# 2 - Setting up a Node development environment

Ahora que sabes para que sirve Express, nosotros te vamos a mostrar cómo preparar y testear un entorno de desarrollo Node/Express en: Windows, Linux (Ubuntu), y macOS. Este artículo te va a dar todo lo que se necesita para poder empezar a desarrollar apps en Express, sin importar el sistema operativo que se use.

## Express ambiente de desarrollo introducción

*Node* y *Express* hacen muy fácil configurar su computadora con el propósito de iniciar el desarrollo de aplicaciones web. Esta sección provee una reseña de qué herramientas son necesarias, explica algunos de los métodos más simples para instalar Node (y Express) en Ubuntu, macOS y Windows, y muestra cómo puede probar su instalación.

### Qué es el ambiente de desarrollo Express?

El ambiente de desarrollo *Express* incluye una instalación de *Nodejs*, el *NPM administrador de paquetes*, y (opcionalmente) el Generador de Aplicaciones de *Express* en su computadora local.

*Node* y el administrador de paquetes *NPM* se instalan juntos desde paquetes binarios, instaladores, administradores de paquetes del sistema operativo o desde las fuentes (como se muestra en las siguientes secciones). *Express* es entonces instalado por NPM como una dependencia individual de sus aplicaciones web *Express* (juntamente con otras librerías como motores de plantillas, controladores de bases de datos, middleware de autenticación, middleware para servir archivos estáticos, etc.)

*NPM* puede ser usado también para (globalmente) instalar el Generador de Aplicaciones de *Express*, una herramienta manual para crear la estructura de las web apps de *Express* que siguen el patrón MVC . El generador de aplicaciones es opcional porque no necesita utilizar esta herramienta para crear apps que usan Express, o construir apps Express que tienen el mismo diseño arquitectónico o dependencias. No obstante estaremos usandolo, porque hace mucho más fácil, y promueve una estrucura modular de aplicación.

**Nota:** A diferencia de otros frameworks web , el ambiente de desarrollo no incluye un servidor web independiente. Una aplicación web *Node*/*Express* crea y ejecuta su propio servidor web!

Hay otras herramientas periféricas que son parte de un ambiente de desarrollo típico, incluyendo editores de texto o IDEs para edición de código, y

herramientas de administración de control de fuentes como Git para administrar con seguridad diferentes versiones de su codigo. Asumimos que usted ya tiene instaladas esta clase de herramientas (en particular un editor de texto).

### Qué sistemas operativos son soportados?

*Node* puede ser ejecutado en Windows, macOS, varias "versiones" de Linux, Docker, etc. (hay una lista completa de páginas de Downloads de nodejs). Casi cualquier computadora personal podría tener el desempeño necesario para ejecutar Node durante el desarrollo. *Express* es ejecutado en un ambiente *Node*, y por lo tanto puede ejecutarse en cualquier plataforma que ejecute *Node*.

En este articulo proveemos instruciones para configurarlo para Windows, macOS, and Ubuntu Linux.

### ¿Qué versión de Node/Express puedo usar?

Hay varias versiones de Node — recientes que contienen reparación de bugs, soporte para versiones más recientes de ECMAScript (JavaScript) estándares, y mejoras a las APIs de Node .

Generalmente se debe usar la versión más reciente *LTS (soporte de largo-plazo),* una versión como esta es más estable que la versión "actual", mientras que sigue teniendo características relativamente recientes (y continua siendo activamente actualizado). Debería utilizar la versión *Actual* si necesita una característica que no está presente en la versión LTS.

Para *Express* siempre se debe utilizar la versión más reciente.

### ¿Qué pasa con bases de datos y otras dependencias?

Otras dependencias, tales como los controladores de bases de datos, motores de plantillas, motores de autenticación, etc. son parte de la aplicación, y son importadas dentro del ambiente de la aplicación utilizando el administrador de paquetes NPM. Estos los discutiremos en artículos posteriores app-specific.

## Instalar Node

Para poder utilizar *Express* primero tiene que instalar *Nodejs* y el Administrador de Paquetes de Node (NPM) en su sistema operativo. Las siguientes secciones explican la forma más fácil de instalar la versión Soporte de Largo-Plazo (SLP) de Nodejs en Ubuntu Linux 16.04, macOS, y Windows 10.

**Tip:** Las secciones de abajo muestran la forma más facil de instalar *Node* y *NPM* en nuestras plataformas de sistemas operativo a elegir. Si está utilizando otro SO o solo quiere ver alguna de otros enfoques para las plataformas

actuales entonces vea Instalando Node.js vía administrador de paquetes (nodejs.org).

