

**user.json:** Las cuentas de los alumnos

```
{"user1":{"nombre":"student76","password":"studentApp"},

"user2":{"nombre":"student77","password":"studentApp"}}
```

**repository.json:** Los repositorios a crearse

```
{"practice1":{"name":"numero-natural","practice":"https://master-class:master-
password@bitbucket.org/phantro/base-java.git","central":"https://master-
class:master-password@bitbucket.org/phantro/base-java.git"},

"practice2":{"name":"fracciones","practice":"https://master-class:master-
password@bitbucket.org/phantro/base-java.git","central":"https://master-
class:master-password@bitbucket.org/phantro/base-java.git"}}
```

La clave "name" es el nombre que se le asignará al repositorio a crea, "practice" es el repositorio que se clonará para cada alumno y la clave "central" es el repositorio donde se encuentra ubicada las pruebas unitarias del repositorio, estos son repositorios externos que al momento de la configuración previa se modificarán para subirlos a GitLab.

**admin.json:**

```
{"name":"adminapp", "password":"adminapp"}
```

Estos archivos JSON se escriben en la receta

```
- copy: dest={{ jenkinsnode }}/user.json
content='{"user1":{"nombre":"studentapp","password":"studentapp"},"user2":{"nombre"
:"franzapp","password":"franzapp"}}'
```

De esta manera copiamos las cuentas de los alumnos al fichero "user.json". De igual manera lo realizamos para los archivos "admin.json" y "repository.json".

Para la instalación de los servicios necesarios para el portafolio virtual usaremos Ansible y se detallará la instalación de Git y Java, necesarios para este proyecto, para el caso puntual de Java la actualización de los repositorios de paquetes. La receta RADL con la configuración completa de los demás servicios se encuentra en la sección de adjuntos en este documento.

```
- name: Install Git 0f485338872
apt: name=git update_cache=yes state=latest

- name: Add repo for java 8 0f168424895
apt_repository: repo='ppa:openjdk-r/ppa' state=present

- name: Install java 8
apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
sudo: yes
```

Se instala la última versión Git y Java. Si ya estaba instalado, entonces se actualiza a la última versión disponible en los repositorios de paquetes de la distribución Linux utilizada (en este caso Ubuntu 14.04 LTS).

### 4.1.4 Etapa 4. Despliegue:

El proceso de despliegue se realiza a partir de la receta RADL que hemos generado y explicado en las anteriores etapas. El servicio proporcionado por el IM permite desplegar la infraestructura en el Cloud que utilizaremos en este proyecto. Para interactuar con este servicio lo realizamos usando REST (Representational State Transfer), un estilo de arquitectura software para sistemas hipermedia distribuidos [31]. También se utilizó en un principio el servicio XML-RPC, un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP para el despliegue [32]. En la actualidad, la aplicación utiliza el API REST del IM para desplegar la infraestructura.

**La comunicación con el Infrastructure Manager mediante REST**

Para crear y configurar una infraestructura los requisitos tienen que estar especificados en el documento RADL. Las recetas, las cuentas de los alumnos y la cuenta del instructor se guardarán en una base de datos MySQL.

En todas las peticiones al IM adjuntamos la AUTORIZACIÓN en la cabecera de la petición explicada en el punto [3.2.2]. La AUTORIZACIÓN se guardará en el servidor PHP [33] en unas variables de sesión. Al momento del despliegue esta variable estará activa durante 12 horas. Pasado este tiempo se eliminará del servidor y de la base de datos. La aplicación realizada en PHP realizará la petición POST con los datos que el instructor ha introducido en la interfaz web. Si la petición que realizamos tiene éxito el IM nos enviará un identificador URI (que es una cadena corta de caracteres que identifica inequívocamente un recurso servicio, página, etc. [34]), de la nueva

infraestructura. Este identificador lo utilizaremos para luego consultar el estado del despliegue y la contextualización.

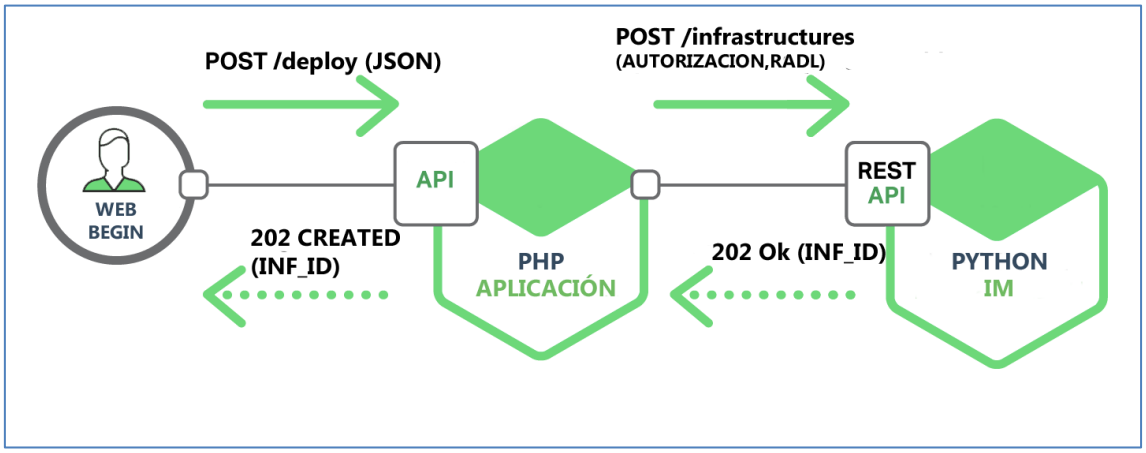



FIGURA 12. DESPLIEGUE DE INFRAESTRUCTURA

El identificador que nos devuelve el IM es como el que se muestra a continuación:

```
{ "INF_ID" : "72e418ca-b546-11e6-a466-300000000002" }
```

Este identificador lo enviamos a la interfaz web para que el usuario pueda seguir el proceso de despliegue y le informe al momento de terminar el despliegue.

La URL destinada a realizar el seguimiento del despliegue "/resume/java/INF_ID". En el cuerpo de esta página tendremos un método asíncrono en JavaScript [35] que cada 60 s. usando la librería Jquery [36], realizará una petición GET a la aplicación en PHP. Este método tiene la URL "/message/INF_ID".

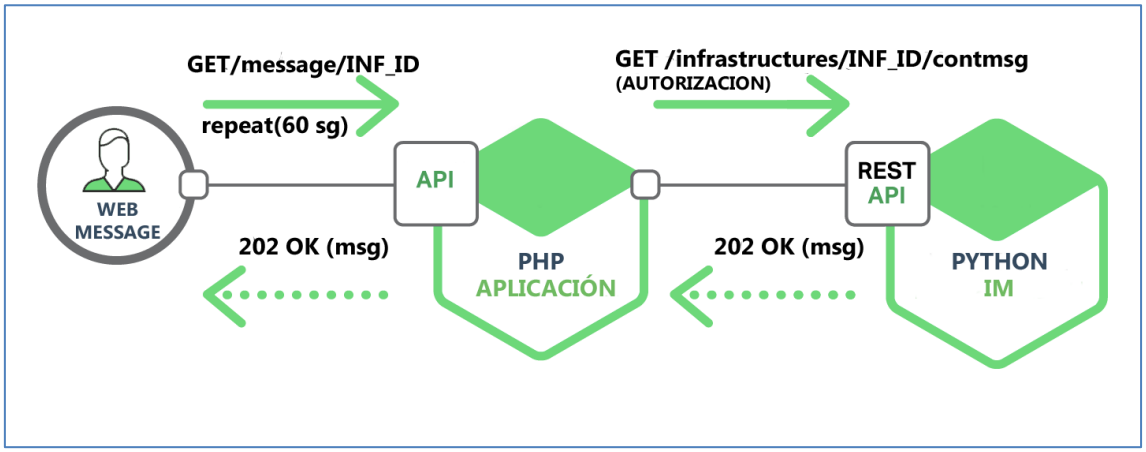



FIGURA 13. CONSULTA ESTADO DEL DESPLIEGUE

Devuelve el estado actual del despliegue. Para mostrar en la interfaz web a través de efectos visuales creados con CSS [37] y mostrado con modificaciones a las clases CSS usando JavaScript, al momento de realizar la receta colocamos identificadores únicos que tendrán los siguientes códigos:

```
["0f485338872"=>"ansible", "0f168424895"=>"git","0f452458482"=>"java",
"0f412483458"=>"jenkins"]
```

Si se encuentra uno de estos códigos en la respuesta mostrará al usuario que servicio se está configurando. Esta técnica la utilizamos también para saber si el despliegue ha terminado. Al encontrar el código "0f412483458" sabremos que ha terminado de configurar todas las instancias.

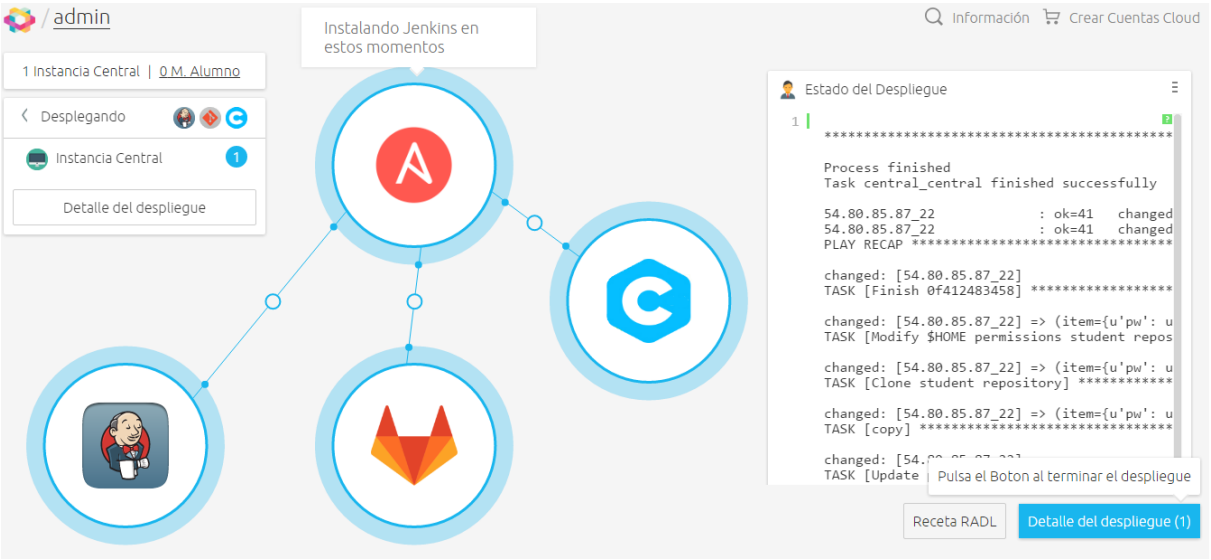



FIGURA 14. ESTADO DEL DESPLIEGUE

Las cuentas de los alumnos y de los servicios los guardaremos en nuestra base de datos. Los datos que necesitamos obtener del IM son:

- IP pública
- IP privada
- Llaves privada de acceso .pem de cada instancia

Toda esta información la obtenemos realizando una petición

***GET*** */infrastructures/INF_ID/vms/id*

Donde "INF_ID" es el identificador único del despliegue y el "id" es el identificador de las instancias desplegadas, por defecto el "id=0" corresponde a la "instancia central" y del "id=1..N" corresponde a las instancias de los alumnos.

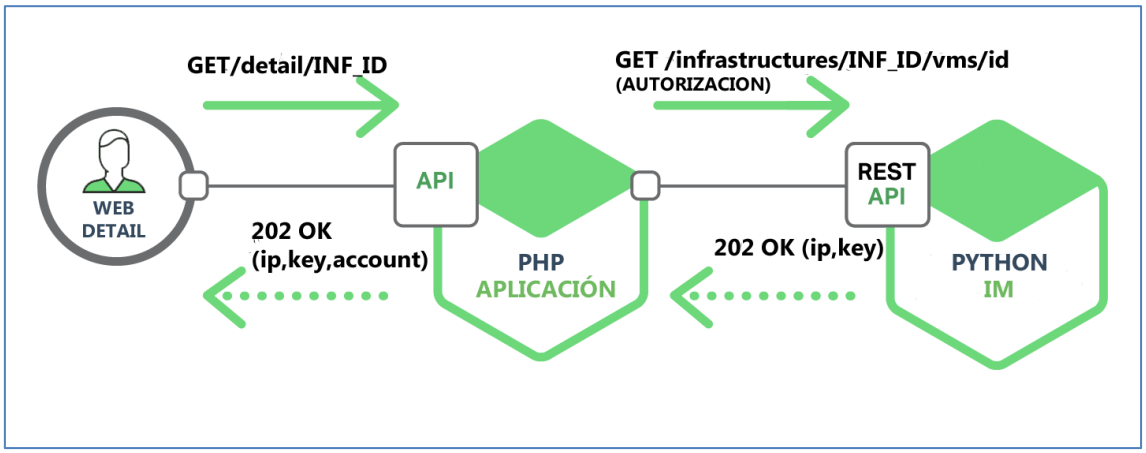



FIGURA 15. INFORMACIÓN DE CUENTAS

### 4.1.5 ETAPA 5. CONFIGURACIÓN DE INSTANCIAS:

Con la receta RADL lo que hemos hechos es instalar todos los paquetes software necesarios, pero estas herramientas no están configuradas, tan solo las tenemos instaladas en cada instancia desplegada, lo que se realizará antes terminar el despliegue es la configuración de estos servicios. Una vez verificado que todos los servicios están activos configuramos GitLab y Jenkins, para que al momento de que el instructor ingrese a los distintos servicios estén previamente configurados y listo para funcionar.

