GitLab

Efficient DevSecOps workflows with reusable CI/CD Components

Chemnitz Linux Days 2025, 2025-03-22

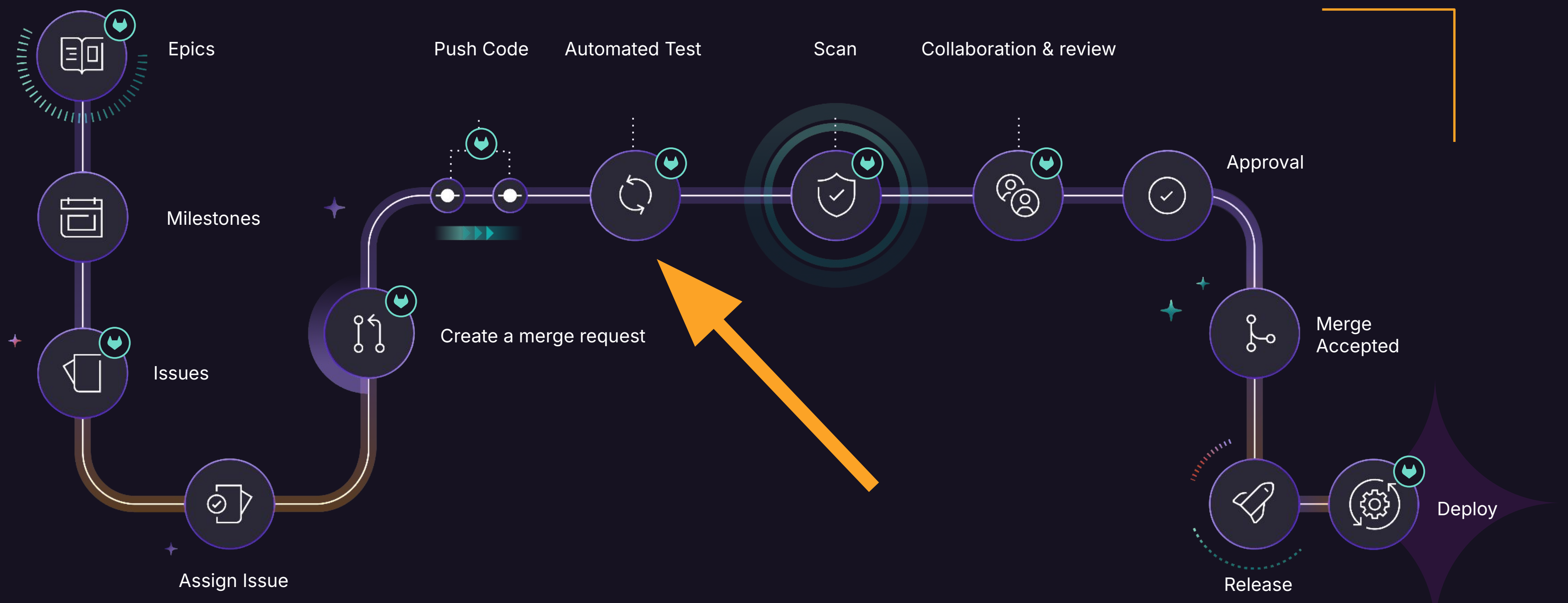
Recording

GitLab

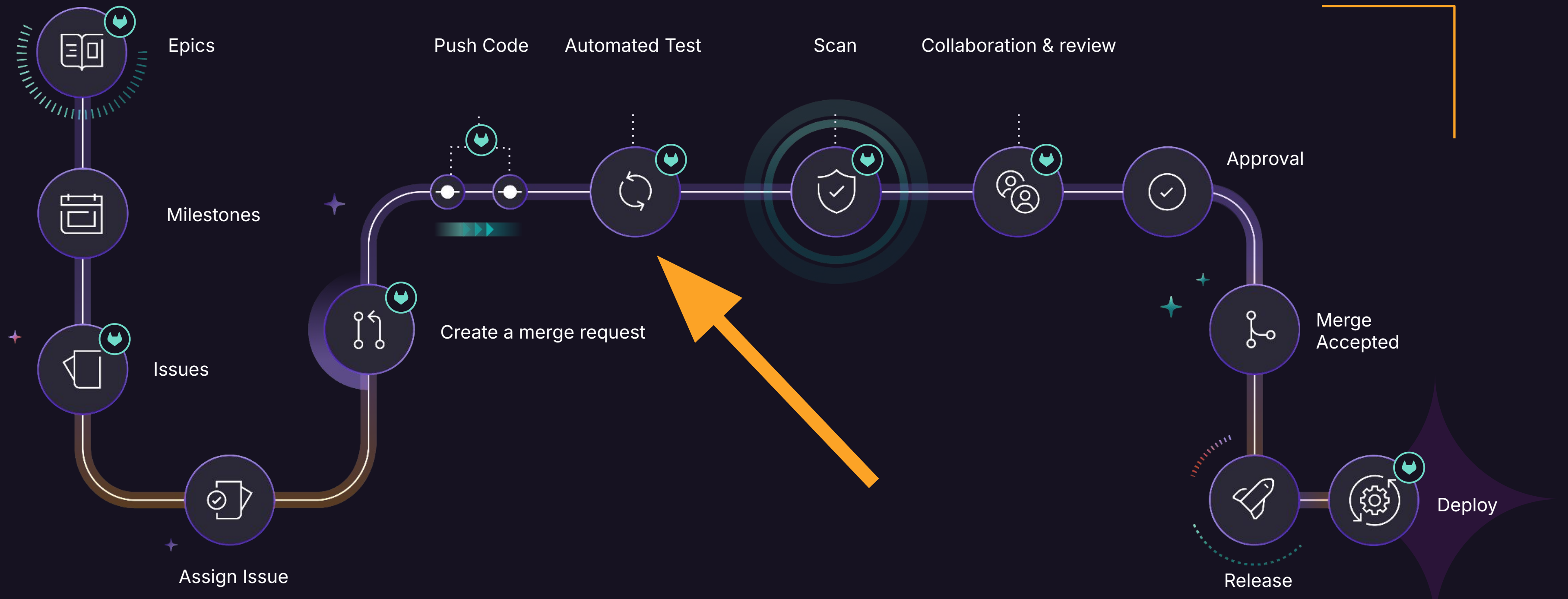
Michael Friedrich

Staff Developer Advocate
@dnsmichi

Where are you in your DevSecOps journey?



What is the most inefficient task?

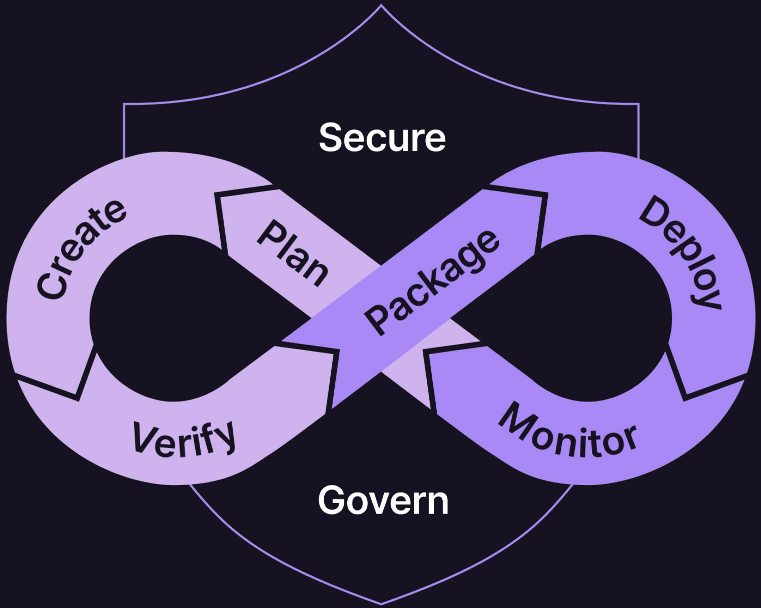


GitLab

Efficient DevSecOps?

DevSecOps is all about speed

- Testing
- Continuous Integration (CI)
- Continuous Deployment (CD)
- Vulnerability Scanning
- Monitoring/Observability
- Infrastructure as Code
- Platform engineering



CI/CD Pipelines

How fast are you able to

- Create a CI/CD pipeline with multiple steps
- Document the usage
- Modularize pipelines
- Create reusable modules for your team
- Optimize and improve modules *without breaking everything*

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"PIPELINE RUNNING"
HEY! GET BACK TO WORK!
PIPELINE RUNNING
OH. CARRY ON.
IMAGE AND FONT BY HTTPS://XKCD.COM

Image generated with https://gitlab.com/gitlab-da/projects/go-excusegen which itself is based on the XKCD comic at https://xkcd.com/303/

Discover, use, share

How to discover?

Internet search
LLM / Docs Chat
Developer portal
Ask your peers

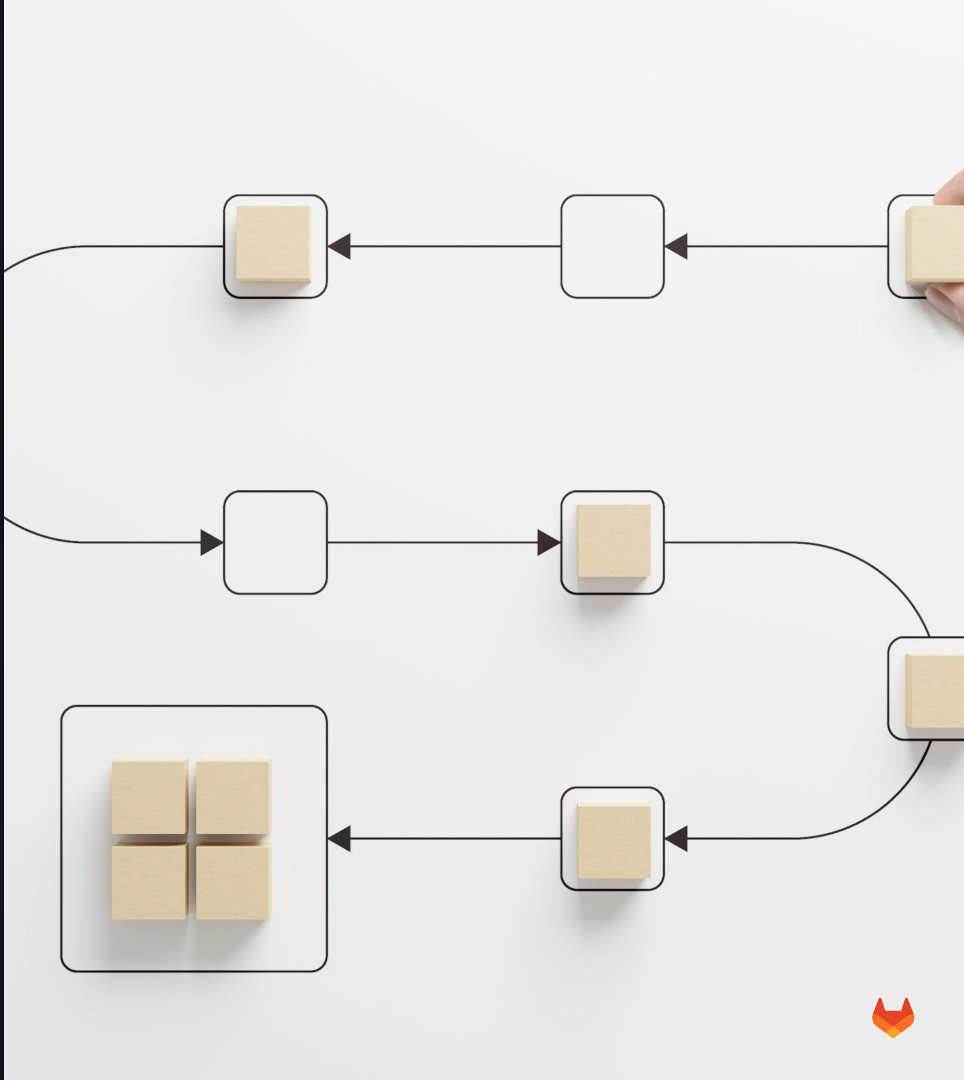
How to use?

Documentation
Wiki
Read the code

How to share?

Publish
Versioning
Share with your peers

More CI/CD inefficiencies



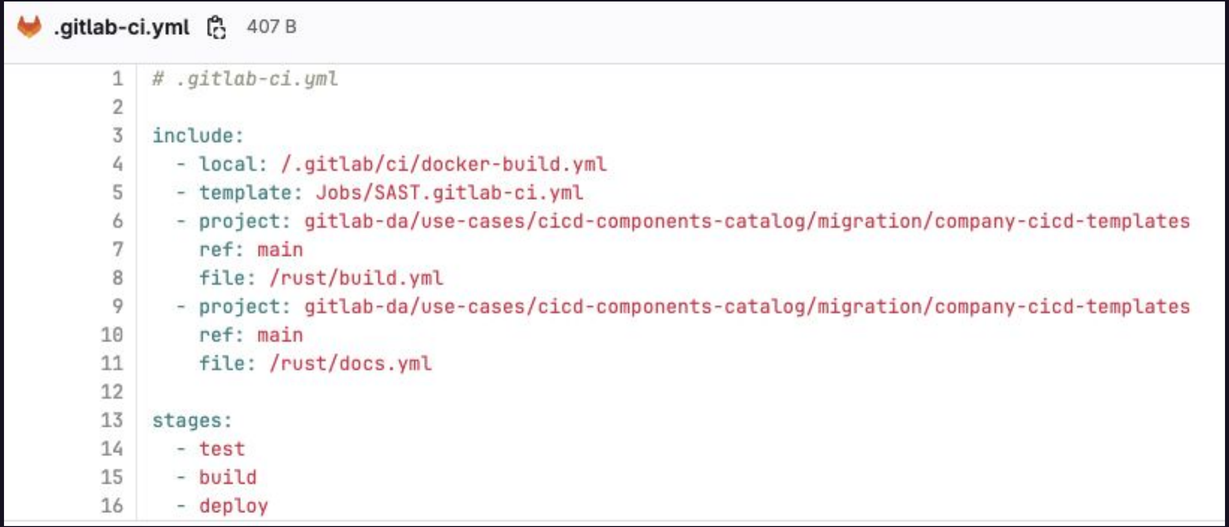
Traditional Pipeline Composition in GitLab CI/CD

Include CI/CD definitions from other repositories

Local to the project

Template: SAST

Project ref file in a different group

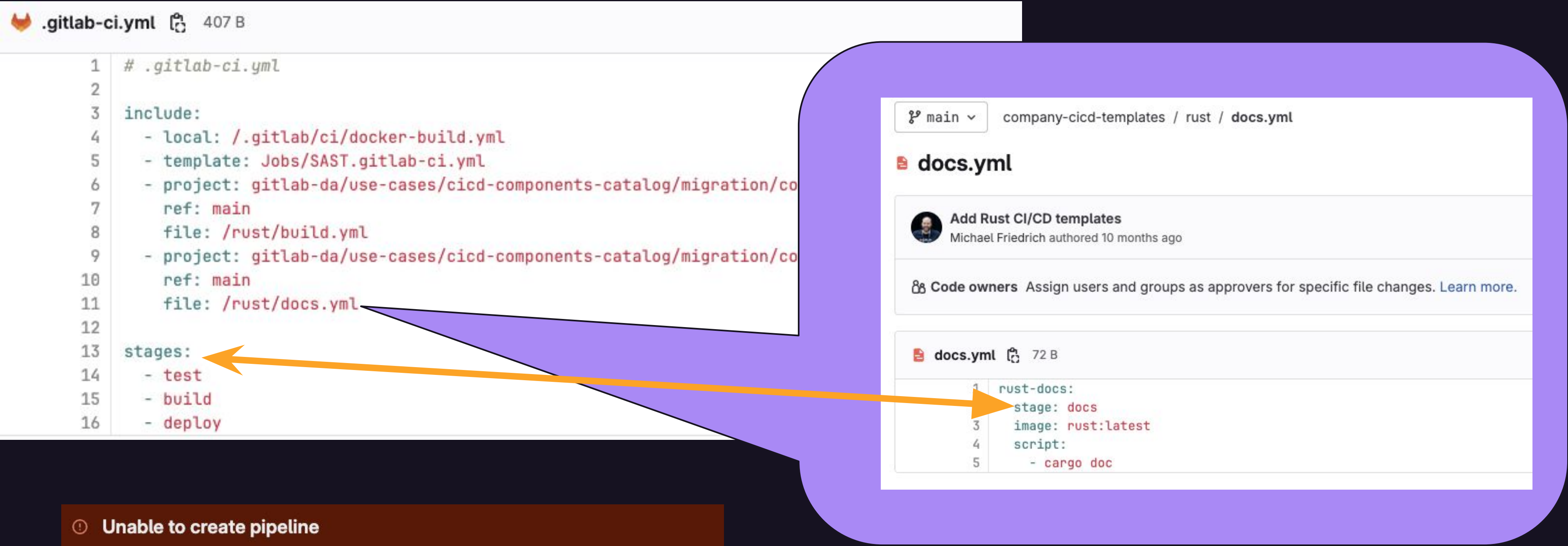


.gitlab-ci.yml 407 B

```yaml
# .gitlab-ci.yml

include:
  - local: /.gitlab/ci/docker-build.yml
  - template: Jobs/SAST.gitlab-ci.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/company-cicd-templates
    ref: main
    file: /rust/build.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/company-cicd-templates
    ref: main
    file: /rust/docs.yml

stages:
  - test
  - build
  - deploy
```

Traditional Pipeline Composition in GitLab CI/CD

Frequent problem: stage mismatch in consumer and include 😫



.gitlab-ci.yml 407 B

```yaml
# .gitlab-ci.yml

include:
  - local: /.gitlab/ci/docker-build.yml
  - template: Jobs/SAST.gitlab-ci.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/co
    ref: main
    file: /rust/build.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/co
    ref: main
    file: /rust/docs.yml

stages:
  - test
  - build
  - deploy
```

main company-cicd-templates / rust / docs.yml

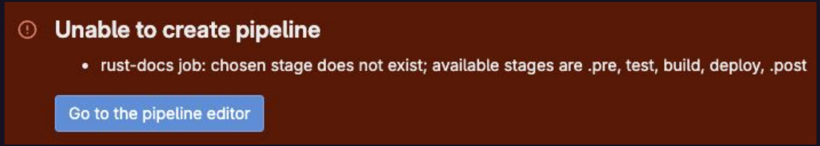
docs.yml

Add Rust CI/CD templates
Michael Friedrich authored 10 months ago

Code owners Assign users and groups as approvers for specific file changes. Learn more.

docs.yml 72 B

```yaml
rust-docs:
  stage: docs
  image: rust:latest
  script:
    - cargo doc
```

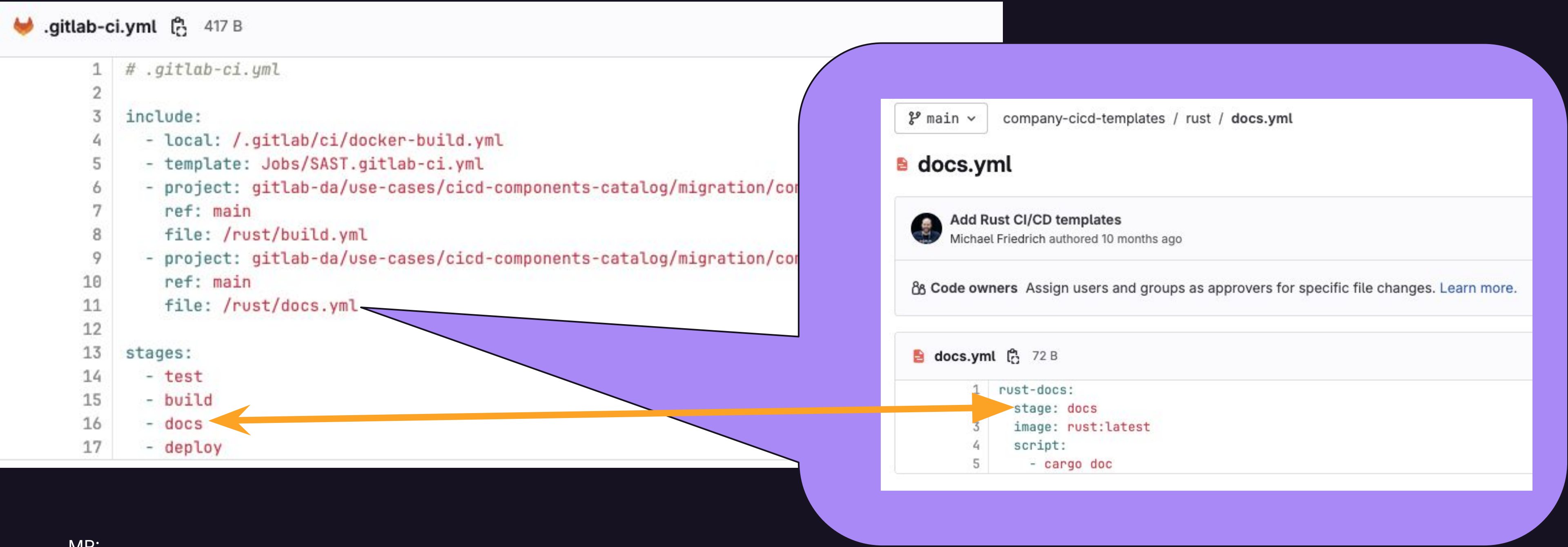


Unable to create pipeline

- rust-docs job: chosen stage does not exist; available stages are .pre, test, build, deploy, .post

Traditional Pipeline Composition in GitLab CI/CD

Solution 1: add the stage 🤨



.gitlab-ci.yml 417 B

```
# .gitlab-ci.yml

include:
  - local: /.gitlab/ci/docker-build.yml
  - template: Jobs/SAST.gitlab-ci.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/co
    ref: main
    file: /rust/build.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/co
    ref: main
    file: /rust/docs.yml

stages:
  - test
  - build
  - docs
  - deploy
```

main company-cicd-templates / rust / docs.yml

docs.yml

Add Rust CI/CD templates
Michael Friedrich authored 10 months ago

Code owners Assign users and groups as approvers for specific file changes. Learn more.

