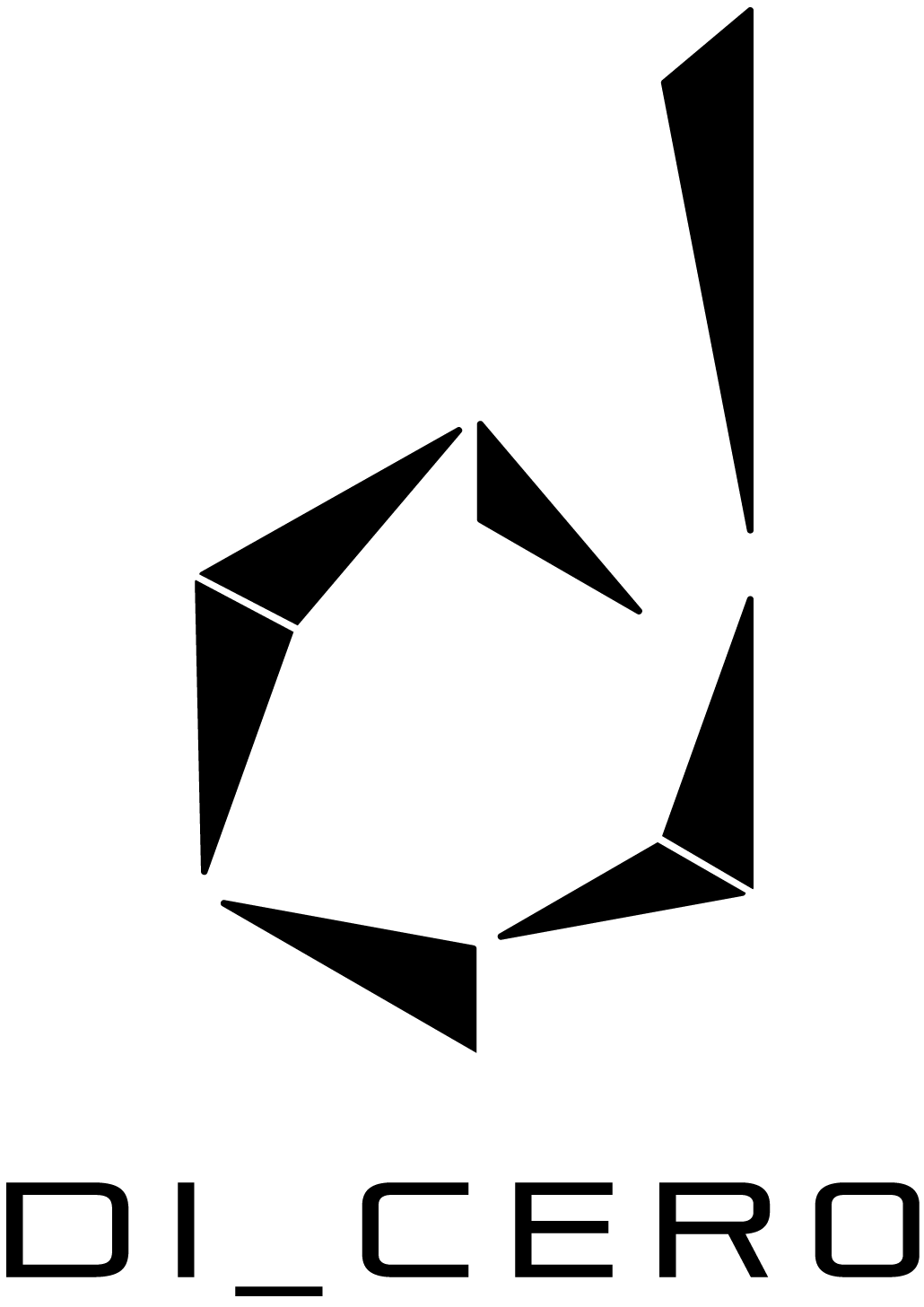
INGENIERÍA MECATRÓNICA



Diego Cervantes Rodríguez

Programación: Desarrollo Backend

Postman, Insomnium, Python, etc.

Introducción a la

Programación Backend

Contenido

[Backend Developer 2](#_Toc203728557)

[Backend, Frontend y Fullstack Developer 2](#_Toc203728558)

[Cómo se Construye el Backend: **Concepto de API** 4](#_Toc203728559)

[HTTP: HyperText Transfer Protocol 4](#_Toc203728560)

[Estructura REST: API con Protocolo HTTP 6](#_Toc203728561)

[API Testing: **Insomnia y Postman** 7](#_Toc203728562)

[Cloud (Nube) 12](#_Toc203728563)

[DevOps 13](#_Toc203728564)

[Arquitectura de Servidores 14](#_Toc203728565)

[Heroku 15](#_Toc203728566)

[Cookies y Sesiones 16](#_Toc203728567)

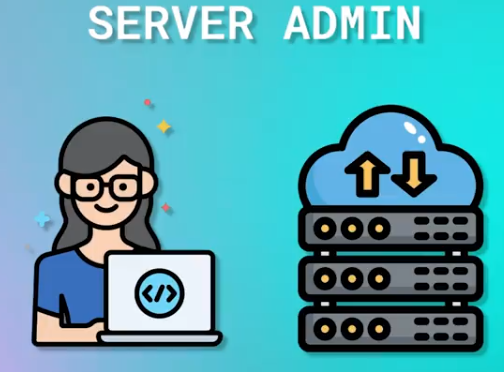
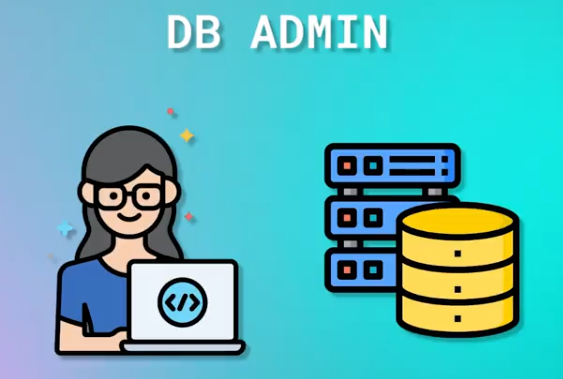
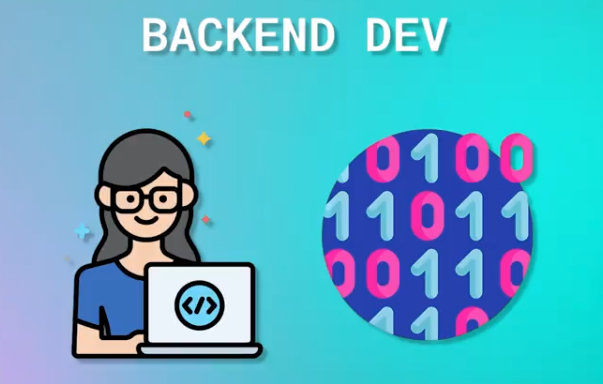
[Bases de Datos 17](#_Toc203728568)

