***How to start with distributed systems? Beginner's guide to scaling systems. (Análisis)***

Para este análisis tomemos en cuenta el ejemplo mencionado en el video acerca de tener una pizzería o algún establecimiento de comida corrida, pensemos que inicialmente para nuestro establecimiento sólo contamos con un chef. A continuación, se describirán varios puntos a considerar para escalar nuestro establecimiento usándolo como analogía para referirnos al escalado de los sistemas distribuidos.

**Vertical Scaling**

En el video se menciona que en primera instancia lo que se hace es exigirle más rendimiento al único chef que se tiene, en computación con esto se refiere a que se tienen que optimizar procesos e incrementar el tráfico de datos utilizando los mismos recursos, esto es llamado vertical *scaling o escalado vertical***.**

Con esto se refiere a añadir más capacidad (CPU/RAM/DISCO) al servidor dependiendo de la demanda. Esta técnica es comúnmente usada en pequeñas y medianas compañías.

**Pre-programming y Cron Job**

Una manera de mejorar la eficacia que tiene el chef para realizar pedidos es tener de antemano algunos de los ingredientes y de las preparaciones ya listas (Se prefiere que esto se haga en horas donde no existe una alta demanda), de esta manera cuando exista una demanda alta los pedidos podrán ser satisfechos más rápido.

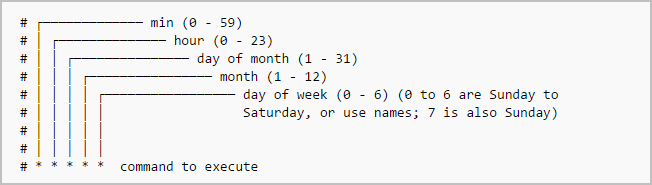
Esto es llamado *pre-programación*, y se refiere al hecho de procesar datos que probablemente más tarde se utilizarán, esto se realiza en las horas que el sistema no tiene muchas peticiones (Normalmente de madrugada). Existe un software llamada Cron, la cual ayuda a programar tareas de administración y mantenimiento, para que estas sean realizadas en horarios específicos durante periodos específicos; este software resulta muy útil ya que permite facilitar tareas que podrían llegar a ser muy pesadas como lo puede ser descargar archivos de internet.

Ilustración Cron Job

**Backups**

¿Qué pasaría si de pronto el único chef con el que se cuenta enferma? Es una buena práctica tener personal que funja como reemplazo en caso de emergencias, ya que con esto se evitan los contratiempos.

Los *respaldos*o *backups*son copias del sistema que, en caso de fallar un nodo, se encargan de tomar las peticiones o funcionamientos del sistema que tuvieron un fallo, estos pueden ser de distintos tipos como el estado operacional, arquitectura o de datos almacenados en base de datos.

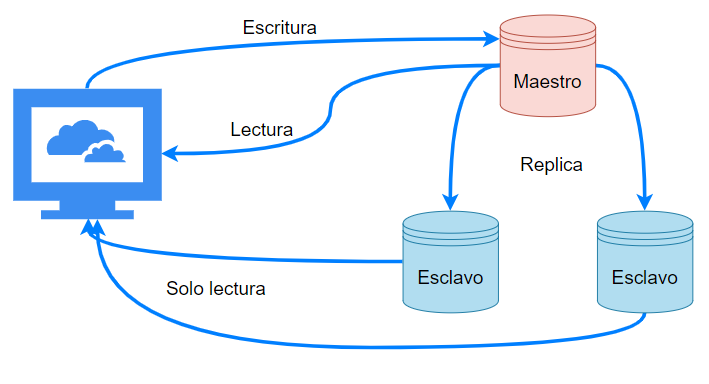
La arquitectura maestro esclavo es una muy popular para realizar respaldos ya que hace una o más replicas de la base de datos y que estas a su vez hacen un log de que han sido actualizadas. Este tipo de arquitectura puede ser síncrona o asíncrona.

Ilustración Arquitectura Maestro - Esclavo

**Horizontal scaling**

Reforzando la idea del punto anterior, ahora en lugar de sólo contar con un chef ¿Por qué no contar con varios? De esta manera las capacidades de atención mejorarían y de esta manera el negocio podría expandir su capacidad de clientes.

