CalCOFI.io

Documentation

Benjamin D. Best Erin Satterthwaite

2026-04-29

Table of contents

1 Process

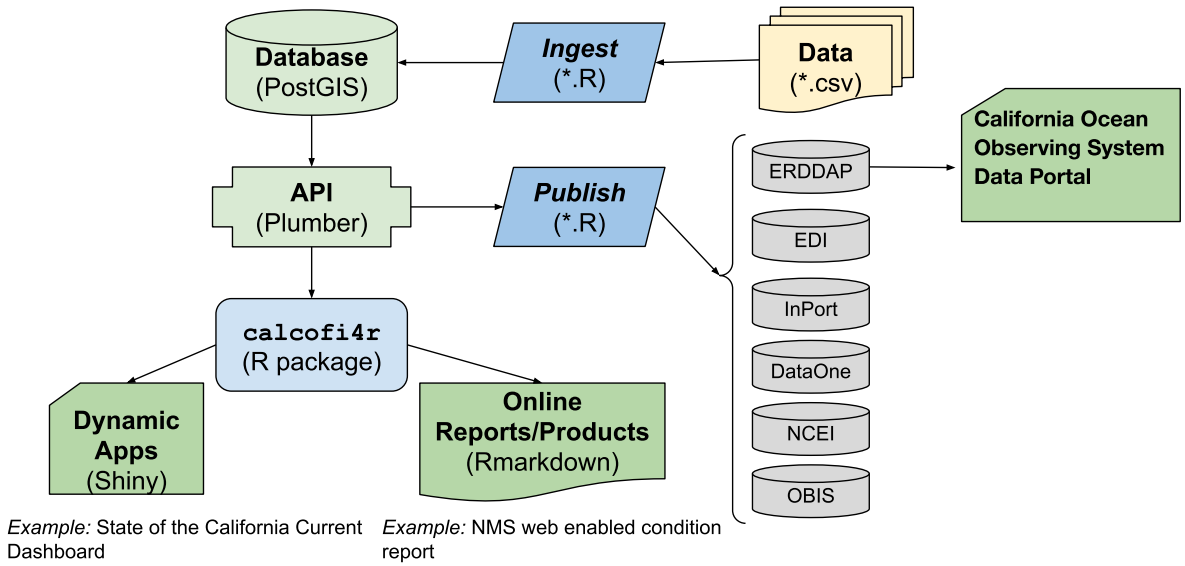


Figure 1.1: CalCOFI data workflow.

The original raw **data**, most often in tabular format [e.g., comma-separated value (*.csv)], gets **ingest**ed into the **database** by R scripts that use functions and lookup data tables in the R package `calcofi4r` where functions are organized into *Read*, *Analyze* and *Visualize* concepts. The application programming interface (**API**) provides a program-language-agnostic public interface for rendering subsets of data and custom visualizations given a set of documented input parameters for feeding interactive applications (**Apps**) using Shiny (or any other web application framework) and **reports** using Rmarkdown (or any other report templating framework). Finally, R scripts will **publish** metadata (as Ecological Metadata Language) and data packages (e.g., in Darwin format) for discovery on a variety of data ***portals*** oriented around slicing the tabular or gridded data (ERDDAP), biogeographic analysis (OBIS), long-term archive (DataOne, NCEI) or metadata discovery (InPort). The **database** will be spatially enabled by PostGIS for summarizing any and all data by ***Areas of Interest*** (AoIs), whether pre-defined (e.g., sanctuaries, MPAs, counties, etc.) or arbitrary new areas. (Figure 1.1)

2 Reports

2.1 Sanctuaries

- Channel Islands WebCR
  web-enabled Condition Report
  - Forage Fish
    example of using calcofi4r functions that pull from the API
- UCSB Student Capstone

3 Applications

- CalCOFI Oceanography
  oceanographic summarization by arbitrary area of interest and sampling period
- UCSB Student Capstone

4 Maps

4.1 Overview

The CalCOFI interactive app renders **hexagonal heatmaps** of species occurrence and oceanographic variables — sardine larvae densities, sea-surface temperature, salinity, chlorophyll — across the entire CalCOFI sampling area, at any zoom level, in seconds. The engine that makes this possible is **H3T**, a tile service that converts a database query into a stream of hexagon tiles on demand.

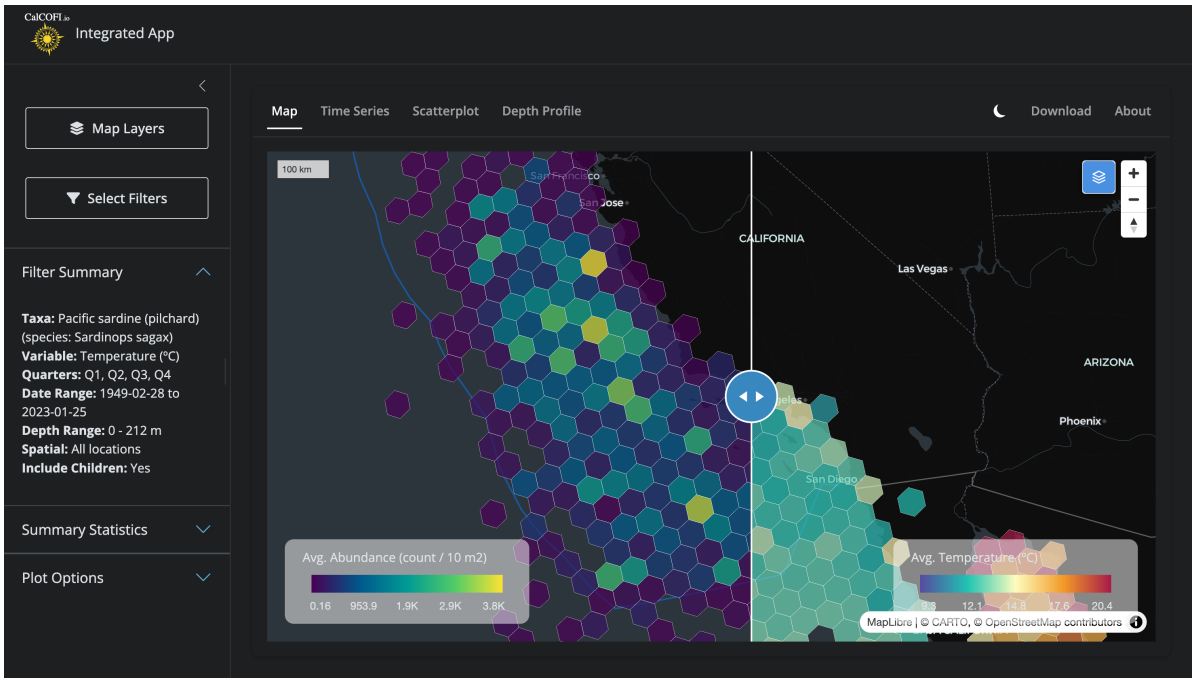


Figure 4.1: Side-by-side comparison in the CalCOFI Integrated App: hexagons summarizing **biology** (Pacific sardine larvae abundance, left) and **environment** (sea-surface temperature, right) over the same area, era, and quarters, with a swipeable divider for direct visual comparison. Both layers are powered by the H3T tile service.

This chapter explains how H3T works, visually, for both the curious oceanographer and the developer who wants to extend it. Throughout, the guiding principle is:

DuckDB is the single source of truth. H3T tiles are a performance layer built on top of it — never an authoritative copy of the data.

4.2 Why hexagons?

H3 is Uber's hexagonal hierarchical geospatial index. Each cell is a hexagon, and every cell at resolution N nests neatly inside one parent cell at resolution $N - 1$. Hexagons have three useful properties for ecological data:

- **Equal area at a given resolution**, so densities are directly comparable across the map.
- **Every neighbor is the same distance away** (unlike a square grid, which has both edge and corner neighbors).
- **No latitude distortion** — a res-5 hex off Point Conception covers the same area as a res-5 hex off Cabo San Lucas.

Compared to a raster of pixels, hexagons aggregate point observations (a net tow, a CTD cast, a larvae count) without imposing an arbitrary north–south orientation, and the parent–child nesting lets the same dataset render coarsely when you're zoomed out and finely when you're zoomed in.

