Developer Guide

Generated on 2026-05-24

Contents

Developer

Welcome to the developer guide for Analytics Platform.

In this guide you can find the following.

- REST API documentation based on the OpenAPI (Swagger) standard.
- MCP server documentation.
- Library of generic SQL queries for data warehouses.

REST API

The REST API is ideal for developer seeking to build integrations, clients and web apps on top of the platform.

Authenticaton

The AP REST API supports *basic auth* for client authentication. Basic auth is a straightforward HTTP challenge-response framework that requires a client to provide a username and password to gain access to a resource. To transmit these credentials, the pair is combined into a single string, encoded using Base64, and passed within the `Authorization` header of the request. Basic auth is supported in all major HTTP clients.

Architecture

The AP API adheres to many of the principles behind the REST architectural style.

- **Resources:** The fundamental building blocks are referred to as resources. A resource can be anything exposed to the web, from a document to a business process.
- **Unique IDs:** All resources can be uniquely identified by a *URI*, also referred to as a *URL*.
- **Content negotiation:** JSON is the default request and response format. You can indicate that you are interested in a specific format by supplying an *Accept* HTTP header or a file extension.
- **HTTP methods:** Interactions with the API requires the use of *HTTP methods*. Use a *GET* request to retrieve a resource, a *POST* request to create a new resource, a *PUT* request to update (replace) an existing resource and a *DELETE* request to remove a resource.

MCP server

The MCP server provides a range of MCP tools and resources, offering a standardized interface for developers to integrate the data orchestration and analytical capabilities of AP directly into LLM-based ecosystems. This integration exposes tools and resources for data catalog access, SQL query execution, and workflow management, allowing external models to interact with the platform as a native extension of their reasoning capabilities. With the AP MCP server, developers can build clients such as autonomous data agents, AI assistant integrations for platforms like Claude and ChatGPT and custom developer tools such as CLIs.

API reference

Analytics Platform provides a comprehensive REST API which covers most platform services.

API reference documentation for the AP backend services is linked to below.

- Identity
- Data pipeline
- DHIS2 Superset gateway

MCP server reference

AP exposes a Model Context Protocol (MCP) server that gives LLM agents and integrations structured, typed access to analytics data. Use it to build data agents, AI-powered dashboards, reporting integrations, or any client that needs to query or explore health and program data.

The server provides three capabilities:

- **Query data warehouse:** Translate natural-language questions into SQL and execute them against a client's analytics database.
- **Query DHIS2 analytics:** Run structured analytics queries against a DHIS2 instance and get results in a uniform tabular format.
- **Discover metadata:** Read database schema structure and resolve human-readable concepts (e.g. *"malaria"*, *"ANC first visit"*) to DHIS2 UIDs via semantic search.

Connection

The MCP server is served over the standard MCP transport: JSON-RPC for message serialization, supporting two main transport mechanisms: STDIO (Standard Input/Output) and HTTP/SSE (Server-Sent Events).

Every tool and resource requires a `client` identifier parameter. The client identifier is scopes the request to the data storage and data warehouse of a specific tenant (client).

Resources

Resources are read-only, context-providing endpoints that an agent can fetch before or alongside tool calls.

Database schema metadata

`db://{client}/metadata`

Returns JSON describing the full database structure (all schemas, tables, and columns) for the given client.

URI template	`db://{client}/metadata`
Path parameter	`client`: Client identifier
Returns	JSON-serialised database metadata

Use this resource to give an agent a complete picture of available tables and columns before generating SQL or natural-language queries.

Example URI

`db://zH65n3YxcHj/metadata`

Filtered database metadata

`db://{client}/metadata/{schemas}`

Same as above, but restricted to a comma-separated list of schema names. Prefer this over the unfiltered variant when the data warehouse contains many schemas and the agent only needs to reason about a subset.

URI template	`db://{client}/metadata/{schemas}`
Path parameters	`client` — client identifier`schemas` — comma-separated schema names, e.g. `public,staging`
Returns	JSON-serialised database metadata for the listed schemas only

Example URI

`db://zH65n3YxcHj/metadata/public,reporting`

Tools

Natural-language analytics

`queryDatabase`

Translates a free-text question into SQL, executes it against the client's analytics database, and returns the result set along with the generated SQL.

Parameters

Name	Type	Required	Description
`client`	`string`	yes	Client identifier
`query`	`string`	yes	Natural-language query, e.g. "Show total ANC visits by district for 2024"
`schemas`	`string[]`	no	Schema name filter. Restricts SQL generation to the listed schemas. Pass an empty list to search all schemas.

