



Practica 7

Alberto Jiménez Serrano

January 2024

Índice

1. Introducción	2
2. Instalaciones	2
3. Configuración	4
4. Gestión de dependencias	7
5. Automatización de tareas	8
6. Entrega continua: Jenkins	11
6.1. Configuración	11
6.2. Flujo de trabajo	15
7. Conclusión	17

1. Introducción

Vamos a crear un flujo de trabajo automatizado para implementar y desplegar una aplicación web en un entorno de producción utilizando herramientas de entrega continua.

Para esta práctica partiremos de una página web ya creada por mí en la que hay distintos juegos clásicos.

Todo el proceso se podrá encontrar en el siguiente GitHub:

<https://github.com/Albertojserr/Practica7>

2. Instalaciones

Lo primero va a ser configurar nginx. Nginx es un servidor web de código abierto y un proxy para protocolos de correo electrónico. Según el sistema operativo que estés utilizando variará la forma de configurar.

En distribuciones basadas en Ubuntu, para instalar Nginx primero actualizaremos los paquetes usando apt, es una herramienta informática de gestión de paquetes desarrollada por Debian, y luego usando sudo apt install nginx podremos instalarlo.

```
(base) alberto@slimbook-ONE:~$ sudo apt update
[sudo] contraseña para alberto:
Lo siento, pruebe otra vez.
[sudo] contraseña para alberto:
Obj:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Des:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Des:3 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Des:4 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease [3.626 B]
Des:5 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Des:6 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1.065 kB]
Des:7 http://es.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [550 kB]
Des:8 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [831 kB]
]
Des:9 http://es.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1.280 kB]
Des:10 http://security.ubuntu.com/ubuntu jammy-security/universe i386 Packages [585 kB]
]
Des:11 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [158 kB]
Des:12 http://es.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [262 kB]
Des:13 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1.031 kB]
Des:14 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe i386 Packages [681 kB]
Des:15 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [230 kB]
Descargados 7.016 kB en 3s (2.193 kB/s)
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se pueden actualizar 70 paquetes. Ejecute «apt list --upgradable» para verlos.
(base) alberto@slimbook-ONE:~$ sudo apt install nginx
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
nginx ya está en su versión más reciente (1.18.0-6ubuntu14.4).
El paquete indicado a continuación se instaló de forma automática y ya no es necesario
libfltk1.1
```

Después de instalarlo, puedes iniciar el servicio de Nginx con el comando:
`sudo systemctl start nginx`
Y para ver el estado en el que se encuentra, si está encendido o apagado, podemos verlo con:

```
(base) alberto@slimbook-ONE:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset:...
```

Podemos indicar que se inicialice nginx al encender el ordenador usando "sudo systemctl enable nginx".

Ahora instalaremos Node.js que es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript.

Se puede instalar Node.js desde el sitio oficial o también en sistemas basados en Debian/Ubuntu, puedes usar los siguientes comandos:

```
sudo apt install nodejs npm
```

Además, a la vez, instalaremos npm, que es un gestor de paquetes, ya que lo necesitaremos más adelante para instalar las dependencias específicas de la aplicación.

Podemos ver sus versiones utilizando "node -v" "npm -v"

```
(base) alberto@slimbook-ONE:~$ node -v
v12.22.9
(base) alberto@slimbook-ONE:~$ npm -v
8.5.1
```

3. Configuración

Lo primero que haremos será `/etc/hosts` y añadiremos una dirección únicamente para esta práctica que se llamará `practica7`. Después, nos iremos al directorio `/etc/nginx` en el que se encuentra la configuración principal de Nginx. El archivo de configuración principal es `/etc/nginx/nginx.conf`.

Los archivos de configuración de sitios individuales suelen ubicarse en `/etc/nginx/sites-available/` y los activaremos creando enlaces simbólicos a la carpeta `/etc/nginx/sites-enabled/`.

En `/etc/nginx/sites-available/` crearemos un archivo `practica7` y lo editaremos con la función `nano`.

