+++ title = "Build a logs data source plugin" +++

# Build a logs data source plugin

This guide explains how to build a logs data source plugin.

Data sources in Grafana supports both metrics and log data. The steps to build a logs data source plugin are largely the same as for a metrics data source. This guide assumes that you're already familiar with how to [Build a data source plugin]({{< relref "/tutorials/build-a-data-source-plugin.md" >}}) for metrics.

## Add logs support to your data source

To add logs support to an existing data source, you need to:

- Enable logs support
- Construct the log data
- (Optional) Add preferred visualisation type hint to the data frame

### Enable logs support

Tell Grafana that your data source plugin can return log data, by adding `"logs": true` to the [plugin.json]({{< relref "metadata.md" >}}) file.

```
{
  "logs": true
}
```

### Construct the log data

Just like for metrics data, Grafana expects your plugin to return log data as a [data frame]({{< relref "data-frames.md" >}}).

To return log data, return a data frame with at least one time field and one text field from the data source's `query` method.

**Example:**

```
const frame = new MutableDataFrame({
  refId: query.refId,
  fields: [
    { name: 'time', type: FieldType.time },
    { name: 'content', type: FieldType.string },
  ],
});

frame.add({ time: 1589189388597, content: 'user registered' });
frame.add({ time: 1589189406480, content: 'user logged in' });
```

That's all you need to start returning log data from your data source. Go ahead and try it out in [Explore]({{< relref "../../explore/_index.md" >}}) or by adding a [Logs panel]({{< relref "../../visualizations/logs-panel.md" >}}).

Congratulations, you just wrote your first logs data source plugin! Next, let's look at a couple of features that can further improve the experience for the user.

### (Optional) Add preferred visualisation type hint to the data frame

To make sure Grafana recognizes data as logs and shows logs visualization automatically in Explore you have do set `meta.preferredVisualisationType` to `'logs'` in the returned data frame. See [Selecting preferred visualisation section]({{< relref "add-support-for-explore-queries.md#selecting-preferred-visualisation" >}})

**Example:**

```
const frame = new MutableDataFrame({
  refId: query.refId,
  meta: {
    preferredVisualisationType: 'logs',
  },
  fields: [
    { name: 'time', type: FieldType.time },
    { name: 'content', type: FieldType.string },
  ],
});
```

## Add labels to your logs

To help filter log lines, many log systems let you query logs based on metadata, or *labels*.

You can add labels to a stream of logs by setting the labels property on the Field.

**Example**:

```
const frame = new MutableDataFrame({
  refId: query.refId,
  fields: [
    { name: 'time', type: FieldType.time },
    { name: 'content', type: FieldType.string, labels: { filename: 'file.txt' } },
  ],
});

frame.add({ time: 1589189388597, content: 'user registered' });
frame.add({ time: 1589189406480, content: 'user logged in' });
```

## Extract detected fields from your logs

You can add additional information about each log line by adding more data frame fields.

If a data frame has more than one text field, then Grafana assumes the first field in the data frame to be the actual log line. Any subsequent text fields are treated as [detected fields]({{< relref "../../explore/_index.md#labels-and-detected-fields" >}}).

While you can add any number of custom fields to your data frame, Grafana comes with a couple of dedicated fields: `levels` and `id`. Let's have a closer look at each one.

### Levels

To set the level for each log line, add a `level` field.

**Example:**

```
const frame = new MutableDataFrame({
  refId: query.refId,
  fields: [
    { name: 'time', type: FieldType.time },
    { name: 'content', type: FieldType.string, labels: { filename: 'file.txt' } },
    { name: 'level', type: FieldType.string },
  ],
});

frame.add({ time: 1589189388597, content: 'user registered', level: 'info' });
frame.add({ time: 1589189406480, content: 'unknown error', level: 'error' });
```

### Unique log lines

By default, Grafana offers basic support for deduplicating log lines. You can improve the support by adding an `id` field to explicitly assign identifiers to each log line.

**Example:**

```
const frame = new MutableDataFrame({
  refId: query.refId,
  fields: [
    { name: 'time', type: FieldType.time },
    { name: 'content', type: FieldType.string, labels: { filename: 'file.txt' } },
    { name: 'level', type: FieldType.string },
    { name: 'id', type: FieldType.string },
  ],
});

frame.add({ time: 1589189388597, content: 'user registered', level: 'info', id:
'd3b07384d113edec49eaa6238ad5ff00' });
frame.add({ time: 1589189406480, content: 'unknown error', level: 'error', id:
'c157a79031e1c40f85931829bc5fc552' });
```