# **Introducción**

Completado

100 XP

* 1 minuto

Es un desarrollador de software en una tienda en línea que se especializa en libros infantiles. Va a crear un nuevo sitio web para su negocio. Aún no ha definido por completo las necesidades exactas de este sitio web, pero crear el sitio de principio a fin será su responsabilidad. Tiene mucha experiencia con JavaScript, por lo que quiere encontrar una solución que se adapte bien a sus puntos fuertes.

Ha oído hablar de la pila MEAN (MongoDB, Express.js, AngularJS y Node.js) y sabe que usa JavaScript. Así que decide probarla y compila una pila MEAN y una aplicación web básica que almacena información sobre libros. Puede usar lo que ha aprendido para empezar con el nuevo sitio web.

## **Objetivos de aprendizaje**

En este módulo, aprenderá a:

* Decidir si la pila web MEAN es la opción adecuada
* Crear una máquina virtual Ubuntu Linux para hospedar la aplicación web
* Instalar los componentes de la pila MEAN en la máquina virtual
* Crear una aplicación web básica en la pila MEAN

# **Decidir si MEAN es la opción adecuada para usted**

Completado

100 XP

* 3 minutos

MEAN es una pila de desarrollo para compilar y hospedar aplicaciones web. MEAN es el acrónimo de sus componentes: MongoDB, Express, AngularJS y Node.js.

La razón principal por la que podría valorar la posibilidad de usar MEAN es que está familiarizado con JavaScript. Estas son algunas otras razones por las que es recomendable elegir MEAN u otra pila de desarrollo para la próxima aplicación web.

## **¿Por qué debería elegir MEAN?**

Todos los componentes de la pila MEAN son confiables, bien conocidos y de código abierto, pero también lo son muchas otras pilas de desarrollo. Estas son algunas de las razones por las que podría elegir MEAN en lugar de otras pilas de desarrollo.

Los datos no están muy estructurados

MongoDB es lo que se denomina una base de datos NoSQL. Una base de datos NoSQL no necesita que los datos se estructuren de una manera predefinida como lo estarían con una base de datos relacional, como Microsoft SQL Server o MySQL. En su lugar, MongoDB almacena sus datos en documentos similares a JSON que no requieren las estructuras de datos estrictas que necesitan MySQL u otras bases de datos relacionales.

MEAN está bien documentado

Todos los componentes de la pila MEAN son populares en la actualidad. Los recursos para trabajar con MongoDB, Express, AngularJS y Node.js son fáciles de encontrar.

MEAN se ejecuta prácticamente en cualquier entorno

También puede desarrollar aplicaciones de la pila MEAN desde su entorno de desarrollo favorito, ya sea Windows, macOS o Linux.

## **¿Por qué MEAN podría no ser la opción adecuada para mí?**

Estas son algunas razones por las que podría elegir una pila de desarrollo que no sea MEAN.

Incluso si decide que MEAN no es adecuada para usted, puede estar interesado en este módulo. Muchos de los patrones que verá se aplican a otros tipos de marcos de trabajo de aplicaciones web.

Sus datos están muy estructurados

Si sus datos están muy estructurados, podría beneficiarse de colocarlos en una base de datos relacional, como Microsoft SQL Server o MySQL.

JavaScript no es su aptitud principal

Si prefiere otro lenguaje en lugar de JavaScript, puede haber un marco alternativo a su alcance.

Por ejemplo, puede que la pila LAMP, que consta de Linux, Apache, MySQL y PHP (a veces con Perl o Python en lugar de PHP), se adapte mejor a sus puntos fuertes y experiencia.

Al igual que la mayoría de los marcos de trabajo de las aplicaciones, puede ejecutar la aplicación de la pila MEAN en muchos entornos diferentes. Puede ejecutar la aplicación en un equipo físico en su sala de servidores, en una máquina virtual o en contenedores.

Aquí ejecutará la aplicación en una máquina virtual que se ejecuta en Azure. MEAN es compatible con varios sistemas operativos diferentes. Con fines de aprendizaje, aquí usará Ubuntu Linux.

## **Creación de una máquina virtual Ubuntu Linux**

Normalmente, crearía un *grupo de recursos* antes de crear otros recursos en Azure. Un grupo de recursos es un contenedor que almacena los recursos relacionados de una solución de Azure. En este ejercicio, el espacio aislado de Azure le proporciona un grupo de recursos. Si trabaja en su propia suscripción de Azure, use el siguiente comando para crear un grupo de recursos en una ubicación cercana.

Nota

El código de ejemplo siguiente es para que lo use con su propia cuenta, no con este espacio aislado.