[Referencias 20](#_Toc203728569)

# Backend Developer

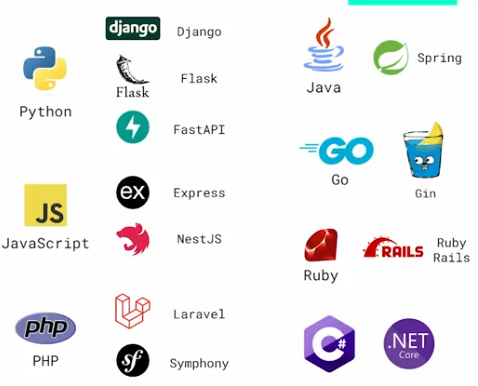
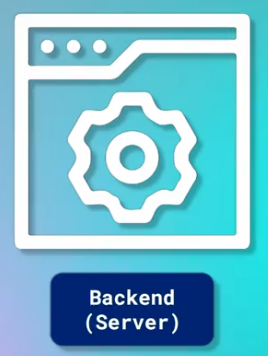
El rol principal del backend developer es el de escribir código que tenga que ver con reglas de negocio, ya sea validación, autorización de usuarios, conexión a bases de datos, manejo de datos, etc. El cual se montará sobre un servidor y será expuesto a millones de usuarios. Sin embargo, como backend también se puede adoptar los roles de:

* **DB Admin:** Este se encarga de gestionar una base de datos, tomando en cuenta sus políticas, disposición, seguridad, etc.
* **Server Admin:** Este se encarga de gestionar la seguridad de los servidores donde corre el código del backend.

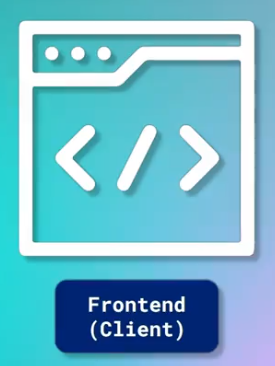


# Backend, Frontend y Fullstack Developer

* **Bakend Developer:** Desarrolla y pone a disposición **servicios** en los cuales los **clientes** **(plataformas frontend como páginas web, aplicaciones móviles, microcontroladores, etc.)** empiezan a conectarse para **extraer datos**.
  + **Los lenguajes más populares y sus frameworks para escribir servicios de backend en un servidor son**: Python (Django o FastAPI), JavaScript (Express de Node.js), TypeScript (NestJS), PHP (Laravel), Java (Spring), C# (.NET), Go (Gin), Ruby (Ruby on Rails), etc.



* **Frontend Developer:** Se enfoca en el área del **renderizado**, esto se refiere a la conexión con un servicio desplegado por el backend de una aplicación para poder **dibujar datos u otras cosas en pantalla**. Los diferentes clientes frontend pueden ser:
  + **Navegadores:** Es el más popular, ya que se encarga de conectarse a un servicio de backend que está montado en un servidor para mostrar los datos HTML que se despliegan (renderizan) en un sitio web. Los navegadores soportan 3 tipos de datos que puede devolver el servicio de un backend, los cuales son:
    - **HTML o Markdown:** Como backend developer se podrá crear un servicio que renderice un código HTML para que muestre diferentes datos en el cliente frontend o para hacer blogs se pueden renderizar de igual forma archivos markdown que sirven para creación de textos con formato.
    - **CSS:** Es código de estilo para la estructura HTML de un sitio web, este cuenta con librerías como Bootstrap, Tailwind, Foundation, etc. Para realizar de forma más sencilla el diseño de una página con código CSS.
    - **JavaScript:** Este se utiliza para proporcionar con efectos o dinamismo a un sitio web, los cuales son utilizados más que nada con frameworks como Angular, React.js, Vue.js, etc. Para solicitar información al backend y renderizarla.
  + **Smartphones:** Los cuales extraen datos para mostrar las aplicaciones móviles de los celulares. Los teléfonos soportan 3 tipos de datos que puede devolver el servicio de un backend, los cuales son:
    - **iOS Nativo:** Estos trabajan con código Swift u Objective C y solo pueden ser utilizados en sistemas operativos de Apple.
    - **Android Nativo:** Estos pueden recibir código Java o Kotlin y solo pueden ser utilizados en dispositivos con sistema operativo Android.
    - **Cross Platforms:** Estos pueden recibir ya sea código de React Native, Flutter o .NET MAUI, lo interesante de este método es que se puede utilizar en dispositivos con sistema operativo de Apple o Android, pero tiene algunas desventajas como el no poder realizar de forma tan sencilla el uso de sensores como GPS, Cámaras, etc.
  + **Microcontroladores:** Los cuales pueden crear dispositivos IoT que se conecten y manden o extraigan datos de algún servidor o base de datos.

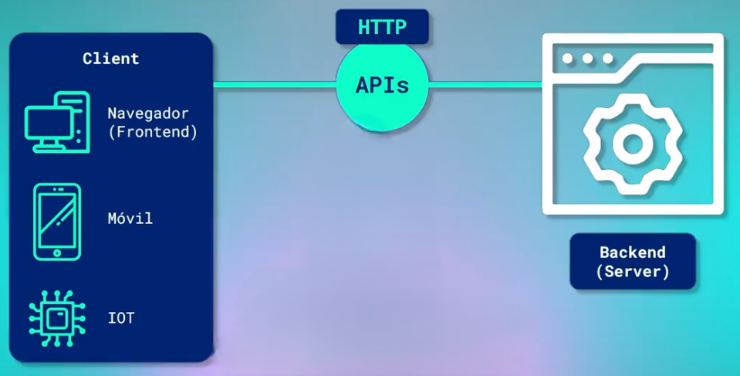
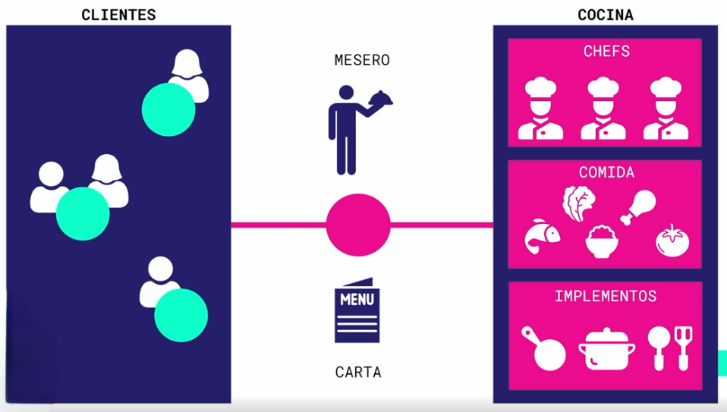


## Cómo se Construye el Backend: **Concepto de API**

Una vez que se tiene el cliente (frontend) y el servicio (backend), lo que se hace para conectarlos es crear una API (Application Programming Interface), la cual se comunica por algo llamado métodos HTTP.

