From supermarket prices to decisions

Basket Desk SG collects daily online grocery listings from FairPrice, Sheng Siong and Cold Storage, preserves their history, and makes the records available through a public, configurable dashboard.

The business question

Which item became more expensive, at which supermarket, between which observations - and how does its price compare with the same item elsewhere? A current storefront answers "what is listed now"; a dated history lets us investigate change.

A 60-second explanation

"I built a daily price-history pipeline for three Singapore supermarkets. Their storefronts return data in different formats, so I use a separate collector for each and standardise the records into one shared schema. I preserve each day's observations in Cloudflare, verify them before converting them to Parquet, and query the history with DuckDB. I match products conservatively so different pack sizes are not treated as equivalent. Finally, I publish date-partitioned data to an interactive dashboard where users choose fields, filters, tables or charts, download records and save their setup. The key engineering concerns are reliable coverage, honest price comparisons and keeping the system within a small resource budget."

Follow the data in six steps

1. Collect Cloudflare and Windows collectors read the catalogues.	**2. Standardise** Map source values into 18 consistent columns.	**3. Preserve** Cloudflare D1 holds snapshots; KV holds collection reports.
4. Analyse Read Parquet as a shared DuckDB table; derive changes and matches.	**5. Publish** Export compressed date partitions to Cloudflare Pages.	**6. Explore** Filter, compare, chart, export and save a dashboard.

Evolution: source feasibility came first, followed by daily history, comparable tables and a public MVP. User feedback then shaped the Aeolus-inspired field shelves, browser-worker queries, partition-date filters, explanations and saved views.

This is an implemented proof of concept. It records observed online listings; it is not a live checkout quote, a complete physical-store survey or an official inflation measure.

1. Extract the catalogue, one source at a time

A collector is a small program that discovers departments, requests their product pages and records the returned listings. It aims to cover the accessible catalogue, rather than a hand-entered shopping list. A separate adapter handles each retailer's format.

Source and runner	How the implemented collector works
FairPrice Cloudflare	Reads department navigation from the homepage, then uses the storefront's `layout/category/v2` and `product/v2` endpoints under `website-api.omni.fairprice.com.sg/api/`. Requests specify delivery store 165. Where a category's advertised count exceeds its accessible pages, it tries price-band, brand or child-category subdivisions and checks coverage.
Sheng Siong Windows workstation	Uses the storefront WebSocket at `wss://shengsiong.com.sg/websocket`. `Categories.getByLevels` discovers categories and `Products.getByAllSlugs` returns listings in growing batches of 80. The collector checks the previous batch boundary, then accepts only new records.
Cold Storage Cloudflare	Uses homepage navigation and the category service at `csp.coldstorage.com.sg/api/catalog/category/all`. Category pages expose the category identifier; a storefront server action returns pages of 30 products. The collector decodes the response and checks page numbers and advertised totals.

Why use two execution environments?

FairPrice and Cold Storage run in Cloudflare without the laptop. Sheng Siong's WebSocket worked from the workstation but did not establish correctly from Cloudflare, so its scheduled collector runs locally and uploads to the same archive. This is a practical dependency, not a fully cloud-based design.

How requests stay controlled

- FairPrice uses one request in flight, with starts at least one second apart. Rate limiting can trigger up to two retries, respecting the requested wait where supported and slowing subsequent requests.
- Individual network requests have a 25-second timeout. Ordinary transient page failures receive a bounded retry; continued errors become recorded gaps.
- Access rejection or exhausted rate-limit retries stop the affected run. The collectors do not assume that every advertised item was retrieved.

These are storefront interfaces used by the current implementation, not contracted public developer APIs. Interface changes, access restrictions and permitted reuse require ongoing review. Giant is not currently collected.

2. Turn different responses into consistent records

Standardisation means mapping differently named source values into the same columns. For example, one retailer's `productId` and another's `id` become `product_id`. The retailer is retained, so IDs from different companies cannot be confused.

What does one row represent?

One row is one retained product listing, from one retailer and storefront context, in one daily collection. Products appearing in several categories are deduplicated by product ID within that retailer's run; the first accepted occurrence is kept. This also means the category is the retained source category, not a universal grocery taxonomy.

