

Universidad Tecnológica Nacional
Facultad Regional de Córdoba

Integración Continua

Preparado por

Grupo 2 - Cátedra Ingeniería de Software.

Capitanio, Alejandro - de la Orden, Lourdes - Fassino, María Belén - García,
Rodrigo Nahuel - Jover, María Magdalena - Olalde, Tomás Matias

para

Ingenieros en Sistemas de Información / Técnicos / Personas afines a la
tecnología

Fecha de entrega: 30 de Agosto, 2019

RESUMEN

Las metodologías ágiles aplicadas al desarrollo de software están siendo usadas prácticamente en todas las empresas, desde las más pequeñas hasta las que tienen ya muchos años en el mercado, gracias a que son realmente efectivas, ya que, son muy flexibles en cuanto a los cambios, y con una correcta implementación, las probabilidades de que el software entregado sea útil para el cliente son muy altas. Los resultados del estudio darán a conocer las ventajas de esta implementación y cuáles son las tecnologías básicas para implementarlo en un proyecto real.

Creemos que la mejor manera de entender cómo funciona esta práctica es de manera práctica, por lo que vamos a crear una implementación muy básica de lo que sería la Integración Continua con Jenkins, Git, GitHub, Docker y un servidor en la nube

PALABRAS CLAVE

Software, Integración Continua, Servidor, Metodologías Ágiles, Commit, Push, Pull, Merge, Comando, Código, Desarrollador, API, Testing, Test Case, Root, Kernel, Contenedor, Rama, Consola, Web, Sistema Operativo, Máquina Remota, IP, Software Libre, Puerto de Conexión, Repositorio, Http, Plugin, Link.

TABLA DE CONTENIDOS

RESUMEN	1
INTRODUCCIÓN	3
CONCEPTOS TEÓRICOS	4
Continuous Integration	4
Ciclo de vida básico	4
Tecnologías a utilizar	5
DESARROLLO	7
1. Servidor de integración continua	7
1.1. Cuenta Digital Ocean	7
1.2. Droplet	7
1.3 Visualizar servidor	8
1.4. Ingresar al servidor	9
2. Instalación de Jenkins	9
3. Crear un proyecto en Jenkins	11
3.1. Crear nuevo trabajo en Jenkins	11
3.2. Configuración inicial del trabajo	12
3.3. Instalación de plugins	13
4. Construcción del proyecto	14
5. Crear contenedor de Docker	16
5.1. Contenedor Docker	16
5.2. Clonar repositorio Docker	16
5.3. Construir proyecto Docker	17
5.4. Ejecutar contenedor Docker	17
5.5. Verificar ejecución de Docker	18
6. Crear Repositorio en Docker	18
6.1. Configurar repositorio Docker	18
6.2. Configuración de Docker dentro de Jenkins	19
7. Desplegar API	21
7.1. Construir proyecto en Jenkins	21
7.2. Testing despliegue de aplicación	23
RESULTADOS	24
CONCLUSION	25
REFERENCIAS	26

INTRODUCCIÓN

La integración continua es una práctica de desarrollo de software en la cual los integrantes del grupo combinan los cambios de código periódicamente y se ejecutan tareas automáticas en base a esos cambios. Esta práctica brinda muchas ventajas y se complementa perfectamente con las metodologías ágiles. El problema es que se necesita implementar varias herramientas de desarrollo para poder lograr una integración continua dentro de un proyecto de software, por lo que la o las personas que configuren un ambiente con estas características debe conocer cada una de esas herramientas y cómo se comunican entre sí. Partiendo de este problema, en el desarrollo de este reporte vamos a dar un enfoque práctico al mismo, intentando crear un ambiente de integración continua implementando alguna de las herramientas que suelen utilizarse en ambientes profesionales, ya que, se puede encontrar mucha información teórica sobre cada una de las herramientas, pero no se encuentra tanto cuando se busca información para implementar todas juntas.

Resumidamente, se contratará un servidor en la nube con la intención de instalar Jenkins para que desde Jenkins nos conectemos a un repositorio donde ya hay almacenada una simple api desarrollada en Node.js y que al momento de desplegar nuestra aplicación se despliegue dentro de una imagen de Docker.

El objetivo de este proceso es entender cuáles son las mínimas configuraciones que hay que realizar para implantar integración continua, viendo desde un punto de vista práctico que no es tan difícil configurar estas herramientas y que las ventajas que brinda la integración continua son muchas.

En los resultados esperamos tener funcionando todas estas herramientas y poder realizar pruebas con las mismas, ya que estarán todas alojadas en un servidor de integración continua a disposición de todos los integrantes del equipo. Además, tener el conocimiento y la documentación para replicar este proceso con mucho más detalle en proyectos futuros.

CONCEPTOS TEÓRICOS

A continuación se tratará brevemente cuales son los conceptos más importantes antes de comenzar con el desarrollo práctico.

Continuous Integration (CI)

Es una forma de trabajo en la que continuamente se actualiza en un repositorio los avances que un integrante del proyecto realiza durante el día, como mínimo se pide realizar un commit diario de los cambios realizados.

Cuando el desarrollador haga commit recibirá una respuesta de manera automática por parte del servidor, donde estará alojado el proyecto. Esta respuesta hacia el desarrollador puede estar constituida por información muy variada ya que se construye en el servidor. La información recibida puede contener:

- Test Cases con éxito y con error.
- Errores de sintaxis en el código, si es que existe algún error.
- Dónde ocurrieron los errores al momento de compilar el código.
- Si se pudo mergear con el código o es necesario realizar cambios.

Esto permite detectar errores muy rápidamente e incrementa la calidad del código. Lo que hace que cada avance en el proyecto sea mucho más limpio, y que realmente sea un avance y no se tenga que volver constantemente a resolver antiguos errores.

Ciclo de vida básico

Si bien más adelante se describirá el proceso con más detalle, un ciclo de vida básico puede ser el siguiente:

1. Un desarrollador en un día de trabajo normal hace una copia del código que está almacenado en un servidor como puede ser GitHub, BitBucket, etc.
2. Realiza cambios sobre el código, ya sea corrección de errores o nuevas funcionalidades.
3. El desarrollador sube los cambios realizados a un repositorio por medio de un commit.
4. Al momento de subir el código al repositorio se ejecutan los Test Cases configurados previamente sobre todo el proyecto, y en caso de detectar algún error se notifica al desarrollador.
5. El desarrollador arregla los cambios y vuelve a subir el código hasta que los cambios estén correctamente corregidos.

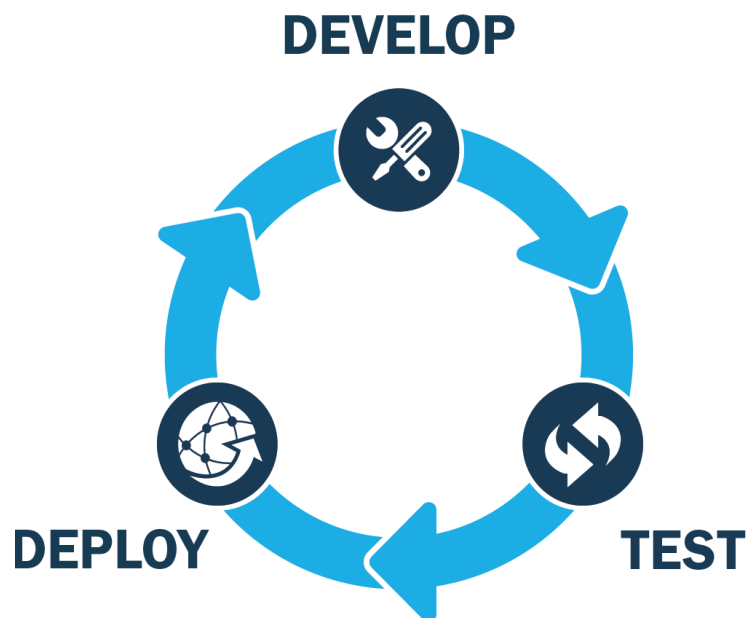


Figura 1

Tecnologías a utilizar

A continuación daremos un panorama general de las herramientas que vamos a utilizar para implantar Integración Continua.