Para explicar el funcionamiento de las APIs, se utiliza una analogía de un mesero, donde se cuenta con uno o varios comensales (cliente frontend) y la cocina (servicio backend) de un restaurante:

* **Comensales:** Estos representan cada navegador, teléfono o microcontrolador utilizado por un usuario, que simboliza un cliente frontend, el cual va a solicitar utilizar el servicio proporcionado por el backend (cocina).
* **Carta o Menú:** Esto representa la pantalla del sitio web, aplicación móvil o interfaz IoT con la cual el usuario (navegador, teléfono o microcontrolador) puede ver el listado de funciones que puede solicitar.
* **Mesero:** Este representa la API, que se encarga de recoger la solicitud del comensal ya que haya visto la carta y pedir que se prepare esto a la cocina (el backend).
* **Cocina:** Esto representa el backend completo del servicio, el cual utiliza las siguientes partes para realizar su función.
  + **Chefs:** Endpoints de desarrollo, estos son un listado de URLs que reciben y devuelven un tipo de dato en específico cada vez que sean utilizados.
  + **Almacén de Comida:** Esto representa las bases de datos, las cuales ya cuentan con datos que hayan sido recabados del frontend, sean llenadas de forma manual o se encuentren en un data warehouse para responder a las solicitudes de los clientes.
  + **Implementos, Cubiertos o Herramientas:** Estos representan las librerías con las que cuenta cada lenguaje de programación para permitirnos conectarnos a bases de datos, realizar autentificaciones de usuarios, etc.



En resumen, las APIs nos permiten comunicarnos con código backend comunicarnos con otra parte de un mismo sistema o con otro sistema con el protocolo HTTP haciendo uso de endpoints para recibir como valor de retorno un código que renderice el frontend o datos en formato JSON o XML.

### HTTP: HyperText Transfer Protocol

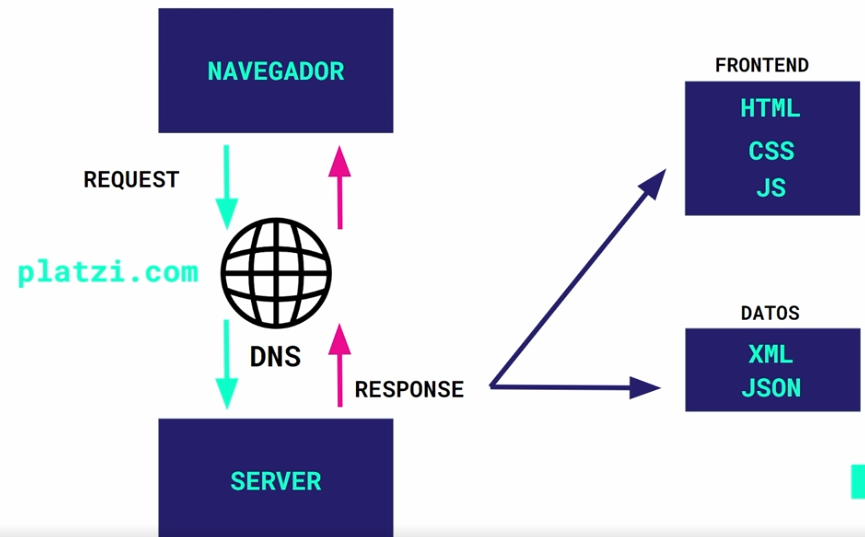
El protocolo HTTP se conforma de una URL que nos ayuda a explorar y exponer servicios web, es la forma en la que el frontend y el backend interactúan entre sí, el cual se conforma de las siguientes partes:

* **Protocolo:** Esta es la parte del URL que indica el protocolo que se está utilizando, ya sea ftp (para transmisión de archivos), http o https (que contiene una capa de seguridad).
* **Dominio:** Esta parte se compra a través de un servicio de host, que nos provee con un servidor DNS, el cual nos permite tener la propiedad de una dirección IP, la cual por motivos de facilidad se ve como una palabra en vez de números, llamado dominio y es como la gente podrá acceder a cada sitio web en específico.
* **Ruta o Endpoint:** Este va separado del dominio por medio de un símbolo de slash (/)
  + **Ruta:** Si nos estamos refiriendo a una ruta nos permitirá acceder a cada carpeta que contenga una imagen, archivo, etc.
  + **Endpoint:** Si nos referimos a un endpoint, este permite acceder a un dato en específico que es expuesto a través del backend en formato JSON o XML para que el frontend pueda ser renderizado o actualizado o a una nueva plantilla de frontend, la cual devolverá en su repuesta un código que pueda ser interpretado por el cliente, ya sea HTML, CSS, JavaScript, Java, Kotlin, Swift, etc.



* **Status HTTP:** Hay algo muy interesante de los protocolos HTTP y son los códigos de estado (status), estos nos dan información para saber el estado en el que se encuentra la petición que realizó el cliente hacia el servicio. El significado de estos se encuentra categorizado por rango, el cual será explicado a continuación:
  + **100-199:** Estos indican información de estado del servidor hacia el cliente, que puede ser como la 102 = Procesando datos.
  + **200-299:** Este es el rango más utilizado porque denota éxito en alguna acción del servidor, por ejemplo 200 = Ok, 201 = Algo fue creado de forma exitosa, 204 = La acción fue exitosa pero no retornó ningún dato, etc.
  + **300-399:** El rango de los 300 indica que algún recurso ha sido movido de lugar, por ejemplo 301 = El recurso que antes estaba aquí, fue movido de lugar, 307 = El recurso fue temporalmente movido, 308 = El recurso fue permanentemente movido, etc.
  + **400-499**: Representan errores de la parte del cliente, esto normalmente ocurre porque el cliente envió una solicitud de forma errónea, por ejemplo 401 = No autorizado, 404 = No encontrado, 409 = Existe conflicto porque los datos están siendo mandados de forma incorrecta.
  + **500-599:** Representan errores que ocurren de parte del servidor, del código que está ejecutándose, por ejemplo 500 = Error interno del servidor, 502 = Bad Gateway, 504 = El tiempo de espera del servidor se ha sobrepasado.

