Postgres Local First

App code works directly with client-side embedded database which auto syncs with backend database

Reads & writes go to local DB first

Benifits

App devs	End users
simple state mgmt	Instant reactive UX
Simple backend	Offline first
↓ backend compute	Real time multi-user collaboration

Primarily use Postgres & SQLite

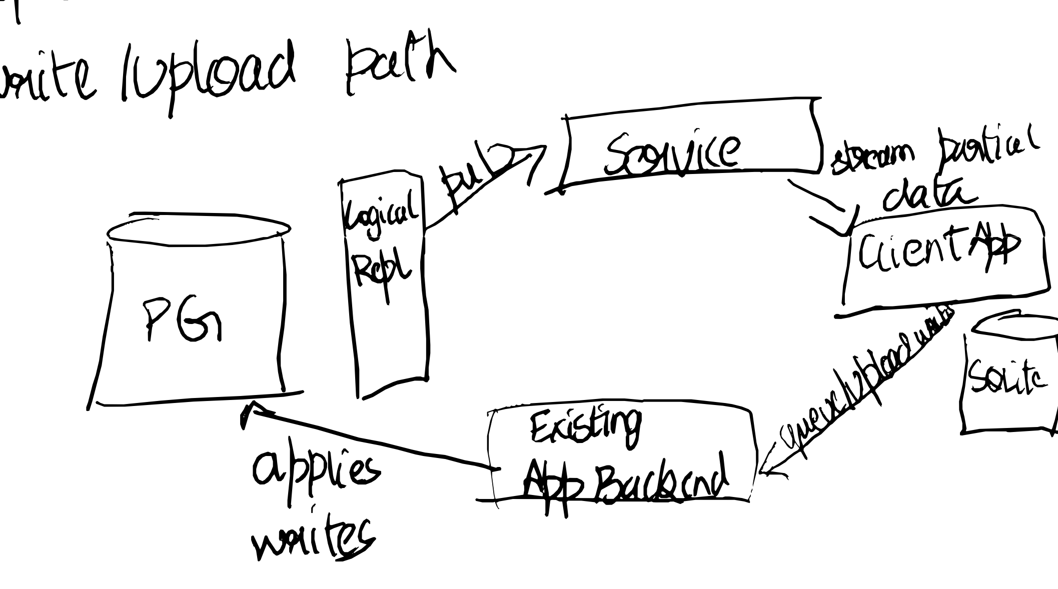
SQLite is hard to beat on client-side
- → Battle tested (trillion SQLite DB in active use)
- → Perf & aggregations

Objectives

- → Treat PG DB as sacrosanct
  - → Min changes to schema, config
  - → Don't bypass user logic
- → Dynamic partial replication

Architecture

Separate Read/Download path from write/upload path



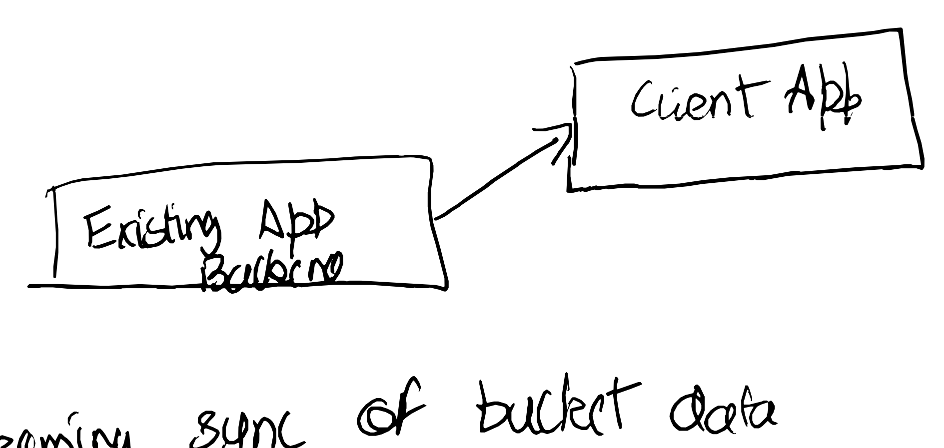
CRDT is key

Implementing dynamic partial Replication
- → Make arbitrary queries & keep results sets in sync
- → Problematic for high scalability

Solution
- → Buckets of data
- → Shared between clients
- → Changes to buckets are synced

Based on sync rules, replicate & pre-process data from PG. First take snapshot & then update incrementally (using logical replication)
- → Checkpoints keep track of LSNs
- → Store recent history of ops for each row
- → Rep as JSON
- → ops indexed by 'operation ID' which is strictly ↑
  - → Allows query of ordered ops efficiently

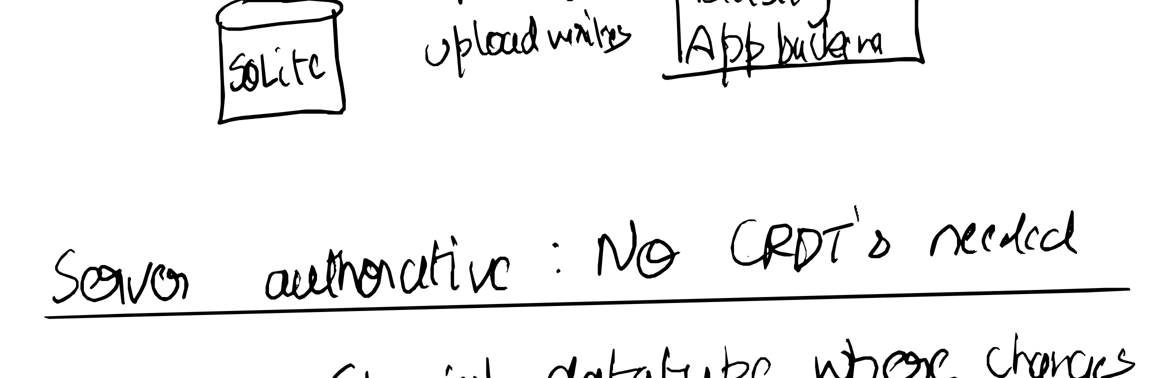
3. Auth users using JWTs



4. Streaming sync of bucket data from service to clients

5. On client, data is persisted to SQLite
   - → Type mapping from PG → SQLite
   - → Replicated data stored in schemaless format
   - → client side Schema
   - → Schema is applied as SQLite views on top of schemaless data
   - → Live reactive query hooks: Update UI when data changes. (React hooks)

6. Syncing local changes
   - → Changes written to local SQLite DB & also upload queue
   - → Dev defines own func for uploading changes to backend & write to PG



Server authoritative: No CRDT's needed

CRDT → Special datatype where changes can be merged in any order & resolution is always deterministic

However, NO CRDT's used in this version:

Alternative: Always merge changes in same order

Guarenteeing consistency

- → Track write ckpts on server
  - → Ckpts have op-id & Postgres LSN
- → Local client sides writes to SQLite database
  - → Applied on top of last recvd ckpt from server
  - → Added to upload queue
- → Client retrives latest ckpt from server AFTER it has finished uploading writes
  - → Client update local state to match server state.

Guaranteeing data integrity

- → Checksum computation on bucket. when mismatch, download from bucket

Problems & Solution

1. Handling schema/DDL changes

   DDL changes not propageted by PG in logical replication

   Hence, schemaless

   DDL → CREATE, DROP, RENAME, REPLICA IDENTITY

2. LSN's that overlap btw transactions
   - → LSN's are not guaranteed to be monotonically increasing
   - → sharded DB support
   - → Reprocess sync data

3. Accessing TOASTed values

4. Handling diff row id permutations

   Implemented support for diff permutations