Tenemos que tener en cuenta que existen muchas herramientas y formas diferentes para poder realizar Integración Continua y que tenga el mismo resultado o todavía mejor.

Git: Es un sistema de control de versiones muy potente que permite gestionar ramas de trabajo, conflictos, integración de código y muchas funcionalidades que brindan control completo del código y estado del mismo.

GitHub: Git nos provee la funcionalidad de tener un repositorio local para poder ir guardando los cambios a medida que escribimos código, pero no hay forma de recuperar el código en caso de que nuestra computadora deje de funcionar, ahí es donde entra GitHub. GitHub nos da la posibilidad de alojar nuestro código en la nube con una pequeña cantidad de comandos, lo que hace que nuestro código siempre esté seguro y disponible.

También podemos compartir nuestro código, tener un registro de quienes fueron las personas que modificaron el mismo y una cantidad muy grande de funcionalidades adicionales.

Jenkins: Esta herramienta nos permite configurar fácilmente un ambiente para la integración continua para la mayoría de los lenguajes de programación a través de pipelines.

Docker: Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del Kernel Linux para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

Además, permite crear tareas automáticas que se ejecutan cuando sucede algún cambio en el repositorio.

A continuación intentaremos configurar un ambiente de CI. Reiteramos que existen muchas formas y herramientas distintas para poner en producción un ambiente de CI, estas son algunas de las más conocidas:



Figura 2

Nosotros nos enfocaremos en el uso de **Jenkins** por la gran cantidad de información que existe en la web.

DESARROLLO

A continuación veremos una representación de la arquitectura de servidores que vamos a utilizar para la implementación.

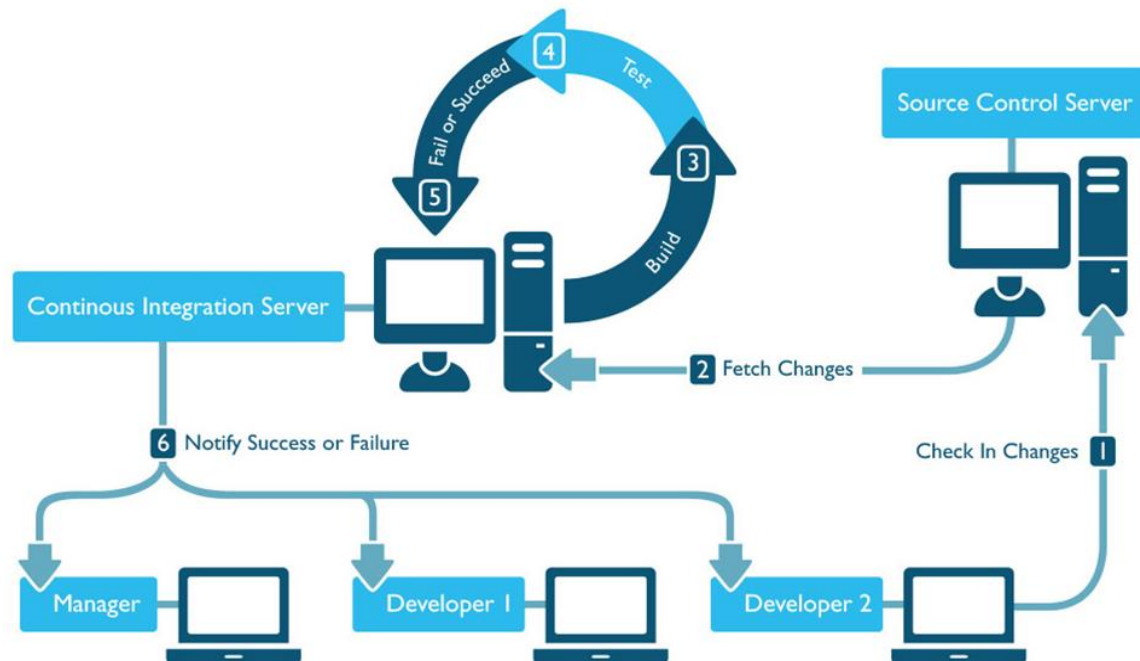


Figura 3

En esta implementación se aloja a la herramienta de integración continua (Jenkins) en un servidor dedicado que podemos contratar de forma gratuita por un mes para poder realizar este tipo de pruebas. Este servidor se puede contratar por medio de **DigitalOcean** que es una empresa que brinda dicho servicio, además de la facilidad de poder manejar el servidor por una consola desde la misma web y distintas métricas sobre el estado del servidor.

1. Servidor de Integración continua.

Necesitamos obtener un servidor para realizar la instalación de las herramientas que vamos a utilizar, por lo que obtendremos uno.

1.1 Cuenta Digital Ocean

Comenzaremos por crearnos una cuenta en <https://www.digitalocean.com/>.

DigitalOcean es una empresa que brinda diferentes servicios y uno de esas es la posibilidad de crear un servidor de una manera muy simple.

1.2 Droplet

Cuando hayamos creado la cuenta nos dirigimos al panel de herramientas que está a nuestra derecha y seleccionamos la opción Droplets.

Un **Droplet** es un servidor que podemos administrar de una forma muy sencilla por medio de una consola. Esta consola nos permite ingresar a nuestro servidor e instalar las distintas herramientas que vamos a necesitar.

En la Figura 4 veremos cuáles son las configuraciones iniciales de nuestro servidor, la más importante de ellas es el sistema operativo con el que va a contar nuestro servidor. Se recomienda seleccionar la última versión estable de Ubuntu que es la que vamos a estar utilizando para este estudio, en nuestro caso seleccionamos **Ubuntu 18.04 (LTS)**.

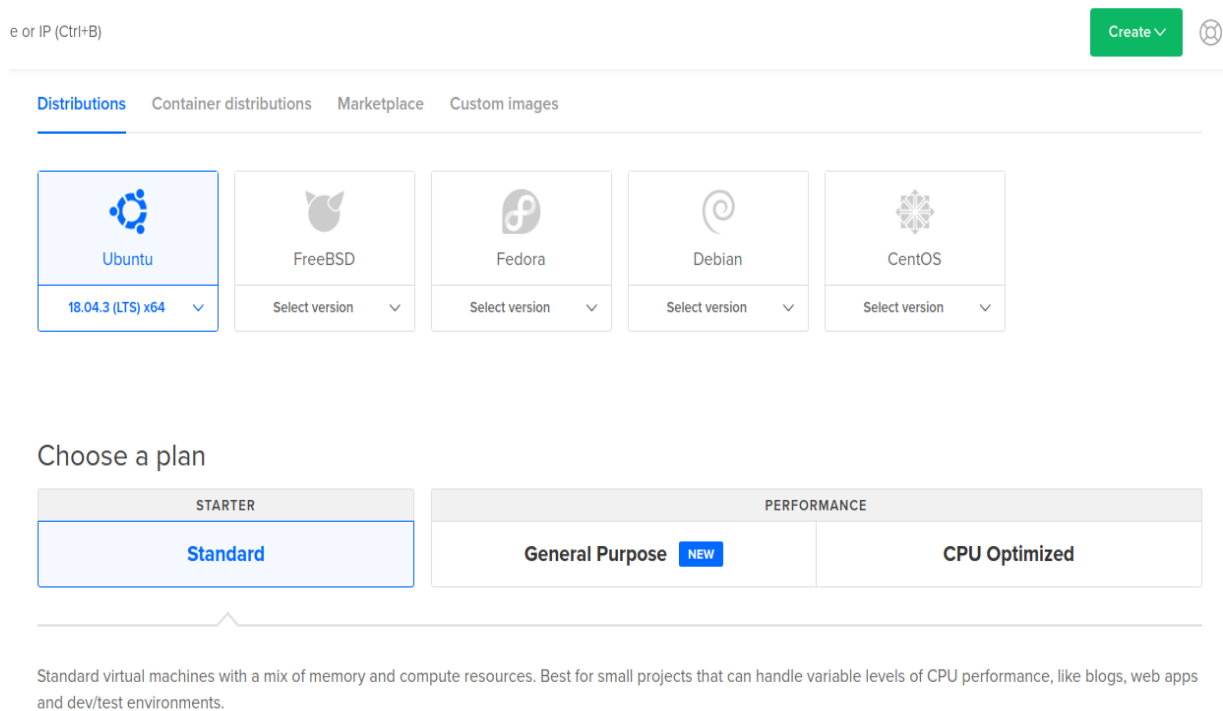


Figura 4

1.3 Visualizar servidor

Luego de que creamos nuestro servidor en nuestro panel principal tendremos la posibilidad de visualizar el estado de nuestro servidor y la opción para poder conectarnos con el mismo por medio una consola, como se observa en la Figura 5.