Para explicar todos estos tipos de respuesta del servidor en forma divertida con memes se cuenta con el sitio: [HTTP Cats](https://http.cat/)

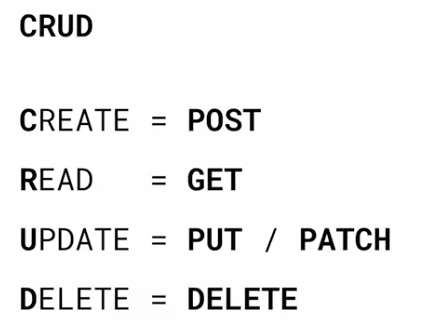


### Estructura REST: API con Protocolo HTTP

El estándar o estructura REST API se utiliza para desarrollar APIs que funcionen a través del protocolo HTTP, es por eso que es la más utilizada. La forma en la que se piden recursos o datos a los endpoints de las REST APIs es a través de su dominio y normalmente devuelven su resultado en formato JSON, que luego serán procesador por un cliente web, smartphone o un microcontrolador.

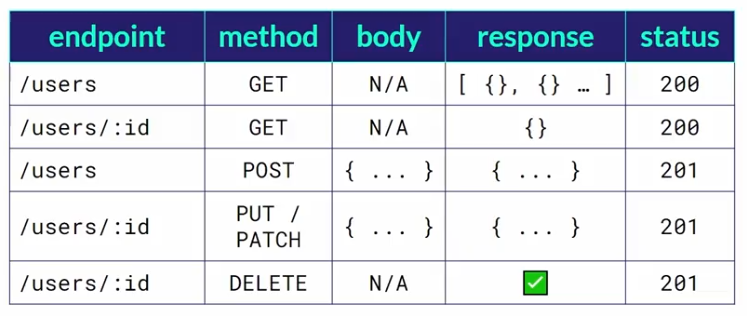
La forma en la que funcionan las REST API es proporcionando como mínimo métodos CRUD que permitan crear, leer, actualizar o borrar datos de un database a través de sus endpoints:

* **Create =** Método **Post** del protocolo HTTP.
* **Read =** Método **Get** del protocolo HTTP.
* **Update =** Métodos **Put/Patch** del protocolo HTTP.
* **Delete =** Método **Delete** del protocolo HTTP.

Cada que se cree un endpoint se debe indicar:

1. El nombre del endpoint.
2. Cuál es el método HTTP que se ejecuta al acceder a dicho endpoint
3. El body que contiene los datos que utilizará el método HTTP para realizar su función.
4. La respuesta que se obtendrá del servidor al utilizar el endpoint.
5. El número de status HTTP que se espera recibir.



## API Testing: **Insomnia y Postman**

Utilizaremos la herramienta de Insomnia y para testear una API de prueba de Platzi que se encuentra en el siguiente dominio: <https://fakeapi.platzi.com/>

Esta API lo que muestra es una tienda en línea de mentira, la cual tiene productos, imágenes de productos, precios, categorías, etc. y utiliza una arquitectura REST, para utilizarla veremos su documentación y utilizaremos algunos de sus endpoints para ver cómo funciona la herramienta:

En las documentaciones de APIs lo que podremos ver será:

* **Método HTTP que se utiliza +** el dominio **+** **el nombre del endpoint**.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

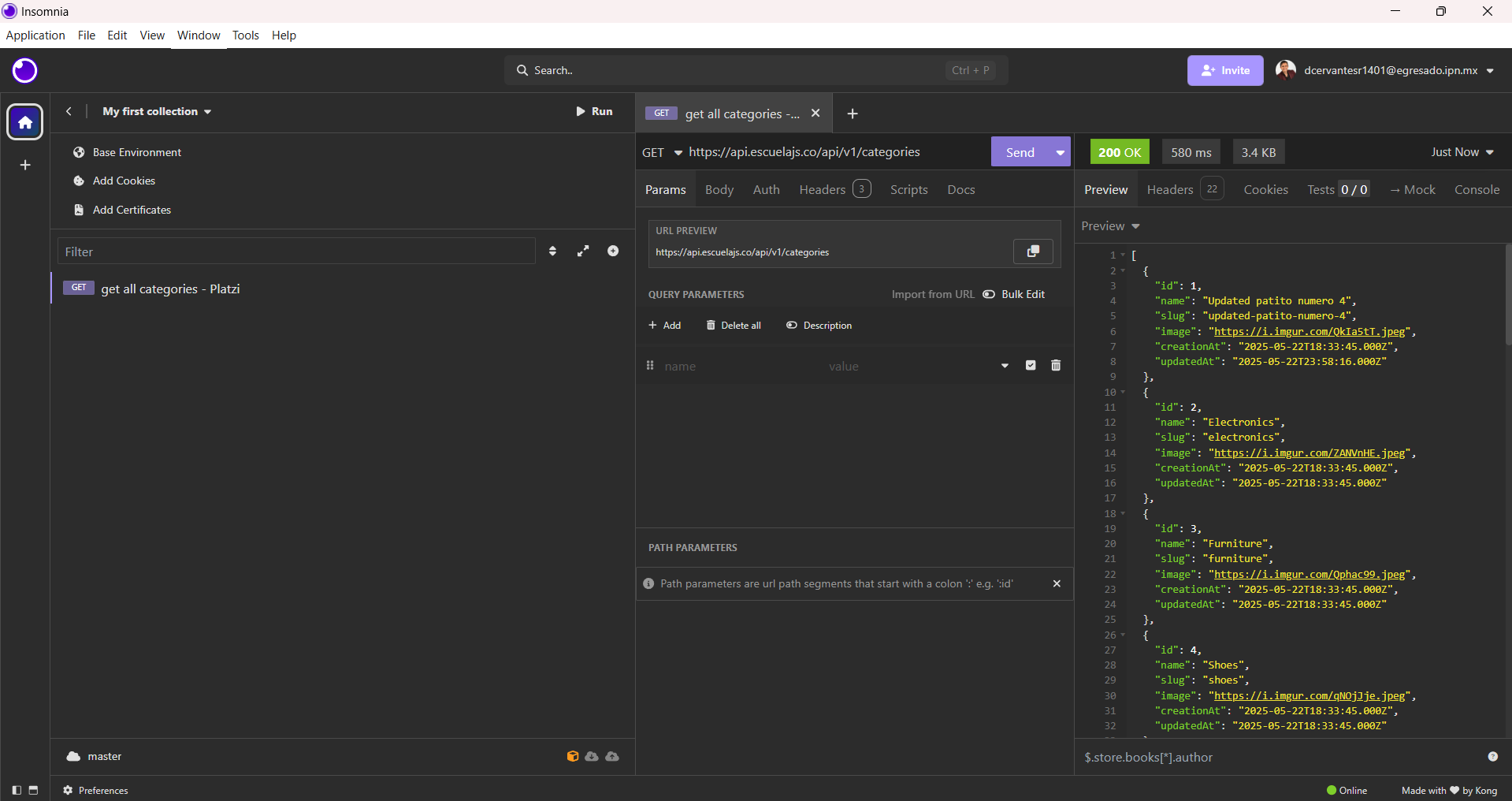
El software de Insomnia puede ser descargado del siguiente enlace: <https://insomnia.rest/download>

