



Clase 33. Programación Backend

Productos Cloud: Heroku



OBJETIVOS DE LA CLASE

- Explorar y aplicar el concepto de control de versiones.
- Utilizar Git para tu proyecto.
- Conocer PaaS y Heroku y sus funciones.
- Implementar un proyecto de Node en Heroku.

CRONOGRAMA DEL CURSO

Clase 32



**Logs, profiling & debug
Parte II**

Clase 33



**Productos Cloud:
Heroku**

Clase 34



Productos Cloud: AWS

CONTROL DE VERSIONES



CONTROL DE VERSIONES

¿De qué se trata?

Es una manera de registrar los cambios realizados sobre un archivo (o conjunto de archivos) a lo largo del tiempo, permitiendo recuperar versiones específicas más adelante.



¿De qué se trata?

A lo largo del tiempo ha habido varias mejoras sobre cómo manejar estas versiones, llegando a lo que hoy se conoce como **Sistemas de Control de Versiones Distribuidos**.

Su idea parte de que cada desarrollador de un proyecto tenga una copia local de todo el proyecto. De esta manera se construye una red distribuida de repositorios, en la que cada desarrollador puede trabajar de manera aislada pero teniendo un mecanismo de resolución de conflictos mucho mejor que en versiones anteriores.



Sistemas de Control de Versiones Distribuidos



- Al no existir un repositorio central, cada desarrollador puede trabajar a su propio ritmo, almacenar los cambios a nivel local y mezclar los conflictos que se presenten sólo cuando se requiera.
- Como cada usuario tiene una copia completa del proyecto, el riesgo por una caída del servidor, un repositorio dañado o cualquier otro tipo de pérdida de datos es bastante bajo.



En resumen:



- Podemos volver a cualquier **estado anterior** de nuestro proyecto.
- Podemos tener una historia de cuáles fueron los cambios en el tiempo.
- Sobre estos podremos saber **cuándo, cómo y quién** los realizó.
- Permite la **colaboración** en un proyecto.
- Permite desarrollar **versiones** de un mismo proyecto a la vez.

GIT



Git es una herramienta para llevar a cabo el control de versiones. Es uno de los sistemas de control de versiones más utilizado.



Conceptos de Git

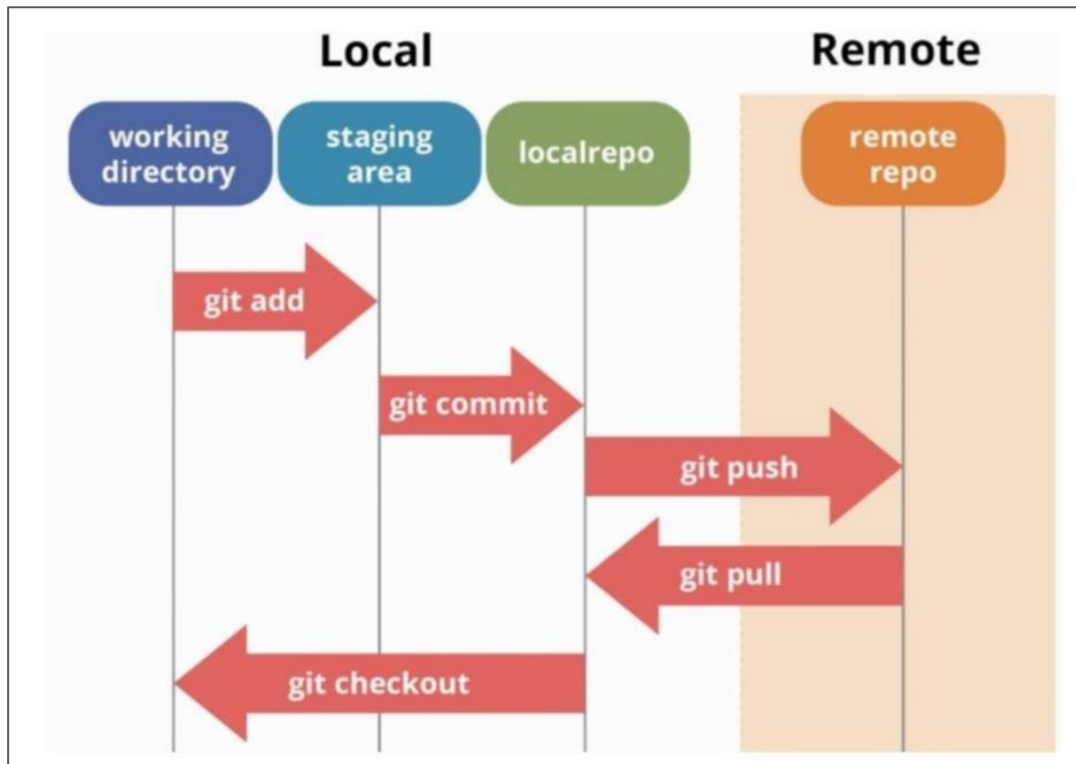
- **Repositorio remoto:** Es el lugar centralizado donde se guardan los archivos.
- **Repositorio local:** Es el lugar dentro de la computadora donde se guardan los archivos.
- **Working directory:** Copia del repositorio local (donde voy a empezar a trabajar).
- **Versión:** Captura del repositorio en un determinado momento.
- **Commit:** Modificaciones que le hacemos a los archivos del repositorio en nuestra computadora.



- **Tag:** Es una versión a la que le damos cierta importancia. Ej.: 1.0.2.
- **Push:** Registrar los commits en el repositorio REMOTO.
- **Pull:** Obtener los cambios en el repositorio REMOTO.
- **Conflicts:** Cuando dos o más personas modifican la misma línea de un archivo.
- **Resolución de conflicto:** Decidir cuál es la mejor versión que queremos del archivo modificado.
- **Branch:** secuencia de commits sucesivos, que conforman una ramificación en la línea temporal de un proyecto. Por convención tenemos una llamada 'master', aunque actualmente se está reemplazando por 'main', y puede haber otras más.



Esquema del flujo básico de Git





Instalación de Git

Ejemplo
en vivo



Para comenzar a usar Git, primero debemos instalarlo.

1. Para eso, descarga Git [para Windows](#) o [para Mac](#).
2. Luego, con el siguiente comando en consola podemos chequear que se haya instalado correctamente y la versión instalada.

```
$ git --version
```

3. Si se instaló bien, en consola nos debe salir algo parecido a esto:

```
git version 2.30.0.windows.2
```



Repositorio remoto en Github

Ejemplo
en vivo



Para trabajar con los repositorios en forma remota vamos a usar Github.

1. Para esto, debemos crearnos una cuenta en su página <https://github.com/>
2. Entramos a la página y hacemos click en el botón de registro (*sign up*).
3. Llenamos el formulario y ya tenemos nuestra cuenta de Github para subir los repositorios.

The screenshot shows the GitHub registration page. At the top, it says 'Join Github' and 'Create your account'. There are three input fields: 'Username *', 'Email address *', and 'Password *'. Below the password field, there is a note: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)'. There is a checkbox for 'Email preferences' with the text 'Send me occasional product updates, announcements, and offers.' Below this is a 'Verify your account' section with a large box containing the text 'Solucione este rompecabezas para que sepamos que es una persona real' and a 'Verificar' button. At the bottom of the form is a green 'Create account' button. Below the button, there is a small disclaimer: 'By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.'



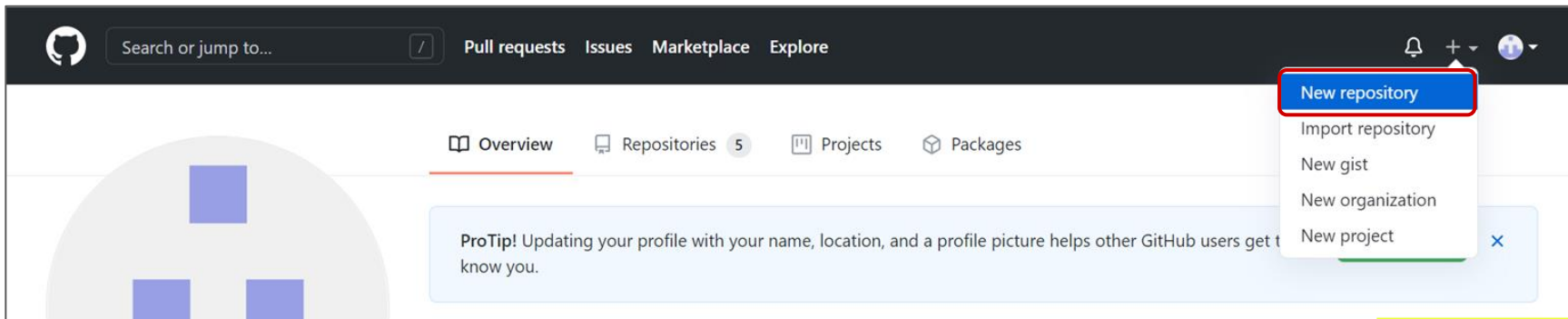
Repositorio remoto en Github

Ejemplo
en vivo



Podemos crear un repositorio directamente remoto en el Github, luego usarlo localmente y subir los cambios.