Decision	Why it matters
Save every day's price	Unchanged prices are recorded again. This distinguishes "observed and unchanged" from "not collected".
Store money as cents	Whole-number `regular_cents`, `offer_cents` and `deposit_cents` keep monetary values consistent. The dashboard converts them to SGD by dividing by 100.
Keep missing values empty	A missing offer is not a zero-dollar offer. Stock can be true, false or unknown. An unidentified product or blank name is rejected and counted.
Keep identity and evidence	Names, pack size, brand, barcodes, source context, timestamps and available promotion details support later checks.

"Regular price" has source-specific meaning

FairPrice: the configured store's reference price is used, with a source fallback. The summary offer only accepts qualifying unrestricted, single-unit fixed-price offers within their validity period. Other collected promotion rules remain in Offer details.

Sheng Siong: when the source's reference price exceeds its current price, the higher value becomes Regular price and the current value becomes Recorded offer. Otherwise the current value becomes Regular price.

Cold Storage: the source price becomes Regular price; a positive, lower promotional price becomes Recorded offer. A pack measurement can be inferred from the title when a size field is missing.

Three dates, three different roles

`observed_date` identifies the Singapore date assigned to the collection run. `observed_at` is the page observation timestamp, displayed in UTC. `p_date` is a derived reporting partition: **observed_date minus one calendar day**.

A run dated 10 November 2026 has `p_date = 2026-11-09`. Both dates remain available. This relabelling does not turn a price collected on 10 November into an observation made on 9 November; use Observed date/time when explaining evidence.

3. Preserve history, then expose a queryable table

The system separates collection storage from analytical storage. The cloud archive protects daily history; the local warehouse makes that history convenient to query.

Layer	What is stored and why
Temporary collection Durable Object	A Cloudflare component with durable state keeps the current queue, progress and compressed pages. Each cloud retailer has its own collector. Page progress survives between scheduled processing steps.
Daily archive Cloudflare D1	Deduplicated records are written as one JSON record per line, then gzip-compressed. D1 holds these bytes in chunks, with a separate snapshot record containing the size, chunk count and SHA-256 checksum.
Collection reports Cloudflare KV	A manifest records the date, status, row count, gaps and archive location. Separate pointers identify the latest outcome and the latest complete run.
Analytical files Local Parquet	The downloader checks the archive checksum, size and row count, then writes one compressed Parquet file per retailer and collection date. Parquet is a compact, typed table format suited to analysis.
Warehouse queries DuckDB	DuckDB reads the Parquet files directly. It exposes their combined contents as `price_observations` and builds analytical SQL views. No always-on database server is required.

How three sources become one table

Each adapter produces the same 18-column schema. The files are then **stacked as rows**, matching columns by name. This is a union, not a product match: a FairPrice listing and a Cold Storage listing remain separate observations. Source identity and context are retained in every row.

DuckDB creates the combined table view when queried. The archive is not one large grocery SQL table inside D1: D1 stores compressed snapshots, while DuckDB supplies the analytical table over local files.

What "raw data" means in this project

The collectors receive full website responses, but the long-term archive retains **selected, standardised fields**. It does not retain every original HTML/API response or image. The manifest's `rawBytes` is a received-volume counter, not a retained raw-file archive. Dashboard "Source rows" are those processed observations, with reporting and matching fields added.

This saves storage and simplifies analysis. The trade-off is that a newly desired source field generally needs future collection; it cannot always be recovered from past snapshots.

4. Derive useful answers without misleading matches

Warehouse view	Business meaning
`price_observations`	All downloaded dated observations. This is the base history.
`retailer_products`	The latest collected record for each retailer, storefront context and product ID. Different products can have different latest dates.
`product_offers`	Expands collected promotion arrays into separate offer rows for analysis.
`price_changes`	Compares successive non-missing Regular prices for the same retailer, context, product ID, pack size and currency. Returns only changed values.
`product_comparisons`	Pairs in-stock listings across retailers on the same observed date, with matching currency, pack and sufficient identity evidence.

A real price-change example

Cold Storage's **Dayplus Banana Bun 80g** was recorded at **S$1.65** on 9 November 2026 at 15:25 SGT and **S$1.80** on 10 November at 01:34 SGT. The warehouse reports a **S$0.15 increase, or 9.09%**.

Calculation: change = new price - previous price; percentage change = change / previous price x 100. A previous price of zero produces no percentage. This example comes from the local collected history, not a hypothetical price.

The evidence brackets the change between two observations. It does not identify the exact moment the retailer changed its price. If a day's collection is missing, the comparison spans the gap. A promotional reference price alone is not proof of a historical increase.

