ChatGPT

OpenObserve Capabilities for Schema and Dashboard Integration

Retrieving Stream Schemas via OpenObserve API

OpenObserve provides a **built-in API to fetch the schema of any log/metric/trace stream**. For each stream (equivalent to an index), you can call the schema endpoint to retrieve all field names and data types. The official documentation specifies an HTTP GET endpoint of the form:

```
GET /api/{organization}/streams/{stream}/schema?type={StreamType}
```

For example, to get the schema of a logs stream, you would replace `{organization}` with your org identifier (e.g. "default") and `{stream}` with the stream name. The `type` query param can be `logs`, `metrics`, or `traces` (defaults to logs) [1]. This is exactly what your sample `curl` command was doing for `walmart_app_analytics`. When called with proper authentication (discussed below), this endpoint returns a JSON object containing the stream's metadata and schema. For instance, the response includes a `"schema"` array with each field's name and type [2].

Feasibility: This is completely feasible using OpenObserve's inbuilt API. It's an officially supported endpoint, so no hacks are needed. You can even list all streams first via the `GET /api/{org}/streams` API (with `fetchSchema=true` to include schemas in one call) [3] [4], or query specific streams individually. The onboarding team can programmatically retrieve each stream's schema from your multiple OpenObserve URLs using these endpoints. Once the raw schema JSON is obtained, it can be fed into an LLM to generate human-friendly field descriptions. Those descriptions can then be reviewed and merged (e.g. via a documentation PR or into your RAG knowledge base). In short, **OpenObserve's API fully supports extracting schemas**, enabling your workflow of enhancing schemas with LLM-generated descriptions.

Obtaining Top Field Values for Context

To enrich field descriptions, you mentioned getting the "top 10 values" of each field. OpenObserve makes this feasible as well. There are a couple of approaches:

- **Using the Search Query API:** You could run an aggregation query (for example, an SQL query using `COUNT()` or the `approx_topk` function) to find the most common values per field. OpenObserve's search APIs allow running such queries programmatically [5]. However, crafting and running separate queries for each field might be tedious.

- **Using the `_values` Endpoint:** OpenObserve actually has a convenient **"Search Values" API** that returns the most frequent values for specified fields. The documented endpoint is:

```
GET /api/{organization}/{stream}/_values?
fields=field1,field2&size=10&start_time=...&end_time=...
```

This will retrieve up to the top 10 values for the given field(s) over the specified time range [6]. The response includes each field's values along with their occurrence counts [7] (e.g. `zo_sql_key` for the value and `zo_sql_num` for its frequency). Using this API, the onboarding team can programmatically fetch the common values for each field in a stream without manually writing queries. These values can then be appended to the LLM-generated description to provide examples or context.

Feasibility: Very high. This is an inbuilt capability of OpenObserve. By leveraging the `_values` API (or custom queries if needed), you can automate retrieval of sample field values. This helps the LLM produce more accurate and informative descriptions. All of this can be done through OpenObserve's APIs – no external database or manual exports required.

Accessing Dashboard Configurations via API

Your use case also involves processing dashboard definitions – specifically extracting each panel's query (or VRL function query), associated stream, and title in order to improve the titles. While the OpenObserve documentation doesn't list a formal "dashboard API" in the same way as streams, **the functionality is available**. In fact, the OpenObserve UI itself uses API calls to manage dashboards. There are two key approaches:

- **Exporting Dashboards Individually:** OpenObserve allows you to **export a dashboard as a JSON file** from the UI [8]. This JSON contains the dashboard's full configuration, including all panels and their details (title, query, any transformation logic, etc.). The official docs note that the *Export Dashboard* button will download the dashboard configuration JSON [8]. Programmatically, you can replicate this by calling the underlying endpoint. Typically, it looks like:

  ```
  GET /api/{organization}/dashboards/{dashboardId}
  ```

  (or a similar endpoint) to retrieve the JSON. Each panel in the JSON will have fields such as its `title`, the `query` or `vrlFunctionQuery` it runs, and the `stream` it queries. With authentication, you can iterate over dashboards to pull their JSON.

- **Listing Dashboards via API:** OpenObserve also appears to offer an API to list dashboards (as suggested by your provided `curl` with `page_num` and `folder` parameters). Although not explicitly documented on the public docs, the presence of a `GET /api/{org}/dashboards?...` endpoint is confirmed by the UI's behavior. For example, listing dashboards in a folder (or all folders) returns a list of dashboard metadata and possibly their panel definitions. In your case, using `page_size=1000` and specifying the folder ID likely returned all dashboards in that folder including panels data. Once you have that JSON (either via a bulk list or individual exports), you have what you need: each panel's title, query text, and associated stream.