→ Para crear un repositorio remoto desde nuestra cuenta, vamos al botón “+” y elegimos “*New Repository*”.



CODER HOUSE



Repositorio remoto en Github

Ejemplo
en vivo





Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?



[Import a repository.](#)

Owner * / Repository name *

 username / nombre 

Great repository names are short and memorable. Need inspiration? How about [fuzzy-fiesta](#)?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)
- ☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)
- ☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

- Solo es necesario elegir el nombre del repositorio y ponerlo en “*Repository name*”.
- Los repositorios en general en Github los hacemos públicos.
- Se puede agregar un archivo README para poner especificaciones o explicaciones del proyecto que va a tener el repositorio.

CODER HOUSE



Crear un repositorio local

Ejemplo
en vivo



- Para crear un repositorio local, abrimos una consola en la carpeta en la que queremos que se cree y ponemos el comando:

```
$ git init
```

- Ya tenemos entonces una carpeta vacía en la que podemos agregar los archivos de nuestro proyecto y usar los comandos de Git.
- Para configurar este repositorio para poder luego subirlo a nuestra cuenta de Github, usamos los siguientes comandos de consola:

```
$ git config user.name "mi usuario de github"
```

```
$ git config user.email "mail de la cuenta de github"
```



Conectar repositorio local y remoto

Ejemplo
en vivo



- Para conectar el repositorio local creado con el que creamos remoto en Github, primero debemos obtener la url del repositorio remoto, abiriéndolo desde nuestra cuenta de Github. El link debe ser de la forma:

`https://github.com/username/repositoryName.git`

- De este modo, en consola, en la carpeta que tenemos el repositorio local, ponemos el comando:

```
$ git remote add origin https://github.com/username/repositoryName.git
```



Agregar archivos al repositorio

Ejemplo
en vivo



- Podemos agregar los archivos de nuestro proyecto al repositorio local con el siguiente comando:

→ Para agregar un solo archivo:

```
$ git add <nombre del archivo>
```

→ Para agregar todos los archivos que no estén ya en el repositorio o que hayan tenido cambios:

```
git add .
```

- Luego, debemos hacer un *commit* de los archivos agregados para generar un punto en la línea de tiempo y entonces poder volver a ese punto en caso de errores posteriores, es decir, controlar las versiones.

```
$ git commit -m "nombre del commit"
```

- Usamos el comando:



Subir y descargar archivos del repositorio remoto

Ejemplo
en vivo



- Luego, para subir los archivos que agregamos del repositorio local al remoto usamos el comando:

```
$ git push origin master
```

- Para descargar archivos al repositorio local que están en el repositorio remoto usamos el comando:

```
$ git pull origin master
```

👉 Pueden ser, por ejemplo, archivos subidos por otro usuario al mismo repositorio y entonces no lo tendremos en el local.

CODER HOUSE



Clonar un repositorio remoto

Ejemplo
en vivo



- Si estamos trabajando en equipo, uno de los integrantes puede crear el repositorio y subir al repositorio remoto los archivos iniciales del proyecto con los comandos que ya vimos.
- Si queremos tener luego ese repositorio localmente en nuestras computadoras, vamos a tener que clonarlo.
- Para eso, necesitamos la misma url del repositorio remoto que usamos anteriormente (del repositorio que nos queramos clonar) y usamos el siguiente comando en consola en la carpeta en la que queremos que esté el proyecto:

```
$ git clone https://github.com/username/repositoryName.git
```

TRABAJANDO CON BRANCHES EN GIT



¿Para qué son?

Las **ramas** (***branches***) son utilizadas para desarrollar **funcionalidades aisladas** unas de otras.

- La **rama *master*** es la rama principal y "por defecto" cuando creas un repositorio. Es la rama de producción que se suele usar solo para lo que ya esté listo.
- Luego, se suele tener una **rama** llamada ***dev***, en la cual se van fusionando las nuevas funcionalidades una vez que se desarrollan y se aprueba su correcto funcionamiento. Es la rama de desarrollo.
- Finalmente, se van creando **otras ramas** para ir desarrollando las distintas funcionalidades y no tener problemas por posibles errores que puedan surgir de estos cambios (*se hace de forma aislada*).



Trabajando con branches en Git

Ejemplo
en vivo



- Para crear una nueva rama y posicionarnos sobre ella en el repositorio usamos el `$ git checkout -b <nombre rama>`
- Para cambiar entre ramas ya creadas (master/main o cualquier otra) usamos el `$ git checkout <nombre rama>`
- Para ver el listado de todas las ramas del repositorio `$ git branch`
- Para borrar una rama creada usamos el comando `$ git branch -d <nombre rama>`



Subir y descargar de ramas

Ejemplo
en vivo



- Todos los comandos anteriores se ejecutan sobre el repositorio local.
- Para que la rama se cree luego en el repositorio remoto, hay que hacerle un push.

```
$ git push origin <nombre rama>
```
- Para actualizar el repositorio local en una rama en particular que se pudo haber actualizado en el remoto, nos posicionamos en la consola en el proyecto y vamos a la rama que queremos actualizar. Ahí ejecutamos

```
$ git pull
```

 do:



Fusionar ramas

Ejemplo
en vivo



- Una vez que terminamos la funcionalidad en una rama, debemos fusionarla a la rama de *Dev* o *Master*, según como estemos trabajando.
- También, nos puede pasar que para desarrollar una funcionalidad necesitamos funcionalidades de otra rama que todavía no fue fusionada a *Dev* o *Master*, entonces debemos fusionarla con la nuestra.
- Para hacer esto, necesitamos fusionar la rama a la que le queremos fusionar cosas, y ejecutamos el comando de **Merge**:

```
$ git merge <nombre rama que queremos fusionar>
```

CODER HOUSE



UTILIZAR GIT EN UN PROYECTO

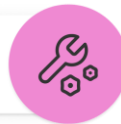
Tiempo: 20 minutos

⚠ Si aún no la tienes, corre a crearte una cuenta en [Github](#) (este desafío lo requiere).

CODER HOUSE

Utilizar Git en un proyecto

Desafío
generico

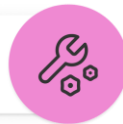


Tiempo: 20 minutos

- Crea una carpeta local llamada proyectoGit-persona_1.
- Inicializa un proyecto de git en esa carpeta.
- Crea un servidor en node.js y express dentro de esa carpeta que en su ruta raiz devuelva un html, el cual tendrá un título h1 con el contenido: 'Proyecto Node.js con git'. La página web tendrá una hoja de estilos externa que dará color azul a nuestro título y un script externo que represente el contenido del h1.
- Prueba el servidor en forma local.
- Crea el archivo .gitignore agregando la carpeta node_modules para no incluirla en el repo.
- Agrega los archivos en el stage area.

Utilizar Git en un proyecto

Desafío
generico



Tiempo: 20 minutos

- Commitea los cambios en el repo local con un mensaje que represente la versión inicial: ej: v0.0.1.
- Crea un repo en github llamado proyectoGit.
- Agrega el repo remoto al proyecto de git.
- Sube el repo local al remoto.
- Revisa si en el repo remoto está el código subido con la versión que pusimos en el commit.
- Clona el repo remoto en una carpeta llamada proyectoGit-persona_2
- Restaura las dependencias y verificar que todo funcione en esta carpeta.
- Agrega un subtítulo h2 al documento que diga: '¡Bienvenidos!'.

Utilizar Git en un proyecto

Desafío
generico



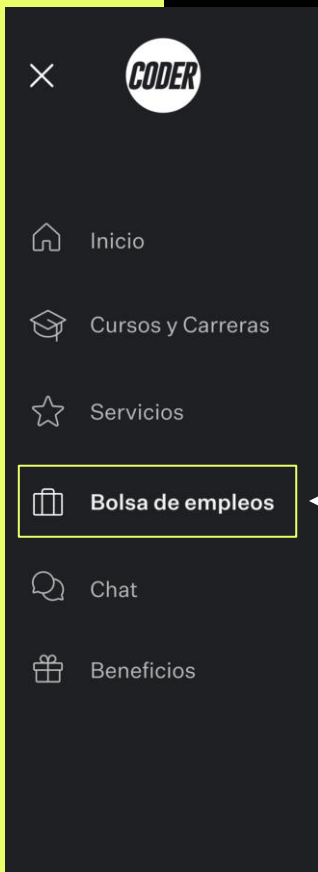
Tiempo: 20 minutos

- Realiza todo lo necesario para subir los cambios al repo remoto desde esta carpeta, con un mensaje de revisión: v0.0.2.
- Verifica en github el código subido con la nueva revisión.
- En la carpeta proyectoGit-persona_1 realiza un pull comprobando que el código quede igual al de la carpeta proyectoGit-persona_2 (sincronizamos el repo local con el remoto).



