1. Microservicios principales

Cada uno tendrá su propia lógica, controladores, base de datos (o esquema en la misma DB si lo prefieres) y API independiente.

| Microservicio | Descripción | Tablas que usa |
| --- | --- | --- |
| Auth Service | Maneja autenticación y seguridad de usuarios de tienda y clientes | User |
| Store Service | Administra tiendas licenciadas | Store, License |
| Catalog Service | Administra categorías, subcategorías y productos | MainCategory, Subcategory, Product |
| Order Service | Procesa pedidos y pagos | Order, OrderItem |
| Notification Service | Envía correos, SMS o notificaciones push | No depende directamente de tablas, usa eventos |
| Payment Service | Integra pasarela local o contraentrega | Order |
| Analytics Service | Estadísticas de ventas, productos más vendidos, etc. | Todas |

## **Comunicación entre microservicios**

* **Síncrona (HTTP/REST)** → cuando se necesite una respuesta inmediata.  
  Ej: Order Service consulta a Catalog Service para validar stock.
* **Asíncrona (Mensajería)** → para eventos y notificaciones.  
  Ej: RabbitMQ o Kafka para enviar un evento “pedido\_creado” que escucha Notification Service.

1. Arquitectura general



## **Docker y Kubernetes**

* **Docker** → cada microservicio en su propio contenedor con NestJS.
* **Kubernetes** → orquesta los microservicios, maneja escalabilidad y disponibilidad.
* **Base de datos** → puede estar en un solo clúster MySQL o uno por servicio.
* **Broker** → RabbitMQ también en Kubernetes.

## ****Flujo de un pedido****

1. Cliente inicia sesión (**Auth Service**).
2. Consulta catálogo (**Catalog Service**).
3. Agrega productos al carrito (**Frontend local**, no en la DB).
4. Confirma pedido (**Order Service**).
5. Valida stock (**Catalog Service**).
6. Procesa pago (**Payment Service**).
7. Crea registro de pedido (**Order Service**).
8. Emite evento pedido\_creado (**RabbitMQ**).
9. Notification Service envía mensaje al cliente.
10. Analytics Service guarda el evento para estadísticas.