**Configuraciones de GitLab**

Para utilizar cualquier servicio de GitLab como el de crear nuevas cuentas de usuario repositorios nuevos, y configuración en el entorno GitLab es necesario tener un token de autenticación y para ellos debemos de tener primeramente un usuario y una contraseña para obtener este token. Este usuario tiene que tener todos los privilegios. Al instalar GitLab el usuario con todos los privilegios es "root", la contraseña del usuario root en versiones anteriores a la versión "V.7" tenía un valor por defecto que era "5iveL!fe" en la versión "V.8" y posteriores modificaron esta característica por lo cual, a partir de esta versión, la contraseña se introducía en el primer inicio de la aplicación en la interfaz web, pudiendo introducir la contraseña del usuario root. Por lo

tanto, necesitamos que el usuario tenga que introducir la contraseña para luego ejecutar nuestro scripts para crear nuevos usuario y repositorios. Esta opción ponía un freno a la hora de realizar una configuración totalmente automatizada. Lo que se realizó para resolver este problema es utilizar la consola de gitlab-rails para cambiar la contraseña. Por seguridad a esta consola se puede acceder desde la máquina donde se instaló GitLab y con los privilegios sudo, para hacerlo de forma automática definimos un archivo con los comandos a utilizar en la consola ejemplo:

```
u = User.where(id:1).first
u.password = "12345678"
u.password_confirmation = "12345678"
u.save!
exit
```

Con la primera instrucción *User.where(id:1).first* obtenemos la cuenta de root y agregamos el password y confirmación. Luego, guardamos y cerramos la consola.

Una vez hemos configurado correctamente la cuenta root de GitLab ya podemos obtener un token para crear nuevos usuario y repositorios.

Como segundo paso, creamos las cuentas de los alumnos en GitLab con el usuario y la contraseña que colocaron en el momento de definir las cuentas de alumnos.

Para crear estas cuentas usaremos CURL para el envío de peticiones http o https y la API "v3" de GitLab se lo realizará enviando peticiones a la URL del servicio desplegado. Este script lo generamos desde Node.js.

```
curl --header "PRIVATE-TOKEN: '+token.private_token+'" -X POST '+'"'+PROTOCOLO +
'://' + IP + ':' + PORT + '/' + API + '/' + USERS + '?email=' + userStudent +
'@upv.es&password=' + passwordStudent + '&username=' + userStudent + '&name=' +
userStudent + '"
```

Como parámetro necesitamos darle la IP donde se encuentra el servicio. Si lo realizamos en la instancia central solo necesitamos darle localhost como IP. Por defecto GitLab utiliza 2 puertos: el puerto 8080 y el puerto 80 la aplicación de GitLab se ejecuta en el puerto 8080 y utiliza Ngnix (servidor web/proxy inverso) para redirigir el puerto 80 al puerto 8080.

Definimos el recurso a utilizar, "/users", este recurso tiene como requisitos mínimos para crear un usuario los parámetros:

```
username: studentapp
password: passwordstudent
name: student
email: studentapp@upv.es
```

Se tiene más parámetros para configurar un usuario, pero estos son los necesarios para crear la cuenta.

Una vez creadas todas las cuentas de los usuarios creamos los repositorios para cada alumno. De acuerdo al número de prácticas del portafolio lo creamos de la siguiente manera:

```
'curl --header "PRIVATE-TOKEN: '+token.private_token+'" -X POST '+'"'+PROTOCOLO +
'://' + IP + ':' + PORT + '/' + API + '/' + PROYECT + '/' + nameUser+ '?name=' +
nameProyect + '"';
```

Como requisito para crear un repositorio nuevo debemos de especificar el “id” del usuario y el nombre del proyecto que deseamos crear. Por defecto, los repositorios se crean como privados. Para crearlo de tipo público lo tendríamos que definir en la petición o posteriormente modificando las propiedades [38].

Ya creadas las cuentas de usuario y todos los repositorios de cada alumno, lo siguiente que debemos hacer en el caso de Java, es configurar cada proyecto para que sea único. Cuando Jenkins tenga de realizar las pruebas unitarias y tenga que registrar en SonarQube, los proyectos analizados tiene que identificarlos como un proyecto único para cada alumno y para cada práctica del portafolio. Para ello editaremos el proyecto de tal forma que cada proyecto sea único para cada práctica del alumno y así poder analizarlo de forma independiente.

Clonamos los repositorios definidos al comienzo usando una librería "shelljs" [39] de Node.js que permite ejecutar código shell desde Node.js. Una vez clonado, configuramos el proyecto antes de subirlo a nuestros repositorios de GitLab:

```
SUBSTRING="' + nameStudent + '"
FOLDER=/"' + FOLDER + '"
PROYECT="' + nameProyect + '"
RUTA=$PWD$FOLDER
'sed s/"{{ item.name }}"/$PROYECT-$SUBSTRING/g $RUTA/pom.j2 > $RUTA/pom.xml'
'sed s/"{{ item.name }}"/$PROYECT-$SUBSTRING/g $RUTA/JunitTesting/pom.j2 >
$RUTA/JunitTesting/pom.xml'
```

Modificamos los documentos XML para que tengan el nombre del alumno y la práctica, y una vez tenemos modificado los documentos XML realizamos las siguientes instrucciones:

```
'cd $RUTA && git init' +
'cd $RUTA && git config --global user.name $SUBSTRING' +
'cd $RUTA && git config --global user.email $SUBSTRING\@upv.es' +
'cd $RUTA && git remote add origin ' + urlRepo +
'cd $RUTA && git add . ' +
'cd $RUTA && git commit -m "Config pom.xml"' +
'cd $RUTA && git push -u origin master';
```

Iniciamos un repositorio local y agregamos la cuenta de usuario para subir los cambios al repositorio de GitLab. No es necesario que el *name* sea el mismo que el que se creó al iniciar el proyecto ni el *email*. Estos datos los utiliza Git para identificar qué usuarios suben los cambios a los repositorios. Esto nos da la posibilidad de que varios usuarios puedan subir cambios a un mismo repositorio, es decir, la posibilidad de que el alumno pueda subir desde diferentes máquinas y poder identificar qué dispositivo ha subido estos cambios.

Terminadas estas configuraciones podemos en las instancias de cada alumno descargar los repositorios directamente desde GitLab ya configurados y listos para utilizar.

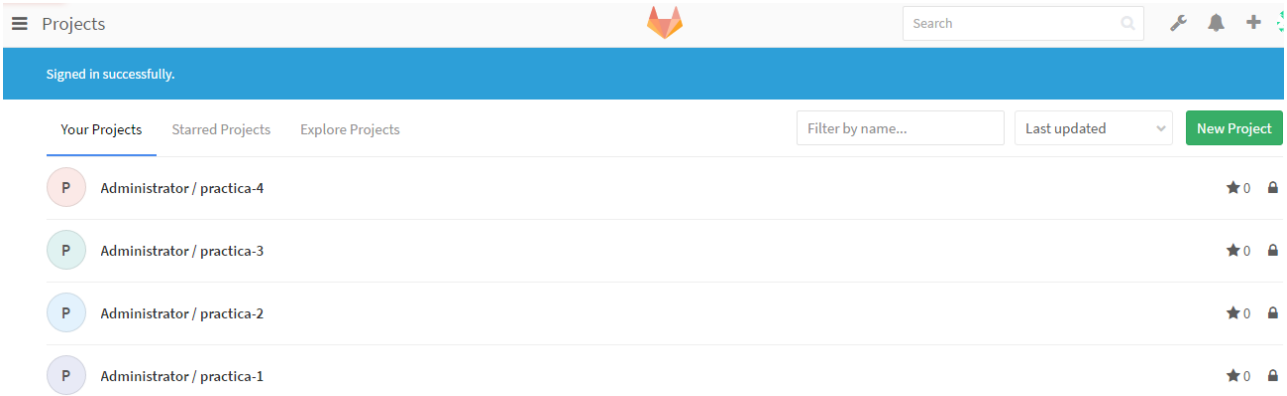



FIGURA 16. GITLAB

**Configuración Jenkins**

Una vez desplegados los servicios como últimos pasos para la configuración de la "instancia central" es la configuración de las herramientas para realizar las pruebas unitarias. Instalaremos los siguientes plugins en Jenkins [40] necesarios para un correcto funcionamiento.

Los plugins a instalar en ambas opciones de despliegue en el caso de Java y de C serán los siguientes:

```
git@current
greenballs@current
github@current
```

en el caso de Java necesitamos instalar el plugin de SonarQube en Jenkins

```
sonar@current
```

Estas importaciones lo realizamos a través de llamadas Http a Jenkins usando CURL. Las instrucciones se ejecutarán sobre localhost dado que la instrucción se ejecutará en la "instancia central".

```
curl -X POST http://localhost:8089/pluginManager/installNecessaryPlugins -d
'<install plugin="git@current" />'

curl -X POST http://localhost:8089/pluginManager/installNecessaryPlugins -d
'<install plugin="greenballs@current" />'

curl -X POST http://localhost:8089/pluginManager/installNecessaryPlugins -d
'<install plugin="github@current" />'
```

En el caso de haber elegido como lenguaje de programación Java:

```
curl -X POST http://localhost:8089/pluginManager/installNecessaryPlugins -d
'<install plugin="sonar@current" />'
```

En el caso de Java es necesario que configuremos Maven en Jenkins y para ello utilizaremos un script Groovy (un lenguaje de programación orientado a objetos implementado sobre la plataforma Java), que permite configurar Jenkins a través de código Java.

Hay dos maneras de importar la configuración de Maven Jenkins.

La primera opción es dejar que Jenkins descargue y configure Maven. Debemos definir qué versión deseamos instalar. El siguiente código será:

```
import jenkins.model.*

println "Adding an auto installer for Maven 3.0.4"

def mavenPluginExtension =
Jenkins.instance.getExtensionList(hudson.tasks.Maven.DescriptorImpl.class)[0]

def asList = (mavenPluginExtension.installations as List)
asList.add(new hudson.tasks.Maven.MavenInstallation('maven-3', null, [new
hudson.tools.InstallSourceProperty([new
hudson.tasks.Maven.MavenInstaller("3.0.4")])]))

mavenPluginExtension.installations = asList

mavenPluginExtension.save()

println "OK - Maven auto-installer (from Apache) added for 3.3.9"
```

En el caso de que tengamos configurado de forma manual Maven lo que debemos de realizar en Jenkins es registrar dónde tenemos configurado Maven y que pueda utilizar esta configuración. Lo realizaremos con este script Groovy:

```
import jenkins.model.*

println "Adding an config for Maven 3.0.4"

a=Jenkins.instance.getExtensionList(hudson.tasks.Maven.DescriptorImpl.class)[0];
b=(a.installations as List);
b.add(new hudson.tasks.Maven.MavenInstallation("MAVEN3", "/usr/local/apache-maven-
3.0.4", []));
a.installations=b
a.save()

println "OK - Maven auto-installer (from Apache) added for 3.0.4"
```

Ambos scripts los podemos lanzar usando CURL a la ruta http://localhost:8089/script el cual en la petición tenemos que adjuntar el script Groovy. Estas instrucciones la lanzaremos en Bash (un intérprete de órdenes basado en la shell de Unix).

```
somesscript=$(cat ./node-jenkins/somescript-maven.groovy)
curl -d "script=$somesscript" http://localhost:8089/script
```

En el caso de Java para la instancia *m1.small* solo utilizaremos 1 thread para la ejecución de Jobs en Jenkins. El motivo de utilizar esta opción y no la que viene por defecto en Jenkins, que es de 2 threads, es que al momento de realizar algunas pruebas con proyectos reales se ha identificado que si 2 procesos están al mismo

tiempo registrando los proyectos en SonarQube tiene problemas de conexión usando la base de datos Mysql e hibernate, que es una herramienta de Mapeo objeto-relacional (ORM), para la plataforma Java. Debido a las características de las instancias tiene lo mínimo recomendado para ejecutar SonarQube. Lo que sucede es que solo 1 Job se registra con éxito, las pruebas unitarias realizadas eran correctas y el otro Job produce un error al registrar este proyecto en SonarQube.

