Developer guide

This guide helps you get started developing Grafana.

Before you begin, you might want to read How to contribute to Grafana as a junior dev by Ivana Huckova.

Dependencies

Make sure you have the following dependencies installed before setting up your developer environment:

- Git
- Go (see go.mod for minimum required version)
- Node.js (Long Term Support)
- Yarn

macOS

We recommend using Homebrew for installing any missing dependencies:

```
brew install git
brew install go
brew install node@16
npm install -g yarn
```

Windows

If you are running Grafana on Windows 10, we recommend installing the Windows Subsystem for Linux (WSL). For installation instructions, refer to our Grafana setup guide for Windows environment.

Download Grafana

We recommend using the Git command-line interface to download the source code for the Grafana project:

1. Open a terminal and run `git clone https://github.com/grafana/grafana.git`. This command downloads Grafana to a new `grafana` directory in your current directory.
2. Open the `grafana` directory in your favorite code editor.

For alternative ways of cloning the Grafana repository, please refer to GitHub's cloning a repository documentation.

Warning: Do not use `go get` to download Grafana. Recent versions of Go have added behavior which isn't compatible with the way the Grafana repository is structured.

Configure Editors

For some IDEs, additional configuration may be needed for Typescript to work with Yarn plug'n'play. For VSCode and Vim, it's as easy as running `yarn dlx @yarnpkg/sdks vscode` or `yarn dlx @yarnpkg/sdks vim`, respectively.

More information can be found here.

Build Grafana

Grafana consists of two components; the *frontend*, and the *backend*.

Frontend

Before we can build the frontend assets, we need to install the dependencies:

```
yarn install --immutable
```

After the command has finished, we can start building our source code:

```
yarn start
```

Once `yarn start` has built the assets, it will continue to do so whenever any of the files change. This means you don't have to manually build the assets every time you change the code.

Next, we'll build the web server that will serve the frontend assets we just built.

Backend

Build and run the backend by running `make run` in the root directory of the repository. This command compiles the Go source code and starts a web server.

> Are you having problems with too many open files?

By default, you can access the web server at `http://localhost:3000/`.

Log in using the default credentials:

username	password
`admin`	`admin`

When you log in for the first time, Grafana asks you to change your password.

Building on Windows The Grafana backend includes SQLite which requires GCC to compile. So in order to compile Grafana on Windows you need to install GCC. We recommend TDM-GCC. Eventually, if you use Scoop, you can install GCC through that.

You can build the back-end as follows:

1. Follow the instructions to install the Wire tool.
2. Generate code using Wire:

```
# Normally Wire tool installed at $GOPATH/bin/wire.exe
<Wire tool install path> gen -tags oss ./pkg/server ./pkg/cmd/grafana-cli/runner
```

3. Build the Grafana binaries:

```
go run build.go build
```

The Grafana binaries will be in bin\windows-amd64. Alternately, if you wish to use the `make` command, install Make for Windows and use it in a Unix shell (f.ex. Git Bash).

Test Grafana

The test suite consists of three types of tests: *Frontend tests*, *backend tests*, and *end-to-end tests*.

Run frontend tests

We use jest for our frontend tests. Run them using Yarn:

```
yarn test
```

Run backend tests

If you're developing for the backend, run the tests with the standard Go tool:

```
go test -v ./pkg/...
```

On Windows Running the backend tests on Windows currently needs some tweaking, so use the build.go script:

```
go run build.go test
```

Run PostgreSQL and MySQL integration tests

To run PostgreSQL and MySQL integration tests locally, you need to start the docker blocks for MySQL and/or PostgreSQL test data sources by running `make devenv sources=mysql_tests,postgres_tests`. When your test data sources are running, you can execute integration tests by running:

```
GRAFANA_TEST_DB=mysql go test -covermode=atomic -tags=integration ./pkg/...
```

and/or

```
GRAFANA_TEST_DB=postgres go test -covermode=atomic -tags=integration ./pkg/...
```

Run end-to-end tests

The end to end tests in Grafana use Cypress to run automated scripts in a headless Chromium browser. Read more about our e2e framework.