Pondremos lo siguiente:

```
GNU nano 6.2 /etc/nginx/sites-available/practica7
server {
    listen 80;
    server_name practica7;

    location / {
        root /var/www/practica7;
        index index.html;
    }
}
```

[9 líneas leídas]

^G Ayuda ^O Guardar ^W Buscar ^K Cortar ^T Ejecutar ^C Ubicación
^X Salir ^R Leer fich. ^_ Reemplazar ^U Pegar ^J Justificar ^_ Ir a línea

Ponemos que escuche en el puerto 80, el nombre del servidor que es practica7, lo hemos establecido anteriormente, luego la ruta en la que nos encontramos el index de la página que será la página inicial.

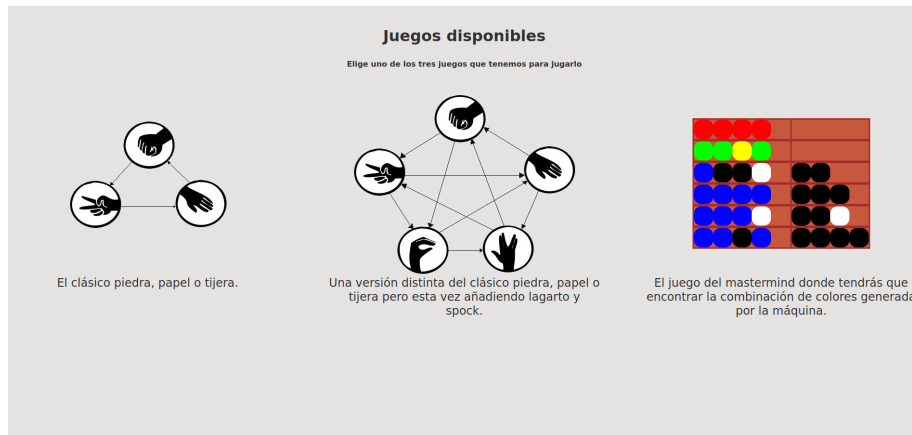
Ahora crearemos un enlace simbólico con el siguiente comando:

"sudo ln -s /etc/nginx/sites-available/practica7 /etc/nginx/sites-enabled/"

Para ver si lo hemos hecho correctamente y no hay fallos, como por ejemplo espacios en la ruta que no le permite acceder correctamente, podemos verlo de la siguiente manera:

```
(base) alberto@slimbook-ONE:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
(base) alberto@slimbook-ONE:~$ sudo systemctl restart nginx
```

Una vez visto que la configuración está bien reiniciamos nginx y ya podremos acceder a la página web.



Si se está usando un firewall, hay que asegurarse de permitir el tráfico en el puerto 80 para HTTP o el puerto 443 para HTTPS mediante "sudo ufw allow 80".

Para gestionar las variables de entorno en Node.js utilizaremos el módulo dotenv:

```
npm install dotenv
```

Después, crearemos un archivo .env en el directorio raíz del proyecto con las siguientes variables:

```
DB_URL=url_base_de_datos
```

```
AUTH_CREDENTIALS=usuario:contraseña
```

Estas variables de entorno las podremos luego utilizar en nuestro archivo JavaScript de la siguiente forma:

```
const dbUrl = process.env.DB_URL;
```

```
const authCredentials = process.env.AUTH_CREDENTIALS;
```

Al utilizar nuestra aplicación un servidor Node.js en lugar de servir archivos estáticos directamente, podemos configurar Nginx como un proxy inverso. Esto se hace agregando una configuración de proxy en la sección de location del archivo de configuración del sitio.

```
location / {
    proxy_pass http://practica:80;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
```

Después de hacer los cambios hay que reiniciar nginx como antes.

4. Gestión de dependencias

Ya tenemos instalado npm, ahora lo que haremos será iniciar un proyecto Node.js ejecutando en el directorio del proyecto el comando "npm init" lo que, mediante una serie de parámetros que nos pedirá, creará un archivo package.json. Este archivo almacenará información sobre tu proyecto, incluyendo las dependencias.