CLI de Azure

Copiar

az group create \

--name <resource-group-name> \

--location <resource-group-location>

1. En Cloud Shell, ejecute el comando az vm create para crear una máquina virtual Ubuntu.
2. CLI de Azure
3. Copiar

az vm create \

--resource-group [sandbox resource group name] \

--name MeanStack \

--image Canonical:UbuntuServer:16.04-LTS:latest \

--admin-username azureuser \

--generate-ssh-keys

1. El comando tarda unos dos minutos en completarse. Cuando finalice el comando, verá una salida similar a esta.
2. JSON
3. Copiar

(Output Example)

{

"fqdns": "",

"id": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myResourceGroup/providers/Microsoft.Compute/virtualMachines/MeanStack",

"location": "eastus",

"macAddress": "00-0D-3A-1E-1B-3B",

"powerState": "VM running",

"privateIpAddress": "10.0.0.5",

"publicIpAddress": "104.211.9.245",

"resourceGroup": "[sandbox resource group name]",

"zones": ""

}

1. El nombre de la máquina virtual es "MeanStack". Usará este nombre en futuros comandos para identificar la máquina virtual con la que quiera trabajar.
2. Abra el puerto 80 en la máquina virtual para permitir el tráfico HTTP entrante a la aplicación web que creará después.
3. CLI de Azure
4. Copiar

az vm open-port \

--port 80 \

--resource-group [sandbox resource group name] \

--name MeanStack

1. Cree una conexión SSH a la máquina virtual.  
   Aunque el resultado del comando az vm create muestra la dirección IP pública de la máquina virtual, puede que le sea útil almacenar la dirección en una variable de Bash.  
   Para comenzar, ejecute az vm show. Este comando guarda la dirección IP en una variable de Bash denominada ipaddress.
2. CLI de Azure
3. Copiar

ipaddress=$(az vm show \

--name MeanStack \

--resource-group [sandbox resource group name] \

--show-details \

--query [publicIps] \

--output tsv)

1. Conéctese a la máquina virtual de esta forma.
2. Bash
3. Copiar

ssh azureuser@$ipaddress

1. Cuando se le solicite, conteste "yes" (sí) para guardar la identidad de la máquina virtual de forma local para que las futuras conexiones sean de confianza.  
   Usará la conexión SSH para configurar el software en la máquina virtual en las secciones siguientes.

## **Resumen**

Con la máquina virtual Ubuntu lista, ya puede instalar cada componente de la pila MEAN. Comenzará instalando MongoDB.

Muchas aplicaciones requieren una base de datos. Aquí instalará MongoDB, la "M" de la pila MEAN. Es una conocida solución de base de datos NoSQL que es gratuita y de código abierto. Una base de datos NoSQL no necesita que los datos se estructuren de una manera predefinida como lo estarían con una base de datos relacional como SQL Server o MySQL.

MongoDB almacena sus datos en documentos similares a JSON que no requieren estructuras de datos rígidas. Se interactúa con MongoDB mediante consultas y comandos que se envían por medio de la notación de objetos JavaScript o JSON.

## **¿Qué ediciones de MongoDB están disponibles?**

MongoDB proporciona dos ediciones:

* MongoDB Community Server
* MongoDB Enterprise Server

Aquí instalará MongoDB Community Server. Más adelante usará MongoDB para almacenar información sobre libros.

## **¿Cómo se instala MongoDB?**

Puede instalar MongoDB en Linux, macOS y Windows. Con fines de aprendizaje, aquí instalará MongoDB en Ubuntu mediante el administrador de paquetes apt de Ubuntu.

El proceso de instalación varía según el sistema operativo. Si no está familiarizado con Ubuntu, puede seguir leyendo para hacerse una idea de cómo funciona todo.

Más adelante, puede [consultar la guía de instalación](https://docs.mongodb.com/manual/administration/install-community) para obtener más información.

## **Instalación de MongoDB**

Aquí instalará MongoDB con tan solo unos comandos. Trabajará desde la conexión SSH a la máquina virtual Ubuntu que ha creado en la unidad anterior.

Nota

La actualización puede tardar hasta 10 minutos.

1. En primer lugar, vamos a asegurarnos de que todos los paquetes actuales están actualizados.
2. Bash
3. Copiar

sudo apt update && sudo apt upgrade -y

1. Nota  
   La parte sudo significa que queremos ejecutar el comando con privilegios administrativos.
2. Instale el paquete de MongoDB.
3. Bash
4. Copiar

sudo apt-get install -y mongodb

1. Una vez finalizada la instalación, el servicio se debería iniciar automáticamente. Para confirmarlo, ejecute este comando.
2. Bash
3. Copiar

sudo systemctl status mongodb

1. Debería ver que el servicio se está ejecutando.
2. Resultado
3. Copiar

azureuser@MeanStack:~$ sudo systemctl status mongodb

● mongodb.service - An object/document-oriented database

Loaded: loaded (/lib/systemd/system/mongodb.service; enabled; vendor preset: enabled)

Active: active (running) since Thu 2019-08-22 16:46:30 UTC; 9s ago

Docs: man:mongod(1)

Main PID: 18360 (mongod)

CGroup: /system.slice/mongodb.service

└─18360 /usr/bin/mongod --config /etc/mongodb.conf

Aug 22 16:46:30 MeanStack systemd[1]: Started An object/document-oriented database.