How identical products are matched

1. Require the same recorded pack key and currency. The pack key removes whitespace and case differences; it does not generally convert "1 kg" into "1000 g".
2. Prefer a shared barcode plus matching pack. Otherwise require exact normalised title/brand word agreement, an explicit supported pack measurement and additional ambiguity checks. Conflicting non-empty barcodes prevent a name-only match.
3. Exclude ambiguous many-to-many matches. A dashboard comparison group contains at most one listing per retailer, and every pair in the group must have direct matching evidence.

Accepted pair evidence from collected history creates a Comparable product label for the matching listings. The original names and Match evidence remain visible. Unmatched observations are still included in the dashboard; remove the Comparable product filter to search them.

The matching policy favours defensible comparisons over maximum match coverage. It can miss valid equivalents and cannot guarantee that every accepted match is perfect.

5. Run daily and publish only available evidence

Stage	Current schedule and behaviour
Start collection	Daily from **01:30 Singapore time**. A two-minute Cloudflare trigger starts or resumes FairPrice and Cold Storage work; it does not create a fresh full snapshot every two minutes. Sheng Siong starts through Windows Task Scheduler.
Recover locally	The Sheng Siong task has a sign-in check, missed-start handling and up to three task retries at 20-minute intervals. A pending upload can be reused. A day entirely offline has no Sheng Siong observation.
Check readiness	The Windows publisher checks every 20 minutes from 01:30 and at sign-in. Normally it waits for all three retailers to reach a final status. From 07:30 it can publish available archives if both cloud collectors finished and Sheng Siong remains pending.
Verify and export	The publisher downloads new or changed archived snapshots, rebuilds comparisons and exports all locally cached observations. It produces a small metadata index and one compressed file per `p_date`, with a content hash in each data filename.
Deploy	The application, documents, index and data partitions are deployed together to Cloudflare Pages. A local lock prevents overlapping publishers; the input fingerprint avoids unnecessary data publications.

Collection time is not publication time

01:30 is the start, not a promise that new data is already on the website. Collection duration, retries and workstation availability affect publication. Sheng Siong collection and dashboard publication require the workstation to be on, connected and able to run the signed-in tasks.

A final status can be **complete, partial, failed or blocked**. A partial archive can still be useful and is published with its coverage status. "All collectors finished" does not mean "all listings were collected".

Verified history at documentation time

Collection date	Reporting p_date	Observations
9 Nov 2026	8 Nov 2026	54,028
10 Nov 2026	9 Nov 2026	56,262
Total	Two reporting partitions	**110,290**

For the 10 November collection, FairPrice recorded 38,272 listings with 84 gap flags; Cold Storage recorded 10,250 and Sheng Siong 7,740, both marked complete. These are dated facts for this walkthrough, not live counters.

Reopening the dashboard fetches the current published index. Its latest reporting date therefore reflects available published data, even when a scheduled run is delayed.

6. Give users a simple analytical workbench

The public dashboard uses Aeolus-inspired field shelves. Users build a question by choosing **Dimension**, **Metrics** and **Filters**, then clicking **Run query**. They can drag fields or use the add buttons.

Control	What the user gets
Dimension	Every exported column is available, including raw price fields. Adding a price here displays/groups by that value; it does not automatically turn the field into MIN(price).
Metrics	Explicit SUM, AVG, MIN, MAX, COUNT or distinct count, or a calculated formula. Up to six metrics are supported. Hover/tap explanations show field meaning and a SQL equivalent for calculations.
Filters	Date ranges, selected values, text search and numeric ranges. Filters combine with AND. `p_date` defaults to Latest available; a fixed date range stays fixed.
Views	Grouped Table, Source rows, bar, line and scatter formats. Tables show 25 rows per page; charts show a selectable first 12, 30 or 60 groups, with point details.
Download CSV	Grouped results or filtered source rows, up to 1,000,000 matching rows per file. The download is not restricted to the visible table page.
Save dashboard	Remembers selected fields, formulas, filters, view, chart settings and sorting in this browser. No login is needed. Other devices have separate saved setups.

Why it stays more responsive

A background browser worker loads relevant date partitions and performs filtering, grouping, sorting and CSV preparation. The main page receives small chart previews or table pages rather than rendering tens of thousands of rows at once. A date filter avoids loading unrelated days; removing it may require all historical partitions.

This is client-side analysis, not a server query API. The SQL shown for a metric explains its meaning; the browser actually evaluates the supported expression in JavaScript. Large all-history queries and million-row exports are still bounded by the device's memory and download speed.