Returns — `DataQueryResponse`

Example

```
{
  "client": "zH65n3YxcHj",
  "query": "What were the top 5 districts by malaria cases last quarter?",
  "schemas": ["public"]
}
```

Notes

- The tool reads schema metadata from the resource endpoint internally. Providing a `schemas` filter speeds up context retrieval and improves SQL accuracy on large databases.
- On error the tool returns a response with `null` for `result` and `sqlQuery`, inspect `duration` to detect a timeout.

Semantic metadata search

`searchMetadata`

Generates a vector embedding for the given phrase and runs a cosine-similarity search against the `metadata_embedding` table in the client's data warehouse. Use this to resolve human-readable terms (e.g. *"malaria"*, *"ANC first visit"*) to DHIS2 UIDs before calling `queryAnalytics`.

Parameters

Name	Type	Required	Description
`client`	`string`	Yes	Client identifier
`phrase`	`string`	Yes	Natural-language phrase or keyword to search for
`dataPipeline`	`string`	Yes	ID of the DHIS2 data pipeline whose embeddings table to search
`objectTypes`	`string[]`	No	Metadata types to restrict results to, e.g. `["data_element","indicator","program"]`. Pass empty list to search all.
`limit`	`integer`	No	Maximum number of results to return (default 10)

Returns — `MetadataSearchResult[]` (see MetadataSearchResult)

Example

```
{
  "client": "zH65n3YxcHj",
  "phrase": "Antenatal care first visit",
  "dataPipeline": "dhis2-main",
  "objectTypes": ["data_element", "indicator"],
```

```
  "limit": 5
}
```

Typical agent workflow

1. Call `searchMetadata` to resolve concept names to UIDs.
2. Use the returned UIDs as items in the `dx` dimension of `queryAnalytics`.

DHIS2 analytics query

`queryAnalytics`

Executes a DHIS2 `/api/analytics` request against the DHIS2 instance configured in the given data pipeline. Returns results in the same `DataTable` shape as `queryDatabase`, so downstream processing is uniform.

Parameters

Name	Type	Required	Description
`client`	`string`	yes	Client identifier
`dataPipeline`	`string`	yes	ID of the DHIS2 data pipeline to query
`queryJson`	`string`	yes	JSON-serialised `AnalyticsQuery` object (see below)

Returns — `AnalyticsQueryResponse`

`queryJson` schema `queryJson` is a JSON string with the following top-level fields:

Field	Type	Required	Description
`dimensions`	`QueryDimension[]`	yes	Dimensions to break down by; at least one required. See standard dimensions.
`filters`	`QueryDimension[]`	no	Dimensions to filter on without returning them as columns.
`aggregationType`	`string`	no	Override aggregation function: `SUM`, `AVERAGE`, `COUNT`, `STDDEV`, `VARIANCE`, `MIN`, `MAX`, `NONE`.
`startDate`	`string`	no	Start of date range in `yyyy-MM-dd` format. Use as an alternative to `pe`.
`endDate`	`string`	no	End of date range in `yyyy-MM-dd` format. Must be provided together with `startDate`.

Field	Type	Required	Description
`outputIdScheme`	`string`	no	Output identifier scheme: `UID` (default), `NAME`, `CODE`.

Each `QueryDimension` object has the shape:

```
{ "dimension": "<key>", "items": ["<item1>", "<item2>"] }
```

Standard dimensions

Dimension key	Meaning	Supported item formats
`dx`	Data elements, indicators, program indicators, or data sets	DHIS2 UIDs
`ou`	Organisation units	DHIS2 UIDs; `LEVEL-<n>` for all units at a level; `OU_GROUP-<uid>` for a group
`pe`	Periods	Fixed periods (`202401`, `2024Q1`) or relative codes (`LAST_12_MONTHS`, `THIS_YEAR`, `LAST_YEAR`)

Dimension placement rules The validator enforces the following rules before executing the query. Errors are returned in the `errors` array of the response rather than as exceptions.

- `dimensions` must contain at least one entry.
- `dx` must appear in exactly one of `dimensions` or `filters`.
- `ou` must appear in exactly one of `dimensions` or `filters`.
- `pe` must appear in exactly one of `dimensions` or `filters` **unless** `startDate`/`endDate` are used, in which case `pe` must be omitted entirely.
- A dimension key cannot appear in both `dimensions` and `filters`.
- Every dimension entry must include at least one item.

Warnings The `warnings` array in the response may contain the following informational strings:

Warning	Meaning
No rows returned for the requested ou and pe combination	The query succeeded but DHIS2 returned an empty data set.
Result set was truncated by DHIS2	DHIS2 cut the result set short. Narrow the query scope.

