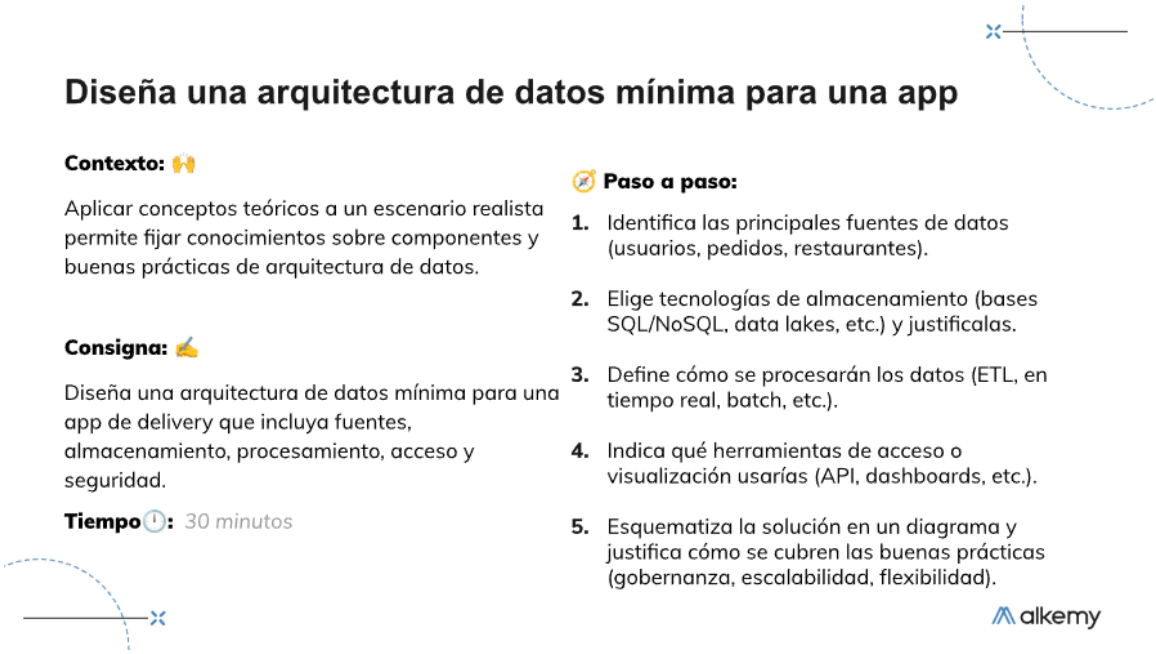
Arquitectura de datos mínima – App de Delivery



Propuesta mínima viable para una aplicación de delivery (tipo Rappi/UberEats) que cubre fuentes, almacenamiento, procesamiento, acceso/visualización y buenas prácticas.

1) Fuentes de datos

• App Cliente: registro, login, carrito, pedidos, geolocalización.

• App Repartidor: GPS en tiempo real, estados del pedido (asignado, retirado, entregado).

• Portal Restaurante: alta de menús, precios, stock, tiempos de preparación.

• Pasarela de pago: autorizaciones y liquidaciones.

• Soporte / notificaciones: tickets, push/email/SMS.

2) Almacenamiento (mínimo viable)

• OLTP relacional (PostgreSQL/Aurora): usuarios, direcciones, pedidos, ítems, pagos.

• NoSQL clave-valor (DynamoDB/Redis): carrito, sesiones y claves de acceso de alta concurrencia.

• Event streaming (Kafka/Kinesis): eventos de pedidos, telemetría GPS.

• Data Lake (S3/ADLS – parquet): datos crudos, logs, recibos, snapshots.

• Data Warehouse (BigQuery/Redshift/Snowflake): métricas y BI.

3) Procesamiento

• Tiempo real: Kinesis/Kafka → Flink/Spark Streaming/Lambda. Casos: ETA, asignación de repartidor, alertas.

• Batch/ELT: Airflow/Glue orquesta cargas al DW, auditoría de calidad de datos, reportes financieros.

