+++ title = "Backend plugins" keywords = ["grafana", "plugins", "backend", "plugin", "backend-plugins", "documentation"] aliases = ["/docs/grafana/latest/plugins/developing/backend-plugins-guide/"] +++

Backend plugins

Grafana added support for plugins in version 3.0 and this enabled the Grafana community to create panel plugins and data source plugins. It was wildly successful and has made Grafana much more useful as you can integrate it with anything and do any type of custom visualization that you want.

However, one limitation with these plugins are that they execute on the client-side (in the browser) which makes it hard to support certain use cases/features, e.g. enable Grafana Alerting for data sources. Grafana v7.0 adds official support for backend plugins which removes this limitation. At the same time it gives plugin developers the possibility to extend Grafana in new and interesting ways, with code running in the backend (server side).

We use the term *backend plugin* to denote that a plugin has a backend component. Still, normally a backend plugin requires frontend components as well. This is for example true for backend data source plugins which normally need configuration and query editor components implemented for the frontend.

Data source plugins can be extended with a backend component. In the future we plan to support additional types and possibly new kinds of plugins, such as [notifiers for Grafana Alerting]({{< relref "../../../alerting/old-alerting/notifications.md" >}}) and custom authentication to name a few.

Use cases for implementing a backend plugin

The following examples gives you an idea of why you'd consider implementing a backend plugin:

- Enable [Grafana Alerting]({{< relref "../../../alerting" >}}) for data sources.
- Connect to non-HTTP services that normally can't be connected to from a web browser, e.g. SQL database servers.
- Keep state between users, e.g. query caching for data sources.
- Use custom authentication methods and/or authorization checks that aren't supported in Grafana.
- Use a custom data source request proxy, see [Resources]({{< relref "#resources" >}}).

Grafana’s backend plugin system

The Grafana backend plugin system is based on the go-plugin library by HashiCorp. The Grafana server launches each backend plugin as a subprocess and communicates with it over gRPC. This approach has a number of benefits:

- Plugins can’t crash your grafana process: a panic in a plugin doesn’t panic the server.
- Plugins are easy to develop: just write a Go application and run `go build` (or use any other language which supports gRPC).
- Plugins can be relatively secure: The plugin only has access to the interfaces and arguments that are given to it, not to the entire memory space of the process.

Grafana's backend plugin system exposes a couple of different capabilities, or building blocks, that a backend plugin can implement:

- Query data
- Resources
- Health checks
- Collect metrics

Query data

The query data capability allows a backend plugin to handle data source queries that are submitted from a [dashboard]({{< relref "../../../dashboards/_index.md" >}}), [Explore]({{< relref "../../../explore/_index.md" >}}) or [Grafana Alerting]({{< relref "../../../alerting" >}}). The response contains [data frames]({{< relref "../data-frames.md" >}}), which are used to visualize metrics, logs, and traces. The query data capability is required to implement for a backend data source plugin.

Resources

The resources capability allows a backend plugin to handle custom HTTP requests sent to the Grafana HTTP API and respond with custom HTTP responses. Here, the request and response formats can vary, e.g. JSON, plain text, HTML or static resources (files, images) etc. Compared to the query data capability where the response contains data frames, resources give the plugin developer a lot of flexibility for extending and open up Grafana for new and interesting use cases.

Examples of use cases for implementing resources:

- Implement a custom data source proxy in case certain authentication/authorization or other requirements are required/needed that are not supported in Grafana's [built-in data proxy]({{< relref "../../../http_api/data_source.md#data-source-proxy-calls" >}}).
- Return data or information in a format suitable to use within a data source query editor to provide auto-complete functionality.
- Return static resources, such as images or files.
- Send a command to a device, such as a micro controller or IOT device.
- Request information from a device, such as a micro controller or IOT device.
- Extend Grafana's HTTP API with custom resources, methods and actions.
- Use chunked transfer encoding to return large data responses in chunks or to enable "basic" streaming capabilities.

Health checks

The health checks capability allows a backend plugin to return the status of the plugin. For data source backend plugins the health check will automatically be called when you do *Save & Test* in the UI when editing a data source. A plugin's health check endpoint is exposed in the Grafana HTTP API and allows external systems to continuously poll the plugin's health to make sure it's running and working as expected.

Collect metrics

A backend plugin can collect and return runtime, process and custom metrics using the text-based Prometheus exposition format. If you're using the [Grafana Plugin SDK for Go]({{< relref "grafana-plugin-sdk-for-go.md" >}}) to implement your backend plugin, then the Prometheus instrumentation library for Go applications is built-in, and gives you Go runtime metrics and process metrics out of the box. By using the Prometheus instrumentation library you can add custom metrics to instrument your backend plugin.

A metrics endpoint (`/api/plugins/<plugin id>/metrics`) for a plugin is available in the Grafana HTTP API and allows a Prometheus instance to be configured to scrape the metrics.