GitLab

Efficient DevSecOps with CI/CD Components

Open Source @ Siemens, 2024-05-14

GitLab

Michael Friedrich

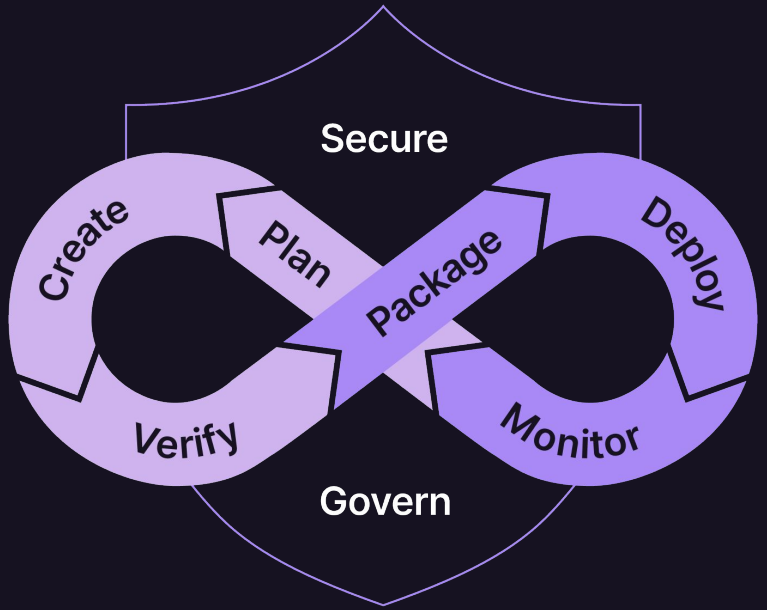
Senior Developer Advocate
@dnsmichi

GitLab

Efficient DevSecOps?

DevSecOps is all about speed

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



DevSecOps is all about speed

How fast are you able to

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

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

DevSecOps is all about speed

How to discover?

Google / LLM/RAG
Developer portal
Ask your peers

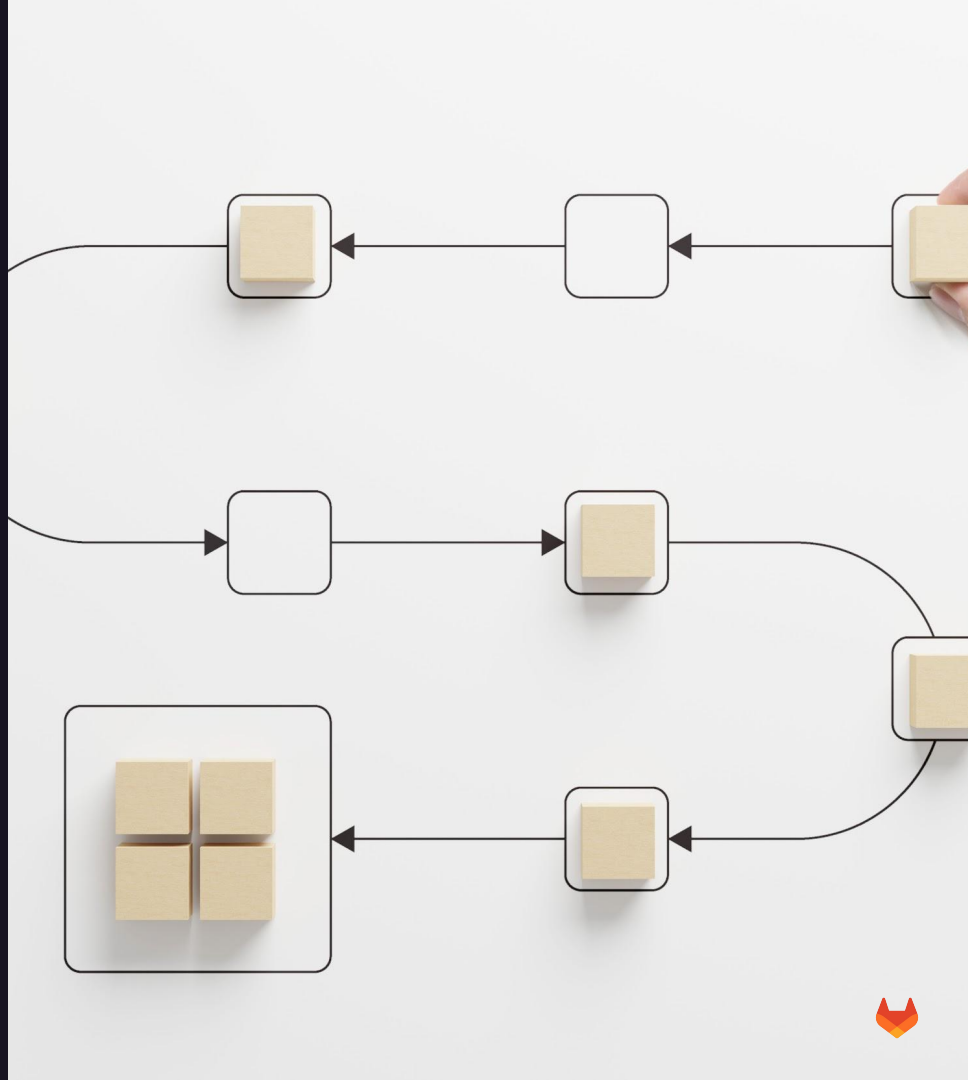
How to use?

Documentation
Wiki
Read the code

How to share?

Publish
Versioning
Tell it your peers

Other areas of improvement



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

```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
```


Traditional Pipeline Composition in GitLab CI/CD

Frequent problem: stage mismatch in consumer and include 😫

```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
```

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

Add Rust CI/CD templates
Michael Friedrich authored 2 hours 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

Go to the pipeline editor

Traditional Pipeline Composition in GitLab CI/CD

Solution 1: add the stage 🤨

```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
  - docs
  - deploy
```

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

Add Rust CI/CD templates
Michael Friedrich authored 2 hours 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

Traditional Pipeline Composition in GitLab CI/CD

Solution 2: override the job 🥲

```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

rust-docs:
  stage: build

stages:
  - test
  - build
  - deploy
```

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

Add Rust CI/CD templates
Michael Friedrich authored 2 hours 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/3

GitLab

CI/CD Components

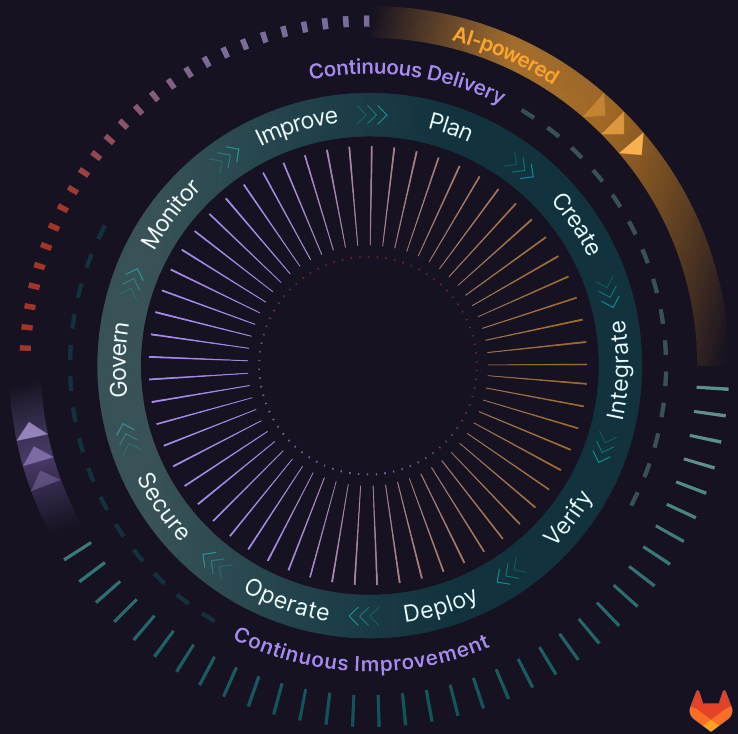
Streamline and automate pipeline creation