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En la interfaz de Insomnia se tiene lo siguiente:

* Del lado izquierdo superior podemos ver el environment, colección o documento en el que nos encontramos, lo cual se refiere a
  + Debajo y del lado izquierdo podemos ver toda la lista de endpoints a los que podemos acceder. Cada vez que utilicemos uno nuevo, podemos asignarle un nombre y guardarlo en nuestra colección de endpoints.
    - Los endpoints guardados indicarán el método HTTP que utilizan y su nombre asignado.
      * Para crear un nuevo request y por lo tanto, guardar un nuevo endpoint, debemos dar clic en el botón de + que se encuentra al lado derecho del filtro de requests.
    - De igual forma estos pueden ser categorizados en folders.
* En la parte media superior podemos introducir la siguiente estructura del endpoint:
  + **Método HTTP que se utiliza +** el dominio **+** **el nombre del endpoint**.
  + Abajo del endpoint se pueden introducir sus siguientes características:
    - **Params:** Se utiliza para ver la endpoint completa y si se quieren añadir elementos como un nombre y un valor a la URL.
    - **Body:** Es el cuerpo del mensaje que se quiere mandar, osea los datos que espera, esto se utiliza solamente cuando se ejecutan comandos HTTP POST, PUT, PATCH o DELETE.
      * En esta parte se puede escoger el formato del body, ya sea en formato JSON, XML, etc.
    - **Auth:** Es el método de autenticación, para validación de usuarios o seguridad.
    - **Headers:** Es el encabezado del body del mensaje que se manda a la API.
    - **Scripts:** Para poder escribir código backend que recabe datos.
    - **Docs:** Es la documentación del endpoint.
* En la parte derecha, podemos ver la respuesta de la solicitud que se realiza hacia cada endpoint de la API.
  + En la parte superior izquierda de la sección derecha podemos ver:
    - El HTTP status code que devolvió la solicitud.
    - El tiempo que se tardó en recuperarla.
    - El peso de la respuesta dada en bits de memoria.
  + Debajo de la franja de estado de la solicitud, se encuentran opciones donde se puede ver un preview de la respuesta, solo su header, las cookies que retornó y su timeline.

 Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto. Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

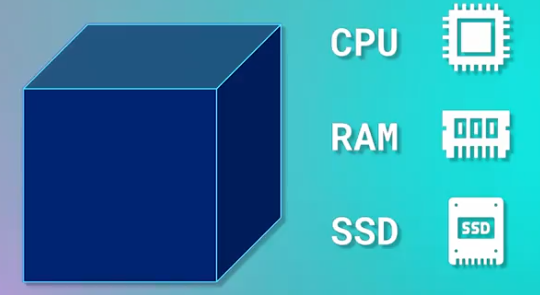
 Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

La diferencia entre los métodos HTTP PUT y PATCH es que PATCH te permite cambiar solo uno de los datos del JSON, mientras que PUT te obliga a mencionarlos todos, aunque el valor sea el mismo en algunos y solo cambien otros.

# Cloud (Nube)

La nube se compone de uno o varios ordenadores que sirven para exponer nuestros servicios de código backend (también llamados granjas de servidores o data centers), los cuales cuentan con CPU, memoria RAM, discos duros, etc. y donde se alojan los servicios y scripts de código a donde los clientes (web, móvil o microcontroladores) acceden por medio de requests con métodos HTTP. Los proveedores de nube más conocidos son AWS (Amazon Web Service), Digital Ocean, Azure (Microsoft), GCP (Google Cloud Platform).



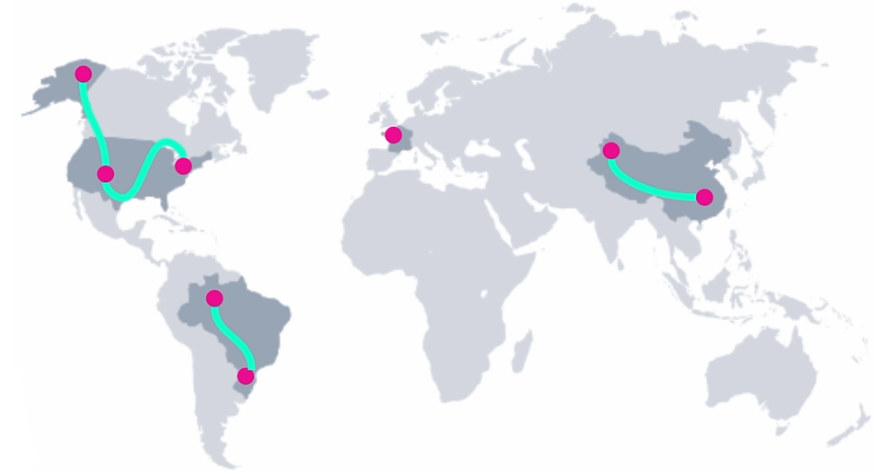
Los **data centers** pueden no estar solo localizados en una región geográfica, sino en varias, con el propósito de dar respuestas más rápidas a los diferentes requests que nuestros servicios puedan recibir. Logrando una menor **latencia o delay** en la recepción de los datos que estamos pidiendo. La **latencia** se refiere al tiempo que tardan los datos en pasar de un punto a otro dentro de una red. Esto implica que dentro de una **nube** puedo tener mi sistema o servicio replicado en diferentes locaciones estratégicas, dependiendo de dónde se encuentren físicamente mis clientes.



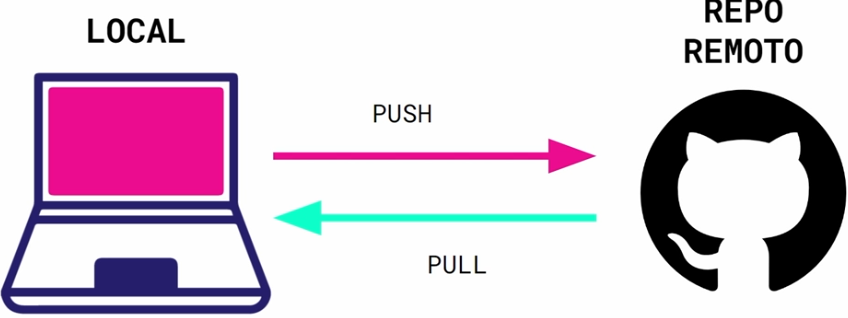
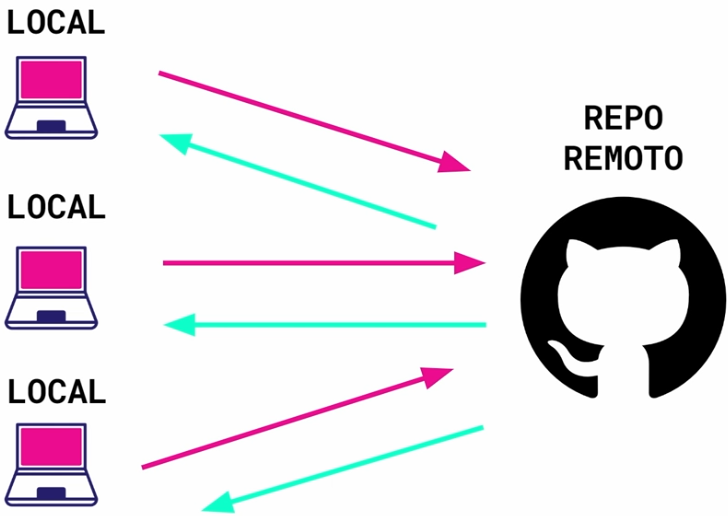
## DevOps

