

# REST



Attribution-NonCommercial-ShareAlike  
4.0 International (CC BY-NC-SA 4.0)

**paco@portadaalta.es**

# Índice

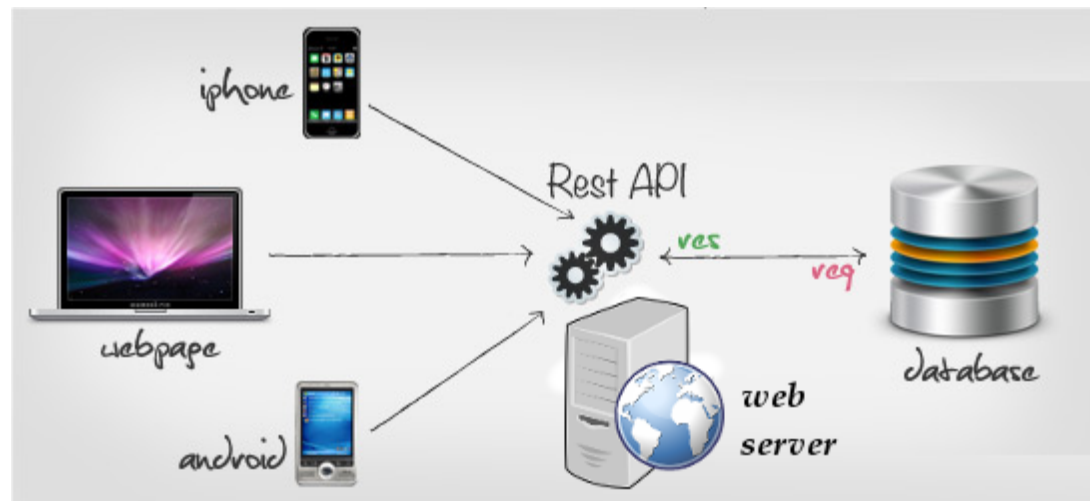
- Introducción
- REST
- Alojamiento web
- Laravel
- Cliente Android

# Introducción

Vamos a crear un servicio web que facilite el acceso a la información almacenada en la base de datos a través de un API RESTful service.

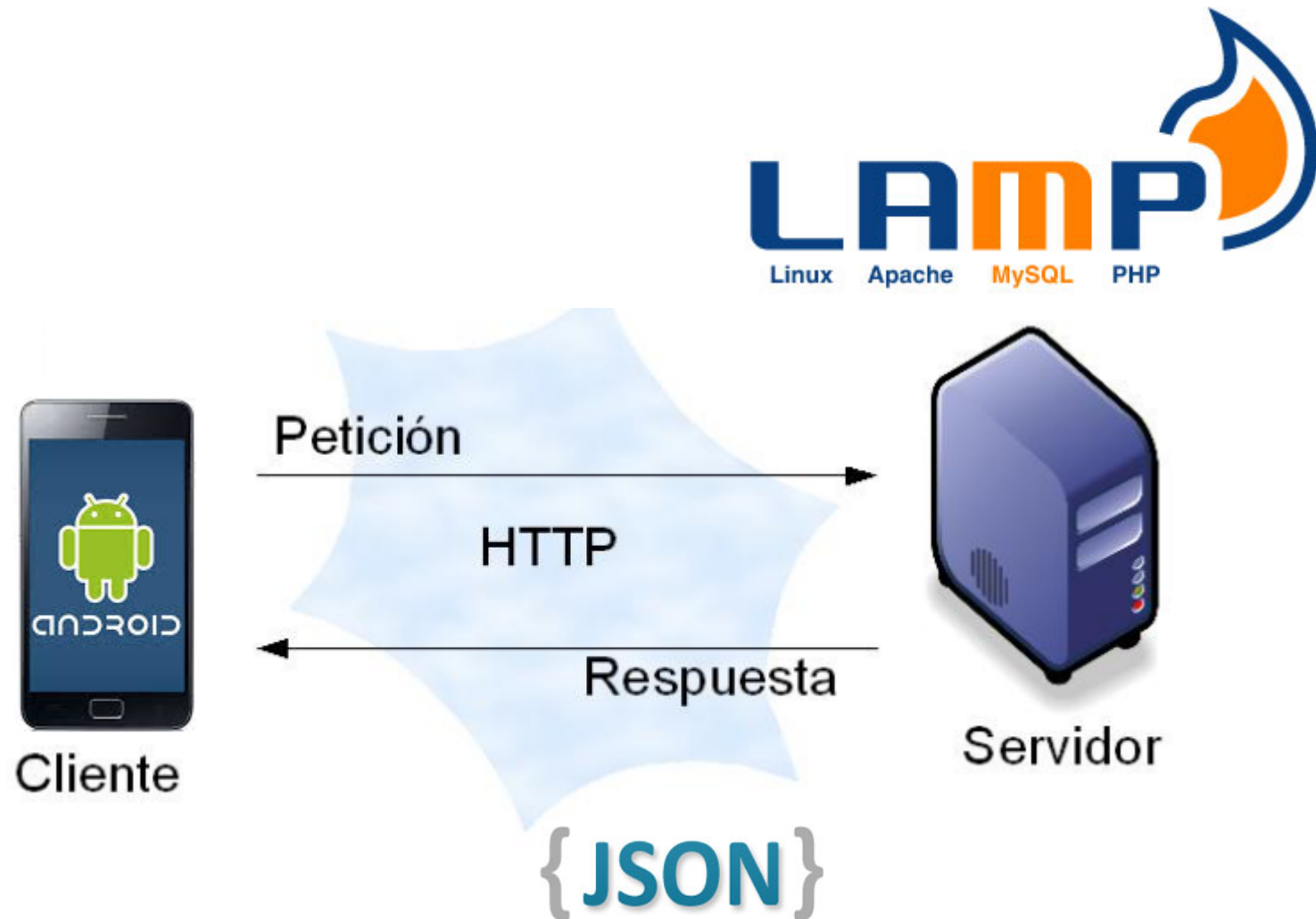
La base de datos estará situada en la red (en un servidor local o en Internet).

También crearemos una aplicación Android que se comunice con el servicio web y permita manejar la información en la base de datos.



# Introducción

¿Conocimientos previos?



