

Proyecto de software basado en herramientas de integración continua

Integración continua

Entrega Semana 5

Integrantes:

Lisbeth Alvarez Zuluaga

Andres Felipe Triana Gallego

Politécnico Grancolombiano

Ingeniería de Software

2023

Contenido:.....	2
-----------------	---

### **Entrega semana 3**

Proyecto a realizar y lenguaje .....	3
--------------------------------------	---

Justificación .....	4
---------------------	---

Alcances.....	5
---------------	---

Enlace repositorio y Contexto del código .....	6
--	---

### **Entrega semana 5**

Implementación Jenkins .....	8
------------------------------	---

## **Proyecto a realizar y lenguaje**

Realizaremos un ecommerce que nos permita poner en práctica todo lo aprendido, además de generar un entregable de alta confiabilidad, alta y rápida respuesta, fácil escalabilidad e integración. Para esto utilizaremos como base:

**Front-end:** JavaScript es esencial para el desarrollo del front-end de un sitio web de comercio electrónico. Aquí podemos utilizar bibliotecas y frameworks como React, Angular o Vue.js para crear interfaces de usuario interactivas y atractivas. Estas tecnologías nos permiten construir páginas de producto, carritos de compra, formularios de registro y otros componentes front-end.

**Back-end:** Para el desarrollo del back-end, Node.js nos parece una excelente opción. Node.js nos permite utilizar JavaScript en el lado del servidor. Puedes construir una API RESTful o GraphQL para gestionar la lógica empresarial, manejar la autenticación de usuarios, procesar pedidos y gestionar la base de datos.

**Base de datos:** JavaScript no es un lenguaje de base de datos, por lo que necesitaremos una base de datos para almacenar la información de productos, pedidos, usuarios, etc. Usaremos bases de datos relacionales MySQL.

## **Justificación**

Crear un ecommerce con las características mencionadas en la respuesta anterior que permita emplear herramientas para la integración continua, utilizando JavaScript tanto en el front-end como en el back-end, nos da ventajas y aspectos positivos. Aquí hay algunos de ellos:

### **Ventajas:**

**Versatilidad tecnológica:** El uso de JavaScript en el front-end y Node.js en el back-end nos brinda una plataforma versátil y unificada para el desarrollo.

**Interactividad y experiencia de usuario mejorada:** JavaScript nos permite crear interfaces de usuario altamente interactivas y atractivas. Los frameworks como React, Angular y Vue.js son ideales para proporcionar a los usuarios una experiencia de compra más agradable y personalizada.

**Eficiencia en el desarrollo:** Al utilizar JavaScript en todo el ciclo de desarrollo, nos permite optimizar la eficiencia y la velocidad de desarrollo.

**Comunidad y recursos abundantes:** JavaScript tiene una comunidad de desarrollo activa y una gran cantidad de recursos, tutoriales y bibliotecas disponibles para facilitar el desarrollo y resolver problemas comunes.

**Escalabilidad y rendimiento:** Node.js es conocido por su capacidad de manejar un alto volumen de conexiones simultáneas, lo que lo hace adecuado para sitios web de comercio electrónico con un tráfico creciente.

**Seguridad:** Con un enfoque adecuado en la seguridad, podemos garantizar que las transacciones en línea y los datos de los usuarios estén protegidos.

Flexibilidad en la elección de bases de datos: JavaScript nos permite elegir entre una variedad de bases de datos, ya sea una base de datos relacional o NoSQL, según nuestras necesidades y preferencias.

### **Alcances:**

Gestión de productos: Poder administrar eficientemente una amplia variedad de productos, descripciones, precios y categorías en nuestro ecommerce.

Procesamiento de pedidos: Lograr automatizar el proceso de procesamiento de pedidos, desde la selección de productos hasta la confirmación y envío.

Gestión de usuarios y autenticación: Poder implementar un sistema de autenticación seguro para que los usuarios puedan crear cuentas, iniciar sesión y realizar un seguimiento de sus pedidos.

Pasarelas de pago: Integrar pasarelas de pago seguras y confiables para procesar transacciones en línea.

Seguimiento de inventario: Realizar seguimientos del inventario en tiempo real y gestiona el stock de productos.