4.3 The problem: aggregating millions of points at every zoom

Before H3T, the int-app pre-computed every hex aggregation across **all 10 H3 resolutions** for every species and environmental variable at startup. The Shiny app then shipped the relevant resolution to the browser for each zoom change. This gave the app two problems:

1. **Slow startup.** Building those aggregations against the full bottle and ichthyo tables took >10 s every time the app launched.
2. **Heavy memory.** A multi-resolution `sf` grid for one variable is hundreds of megabytes; for dozens of variables it crowded out everything else on the Shiny server.

This is the same class of problem that the MarineSensitivity stack solved for rasters with a TiTiler factory + Varnish cache. H3T is the equivalent move for **hexagons**.

4.4 The solution: query-as-URL hexagonal tiles

The architecture rests on three ideas, each covered below: the H3 resolution that's served depends on the zoom level; the entire tile request lifecycle is fronted by a cache; and a `release` parameter handles cache invalidation automatically when new data drops.

4.4.1 Resolution follows zoom

When the user zooms out to see the whole California Current, they don't need 7-meter hexagons — a 22 km hex is finer than the underlying sampling grid anyway. When they zoom in to a single CalCOFI line, they want as much detail as the data supports. H3T maps web-map zoom levels to H3 resolutions on the server side: zoom 0–1 → res 1 (~1106 km hexes), zoom 5–7 → res 5 (~22 km hexes), zoom 11+ → res 10 (~7.5 m hexes), and so on (Figure 4.2).

The trick is in the SQL: the int-app sends a **template** with a literal placeholder `hex_h3res{{res}}` (see `build_sp_sql()` in `int-app/app/functions_h3t.R`), and the API substitutes the right resolution for each tile before executing. One query string, ten zoom levels, one shared cache namespace.

4.4.2 Request lifecycle

When the user pans or zooms, MapLibre asks for tiles. Each tile URL looks like this:

```
h3tiles://h3t.calcofi.io/h3t/{z}/{x}/{y}.h3t?q=<base64-SQL>&release=v2026.04.08
```

The clever bit is the `q` parameter: it's the entire `SELECT` statement, base64-encoded with URL-safe characters (RFC 4648 §5). That means **the URL itself describes the data to render**, so two users asking for "average temperature for sardines, all quarters, 1949–2024" hit the exact same cache entry — no session state, no server-side query registry.

What happens next is shown in Figure 4.3. Caddy terminates TLS and forwards to Varnish, which checks its cache. On a hit, the user gets back a chunk of h3j JSON in well under 100 ms. On a miss, Varnish forwards to the Plumber API, which:

1. **Decodes** the base64 SQL.
2. **Validates** it with `sqlglot`: single statement, must be `SELECT` (or `WITH … SELECT`), must project exactly `cell_id, value, [n]`, no `read_csv`/`attach`/`load`, no shell calls, no schema beyond the published tables.
3. **Substitutes** `{res}` from the zoom level and wraps the inner query in a viewport bounding-box filter.
4. **Executes** against a read-only DuckDB connection with a 3-second timeout and a 50,000-row cap.
5. **Formats** the result as h3j JSON: `{"cells": [{"h3id": "8a1941a29a7ffff", "value": 12.37, "n": 42}, …]}`.

The validation, read-only mode, timeout, and row cap are stacked guardrails: the public URL accepts arbitrary SQL because **none of the layers will let it do harm**. Even if a determined actor slipped a malformed statement past sqlglot, the DuckDB connection has no write permission, no extension load, and a hard execution budget.

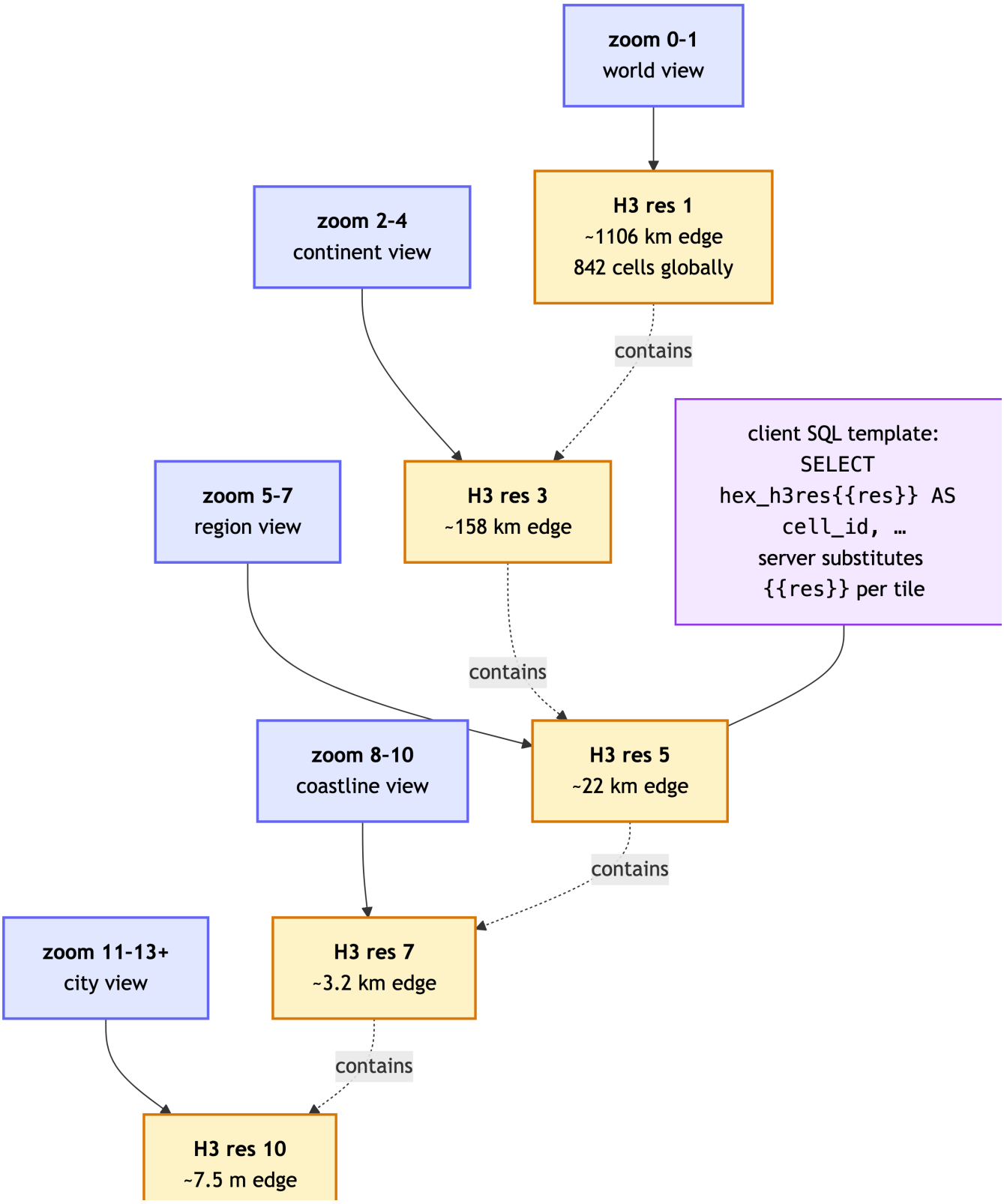


Figure 4.2: Web-map zoom level determines which H3 resolution the server returns. The same SQL template — `hex_h3res{{res}}` — adapts to each tile by server-side substitution. Hexagons at finer resolutions nest inside their coarser parents, so the data hierarchy and the visual hierarchy stay aligned.