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. Testable
5. Release and publish with versioning
6. Allows others to contribute (via 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 pipeline configuration (exists today)
- **Components** - Reusable unit of pipeline configuration
- **Catalog** - Collection of projects that provide components, global instance scope

GitLab

Inside a component

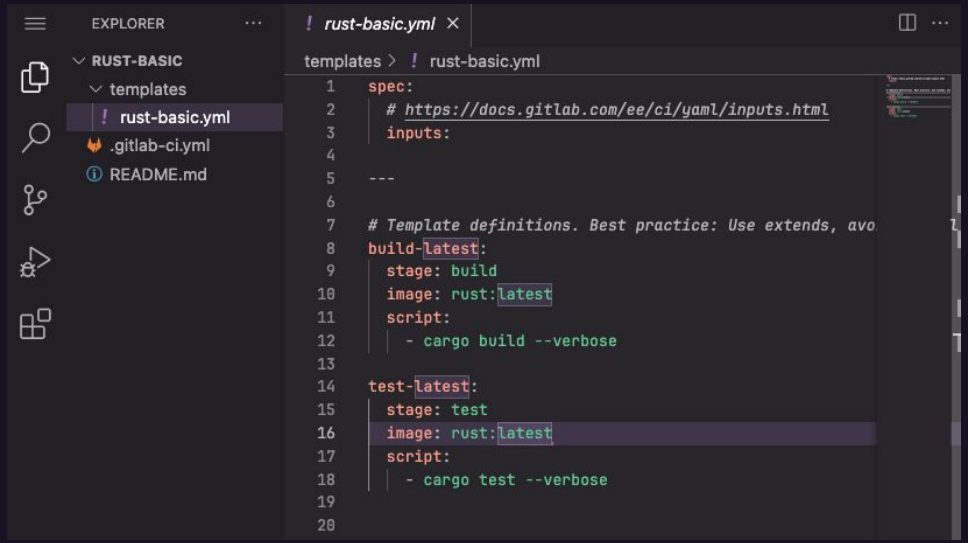
CI/CD component overview

Directory structure

Spec for **metadata** and **inputs**

Template definitions

Project in GitLab, release workflows



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” https://go.gitlab.com/kB4VQD

.gitlab-ci.yml 701 B

Edit | Lock | Replace | Delete

```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
```

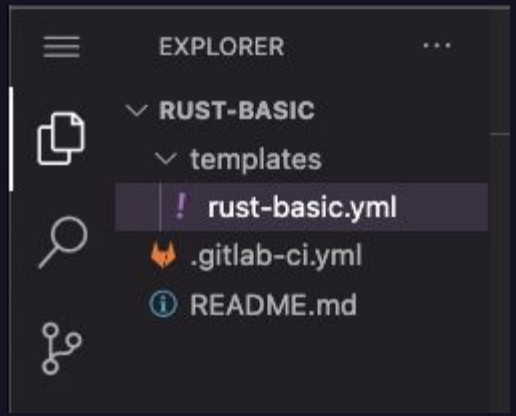
Basic Rust component

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

Create a GitLab project

Directory tree

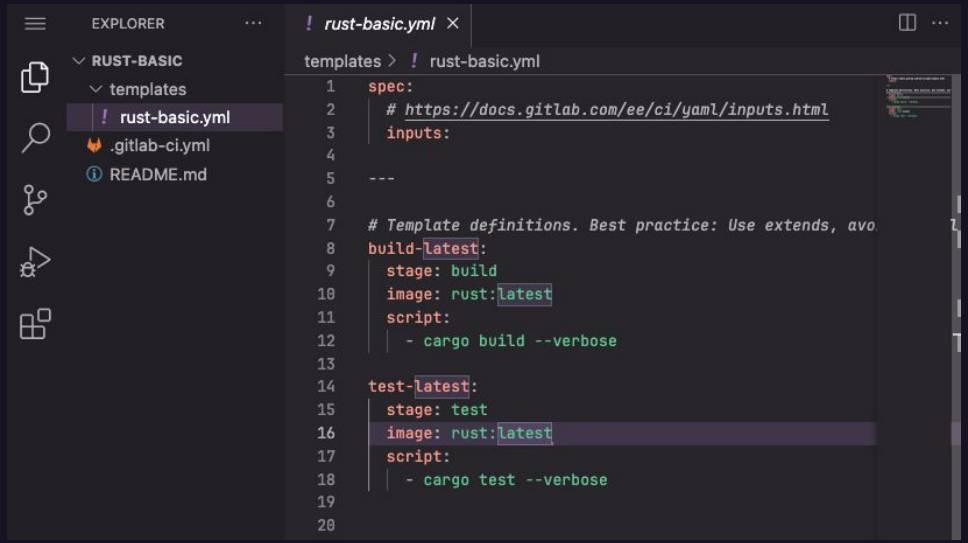
templates/ ← component type
  rust-basic.yml ← component template name
README.md ← How to use
.gitlab-ci.yml ← testing the component



CI/CD jobs in Rust component

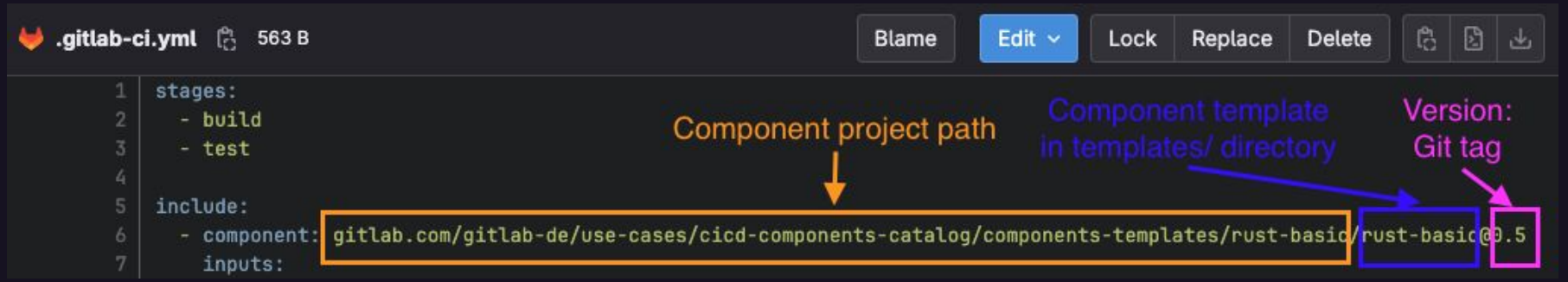
MVC (minimal viable change):

1. 💡 spec with empty inputs
2. 🌱 Two jobs
   a. stage
   b. image
   c. script
3. 🤔 No optimization yet



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

Consume Rust CI/CD component



The *include* keyword supports the *components* keyword.