BREAK

¡5/10 MINUTOS Y VOLVEMOS!



Nuevo

¡Lanzamos la Bolsa de Empleos!

Un espacio para seguir **potenciando tu carrera** y que tengas más **oportunidades de inserción laboral**.

Podrás encontrar la **Bolsa de Empleos** en el menú izquierdo de la plataforma.

Te invitamos a conocerla y ¡postularte a tu futuro trabajo!

Conócela

PAAS



¿Qué es?



- Paas (plataforma como servicio) es un entorno de desarrollo e implementación completo en la nube.
- Cuenta con recursos que permiten generar “de todo”: desde aplicaciones sencillas basadas en la nube hasta aplicaciones empresariales sofisticadas habilitadas para la nube.
- Se compran los recursos que necesitamos a un proveedor de servicios en la nube, a los que accedemos a través de internet, pero solo pagamos por el uso que hacemos de ellos.
- PaaS incluye infraestructura (servidores, almacenamiento y redes), tanto como middleware, herramientas de desarrollo, servicios de inteligencia empresarial (BI), sistemas de administración de bases de datos, etc.



- Está diseñado para sustentar el ciclo de vida completo de las aplicaciones web: compilación, pruebas, implementación, administración y actualización.
- Nos permite evitar el gasto y la complejidad que suponen la compra y la administración de licencias de software, la infraestructura de aplicaciones y el middleware subyacentes, los orquestadores de contenedores o las herramientas de desarrollo y otros recursos.
- Administramos las aplicaciones y los servicios que desarrollamos y, normalmente, el proveedor de servicios en la nube administra todo lo demás.



Escenarios habituales de uso de PaaS



- **Framework de desarrollo.** PaaS proporciona un framework que los desarrolladores pueden ampliar para desarrollar o personalizar aplicaciones basadas en la nube. Permite a los desarrolladores crear aplicaciones usando componentes de software integrados.
- **Análisis o inteligencia empresarial.** Las herramientas que se proporcionan en PaaS permiten a las empresas realizar análisis de datos, obtener cierta información y entonces poder predecir los resultados para mejorar las previsiones de la empresa.
- **Servicios adicionales.** Los proveedores de PaaS pueden ofrecer otros servicios que mejoren las aplicaciones, como flujo de trabajo, directorios, seguridad y programación.



Ventajas del uso de PaaS



- **Reducir el tiempo de programación.** Las herramientas de desarrollo de PaaS pueden reducir el tiempo que se tarda en programar aplicaciones nuevas con componentes de aplicación preprogramados.
- **Agregar más funcionalidad de desarrollo sin incorporar más personal.** Los componentes de plataforma como servicio pueden aportar al equipo de desarrollo nuevas características sin necesidad de contratar personal especializado.
- **Desarrollar para varias plataformas con más facilidad.** Algunos proveedores de servicios ofrecen opciones de desarrollo para varias plataformas, lo que agiliza y facilita el desarrollo de aplicaciones multiplataforma.



Ventajas del uso de PaaS



- **Usar herramientas sofisticadas a un precio asequible.** Gracias a un modelo de pago por uso, personas y empresas podemos usar software de desarrollo sofisticado y herramientas de inteligencia empresarial y análisis cuya compra no se podrían permitir.
- **Colaboración en equipos de desarrollo distribuidos geográficamente.** Como se accede a través de Internet, los equipos de desarrollo pueden colaborar en proyectos estando en países diferentes.
- **Administrar el ciclo de vida de las aplicaciones con eficacia.** PaaS proporciona todas las características necesarias para sustentar el ciclo de vida completo de las aplicaciones web: compilación, pruebas, implementación, administración y actualización, dentro del mismo entorno integrado.

HEROKU

CODER HOUSE

¿Qué es?



- Heroku es una plataforma en la nube que ofrece servicio para alojar e implementar aplicaciones web en varios lenguajes de programación, como Node.js, entre otros.
- Las aplicaciones se corren desde un servidor Heroku usando Heroku DNS Server para apuntar al dominio de la aplicación (nombreaplicacion.herokuapp.com).
- Cada aplicación corre sobre un motor a través de una “red de bancos de prueba” que consta de varios servidores. El servidor Git de Heroku maneja los repositorios de las aplicaciones que son subidas por los usuarios.

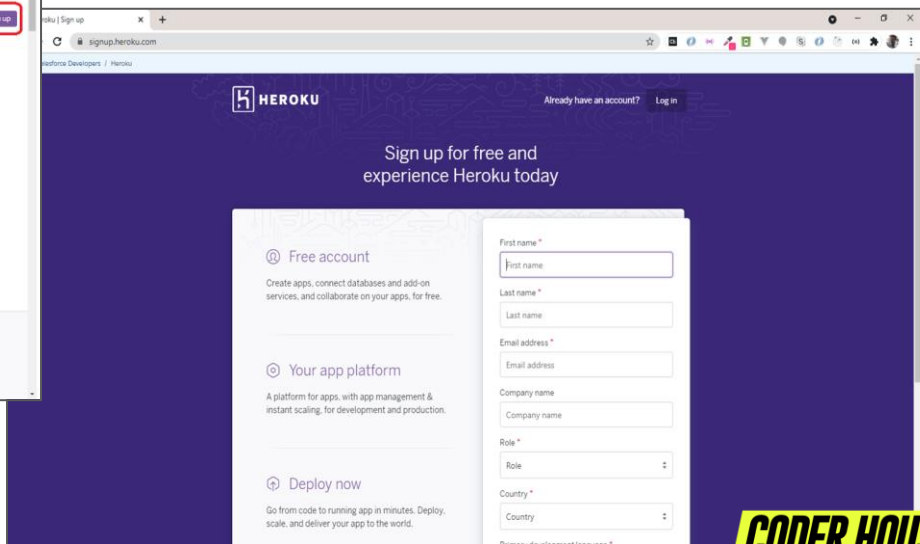
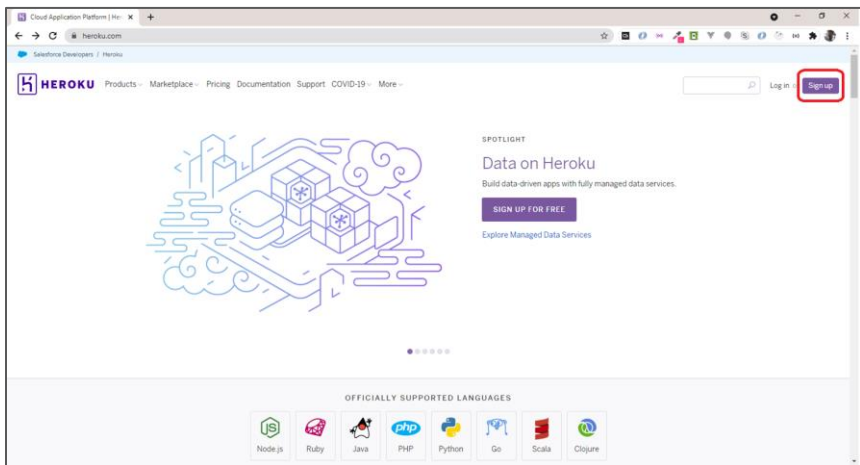


Empezar a usar Heroku

Ejemplo
en vivo



Para empezar a usar Heroku, primero debemos hacer es crear una cuenta en [Heroku login](#).



CODER HOUSE



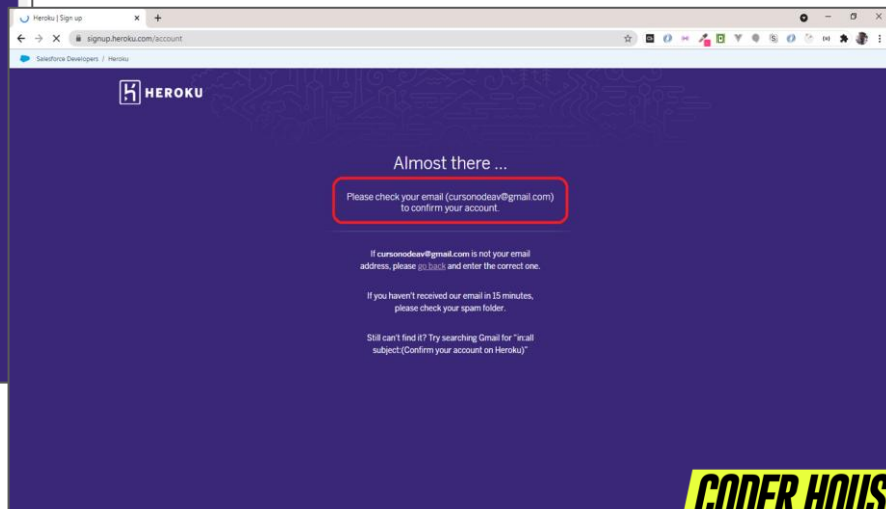
Empezar a usar Heroku

Ejemplo
en vivo



Llenamos el formulario de registro seleccionando Node como lenguaje de desarrollo.