To run the tests:

```
yarn e2e
```

By default, the end-to-end tests starts a Grafana instance listening on `localhost:3001`. To use a specific URL, set the `BASE_URL` environment variable:

```
BASE_URL=http://localhost:3333 yarn e2e
```

To follow the tests in the browser while they're running, use the `yarn e2e:debug`.

```
yarn e2e:debug
```

If you want to pick a test first, use the `yarn e2e:dev`, to pick a test and follow the test in the browser while it runs.

```
yarn e2e:dev
```

Configure Grafana for development

The default configuration, `defaults.ini`, is located in the `conf` directory.

To override the default configuration, create a `custom.ini` file in the `conf` directory. You only need to add the options you wish to override.

Enable the development mode, by adding the following line in your `custom.ini`:

```
app_mode = development
```

Add data sources

By now, you should be able to build and test a change you've made to the Grafana source code. In most cases, you need to add at least one data source to verify the change.

To set up data sources for your development environment, go to the devenv directory in the Grafana repository:

```
cd devenv
```

Run the `setup.sh` script to set up a set of data sources and dashboards in your local Grafana instance. The script creates a set of data sources called **gdev-<type>**, and a set of dashboards located in a folder called **gdev dashboards**.

Some of the data sources require databases to run in the background.

Installing and configuring databases can be a tricky business. Grafana uses Docker to make the task of setting up databases a little easier. Make sure you install Docker before proceeding to the next step.

In the root directory of your Grafana repository, run the following command:

```
make devenv sources=influxdb,loki
```

The script generates a Docker Compose file with the databases you specify as `sources`, and runs them in the background.

See the repository for all the available data sources. Note that some data sources have specific Docker images for macOS, e.g. `nginx_proxy_mac`.

Build a Docker image

To build a Docker image, run:

```
make build-docker-full
```

The resulting image will be tagged as grafana/grafana:dev.

Note: If you are using Docker for macOS, be sure to set the memory limit to be larger than 2 GiB. Otherwise, `grunt build` may fail. The memory limit settings are available under **Docker Desktop** -> **Preferences** -> **Advanced**.

Troubleshooting

Are you having issues with setting up your environment? Here are some tips that might help.

Too many open files when running `make run`

Depending on your environment, you may have to increase the maximum number of open files allowed. For the rest of this section, we will assume you are on a Unix like OS (e.g. Linux/macOS), where you can control the maximum number of open files through the ulimit shell command.

To see how many open files are allowed, run:

```
ulimit -a
```

To change the number of open files allowed, run:

```
ulimit -S -n 4096
```

The number of files needed may be different on your environment. To determine the number of open files needed by `make run`, run:

```
find ./conf ./pkg ./public/views | wc -l
```

Another alternative is to limit the files being watched. The directories that are watched for changes are listed in the `.bra.toml` file in the root directory.

To retain your `ulimit` configuration, i.e. so it will be remembered for future sessions, you need to commit it to your command line shell initialization file. Which file this will be depends on the shell you are using, here are some examples:

- zsh -> ~/.zshrc
- bash -> ~/.bashrc

Commit your ulimit configuration to your shell initialization file as follows ($LIMIT being your chosen limit and $INIT_FILE being the initialization file for your shell):

```
echo ulimit -S -n $LIMIT >> $INIT_FILE
```

Your command shell should read the initialization file in question every time it gets started, and apply your `ulimit` command.

For some people, typically using the bash shell, ulimit fails with an error similar to the following:

```
ulimit: open files: cannot modify limit: Operation not permitted
```

If that happens to you, chances are you've already set a lower limit and your shell won't let you set a higher one. Try looking in your shell initialization files (~/.bashrc typically), if there's already a ulimit command that you can tweak.

Next steps

- Read our style guides.
- Learn how to Create a pull request.
- Read How to contribute to Grafana as a junior dev by Ivana Huckova.
- Read about the architecture.