Lo resolvemos dando a Jenkins 1 solo ejecutor. Cambiamos estos con un script Groovy:

```
import jenkins.model.*
def instance = Jenkins.getInstance()
instance.setNumExecutors(1)
instance.save()
```

```
somesthread=$(cat ./node-jenkins/somethread.groovy)
curl -d "script=$somesthread" http://localhost:8089/script
```

Para la configuración de los Jobs en Jenkins, tenemos un archivo java.xml. Este es un backup de un proyecto que se creó de forma manual y exportado desde Jenkins usando una librería llamada "Jenkins" [41] en Node.js. Obtenemos estos backup y lo modificaremos cambiando el nombre del proyecto y los repositorios del backup y llamamos a una función que permite crear Jobs en Jenkins a partir de archivos .xml y el nombre de cada Job que tendrá. Por último, falta colocar el usuario administrador para que los usuarios tengan todos los privilegios. De igual manera usaremos Groovy para definirlo.

```
import jenkins.model.*
import hudson.security.*

def instance = Jenkins.getInstance()

def hudsonRealm = new HudsonPrivateSecurityRealm(false)
hudsonRealm.createAccount("adminapp","adminapp")
instance.setSecurityRealm(hudsonRealm)
def strategy = new GlobalMatrixAuthorizationStrategy()
strategy.add(Jenkins.ADMINISTER, "adminapp")
instance.setAuthorizationStrategy(strategy)
instance.save()
```

Creamos un nuevo usuario que será adminapp y su password adminapp lo realizamos con la instrucción

```
hudsonRealm.createAccount("adminapp","adminapp")
```

y asignamos todos los privilegios a este nuevo usuario como administrador

```
strategy.add(Jenkins.ADMINISTER, "adminapp")
```

```
someuser=$(cat ./node-jenkins/someuser.groovy)
curl -d "script=$someuser" http://localhost:8089/script
```

Una vez enviamos estas instrucciones a Jenkins habremos terminado la configuración y posterior a esto restauramos Jenkins.

```
curl -X POST -u adminapp:adminapp http://localhost:8089/safeRestart
```

Con el flag -u damos en la petición el usuario y la contraseña y enviamos la petición de "safeRestart" a Jenkins lo que permite que una vez se termine los Jobs que están en ejecución reiniciar el servicio, a diferencia de utilizar la opción "restart" que reinicia Jenkins inmediatamente.

En efecto, para el correcto funcionamiento de los plugins es necesario que Jenkins se reinicie.

Las posteriores instrucciones que enviamos a Jenkins tanto para instalar un plugin como para configurar Jenkins usando Groovy requieren enviarle el usuario y contraseña y Jenkins verificará los permisos que tenga este usuario y decidirá si realiza la configuración enviada para su ejecución.

### 4.1.6 Etapa 6. Ingreso y uso:

Terminado el despliegue, el usuario de nuestra aplicación ("instructor") puede acceder a un descripción de toda la configuración realizada:

- Cuentas de usuarios creadas.
- Cuentas de los repositorios en GitLab.
- Las claves .pem de cada instancia desplegada.
- La cuenta con privilegios de sudo de cada instancia.
- Los servicios desplegados los usuarios y contraseñas de cada servicio.
- La IP pública e IP privada de las instancias desplegadas.

- ∇ Acceso a las máquinas a través de escritorio remoto.
- ∇ Imprimir Informe de las cuentas de usuarios y servicios.

Descargar la receta en formato RADL para poder modificarla si lo desea y desplegarla en el IM en la web oficial o usarlo con Ansible "on premise".


FIGURA 17. DETALLE DE CUENTAS

### 4.1.7 ETAPA 7. PRUEBAS Y EXAMINACIÓN:

**Proyecto de código en lenguaje C**

El proyecto basado en código C tiene 2 repositorios

- ∇ Repositorio de cada alumno.
- ∇ Repositorio de pruebas unitarias que tiene acceso el instructor.

**Repositorio alumno:** Tendrá el fichero Student.c el cual tiene el método "StrToUpper" que es para convertir palabras de minúsculas a mayúsculas. Para todas las prácticas estará este método como ejemplo.

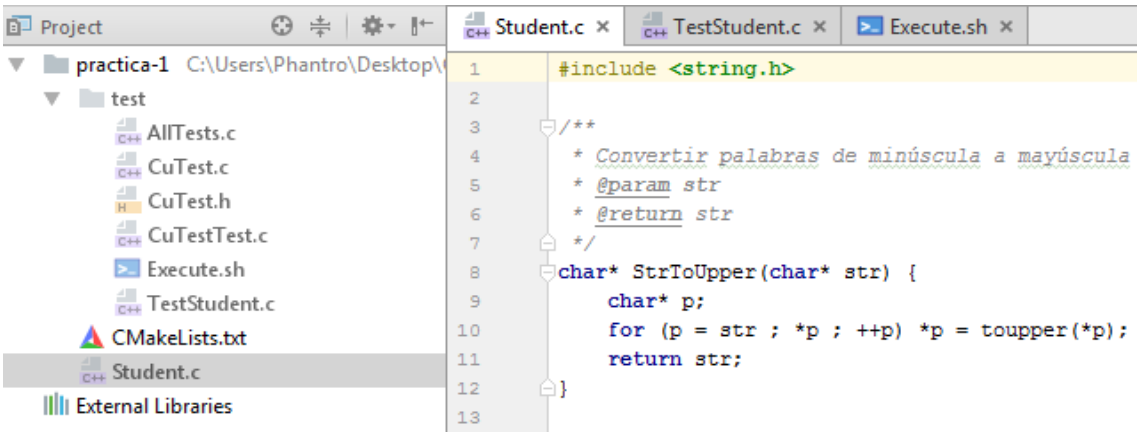



FIGURA 18. MÉTODO DE EJEMPLO C

**Repositorio de pruebas unitarias:** Este repositorio contiene las pruebas unitarias. El fichero "TestStudent.c" cuenta con la prueba unitario del método "StrToUpper" creado por defecto en cada repositorio de alumno.

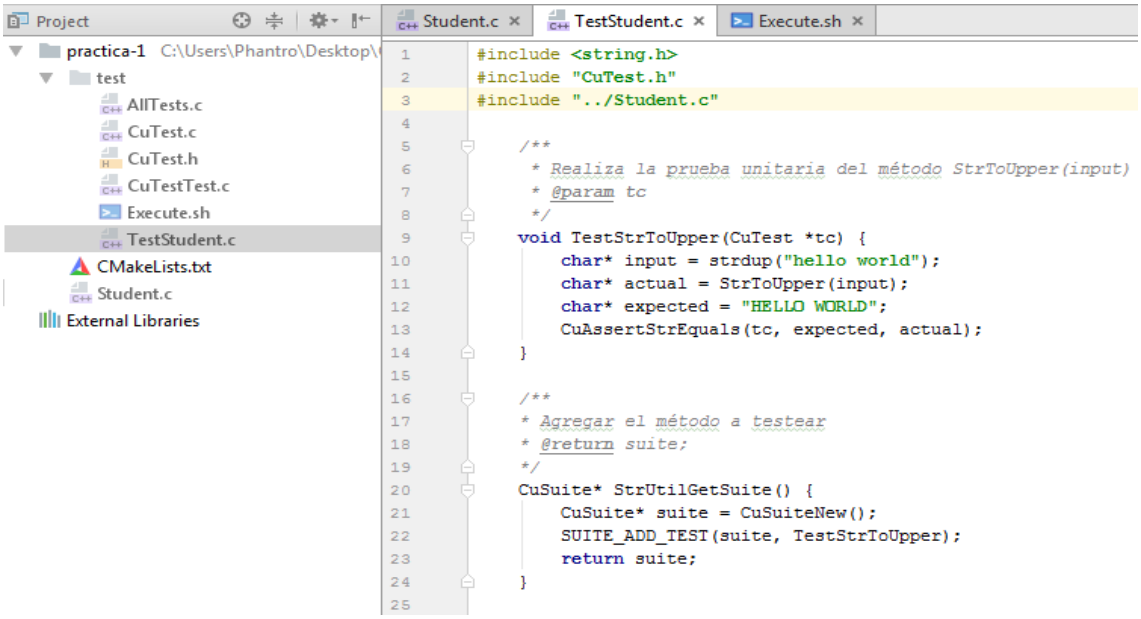



FIGURA 19. PRUEBAS UNITARIAS C

Para compilar y realizar las pruebas unitarias tendremos que ejecutar script "sh Execute.sh" el cual ejecutaremos desde Jenkins o desde la terminal.

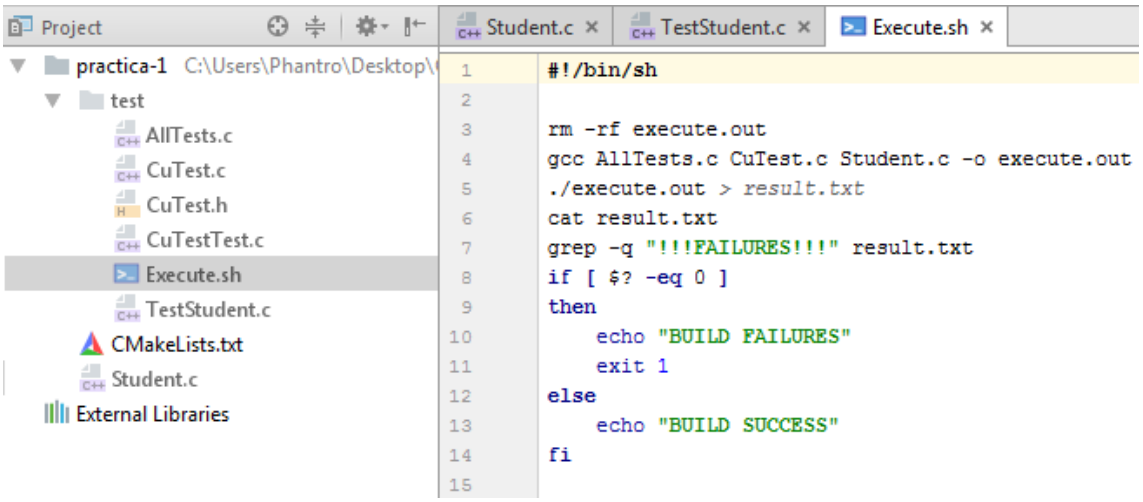



FIGURA 20. EJECUTAR PRUEBAS UNITARIAS C

### Proyecto en lenguaje Java

El proyecto Java tiene 2 repositorios que juntos forman parte de un proyecto Maven:

- ∇ Repositorio de cada alumno.
- ∇ Repositorio de prueba unitarias al que tiene acceso el instructor.

**Repositorio de cada alumno:** El código fuente Java tendrá un método "greet(String)" que concatena al input introducido "Hello " + cadena. Es un método de ejemplo para el alumno.

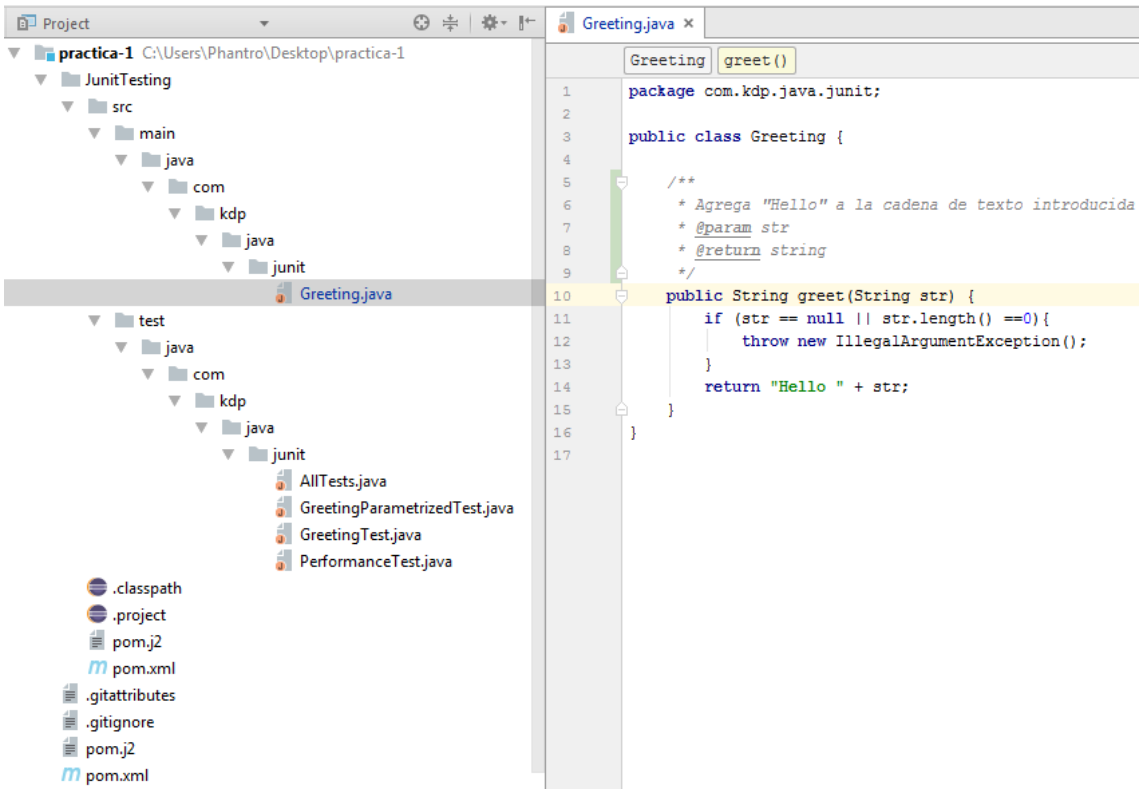



FIGURA 21. CLASS EJEMPLO DE JAVA

**Repositorio de pruebas unitarias:** El repositorio de pruebas unitarias tendrán los ficheros "GreetingParametrizedTest.java" que contiene las pruebas parametrizadas que nos brinda la opción de añadir múltiples valores para analizar el método.

