MANUAL: Continuous Integration Setup with GitLab CI and SonarQube

Introduction: GÉANT Project GitLab

GitLab is an open-source code hosting platform for collaboration and version control. It can be installed on your servers to host your codes privately. GitLab provides Source Code Management (SCM) functionality similar to GitHub and BitBucket.

GÉANT Project GitLab is the instance of GitLab used by the developers in the GÉANT communities and GitLab CI/CD is a tool built into GitLab for software development of CI/CD pipelines.

It allows performing a defined task upon a list of defined actions. By default, every action on the repository triggers a task (commit, merge, tag creation), but the user is entitled to restrict the number of actions.

For the correct trigger of the SonarQube task, GitLab CI uses a "runner" that is already configured in our GÉANT Project GitLab deployment. Momentarily the runner is configured as a "shared" process (*shared runner*). This means that one runner is used to trigger the SonarQube analysis for all GitLab projects that use the CI/CD pipeline. For this, the runner queues all incoming requests and works on them in a sequential way. In the past, not many projects were using SonarQube at the same time. Therefore, we assume that momentarily the usage of the "shared runner" will have no impact on the processing time of your project in SonarQube.

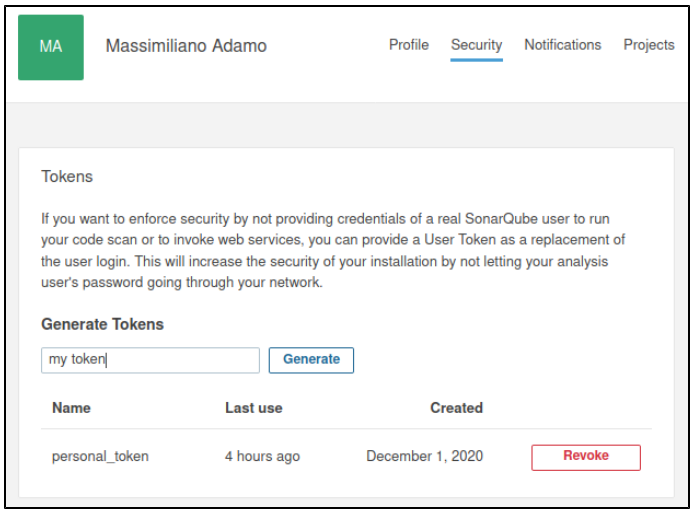
Prerequisites

Create a token in SonarQube

You need to have a token to authorize the CI against SonarQube. In fact, the CI must be able to push the outcome of the scan to SonarQube.

You'll use the token to provide a value for the variable SONAR_LOGIN (see the point: 4.2 Project Variables).

You can create a personal token under the security settings of your SonarQube account (*My Account Security*):

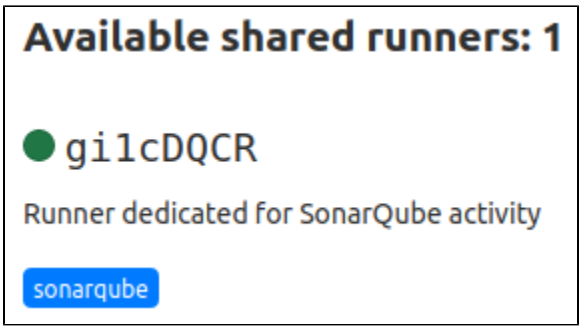


Setup your project in the GÉANT deployment of GitLab

Create your project in the GÉANT Project GitLab deployment.

> **Hint**
>
> As the creator of the project, you usually have the member permissions for the *Owner* role. If you want other team members to access and change the variables and the runner they must be either *Maintainer* or *Owner* of this project. Otherwise, they only need a role authorized to commit and push to the repository (e.g. Developer).

With your member permission set to *Owner* or *Maintainer* the aforementioned "runner" information will be visible to you in the *"Shared runners"* section, similar to the following picture (*Settings CI/CD Runners*):



The tag name is *"sonarqube"* and the runner is intended to be a Docker runner.