Requires a component path:

example.gitlab.com/namespace/component/component-template-name@tagged-version

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

Dynamic inputs

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

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

Usage:

$[[inputs.*variable_name*]]

Example: Define default stage names for build/test

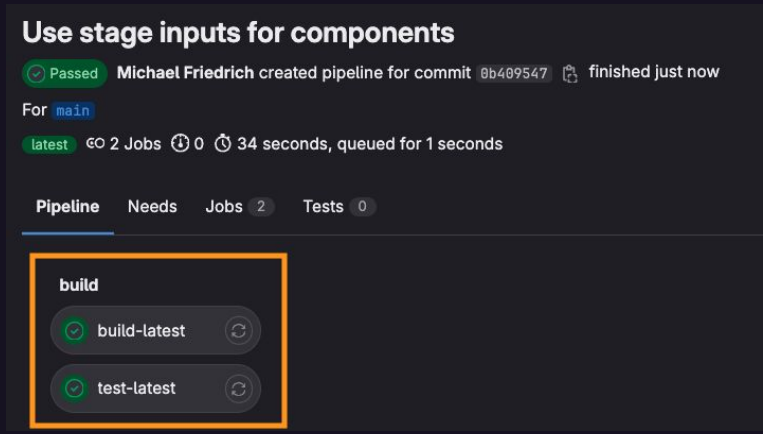
templates/rust-basic.yml

```
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'

---

@@ -27,13 +33,13 @@ variables:

# Template definitions. Best practice: Use extends,
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.



.gitlab-ci.yml

```yaml
stages:
  - build
  - test

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 test as input

Use image tags from https://hub.docker.com/_/rust/tags

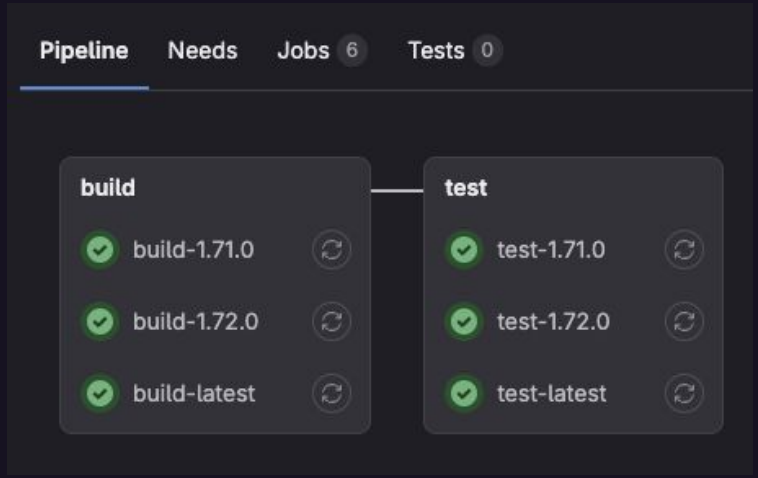
templates/rust-basic.yml +8 −3 Viewed

```
@@ -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



.gitlab-ci.yml 563 B

Blame | Edit | Lock | Replace | Delete

Component project path

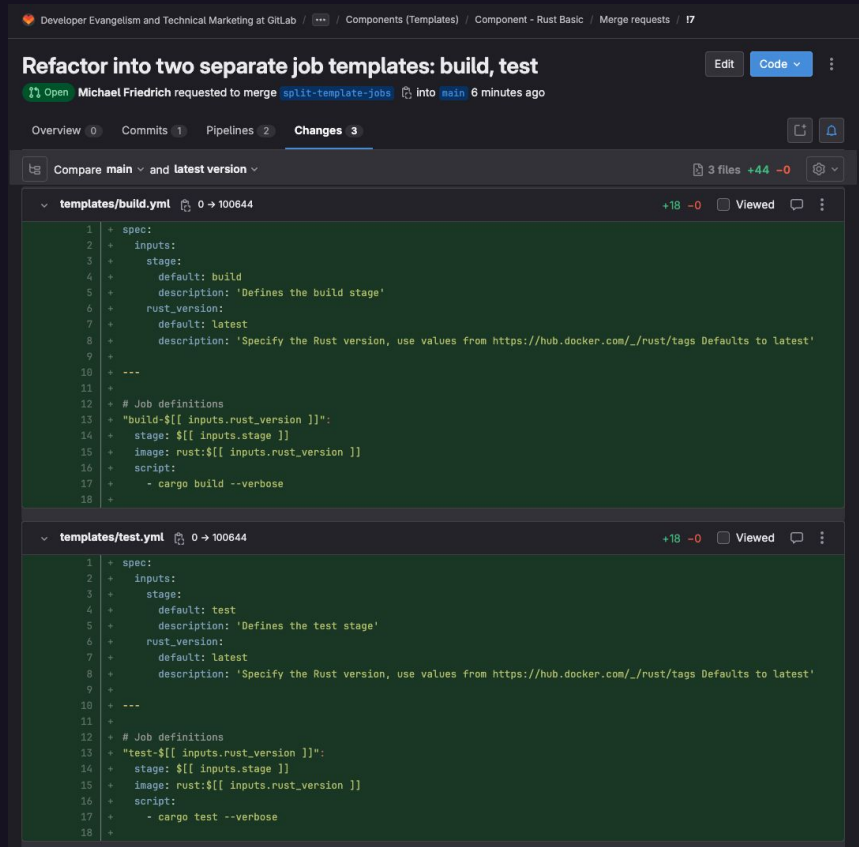
Component template in templates/ directory

Version: Git tag