1. Ejecute mongod --version para comprobar la instalación.
2. Bash
3. Copiar

mongod --version

Mantenga la conexión SSH abierta para la parte siguiente.

Aquí instalará Node.js, la N del acrónimo MEAN. Como MongoDB, Node.js es de código abierto.

Node.js actuará como el host del lado servidor para la aplicación web y controlará el tráfico HTTP entrante. Node.js también le proporciona una manera de comunicarse con la instalación de MongoDB, que verá más adelante.

## **¿Qué versiones de Node.js están disponibles?**

Puede obtener Node.js de dos maneras:

* Long Term Support (LTS) (Compatibilidad a largo plazo): por lo general, se considera más estable y es recomendable para la mayoría de los usuarios y para entornos de producción.
* Current (Actual): es para quienes quieren experimentar con las características más recientes. Dado que se pueden introducir cambios importantes entre las versiones, no se recomienda para entornos de producción.

Aquí usará Node.js LTS.

## **¿Cómo se instala Node.js?**

Como MongoDB, puede ejecutar Node.js en Windows, macOS y Linux. Node.js también admite los sistemas operativos basados en Unix como SunOS y AIX.

De igual forma que con MongoDB, aquí registrará el repositorio de Node.js para que apt pueda localizar el paquete.

Recuerde que está trabajando con una máquina virtual Ubuntu. Más adelante, puede [consultar la guía de instalación](https://nodejs.org/en/download/package-manager) para obtener información sobre cómo instalar Node.js en su plataforma favorita.

## **Instalación de Node.js**

Aquí instalará Node.js. Como con MongoDB, el proceso implica registrar el repositorio de Node.js para que apt pueda localizar el paquete.

Importante

En este caso, trabajará desde la conexión SSH a la máquina virtual Ubuntu que ha creado anteriormente en este módulo.

1. Registre el repositorio de Node.js para que el administrador de paquetes pueda localizar los paquetes de esta forma. Este paso puede tardar hasta 10 minutos o más, en función de la conexión a Internet.
2. Bash
3. Copiar

curl -sL https://deb.nodesource.com/setup\_8.x | sudo -E bash -

1. Instale el paquete de Node.js.
2. Bash
3. Copiar

sudo apt install nodejs

1. Ejecute nodejs -v para comprobar la instalación.
2. Bash
3. Copiar

nodejs -v

1. La salida muestra que tiene la versión de LTS más reciente de Node.js.

## **Cierre de la sesión de SSH**

Por ahora, ya ha terminado de trabajar directamente en la máquina virtual. Ejecute exit para salir de la sesión SSH a la máquina virtual.

Bash

Copiar

exit

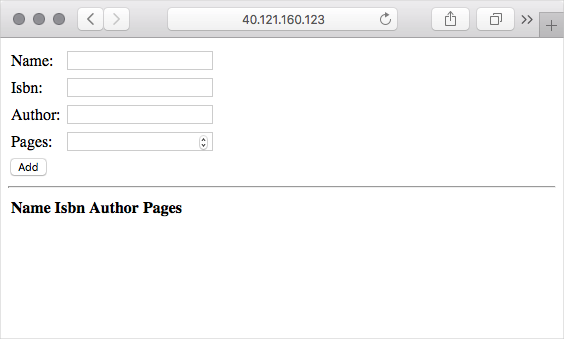
Ha vuelto a la sesión de Cloud Shell.

Hasta ahora, tiene MongoDB y Node.js instalados en la máquina virtual Ubuntu. Ha llegado el momento de crear una aplicación web básica para ver todo en acción. A lo largo del camino, verá cómo encajan AngularJS y Express.

Una excelente manera de aprender es con un ejemplo. La aplicación web que compilará implementa una base de datos de libros básica. La aplicación web le permite mostrar información sobre los libros, agregar libros nuevos y eliminar los existentes.

La aplicación web que verá aquí demuestra muchos conceptos que se aplican a la mayoría de las aplicaciones web de la pila MEAN. Según sus necesidades e intereses, puede explorar las características que necesita para compilar sus propias aplicaciones de la pila MEAN.

Este es el aspecto que tendrá la aplicación web de libros.



Así es como encaja cada componente de la pila MEAN.

* MongoDB almacena información sobre los libros.
* Express enruta cada solicitud HTTP al controlador adecuado.
* AngularJS conecta la interfaz de usuario con la lógica de negocios del programa.
* Node.js hospeda la aplicación del lado servidor.

Importante

Con fines de aprendizaje, aquí creará una aplicación web básica. Su objetivo es probar la pila MEAN y ayudarle a hacerse una idea de cómo funciona. La aplicación no es lo suficientemente segura ni está preparada para su uso en producción.