Vamos a instalar una serie de dependencias, además de dotenv que ya la instalamos anteriormente. Para ello usamos:

npm install dependencia

Instalaremos las dependencias express, un framework web minimalista para Node.js, jsonwebtoken, una implementación de JSON Web Tokens, y web que es un módulo de nodos diseñado para crear aplicaciones web.

```
(base) alberto@slimbook-ONE:~/Documentos/UAX/Curso3/Cuatri2/Ironhack/IronhackTrabajoFinal$ npm install jsonwebtoken
added 17 packages, and audited 19 packages in 4s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
(base) alberto@slimbook-ONE:~/Documentos/UAX/Curso3/Cuatri2/Ironhack/IronhackTrabajoFinal$ npm install express
added 60 packages, and audited 79 packages in 5s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
(base) alberto@slimbook-ONE:~/Documentos/UAX/Curso3/Cuatri2/Ironhack/IronhackTrabajoFinal$ npm install web
added 1 package, and audited 80 packages in 3s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```

package.json x
1 {
2   "name": "practica7",
3   "version": "1.0.0",
4   "description": "Este es el package json de la practica 7 de la asignatura de
5   ",
6   "main": "index.js",
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit 1"
9   },
10  "repository": {
11    "type": "git",
12    "url": "git+https://github.com/Albertojsserr/Practica7.git"
13  },
14  "author": "",
15  "license": "ISC",
16  "bugs": {
17    "url": "https://github.com/Albertojsserr/Practica7/issues"
18  },
19  "homepage": "https://github.com/Albertojsserr/Practica7#readme",
20  "dependencies": {
21    "dotenv": "^16.3.2",
22    "express": "^4.18.2",
23    "jsonwebtoken": "^9.0.2",
24    "web": "^0.0.2"
25  }

```

5. Automatización de tareas

Vamos a realizar un script de construcción que es aquel que suele realizar tareas como compilación de código fuente y generación de archivos estáticos. Utilizaremos Webpack como herramienta de construcción. Para ello instalaremos webpack:

`npm install webpack webpack-cli --save-dev`

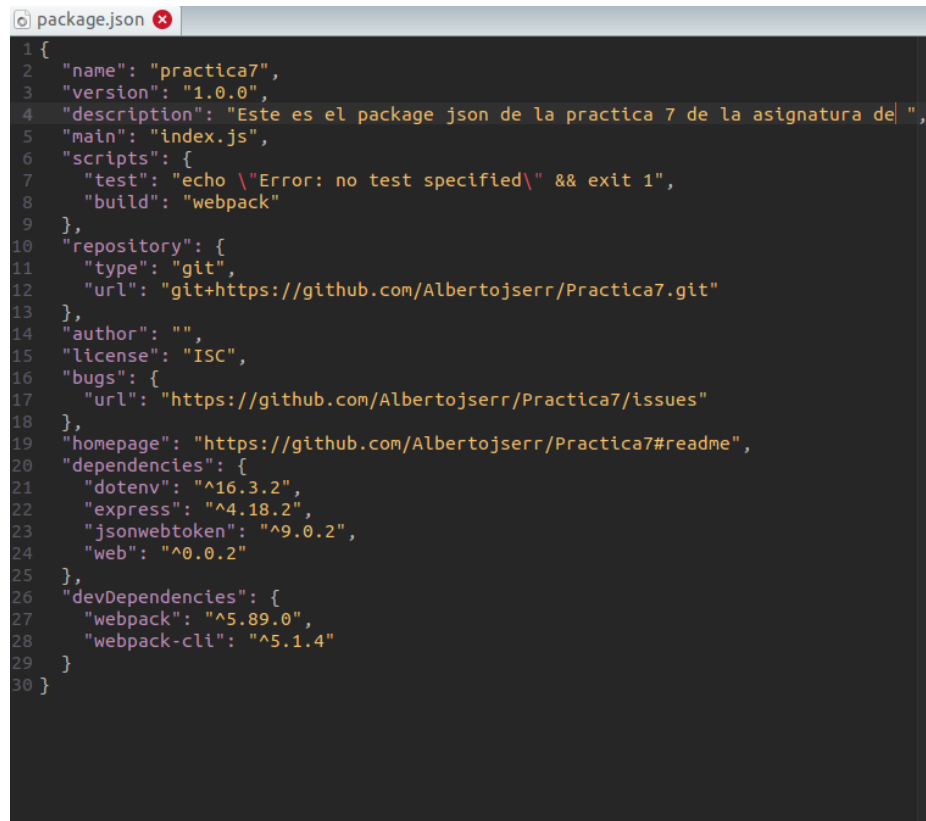
Crearemos un archivo de configuración llamado `webpack.config.js`.

```

webpack.config.js x
1 const path = require('path');
2
3 module.exports = {
4   entry: './js/script.js',
5   output: {
6     filename: 'script1.js',
7     path: path.resolve(__dirname, 'dist'),
8   },
9 };