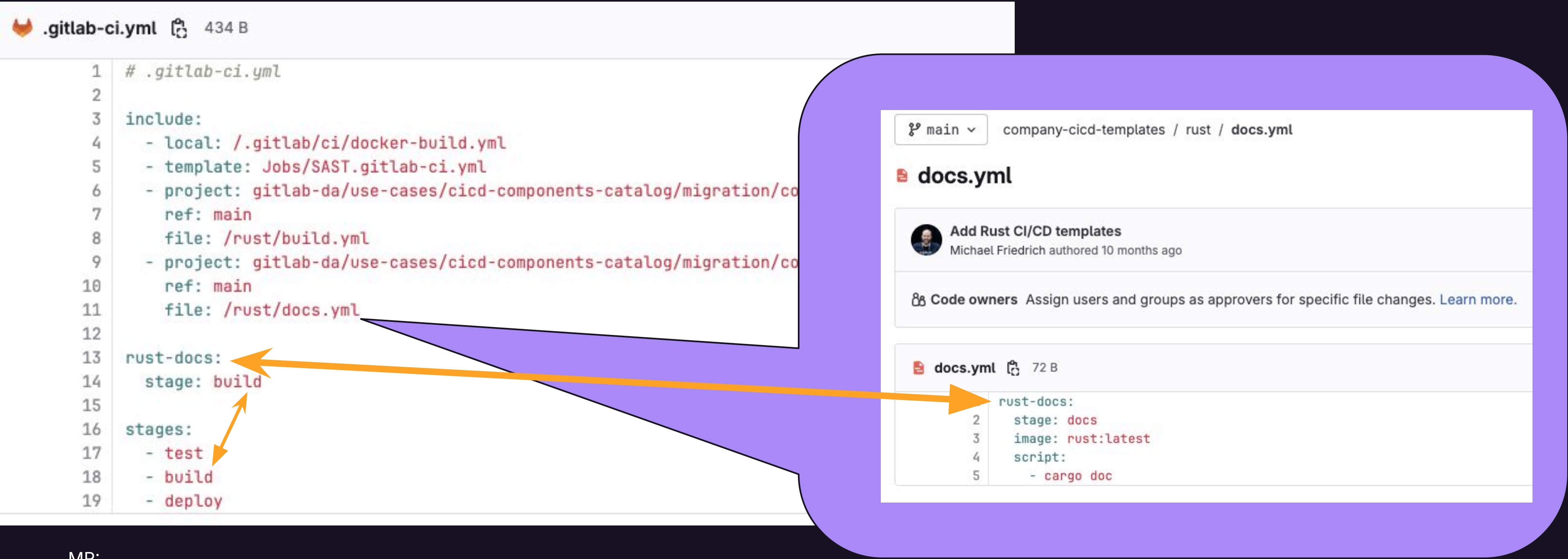
docs.yml 72 B

```
rust-docs:
  stage: docs
  image: rust:latest
  script:
    - cargo doc
```

MR:
https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/migration/include-stages-rust-gitlab-api/-/merge_requests/2

Traditional Pipeline Composition in GitLab CI/CD

Solution 2: override the job 🥴



.gitlab-ci.yml 434 B

```yaml
# .gitlab-ci.yml

include:
  - local: /.gitlab/ci/docker-build.yml
  - template: Jobs/SAST.gitlab-ci.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/co
    ref: main
    file: /rust/build.yml
  - project: gitlab-da/use-cases/cicd-components-catalog/migration/co
    ref: main
    file: /rust/docs.yml

rust-docs:
  stage: build

stages:
  - test
  - build
  - deploy
```

main company-cicd-templates / rust / docs.yml

docs.yml

Add Rust CI/CD templates
Michael Friedrich authored 10 months ago

Code owners Assign users and groups as approvers for specific file changes. Learn more.

docs.yml 72 B

```yaml
rust-docs:
  stage: docs
  image: rust:latest
  script:
    - cargo doc
```

MR:
https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/migration/include-stages-rust-gitlab-api/-/merge_requests/2

GitLab

CI/CD Components

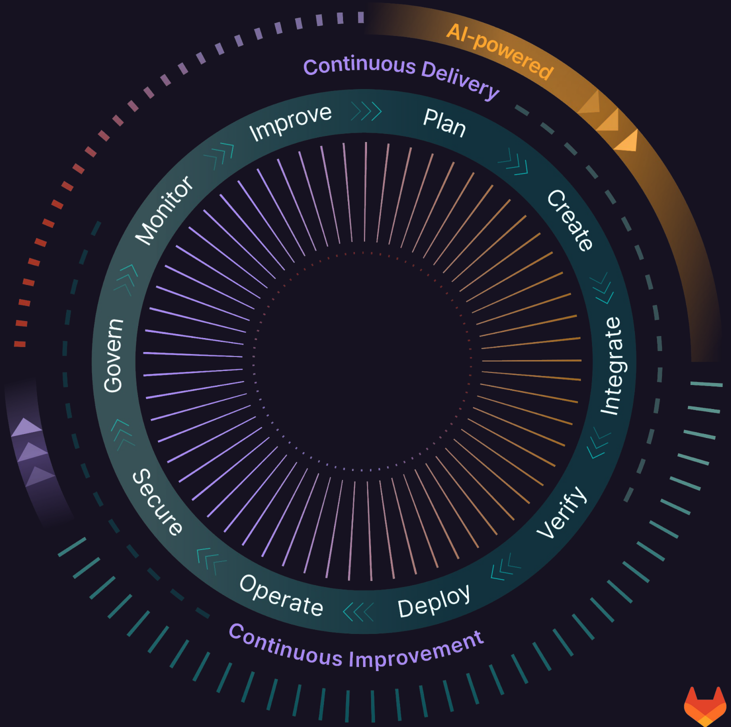
Streamline and automate pipeline creation

Easy to assemble pipeline components

→ consistent and repeatable workflows

Shareable
Reusable
Discoverable

→ across teams to improve collaboration and increase DevSecOps efficiency



CI/CD component

1. Modularized job definition
2. Documented purpose
3. Parameter specification
4. Test automation
5. Versions with traditional release/publish workflow
6. Everyone can contribute - GitLab Merge Requests

CI/CD catalog

Collection of available CI/CD Components

1. Explorable index (Single source of Truth)
2. Public or internal
3. Enables reusability

Shared ownership

1. Instance maintainers
2. GitLab maintained components
3. Components created by verified GitLab partners
4. Community contributions

Adoption path

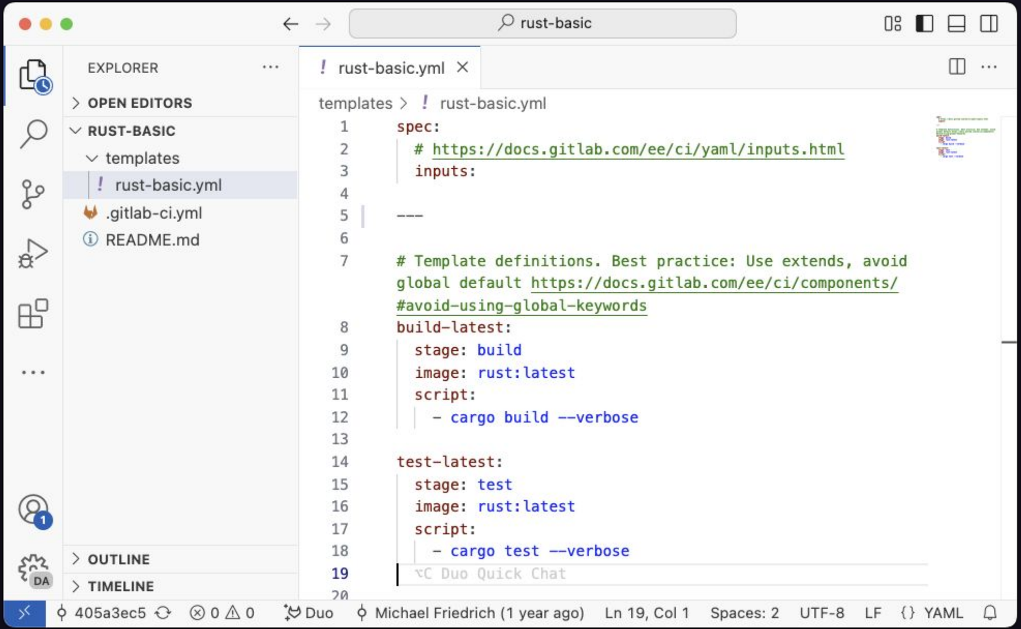
- **Templates** - Any part of CI/CD pipeline configuration (exists today)
- **Components** - Reusable unit of pipeline configuration
- **Catalog** - Collection of components, searchable on global instance
- **New component types** - future

GitLab

Inside a component

CI/CD component overview

- Directory structure
- Spec for **metadata** and **inputs**
- **Template** definitions
- Project in GitLab, test and release workflows in CI/CD



```yaml
spec:
  # https://docs.gitlab.com/ee/ci/yaml/inputs.html
  inputs:

---

# Template definitions. Best practice: Use extends, avoid global default https://docs.gitlab.com/ee/ci/components/#avoid-using-global-keywords
build-latest:
  stage: build
  image: rust:latest
  script:
    - cargo build --verbose

test-latest:
  stage: test
  image: rust:latest
  script:
    - cargo test --verbose
```

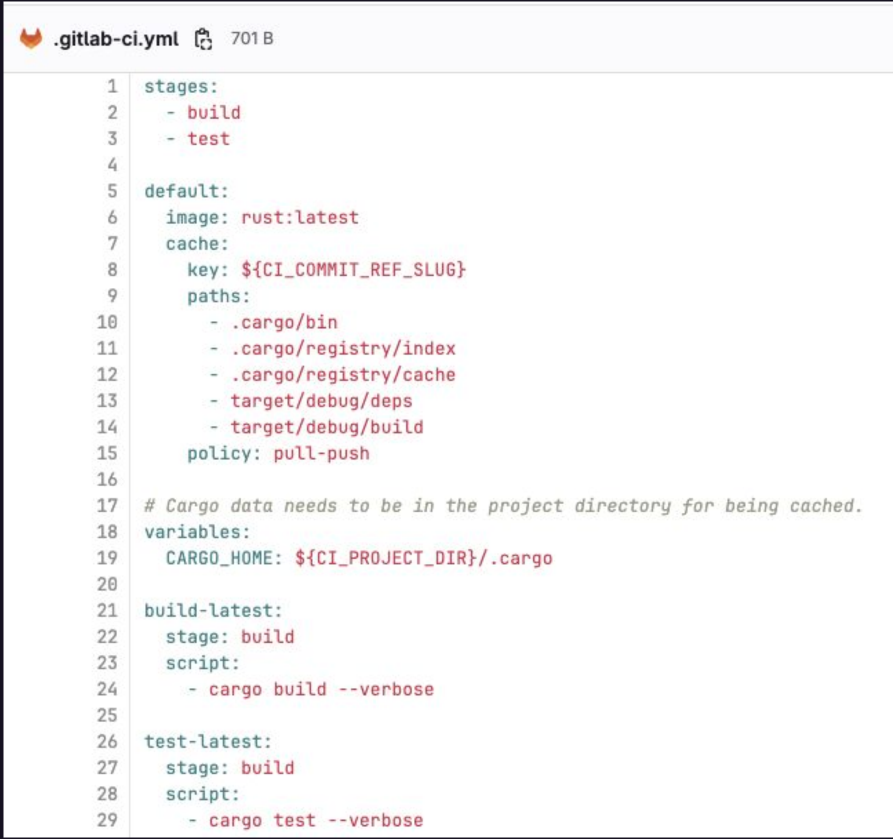
GitLab

Create and Consume a CI/CD Component

Practical Example: Rust

Refactor Rust CI/CD config from the blog post “Learning Rust with a little help from AI”

Blog: https://about.gitlab.com/blog/2023/08/10/learning-rust-with-a-little-help-from-ai-code-suggestions-getting-started/
MR: https://gitlab.com/gitlab-da/use-cases/ai/learn-with-ai/learn-rust-ai/-/merge_requests/1/diffs



.gitlab-ci.yml 701 B

```yaml
stages:
  - build
  - test

default:
  image: rust:latest
  cache:
    key: ${CI_COMMIT_REF_SLUG}
    paths:
      - .cargo/bin
      - .cargo/registry/index
      - .cargo/registry/cache
      - target/debug/deps
      - target/debug/build
    policy: pull-push

# Cargo data needs to be in the project directory for being cached.
variables:
  CARGO_HOME: ${CI_PROJECT_DIR}/.cargo

build-latest:
  stage: build
  script:
    - cargo build --verbose

test-latest:
  stage: build
  script:
    - cargo test --verbose
```

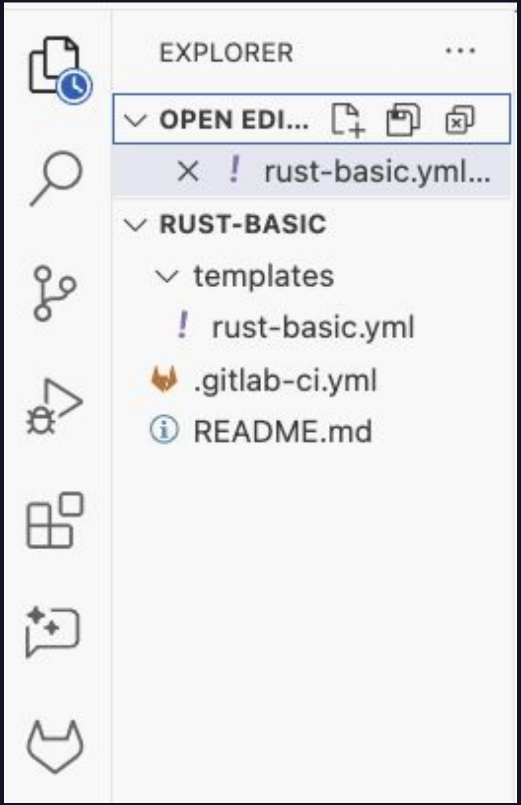
Basic Rust component

Start at https://docs.gitlab.com/ee/ci/components/

Create a GitLab project and open the Web IDE

Directory tree

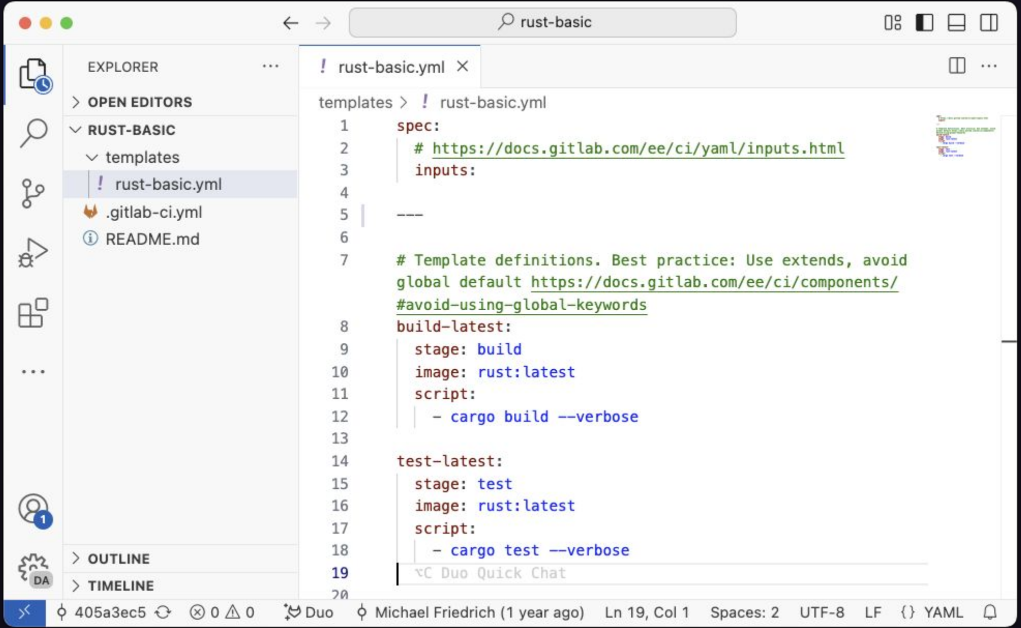
templates/	← component type
rust-basic.yml	← component template name (w/o .yml)
README.md	← How to use
.gitlab-ci.yml	← testing the component



CI/CD jobs in Rust component

MVC (minimal valuable change):

1. 💡 **spec** with empty inputs
2. 🌱 Two **jobs:** *build-latest, test-latest*
   a. stage
   b. image
   c. script
3. **Template name**: rust-basic
4. 🤔 No optimization yet

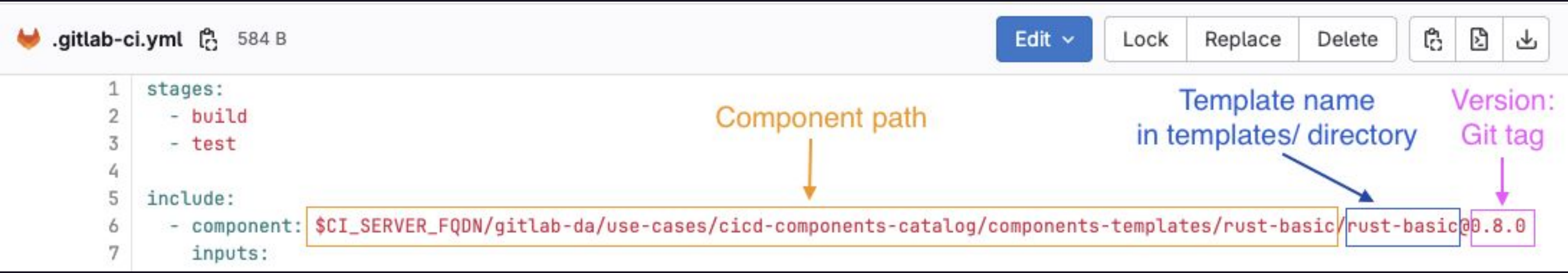


```yaml
spec:
  # https://docs.gitlab.com/ee/ci/yaml/inputs.html
  inputs:

---

# Template definitions. Best practice: Use extends, avoid global default https://docs.gitlab.com/ee/ci/components/#avoid-using-global-keywords
build-latest:
  stage: build
  image: rust:latest
  script:
    - cargo build --verbose

test-latest:
  stage: test
  image: rust:latest
  script:
    - cargo test --verbose
```

Demo component in https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic

Consume Rust CI/CD component

Let's test this first iteration.



```yaml
stages:
  - build
  - test

include:
  - component: $CI_SERVER_FQDN/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.8.0
    inputs:
```

The *include* keyword supports the *components* keyword.
The variable *CI_SERVER_FQDN* ensures that components can be used on any self-managed or SaaS instance without editing.

Requires a component path:
$CI_SERVER_FQDN/namespace/component/component-template-name@tagged-version

Remember the component directory tree? Omit *templates* in the path.