**DevOps** es la parte donde el **desarrollo de código** y la **gestión de operaciones** se unifica, describiendo la forma en la que todos los miembros de un equipo seguirán un flujo de trabajo para lograr que cierto código que se está trabajando de forma local (en nuestro ordenador) llegue a la nube (granja de servidores o data center) y podamos exponerlo (hacer deploy) a miles de usuarios para que lo utilicen.

Para ello, se puede configurar la nube (data center) para que se replique este código de forma estratégica en ciertas zonas geográficas para reducir la latencia que perciban nuestros usuarios.

Para ello se utilizan herramientas de gestión de versiones como GitHub, pudiendo así trabajar varios desarrolladores de código en un mismo código dentro de un mismo repositorio.

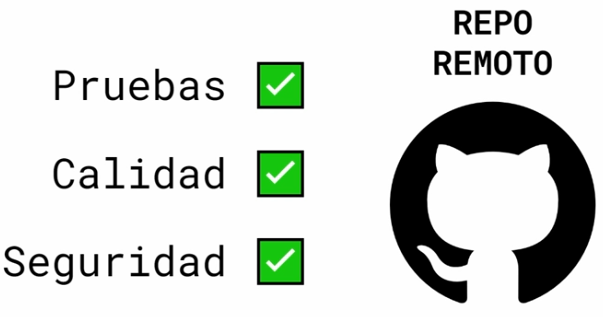
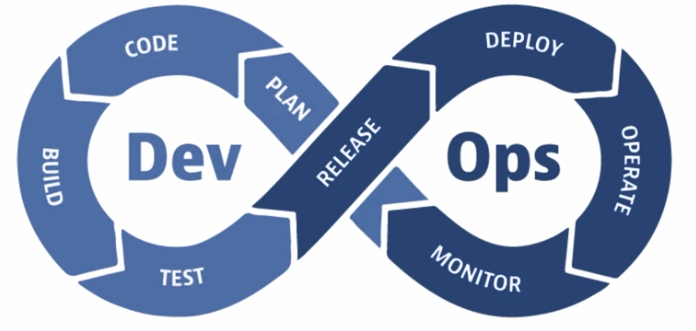
 

Para que se tenga un orden en la forma que van contribuyendo los miembros de un equipo en un repositorio existe el concepto de rama, donde se tiene la rama principal main, las ramas de desarrollo fix, feature y develop para arreglar bugs o agregar funcionalidades al código.

La rama principal que contiene todo el código que saldrá a producción (es decir, el que se subirá a la **nube** para ser expuesto a los usuarios) debe confirmar que el código siga cierto estándar, asegurando la calidad y seguridad del software a través de pruebas unitarias, esto normalmente se realiza por un rol del equipo de desarrollo llamado **Automation** y es muy importante, ya que si el código que ha sido lanzado a producción tiene un error o falla de seguridad, este error será expuesto a millones de usuarios, causando pérdida de datos o de usuarios y por eso es que el rol de **DevOps** es de suma importancia, ya que:

**PLAN:** Planifica el desarrollo del código **→ CODE:** Desarrolla el código de forma local **→ BUILD:** Lo construye **→ TEST:** Lo prueba o testea **→ RELEASE:** Realiza su lanzamiento**→ DEPLOY:** Lo sube a la nube, exponiéndolo a todos los usuarios **→ OPERATE:** Opera el código en producción, gestionando temas de latencia dependiendo de la ubicación geográfica de los usuarios **→ MONITOR:** Monitorea su funcionamiento a través del rol de QA (Quality Assurance).

Este flujo de trabajo no es estático, sino que se repite creando iteraciones de estos pasos y una de las herramientas más importantes para realizarlo es un servidor de repositorios como GitHub.

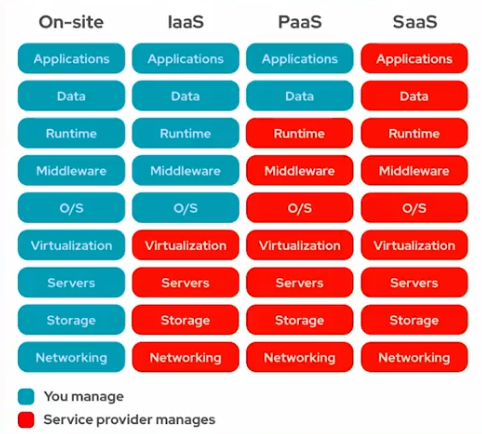
## Arquitectura de Servidores

Recordemos que la nube se compone de varios servidores, llamado granja de servidores, en donde cada uno cuenta con CPU, memoria RAM, disco duro de estado sólido SSD o mecánico, etc. Sin embargo, en estas granjas de servidores se puede adoptar una arquitectura específica, dependiendo del tipo de servicio que se vaya a proporcionar a los usuarios.



* **On Site:** Esto se refiere a cuando temenos un servidor físico propio, donde debemos manejar manualmente todas sus operaciones.
* **IaaS (Infrastructure as a Service)**: Esta arquitectura de servidores es la que mayor control nos proporciona de las 3 arquitecturas, pudiendo manejar de forma manual en los servidores la aplicación, sus datos, el periodo de tiempo en el que mi programa se tarda en ejecutarse, el sistema operativo, etc. **Un ejemplo de este es Digital Ocean.**
* **PaaS (Platform as a Service)**: Aquí se cuenta con un poco más de capacidad de administración de las características del servidor como la aplicación y los datos que se manejan, necesitando en ellas intervención manual. **Un ejemplo de este es Firebase de Google.**
* **SaaS (Software as a Service)**: En esta arquitectura se cuenta con poco control en el estado de red, en los servidores, en el almacenamiento, sistema operativo, datos, ni casi nada porque es un sistema que se maneja prácticamente solo, sin necesidad de intervenciones manuales. **Un ejemplo de este son Google Drive y Slack.**

La arquitectura de servidores más conveniente a utilizarse durante el desarrollo de un software es la **PaaS**, proporcionando un control medio del servidor, ya que no siempre es necesario involucrarse en conceptos de bajo nivel como los recursos de red, storage, etc. En esta arquitectura se gestiona la **aplicación (osea el código)** y la **data (osea las bases de datos)**.



