# Despliegue de aplicaciones en servidores remotos

Ahora que ya sabemos cómo configurar un servidor remoto para acceder a él, veremos en este documento cómo instalar el software necesario para desplegar aplicaciones web en dicho servidor.

> **NOTA IMPORTANTE**: conviene que no instales nada de lo que se explica aquí directamente en el VPS, ya que se dan las nociones importantes para instalar diversos servidores web (Apache, Nginx), diversas bases de datos (MariaDB, MongoDB), pero posiblemente no vayas a tener todas esas cosas en el servidor. Limítate a instalar lo que se indica en los ejercicios nada más, que será lo que utilicemos en el curso.

## 1. Pasos principales para despliegue de aplicaciones Node.js

En primer lugar veremos qué pasos son los recomendables para poder desplegar cómodamente aplicaciones Node.js en servidores remotos.

### 1.1. Instalación de Apache

Apache es, hoy por hoy, el servidor más utilizado para desarrollo de aplicaciones web. Su instalación y configuración varían ligeramente de un sistema operativo a otro, especialmente en cuanto al nombre y ubicación de los ejecutables y de los ficheros de configuración. Aquí veremos el caso que nos ocupa: instalarlo en una distribución Debian para nuestro VPS.

Para instalar Apache en Debian, escribimos este comando:

```
sudo apt-get install apache2
```

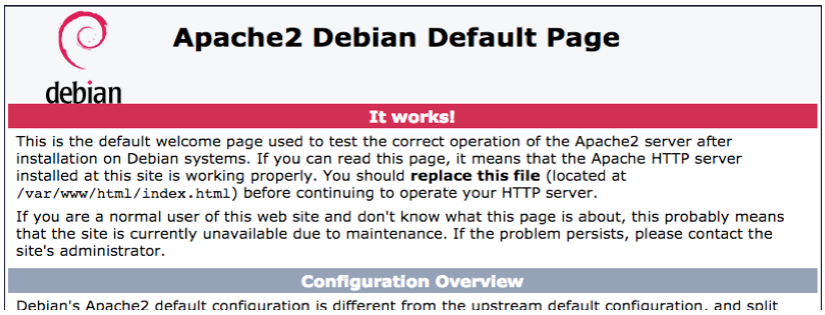
Automáticamente, Apache quedará instalado como servicio, y se iniciará con el sistema operativo. Podemos iniciar, detener o reiniciar el servidor con estos comandos (la opción `reload` es más ligera y menos profunda que `restart`, y se emplea para recargar los archivos de configuración simplemente):

```
sudo systemctl start apache2
sudo systemctl stop apache2
sudo systemctl restart apache2
sudo systemctl reload apache2
```

O también con estos otros:

```
sudo /etc/init.d/apache2 start
sudo /etc/init.d/apache2 stop
sudo /etc/init.d/apache2 reload
sudo /etc/init.d/apache2 restart
```

Por defecto, Apache queda escuchando en el puerto 80, así que para probar que está en marcha, podemos acceder a la URL que nos dieron para nuestro VPS al registrar (por ejemplo, *vps-xxxxxxxx.vps.ovh.net*). Si hemos contratado un nombre de dominio y lo hemos asociado a la IP de nuestro VPS, también lo podemos usar para acceder a Apache. Veremos una página de bienvenida similar a ésta:




### 1.1.1. Configuración básica

La configuración de Apache en Linux Debian se encuentra distribuida en varios archivos, todos ellos ubicados en la carpeta `/etc/apache2`. Antes de editar cualquiera de estos archivos, conviene hacer una copia de seguridad de su estado original, por si los cambios dejan al archivo con errores, y Apache no puede iniciar. Por ejemplo:

```
sudo cp /etc/apache2/apache2.conf /etc/apache2/apache2.conf.original
```

El **archivo principal de configuración** de Apache, dentro de la carpeta anterior, es `apache2.conf`. En él existen referencias a los otros archivos de configuración, que permiten gestionar puertos, hosts virtuales y otros elementos. Dentro de este archivo principal en sí, podemos configurar cosas como:

- La carpeta por defecto de los archivos de configuración (parámetro `ServerRoot`)
- La carpeta raíz donde se ubicarán los documentos web (parámetro `DocumentRoot`). Por defecto, el valor de este último elemento es `var/www/html`.
- Nombres de archivos por defecto a cargar si no se especifica ninguno en la URL (parámetro `DirectoryIndex`).
- Archivo donde se almacenarán los mensajes de error producidos (parámetro `ErrorLog`, cuyo valor por defecto es `/var/log/apache2/error.log`)
- ... etc.