Dynamic inputs

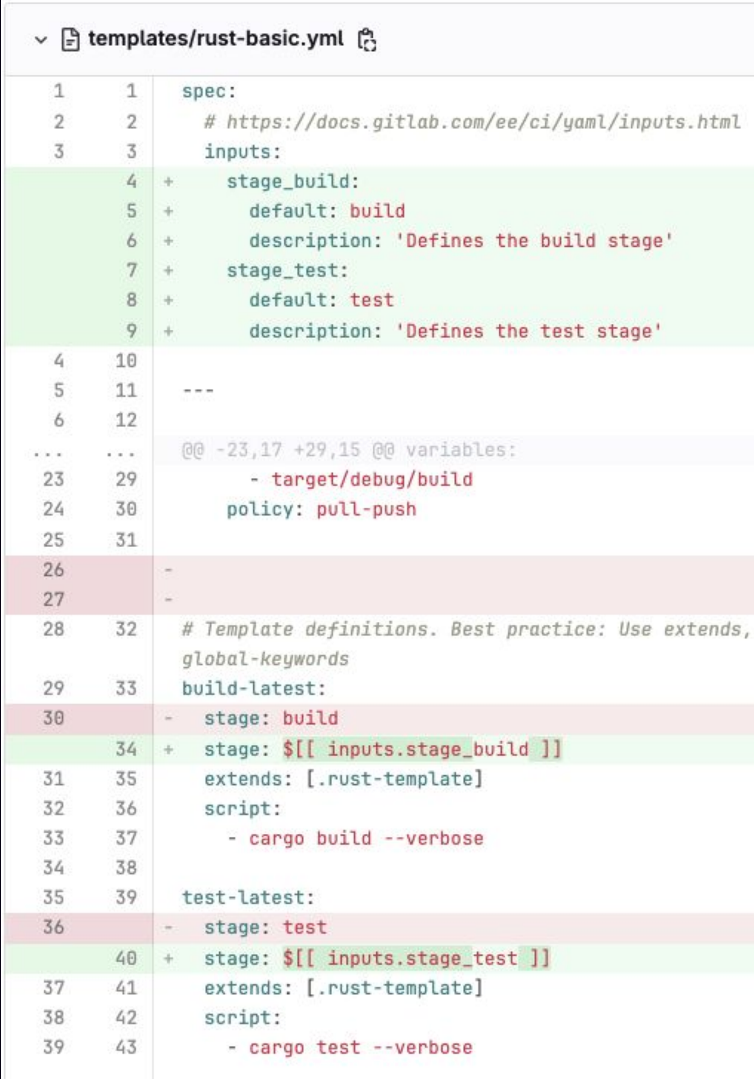
spec:
  inputs:
    variable_name:
      default: ***variable_default_value***
      description: ***variable_description***

Usage:

$[[inputs.***variable_name***]]

Example: Define default stage names for build/test

https://docs.gitlab.com/ee/ci/yaml/inputs.html



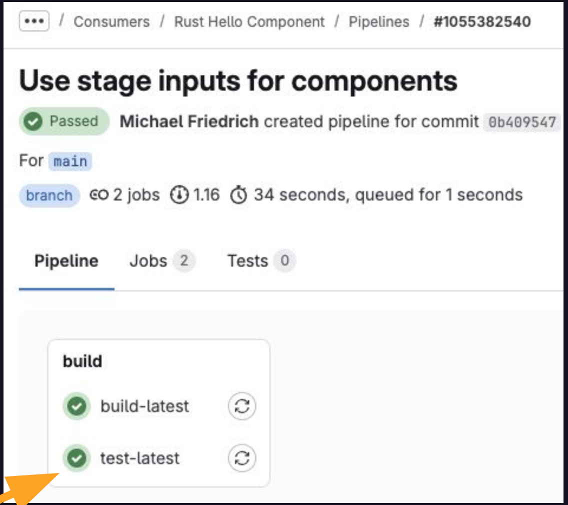
templates/rust-basic.yml

```diff
 spec:
   # https://docs.gitlab.com/ee/ci/yaml/inputs.html
   inputs:
+    stage_build:
+      default: build
+      description: 'Defines the build stage'
+    stage_test:
+      default: test
+      description: 'Defines the test stage'

 ---

@@ -23,17 +29,15 @@ variables:
       - target/debug/build
     policy: pull-push

-
-
 # Template definitions. Best practice: Use extends,
 global-keywords
 build-latest:
-  stage: build
+  stage: $[[ inputs.stage_build ]]
   extends: [.rust-template]
   script:
     - cargo build --verbose

 test-latest:
-  stage: test
+  stage: $[[ inputs.stage_test ]]
   extends: [.rust-template]
   script:
     - cargo test --verbose
```

Dynamic inputs - validation

Specify the component inputs

For testing, use the **same stage value *build*** which is different from the default.



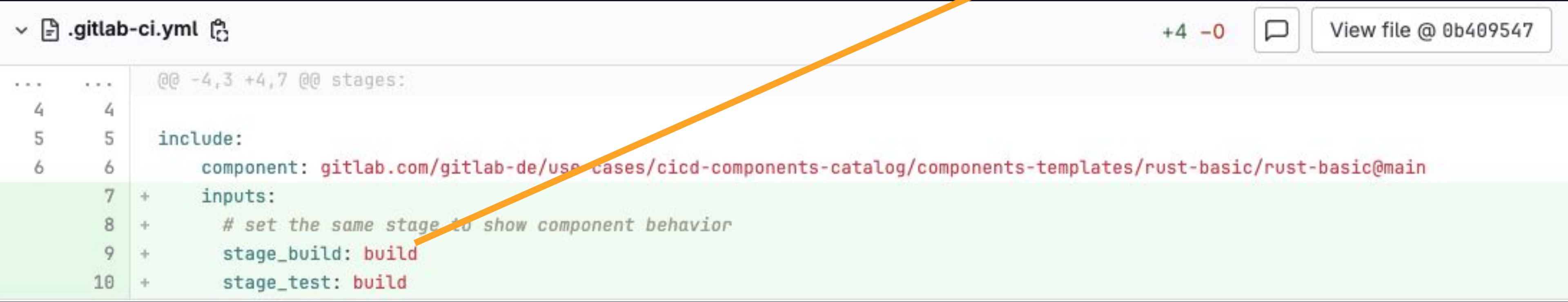
... / Consumers / Rust Hello Component / Pipelines / **#1055382540**

Use stage inputs for components

Passed **Michael Friedrich** created pipeline for commit `0b409547`

For `main`

branch 2 jobs 1.16 34 seconds, queued for 1 seconds

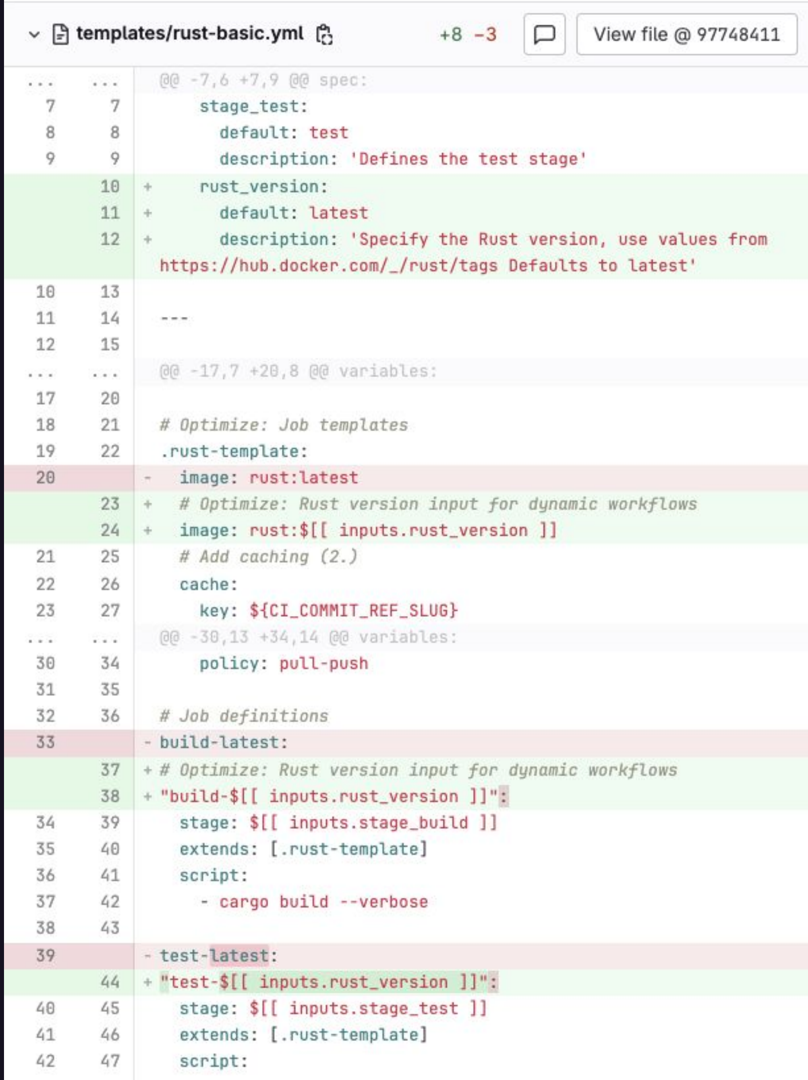


```
@@ -4,3 +4,7 @@ stages:

 include:
     component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@main
+    inputs:
+      # set the same stage to show component behavior
+      stage_build: build
+      stage_test: build
```

Dynamic inputs ++

Replace *latest* value in job names and images

- Define the Rust version to use as input
- Use image tags from https://hub.docker.com/_/rust/tags



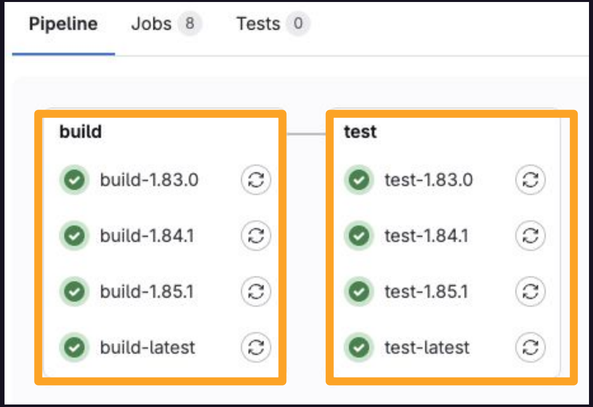
templates/rust-basic.yml +8 −3 View file @ 97748411

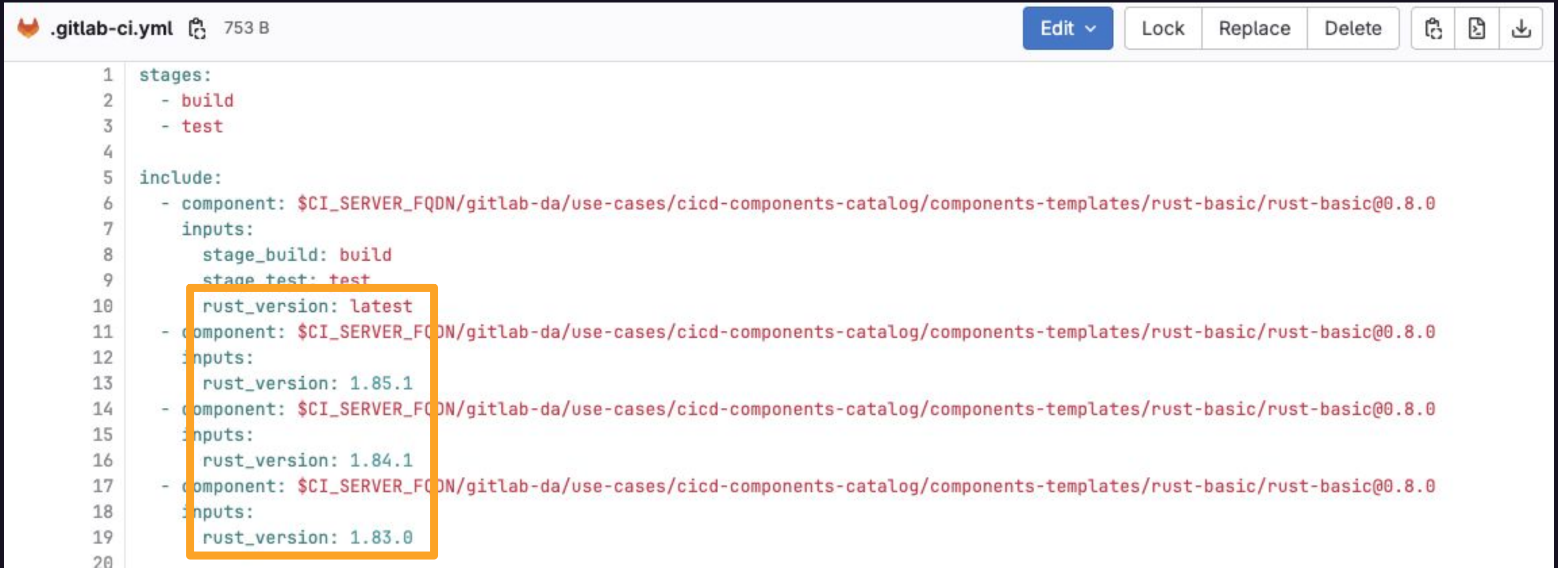
```diff
@@ -7,6 +7,9 @@ spec:
     stage_test:
       default: test
       description: 'Defines the test stage'
+    rust_version:
+      default: latest
+      description: 'Specify the Rust version, use values from https://hub.docker.com/_/rust/tags Defaults to latest'

 ---

@@ -17,7 +20,8 @@ variables:

 # Optimize: Job templates
 .rust-template:
-  image: rust:latest
+  # Optimize: Rust version input for dynamic workflows
+  image: rust:$[[ inputs.rust_version ]]
   # Add caching (2.)
   cache:
     key: ${CI_COMMIT_REF_SLUG}
@@ -30,13 +34,14 @@ variables:
     policy: pull-push

 # Job definitions
-build-latest:
+# Optimize: Rust version input for dynamic workflows
+"build-$[[ inputs.rust_version ]]":
   stage: $[[ inputs.stage_build ]]
   extends: [.rust-template]
   script:
     - cargo build --verbose

-test-latest:
+"test-$[[ inputs.rust_version ]]":
   stage: $[[ inputs.stage_test ]]
   extends: [.rust-template]
   script:
```

Dynamic job names

⚡ Job names based on input

💥 Reusable components

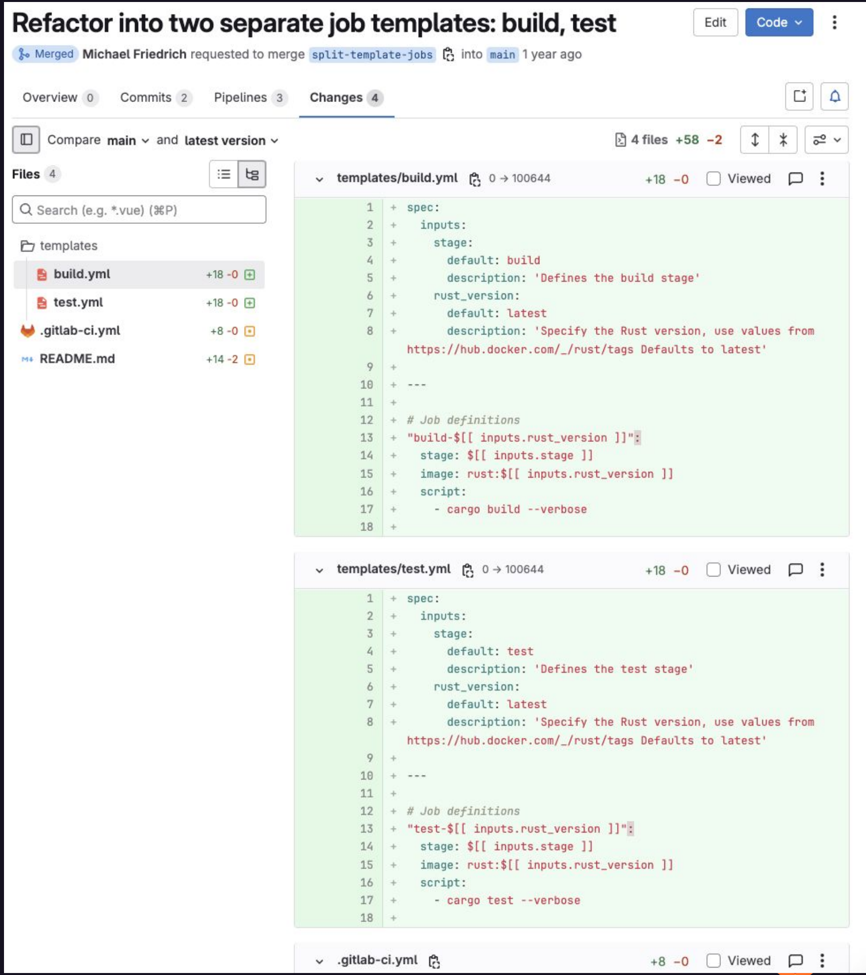




```yaml
stages:
  - build
  - test

include:
  - component: $CI_SERVER_FQDN/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.8.0
    inputs:
      stage_build: build
      stage_test: test
      rust_version: latest
  - component: $CI_SERVER_FQDN/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.8.0
    inputs:
      rust_version: 1.85.1
  - component: $CI_SERVER_FQDN/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.8.0
    inputs:
      rust_version: 1.84.1
  - component: $CI_SERVER_FQDN/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.8.0
    inputs:
      rust_version: 1.83.0
```

Optimize CI/CD component

1. Split the single template into job specific templates
   a. build
   b. test
2. Consider
   a. Adding more inputs
   b. Default inputs values
   c. Caching
3. Avoid global settings (no *default* keyword)



Refactor into two separate job templates: build, test

Merged Michael Friedrich requested to merge `split-template-jobs` into `main` 1 year ago

templates/build.yml 0 → 100644 +18 −0

```
+ spec:
+   inputs:
+     stage:
+       default: build
+       description: 'Defines the build stage'
+     rust_version:
+       default: latest
+       description: 'Specify the Rust version, use values from https://hub.docker.com/_/rust/tags Defaults to latest'
+
+ ---
+
+ # Job definitions
+ "build-$[[ inputs.rust_version ]]":
+   stage: $[[ inputs.stage ]]
+   image: rust:$[[ inputs.rust_version ]]
+   script:
+     - cargo build --verbose
+
```

templates/test.yml 0 → 100644 +18 −0

```
+ spec:
+   inputs:
+     stage:
+       default: test
+       description: 'Defines the test stage'
+     rust_version:
+       default: latest
+       description: 'Specify the Rust version, use values from https://hub.docker.com/_/rust/tags Defaults to latest'
+
+ ---
+
+ # Job definitions
+ "test-$[[ inputs.rust_version ]]":
+   stage: $[[ inputs.stage ]]
+   image: rust:$[[ inputs.rust_version ]]
+   script:
+     - cargo test --verbose
+
```

.gitlab-ci.yml +8 −0

GitLab

Maintain CI/CD Components

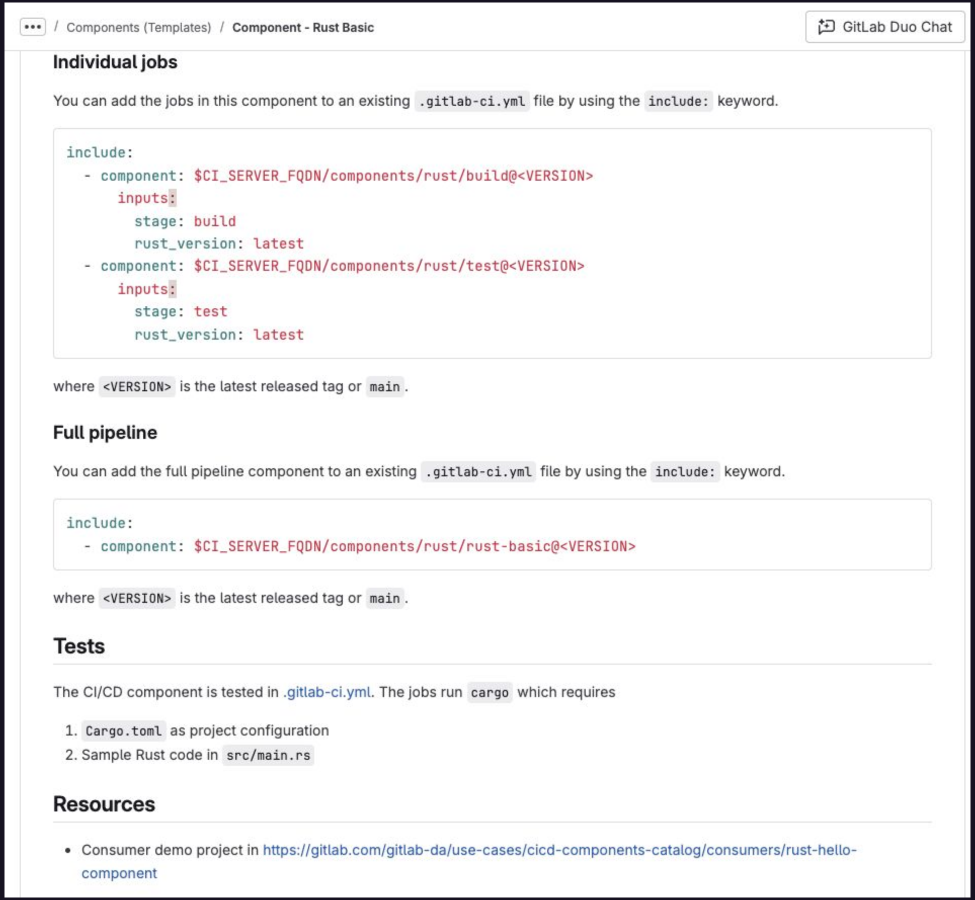
Documentation, tests

Documentation

- README.md best practices

  1. Purpose of the component
  2. Usage
  3. Testing & Development

- Inputs documentation is automatically generated in the CI/CD catalog



Individual jobs

You can add the jobs in this component to an existing `.gitlab-ci.yml` file by using the `include:` keyword.

```
include:
  - component: $CI_SERVER_FQDN/components/rust/build@<VERSION>
    inputs:
      stage: build
      rust_version: latest
  - component: $CI_SERVER_FQDN/components/rust/test@<VERSION>
    inputs:
      stage: test
      rust_version: latest
```

where `<VERSION>` is the latest released tag or `main`.

