Welcome to Production Systems!

ProdSys is in charge of the following codebases and associated processes:

- covidcast-indicators - data pipelines
- delphi-epidata - database, data ingestion, pre-covid pipelines, API server, and public API documentation
- forecast-eval - public dashboard for forecast evaluation
- www-covidcast - Web frontend for data visualizations on the Delphi website
- www-main - General layout, blog posts, and other static content for the Delphi website
- fault-record - GUI for tracking and browsing data faults (still internal)
- (deprecated) covid-19 - prototypes from early April 2020
- (deprecated) covidcast - API clients and public statistical tools (shared with Tooling)

We have established team practices for tracking work, submitting changes, and code review.

Our primary Slack channel is #engineering. We do daily asynchronous “standups” in the #engineering-checkin Slack channel. Check-ins are optional and mostly to help you remember, tag people you're blocked on, or ask for help. Other Slack channels you may be interested in are in their own section below.

We meet weekly for round-table updates and a walkthrough talk. Notes and links to walkthrough recordings

We are slowly developing a pipeline operations and triage manual (campus VPN and Andrew login required). Additional procedure documentation is in the delphi-prod-procedures folder on sharedrive.

We have a Miro board with system diagrams and brief explanations of our tech stack (Log in with CMU SSO).

Other sources of documentation:

- the covidcast-indicators README - procedure for creating a new indicator, getting it approved for publishing in the API, and loading it into the automation system
- the README of each indicator module (e.g. JHU) - procedure to install, run, lint, and test the indicator
- the indicator module template (Python; R) including the code review guide - outlines common structures used by all indicators. If it's not in the template, indicators may differ in their approach.
- the delphi_utils module - centralized implementations of common tasks, including reading from params, smoothing, geographical aggregation, and construction of diff-based data issues.

The rest of this document describes processes not included elsewhere.

How a dataset becomes an indicator

This is the general extract-transform-load procedure used by all COVIDcast indicators.

Each indicator pipeline is managed through Cronicle, a job scheduler. Cronicle automatically tracks success/failure and resource usage statistics and handles semaphores so that database operations don't run into each other.

First, an indicator module in the covidcast-indicators codebase does the following:

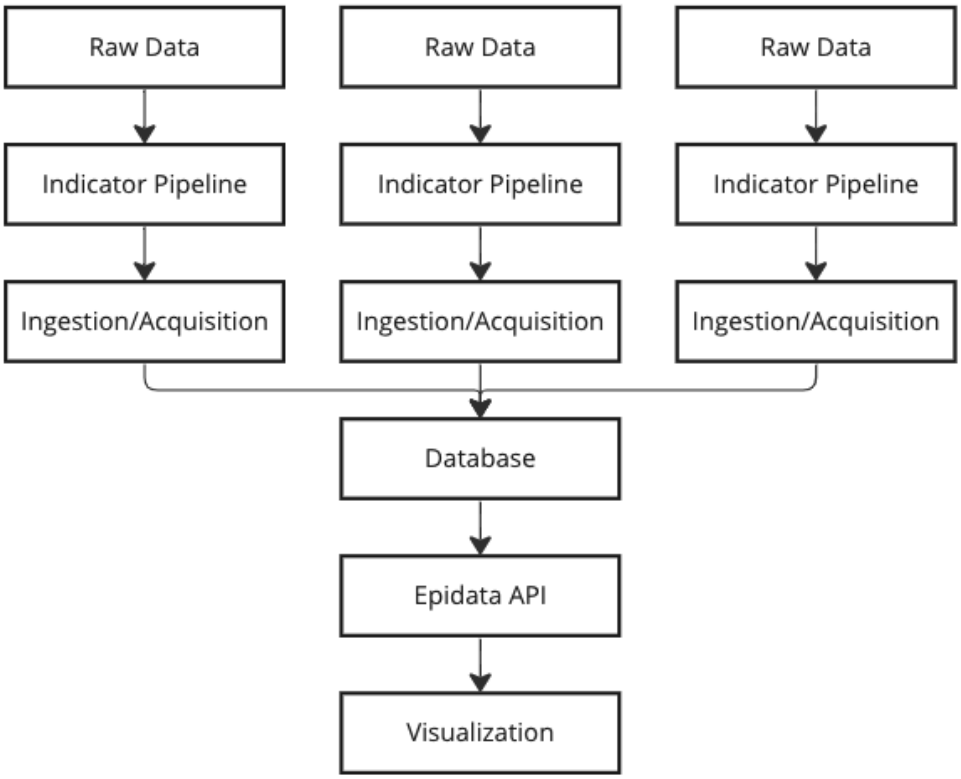
1. Download data from the source. This could be via an API query, scraping a website, an SFTP or S3 dropbox, or an email attachment.
2. Process the source data to extract one or more time series signals. A signal includes a value, standard error, and sample size for each base-level region on each unit of time (a day, or an epidemiological week "epiweek").
3. Aggregate each signal to higher geographic levels. For example, we generate data at the state level by combining data at the county level.
4. Output each signal into a set of CSV files with a fixed format.
5. Run a set of checks on the output to make sure it will be accepted by the acquisition code, and hunt for common signs of buggy code or bad source data. Eventually major failures here will be used to block future steps, but for now we just log any problems we find. This component is called Validator.
6. Compare today's output with a cached version of what's currently in the API to convert the dense output to a diff. This reduces the size of each update, often dramatically. Not yet used by doctor-visits or hospital-admissions. This component is called ArchiveDiffer.
7. Deliver the CSV output files to the `receiving/` directory on the acquisition server (AKA primary AKA midas).

