arquitectura **híbrida (monolito + microservicios)** para MotusCar.

Los microservicios se usan en componentes **críticos que requieren escalabilidad independiente, tecnologías especializadas o aislamiento de seguridad**.

**Ventajas:**

* Desarrollo rápido (APIs con DRF).
* Transacciones ACID (ej: reserva + pago en misma DB).
* Muchas entidades se afectan entre si, es mas fácil editarlas en la misma DB (ej: al añadir un vehiculo a un usuario se podrá hacer en la misma DB, ya que no necesita mucha escalabilidad)

### ****¿Cuándo Evitar Microservicios?****

* Si el equipo es pequeño (< 5 devs).
* Si no hay necesidad real de escalar componentes por separado.

 **Mayor sobrecarga operacional**: Cada microservicio necesita:

* Su propia base de datos.
* Configuración de CI/CD independiente.
* Monitoreo y logs separados.

 **Comunicación compleja**: APIs REST/gRPC entre servicios generan latencia y requieren manejo de errores (retries, circuit breakers).

* **Menos servicios = menos infraestructura que mantener**.

### ****Problema con Microservicios Puros****:

* **Llamadas entre servicios** (incluso en la misma red) añaden latencia.
* Ejemplo: Para mostrar un taller, el frontend debe llamar a:
  1. **Servicio de usuarios** (datos del dueño).
  2. **Servicio de talleres** (info del local).
  3. **Servicio de reseñas** (puntuación).  
     → **3 llamadas HTTP = más lento que una consulta JOIN en SQL**.

**Microservicios**

**1. Geolocalización en Tiempo Real**

**¿Por qué?**

* Consultas geoespaciales complejas (PostGIS).
* Alto tráfico en búsquedas de talleres cercanos.

**Tecnología:**

* **Django + PostGIS** (el mismo stack, pero desacoplado).

### ****2. Sistema de Pagos****

**¿Por qué?**

* Requiere aislamiento por seguridad (PCI DSS).
* Integración con APIs externas (Stripe, Webpay).

**Tecnología:**

* **Node.js + Express** (mejor soporte para webhooks de pagos).

### ****3. Notificaciones y Comunicaciones****

**¿Por qué?**

* Envío masivo de emails/SMS (Celery puede saturar el monolito).

**Tecnología:**

* **Python + FastAPI + Celery** (o AWS SQS).