## **¿Qué ocurre con Express?**

Hasta ahora, ha instalado MongoDB y Node.js en la máquina virtual. ¿Qué ocurre con Express, la E del acrónimo MEAN?

Express es un marco de servidor web que se ha compilado para Node.js y que simplifica el proceso de creación de aplicaciones web.

El objetivo principal de Express es controlar el enrutamiento de solicitudes. *Enrutamiento* hace referencia a cómo responde la aplicación a una solicitud a un punto de conexión concreto. Los puntos de conexión están formados por una ruta de acceso, o URI, y un método de solicitud, como GET o POST. Por ejemplo, podría responder a una solicitud GET al punto de conexión /book al proporcionar la lista de todos los libros de la base de datos. Podría responder a una solicitud POST al mismo punto de conexión al agregar una entrada a la base de datos en función de los campos que haya rellenado el usuario en un formulario web.

En la aplicación web que compilará en breve, usará Express para enrutar las solicitudes HTTP y devolver el contenido web a su usuario. Express también puede ayudar a las aplicaciones web a trabajar con cookies HTTP y a procesar cadenas de consulta.

Express es un paquete de Node.js. Usará la utilidad npm que viene con Node.js para instalar y administrar paquetes de Node.js. Más adelante en esta parte, creará un archivo denominado package.json para definir Express y otras dependencias, y después ejecutará el comando npm install para instalar estas dependencias.

## **¿Qué ocurre con AngularJS?**

Como Express, aún no ha instalado AngularJS, la A del acrónimo MEAN.

AngularJS hace que las aplicaciones web sean fáciles de escribir y probar porque le permite separar mejor la *apariencia* de la página web, el código HTML, de cómo se comporta la página web. Si está familiarizado con el patrón Modelo-Vista-Controlador (MVC) o el concepto de enlace de datos, AngularJS le resultará familiar.

AngularJS es lo que denominamos un marco front-end de JavaScript, lo que significa que solo tiene que estar disponible en el cliente que accede a la aplicación. En otras palabras, AngularJS se ejecuta en el explorador web del usuario, no en su servidor web. Y dado que AngularJS es JavaScript, puede usarlo para capturar fácilmente los datos del servidor web para mostrarlos en la página.

En realidad, AngularJS no se *instala*. En su lugar, agrega una referencia al archivo JavaScript en la página HTML, tal y como se hace con otras bibliotecas de JavaScript. Hay varias maneras de incluir AngularJS en las páginas web. Aquí cargará AngularJS desde una red de entrega de contenido o CDN. Una red CDN es una forma de distribuir imágenes, vídeos y otro contenido de forma geográfica para mejorar las velocidades de descarga.

No agregue este código todavía, pero aquí tiene un ejemplo que carga AngularJS desde una red CDN. Normalmente, agregaría este código a la sección <head> de la página HTML.

HTML

Copiar

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.2/angular.min.js"></script>

Nota

No confunda AngularJS con Angular. Aunque muchos de los conceptos son similares en ambos, AngularJS es el predecesor de Angular. AngularJS se sigue usando habitualmente para compilar aplicaciones web. Mientras que AngularJS se basa en JavaScript, Angular se basa en TypeScript, un lenguaje de programación que facilita la escritura de programas de JavaScript.

## **¿Cómo compilo la aplicación?**

Aquí seguirá un proceso básico. Primero escribirá el código de la aplicación en Cloud Shell y, luego, usará el protocolo de copia de seguridad (SCP) para copiar los archivos en la máquina virtual. Después, iniciará la aplicación de Node.js y verá los resultados en el explorador.

En la práctica, normalmente escribiría y probaría la aplicación web en un entorno más local, como un portátil o una máquina virtual que ejecute de forma local. Después, podría almacenar el código en un sistema de control de versiones como Git y usar un sistema de integración continua y entrega continua, o CI/CD, como Azure DevOps para probar los cambios y cargarlos en la máquina virtual. Le indicaremos más recursos al final de este módulo.

## **Creación de la aplicación web de libros**

Aquí creará todo el código, los scripts y los archivos HTML que forman la aplicación web. Para mayor brevedad, resaltaremos las partes importantes de cada archivo, pero no entraremos en detalles.

Si aún está conectado a la máquina virtual por medio de SSH, ejecute exit para salir de la sesión de SSH y volver a Cloud Shell.

Bash

Copiar

exit

Ha vuelto a la sesión de Cloud Shell.