El **archivo de gestión de puertos** también se encuentra dentro de la carpeta `/etc/apache2`. En concreto, es el archivo `ports.conf`, y contiene los puertos que se habilitan para trabajar con Apache. Por defecto,

hay dos puertos habilitados:

- El puerto 80 para conexiones HTTP
- El puerto 443 para conexiones HTTPS

Podemos editar cualquiera de estos puertos para hacer que Apache escuche estas conexiones por otros puertos, y podemos habilitar más que esos dos puertos, añadiendo más directivas `Listen` a las que ya hay.

> **Ejercicio 1:**
>
> Instala Apache en tu VPS siguiendo las instrucciones indicadas en este apartado. Edita el fichero `ports.conf` para habilitar también los puertos 8000 y 8080. Reinicia el servidor para que incluya los cambios.

### 1.1.2. Definiendo hosts virtuales

Los hosts virtuales permiten que un solo ordenador pueda alojar múltiples dominios y páginas web, de forma que una sola dirección IP puede responder a diferentes nombres de dominio. También un host virtual permite asignar un subdominio a una aplicación concreta, ubicada en una carpeta determinada. Por ejemplo, si nuestro dominio es *daw.ovh*, podemos definir un subdominio llamado *pruebas.daw.ovh*, y colocar la web en la carpeta */home/usuario/pruebas*, por ejemplo.

En la carpeta `/etc/apache2/sites-available` se tienen definidos los hosts virtuales, y en la carpeta `/etc/apache2/sites-enable` se tienen enlaces simbólicos a los hosts de la carpeta anterior que están activados o habilitados actualmente.

El servidor web Apache se instala por defecto con un host virtual ya definido. La configuración de este host la podemos encontrar en `/etc/apache2/sites-available/000-default.conf`, y alude a la carpeta de documentos por defecto `/var/www/html`. Por defecto este sitio virtual está habilitado, por lo que podemos comprobar que existe un enlace simbólico a este fichero en el directorio `/etc/apache2/sites-enable`.

Supongamos que queremos configurar un *host* virtual para el subdominio *maycalle.ovh* (que ya tendremos registrado previamente en OVH), de forma que atienda peticiones en el puerto 80 y cargue la web que tenemos almacenada en `/home/usuario/librosweb`. En este caso, debemos crear un archivo de configuración en `/etc/apache2/sites-available`. Podemos copiar el que ya hay y darle otro nombre:

```
cd /etc/apache2/sites-available
cp 000-default.conf 001-librosweb.conf
```

Después, editamos este nuevo archivo `001-librosweb.conf` y lo configuramos con los datos del nombre de dominio o subdominio asociado, y carpeta de la web. Añadimos también un bloque `<Directory>` para dar permisos de acceso a la carpeta en cuestión.

```
<VirtualHost *:80>
    ServerAdmin may.calle@gmail.com
    ServerName maycalle.ovh
    DocumentRoot "/home/debian/librosweb"
    DirectoryIndex index.js
    <Directory "/home/debian/librosweb">
        Require all granted
    </Directory>
</VirtualHost>
```

Finalmente, debemos habilitar este nuevo sitio, para lo que usaremos el comando `ln` para crear un *alias* del archivo en la carpeta `/etc/apache2/sites-enabled`. El siguiente comando se escribe en una sola línea:

```
sudo ln -s /etc/apache2/sites-available/001-librosweb.conf
/etc/apache2/sites-enabled
```

Para deshabilitar el sitio simplemente tenemos que borrar el enlace simbólico de la otra carpeta.

Tras los cambios realizados, deberemos reiniciar el servidor para que los incorpore:

```
sudo systemctl reload apache2
```

Veremos que al visitar `maycalle.ovh` en el navegador se muestra el contenido del archivo *index.js* en lugar de ejecutarse, esto es porque Apache, por defecto, no procesa archivos *.js* a diferencia de los archivos *.php*. Por lo tanto, Apache está sirviendo el archivo index.js como un archivo estático. Una posible solución es configurar Apache para actuar como un proxy inverso que reenvíe las solicitudes al puerto donde se ejecuta tu aplicación Node.js o utilizar **Phusion Passenger**, que veremos más adelante.