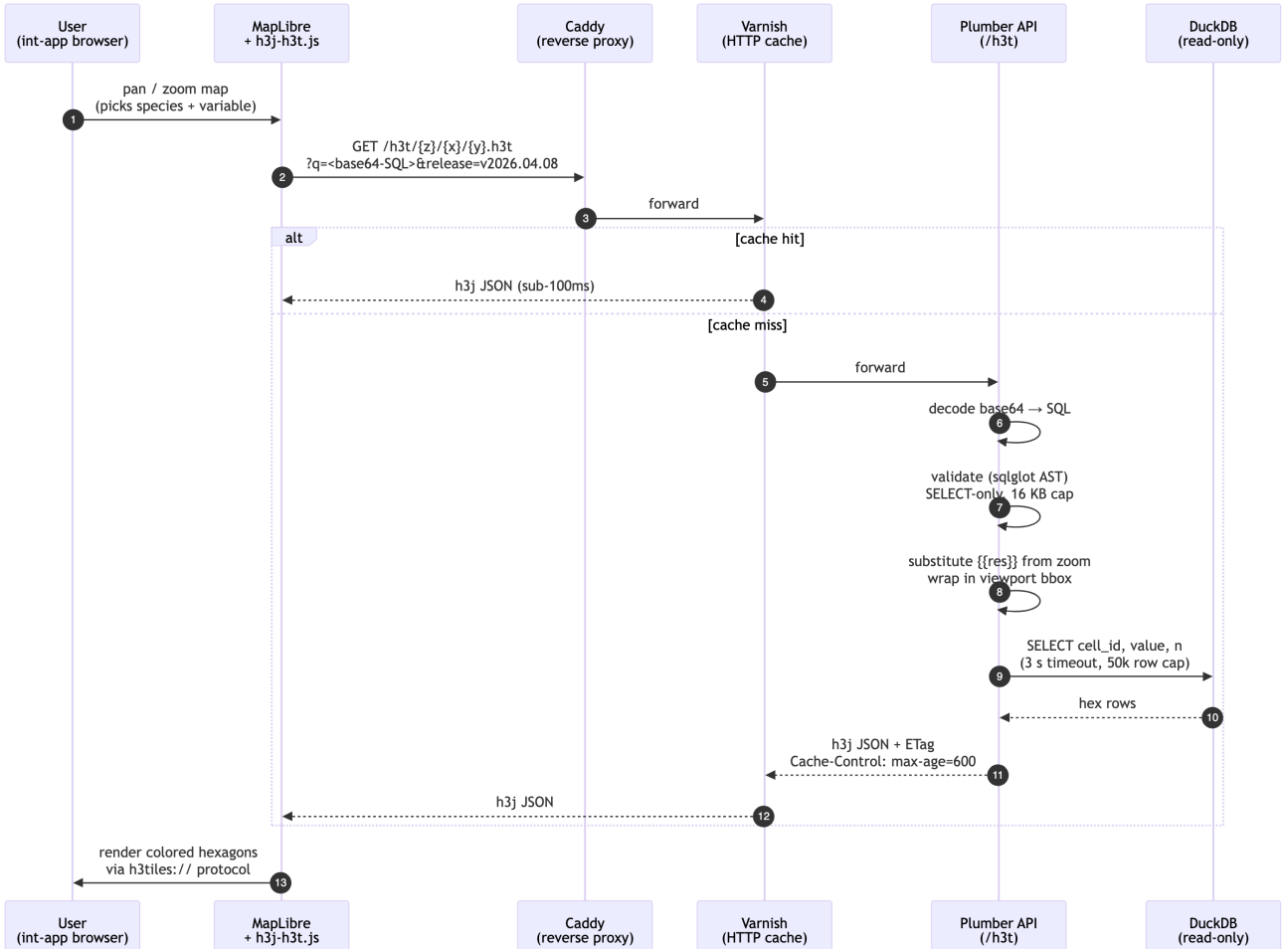


Figure 4.3: Tile request lifecycle. The browser asks for a tile by URL; Varnish caches identical URLs; on a miss, the Plumber API decodes and validates the SQL, runs it against a read-only DuckDB, and returns h3j-formatted hexagons. Validation + read-only DB + timeouts mean the public endpoint can accept arbitrary SQL safely.

4.4.3 Cache invalidation by release

Caches are easy to fill but notoriously hard to clear. H3T sidesteps the hard part entirely with a single trick: the `release` query parameter (Figure 4.4).

When CalCOFI publishes a new database release (e.g., `v2026.04.08`, see Database), a symlink at the API server is swapped to point at the new file. The next time the int-app loads, it asks `/h3t/meta` for the current release version, gets back `v2026.04.08`, and passes that into every subsequent tile URL. Because the URL is now different, Varnish **automatically cache-misses** — and the new tiles are generated against the fresh data on first request. Old tiles linger harmlessly in cache until evicted by LRU; they'd still serve correct data for their old release if anyone asked.

No purge command, no cache key surgery. Just: change one parameter, get a clean cache layer for free.

4.5 A worked example: sardine + temperature

To make this concrete, here is the full life of one click in the int-app:

1. **User picks** *Pacific sardine* (`Sardinops sagax`) and *temperature* in the sidebar, with date range 1949–2024 and all four quarters.
2. **`build_sp_sql()`** assembles a SQL template:

```
SELECT hex_h3res{{res}} AS cell_id,
       AVG(std_tally)   AS value,
       COUNT(*)         AS n
  FROM bio_obs
 WHERE scientific_name = 'Sardinops sagax'
   AND quarter IN (1, 2, 3, 4)
   AND time_start BETWEEN '1949-01-01' AND '2024-12-31'
 GROUP BY 1
```

3. **`h3t_b64()`** base64-encodes the SQL → `q=U0VMRUNUIGhleF9oM3Jlc3t7cmVzfX0gQVMgY2VsbF9pZCwKICAgIC`
4. **`fetch_h3t_stats()`** calls `/h3t/stats?q=…&release=v2026.04.08` once to get `{min, max, p02, p98, n}`. The 2nd–98th percentiles drive a robust color scale that ignores outliers.
5. **MapLibre** requests tiles as the user pans:

   `h3tiles://h3t.calcofi.io/h3t/4/3/6.h3t?q=U0VMRUNU…&release=v2026.04.08`

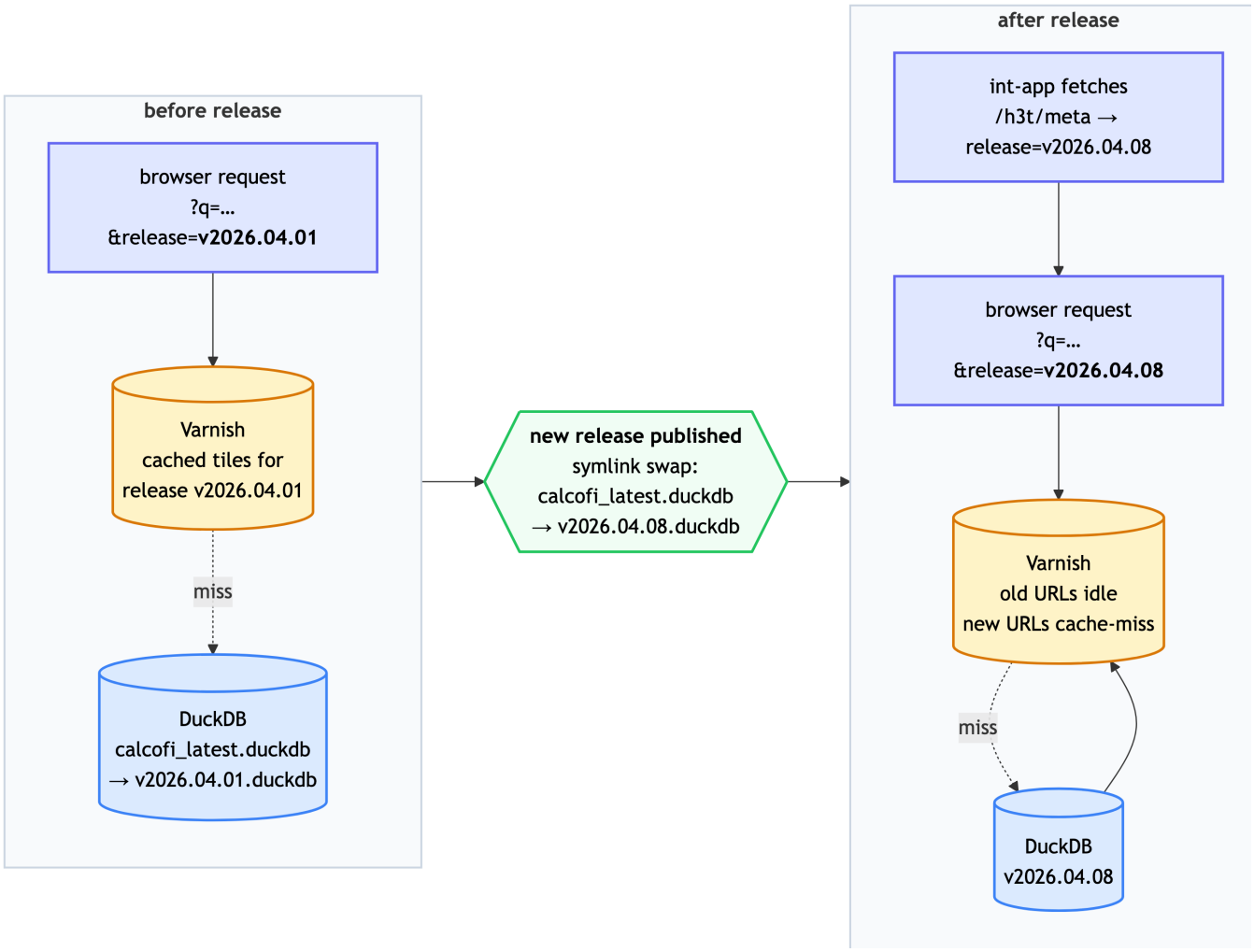


Figure 4.4: Cache invalidation by release. The `release` query parameter is part of every tile URL. When a new database release is published, the int-app picks up the new version from `/h3t/meta`, all subsequent tile URLs change, and Varnish refills naturally — no manual purge.

6. **Plumber** decodes, substitutes `{res}` (zoom 4 → res 3), wraps in a bounding-box filter for tile (4, 3, 6), runs the query, and returns:


```
{
  "cells": [
    { "h3id": "83390cfffffffff", "value": 12.37, "n": 42 },
    { "h3id": "83390dfffffffff", "value": 14.02, "n": 38 },
    { "h3id": "83390efffffffff", "value": 11.85, "n": 51 }
  ]
}
```


7. **MapLibre** paints each hex by interpolating the stats palette between p02 and p98. The user sees a smooth choropleth update as new tiles arrive.

The first time this query runs anywhere, it takes ~200–800 ms (Varnish miss, DuckDB execution). Every subsequent user requesting "sardines, all quarters, 1949–2024" hits Varnish and renders in under 100 ms.

4.6 How the data gets there

The H3T service is the *last* link in the CalCOFI data pipeline, not the first:

- Source data (Google Drive, NCEI, ERDDAP, SWFSC) is ingested through Quarto notebooks in `CalCOFI/workflows` into a working DuckDB database.
- A `release_database.qmd` step validates, freezes, and publishes a versioned DuckDB file to the API server.
- The H3T API mounts that file read-only and serves tiles from it.

For the schema details, see Database. For the publish path, see Portals.

4.7 Repositories and code

Repository	Role
CalCOFI/api-h3t	Plumber API, sqlglot validator, DuckDB driver, deployment configs
CalCOFI/int-app	Shiny consumer; SQL builders and map widgets in `app/functions_h3t.R`
bbest/mapgl @ feat/add-h3t-source	`add_h3t_source()` extension to the mapgl R package

Repository	Role
bbest/h3j-h3t @ fix/maplibre-v3-compat	fix h3t protocol for MapLibre v3+/v4 and multiple sources on one page in INSPIDE/h3j-h3t

For the full technical contract — every endpoint, every validation rule, every deployment knob — see the `api-h3t` README and `deploy.md`.

5 API

The raw interface to the Application Programming Interface (API) is available at:

- api.calcofi.io

Here we will provide more guidance on how to use the API functions with documented input arguments, output results and examples of use.

5.1 `/variables`: get list of variables for timeseries

Get list of variables for use in `/timeseries`

5.2 `/species_groups`: get species groups for larvae

Not yet working. Get list of species groups for use with variables `larvae_counts.count` in `/timeseries`

5.3 `/timeseries`: get time series data

5.4 `/cruises`: get list of cruises

Get list of cruises with summary stats as CSV table for time (`date_beg`)

5.5 `/raster`: get raster map of variable

Get raster of variable

5.6 `/cruise_lines`: get station lines from cruises

Get station lines from cruises (with more than one cast)

5.7 `/cruise_line_profile`

Get profile at depths for given variable of casts along line of stations

6 Database

6.1 Database naming conventions

> There are only two hard things in Computer Science: cache invalidation and naming things. – Phil Karlton (Netscape architect)

We're circling the wagons to come up with the best conventions for naming. Here are some ideas:

- Learn SQL: Naming Conventions
- Best Practices for Database Naming Conventions - Drygast.NET

6.1.1 Name tables

- Table names are **singular** and use all lower case.
  - Example: `cruise`, `site`, `species`, `lookup` (not `cruises`, `sites`, `lookups`)

6.1.2 Name columns

- To name columns, use **snake-case** (i.e., lower-case with underscores) so as to prevent the need to quote SQL statements. (TIP: Use `janitor::clean_names()` to convert a table.)
- Unique **identifiers** are suffixed with:
  - `*_id` for unique integer keys;
  - `*_uuid` for universally unique identifiers as defined by RFC 4122 and stored in Postgres as UUID Type.
  - `*_key` for unique string keys;
  - `*_seq` for auto-incrementing sequence integer keys.
- Suffix with **units** where applicable (e.g., `*_m` for meters, `*_km` for kilometers, `degc` for degrees Celsius). See units vignette.

- Set geometry column to **`geom`** (used by PostGIS spatial extension). If the table has multiple geometry columns, use `geom` for the default geometry column and `geom_{type}` for additional geometry columns (e.g., `geom_point`, `geom_line`, `geom_polygon`).

6.1.3 Primary key conventions

Prefer natural keys (meaningful domain identifiers) over surrogate keys where stable:

- `cruise_key` = 'YYMMKK' (e.g., `2401NH` = January 2024, New Horizon)
- `ship_key` = 2-letter ship code (e.g., `NH`)
- `tow_type_key` = tow type code (e.g., `CB`)

Sequential integer keys should have explicit sort order documented for reproducibility:

- `site_id` sorted by `cruise_key, orderocc`
- `tow_id` sorted by `site_id, time_start`
- `net_id` sorted by `tow_id, side`
- `ichthyo_id` sorted by `net_id, species_id, life_stage, measurement_type, measurement_value`

Avoid UUIDs in output tables: Use `_source_uuid` in Working DuckLake for provenance tracking only (stripped in frozen releases).

6.2 Use Unicode for text

The default character encoding for Postgresql is unicode (`UTF8`), which allows for international characters, accents and special characters. Improper encoding can royally mess up basic text.

Logging into the server, we can see this with the following command:

```
docker exec -it postgis psql -l
```

```
                                  List of databases
        Name         | Owner | Encoding |  Collate   |   Ctype    | Access privileges
---------------------+-------+----------+------------+------------+-------------------
 gis                 | admin | UTF8     | en_US.utf8 | en_US.utf8 | =Tc/admin         +
                     |       |          |            |            | admin=CTc/admin   +
                     |       |          |            |            | ro_user=c/admin
 lter_core_metabase  | admin | UTF8     | en_US.utf8 | en_US.utf8 | =Tc/admin         +
                     |       |          |            |            | admin=CTc/admin   +
                     |       |          |            |            | rw_user=c/admin
 postgres            | admin | UTF8     | en_US.utf8 | en_US.utf8 |
```

```
 template0          | admin | UTF8     | en_US.utf8 | en_US.utf8 | =c/admin         +
                    |       |          |            |            | admin=CTc/admin
 template1          | admin | UTF8     | en_US.utf8 | en_US.utf8 | =c/admin         +
                    |       |          |            |            | admin=CTc/admin
 template_postgis   | admin | UTF8     | en_US.utf8 | en_US.utf8 |
(6 rows)
```

Use Unicode (`utf-8` in Python or `UTF8` in Postgresql) encoding for all database text values to support international characters and documentation (i.e., tabs, etc for markdown conversion).