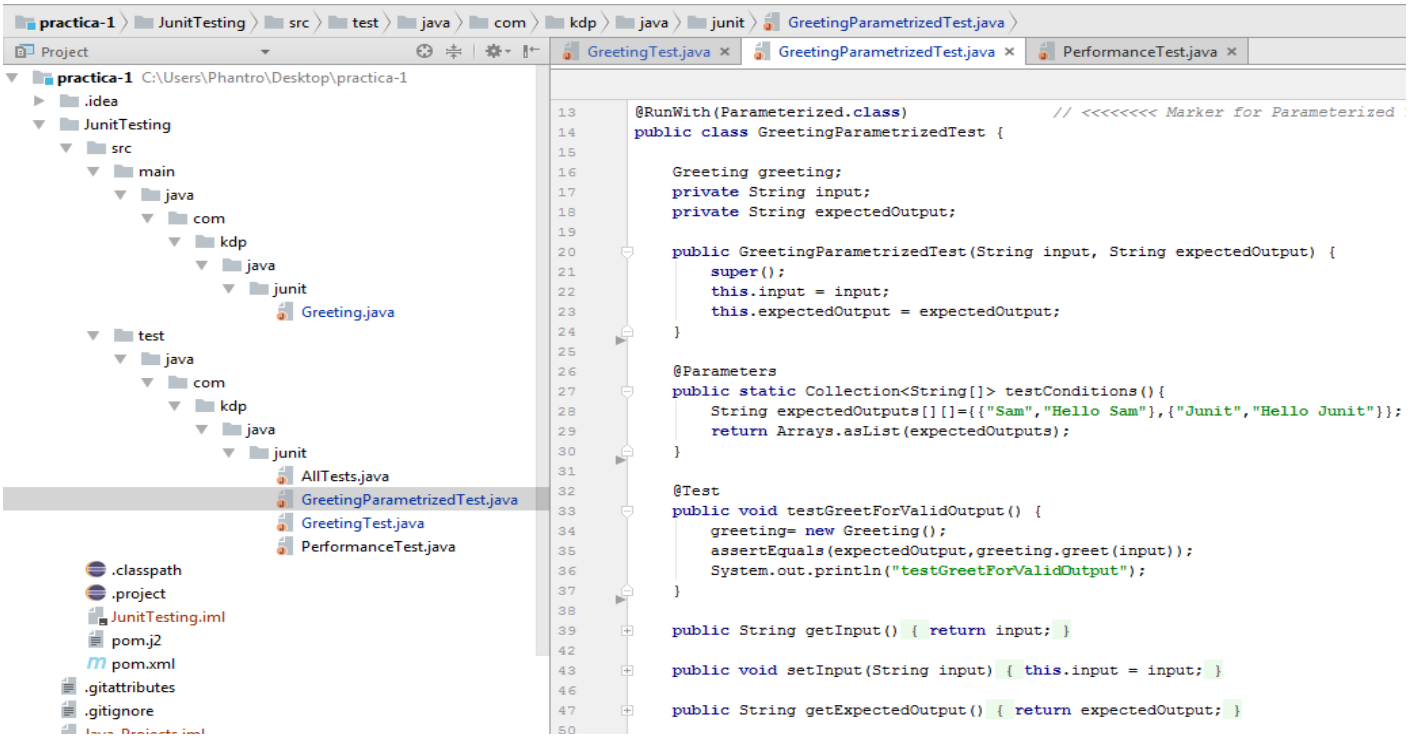



FIGURA 22. PRUEBAS PARAMETRIZADAS JAVA

Este tipo de pruebas unitarias son las más utilizadas en el desarrollo y persiguen instanciar una clase, agregar los input y comparar el resultado a través del método

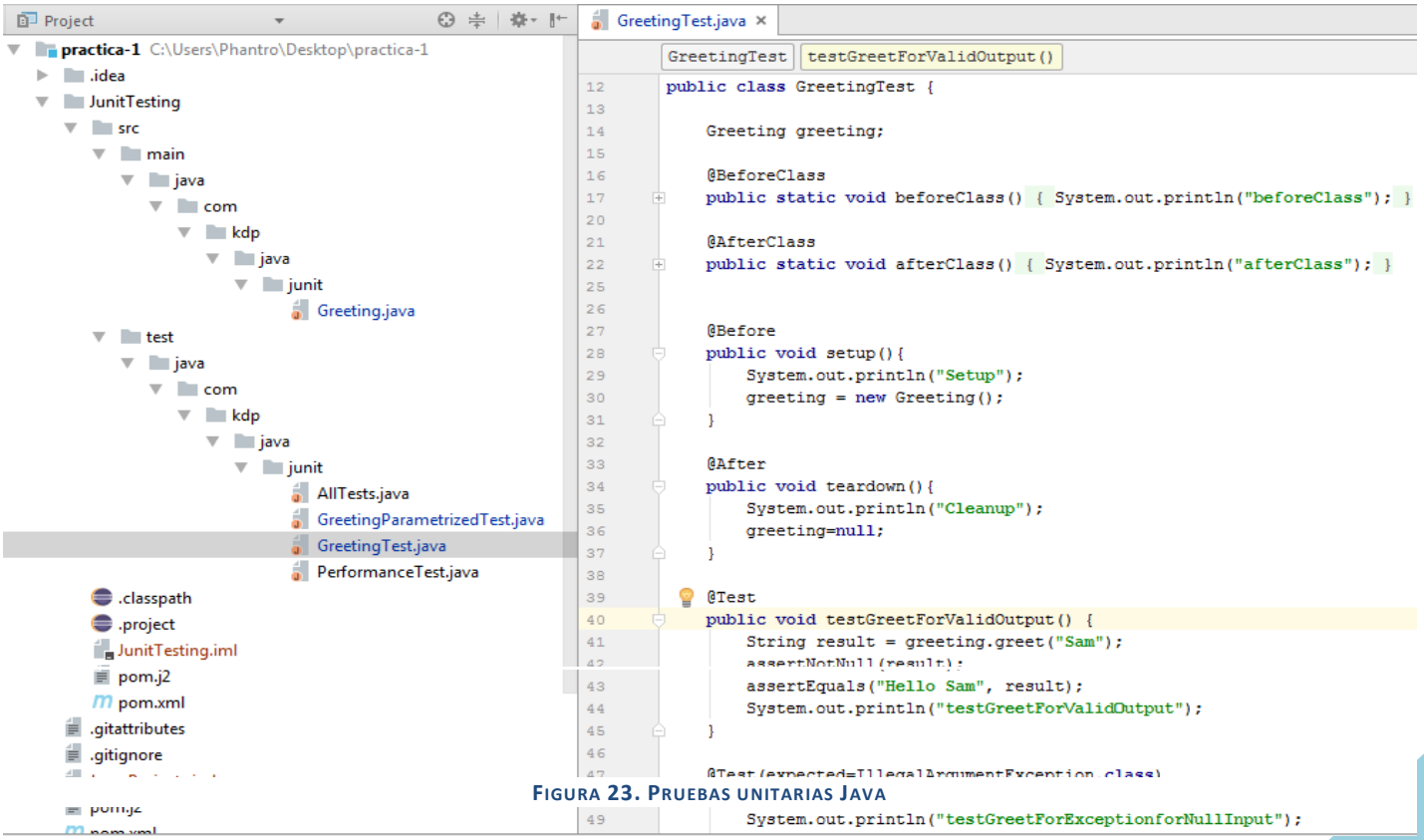



FIGURA 23. PRUEBAS UNITARIAS JAVA

assertEquals para comparar si son iguales [42].

En el caso del despliegue de aplicaciones Java seguiremos las siguientes instrucciones:

Ingresamos a Jenkins con la cuenta de usuario administrador que nos proporciona el detalle del despliegue. Tenemos que configurar la instalación de Sonar y para ello nos vamos a la sección de Administrar Jenkins -> y luego seleccionamos Configurar Sistema -> nos desplazamos hasta la sección SonarQube servers y lo único que tenemos que configurar es habilitar el campo "Enable injection of Sonar" y colocar dar click en "add SonarQube", aparecen campos editables donde tendremos que colocar nombre a nuestra conexión "name => sonar" y la IP de la instancia central ejemplo "Url servidor => 107.22.67.78:9000" el 9000 es el puerto que utiliza SonarQube. Con ello lo tendremos completo y configurado nuestro Jenkins para ejecutar los Jobs de las prácticas de los alumnos. Como recomendación se tiene que ejecutar todo los Jobs creados para verificar que los repositorios y el registro en SonarQube sea el correcto.


FIGURA 24. RESUMEN JOBS JENKINS

Una vez se ejecuta un Job de prácticas en la sección de consola podemos apreciar los detalles de las pruebas unitarias ejecutadas y posterior registro del proyecto en SonarQube el cual nos dirá cuál es la URL para acceder al resumen de las prácticas.

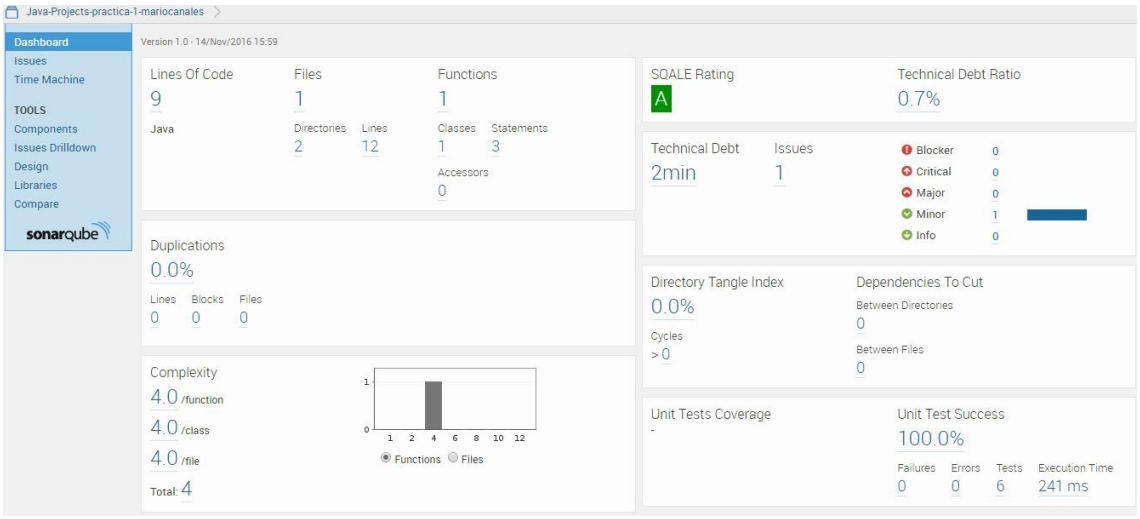



**FIGURA 25. DASHBOARD SONARQUBE**

En la primera columna en la parte izquierda del *dashboard*, nos encontramos con métricas relacionadas con las *"métricas de tamaño"* como líneas de código, número de ficheros, etc. También las que nos muestran el porcentaje de código duplicado que tenemos en el proyecto o las relacionadas con la complejidad de clase y funciones.

Una primera aproximación de cómo interpretar estas métricas sería la siguiente:

- **Métricas de tamaño**: Se refiere a la magnitud del mismo.
- **Porcentaje de código duplicado**: En cuanto menos código duplicado tengamos, en términos de calidad es mejor. Seguramente tengamos un mejor diseño.
- **Complejidad**: Es intuitivo que una complejidad menor es mejor que una más alta.

Se suele recomendar unos valores por debajo un 1% o 2% de código duplicado, complejidad de método por debajo de 5 y de clase por debajo de 10. Estos son valores que deben revisarse en función de la cultura y contexto de la proyecto.

Vamos por la parte derecha del *dashboard*, "Deuda Técnica" (*Technical Debt*) y *"SQALE rating"*. Sirve para saber el estado de salud de los proyectos [43].

# 5 TIEMPOS DEL DESPLIEGUE

La aplicación tiene una base de datos que registra todos los despliegues que se realizan tenemos un histórico de los despliegues exitoso y los despliegues fallidos. Estos últimos debido a mantenimiento de los repositorios de código como ejemplo puede

ser los repositorios de Java 8, por temas de mantenimiento lo cual no permite que podamos descargar paquetes de software necesarios.

Los tiempos lo registramos al momento de generar las recetas y se guardan en base de datos en el campos "created_at" de la tabla "radl". Con esto ya tenemos el tiempo inicial. Los tiempos finales del despliegue se obtienen a partir de la técnica que usamos anteriormente para saber en qué estado del despliegue se encuentra nuestra infraestructura. Cada 60 segundos realizamos una petición al IM para que nos informe del estado de nuestro despliegue. Por lo tanto, los tiempos que se mostraran en esta sección son orientativos. Con esta información verificamos si dentro del texto obtenido se encuentra esta cadena "0f412483458" la cual a nosotros nos indica que la última instrucción de Ansible se ha realizado. Si lo encontramos guardamos en la base de datos en el campo "finish_date" y marcamos el campo "finish" a 1 indicando que se ha terminado este despliegue.