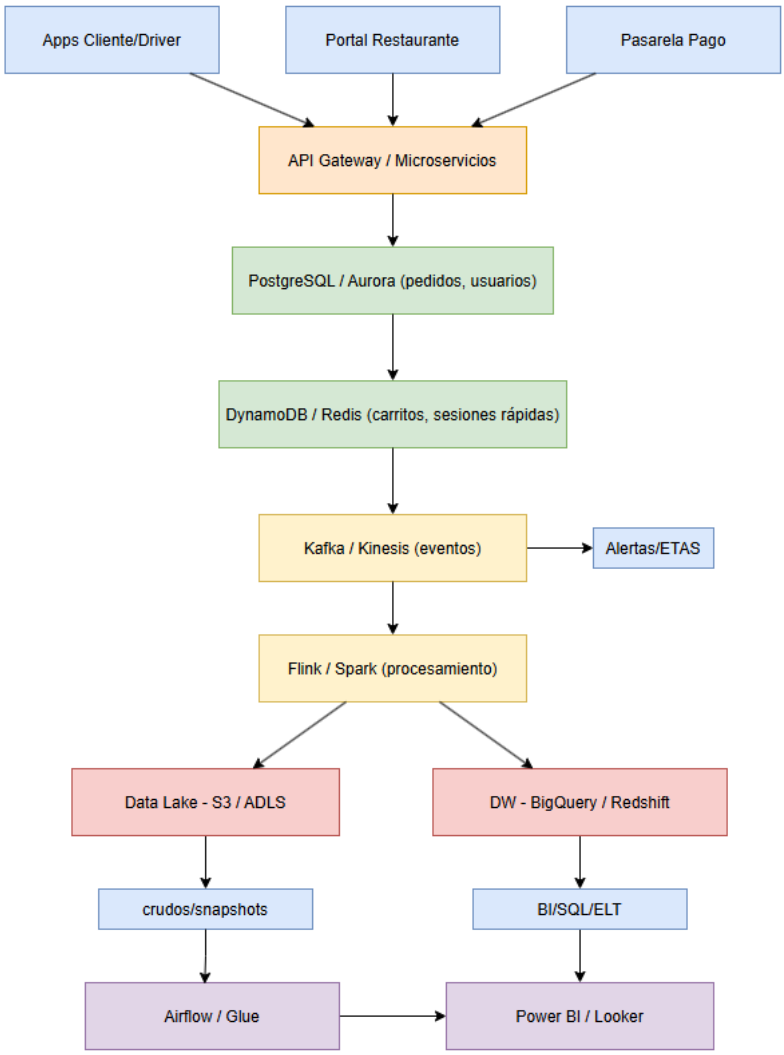
4) Acceso y visualización

• APIs (API Gateway + microservicios Node/Go/Python). GraphQL opcional para apps.

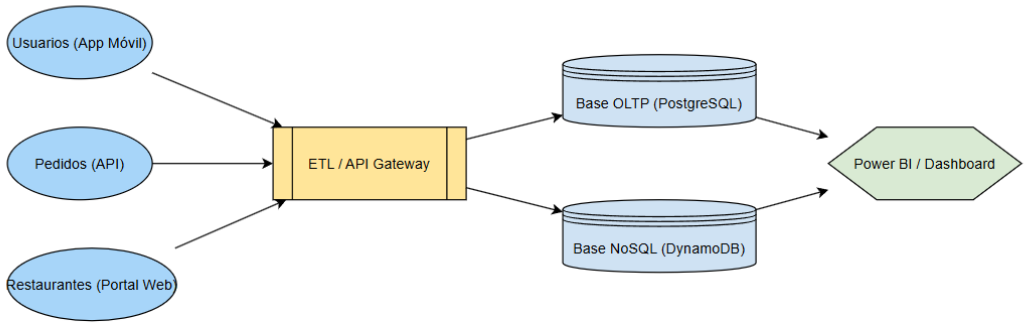
• BI (Looker/Power BI/Tableau) conectado al DW.

• Notificaciones (SNS/SES/Firebase).