```yaml
stages:
  - build
  - test

include:
  - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
    inputs:
      stage_build: build
      stage_test: test
      rust_version: latest
  - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
    inputs:
      rust_version: 1.72.0
  - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
    inputs:
      rust_version: 1.71.0
```

Optimize CI/CD component

1. Split the single template into job specific templates
   a. build
   b. test
2. Add more inputs
3. Caching
4. Avoid global settings



GitLab

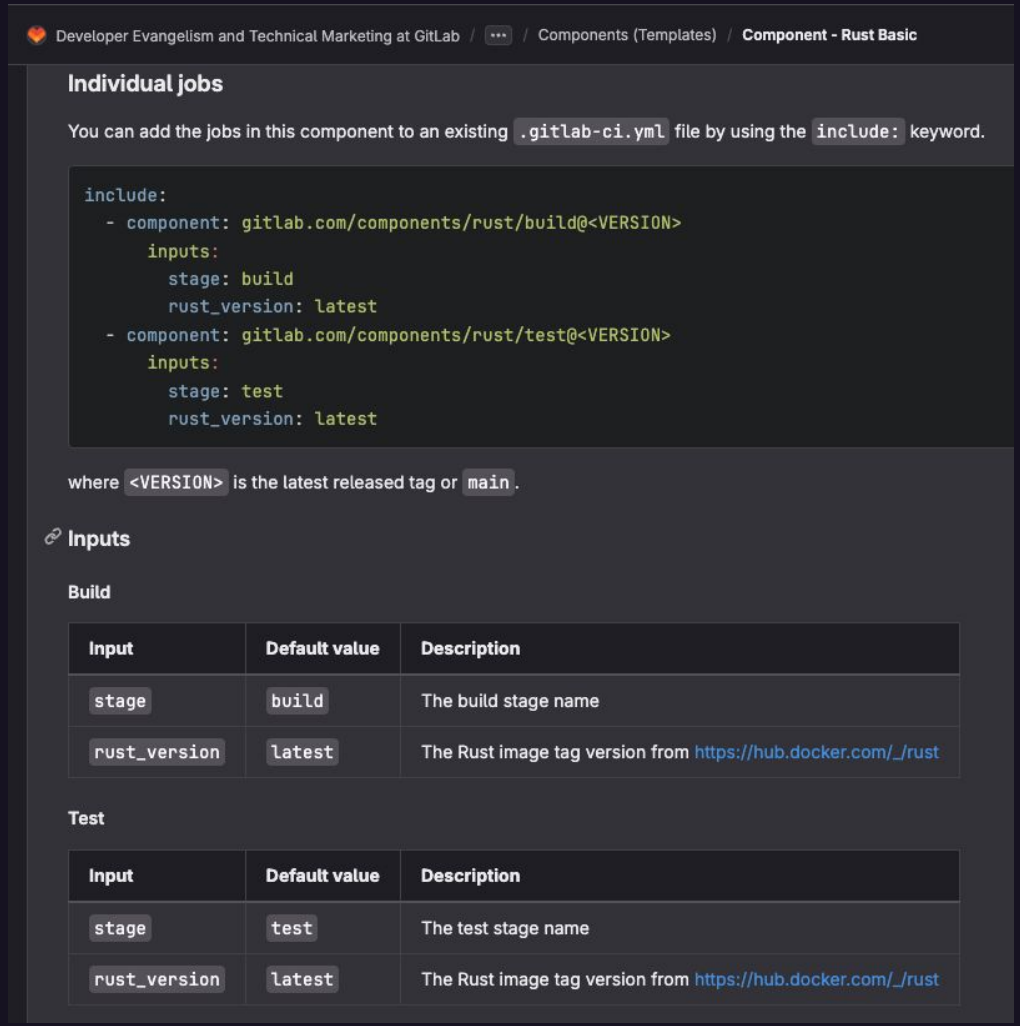
Maintain CI/CD Components

Documentation, tests

Documentation

README.md best practices

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



Testing a CI/CD component

Include the component in CI/CD

Test different input values for different templates

Use pre-defined CI/CD variables

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

.gitlab-ci.yml 1.56 KiB Blame Edit

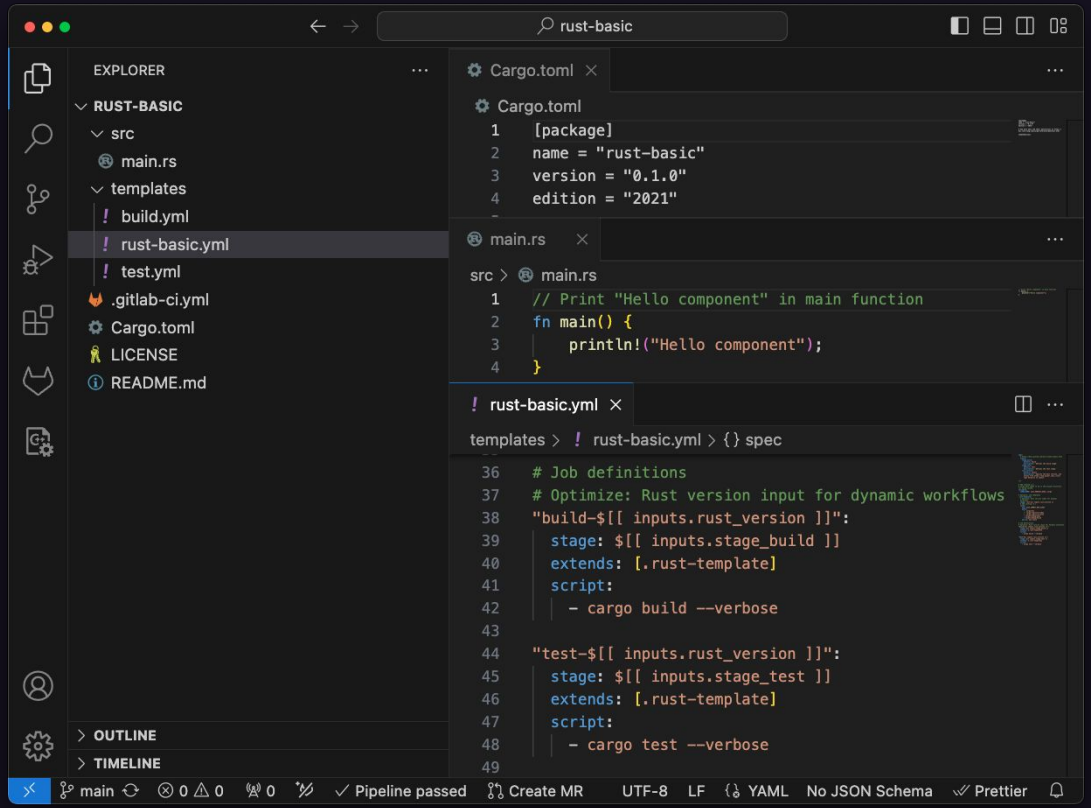
```yaml
include:
  # include the component located in the current project from the current SHA
  - component: gitlab.com/$CI_PROJECT_PATH/rust-basic@$CI_COMMIT_SHA
    inputs:
      stage_build: build
      stage_test: test
      rust_version: latest
  - component: gitlab.com/$CI_PROJECT_PATH/build@$CI_COMMIT_SHA
    inputs:
      rust_version: 1.72.0
      #stage: build
  - component: gitlab.com/$CI_PROJECT_PATH/test@$CI_COMMIT_SHA
    inputs:
      rust_version: 1.72.0
      # stage: test

