

# DevOps. Ejemplo de integración continua de procesos en un Área de Mercados y Tesorería

Juan Erasmo Plúa Gutiérrez

**Resumen**—DevOps es una metodología de desarrollo en la Ingeniería del Software basada en la comunicación, colaboración e integración entre los desarrolladores del software y los profesionales de operaciones IT. Esta metodología comporta unos beneficios dentro de la empresa como la mejora en la calidad del código y su implementación, incremento de la productividad, disminución de los costes y una alta frecuencia de entrega de paquetes de desarrollo. Este proyecto tiene como objetivo implementar y configurar una serie de herramientas que permitan aplicar este modelo de desarrollo de software dentro de la empresa, en concreto, en el Área de Mercados y Tesorería de Everis, empresa colaboradora del proyecto.

**Palabras clave**—DevOps, integración continua, automatización de procesos, Jenkins, Git, Docker, Tomcat, TomEE, AWS, Amazon Web Services, Nginx, Java, Web Application.

**Abstract**—DevOps is a development methodology in Software Engineering based on communication, collaboration and integration between software developers and IT operation professionals. This methodology involves benefits within the company such as the improvement in the quality of the code and its implementation, increased productivity, decreased costs and high frequency of delivery of development packages. This project intends to implement and configure a series of tools that this software development model can apply within the company, specifically, in the Markets and Treasury Area of Everis, project partner company.

**Index Terms**—DevOps, continuous integration, process automation, Jenkins, Git, Docker, Tomcat, TomEE, AWS, Amazon Web Services, Nginx, Java, Web application.

## 1 INTRODUCCIÓN

ESTE proyecto es una propuesta de la empresa Everis para realizar un ejemplo práctico de como implementar y configurar una serie de herramientas que conformen un nuevo modelo, idea o metodología como lo es el DevOps, y será llevado a cabo dentro del Área de Mercados y Tesorería.

DevOps (*Development and Operations*) surgió a finales de la pasada década como una solución necesaria a adoptar para un mejor funcionamiento del desarrollo ágil, el cual se basa en un desarrollo iterativo e incremental donde las entregas de paquetes de código son cada vez de mayor frecuencia (Fig. 1).

Es muy común que durante este proceso se produzcan errores de todo tipo y que el usuario o cliente dirija las responsabilidades del problema al departamento IT, lo que acaba desembocando en un conflicto entre ambos.

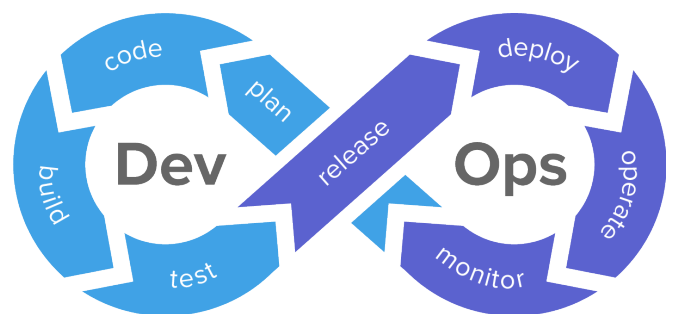


Fig. 1. Flujo de tareas de DevOps (nub8.net)

Lo que propone DevOps es establecer una amplia comunicación, colaboración e integración entre los desarrolladores que forman el departamento IT y los profesionales que forman el departamento de operaciones, ya que estos elementos son clave para que el sistema funcione y como consecuencia se vea reflejado en beneficios para la empresa. Para ello, se han de implementar una serie de herramientas que faciliten esta metodología de trabajo:

- E-mail de contacto: [JuanErasmo.Plua@e-campus.uab.cat](mailto:JuanErasmo.Plua@e-campus.uab.cat)
- Mención realizada: Ingeniería del Software
- Trabajo tutorizado por: Yolanda Benítez (Ciencias de la Computación) y Daniel Cufí (Empresa Colaboradora)
- Curso 2019/20

- Administración del código.
- Integración continua para la compilación, ejecución

- de pruebas y detección de errores del código.
- Repositorios de artefactos donde almacenar el código durante el desarrollo.
- Gestión de cambios
- Configuración de la infraestructura.
- Monitoreo del rendimiento de la aplicación.

Si bien es cierto que no existe una implementación determinada ni tampoco el uso de un software en el mercado en concreto para cada una de las herramientas mencionadas. En el siguiente apartado se analizará cuáles son las diferentes alternativas que podemos encontrar.

Por último, cabe decir que en la empresa existen ciertos proyectos en los que ya se implementa la metodología DevOps, pero que tienen características que difieren con las de este proyecto como por ejemplo el tipo de herramienta de control de versiones. De igual manera, se tendrán en cuenta a la hora de poder adaptarlas a nuestras necesidades.

## 2 ESTADO DEL ARTE

DevOps se comprende de una serie de herramientas que ayudan a aportar valor a nuestro producto. En este apartado se hará un estudio de cuáles son las tecnologías más populares para implementar DevOps.

Para la administración del código necesitaremos lo que se denomina SCM (*Source Code Management*), lo cual permite llevar una gestión de los cambios del código fuente, así como el control de versiones de este. Entre los más conocidos y utilizados tenemos Subversion y Git. La característica principal del primero es su simplicidad ya que tiene todo lo que un sistema de control de versiones debe tener, por el contrario, al ser un sistema centralizado requiere de conexión a internet para poder trabajar. Por otro lado, Git puede parecer algo más complejo debido a sus múltiples comandos y opciones, pero esto nos permite una mayor flexibilidad con nuestro proyecto, además no requiere de conexión a internet ya que cada usuario posee una copia del repositorio en local.

Como herramienta de integración continua tenemos Jenkins, el cual es un servidor que ofrece un entorno visual sencillo de usar. Se encarga de la construcción, test, entrega y despliegue de nuestro proyecto, además cuenta con un amplio catálogo de plugins que nos permitirán funcionalidades adicionales. Como alternativa tenemos Bamboo de Atlassian.

Cuando tenemos entornos muy complejos necesitamos hacer pruebas con entornos que sean muy similares o que tengan las mismas características donde el software va a ser instalado. Para ello existe la virtualización, donde los testers podrán hacer tantas réplicas sean necesarias del entorno para lanzar pruebas. Docker y Vagrant son las herramientas más conocidas.

Por último, hoy en día cada vez son más las empresas que contratan los servicios de *plataformas cloud*, que además de ahorrar costos en recursos también nos pueden proveer de aplicaciones que nos ayudan a llevar una completa gestión de nuestra infraestructura. Entre las más conocidas tenemos: Amazon Web Services, Microsoft Azure, IBM Cloud o Red Hat Open Shift.

