**¿Qué es Node.js?**

**Node** es una plataforma *OpenSource* para desarrollar aplicaciones, que está enfocada en tiempo real. Es similar a la máquina virtual de *Java*.

**Node** es muy útil para aplicaciones que tienen alta concurrencia. Utiliza **JavaScript** como lenguaje de programación y patrones de programación asíncronos y orientados a objetos.

¿Por qué decimos que **Node.js** es una plataforma simple?

1. Es una librería estándar es donde el código va a morir. Mantener una plataforma estándar actualizada puede ser un problema.
2. ***npm*** y la ***modularidad*** de **Node** abre un ecosistema de trabajo muy grande.
3. El modelo de trabajo *Open Source* beneficia mucho a la plataforma.

**Node** existe gracias a un componente principal que se llama **V8**, que es el que nos permite utilizar **JavaScript** del lado del servidor.

* Node.js  
  Es una plataforma open source para desarrolar aplicaciones enfocadas en tiempo real. (Parecidas a JVM de Java o Framework .NET de Microsoft).  
  Ideal para alta concurrencia  
  Te permite desbloquear muchos recursos  
  Fullstack (desarrollo con Node.js en el server y js en el front).

Node.js simple para desarrollar:

Aplicaciones centradas en red,  
Servidores en tiempo real.  
Aplicaciones modulares  
Utiliza patrones de programación asincrono y orientado a eventos, pero esta es la clave que hace node.js especial.

El core de Node.js es muy pequeño, es simple de entender, el crecimiento de la plataforma en es base a la comunidad, creando modulos enfocados en resolver ciertos problemas. Mantenidos por el user-land (npm) .

V8 -> es la maquina virtual de chrome, es lo que permite usar node.js del lado del servidor.

LIBUV -> es una librería que se encarga de manejar el loop de las tareas y las operaciones de entrada y salida de archivos…

zlib -> Encriptación  
openssl -> para comunicación segura  
http\_parse-> para manejar todo el parseo de http

Networking and i/O  
La mayoría de las aplicaciones de Node.js son enfocadas para la red y de entrada y salida.  
Diseñado para manejar grandes cargas de datos de entrada y salida.  
Empoderar la web moderna.

**Características de la plataforma Node.js**

**Networking** y *entrada* y *salida*: Más de 1/4 del core de **Node.js** está dedicado a *Networking.*  
**Node.js** está especialmente diseñado para tipos de trabajo de entrada y salida.

**Node.js** es *modular*. **npm** es el registro de paquetes *OpenSource* más grande del mundo.  
El sistema modular de **Node.js** resuelve los problemas *dependency hell*.

La característica principal de **Node.js** es que podemos trabajar de manera asíncrona y orientada a objetos.

**Node.js** es muy bueno para aplicaciones en tiempo real o para orquestación de recursos entre múltiples aplicaciones.

**Node.js** no es bueno para computaciones que requieren uso de la CPU, ni para desarrollar aplicaciones como Systems programming.

**¿Qué tipo de aplicaciones podemos desarrollar con Node.js?**

**Node.js** es el proyecto *open source* con el crecimiento más rápido del mundo en este momento. Muchas empresas están empezando no solo a utilizarlo sino a contribuir y patrocinar y tienen empleados dedicados a que aporten al proyecto y eso es lo que permite que tenga un crecimiento tan grande.

¿Qué tipo de aplicaciones podemos desarrollar?

**- Aplicaciones de escritorio.**  
**- Dispositivos embebidos.**  
**- Servidores web.**  
**- Aplicaciones móviles: hace poco anunciaron el soporte para Android.**

**Cómo instalar Node.js en macOS**

Antes de empezar a desarrollar el proyecto es importante tener instaladas las aplicaciones que vamos a utilizar.  
En el vídeo vamos a ver la instalación en *macOS*, pero mas adelante vas a encontrar los artículos para otros sistemas operativos.

El primer paso es ir a [**nodejs.org**](https://platzi.com/clases/nodejs/concepto/preparando-el-entorno-de-desarrollo/como-instalar-nodejs-en-macos/material/url). Ahí vas a encontrar dos versiones, la actual y la **LTS**, que significa *Long Term Support*. Debemos elegir la de la rama 8, que es la actual porque es la que tiene todos los últimos features y se va a convertir en la **LTS** muy pronto.

Para verificar que **Node.js** quedó instalado vamos a la terminal y ejecutamos:  
$ node -v  
Con esto veremos la versión de **Node** que hemos descargado.

