Backend plugins

The Grafana plugin system for backend development allows you to integrate Grafana with virtually anything and oler custom visualizations. The system is based on HashiCorp's Go Plugin System over RPC. Our implementation of the Grafana server launches each backend plugin as a subprocess and communicates with it over gRPC.

This document explains the system's background, use cases, benefits, and key features.

Background

Grafana added support for *frontend plugins* in version 3.0 so that the Grafana community could create custom panels and data sources. It was wildly successful and has made Grafana much more useful for our user community.

However, one limitation of these plugins is that they run on the client side, in the browser. Therefore, they can't support use cases that require server-side features.

Since Grafana v7.0, we have supported server-side plugins that remove this limitation. We use the term *backend plugin* to denote that a plugin has a backend component. A backend plugin usually requires frontend components as well. For example, some backend data source plugins need query editor components on the frontend.

Use cases for implementing a backend plugin

The following examples give some common use cases for backend plugins:

- Support Grafana Alerting, Recorded Queries and Query and resource caching for data sources.
- Connect to SQL database servers and other non-HTTP services that normally can't be connected to from a browser.
- Keep state between users, for example, by implementing custom 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 (refer to Resources for more information).

Benefits for plugin development

Grafana's approach has benefits for developers:

- **Stability:** Plugins can't crash your Grafana process: a panic in a plugin doesn't panic the server.
- **Ease of development:** Grafana provides an olicially supported SDK for Go and tooling to help create plugins.
- **Security:** Plugins only have access to the interfaces and arguments given to them, not to the entire memory space of the process.

Capabilities of the backend plugin system

Grafana's backend plugin system exposes several key capabilities, or building blocks, that your backend plugin can implement:

- Query data
- Resources
- Health checks
- Collect metrics
- Streaming

Query data

The query data capability allows a backend plugin to handle data source queries that are submitted from a dashboard, Explore or Grafana Alerting. The response contains data frames, which are used to visualize metrics, logs, and traces.

note

Backend data source plugins are required to implement the query data capability.

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. For example, you can use JSON, plain text, HTML, or static resources such as images and files, and so on.

Compared to the query data capability, where the response contains data frames, the resources capability gives the plugin developer more flexibility for extending and opening up Grafana for new and interesting use cases.

Use cases for implementing resources:

- Implement a custom data source proxy to provide certain authentication, authorization, or other requirements that are not supported in Grafana's built-in data proxy.
- Return data or information in a format suitable for 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 microcontroller or IoT device.
- Request information from a device, such as a microcontroller 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 certain 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 is automatically called when a user edits a data source and selects *Save & Test* in the UI.

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 that 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 to implement your backend plugin, then the Prometheus instrumentation library for Go applications is built-in. This SDK gives you Go runtime metrics and process metrics out of the box. To add custom metrics to instrument your backend plugin, refer to Implement metrics in your plugin.

Streaming

The streaming capability allows a backend plugin to handle data source queries that are streaming. For more information, refer to the tutorial for a streaming data source plugin.

Data communication model

Grafana uses a communication model where you can opt in to instance management to simplify the development process. If you do, then all necessary information (configuration) is provided in each request to a backend plugin, allowing the plugin to fulfill the request and return a response. This model simplifies for plugin authors not having to keep track of or request additional state to fulfill a request.

Caching and connection pooling

Grafana provides instance management in the backend plugin SDK to ease working with multiple configured Grafana data sources or apps, referred to as instances. This allows a plugin to simply keep state cleanly separated between instances. The SDK makes sure to optimize plugin resources by caching said instances in memory until their configuration changes in Grafana. Refer to the HTTP Backend plugin example or the App with backend example, which shows how to use the instance management for data source and app plugins.

Mentioned instance state is especially useful for holding client connections to downstream servers, such as HTTP, gRPC, TCP, UDP, and so on, to enable usage of connection pooling that optimizes usage and connection reuse to a downstream server. By using connection pooling, the plugin avoids using all of the machine's available TCP connections.

For an example of a plugin supporting connection pooling, refer to the HTTP Backend plugin example, which shows each plugin instance creating an HTTP client that will be reused throughout the lifetime of the instance and thereby reuse HTTP connections.