# Introducción

## Servidor LAMP

**LAMP:**



**L**inux

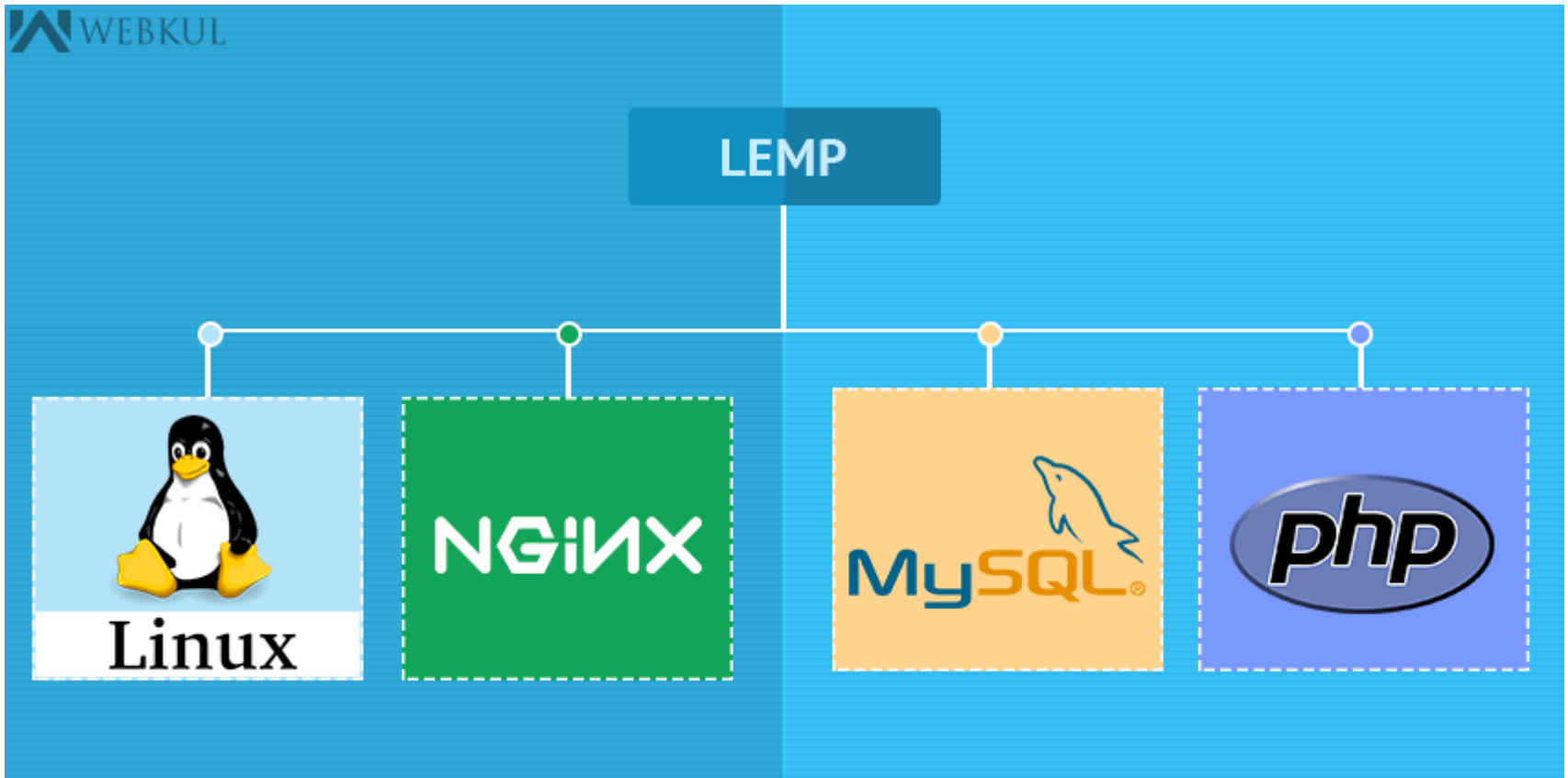


**A**pache



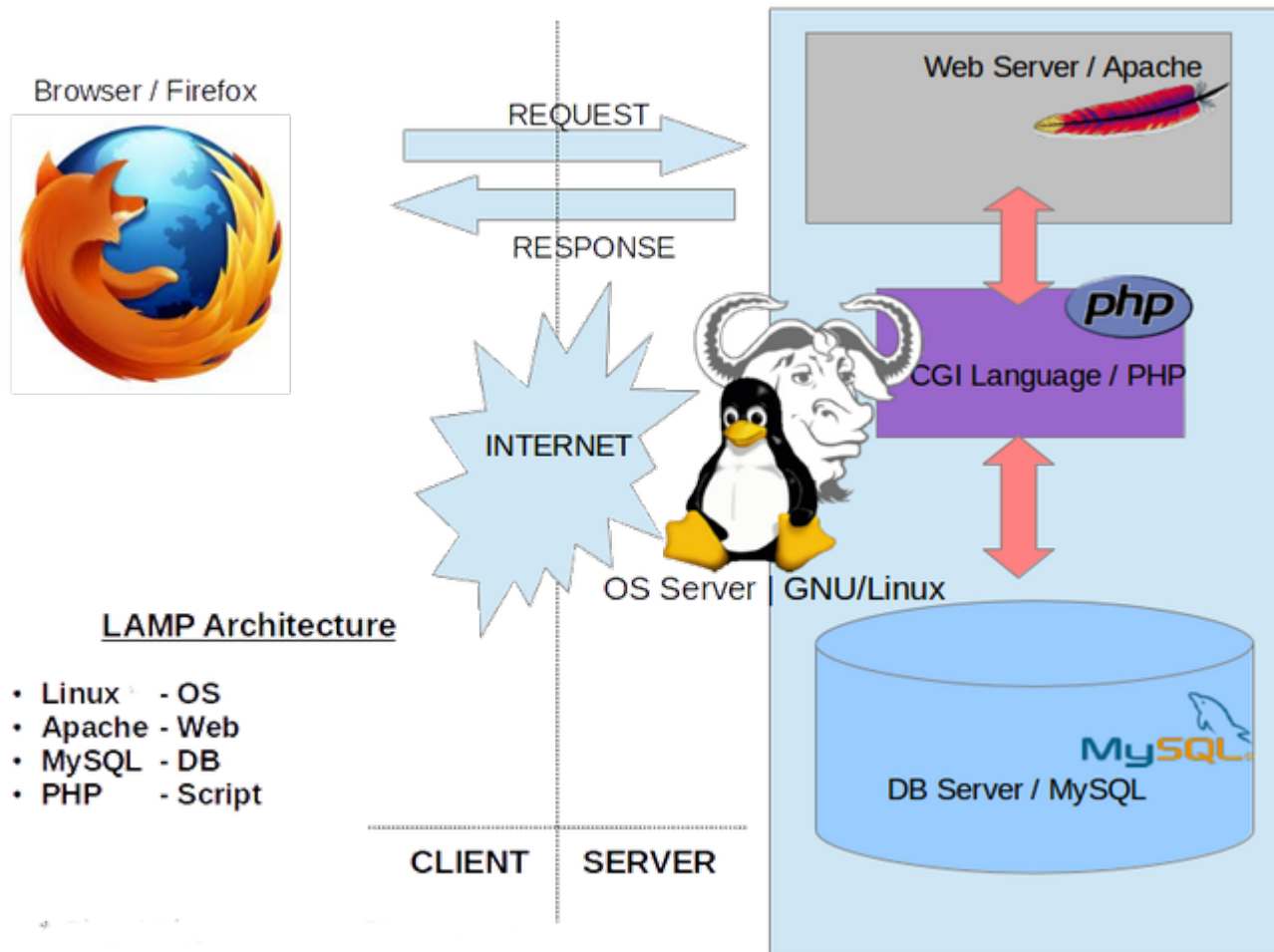
# Introducción

## Servidor LEMP



# Introducción

## Modelo Cliente - Servidor



# Introducción

## **Instalación del servidor LAMP**

[Cómo instalar LAMP en Ubuntu 16.04](#)

[Instalar LAMP en Ubuntu 16.04](#)



# Introducción

## **Instalación del servidor LEMP**

Parar el servidor web apache, si estuviese instalado:  
`sudo service apache2 stop`

[Cómo instalar LEMP en Ubuntu 16.04](#)

[Cómo instalar LEMP en Linux](#)

# Introducción

## Instalación del servidor LEMP

Configurar Nginx en el puerto 8080:

Fichero /etc/nginx/sites-available/default:

```
server {  
    listen 8080 default_server;  
    listen [::]:8080 default_server;  
    root /var/www/html;  
    # Crear la carpeta /var/www/nginx  
    # root /var/www/nginx;  
    location / {  
        autoindex on;  # crear un índice, si no existe index.html  
    }  
}
```

# Introducción

## Instalación del servidor LEMP

Comprobar la configuración de Nginx  
`sudo nginx -t`

Recargar el servidor Nginx:  
`sudo service nginx reload`

Reiniciar el servidor web apache:  
`sudo service apache2 start`

## Comprobación:

<http://localhost/>

<http://localhost:8080>

# Introducción

## Instalación de phpMyAdmin

How to install phpMyAdmin on Ubuntu 16.04

Configuración de phpMyAdmin con Nginx en ubuntu 16.04:

```
sudo apt-get update  
sudo apt-get install phpmyadmin  
(marcar la configuración para Apache)
```

```
sudo ln -s /usr/share/phpmyadmin /var/www/html/phpmyadmin  
sudo phpenmod mcrypt  
sudo service php7.0-fpm restart
```

Error 404: 404 after logging in phpmyadmin with Nginx

# Introducción

## Comprobación del servidor LAMP / LEMP

Crear el fichero info.php en la carpeta de documentos del servidor web:

```
cd /var/www/html  
sudo nano info.php  
    <?php  
    phpinfo();  
    ?>
```

<http://localhost/phpmyadmin>

localhost:8080/info.php

**PHP Version 7.0.22-0ubuntu0.16.04.1**

System	Linux portatil 4.8.0-53-ger
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/fpm
Loaded Configuration File	/etc/php/7.0/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/fpm/conf.d

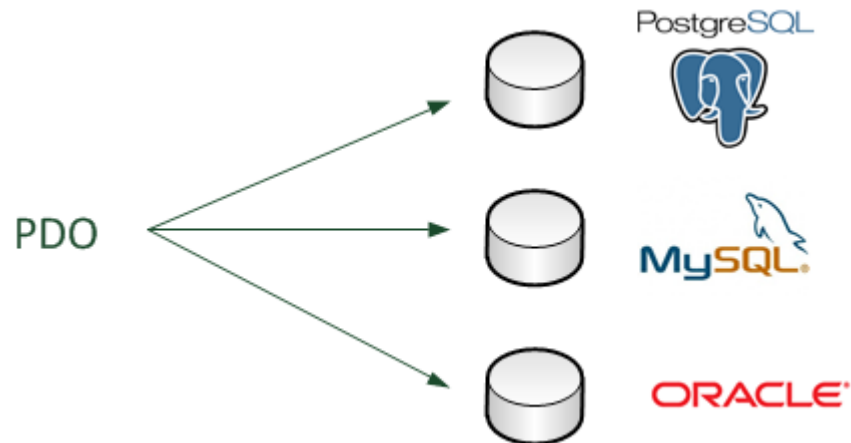
# Introducción

## PHP

### Elegir una API en PHP

// PDO

```
$pdo = new PDO('mysql:host=ejemplo.com;dbname=basedatos', 'usuario', 'contraseña');  
$sentencia = $pdo->query("SELECT 'Hola, querido usuario de MySQL!' AS _message FROM DUAL");  
$fila = $sentencia->fetch(PDO::FETCH_ASSOC);  
echo htmlentities($fila['_message']);
```



# Introducción



# Introducción

```
.  
├─ Controller  
│   ├── borraOferta.php  
│   ├── grabaOferta.php  
│   ├── index.php  
│   └─ nuevaOferta.php  
├─ index.php  
├─ Model  
│   ├── Oferta.php  
│   └─ PizzeriaDB.php  
└─ View  
    ├── formularioOferta.php  
    ├── images  
    │   ├── pizza1.jpg  
    │   ├── pizza2.jpg  
    │   └─ pizza3.jpg  
    └─ listado.php
```



# Introducción

## Creación de la base de datos

consola:

```
usuario@equipo ~ $ sudo mysql -u root -p
```

Enter password:

```
MariaDB> CREATE USER 'alumno'@'localhost' IDENTIFIED BY 'malaga2018';
```

```
MariaDB> GRANT ALL PRIVILEGES ON *.* TO 'alumno'@'localhost';
```

```
MariaDB> CREATE DATABASE IF NOT EXISTS pizzeria;
```

```
MariaDB> CREATE USER 'usuariopizzeria'@'localhost' IDENTIFIED BY 'malaga2018';
```

```
MariaDB> GRANT ALL PRIVILEGES ON pizzeria.* TO 'usuariopizzeria'@'localhost';
```

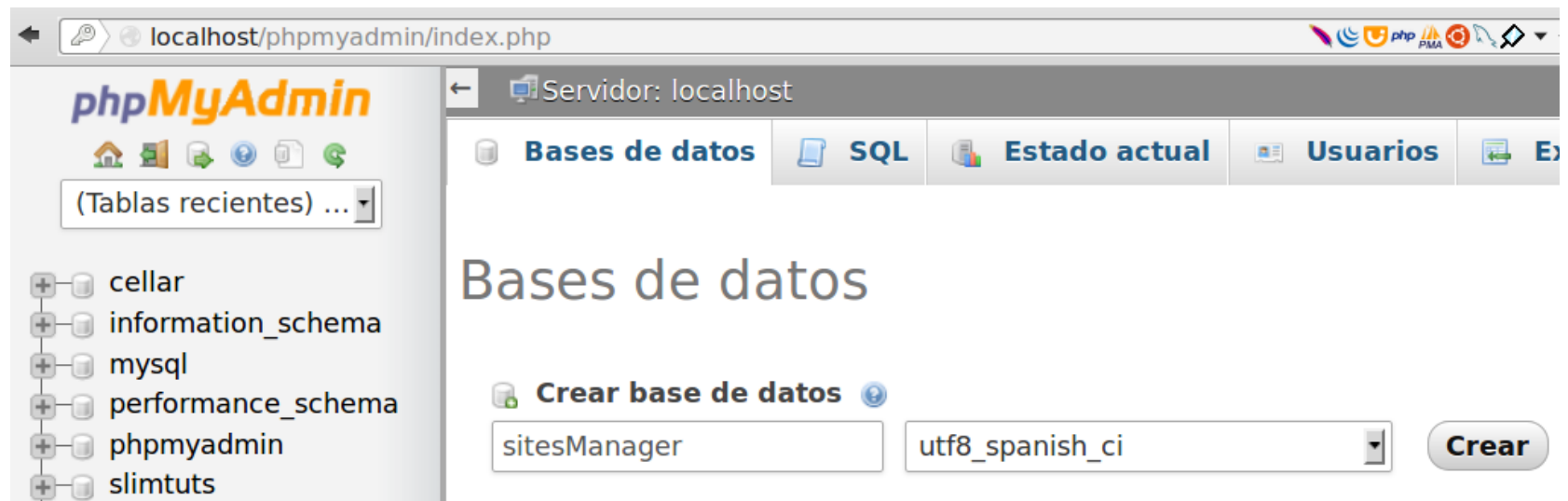
```
MariaDB> USE pizzeria;
```

```
MariaDB> (ejecutar las instrucciones en oferta.sql)
```

# Introducción

## Creación de la base de datos

phpMyAdmin:



# Introducción

Descargar [OfertasPizzeriaBasico](#)

Descomprimir los ficheros en una carpeta del servidor web

Modificar Model/PizzeriaDB.php:

```
<?php
```

```
abstract class PizzeriaDB {  
    private static $server = 'localhost';  
    private static $db = 'pizzeria';  
    private static $user = 'usuariopizzeria';  
    private static $password = 'malaga2018';  
}
```

# Introducción

Comprobar el funcionamiento



← → ↻ 🏠 localhost:8080/mvc/OfertasPizzeriaBasico/Controller/index.php

## Pizzeria Peachepe

[Nueva oferta](#)

---

### Bebida gratis pidiendo dos pizzas



Pidiendo dos pizzas de cualquier tipo te regalamos dos bebidas (no incluye bebidas alcohólicas de alta

# Introducción

¿ ORM ?