## 3 OBJETIVOS

El objetivo principal del proyecto se centra en implementar y configurar una serie de herramientas que permitan aplicar la metodología DevOps para el desarrollo de una *Aplicación Web* en lenguaje Java.

Es muy común que durante el ciclo de vida de un proyecto surjan problemas e inconvenientes, que dependan directa o indirectamente de nosotros, o que simplemente no se puedan predecir al inicio, por esta razón y para que el proyecto tenga una base y cumpla unos mínimos, los objetivos se categorizan en tres clases: prioritarios, generales y opcionales.

### 3.1 Prioritarios

1. El diseño de la arquitectura ha de ser de fácil mantenimiento.
2. El diseño de la arquitectura ha de ser fácilmente ampliable.
3. Tener un sistema de control de versiones integrado en el sistema.
4. Configuración y administración del servidor Jenkins como sistema de automatización.
  - 4.1. Chequeo continuo de modificaciones del repositorio principal.
  - 4.2. Automatización de la compilación del código fuente.
  - 4.3. Automatización del test del software.
  - 4.4. Automatización de la entrega y despliegue del software.

### 3.2 Generales

- 4.5. Monitoreo de los pasos en la construcción del software.
- 4.6. Notificación a los desarrolladores de los posibles errores.
5. Configuración de los servidores de aplicaciones apta para el despliegue.

### 3.3 Opcionales

6. Configuración de servidor *proxy* que implemente balanceo *Round Robin* de los servidores de aplicaciones

Como se puede observar, a excepción de los objetivos de diseño y de la integración de un sistema de control de versiones, el núcleo de este proyecto está en el servidor Jenkins ya que es el sistema que se encargará de la automatización de los procesos, en los que la mayoría constan como objetivos prioritarios.

## 4 METODOLOGÍA

Como este proyecto está orientado a promover el desarrollo ágil, a la hora de decidir una metodología de trabajo también se tuvo en cuenta trabajar con metodologías ágiles, ya que permiten gestionar nuestro proyecto de forma flexible y autónoma reduciendo costes e incrementando nuestra productividad. Dentro de las más utilizadas tenemos la Programación Extrema, Scrum y Kanban. Se consideró que para este proyecto la mejor opción sería la de Kanban, ya que quizás entre las tres sea la más sencilla de aplicar en un proyecto el cual consta de un trabajo individual que, aunque también está orientado al trabajo en equipo, no requiere de roles u otras características complejas que en este caso no son necesarias.

El método Kanban junto con la ayuda de la aplicación Trello ha permitido tener una visión general del flujo de trabajo y el estado en que se encontraban las tareas en cada momento (Fig. 2).

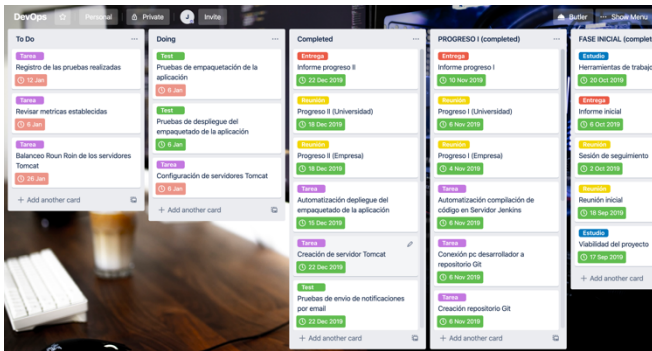


Fig. 2. Tablero de tareas de Trello

Este control ha permitido que se vayan cumpliendo los plazos estimados y llevar una carga equilibrada de trabajo evitando cuellos de botella. Además, después de evaluar los problemas surgidos en cada iteración se ha intentado incorporar mejoras, en la mayor medida de lo posible, para tener una mejor visión general y eficiencia en el proyecto.

### 4.1 Planificación

Para llevar un control a medida en el avance y poder conseguir los objetivos de este proyecto se elaboró una planificación. Esta planificación se dividió en cinco fases con un conjunto de actividades a realizar en cada una:

- **Fase inicial.** Se definieron las necesidades y requisitos, se marcaron los objetivos, se elaboró una planificación con unas tareas a realizar y se organizó una reunión con los tutores para presentar el proyecto.
- **Diseño.** Se especificaron las herramientas a utilizar para implementar las funcionalidades del sistema y se elaboró un diagrama general de la infraestructura del sistema.
- **Desarrollo.** Se llevaron a cabo las implementaciones y configuraciones de cada uno de los componentes y funcionalidades del sistema, así como la integración de todos los componentes en la

infraestructura.

- **Pruebas.** Se realizaron las pruebas que validarán los requisitos, se analizaron los resultados y variaciones respecto a las estimaciones esperadas.
- **Cierre.** Se aseguró que todas las actividades del proyecto hayan quedado correctamente finalizadas, se realizó una evaluación general de las tareas implementadas.

En el apéndice (A1) se podrá encontrar el diagrama de Gantt con las fases y tareas a lo largo del ciclo de vida de este proyecto.

## 5 REQUISITOS

A continuación, se enumerarán los requisitos funcionales y no funcionales que determinarán el comportamiento de nuestro sistema. A la hora de recoger los requisitos se ha tenido en cuenta el “qué” y el “cómo” hará el sistema en general, es decir, el conjunto de herramientas que forman nuestro sistema.

### 5.1 Funcionales

La siguiente lista recoge los requisitos funcionales:

- **RF1.** El sistema ha de permitir trabajar en equipo.
- **RF2.** El sistema ha de revisar el repositorio central de manera continua.
- **RF3.** El sistema ha de automatizar la compilación del código fuente.
- **RF4.** El sistema ha de automatizar los tests del código fuente.
- **RF5.** El sistema ha de automatizar la entrega (empaquetado) del código fuente.
- **RF6.** El sistema ha de automatizar el despliegue del código fuente a los servidores de aplicaciones.
- **RF7.** El sistema ha de notificar a los desarrolladores del resultado de la ejecución de la cadena de procesos por correo electrónico.
- **RF8.** Los contenedores han de poder establecer conexión SSH entre sí.
- **RF9.** El sistema ha de tener un servidor proxy que implemente balanceo *Round Robin* de los servidores de aplicaciones.