Figura 5

1.4 Ingresar al servidor

Por medio de la consola de comandos podremos instalar Jenkins y Docker que van a ser las dos herramientas principales que se va a alojar en nuestro servidor.

Para conectarnos al servidor solo debemos utilizar la dirección **IPv4** que nos brindan para la conexión, y por ser la primera vez se solicitará que creamos una contraseña para poder ser identificados cada vez que nos conectemos desde una máquina remota.

Por defecto tendremos el usuario **root** que es el que se nos va a solicitar cada vez que queramos ingresar al servidor.

2. Instalación de Jenkins.

La instalación de jenkins la haremos por medio de la consola con el siguiente comando:

```
wget http://raw.githubusercontent.com/wardviaene/jenkins-course/master/scripts/install_jenkins.sh
```

Este comando se encarga de descargar desde un repositorio un archivo que contiene una serie de scripts que se encargaran de la instalación de jenkins en nuestro servidor.

Cuando se termine de descargar ejecutamos el comando:

```
bash install_jenkins.sh
```

Que ejecuta una serie de scripts contenidos en ese archivo, que van a ser los encargados de instalar Jenkins en el servidor.

```
DigitalOcean Droplet Console - Google Chrome
DigitalOcean, LLC [US] | cloud.digitalocean.com/droplets/156088623/console?no_layout=true&i=a1ad92
Hit:3 https://apt.dockerproject.org/repo ubuntu-xenial InRelease
Hit:4 http://mirrors.digitalocean.com/ubuntu bionic-updates InRelease
Hit:5 http://mirrors.digitalocean.com/ubuntu bionic-backports InRelease
Fetched 242 kB in 1s (325 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
docker-engine is already the newest version (17.05.0~ce-0~ubuntu-xenial).
The following package was automatically installed and is no longer required:
  grub-pc-bin
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
Synchronizing state of docker.service with SysV service script with /lib/systemd
/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
usermod: user 'ubuntu' does not exist
docker: Error response from daemon: Conflict. The container name "/jenkins" is a
lready in use by container "9a36da2be33f6273ee0acb5939a3f170b8634adc18e0b628ed6
7c9b290f5d1a". You have to remove (or rename) that container to be able to reuse
that name.
See 'docker run --help'.
Jenkins installed
You should now be able to access jenkins at: http://157.245.13.179:8080
root@ubuntu-s-1vcpu-2gb-nyc3-01:~# _
```

Figura 6

Como podemos observar en la Figura 6, cuando terminamos de ejecutar este comando, se nos brinda una dirección **http://157.254.13.179:8080** la cual nos permite conectarnos a Jenkins. Esta dirección está compuesta por la dirección Ipv4 de nuestro servidor y por el puerto 8080 que es donde se está ejecutando Jenkins.

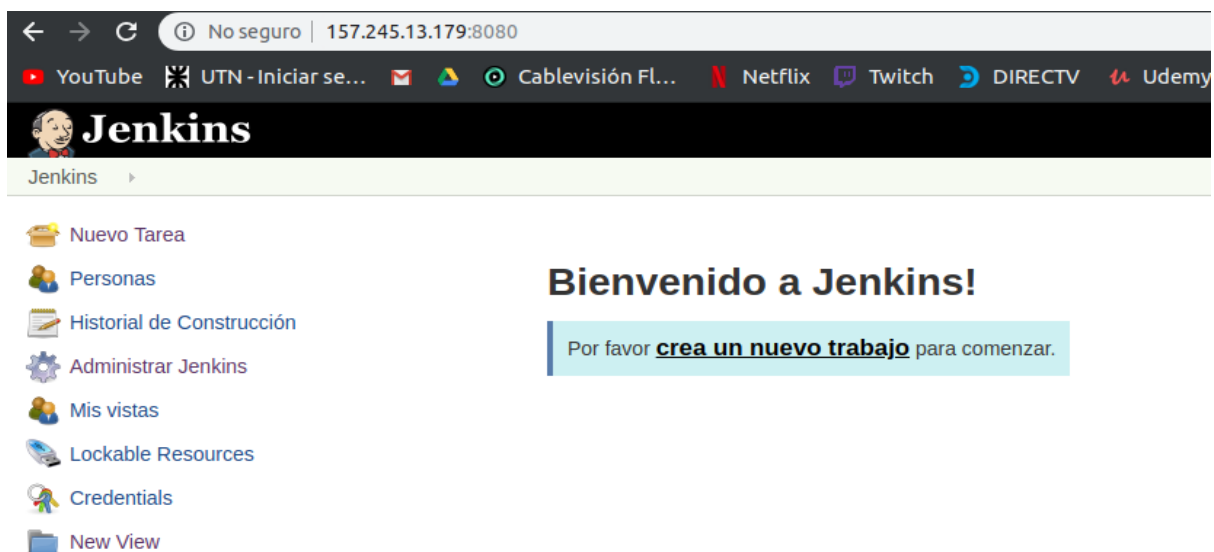


Figura 7

En caso de querer ver cuáles fueron precisamente los comandos que se ejecutaron, ejecute el siguiente comando dentro de la consola de administración del servidor.

```
cat install_jenkins.sh
```

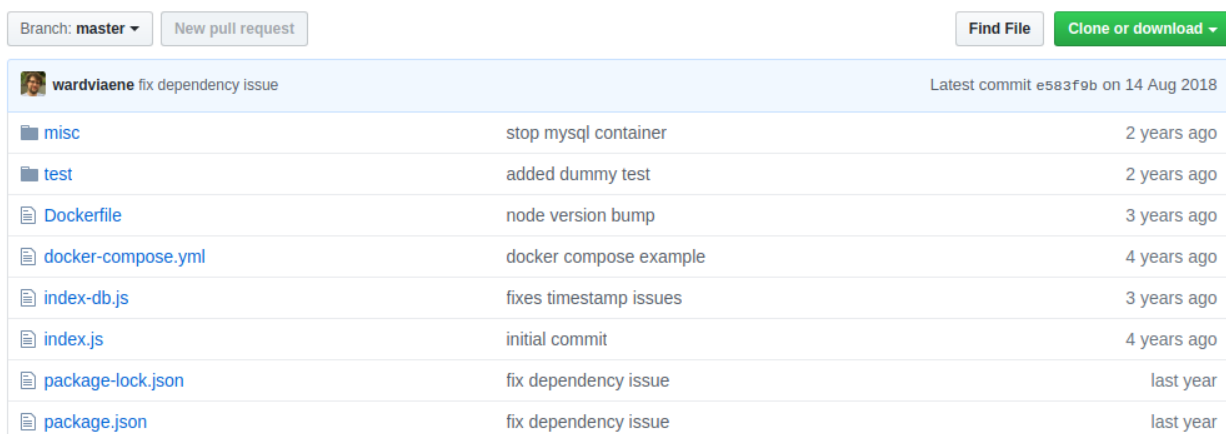
3. Crear un proyecto en Jenkins

Con el el servidor de Integración Continua en la nube funcionando correctamente, vamos a proceder a crear un nuevo proyecto.

