LOGZILLA DOCUMENTATION

Making Queries

Create LogZilla queries through POST /api/query, handle asynchronous 202 responses with query_id polling, and subscribe to WebSocket result streams

LogZilla API · Generated May 3, 2026 · logzilla.ai/docs/logzilla-api/making-queries

Making Queries

The LogZilla Query API provides fundamental operations for data retrieval and analysis, including query creation, asynchronous request handling, and websocket connections for real-time updates.

Prerequisites

- Valid LogZilla API authentication token (see Getting Started (https://www.logzilla.ai/docs/logzilla-api/getting-started))
- Understanding of REST API concepts and HTTP methods
- HTTP client capable of POST requests
- JSON data format familiarity

Creating a New Query

A new query is created through `POST /api/query`, and always includes two parameters (usually with JSON body):

type
Indicates which query you want to perform. See Event Queries (https://www.logzilla.ai/docs/logzilla-api/event-query-types) and System/Admin Queries (https://www.logzilla.ai/docs/logzilla-api/system-query-types) for more detail.

params
A JSON object containing the parameters for the query. Every query type has a different list of available parameters.

After creating a query you can get its results either immediately (if it was able to complete in 1 second) with response `200 "OK"`, or (for requests which must be completed asynchronously) a status of `202 "ACCEPTED"` with response body containing a `query_id`.

Asynchronous Requests

NOTE: Although you can query for results at any time with `GET /api/query/<id>` the recommended way of getting query results is to use websockets and subscriptions (see below).

If the initial query returns `202 "ACCEPTED"` check for the query results using the query id value returned from the first query using `GET /api/query/<id>` to get updated results.

Relative Time Queries

Asynchronous queries that use relative time ranges, like "last hour", are perpetually updated to represent changing time, even if they have status `SUCCESS`.

Polling Query Results

To retrieve the current data of an existing query (whether currently processing or not) use `GET /api/query/<id>`.

`GET /api/query/<id>` can return paged results of the data by providing additional parameters of `page_size=<num of results>` and `page=<page number>`. The HTTP result message is always returned immediately but the query status (in the returned JSON) could indicate that the query is incomplete (query status `IN_PROGRESS`) or even as of yet empty (query status `PENDING`).

For example, if the query is not completed immediately the received response would be:

```
{
  "query_id": "72bc846140344b4da3cdcfb831174a3e",
  "status": "IN_PROGRESS",
  "type": "Search",
  "base_time": 1416233863,
  "results": {
    "..."
  },
  "params": {
    "sort": [
      "first_occurrence"
    ],
    "filter": [],
    "page": 1,
    "page_size": 100,
    "time_range": {
      "ts_from": 1000,
      "ts_to": 10000
    }
  },
  "owner_id": 1
}
```

When a query is completed (possibly immediately) the response would be:

```
{
  "query_id": "72bc846140344b4da3cdcfb831174a3e",
  "status": "SUCCESS",
  "type": "Search",
  "base_time": 1416233863,
  "results": {
```

```
      "..."
    },
    "params": {
      "sort": [
        "first_occurrence"
      ],
      "filter": [],
      "page": 1,
      "page_size": 100,
      "time_range": {
        "ts_from": 1000,
        "ts_to": 10000
      }
    },
    "owner_id": 1
}
```

Getting Query Results via Websocket

The recommended way of getting query results, especially for widgets, is using websockets. Using the websocket method (vs. `GET /api/query/`) provides initial calculation results, partial results for asynchronous queries, and final results of the query.

Note: You must first create a query with `POST /api/query` and only then use websockets for polling the results.

Websockets for the API are available under `/ws/live-updates` and, after establishing the connection, allows for real-time subscription and unsubscription on events of interest.

Websocket operations should be sent using encoded JSON with an array of commands and parameters, for example:

```
["subscribe", "widget", 2]
```

Subscription to a particular query or widget can be accomplished by providing the appropriate entity id, for which query id is a string and widget id is an integer, subscription to the whole dashboard can be requested in which case websocket updates will then include get updates for all widgets on that dashboard.

Unsubscribing can be accomplished either by providing the same parameters that were used for the subscription, or removal of all subscriptions with:

```
["unsubscribe", "all"]
```

After successful subscription/unsubscription a confirmation result will be returned, which always contains the list of currently subscribed items:

```
["subscription-update", {"query": [], "widget": [2], "dashboard": []}]
```

Once subscribed updates for the requested objects will begin. Each update is a separate message as follows:

```
["widget-update", {
    "widget_id": 2,
    "dashboard_id": null,
    "data": {
        "status": "SUCCESS",
        "query_id": "4f29934c97b1c0857c2341c3cb188371",
        "results": {
            "totals": "...",
            "details": "..."
        }
    }
}]
```

For widgets that are directly subscribed the dashboard_id as shown above will be null. For query subscriptions, both dashboard_id and widget_id will be null. The data field contains exactly the same content that `GET /api/query/<id>` would return, as indicated in the documentation for each particular request type.

Verification

Test query creation with a simple request:

```
curl -X POST -H "Authorization: token YOUR_TOKEN_HERE" \
     -H "Content-Type: application/json" \
     -d '{"type": "Search", "params": {"time_range": {"preset": "last_1_hours"}}}' \
     "http://your-logzilla-server/api/query"
```

Successful query creation returns a JSON response with query_id.

References

- Query API Parameters (https://www.logzilla.ai/docs/logzilla-api/query-api-parameters) - Common parameters and filtering options
- Event Queries (https://www.logzilla.ai/docs/logzilla-api/event-query-types) and System/Admin Queries (https://www.logzilla.ai/docs/logzilla-api/system-query-types) - Detailed documentation for all query types