stages:
  - build
  - test
  - release
```

Testing a CI/CD component

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

Rust (run *cargo init*)

Cargo.toml configuration
src/main.rs source code



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 resource

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

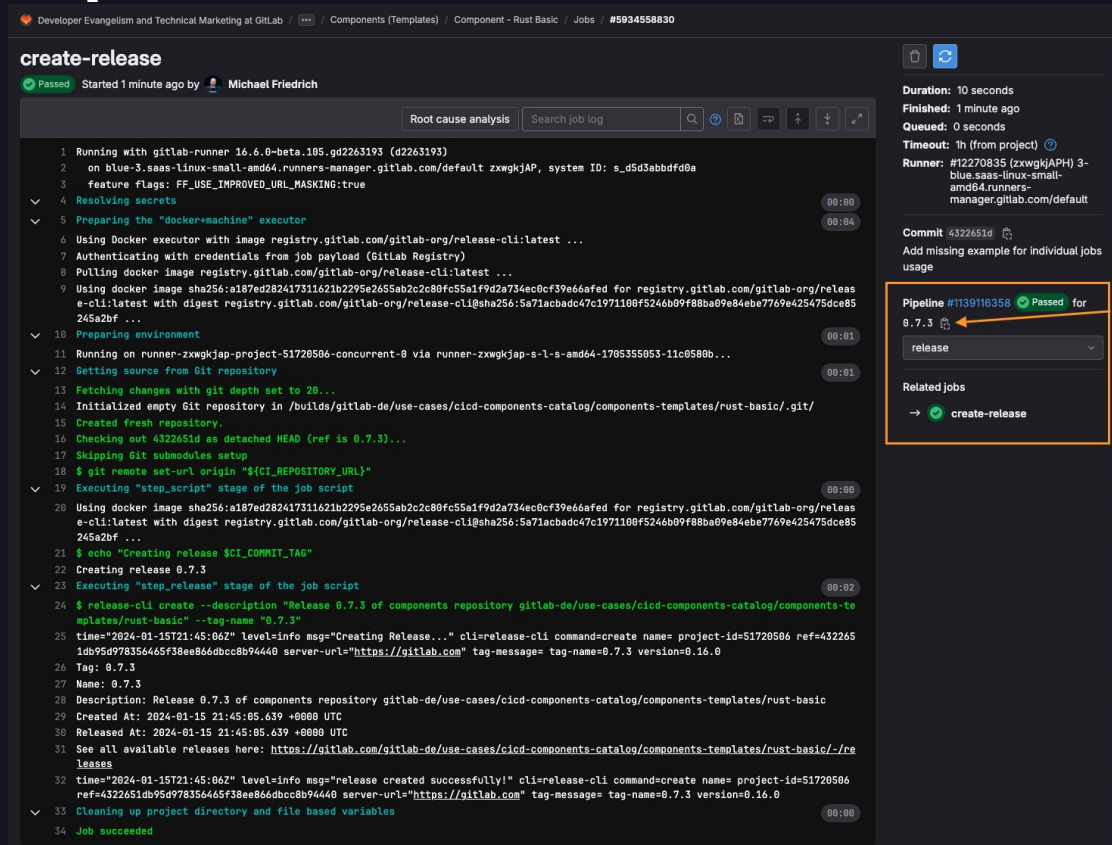
Release a CI/CD component

Automated in CI/CD

Release notes generated using release-cli

Published into the CI/CD Catalog

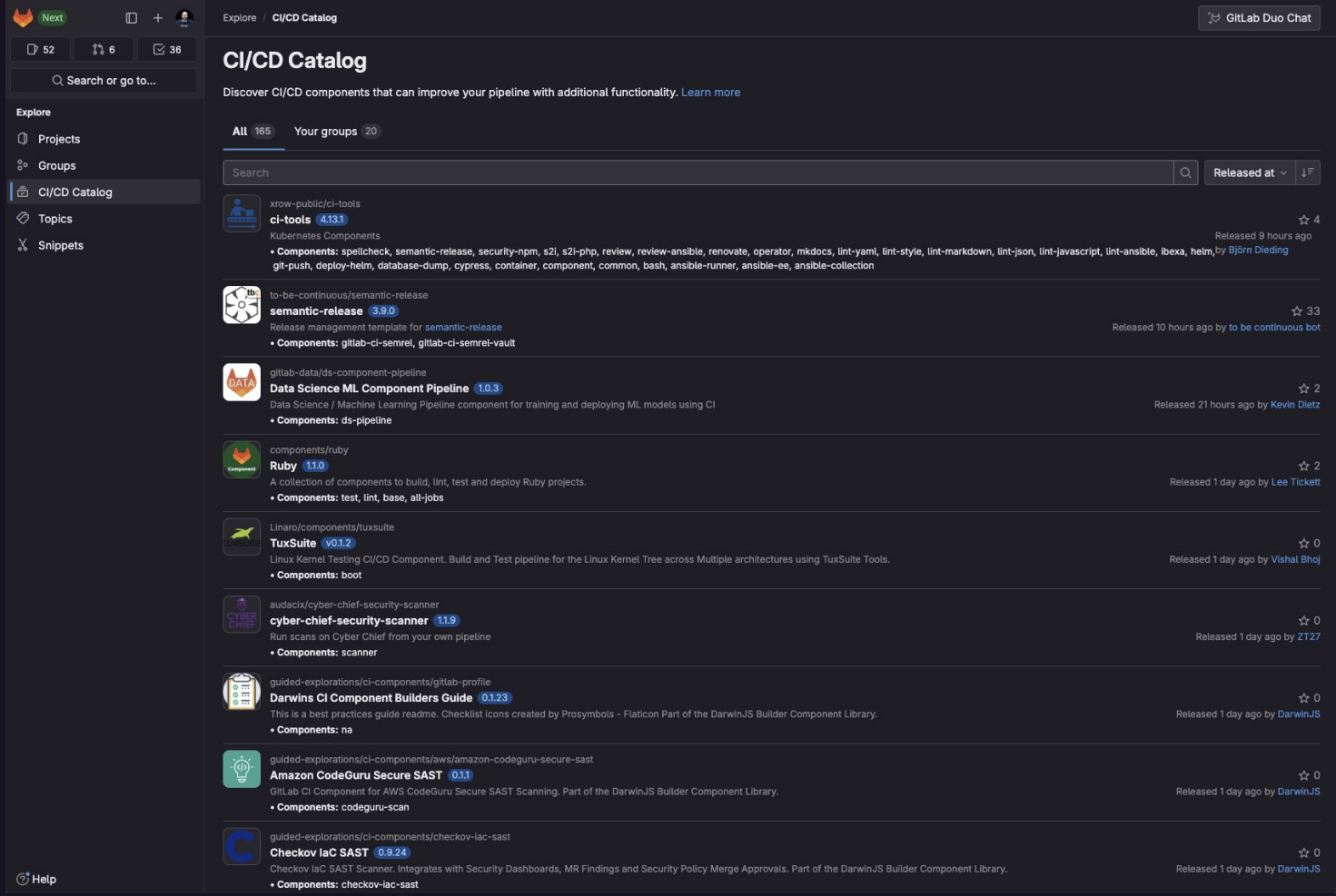
User action: Create and push a Git tag (semantic version)



CI/CD catalog

Search or go to > Explore > CI/CD catalog

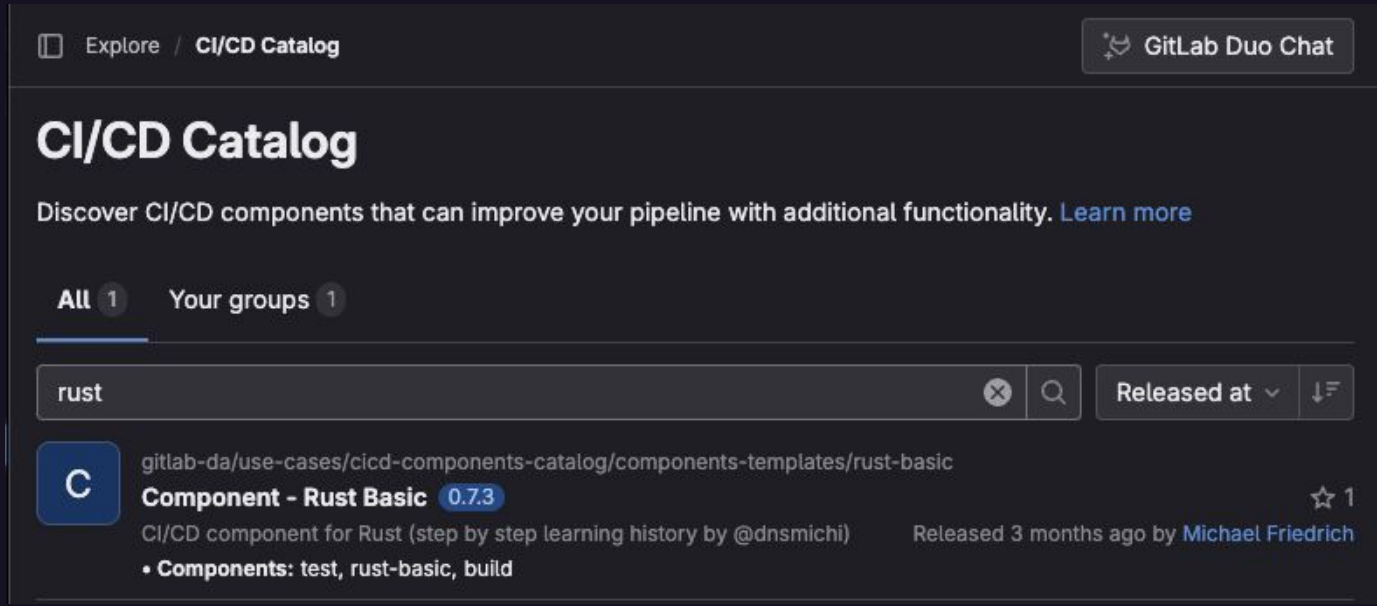
https://gitlab.com/explore/catalog



CI/CD catalog

Search for the component name

https://gitlab.com/explore/catalog



GitLab

Component Types

Photo by Karsten Würth on Unsplash

Component types

First iteration: Templates
https://docs.gitlab.com/ee/ci/components/#directory-structure

Next iteration: Steps
https://docs.gitlab.com/ee/architecture/blueprints/gitlab_steps/
Experiment feedback: https://gitlab.com/gitlab-org/step-runner/-/issues/10
Direction: https://about.gitlab.com/direction/ci/#cross-section-efforts

Direction: Expanding catalog resource types
https://about.gitlab.com/direction/verify/component_catalog/#expanding-catalog-resource-types

Glossary
https://docs.gitlab.com/ee/architecture/blueprints/ci_pipeline_components/#glossary

Photo by Bernard Hermant on Unsplash

CI/CD Steps

Status: Experiment.

https://docs.gitlab.com/ee/ci/steps/
https://gitlab.com/gitlab-org/ci-cd/runner-tools/echo-step/-/blob/master/step.yml?ref_type=heads