Para este estudio nosotros ya hemos creado una API en un repositorio que va a servir de prueba para la configuración de nuestro proyecto. Esta API es muy sencilla y su finalidad es solo para comprender cómo sería desplegar un proyecto en la realidad.

Repositorio: <https://github.com/wardviaene/docker-demo>

Como se puede ver en la Figura 8, dentro del repositorio nos encontramos con una simple API de prueba creada con Node.js.



Branch: master ▾ New pull request		Find File	Clone or download ▾
wardviaene fix dependency issue		Latest commit e5a3f9b on 14 Aug 2018	
misc	stop mysql container	2 years ago	
test	added dummy test	2 years ago	
Dockerfile	node version bump	3 years ago	
docker-compose.yml	docker compose example	4 years ago	
index-db.js	fixes timestamp issues	3 years ago	
index.js	initial commit	4 years ago	
package-lock.json	fix dependency issue	last year	
package.json	fix dependency issue	last year	

Figura 8

Node.js es un compilador de código que te permite escribir aplicaciones del lado del servidor con javascript de forma muy rápida, por lo que nos fue de mucha utilidad para el caso de estudio.

Para más información sobre Node.js puede ir a la documentación de su página oficial, que se encuentra en la sección de Referencias.

El ítem de configuración que nos interesa en esta aplicación es “Dockerfile”. Este archivo contiene una simple configuración para Docker, que vamos a utilizar luego.

3.1 Crear nuevo trabajo en Jenkins

Para crear un proyecto en Jenkins lo que tenemos que hacer es crear una instancia desde Jenkins hacia el repositorio que contiene la aplicación, para eso clickeamos en donde dice crear nuevo trabajo.

Por favor **crea un nuevo trabajo** para comenzar.

Figura 9

Nos aparecerá un panel donde debemos ingresar el nombre de nuestro trabajo y seleccionaremos la opción crear un proyecto de estilo libre. Ver Figura 10.

Figura 10

Nos aparecerá un panel de configuración de nuestro trabajo donde tendremos que crear la instancia de nuestro repositorio, copiando el link de conexión del mismo. Ver Figura 11.

Figura 11

3.2 Configuración inicial del trabajo

Nuestra API, como la mayoría, utiliza módulos de terceros para poder funcionar, como por ejemplo el framework **Express** para aplicaciones **REST** que utilizamos. Por lo que debemos decirle a Jenkins que cuando cree el proyecto además de hacer la referencia hacia el repositorio, también debe ejecutar un comando específico para descargar los paquetes que va a utilizar nuestra API. Ese proceso se puede describir en la pestaña Ejecutar en el mismo panel de configuración. Ver Figura 12.

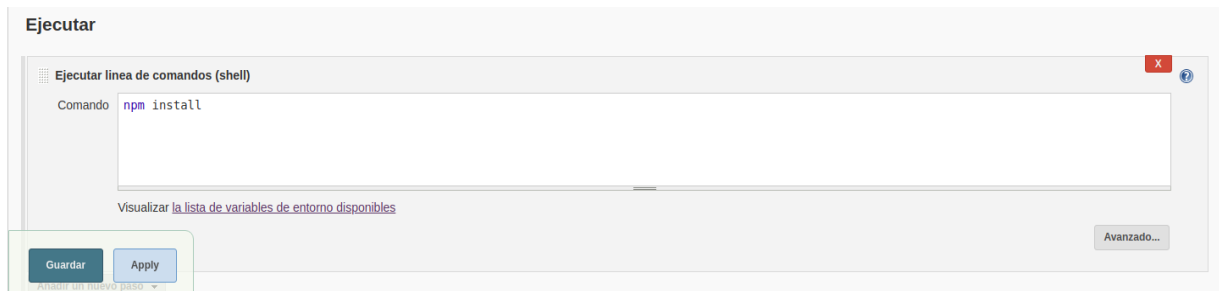


Figura 12

npm install

Ese comando se va a ejecutar cuando construyamos nuestro trabajo y es el encargado de descargar todos los módulos que se encuentran en el archivo package.json. Este es un ítem de configuración que contiene todas las dependencias de nuestro proyecto.

El trabajo ya creado se verá en nuestro panel principal como se muestra en la Figura 13

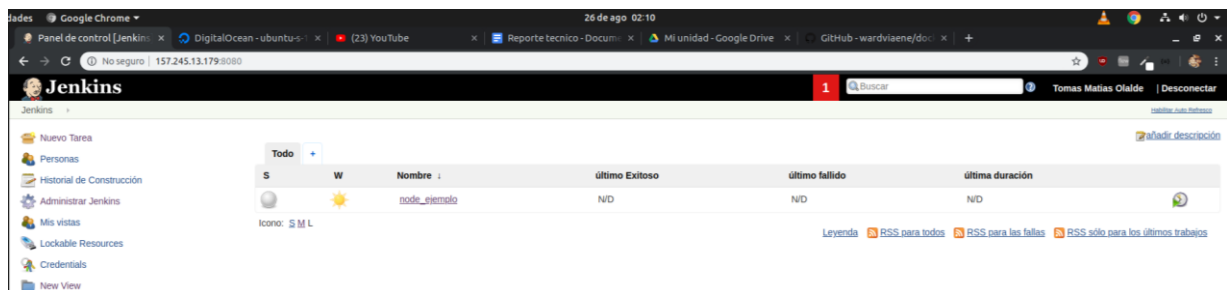


Figura 13

3.3 Instalación de plugins

Ahora con nuestro proyecto ya creado debemos instalar algunas extensiones para que nuestro proyecto quede listo para ser construido.

En el menú de opciones, en la sección Administrar Jenkins debemos ingresar a Administrar Plugins, como se ve en la Figura 14.



Figura 14

Una vez dentro debemos buscar los plugins para Node.JS y para Docker.

- Plugin para Node.Js: **NodeJS**
- Plugin para Docker: **CloudBees Docker Build and Publish plugin**

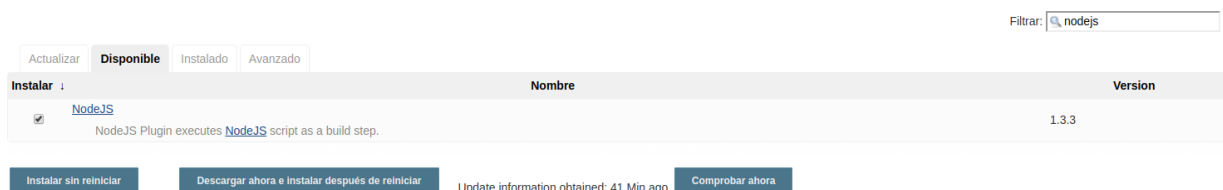


Figura 15

De esa manera los buscamos y seleccionamos *Descargar ahora e instalar después de reiniciar*. Esta opción hará que Jenkins se reinicie una vez termine de descargar e instalar los plugins.

4. Construcción del proyecto

Tenemos nuestra api ya creada, junto con su respectiva referencia en el proyecto de Jenkins, los plugins que se van a utilizar y las configuraciones de los comandos que queremos ejecutar, por lo que solo nos queda ejecutar el proyecto para que se descargen todas esas configuraciones en un proyecto de forma física en nuestro servidor.