Full pipeline

You can add the full pipeline component to an existing `.gitlab-ci.yml` file by using the `include:` keyword.

```
include:
  - component: $CI_SERVER_FQDN/components/rust/rust-basic@<VERSION>
```

where `<VERSION>` is the latest released tag or `main`.

Tests

The CI/CD component is tested in .gitlab-ci.yml. The jobs run `cargo` which requires

1. `Cargo.toml` as project configuration
2. Sample Rust code in `src/main.rs`

Resources

- Consumer demo project in https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/consumers/rust-hello-component

Demo component in https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic

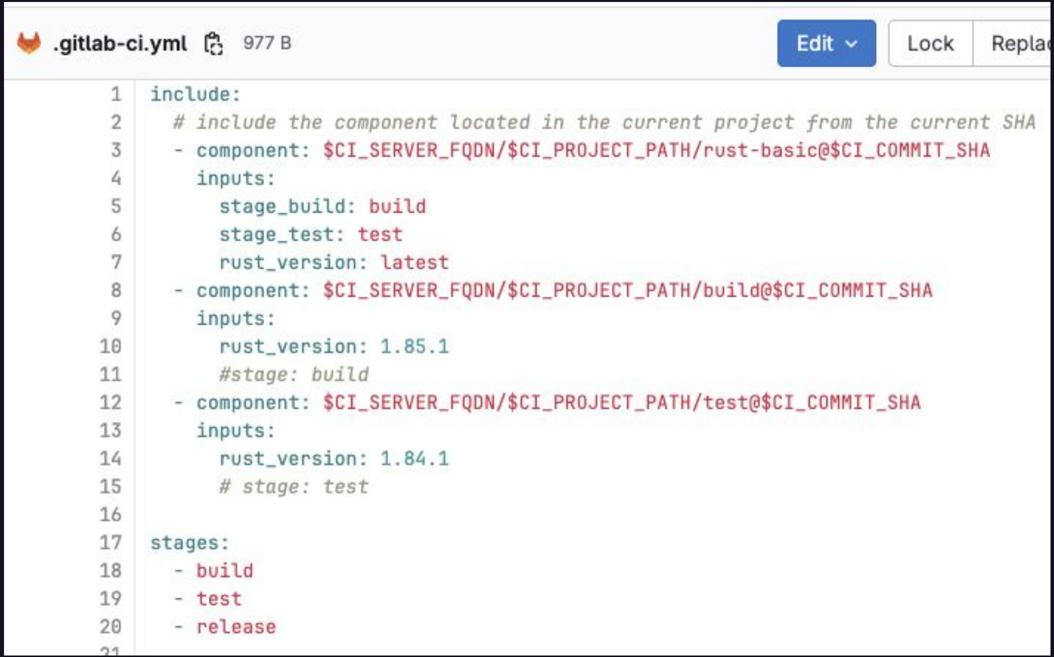
Testing a CI/CD component

Include the component in CI/CD in the same project

Test different input values for different templates

Use pre-defined CI/CD variables

https://docs.gitlab.com/ci/components/examples/#test-a-component



```yaml
include:
  # include the component located in the current project from the current SHA
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/rust-basic@$CI_COMMIT_SHA
    inputs:
      stage_build: build
      stage_test: test
      rust_version: latest
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/build@$CI_COMMIT_SHA
    inputs:
      rust_version: 1.85.1
      #stage: build
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/test@$CI_COMMIT_SHA
    inputs:
      rust_version: 1.84.1
      # stage: test

stages:
  - build
  - test
  - release
```

Demo component in https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic

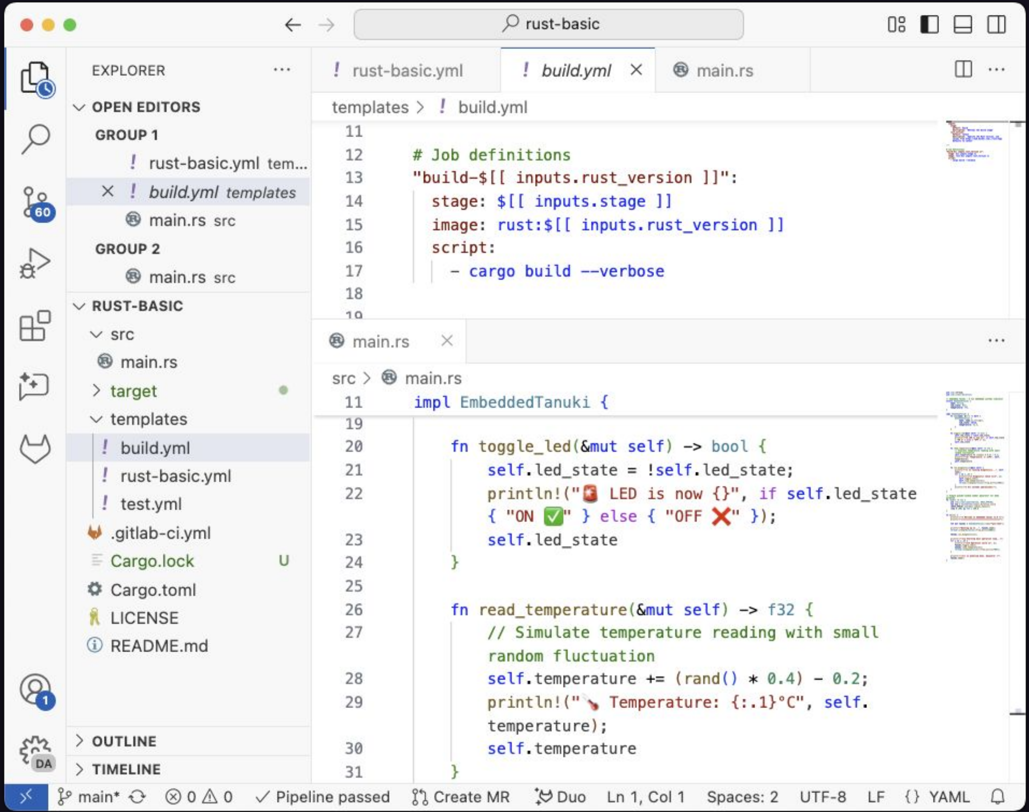
Testing with app code

Include source code, configuration, etc. environment in the component

Rust (run *cargo init*)

Cargo.toml configuration
src/main.rs source code

Advanced testing example with parallel matrix builds in the Go component: https://gitlab.com/components/go/-/blob/main/.gitlab-ci.yml?ref_type=heads



```yaml
  # Job definitions
  "build-$[[ inputs.rust_version ]]":
    stage: $[[ inputs.stage ]]
    image: rust:$[[ inputs.rust_version ]]
    script:
      - cargo build --verbose
```

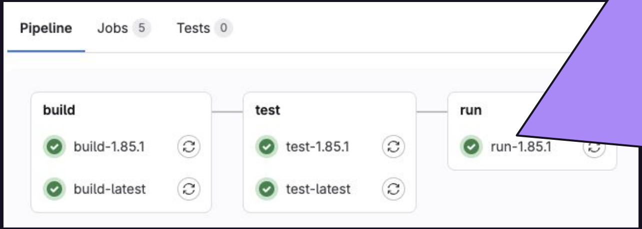
```rust
impl EmbeddedTanuki {

    fn toggle_led(&mut self) -> bool {
        self.led_state = !self.led_state;
        println!("🚨 LED is now {}", if self.led_state
        { "ON ✅" } else { "OFF ❌" });
        self.led_state
    }

    fn read_temperature(&mut self) -> f32 {
        // Simulate temperature reading with small
        random fluctuation
        self.temperature += (rand() * 0.4) - 0.2;
        println!("🌡️ Temperature: {:.1}°C", self.
        temperature);
        self.temperature
    }
```

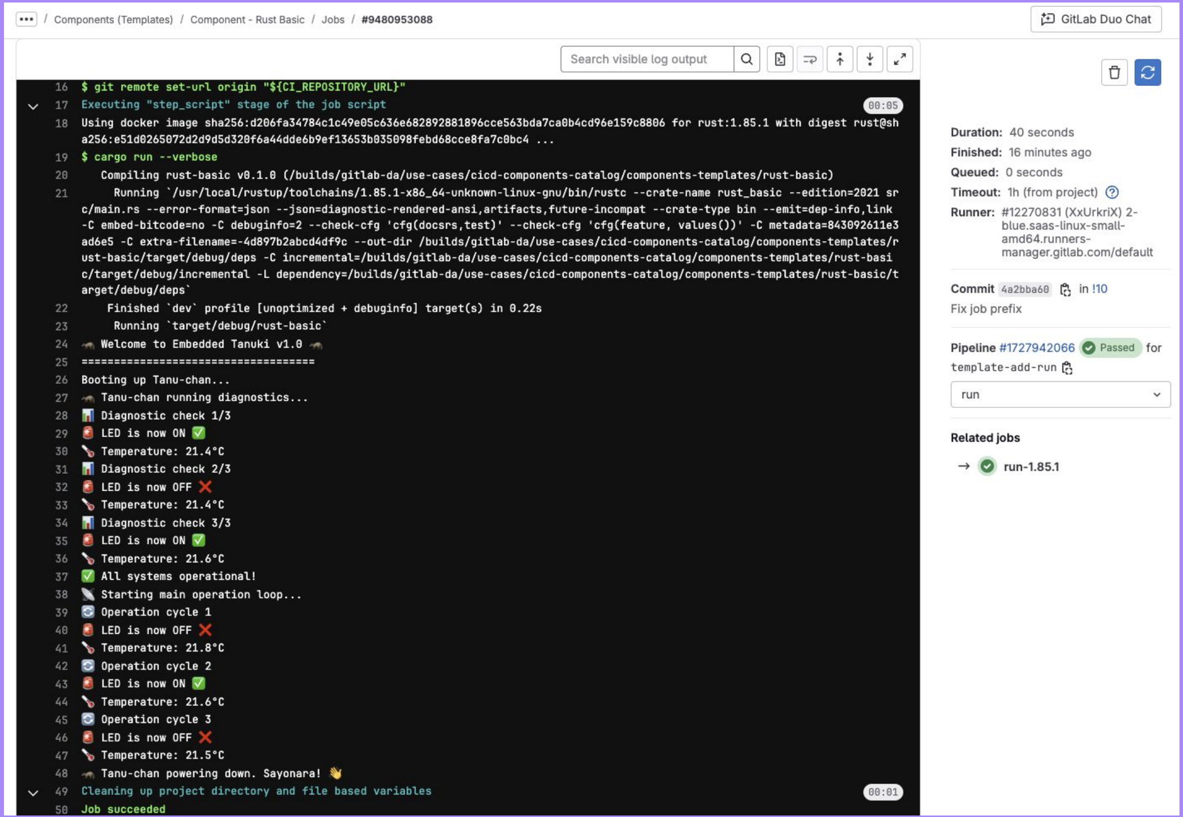
Iterate

Add more component templates and inputs as needed

→ run template for “cargo run”

Verify CI/CD pipelines





Components (Templates) / Component - Rust Basic / Jobs / #9480953088
GitLab Duo Chat
Search visible log output

```
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Executing "step_script" stage of the job script                                    00:05
Using docker image sha256:d206fa34784c1c49e05c636e682892881896cce563bda7ca0b4cd96e159c8806 for rust:1.85.1 with digest rust@sh
a256:e51d0265072d2d9d5d320f6a44dde6b9ef13653b035098febd68cce8fa7c0bc4 ...
$ cargo run --verbose
   Compiling rust-basic v0.1.0 (/builds/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic)
     Running `/usr/local/rustup/toolchains/1.85.1-x86_64-unknown-linux-gnu/bin/rustc --crate-name rust_basic --edition=2021 sr
c/main.rs --error-format=json --json=diagnostic-rendered-ansi,artifacts,future-incompat --crate-type bin --emit=dep-info,link
-C embed-bitcode=no -C debuginfo=2 --check-cfg 'cfg(docsrs,test)' --check-cfg 'cfg(feature, values())' -C metadata=843092611e3
ad6e5 -C extra-filename=-4d897b2abcd4df9c --out-dir /builds/gitlab-da/use-cases/cicd-components-catalog/components-templates/r
ust-basic/target/debug/deps -C incremental=/builds/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basi
c/target/debug/incremental -L dependency=/builds/gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic/t
arget/debug/deps`
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.22s
     Running `target/debug/rust-basic`
🦡 Welcome to Embedded Tanuki v1.0 🦡
====================================
Booting up Tanu-chan...
🦡 Tanu-chan running diagnostics...
📊 Diagnostic check 1/3
🚨 LED is now ON ✅
🌡️ Temperature: 21.4°C
📊 Diagnostic check 2/3
🚨 LED is now OFF ❌
🌡️ Temperature: 21.4°C
📊 Diagnostic check 3/3
🚨 LED is now ON ✅
🌡️ Temperature: 21.6°C
✅ All systems operational!
📡 Starting main operation loop...
🔄 Operation cycle 1
🚨 LED is now OFF ❌
🌡️ Temperature: 21.8°C
🔄 Operation cycle 2
🚨 LED is now ON ✅
🌡️ Temperature: 21.6°C
🔄 Operation cycle 3
🚨 LED is now OFF ❌
🌡️ Temperature: 21.5°C
🦡 Tanu-chan powering down. Sayonara! 👋
Cleaning up project directory and file based variables                             00:01
Job succeeded
```

Duration: 40 seconds
Finished: 16 minutes ago
Queued: 0 seconds
Timeout: 1h (from project)
Runner: #12270831 (XxUrkriX) 2-blue.saas-linux-small-amd64.runners-manager.gitlab.com/default

Commit `4a2bba60` in !10
Fix job prefix

Pipeline #1727942066 Passed for `template-add-run`

run

Related jobs
→ run-1.85.1

GitLab

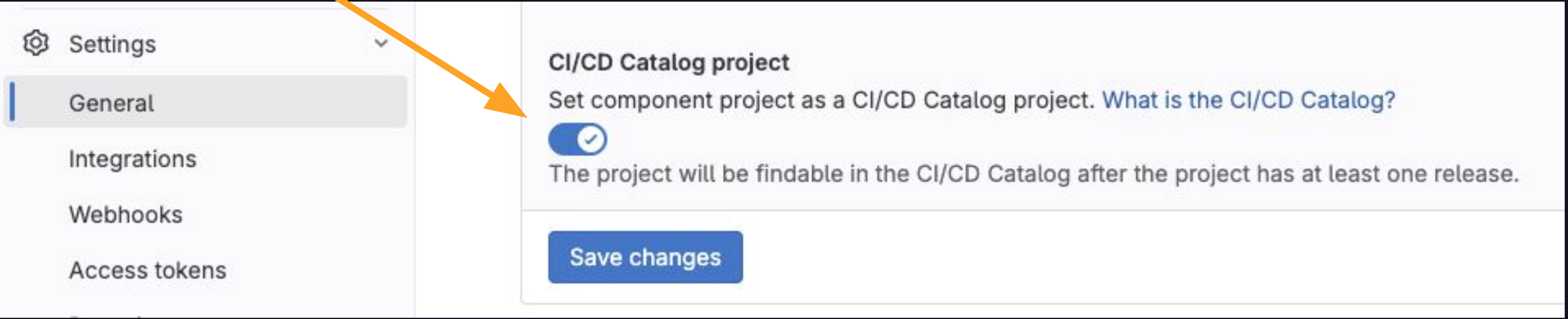
Visibility

CI/CD Catalog

Add Rust component to CI/CD catalog

Project > Settings >

1. Add description
2. Visibility, project features, permissions > CI/CD catalog project
3. **Toggle enabled and save changes**



CI/CD Catalog project

Set component project as a CI/CD Catalog project. What is the CI/CD Catalog?

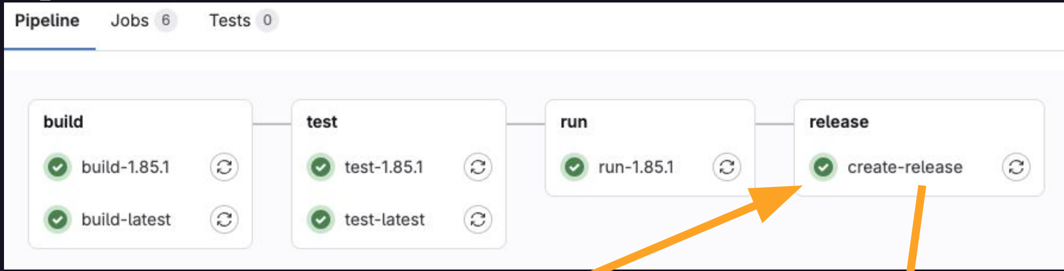
The project will be findable in the CI/CD Catalog after the project has at least one release.

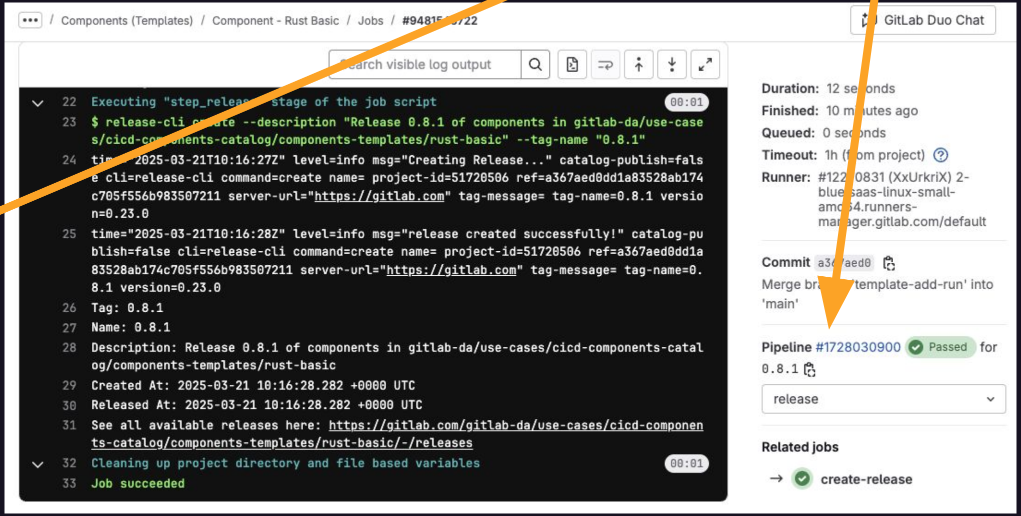
https://docs.gitlab.com/ci/components/#set-a-component-project-as-a-catalog-project

Release a CI/CD component

- Automated in CI/CD
- Automated Release notes
- Published into the CI/CD Catalog

User action: Create and push a Git tag (semantic version)




Components (Templates) / Component - Rust Basic / Jobs / #94815[illegible]722
GitLab Duo Chat
Search visible log output

```
Executing "step_release" stage of the job script                                  00:01
$ release-cli create --description "Release 0.8.1 of components in gitlab-da/use-case
s/cicd-components-catalog/components-templates/rust-basic" --tag-name "0.8.1"
time="2025-03-21T10:16:27Z" level=info msg="Creating Release..." catalog-publish=fals
e cli=release-cli command=create name= project-id=51720506 ref=a367aed0dd1a83528ab174
c705f556b983507211 server-url="https://gitlab.com" tag-message= tag-name=0.8.1 versio
n=0.23.0
time="2025-03-21T10:16:28Z" level=info msg="release created successfully!" catalog-pu
blish=false cli=release-cli command=create name= project-id=51720506 ref=a367aed0dd1a
83528ab174c705f556b983507211 server-url="https://gitlab.com" tag-message= tag-name=0.
8.1 version=0.23.0
Tag: 0.8.1
Name: 0.8.1
Description: Release 0.8.1 of components in gitlab-da/use-cases/cicd-components-catal
og/components-templates/rust-basic
Created At: 2025-03-21 10:16:28.282 +0000 UTC
Released At: 2025-03-21 10:16:28.282 +0000 UTC
See all available releases here: https://gitlab.com/gitlab-da/use-cases/cicd-componen
ts-catalog/components-templates/rust-basic/-/releases
Cleaning up project directory and file based variables                            00:01
Job succeeded
```

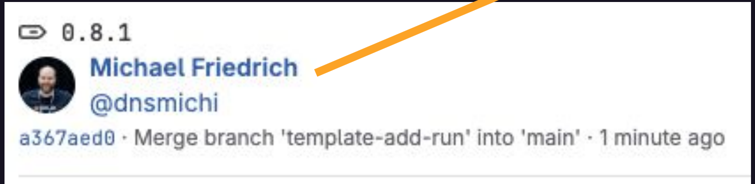
Duration: 12 seconds
Finished: 10 minutes ago
Queued: 0 seconds
Timeout: 1h (from project)
Runner: #122[illegible]0831 (XxUrkriX) 2-blue[illegible]saas-linux-small-amd64.runners-manager.gitlab.com/default

Commit a3[illegible]aed0
Merge branch 'template-add-run' into 'main'