> **Ejercicio 2:**
>
> Para realizar este ejercicio seguiremos estos pasos:
>
> 1. Descarga este ejemplo de proyecto Node.js que hemos utilizado en alguna sesión anterior. Contiene un código simple para probar la autenticación basada en sesiones con Express (sin base de datos). Descomprímelo en una carpeta
> 2. Súbelo a un repositorio tuyo en GitHub. Para ello:
>    - Crea un repositorio nuevo con el nombre que quieras (por ejemplo *PruebaSesionesExpress*), y añade un fichero *README*
>    - Clona el repositorio en tu ordenador, y copia dentro el código del proyecto descomprimido del paso 1.
>    - Añade un fichero `.gitignore` que incluya la carpeta `node_modules`
>    - Haz un *Commit* y un *Push* para subir los cambios a GitHub

3. Abre sesión en tu VPS y descarga el proyecto en tu carpeta de usuario (utiliza *git clone* para clonar el repositorio la primera vez, y *git pull* si haces cambios las siguientes veces).
4. Añade un *virtual host* en Apache (carpeta */etc/apache2/sites-available*) para que acceda a esa carpeta por el puerto 8080:

```
<VirtualHost *:8080>
    ServerAdmin tu_correo@gmail.com
    ServerName vps-XXXXXX-vps.ovh.net
    DocumentRoot "/home/debian/PruebaSesionesExpress"
    DirectoryIndex index.js
    <Directory "/home/debian/PruebaSesionesExpress">
        Require all granted
    </Directory>
</VirtualHost>
```

5. Recuerda añadir también el enlace simbólico en la carpeta *sites-enabled* Reinicia el servidor y prueba a cargar el fichero *index.js* (se mostrará directamente el código fuente)

## 1.2. Node.js

A la hora de instalar Node.js, podemos emplear el gestor de versiones *NVM* o bien la distribución correspondiente de Node para nuestro sistema operativo. En este documento se explica detalladamente cómo instalar una u otra opción.

> **Ejercicio 3:**
>
> Instala Node.js (la última versión LTS disponible) en tu VPS. Es preferible que lo instales desde la web oficial, sin usar NVM, siguiendo estas instrucciones para Linux.

## 1.3. Enlazar Node con Apache a través del servidor Passenger

Es habitual que en un servidor remoto coexistan distintos servidores web, como por ejemplo Node y Apache, o Apache y Nginx. En este caso, no es muy recomendable tener que acceder a las aplicaciones Node por un puerto distinto al puerto por defecto, salvo que sean aplicaciones de prueba o para uso particular, pero tampoco podemos dejar escuchando a Node por el puerto 80 si ya hay otro servidor ocupando ese puerto... salvo que comuniquemos ambos servidores.

*Passenger* es un servidor de aplicaciones que permite trabajar con aplicaciones desarrolladas en varios frameworks, como por ejemplo Ruby o Node. Monitoriza las aplicaciones para volver a levantarlas si caen, y ofrece algunas ventajas adicionales, como el balanceo de carga. Además, dispone de módulos para integrarse con los servidores Apache o Nginx, de forma que, utilizando los mismos puertos, podemos hacer que estos servidores "pasen" a Passenger las aplicaciones que indiquemos en los correspondientes archivos de configuración.

### 1.3.1. Instalación de Passenger

Instalamos Passenger y el módulo para Apache a través del repositorio oficial. Cada comando *sudo* a continuación debe introducirse en una sola línea independiente del resto, en el mismo orden.

```
sudo apt-get install -y dirmngr gnupg

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
--recv-keys 561F9B9CAC40B2F7

sudo apt-get install -y apt-transport-https ca-certificates

sudo sh -c 'echo deb https://oss-binaries.phusionpassenger.com/
apt/passenger bullseye main > /etc/apt/sources.list.d/passenger.list'

sudo apt-get update

sudo apt-get install -y libapache2-mod-passenger
```

Habilitamos el módulo Passenger para apache, y reiniciamos el servidor

```
sudo a2enmod passenger
sudo systemctl restart apache2
```

Verificar la instalación antes de continuar.

```
sudo /usr/bin/passenger-config validate-install
```

### 1.3.2. Configuración de un *host virtual* con Passenger

Añadir un nuevo archivo de configuración de *virtual host* para nuestra aplicación Node:

```
sudo nano /etc/apache2/sites-available/appnode.conf
```

Editar el archivo para dejarlo con esta apariencia:

```
<VirtualHost *:80>

    ServerName tu_dominio.com

    DocumentRoot /home/usuario/web_node
    PassengerAppRoot /home/usuario/web_node

    PassengerAppType node
    PassengerStartupFile app.js

    <Directory /home/usuario/web_node>
      Allow from all
      Options -MultiViews
      Require all granted
    </Directory>

</VirtualHost>
```

Habilitar el nuevo *virtual host*, y reiniciar Apache para aceptar los cambios:

```
sudo ln -s /etc/apache2/sites-available/app.node.conf
/etc/apache2/sites-enabled

sudo systemctl reload apache2
```

Ahora nuestra nueva web ya debe estar visible y operativa. Podemos también comprobarlo con el comando `curl`, además del navegador:

```
curl http://www.tu_dominio.com
```

> **NOTA**: obviamente, tanto el dominio de `ServerName` como las carpetas indicadas en `DocumentRoot`, `PassengerAppRoot` y `Directory` las adaptaremos a las que requiera nuestra aplicación.