The screenshot shows the Heroku sign-up page. On the left, there's a sidebar with 'Your app platform' and 'Deploy now' sections. The main form on the right includes fields for 'Last name', 'Email address', 'Company name', 'Role', 'Country', and 'Primary development language'. A red arrow points to the 'Primary development language' dropdown, which is currently set to 'Node.js'. Below the form, there's a checkbox for 'I'm not a robot' and a blue 'CREATE FREE ACCOUNT' button. At the bottom, there's a link to the 'Terms of Service' and 'Privacy Policy'.



CODER HOUSE

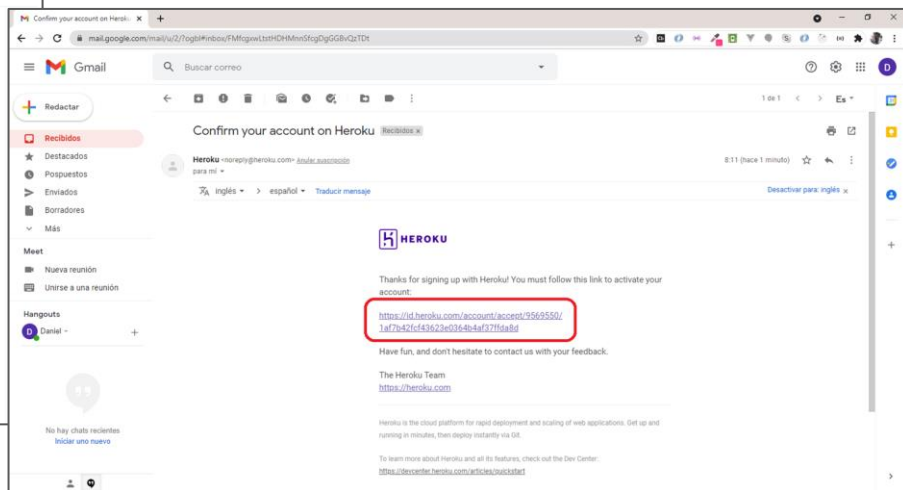
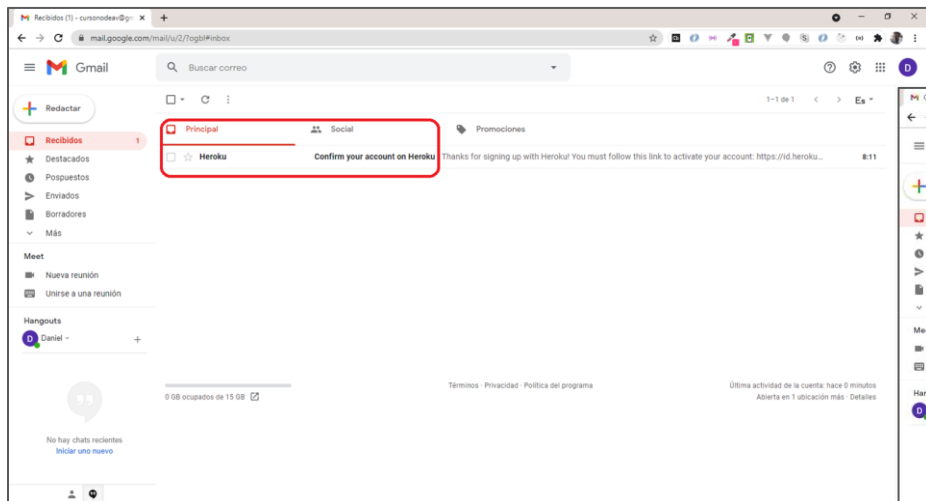


Empezar a usar Heroku

Ejemplo
en vivo



Vamos al mail con el que nos registramos, y confirmamos el mismo con el link que nos envía Heroku.



CODER HOUSE



Empezar a usar Heroku

Ejemplo
en vivo



Debemos ahora elegir una contraseña, y luego podremos iniciar sesión.

Confirm your account on Heroku | Heroku | Welcome

signup.heroku.com/confirm

Set your password

Create your password and log in to your Heroku account.

Create a new password *

New password

Password confirmation *

Confirm new password

Password requirements

- Must be a minimum of 8 characters.
- Must contain letters, numbers, and symbols.
- Passwords must match.

SET PASSWORD AND LOG IN

Confirm your account on Heroku | Heroku | Welcome

signup.heroku.com/confirm

Set your password

Create your password and log in to your Heroku account.

Create a new password *

Password confirmation *

Password requirements

- ✓ Must be a minimum of 8 characters.
- ✓ Must contain letters, numbers, and symbols.
- ✓ Passwords must match.

SET PASSWORD AND LOG IN

CODER HOUSE

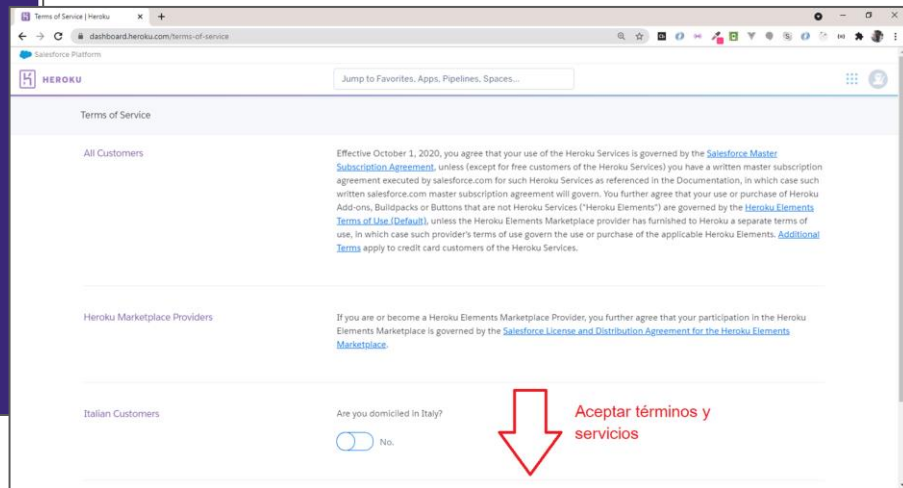
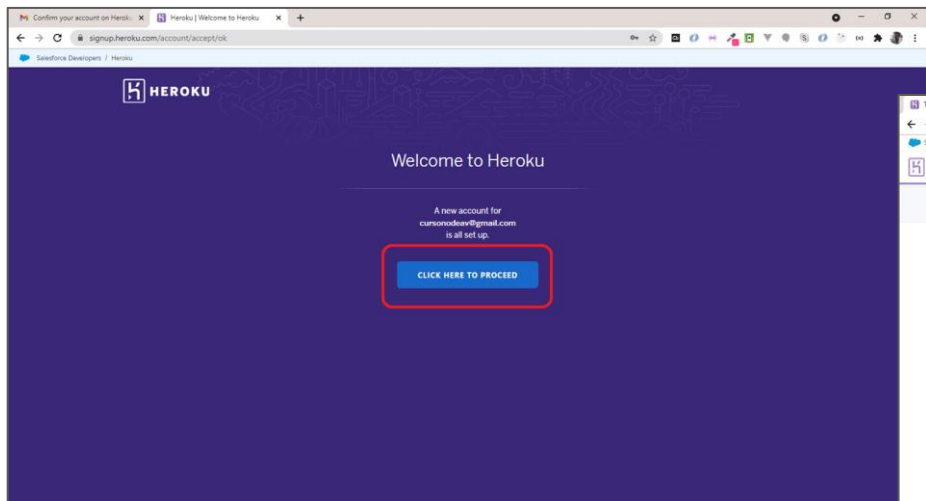


Empezar a usar Heroku

Ejemplo
en vivo



Luego tenemos que aceptar los términos y condiciones.