### Heroku

Existen herramientas con arquitecturas que ya están configuradas y manejadas como **PaaS (Plataforma como Servicio)**, una de ellas es Heroku, que es un sistema donde podemos desplegar (deploy) nuestro código y toda la parte de configuración de servidores se proporciona como parte de su servicio, dejando que nosotros solo nos preocupemos del **código (aplicación)** y la **base de datos (data)**.

# Cookies y Sesiones

Las cookies y las sesiones son tecnologías que almacenan información de los usuarios en los navegadores para reconocerlos de forma rápida, brindando así una mejor experiencia en nuestras aplicaciones (código subido a la nube). Las cookies pueden almacenar datos como país de origen, el idioma que el usuario seleccionó previamente al desplegar la información de la aplicación, mantener una sesión (login con usuario y contraseña) abierta, etc.

La forma en la que esto se realiza es a través del navegador (cliente) hacia el servidor:

Donde primero **el cliente realiza una solicitud al servidor** **→** Luego **el servidor le manda una cookie con los datos que quiere recabar** sobre el cliente **→** **La cookie se queda almacenada en el navegador** con los datos almacenados del usuario **→** Y finalmente **esa cookie es retornada al servidor con los datos que extrajo del navegador** siempre que el cliente le haga solicitudes a la nube que tiene almacenado nuestro servicio.

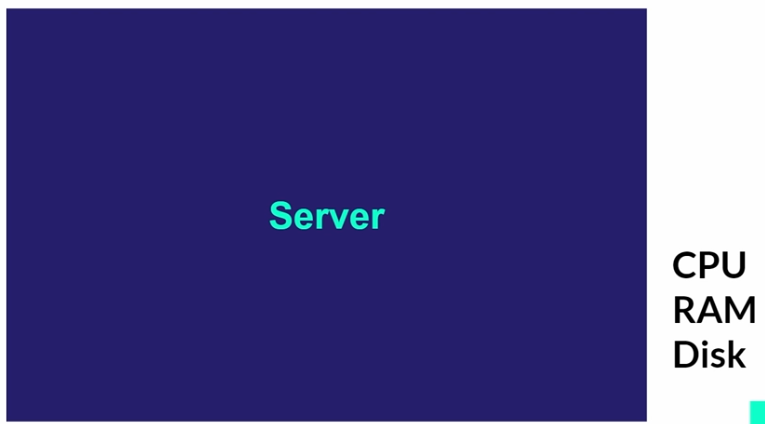
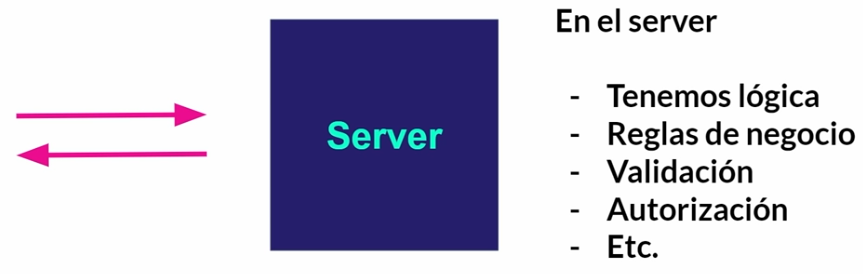


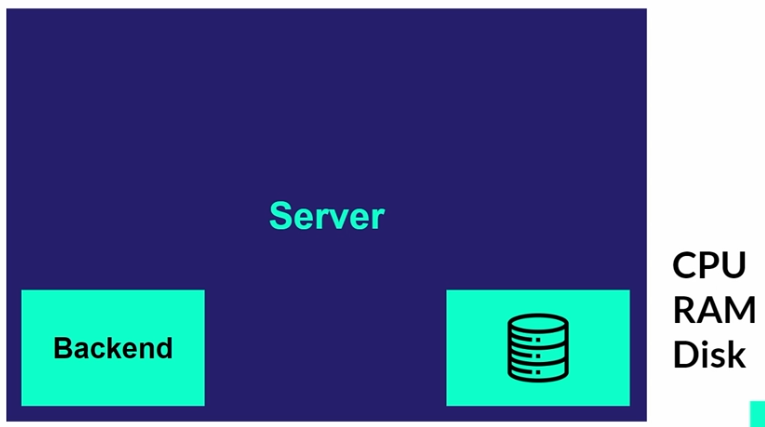
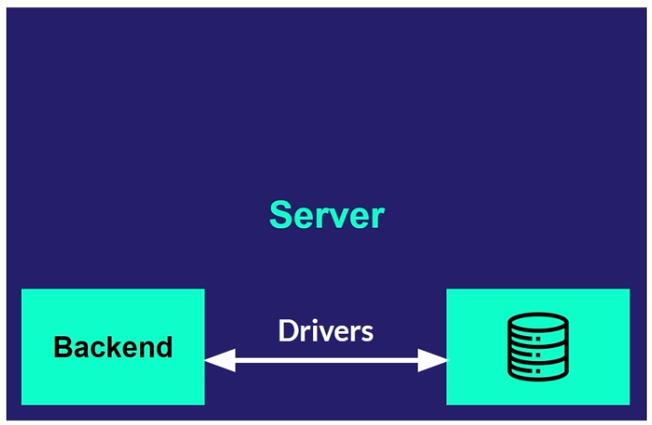
 