# Introducción

Descargar [NotORM](#) y descomprimirlo

Fichero config.php:

```
<?php  
define('HOST', "localhost");  
define('USER', "usuariolibrary");  
define('PASSWORD', "malaga2018");  
define('DATABASE', "library");  
define('TABLE', "books");
```

# Introducción

```
<?php

include "config.php";
include "NotORM.php";

try {
    $pdo = new PDO('mysql:host=' . HOST . ';dbname=' . DATABASE . ';charset=utf8mb4', USER, PASSWORD);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); $library = new NotORM($pdo);

    $libros = $library->books();
    echo "Books: <br />";
    foreach ($libros as $libro) {
        echo $libro["id"] . " " . $libro["title"] . "<br />";
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
```

# ¿Qué es REST?

REST no es . . .

una tecnología

un framework

un patrón de diseño

un protocolo

un estándar



# ¿Qué es REST?

**REST** (**RE**presentational **S**tate **T**ransfer) fue definido en el año 2000 por [Roy Fielding](#) (coautor principal también de la especificación HTTP) en su tesis doctoral y proporciona una forma sencilla de interacción entre sistemas, la mayor parte de las veces a través de un navegador web y HTTP.

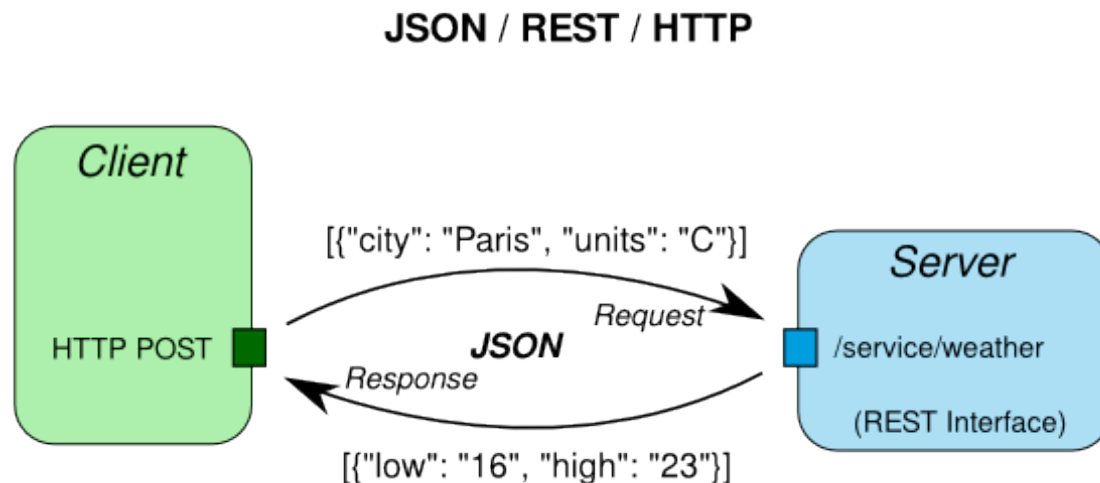
REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es mucho más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

REST es un estilo arquitectónico, un conjunto de convenciones para aplicaciones web y servicios web, que se centra principalmente en la manipulación de recursos a través de especificaciones HTTP.

# ¿Qué es REST?

Podemos decir que REST es una interfaz web estándar y simple que nos permite interactuar con servicios web de una manera muy cómoda.

Por lo tanto, REST es el tipo de arquitectura más natural y estándar para crear un **API** (Application Programming Interface) para servicios orientados a Internet.



# ¿Qué es REST?

Características fundamentales de REST:

- **Un protocolo cliente/servidor sin estado.**

Cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST).

# ¿Qué es REST?

Características fundamentales de REST:

- **Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información.**

HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.

Con frecuencia estas operaciones se equiparan a las operaciones CRUD (**C**reate, **R**ead, **U**ppdate, **D**elete) que se requieren para la persistencia de datos.

# ¿Qué es REST?

Características fundamentales de REST:

- **Una sintaxis universal para identificar los recursos.**

En un sistema REST, cada recurso es accesible únicamente a través de su URI.

# ¿Qué es REST?

Características fundamentales de REST:

- **El uso de hipermedios (HATEOAS).**

La representación de este estado en un sistema REST se hace típicamente usando HTML, JSON o XML.

Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

# ¿Qué es REST?

## Uso correcto de URIs

Cuando desarrollamos una web o una aplicación web, las URLs nos permiten acceder a cada uno de las páginas, secciones o documentos del sitio web. Cada página, información en una sección, archivo, cuando hablamos de REST, los nombramos como recursos. El recurso, por lo tanto, es la información a la que vamos a acceder o modificar o borrar, independientemente de su formato.

Las URL, Uniform Resource Locator, son un tipo de [URI](#), Uniform Resource Identifier, que además de permitir identificar de forma única el recurso, nos permite localizarlo para poder acceder a él o compartir su ubicación.

# ¿Qué es REST?

## Uso correcto de URIs

Una URL se estructura de la siguiente forma:

`{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}`

Ejemplo, predicción meteorológica para Málaga:

<http://www.openweathermap.org/city/2514256>



# ¿Qué es REST?

## Uso correcto de URIs

Una URI es, esencialmente, un identificador de un recurso. Veamos el siguiente ejemplo:

### **GET /amigos**

Podríamos llamar a esto un recurso. Cuando esta ruta es llamada, siguiendo los patrones REST, se obtendrán todos los amigos (generalmente de una base de datos) y se mostrarán en pantalla o se devolverán en un formato determinado (HTML, JSON, XML) a quien lo solicite.

Pero, ¿cómo haremos si queremos especificar un amigo en particular?

### **GET /amigos/eva**

Como se puede ver, es fácilmente comprensible. Esa es una de las claves de la arquitectura RESTful. Permite el uso de URIs que son manejables por los humanos y las máquinas de forma fácil.

Se puede pensar en un recurso como un nombre en plural: contactos, usuarios, fotos, etc.

# ¿Qué es REST?

## Métodos HTTP

Cada petición HTTP especifica un método, o un verbo, en sus encabezados. Generalmente se usan GET y POST. Por defecto, el verbo utilizado cuando accedemos o vemos una página web es GET.

Para cualquier URI dada, podemos referenciar hasta 4 tipos diferentes de métodos: GET, POST, PUT y DELETE.