```yaml
# (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
```

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.

```yaml
# 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.

```yaml
# 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]
```

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) ]]
```

Practical examples: Dynamic inputs

Arrays: Runner tags as inputs
https://docs.gitlab.com/ee/ci/runners/hosted_runners/linux.html

String: Image as inputs
https://gitlab.com/explore/catalog/to-be-continuous/python

```yaml
include:
  # 1: include the component
  - component: gitlab.com/to-be-continuous/python/gitlab-ci-python@6.11.1
    # 2: set/override component inputs
    inputs:
      image: registry.hub.docker.com/library/python:3.10
      pytest-enabled: true
```


GitLab

Benefits

Photo by Tim Marshall on Unsplash

Building blocks

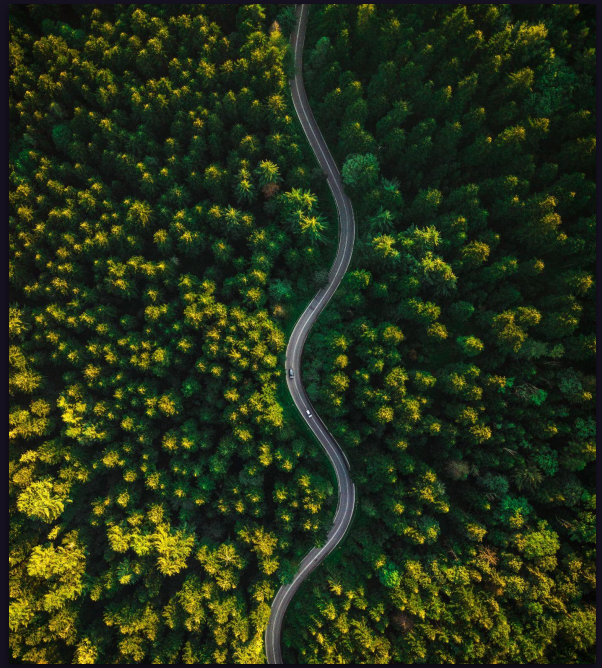
Programming languages - lint, build, test best practices

Efficiency best practices

Dynamic workflows

Version control, dependencies, automated tests

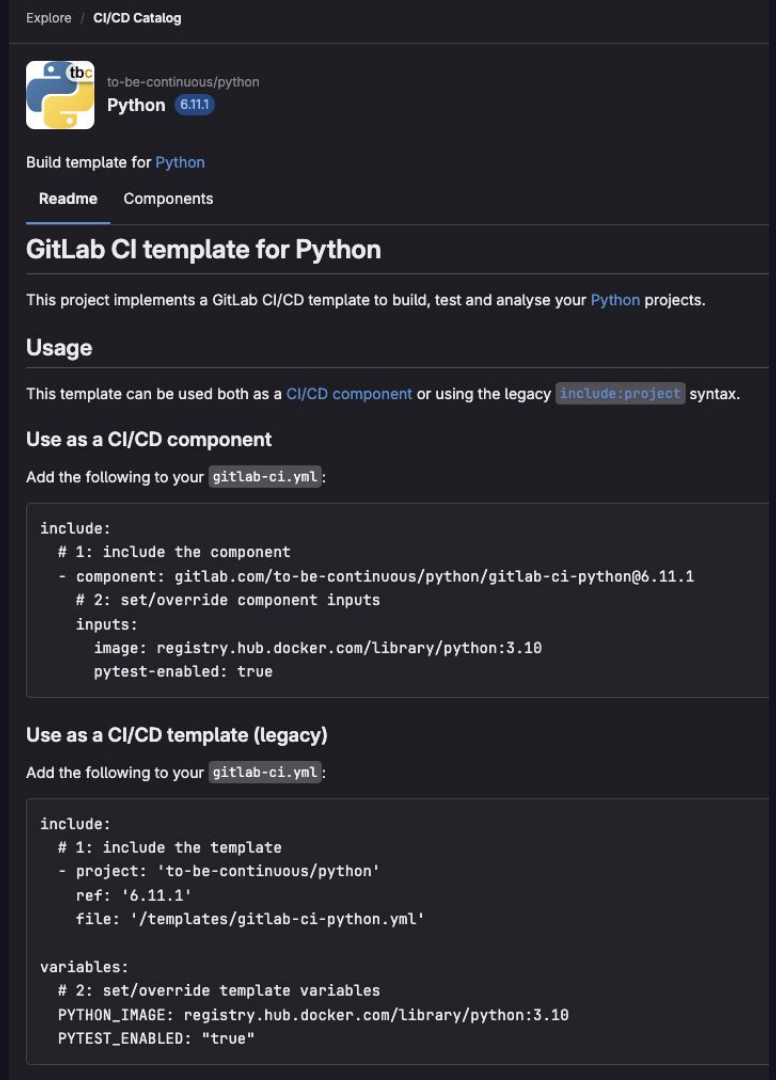
Visibility



Contributed components

Highlight: to-be-continuous - Python

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