> **NOTA**: las propiedades `DocumentRoot` y `PassengerAppRoot` normalmente apuntarán a la misma carpeta, donde se encuentre el archivo principal de nuestra aplicación Node.js. Si estamos utilizando algún framework especial, como Nest.js, donde no hay un archivo principal específico (en este caso, la aplicación se lanza con *npm run* y alguna opción predefinida en el archivo *package.json*), entonces debemos construir la aplicación (en el caso de Nest.js, usamos `npm run build`), y las propiedades anteriores del archivo de configuración de Apache deben apuntar a la subcarpeta `dist` dentro del proyecto, siendo el archivo principal (propiedad `PassengerStartupFile`) el archivo `main.js`, dentro de dicha carpeta.

Estos pasos están sacados de la web oficial. También allí se puede obtener cómo instalar Passenger de forma autónoma (*standalone*), y también integrado con Nginx.

> **Ejercicio 4:**
>
> Completa los siguientes pasos:
>
> 1. Instala Passenger siguiendo los pasos indicados en este apartado
> 2. Modifica el *virtual host* que has creado antes en el *Ejercicio 2* para configurarlo con Passenger:
>
> ```
> <VirtualHost *:8080>
>     ServerAdmin tu_correo@gmail.com
>     ServerName vps-XXXXXX-vps.ovh.net
>     DocumentRoot "/home/debian/PruebaSesionesExpress"
>     PassengerAppRoot "/home/debian/PruebaSesionesExpress"
>     PassengerAppType node
>     PassengerStartupFile index.js
>     <Directory "/home/debian/PruebaSesionesExpress">
>         Allow from all
>         Options -MultiViews
>         Require all granted
>     </Directory>
> </VirtualHost>
> ```
>
> 3. Reinicia Apache
> 4. Ejecuta *npm install* en tu carpeta */home/debian/PruebaSesionesExpress* para que se instalen los paquetes necesarios.
> 5. Prueba a acceder de nuevo a la web (ruta raíz) y ahora ya debería verse todo correctamente.

# 2. Otras instalaciones adicionales o alternativas

Como alternativas o complementos a los pasos seguidos en el apartado anterior, aquí vamos a indicar otras opciones que podemos tener en cuenta a la hora de ser instaladas en el VPS.

## 2.1. Nginx

Cuando uno oye hablar de servidores web, posiblemente Apache sea el primero que le viene a la mente. Y no es casualidad, ya que en torno al 30 o 35% del tráfico web se sirve actualmente con este servidor. Sin embargo, en los últimos años han surgido alternativas al mismo, y una de las que ha llegado con más fuerza es Nginx. Actualmente sirve más del 20% de las webs existentes.

Nginx es un servidor HTTP ligero, de origen ruso. Inicialmente se creó para dar soporte a una web que recibía en torno a 500 millones de visitas diarias, y actualmente también se emplea como servidor base de otros sitios web conocidos, como Reddit o Dropbox.

Entre las características que podríamos citar de Nginx, las más relevantes son:

- Está basado en un **modelo de conexiones asíncrono**. Es decir, en lugar de colapsar la memoria con cientos de procesos que gestionan los cientos de conexiones simultáneas de una web, Nginx pone en marcha un proceso por núcleo de procesador, y todos ellos gestionan a la vez miles de conexiones. Esto permite una carga mucho menor de trabajo para la CPU, y menos consumo de memoria.
- Es **open source**, y funciona en distintas plataformas, aunque para su uso en producción se aconsejan distribuciones Linux, como Ubuntu, Debian o CentOS, entre otras.
- Está basado en **módulos**, un potente y eficaz sistema de plugins que permite dotar de funcionalidades adicionales al servidor, si se requiere. El propio núcleo de Nginx ya viene con una gran variedad de módulos incorporados, pero también podemos descargar otros módulos de terceros e instalarlos. En este sentido, tiene una filosofía similar a Node.js, por ejemplo.

Con todo lo anterior, podemos concluir que Nginx es una mezcla de eficiencia, velocidad y potencia, y por ello cada día es una alternativa más sólida a Apache.