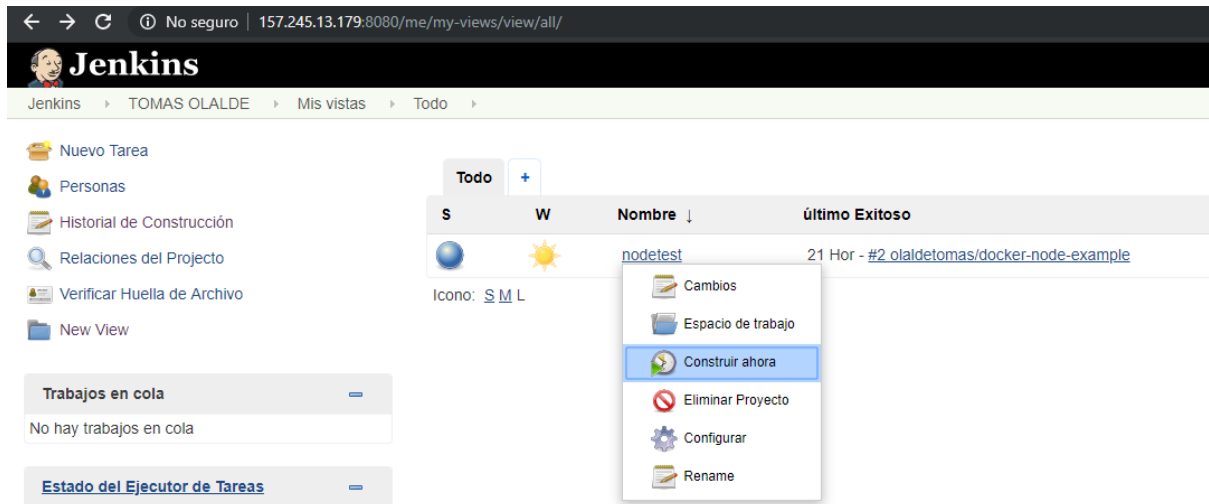


Figura 16

Se comenzará a generar nuestro proyecto y cuando finalice veremos la siguiente salida en la consola de jenkins.

Salida de consola

```
Started by user Tomas Matias Olalde
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/node_ejemplo
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/wardviaene/docker-demo.git
> git init /var/jenkins_home/workspace/node_ejemplo # timeout=10
Fetching upstream changes from https://github.com/wardviaene/docker-demo.git
> git --version # timeout=10
> git fetch --tags --progress https://github.com/wardviaene/docker-demo.git +refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url https://github.com/wardviaene/docker-demo.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/wardviaene/docker-demo.git # timeout=10
Fetching upstream changes from https://github.com/wardviaene/docker-demo.git
> git fetch --tags --progress https://github.com/wardviaene/docker-demo.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision e583f9bd4b2b44620bdf3b92b2054ab89ae8084 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f e583f9bd4b2b44620bdf3b92b2054ab89ae8084
Commit message: "fix dependency issue"
First time build. Skipping changelog.
Unpacking https://nodejs.org/dist/v12.9.0/node-v12.9.0-linux-x64.tar.gz to /var/jenkins_home/tools/jenkins.plugins.nodejs.
[node_ejemplo] $ /bin/sh -xe /tmp/jenkins5488228293931521731.sh
+ npm install
added 94 packages from 485 contributors and audited 171 packages in 4.084s
found 0 vulnerabilities

Finished: SUCCESS
```

Figura 17

Si observamos la Figura 17, en la salida por consola podemos ver la ruta en donde Jenkins creó nuestro proyecto y además, la ejecución del comando `npm install` que era el encargado de descargar las dependencias de nuestra api.

Ejecutando el siguiente comando en nuestro servidor podemos ver que efectivamente nuestro proyecto fue creado.

```
ls /var/jenkins_home/workspace/node_ejemplo
```

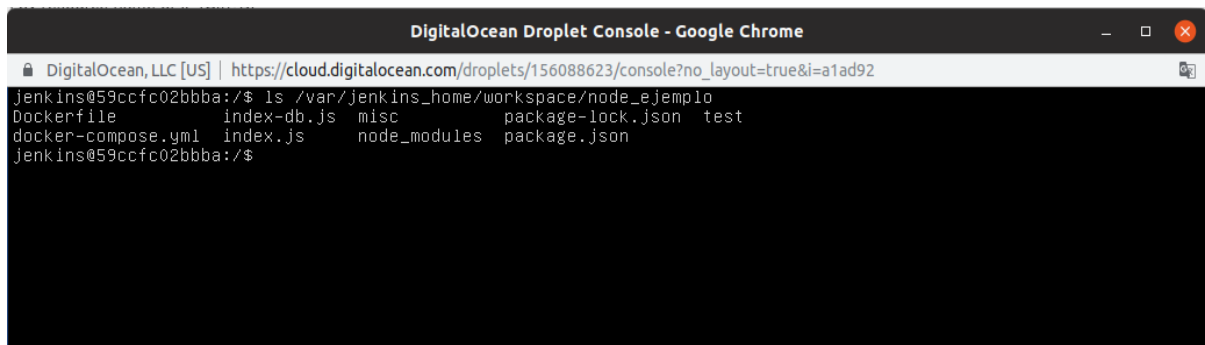


Figura 18

5. Crear contenedor de Docker

Para esta última parte lo vamos a desplegar nuestra api en un contenedor de Docker. Esta última parte podría ser evitada y correr nuestra api directamente en el servidor, pero creemos que es mucho más flexible si utilizamos un contenedor de Docker.

5.1 Contenedor Docker

Para este estudio ya tenemos un contenedor de Docker configurado el cual se encuentra en un repositorio:

Repositorio: <https://github.com/wardviaene/jenkins-docker>

Dentro del repositorio nos encontramos con un archivo llamado Dockerfile que contiene la configuración para nuestro contenedor. Ver Figura 19

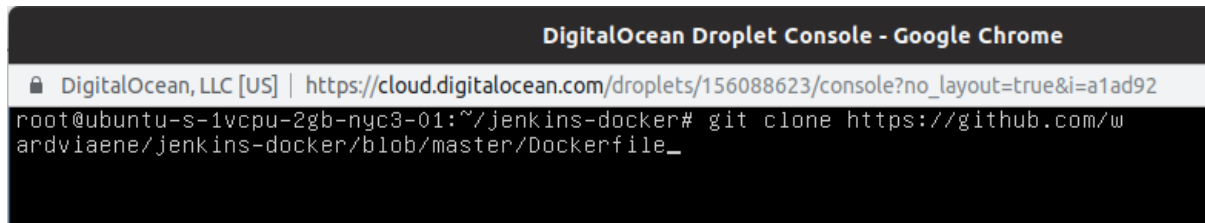


Figura 19

5.2 Clonar repositorio Docker

Para utilizar esta imagen debemos ir a la consola de comandos de nuestro servidor y clonar el repositorio, para poder contar con este archivo.

```
git clone https://github.com/wardviaene/jenkins-docker
```