Análisis y métricas: Implementar herramientas de análisis para rastrear el rendimiento de la tienda, las conversiones y el comportamiento de los usuarios.

Personalización y recomendaciones: Poder personalizar la experiencia de compra de los usuarios y ofrecer recomendaciones de productos relevantes.

- **Enlace del repositorio:** <https://github.com/lisalvarez18/proyecto-ecommerce>

- Estructura básica del proyecto node js la cual fue creada usando el administrador de paquetes NPM.

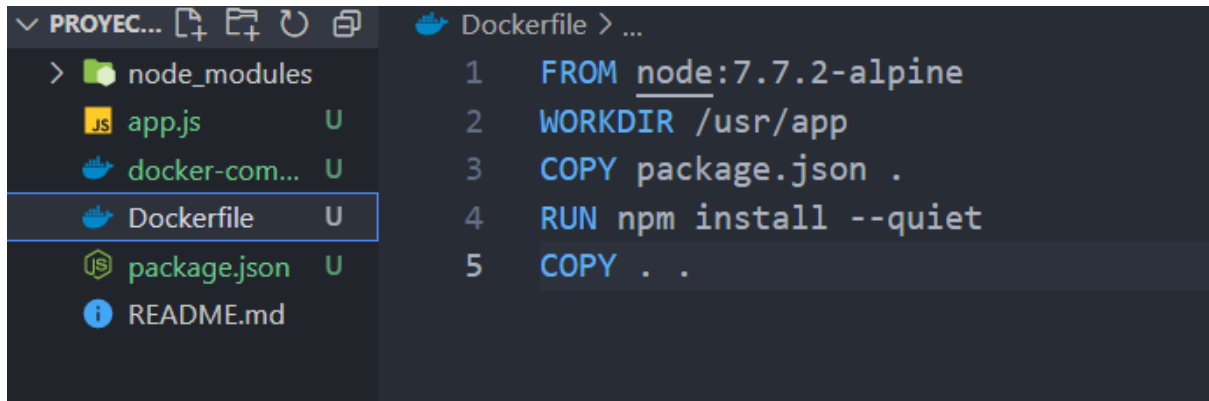
```
1 {
2   "name": "proyecto-ecommerce",
3   "version": "1.0.0",
4   "description": "Proyecto de comercio electrónico",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "repository": {
10    "type": "git",
11    "url": "git+https://github.com/lisalvarez18/proyecto-ecommerce.git"
12  },
13  "author": "",
14  "license": "ISC",
15  "bugs": {
16    "url": "https://github.com/lisalvarez18/proyecto-ecommerce/issues"
17  },
18  "homepage": "https://github.com/lisalvarez18/proyecto-ecommerce#readme"
19 }
20
```

- Hola mundo en node js usando un servidor http.

```
> node_modules
  app.js U
  docker-com... U
  Dockerfile U
  package.json U
  README.md

1
2 var http = require('http');
3 var server = http.createServer();
4
5 function mensaje(petic, resp) {
6   resp.writeHead(200, {'content-type': 'text/plain'});
7   resp.write('Hola Mundo');
8   resp.end();
9 }
10 server.on('request', mensaje);
11
12 server.listen(3000, function () {
13   console.log('La Aplicación está corriendo en el puerto 3000');
14 });
```