Otra forma de instalar, si trabajas con proyectos de **Node** que utilizan diferentes versiones, puedes utilizar **nvm**. Esta herramienta te permite cambiar entre versiones de **Node** fácilmente.

**Cómo instalar PostgreSQL y Redis en macOS**

Vamos a utilizar una base de datos relacional llamada **PostgreSQL**.

*Mac* no tiene por defecto un manejador de paquetes entonces vamos a tener un repositorio de aplicaciones que podemos instalar en la línea de comandos, que se llama **Homebrew**. Después de esto ya podremos instalar **PostgreSQL**.

Una vez hemos instalado **Homebrew** podemos instalar **Postgres** ejecutando:  
$brew install postgresql

Y para inicializar:  
$ brew services start postgresql

Con brew services podemos inicializar **Postgres**.

Igualmente vamos a instalar Redis, que es una base de datos no relacional. Se instala de la misma forma:  
$ brew install redis

Y para inicializar:  
$ brew services start redis

**Cómo instalar Visual Studio Code en macOS**

**Visual Studio Code** es el editor que vamos a utilizar en el curso, sin embargo puedes utilizar el editor que más te guste.  
Una de las ventajas de este editor es que tiene un buen soporte para **Node.js**, además de que está desarrollado en **Node**.  
La instalación es sencilla y en la documentación nos dan las guías sobre cómo utilizarlo mejor.  
Vamos a utilizar un paquete de desarrollo que se llama **VS Code** for **Node.js** que se puede descargar directamente desde el editor.

**Cómo instalar Ansible en macOS**

Otra herramienta que vamos a utilizar es **Ansible**, que nos sirve para automatizar tareas de infraestructura. La vamos a utilizar para crear *scripts* de manera automatizada.

Para instalar **Ansible** vamos a utilizar **Homebrew** con el comando:  
$ brew install ansible

Por otro lado vamos a descargar **Vagrant**, que nos permite tener ambientes de desarrollo o servidores virtuales para hacer primero un *deploy* local antes de hacer despliegue en la nube.

**Estructura básica de un módulo de Node.js**

Para empezar con el proyecto vamos a crear un módulo de **Node.js,** con sus respectivas pruebas unitarias. El módulo de Persistencia en Bases de Datos.

Vamos a crear el directorio del proyecto y vamos a utilizar un patrón de mono repo, donde tenemos un único repositorio de Git y varios proyectos adentro.

Un módulo de Node por lo general tiene un archivo que se llama package.json, que tiene la metadata del módulo. Este archivo se puede crear a mano o utilizando npm con una herramienta que nos permite crearlo automáticamente.

Para empezar a implementar nuestro módulo creamos el archivo **index.js** y nos vamos a **Visual Studio Code**, donde podemos ver nuestros archivos y empezar a trabajar con ellos. Una buena práctica es crear el archivo [**README.md**](http://readme.md/) con instrucciones de instalación y cómo contribuir al proyecto. Recuerda que este es el archivo de documentación de tu proyecto.

Como otra buena práctica utilizaremos **JavaScript** en modo estricto, para empezar a definir la forma básica del módulo. La forma de escribir el objeto que utilizamos en el código es de **ECMAScript 2016.**

**Definición de entidades de base de datos**

Lo primero que tenemos que hacer es definir las entidades de las bases de datos que vamos a utilizar. En nuestro proyecto vamos a usar **PostgreSQL,** una base de datos relacional.

El Agente va a conectarse al servidor en tiempo real y cada cierto tiempo va a reportar una métrica.  
La Métrica es cualquier valor que tiene un tipo, que va a ser almacenado en la base de datos.

**Implementación de modelos con sequelize**

**Sequelize** es un modelo de programación de mapeo de *objeto-relacional* basado en promesas, para **Node.js.**

**Dos tipos de objetos:** modelos *nativos* de **sequelize** y *servicios.*

**Singleton** es un objeto que solo tiene una instancia. Cada vez que llamemos a una función no va a crear múltiples instancias.

**Implementación de un módulo básico de base de datos**

Vimos cómo definimos los modelos en el video pasado. Ahora vamos a ver cómo podemos relacionar los modelos entre si.

Primero vamos a necesitar las funciones que definimos anteriormente. Para poner a interactuar los modelos entre si debemos hacer la configuración de la base de datos y los modelos.

**Implementación de script de inicialización de base de datos**

Cuando ejecutemos el proyecto por primera vez necesitaremos tener creada la base de datos, y crear un usuario con permisos para alimentar esa base de datos.