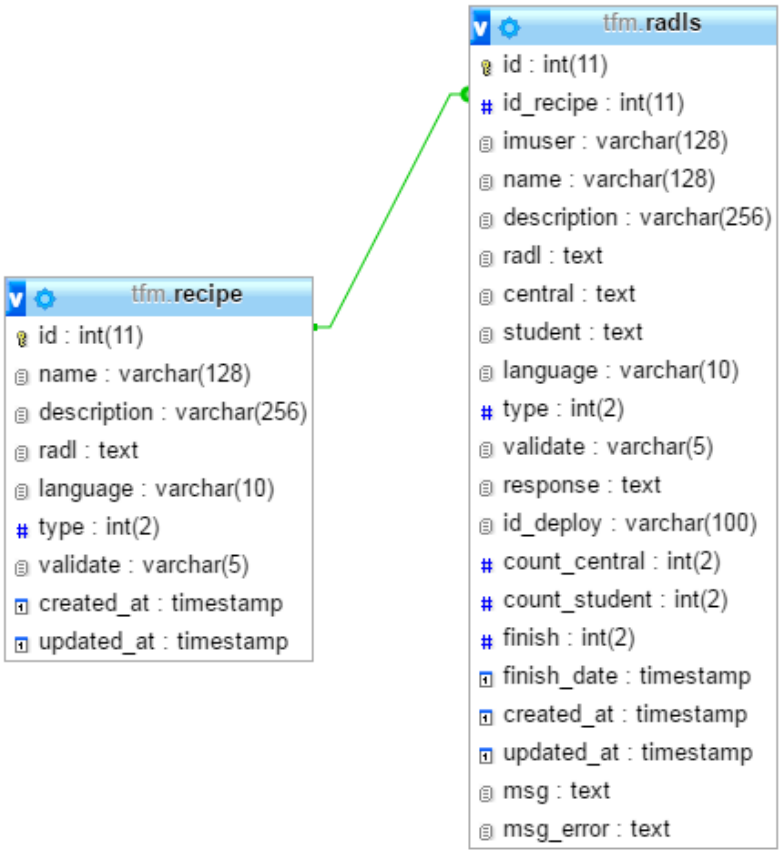



FIGURA 26. ESQUEMA DE BASE DATOS

Esto se realiza para tener métricas de la tasa de éxito en la generación y despliegue de nuestras recetas RADL y para realizar cambios oportunos en caso de servicios externos como repositorios y librerías no estén disponibles, que son necesarias para la instalación de nuestra instancia a desplegar.

**Escenario:** Se ha desplegado la "instancia central" con el soporte para los 2 lenguajes de programación con las siguientes especificaciones

- Instancia central m1.small
- 10 cuentas de alumnos
- 5 repositorios para cada alumno correspondiente a sus prácticas. En total tendremos 50 repositorios. A medida que se aumente el número de alumnos y de repositorio los tiempos varían debido a los tiempos en crear nuevas cuentas, repositorios y posterior subida del código de ejemplo a estos repositorios creados.
- 5 Repositorio de Test unitarios para el instructor.

Tiempos obtenidos en el despliegue de la instancia central sin instancias de alumnos:

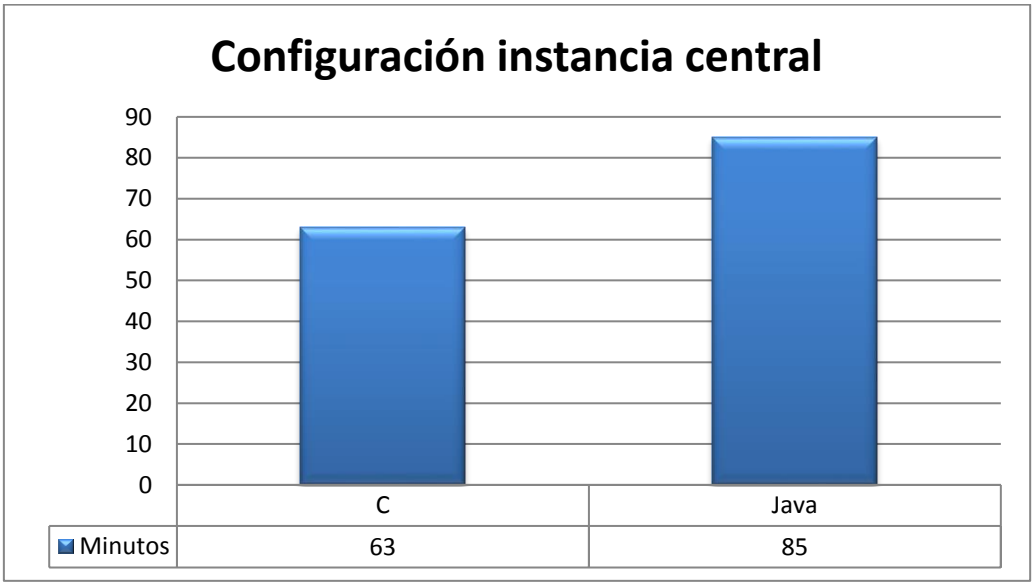



FIGURA 27. TIEMPO DE DESPLIEGUE INSTANCIA CENTRAL (EN MINUTOS)

Estos tiempos son elevados debido a que utilizamos una imagen de máquina virtual sin ningún servicio requerido instalado, y tenemos que instalar todos los servicios. Esto nos penaliza en los tiempos de despliegue pero hace que nuestra receta generada

funcione en otros servicios Cloud compatibles con el IM, y que los cambios que se tengan que realizar a esta receta sean cambiar:

```
disk.0.image.url = 'aws://us-east-1/ami-c93a83de'
```

Por la imagen de otro servicio Cloud. Los tiempos para desplegar un entorno para evaluar proyectos basados en lenguaje C son menores al correspondiente despliegue para evaluar proyectos Java debido a que en este último lenguaje tenemos que instalar Maven y SonarQube con su respectivo plugin. Si escogemos como lenguaje para el portafolio virtual Java o C, en los dos casos se instalará Java y Node.js. Java porque es requisito para ejecutar Jenkins y Node.js para realizar scripts que crean cuentas, repositorios en GitLab y crean Jobs en Jenkins de cada práctica.

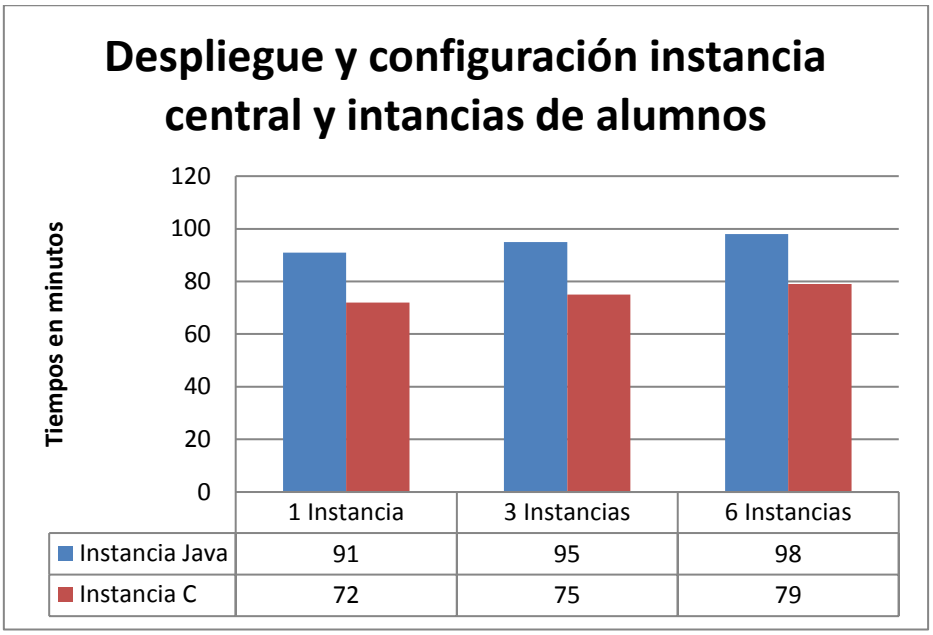



FIGURA 28. TIEMPO DESPLIEGUE Y CONFIGURACIÓN MÚLTIPLES INSTANCIAS (EN MINUTOS)

Los tiempos son similares si desplegamos 1,3,6 Instancias de alumnos. Esto ocurre porque las instancias de alumnos se configuran en paralelo y el IM precisa el mismo tiempo para desplegar 1 que para desplegar 6 instancias.

En la base de datos contamos con campos que nos indica a qué lenguaje pertenece cada receta desplegada y con estos datos podemos sacar gráficas de cual es lenguaje que cuenta con un mayor número de despliegue y el número de instancia de alumnos.

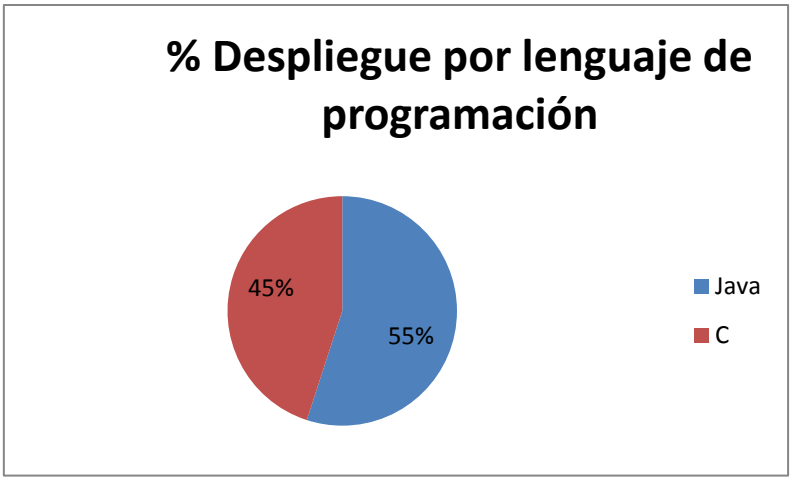



FIGURA 29. DESPLIEGUES POR LENGUAJE DE PROGRAMACIÓN

# 6 TRABAJOS FUTUROS

Una de las tareas futuras de este proyecto es añadir nuevos tipos de proveedores Cloud como ser OpenNebula, Microsoft Azure o Google Cloud Platform, y que permita a esta aplicación abarcar el mayor número de infraestructuras posibles. De esta forma, darle al instructor la posibilidad de elegir el proveedor que se adapte a su presupuesto. El IM tiene soporte para desplegar nuestra infraestructura en dichos proveedores. Lo que tenemos que tener en cuenta para el despliegue en estos proveedores son las diferentes credenciales.

```
id = one; type = OpenNebula; host = osenserver:2633; username = user; password =
pass

id = gce; type = GCE; username = username.apps.googleusercontent.com; password =
pass; project = projectname

id = azure; type = Azure; username = subscription-id; public_key = file(cert.pem);
private_key = file(key.pem)
```

El IM tiene una convención de nombres para las URIs a la hora del registro de IMVs en el VMRC, cuando las IMVs están almacenadas en los propios proveedores Cloud. El uso de esta convención permite al IM conocer su ubicación y cómo acceder a ellas. El campo de protocolo de la URI se utiliza para especificar el tipo de proveedor Cloud: one (OpenNebula), azr (Azure), gce (Google App). En el caso de OpenNebula, los campos server y port tienen su función por estándar y la ruta se utiliza para especificar el ID de las imágenes, un número entero para OpenNebula. En el caso de gce, el campo

de región se utiliza para especificar la región en la que se almacena la imagen y la ruta de acceso para almacenar el nombre de la imagen.

```
one://<server>:<port>/<image-id>
gce://<region>/<image-id>
azr://<image-id>
```

Con estas modificaciones a la generación de recetas RADL de nuestra aplicación será capaz de realizar el despliegue del portafolio en actividades de programación en otros proveedores Cloud.

Para incluir nuevos lenguajes de programación las modificaciones que se deberá de tener en cuenta:

- La creación de recetas genérica tanto para la instancia central y la instancia de los alumnos incluyendo la instalación de este lenguaje y el repositorio de configuración que permite la interacción con GitLab y Jenkins.
- Adecuación de un proyecto de dicho lenguaje de programación. Este proyecto debe de incluir dos archivos como mínimo, dependerá del lenguaje de programación a incluir y de los tipos de pruebas unitarias a realizar, un archivo que describa un conjunto de métodos. Si es un lenguaje orientado a objetos, definimos una plantilla o clase que describe las características y el comportamiento de un conjunto de objetos, y segundo archivo que realice las pruebas unitarias a estos métodos. Estos archivos estarán almacenados en repositorios externos como GitHub, Bitbucket, etc. y serán clonados y subidos a los repositorios de GitLab que desplegamos.
- Finalmente, se deberá crear un Job en Jenkins que incluya el proyecto recientemente creado que realice la ejecución de las pruebas unitarias. Con esto terminado, se deberá realizar un backup del Job en Jenkins y extraer el fichero XML.

Con estos pasos incluiremos un nuevo lenguaje de programación a nuestra aplicación.

Como tarea más ambiciosa tenemos la reconfiguración de la infraestructura desplegada desde la *instancia central* usando una interfaz web configurada para cumplir con estas expectativas:

1) Incluir nuevas cuentas de alumnos y modificar las existentes.
2) Crear nuevos repositorios y Jobs.
3) Agregar nuevas Instancias y eliminar las existentes.