### 5.2 No funcionales

La siguiente lista recoge los requisitos no funcionales:

- **RNF1.** El proyecto en desarrollo ha de ser del tipo *Dynamic Web Project* del programa Eclipse.
- **RNF2.** La máquina que almacenará los repositorios centrales tendrá que ser de distribución Linux.
- **RNF3.** El sistema de control de versiones ha de ser Git.
- **RNF4.** El sistema que automatizará la cadena de procesos ha de ser un servidor Jenkins.
- **RNF5.** La ejecución de la cadena de procesos del servidor Jenkins no ha de ser superior a 5 minutos.

## 6 DISEÑO

En la fase de diseño, una vez estudiado el mercado y evaluado las diferentes alternativas para determinar cuáles serían las herramientas por usar, implementar y configurar, se procedió a diseñar el aspecto que tendría la arquitectura del sistema (Fig. 3).

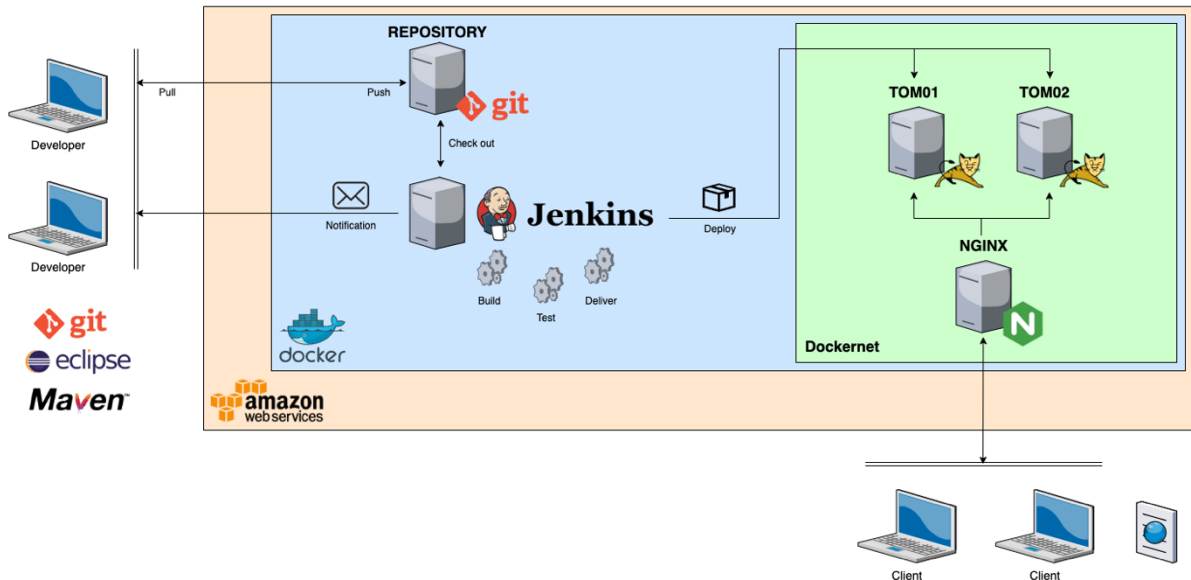


Fig. 3. Arquitectura del sistema

Como se puede observar, la arquitectura general del sistema está formada por cinco contenedores Docker. En primer lugar, tenemos un contenedor con un servidor Git que cumple la función de administrador del código fuente, en el cual se almacenarán los repositorios centrales del proyecto en desarrollo. En segundo lugar, tenemos el contenedor con un servidor Jenkins, núcleo del sistema, donde se automatizará la ejecución de una cadena de procesos. En tercer lugar, tenemos dos contenedores con servidores aplicaciones web TomEE (*Tomcat Enterprise Edition*) donde se desplegarán los paquetes de código con la aplicación web. Y, por último, tenemos un contenedor con un servidor Nginx que hará el balanceo de los dos servidores TomEE. Estos tres últimos contenedores pertenecerán a una red interna Docker.

El diagrama de la arquitectura fue elaborado con la aplicación web Draw.io, la cual es una herramienta gratuita online para construir todo tipo de diagramas. Ha resultado bastante útil gracias a que dispone de una gran variedad de iconos que ayudan a representar muy bien las aplicaciones que integran el sistema.

En los siguientes subapartados se expondrán las razones de haber utilizado Amazon Web Services y Docker, y, también analizaremos las ventajas y desventajas de cada una de ellas.

### 6.1 AWS

Utilizar Amazon Web Services en este proyecto fue una idea que se fijó en el momento en que la empresa, Everis, propuso el proyecto. Una de las motivaciones que se tuvo en cuenta en este proyecto fue tener la posibilidad de dedicar horas tanto en horario de oficina como fuera. El hecho de tener el proyecto en la nube posibilita acceder a él

desde cualquier lugar con acceso a internet. Además, la duración de este proyecto entraba dentro del periodo de prueba que ofrece AWS para usar sus servicios gratuitamente, evidentemente ofrece AWS para usar sus servicios gratuitamente, evidentemente con limitaciones de recursos. Por último, esta opción también podría ser valorada por la empresa cuando éste u otros proyectos se implanten.

### 6.2 Docker

Teniendo en cuenta que en este proyecto se iba a tratar con diferentes entornos, se tomó la decisión de utilizar Docker. Esta tecnología permite escoger una gran variedad de imágenes ya creadas en Dockerhub (<https://hub.docker.com>) con las características que más se ajustaran a las necesidades del proyecto. Alternativamente, permite escoger estas imágenes y complementarlas con acciones adicionales en un documento Dockerfile para crear propias imágenes.

A la hora de realizar pruebas, Docker permite hacer tantas réplicas como necesitemos de manera rápida y eficiente gracias a que el despliegue de las aplicaciones es automático dentro de los contenedores, proporcionando una capa adicional de abstracción del sistema operativo.

Para poder gestionar y monitorear todos los contenedores, imágenes, volúmenes, redes, entre otros, se levantó uno con la aplicación Portainer la cual presenta una interfaz bastante amigable. A continuación, en la Fig. 4, se muestra una captura de la vista con la lista de los contenedores:



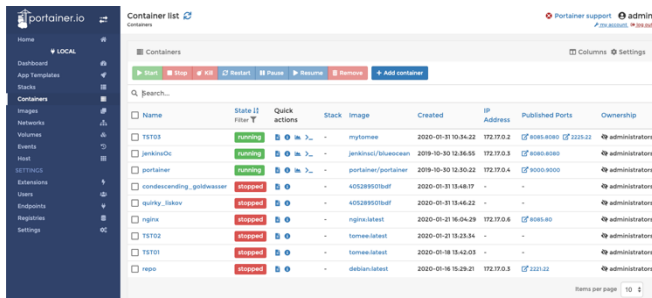


Fig. 4. Portainer, administrador de contenedores

## 7 IMPLEMENTACIÓN Y CONFIGURACIÓN

Esta sección explica los diferentes aspectos técnicos de la fase de desarrollo del proyecto. Se ha seguido el orden cronológico en la que se ha ido implementando y configurando cada una de las aplicaciones que complementan el sistema.

