Data engineering task | Advisory data pipeline

Context

Echo's advisory is the backbone of how we translate raw vulnerability data into trusted, explainable security outcomes.

Today, Echo publishes a public advisory dataset (`data.json`) that lists packages, CVEs, affected versions, and known fixed versions. This dataset is generated from Echo's scans and factory pipeline, including auto-fixes and custom patches, and is consumed by all scanner integrations and in-product CVE views.

Many CVEs do not have a known fix when they first appear. Others are later determined to be not applicable or won't fix based on upstream clarification. These determinations change over time as upstream ecosystems evolve.

This task is about designing, and partially prototyping, the system that continuously evaluates Echo's advisory data against upstream signals and internal inputs, and maintains a single, correct, and explainable source of truth.

Scope

Your goal is to design a system that periodically analyzes and enriches Echo's advisory data using upstream sources, and produces a correct, explainable advisory dataset over time.

You will be given:

- Echo's public advisory dataset: https://advisory.echohq.com/data.json
- A CSV of CVEs that should be treated as not applicable - This CSV simulates a live database table maintained by Echo's research and AI systems, and may change on every run. It should be treated as a continuously updated input, not a static artifact or source of truth.

You are expected to build logic that:

- Periodically analyzes all CVEs without a fixed version
    - If a fix exists upstream, enrich the advisory with it

    - If no fix exists, map the CVE to the correct state (not applicable, won't fix, or pending upstream)
- Bases decisions on upstream signals such as distro advisories, CVE metadata, or upstream discussions, combined with the live not-applicable feed
- Maintains a single advisory source of truth that includes
    - Current advisory state
    - Customer-facing explanations for every decision

What to focus on

- Advisory lifecycle
    - How CVEs move from unknown → pending → final states
    - How explanations are generated and preserved over time
- Upstream enrichment
    - Which public sources you rely on and why
    - How conflicting, partial, or stale upstream data is handled
    - How de-duplication across sources is enforced
- Efficiency & scale
    - Avoiding re-processing CVEs in final states
    - Rate limiting, retries, and backoff
    - Incremental updates vs full re-analysis
- System design
    - Migration path from the current advisory setup
    - Architecture and core tech stack choices
    - Extensibility for new feeds and advisory signals
- Observability & trust
    - Pipeline visibility and monitoring
    - Detection of stalled CVEs, data quality / integrity regressions, or silent failures

Deliverables

1. Technical design

Produce a concise technical design focused on system architecture and implementation. It should cover:

- System boundaries and ownership
- Data ingestion layer
- Core storage and data model
  - Representation of current advisory state
  - Customer-facing explanation fields and provenance
- Processing and orchestration
  - Periodic analysis flow
  - State transition logic
  - Handling of live input changes
- Efficiency mechanisms
- Reliability and observability
  - Failure modes and recovery
  - Monitoring, alerting, and data quality guarantees

2. Short deck (max 5 slides)

To be presented live. Cover:

- Advisory data model, pipeline and lifecycle
- End-to-end architecture and tech stack
- Key tradeoffs and non-goals
- How explainability is preserved for customers

3. Prototype

Build a small prototype that:

- Consumes `data.json` and the live CSV input
- Enriches CVEs using at least one upstream source

- Assigns `fixed`, `pending_upstream`, or `not_applicable` states with explanations
- Emits an updated advisory view

This does not need to be close to production-grade. It should demonstrate sound judgment, feasibility, and clarity.

What matters to us

- Ability to reason about high-volume, messy security data
- Strong instincts around decision modeling and temporal data
- Clear separation between ingestion, enrichment, and decisioning
- Evidence you can build this yourself and later grow the infrastructure and team around it