### 2.1.1. Instalación y puesta en marcha

Veamos cómo descargar, instalar y poner en marcha una instancia básica de Nginx. Los primeros pasos son similares a cualquier otra herramienta bajo sistemas Linux: actualizamos repositorios e instalamos el paquete:

```
sudo apt update
sudo apt install nginx
```

> **NOTA**: si ya tenemos una instancia de otro servidor (por ejemplo, Apache), corriendo en el puerto 80, es conveniente que detengamos el servicio temporalmente, para que Nginx pueda completar su instalación y puesta en marcha en dicho puerto 80. Más adelante veremos cómo podemos cambiar el puerto y hacer que cada servidor tenga el suyo, si es necesario.

Por defecto, Nginx utiliza la misma carpeta de documentos web que Apache (`/var/www/html`), por lo que, si tenemos Apache ya instalado, Nginx utilizará esta página de inicio, y dejará la suya renombrada en la misma carpeta, en un segundo plano. Si Nginx es nuestro primer servidor, entonces al conectar al puerto 80 desde nuestro dominio (por ejemplo, *vps112233.ovh.net* o similar), veremos la página de bienvenida de Nginx.


El archivo de configuración de Nginx es `/etc/nginx/nginx.conf`. Internamente redirige o incluye otros archivos de configuración adicionales, y de hecho, gran parte de la configuración que necesitemos la haremos en esos archivos.

### 2.1.2. Añadir hosts virtuales

La mecánica de Nginx para habilitar sitios web es similar a la utilizada por Apache: existe una carpeta `/etc/nginx/sites-available` donde definir archivos de configuración para los distintos hosts virtuales que tengamos (además de la propia carpeta `/var/www/html`), y luego existe una carpeta paralela `/etc/nginx/sites-enabled` con enlaces simbólicos a los sitios de la carpeta anterior que queramos tener activos en cada momento.

Para añadir un nuevo host virtual en Nginx, basta con añadir un nuevo archivo a la carpeta `/etc/nginx/sites-available`. Ya existe un archivo `default` que apunta a la carpeta por defecto `/var/www/html`, y lo podemos utilizar para crear un duplicado, por ejemplo, `misitio.com`, con una apariencia como ésta:

```
server {
    listen 80;
    listen [::]:80;

    root /home/usuario/pruebas;
    index index.html index.htm;

    server_name misitio.com;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

- La directiva `listen` se emplea para indicar el puerto por el que escuchar. La segunda directiva `listen` con la sintaxis `[::]` se emplea para direcciones IPv6.
- La directiva `root` indica la carpeta raíz de los documentos de este host virtual
- La directiva `index` contiene las páginas que se van a cargar cuando se acceda a la carpeta de la web sin indicar ninguna. En este caso se buscará primero la página `index.html`, y si no se encuentra, se buscará `index.htm` en segundo lugar. También podríamos incluir aquí `index.php`, por ejemplo, si trabajamos con PHP.
- La directiva `server_name` es el nombre asociado al host virtual (normalmente, un alias o subdominio)
- Los grupos `location` se emplean para configuraciones particulares de ciertas rutas. En este caso, se emite un código 404 en el caso de que el archivo solicitado no se encuentre.

Una vez tenemos el sitio configurado, para hacerlo visible debemos crear un enlace simbólico de este archivo en la carpeta `sites-enabled`, con el siguiente comando (en una sola línea):

```
sudo ln -s /etc/nginx/sites-available/misitio.com
/etc/nginx/sites-enabled
```

### 2.1.3. Redirección de peticiones

En algunos tipos de aplicaciones, como por ejemplo aplicaciones SPA (*Single Page Applications*), o hechas con frameworks como Angular, puede resultar interesante redirigir cualquier URL que solicite el cliente a un único recurso (por ejemplo, a la página `index.html` en la raíz de la aplicación). Para eso, podemos crear un archivo de configuración (copia de `default.conf`, como en el caso anterior), como éste. La línea con la instrucción `rewrite` se encarga de la redirección propiamente dicha.