### **Creación de los archivos**

1. En Cloud Shell, ejecute estos comandos para crear las carpetas y los archivos de la aplicación web.
2. Bash
3. Copiar

cd ~

mkdir Books

touch Books/server.js

touch Books/package.json

mkdir Books/app

touch Books/app/model.js

touch Books/app/routes.js

mkdir Books/public

touch Books/public/script.js

touch Books/public/index.html

1. Esto es lo que incluye:
   * Books: el directorio raíz del proyecto.
     + server.js define el punto de entrada a la aplicación web. Carga los paquetes de Node.js necesarios, especifica el puerto en el que escuchar y empieza a escuchar el tráfico HTTP entrante.
     + package.json proporciona información sobre la aplicación, como su nombre, descripción y qué paquetes de Node.js debe ejecutar la aplicación.
   * Books/app: contiene código que se ejecuta en el servidor.
     + model.js define la conexión de base de datos y el esquema. Piense en ella como el modelo de datos de la aplicación.
     + routes.js controla el enrutamiento de solicitudes. Por ejemplo, define las solicitudes GET al punto de conexión /book al proporcionar la lista de todos los libros de la base de datos.
   * Books/public: contiene archivos que se entregan directamente en el explorador del cliente.
     + index.html contiene la página de índice. Contiene un formulario web que permite al usuario enviar información sobre los libros. También muestra todos los libros de la base de datos y le permite eliminar las entradas de la base de datos.
     + script.js contiene el código JavaScript que se ejecuta en el explorador del usuario. Puede enviar solicitudes al servidor para mostrar libros, agregar libros a la base de datos y eliminar libros de la base de datos.
2. Ejecute el comando code para abrir los archivos mediante el editor de Cloud Shell.
3. Bash
4. Copiar

code Books

### **Creación del modelo de datos**

1. En el editor, abra app/model.js y agregue lo siguiente.
2. JavaScript
3. Copiar

var mongoose = require('mongoose');

var dbHost = 'mongodb://localhost:27017/Books';

mongoose.connect(dbHost, { useNewUrlParser: true } );

mongoose.connection;

mongoose.set('debug', true);

var bookSchema = mongoose.Schema( {

name: String,

isbn: {type: String, index: true},

author: String,

pages: Number

});

var Book = mongoose.model('Book', bookSchema);

module.exports = Book;

1. Importante  
   Cada vez que pegue o cambie código en un archivo en el editor, no olvide guardarlo posteriormente mediante el menú "..." o la tecla de aceleración (CTRL+S en Windows y Linux, o Comando+S en macOS).  
   Este código usa Mongoose para simplificar el proceso de transferencia de datos dentro y fuera de MongoDB. Mongoose es un sistema basado en esquemas para modelar datos. El código define un documento de base de datos denominado "Book" (Libro) con el esquema proporcionado. El esquema define cuatro campos que describen un único libro:
   * El nombre o título del libro.
   * Su Número Internacional Normalizado del Libro o ISBN, que identifica el libro de forma única.
   * Su autor.
   * El número de páginas que contiene.
2. Después, creará los controladores HTTP que asignan las solicitudes GET, POST y DELETE a las operaciones de base de datos.

### **Creación de las rutas de Express que controlan las solicitudes HTTP**

1. En el editor, abra app/routes.js y agregue el siguiente código.
2. JavaScript
3. Copiar

var path = require('path');

var Book = require('./model');

var routes = function(app) {

app.get('/book', function(req, res) {

Book.find({}, function(err, result) {

if ( err ) throw err;

res.json(result);

});

});

app.post('/book', function(req, res) {

var book = new Book( {

name:req.body.name,

isbn:req.body.isbn,

author:req.body.author,

pages:req.body.pages

});

book.save(function(err, result) {

if ( err ) throw err;

res.json( {

message:"Successfully added book",

book:result

});

});

});

app.delete("/book/:isbn", function(req, res) {

Book.findOneAndRemove(req.query, function(err, result) {

if ( err ) throw err;

res.json( {

message: "Successfully deleted the book",

book: result

});

});

});

app.get('\*', function(req, res) {

res.sendFile(path.join(\_\_dirname + '/public', 'index.html'));

});

};

module.exports = routes;

1. Este código crea cuatro rutas para la aplicación. Aquí tiene una breve descripción de cada una.

| **Verbo HTTP** | **Punto de conexión** | **Descripción** |
| --- | --- | --- |
| GET | /book | Recupera todos los libros de la base de datos. |
| POST | /book | Crea un objeto Book en función de los campos que haya proporcionado el usuario en el formulario web y escribe ese objeto en la base de datos. |
| DELETE | /book/:isbn | Elimina el libro tal y como identifica su ISBN de la base de datos. |
| GET | \* | Devuelve la página de índice cuando no coincide ninguna otra ruta. |

1. Express puede entregar respuestas HTTP directamente al código de control de ruta, o bien puede entregar contenido estático desde los archivos. Este código muestra ambas opciones. Las tres primeras rutas devuelven datos JSON para las solicitudes de la API de libros. La cuarta ruta (el caso predeterminado) devuelve el contenido del archivo de índice index.html.