CODER HOUSE

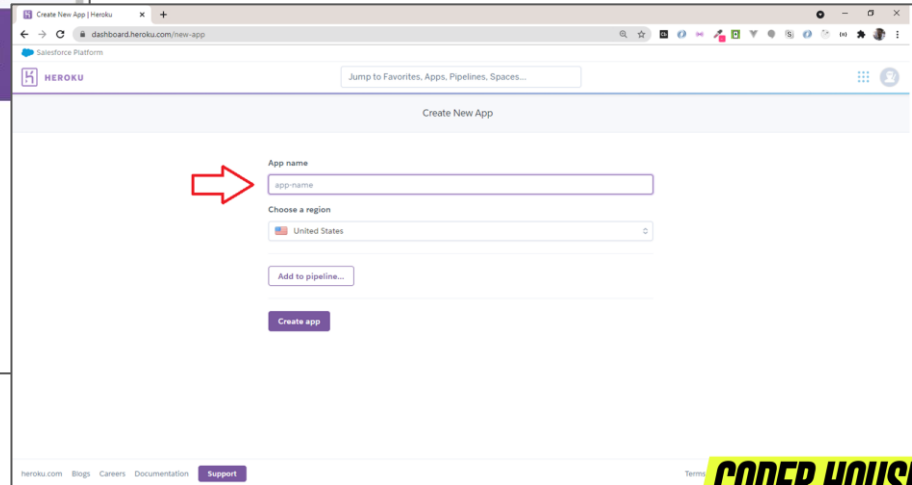
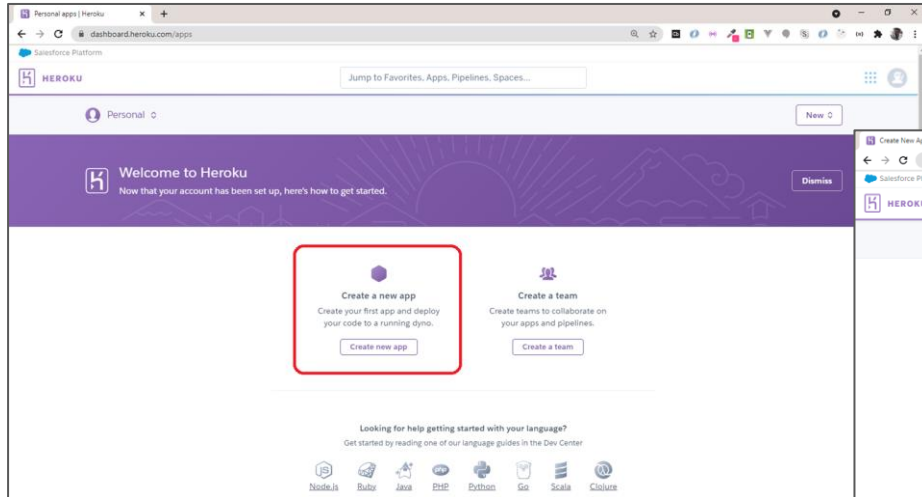


Crear una App en Heroku

Ejemplo
en vivo



Ya podemos ahora crear nuestra primera App en Heroku. Elegimos el nombre y la región y la creamos. En este caso la App se llama “serverch”.



CODER HOUSE

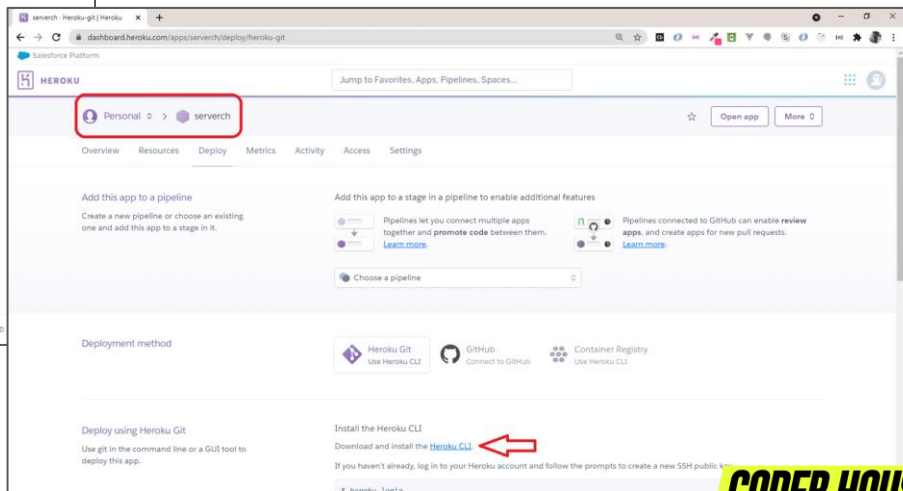
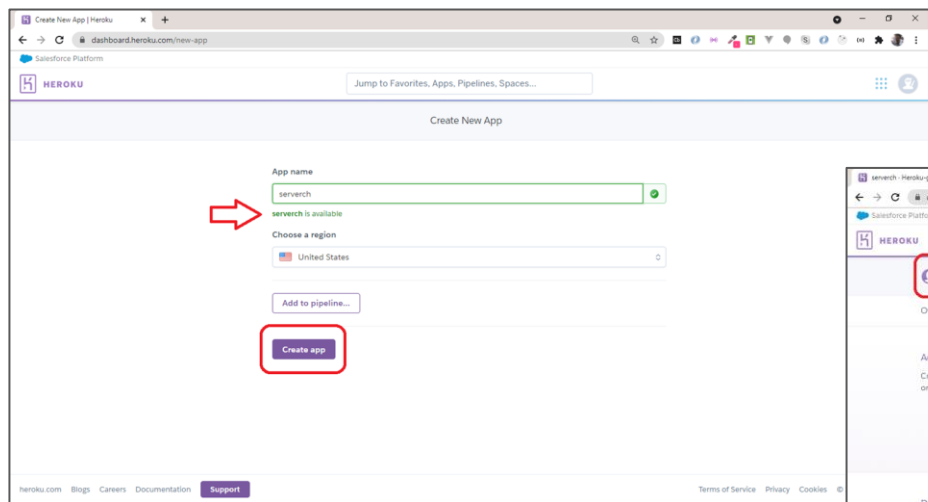


Crear una App en Heroku

Ejemplo
en vivo



Luego, debemos instalar *Heroku CLI* con el link que nos aparece abajo donde indica la flecha.



CODER HOUSE



Crear una App en Heroku

Ejemplo
en vivo



Use git in the command line or a GUI tool to deploy this app.

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/  
$ git init  
$ heroku git:remote -a serverch
```

Deploy your application

Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add -  
$ git commit -am "make it better"  
$ git push heroku master
```

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Existing Git repository

For existing repositories, simply add the `heroku` remote

```
$ heroku git:remote -a serverch
```

Heroku Dev Center

Command Line / The Heroku CLI

The Heroku CLI

Last updated April 27, 2021

Table of Contents

- Download and install
- Other installation methods
- Verifying your installation
- Getting started
- Staying up to date
- Useful CLI plugins
- CLI architecture
- Troubleshooting
- Uninstalling the Heroku CLI

The Heroku Command Line Interface (CLI) makes it easy to create and manage your Heroku apps directly from the terminal. It's an essential part of using Heroku.

Download and install

CODER HOUSE



Crear una App en Heroku

Ejemplo
en vivo



De esta forma descargamos el Heroku CLI y luego lo instalamos.

The Heroku CLI | Heroku Dev Co

devcenter.heroku.com/articles/heroku-cli

Accounts & Billing
Troubleshooting & Support

The Heroku Command Line Interface (CLI) makes it easy to create and manage your Heroku apps directly from the terminal. It's an essential part of using Heroku.

Download and install

The Heroku CLI requires Git, the popular version control system. If you don't already have Git installed, complete the following before installing the CLI:

- Git installation
- First-time Git setup

Currently, the Windows installers may display a warning titled "Windows protected your PC". To run the installation when this warning is shown, click "More info", verify the publisher as "salesforce.com, inc.", then click the "Run anyway" button.

macOS

```
$ brew tap heroku/brew && brew install heroku
```

Windows

Download the appropriate installer for your Windows installation:

- 64-bit installer
- 32-bit installer

macOS

```
$ brew tap heroku/brew && brew install heroku
```

Windows

Download the appropriate installer for your Windows installation:

- 64-bit installer
- 32-bit installer

Ubuntu 16+

Run the following from your terminal:

```
curl -fsSL https://raw.githubusercontent.com/heroku/heroku-cli/master/install.sh | sh
```

heroku@64bit

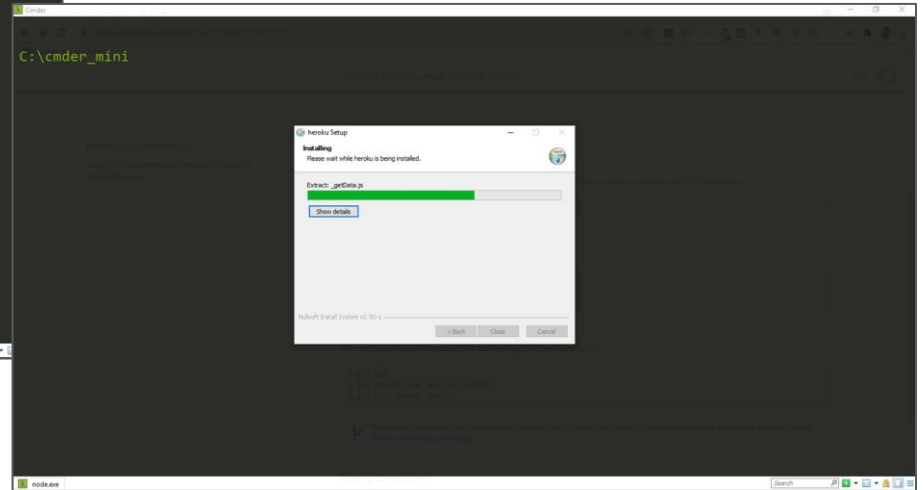
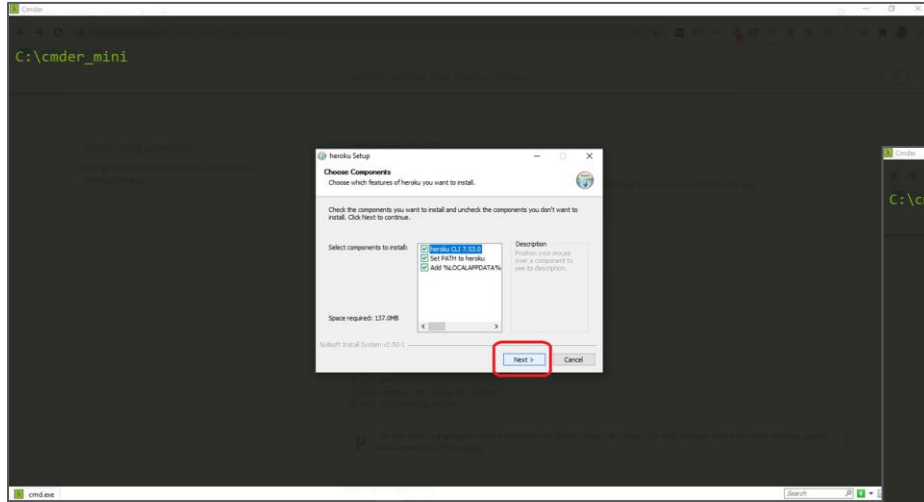
Monitor tools

CODER HOUSE



Crear una App en Heroku

Ejemplo
en vivo



CODER HOUSE



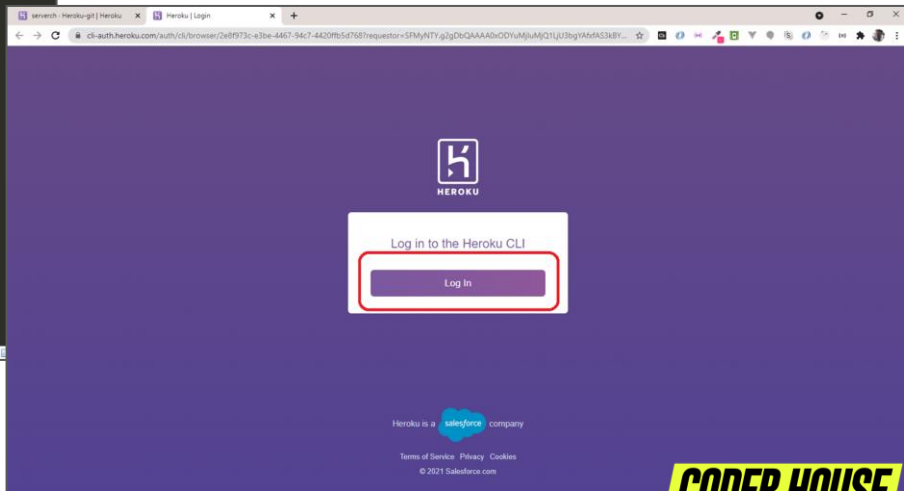
Crear una App en Heroku

Ejemplo
en vivo



En consola, ponemos el comando `$ heroku login` y luego hacemos el login desde la página.

```
C:\cmdr_mini
λ heroku login
» Warning: Our terms of service have changed: https://dashboard.heroku.com/terms-of-service
heroku: Press any key to open up the browser to login or q to exit: |
```



CODER HOUSE

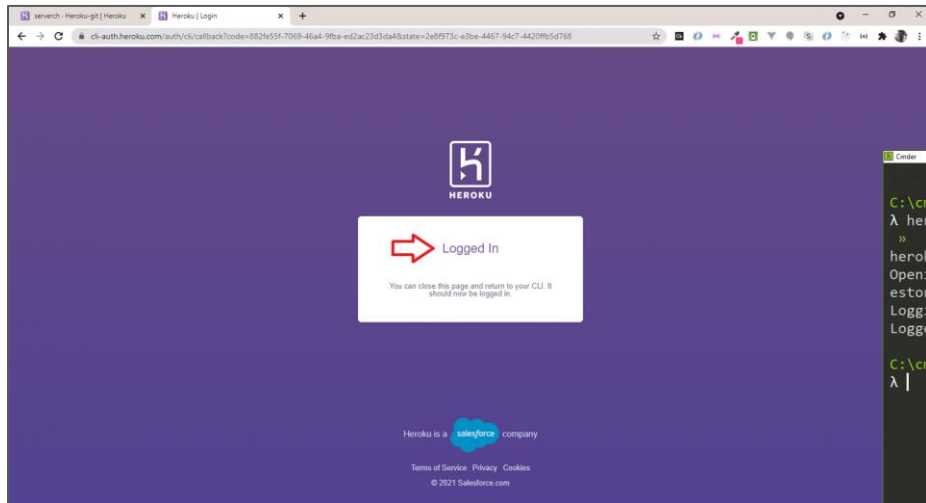


Crear una App en Heroku

Ejemplo
en vivo



Una vez hecho el login en la página, vemos que en consola lo toma y se loguea.



```
C:\cmdr_mini
λ heroku login
» Warning: Our terms of service have changed: https://dashboard.heroku.com/terms-of-service
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/2e8f973c-e3be-4467-94c7-4420ffb5d768?requestor=SFMyNTY.g2gDbQAAAA8xODYuMjIuMjQ1LjU3bgYAfxfAS3kBYgABUYA.kwhWXSaoNN3nSRG5RIbs0YY8jGz8WsfBC3vYzL6Pp8
Logging in... done
Logged in as cursonodeav@gmail.com

C:\cmdr_mini
λ |
```

CODER HOUSE



Crear una App en Heroku

Ejemplo
en vivo



- Vamos a usar Git para el proyecto, por lo que iniciamos git en la carpeta del proyecto, y usamos luego todos los comandos express que se muestran.
- Y en el proyecto, hacemos un server y una ruta que simplemente mande un mensaje.

Paso 1 →

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the [Heroku CLI](#)

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/  
$ git init  
$ heroku git:remote -a serverch
```

Deploy your application

Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add .  
$ git commit -am "make it better"  
$ git push heroku master
```

Paso 2 →

```
const express = require('express')  
const app = express()  
  
app.get('/mensaje', (req,res) => {  
  res.send('Hola Node.js desde Heroku!');  
})  
  
const PORT = process.env.PORT || 8080  
const server = app.listen(PORT, () => {  
  console.log(`Servidor express escuchando en el puerto ${PORT}`)  
})  
server.on('error', error => console.log(`Error en servidor ${error}`))
```

