Dataproc Teleport Github repo - Release Automation

# How should the commits be written using Conventional Commits standards?

The commit message should be structured as follows:

**<type>**[optional scope]**: <description>**

[optional body]

[optional footer(s)]

The most important prefixes types you should have in mind are:

* **fix**: which represents bug fixes, and correlates to a SemVer patch.
* **feat**: which represents a new feature, and correlates to a SemVer minor.
* **feat!:** which represents a breaking change (indicated by the **!**) and will result in a SemVer major.
* types other than fix: and feat: are also allowed viz. **build:, chore:, ci:, docs:, refactor:, perf:, test:**

# Example Commit Messages

### [Using **fix:**] Commit message for a patch

fix: Enabled csv support for GCStoGCS template

### [Using **feat:**] Commit message to enable a new feature

feat: Built Python GCStoGCS template for Dataproc

### [Using **feat!:**]Commit message with ! to draw attention to breaking change

feat!: Java & Python Dataproc templates released for customer usage now

### Commit message for a build or regular chore jobs

chore: Updated a link on the README file

build: Completed build for GCStoGCS template

refactor: Updated variable name for GCStoGCS template

### Commit message for integration testing, performance enhancement or automation

test: Integration testing completed for GCStoGCS template

perf: Validated GCStoGCS template with 100GB load

ci: Implemented the Jenkins automation for GCStoGCS template

### Commit message for publishing/documentation

docs: correct spelling of CHANGELOG

### Commit message with multi-paragraph body and multiple footers

fix: prevent racing of requests

Introduce a request id and a reference to latest request. Dismiss

incoming responses other than from latest request.

Remove timeouts which were used to mitigate the racing issue but are

obsolete now.

# What is the Commit-checker bot?

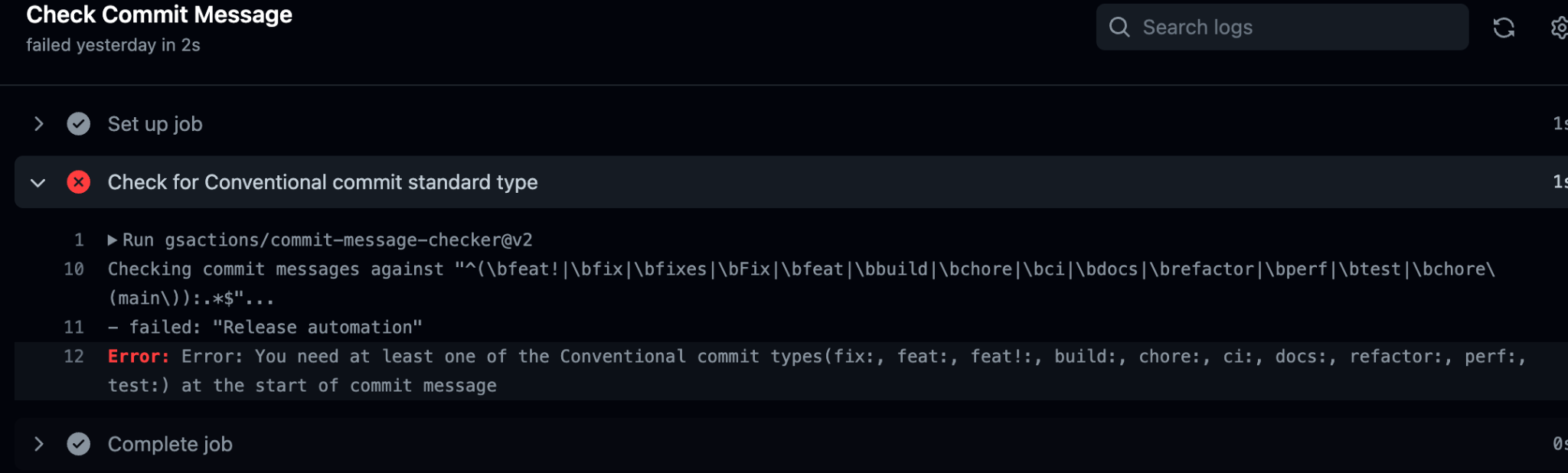
Everytime a PR is created, the Commit-checker bot validates whether the commit message syntax contains the allowed prefix or not. Below image shows the example for a Commit check failed and two Commit checks passed.



Ideally, a PR should not be merged until all the checks are passed

# What to do if Commit-checker check fails?

A Commit-checker bot check fails when the above stated conventional commit standards are not followed while pushing a PR. A detailed error would read something like below:



To fix the error, one can push a commit again with a valid prefix added to the commit message. Once, The same PR gets updated with a valid commit message, the Commit-checker bot gets triggered automatically and will successfully pass the consequent checks.

# Why should conventional commit standards be followed?

Conventional commits help with the ease of readability and understanding of the Git history. With a growing github repo like dataproc-templates, generating the change logs with every new release becomes important.

By using conventional commits, we can now generate the Change Log automatically at each release using a release-please bot. The bot will look at the commits published and the basis of the type prefix mentioned in the commits - it will be able to segregate the features, patches and documentation created in a new release.

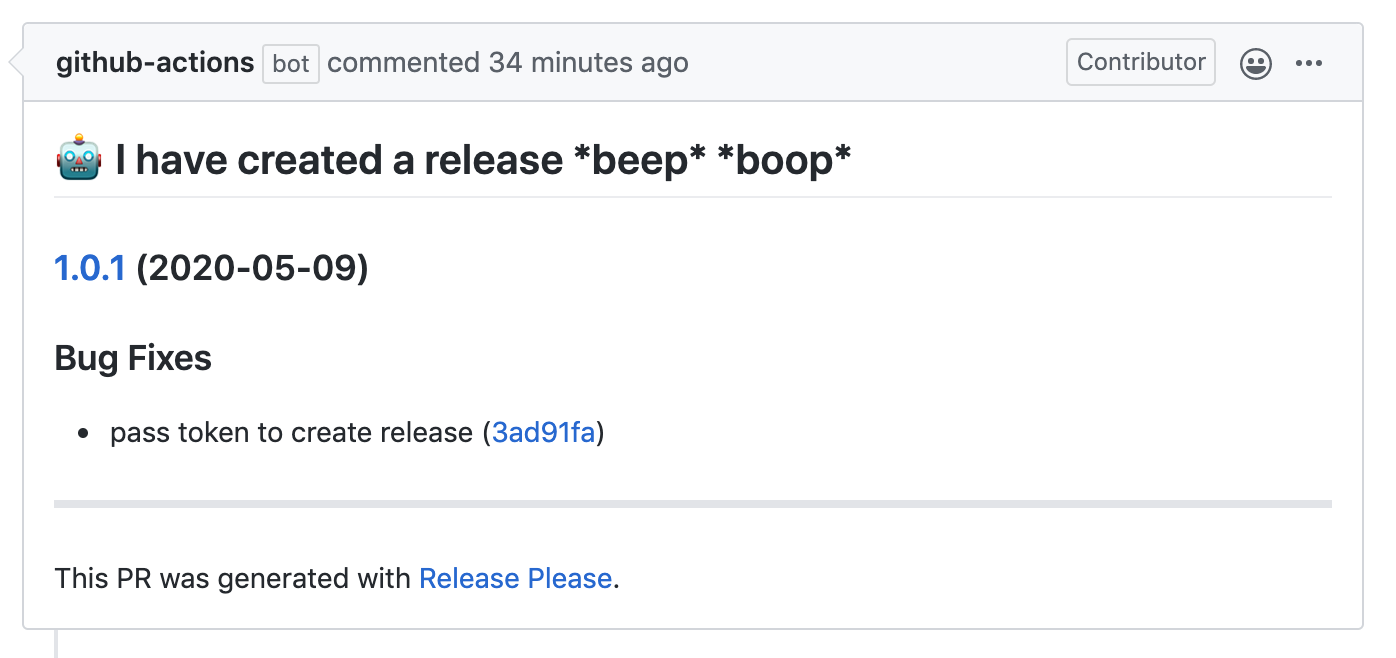
# release-please bot

Release Please assumes you are using Conventional Commit messages. Release Please automates CHANGELOG generation, the creation of GitHub releases, and version bumps for your projects.

It does so by parsing your git history, looking for [Conventional Commit messages](https://www.conventionalcommits.org/), and creating release PRs.

# What's a Release PR?

Rather than continuously releasing what's landed to your default branch, release-please maintains Release PRs:



These Release PRs are kept up-to-date as additional work is merged. When you're ready to tag a release, simply merge the release PR. Both squash-merge and merge commits work with Release PRs.

When the Release PR is merged, release-please takes the following steps:

1. Updates your changelog file (for example CHANGELOG.md)
2. Creates a GitHub Release based on the tag with the version number

# Further Reading:

* Conventional Commits: <https://www.conventionalcommits.org/>
* Release please Documentation: <https://github.com/googleapis/release-please#release-please>
* Semantic Versioning: <https://semver.org/>
* Google Release-please Github Action: <https://github.com/google-github-actions/release-please-action>