### **Creación de la aplicación de JavaScript del lado cliente**

1. En el editor, abra public/script.js y agregue este código:
2. JavaScript
3. Copiar

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope, $http) {

var getData = function() {

return $http( {

method: 'GET',

url: '/book'

}).then(function successCallback(response) {

$scope.books = response.data;

}, function errorCallback(response) {

console.log('Error: ' + response);

});

};

getData();

$scope.del\_book = function(book) {

$http( {

method: 'DELETE',

url: '/book/:isbn',

params: {'isbn': book.isbn}

}).then(function successCallback(response) {

console.log(response);

return getData();

}, function errorCallback(response) {

console.log('Error: ' + response);

});

};

$scope.add\_book = function() {

var body = '{ "name": "' + $scope.Name +

'", "isbn": "' + $scope.Isbn +

'", "author": "' + $scope.Author +

'", "pages": "' + $scope.Pages + '" }';

$http({

method: 'POST',

url: '/book',

data: body

}).then(function successCallback(response) {

console.log(response);

return getData();

}, function errorCallback(response) {

console.log('Error: ' + response);

});

};

});

1. Observe cómo este código define un módulo denominado "myApp" y un controlador denominado "myCtrl". No entraremos en detalles sobre cómo funcionan aquí el módulo y los controladores, pero usará estos nombres en el paso siguiente para enlazar la interfaz de usuario (código HTML) con la lógica de negocios de la aplicación.  
   Antes ha creado cuatro rutas que controlan varias operaciones GET, POST y DELETE en el servidor. Este código es similar a esas mismas operaciones, pero desde el lado cliente (el explorador web del usuario).  
   La función getData, por ejemplo, envía una solicitud GET al punto de conexión /book. Recuerde que el servidor controla esta solicitud al recuperar información sobre todos los libros de la base de datos y devolver esa información como datos JSON. Observe cómo los datos JSON resultantes se asignan a la variable $scope.books. Verá cómo esto afecta a lo que ve el usuario en la página web en el paso siguiente.  
   Este código llama a la función getData cuando se carga la página. Puede examinar las funciones del\_book y add\_book para hacerse una idea de cómo funcionan. No necesita que el código del lado cliente coincida con el controlador predeterminado del servidor porque el controlador predeterminado devuelve la página de índice y no los datos JSON.

### **Creación de la interfaz del usuario**

1. En el editor, abra public/index.html y agregue este código:
2. HTML
3. Copiar

<!doctype html>

<html ng-app="myApp" ng-controller="myCtrl">

<head>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.2/angular.min.js"></script>

<script src="script.js"></script>

</head>

<body>

<div>

<table>

<tr>

<td>Name:</td>

<td><input type="text" ng-model="Name"></td>

</tr>

<tr>

<td>Isbn:</td>

<td><input type="text" ng-model="Isbn"></td>

</tr>

<tr>

<td>Author:</td>

<td><input type="text" ng-model="Author"></td>

</tr>

<tr>

<td>Pages:</td>

<td><input type="number" ng-model="Pages"></td>

</tr>

</table>

<button ng-click="add\_book()">Add</button>

</div>

<hr>

<div>

<table>

<tr>

<th>Name</th>

<th>Isbn</th>

<th>Author</th>

<th>Pages</th>

</tr>

<tr ng-repeat="book in books">

<td><input type="button" value="Delete" data-ng-click="del\_book(book)"></td>

<td>{{book.name}}</td>

<td>{{book.isbn}}</td>

<td>{{book.author}}</td>

<td>{{book.pages}}</td>

</tr>

</table>

</div>

</body>

</html>

1. Este código crea un formulario HTML básico con cuatro campos para enviar datos de libros y una tabla que muestra todos los libros que están almacenados en la base de datos.  
   Aunque se trata de código HTML estándar, puede que no conozca los atributos HTML ng-. Estos atributos HTML conectan el código de AngularJS con la interfaz de usuario. Por ejemplo, al seleccionar Agregar, AngularJS llama a la función add\_book, que envía los datos del formulario al servidor.  
   Puede examinar el código aquí para hacerse una idea de cómo se relaciona cada uno de los atributos ng- con la lógica de negocios de la aplicación.

### **Creación del servidor de Express que hospedará la aplicación**

1. En el editor, abra server.js y agregue este código.
2. JavaScript
3. Copiar

var express = require('express');

var bodyParser = require('body-parser');

var app = express();

app.use(express.static(\_\_dirname + '/public'));

app.use(bodyParser.json());

require('./app/routes')(app);

app.set('port', 80);

app.listen(app.get('port'), function() {

console.log('Server up: http://localhost:' + app.get('port'));

});

1. Este código crea la propia aplicación web. Entrega archivos estáticos del directorio public y usa las rutas que ha definido anteriormente para controlar las solicitudes.