Next, the acquisition:covidcast component of the delphi-epidata codebase does the following:

8. Look in the receiving/ folder to see if any new data files are available for this indicator. If there are, then:
    a. Import the new data into the epimetric_full table of the epidata.covid database, filling in the columns as follows:
        i. source: parsed from the command line argument
        ii. signal: parsed from the filename
        iii. time_type: parsed from the filename
        iv. time_value: parsed from the filename
        v. geo_type: parsed from the filename
        vi. geo_value: parsed from each row of the csv file
        vii. value: parsed from each row of the csv file

viii. se: parsed from each row of the csv file
ix. sample_size: parsed from each row of the csv file
x. issue: whatever now is in time_type units
xi. lag: the difference in time_type units from now to time_value
xii. value_updated_timestamp: now

b. Update the epimetric_latest table with any new keys or new versions of existing keys.



As often as we can (May 2023: every 3 hours), we generate the following analytics:

9. Update the metadata cache:
    a. For every (source, signal, time_type, geo_type) tuple, compute the minimum and maximum time_value, value, issue, and lag, the number of regions with data, and the total mean and stdev, and cache them in a JSON string. This information is used by the map to determine appropriate axis and colorscale limits, and is used by researchers to determine how far back in time they can go when using our API to generate training data for their models.

Each night, we generate the following analytics:

10. Check that all indicators are being updated in a timely fashion. If not, call for help.

    a. This is done using a utility called "Sir Complains-a-lot" because it used to complain an awful lot. It complains less these days.
11. Compute coverage and latency for a representative signal from each timeseries family.

PRDs: Project Requirements Documents

We've adapted an artifact and accompanying practice from Google called a PRD that helps crystallize ideas for new projects or thorny design decisions.

- PRD Template - to make a new PRD, open this link, then choose "Make a copy" from the File menu
- Production Systems -specific PRDs
- All-Delphi PRDs

It would be tedious to write up a PRD for every little thing, so we don't do that. What kinds of tasks are PRDs useful for?

- **things with potentially many moving parts** (so that it's easier to see how they interact)
- **things with potentially many design decisions** (so that's there's a common record; so that it's easier to see how their consequences interact)
- **things with potentially many stakeholders** (so that there's a common record; so that it's easier to see how their constraints interact)
- **things with a potentially long implementation time** (so that there's a common record)
- **things that are hard to explain** without a bunch of background information / context (so that there's a common record)
- **things you plan to delegate** to another team, especially to junior staff (so that there's a common record)

Slack Channels

We use Slack a lot!