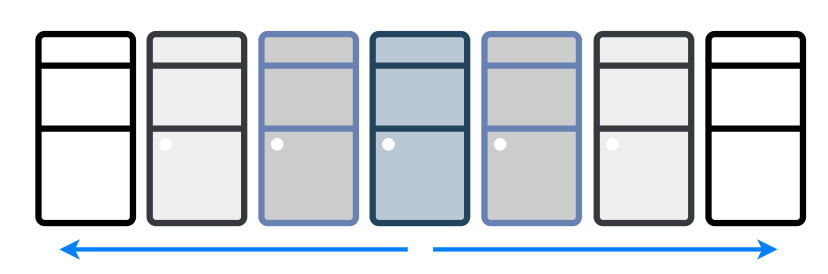
Bien pues, el *escalado horizontal* significa añadir más máquinas a el pool de recursos, a diferencia del escalado vertical, en este no se mejora el rendimiento de las maquinas, sino que se aumenta el número de estas.

Ilustración Horizontal scaling

**Arquitectura de Microservicios**

Tomando en cuenta el escalado horizontal, supongamos ahora que los chefs ahora cada uno tiene una especialización, de 3 chefs, 2 se dedican a cocinar pizza y 1 a cocinar hamburguesas. Lo más favorable sería que las órdenes de hamburguesas sean atendidas por el correspondiente chef, y lo mismo con las de pizza.

Cuando en un sistema se obtienen peticiones, lo más favorable es que el nodo que más comúnmente atiende ese tipo de peticiones sea el que lo atienda, de esta manea cada nodo está especializado en resolver cierto tipo de tareas y con esto se mejora la velocidad con la que se resuelven las peticiones.

Volviendo al ejemplo de los chefs, supongamos que ahora existen grupos de chefs y que cada grupo está compuesto de varios integrantes, todos de tamaños diferentes, entonces la mayoría de las peticiones se cargarán a cada grupo dependiendo de su tipo, y a su vez cada grupo de chefs ya es independiente de los demás, aunque todos los grupos sigan siendo parte del mismo negocio.

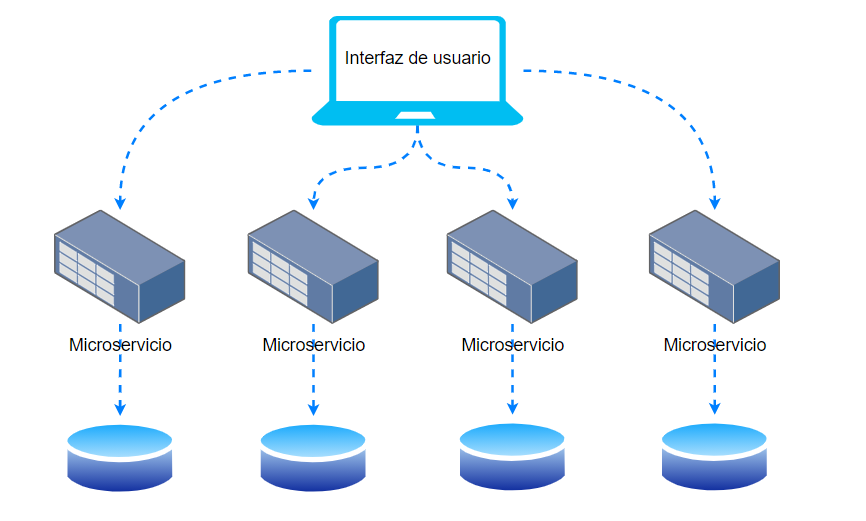
En los sistemas distribuidos lo que se plantea es una *arquitectura de microservicios*, de tal manera que a cada servidor o grupo de servidores se le asignan responsabilidades que son independientes y autónomas de las de todos los demás. La gran ventaja de los microservicios es que, si se requiere modificar solamente un servicio, no es necesario alterar toda la arquitectura.