### 7.1 Git

#### 7.1.1 Preparación del repositorio central

Para almacenar el código fuente se configuró un servidor Git sobre un Docker con una imagen de un sistema de distribución Linux, en este caso, Debian. Primero, se creó un usuario con el cual los desarrolladores tendrán acceso al repositorio central mediante el protocolo SSH, por lo tanto, se habilitó dicho servicio y se abrió el puerto 22. Posteriormente, se creó la carpeta `.ssh` la cual almacenará un archivo `authorized_keys` con las claves públicas de los desarrolladores con acceso.

Tras esto, vino el paso principal de este apartado. Se preparó un repositorio vacío con el comando `git init` con la opción `--bare` para inicializar un repositorio sin carpeta de trabajo, es decir, un repositorio central el cual almacenará el historial de versiones del repositorio y que se podrá compartir:

```
$ mkdir my-appweb.git
$ cd my-appweb.git/
$ git init --bare
```

#### 7.1.2 Configuración de Eclipse

Una vez creado el repositorio central, desde Eclipse se creó el tipo de proyecto que se va a desarrollar, *Dynamic Web Project*. Hay que marcar la opción *Add Project to an EAR*, esta opción hará que posteriormente el proyecto pueda empaquetar todos sus módulos en un solo archivo que será desplegado en los servidores de aplicaciones TomEE, es una manera estándar de empaquetar aplicaciones. A partir de ahora, visualmente se tendrá dos proyectos en Eclipse, uno WEB y otro EAR, por lo que se necesitará, además de otro repositorio, repetir los pasos que vienen a continuación.

Se convierte el proyecto en desarrollo a uno basado en Maven, haciendo clic derecho sobre el proyecto y seleccionando la opción *Configure > Convert to Maven Project*. Se abrirá una ventana donde se tendrá que especificar el nombre de grupo (el mismo en ambos proyectos), artefacto, versión y tipo de empaquetado como se muestra en la Fig. 5:

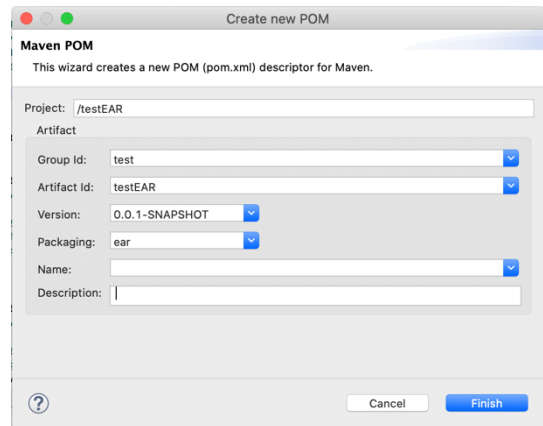


Fig. 5. Eclipse, convertir a proyecto Maven

Al darle a *Finish* se generará automáticamente un archivo POM (*Project Object Model*), el cual es un fichero XML, que contendrá y se deberá añadir la información principal acerca del proyecto. A continuación, en la Fig. 6 se muestra el archivo `pom.xml` correspondiente al proyecto EAR.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>my-webapp</groupId>
5   <artifactId>my-webapp-ear</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <packaging>ear</packaging>
8   <properties>
9     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
10  </properties>
11  <dependencies>
12    <dependency>
13      <groupId>my-webapp</groupId>
14      <artifactId>my-webapp</artifactId>
15      <version>0.0.1-SNAPSHOT</version>
16      <type>war</type>
17    </dependency>
18  </dependencies>
19  <build>
20    <plugins>
21      <plugin>
22        <artifactId>maven-ear-plugin</artifactId>
23        <version>2.10</version>
24        <configuration>
25          <modules>
26            <webModule>
27              <groupId>my-webapp</groupId>
28              <artifactId>my-webapp</artifactId>
29              <uri>my-webapp-0.0.1-SNAPSHOT.war</uri>
30              <contextRoot>/my-webapp</contextRoot>
31            </webModule>
32          </modules>
33        </configuration>
34      </plugin>
35    </plugins>
36  </build>
37 </project>
```

Fig. 6. Archivo `pom.xml` del proyecto EAR

Como se puede observar, el archivo muestra como está estructurado el proyecto para que a la hora de construirlo lo haga de manera coherente a las versiones indicadas y a las dependencias que éste está relacionado, en este caso, a nuestro proyecto WEB.

Se tendrá que relacionar este proyecto al repositorio central. Se inicializa un repositorio git dentro de la carpeta del proyecto, se añaden todos los archivos, se hace un *commit*, y se relaciona el repositorio central con un comando que presentará un aspecto como el siguiente:

```
$ git remote add origin ssh://<usuario>@<dominio.com>:22/~ /my-webapp.git
```

Finalmente, se hace un *push* al repositorio central y nuestro proyecto estará listo para compartir.

### 7.1.3 Clonar proyecto

Una vez el proyecto WEB ya está en el servidor Git, cualquier desarrollador que esté autorizado a acceder podrá clonarlo en local y empezar a colaborar en el proyecto. Desde Eclipse se puede importar el proyecto directamente desde el menú: *File > Import > Projects From Git > Existing Local Repository > Clone URI*, y copiando la dirección (Fig. 7)

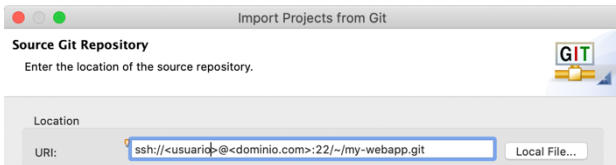


Fig. 7. Eclipse. Importar proyecto Git

## 7.2 Jenkins

En esta sección describe como configurar el servidor Jenkins para poder ejecutar la cadena de procesos que necesitamos automatizar. Jenkins admite herramientas de control de versiones como Subversion, Git, CVS, entre otros; y ejecutar proyectos basados en Apache Ant y Apache Maven; también dispone de gran variedad de *plugins* que ayudarán a cambiar el comportamiento de Jenkins o añadir nuevas funcionalidades.

### 7.2.1 Creación del Pipeline

Antes que nada, se tiene que autorizar el acceso entre del servidor Jenkins al servidor Git. Se genera claves RSA, la clave pública se añade al archivo *authorized\_keys* del usuario *git*, y la clave privada se la añade a las credenciales de Jenkins en el menú: *Jenkins > Credentials > System > Global credentials > add Credentials*. Creamos un nuevo Pipeline en el que organizaremos nuestras tareas de manera libre (Fig. 8): *Jenkins > Nueva Tarea > Pipeline*.

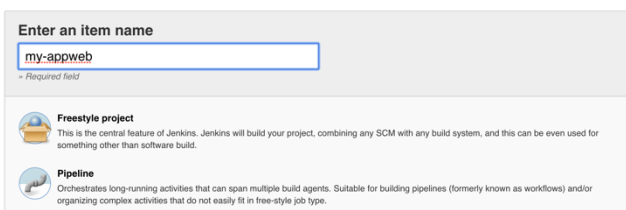


Fig. 8. Crear Pipeline

Al estar usando Docker debemos de coger como referencia las IP dentro de Docker. Si estamos usando la

aplicación Portainer, mencionada anteriormente, podremos ver que IP corresponde a cada contenedor. En caso contrario, podremos utilizar el comando:

```
$ docker inspect <nombre_contenedor>
```

Se mostrará toda la información relacionada al contenedor, en este caso necesitamos la de el servidor Git y el campo "IPAddress". Con toda esta información ya podremos conectarnos al repositorio como muestra la Fig. 9.

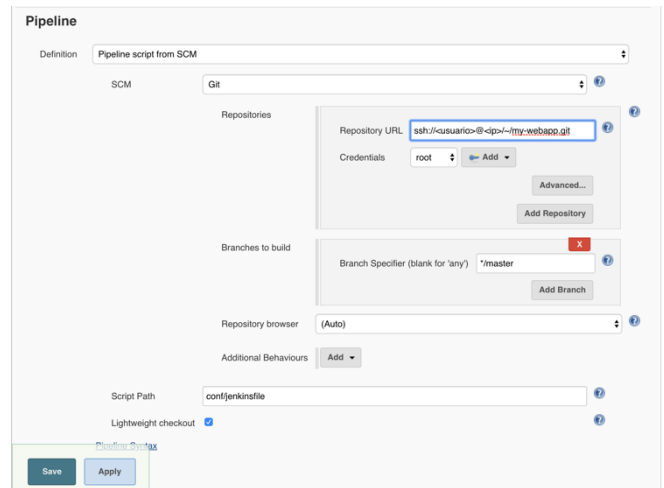


Fig. 9. Conexión servidor Jenkins a servidor Git

### 7.2.2 Revisión continua del repositorio

Uno de los objetivos era que se hiciera una revisión continua de cambios sobre el repositorio. Para ello utilizaremos el sistema cron, que es un administrador regular de procesos en segundo plano. El formato de configuración es el siguiente (Fig. 10):

```
----- minuto (0-59)
----- hora (0-23)
----- día del mes (1-31)
----- mes (1-12) o jan,feb,mar,apr,may,jun,jul... (meses en inglés)
----- día de la semana (0-6) (domingo=0 ó 7) o sun,mon,tue,wed,thu,fri,sat
* * * * * comando a ejecutar
```

Fig. 10. Formato cron (Wikipedia)

Para ello marcamos la opción *Consultar repositorio (SCM)* y rellenamos el campo *Programador*, si queremos que se revise el repositorio cada dos minutos sería:

```
* / 2 * * * *
```

Cuando existan cambios en el repositorio actualizará comenzará a ejecutarse la cadena de procesos que implementaremos en los siguientes apartados.

### 7.2.3 Construcción, test y entrega

El *pipeline* se declara en un fichero, si observamos la Fig. 9 estará en la carpeta *conf*, se almacena y se versiona junto con el código en un fichero comúnmente llamado *Jenkinsfile*. Este fichero define las fases (*stages*) de la que está compuesto nuestro flujo.

El proyecto en desarrollo al estar basado en Maven, compilaremos el código sobre una imagen Maven de Docker, que hará de agente para poder correr el código. Esta imagen solo estará activa durante la ejecución del *pipeline*. A continuación, se muestra (Fig. 11) las fases de compilación, test y entrega del proyecto WEB.

```

1 pipeline {
2   agent {
3     docker {
4       image 'maven:3-alpine'
5       args '-v /home/ec2-user/.m2:/root/.m2'
6     }
7   }
8   stages {
9     stage('Build') {
10      steps {
11        sh 'mvn -B -DskipTests clean package'
12      }
13    }
14    stage('Test') {
15      steps {
16        sh 'mvn test'
17      }
18    }
19    stage('Deliver') {
20      steps {
21        sh 'mvn war:war install:install help:evaluate -Dexpression=project.name'
22      }
23    }
24  }
25 }
26 }

```

Fig. 11. Stages de compilación, test y entrega

En la fase de entrega, en la que se empaqueta el proyecto, deberá ser modificada la línea 22. En el proyecto EAR se cambiará el tipo de empaquetado:

```
sh 'mvn ear:ear install:install help:evaluate -Dexpression=project.name'
```

El paquete quedará almacenado en la carpeta *target* del proyecto en el *workspace* del servidor Jenkins y en el directorio */root/.m2* del usuario de la máquina donde tenemos instalado el Docker.

## 7.2.4 Despliegue

La fase de despliegue la implementaremos únicamente en el *Jenkinsfile* del proyecto EAR, ya que éste ya contiene el paquete war que corresponde al módulo web. Para esta fase se supondrá que ya tenemos levantados los servidores TomEE donde desplegaremos nuestra aplicación web (en el punto 7.3 se explicará en detalle la configuración).

La transmisión del archivo se hará usando el comando *scp* por lo que nuevamente necesitamos las credenciales para poder conectarnos, esta vez a los servidores TomEE. Es necesario instalar el *SSH Agent Plugin* y especificar la credencial indicando su ID como muestra la Fig. 12 en la línea 30.