## macOS y Windows

Instalar *Node* y *NPM* en Windows y macOS es sencillo, porque simplemente debe utilizar el instalador provisto:

1. Descargue el instalador requerido:
    1. Vaya a https://nodejs.org/es/
    2. Seleccione el botón para descargar la versión LTS que es "Recomendada la mayoría de los usuarios".
2. Instale Node al dar doble-click en el archivo de descarga y en seguida la instalación inicia.

## Ubuntu 20.04

La forma más fácil de instalar la versión LTS de Node 10.x es la usar el administrador de paquetes para obtenerlo del repositorio de distribuciones *binarias* de Ubuntu. Esto puede ser hecho muy simple al ejecutar los siguientes dos comandos en su terminal:

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -

sudo apt-get install -y nodejs
```

**Advertencia:** No instale directamente desde los repositorios normales de Ubuntu porque pueden contener versiones muy antiguas de Node.

## Probar su instalación de Nodejs y NPM

La forma más fácil de probar que Node está instalado es ejecutar el comando "versión" en su prompt de terminal/command y checar que una cadena de versión es devuelta:

```
>node -v

v16.13.1
```

El administrador de paquetes *NPM* de *Nodejs* también debería haber sido instalado y puede ser probado de la misma forma:

```
>npm -v
```

```
8.1.2
```

Como una prueba un poco más emocionante creemos un muy básico "básico servidor node" que simplemente imprima "Hola Mundo" en el browser cuando visite la URL correcta en él:

1. Copie el siguiente texto en un archivo llamado **holanode.js**. Este utiliza características básicas de Node (nada desde Express) y algo de sintáxis ES6:

```
//Load HTTP module

const http = require("http");

const hostname = '127.0.0.1';

const port = 3000;



//Create HTTP server and listen on port 3000 for requests

const server = http.createServer((req, res) => {



  //Set the response HTTP header with HTTP status and Content type

  res.statusCode = 200;

  res.setHeader('Content-Type', 'text/plain');

  res.end('Hello World\n');

});



//listen for request on port 3000, and as a callback function have
the port listened on logged

server.listen(port, hostname, () => {

  console.log(`Server running at http://${hostname}:${port}/`);
```

```
});
```

El código importa el módulo "http" y lo usa para crear un servidor (`createServer()`) que escucha las solicitudes HTTP en el puerto 3000. Luego, el script imprime un mensaje en la consola con la URL del navegador puede usar para probar el servidor. La función `createServer()` toma como argumento una función callback que se invocará cuando se reciba una solicitud HTTP — esto simplemente devuelve una respuesta con un código de estado HTTP de 200 ("OK") y el texto sin formato "Hello World".

**Nota:** ¡No se preocupe si aún no comprende exactamente lo que está haciendo este código! ¡Explicaremos nuestro código con mayor detalle una vez que comencemos a usar Express!

2. Inicie el servidor navegando en el mismo directorio que su archivo `hellonode.js` en su símbolo del sistema, y llamando a `node` junto con el nombre del script, así:

```
>node hellonode.js

Server running at http://127.0.0.1:3000/
```

3. Navega a la URL http://127.0.0.1:3000 . Sí todo está funciona, el navegador simplemente debe mostrar la cadena de texto "Hello World".

# Usando NPM

Junto al propio node, NPM es la herramienta más importante para trabajar con aplicaciones de node. NPM se usa para obtener los paquetes (bibliotecas de JavaScript) que una aplicación necesita para el desarrollo, las pruebas y/o la producción, y también se puede usar para ejecutar pruebas y herramientas utilizadas en el proceso de desarrollo.

**Nota:** Desde la perspectiva de Node, Express es solo otro paquete que necesita instalar usando NPM y luego requerir en su propio código.

Se puede usar NPM manualmente para buscar por separado cada paquete necesario. Por lo general, administramos las dependencias utilizando un archivo de definición de texto plano llamado package.json. Este archivo enumera todas las dependencias para un "paquete" de JavaScript específico, incluido el nombre del paquete, la versión, la descripción, el archivo inicial a ejecutar, las dependencias de producción, las dependencias de desarrollo, las versiones de Node con las que puede trabajar, etc. El archivo package.json debería contener todo lo que NPM necesita para buscar y ejecutar su aplicación (si estuviera

escribiendo una biblioteca reutilizable, podría usar esta definición para cargar su paquete en el repositorio npm y ponerlo a disposición de otros usuarios).

## Agregando dependencias

Los siguientes pasos muestran cómo puede usar NPM para descargar un paquete, guardarlo en las dependencias del proyecto y luego requerirlo en una aplicación Node.

**Nota:** Aquí mostramos las instrucciones para buscar e instalar el paquete *Express*. Más adelante mostraremos cómo este paquete y otros ya están especificados para nosotros utilizando el *Generador de aplicaciones Express*. Esta sección se proporciona porque es útil para comprender cómo funciona NPM y qué está creando el generador de aplicaciones.

1. Primero cree un directorio para su nueva aplicación y acceda a él:

```
myapp

cd myapp
```

2. Use el comando `npm init` para crear un archivo **package.json** para su aplicación. Este comando le solicita varias cosas, incluido el nombre y la versión de su aplicación y el nombre del archivo de punto de entrada inicial (de forma predeterminada, esto es **index.js**). Por ahora, solo acepte los valores predeterminados:

```
npm init
```

   Si muestra el archivo **package.json** (`cat package.json`), verá los valores predeterminados que aceptó, que finalizarán con la licencia.