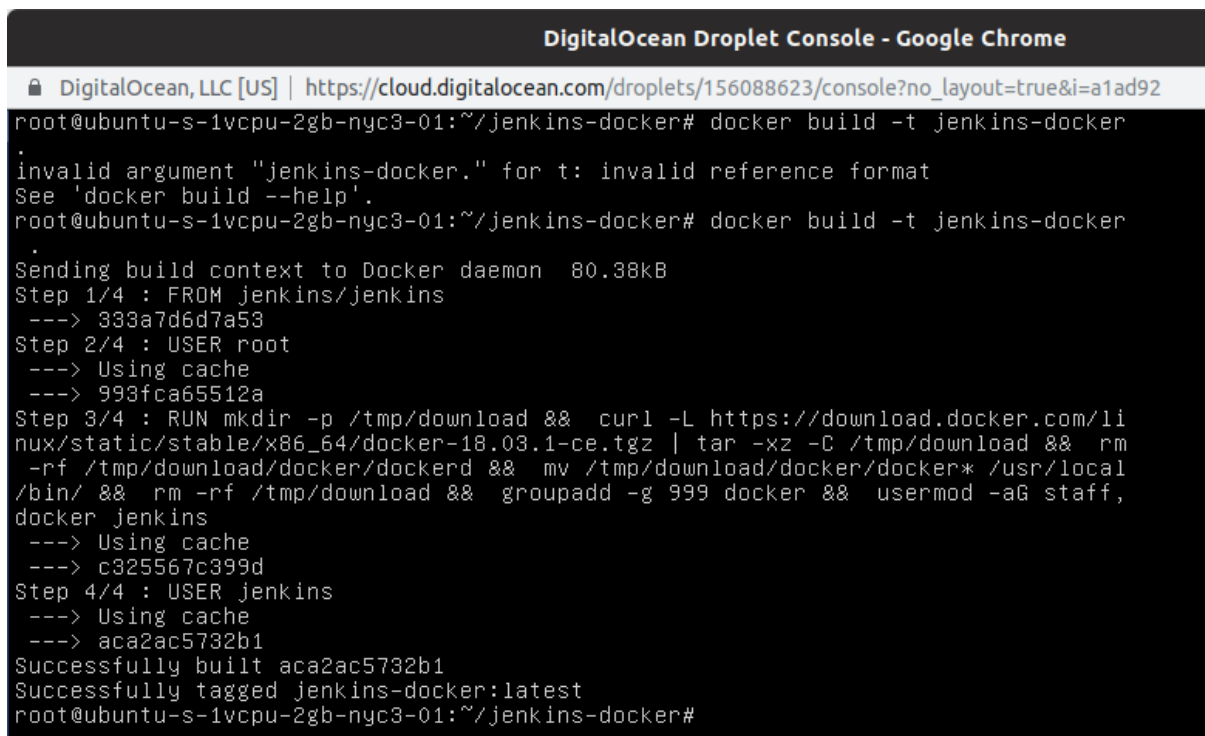
The screenshot shows a terminal window titled "DigitalOcean Droplet Console - Google Chrome". The address bar displays "DigitalOcean, LLC [US] | https://cloud.digitalocean.com/droplets/156088623/console?no_layout=true&i=a1ad92". The terminal prompt is "root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker#". The command entered is "git clone https://github.com/wardviaene/jenkins-docker/blob/master/Dockerfile_".

Figura 20

5.3 Construir proyecto Docker

Ya tenemos el archivo en nuestro servidor y también contamos con el Plugin de Docker antes instalado, por lo que podemos crear el contenedor de Docker con el siguiente comando.

```
docker build -t jenkins-docker
```



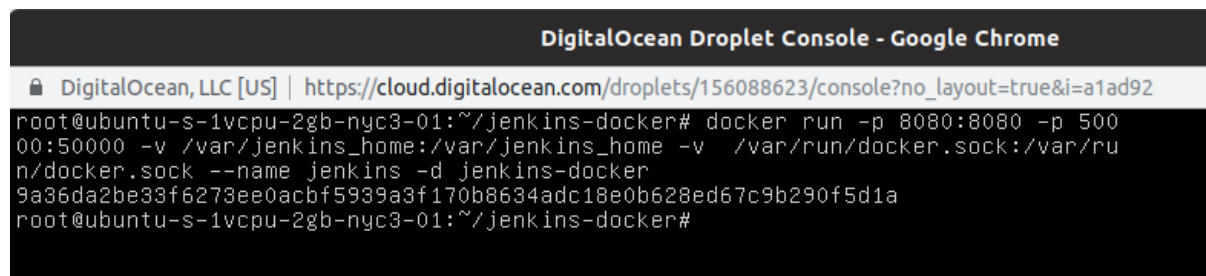
The screenshot shows a terminal window titled "DigitalOcean Droplet Console - Google Chrome". The address bar displays "DigitalOcean, LLC [US] | https://cloud.digitalocean.com/droplets/156088623/console?no_layout=true&i=a1ad92". The terminal prompt is "root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker#". The command entered is "docker build -t jenkins-docker". The output shows the build process: "Sending build context to Docker daemon 80.38kB", "Step 1/4 : FROM jenkins/jenkins", "Step 2/4 : USER root", "Step 3/4 : RUN mkdir -p /tmp/download && curl -L https://download.docker.com/linux/static/stable/x86_64/docker-18.03.1-ce.tgz | tar -xz -C /tmp/download && rm -rf /tmp/download/docker/dockerd && mv /tmp/download/docker/docker* /usr/local/bin/ && rm -rf /tmp/download && groupadd -g 999 docker && usermod -aG staff, docker jenkins", "Step 4/4 : USER jenkins", "Successfully built aca2ac5732b1", "Successfully tagged jenkins-docker:latest".

Figura 21

5.4 Ejecutar contenedor Docker

Ejecutamos el contenedor de Docker.

```
docker run -p 8080:8080 -p 50000:50000 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -d jenkins-docker
```



DigitalOcean Droplet Console - Google Chrome

DigitalOcean, LLC [US] | https://cloud.digitalocean.com/droplets/156088623/console?no_layout=true&i=a1ad92

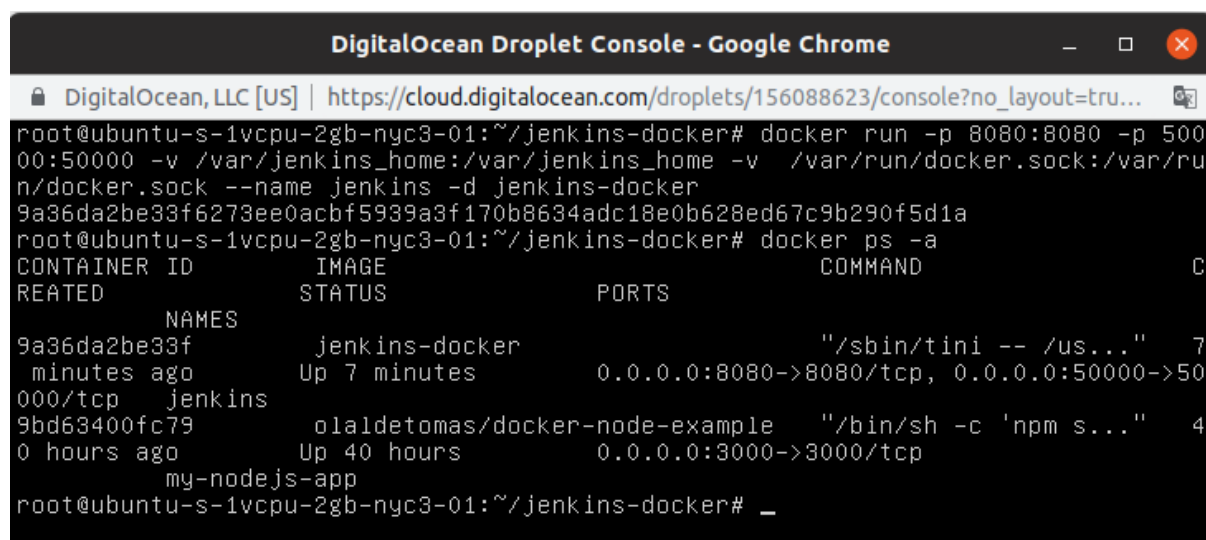
```
root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker# docker run -p 8080:8080 -p 50000:50000 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -d jenkins-docker
9a36da2be33f6273ee0acbf5939a3f170b8634adc18e0b628ed67c9b290f5d1a
root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker#
```

Figura 22

5.5 Verificar ejecución de Docker

Para ver si nuestro contenedor se está ejecutando correctamente podemos ejecutar el siguiente comando:

```
docker ps -a
```



DigitalOcean Droplet Console - Google Chrome

DigitalOcean, LLC [US] | https://cloud.digitalocean.com/droplets/156088623/console?no_layout=tru...

```
root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker# docker run -p 8080:8080 -p 50000:50000 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -d jenkins-docker
9a36da2be33f6273ee0acbf5939a3f170b8634adc18e0b628ed67c9b290f5d1a
root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker# docker ps -a
```

CONTAINER ID	IMAGE	STATUS	PORTS	COMMAND
9a36da2be33f	jenkins-docker	Up 7 minutes	0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000/tcp	"/sbin/tini -- /us..."
9bd63400fc79	olaldetomas/docker-node-example	Up 40 hours	0.0.0.0:3000->3000/tcp	"/bin/sh -c 'npm s..."

```
root@ubuntu-s-1vcpu-2gb-nyc3-01:~/jenkins-docker#
```

Figura 23

Si bien la consola que nos brinda DigitalOcean no reescala el texto cuando la expandimos, podemos ver que nuestro contenedor se está ejecutando correctamente.