# 7 CONCLUSIÓN

Este TFM ha desarrollado una plataforma web para la definición de Entornos Virtuales (EVs) computacionales, que son desplegados sobre una plataforma de Cloud público (concretamente Amazon Web Services), para soportar la evaluación automatizada de portafolios de actividades de programación.

Como conclusión se observa que el empleo de los EVs fomentará una evaluación objetiva del portafolio de una asignatura de programación. Para el alumno, brinda un entorno que utiliza habitualmente en los laboratorios de prácticas, de forma rápida y sencilla, sin necesidad de lidiar con detalles de instalación y configuración de software. Este trabajo se complementa con la integración de la herramienta de auto-testeo de programas.

Dichas mejoras son:

- ∇ Accesibilidad a los EVs necesarios para la realización de las prácticas (software, licencias, IDE, Repositorios, etc.), de tal forma que todos los alumnos tengan un entorno de trabajo homogéneo con las prácticas descargadas de sus repositorios personales creado en el despliegue, además de poder acceder desde un navegador web en dispositivos móviles y desktop, modificar sus prácticas usando GitLab web.
- ∇ Procesamiento de pruebas unitarias con proyectos pre-configurados desde el momento de despliegue dando al instructor un entorno listo para realizar sus actividades académicas y realizar las pruebas unitarias de cada método pertinente a la práctica educativa. Posteriormente, es posible tener informes e indicadores de calidad de las prácticas que cumplen con las pruebas unitarias definidos por el instructor.
- ∇ Un generador de recetas RADL, que a partir de los requerimientos introducidos por el instructor estos plasmarlos en el lenguaje declarativo de alto nivel que

soporta el IM: RADL. La descarga de esta receta y modificarla, aumentando servicios adicionales para este proyecto y desplegarla en el IM. Evitando la necesidad de crear previamente las imágenes de máquinas virtuales pre-configuradas.

- ∇ Si se despliega desde la aplicación se tiene un historial de todos los despliegues realizados guardados en base de datos.
- ∇ La aplicación creada en este TFM que utiliza tecnologías y proyectos Open Source, y se ha liberada bajo la licencia "Apache License, Version 2.0" se ha puesto a disposición de la comunidad[11].

[1] El código fuente del proyecto está disponible en: https://github.com/grycap/odisea

# 8 BIBLIOGRAFÍA


[1] *Máquinas Virtuales en las clases de Informática.* **Martín, Pedro Pablo Gómez.** Madrid : s.n., 2006.

[2] *The Financial impact of teaching surgical.* **M. BRIDGES, D.L. DIAMOND.** s.l. : American journal of surgery, 1999.

[3] *Creación de entornos virtuales para fomentar el trabajo en grupo y la racionalización de recursos.* **A. MARTÍNEZ, N. CASTILLA, D. SEGRELLES, G. MOLTÓ.** Valencia (ES) : XXII Congreso Universitario de Innovación Educativa en las Enseñanzas Técnicas, 2014.

[4] *The NIST Definition of Cloud Computing.* **Peter Mell, Timothy Grance.** 2011, National Institute of Standards and Technology, pág. 7.

[5] **Fernández, Miguel Caballer.** *Gestión de infraestructuras virtuales configuradas dinámicamente.* Valencia (ES) : Universitat Politècnica de València, 2014.

[6] *Overview of Amazon Web Services.* **Jinesh Varia, Sajee Mathew.** s.l. : Amazon Web Services, 2014.

[7] **Virtual Machine image Repository & Catalog.** http://www.grycap.upv.es/vmrc/documentation.php. [En línea] Grycap, 2015. [Citado el: 20 de 5 de 2016.]

[8] **Infrastructure Manager.** http://www.grycap.upv.es/im/documentation.php. [En línea] 2015. [Citado el: 4 de 5 de 2016.]

[9] **Geerling, Jeff.** *Ansible for DevOps.* s.l. : ANSIBLE EBOOKS, 2014.

[10] **Straub, Scott Chacon and Bend.** *Pro Git.* s.l. : Apress, 2014.

[11] **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** *Control de versiones con Subversion.* s.l. : Subversion, 2002.

[12] **Perforce.** https://www.perforce.com/customers/success-stories/cig-controls-growing-universe-perforce-helix. [En línea] 2012. [Citado el: 4 de 5 de 2016.]

[13] **GIT.** https://git-scm.com/book/es/v2/Git-en-el-Servidor-GitLab. [En línea] 7 de 2013.

[14] **Oterino, Ana M. Del Carmen García.** http://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.html. [En línea] 2014. [Citado el: 8 de 5 de 2016.]

[15] **Jalis, Asim.** http://cutest.sourceforge.net/. [En línea] 2003. [Citado el: 10 de 7 de 2016.]

[16] **JUnit.** http://junit.org/junit4/cookbook.html. [En línea] 2002. [Citado el: 29 de 5 de 2016.]

[17] **Tutorials Point.** https://www.tutorialspoint.com/junit/index.htm. [En línea] 2016. [Citado el: 20 de 5 de 2016.]

[18] **Ortíz, David.** http://www.agiliacenter.com/blog/integracion-continua-y-jenkins/. [En línea] 28 de 1 de 2014. [Citado el: 8 de 5 de 2016.]

[19] **Drone.** https://drone.io/. [En línea] 2013. [Citado el: 20 de 6 de 2016.]

[20] **Warner, Brian.** http://buildbot.net/. [En línea] 29 de 4 de 2003. [Citado el: 18 de 5 de 2016.]

[21] **Travis CI community.** https://travis-ci.org/. [En línea] 2011. [Citado el: 16 de 8 de 2016.]

[22] **Caro, Federico.** https://www.paradigmadigital.com/dev/sonar-medida-de-la-calidad-de-tu-codigo/. [En línea] 16 de 9 de 2010. [Citado el: 19 de 6 de 2016.]

[23] **SonarQube.** http://www.sonarqube.org/. [En línea] 2008. [Citado el: 19 de 6 de 2016.]

[24] **Zhang, Scott.** https://docs.microsoft.com/es-es/azure/virtual-machines/virtual-machines-linux-classic-remote-desktop. [En línea] 19 de 8 de 2016. [Citado el: 25 de 10 de 2016.]

[25] **LXDE.** https://wiki.lxde.org/en/Main_Page. [En línea] 2006. [Citado el: 25 de 10 de 2016.]

[26] **CodeBlocks.** http://www.codeblocks.org/. [En línea] 2005. [Citado el: 28 de 9 de 2016.]

[27] **Gitlab.** https://about.gitlab.com/. [En línea] 2011. [Citado el: 28 de 9 de 2016.]

[28] **Amazon Web Service.** https://calculator.s3.amazonaws.com/index.html. [En línea] 2016. [Citado el: 8 de 6 de 2016.]

[29] *Uniform Resource Locators (URL).* **L. Masinter, Xerox Corporation, M. McCahill.** s.l. : University of Minnesota, 1994.

[30] **Node.js.** https://nodejs.org/es/. [En línea] 2016. [Citado el: 19 de 7 de 2016.]

[31] *Architectural Styles and the Design of Network-based Software Architectures.* **Fielding, Roy Thomas.** California : UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[32] **Cerami, Ethan.** *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI.* s.l. : O'REILLY, 2002.

[33] **Laravel.** https://laravel.com/. [En línea] 2011. [Citado el: 3 de 4 de 2016.]

[34] *Uniform Resource Identifier (URI).* **Fielding, Roy.** s.l. : Adobe Systems, 2005.

[35] **Fuentes, Juan Mariano.** *Manual de AJAX.* 2009.

[36] **Jonathan Chaffer, Karl Swedberg.** *Learning jQuery.* s.l. : PACKT, 2013.

[37] **Ubuntu.** https://jujucharms.com/. [En línea] 2016. [Citado el: 10 de 4 de 2016.]

[38] **Gitlab.** https://docs.gitlab.com/ee/api/README.html. [En línea] 2011. [Citado el: 29 de 7 de 2016.]

[39] **Fischer, Nate.** http://documentup.com/shelljs/shelljs. [En línea] 2016. [Citado el: 10 de 5 de 2016.]

[40] **Jenkins.** https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Script+Console. [En línea] 2016. [Citado el: 6 de 6 de 2016.]

[41] **Silas.** https://www.npmjs.com/package/jenkins. [En línea] 2016. [Citado el: 26 de 7 de 2016.]

[42] *JUnit: unit testing and coiling in tandem.* **Louridas, P.** s.l. : IEEE Software, 2005, Vol. 22.

[43] **Rafael, Borja Lázaro de.** https://www.adictosaltrabajo.com/tutoriales/sonarqube-4-5-2/. [En línea] 2015. [Citado el: 9 de 8 de 2016.]

[44] **AWS Educate.** https://aws.amazon.com/es/education/awseducate/

[45] **RADL.** Resource and Application Description Language. http://imdocs.readthedocs.io/en/latest/radl.html

[46] **Ansible.** www.ansible.com

[47] **Docker.** www.docker.com

[48] **Complejidad Ciclomática.** https://es.wikipedia.org/wiki/Complejidad_ciclom%C3%A1tica

# 9 ANEXOS

Receta RADL del lenguaje de programación C, generado desde la aplicación para la "instancia central"

```
network publica (
  outbound = 'yes' and outports = '22,80,8089,3389,9000'
)
network privada ()
system central (
  cpu.arch='x86_64' and
  cpu.count>=1 and
  memory.size>=1740m and
  net_interface.0.connection = 'publica' and
  net_interface.1.connection = 'privada' and
  net_interface.0.dns_name = 'central-mv' and
  net_interface.1.dns_name = 'central-priv' and
  disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
  disk.0.os.name='linux' and
  disk.0.os.credentials.username='ubuntu' and
  disk.0.os.version='14.04' and
  disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)


configure central (
@begin
---
  - vars:
      accounts:
        - { name: "studentapp", pw: "gu2KmqcJp0Yyo", pass: "studentapp"}
      admin:
        - { name: "adminapp", pw: "gu2KmqcJp0Yyo", pass: "adminapp"}
      ipmaster: localhost
      jenkinsnode: /home/ubuntu/node-jenkins
      jenkinsnodeurl: https://master-class:master-
password@bitbucket.org/phantro/node-jenkins-c.git
      jenkinsserver: https://master-class:master-
password@bitbucket.org/phantro/jenkins-server.git
      jenkins: /home/ubuntu/jenkins-server
      userproyect: proyect-user.sh

    tasks:
    - name: Install Git 0f485338872
      apt: name=git update_cache=yes state=latest

    - name: Install Curl
      apt: name=curl update_cache=yes state=latest

    - name: Update repository Node.js
      shell: 'curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -'

    - name: Install Node.js
      apt: name=nodejs update_cache=yes state=latest

    - name: Install Dependency Node.js
      apt: name=build-essential update_cache=yes state=latest

    - name: Create user accounts student
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: accounts
```

```
    - name: Modify $HOME permissions student
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: accounts

    - name: Create user admin
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: admin

    - name: Modify $HOME permissions admin
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: admin

    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
    - service: name=ssh state=restarted

    - name: Add sudo admin
      shell: 'adduser {{ item.name }} sudo'
      with_items: admin

    - name: Install Openssh
      apt: name=openssh-server update_cache=yes state=latest
    - name: Install Certificates
      apt: name=ca-certificates update_cache=yes state=latest
    - name: Update repository sh
      shell: 'curl -sS
https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh |
sudo bash'
    - name: Install Gitlab
      apt: name=gitlab-ce update_cache=yes state=latest

    - name: Initial Gitlab
      shell: 'gitlab-ctl reconfigure'

    - name: App | Cloning repos + backup proyect
      git: repo={{ jenkinsnodeurl }}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkinsnode }}"
          repo: PrimaryRepo

    - copy: dest={{ jenkinsnode }}/user.json
content='{"user1":{"nombre":"studentapp","password":"studentapp"}}'

    - copy: dest={{ jenkinsnode }}/admin.json
content='{"user1":{"name":"adminapp","password":"adminapp"}}'

    - copy: dest={{ jenkinsnode }}/repository.json
content='{"practice1":{"name":"practica-1","practice":"https://master-class:master-
password@bitbucket.org/phantro/completo-prueba.git","central":"https://master-
class:master-password@bitbucket.org/phantro/prueba-unitaria-c.git"}}'

    - name: Add repo for java 8 0f168424895
      apt_repository: repo='ppa:openjdk-r/ppa' state=present

    - name: Install java 8
      apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
```

```
      sudo: yes

    - lineinfile: dest=/etc/bash.bashrc regexp="export JAVA_HOME=/usr/lib/jvm/java-
8-oracle/" line="export JAVA_HOME=/usr/lib/jvm/java-8-oracle/" create=yes

    - name: App | Cloning repos + submodules + jenkins 0f452458482
      git: repo={{jenkinsserver}}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkins }}"
          repo: PrimaryRepo

    - name: Install unzip
      apt: pkg=unzip state=latest update_cache=yes

    - name: Run Jenkins
      shell: '{{jenkins}}/jenkins.sh start'

    - command: /bin/sleep 180

    - name: Install codeblocks
      apt: name=codeblocks update_cache=yes state=latest

    - name: Install lxde
      apt: name=lxde update_cache=yes state=latest

    - name: Run lxdm
      shell: 'start lxdm'

    - name: Install xrdp
      apt: name=xrdp update_cache=yes state=latest

    - name: Wait for Jenkins to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}:8089/cli/"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("200 OK") != -1) and (result.stdout.find("Please wait while")
== -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Wait for Gitlab to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}/api/v3/user"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("401 Unauthorized") != -1) and (result.stdout.find("Please wait
while") == -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Update pluguin Jenkins and Proyect Gitlab
      shell: 'sh {{jenkinsnode}}/ejecutar.sh >> detail.txt'

    - copy: dest=/home/{{ item.name }}/{{ userproyect }} content='#!/bin/bash\n git
clone http://{{ item.name }}:{{ item.pass }}@localhost/{{ item.name }}/practica-
1.git /home/{{ item.name }}/practica-1\n ' mode=0755
      with_items: accounts

    - name: Clone student repository
      shell: 'sh /home/{{ item.name }}/{{ userproyect }}'
```

```
      with_items: accounts

    - name: Modify $HOME permissions student repository
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700 recurse=yes
      with_items: accounts

    - name: Finish 0f412483458
      shell: 'echo "Finish Despliegue"'
@end
)

deploy central 1
```