```

27 stage('Deploy') {
28   steps {
29     sshagent(credentials: ['c9aee383-ec26-4575-8ca1-e41cc29279d4']) {
30       sh 'ssh -o StrictHostKeyChecking=no 172.17.0.4 uname -a'
31       sh 'ssh 172.17.0.4 "rm -rf /usr/local/tomee/webapps/my-webapp"'
32       sh 'scp -r /var/jenkins_home/workspace/my-webapp-ear/target/my-webapp-ear.ear 172.17.0.4:/usr/local/tomee/webapps/'
33     }
34   }
35 }

```

Fig. 12. Stage de despliegue

El servidor TomEE almacena las aplicaciones en el directorio */usr/local/tomee/webapps*. Borrarnos el contenido del directorio, en caso de que existan versiones anteriores (línea 31), y copiamos el paquete en cuestión (línea 32).

## 7.2.5 Notificación

Para enviar un correo con el resultado obtenido de la ejecución de la cadena de procesos, independiente si ha sido exitoso o fallido, utilizaremos el *Email Extension Plugin* y que requerirá un servidor SMTP (*Simple Mail Transfer Protocol*) y una cuenta Gmail en el menú: *Jenkins > Configurar Sistema > Extended E-mail Notification*, como se muestra en la Fig. 13:

Fig. 13. Configuración de *Extended E-mail Notification*

Y en el *Jenkinsfile* añadir lo siguiente (Fig. 14):

```

38 post {
39   success {
40     email {
41       to: 'JuanErasmoplua@campus.uab.cat',
42       subject: 'Build: ${env.JOB_NAME} - Success',
43       attachLog: true,
44       body: "Job Success - \"${env.JOB_NAME}\" build: ${env.BUILD_NUMBER}\n\n"
45     }
46   }
47   failure {
48     email {
49       to: 'JuanErasmoplua@campus.uab.cat',
50       subject: 'Build: ${env.JOB_NAME} - Failed',
51       attachLog: true,
52       body: "Job Failed - \"${env.JOB_NAME}\" build: ${env.BUILD_NUMBER}\n\n"
53     }
54   }
55 }

```

Fig. 14. Stage de enviar notificación por correo

El archivo *build.log* también lo adjuntamos al correo.

## 7.3 TomEE

El despliegue de la aplicación web se hará sobre los servidores web TomEE. En esta sección se aprovechará para explicar como crear una imagen propia a partir de un *Dockerfile*. Un *Dockerfile* es un archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen. Por ejemplo, nuestro *Dockerfile* para crear una imagen TomEE sería:

```
FROM tomee:8-jre-7.1.0-webprofile
```

```

RUN set -x \
  && apt-get update \
  && apt-get upgrade \
  && apt-get install ssh -y \
  && service ssh start \
  && mkdir /root/.ssh \
  && set +x

```

```
COPY webapps /usr/local/tomee/webapps
```

```
COPY .ssh /root/.ssh
```

```
EXPOSE 22
```

A partir de (*FROM*) una imagen TomEE ya creada, podemos ejecutar (*RUN*) una serie de comandos, copiar (*COPY*) archivos y carpetas; y, por último, abrir (*EXPOSE*) puertos que necesitemos en la nueva imagen. Ahora se podrá crear la nueva imagen a partir del *Dockerfile* dándole un nombre y una etiqueta (*tag*) que se quiera con el siguiente comando:

```
$ sudo docker build -t <nombre>:<tag> .
```

Posteriormente, se podrá crear tantas réplicas como se requiera con el comando *docker run* usando esta nueva imagen.

## 7.4 Nginx

Se crea un servidor balanceador Nginx que balancee los dos servidores de aplicaciones TomEE utilizando el método *Round Robin*. Será necesario crear una red Docker interna a la cual se conectarán el servidor Nginx y los servidores TomEE.

```
$ docker network create -d bridge mynet
$ docker network connect mynet nginx
$ docker network connect mynet tom01
$ docker network connect mynet tom02
```

Se creará un fichero *load-balancer.conf* que tendrá la configuración de balanceo del servidor Nginx.

```
upstream mywebapp {
    server 172.19.0.3:8080;
    server 172.19.0.2:8080;
}

server {
    listen 80;
    root /my-webapp;
    index index.html index.jsp;