```
{

  "name": "myapp",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"
```

```
  },

  "author": "",

  "license": "ISC"

}
```

3. Ahora instale Express en el directorio `myapp` y guárdelo en la lista de dependencias de su archivo package.json (opción --save):

```
npm install express --save
```

La sección de dependencias de su **package.json** ahora aparecerá al final del archivo **package.json** e incluirá *Express*.

```
{

  "name": "myapp",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "",

  "license": "ISC",

  "dependencies": {

    "express": "^4.16.3"

  }

}
```

4. Para usar la biblioteca, llame a la función `require()` como se muestra a continuación en su archivo **index.js**.

```
const express = require('express')

const app = express();



app.get('/', (req, res) => {

  res.send('Hello World!')

});

app.listen(8000, () => {

  console.log('Example app listening on port 8000!')

});
```

Este código muestra una aplicación web mínima "HelloWorld" Express. Esto importa el módulo "express" y lo usa para crear un servidor (`app`) que escucha las solicitudes HTTP en el puerto 8000 e imprime un mensaje en la consola que indica qué URL del navegador puede usar para probar el servidor. La función `app.get()` solo responde a las solicitudes HTTP `GET` con la ruta URL especificada (`'/'`), en este caso llamando a una función para enviar nuestro mensaje Hello World! .

5. Cree un archivo llamado **index.js** en la raíz del directorio de la aplicación "myapp" y dele el contenido que se muestra arriba.

6. Puede iniciar el servidor llamando a node con el script en su símbolo del sistema:

```
> node index.js

Example app listening on port 8000
```

7. Navega a la URL (http://127.0.0.1:8000/). Sí todo está funciona, el navegador simplemente debe mostrar la cadena de texto "Hello World".

## Dependencias de Desarrollo

Si una dependencia solo se usa durante el desarrollo, debe guardarla como una "dependencia de desarrollo" (para que los usuarios de su paquete no tengan que instalarla en producción). Por ejemplo, para usar la popular herramienta Linting JavaScript eslint llamaría a NPM como se muestra a continuación:

```
npm install eslint --save-dev
```

La siguiente entrada se agregaría al **paquete.json** de su aplicación:

```
  "devDependencies": {

    "eslint": "^4.12.1"

  }
```

**Nota:** "Linters" son herramientas que realizan análisis estáticos en el software para reconocer e informar la adhesión / no adhesión a algún conjunto de mejores prácticas de codificación.

## Ejecutando tareas

Además de definir y buscar dependencias, también puede definir scripts con nombre en sus archivos package.json y llamar a NPM para ejecutarlos con el comando run-script. Este enfoque se usa comúnmente para automatizar las pruebas en ejecución y partes de la cadena de herramientas de desarrollo o construcción (por ejemplo, ejecutar herramientas para minimizar JavaScript, reducir imágenes, LINT/analizar su código, etc.).

**Nota:** Los ejecutadores de tareas como Gulp y Grunt también se pueden usar para ejecutar pruebas y otras herramientas externas.

Por ejemplo, para definir un script para ejecutar la dependencia de desarrollo de *eslint* que especificamos en la sección anterior, podríamos agregar el siguiente bloque de script a nuestro archivo **package.json** (suponiendo que el origen de nuestra aplicación esté en una carpeta /src/js):

```
"scripts": {

  ...

  "lint": "eslint src/js"
```

```
  ...

}
```

Para explicar un poco más, `eslint src/js` es un comando que podríamos ingresar en nuestra línea de terminal/linea de comandos para ejecutar `eslint` en archivos JavaScript contenidos en el directorio `src/js` dentro de nuestro directorio de aplicaciones. Incluir lo anterior dentro del archivo package.json de nuestra aplicación proporciona un acceso directo para este comando: `lint`.

Entonces podríamos ejecutar eslint usando NPM llamando a:

```
npm run-script lint

# OR (using the alias)

npm run lint
```

Es posible que este ejemplo no parezca más corto que el comando original, pero puede incluir comandos mucho más grandes dentro de sus scripts npm, incluidas cadenas de comandos múltiples. Puede identificar un solo script npm que ejecute todas sus pruebas a la vez.

# Instalando Express Application Generator

La herramienta Express Application Generator genera un "esqueleto" de la aplicación Express. Instale el generador usando NPM como se muestra (el indicador `-g` instala la herramienta globalmente para que pueda llamarla desde cualquier lugar):