CODER HOUSE

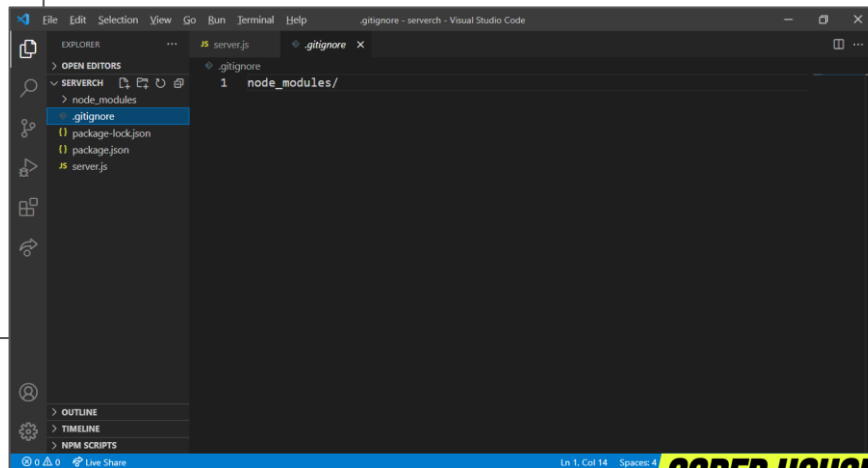
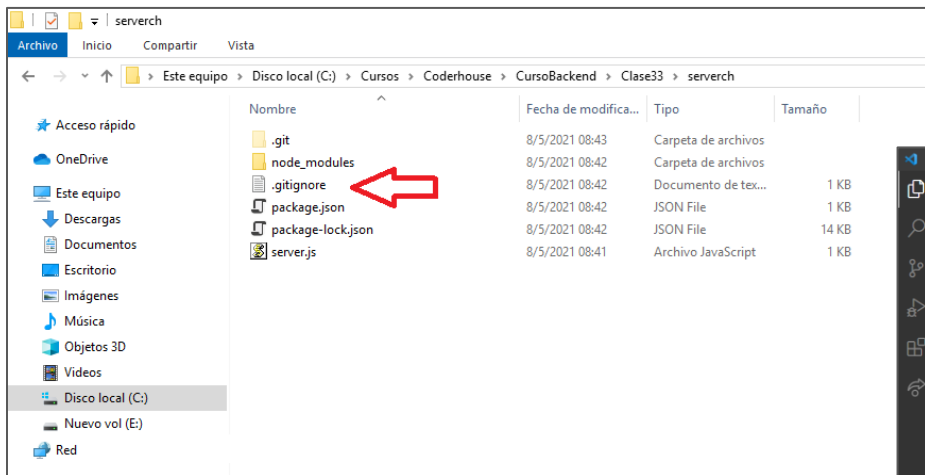


Crear una App en Heroku

Ejemplo
en vivo



Creamos el archivo **.gitignore**. Lo utilizamos para poner todo lo que no queremos que se suba a Git, como por ejemplo la carpeta `node_modules`.





Crear una App en Heroku

Ejemplo
en vivo



Seguimos estos pasos de comandos en la consola:

```
C:\cmdr_mini
λ cd C:\Cursos\Coderhouse\CursoBackend\Clase33

C:\Cursos\Coderhouse\CursoBackend\Clase33
λ cd serverch\

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (serverch@1.0.0)
λ ls
node_modules/ package.json package-lock.json server.js

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (serverch@1.0.0)
λ ls -a
./ ../ .gitignore node_modules/ package.json package-lock.json server.js

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (serverch@1.0.0)
λ git init
Initialized empty Git repository in C:/Cursos/Coderhouse/CursoBackend/Clase33/serverch/.git/

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ ls -a
./ ../ .git/ .gitignore node_modules/ package.json package-lock.json server.js

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ heroku git:remote -a serverch
set git remote heroku to https://git.heroku.com/serverch.git

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
```

Paso 1



```
C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ git add .
warning: LF will be replaced by CRLF in package-lock.json.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in package.json.
The file will have its original line endings in your working directory

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ git commit -am "v0.0.1"
[master (root-commit) 22ced5b] v0.0.1
4 files changed, 403 insertions(+)
create mode 100644 .gitignore
create mode 100644 package-lock.json
create mode 100644 package.json
create mode 100644 server.js

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ git push heroku master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 5.02 KiB | 5.02 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
```

Paso 2



CODER HOUSE



Crear una App en Heroku

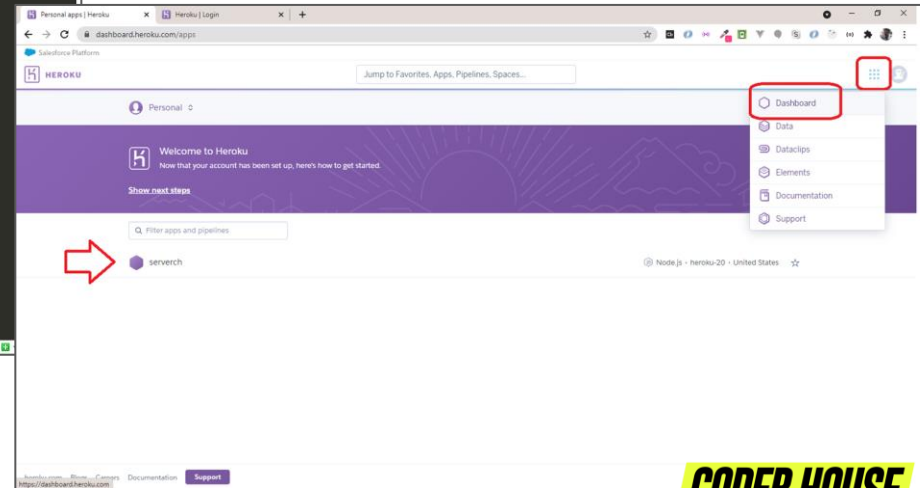
Ejemplo
en vivo



Una vez que terminamos los pasos en la consola, vamos al dashboard de Heroku y podemos ver nuestro proyecto subido.

```
remote: ----- Build
remote: ----- Caching build
remote:      - node_modules
remote: ----- Pruning devDependencies
remote:      audited 50 packages in 1.02s
remote:      found 0 vulnerabilities
remote: ----- Build succeeded!
remote: ----- Discovering process types
remote:      Procfile declares types --> (none)
remote:      Default types for buildpack --> web
remote: ----- Compressing...
remote:      Done: 32.6M
remote: ----- Launching...
remote:      Released v3
remote:      https://serverch.herokuapp.com/ deployed to Heroku
remote: Verifying deploy... done.
To https://git.heroku.com/serverch.git
 * [new branch]      master -> master

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ
```



CODER HOUSE

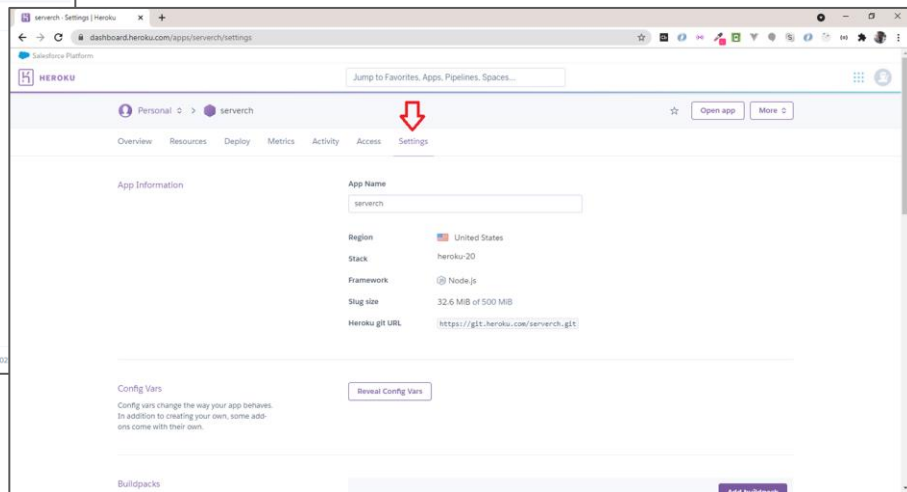
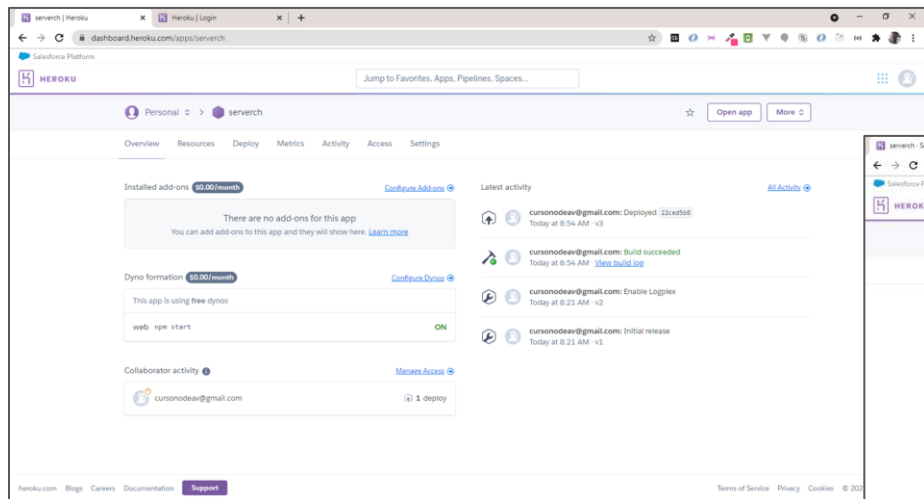


Crear una App en Heroku

Ejemplo
en vivo



Abrimos el proyecto y vamos a las configuraciones.