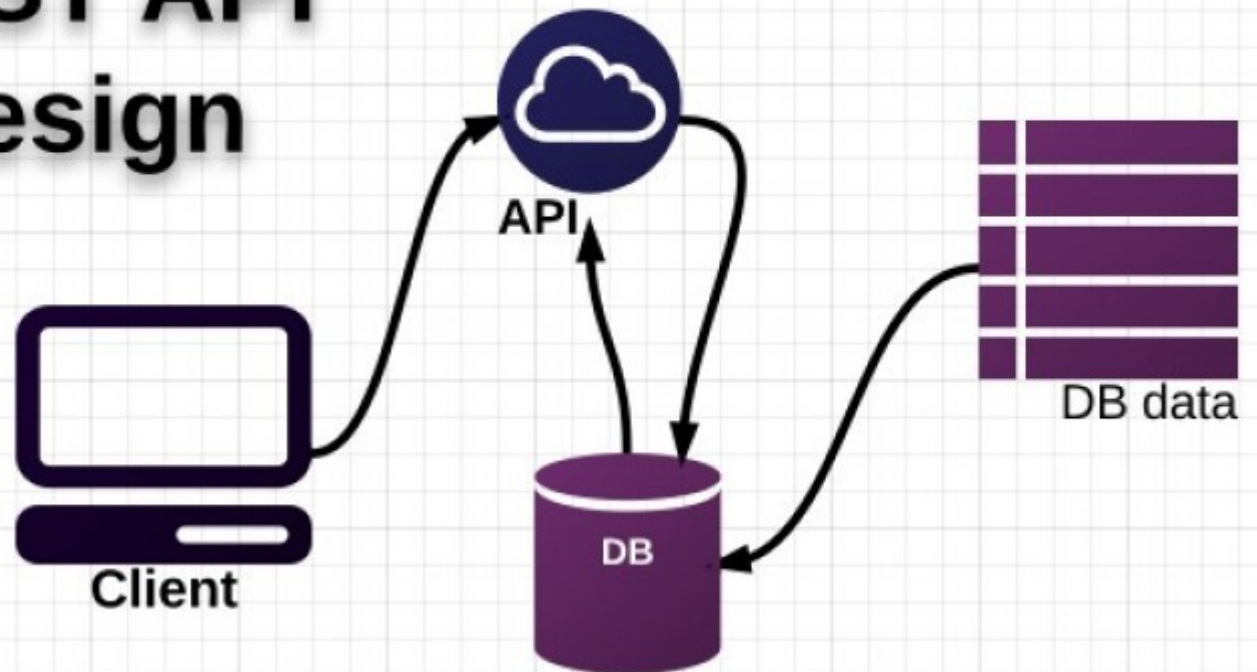
Esencialmente, estos verbos HTTP indican al servidor que hacer con los datos especificados en la URI. Una forma fácil de asociar estos verbos a las acciones realizadas, es comparándolo con CRUD (Create-Read-Update-Delete).

<b>GET</b>	<b>-&gt;</b>	<b>READ</b>
<b>POST</b>	<b>-&gt;</b>	<b>CREATE</b>
<b>PUT</b>	<b>-&gt;</b>	<b>UPDATE</b>
<b>DELETE</b>	<b>-&gt;</b>	<b>DELETE</b>

# ¿Qué es REST?

## REST API Design

**GET** /tasks - display all tasks  
**POST** /tasks - create a new task  
**GET** /tasks/{id} - display a task by ID  
**PUT** /tasks/{id} - update a task by ID  
**DELETE** /tasks/{id} - delete a task by ID



# ¿Qué es REST?

## Verbos disponibles en REST

Ejemplos de URIs que son no RESTful y que no se recomienda utilizar:

/tweets/añadirNuevoTweet.php  
/amigos/borrarAmigoPorNombre.php  
/contactos/actualizarContacto.php

Si trabajamos con REST, esto sería un error de base y puede darnos problemas incluso a la hora de nombrar nuestros recursos, obligándonos a poner verbos en las URLs.

Ejemplos de URIs que son RESTful y que serían un buen ejemplo:

GET /facturas	Nos permite acceder al listado de facturas
GET /facturas/123	Nos permite acceder al detalle de una factura
POST /facturas	Nos permite crear una factura nueva
PUT /facturas/123	Nos permite modificar la factura
DELETE /facturas/123	Nos permite eliminar la factura

# ¿Qué es REST?

## ¿Qué es HATEOAS?

HATEOAS es un acrónimo de Hypermedia As The Engine Of Application State (hipermedia como motor del estado de la aplicación). Significa algo así como que, dado un punto de entrada genérico de nuestra API REST, podemos ser capaces de descubrir sus recursos basándonos únicamente en las respuestas del servidor.

Dicho de otro modo, cuando el servidor nos devuelva la representación de un recurso (JSON, XML...), parte de la información devuelta contendrá identificadores únicos en forma de hipervínculos a otros recursos asociados.

# ¿Qué es REST?

## ¿Qué es HATEOAS?

Ejemplo: Imaginemos que tenemos un API de un concesionario de coches donde nuestros clientes, evidentemente, compran coches. Supongamos que queremos obtener los datos del cliente con id igual a 78. Haríamos una petición de este estilo:

Request URL: `http://miservidor.com/concesionario/api/v1/clientes/78`  
Request Method: GET

Y obtendríamos algo como:

```
{
  "id": 78,
  "nombre": "Juan",
  "apellido": "García",
  "coches": [
    {
      "id": 1033
    },
    {
      "id": 3889
    }
  ]
}
```

# ¿Qué es REST?

## ¿Qué es HATEOAS?

Con esto ya sabemos que nuestro cliente compró dos coches pero, ¿cómo accedemos a la representación de esos dos recursos?

Sin consultar la documentación del API no tenemos forma de obtener la URL que identifique de forma única a cada uno de los coches. Además, aunque supiésemos conformar la URL de acceso a los recursos, cualquier cliente que quisiese consumir los recursos debería tener la responsabilidad de construir dicha URL.

Por último, ¿qué ocurriría si la URL cambiase?, habría que cambiar todos los clientes que consumen los recursos.

# ¿Qué es REST?

## ¿Qué es HATEOAS?

Siguiendo el principio HATEOAS la respuesta sería algo como:

```
{
  "id": 78,
  "nombre": "Juan",
  "apellido": "García",
  "coches": [
    {
      "coche": "http://miservidor.com/concesionario/api/v1/clientes/78/coches/1033"
    },
    {
      "coche": "http://miservidor.com/concesionario/api/v1/clientes/78/coches/3889"
    }
  ]
}
```

De esta forma, ya sabemos dónde debemos ir a buscar los recursos relacionados (coches) con nuestro recurso original (cliente) gracias a la respuesta del servidor (hypertext-driven).

Sin seguir este principio, nuestra API nunca seguirá el verdadero estilo arquitectónico REST (según su principal promotor: Roy Fielding).



# ¿Qué es REST?

## HATEOAS

Hypermedia as the engine of application state

- The model of application is an **engine** that moves from one **state** to another by picking **alternative state transitions** in current set of **representations**
- Or simply put:
  - State: where the user is
    - i.e., current resources
  - Transitions: instructions on user's next steps
    - i.e., links / forms from current resources to others

# ¿Qué es REST?

## Códigos de estado.

Uno de los errores más frecuentes a la hora de construir una API suele ser el reinventar la rueda creando nuestras propias herramientas en lugar de utilizar las que ya han sido creadas, pensadas y probadas. Las ruedas más reinventadas en el desarrollo de APIs son los códigos de error y códigos de estado.