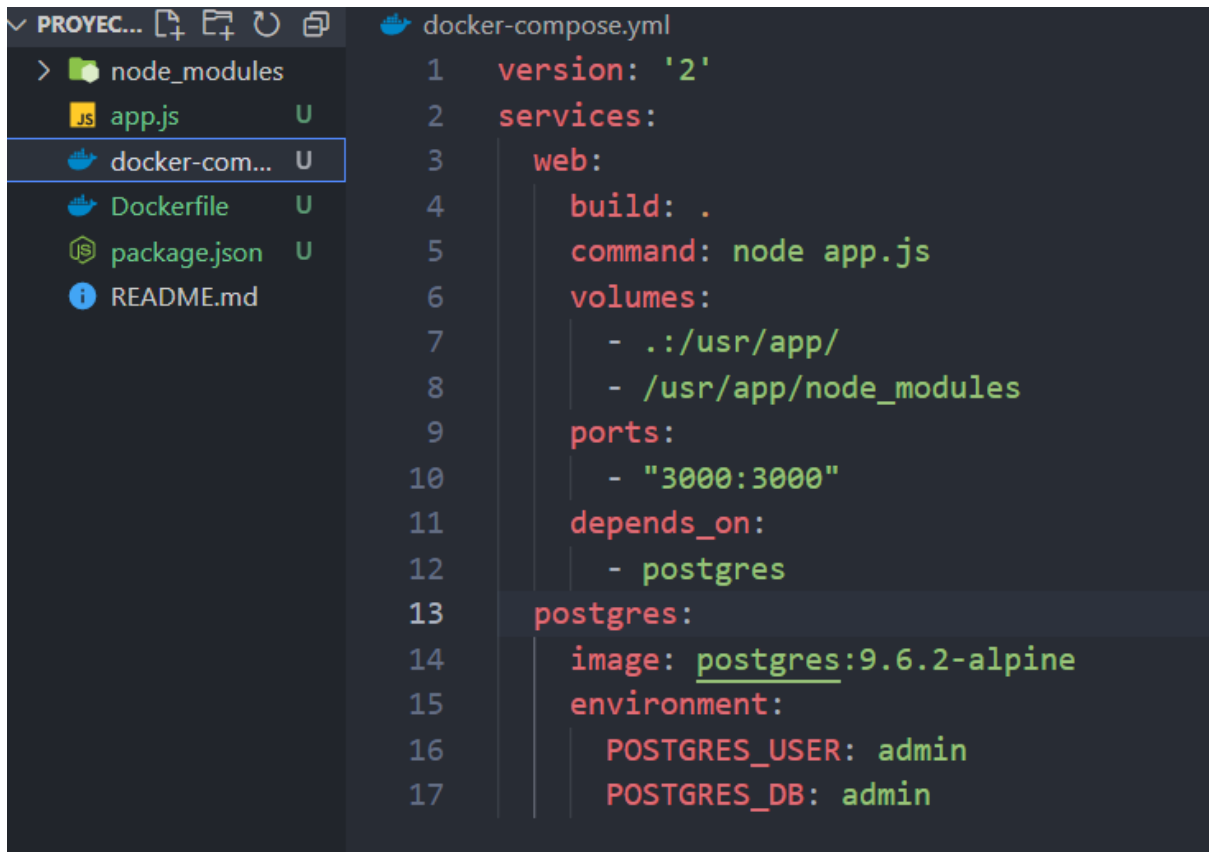
- Este código se usa para crear una imagen de la aplicación.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PROYEC...' with a folder 'node\_modules' and files 'app.js', 'docker-com...', 'Dockerfile', 'package.json', and 'README.md'. The 'Dockerfile' file is selected. The code editor shows the content of the Dockerfile, which consists of five lines of Docker instructions.

```
1 FROM node:7.7.2-alpine
2 WORKDIR /usr/app
3 COPY package.json .
4 RUN npm install --quiet
5 COPY . .
```

- Este código se usa para levantar dos contenedores, uno basado en la imagen de nuestra aplicación y el otro basado en la imagen de la base de datos de postgres.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PROYEC...' with a folder 'node\_modules' and files 'app.js', 'docker-com...', 'Dockerfile', 'package.json', and 'README.md'. The 'docker-com...' file is selected. The code editor shows the content of the docker-compose.yml file, which consists of 17 lines of YAML configuration.

```
1 version: '2'
2 services:
3   web:
4     build: .
5     command: node app.js
6     volumes:
7       - ../usr/app/
8       - /usr/app/node_modules
9     ports:
10      - "3000:3000"
11     depends_on:
12       - postgres
13   postgres:
14     image: postgres:9.6.2-alpine
15     environment:
16       POSTGRES_USER: admin
17       POSTGRES_DB: admin
```

## Implementación Jenkins

- En este archivo docker file creamos una imagen propia de Jenkins la cual va a tener instalado Docker.

```
jenkins > 🐳 Dockerfile > ...
1  FROM jenkins/jenkins:lts
2  USER root
3  RUN apt-get update
4  RUN curl -sSL https://get.docker.com/ | sh
```