```

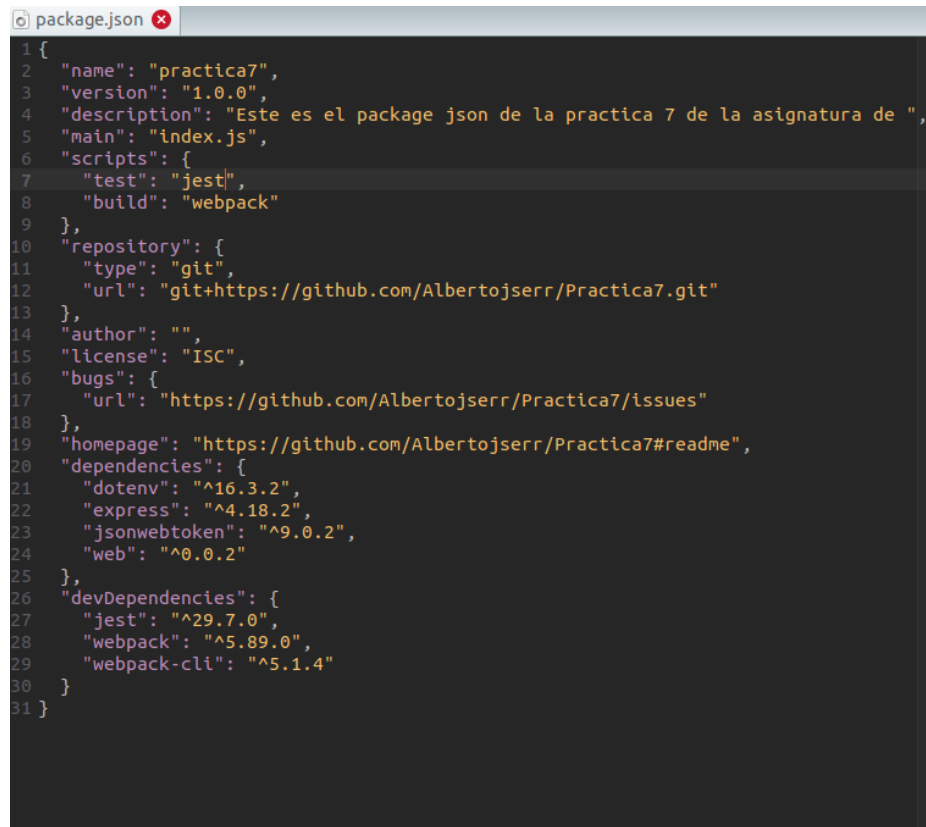
Y podemos observar en el `package.json` como las dependencias se han actualizado y que hemos añadido un script.



```
1 {
2   "name": "practica7",
3   "version": "1.0.0",
4   "description": "Este es el package json de la practica 7 de la asignatura de ",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1",
8     "build": "webpack"
9   },
10  "repository": {
11    "type": "git",
12    "url": "git+https://github.com/Albertojserr/Practica7.git"
13  },
14  "author": "",
15  "license": "ISC",
16  "bugs": {
17    "url": "https://github.com/Albertojserr/Practica7/issues"
18  },
19  "homepage": "https://github.com/Albertojserr/Practica7#readme",
20  "dependencies": {
21    "dotenv": "^16.3.2",
22    "express": "^4.18.2",
23    "jsonwebtoken": "^9.0.2",
24    "web": "^0.0.2"
25  },
26  "devDependencies": {
27    "webpack": "^5.89.0",
28    "webpack-cli": "^5.1.4"
29  }
30 }
```

