Merge a pull request

When a pull request has been reviewed and approved by at least one person and all checks have passed it's time to merge the pull request.

Who is expected to merge a pull request?

Maintainers are responsible for merging all pull requests. If a maintainer has opened a pull request the general rule is that the same maintainer merges the pull request. If a non-maintainer has opened a pull request it's suggested that one of the maintainers reviewing the pull request merges the pull request.

Checklist/summary

The following checklist/summary should give you a quick overview of what to ask/consider before merging a pull request.

- Reviewed and approved?
- All checks passed?
- Proper pull request title?
- Milestone assigned?
- Add to changelog/release notes?
- Needs backporting?

Before merge

Before actually merging a pull request there's a couple of things to take into consideration.

Status checks

Before you can merge a pull request, it must have a review approval, and all the required status checks must pass.

Format the pull request title

The pull request title should be formatted according to `<Area>: <Summary>`. Keep the summary short and understandable for the community as a whole.

All commits in a pull request are squashed when merged and the pull request title will be the default subject line of the squashed commit message. It's also used for changelog/release notes.

See formatting guidelines for more information.

Assign a milestone

A milestone **should** be added to every pull request. Several things in the Grafana release process requires at least pull requests to be in a milestone, for example generating changelog/release notes.

This makes it easier to track what changes go into a certain release. Without this information, release managers have to go through git commits which is not an efficient process.

Include in changelog and release notes?

At Grafana we generate the changelog and release notes based on merged pull requests. Including changes in the changelog/release notes is very important to provide a somewhat complete picture of what changes a Grafana release actually includes.

Exactly what changes should be added to the changelog is hard to answer but some general guidance would be any change that you think would be interesting for the community as a whole. Use your best judgement and/or ask other maintainers for advice.

There's a GitHub action available in the repository named Update changelog that can manually be triggered to re-generate the changelog and release notes for any release.

To include a pull request in the changelog/release notes the general rule of thumb is that a milestone should be assigned and labeled with `add to changelog`.

The changelog/release notes are divided into sections and here's a description of how you make a pull request show up in a certain section.

Features and enhancements:

Milestone assigned and labeled with `add to changelog` and any of the other section rules don't apply.

Bug fixes:

Milestone assigned and labeled with `add to changelog` and either labeled with `type/bug` or the pull request title contains `fix` or `fixes`.

Plugin development fixes & changes:

Milestone assigned and labeled with `area/grafana/toolkit`, `area/grafana/ui` or `area/grafana/runtime`.

Deprecations:

In case the pull request introduces a deprecation you should document this. Label the pull request with `add to changelog` and use the following template at the end of the pull request description describing the deprecation change.

```
# Deprecation notice

<Deprecation description>
```

Breaking changes:

In case the pull request introduces a breaking change you should document this. Label the pull request with `add to changelog` and `breaking change` and use the following template at the end of the pull request description describing the breaking change.

```
# Release notice breaking change

<Breaking change description>
```

Should the pull request be backported?

An active decision of backporting needs to be taken for every pull request. There's a pull request check named **Backport Check** that will enforce this. By adding/removing labels on the pull request the check will be re-evaluated.

No backport

If you don't want to backport you need to add a label named **no-backport** to the pull request.

Backport

If your pull request has changes that need to go into one or several existing release branches you need to backport the changes. Please refer to Backport PR for detailed instructions.

The general rule of thumb regarding what changes goes into what release is:

- bug fixes should be released in patch releases, e.g. v8.1.3, if the bug was introduced in the same major/minor or lower patch version.
- new features should go into the next major/minor version, e.g. v8.0.0, v8.2.0.

Some examples when backport is required:

- The change needs to be released in the next upcoming patch release, e.g. v8.1.3, so you have to backport it, e.g. into the v8.1.x release branch.
- You have a change to be released in the next major/minor release, e.g. v8.0.0, and there's already a release branch, e.g. v8.0.x, you have to backport it, e.g. into the v8.0.x release branch.
- The change includes documentation changes that needs to be updated for one or multiple older versions, then you have to backport it to each release branch.

Some examples when backport is not required:

- The change is supposed to be released in the next major/minor release, e.g. v8.0.0, but the release branch, e.g. v8.0.x, has not yet been created.

> ***Note:*** *You can still backport a pull request after it's been merged.*

Doing the actual merge

Time to actually merge the pull request changes. All commits in a pull request are squashed, hence the GitHub `Squash and merge` button is used to initialize the merge.

This will present you with options allowing you to optionally change the commit message before merging. Please remember that developers might use the commit information when reviewing changes of files, doing git blame and resolving merge conflicts etc., trying to quickly figure out what the actual change was. But there's not really any best practices around this, the following is an attempt to bring some guidance.

Do:

- Make sure the pull request title is formatted properly before merging, this will automatically give you a good and short summary of the commit/change.
- Leave `Co-authored-by:` lines as is so that co-authors will be accounted for the contribution.
- Remove any commit information that doesn't bring any context to the change.

Consider:

- Add any references to issues that the pull request fixes/closes/references to ease giving quick context to things. Doing this allows cross-reference between the commit and referenced issue(s).

Finalize the merge by clicking on the `Confirm squash and merge` button.

After the merge

Make sure to close any referenced/related issues. It's recommended to assign the same milestone on the issues that the pull request fixes/closes, but not required.