    location / {
        proxy_pass http://mywebapp;
    }
}
```

Para que esta configuración se aplique se deberá eliminar la que esta por defecto en el mismo directorio donde se deberá dejar la nueva (*/etc/nginx/conf.d/*).

## 8 PRUEBAS

Al comienzo de las pruebas se decidió que era importante realizar manualmente la construcción del código desde eclipse y el despliegue en un servidor TomEE en local. De esta manera, resultó bastante útil ya se que pudo observar el comportamiento que debía tener el sistema y tenerlo como referencia durante esta fase.

En la configuración de los entornos fue importante tener clara la relación entre cada capa de la infraestructura. La IP y el DNS público de la instancia de AWS variaba cada

vez que se hacía un reinicio de ella al igual que las IP docker de los contenedores, que se asignaban en el orden que se levantaban y había que tener identificada la asignación y relación de los puertos de cada componente.

Las pruebas se realizaron prácticamente en paralelo a la implementación y configuración de cada componente y, en primera instancia, de manera independiente al resto de la infraestructura. Por ejemplo, se intentó acceder a los repositorios centrales desde máquinas que sí y que no estaban autorizadas en el archivo *authorized\_keys* del usuario del servidor Git. Los contenedores Docker de los servidores TomEE se levantaron con paquetes ya incluidos dentro del directorio de despliegue y también se desplegaron manualmente paquetes con un empaquetado existoso y fallido.

La misma metodología se siguió para los procesos que se ejecutan de manera automática en el servidor Jenkins. Se creó un *pipeline* diferente para cada tarea. Se comprobó que la revisión del repositorio fuera continua y que cumpliera la frecuencia establecida subiendo código al repositorio de manera irregular. Al tener dos proyectos dependientes el uno del otro, se comprobó que la estructura del fichero *pom.xml* con correspondiera con la construcción y que el empaquetado WEB y EAR se pudieran desplegar en un servidor, el primero en uno Tomcat, y el segundo en uno TomEE local. El envío de notificación por correo se validó que el mensaje correspondiera al resultado de la ejecución del *pipeline* obtenido forzando errores en sus fases.

Cada componente que pasaba las pruebas individuales de manera satisfactoria se lo integraba a la infraestructura para realizar pruebas de integración. Se comprobó el acceso entre las aplicaciones por protocolo SSH y se juntaron todas las fases en un solo *pipeline* a ejecutar el servidor Jenkins. A continuación, se muestra el *pipeline* completo del proyecto WEB (Fig. 15) y del proyecto EAR (Fig.16).

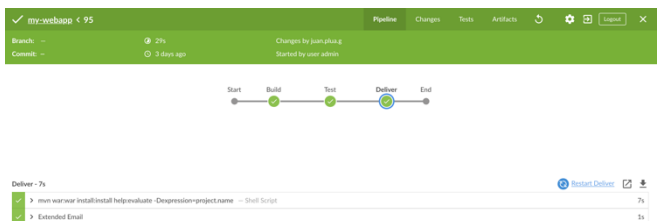


Fig. 15. Prueba satisfactoria *pipeline* del proyecto WEB

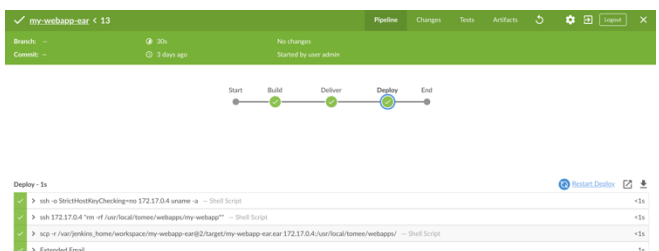


Fig. 16. Prueba satisfactoria *pipeline* del proyecto EAR



Jenkins cuenta con un monitoreo de cada ejecución que ha sido determinante en la mayoría de las pruebas para identificar los errores producidos, por este motivo se adjuntó el fichero de *log* a la notificación por correo a los desarrolladores. En el apéndice (A2) se podrá encontrar capturas de la recepción de la notificación por correo.

## 9 RESULTADOS

Los resultados obtenidos se basan en las diferentes validaciones que se han realizado en el apartado de pruebas donde se ha comprobado cada una de las funcionalidades que se han implementado.

Los objetivos de diseño, los cuáles pedían que fuera de fácil mantenimiento y ampliación se logró gracias al uso de *Dockerfiles* para crear imágenes propias y a las pruebas realizadas donde se validaba cada aplicación de forma independiente e integrada al sistema.

Uno de los objetivos prioritarios era tener un sistema de control de versiones, este se consiguió integrando un servidor *Git* en el sistema, el cual permite almacenar código en los repositorios centrales de proyectos en desarrollo, y de un sistema de claves que sólo permite acceder desde máquinas autorizadas.

El objetivo prioritario, del cual derivaban varios subobjetivos, era la automatización de procesos en el servidor Jenkins, éste hace que el sistema esté pendiente de los cambios que se producen en el repositorio central haciendo revisiones continuas con una frecuencia de 2 minutos. Cada vez que se produce un cambio inicia la ejecución del *pipeline*. Compila el código de proyectos Java basados en Apache Maven para encontrar errores, realiza los tests para revisar las métricas establecidas, guarda el paquete resultante y lo despliega en los servidores de aplicaciones TomEE para que el cliente pueda utilizar la nueva entrega o los desarrolladores puedan realizar las pruebas pertinentes. Además, se envía una notificación con el resultado de la ejecución y el fichero *build.log* obtenido por correo electrónico a los desarrolladores del proyecto. El tiempo que se dedica a la construcción varía dependiendo del tamaño del proyecto, pero considerando un proyecto pequeño los resultados obtenidos de la ejecución del *pipeline* al completo se consiguieron tiempos que no superaban el minuto de duración.

Y, como objetivos generales y opcionales, el sistema integra dos servidores de aplicaciones TomEE (*Tomcat Enterprise Edition*) que funciona como contenedor de *servlets* que es donde se despliega la aplicación web. Además de un servidor *proxy* Nginx que implementa un balanceo *Round Robin* de los servidores TomEE.

## 10 CONCLUSIONES

Los objetivos marcados se han alcanzado, incluso los que eran opcionales dando como resultado un sistema que permite complementar el desarrollo con metodologías

ágiles en equipos de software ayudando en un incremento de la frecuencia de entrega de paquetes de código de manera ordenada y coherente, mejorando la calidad de código, disminuyendo los costes e incrementando la productividad.

La planificación y la metodología de trabajo Kanban han permitido hacer un seguimiento y una evaluación constante, y aplicar mejoras durante el ciclo de vida del proyecto.

Por último, trabajar con *Amazon Web Services* y *Docker* fue una decisión acertada ya que permitieron agilidad a la hora de experimentar con las funcionalidades aplicaciones, realizar pruebas y guardar imágenes ya configuradas para poder replicarlas.