### **Definición de las dependencias y la información del paquete**

Recuerde que package.json proporciona información sobre la aplicación, como su nombre, descripción y qué paquetes de Node.js debe ejecutar la aplicación.

1. En el editor, abra package.json y agregue este código.
2. JSON
3. Copiar

{

"name": "books",

"description": "Sample web app that manages book information.",

"license": "MIT",

"repository": {

"type": "git",

"url": "https://github.com/MicrosoftDocs/mslearn-build-a-web-app-with-mean-on-a-linux-vm"

},

"main": "server.js",

"dependencies": {

"express": "~4.16",

"mongoose": "~5.3",

"body-parser": "~1.18"

}

}

Verá información o metadatos sobre la aplicación, como su nombre, descripción y licencia.

El campo repository especifica dónde se mantiene el código. Como referencia, más adelante puede revisar el código en GitHub en la URL que se muestra aquí.

El campo main define el punto de entrada de la aplicación. Se proporciona aquí para completar, pero no es importante porque no tiene previsto publicar la aplicación como un paquete de Node.js para que otros usuarios la descarguen y usen.

El campo dependencies es importante. Define los paquetes de Node.js que necesita la aplicación. En resumen, se conectará a la máquina virtual una segunda vez y ejecutará el comando npm install para instalar estos paquetes.

Los paquetes de Node suelen usar el esquema de control de versiones [Versionamiento Semántico](https://semver.org/). El número de versión contiene tres componentes: versión principal, versión secundaria y revisión. La notación de tilde ~ indica a npm que instale la versión de revisión más reciente de las versiones principales y secundarias proporcionadas. Las versiones que ve aquí son las más recientes con las que se ha probado este módulo. En la práctica, puede aumentar la versión con el paso del tiempo conforme actualiza y prueba la aplicación para usar las características más recientes que proporciona cada paquete dependiente.

### **Copia de los archivos en la máquina virtual**

1. Ya ha terminado de editar los archivos. Asegúrese de que guarda los cambios de cada archivo y después cierre el editor.  
   Para cerrar el editor, seleccione los puntos suspensivos situados en la esquina y luego Cerrar editor.
2. Ejecute el siguiente comando scp para copiar el contenido del directorio ~/Books de la sesión de Cloud Shell en el mismo nombre de directorio en la máquina virtual.
3. Bash
4. Copiar

scp -r ~/Books azureuser@$ipaddress:~/Books

## **Instalación de los paquetes adicionales de Node**

Supongamos que, durante el proceso de desarrollo, ha identificado los paquetes adicionales de Node que quiere usar. Por ejemplo, recuerde que app/model.js comienza con esta línea.

JavaScript

Copiar

var mongoose = require('mongoose');

Recuerde que la aplicación usa Mongoose para ayudar a transferir datos dentro y fuera de la base de datos de MongoDB.

La aplicación también requiere Express y los paquetes de body-parser. body-parser es un complemento que permite que Express trabaje con datos del formulario web que ha enviado el cliente.

Conéctese a la máquina virtual e instale los paquetes que ha especificado en package.json.

1. Antes de conectarse a la máquina virtual, asegúrese de que tiene a mano la dirección IP de la máquina virtual. Si no la tiene, ejecute estos comandos en Cloud Shell para recuperarla.
2. CLI de Azure
3. Copiar

ipaddress=$(az vm show \

--name MeanStack \

--resource-group [sandbox resource group name] \

--show-details \

--query [publicIps] \

--output tsv)

1. Bash
2. Copiar

echo $ipaddress

1. Como ha hecho antes, cree una conexión SSH a la máquina virtual.
2. Bash
3. Copiar

ssh azureuser@$ipaddress

1. Vaya al directorio Books dentro del directorio de inicio.
2. Bash
3. Copiar

cd ~/Books

1. Ejecute npm install para instalar los paquetes dependientes.
2. Bash
3. Copiar

sudo apt install npm

npm install

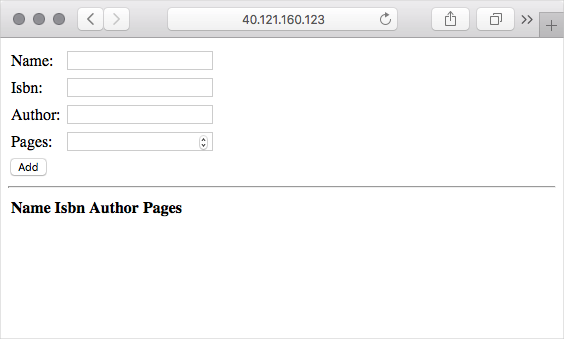
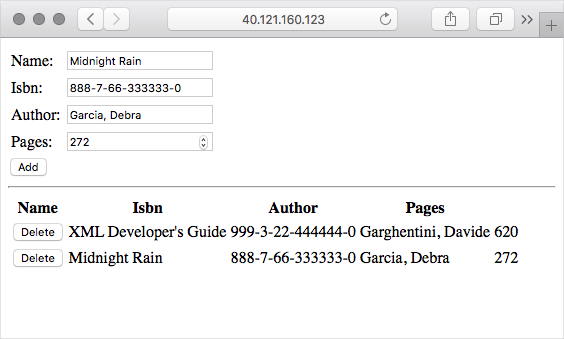
Mantenga la conexión SSH abierta para la parte siguiente.

