Version 2
Date: 10.10.2025

DataSignal User Manual

Having up-to-date and high-quality data is essential in a fast-paced business environment. **DataSignal** is a powerful data observation system designed to provide you with real-time data monitoring, enabling you to track, analyze, and respond to changes instantly. It allows you to connect sources, set up regular data reviews on a schedule, and track key metrics with minimal latency. In case of critical deviations, the system will automatically generate a notification and send it to the selected channel. This will help you quickly respond to changes and make decisions based on reliable information.

User Guide

About This Guide

This guide demonstrates how DataSignal works through a step-by-step setup for one of the possible data monitoring use cases. By following the example, you'll learn the core principles of DataSignal's operation and see how the configuration looks in practice.

Our goal is to provide a practical example that helps you understand how to use DataSignal to monitor your data effectively.

In this section, we will use cURL (although you can use any **API Interaction Tools** tool you prefer, such as Postman, Insomnia, etc.) to demonstrate how to send HTTP requests to the DataSignal API. To illustrate the process, we'll start with a simple example that guides you through the basic steps.

For demonstration purposes, we will use **PostgreSQL** as the database containing the observed data. However, DataSignal is not limited to PostgreSQL — it supports connections to various other database types, giving you flexibility to work with the systems that best fit your needs.

Use Case

Suppose you manage a chain of stores in several cities and accumulate large amounts of sales data every day. Some of the information is stored in a table with store reference data, and some on a table with transaction details: sales amount, number of returns, average check, etc. Agree that it is important to see these indicators in dynamics, compare them with planned values, and respond in a timely manner to any critical deviations.

Actual values may differ from our expectations or forecasts. Sales volumes or the number of visits can suddenly decrease, returns can increase, or the average check can fall. Such changes are often the first signals of possible problems, and it is extremely important to notice them in time to avoid negative consequences for the business.

Thus, with DataSignal you get a proactive management tool: critical deviations are detected at an early stage, the risk of financial losses is reduced, stability is ensured, and the speed and quality of decision-making are increased.

Getting Started with DataSignal API

In this section, we will take a detailed look at the process of setting up monitoring using the DataSignal API.

Authenticating with the DataSignal API

Step-by-step guide to setting up secure access credentials

Before you can send requests to the DataSignal API, you must be properly authenticated. This ensures that only authorized users can access the system and that your data remains secure.

1. Obtain Your API Credentials

Contact your system administrator or refer to your account settings in DataSignal to provide your user credentials or generate an **access token** for you. Keep these credentials secure — never share them publicly or commit them to source control.

An example is how you can receive token for next requests.

```
curl -L "http://<host>/api/v1/auth" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d "username=<username>" \
-d "password=<password>"
```

2. Adding Your Token to API Requests

To make requests to the DataSignal API, you must use the token generated in the previous step during the authorization process.

For example, the following request checks which user account you are currently authenticated as:

```
curl -L "http://<host>/api/v1/whoami" \
-H "Authorization: Bearer <token>"
```

If you are using an API interaction tool (such as Postman or Insomnia), configure the authorization as follows:

1. Open your request.
2. Go to the **Authorization** tab.
3. Select the appropriate type — **Bearer Token**.
4. Paste your token into the provided field.

Adding Channel

To ensure the possibility to send notifications to users, you first need to configure a Channel component.

A Channel specifies how the user will be notified (for example, by email or in other way) and specifies the format of the notification. The channel object stores all the information needed to deliver messages to the user in the desired manner.

Create the Channel Object via API

To create a Channel object, you need to send a POST request to the */api/v1/channels* endpoint with the required configuration details in JSON format.

Below is a sample JSON payload for creating an email channel:

```
{
  "data": {
    "type": "channel",
    "id": "channel_id",
    "attributes": {
      "name": "channel_name",
      "description": "The name of channel",
      "type": "email",
      "subject": "Message Subject",
      "host": "smtp.gmail.com",
      "port": "465",
      "username": " your_name",
      "password": "password",
      "recipients": ["your_email@gmail.com"],
      "tags": {
        "additionalProp1": "string"
      }
    }
  }
}
```

Below is an example of creating an **Email Channel** using the DataSignal API.

```
curl -L -X POST "http://<host>/api/v1/channels" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <token>" \
--data-raw "{
  "data": {
    "type": "channel",
    "id": "<channel_id>",
    "attributes": {
      "name": "<channel_name>",
      "description": "<Description of channel>",
      "type": "email",
      "subject": "<Your Message Subject>",
      "host": "smtp.gmail.com",
      "port": "465",
      "username": "<username>",
      "password": "<password>",
      "recipients": ["<recipient1_email>", "<recipient2_email>",],
      "tags": {"<tag_name>": "<value>"}
    }
  }
}"
```

By substituting the values you need into the request, you will create a channel object through which you can send notifications.