Pipeline #1728030900 Passed for 0.8.1

release

Related jobs
→ create-release

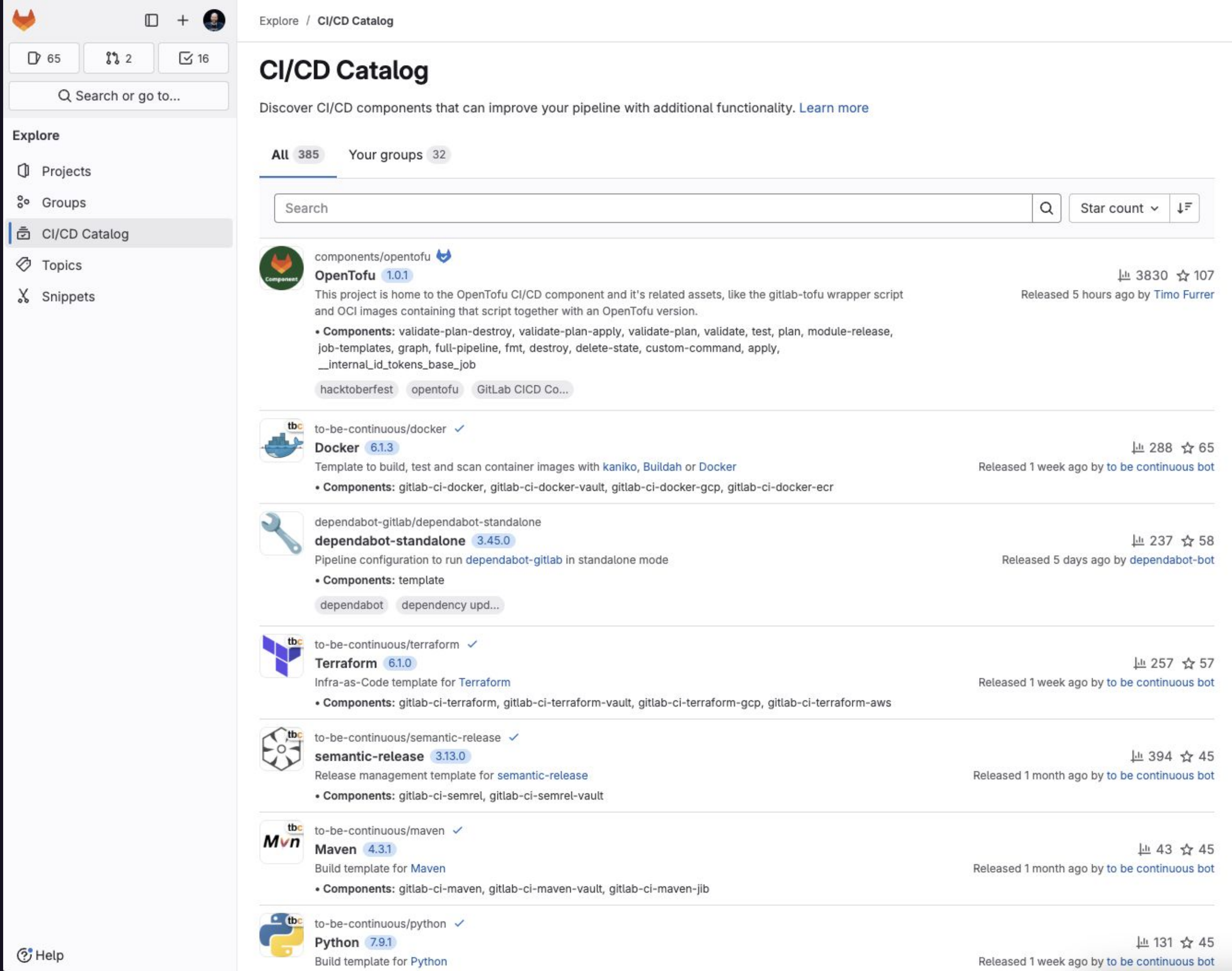


0.8.1
Michael Friedrich
@dnsmichi
a367aed0 · Merge branch 'template-add-run' into 'main' · 1 minute ago

CI/CD catalog

Search or go to > Explore > CI/CD catalog

https://gitlab.com/explore/catalog



CI/CD Catalog

Discover CI/CD components that can improve your pipeline with additional functionality. Learn more

All 385 Your groups 32

Search | Star count

components/opentofu
OpenTofu 1.0.1 — 3830 ☆ 107
Released 5 hours ago by Timo Furrer
This project is home to the OpenTofu CI/CD component and it's related assets, like the gitlab-tofu wrapper script and OCI images containing that script together with an OpenTofu version.
- **Components:** validate-plan-destroy, validate-plan-apply, validate-plan, validate, test, plan, module-release, job-templates, graph, full-pipeline, fmt, destroy, delete-state, custom-command, apply, __internal_id_tokens_base_job

hacktoberfest · opentofu · GitLab CICD Co...

to-be-continuous/docker ✓
Docker 6.1.3 — 288 ☆ 65
Released 1 week ago by to be continuous bot
Template to build, test and scan container images with kaniko, Buildah or Docker
- **Components:** gitlab-ci-docker, gitlab-ci-docker-vault, gitlab-ci-docker-gcp, gitlab-ci-docker-ecr

dependabot-gitlab/dependabot-standalone
dependabot-standalone 3.45.0 — 237 ☆ 58
Released 5 days ago by dependabot-bot
Pipeline configuration to run dependabot-gitlab in standalone mode
- **Components:** template

dependabot · dependency upd...

to-be-continuous/terraform ✓
Terraform 6.1.0 — 257 ☆ 57
Released 1 week ago by to be continuous bot
Infra-as-Code template for Terraform
- **Components:** gitlab-ci-terraform, gitlab-ci-terraform-vault, gitlab-ci-terraform-gcp, gitlab-ci-terraform-aws

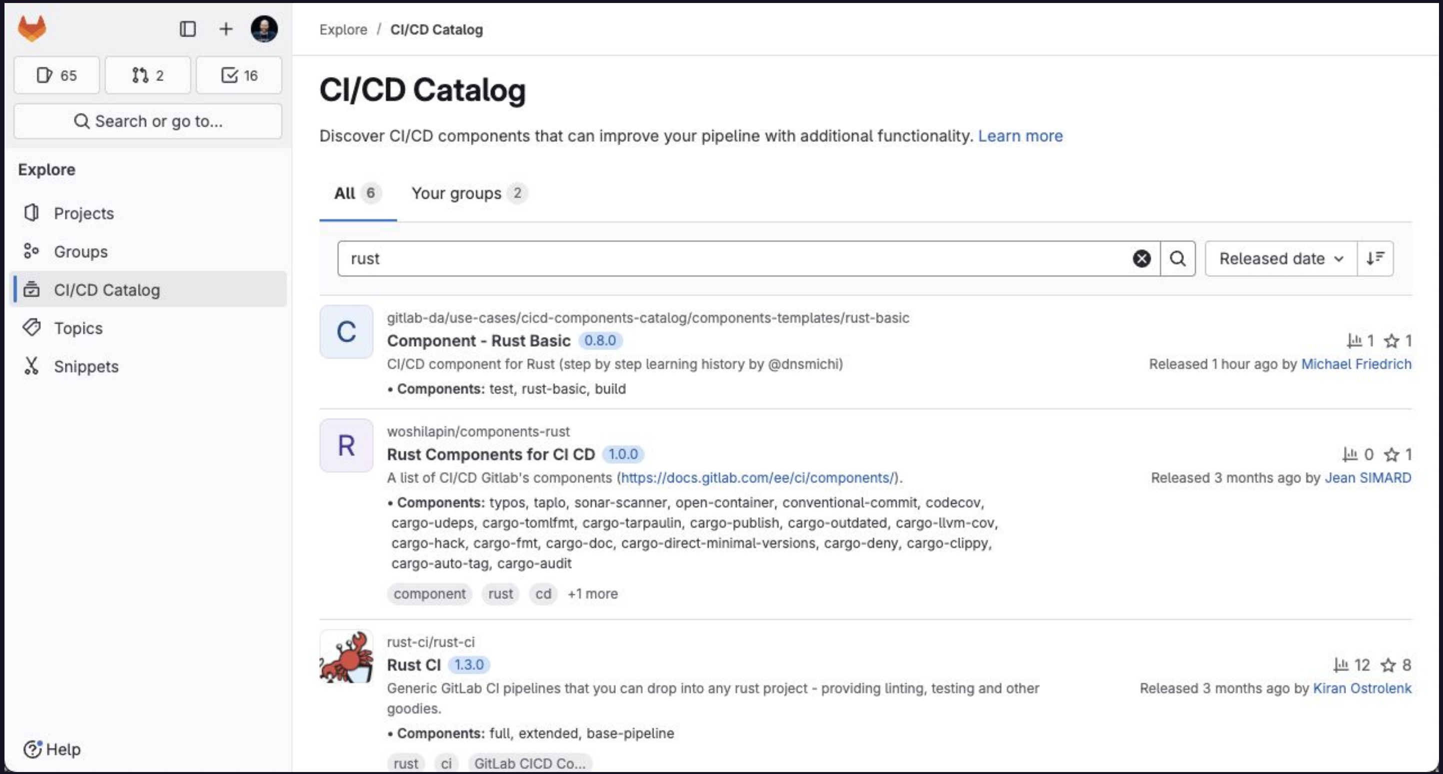
to-be-continuous/semantic-release ✓
semantic-release 3.13.0 — 394 ☆ 45
Released 1 month ago by to be continuous bot
Release management template for semantic-release
- **Components:** gitlab-ci-semrel, gitlab-ci-semrel-vault

to-be-continuous/maven ✓
Maven 4.3.1 — 43 ☆ 45
Released 1 month ago by to be continuous bot
Build template for Maven
- **Components:** gitlab-ci-maven, gitlab-ci-maven-vault, gitlab-ci-maven-jib

to-be-continuous/python ✓
Python 7.9.1 — 131 ☆ 45
Released 1 week ago by to be continuous bot
Build template for Python

CI/CD catalog

- Search for the component name
- Sort by release date, popularity, and more



CI/CD Catalog

Discover CI/CD components that can improve your pipeline with additional functionality. Learn more

All 6 | Your groups 2

Search: rust | Released date

gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic
Component - Rust Basic 0.8.0
CI/CD component for Rust (step by step learning history by @dnsmichi)
- **Components:** test, rust-basic, build

1 ☆ 1 — Released 1 hour ago by Michael Friedrich

woshilapin/components-rust
Rust Components for CI CD 1.0.0
A list of CI/CD Gitlab's components (https://docs.gitlab.com/ee/ci/components/).
- **Components:** typos, taplo, sonar-scanner, open-container, conventional-commit, codecov, cargo-udeps, cargo-tomlfmt, cargo-tarpaulin, cargo-publish, cargo-outdated, cargo-llvm-cov, cargo-hack, cargo-fmt, cargo-doc, cargo-direct-minimal-versions, cargo-deny, cargo-clippy, cargo-auto-tag, cargo-audit

component · rust · cd · +1 more

0 ☆ 1 — Released 3 months ago by Jean SIMARD

rust-ci/rust-ci
Rust CI 1.3.0
Generic GitLab CI pipelines that you can drop into any rust project - providing linting, testing and other goodies.
- **Components:** full, extended, base-pipeline

rust · ci · GitLab CICD Co...

12 ☆ 8 — Released 3 months ago by Kiran Ostrolenk

https://gitlab.com/explore/catalog

GitLab

Component Direction

Photo by Karsten Würth on Unsplash

Component direction

First iteration: Templates

Next iteration: Steps for reusable job pieces

Direction: Expanding catalog resource types

Direction: Support inputs for pipelines

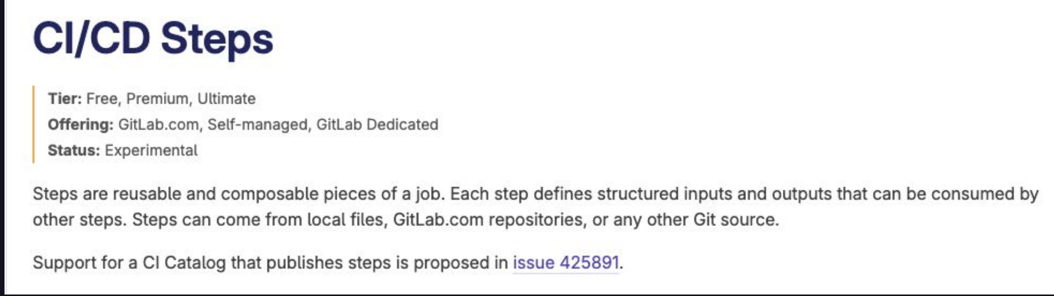
Direction: FY26 Pipeline Modularity

Photo by Bernard Hermant on Unsplash

CI/CD Steps

Steps are composable pieces of a job, replacing the *script* section.
Accepts inputs, can be reused and combined with CI/CD Components.

Status: Experiment. Add ideas & feedback.

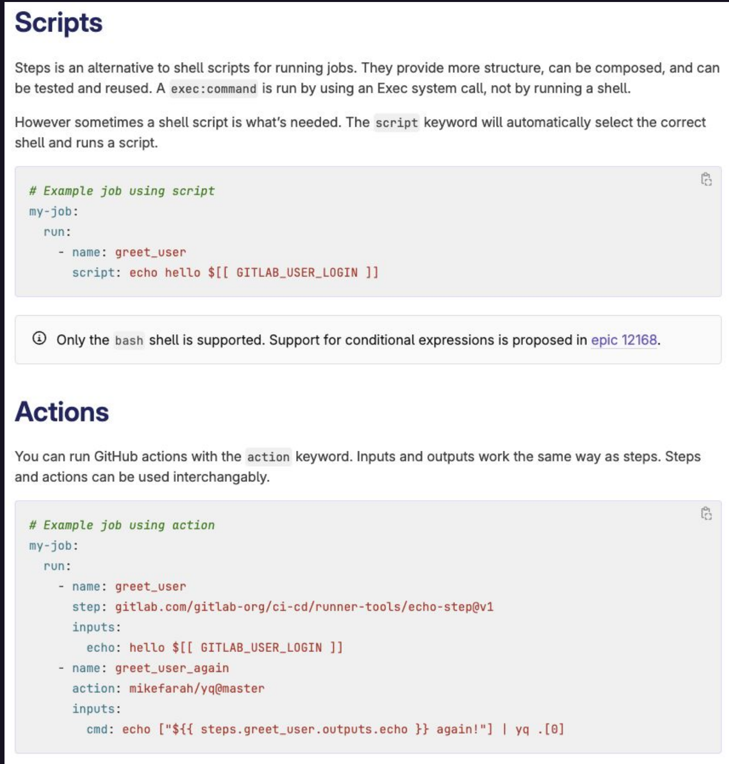


CI/CD Steps

Tier: Free, Premium, Ultimate
Offering: GitLab.com, Self-managed, GitLab Dedicated
Status: Experimental

Steps are reusable and composable pieces of a job. Each step defines structured inputs and outputs that can be consumed by other steps. Steps can come from local files, GitLab.com repositories, or any other Git source.

Support for a CI Catalog that publishes steps is proposed in issue 425891.



Scripts

Steps is an alternative to shell scripts for running jobs. They provide more structure, can be composed, and can be tested and reused. A `exec:command` is run by using an Exec system call, not by running a shell.

However sometimes a shell script is what's needed. The `script` keyword will automatically select the correct shell and runs a script.

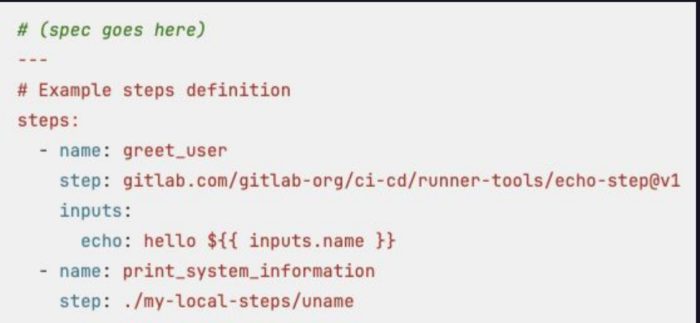
```
# Example job using script
my-job:
  run:
    - name: greet_user
      script: echo hello $[[ GITLAB_USER_LOGIN ]]
```

Only the `bash` shell is supported. Support for conditional expressions is proposed in epic 12168.

Actions

You can run GitHub actions with the `action` keyword. Inputs and outputs work the same way as steps. Steps and actions can be used interchangably.

```
# Example job using action
my-job:
  run:
    - name: greet_user
      step: gitlab.com/gitlab-org/ci-cd/runner-tools/echo-step@v1
      inputs:
        echo: hello $[[ GITLAB_USER_LOGIN ]]
    - name: greet_user_again
      action: mikefarah/yq@master
      inputs:
        cmd: echo ["${{ steps.greet_user.outputs.echo }} again!"] | yq .[0]
```



```
# (spec goes here)
---
# Example steps definition
steps:
  - name: greet_user
    step: gitlab.com/gitlab-org/ci-cd/runner-tools/echo-step@v1
    inputs:
      echo: hello ${{ inputs.name }}
  - name: print_system_information
    step: ./my-local-steps/uname
```

GitLab

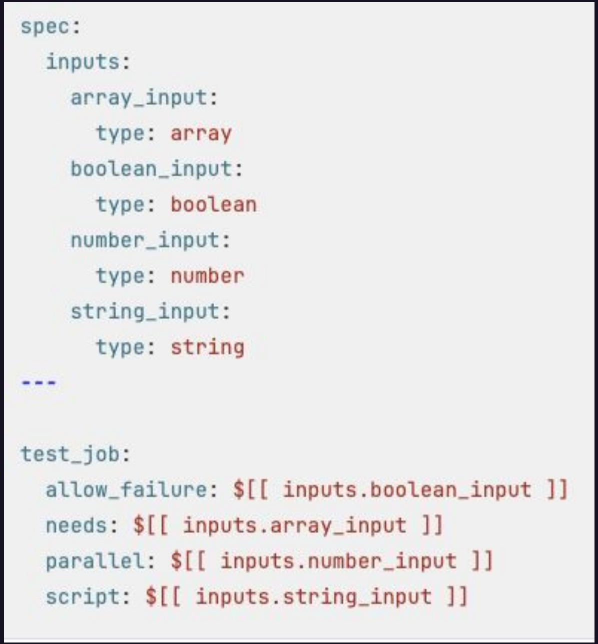
Efficiency tips

Best practices

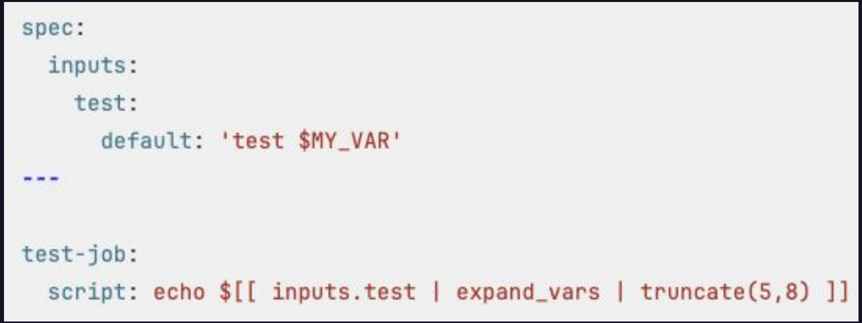
Input types

1. String (default)
   → stage, image, script
2. Number
   → parallel
3. Boolean
   → allow_failure
4. Array
   → needs, rules
5. Functions to manipulate values

https://docs.gitlab.com/ee/ci/yaml/inputs.html#input-types
https://docs.gitlab.com/ee/ci/yaml/inputs.html#specify-functions-to-manipulate-input-values
https://docs.gitlab.com/ee/ci/yaml/#job-keywords



```yaml
spec:
  inputs:
    array_input:
      type: array
    boolean_input:
      type: boolean
    number_input:
      type: number
    string_input:
      type: string
---

test_job:
  allow_failure: $[[ inputs.boolean_input ]]
  needs: $[[ inputs.array_input ]]
  parallel: $[[ inputs.number_input ]]
  script: $[[ inputs.string_input ]]
```



```yaml
spec:
  inputs:
    test:
      default: 'test $MY_VAR'
---

test-job:
  script: echo $[[ inputs.test | expand_vars | truncate(5,8) ]]
```

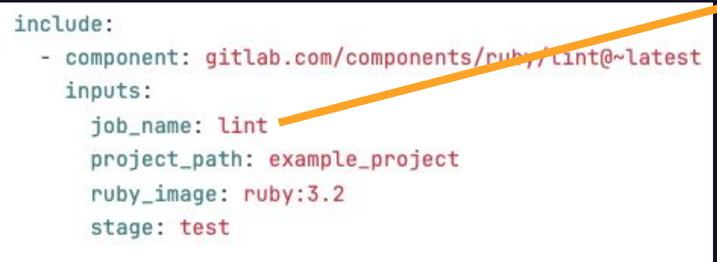
Dynamic inputs example

Arrays:

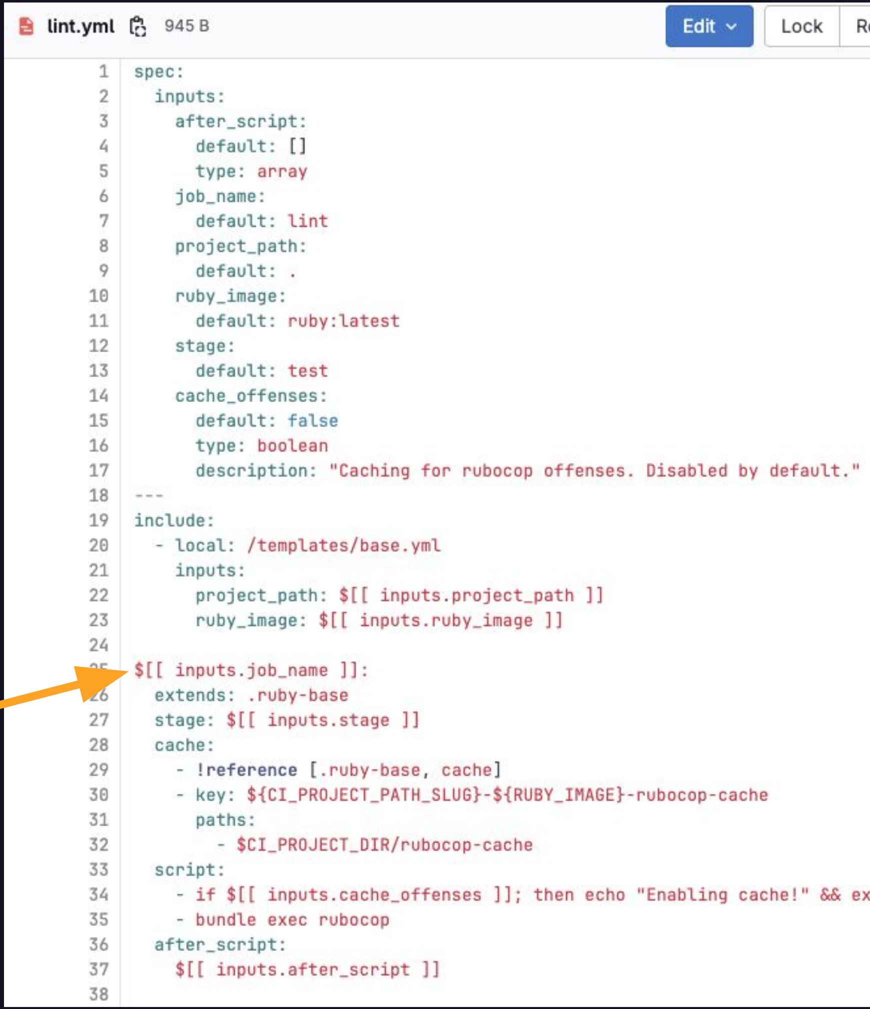
1. After script execution
   https://gitlab.com/components/ruby
2. Cache paths
   https://gitlab.com/explore/catalog/rust-ci/rust-ci

String:

1. Container image
   https://gitlab.com/explore/catalog/to-be-continuous/python
2. Dynamic job name
   https://gitlab.com/components/ruby



```yaml
include:
  - component: gitlab.com/components/ruby/lint@~latest
    inputs:
      job_name: lint
      project_path: example_project
      ruby_image: ruby:3.2
      stage: test
```



lint.yml 945 B Edit Lock

```yaml
spec:
  inputs:
    after_script:
      default: []
      type: array
    job_name:
      default: lint
    project_path:
      default: .
    ruby_image:
      default: ruby:latest
    stage:
      default: test
    cache_offenses:
      default: false
      type: boolean
      description: "Caching for rubocop offenses. Disabled by default."
---
include:
  - local: /templates/base.yml
    inputs:
      project_path: $[[ inputs.project_path ]]
      ruby_image: $[[ inputs.ruby_image ]]

$[[ inputs.job_name ]]:
  extends: .ruby-base
  stage: $[[ inputs.stage ]]
  cache:
    - !reference [.ruby-base, cache]
    - key: ${CI_PROJECT_PATH_SLUG}-${RUBY_IMAGE}-rubocop-cache
      paths:
        - $CI_PROJECT_DIR/rubocop-cache
  script:
    - if $[[ inputs.cache_offenses ]]; then echo "Enabling cache!" && ex
    - bundle exec rubocop
  after_script:
    $[[ inputs.after_script ]]
```

Security best practices

For component users

1. Use pinned versions
2. Store secrets securely
3. Securely handle cache and artifacts

4. Review CI/CD component changes
5. Audit and review component source code
6. Minimize access to credentials and tokens

For component maintainers

1. Discourage using latest
2. Use two-factor authentication (2FA)
3. Use protected branches
4. Review changes carefully

Photo by Michael Rodock on Unsplash

GitLab

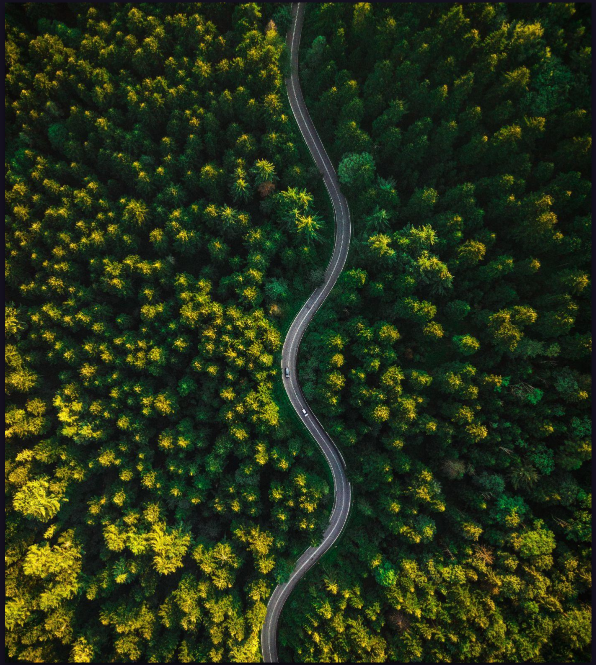
Benefits

Component highlights
and use cases

Photo by Tim Marshall on Unsplash

Building blocks

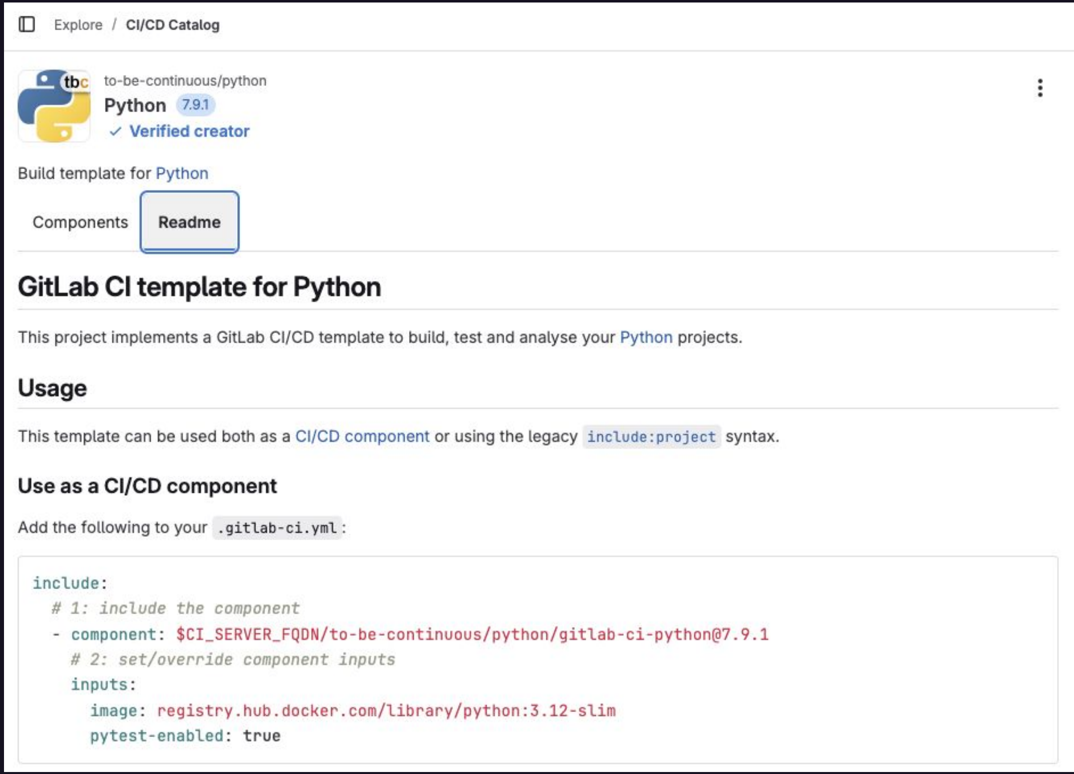
- Programming languages - lint, build, test best practices
- Efficiency best practices
- Dynamic workflows
- Version control, dependencies, automated tests
- Single source of Truth and Visibility



Verified creator components

- Highlight: to-be-continuous - Python

https://gitlab.com/explore/catalog/to-be-continuous/python



Build template for Python

GitLab CI template for Python

This project implements a GitLab CI/CD template to build, test and analyse your Python projects.

Usage

This template can be used both as a CI/CD component or using the legacy `include:project` syntax.

Use as a CI/CD component

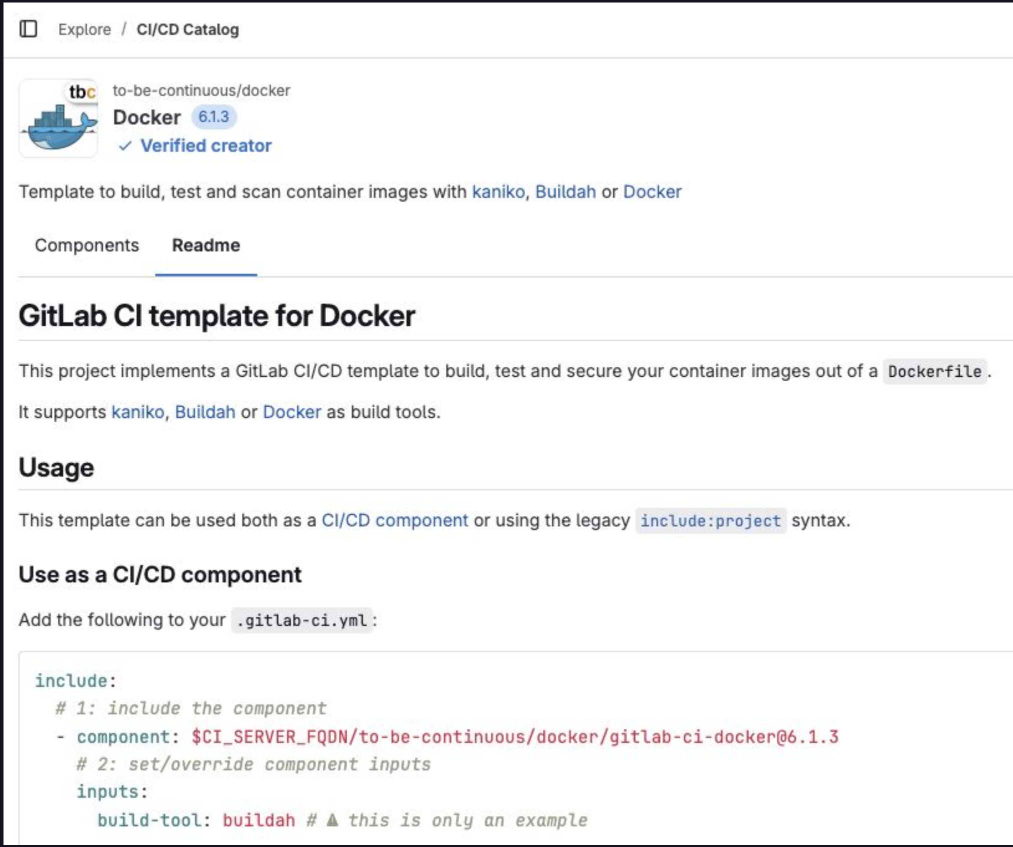
Add the following to your `.gitlab-ci.yml`:

```yaml
include:
  # 1: include the component
  - component: $CI_SERVER_FQDN/to-be-continuous/python/gitlab-ci-python@7.9.1
    # 2: set/override component inputs
    inputs:
      image: registry.hub.docker.com/library/python:3.12-slim
      pytest-enabled: true
```

Verified creator components

- Highlight: to-be-continuous - Docker

https://gitlab.com/explore/catalog/to-be-continuous/docker



Template to build, test and scan container images with kaniko, Buildah or Docker

GitLab CI template for Docker

This project implements a GitLab CI/CD template to build, test and secure your container images out of a `Dockerfile`.

It supports kaniko, Buildah or Docker as build tools.

Usage

This template can be used both as a CI/CD component or using the legacy `include:project` syntax.

Use as a CI/CD component

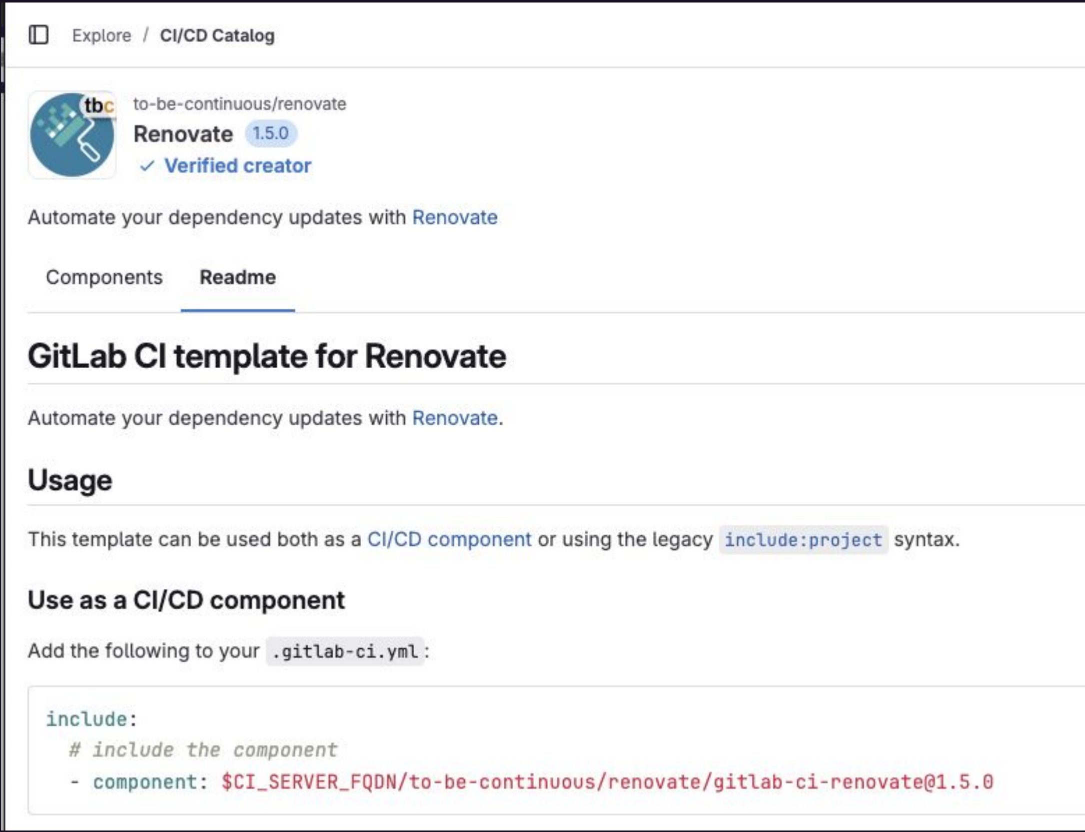
Add the following to your `.gitlab-ci.yml`:

```
include:
  # 1: include the component
  - component: $CI_SERVER_FQDN/to-be-continuous/docker/gitlab-ci-docker@6.1.3
    # 2: set/override component inputs
    inputs:
      build-tool: buildah # ⚠ this is only an example
```

Verified creator components

Highlight: to-be-continuous - Renovate

https://gitlab.com/explore/catalog/to-be-continuous/renovate



to-be-continuous/renovate

Renovate 1.5.0

✓ Verified creator

Automate your dependency updates with Renovate

Components Readme

GitLab CI template for Renovate

Automate your dependency updates with Renovate.

Usage

This template can be used both as a CI/CD component or using the legacy `include:project` syntax.

Use as a CI/CD component

Add the following to your `.gitlab-ci.yml`:

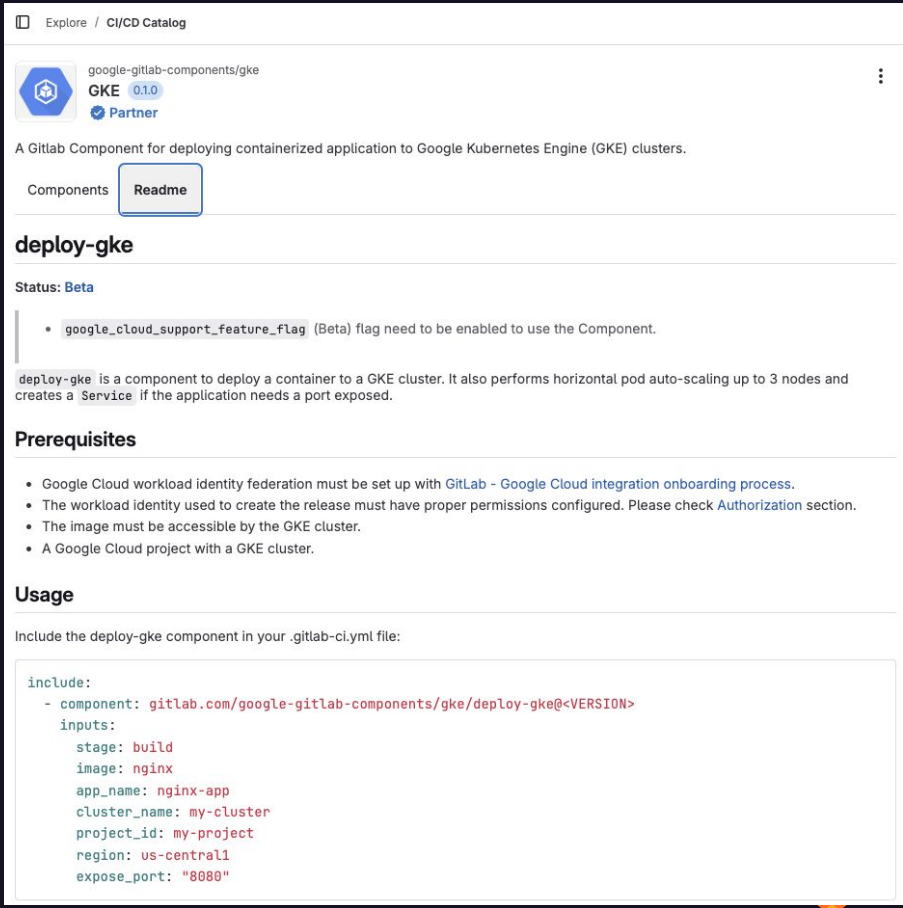
```yaml
include:
  # include the component
  - component: $CI_SERVER_FQDN/to-be-continuous/renovate/gitlab-ci-renovate@1.5.0
```

Partner components

Highlight: Google - GKE

https://gitlab.com/explore/catalog/google-gitlab-components/gke

Learn more: https://about.gitlab.com/blog/2024/04/09/gitlab-google-cloud-integrations-now-in-public-beta/#automate-cicd



A Gitlab Component for deploying containerized application to Google Kubernetes Engine (GKE) clusters.

deploy-gke

Status: Beta

- `google_cloud_support_feature_flag` (Beta) flag need to be enabled to use the Component.

`deploy-gke` is a component to deploy a container to a GKE cluster. It also performs horizontal pod auto-scaling up to 3 nodes and creates a `Service` if the application needs a port exposed.

Prerequisites

- Google Cloud workload identity federation must be set up with GitLab - Google Cloud integration onboarding process.
- The workload identity used to create the release must have proper permissions configured. Please check Authorization section.
- The image must be accessible by the GKE cluster.
- A Google Cloud project with a GKE cluster.