```
server {
    listen       80;
    server_name  misitio.com;

    root /home/debian/pruebas;
    rewrite ^\/.+$ /index.html last;

    location / {
    }
    ...
```

### 2.1.4. El archivo *favicon.ico* y los errores en el *log*

En algunos casos, al cargar una URL se tiende a buscar el archivo por defecto "favicon.ico", y si no se encuentra nos podemos encontrar con mensaje de error en el navegador, o con el correspondiente mensaje de log en el archivo de errores. Para evitar esto, podemos añadir esta directiva a la configuración de nuestro host virtual:

```
location = /favicon.ico {
    return 204;
    access_log     off;
    log_not_found  off;
}
```

Por otra parte, ante cualquier error en la ejecución de Nginx, se generarán los correspondientes mensajes en el archivo de logs, ubicado en `/var/log/nginx/error.log`. Podemos consultarlo si algo va mal para tener algo más de información sobre el error. Si es debido a un error de sintaxis en algún fichero de configuración, podemos tener alguna pista de en qué archivo y línea está el error.

## 2.2. Definir *hosts* virtuales desde Node.js

En la mayoría de ejemplos que hemos visto o veremos realizaremos una única aplicación Node que pondremos en marcha por sí misma. Pero es posible que en un mismo servidor tengamos coexistiendo varias aplicaciones, cada una de ellas asociada a un subdominio diferente. Si hemos configurado Passenger con

Apache podemos definir un *host virtual* para cada aplicación, con su dominio asociado, que apunte a cada carpeta.

Sin embargo, de forma alternativa, podemos gestionar todas las aplicaciones y *hosts virtuales* desde Node.js de otra forma. Para empezar, puede ser buena idea definir una carpeta donde ubicar todos los proyectos (en subcarpetas independientes). Podemos crear una carpeta llamada "*ProyectosNode*" dentro de nuestra carpeta de usuario, con una subcarpeta para cada proyecto.

Supongamos que definimos un primer proyecto llamado "*Principal*" en nuestra carpeta de "*ProyectosNode*". Instalamos Express en dicha carpeta, y dejamos un archivo `app.js` como este:

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

let app = express();

app.get('/', (req, res) => {
    res.send('Bienvenido a la web principal');
});

module.exports.app = app;
```

Del mismo modo, podemos crear otro proyecto, llamado "*Pruebas*", por ejemplo, en nuestra carpeta de "*ProyectosNode*". Instalamos nuevamente Express y dejamos el archivo `app.js` así:

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

let app = express();

app.get('/', (req, res) => {
    res.send('Bienvenido a la web de pruebas');
});

module.exports.app = app;
```

Ahora vamos a crear una tercera carpeta llamada "*Manager*", en la misma carpeta de "*ProyectosNode*". Instalamos también Express, y una librería adicional llamada `vhost`, que se encarga de asociar hosts virtuales con aplicaciones Express. El archivo `app.js` quedará como sigue:

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

let app = express();

app.use(vhost('nachoib.ovh',
              require(__dirname + '/../Principal/app').app));
app.use(vhost('pruebas.nachoib.ovh',
              require(__dirname + '/../Pruebas/app').app));
app.listen(3000);
```

En este último caso, lo que hacemos es asociar el nombre *nachoib.ovh* con la aplicación "*Principal*", y el subdominio *pruebas.nachoib.ovh* con el proyecto de "*Pruebas*". Si ponemos en marcha esta última aplicación, podemos acceder a las dos webs por separado con las correspondientes URLS:

- *http://nachoib.ovh:3000*
- *http://pruebas.nachoib.ovh:3000*

Alternativamente, podemos *mapear* Passenger con esta última aplicación *Manager* únicamente, y que sea ella la que se encargue de redirigir a una u otra app en función del dominio. Aunque, en este último caso, quizá sea más cómodo configurar *hosts* virtuales independientes en Apache con Passenger.

## 2.3. Automatizar la puesta en marcha del servidor Node con *forever*

Como alternativa al uso de *Passenger* con Apache, podemos optar por no instalar Apache y gestionar el arranque de las aplicaciones Node con la herramienta *forever* para que se inicien de forma automática al iniciar el sistema. Para ello, seguimos estos pasos (suponiendo que los hagamos como usuario *root*, de lo contrario deberemos anteponer el comando `sudo`). Tomaremos como ejemplo la aplicación *Manager* que hemos hecho en el apartado anterior.

1. Instalamos, si no lo tenemos ya, el módulo `forever` del repositorio de NPM. Este módulo permite rearrancar de forma automática una aplicación Node, incluso cuando se produce un error que la hace finalizar.

```
npm install forever -g
```

2. Creamos un script en la carpeta `/etc/init.d`, por ejemplo:

```
nano /etc/init.d/servidor-node
```

3. Editamos el archivo con este contenido (poniendo la carpeta donde estará el proyecto Node con el Manager, si no coincide con la que se indica a continuación):

```
#!/bin/sh
FOREVER="/usr/bin/forever"
APP_PATH="/home/usuario/ProyectosNode/Manager/app.js"
USER="usuario"
su - $USER -c "$FOREVER start $APP_PATH"
```

> **Nota:** Recuerda que debes cambiar el usuario y la ruta del proyecto de acuerdo a tus necesidades. En el VPS de OVH el usuario es "debian".