Configure your GitLab Project

To enable GitLab to automatically execute the CI/CD runner, some variables need to be defined. Some of them had already been set on a global ("system") level, others must be set by your project.

Global Variables

The following two variables are required. They are already set up globally in GitLab, so that you don't need to define them again.

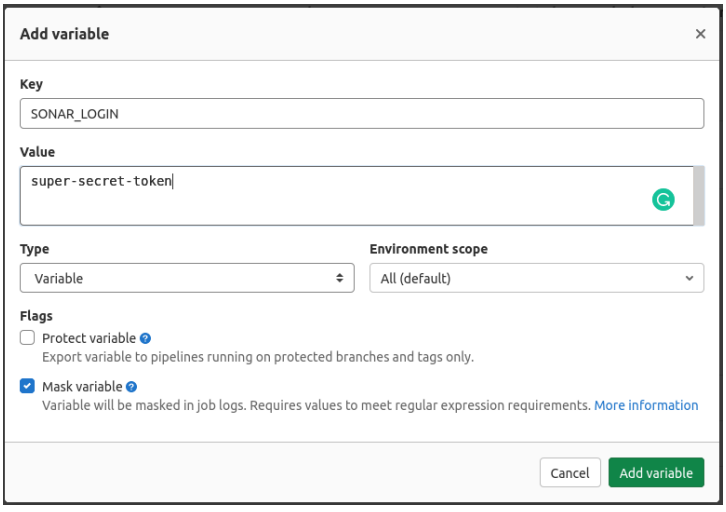
- SONAR_HOST_URL = https://sonarqube.software.geant.org
- SONAR_SCANNER_CLI_VERSION = 4.6

Project Variables

The user will have to set a token variable at the project level (*Settings CI/CD Variables Add Variable* - it is recommended to use a "masked" type of variable).

- SONAR_LOGIN = XXXXX-XXXXX-XXXX

> **Hint**
>
> The value of SONAR_LOGIN must be **identical** to the value of the SonarQube token as defined in Step 2.1 Create a token in SonarQube.



Create the file .gitlab-ci.yml

The file that is responsible to define the appropriate list of tasks that need to be executed after a "git push" is the file *.gitlab-ci.yml*. This file must be placed in the root directory of your project.

If the file is absent, the CI is skipped (i.e.: SonarQube won't be triggered).

Some of the variables used in the YAML file are based on other existing variables already defined before. They are listed to give you a detailed understanding but they don't require any action (definition or adaptation) from your side.

- SONAR_SCANNER_CLI_VERSION (see step 2: it's defined globally).
- CI_PROJECT_DIR: points to the root directory of your project (it's a built-in variable and it cannot be changed).
- CI_PROJECT_NAME: corresponds to the name of the project and this same name will be assigned to the SonarQube project (it's a built-in variable and it cannot be changed).

.gitlab-ci.yml

```
---
stages:
  - sonarqube_stage
sonarqube_ci_stanza:
# free name
  image: sonarsource/sonar-scanner-cli:${SONAR_SCANNER_CLI_VERSION}
  stage:
sonarqube_stage
# free name
  variables:
    SONAR_PROJECT_BASE_DIR: "$CI_PROJECT_DIR"
  script:
    - /usr/bin/entrypoint.sh sonar-scanner -Dsonar.projectKey="$CI_PROJECT_NAME"          # sonar.projectKey
defines the name of the project in SonarQube.

# In this example it uses the same name of the project in GitLab.

tags:
# NOTE: this is NOT a git tag name.
    -
sonarqube
# It is a GitLab tag associated in the runner section (see Step 3.

# Setup your project in the GÉANT deployment of GitLab)
```