Usage

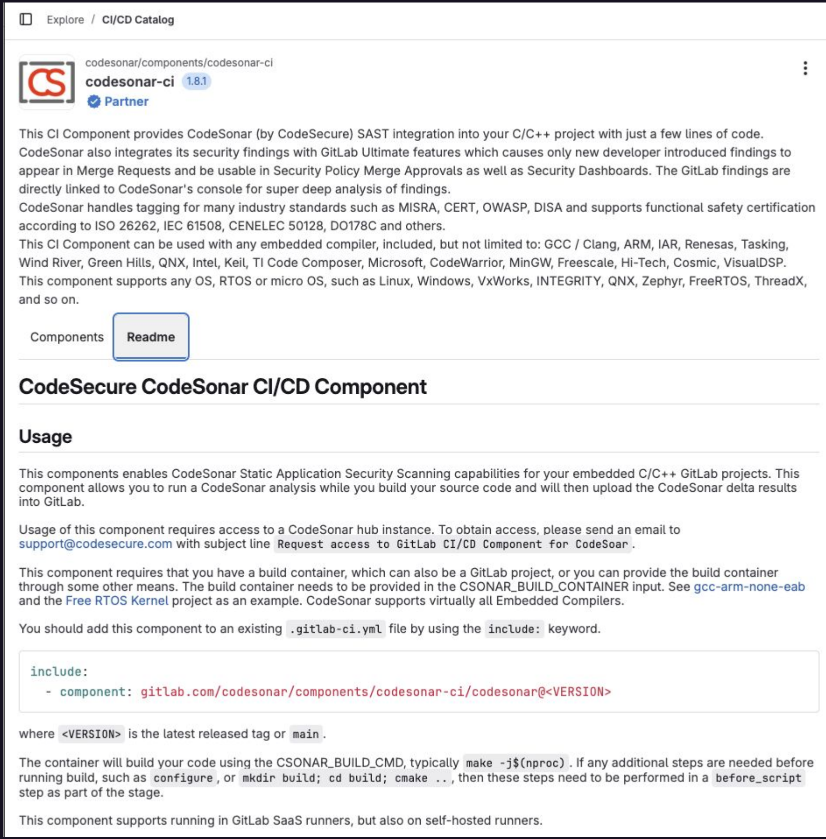
Include the deploy-gke component in your .gitlab-ci.yml file:

```yaml
include:
  - component: gitlab.com/google-gitlab-components/gke/deploy-gke@<VERSION>
    inputs:
      stage: build
      image: nginx
      app_name: nginx-app
      cluster_name: my-cluster
      project_id: my-project
      region: us-central1
      expose_port: "8080"
```

Partner components

Highlight: CodeSonar CodeSecure - Embedded Security scanning

https://gitlab.com/explore/catalog/codesonar/components/codesonar-ci


Explore / CI/CD Catalog

codesonar/components/codesonar-ci
codesonar-ci 1.8.1
Partner

This CI Component provides CodeSonar (by CodeSecure) SAST integration into your C/C++ project with just a few lines of code. CodeSonar also integrates its security findings with GitLab Ultimate features which causes only new developer introduced findings to appear in Merge Requests and be usable in Security Policy Merge Approvals as well as Security Dashboards. The GitLab findings are directly linked to CodeSonar's console for super deep analysis of findings.
CodeSonar handles tagging for many industry standards such as MISRA, CERT, OWASP, DISA and supports functional safety certification according to ISO 26262, IEC 61508, CENELEC 50128, DO178C and others.
This CI Component can be used with any embedded compiler, included, but not limited to: GCC / Clang, ARM, IAR, Renesas, Tasking, Wind River, Green Hills, QNX, Intel, Keil, TI Code Composer, Microsoft, CodeWarrior, MinGW, Freescale, Hi-Tech, Cosmic, VisualDSP.
This component supports any OS, RTOS or micro OS, such as Linux, Windows, VxWorks, INTEGRITY, QNX, Zephyr, FreeRTOS, ThreadX, and so on.

Components | Readme

CodeSecure CodeSonar CI/CD Component

Usage

This components enables CodeSonar Static Application Security Scanning capabilities for your embedded C/C++ GitLab projects. This component allows you to run a CodeSonar analysis while you build your source code and will then upload the CodeSonar delta results into GitLab.

Usage of this component requires access to a CodeSonar hub instance. To obtain access, please send an email to support@codesecure.com with subject line `Request access to GitLab CI/CD Component for CodeSoar`.

This component requires that you have a build container, which can also be a GitLab project, or you can provide the build container through some other means. The build container needs to be provided in the CSONAR_BUILD_CONTAINER input. See gcc-arm-none-eab and the Free RTOS Kernel project as an example. CodeSonar supports virtually all Embedded Compilers.

You should add this component to an existing `.gitlab-ci.yml` file by using the `include:` keyword.

```
include:
  - component: gitlab.com/codesonar/components/codesonar-ci/codesonar@<VERSION>
```

where `<VERSION>` is the latest released tag or `main`.

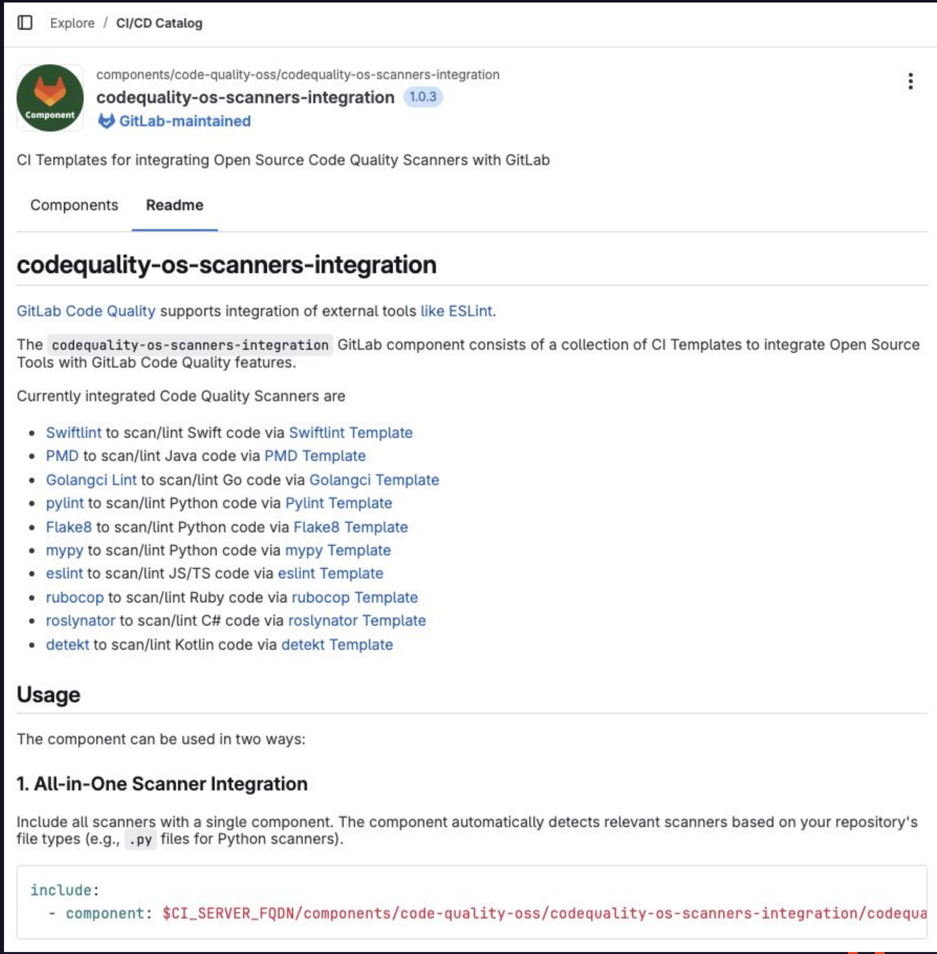
The container will build your code using the CSONAR_BUILD_CMD, typically `make -j$(nproc)`. If any additional steps are needed before running build, such as `configure`, or `mkdir build; cd build; cmake ..`, then these steps need to be performed in a `before_script` step as part of the stage.

This component supports running in GitLab SaaS runners, but also on self-hosted runners.

GitLab-maintained components

Highlight: Code Quality OSS

https://gitlab.com/explore/catalog/components/code-quality-oss/codequality-os-scanners-integration



components/code-quality-oss/codequality-os-scanners-integration

codequality-os-scanners-integration 1.0.3

GitLab-maintained

CI Templates for integrating Open Source Code Quality Scanners with GitLab

codequality-os-scanners-integration

GitLab Code Quality supports integration of external tools like ESLint.

The `codequality-os-scanners-integration` GitLab component consists of a collection of CI Templates to integrate Open Source Tools with GitLab Code Quality features.

Currently integrated Code Quality Scanners are

- Swiftlint to scan/lint Swift code via Swiftlint Template
- PMD to scan/lint Java code via PMD Template
- Golangci Lint to scan/lint Go code via Golangci Template
- pylint to scan/lint Python code via Pylint Template
- Flake8 to scan/lint Python code via Flake8 Template
- mypy to scan/lint Python code via mypy Template
- eslint to scan/lint JS/TS code via eslint Template
- rubocop to scan/lint Ruby code via rubocop Template
- roslynator to scan/lint C# code via roslynator Template
- detekt to scan/lint Kotlin code via detekt Template

Usage

The component can be used in two ways:

1. All-in-One Scanner Integration

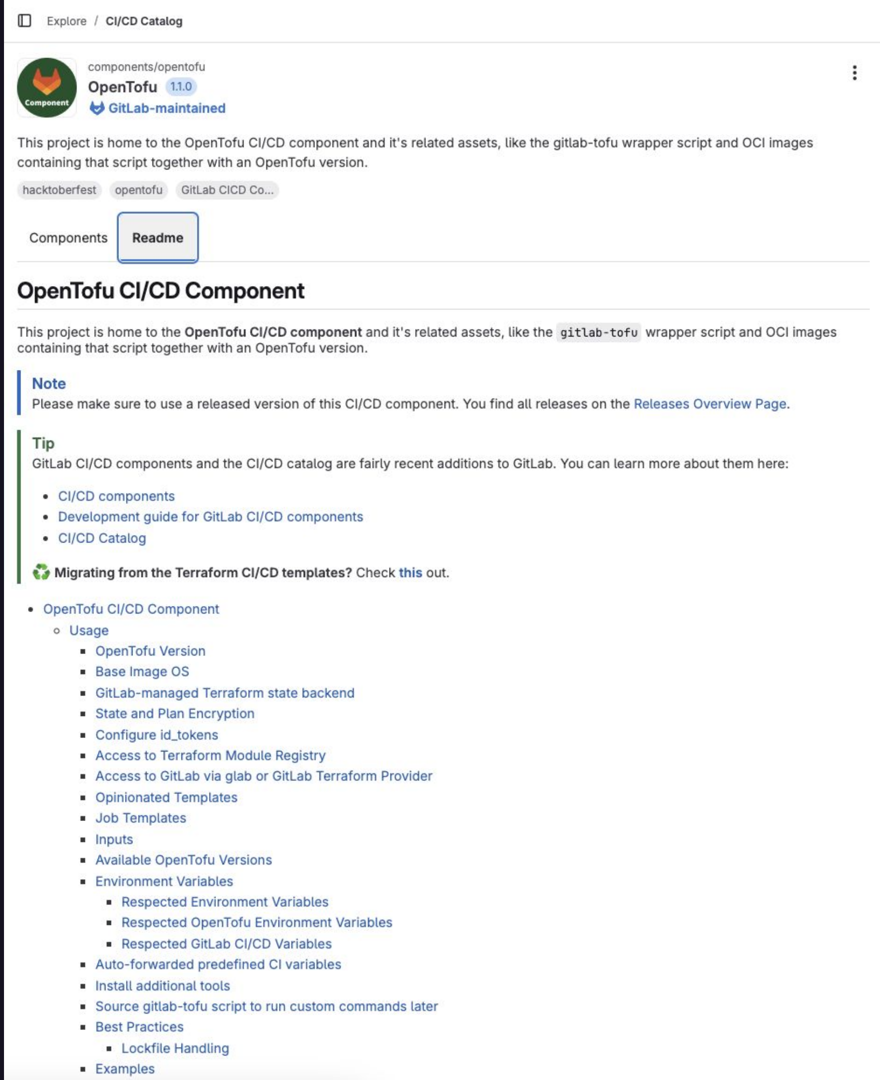
Include all scanners with a single component. The component automatically detects relevant scanners based on your repository's file types (e.g., `.py` files for Python scanners).

```
include:
  - component: $CI_SERVER_FQDN/components/code-quality-oss/codequality-os-scanners-integration/codequa
```

GitLab-maintained components

Highlight: OpenTofu

https://gitlab.com/explore/catalog/components/opentofu



This project is home to the OpenTofu CI/CD component and it's related assets, like the gitlab-tofu wrapper script and OCI images containing that script together with an OpenTofu version.

OpenTofu CI/CD Component

This project is home to the **OpenTofu CI/CD component** and it's related assets, like the `gitlab-tofu` wrapper script and OCI images containing that script together with an OpenTofu version.

Note
Please make sure to use a released version of this CI/CD component. You find all releases on the Releases Overview Page.

Tip
GitLab CI/CD components and the CI/CD catalog are fairly recent additions to GitLab. You can learn more about them here:

- CI/CD components
- Development guide for GitLab CI/CD components
- CI/CD Catalog

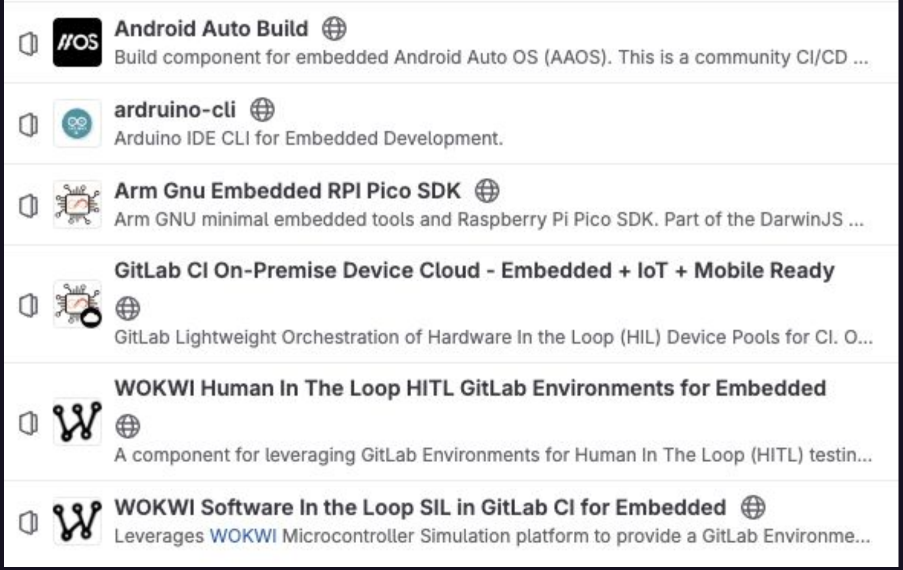
♻️ **Migrating from the Terraform CI/CD templates?** Check this out.

Use case: Hardware in the loop

Use: GitLab CI Components for Embedded

Learn: Embedded DevOps Workshop - A Self-Paced POC for Refactoring to GitLab CI and Modern Security and Compliance

Watch: Embedded DevOps Hardware in the CI Loop and The Transformative Power of Sharing Work-in-Progress


Android Auto Build
Build component for embedded Android Auto OS (AAOS). This is a community CI/CD ...

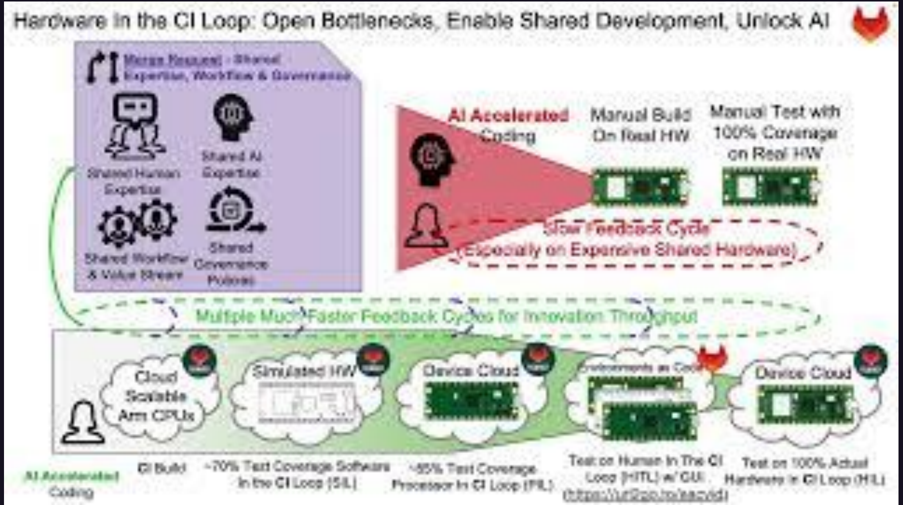
ardruino-cli
Arduino IDE CLI for Embedded Development.

Arm Gnu Embedded RPI Pico SDK
Arm GNU minimal embedded tools and Raspberry Pi Pico SDK. Part of the DarwinJS ...

GitLab CI On-Premise Device Cloud - Embedded + IoT + Mobile Ready
GitLab Lightweight Orchestration of Hardware In the Loop (HIL) Device Pools for CI. O...

WOKWI Human In The Loop HITL GitLab Environments for Embedded
A component for leveraging GitLab Environments for Human In The Loop (HITL) testin...

WOKWI Software In the Loop SIL in GitLab CI for Embedded
Leverages WOKWI Microcontroller Simulation platform to provide a GitLab Environme...


Hardware in the CI Loop: Open Bottlenecks, Enable Shared Development, Unlock AI
Merge Request - Shared Expertise, Workflow & Governance
Shared Human Expertise
Shared AI Expertise
Shared Workflow & Value Stream
Shared Governance Policies
AI Accelerated Coding
Manual Build On Real HW
Manual Test with 100% Coverage on Real HW
Slow Feedback Cycle (Especially on Expensive Shared Hardware)
Multiple Much Faster Feedback Cycles for Innovation Throughput
Cloud Scalable Arm CPUs
Simulated HW
Device Cloud
Environments as Code
Device Cloud
AI Accelerated Coding
CI Build
~70% Test Coverage Software In the CI Loop (SIL)
~85% Test Coverage Processor In CI Loop (PIL)
Test on Human In The CI Loop (HITL) w/ GUI
Test on 100% Actual Hardware in CI Loop (HIL)

More use cases

Get inspired to create your own CI/CD components:

1. Container image builds: Docker-in-Docker, podman, Kaniko
2. Programming language workflows including best practices
3. Platform engineering, developer experience
4. Continuous releases and deployments
5. IaC and Observability
6. Security scanning, Supply Chain Security, SLSA, SBOM
7. Embedded, automotive, telco, aerospace, medical

GitLab

Everyone can contribute

Create

Photo by Johannes Andersson on Unsplash

Start today

- Create a component project
- Convert CI/CD templates into components
- Replace hardcoded values with inputs
- Development guide for GitLab CI/CD components
- Join our community: Forum, Discord

FAQ

1. CI/CD Templates are deprecated?
   a. No. They are used within component types, too.
2. Where to start with components?
   a. Start where pipelines failing for 1+ teams
   b. Review more pipelines and find common patterns and usages
   c. Contribute to the GitLab CI/CD Catalog locally and at GitLab
   d. Share your feedback in GitLab.com epics and issues!
3. Don't get confused about steps, they are still experimental
4. Visit the GitLab blog - useful resources like FAQs and more
5. More interest? Follow this epic

Meme by David Bell on Twitter/X

GitLab

Migration workshop

Optional: Go CI/CD template migration practical example

Migration workshop: Go

Analyze existing CI/CD template

Split jobs into specific component templates

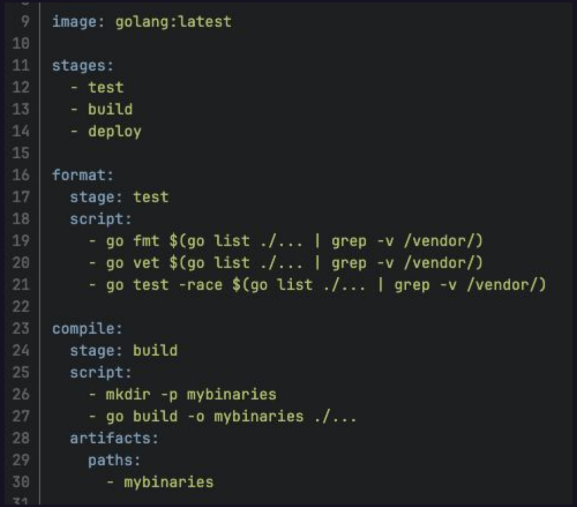
Optimize with dynamic inputs

Test component with source code

Release component

1 Analyze existing CI/CD template

1. The *image* configuration is *global*.
   → Needs to be moved into job definition.
2. The *format* job runs multiple go commands, including *go test*
   → Split the jobs into *format* and *test*
3. The *compile* job runs *go build*.
   → Rename job.