```
npm install express-generator -g
```

Para crear una aplicación *Express* llamada "helloworld" con la configuración predeterminada, navegue hasta donde desea crearla y ejecute la aplicación como se muestra:

```
express helloworld
```

**Nota:** También puede especificar la biblioteca de plantillas para usar y una serie de otras configuraciones. Use el comando `--help` para ver todas las opciones:

```
express --help
```

NPM creará la nueva aplicación Express en una subcarpeta de su ubicación actual, mostrando el progreso de la compilación en la consola. Al finalizar, la herramienta mostrará los comandos que necesita ingresar para instalar las dependencias de Node e iniciar la aplicación.

La nueva aplicación tendrá un archivo **package.json** en su directorio raíz. Puede abrir esto para ver qué dependencias están instaladas, incluidas Express y la biblioteca de plantillas Jade:

```
{
  "name": "helloworld",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.18.2",
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.9",
    "express": "~4.15.5",
    "jade": "~1.11.0",
    "morgan": "~1.9.0",
    "serve-favicon": "~2.4.5"
  }
}
```

Instale todas las dependencias para la aplicación helloworld usando NPM como se muestra:

```
cd helloworld

npm install
```

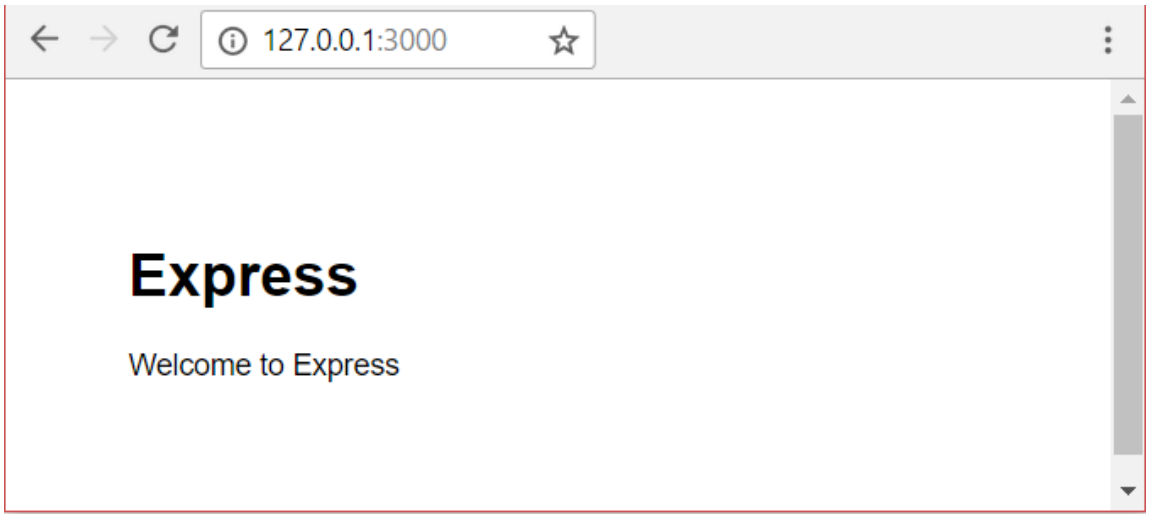
Luego ejecute la aplicación (los comandos son ligeramente diferentes para Windows y Linux/macOS), como se muestra a continuación:

```
#  Ejecute helloworld en Windows con símbolo del sistema

SET DEBUG=helloworld:* & npm start



#  Ejecute helloworld en Windows con PowerShell

SET DEBUG=helloworld:* | npm start



#  Ejecute helloworld en Linux/macOS

DEBUG=helloworld:* npm start
```

El comando DEBUG crea registros útiles, lo que resulta en una salida como la que se muestra a continuación.

```
>SET DEBUG=helloworld:* & npm start



> helloworld@0.0.0 start D:\Github\expresstests\helloworld

> node ./bin/www



  helloworld:server Listening on port 3000 +0ms
```

Abra un navegador y navegue a http://127.0.0.1:3000/ para ver la página de bienvenida Express predeterminada.




Hablaremos más sobre la aplicación generada cuando lleguemos al artículo sobre la generación de una aplicación esqueleto.

## Resumen

Ahora tiene un entorno de desarrollo de Node en funcionamiento en su computadora que puede usarse para crear aplicaciones web Express. También ha visto cómo se puede usar NPM para importar Express en una aplicación, y también cómo puede crear aplicaciones usando la herramienta Express Application Generator y luego ejecutarlas.

En el siguiente artículo, comenzaremos a trabajar a través de un tutorial para crear una aplicación web completa utilizando este entorno y las herramientas asociadas.