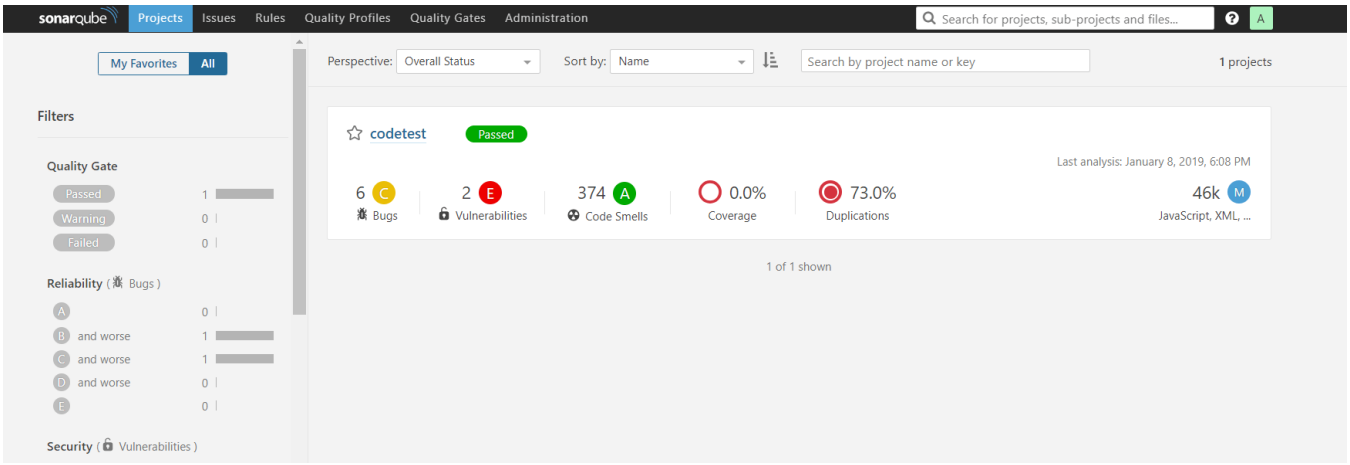
GitLab CI extra settings

The CI will trigger SonarQube on each event by default (commit to every branch, tags creation, merge...). It's up to the user to limit the triggers for the CI or skip the CI on certain branches.

To restrict the triggers, the user can play with the `rules` parameter: https://docs.gitlab.com/ee/ci/yaml/README.html#rules in conjunction with the `if` clause, as explained in the documentation.

Step 3: Check SonarQube dashboard

Also, you can see the project code reports from within SonarQube.



That's it! You have successfully integrated GitLab and SonarQube.

Additional information

sonar-scanner-cli-docker

the following script is run by the container to trigger the SonarQube.

https://github.com/SonarSource/sonar-scanner-cli-docker/blob/master/4/bin/entrypoint.sh

entrypoint.sh

```
#!/bin/bash

set -euo pipefail

declare -a args=()

add_env_var_as_env_prop() {
  if [ "$1" ]; then
    args+=("-D$2=$1")
  fi
}

# if nothing is passed, assume we want to run sonar-scanner
if [[ "$#" == 0 ]]; then
  set -- sonar-scanner
fi

# if first arg looks like a flag, assume we want to run sonar-scanner with flags
if [[ "${1#-}" != "${1}" ]] || [[ -z "$(command -v "${1}")" ]]; then
  set -- sonar-scanner "$@"
fi

if [[ "$1" = 'sonar-scanner' ]]; then
  add_env_var_as_env_prop "${SONAR_LOGIN:-}" "sonar.login"
  add_env_var_as_env_prop "${SONAR_PASSWORD:-}" "sonar.password"
  add_env_var_as_env_prop "${SONAR_PROJECT_BASE_DIR:-}" "sonar.projectBaseDir"
  if [ ${#args[@]} -ne 0 ]; then
    set -- sonar-scanner "${args[@]}" "${@:2}"
  fi
fi

exec "$@"
```

Links

- GÉANT instance of SonarQube
- Information regarding GÉANT instance of GitLab GitLab - Source Code Repository