Example: compare one product fairly

Keep the latest `p_date`, select a Comparable product, and display Supermarket, Source product name, Pack size, Regular price and Recorded offer in a table. Check Match evidence, Recorded stock and Offer details before concluding which listing is cheaper.

Saving remembers the setup, not frozen prices. Reopening or refreshing runs it against newly published data. A page left open keeps its existing results until reloaded. See the linked Dashboard guide for click-by-click instructions.

7. Make quality and limitations visible

Risk	Control and remaining limitation
Incomplete catalogues	Validate source shapes, page metadata, repeated pages, advertised counts and capped categories. A drop below 80% of the previous complete catalogue is flagged. These checks detect problems; they do not prove universal coverage.
Duplicate records	Deduplicate by product ID within a retailer's daily run. Repeated observations across different dates are intentionally retained.
Corrupt transfers	Verify archive byte size, checksum for D1 snapshots and expected row count before committing the local Parquet file. Temporary files are cleaned up after conversion.
Misleading comparisons	Separate regular prices, offers and deposits. Require product/pack identity for cross-retailer comparisons. Missing values remain missing; collection gaps are not filled with invented observations.
Source or machine failure	Persist cloud progress, use bounded retries and retain pending local uploads. The website keeps serving its last published data; local collection/publication can be delayed while the workstation is unavailable.

Resource controls are finite, not unlimited retention

The implementation allocates four D1 archives with a **360 MB compressed-payload budget each**, or 1.44 GB combined. A single archived snapshot is guarded at 15 MB; the publisher checks a 25 MiB ceiling per website asset. A full archive reports a storage block and retains unpublished staged data. There is no automatic history-deletion policy.

The local uploader/download cache enforces a **512 MiB project-data cap** and requires **10 GiB free disk**. Warehouse queries use a 384 MB DuckDB memory limit and 256 MB spill limit. The exporter has a 384 MB DuckDB limit; its Python objects and the browser are additional memory users. These guards do not cap all disk and memory use on the computer.

The three 10 November archives total about **3.62 MB**; the six local Parquet files for both days total **5.80 MB**. Those are measured sizes, not a promise about future growth. Storage retention and browser performance need reassessment as history grows.

Current scope and the next sensible improvements

- **Built:** daily observations for three sources, history, SQL analysis, conservative matching, a public workbench, CSV and saved browser views.
- **Not built:** Giant ingestion, CPI/utility integration, a validated cheapest-basket engine, automatic price alerts or server-side saved dashboards.
- **Next:** resolve persistent coverage gaps, improve matching validation, move remaining workstation dependencies to a suitable unattended runner, and plan retention/server-side queries before growth exceeds current budgets.

The current stack uses JavaScript collectors, Cloudflare scheduling/storage, Python, Parquet, DuckDB and a static web app. Dagster, dbt, FastAPI and R2 were considered in the proposal but are not part of the deployed execution path.

8. A five-minute showcase

1. **0:00-0:45 | Start with the question.** "I want to know which supermarket item changed price over time, and compare equivalent products." Show the six-step workflow on the opening page.
2. **0:45-1:30 | Explain the engineering choice.** Three source adapters produce one standard observation schema. Daily snapshots keep unchanged prices, so missing collection is not confused with a stable price.
3. **1:30-2:30 | Show a real historical change.** Set `p_date` to 8-9 Nov 2026, remove Comparable product, filter Supermarket to Cold Storage and Source product name to contain "Dayplus Banana Bun". Display Observed date, Source product name, Pack size and Regular price. The collected values are S$1.65 and S$1.80.
4. **2:30-3:15 | Explain evidence.** Point out the two observation dates and the 9.09% calculation. Explain that the change occurred between observations, not at a known exact time. For a line chart, keep the item filter, group by Observed date and use AVG(Regular price) only after confirming one item per day.
5. **3:15-4:00 | Show cross-retailer comparison.** Reset to the sample matched product. Show original listing names, pack size, regular price and offer price. Explain barcode/pack or exact title/brand matching and why unmatched products remain searchable.
6. **4:00-5:00 | Close with usability and trade-offs.** Switch Table/Source rows, show CSV and Save dashboard, then open Data coverage. Explain the remaining workstation dependency and partial FairPrice coverage honestly.