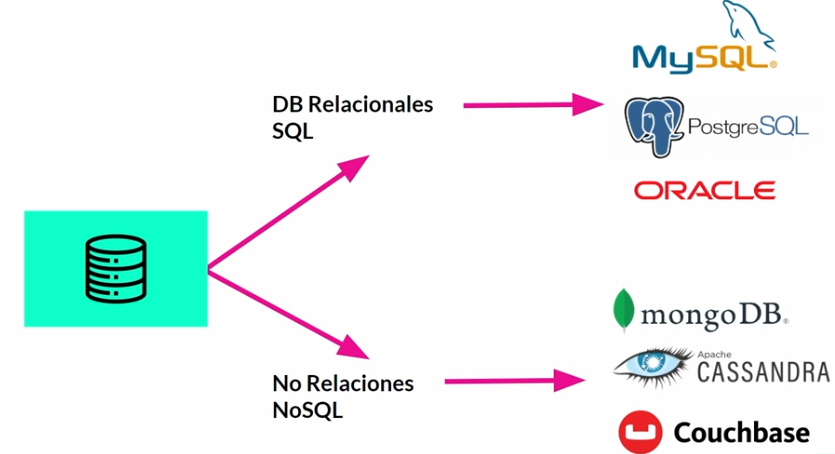
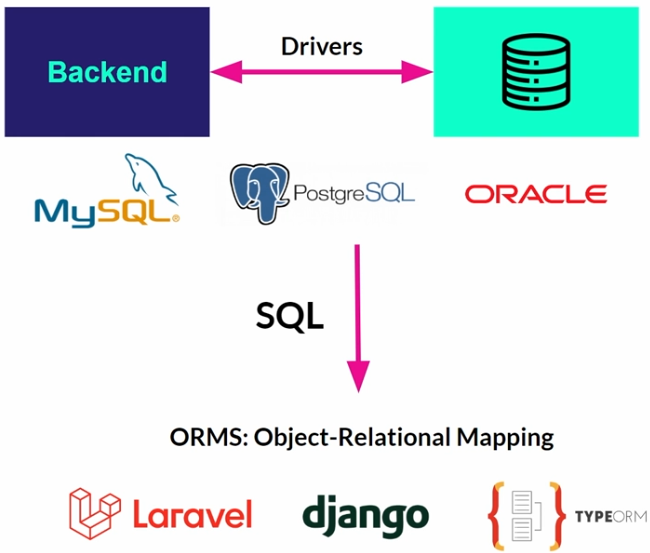
El problema con esto es que solo funciona en servicios web, si esto se quiere realizar en aplicaciones móviles nativas Android o iOS o con microcontroladores, esta tecnología no funciona. El equivalente para este tipo de clientes es algo llamado JWT (JSON Web Token), la cual nos permite validar por medio de tokens dicha información. Aunque cabe mencionar que los JWT si se pueden utilizar también con aplicaciones web.

# Bases de Datos

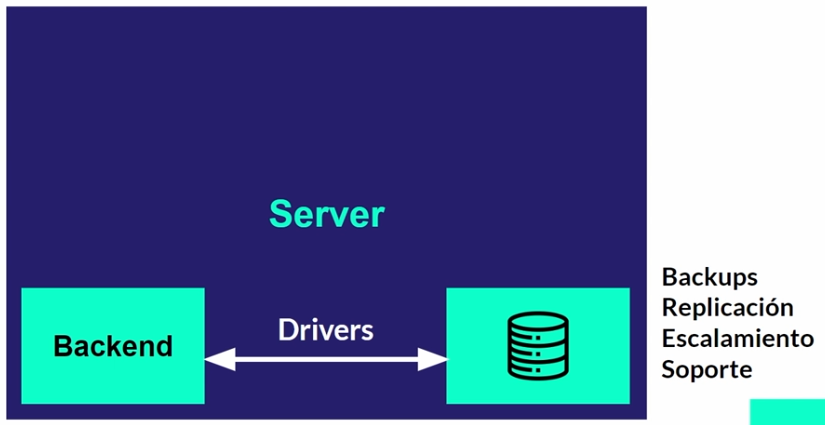
Las bases de datos son plataformas intermedias entre un servidor y un cliente, ya que cuando el cliente interactúa con el servidor, se pueden ingresar datos importantes y relevantes con su operación, pero como este cuenta con recursos limitados como CPU, RAM, Disco Duro, etc. los cuales son utilizados para realizar su operación a través de sus servicios de backend como APIs (no importando si se realiza con uno o varios frameworks o lenguajes de programación), dichos datos son pasados a elementos llamados databases a través de un servicio de base de datos que de igual forma se encuentra dentro del servidor, almacenándolos de forma intermedia, esta operación entre el backend y la base de datos se realiza a través de algo llamado Drivers, donde cada base de datos específica, ya sea relacional o no relacional y dependiendo de igual forma de su tipo, tiene su propio driver, el cual puede estar adaptado para conectarse a distintos lenguaje de programación en el backend. Un ejemplo de esto es como la base de datos de PostrgreSQL tiene distintos drivers para conectarse a código Python, JavaScript, PHP, C# .NET, etc. hasta pudiendo utilizar métodos orientados a objetos para extraer datos a través de un lenguaje orientado a objetos llamado ORM (Object Relational Mapping).



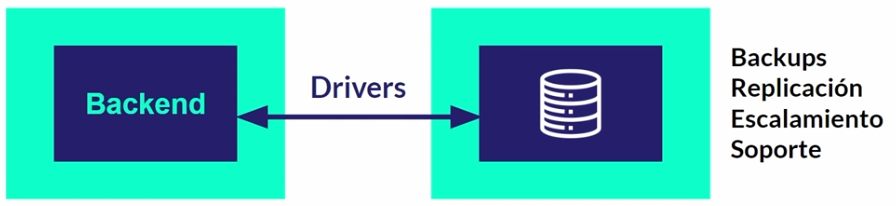
 

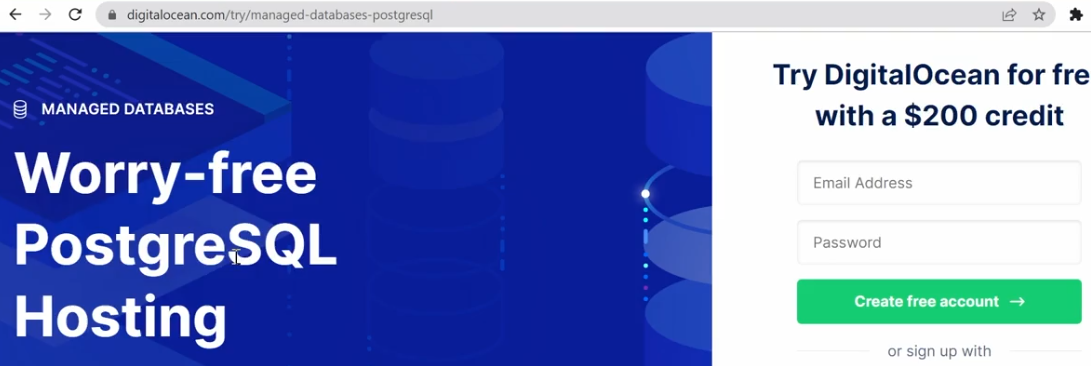
 

Cuando se haya recabado un gran número de datos, estos pueden ser replicados hacia un data warehouse, para luego analizarlos y realizar operaciones como machine learning, business intelligence, etc. Esto si entra dentro del rol de backend, pero los que realizan tareas de backups y soporte son correspondientes al rol de DB Admin.



Aunque existen servicios que pueden realizar todo este tipo de tareas de forma automática, delegando esta parte de administración de la base de datos. Como por ejemplo Heroku, Digital Ocean, MongoDB Altas, Coucbase, etc. que de igual forma proporciona este tipo de servicios.





# Referencias

Platzi, Nicolás Molina, “Curso de Introducción al Desarrollo Backend”, 2018 [Online], Available: https://platzi.com/home/clases/4656-backend/56005-los-roles-del-desarrollo-backend/