Receta RADL del lenguaje de programación C, generado desde la aplicación para la "instancia central" e "instancia de alumnos"

```
network publica (
  outbound = 'yes' and outports = '22,80,8089,3389,9000'
)
network privada ()
system central (
  cpu.arch='x86_64' and
  cpu.count>=1 and
  memory.size>=1740m and
  net_interface.0.connection = 'publica' and
  net_interface.1.connection = 'privada' and
  net_interface.0.dns_name = 'central-mv' and
  net_interface.1.dns_name = 'central-priv' and
  disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
  disk.0.os.name='linux' and
  disk.0.os.credentials.username='ubuntu' and
  disk.0.os.version='14.04' and
  disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)
system mv (
  cpu.arch='x86_64' and
  cpu.count>=1 and
  memory.size>=512m and
  net_interface.0.connection = 'publica' and
  net_interface.1.connection = 'privada' and
  net_interface.0.dns_name = 'student-mv-#N#' and
  net_interface.1.dns_name = 'student-priv-#N#' and
  disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
  disk.0.os.name='linux' and
  disk.0.os.credentials.username='ubuntu' and
  disk.0.os.version='14.04' and
  disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)

configure central (
@begin
---
  - vars:
      accounts:
        - { name: "studentapp", pw: "gu2KmqcJp0Yyo", pass: "studentapp"}
      admin:
        - { name: "adminapp", pw: "gu2KmqcJp0Yyo", pass: "adminapp"}
      ipmaster: localhost
      jenkinsnode: /home/ubuntu/node-jenkins
```

```
      jenkinsnodeurl: https://master-class:master-
password@bitbucket.org/phantro/node-jenkins-c.git
      jenkinsserver: https://master-class:master-
password@bitbucket.org/phantro/jenkins-server.git
      jenkins: /home/ubuntu/jenkins-server
      userproyect: proyect-user.sh

    tasks:
    - name: Install Git 0f485338872
      apt: name=git update_cache=yes state=latest

    - name: Install Curl
      apt: name=curl update_cache=yes state=latest

    - name: Update repository Node.js
      shell: 'curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -'

    - name: Install Node.js
      apt: name=nodejs update_cache=yes state=latest

    - name: Install Dependency Node.js
      apt: name=build-essential update_cache=yes state=latest

    - name: Create user accounts student
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: accounts

    - name: Modify $HOME permissions student
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: accounts

    - name: Create user admin
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: admin

    - name: Modify $HOME permissions admin
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: admin

    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
    - service: name=ssh state=restarted

    - name: Add sudo admin
      shell: 'adduser {{ item.name }} sudo'
      with_items: admin

    - name: Install Openssh
      apt: name=openssh-server update_cache=yes state=latest
    - name: Install Certificates
      apt: name=ca-certificates update_cache=yes state=latest
    - name: Update repository sh
      shell: 'curl -sS
https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh |
sudo bash'
    - name: Install Gitlab
      apt: name=gitlab-ce update_cache=yes state=latest

    - name: Initial Gitlab
      shell: 'gitlab-ctl reconfigure'

    - name: App | Cloning repos + backup proyect
```

```
      git: repo={{ jenkinsnodeurl }}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkinsnode }}"
          repo: PrimaryRepo

    - copy: dest={{ jenkinsnode }}/user.json
content='{"user1":{"nombre":"studentapp","password":"studentapp"}}'
    - copy: dest={{ jenkinsnode }}/admin.json
content='{"user1":{"name":"adminapp","password":"adminapp"}}'
    - copy: dest={{ jenkinsnode }}/repository.json
content='{"practice1":{"name":"practica-1","practice":"https://master-class:master-
password@bitbucket.org/phantro/completo-prueba.git","central":"https://master-
class:master-password@bitbucket.org/phantro/prueba-unitaria-c.git"}}'

    - name: Add repo for java 8 0f168424895
      apt_repository: repo='ppa:openjdk-r/ppa' state=present

    - name: Install java 8
      apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
      sudo: yes

    - lineinfile: dest=/etc/bash.bashrc regexp="export JAVA_HOME=/usr/lib/jvm/java-
8-oracle/" line="export JAVA_HOME=/usr/lib/jvm/java-8-oracle/" create=yes

    - name: App | Cloning repos + submodules + jenkins 0f452458482
      git: repo={{jenkinsserver}}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkins }}"
          repo: PrimaryRepo

    - name: Install unzip
      apt: pkg=unzip state=latest update_cache=yes

    - name: Run Jenkins
      shell: '{{jenkins}}/jenkins.sh start'

    - command: /bin/sleep 180

    - name: Install codeblocks
      apt: name=codeblocks update_cache=yes state=latest

    - name: Install lxde
      apt: name=lxde update_cache=yes state=latest

    - name: Run lxdm
      shell: 'start lxdm'

    - name: Install xrdp
      apt: name=xrdp update_cache=yes state=latest

    - name: Wait for Jenkins to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}:8089/cli/"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("200 OK") != -1) and (result.stdout.find("Please wait while")
== -1)
```

```
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Wait for Gitlab to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}/api/v3/user"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("401 Unauthorized") != -1) and (result.stdout.find("Please wait
while") == -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Update pluguin Jenkins and Proyect Gitlab
      shell: 'sh {{jenkinsnode}}/ejecutar.sh >> detail.txt'

    - copy: dest=/home/{{ item.name }}/{{ userproyect }} content='#!/bin/bash\n git
clone http://{{ item.name }}:{{ item.pass }}@central-priv/{{ item.name }}/practica-
1.git /home/{{ item.name }}/practica-1\n ' mode=0755
      with_items: accounts

    - name: Clone student repository
      shell: 'sh /home/{{ item.name }}/{{ userproyect }}'
      with_items: accounts

    - name: Modify $HOME permissions student repository
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700 recurse=yes
      with_items: accounts

@end
)

configure mv (
@begin
---
  - vars:
      accounts:
        - { name: "studentapp", pw: "gu2KmqcJp0Yyo", pass: "studentapp"}
      admin:
        - { name: "adminapp", pw: "gu2KmqcJp0Yyo", pass: "adminapp"}
      ipmaster: central-priv
      userproyect : proyect-user.sh
    tasks:
    - name: Install Git
      apt: name=git update_cache=yes state=latest

    - name: Create user accounts student
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: accounts

    - name: Modify $HOME permissions student
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: accounts

    - name: Create user admin
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: admin

    - name: Modify $HOME permissions admin
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: admin
```

```
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
    - service: name=ssh state=restarted

    - name: Add sudo admin
      shell: 'adduser {{ item.name }} sudo'
      with_items: admin

    - name: Add repo for java 8 0f168424895
      apt_repository: repo='ppa:openjdk-r/ppa' state=present

    - name: Install java 8
      apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
      sudo: yes

    - lineinfile: dest=/etc/bash.bashrc regexp="export JAVA_HOME=/usr/lib/jvm/java-
8-oracle/" line="export JAVA_HOME=/usr/lib/jvm/java-8-oracle/" create=yes

    - copy: dest=/home/{{ item.name }}/{{ userproyect }} content='#!/bin/bash\n git
clone http://{{ item.name }}:{{ item.pass }}@central-priv/{{ item.name }}/practica-
1.git /home/{{ item.name }}/practica-1\n ' mode=0755
      with_items: accounts

    - name: Clone student repository
      shell: 'sh /home/{{ item.name }}/{{ userproyect }}'
      with_items: accounts

    - name: Modify $HOME permissions student repository
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700 recurse=yes
      with_items: accounts

    - name: Install codeblocks
      apt: name=codeblocks update_cache=yes state=latest

    - name: Install lxde
      apt: name=lxde update_cache=yes state=latest

    - name: Run lxdm
      shell: 'start lxdm'

    - name: Install xrdp
      apt: name=xrdp update_cache=yes state=latest

    - name: Finish 0f412483458
      shell: 'echo "Finish Despliegue"'
@end
)

deploy central 1
deploy mv 1

contextualize (
    system central configure central step 1
    system mv configure mv step 2
)
```

Receta RADL del lenguaje de programación Java, generado desde la aplicación para la "instancia central"

```
network publica (
  outbound = 'yes' and outports = '22,80,8089,3389,9000'
)
network privada ()
system central (
  cpu.arch='x86_64' and
  cpu.count>=1 and
  memory.size>=1740m and
  net_interface.0.connection = 'publica' and
  net_interface.1.connection = 'privada' and
  net_interface.0.dns_name = 'central-mv' and
  net_interface.1.dns_name = 'central-priv' and
  disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
  disk.0.os.name='linux' and
  disk.0.os.credentials.username='ubuntu' and
  disk.0.os.version='14.04' and
  disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)


configure central (
@begin
---
  - vars:
      accounts:
        - { name: "studentapp", pw: "gu2KmqcJp0Yyo", pass: "studentapp"}
      admin:
        - { name: "adminapp", pw: "gu2KmqcJp0Yyo", pass: "adminapp"}
      home: /home/ubuntu/
      baseurl: https://master-class:master-password@bitbucket.org/phantro/base-
java.git
      ipmaster: localhost
      jenkinsnode: /home/ubuntu/node-jenkins
      jenkinsnodeurl: https://master-class:master-
password@bitbucket.org/phantro/node-jenkins.git
      jenkinsserver: https://master-class:master-
password@bitbucket.org/phantro/jenkins-server.git
      jenkins: /home/ubuntu/jenkins-server
      userproyect: proyect-user.sh

    tasks:
    - name: Install Git 0f485338872
      apt: name=git update_cache=yes state=latest

    - name: Install Curl
      apt: name=curl update_cache=yes state=latest

    - name: Update repository Node.js
      shell: 'curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -'

    - name: Install Node.js
      apt: name=nodejs update_cache=yes state=latest

    - name: Install Dependency Node.js
      apt: name=build-essential update_cache=yes state=latest

    - name: Create user accounts student
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: accounts

    - name: Modify $HOME permissions student
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: accounts
```

```
    - name: Create user admin
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: admin

    - name: Modify $HOME permissions admin
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: admin

    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
    - service: name=ssh state=restarted

    - name: Add sudo admin
      shell: 'adduser {{ item.name }} sudo'
      with_items: admin

    - name: Install Openssh
      apt: name=openssh-server update_cache=yes state=latest
    - name: Install Certificates
      apt: name=ca-certificates update_cache=yes state=latest
    - name: Update repository sh
      shell: 'curl -sS
https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh |
sudo bash'
    - name: Install Gitlab
      apt: name=gitlab-ce update_cache=yes state=latest

    - name: Initial Gitlab
      shell: 'gitlab-ctl reconfigure'

    - name: App | Cloning repos + backup proyect
      git: repo={{ jenkinsnodeurl }}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkinsnode }}"
          repo: PrimaryRepo

    - copy: dest={{ jenkinsnode }}/user.json
content='{"user1":{"nombre":"studentapp","password":"studentapp"}}'
    - copy: dest={{ jenkinsnode }}/admin.json
content='{"user1":{"name":"adminapp","password":"adminapp"}}
    - copy: dest={{ jenkinsnode }}/repository.json
content='{"practice1":{"name":"practica-1","practice":"https://master-class:master-
password@bitbucket.org/phantro/base-java.git","central":"https://master-
class:master-password@bitbucket.org/phantro/base-java.git"}}'

    - name: Add repo for java 8 0f168424895
      apt_repository: repo='ppa:openjdk-r/ppa' state=present

    - name: Install java 8
      apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
      sudo: yes

    - lineinfile: dest=/etc/bash.bashrc regexp="export JAVA_HOME=/usr/lib/jvm/java-
8-oracle/" line="export JAVA_HOME=/usr/lib/jvm/java-8-oracle/" create=yes

    - name: App | Cloning repos + submodules + jenkins 0f452458482
      git: repo={{jenkinsserver}}
           dest={{ item.dest }}
```

```
            accept_hostkey=yes
            force=yes
            recursive=no
      with_items:
        -
          dest: "{{ jenkins }}"
          repo: PrimaryRepo

    - name: Download Maven
      get_url: url=http://archive.apache.org/dist/maven/binaries/apache-maven-
3.0.4-bin.tar.gz dest={{home}}

    - name: Unarchive maven
      unarchive: src={{home}}apache-maven-3.0.4-bin.tar.gz dest={{home}} copy=no

    - name: Delete Apache.tar.gz
      file: path={{home}}apache-maven-3.0.4-bin.tar.gz state=absent recurse=no

    - command: mv {{home}}apache-maven-3.0.4 /usr/local

    - name: Run Maven
      shell: 'sudo ln -s /usr/local/apache-maven-3.0.4/bin/mvn /usr/bin/mvn'

    - name: Download Sonar
      get_url:
url=https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-4.5.7.zip
dest={{home}}

    - name: Install unzip
      apt: pkg=unzip state=latest update_cache=yes

    - name: Unarchive Sonar
      unarchive: src={{home}}sonarqube-4.5.7.zip dest={{home}} copy=no

    - name: Delete Sonar.zip
      file: path={{home}}sonarqube-4.5.7.zip state=absent recurse=no

    - name: Run Sonar
      shell: 'cd {{home}}sonarqube-4.5.7 && bin/linux-x86-64/sonar.sh console &'

    - command: /bin/sleep 180

    - name: Run Jenkins
      shell: '{{jenkins}}/jenkins.sh start'

    - command: /bin/sleep 180

    - name: Install codeblocks
      apt: name=codeblocks update_cache=yes state=latest

    - name: Install lxde
      apt: name=lxde update_cache=yes state=latest

    - name: Run lxdm
      shell: 'start lxdm'

    - name: Install xrdp
      apt: name=xrdp update_cache=yes state=latest

    - name: Wait for Jenkins to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}:8089/cli/"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("200 OK") != -1) and (result.stdout.find("Please wait while")
== -1)
      retries: "60"
```

```
      delay: "5"
      changed_when: false

    - name: Wait for Gitlab to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}/api/v3/user"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("401 Unauthorized") != -1) and (result.stdout.find("Please wait
while") == -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Wait for Sonar to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}:9000/"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("200 OK") != -1) and (result.stdout.find("Please wait while")
== -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Update pluguin Jenkins and Proyect Gitlab
      shell: 'sh {{jenkinsnode}}/ejecutar.sh >> detail.txt'

    - copy: dest=/home/{{ item.name }}/{{ userproyect }} content='#!/bin/bash\n git
clone http://{{ item.name }}:{{ item.pass }}@localhost/{{ item.name }}/practica-
1.git /home/{{ item.name }}/practica-1\n ' mode=0755
      with_items: accounts

    - name: Clone student repository
      shell: 'sh /home/{{ item.name }}/{{ userproyect }}'
      with_items: accounts

    - name: Modify $HOME permissions student repository
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700 recurse=yes
      with_items: accounts

    - name: Finish 0f412483458
      shell: 'echo "Finish Despliegue"'
@end
)

deploy central 1
```