Pay attention to the ***id*** field, you will refer to the channel identifier you specified at this stage when creating the notification rule.

Adding Connection

To connect to a database or data warehouse, you first need to configure a Connection component.

A Connection specifies how the system will access your data source and contains all the necessary connection details such as host, port, authentication credentials.

Create the Connection Object via API

To create a **Connection** object, you need to send a POST request to the */api/v1/connections* endpoint with the required configuration details in JSON format.

Below is a sample JSON payload for creating connection to PostgreSQl:

```
{
  "data": {
    "type": "connection",
    "id": "connection_postgres",
    "attributes": {
      "name": "connection_postgres",
      "description": "Postgres connection",
      "engine": "postgres",
      "host": "db_host",
      "port": 5432,
      "username": "username",
      "password": "password",
      "database": "shop_db",
      "tags": {}
    }
  }
}
```

Accordingly, to create the connection object, you ought to make a POST request:

```
curl -L -X POST "http://<host>/api/v1/connections" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <token>" \
--data-raw "{
  "data": {
    "type": "connection",
    "id": "<connection_in>",
    "attributes": {
      "name": "<connection_name> ",
      "description": "Connection",
      "engine": "postgres",
      "host": "<db_host>",
      "port": 5432,
      "username": "<username>",
      "password": "<password>",
      "database": "<db_name>",
      "tags": {}
    }
  }
}"
```

After these two steps, you have already had the connection to your data source and channel where you expected to observe notifications.

Adding Dataset

Dataset is an abstraction on top of data, which allows you to define where the data is stored and what dimensions and metrics it contains. To monitor your data – you should specify the dataset object.

To configure the dataset's objects payload in request, you should be aware of main components that are required to be specified.

- **Connection** – object that you specify before, defined the connection to the data source (specify here id).
- **Dimensions** - map of dimensions, where dimension is a categorical column which can be used to filter or group the data.
- **Metrics** - map of metrics, where metric is numerical column, which can be used to aggregate the data.
- **Table** - represents table in the data source.

Create the Dataset Object via API

Accordingly, to create the **Dataset** object, you ought to make a POST request to */api/v1/datasets* endpoint with the required configuration details in JSON format.

```
curl -L -X POST "http://<host>/api/v1/datasets" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <token>" \
--data-raw "{
  <payload>
}"
```

Below is a sample JSON payload for creating dataset:

```
{
  "data": {
    "type": "dataset",
    "id": "shop_ds",
    "attributes": {
      "name": "Shops",
      "description": "This is a dataset with required metrics.",
      "connection": "<connection_to_db>",
      "dimensions": {
        ...
      },
      "metrics": {
        ...
      },
      "table": {
        ...
      }
    }
  }
}
```

Having configured the payload applicable for your case you will receive create appropriate dataset, for further monitoring.

Let's take a closer look at Dataset components.

Dimensions

Dimension represents some categorical indicator that determines how your data will be analyzed. They are not subject to aggregation, but they can help structure metrics and make data more informative. Roughly speaking, we can relate dimensions to the answer to the question "*by what criteria do we look at the data?*".

How to create *dimension* attribute

You ought to specify attributes according to your scenario.

- *name* – this is a name that you provide to dimension (optional).
- *description* – some descriptions that you can provide.
- *expression* – optional field allows you to specify expression for extracting the dimension from the data source, required if field is not set.
- *field* – this is a name of column in data source (required if expression not set).
- *type* – dimension type (default: string)
- *granularity* – date/time granularity (only if type is date/datetime; possible: year, month, week, day, hour, minute, second)

Metrics

Metric represents numerical indicators that can be aggregated (sum, average, maximum, minimum, etc.). They answer the question "*what exactly are we measuring?*" and are the basis for monitoring and verification rules.

How to create *metrics* attribute

You ought to specify attributes according to your scenario.