Como hemos visto, en el script de test tenemos un error ya que no hay ningún test especificado por lo que vamos a utilizar Jest para las pruebas automatizadas.

```
npm install jest --save-dev
```

A screenshot of a code editor window titled 'package.json'. The editor shows a JSON configuration for a project named 'practica7'. The configuration includes fields for name, version, description, main script, test script (jest), build script (webpack), repository (git), author, license (ISC), bugs, homepage, dependencies (dotenv, express, jsonwebtoken, web), and devDependencies (jest, webpack, webpack-cli).

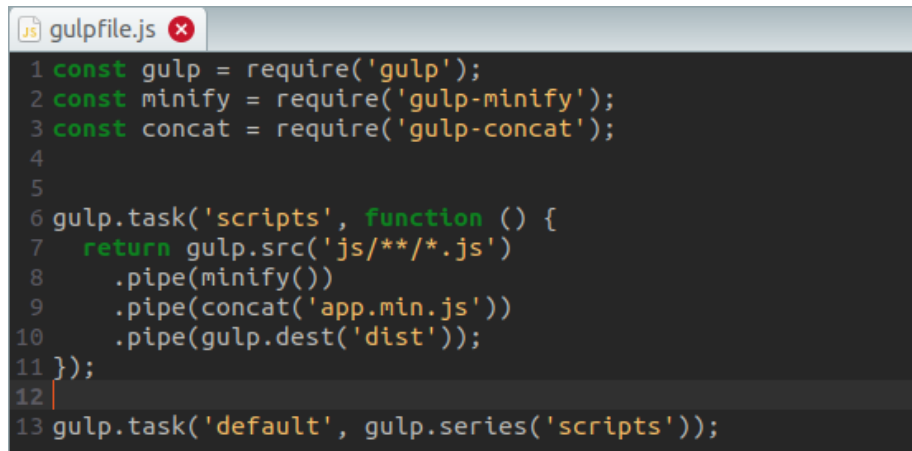
```
1 {
2   "name": "practica7",
3   "version": "1.0.0",
4   "description": "Este es el package json de la practica 7 de la asignatura de ",
5   "main": "index.js",
6   "scripts": {
7     "test": "jest",
8     "build": "webpack"
9   },
10  "repository": {
11    "type": "git",
12    "url": "git+https://github.com/Albertojserr/Practica7.git"
13  },
14  "author": "",
15  "license": "ISC",
16  "bugs": {
17    "url": "https://github.com/Albertojserr/Practica7/issues"
18  },
19  "homepage": "https://github.com/Albertojserr/Practica7#readme",
20  "dependencies": {
21    "dotenv": "^16.3.2",
22    "express": "^4.18.2",
23    "jsonwebtoken": "^9.0.2",
24    "web": "^0.0.2"
25  },
26  "devDependencies": {
27    "jest": "^29.7.0",
28    "webpack": "^5.89.0",
29    "webpack-cli": "^5.1.4"
30  }
31 }
```

Ahora vamos a utilizar la herramienta de automatización Gulp para automatizar tareas.

```
npm install -g gulp
```

```
npm install gulp --save-dev
```

Crearemos un archivo de configuración para Gulp llamado gulpfile.js.



```

1 const gulp = require('gulp');
2 const minify = require('gulp-minify');
3 const concat = require('gulp-concat');
4
5
6 gulp.task('scripts', function () {
7   return gulp.src('js/**/*.js')
8     .pipe(minify())
9     .pipe(concat('app.min.js'))
10    .pipe(gulp.dest('dist'));
11 });
12
13 gulp.task('default', gulp.series('scripts'));

```

Instalaremos las dependencias necesarias: `npm install gulp-minify gulp-concat --save-dev` Y ya podremos ejecutar las tareas de Gulp ejecutando el comando "gulp".