5) Diagrama lógico



5.a) Diagrama simplificado:



6) Esquema relacional (OLTP - PostgreSQL)

DDL sugerida (extracto):

```
CREATE TABLE users (
  user_id        BIGSERIAL PRIMARY KEY,
  name           VARCHAR(120) NOT NULL,
  email          VARCHAR(160) UNIQUE NOT NULL,
  phone          VARCHAR(30),
  created_at     TIMESTAMP NOT NULL DEFAULT now()
);

CREATE TABLE restaurants (
  restaurant_id   BIGSERIAL PRIMARY KEY,
  name           VARCHAR(160) NOT NULL,
  address        VARCHAR(240),
  city           VARCHAR(80),
  created_at     TIMESTAMP NOT NULL DEFAULT now()
);

CREATE TABLE orders (
  order_id       BIGSERIAL PRIMARY KEY,
  user_id        BIGINT NOT NULL REFERENCES users(user_id),
  restaurant_id   BIGINT NOT NULL REFERENCES restaurants(restaurant_id),
  status         VARCHAR(24) NOT NULL, -- created|accepted|picked|delivered|canceled
  total_amount    NUMERIC(12,2) NOT NULL,
  payment_status  VARCHAR(24) NOT NULL, -- pending|approved|failed|refunded
  created_at     TIMESTAMP NOT NULL DEFAULT now(),
  updated_at     TIMESTAMP
```

```
);

CREATE TABLE order_items (
  order_item_id    BIGSERIAL PRIMARY KEY,
  order_id         BIGINT NOT NULL REFERENCES orders(order_id) ON DELETE CASCADE,
  sku              VARCHAR(80) NOT NULL,
  name             VARCHAR(160),
  qty              INT NOT NULL,
  unit_price       NUMERIC(12,2) NOT NULL
);

CREATE INDEX idx_orders_restaurant_created ON orders(restaurant_id, created_at);
CREATE INDEX idx_order_items_order ON order_items(order_id);
```

7) Tabla NoSQL (DynamoDB) – diseño de clave

• Tabla: delivery_app
• PK (partition key): pk | SK (sort key): sk
• Ejemplos de ítems:

- Carritos: pk='CART#<user_id>' sk='ITEM#<sku>'

- Sesiones: pk='SESSION#<user_id>' sk='TOKEN#<ts>'

- Pedidos (solo cache/lookup rápido): pk='ORDER#<order_id>' sk='META'

• GSI1 (por usuario): GSI1PK='USER#<user_id>' GSI1SK='ORDER#<created_at>' (para listar pedidos del usuario)

8) Ejemplos JSON (eventos/documentos)

a) Evento de pedido (stream):

```
{
  "event_type": "ORDER_STATUS_CHANGED",
  "order_id": 9123456,
  "user_id": 1001,
  "restaurant_id": 501,
  "old_status": "accepted",
  "new_status": "picked",
  "driver_id": 3007,
  "ts_event": "2025-08-08T18:35:22Z",
  "geo": {"lat": -33.456, "lng": -70.645}
}
```

b) Documento de carrito (DynamoDB estilo JSON):

```
{
 "pk": "CART#1001",
 "sk": "ITEM#SKU-123",
 "qty": 2,
 "price": 7.50,
 "restaurant_id": 501,
 "added_at": "2025-08-08T18:20:01Z",
 "GSI1PK": "USER#1001",
 "GSI1SK": "CART#2025-08-08T18:20:01Z"
}
```

c) Menú restaurante (para lake/NoSQL):

```
{
 "restaurant_id": 501,
 "menu_date": "2025-08-01",
 "items": [
  {"sku":"SKU-123","name":"Burger","price":7.50,"tags":["combo","beef"]},
  {"sku":"SKU-222","name":"Veggie Wrap","price":6.20,"tags":["vegan"]}
 ]
}
```

9) Buenas prácticas clave

• Gobernanza y seguridad: IAM/RBAC, KMS, cifrado en tránsito y en reposo, catálogo de datos, linaje.

• Escalabilidad: particiones por fecha/tienda en DW; diseño de PK/SK y GSIs en DynamoDB; autoscaling de streams.

• Calidad de datos: validaciones en el ELT (Great Expectations), DLQs para eventos defectuosos, contratos de esquemas (Avro/Protobuf).

• Costos: Lake para crudos, DW para consultas; caché para rutas críticas; compresión columnar (Parquet/Snappy).