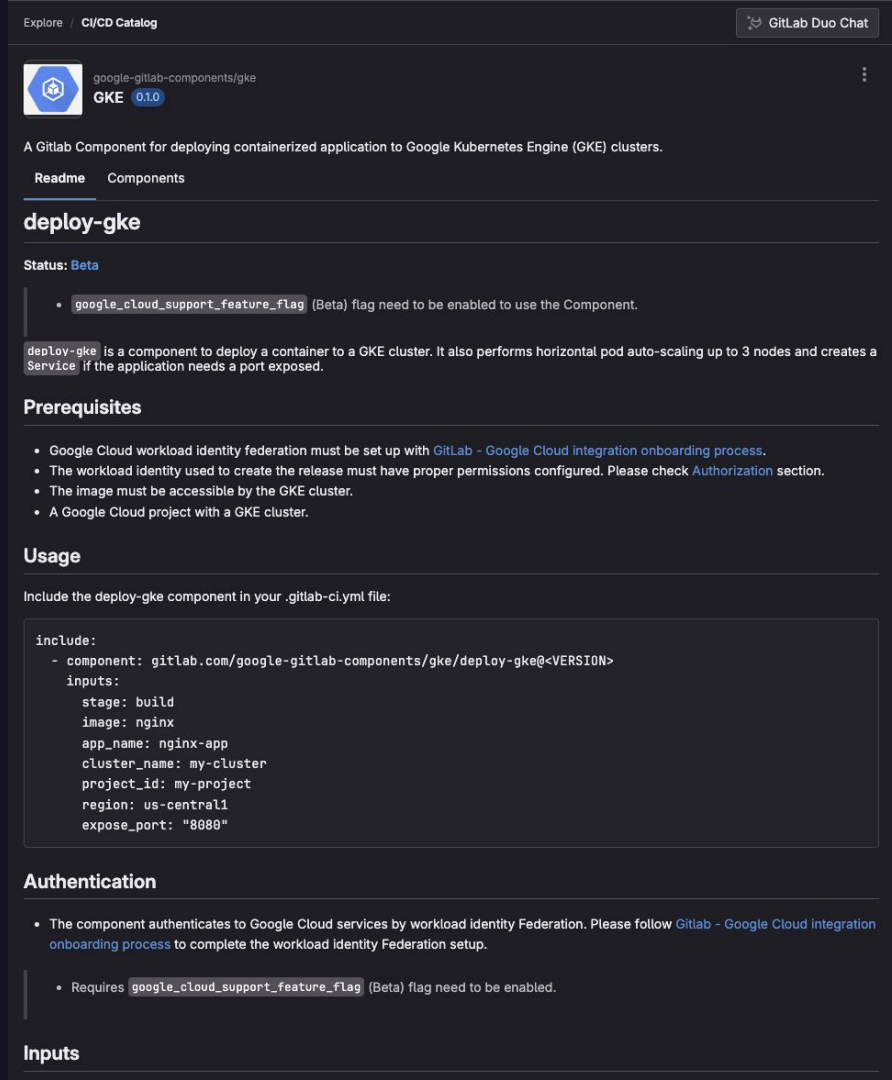


Contributed components

Highlight: Google - GKE

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

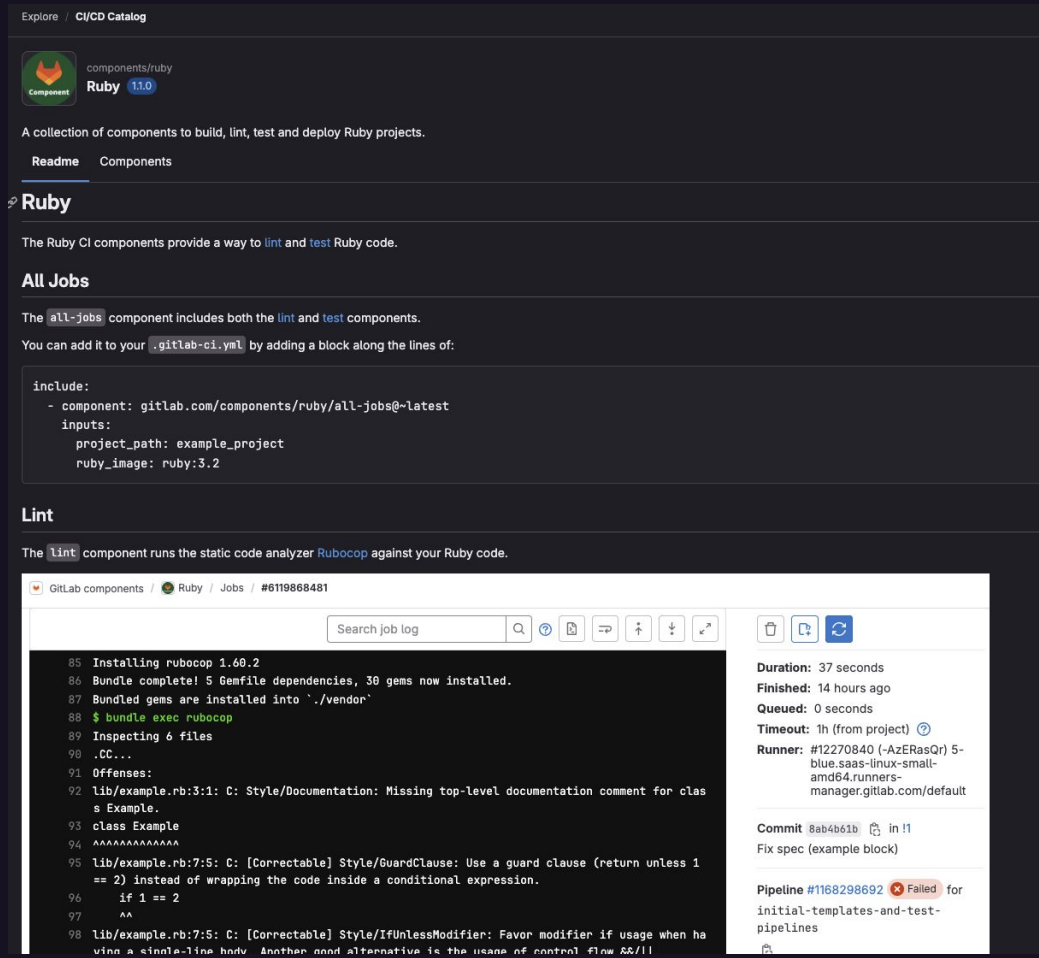
Learn more: https://about.gitlab.com/blog/2024/04/09/gitlab-google-cloud-integrations-now-in-public-beta/#automate-cicd



Maintained components

Highlight: GitLab - Ruby

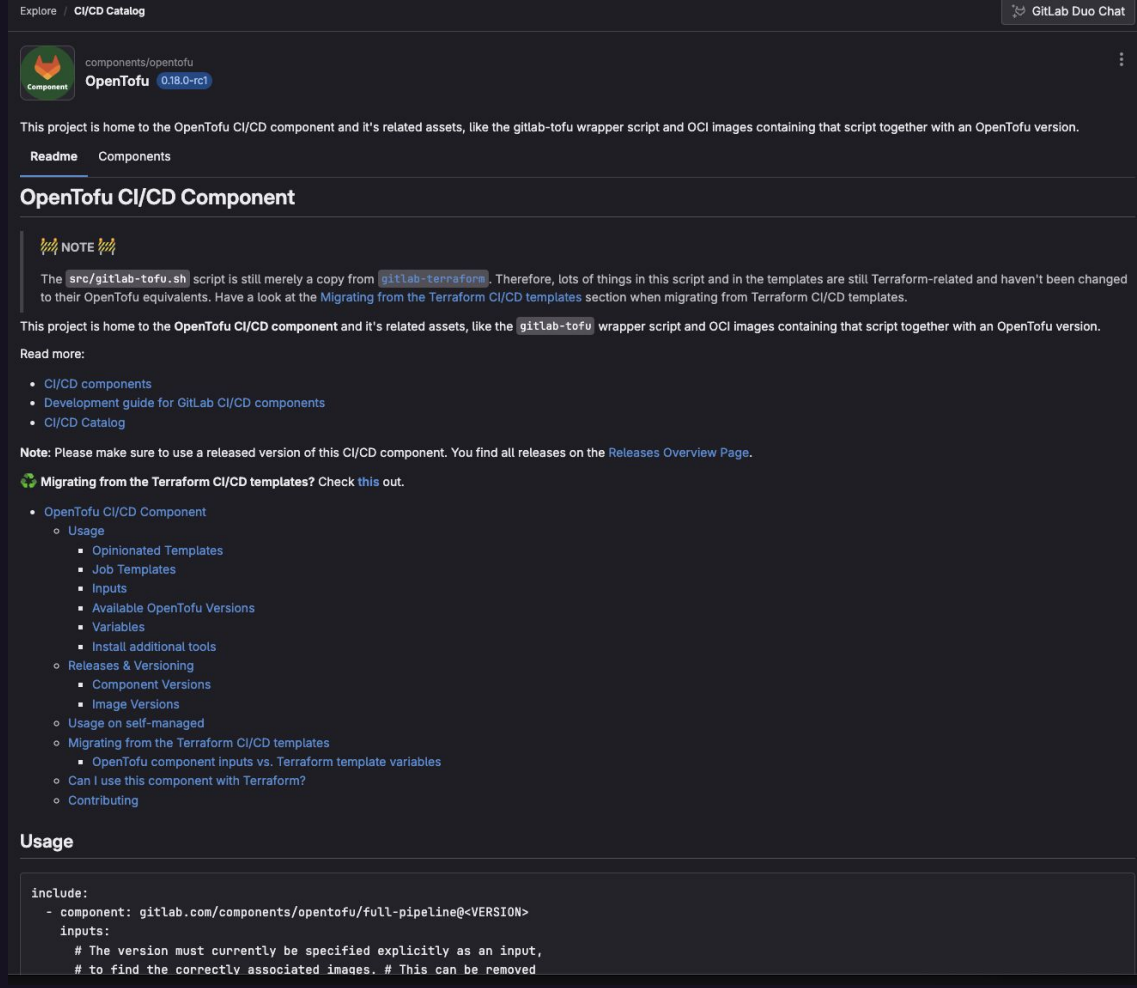
https://gitlab.com/explore/catalog/components/ruby



Maintained components

Highlight: GitLab - OpenTofu

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



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, Edge
8. MLOps, Machine Learning, Data Science

GitLab

Everyone can contribute

Create

Photo by Johannes Andersson on Unsplash

Start today

Create a component repository
https://docs.gitlab.com/ee/ci/components/#create-a-components-repository

Convert CI/CD templates into components
https://docs.gitlab.com/ee/ci/components/#convert-a-cicd-template-to-a-component

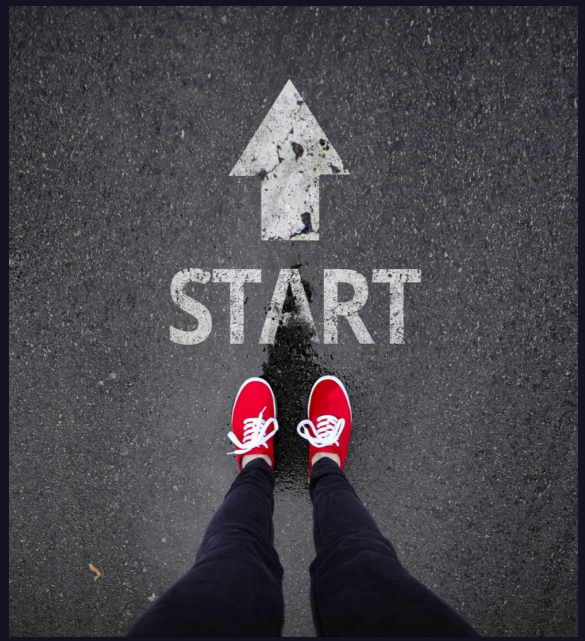
Release a component
https://docs.gitlab.com/ee/ci/components/#release-a-component

Development guide for GitLab CI/CD components
https://docs.gitlab.com/ee/development/cicd/components

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.