Ilustración Arquitectura de Microservicios

**Sistemas Distribuidos (Particionar)**

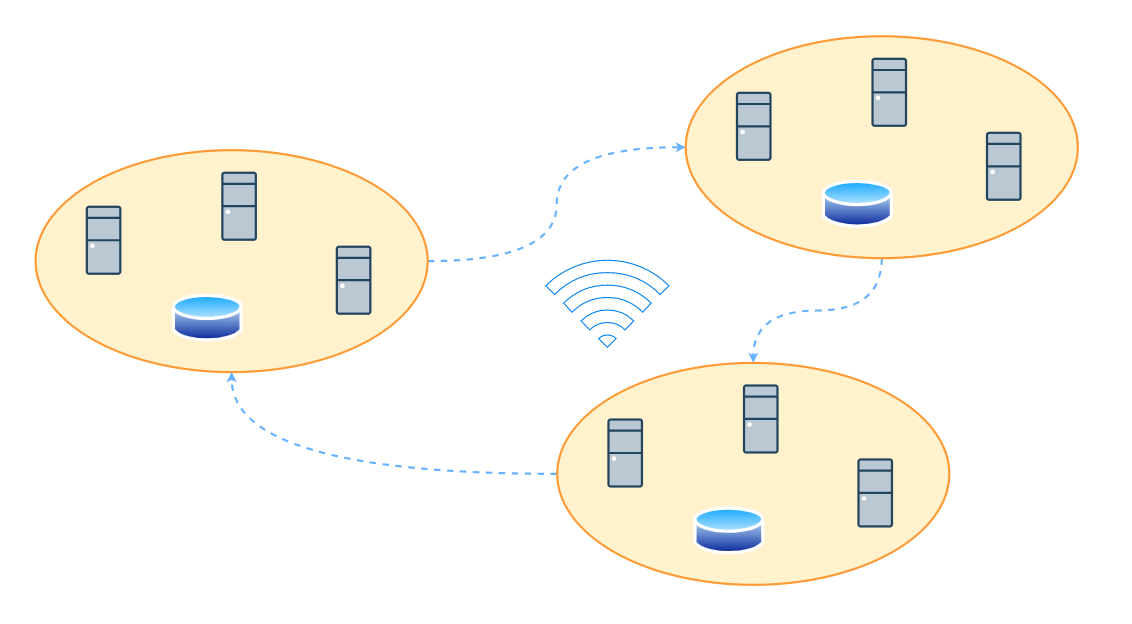
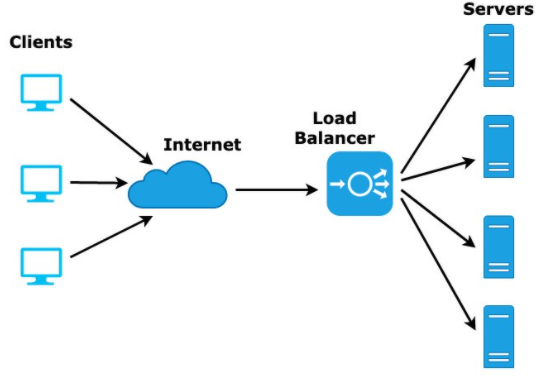
Supongamos que ahora colocamos varios puntos de distribución en distintas locaciones geográficas, lo que se pretende es no tener “todos los huevos en una sola canasta”, procurando que, en caso de que suceda algún desperfecto (apagón de luz, se venzan los permisos para vender, etc.), algún otro punto pueda hacerse cargo de las peticiones pendientes y de estar forma no se pierde dinero.

Ilustración Sistema Distribuido

Los *sistemas distribuidos* funcionan de una forma similar, aunque además de ser tolerante a fallos, tiene más intenciones como las de encontrarse más cerca de todos los usuarios sin importar su posición global, para de estar forma poder atender a las peticiones de la manera más eficiente.

**Load Balancer**

Supongamos que de dos sucursales que se tienen, se quiere saber cuál de ambas ofrece un servicio más eficiente al cliente, supongamos colocar un operador central el cuál se encarga de decidir a cuál de ambas enviará la petición. Bien pues, supongamos que la primera sucursal es muy popular y por tanto muy demandada, y tardaría en entregarle al cliente 1 hora y media entre la cola de espera, la preparación y la entrega; mientras que la sucursal 2 tarda tan sólo 25 min en realizar todo esto. Este lugar central que se encarga de tomar las decisiones se llama un *load balancer***,** y su función es tomar las peticiones de los clientes y decidir cuál de todas las sucursales otorgará una mejor experiencia al usuario, y además significa ahorrar tiempo y ganar más dinero.