Cuando realizamos una operación, es vital saber si dicha operación se ha realizado con éxito o, en caso contrario, por qué ha fallado. Un error común sería, por ejemplo:

<pre>Petición ===== PUT /facturas/123  Respuesta ===== Status Code 200 Content: {   success: false,   code: 734,   error: "datos insuficientes" }</pre>	<pre>Petición ===== PUT /facturas/123  Respuesta ===== Status Code 400 Content: {   message: "falta un id de cliente para la factura" }</pre>
<b>Erróneo</b>	<b>Correcto</b>

# ¿Qué es REST?

## Códigos de estado.

En este ejemplo se devuelve un código de estado 200, que significa que la petición se ha realizado correctamente. Sin embargo, estamos devolviendo en el cuerpo de la respuesta un error y no el recurso solicitado en la URL.

Este es un error común que tiene varios inconvenientes:

- No es REST ni es estándar.
- El cliente que acceda a este API debe conocer el funcionamiento especial y cómo tratar los errores de la misma, por lo que requiere un esfuerzo adicional importante para usarlo.
- Tenemos que preocuparnos por mantener nuestros propios códigos o mensajes de error.

# ¿Qué es REST?

## Códigos de estado.

HTTP tiene un abanico muy amplio que cubre todas las posibles indicaciones que deberemos añadir en nuestras respuestas cuando las operaciones han ido bien o mal.

Es importante conocerlos y saber cuándo utilizarlos, independientemente de que desarrolles siguiendo REST.

HTTP Status Codes		
	1xx - Informational	>
	2xx - Success	>
	3xx - Redirection	>
	4xx - Client Error	>
	5xx - Server Error	>

# Errores

## Códigos de estado HTTP

### **2xx: Peticiones correctas**

Esta clase de código de estado indica que la petición fue recibida correctamente, entendida y aceptada.

#### **200 OK**

Respuesta estándar para peticiones correctas.

#### **201 Creado**

La petición ha sido completada y ha resultado en la creación de un nuevo recurso.

#### **202 Aceptada**

La petición ha sido aceptada para procesamiento, pero este no ha sido completado. La petición eventualmente pudiere no ser satisfecha, ya que podría ser no permitida o prohibida cuando el procesamiento tenga lugar.

### **4xx Errores del cliente**

#### **404 No encontrado**

Recurso no encontrado. Se utiliza cuando el servidor web no encuentra la página o recurso solicitado.

#### **409 Conflicto**

Indica que la solicitud no pudo ser procesada debido a un conflicto con el estado actual del recurso que ésta identifica.

# Errores

## HTTP Status Codes

# ¿Qué es REST?

## **Tipos y formatos de contenido.**

Cuando hablamos sobre URLs, vimos también que no era correcto indicar el tipo de formato de un recurso al cual queremos acceder o manipular. HTTP nos permite especificar en qué formato queremos recibir el recurso, pudiendo indicar varios en orden de preferencia, para ello utilizamos el header `Accept`.

Nuestra API devolverá el recurso en el primer formato disponible y, de no poder mostrar el recurso en ninguno de los formatos indicados por el cliente mediante el header `Accept`, devolverá el código de estado HTTP 406.

# ¿Qué es REST?

## Tipos y formatos de contenido.

En la respuesta, se devolverá el header Content-Type, para que el cliente sepa qué formato se devuelve, por ejemplo:

Petición

=====

GET /facturas/123

Accept: application/epub+zip , application/pdf, application/json

Respuesta

=====

Status Code 200

Content-Type: application/pdf

En este caso, el cliente solicita la factura en epub comprimido con ZIP y de no tenerlo, en pdf o json por orden de preferencia. El servidor le devuelve finalmente la factura en pdf.



# ¿Qué es REST?

**Para saber más:**

REST CookBook

Arquitectura Rest

Buenas prácticas para el Diseño de una API RESTful Pragmática

- Debería ser sencilla, intuitiva y consistente

- Una API es una interfaz de usuario (UI) para un desarrollador

- Usa acciones y URLs RESTful

- SSL en todos lados y todo el tiempo

- Documentación . . .

# Ejemplos de RESTful Web Services

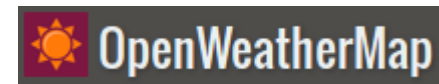
Twitter: [REST APIs](#)



Flickr: [App Garden](#)



OpenWeatherMap: [API](#)



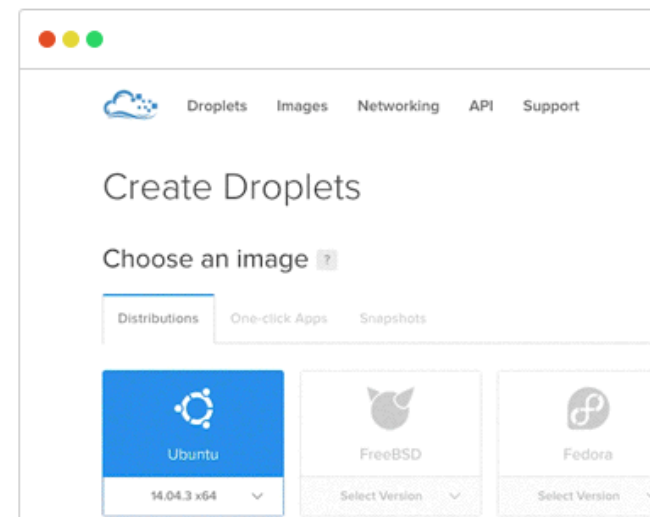
# Alojamiento web

DigitalOcean

Simple Cloud Hosting,  
Built for Developers.

---

Deploy an SSD cloud server  
in 55 seconds.



# Alojamiento web

## How To Create Your First DigitalOcean Droplet

### Create Droplets

Choose an image 

Distributions

Container distributions

One-click apps



Discourse 2.0.20170531 on 16.04



Django 1.8.7 on 16.04



Dokku 0.11.3 on 16.04



Ghost 1.21.1 on 16.04



LAMP on 16.04



LEMP on 16.04

# Alojamiento web

Initial server setup with Ubuntu 16.04

# Alojamiento web

```
$ ssh root@your_server_ip  
(poner nueva password)
```

```
# adduser nuevoUsuario  
# usermod -aG sudo nuevoUsuario  
# exit
```

```
$ ssh nuevoUsuario@your_ip_server  
$ sudo nano /etc/ssh/sshd_config  
(PermitRootLogin no)  
$ sudo service ssh reload
```

# dns









# dns

How to set up a host name

How to Point to DigitalOcean Nameservers

Nameservers:

#	Nameserver	IP Address	Status
1	NS1.DIGITALOCEAN.COM		 
2	NS2.DIGITALOCEAN.COM		 
3	NS3.DIGITALOCEAN.COM		 

Add Nameserver

Save

Cancel

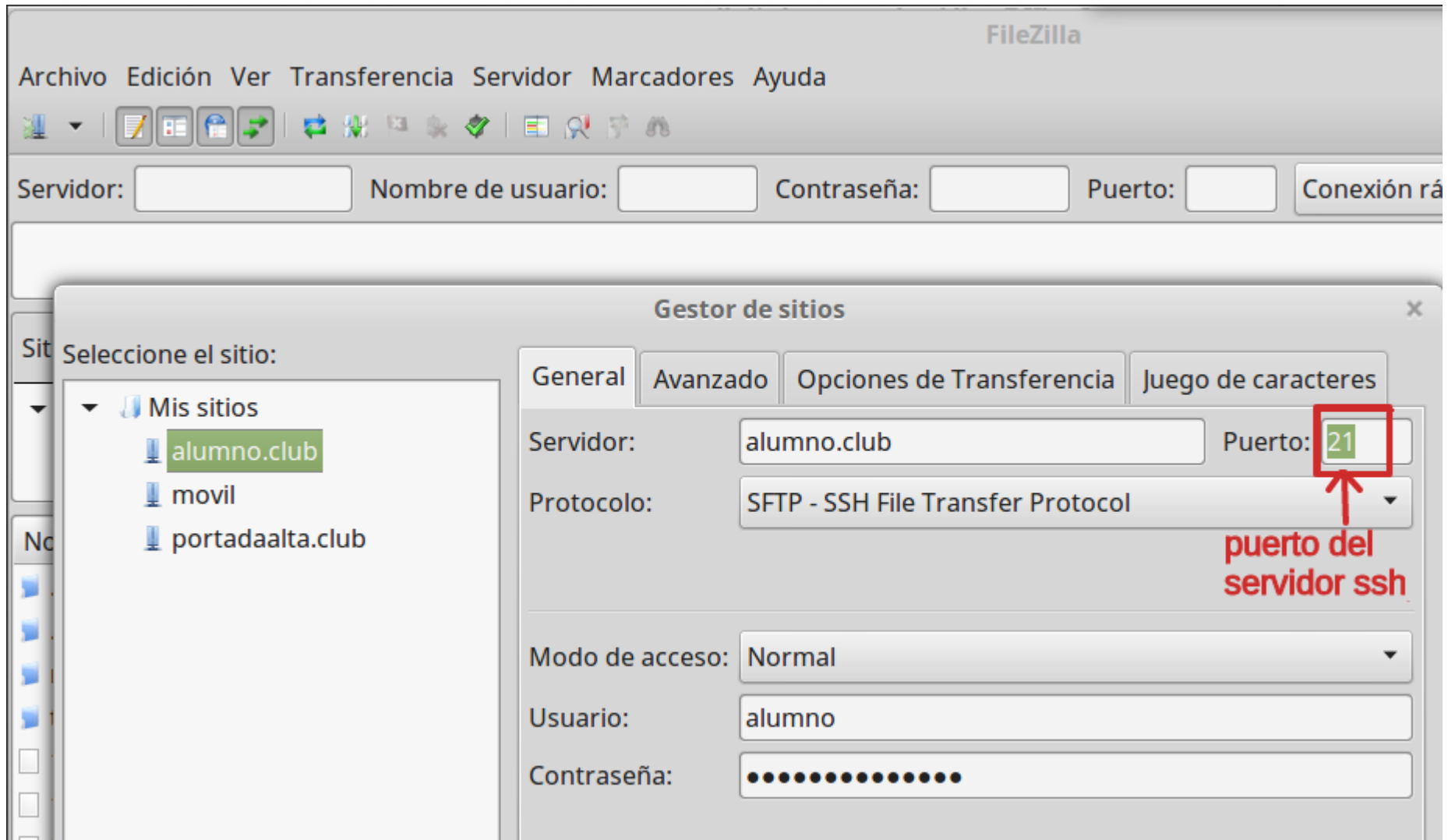


# dns

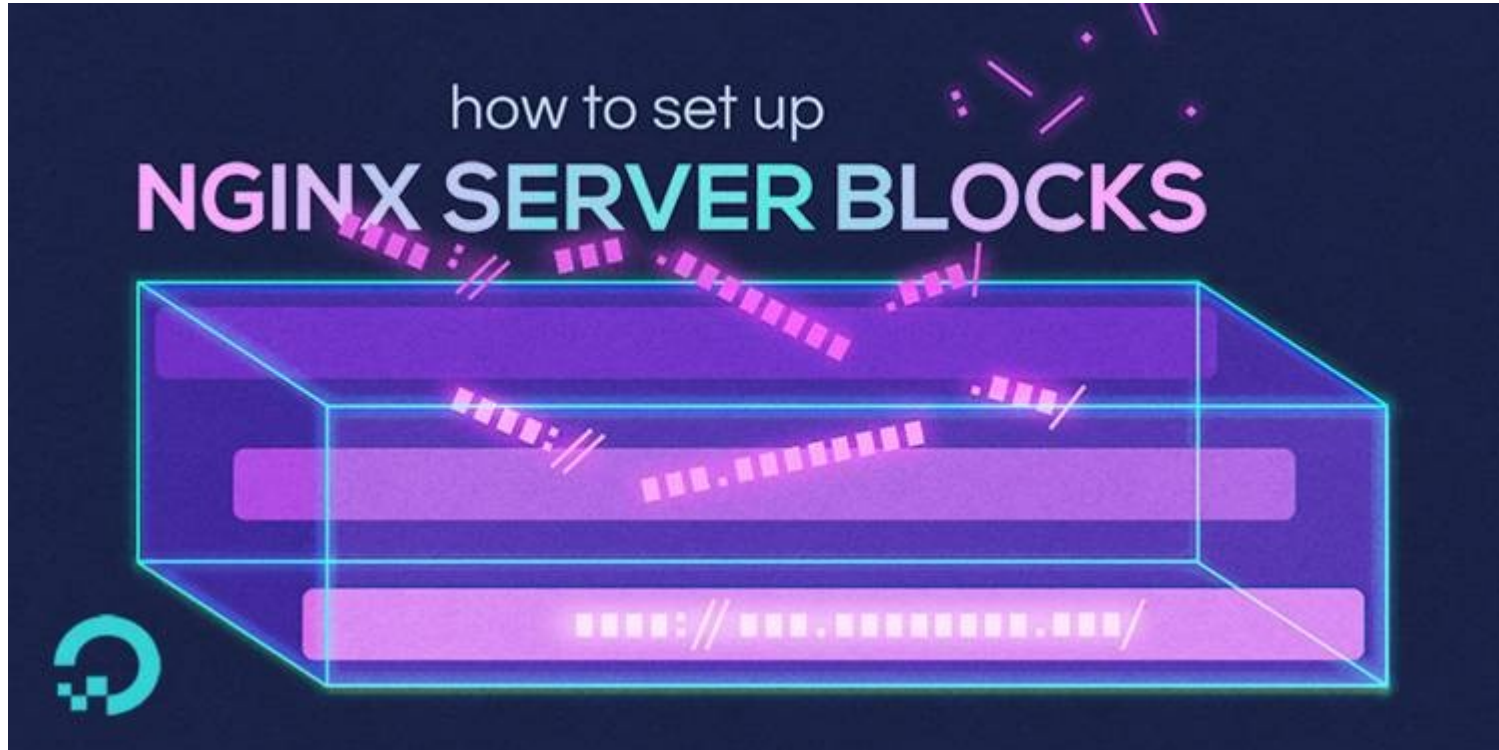


# Subir archivos

sudo apt install filezilla



# Virtual Hosts en Nginx



[How To Set Up Nginx Server Blocks \(Virtual Hosts\) on Ubuntu 16.04](#)