- In **Python**, use `pandas` to read (`read_csv()`) and write (`to_csv()`) with UTF-8 encoding (i.e., `encoding='utf-8'`).:

```python
import pandas as pd
from sqlalchemy import create_engine
engine = create_engine('postgresql://user:password@localhost:5432/dbname')

# read from a csv file
df = pd.read_csv('file.csv', encoding='utf-8')

# write to PostgreSQL
df.to_sql('table_name', engine, if_exists='replace', index=False, method='multi', chunks

# read from PostgreSQL
df = pd.read_sql('SELECT * FROM table_name', engine, encoding='utf-8')

# write to a csv file with UTF-8 encoding
df.to_csv('file.csv', index=False, encoding='utf-8')
```

- In **R**, use `readr` to read (`read_csv()`) and write (`write_excel_csv()`) to force UTF-8 encoding.

```r
library(readr)
library(DBI)
library(RPostgres)

# connect to PostgreSQL
con <- dbConnect(RPostgres::Postgres(), dbname = "dbname", host = "localhost", port = 54

# read from a csv file
df <- read_csv('file.csv', locale = locale(encoding = 'UTF-8'))  # explicit
df <- read_csv('file.csv')                                       # implicit
```

```r
# write to PostgreSQL
dbWriteTable(con, 'table_name', df, overwrite = TRUE)

# read from PostgreSQL
df <- dbReadTable(con, 'table_name')

# write to a csv file with UTF-8 encoding
write_excel_csv(df, 'file.csv', locale = locale(encoding = 'UTF-8'))  # explicit
write_excel_csv(df, 'file.csv')                                       # implicit
```

6.3 Integrated database ingestion strategy

6.3.1 Overview

The CalCOFI database uses a two-schema strategy for development and production:

- **`dev` schema**: Development schema where new datasets, tables, fields, and relationships are ingested and QA/QC'd. This schema is recreated fresh with each ingestion run using the master ingestion script.
- **`prod` schema**: Production schema for stable, versioned data used by public APIs, apps, and data portals (OBIS, EDI, ERDDAP). Once `dev` is validated, it's copied to `prod` with a version number.

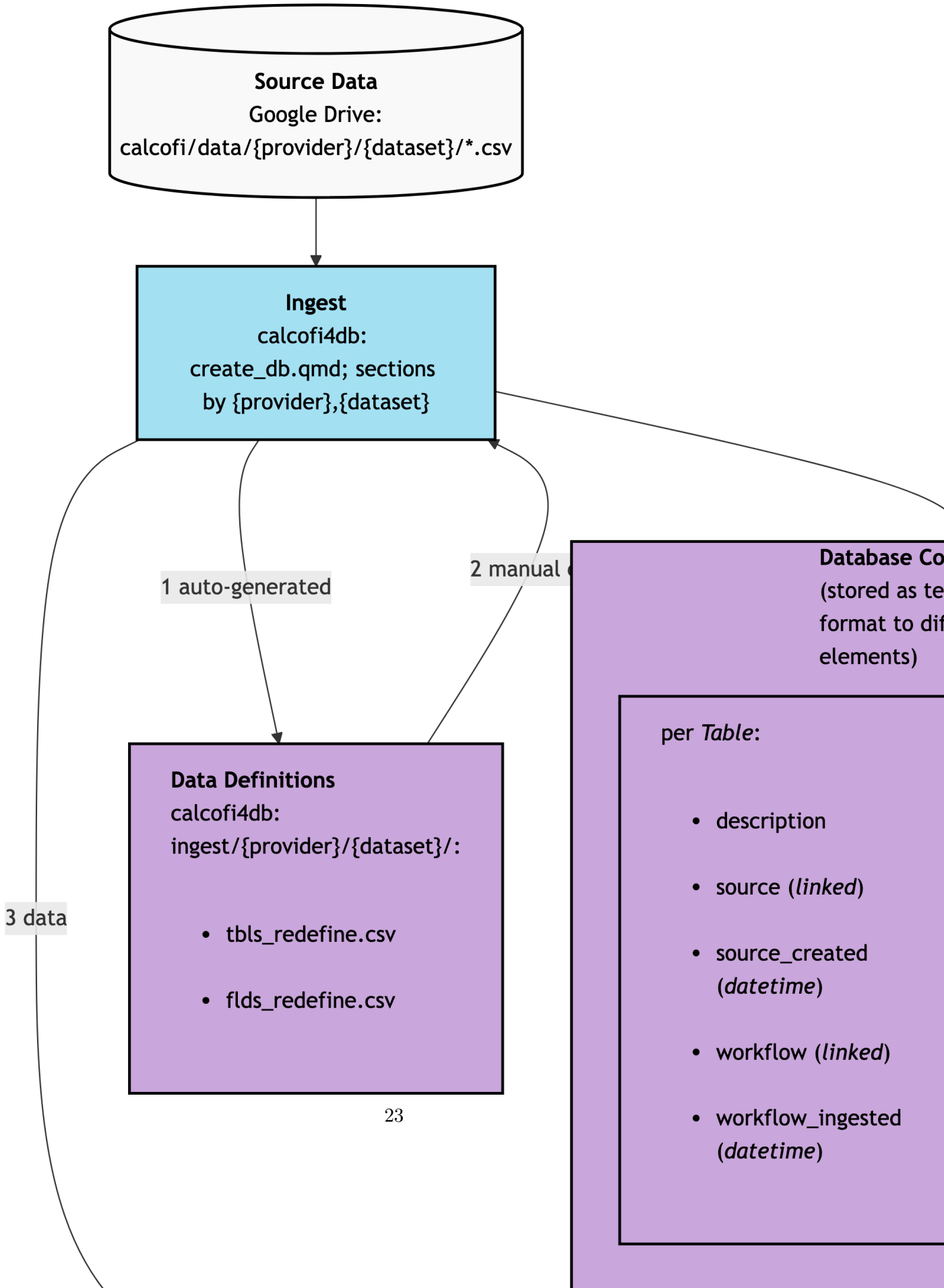
6.3.2 Master ingestion workflow

All datasets are ingested using a single master Quarto script `calcofi4db/inst/create_db.qmd` that:

1. **Drops and recreates** the `dev` schema (fresh start each run)
2. **Ingests multiple datasets** from Google Drive source files (CSV, potentially SHP/NC in future)
3. **Applies transformations** using redefinition files (`tbls_redefine.csv`, `flds_redefine.csv`)
4. **Creates relationships** (primary keys, foreign keys, indexes)
5. **Records schema version** with metadata in `schema_version` table

Each dataset section in the master script handles:

- Reading CSV files from Google Drive
- Transforming data according to redefinition rules
- Loading into database tables
- Adding table/field comments with metadata


Source Data
Google Drive:
calcofi/data/{provider}/{dataset}/*.csv
Ingest
calcofi4db:
create_db.qmd; sections
by {provider},{dataset}
1 auto-generated
2 manual
Database Co
(stored as te
format to dif
elements)
per *Table*:
description
source (*linked*)
source_created
(*datetime*)
workflow (*linked*)
workflow_ingested
(*datetime*)
Data Definitions
calcofi4db:
ingest/{provider}/{dataset}/:
tbls_redefine.csv
flds_redefine.csv
3 data

6.3.3 Using calcofi4db package

The calcofi4db package provides streamlined functions for dataset ingestion.