El siguiente comando se usa para construir nuestra imagen de Jenkins.

```
docker build -t lisbeth/jenkins .
```

Este comando es para correr nuestra imagen de jenkins con los respectivos puertos y volúmenes.

```
docker run -p 8080:8080 -p 50000:50000 -v
```

```
<Jenkins_home>:/var/jenkins_home -v
```

```
/var/run/docker.sock:/var/run/docker.sock lisbeth/Jenkins
```

- Luego de que corremos nuestro contenedor de Jenkins se nos proporcionará a través de los logs un password para iniciar la configuración de Jenkins.

```
2023-11-16 21:32:46.013+0000 [id=31] INFO jenkins.install.SetupWizard#init:
*****
*****
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
de7e99dd77aa49a28839c72bdc429351
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****
*****
*****
2023-11-16 21:33:33.841+0000 [id=47] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
2023-11-16 21:33:33.842+0000 [id=47] INFO hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt #1
2023-11-16 21:33:34.280+0000 [id=31] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
```




- Al configurar Jenkins nos pedirá algunos datos (nombre, usuario, contraseña y email) y adicional se sugerirán algunos plugins, en nuestro caso instalamos los sugeridos más node.js.
- En Jenkins procedemos a crear nuestra tarea de pipeline.

Dashboard > All >


### Enter an item name

simple-node-react-npm-app


» Required field



**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

New Item de tipo Pipeline, OK.

- Comenzamos la configuración e ingresamos una breve descripción.

## General

Enabled 

Descripción

Pipeline para construir testear y desplegar nuestro proyecto e-commerce usando npm

Plain text [Visualizar](#)

- Configuramos nuestro pipeline para que funcione desde un script de SCM (supply chain management), y en esta ventana configuramos la url y las credenciales de nuestro repositorio, para que de esta forma jenkins tenga acceso al código.

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/lisalvarez18/proyecto-ecommerce

Credentials ?

lisalvarez18/\*\*\*\*\*

+ Add

Avanzado

Guardar Apply

- Configuramos la rama de la cual Jenkins va a obtener el código.

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

- En este campo de texto ingresamos el path de nuestro Jenkins file, el cual contiene la configuración de nuestro pipeline.

Script Path ?

Jenkinsfile

- Este es el Código de nuestro “Hola mundo”, el cual fue modificado para facilitar los test.

```
app.js > ...
1  const express = require("express");
2  const app = express();
3  const port = process.env.PORT || 3000;
4
5  app.get("/", "/index.html", (req, res) => {
6    res.send("Hola mundo");
7  });
8
9  if (process.env.NODE_ENV !== "test") {
10    app.listen(port, () => {
11      console.log(`La aplicacion esta corriendo en el puerto ${port}`);
12    });
13  }
14
15  module.exports = app;
```

- Estas corresponden a nuestras pruebas unitarias.

```

test.js > ...
1  const server = require("../app");
2  const supertest = require("supertest");
3  const { default: expect } = require("expect");
4
5  describe("Homepage", () => {
6      const checkHome = (supertest_response) => {
7          expect(supertest_response.status).toEqual(200);
8          expect(supertest_response.type).toEqual('text/html');
9          expect(supertest_response.text).toEqual("Hola mundo");
10     }
11     it("GET / return Hola mundo", async () => {
12         const res = await supertest(server).get("/");
13         checkHome(res);
14     });
15     it("GET /index.html return Hola mundo", async () => {
16         const res = await supertest(server).get("/index.html");
17         checkHome(res);
18     });
19 });

```

- La siguiente corresponde a nuestro Jenkins file que contiene nuestra configuración para nuestro pipeline en Jenkins. En este mismo archivo podemos ver las diferentes etapas de nuestro pipeline.