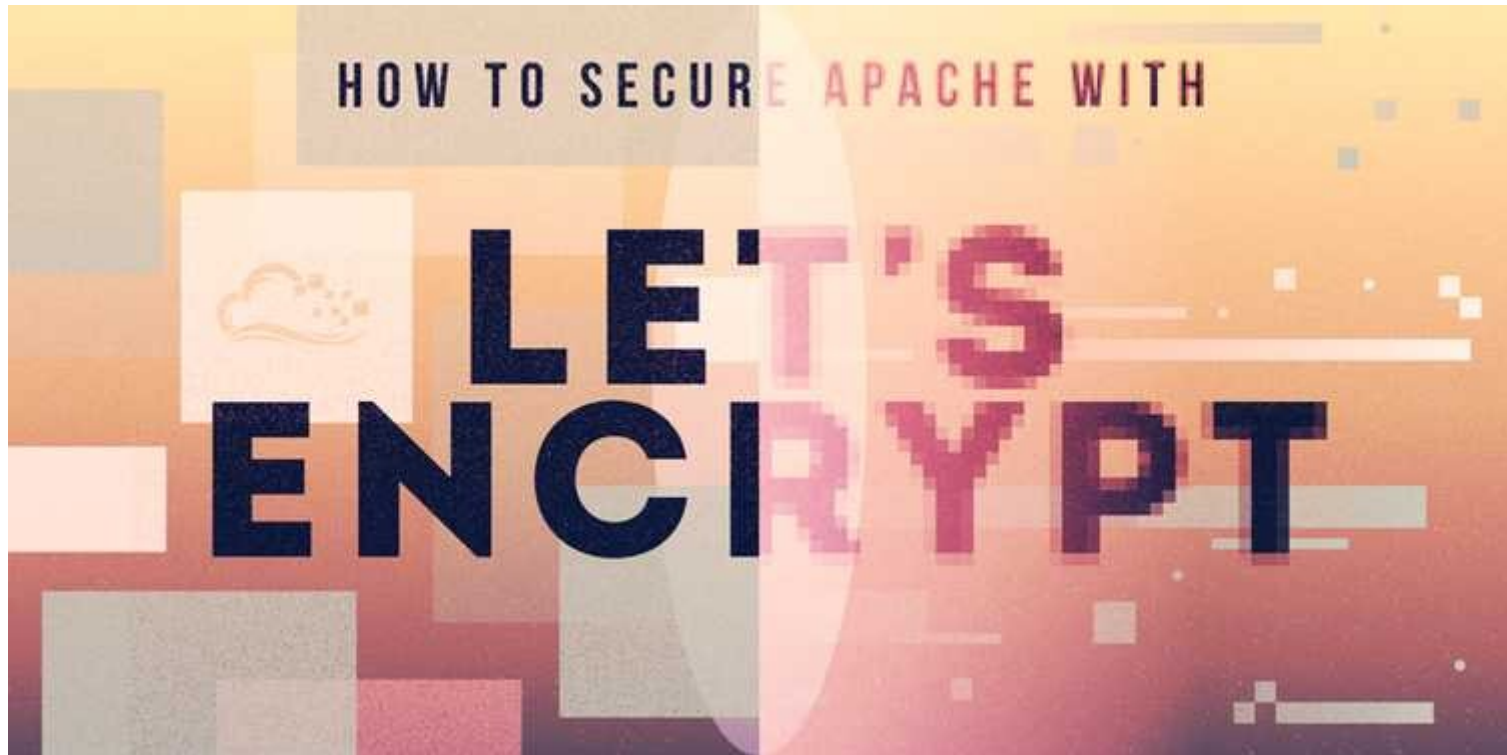
# Certificado SSL con Let's Encrypt

Let's Encrypt is a **free, automated, and open** Certificate Authority.

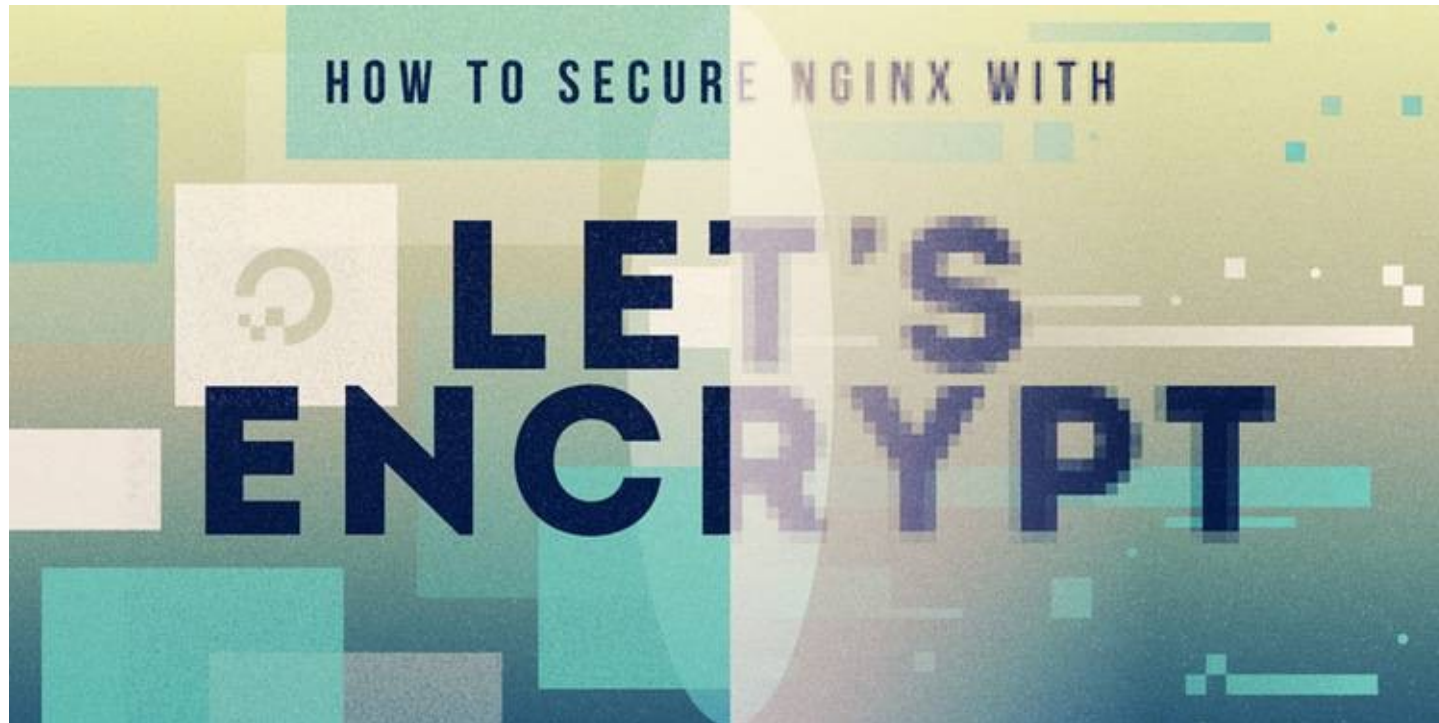
[Get Started](#)

[Donate](#)

# Certificado SSL con Let's Encrypt



# Certificado SSL con Let's Encrypt



# Certificado SSL con Let's Encrypt

How To Secure Apache with Let's Encrypt

Cómo instalar certificados SSL de Let's Encrypt en Nginx

¿Dudas?

¿Sugerencias?



**paco@portadaalta.es**