> ***WARNING: @grafana/toolkit is currently in BETA***.

grafana-toolkit

grafana-toolkit is a CLI that enables efficient development of Grafana plugins. We want to help our community focus on the core value of their plugins rather than all the setup required to develop them.

Getting started

Set up a new plugin with `grafana-toolkit plugin:create` command:

```
npx @grafana/toolkit plugin:create my-grafana-plugin
cd my-grafana-plugin
yarn install
yarn dev
```

> ***Note:*** *If running NPM 7+ the* `npx` *commands mentioned in this article may hang. The workaround is to use* `npx --legacy-peer-deps <command to run>`.

Update your plugin to use grafana-toolkit

Follow the steps below to start using grafana-toolkit in your existing plugin.

1. Add `@grafana/toolkit` package to your project by running `yarn add @grafana/toolkit` or `npm install @grafana/toolkit`.
2. Create `tsconfig.json` file in the root dir of your plugin and paste the code below:

```
{
  "extends": "./node_modules/@grafana/toolkit/src/config/tsconfig.plugin.json",
  "include": ["src", "types"],
  "compilerOptions": {
    "rootDir": "./src",
    "baseUrl": "./src",
    "typeRoots": ["./node_modules/@types"]
  }
}
```

3. Create `.prettierrc.js` file in the root dir of your plugin and paste the code below:

```
module.exports = {

...require('./node_modules/@grafana/toolkit/src/config/prettier.plugin.config.json'),

};
```

4. In your `package.json` file add following scripts:

```
"scripts": {
  "build": "grafana-toolkit plugin:build",
  "test": "grafana-toolkit plugin:test",
  "dev": "grafana-toolkit plugin:dev",
  "watch": "grafana-toolkit plugin:dev --watch"
},
```

Usage

With grafana-toolkit, we give you a CLI that addresses common tasks performed when working on Grafana plugin:

- `grafana-toolkit plugin:create`
- `grafana-toolkit plugin:dev`
- `grafana-toolkit plugin:test`
- `grafana-toolkit plugin:build`
- `grafana-toolkit plugin:sign`

Create your plugin

`grafana-toolkit plugin:create plugin-name`

This command creates a new Grafana plugin from template.

If `plugin-name` is provided, then the template is downloaded to `./plugin-name` directory. Otherwise, it will be downloaded to the current directory.

Develop your plugin

`grafana-toolkit plugin:dev`

This command creates a development build that's easy to play with and debug using common browser tooling.

Available options:

- `-w` , `--watch` - run development task in a watch mode

Test your plugin

`grafana-toolkit plugin:test`

This command runs Jest against your codebase.

Available options:

- `--watch` - Runs tests in interactive watch mode.
- `--coverage` - Reports code coverage.
- `-u` , `--updateSnapshot` - Performs snapshots update.
- `--testNamePattern=<regex>` - Runs test with names that match provided regex (https://jestjs.io/docs/en/cli#testnamepattern-regex).
- `--testPathPattern=<regex>` - Runs test with paths that match provided regex (https://jestjs.io/docs/en/cli#testpathpattern-regex).
- `--maxWorkers=<num>|<string>` - Limit number of Jest workers spawned (https://jestjs.io/docs/en/cli#--maxworkersnumstring)

Build your plugin

`grafana-toolkit plugin:build`

This command creates a production-ready build of your plugin.

Available options:

- `--skipTest` - Skip running tests as part of build. Useful if you're running the build as part of a larger pipeline.
- `--skipLint` - Skip linting as part of build. Useful if you're running the build as part of a larger pipeline.
- `--coverage` - Reports code coverage after the test step of the build.
- `--preserveConsole` - Preserves console statements in the code.

Sign your plugin

`grafana-toolkit plugin:sign`

This command creates a signed MANIFEST.txt file which Grafana uses to validate the integrity of the plugin.

Available options:

- `--signatureType` - The type of Signature you are generating: `private`, `community` or `commercial`
- `--rootUrls` - For private signatures, a list of the Grafana instance URLs that the plugin will be used on

To generate a signature, you will need to sign up for a free account on https://grafana.com, create an API key with the Plugin Publisher role, and pass that in the `GRAFANA_API_KEY` environment variable.

FAQ

Which version of grafana-toolkit should I use?

See Grafana packages versioning guide.

What tools does grafana-toolkit use?

grafana-toolkit comes with TypeScript, ESLint, Prettier, Jest, CSS and SASS support.

How to start using grafana-toolkit in my plugin?

See Updating your plugin to use grafana-toolkit.

Can I use TypeScript to develop Grafana plugins?

Yes! grafana-toolkit supports TypeScript by default.

How can I test my plugin?

grafana-toolkit comes with Jest as a test runner.

Internally at Grafana we use Enzyme. If you are developing React plugin and you want to configure Enzyme as a testing utility, then you need to configure `enzyme-adapter-react`. To do so, create `<YOUR_PLUGIN_DIR>/config/jest-setup.ts` file that will provide necessary setup. Copy the following code into that file to get Enzyme working with React:

```
import { configure } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

configure({ adapter: new Adapter() });
```

You can also set up Jest with shims of your needs by creating `jest-shim.ts` file in the same directory: `<YOUR_PLUGIN_DIR_>/config/jest-shim.ts`

Can I provide custom setup for Jest?

You can provide custom Jest configuration with a `package.json` file. For more details, see Jest docs.

Currently we support following Jest configuration properties:

- `snapshotSerializers`
- `moduleNameMapper`

How can I customize Webpack rules or plugins?

You can provide your own `webpack.config.js` file that exports a `getWebpackConfig` function. We recommend that you extend the standard configuration, but you are free to create your own:

```
const CustomPlugin = require('custom-plugin');

module.exports.getWebpackConfig = (config, options) => ({
  ...config,
  plugins: [...config.plugins, new CustomPlugin()],
});
```

How can I style my plugin?

We support pure CSS, SASS, and CSS-in-JS approach (via Emotion).

Single CSS or SASS file

Create your CSS or SASS file and import it in your plugin entry point (typically `module.ts`):

```
import 'path/to/your/css_or_sass';
```

The styles will be injected via `style` tag during runtime.

> ***Note:*** *that imported static assets will be inlined as base64 URIs. This can be subject of change in the future!*

Theme-specific stylesheets

If you want to provide different stylesheets for dark/light theme, then create `dark.[css|scss]` and `light.[css|scss]` files in the `src/styles` directory of your plugin. grafana-toolkit generates theme-specific stylesheets that are stored in `dist/styles` directory.

In order for Grafana to pick up your theme stylesheets, you need to use `loadPluginCss` from `@grafana/runtime` package. Typically you would do that in the entry point of your plugin:

```
import { loadPluginCss } from '@grafana/runtime';

loadPluginCss({
  dark: 'plugins/<YOUR-PLUGIN-ID>/styles/dark.css',
  light: 'plugins/<YOUR-PLUGIN-ID>/styles/light.css',
});
```

You must add `@grafana/runtime` to your plugin dependencies by running `yarn add @grafana/runtime` or `npm install @grafana/runtime`.

> ***Note:*** *that in this case static files (png, svg, json, html) are all copied to dist directory when the plugin is bundled. Relative paths to those files does not change!*

Emotion

Starting from Grafana 6.2 *our suggested way* for styling plugins is by using Emotion. It's a CSS-in-JS library that we use internally at Grafana. The biggest advantage of using Emotion is that you can access Grafana Theme variables.

To start using Emotion, you first must add it to your plugin dependencies:

```
yarn add "emotion"@10.0.27
```

Then, import `css` function from Emotion:

```
import { css } from 'emotion';
```

Now you are ready to implement your styles:

```
const MyComponent = () => {
  return (
    <div
      className={css`
        background: red;
      `}
    />
  );
};
```

To learn more about using Grafana theme please refer to Theme usage guide

> *We do not support Emotion's `css` prop. Use className instead!*

Can I adjust TypeScript configuration to suit my needs?

Yes! However, it's important that your `tsconfig.json` file contains the following lines:

```
{
  "extends": "./node_modules/@grafana/toolkit/src/config/tsconfig.plugin.json",
  "include": ["src"],
  "compilerOptions": {
    "rootDir": "./src",
    "typeRoots": ["./node_modules/@types"]
  }
}
```

Can I adjust ESLint configuration to suit my needs?

grafana-toolkit comes with default config for ESLint. For now, there is no way to customise ESLint config.

How is Prettier integrated into grafana-toolkit workflow?

When building plugin with `grafana-toolkit plugin:build` task, grafana-toolkit performs Prettier check. If the check detects any Prettier issues, the build will not pass. To avoid such situation we suggest developing plugin with `grafana-toolkit plugin:dev --watch` task running. This task tries to fix Prettier issues automatically.

My editor does not respect Prettier config, what should I do?

In order for your editor to pick up our Prettier config you need to create `.prettierrc.js` file in the root directory of your plugin with following content:

```
module.exports = {

...require('./node_modules/@grafana/toolkit/src/config/prettier.plugin.config.json'),

};
```

How do I add third-party dependencies that are not npm packages?

Put them in the `static` directory in the root of your project. The `static` directory is copied when the plugin is built.

I am getting this message when I run yarn install: `Request failed \"404 Not Found\"`

If you are using version `canary`, this error occurs because a `canary` release unpublishes previous versions leaving `yarn.lock` outdated. Remove `yarn.lock` and run `yarn install` again.

I am getting this message when I run my plugin: `Unable to dynamically transpile ES module A loader plugin needs to be configured via SystemJS.config({ transpiler: 'transpiler-module' }).`

This error occurs when you bundle your plugin using the `grafana-toolkit plugin:dev` task and your code comments include ES2016 code.

There are two issues at play:

- The `grafana-toolkit plugin:dev` task does not remove comments from your bundled package.
- Grafana does not support ES modules.

If your comments include ES2016 code, then SystemJS v0.20.19, which Grafana uses internally to load plugins, interprets your code as an ESM and fails.

To fix this error, remove the ES2016 code from your comments.

I would like to dynamically import modules in my plugin

Create a webpack.config.js with this content (in the root of *your* plugin)

```
// webpack.config.js
const pluginJson = require('./src/plugin.json');
module.exports.getWebpackConfig = (config, options) => ({
  ...config,
  output: {
    ...config.output,
    publicPath: `public/plugins/${pluginJson.id}/`,
  },
});
```

The plugin id is the id written in the plugin.json file.

Contribute to grafana-toolkit

You can contribute to grafana-toolkit by helping develop it or by debugging it.

Develop grafana-toolkit

Typically plugins should be developed using the `@grafana/toolkit` installed from npm. However, when working on the toolkit, you might want to use the local version. Follow the steps below to develop with a local version:

1. Clone Grafana repository.
2. Navigate to the directory you have cloned Grafana repo to and then run `yarn install --immutable`.
3. Navigate to `<GRAFANA_DIR>/packages/grafana-toolkit` and then run `yarn link`.
4. Navigate to the directory where your plugin code is and then run `npx grafana-toolkit plugin:dev --yarnlink`. This adds all dependencies required by grafana-toolkit to your project, as well as link your local grafana-toolkit version to be used by the plugin.

Debug grafana-toolkit

To debug grafana-toolkit you can use standard NodeJS debugging methods (`node --inspect`, `node --inspect-brk`).

To run grafana-toolkit in a debugging session use the following command in the toolkit's directory:

```
node --inspect-brk ./bin/grafana-toolkit.js [task]
```

To run linked grafana-toolkit in a debugging session use the following command in the plugin's directory:

```
node --inspect-brk ./node_modules/@grafana/toolkit/bin/grafana-toolkit.js [task]
```