6.3.3.1 DuckLake Workflow (Recommended)

The preferred approach uses DuckDB with provenance tracking:

```
library(calcofi4db)

# connect to working DuckLake (downloads from GCS if needed)
con <- get_working_ducklake()

# read CSV files from local directory or GCS archive
# - syncs to GCS archive for immutable provenance tracking
# - loads redefinition metadata for column renaming/typing
d <- read_csv_files(
  provider     = "swfsc",
  dataset      = "ichthyo",
  dir_data     = "~/My Drive/projects/calcofi/data-public",
  metadata_dir = "metadata",
  sync_archive = TRUE)

# ingest dataset with automatic provenance tracking
# - transforms data using redefinition files
# - adds _source_file, _source_row, _source_uuid, _ingested_at columns
# - handles uuid column detection automatically
tbl_stats <- ingest_dataset(
  con  = con,
  d    = d,
  mode = "replace")

# save to GCS and close
save_working_ducklake(con)
close_duckdb(con)
```

The `ingest_dataset()` function handles:

- Calling `transform_data()` to apply table/field redefinitions
- Detecting UUID columns automatically for provenance tracking
- Calling `ingest_to_working()` for each table with proper source file tracking
- Returning ingestion statistics (rows input, rows after, timestamps)

6.3.3.2 PostgreSQL Workflow (Legacy)

For PostgreSQL ingestion (being gradually deprecated):

```
library(calcofi4db)
library(DBI)
library(RPostgres)

# connect to database
con <- dbConnect(
  Postgres(),
  dbname   = "gis",
  host     = "localhost",
  port     = 5432,
  user     = "admin",
  password = "postgres")

# read CSV files and metadata
d <- read_csv_files(
  provider = "swfsc",
  dataset  = "ichthyo")

# transform data according to redefinitions
transformed_data <- transform_data(d)

# ingest into dev schema
ingest_csv_to_db(
  con              = con,
  schema           = "dev",
  transformed_data = transformed_data,
  d_flds_rd        = d$d_flds_rd,
  d_gdata          = d$d_gdata,
  workflow_info    = d$workflow_info)

# record schema version
record_schema_version(
  con              = con,
  schema           = "dev",
  version          = "1.0.0",
  description      = "Initial ingestion of NOAA CalCOFI Database",
  script_permalink = "https://github.com/CalCOFI/calcofi4db/blob/main/inst/create_db.qmd")
```

6.3.4 Schema versioning

Each successful ingestion creates a new schema version recorded in the `schema_version` table with:

- **version**: Semantic version number (e.g., "1.0.0", "1.1.0")
- **description**: Changes introduced in this version
- **date_created**: Timestamp of ingestion
- **script_permalink**: GitHub permalink to the versioned ingestion script

Versions are also archived as SQL dumps in Google Drive for reproducibility.

6.3.5 Metadata and documentation

After ingestion, metadata is stored in PostgreSQL `COMMENT`s as JSON at the **table** level:

- **description**: General description and row uniqueness
- **source**: CSV file link to Google Drive
- **source_created**: Source file creation timestamp
- **workflow**: Link to rendered ingestion script
- **workflow_ingested**: Ingestion timestamp

And at the **field** level:

- **description**: Field description
- **units**: SI units where applicable

These comments are exposed via the API db_tables endpoint and rendered with calcofi4r::cc_db_catalog.

6.3.6 Publishing to portals

After `prod` schema is versioned, additional workflows publish data to Portals (ERDDAP, EDI, OBIS, NCEI) using ecological metadata language (EML) via the EML R package, pulling metadata directly from database comments.

6.3.7 OR Describe tables and columns directly

- Use the `COMMENT` clause to add descriptions to tables and columns, either through the GUI pgadmin.calcofi.io (by right-clicking on the table or column and selecting `Properties`) or with SQL. For example:

```
COMMENT ON TABLE public.aoi_fed_sanctuaries IS 'areas of interest (`aoi`) polygons for f
```

- Note the use of **markdown** for including links and formatting (e.g., bold, code, italics), such that the above SQL will render like so:

  > areas of interest (`aoi`) polygons for federal **National Marine Sanctuaries**; loaded by *workflow* load_sanctuaries

- It is especially helpful to link to any ***workflows*** that are responsible for the ingesting or updating of the input data.

6.3.8 Display tables and columns with metadata

- These descriptions can be viewed in the CalCOFI **API** api.calcofi.io as CSV tables (see code in calcofi/api: `plumber.R`):
  - api.calcofi.io/`db_tables`
    fields:
    - `schema`: (only "public" so far)
    - `table_type`: "table", "view", or "materialized view" (none yet)
    - `table`: name of table
    - `table_description`: description of table (possibly in markdown)
  - api.calcofi.io/`db_columns`
    fields:
    - `schema`: (only "public" so far)
    - `table_type`: "table", "view", or "materialized view" (none yet)
    - `table`: name of table
    - `column`: name of column
    - `column_type`: data type of column
    - `column_description`: description of column (possibly in markdown)
- Fetch and display these descriptions into an interactive table with `calcofi4r::cc_db_catalog()`.

6.4 Relationships between tables

- See calcofi/workflows: **clean_db**
- `TODO:` add calcofi/apps: db to show latest tables, columns and relationsips

6.5 Spatial Tips

- Use `ST_Subdivide()` when running spatial joins on large polygons.

7 Portals

7.1 Overview

CalCOFI data is available through various portals, each serving different purposes and user needs. This document outlines the main access points and their characteristics.

7.2 Data Flow

While it would be ideal for CalCOFI data to be available through a single portal, each portal has its strengths and limitations. The following diagram illustrates one possible realization of data flow between CalCOFI data and the portals: from raw data to the integrated database to portals and meta-portals.

In practice, CalCOFI is a partnership with various contributing members, so the authoritative dataset might flow differently, such as from EDI to the database to the other portals. The other portals, such as OBIS or ERDDAP, serve different audiences or purposes. The meta-portals like ODIS and Data.gov then index these portals to provide broader discovery of CalCOFI datasets.

7.3 Portals

While some portals serve as data repositories, others provide advanced data access and visualization tools. The following sections describe the main portals where CalCOFI data is available and their key features.

7.3.1 EDI

Environmental Data Initiative

- Complete dataset archives using DataOne software and EML metadata
- DOIs issued for all datasets ensuring citability
- Full archive allowing for any data file types
- Basic spatial and temporal filtering through web interface