### ****4. Chatbot con IA (G-IA-N o AdrIAn)****

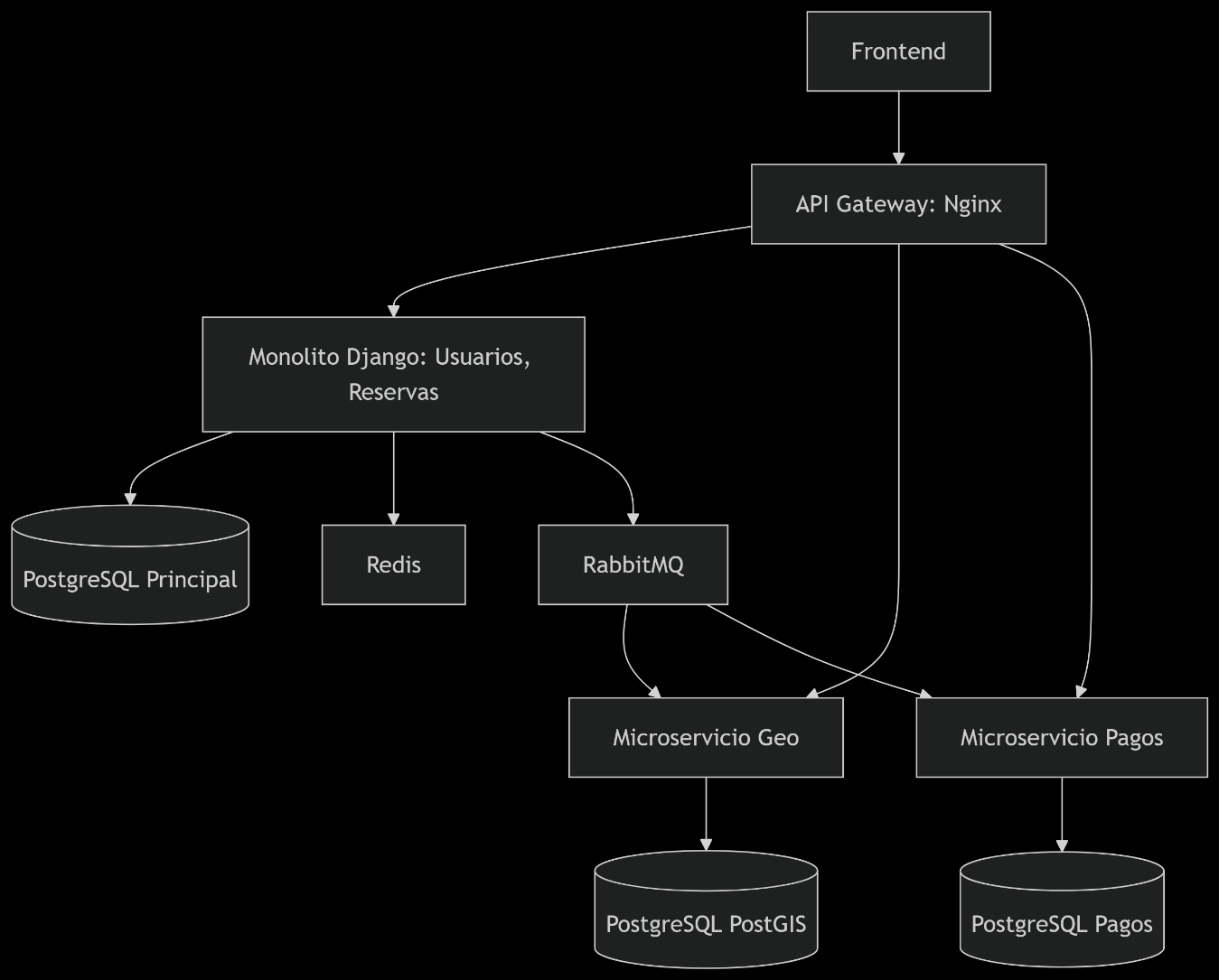
**¿Por qué?**

* Procesamiento de NLP (requiere librerías específicas como TensorFlow).

**Tecnología:**

* **Python + Flask + Transformers** (Hugging Face).

**Ejemplo**



**Estructura de Carpetas**

motuscar/

│

├── core/ # Monolito principal (Django)

│ ├── settings.py # Configuración compartida

│ ├── urls.py # Rutas del monolito

│ └── wsgi.py # WSGI para producción

│

├── apps/ # Módulos del monolito

│ ├── users/ # Autenticación y perfiles

│ ├── bookings/ # Reservas y citas

│ └── reviews/ # Reseñas

│

├── microservices/ # Microservicios estratégicos

│ │

│ ├── geo\_service/ # Geolocalización

│ │ ├── app/ # Código Python

│ │ │ ├── models.py # Modelos PostGIS

│ │ │ └── views.py # API de talleres cercanos

│ │ ├── Dockerfile # Contenedor específico

│ │ └── requirements.txt # Dependencias

│ │

│ ├── payment\_service/ # Procesamiento de pagos

│ │ ├── src/ # Código Node.js

│ │ │ ├── index.js # Endpoints Stripe

│ │ │ └── package.json

│ │ └── Dockerfile

│ │

│ └── notification\_service/ # Notificaciones async

│ ├── tasks.py # Celery + RabbitMQ

│ └── Dockerfile

│

├── gateway/ # API Gateway (Nginx/Kong)

│ ├── nginx.conf # Reglas de enrutamiento

│ └── Dockerfile

│

├── infra/ # Configuración de infraestructura

│ ├── docker-compose.yml # Orquestación de servicios

│ ├── postgres/ # Configs de PostgreSQL

│ └── redis/ # Configs de Redis

│

├── static/ # Assets estáticos (frontend)

├── docs/ # Documentación

└── scripts/ # Scripts de despliegue/backup

**Herramientas y Estructura Clave**

#### **1. Backend**

* **Lenguaje:** Python (Django).
* **Librerías Principales:**
  + Django REST Framework: Para construir APIs.
  + Django Filters: Para búsquedas avanzadas (repuestos/talleres).
  + Celery + RabbitMQ: Tareas asíncronas (emails, notificaciones).
  + Django PostGIS: Para geolocalización.

#### **2. Base de Datos**

* **Principal:** PostgreSQL con PostGIS.
* **Caché:** Redis.
* **Backup:** pg\_dump + AWS S3 (o Google Cloud Storage).

#### **3. Frontend**

* **Web:** React.js + Material UI.
* **Mobile:** React Native (opcional).
* **Mapas:** Mapbox GL JS o Google Maps SDK.

#### **4. Microservicios Clave** (Opcionales, escalan después)

* **Geolocalización:** Django + PostGIS (endpoint /nearby-workshops).
* **Pagos:** Node.js + Stripe API (endpoint /process-payment).

#### **5. DevOps**

* **Contenedores:** Docker (+ Docker Compose para desarrollo).
* **Despliegue:** AWS ECS (Fargate) o Google Cloud Run.
* **CI/CD:** GitHub Actions (automatizar tests y despliegues).
* **Monitorización:** Grafana + Prometheus (métricas), Sentry (errores).

#### **6. Seguridad**

* **Auth:** JWT con djangorestframework-simplejwt.
* **Cifrado:** pgcrypto en PostgreSQL para datos sensibles.
* **SSL:** Certbot (Let’s Encrypt) para HTTPS.