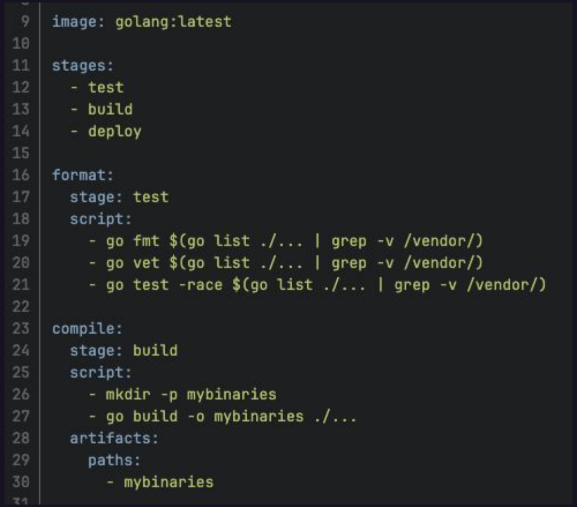


```yaml
image: golang:latest

stages:
  - test
  - build
  - deploy

format:
  stage: test
  script:
    - go fmt $(go list ./... | grep -v /vendor/)
    - go vet $(go list ./... | grep -v /vendor/)
    - go test -race $(go list ./... | grep -v /vendor/)

compile:
  stage: build
  script:
    - mkdir -p mybinaries
    - go build -o mybinaries ./...
  artifacts:
    paths:
      - mybinaries
```

Source:
https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Go.gitlab-ci.yml

2 Define optimization strategies

1. The *stage* attribute is hardcoded.
   → Should be configurable
2. The *image* attribute hardcodes *latest* tag.
   → Make it a configurable input.
3. The *compile* job uses a hard-coded binary output path.
   → Make it configurable.

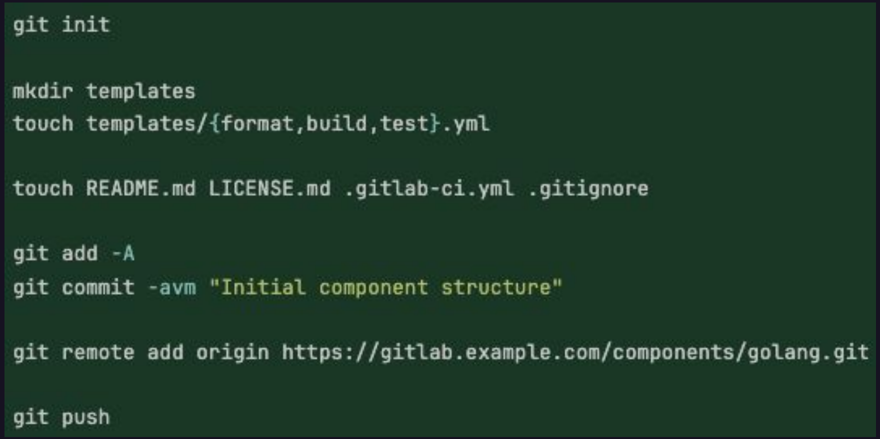


```yaml
image: golang:latest

stages:
  - test
  - build
  - deploy

format:
  stage: test
  script:
    - go fmt $(go list ./... | grep -v /vendor/)
    - go vet $(go list ./... | grep -v /vendor/)
    - go test -race $(go list ./... | grep -v /vendor/)

compile:
  stage: build
  script:
    - mkdir -p mybinaries
    - go build -o mybinaries ./...
  artifacts:
    paths:
      - mybinaries
```

Source:
https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Go.gitlab-ci.yml

3 Create template directory structure

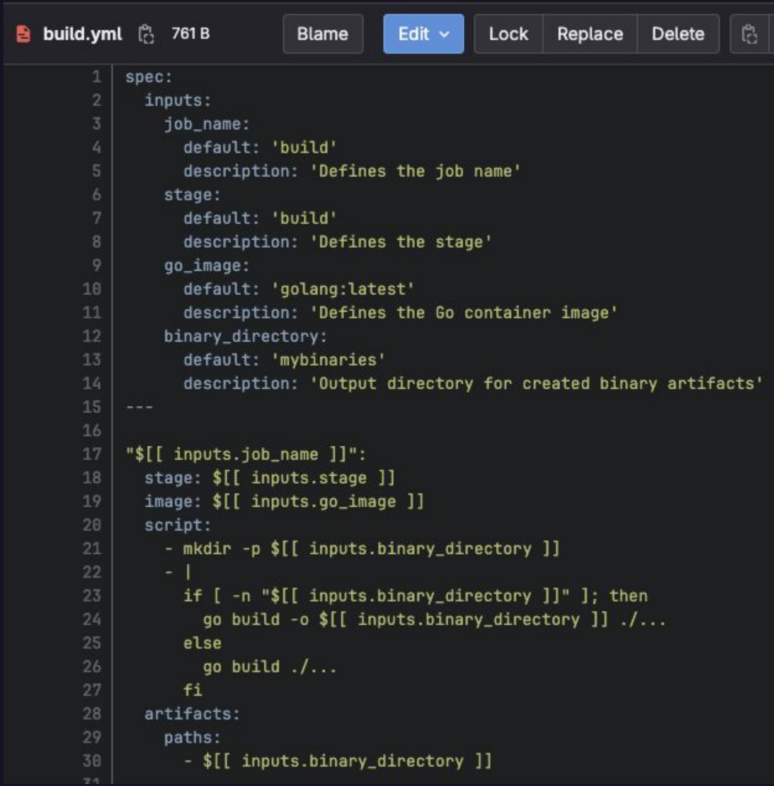
1. One template file for each job: *format, build, test*
2. Create a project, initialize a Git repository
3. Create additional best practice files



```
git init

mkdir templates
touch templates/{format,build,test}.yml

touch README.md LICENSE.md .gitlab-ci.yml .gitignore

git add -A
git commit -avm "Initial component structure"

git remote add origin https://gitlab.example.com/components/golang.git

git push
```

4 CI/CD job templates: build

1. Define inputs: *job_name*, *stage*, *go_image*, *binary_directory*
2. Add **dynamic job name** definition, using *inputs.job_name*
3. Assign *stage* to *inputs.stage*
4. Use *image* from *inputs.go_image*
5. Create binary directory from *inputs.binary_directory*, add to *go build*
6. Define the artifacts path to *inputs.binary_directory*

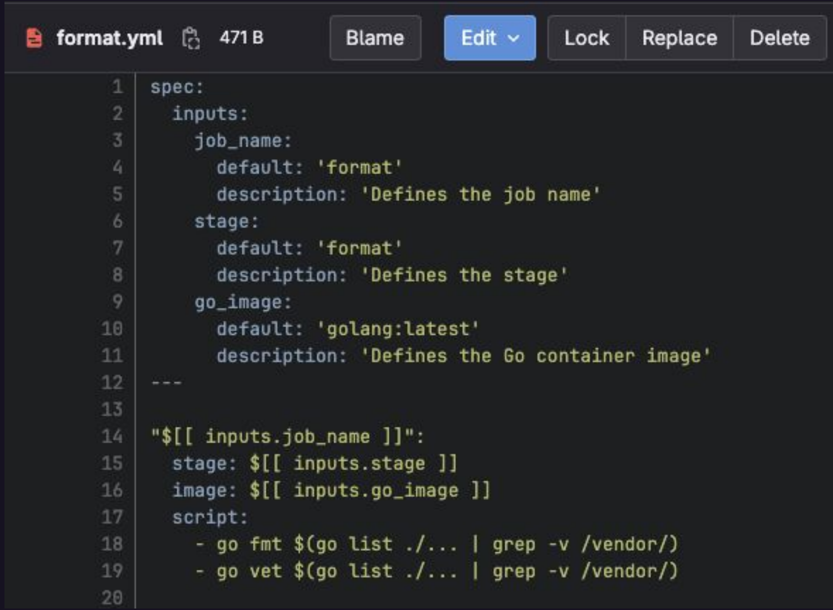


```yaml
spec:
  inputs:
    job_name:
      default: 'build'
      description: 'Defines the job name'
    stage:
      default: 'build'
      description: 'Defines the stage'
    go_image:
      default: 'golang:latest'
      description: 'Defines the Go container image'
    binary_directory:
      default: 'mybinaries'
      description: 'Output directory for created binary artifacts'
---

"$[[ inputs.job_name ]]":
  stage: $[[ inputs.stage ]]
  image: $[[ inputs.go_image ]]
  script:
    - mkdir -p $[[ inputs.binary_directory ]]
    - |
      if [ -n "$[[ inputs.binary_directory ]]" ]; then
        go build -o $[[ inputs.binary_directory ]] ./...
      else
        go build ./...
      fi
  artifacts:
    paths:
      - $[[ inputs.binary_directory ]]
```

Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

5 CI/CD job template: format

1. Follow the same pattern:
2. Inputs: *job_name*, *stage* and *go_image*
   a. Default values, description
3. Dynamic job name, stage, image

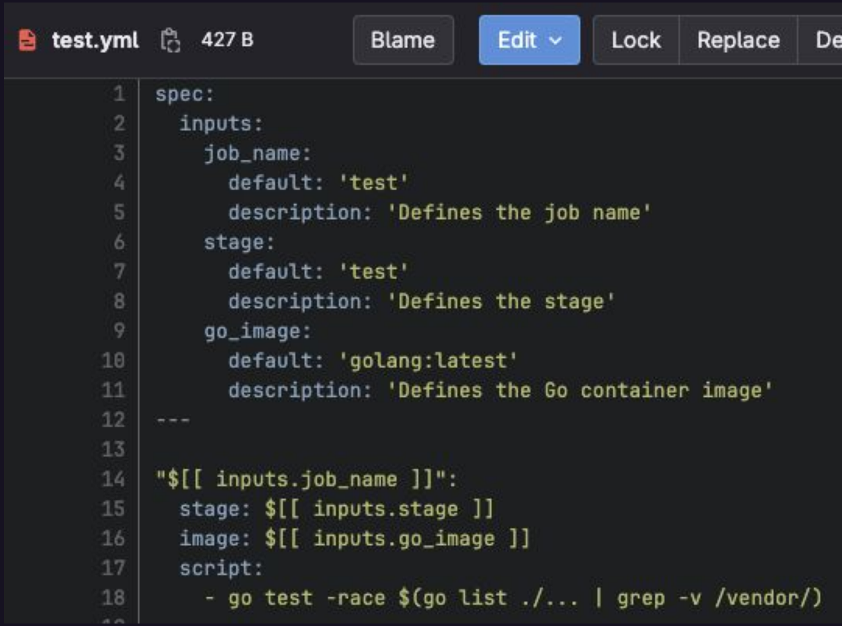


```yaml
spec:
  inputs:
    job_name:
      default: 'format'
      description: 'Defines the job name'
    stage:
      default: 'format'
      description: 'Defines the stage'
    go_image:
      default: 'golang:latest'
      description: 'Defines the Go container image'
---

"$[[ inputs.job_name ]]":
  stage: $[[ inputs.stage ]]
  image: $[[ inputs.go_image ]]
  script:
    - go fmt $(go list ./... | grep -v /vendor/)
    - go vet $(go list ./... | grep -v /vendor/)
```

Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

6 CI/CD job template: test

1. Follow the same pattern:
2. Inputs: *job_name*, *stage* and *go_image*
   a. Default values, description
3. Dynamic job name, stage, image

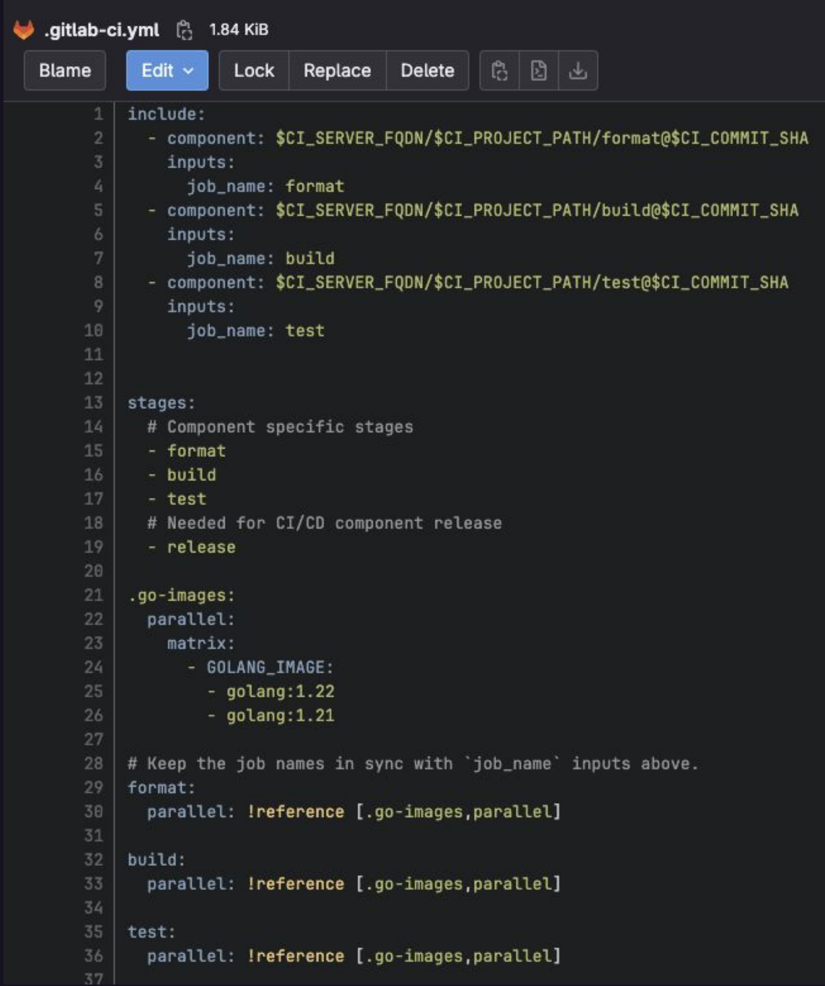


```yaml
spec:
  inputs:
    job_name:
      default: 'test'
      description: 'Defines the job name'
    stage:
      default: 'test'
      description: 'Defines the stage'
    go_image:
      default: 'golang:latest'
      description: 'Defines the Go container image'
---

"$[[ inputs.job_name ]]":
  stage: $[[ inputs.stage ]]
  image: $[[ inputs.go_image ]]
  script:
    - go test -race $(go list ./... | grep -v /vendor/)
```

Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

7 Test CI/CD component

1. Edit .gitlab-ci.yml and add tests
2. Specify multiple input values for *job_name*, *stage*, *image*
   a. Use parallel:matrix for Go images
3. CI_SERVER_FQDN automatically detects your instance URL


.gitlab-ci.yml 1.84 KiB

```yaml
include:
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/format@$CI_COMMIT_SHA
    inputs:
      job_name: format
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/build@$CI_COMMIT_SHA
    inputs:
      job_name: build
  - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/test@$CI_COMMIT_SHA
    inputs:
      job_name: test


stages:
  # Component specific stages
  - format
  - build
  - test
  # Needed for CI/CD component release
  - release

.go-images:
  parallel:
    matrix:
      - GOLANG_IMAGE:
        - golang:1.22
        - golang:1.21

# Keep the job names in sync with `job_name` inputs above.
format:
  parallel: !reference [.go-images,parallel]

build:
  parallel: !reference [.go-images,parallel]

test:
  parallel: !reference [.go-images,parallel]

```

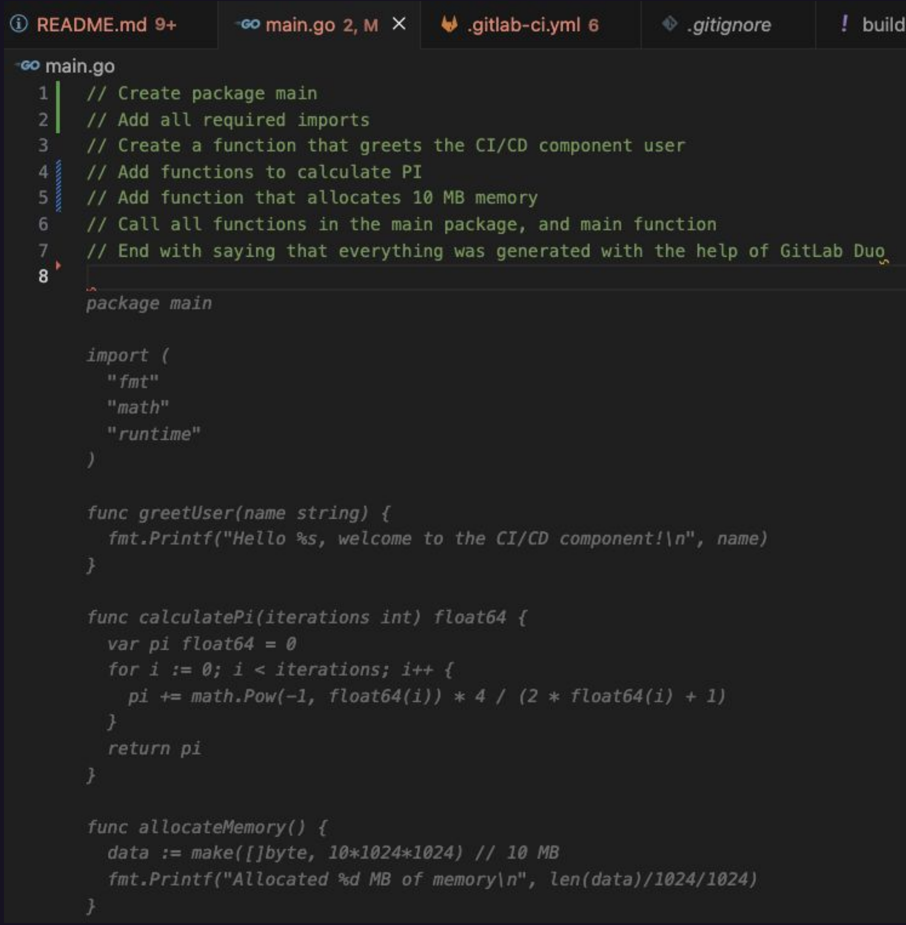
Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

8 Add Go source code

The *go* commands expect a Go project with go.mod, main.go

$ go mod init <project URL without https://>

💡 Tip: Use GitLab Duo Code Suggestions and use comments as prompt instructions to generate code.

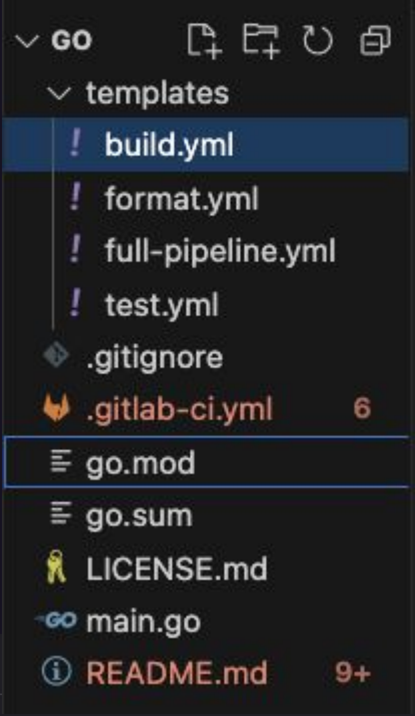


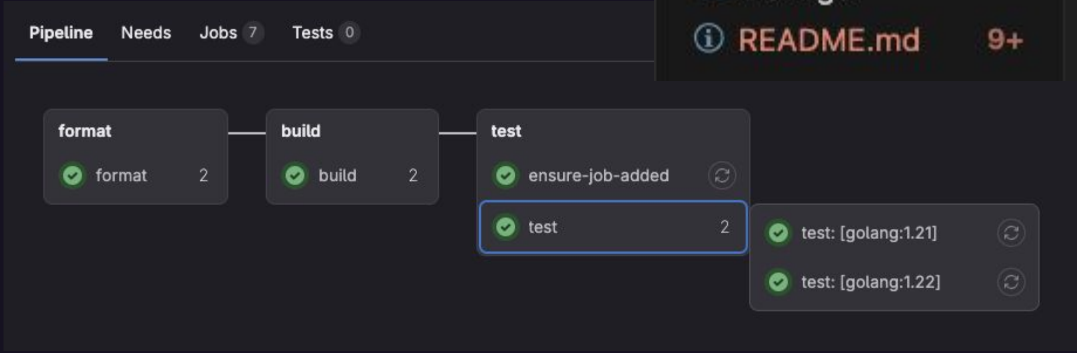
```
// Create package main
// Add all required imports
// Create a function that greets the CI/CD component user
// Add functions to calculate PI
// Add function that allocates 10 MB memory
// Call all functions in the main package, and main function
// End with saying that everything was generated with the help of GitLab Duo

package main

import (
  "fmt"
  "math"
  "runtime"
)

func greetUser(name string) {
  fmt.Printf("Hello %s, welcome to the CI/CD component!\n", name)
}

func calculatePi(iterations int) float64 {
  var pi float64 = 0
  for i := 0; i < iterations; i++ {
    pi += math.Pow(-1, float64(i)) * 4 / (2 * float64(i) + 1)
  }
  return pi
}

func allocateMemory() {
  data := make([]byte, 10*1024*1024) // 10 MB
  fmt.Printf("Allocated %d MB of memory\n", len(data)/1024/1024)
}
```

Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

9 Verify results, add documentation

1. Commit & push the change
2. Add docs best practices for Usage and Inputs
3. Release the component
4. Consume the component in staging/production





https://docs.gitlab.com/ee/ci/components/#best-practices

GitLab

DevSecOps Efficiency with CI/CD components

DevSecOps efficiency with a little help from CI/CD components

- Reusable, self-contained building blocks for GitLab CI/CD
- Visible and discoverable
- Share “best practice” pipeline jobs
- Formalized, and testable input parameters
- Create and consume
- Maintain, test and release
- Inspire collaboration and transparency

Photo by Diego PH on Unsplash

GitLab

Resources

Slides