4. Hacemos el script ejecutable:

```
chmod +x /etc/init.d/servidor-node
```

5. Escribimos este comando para hacer los cambios permanentes al iniciar (es decir, que con cada inicio del sistema se arrancará el manager):

```
update-rc.d /etc/init.d/servidor-node defaults
```

6. Tras esto, debemos poner en marcha el servidor ( `/etc/init.d/servidor-node` ) o reiniciar el servidor VPS para que acepte los cambios introducidos.

## 2.4. Servidores de bases de datos

Vamos a mostrar los pasos para instalar un servidor de base de datos de nuestra elección. Veremos los casos de MariaDB y MongoDB.

### 2.4.1. MariaDB

Para instalar MariaDB en nuestro servidor VPS, accedemos por SSH como usuario root y escribimos el siguiente comando:

```
apt update
apt install mariadb-server
```

> **NOTA**: si queremos instalar MySQL en lugar de MariaDB, los pasos a seguir son los mismos, pero en el comando anterior cambiaremos el paquete `mariadb-server` por `mysql-server` .

Tras la instalación, podemos ejecutar con permisos de *root* el comando:

```
mysql_secure_installation
```

para asegurar algunos aspectos que quedan poco seguros en la instalación por defecto, como la contraseña de root o el acceso remoto o anónimo.

- Podemos establecer la contraseña de root en el primer paso de este asistente
- En el siguiente paso (Remove anonymous users), conviene responder que sí (Y)
- En el tercer paso (Disallow root login remotely), podemos elegir, aunque quizá convenga elegir que sí (Y)
- El siguiente paso (Remove test database and access to it), también podemos responder que sí (Y)
- Finalmente, recargamos la tabla de privilegios de los usuarios (Y) y ya está listo.

Una vez finalizada esta configuración, para iniciar, detener o reiniciar el servidor, escribiremos estos comandos, respectivamente:

/etc/init.d/mysql start /etc/init.d/mysql stop /etc/init.d/mysql restart

### 2.4.2. MongoDB

Para instalar MongoDB en VPS, lo propio es instalarlo como servicio. Los pasos a seguir siempre será mejor consultarlos en la página oficial. En nuestro caso, puedes seguir estos pasos: Instalación de la Edición Community de MongoDB en Debian 11.

Tras esto, ya podemos lanzar y detener Mongo con el servicio:

```
/etc/init.d/mongodb start
/etc/init.d/mongodb stop
/etc/init.d/mongodb restart
```

**Habilitar conexiones externas**

Podríamos utilizar un cliente para conectar a nuestro servidor remoto de MongoDB. Pero antes debemos habilitar MongoDB para aceptar conexiones remotas. Para ello, editamos el archivo `/etc/mongodb.conf` y comentamos esta línea:

```
#bind_ip = 127.0.0.1
```

Reiniciamos el servicio de MongoDB, y después iniciamos nuestro cliente MongoDB (por ejemplo, el *plugin* de Visual Studio Code, o la aplicación *Compass* oficial de Mongo) y creamos una nueva conexión a nuestro servidor (por ejemplo, *vps112233.ovh.net*) por el puerto que sea. Recordemos que el puerto por defecto es 27017, pero se puede modificar con la directiva `port` en el archivo de configuración anterior:

```
port = 27017
```

## 2.5. Instalar PHP y otras funcionalidades

Finalmente, vamos a instalar los módulos de PHP para poder trabajar tanto desde Apache como desde Nginx, si fuese el caso. También instalaremos la aplicación *phpMyAdmin* para gestionar las bases de datos MariaDB de forma remota con esta aplicación web.

### 2.5.1. Instalación de PHP

Explicaremos a continuación cómo instalar PHP de forma global, y cómo configurarla después para el caso concreto de Nginx (la configuración con Apache es inmediata).

En primer lugar, debemos ejecutar los siguientes comandos para descargar PHP del repositorio correspondiente:

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:ondrej/php
sudo apt update
```

Después, instalamos tanto `php7.4` como `php7.4-fpm`, que es la herramienta que utiliza Nginx para tratamiento de páginas PHP.