Likely interviewer questions

Question	A concise answer
Why full daily snapshots?	They preserve evidence of stable prices and enable reproducible historical queries. The trade-off is storage growth.
Why Parquet and DuckDB?	Parquet gives compact, typed historical files. DuckDB queries them directly, avoiding a permanently running warehouse server at this scale.
Why not match by name alone?	Names can hide different sizes or variants. I require stronger evidence and reject ambiguity, accepting fewer matches for more defensible comparisons.
What does p_date mean?	It is the chosen reporting partition, one day before the run date. Exact collection dates and timestamps are retained separately.
What would you improve first?	Coverage and unattended operation, then retention and query scalability. More charts alone would not improve the reliability of the answer.

Avoid claiming that two days establish a long-term trend, that "complete" means every Singapore grocery item, or that the workbench already contains an automatic "all price increases" report. That derived report exists as a warehouse SQL view; users can inspect selected item histories in the dashboard.

9. The dashboard's 21 columns

The base archive has 18 columns. Export adds `p_date`, Comparable product and Match evidence, giving 21 dashboard columns. The three cents columns are presented as SGD, and the barcode list is displayed as text. Original and reporting dates are both retained.

Column	Meaning
Supermarket	The retailer whose storefront supplied the observation.
Comparable product	A shared label assigned only to accepted matched listings; blank when unmatched.
Source product name	The original listing title, useful for search and match review.
p_date	Reporting partition; collection run date minus one day.
Observed date	The Singapore date assigned to the collection run.
Observed time	Page observation timestamp, displayed in UTC; add eight hours for Singapore time.
Category	The retained source category; categories differ between retailers.
Pack size	Recorded or source-title-derived measurement. Check it before comparing products.
Brand	The separate source brand field; it may be blank even when the title names a brand.
Recorded stock	In stock, out of stock or unknown at observation time.
Product ID	The retailer's identifier, kept as text and used together with Supermarket.
Listing slug	The source's URL-style listing identifier when supplied.
Barcodes	Accepted 8-14 digit source barcodes; displayed as comma-separated text.
Seller	The listing seller when the adapter supplies one, such as a marketplace seller.
Currency	SGD in the current collectors.
Store context	`delivery-165` for FairPrice; `storefront-default` for the others.
Offer details	Selected structured promotion details; an empty array is not a full promotion audit.
Match evidence	The matching rule supporting the assigned comparison label.
Regular price	The mapped regular/reference price in SGD; see source-specific definitions.
Recorded offer	The qualifying or mapped offer price in SGD; blank when none was accepted.
Container deposit	The separately collected deposit in SGD, when supplied; missing is not confirmed zero.

These columns start under Dimension. Metrics are explicit calculations over them. For extended definitions and controls, use the Dashboard guide.

10. Supporting implementation map

This document describes the code and stored evidence reviewed on **10 November 2026**. It is a project explanation, not a real-time status page. The dashboard's Data coverage section is the place to check the currently published observations.

Responsibility	Implementation reference
Source adapters and mapping	`sg-grocery-price-pipeline/src/sources.mjs`
Pagination and validation	`src/crawl.mjs`
Cloud scheduling and progress	`src/worker.mjs` and `wrangler.jsonc`
Compressed cloud archive	`src/archive.mjs` and `sql/archive.sql`
Local collection and verified sync	`scripts/local-collect.mjs`, `scripts/warehouse.py` and `scripts/install-sheng-siong-task.ps1`
Analytical views	`sql/views.sql`; used to verify the Dayplus example.
Dashboard data preparation	`basket-desk-sg/export_snapshot.py`
Daily publication	`publish_daily.py` and `install-publish-task.ps1`
Query engine and presentation	`dist/query-worker.js`, `dist/query.js`, `dist/results.js` and `dist/workbench.js`

A small glossary

Pipeline: the repeatable steps from source to usable data. **Schema:** agreed column names, types and meanings. **Snapshot:** a retained set of observations from one collection. **Manifest:** the collection report describing that snapshot. **Partition:** a dated slice that can be loaded separately.

Warehouse: the organised analytical data and query layer. **View:** a reusable SQL definition of a table-shaped answer. **Checksum:** a content fingerprint used to detect damaged or mismatched data. **Aggregation:** combining multiple values into a summary such as an average or count.

Project links

Public dashboard | Dashboard guide | Online project walkthrough

Retailer storefronts: FairPrice, Sheng Siong and Cold Storage. The project uses their observed online listings and does not claim retailer endorsement.

A strong explanation connects every technical choice to an outcome: consistent records, preserved history, defensible comparisons, visible quality and a usable decision tool.