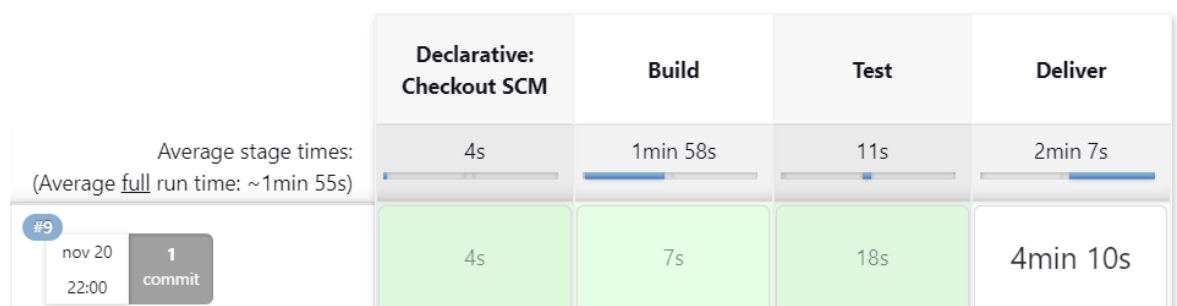
```

Jenkinsfile
1 pipeline {
2   agent {
3     docker {
4       image 'node:20.9.0-alpine3.18'
5       args '-p 3000:3000'
6     }
7   }
8   environment {
9     CI = 'true'
10  }
11  stages {
12    stage('Build') {
13      steps {
14        sh 'npm install'
15      }
16    }
17    stage('Test') {
18      steps {
19        sh 'npm test'
20      }
21    }
22    stage('Deliver') {
23      steps {
24        sh 'node app.js'
25        input message: 'Finished using the web site? (Click "Proceed" to continue)'
26        sh 'kill -INT 888'
27      }
28    }
29  }
30 }

```

Aquí Podemos ver una captura de cada una de las etapas del pipeline en la consola de Jenkins.

## Stage View



- En estos logs que arroja Jenkins cuando ejecutamos nuestro pipeline, podemos apreciar como Jenkins descarga nuestro código desde nuestro repositorio.

```

Started by user Lisbeth Alvarez
Obtained Jenkinsfile from git https://github.com/lisalvarez18/proyecto-e-commerce
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/e-commerce-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
using credential 9bf8bebb-8b77-4817-91e5-8073a45717a0
> git rev-parse --resolve-git-dir /var/jenkins_home/workspace/e-commerce-pipeline/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/lisalvarez18/proyecto-e-commerce # timeout=10
Fetching upstream changes from https://github.com/lisalvarez18/proyecto-e-commerce
> git --version # timeout=10
> git --version # 'git version 2.39.2'
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- https://github.com/lisalvarez18/proyecto-e-commerce +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 28475dcac0a6133978258f8b54591f8e39d8133c (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 28475dcac0a6133978258f8b54591f8e39d8133c # timeout=10
Commit message: "Borrando npm build"
> git rev-list --no-walk e466a87ebda9378d587b0f708d19a3dcb1175402 # timeout=10

```

En las siguientes capturas Podemos apreciar cada una de las etapas de nuestro pipeline.

- **Build:** En esta etapa ejecutamos el comando npm install , el cual descarga todas las dependencias de nuestro proyecto.

```

[Pipeline] { (Build)
[Pipeline] sh
+ npm install

up to date, audited 374 packages in 2s

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

found 0 vulnerabilities

```

- **Test:** En esta etapa se ejecutan las pruebas unitarias.

```
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] sh
+ npm test

> proyecto-ecommerce@1.0.0 test
> cross-env CI=true NODE_ENV=test jest --testTimeout=10000 --detectOpenHandles --forceExit

PASS ./test.js (6.886 s)
  Homepage
    ✓ GET / return Hola mundo (152 ms)
    ✓ GET /index.html return Hola mundo (17 ms)

Test Suites: 1 passed, 1 total
Tests:      2 passed, 2 total
Snapshots:  0 total
Time:       7.291 s
Ran all test suites.
```

- **Deliver:** En esta etapa se ejecuta la aplicación y se detiene intencionalmente después de unos minutos.

```
[Pipeline] stage
[Pipeline] { (Deliver)
[Pipeline] sh
+ node app.js
La aplicacion esta corriendo en el puerto 3000
Sending interrupt signal to process
Aborted by Lisbeth Alvarez
```