Receta RADL del lenguaje de programación Java, generado desde la aplicación para la "instancia central" e "instancia de alumnos"

```
network publica (
  outbound = 'yes' and outports = '22,80,8089,3389,9000'
)
network privada ()
system central (
  cpu.arch='x86_64' and
  cpu.count>=1 and
  memory.size>=1740m and
  net_interface.0.connection = 'publica' and
  net_interface.1.connection = 'privada' and
```

```
  net_interface.0.dns_name = 'central-mv' and
  net_interface.1.dns_name = 'central-priv' and
  disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
  disk.0.os.name='linux' and
  disk.0.os.credentials.username='ubuntu' and
  disk.0.os.version='14.04' and
  disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)
system mv (
  cpu.arch='x86_64' and
  cpu.count>=1 and
  memory.size>=512m and
  net_interface.0.connection = 'publica' and
  net_interface.1.connection = 'privada' and
  net_interface.0.dns_name = 'student-mv-#N#' and
  net_interface.1.dns_name = 'student-priv-#N#' and
  disk.0.image.url = 'aws://us-east-1/ami-c93a83de' and
  disk.0.os.name='linux' and
  disk.0.os.credentials.username='ubuntu' and
  disk.0.os.version='14.04' and
  disk.0.applications contains (name='aws.region.us-east-1' and preinstalled='yes')
)

configure central (
@begin
---
  - vars:
      accounts:
        - { name: "studentapp", pw: "gu2KmqcJp0Yyo", pass: "studentapp"}
      admin:
        - { name: "adminapp", pw: "gu2KmqcJp0Yyo", pass: "adminapp"}
      home: /home/ubuntu/
      baseurl: https://master-class:master-password@bitbucket.org/phantro/base-
java.git
      ipmaster: localhost
      jenkinsnode: /home/ubuntu/node-jenkins
      jenkinsnodeurl: https://master-class:master-
password@bitbucket.org/phantro/node-jenkins.git
      jenkinsserver: https://master-class:master-
password@bitbucket.org/phantro/jenkins-server.git
      jenkins: /home/ubuntu/jenkins-server
      userproyect: proyect-user.sh

    tasks:
    - name: Install Git 0f485338872
      apt: name=git update_cache=yes state=latest

    - name: Install Curl
      apt: name=curl update_cache=yes state=latest

    - name: Update repository Node.js
      shell: 'curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -'

    - name: Install Node.js
      apt: name=nodejs update_cache=yes state=latest

    - name: Install Dependency Node.js
      apt: name=build-essential update_cache=yes state=latest

    - name: Create user accounts student
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: accounts

    - name: Modify $HOME permissions student
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
```

```
      with_items: accounts

    - name: Create user admin
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: admin

    - name: Modify $HOME permissions admin
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: admin

    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
    - service: name=ssh state=restarted

    - name: Add sudo admin
      shell: 'adduser {{ item.name }} sudo'
      with_items: admin

    - name: Install Openssh
      apt: name=openssh-server update_cache=yes state=latest
    - name: Install Certificates
      apt: name=ca-certificates update_cache=yes state=latest
    - name: Update repository sh
      shell: 'curl -sS
https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh |
sudo bash'
    - name: Install Gitlab
      apt: name=gitlab-ce update_cache=yes state=latest

    - name: Initial Gitlab
      shell: 'gitlab-ctl reconfigure'

    - name: App | Cloning repos + backup proyect
      git: repo={{ jenkinsnodeurl }}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkinsnode }}"
          repo: PrimaryRepo

    - copy: dest={{ jenkinsnode }}/user.json
content='{"user1":{"nombre":"studentapp","password":"studentapp"}}'
    - copy: dest={{ jenkinsnode }}/admin.json
content='{"user1":{"name":"adminapp","password":"adminapp"}}'
    - copy: dest={{ jenkinsnode }}/repository.json
content='{"practice1":{"name":"practica-1","practice":"https://master-class:master-
password@bitbucket.org/phantro/base-java.git","central":"https://master-
class:master-password@bitbucket.org/phantro/base-java.git"}}'

    - name: Add repo for java 8 0f168424895
      apt_repository: repo='ppa:openjdk-r/ppa' state=present

    - name: Install java 8
      apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
      sudo: yes

    - lineinfile: dest=/etc/bash.bashrc regexp="export JAVA_HOME=/usr/lib/jvm/java-
8-oracle/" line="export JAVA_HOME=/usr/lib/jvm/java-8-oracle/" create=yes

    - name: App | Cloning repos + submodules + jenkins 0f452458482
```

```
      git: repo={{jenkinsserver}}
           dest={{ item.dest }}
           accept_hostkey=yes
           force=yes
           recursive=no
      with_items:
        -
          dest: "{{ jenkins }}"
          repo: PrimaryRepo

    - name: Download Maven
      get_url: url=http://archive.apache.org/dist/maven/binaries/apache-maven-
3.0.4-bin.tar.gz dest={{home}}

    - name: Unarchive maven
      unarchive: src={{home}}apache-maven-3.0.4-bin.tar.gz dest={{home}} copy=no

    - name: Delete Apache.tar.gz
      file: path={{home}}apache-maven-3.0.4-bin.tar.gz state=absent recurse=no

    - command: mv {{home}}apache-maven-3.0.4 /usr/local

    - name: Run Maven
      shell: 'sudo ln -s /usr/local/apache-maven-3.0.4/bin/mvn /usr/bin/mvn'

    - name: Download Sonar
      get_url:
url=https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-4.5.7.zip
dest={{home}}

    - name: Install unzip
      apt: pkg=unzip state=latest update_cache=yes

    - name: Unarchive Sonar
      unarchive: src={{home}}sonarqube-4.5.7.zip dest={{home}} copy=no

    - name: Delete Sonar.zip
      file: path={{home}}sonarqube-4.5.7.zip state=absent recurse=no

    - name: Run Sonar
      shell: 'cd {{home}}sonarqube-4.5.7 && bin/linux-x86-64/sonar.sh console &'

    - command: /bin/sleep 180

    - name: Run Jenkins
      shell: '{{jenkins}}/jenkins.sh start'

    - command: /bin/sleep 180

    - name: Install codeblocks
      apt: name=codeblocks update_cache=yes state=latest

    - name: Install lxde
      apt: name=lxde update_cache=yes state=latest

    - name: Run lxdm
      shell: 'start lxdm'

    - name: Install xrdp
      apt: name=xrdp update_cache=yes state=latest

    - name: Wait for Jenkins to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}:8089/cli/"
      register: result
```

```
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("200 OK") != -1) and (result.stdout.find("Please wait while")
== -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Wait for Gitlab to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}/api/v3/user"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("401 Unauthorized") != -1) and (result.stdout.find("Please wait
while") == -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Wait for Sonar to start up before proceeding.
      shell: "curl -D - --silent --max-time 5 http://{{ ipmaster }}:9000/"
      register: result
      until: (result.stdout.find("403 Forbidden") != -1) or
(result.stdout.find("200 OK") != -1) and (result.stdout.find("Please wait while")
== -1)
      retries: "60"
      delay: "5"
      changed_when: false

    - name: Update pluguin Jenkins and Proyect Gitlab
      shell: 'sh {{jenkinsnode}}/ejecutar.sh >> detail.txt'

    - copy: dest=/home/{{ item.name }}/{{ userproyect }} content='#!/bin/bash\n git
clone http://{{ item.name }}:{{ item.pass }}@central-priv/{{ item.name }}/practica-
1.git /home/{{ item.name }}/practica-1\n ' mode=0755
      with_items: accounts

    - name: Clone student repository
      shell: 'sh /home/{{ item.name }}/{{ userproyect }}'
      with_items: accounts

    - name: Modify $HOME permissions student repository
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700 recurse=yes
      with_items: accounts

@end
)

configure mv (
@begin
---
  - vars:
      accounts:
        - { name: "studentapp", pw: "gu2KmqcJp0Yyo", pass: "studentapp"}
      admin:
        - { name: "adminapp", pw: "gu2KmqcJp0Yyo", pass: "adminapp"}
      ipmaster: central-priv
      userproyect : proyect-user.sh
    tasks:
    - name: Install Git
      apt: name=git update_cache=yes state=latest

    - name: Create user accounts student
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: accounts

    - name: Modify $HOME permissions student
```

```
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: accounts

    - name: Create user admin
      user: name={{ item.name }} password={{ item.pw }} shell=/bin/bash
      with_items: admin

    - name: Modify $HOME permissions admin
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700
      with_items: admin

    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication no"
line="PasswordAuthentication no" state=absent
    - lineinfile: dest=/etc/ssh/sshd_config regexp="PasswordAuthentication yes"
line="PasswordAuthentication yes" state=present
    - service: name=ssh state=restarted

    - name: Add sudo admin
      shell: 'adduser {{ item.name }} sudo'
      with_items: admin

    - name: Add repo for java 8 0f168424895
      apt_repository: repo='ppa:openjdk-r/ppa' state=present

    - name: Install java 8
      apt: name=openjdk-8-jdk state=latest update-cache=yes force=yes
      sudo: yes

    - lineinfile: dest=/etc/bash.bashrc regexp="export JAVA_HOME=/usr/lib/jvm/java-
8-oracle/" line="export JAVA_HOME=/usr/lib/jvm/java-8-oracle/" create=yes

    - copy: dest=/home/{{ item.name }}/{{ userproyect }} content='#!/bin/bash\n git
clone http://{{ item.name }}:{{ item.pass }}@central-priv/{{ item.name }}/practica-
1.git /home/{{ item.name }}/practica-1\n ' mode=0755
      with_items: accounts

    - name: Clone student repository
      shell: 'sh /home/{{ item.name }}/{{ userproyect }}'
      with_items: accounts

    - name: Modify $HOME permissions student repository
      file: path=/home/{{ item.name }} state=directory owner={{ item.name }}
group={{ item.name }} mode=0700 recurse=yes
      with_items: accounts

    - name: Install codeblocks
      apt: name=codeblocks update_cache=yes state=latest

    - name: Install lxde
      apt: name=lxde update_cache=yes state=latest

    - name: Run lxdm
      shell: 'start lxdm'

    - name: Install xrdp
      apt: name=xrdp update_cache=yes state=latest

    - name: Finish 0f412483458
      shell: 'echo "Finish Despliegue"'
@end
)

deploy central 1
deploy mv 1
```

```
contextualize (
    system central configure central step 1
    system mv configure mv step 2
)
```