```
sudo apt install php7.4
sudo apt install php7.4-fpm
```

Podemos consultar la versión instalada con estos comandos, respectivamente:

```
php -v
php-fpm7.4 -v
```

De forma similar a como hemos instalado FPM, podemos instalar otras extensiones que puedan ser necesarias en un momento dado, haciendo el correspondiente `sudo apt install php7.4-XXXX`, siendo XXXX la extensión en sí (por ejemplo, `php7.4-common`, o `php7.4-mysql`). Aunque en principio no son necesarias para la instalación de base.

En lo que respecta a **Nginx**, nos faltaría habilitar PHP en él. Este paso no es tan automático y sencillo como en Apache, ya que Nginx no dispone de procesamiento PHP nativo, y requiere de unos pasos a seguir, una vez hemos instalado previamente el FPM (*FastCGI Process Manager*).

1. Abrimos el archivo `/etc/php/7.x/fpm/php.ini` (siendo la x la versión que tengamos instalada de PHP). Buscamos la línea de configuración `cgi.fix_pathinfo`, y la dejamos así, guardando el archivo al finalizar:

```
cgi.fix_pathinfo=0
```

2. También puede resultar conveniente, por problemas con permisos, hacer que se atiendan las peticiones PHP por TCP, y no a través de un archivo de socket especial que tiene php-fpm. Para ello, editamos el archivo `/etc/php/7.x/fpm/pool.d/www.conf` y comentamos esta línea y añadimos la que va a continuación:

```
;listen = /run/php/php7.0-fpm.sock
listen = 127.0.0.1:9000
```

3. Reiniciamos el procesador PHP para que acepte los nuevos cambios (sustituimos nuevamente la x por la versión que tengamos de PHP):

```
sudo systemctl restart php7.x-fpm
```

4. Indicamos a Nginx que las solicitudes a páginas PHP las pase al procesador que hemos instalado. Para ello, abrimos el archivo del host virtual que queramos configurar (por ejemplo, `/etc/nginx/sites-available/default`, o cualquier otro que hayamos creado) y hacemos lo siguiente:
    - En primer lugar, añadir el archivo index.php como uno de los posibles archivos de inicio:

```
index index.php index.html ...
```

- Descomentamos el siguiente bloque de directivas, y lo dejamos como sigue (sustituimos, de nuevo, la x por nuestra versión de PHP):

```
location ~ \.php$ {
    include /etc/nginx/fastcgi_params;
    fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
    fastcgi_pass 127.0.0.1:9000;
}
```

Tras estos pasos, reiniciamos Nginx y probamos a cargar algún contenido PHP básico, como la llamada a `phpinfo()`.

### 2.5.2. Administrar MySQL/MariaDB desde *phpMyAdmin*

La forma más sencilla de gestionar nuestras bases de datos MySQL (o MariaDB) quizá sea a través de la aplicación *phpMyAdmin*, el cliente web por excelencia para gestionar las bases de datos de este tipo. Se instala con el siguiente comando (en modo *root* o con permisos de superusuario):

```
apt-get install phpmyadmin
```

Durante la instalación, nos pedirá los siguientes datos:

- Si queremos configurar algún servidor automáticamente para phpMyAdmin. Nos da la opción de Apache (no la de Nginx), así que podemos elegir Apache para que lo configure automáticamente.
- Nos indicará que necesita tener una base de datos por defecto a la que acceder,y que si crea la base de datos por defecto. Podemos contestar que sí.
- Finalmente, nos pedirá el password para acceder a phpMyAdmin desde el navegador, y que lo confirmemos después.

Tras los pasos anteriores, ya podemos acceder a phpMyAdmin con el nombre de nuestro dominio y el puerto por el que esté escuchando Apache. Por ejemplo:

```
http://vps112233.ovh.net/phpmyadmin
```

En el caso de querer utilizar phpMyAdmin con **Nginx**, podemos hacer un enlace simbólico de la carpeta de instalación de phpMyAdmin (`/usr/local/phpmyadmin`) a la carpeta de nuestra aplicación web. Por ejemplo, si queremos acceder desde la carpeta general de aplicaciones de Nginx:

```
ln -s /usr/share/phpmyadmin /var/www/html
```

Después, editamos el archivo de configuración de esa carpeta (en nuestro ejemplo, sería el archivo `/etc/nginx/sites-available/default`, y añadimos la configuración para PHP, si no la tiene. También convendría dejar como globales la ruta raíz de la aplicación (*root*) y los archivos de inicio por defecto. Podría quedar algo así:

```
server {
    listen       80;
    server_name  localhost;

    root   /var/www/html;
    index  index.html index.php;

    location / {
    }

    location ~ \.php$ {
        include /etc/nginx/fastcgi_params;
        fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        fastcgi_pass 127.0.0.1:9000;
    }
    ...
```