Example: Period dimension, output as names

```
{
  "dimensions": [
    {"dimension": "dx", "items": ["Uvn6LCg7dVU", "fbfJHSPpUQD"]},
    {"dimension": "ou", "items": ["LEVEL-2"]},
    {"dimension": "pe", "items": ["LAST_12_MONTHS"]}
  ],
  "filters": [
    {"dimension": "dx", "items": ["cYeuwXTCPkU"]}
  ],
  "aggregationType": "SUM",
  "outputIdScheme": "NAME"
}
```

Example: Date range instead of period dimension

```
{
  "dimensions": [
    {"dimension": "dx", "items": ["Uvn6LCg7dVU"]},
    {"dimension": "ou", "items": ["ImspTQPwCqd"]}
  ],
  "startDate": "2024-01-01",
  "endDate": "2024-12-31"
}
```

Response types

`DataQueryResponse`

Returned by `queryDatabase`.

Field	Type	Description
`startTime`	`string` (ISO-8601)	When the query started
`duration`	`integer`	Total execution time in milliseconds
`sqlQuery`	`string`	The SQL generated from the natural-language input
`pager`	`PagingDto`	Pagination metadata (`page`, `pageSize`, `total`)
`result`	`DataTable`	The result set

`AnalyticsQueryResponse`

Returned by `queryAnalytics`.

Field	Type	Description
`startTime`	`string` (ISO-8601)	When the query started
`duration`	`integer`	Total execution time in milliseconds
`appliedQuery`	`AnalyticsQuery`	The effective query that was sent to DHIS2
`result`	`DataTable`	The normalised result set; `null` on error
`warnings`	`string[]`	Non-fatal warnings (see Warnings)
`errors`	`AnalyticsQueryError[]`	Validation or execution errors

`DataTable`

Shared result shape used by both `queryDatabase` and `queryAnalytics`.

Field	Type	Description
`schema`	`string`	Source schema name (or `"Query response"` for analytics results)
`name`	`string`	Source table name
`columns`	`Column[]`	Column descriptors — each has `label` (display name), `name` (technical name), and `dataType`
`rows`	`any[][]`	Row data; each row is an ordered array matching `columns`. Numeric values are returned as `number`, all others as `string`.
`width`	`integer`	Number of columns (convenience field)
`height`	`integer`	Number of rows (convenience field)
`queryStartTime`	`string` (ISO-8601)	When the underlying query began
`queryDuration`	`integer`	Database execution time in milliseconds

`AnalyticsQueryError`

Field	Type	Description
`field`	`string`	The input field that caused the error (e.g. `"dimensions"`, `"pe"`, `"execution"`)
`message`	`string`	Human-readable error description

`MetadataSearchResult`

Field	Type	Description
`objectType`	`string`	DHIS2 metadata type, e.g. `data_element`, `indicator`, `program`
`uid`	`string`	DHIS2 UID — use as item value in `dx` or other dimensions
`name`	`string`	Full display name
`shortName`	`string`	Abbreviated display name
`code`	`string`	Alphanumeric identifier code (may be empty)

Agent patterns

Natural language analytics

Answering analytics questions in natural language.

```
1. Fetch  db://{client}/metadata/{schemas}          → understand the schema
2. Call   queryDatabase(client, question, schemas)  → get SQL + results
3. Return result.rows to the user
```

DHIS2 analytics

Structured DHIS2 analytics with automatic UID resolution.

```
1. Call   searchMetadata(client, concept, pipeline, objectTypes)  → get UIDs
2. Build  queryJson using the resolved UIDs
3. Call   queryAnalytics(client, pipeline, queryJson)             → get DataTable
4. Return result.rows to the user; surface warnings if non-empty
```

Natural language and DHIS2

Combined natural language analytics questions and DHIS2 analytics in a single agent turn.

```
1. Fetch  db://{client}/metadata                    → full schema context
2. Call   searchMetadata(...)                       → resolve domain terms to UIDs
3. Call   queryAnalytics(...)                       → fetch aggregated figures
4. Call   queryDatabase(...)                        → fetch related warehouse data
5. Synthesise and return
```

Error handling

Scenario	How it surfaces	Recommended agent action
Invalid `queryJson` (validation failure)	`errors` array non-empty, `result` is `null`	Parse `errors[].field` and `errors[].message`. Revise and retry.
DHIS2 API error	`errors[0].field == "execution"`	Surface the message to the user. Do not retry automatically.
Empty result set	`warnings` contains no-rows message	Inform the user. Suggest broadening org unit or period scope.
Truncated result	`warnings` contains truncated message	Suggest narrowing `ou` or `pe` to retrieve complete data.
`queryDatabase` internal error	`result` is `null`, all other fields are `null`	Treat as transient; retry once, then surface failure.
`searchMetadata` internal error	Returns empty array `[]`	Fall back to asking the user for a UID, or try a different phrase.