## **Prueba de la aplicación**

Ya está listo para probar su aplicación web de Node.js.

1. En el directorio ~/Books, ejecute este comando para iniciar la aplicación web.
2. Bash
3. Copiar

sudo nodejs server.js

1. Este comando inicia la aplicación al escuchar en el puerto 80 las solicitudes HTTP entrantes.
2. En otra pestaña del explorador, vaya a la dirección IP pública de su máquina virtual.  
   Verá la página de índice, que incluye un formulario web.  
     
   Intente agregar algunos libros a la base de datos. Cada vez que agrega un libro, la página actualiza la lista completa de libros.  
     
   Para eliminar un libro de la base de datos, también puede seleccionar Eliminar.

# **Resumen**

Completado

200 XP

* 3 minutos

MEAN es una pila de desarrollo para compilar y hospedar aplicaciones web. Recuerde que MEAN es el acrónimo de sus componentes: MongoDB, Express, AngularJS y Node.js.

En este módulo ha aprendido cuándo la pila MEAN es una buena elección para el desarrollo web y cuándo puede resultar conveniente elegir otra cosa. La razón principal por la que podría valorar la posibilidad de usar MEAN es que está familiarizado con JavaScript.

Para ver la pila MEAN en acción, ha creado una máquina virtual Ubuntu en Azure y ha instalado en ella la pila MEAN para el desarrollo web.

Con la pila MEAN preparada, ha creado una aplicación web de inventario de libros básica. En resumen, la aplicación web usa:

* MongoDB para almacenar información sobre libros.
* Express para enrutar cada solicitud HTTP al controlador adecuado.
* AngularJS para conectar la interfaz de usuario con la lógica de negocios del programa.
* Node.js para hospedar la aplicación del lado servidor.

Puede [encontrar el código fuente de la aplicación web](https://github.com/MicrosoftDocs/mslearn-build-a-web-app-with-mean-on-a-linux-vm) en GitHub.

## **Limpieza**

El espacio aislado limpia los recursos automáticamente cuando haya terminado con este módulo.

Al trabajar en una suscripción propia, se recomienda identificar al final de un proyecto si aún necesita los recursos creados. Los recursos que se dejan en ejecución pueden costarle mucho dinero. Puede eliminar los recursos de forma individual o eliminar el grupo de recursos para eliminar todo el conjunto de recursos.

## **Más información**

En este módulo, se ha hecho una idea de cómo funciona la pila MEAN y ha creado una aplicación web básica que la usa. El siguiente paso es comenzar a compilar aplicaciones que solucionen sus propios desafíos empresariales. Después, puede implementar las aplicaciones en Azure y usar procesos automatizados para supervisar las aplicaciones y mejorarlas. Estos son algunos recursos en los que puede obtener más información.

### **Más información sobre el desarrollo de aplicaciones de la pila MEAN**

Obtenga más información sobre los componentes de la pila MEAN y otros paquetes de Node.js que ha usado en este módulo.

* [MongoDB](https://www.mongodb.com/)
* [Express](https://expressjs.com/)
* [AngularJS](https://angularjs.org/)
* [Node.js](https://nodejs.org/)
* [npm](https://www.npmjs.com/)
* [Mongoose](https://www.npmjs.com/package/mongoose)
* [body-parser](https://www.npmjs.com/package/body-parser)

### **Información sobre el servicio Azure Web Apps**

En este módulo, ha usado una máquina virtual para hospedar su aplicación web. Una máquina virtual le ofrece más control sobre el entorno y puede que se adapte mejor a cómo administra actualmente las implementaciones, pero hay otras formas de hospedar aplicaciones web. Consulte [Creación de una aplicación web de Node.js en Azure](https://docs.microsoft.com/es-es/azure/app-service/app-service-web-get-started-nodejs) para aprender a usar el servicio Azure Web Apps para simplificar las implementaciones.

### **Automatización de implementaciones**

También en este módulo, ha usado un proceso principalmente manual para configurar la máquina virtual y ejecutar la aplicación. A medida que crezca el proceso, puede usar un proceso más automatizado para implementar los cambios de forma más rápida y confiable. Consulte [Creación de una canalización de CI/CD para Node.js con Azure DevOps Projects](https://docs.microsoft.com/es-es/azure/devops-project/azure-devops-project-nodejs) para aprender a usar Azure DevOps a fin de implementar la aplicación de Node.js como parte de una canalización de integración continua y entrega continua (CI/CD).