En los sistemas distribuidos un *load balancer* o *balanceador de carga* es un dispositivo que se encarga de distribuir la carga de tráfico en la red, son utilizados para incrementar la capacidad y la eficiencia de las aplicaciones. Estos disminuyen la carga de los servidores, así como también ayudan a la realización de tareas específicas de la aplicación.

Ilustración Load Balancer

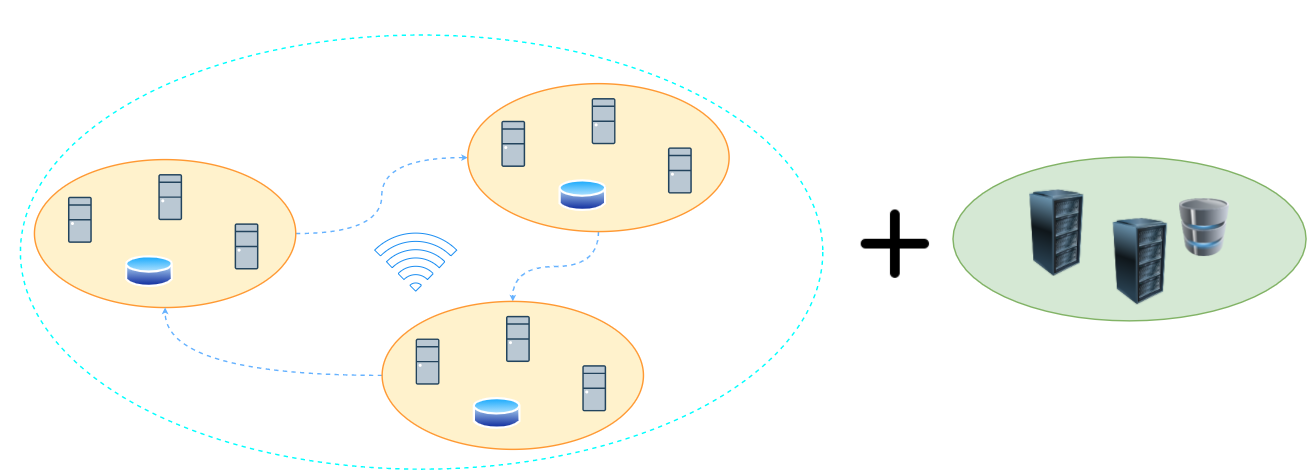
**Decoupling**

Debe haber una separación de las funcionalidades del sistema, para que de esta forma cada parte pueda funcionar de una manera independiente y más eficiente. Estos puntos están conectados sin estar directamente conectados, es un tipo de ambiente operacional en el que los elementos tienen muy poco o nada de conocimiento acerca de otros componentes.

**Logging and Metrics Calculation**

Es importante tener un despliegue de todos los movimientos que se realizan dentro de las máquinas (lun log), posteriormente se pueden utilizar algunas herramientas que puedan ayudar a ver en qué aspectos puede mejorar el sistema, algunas de estas herramientas son los análisis, las auditorías, los reportes y el machine learrning.

**Extensible**

Como desarrollador, nunca se pretende escribir código una y otra vez para un propósito diferente, por ejemplo, algún día puede ser una pizza o al día siguiente una hamburguesa, de esta manera te aseguras de que tu sistema puede crecer sin causar un desastre.

**High Level Design**



Tomando en cuenta todos los puntos anteriores, hay uno que resulta ser de los más importantes y el cuál separa a las grandes aplicaciones con las no tan importantes, y esto es el diseño de alto nivel o *high level design*, todo el diseño del sistema en general. Este describe al sistema y a la arquitectura de la aplicación, todo esto es importante porque es una buena práctica y que además permite tener una mejor organización y en general una idea más concreta de lo que concierne al sistema.

**Conclusiones**

En sistemas distribuidos, es importante contar con diseños de las arquitecturas y funcionalidades de la aplicación; de esta forma, y tomando en cuenta todos los puntos mencionados a lo largo de este análisis, los sistemas pueden ser excelentes de cara al cliente.

Par el cliente, un sistema distribuido debe ser consistente, tolerante a fallos y además debe ofrecer velocidad de respuesta, de esta forma es que las empresas han logrado crecer sus negocios de manera global, nunca dejando atrás que una empresa y su sistema deben estar tomados de la mano cuando se preparan para una expansión.