Feasibility: Achieving this via built-in APIs is **feasible**. You might need to use internal API endpoints (since dashboards are not yet in the published API reference), but these endpoints do exist and are used by OpenObserve's web UI. As long as you include the proper auth header, you can GET the dashboard definitions. After retrieving them, feeding panel titles (and their queries for context) into an LLM is straightforward. The LLM can suggest clearer, more natural-language titles for panels. Those improved titles would then be reviewed and, if approved, you can update the dashboards (likely via an update API or by re-importing the JSON with new titles). In practice, updating could even be automated via an API call (OpenObserve has create/update dashboard APIs under the hood, as indicated by GitHub issues), but manual review via a PR process as you described is wise for quality control. Once updated, these clearer titles can be stored in your "sample queries DB" to improve natural language search through your RAG system.

Authentication and API Access Considerations

All the above API calls require proper authentication. **OpenObserve uses HTTP Basic Authentication** for its API by default. The official docs state that you must include an `Authorization: Basic ...` header in your requests, where the value is a Base64 encoding of `username:password` [9]. (For example, an admin user with password would use `Basic YWRtaW46<base64pwd>`). Ensure you send API requests over HTTPS to protect credentials [10].

In your cURL examples, you included cookies and tokens from an existing session. While that can work (if you reuse a logged-in session's cookies), the more direct method for automation is to use Basic Auth or an API token. If your OpenObserve instance is integrated with SSO (as some enterprise setups are), you may need to obtain a session token or use an API key if supported. Check your deployment's auth configuration – but in general, **the official approach is Basic Auth with an OpenObserve user account**. Once authenticated, the API endpoints for streams, search, and dashboards will respond with the data you need.

Authenticated API behavior is otherwise the same as what you see in the UI: if your user has permissions to view certain streams or dashboards, the API will return those. If not, you'd get a permission error. So the onboarding team's service account (or user credentials) should have access to all relevant streams and dashboards across those multiple URLs/environments.

Feasibility of the LLM-Driven Enhancement Workflow

Bringing it all together, the workflow you described is **highly feasible** with OpenObserve's capabilities:

- **Schema enhancement:** The data (field names and types) is readily obtainable via API, and you can augment it with live data examples (top values) via the values API. OpenObserve doesn't natively generate field descriptions, but leveraging an LLM for that is an external process that complements OpenObserve. Many teams follow a similar approach of documenting schemas with AI assistance. Since all required inputs (schema and sample values) can be fetched automatically, this part can be implemented with relative ease.

- **Dashboard title improvement:** Again, the raw material (dashboard panel metadata) can be fetched from OpenObserve. Crafting better titles is a creative step done by the LLM, which is outside OpenObserve's scope – but nothing stops you from doing it. OpenObserve will accept updated titles either through its UI or via its API. The fact that OpenObserve dashboards are just JSON means they can be version-controlled. Your plan to create a PR for user review before updating is prudent and workable. You might even use the community dashboard GitHub repo approach (as OpenObserve does for sharing dashboards) as inspiration for your internal process.

- **Integration with RAG/NL queries:** By improving field descriptions and panel titles, you enrich the metadata that your Retrieval-Augmented Generation system can draw upon. While OpenObserve itself doesn't provide a RAG system, the outputs you generate (enhanced descriptions, clearer titles) can be stored in your own knowledge base or search index. Because these descriptions are reviewed by subject matter experts, they should boost the accuracy of any natural language query mapping to OpenObserve content. In essence, you are layering an intelligent documentation/search layer on top of OpenObserve – which is a reasonable approach.

In summary, **all parts of your plan are achievable with OpenObserve's built-in APIs and features**. The official documentation confirms the key endpoints for retrieving schemas and values, and the dashboard JSON export functionality covers the panels. With proper authentication and perhaps some scripting, the onboarding team can automate data extraction. The LLM steps are external but fully compatible with this data. After verification, the enriched information can be fed back into your systems (or even into OpenObserve as updated descriptions/titles). This kind of workflow – using OpenObserve as the source of truth and an LLM for augmentation – is not only feasible but aligns well with how OpenObserve is designed (flexible API access, JSON configurations, etc.).

References

- OpenObserve API Authentication and Usage [9]
- Stream Schema API (retrieve index schema) [1] [2]
- Stream List API (list streams, with optional schemas) [3] [4]
- Search Values API (top N values for fields) [6] [7]
- OpenObserve Dashboard Export (JSON configuration) [8]

---

[1] [2] Schema - OpenObserve Documentation
https://openobserve.ai/docs/api/stream/schema/

[3] [4] List - OpenObserve Documentation
https://openobserve.ai/docs/api/stream/list/

[5] [9] [10] API Index - OpenObserve Documentation
https://openobserve.ai/docs/api/

[6] [7] Value - OpenObserve Documentation
https://openobserve.ai/docs/api/search/value/

[8] Manage Dashboards - OpenObserve Documentation
https://openobserve.ai/docs/user-guide/dashboards/manage-dashboards/