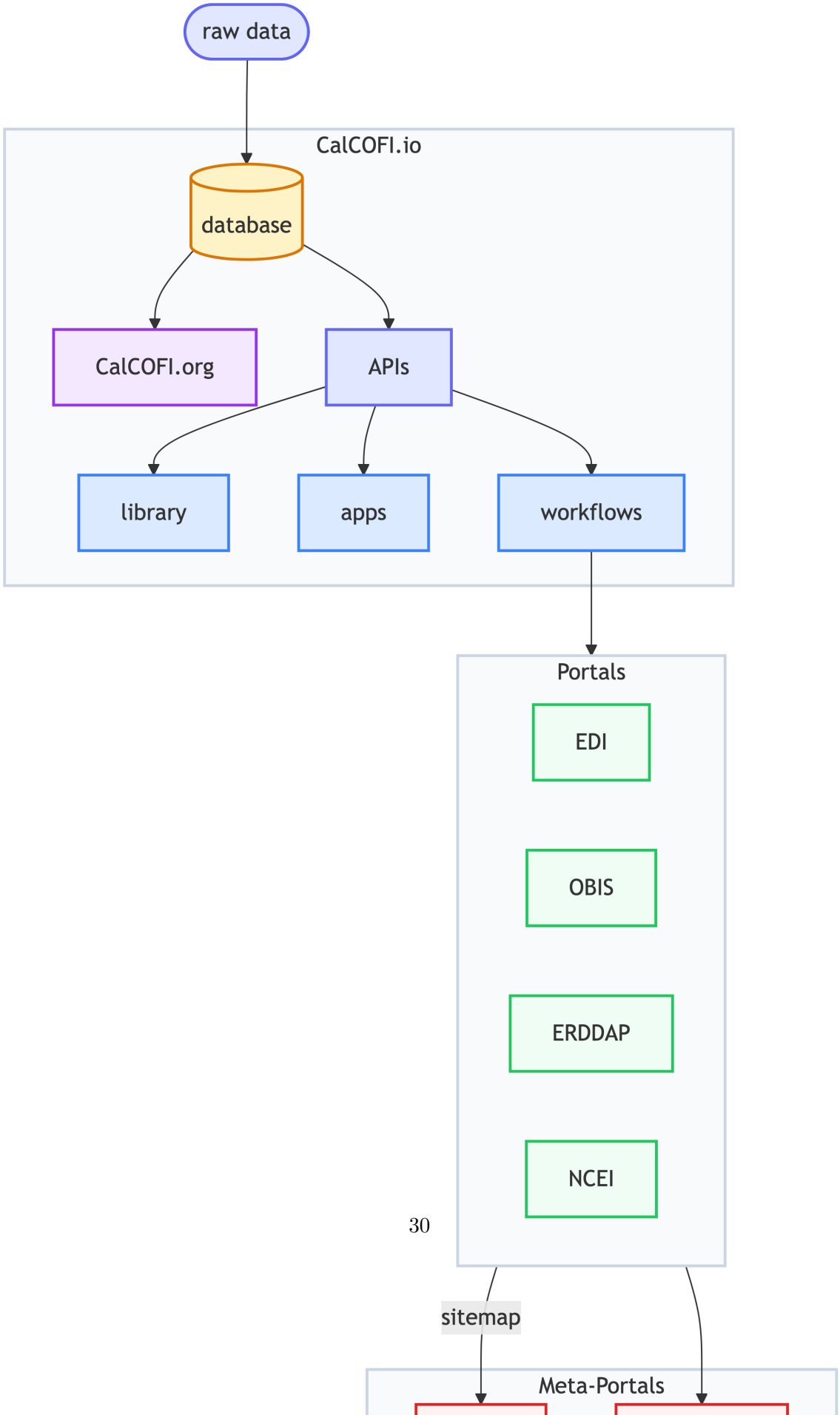

raw data
CalCOFI.io
database
CalCOFI.org
APIs
library
apps
workflows
Portals
EDI
OBIS
ERDDAP
NCEI
sitemap
Meta-Portals

Table 7.1: Portal Capabilities.

	Full Archive	Versioning	DOI Issued	Query by xyt	Query by taxa	Multiple formats
EDI						
NCEI						
OBIS						
ERDDAP						

Capability Legend: = full, = partial, = none

- Download in original formats with metadata
- Access through DataOne API
- Links:
  - EDIrepository.org
  - CalCOFI datasets: EDI query "CalCOFI"

7.3.2 NCEI

National Centers for Environmental Information

- Long-term archival of oceanographic data
- DOIs issued for dataset submissions
- Standardized metadata using ISO 19115-2
- Basic search interface with geographic and temporal filtering
- Data preserved in original submission formats
- Access through NCEI API services
- Links:
  - NCEI Ocean Archive
  - CalCOFI datasets: NCEI search "CalCOFI"

7.3.3 OBIS

Ocean Biodiversity Information System

- Specialized in marine biodiversity data
- Standardized using DarwinCore fields
- Extended measurements supported via extendedMeasurementOrFact
- Powerful filtering by space, time, and taxonomic parameters
- Multiple download formats (CSV, JSON, Darwin Core Archive)
- Full REST API access

- Links:
  - OBIS.org
  - CalCOFI datasets: obis.org/dataset + "calcofi" Keyword

7.3.4 ERDDAP

Environmental Research Division Data Access Program

- Tabular and gridded data server
- Advanced subsetting by space, time, and parameters
- Multiple output formats (CSV, JSON, NetCDF, etc.)
- RESTful API with direct data access
- Built-in data visualization tools
- No persistent identifiers but stable URLs
- Links:
  - ERDDAP
  - CalCOFI datasets:
    * ERDDAP, OceanView - CalCOFI seabirds
    * ERDDAP, CoastWatch - CalCOFI oceanographic

7.4 Metadata

The Ecological Metadata Language (EML) (and using R package EML in workflows) serves as a key standard for describing ecological and environmental data. For CalCOFI, EML metadata files are generated alongside data files, providing structured documentation that enables interoperability across different data portals. This metadata-driven approach allows automated ingestion into various data systems while maintaining data integrity and provenance.

The EML specification provides detailed structure for describing datasets, including:

- Dataset identification and citation
- Geographic and temporal coverage
- Variable definitions and units
- Methods and protocols
- Quality control procedures
- Access and usage rights

This standardized metadata enables automated data transformation and ingestion into various portal systems while preserving the original data context and quality information.

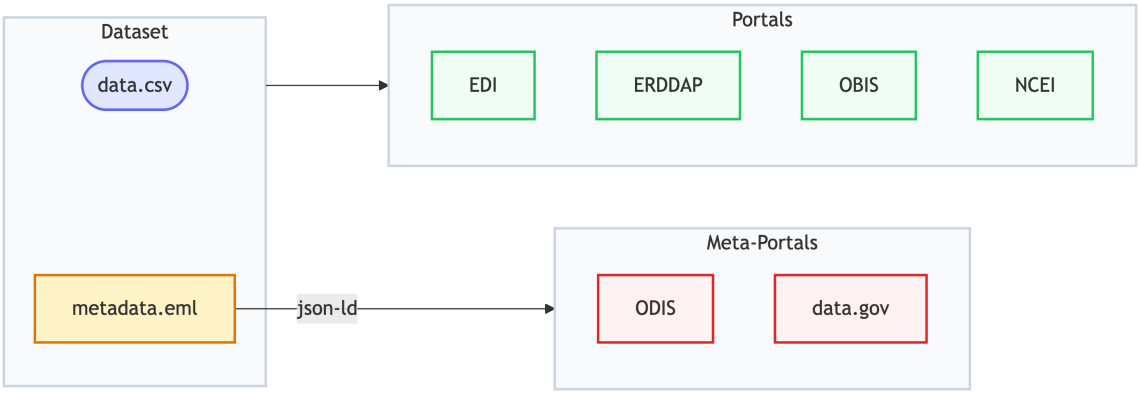


Figure 7.2: Metadata in the form of ecological metadata language (EML) is used to describe the dataset in a consistent manner that can be ingested by the portals.

7.5 Meta-Portals

7.5.1 Google Dataset Search

The JSON-LD metadata in the Portal dataset web pages get indexed by Google Dataset Search through schema.org metadata. This ensures that CalCOFI data is discoverable through Google search and other search engines.

7.5.2 ODIS

Ocean Data Information System

ODIS uses the same technology as Google Dataset Search (schema.org, JSON-LD), but focuses on ocean data. CalCOFI curates a sitemap of authoritative datasets to server to ODIS.org

This federated approach ensures that CalCOFI data remains:

- Discoverable through multiple channels
- Properly cited and attributed
- Integrated with global ocean data systems

7.6 CalCOFI.io Tools

CalCOFI is also developing an integrated database and tools that enable efficient data access and analysis:

7.6.1 APIs

- RESTful endpoints for programmatic access
- Filtering by space, time, and taxonomic parameters
- Relationship queries across tables
- Links:
  - api.calcofi.io
  - tile.calcofi.io

7.6.2 Library

- Direct data access from R
- Built-in analysis functions
- Integration with tidyverse ecosystem
- Link:
  - calcofi.io/calcofi4r

7.6.3 Apps

- Interactive data exploration with Shiny applications
- User-friendly interfaces
- Subset and download data
- Link:
  - calcofi.io, App button

8 Status

8.1 2025-12-01

Over the past several months, CalCOFI's software and data systems have advanced significantly toward a unified, reliable, and user-friendly platform for exploring and publishing CalCOFI data. Work has focused on four main areas: the integrated data app, the underlying database and workflows, pushing to OBIS, and organizing of CTD data.

8.1.1 A More Capable, User-Friendly Integrated App

The **CalCOFI integrated application (`int-app`)** has evolved into a much richer and more intuitive tool for scientists and partners:

- **Taxonomy-aware exploration**
  The app now understands species and their taxonomic relationships. Users can browse taxa hierarchies, see taxonomic ranks, and work with improved species metadata tied to authoritative sources (e.g., WoRMS, ITIS). This makes it easier to find and compare species and groups of species consistently.
- **Better visual experience and theming**
  A new **dark/light theme toggle** has been implemented and refined so that maps, time series, and other plots remain readable and visually consistent. Navigation has been reorganized, with a clearer About page, a guided "tour" of the app, and more intuitive icons and labels, making the app easier to learn and use.
- **Stronger spatial and temporal tools**
  Spatial maps now rely on efficient hexagon grids calculated in the database, improving performance and scalability. Default settings for time and depth matching have been tuned to yield better joins between environmental and biological data out of the box.