### 10.1 Líneas abiertas

El proyecto se puede presentar como una base genérica que puede utilizarse para cualquier proyecto de software en desarrollo que esté basado en Apache Ant o Apache Maven. En este caso se trabajó con un tipo de proyecto de eclipse en concreto porque era un requisito impuesto por el cliente.

Desde el servidor Jenkins se pueden añadir más procesos a la cadena de ejecución que ayuden a mejorar y complementar las necesidades del proyecto en desarrollo.

Por otra parte, gracias al uso de *Dockerfiles* se puede ampliar, por ejemplo, la red de servidores de aplicaciones y *proxy* que en este proyecto no fue posible debido a que los recursos (gratuitos) eran limitados.

## AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi familia que ha estado apoyándome durante estos años de carrera; a Everis, empresa colaboradora, por proponerme el proyecto y la confianza transmitida; y, también, a mis tutores Yolanda Benítez y Daniel Cufí que me han aconsejado a lo largo del proyecto para que fuera bien encaminado y que se cumplieran los objetivos marcados.

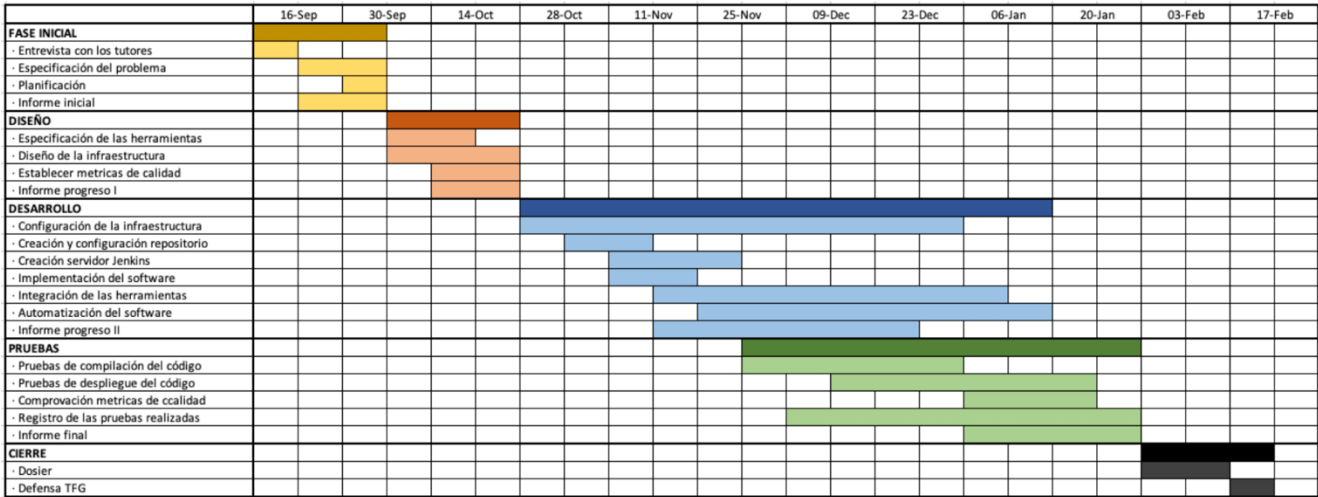
## BIBLIOGRAFIA

- [1] Wikipedia. *DevOps*. [fecha de consulta: 20 de septiembre de 2019]. Disponible en: <https://es.wikipedia.org/wiki/DevOps>
- [2] Profile.es, María José Martín. *¿Qué es DevOps?* [fecha de consulta: 20 de septiembre de 2019]. Disponible en: <https://profile.es/blog/que-es-devops-harder-better-faster-stronger/>
- [3] José Juan Mora Pérez. *DevOps y el camino de baldosas amarillas*, 2015.
- [4] Iebschool, Vanessa Rosselló Villán. *Las metodologías más utilizadas y sus ventajas dentro de la empresa*. [fecha de consulta: 27 de septiembre de 2019]. Disponible en: <https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/>
- [5] Viewnext. *¿Cuáles son las herramientas DevOps?* [fecha de consulta: 29 de septiembre de 2019]. Disponible en: <https://www.viewnext.com/cuales-son-las-herramientas-devops/>
- [6] Kanbantool. *¿Por qué utilizar la metodología Kanban?* [fecha de consulta: 27 de septiembre de 2019]. Disponible en: <https://kanbantool.com/es/metodologia-kanban>

- [7] Wikipedia. *Kanban*. [fecha de consulta: 27 de septiembre de 2019]. Disponible en: [https://es.wikipedia.org/wiki/Kanban\\_\(desarrollo\)](https://es.wikipedia.org/wiki/Kanban_(desarrollo))
- [8] Campus Virtual (UAB). Gestión de Proyectos, curso 2018-19. *Gestión del tiempo*.
- [9] Jenkins.io. *Jenkins*. [fecha de consulta: 7 de octubre de 2019]. Disponible en: <https://jenkins.io/>
- [10] GitHub.com. *Official Jenkins Docker image*. [fecha de consulta: 9 de octubre de 2019]. Disponible en: <https://github.com/jenkinsci/docker/blob/master/README.md>
- [11] Jenkins.io. *Build a Java app with Maven*. [fecha de consulta: 9 de octubre de 2019]. Disponible en: <https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/>
- [12] Portainer.io. *Installing Portainer*. [fecha de consulta: 10 de octubre de 2019]. Disponible en: <https://www.portainer.io/installation/>
- [13] Git-scm.com. *Git en el Servidor*. [fecha de consulta: 4 de noviembre de 2019]. Disponible en: <https://git-scm.com/book/es/v2/Git-en-el-Servidor-Los-Protocolos>
- [14] Maven.apache.org. *Maven*. [fecha de consulta: 6 de noviembre de 2019]. Disponible en: <https://maven.apache.org/>
- [15] Maven.apache.org. *Maven*. [fecha de consulta: 18 de noviembre de 2019]. Disponible en: <https://maven.apache.org/>
- [16] Dockerhub. [fecha de consulta: 20 de noviembre de 2019]. Disponible en: <https://hub.docker.com/>
- [17] Medium, Gustavo Apolinario. *Jenkins sending email on post build*. [fecha de consulta: 2 de diciembre de 2019]. Disponible en: <https://medium.com/@gustavo.guss/>
- [18] Github.com, teeks99. *Jenkins with e-mail*. [fecha de consulta: 2 de diciembre de 2019]. Disponible en: <https://gist.github.com/teeks99/d7c331b3f66a9fe6507cfdaaabb9229>
- [19] Confluence. *Maven properties guide*. [fecha de consulta: 7 de enero de 2020]. Disponible en: <https://cwiki.apache.org/confluence/display/MAVEN/Maven+Properties+Guide>
- [20] Jenkins.io. *Using Docker with Pipeline*. [fecha de consulta: 7 de enero de 2020]. Disponible en: <https://jenkins.io/doc/book/pipeline/docker/>
- [21] Printstacktrace, Szymon Stepniak. *Jenkins Pipeline Environment Variables - The Definitive Guide*. [fecha de consulta: 7 de enero de 2020]. Disponible en: <https://e.printstacktrace.blog/jenkins-pipeline-environment-variables-the-definitive-guide/>
- [22] Avajava, Deron Eriksson. *What are the phases of the maven default lifecycle?*. [fecha de consulta: 7 de enero de 2020]. Disponible en: <http://www.avajava.com/tutorials/lessons/what-are-the-phases-of-the-maven-default-lifecycle.html>
- [23] Nginx.org. *Using nginx as HTTP load balancer*. [fecha de consulta: 8 de enero de 2020]. Disponible en: [http://nginx.org/en/docs/http/load\\_balancing.html](http://nginx.org/en/docs/http/load_balancing.html)
- [24] ackoverflow.com. *Nginx configuration to pass site directly to tomcat webapp with context*. [fecha de consulta: 8 de enero de 2020]. Disponible en: <https://stackoverflow.com/questions/19866203/nginx-configuration-to-pass-site-directly-to-tomcat-webapp-with-context>
- [25] Maven.apache.org. *Using the package phase with the project package type as war / invocation of the war:war goal*. [fecha de consulta: 8 de enero de 2020]. Disponible en: <https://maven.apache.org/plugins/maven-war-plugin/usage.html>
- [26] Wikipedia. *Nginx*. [fecha de consulta: 16 de enero de 2020]. Disponible en: <https://es.wikipedia.org/wiki/Nginx>
- [27] Arquitecturajava, Cecilio Álvarez Caules. *Modulos de Java (III) Enterprise Archive (EAR)*. [fecha de consulta: 17 de enero de 2020]. Disponible en: <https://www.arquitecturajava.com/ear/>
- [28] Openwebinars. *Qué es DockerFile*. [fecha de consulta: 17 de enero de 2020]. Disponible en: <https://openwebinars.net/blog/ques-dockerfile/>


APÉNDICE

A1. DIAGRAMA DE GANTT



A2. Notificación por correo


Build: my-webapp-ear - Success



Jenkins <jenkins.docker.aws@gmail.com>

Mié 05/02/2020 4:04

Usted; juan.erasmo.plua.gutierrez.st@everis.com

 build.log

11 KB

Job Success - "my-webapp-ear" build: 13

Nice!

Love it!

Cute!

¿Las sugerencias anteriores son útiles?

Si

No