CODER HOUSE

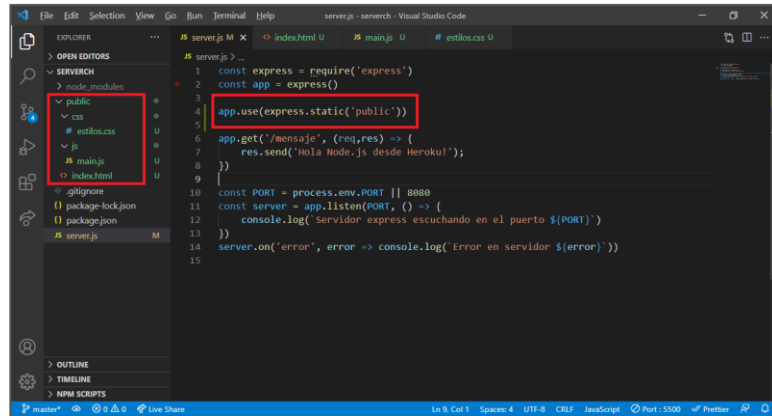
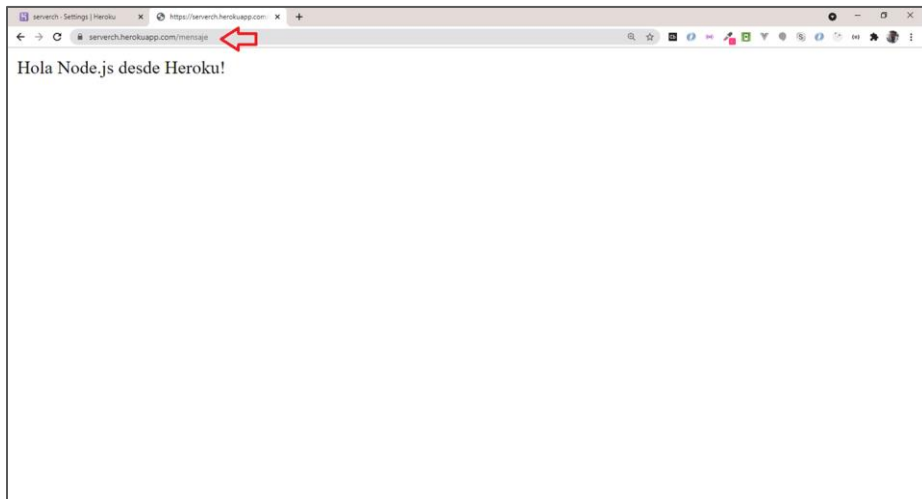


Crear una App en Heroku

Ejemplo
en vivo



- Probamos el proyecto entrando a la ruta configurada anteriormente para ver que esté funcionando.
- Hacemos un cambio y agregamos la carpeta *public* chequeando de que esté puesta como tal en el *server.js*.



CODER HOUSE



Crear una App en Heroku

Ejemplo
en vivo



- Creamos los archivos en la carpeta *public* (html, css y js) y como hubo cambios, debemos subir nuevamente con el paso 2 de los comandos de Git los cambios al repositorio remoto en Heroku.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>App Node.js desde Heroku.com</title>
8   <link rel="stylesheet" href="css/estilos.css">
9 </head>
10 <body>
11   <h1>Hola Node.js desde Heroku</h1>
12
13   <script src="js/main.js"></script>
14 </body>
15 </html>
```

```
C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ git add .

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ git commit -am "v0.0.2"
[master eb77104] v0.0.2
4 files changed, 22 insertions(+)
create mode 100644 public/css/estilos.css
create mode 100644 public/index.html
create mode 100644 public/js/main.js

C:\Cursos\Coderhouse\CursoBackend\Clase33\serverch (master) (serverch@1.0.0)
λ git push heroku master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 1002 bytes | 1002.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: ----- Building on the Heroku-20 stack
remote: ----- Using buildpack: heroku/nodejs
remote: ----- Node.js app detected
remote:
remote: ----- Creating runtime environment
```

CODER HOUSE

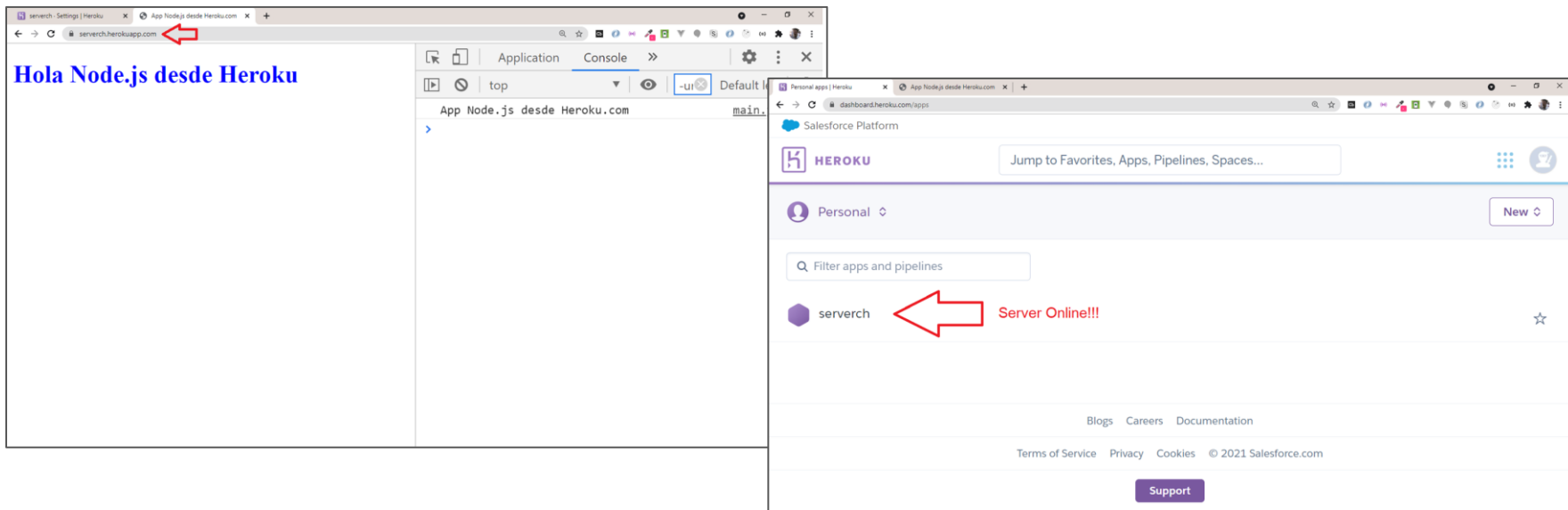


Crear una App en Heroku

Ejemplo
en vivo



- Finalmente tenemos implementada nuestra App **serverch** en el servidor online en Heroku.



CODER HOUSE



IMPLEMENTAR PROYECTO EN HEROKU

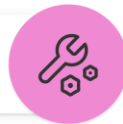
Tiempo: 15 minutos

⚠ Si aún no la tienes, corre a crearte una cuenta en [Heroku](#) (este desafío lo requiere).

CODER HOUSE

Implementar proyecto en Heroku

Desafío
generico



Tiempo: 15 minutos

1. Crea un proyecto de Node.js en Heroku.com utilizando un nombre disponible.
2. Clona el proyecto en el ámbito local dentro de una carpeta con el mismo nombre.
3. Copia el código del servidor realizado en el desafío anterior dentro de esta carpeta (todo menos la carpeta .git, incluyendo .gitignore).
4. Reforma el título h1 en el html para que diga: 'Proyecto Node.js en Heroku.com'.

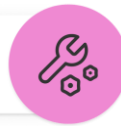


No olvides que el puerto de escucha que ofrece Heroku se encuentra disponible en **PORT**. El proyecto debe permitir seguir trabajando en local en el puerto 8080.

MODER HOUSE

Implementar proyecto en Heroku

Desafío
generico



Tiempo: 15 minutos

- Realiza los pasos correspondientes para agregar este código al repo local clonado de Heroku para luego subirlo a Heroku.com
- Comprueba que el servidor funcione correctamente en la nube a través de la url que provee Heroku para este proyecto.
- Agrega una ruta get `'/info'` que devuelva el mensaje `'Puerto de escucha: '` con el puerto de escucha de node.js. Probar la funcionalidad en forma local.
- Ejecuta los comandos correspondientes para subir esta nueva versión a Heroku.
- Comprueba en la nube la información suministrada por la ruta `'/info'` verificando que el puerto de escucha sea diferente al puerto 8080 local.
- Revisa los logs de la aplicación en forma local, comprobando que el mensaje inicial del puerto de escucha del servidor (con `console.log`) coincida con el de la ruta `info`. **IOUSE**

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Control de versiones, Git y todos sus comandos.
- Qué es PaaS.
- Cómo implementar proyectos en Heroku.



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN