GitLab Artifact Tools Documentation

Release 2.2.1

Mike Haboustak

Aug 07, 2022

Contents

The artifact expiration feature provided by GitLab only supports time-based artifact retention policies. GitLab admins can set a global retention period—specified in days, weeks, months, or years—for all jobs that produce artifacts. At the project level, the global retention period can be replaced per-job in the `.gitlab-ci.yml` CI configuration.

From the community's perspective, this method of expiring artifacts is insufficient.

1. Tagged artifacts are removed because there's no easy way to keep tagged artifacts.
2. You can't reliably fetch artifacts for a project because we can't ensure the latest artifacts are kept for projects that build infrequently.
3. The `keep artifacts` safety valve that's used prevent artifact expiration can't be reset. When the `Keep` button is clicked, the artifacts are kept forever.

Glartifacts extends GitLab's artifact expiration feature to better fit real-world artifact retention needs. It defines several artifact *expiration strategies* that can be applied per-project by GitLab administrators. Once applied, artifact expiration **just works**. The job definitions in `.gitlab-ci.yml` do not need to define when their artifacts are saved or removed. Projects that are configured with time-based expiration continue to work, because glartifacts only updates artifacts that aren't already set to expire.

Warning: Using glartifacts to manage artifact expiration breaks GitLab's keep artifacts feature. Clicking the `Keep` button on a job whose artifacts are managed by glartifacts does nothing. In most cases, a sensible artifact retention policy eliminates the need for the `Keep` button, but that may not be true for your projects.

A GitLab enhancement has been proposed to replace the keep artifacts feature with protected artifacts. If you want to use glartifacts, but also need the `Keep` button, please support the linked proposal.

CHAPTER 1

Requirements

Glartifacts requires Python 3.

Be sure you have a GitLab backup before you start. Glartifacts modifies the `ci_builds.artifacts_expire_at` column in the GitLab database. The next execution of the `expire_build_artifacts_worker` background job will **remove** CI artifacts from the database and file system. Once removed, artifacts are non-recoverable.

CHAPTER 2

Installation

Glartifacts can be installed from PyPI using `pip`

```
pip install glartifacts
```

The default installation is configured for a GitLab Omnibus install. See the *Configuration* section for information on adjusting the settings for your environment.

CHAPTER 3

Expiration Strategies

An expiration strategy is a set of rules used to identify artifacts that can be removed. Each strategy starts by finding a point-in-time where a project's artifacts are considered good and should be kept. Artifacts newer than this point-in-time are kept, artifacts that are older are removed.

3.1 Artifact Dispositions

When an expiration strategy is applied to a project, it assigns an artifact disposition to each CI job. The disposition determines the action that should be taken for artifacts associated with that job.

The following table lists the artifact dispositions used by expiration strategies to categorize artifacts.

Table 1: artifact dispositions

Disposition	Description	Action
`good`	Artifacts identified as the point-in-time where artifacts should be kept.	KEEP
`new`	Artifacts newer than the good artifacts point-in-time.	KEEP
`old`	Artifacts older than the good artifacts point-in-time.	REMOVE
`orphaned`	Artifacts whose job (by name) or branch has been removed.	REMOVE
`tagged`	Artifacts from a tagged build.	KEEP
`expiring`	Artifacts with an expiration date set.	IGNORE

Artifacts with the `good` or `new` disposition are newer than the point-in-time identified by the expiration strategy and are kept. Artifacts with the `old` disposition are older than the point-in-time and will be removed.

Artifacts with the `orphaned` disposition are from a branch or job that has been removed. Branches are removed when they are merged upstream. Jobs are removed when they are renamed in or removed from `.gitlab-ci.yml`. In both cases, the artifacts will also be removed.

Artifacts with the `tagged` disposition are never removed. Each tag represents a release of a project that should be kept forever.

Artifacts with the `expiring` disposition have been previously scheduled for removal and are not modified by the expiration strategy.

3.2 LASTGOOD_PIPELINE

The LASTGOOD_PIPELINE strategy uses the most recent successful pipeline as the good artifacts point-in-time. A successful pipeline is identified by a green checkmark in the GitLab CI pipelines view, where all jobs are either successful, canceled, or marked as allow failure.

This strategy ensures that a complete set of artifacts are kept for each branch.

3.3 LASTGOOD_JOB

The LASTGOOD_JOB strategy uses the most recent successful job as the good artifacts point-in-time. A successful job is identified by a green checkmark in the GitLab CI jobs view.

This strategy is more aggressive than LASTGOOD_PIPELINE because it ensures that only the most recent artifacts for each job are kept, even if the overall pipeline fails.

CHAPTER 4

Configuration

Glartifacts requires access to the GitLab database and Gitaly server. The default connection settings are based on a standard Omnibus install, but can be modified for custom deployments via settings in *glartifacts.conf*. You can also override settings per-invocation using environment variables.

The table below lists the available configuration options:

Table 1: glartifacts.conf settings

Section	Option	ENV variable	Default value
postgres	dbname	GLARTIFACTS_DBNAME	gitlabhq_production
postgres	user	GLARTIFACTS_DBUSER	gitlab
postgres	host	GLARTIFACTS_DBHOST	/var/opt/gitlab/postgresql
postgres	port	GLARTIFACTS_DBPORT	5432
gitaly	address	GLARTIFACTS_GITALYADDR	unix:/var/opt/gitlab/gitaly/gitaly.socket

Glartifacts searches for `glartifacts.conf` in `/etc/glartifacts` and `$HOME/.config/glartifacts`. Settings are merged for each conf file found: `User Settings` > `System Settings` > `Default Settings`.

Example glartifacts.conf

```
[postgres]
user = gitlab
host = /var/opt/gitlab/postgresql
port = 5432

[gitaly]
address = unix:/var/opt/gitlab/gitaly/gitaly.socket
```