- *name* – this is a name that you provide to metric (optional).
- *description* – some descriptions that you can provide.
- *expression* – optional field allows you to specify expression for extracting the metric from the data source, required if field is not set.
- *aggregation* – function for the metric. Allowed values: sum, avg, min, max, count. This attribute makes sense only if the field attribute is set.
- *type* – type of metric. Allowed values: int, float

Tables

The table in a dataset defines a specific data source from which we take both dimensions and metrics. It acts as a kind of *"source of truth*," because the completeness and correctness of all analytics depend on the correct choice of table. Supports the ability to join tables.

How to create *tables* attribute

You ought to specify attributes according to your scenario.

- *name* – this is a name that you provide to metric (required).
- *database* – the name of the database in the data source (optional).
- *schema* – the name of the schema in the data source (optional).
- *relations* – list of relations to other tables, where list is separate object

It is important to note that DataSignal supports **the ability to join data from multiple tables**; to do this we need to specify in table attribute ***relations***.

To configure relations, we need next attributes:

- *join* – Join type. Allowed values: *inner*, *left*, *right*, *full* (required).
- *table* – table object, which defines the table in the data source (required).
- *match_fields* – **the list** of fields for joining. Each field is a *join field object*, that describes below (required).

Join field attributes

- parent – the name of the field in the parent table (required).
- child – the name of the field in the child table (required).

Let's look at an example how to configure dataset with its required inner attributes.

We consider the case of managing a chain of stores. Imagine we have **two tables**, that will be a single dataset for monitoring:

- **metrics** – the table stores time series with data by stores. Consist of attributes: *record_time*, *store_id*, *daily_revenue*, *transaction_value*, *foot_traffic*.
- **stores** - the table contains background information about stores. Consist of attributes: *store_id*, *n_employees*, *store_area*, *city*, etc.

Let's configure dataset objects to observe our tables.

Thus, as **dimensions** we will represent the following attributes:

- *store_id* — unique store identifier for grouping metrics per store, e.g., comparing revenue dynamics across outlets.
- *city/state* — categorical attributes for aggregating data at a higher level, useful for comparing regions or spotting geographic trends.
- *record_time* — time dimension for grouping metrics by period, enabling trend analysis, seasonality detection, and anomaly spotting

```
  "dimensions": {
    "store_id": {
      "expression": "stores.store_id"
    },
    "city": {
      "field": "city"
    },
    "state": {
      "field": "state"
    },
    "record_time": {
      "type": "datetime",
      "field": "record_time",
      "granularity": "day"
    },
    "record_date": {
      "type": "date",
      "expression": "DATE(record_time)",
      "granularity": "day"
    }
  }
```

As a **metrics table** will include the following attributes:

- *daily_revenue* — daily revenue of the store, is one of the key indicators of financial performance.
- *transaction_value* — total amount of transactions per day.
- *foot_traffic* — number of visitors.
- *n_employees* — number of employees in the store.
- *store_area* — area of the store.

```
    "metrics": {
      "total_revenue": {
        "expression": "SUM(daily_revenue)"
      },
      "revenue": {
        "expression": "daily_revenue"
      },
      "transactions": {
        "expression": "SUM(transaction_value)"
      },
      "foot_traffic": {
        "expression": "MIN(foot_traffic)"
      },
      "employees": {
        "expression": "SUM(n_employees)"
      },
      "store_area": {
        "expression": "SUM(store_area)"
      }
    }
```

Configuring **Table** object:

Now we need to configure the table object, where we are going to make join between two tables metrics and stores via the *store_id* field creates a complete dataset.

So, we want to use as data source the table that is equivalent to table would gain because of the query:

```
SELECT * FROM shop.metrics AS m INNER JOIN shop.stores AS s ON m.store_id = s.store_id;
```

Thus, we must specify the following configuration:

```
    "table": {
      "name": "metrics",
      "schema": "shop",
      "relations": [
        {
          "join": "inner",
          "table": {
            "name": "stores",
            "schema": "shop"
          },
          "match_fields": [
            {
              "parent": "store_id",
              "child": "store_id"
            }
          ]
        }
      ]
    }
```

Adding Rule

Rule is the core of the system, is used to define the conditions that must be met for an alert to be generated. They define how frequently the rule is evaluated, the criteria that must be met, and the actions to execute when the rule is triggered.