En este punto ya tenemos nuestra api dentro de un contenedor de docker.

6. Crear Repositorio en Docker

Para poder desplegar nuestra aplicación necesitamos tener una cuenta en Docker y crear un repositorio donde será publicada nuestra api.

En caso de no tener una cuenta en Docker los pasos para crearla son muy sencillos, pueden verse en la página: <https://hub.docker.com/>

6.1 Configurar repositorio Docker

Para crear un repositorio nos dirigimos a la pestaña Repositorios y clickeamos en crear repositorio, como se ve en la Figura 24.

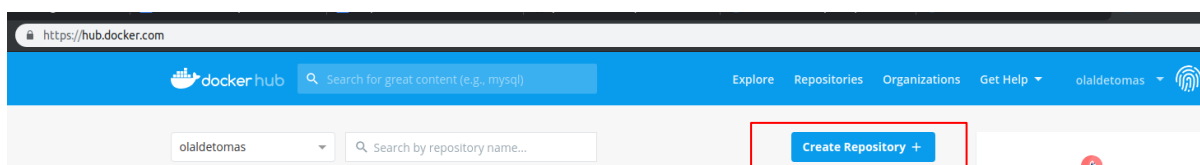


Figura 24

Elegimos un nombre, dejamos los parámetros por defecto y lo guardamos.

A screenshot of the 'Create Repository' form on Docker Hub. The form is titled 'Create Repository' and shows the repository name 'docker-node-example' under the user 'olaldetomas'. There is a 'Description' field. Under 'Visibility', the 'Public' option is selected. A 'Pro tip' box on the right shows CLI commands: 'docker tag local-image:tagname new-repo:tagname' and 'docker push new-repo:tagname'. Below the form, there are 'Build Settings (optional)' and a note about re-linking GitHub or Bitbucket accounts. At the bottom, there are buttons for 'Cancel', 'Create', and 'Create & Build'. The user's profile name 'olaldetomas' is visible in the top right corner.

Figura 25

6.2 Configuración de Docker dentro de Jenkins

Con el repositorio creado en Docker ahora necesitamos configurar en Jenkins la dirección del repositorio donde se va a desplegar la aplicación.

Vamos a nuestro proyecto y en el menú desplegable ingresamos en configurar. Ver Figura 26

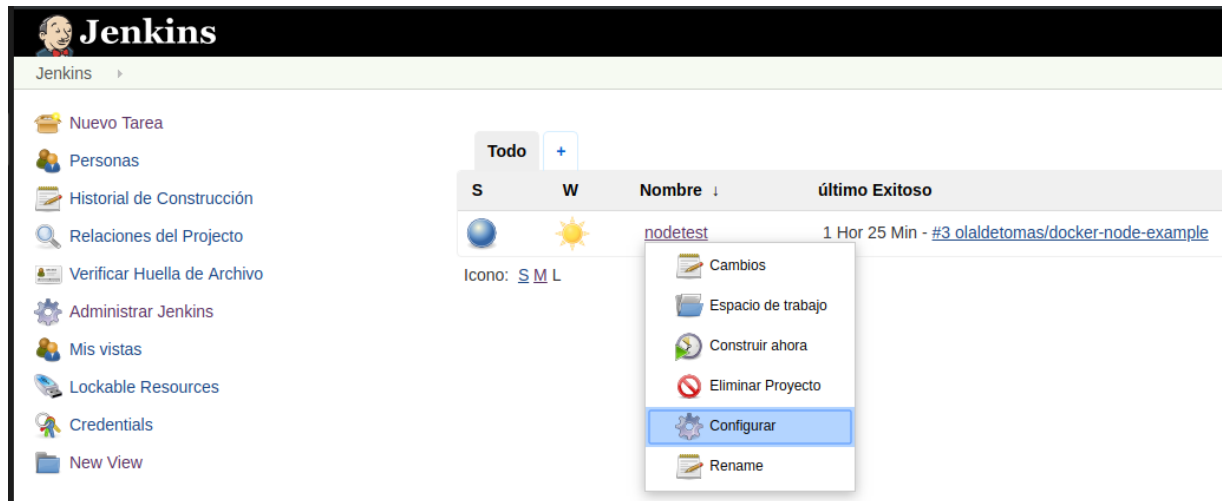


Figura 26

En la pestaña ejecutar, agregamos un nuevo paso. Ver Figura 26.

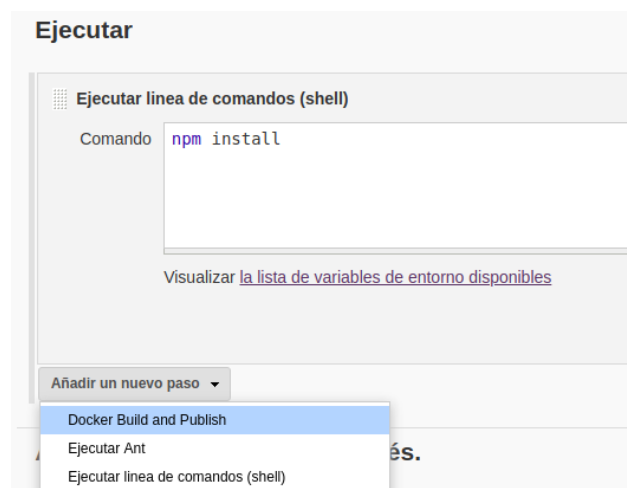


Figura 26

Ahora debemos agregar el nombre del repositorio que creamos en Docker, que está compuesto por nuestro nombre de usuario de Docker seguido del nombre del repositorio.

Figura 27

Como se muestra en la Figura 27, en la parte de Registrar Credenciales debemos agregar nuestro nombre de usuario y contraseña que utilizamos para conectarnos con Jenkins y guardamos los cambios.

7. Desplegar API

Ahora luego de las configuraciones del repositorio de docker, estamos en condiciones de desplegar nuestra simple api en un contenedor de Docker.

7.1 Construir proyecto en Jenkins.

Figura 28

Luego de construir el proyecto veremos en la consola que se ejecutó nuevamente *npm install* pero dentro del contenedor, y que se puede hacer push de la imagen al repositorio que creamos anteriormente. Ver Figura 28.