Overall, the app is moving from a prototype to a **polished, guided interface** that better supports exploratory analysis and communication.

8.1.2 A Stable, Well-Documented Database Foundation

The **CalCOFI database package (`calcofi4db`)** has been formalized and versioned, providing a solid foundation for all downstream tools:

- **Two stable releases** (versions 1.0 and 1.1) have established a reliable baseline for the database, including bottle-level data.
- The project now follows a **clear strategy for separate development and production databases**, reducing risk when making changes and improving reproducibility.
- Data ingestion from NOAA and other sources has been **hardened**, with several rounds of fixes to handle edge cases and ensure that raw files are consistently and correctly translated into the database.

In addition, the package's **online documentation site** has been refreshed so that developers and analysts have up-to-date guidance on how data flow into and through the database.

8.1.3 Unified R Tools and Documentation Around DuckDB

Across the toolchain, CalCOFI has standardized on **DuckDB** as the core data engine:

- The **R package (`calcofi4r`)** now encapsulates key logic originally developed inside the integrated app, so the same high-quality data access and processing is available in scripts, reports, and analyses—not just in the web interface.
- Both the app and R package can connect to **local or remote DuckDB databases**, improving performance and enabling offline or near-offline workflows.
- Documentation in the **`docs` repository** has been updated to describe the full data creation process (from raw data to ready-to-use databases) and to explain the new development/production database strategy. Status documents and helper scripts provide clearer visibility into project progress.

This brings CalCOFI closer to a **coherent, documented platform** where analysts can move seamlessly between app-based exploration and scripted analysis.

8.1.4 Standards-Compliant Publication of Biological Data

The `workflows` repository has seen major progress in turning CalCOFI's biological datasets into **publication-ready products**:

- New workflows now **publish larval data to OBIS**, the global biodiversity information system, with repeatable recipes that combine biological observations with CTD (oceanographic) data.
- The underlying data model for **events and occurrences** has been strengthened to align with international standards (e.g., Darwin Core), including:
  - Clear hierarchies of sampling events,
  - Better handling of life-stage and size information (e.g., egg and larval stages),
  - Automated generation of metadata files required for data archives.
- Additional integrity checks and foreign key relationships help ensure that data are correct and consistent before publication.

These advances substantially improve CalCOFI's ability to **share high-quality, well-structured biodiversity data** with the broader scientific community.

8.1.5 Improved Public Access and Infrastructure

Finally, several changes improve how external users find and access CalCOFI tools:

- The **public website (`CalCOFI.github.io`)** now highlights key applications, including the integrated app and a pollutants-focused app, making them easier to discover.
- Server configuration has been updated so that **Shiny apps are served from a new dedicated domain `app.calcofi.io`** (and still `shiny.calcofi.io`), clarifying the entry point for interactive tools and simplifying operations.

8.1.6 Overall Impact

Together, these developments move CalCOFI toward a **modern, integrated data platform**:

- Scientists and partners gain a more powerful, user-friendly app and R toolkit for exploring CalCOFI data.

- The underlying database and workflows are more robust, testable, and clearly documented.
- CalCOFI's biological data are better positioned for **global visibility and reuse** through standards-compliant publication channels like OBIS.
- Public-facing web presence and infrastructure are cleaner and more aligned, making it easier for stakeholders to find and use CalCOFI resources.

8.2 2025-07-01

This report summarizes the key development activities, major accomplishments, and ongoing work for the first 6 monhts of 2025 across the CalCOFI GitHub repositories: **api, apps, calcofi4db, calcofi4r, docs, server, workflows**. The findings are based on issues and commits from January–July 2025.

8.2.1 API Enhancements

8.2.1.1 New Features & Data Integration

- **Expanded API Options**
  - Added ability to include bottle data and use relaxed criteria for net-to-cast matching (commit).
  - Supported upcast/downcast data downloads (commit).
  - Added Zooplankton biomass and improved ichthyodata output (commit).
- **Performance & Maintenance**
  - Implemented docker compose restart for Plumber API service (commit).
- **Ongoing Work**
  - Migration of database contouring functions to API/app level for improved caching and rendering efficiency.
  - Development of a robust, user-friendly API for seamless DB integration (issue).

8.2.2 Apps Development

8.2.2.1 Visualization & User Interface

- **Continuous Improvements**
  - Multiple commits indicate ongoing enhancement, likely focused on UI, data visualization, and integration with the API (see recent commit log).
  - Close coordination between API and Apps for improved workflows and data access.

8.2.3 calcofi4db: R Package & Data Management

8.2.3.1 R Package Initialization & Data Ingestion

- **New R Package: calcofi4db**
  - Initial commit and setup (commit), including functions for ingesting CSV datasets and metadata.
  - Refined change detection logic for source CSV files, improving tracking of table/field changes (commit).
  - Enhanced documentation and site via pkgdown.
  - Improved function naming and structure for ingestion (commits, commit).

8.2.4 calcofi4r: Spatial & Ecological Data Tools

8.2.4.1 Data Layers, Analysis, and Bug Fixes

- **Spatial Management Layers**
  - Ongoing integration of BOEM Wind Planning Areas, Marine Protected Areas, and SCCWRP management regions (issue, issue).
- **Analysis Functions**
  - Improved packages for ecological and spatial analysis, including new dependencies (commit).
- **User Feedback**
  - Addressing user-reported bugs such as deprecated function calls (issue).

8.2.5 Documentation (docs)

8.2.5.1 Infrastructure & Environment

- **Documentation Site Updates**
  - Added documentation for new packages and ingestion workflows (commit).
  - Improved environment handling for rendering with Quarto and Chromium (multiple commits Jan-Mar 2025).
  - Updated diagrams and edge labels for database documentation.

8.2.6 Server

8.2.6.1 Backend Infrastructure

- **Backend Maintenance**
  - Numerous commits for improving server reliability, configuration, and deployment.
  - Indicates active backend support for API and Apps.

8.2.7 Workflows

8.2.7.1 Data Pipeline, Integration, and Registration

- **Workflow Automation**
  - Multiple commits show ongoing development of data ingestion, harmonization, and visualization workflows (commit, commit).
- **ODIS Registration**
  - Registering datasets with ODIS (using JSON-LD) for broader interoperability (issue).
- **Integration with External Data**
  - Ongoing work to load and harmonize diverse ecological datasets (bottle data, larvae, zooplankton, etc.).
- **Spatial Data Management**

- Continued development of AOI (areas of interest), spatial buffer creation, and integration of management regions.

8.2.8 Key Themes & Impact

8.2.8.1 Integration & Interoperability

- Strong focus on connecting API, Apps, R packages, and backend infrastructure for seamless data access and visualization.
- Enhanced interoperability through ODIS registration and harmonized workflows.

8.2.8.2 Data Accessibility & Usability

- Improvements to API and Apps make ecological data more accessible to researchers and managers.
- Expanded support for spatial management areas and ecological datasets.

8.2.8.3 Infrastructure & Sustainability

- Investments in documentation, backend reliability, and workflow automation contribute to long-term sustainability and reproducibility.

8.2.9 For More Details

- Some results may be incomplete due to API limits.
- To view all commits/issues for 2025, visit each repository's GitHub UI and filter by year.

9 References

9.1 R packages

- API: plumber (Schloerke and Allen 2024)
- docs: Quarto (Allaire and Dervieux 2024)
- apps: Shiny (Chang et al. 2024)

Allaire, JJ, and Christophe Dervieux. 2024. *Quarto: R Interface to Quarto Markdown Publishing System*. https://github.com/quarto-dev/quarto-r.

Chang, Winston, Joe Cheng, JJ Allaire, et al. 2024. *Shiny: Web Application Framework for r*. https://shiny.posit.co/.

Schloerke, Barret, and Jeff Allen. 2024. *Plumber: An API Generator for r*. https://www.rplumber.io.