Channels for people

- For everybody:
    - #engineering - announcements, calls for volunteers or extra eyes, gauging interest in new initiatives, cool tools
    - #engineering-checkin - async standup meeting & affordance for those of us who forget everything that isn't written down
    - #outages - ask and tell about active outages (debugging threads often happen here)
    - #quick-questions - ask literally anything; we're all learning together 💛

- For specific projects or subgroups:
    - #epidata-backend - mostly delphi-epidata development
    - #fault-record - devs and users of the Fault Recordkeeping System
    - #documentation-proj - folks working on the new signals documentation browser
    - #infrastructure - ask and tell about servers & services, maintenance, devops
    - #pipeline-ops - folks keeping the data pipelines running and healthy

Channels for bots

- #engineering-utils - mostly github PRs
- #system-monitoring - exceptions and success/failure notices from data pipelines (debugging threads often happen here)
- #infrastructure-alerts - mostly newrelic
- #cronicle-job-monitoring - 100% cronicle success/failure notices
- #db-system-monitoring - mostly pmm
- #jenkins-ci - 100% jenkins builds

Terminology

See also: Delphi Glossary

- **indicator** - a loose term variously referring to a single data stream, or to a set of data streams sharing some characteristics. For example, "the masks indicator" typically refers to the smothed_wwearing_mask_7d signal currently shown in the CTIS dashboard; "the JHU indicator" typically refers to all signals generated using the cases and deaths data published by Johns Hopkins University's Center for Systems Science and Engineering
- **source** - for most indicators, this is the name of the organization that provided the source dataset. This includes jhu-csse, safegraph, and quidel. Exceptions:
    - The source data for doctor-visits and hospital-admissions is provided to us by a company that wishes to remain unmentioned.
    - The source data for indicator-combination comes from our own API.
- **signal** - In technical conversations, this means a single metric under a single configuration. For example, in doctor-visits, smoothed_cli and smoothed_adj_cli both measure COVID-like illness, but one of them is transformed to remove weekday variations. They are separate signals. More casually, we sometimes talk about e.g. "the community CLI signal" from fb-survey, even though four variations are available (raw vs smoothed; weighted vs unweighted). Even more casually, 'signal' is sometimes used as a synonym for 'indicator', e.g. "the cases and deaths signal".
- **geo level, geo type, regional level** - usually one of: state, county (fips), hospital referral region (hrr), metropolitan statistical area (metro area; msa), designated market area (dma).

- **geo id, region id** - an alphanumeric code identifying a particular state, county, hrr, msa, or dma. For example, the geo id of a state might be 'pa', 'ca', 'oh'; the geo id of a county might be '15215', '02492', '94301'.
- **issue (data versioning)** - like a magazine issue; a collection of data that was published together. For daily signals, the issue date is a day. For weekly signals, the issue date is an epidemiological week ("epiweek"). In COVIDcast we use a diff-based issue that includes only the rows that changed during the time period covered by the issue. Rows that stayed the same are not explicitly confirmed. Rows that were removed are not currently distinguished from rows that stayed the same; this will be addressed in a missingness encoding scheme TBD.
- **issue (github)** - a description of a task or a bug, along with all the notes, comments, commits, and pull requests associated with that task or bug.
- **lag** - the number of time units between when an event occurred and when data about those events were published. It's a purely operational definition: `issue - time_value`.
- **backfill** - what happens when previously-reported values are updated as additional data about those events become available. Backfilled data by definition has greater lag than what was first reported.

Naming Conventions Within Indicator Code

- run.py
    - def run_module()
        - Loads params and passes them along
- constants.py (optional)
    - Indicator-level constants in all caps
- pull.py
    - def pull_data()
        - Pulls data from source
- geo.py (optional)
    - def geo_map()
        - Wrapper around GeoMapper that performs indicator-specific transformations
- data_tools.py
    - Indicator-specific data transformation and smoothing utilities