Salida de consola

```
Started by user TOMAS OLALDE
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/nodetest
No credentials specified
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/wardviaene/docker-demo # timeout=10
Fetching upstream changes from https://github.com/wardviaene/docker-demo
> git --version # timeout=10
> git fetch --tags --progress https://github.com/wardviaene/docker-demo +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision e583f9bd4b2b44620bdf3b92b2054ab89ae8084 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f e583f9bd4b2b44620bdf3b92b2054ab89ae8084
Commit message: "fix dependency issue"
> git rev-list --no-walk e583f9bd4b2b44620bdf3b92b2054ab89ae8084 # timeout=10
[nodetest] $ /bin/sh -xe /tmp/jenkins8036176879559841256.sh
+ npm install
audited 171 packages in 1.432s
found 0 vulnerabilities

[nodetest] $ docker build -t olaldetomas/docker-node-example --pull=true /var/jenkins_home/workspace/nodetest
Sending build context to Docker daemon 5.691MB

Step 1/6 : FROM node:4.6
4.6: Pulling from library/node
Digest: sha256:alcc6d576734c331643f9c4e0e7f572430e8baf9756dc24dab11d87b34bd202e
Status: Image is up to date for node:4.6
--> e834398209c1
Step 2/6 : WORKDIR /app
--> Using cache
--> 4b9a6d073aab
Step 3/6 : ADD . /app
--> Using cache
--> e53c2e03e7d4
Step 4/6 : RUN npm install
--> Using cache
--> ad724cd14d52
Step 5/6 : EXPOSE 3000
--> Using cache
--> 8d13f5071ec1
Step 6/6 : CMD npm start
--> Using cache
--> 10054d37c912
Successfully built 10054d37c912
Successfully tagged olaldetomas/docker-node-example:latest
[nodetest] $ docker inspect 10054d37c912
[nodetest] $ docker push olaldetomas/docker-node-example
The push refers to a repository [docker.io/olaldetomas/docker-node-example]
d03312d27aad: Preparing
b3573bc8ec35: Preparing
a7e41ffda2a0: Preparing
e1da644611ce: Preparing
d79093d63949: Preparing
87cbe568afdd: Preparing
787c930753b4: Preparing
9f17712cba0b: Preparing
223c0d04a137: Preparing
fe4c16cbf7a4: Preparing
87cbe568afdd: Waiting
787c930753b4: Waiting
9f17712cba0b: Waiting
223c0d04a137: Waiting
fe4c16cbf7a4: Waiting
d03312d27aad: Layer already exists
a7e41ffda2a0: Layer already exists
b3573bc8ec35: Layer already exists
e1da644611ce: Layer already exists
d79093d63949: Layer already exists
87cbe568afdd: Layer already exists
9f17712cba0b: Layer already exists
223c0d04a137: Layer already exists
fe4c16cbf7a4: Layer already exists
787c930753b4: Layer already exists
latest: digest: sha256:51d09280dfacd55c60b6c0bae2d8760239d89b736fe612049521fb1c577fe2cd size: 2420
Finished: SUCCESS
```

Figura 29

Para verificar que realmente la imagen fue pusheada con éxito podemos dirigirnos a nuestra cuenta de Docker y ver la última publicación de nuestra imagen. Ver Figura 29.

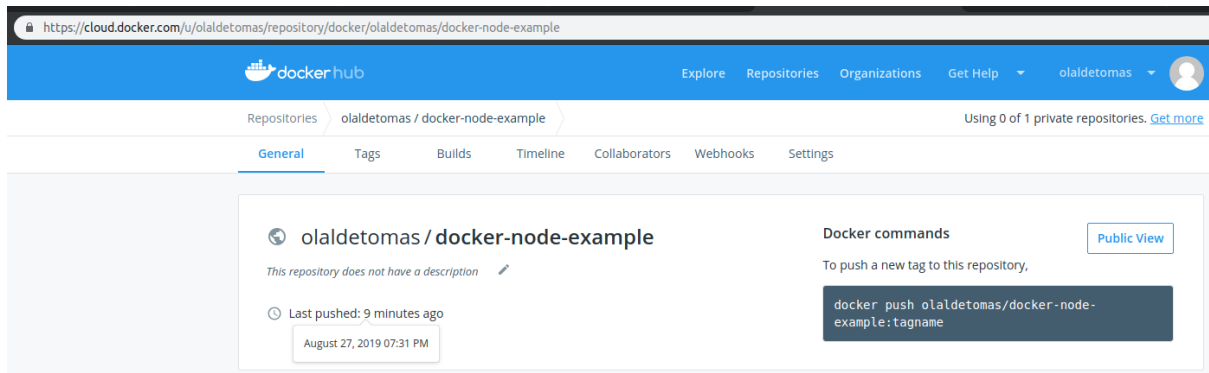


Figura 30

7.2 Testing despliegue de aplicación.

Para testear nuestra aplicación podemos hacerlo desde cualquier computadora pero nosotros vamos a hacer desde el servidor.

Para realizar este proceso ingresamos al servidor y hacemos pull de la imagen creada y la ejecutamos en el puerto 3000.

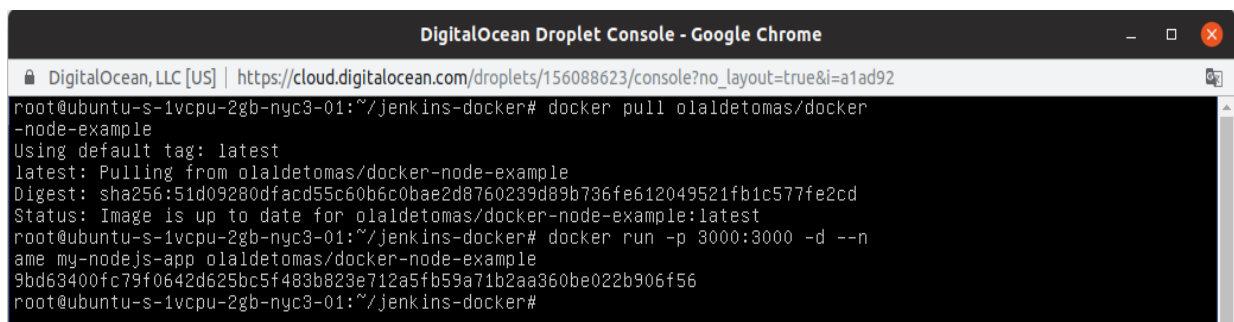


Figura 31

Ingresamos puerto 3000 de nuestro servidor:



Figura 32

RESULTADOS

Si bien los resultados se explican por sí solos daremos una breve explicación de qué fue lo que obtuvimos.

1. Servidor en la nube

Creamos un servidor donde se alojaron todas las herramientas que utilizamos, esto nos da una increíble flexibilidad para la configuración de estas herramientas ya que cualquier persona con las credenciales adecuadas puede ingresar a cualquiera de las herramientas utilizadas para hacer algún tipo de configuración desde cualquier computadora siempre que tenga conexión a internet.

2. Configuramos diferentes herramientas

Configuramos distintas herramientas como Git, GitHub, Jenkins y Docker que son básicas para la Integración Continua y si bien utilizamos apenas el 2% de cada herramienta, aprendimos cómo implementar todas las herramientas juntas lo que consideramos muy importante.

3. Ventajas de Integración Continua.

La calidad en un proyecto que implemente esta práctica se verá realmente incrementada, ya que si se configuran diferentes Test Cases dentro de las Pipelines de Jenkins cada desarrollador tendrá una verificación automática al momento de hacer un commit.

El proceso de despliegue de una aplicación es uno de los más tediosos dentro de muchos proyectos, pero en este caso sería un proceso constante y automático en el que prácticamente no se tendría que hacer nada más que la configuración inicial del mismo.

En caso de querer verificar que fue realmente creado el servidor puede ingresar a **<http://157.245.13.179:8080>** donde vera la pantalla de login de Jenkins. No se pueden brindar la credenciales utilizadas porque las cuentas van a ser utilizadas para seguir profundizando en el tema.

Las direcciones de los repositorios se encuentra a lo largo del desarrollo.

CONCLUSIÓN

El objetivo del reporte fue tener un enfoque práctico para mostrar qué es Integración Continua y poner a disposición de todos los integrantes del equipo una base sólida para continuar con futuros estudios.

En el desarrollo no se llegó a crear automatizaciones de prueba para la aplicación por cuestiones de tiempo, pero se llegó a entender muchos de los conceptos de Integración Continua y como seguir estudiando y en qué temas enfocarnos para realmente construir una implementación más completa y compleja que nos permita una aplicación efectiva en un proyecto real.

REFERENCIAS

[1] <https://jenkins.io/>

[2] <https://git-scm.com/>

[3] <https://www.paradigmadigital.com/dev/pipelines-de-jenkins-evolucion-continuous-delivery/>

[4] <https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html>

[5] <https://www.paradigmadigital.com/dev/pipelines-de-jenkins-evolucion-continuous-delivery/>

[6]

<https://www8.sunydutchess.edu/faculty/akins/documents/TechnicalReportWritingGuidelines.pdf>

[7] <https://docs.docker.com/>

[8] Documento de referencia para la confección del reporte técnico

<https://www8.sunydutchess.edu/faculty/akins/documents/TechnicalReportWritingGuidelines.pdf>