+++ title = "Legacy plugins" aliases = ["/docs/grafana/latest/plugins/development/", "/docs/grafana/next/plugins/datasources/", "/docs/grafana/next/plugins/apps/", "/docs/grafana/next/plugins/panels/", "/docs/grafana/next/plugins/developing/development/"] +++

Legacy plugins

> ***Note:*** *Since Grafana 7.0, writing plugins using Angular is no longer recommended. If you're looking to build a new plugin, refer to [Plugins]({{< relref "../_index.md" >}}).*

You can extend Grafana by writing your own plugins and then share them with other users in our plugin repository.

Grafana already has a strong community of contributors and plugin developers. By making it easier to develop and install plugins with resources such as this guide, we hope that the community can grow even stronger and develop new plugins that we would never think about.

Short version

1. Set up Grafana
2. Clone an example plugin into `/var/lib/grafana/plugins` or `data/plugins` (relative to grafana git repo if you're running development version from source dir)
3. Use one of our example plugins as a starting point

Example plugins

- "Hello World" panel using Angular
- "Hello World" panel using React
- Simple json data source
- Clock panel
- Pie chart panel

You might also be interested in the available tutorials around authoring a plugin.

- Grafana Tutorials

What languages?

Since everything turns into JavaScript, it's up to you to choose which language you want. That said, it's probably a good idea to choose es6 or TypeScript, because we use es6 classes in Grafana. So it's easier to get inspiration from the Grafana repo if you choose one of those languages.

Buildscript

You can use any build system that supports systemjs. All the built content should end up in a folder named `dist` and be committed to the repository. By committing the dist folder, the person who installs your plugin does not have to run any build script. All of our example plugins have a build script configured.

Keep your plugin up to date

New versions of Grafana can sometimes cause plugins to break. Check out our PLUGIN_DEV.md doc for changes in Grafana that can impact your plugin.

Metadata

See the [coding styleguide]({{< relref "style-guide.md" >}}) for details on the metadata.

module.(js|ts)

This is the entry point for every plugin. This is the place where you should export your plugin implementation. Depending on what kind of plugin you are developing you will be expected to export different things. You can find what's expected for [datasource]({{< relref "data-sources.md" >}}), [panels]({{< relref "panels.md" >}}) and [apps]({{< relref "apps.md" >}}) plugins in the documentation.

The Grafana SDK is quite small so far and can be found here:

- SDK file in Grafana

The SDK contains three different plugin classes: PanelCtrl, MetricsPanelCtrl and QueryCtrl. For plugins of the panel type, the module.js file should export one of these. There are some extra classes for [data sources]({{< relref "data-sources.md" >}}).

Example:

```
import { ClockCtrl } from './clock_ctrl';

export { ClockCtrl as PanelCtrl };
```

The module class is also where css for the dark and light themes is imported:

```
import { loadPluginCss } from 'app/plugins/sdk';
import WorldmapCtrl from './worldmap_ctrl';

loadPluginCss({
  dark: 'plugins/grafana-worldmap-panel/css/worldmap.dark.css',
  light: 'plugins/grafana-worldmap-panel/css/worldmap.light.css',
});

export { WorldmapCtrl as PanelCtrl };
```

Start developing your plugin

There are three ways that you can start developing a Grafana plugin.

1. Set up a Grafana development environment. (described here) and place your plugin in the `data/plugins` folder.
2. Install Grafana and place your plugin in the plugins directory which is set in your config file. By default this is `/var/lib/grafana/plugins` on Linux systems.
3. Place your plugin directory anywhere you like and specify it grafana.ini.

We encourage people to set up the full Grafana environment so that you can get inspiration from the rest of the Grafana code base.

When Grafana starts, it scans the plugin folders and mounts every folder that contains a plugin.json file unless the folder contains a subfolder named dist. In that case, Grafana mounts the dist folder instead. This makes it possible to have both built and src content in the same plugin Git repo.

Grafana Events

There are a number of Grafana events that a plugin can hook into:

- `init-edit-mode` can be used to add tabs when editing a panel
- `panel-teardown` can be used for clean up
- `data-received` is an event in that is triggered on data refresh and can be hooked into
- `data-snapshot-load` is an event triggered to load data when in snapshot mode.
- `data-error` is used to handle errors on dashboard refresh.

If a panel receives data and hooks into the `data-received` event then it should handle snapshot mode too. Otherwise the panel will not work if saved as a snapshot. [Getting Plugins to work in Snapshot Mode]({{< relref "snapshot-mode.md" >}}) describes how to add support for this.

Examples

We have three different examples that you can fork/download to get started developing your Grafana plugin.

- simple-json-datasource (small data source plugin for querying json data from backends)
- simple-app-plugin
- clock-panel
- singlestat-panel
- piechart-panel

Other Articles

- [Getting Plugins to work in Snapshot Mode]({{< relref "snapshot-mode.md" >}})
- [Plugin Defaults and Editor Mode]({{< relref "defaults-and-editor-mode.md" >}})
- [Grafana Plugin Code Styleguide]({{< relref "style-guide.md" >}})
- [Grafana Apps]({{< relref "apps.md" >}})
- [Grafana Data Sources]({{< relref "data-sources.md" >}})
- [plugin.json Schema]({{< relref "../metadata.md" >}})