6. Entrega continua: Jenkins

La entrega continua implica la automatización de la integración, pruebas y despliegue de tu aplicación en un entorno de producción cada vez que se realizan cambios en el repositorio.

6.1. Configuración

Primero, añadimos la clave del repositorio:

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg
```

Añadimos los repositorios y actualizamos:

```
sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt update
```

Finalmente podremos instalar jenkins:

```
apt install jenkins
```

Una vez instalado, iniciamos el servicio y lo habilitamos al arranque del servidor:

```
systemctl start jenkins.service
systemctl enable jenkins.service
```

Para poder ver en que puerto se encuentra la aplicación de Jenkins ponemos:

```
systemctl status jenkins.service
```

```
jenkins.service - Jenkins Continuous Integration Server
Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
Active: active (running) since Sun 2024-01-21 17:53:58 CET; 1min 47s ago
Main PID: 10236 (java)
Tasks: 45 (limit: 9288)
Memory: 2.1G
CPU: 2min 15.702s
CGroup: /system.slice/jenkins.service
        └─10236 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
```

Al iniciar nos pide una contraseña que la podemos encontrar si escribimos en una terminal:

```
sudo nano /var/lib/jenkins/secrets/initialAdminPassword
```

Getting Started


Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password



Continue

Al poner la contraseña, nos aparecerá la siguiente pestaña en la que nos dan las opciones de instalar los plugins sugeridos o seleccionar aquellos que queramos. Instalaremos los sugeridos.



Una vez finalice la instalación, deberemos de crear la cuenta administradora.

Getting Started

Create First Admin User

Usuario

admin

Contraseña

Confirma la contraseña

Nombre completo

admin

Dirección de email

admin@admin.com|

Jenkins 2.426.2

[Skip and continue as admin](#)

Save and Continue

Por último, podemos seleccionar la URL de acceso.

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.426.2

Not now [Save and Finish](#)

Con esto terminamos de configurar Jenkins.


6.2. Flujo de trabajo

Ahora lo que haremos será crear un proyecto. Lo llamaremos Practica7 y será de estilo libre que es lo más básico de Jenkins y nos permitirá ejecutar el proyecto combinando cualquier tipo de repositorio con cualquier modo de construcción.

Enter an item name


Practica7

» Required field




Crear un proyecto de estilo libre

Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.




Pipeline

Gestiona actividades de larga duración que pueden abarcar varios agentes de construcción. Apropiado para construir pipelines (conocidas anteriormente como workflows) y/o para la organización de actividades complejas que no se pueden articular fácilmente con tareas de tipo freestyle.




Crear un proyecto multi-configuración

Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.




Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.



Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

Definiremos un flujo de trabajo y nos aseguraremos de que los despliegues se realicen de forma incremental y sin interrupciones en el servicio, para ello nos iremos a la parte de configuración del proyecto y especificaremos el repositorio, que lance ejecuciones concurrentes en caso de ser necesario además de los disparadores de ejecución y en que entorno se ejecuta.

General

Enabled

Descripción

Plain text: Visualizar

☐ Desechar ejecuciones antiguas

☐ Esta ejecución debe parametrizarse

☐ GitHub project

☐ Throttle builds

☒ Lanzar ejecuciones concurrentes en caso de ser necesario

Avanzado

Configurar el origen del código fuente

☐ Ninguno

☒ Git

Git

Repositories

Repository URL:

Credentials:

+ Add

Avanzado

Add Repository

Branches to build

Branch specifier (blank for 'any')

Add Branch

Navegador del repositorio

Additional Behaviours

Añadir

Disparadores de ejecuciones

☐ Lanzar ejecuciones remotas (primero desde scripts)

☐ Construir tras otros proyectos

7. Conclusión

Hemos creado un flujo de trabajo automatizado que nos ha servido para implementar y desplegar una aplicación web en un entorno de producción utilizando herramientas de entrega continua. A lo largo del proyecto, hemos utilizado herramientas diversas como son Nginx, Node.js además de la herramienta de automatización llamada Webpack o un sistema de integración continua, como Jenkins.