The **Rule** has its own requirement to specify concepts such as:

- *enabled* – whether the rule is enabled or not. Default is true.
- *schedule* – the schedule of the rule. Allowed values N minutes, N hours, N days, N weeks. Where N is a number, or cron expression (*required*).
- *dataset* – object that you specify before, defined the dataset will be used for evaluating the rule (specify here id) (*required*).
- *metric* – the metric that will be used for evaluating the rule. Metric should exist in the dataset. Metric should exist in the dataset (*required*).
- *dimensions* – the list of dimensions that will be used for evaluating the rule. Each dimension should exist in the dataset.
- *channels* – the list of channels that will be used for sending notifications (can be used already configured) (*required*).
- ***filters*** – the list of filters that will be used for filtering the data in dataset (not required field, but it is important to be aware with).
- ***criteria*** – the list of criterions that will be used for defining the conditions that must be met for an alert to be generated (*required*).

Create the Rule Object via API

To create a **Rule** object, you need to send a POST request to the */api/v1/rules* endpoint with the required configuration details in JSON format.

The scheme the same:

```
curl -L -X POST "http://<host>/api/v1/rules" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <token>" \
--data-raw "{
  <payload>
}"
```

On a payload you must refer to components that you are going to monitor and use for sending alerts.

Below is a sample JSON payload for creating rules:

```
{
  "data": {
    "type": "rule",
    "id": "<id>",
    "attributes": {
      "name": "<name>",
      "description": "This is description",
      "schedule": "<n granularity>",
      "enabled": false,
      "dataset": "<specified dataset>",
      "dimensions": [<required_dim_name_1>, <required_dim_name_2>],
      "metric": "<observed_metric_name>",
      "filters": […],
      "criteria": […],
      "channels": […]
    }
  }
}
```

Let's take a closer look at Rule components.

Filters

Filters are used to **limit the dataset** to only the relevant subset of data before applying the criteria. They help ensure that rules are evaluated only against the data that matters for your use case.

How to create *filters* attribute

You ought to specify attributes according to your scenario.

- *dimension* – the dimension that will be used for filtering the data in dataset. Dimensions should exist in the dataset.
- *comparison_operator* – the comparison operator. Allowed values: **eq**, **neq**, **gt**, **gte**, **lt**, **lte** (*required*).
- *value* – the value that will be used for filtering the data in dataset (*required*).

Criteria

Criteria define the logical conditions that determine when an alert should be triggered.

How to create *criteria* attribute

You ought to specify attributes according to your scenario.

- *severity* – severity of the criterion. Allowed values: **low**, **medium**, **high** and **critical** (*required*).
- comparison_operator – comparison operator. Allowed values: **eq**, **neq**, **gt**, **gte**, **lt**, **lte** (*required*).

- *threshold* – threshold that will be used for comparing the value of the metric (*required*).

Let's look at an example how to configure rule with its required inner attributes.

We proceed considering the case of managing a chain of stores, where we have two tables, and have already created **dataset**.

Imagine, we want to observe what is happening with our branch in Dawson city, and for us it is very important that the daily profit in that city does not fall below a certain value.

Thus, we need to limit the selection by city using the Filters object:

```
{
  "data": {
    "type": "rule",
    "id": "dawson_shop_revenue",
    "attributes": {
      …
      "filters": [
          {
            "dimension": "city",
            "comparison_operator": "eq",
            "value": "Dawson"
          }
        ]
    }
  }
}
```

Using the Criteria object, we can set the threshold, comparison operator, and importance of the notification. For example, when the profit decreases by a certain value - we will receive a notification:

```
{
  "data": {
    "type": "rule",
    "id": "dawson_shop_revenue",
    "attributes": {
      …
      "metric": "revenue",
      "criteria": [
          {
            "severity": "critical",
            "comparison_operator": "lt",
            "threshold": 120000
          }
        ]
    }
  }
}
```

Thus, in our case, the payload for rule object could be specified as below:

```
{
  "data": {
    "type": "rule",
    "id": "dawson_shop_revenue",
    "attributes": {
      "name": "Dawson shop revenue",
      "description": "This simple rule checks whether the daily revenue from a store in Dawson
does not fall below 120000 ",
      "schedule": "60 minute",
      "enabled": true,
      "dataset": "shop_ds",
      "dimensions": [],
      "metric": "revenue",
      "filters": [
          {
            "dimension": "city",
            "comparison_operator": "eq",
            "value": "Dawson"
          }
        ],
        "criteria": [
          {
            "severity": "critical",
            "comparison_operator": "lt",
            "threshold": 120000,
            "compare_lag": null
          }
        ],
      "channels": ["channel_id"]
    }
  }
}
```

Following all of the above instructions will allow you to set up monitoring in a way that is most convenient for you.