```
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.

```
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

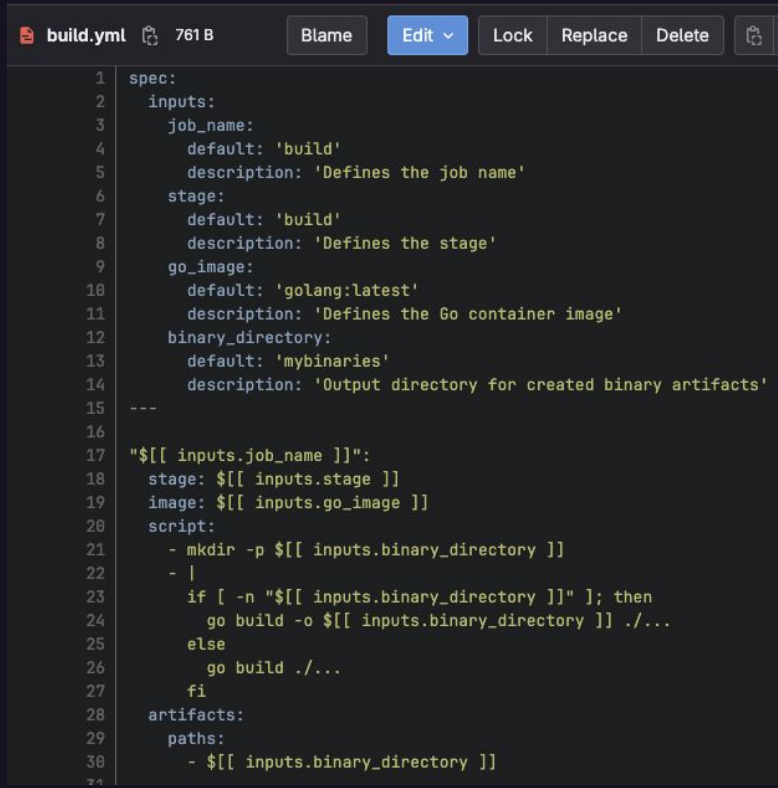
3 Create template directory structure

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*



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

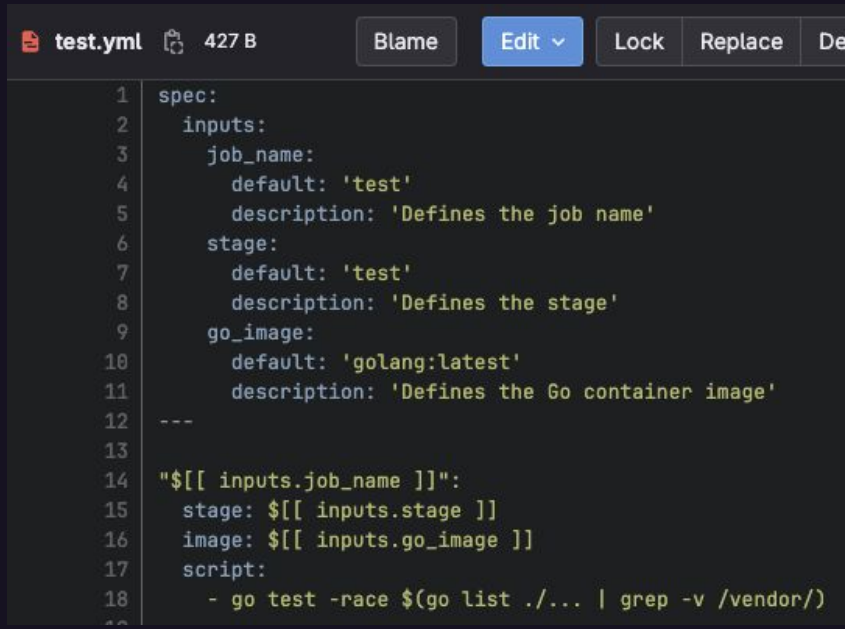
format.yml 471 B Blame Edit Lock Replace Delete

```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

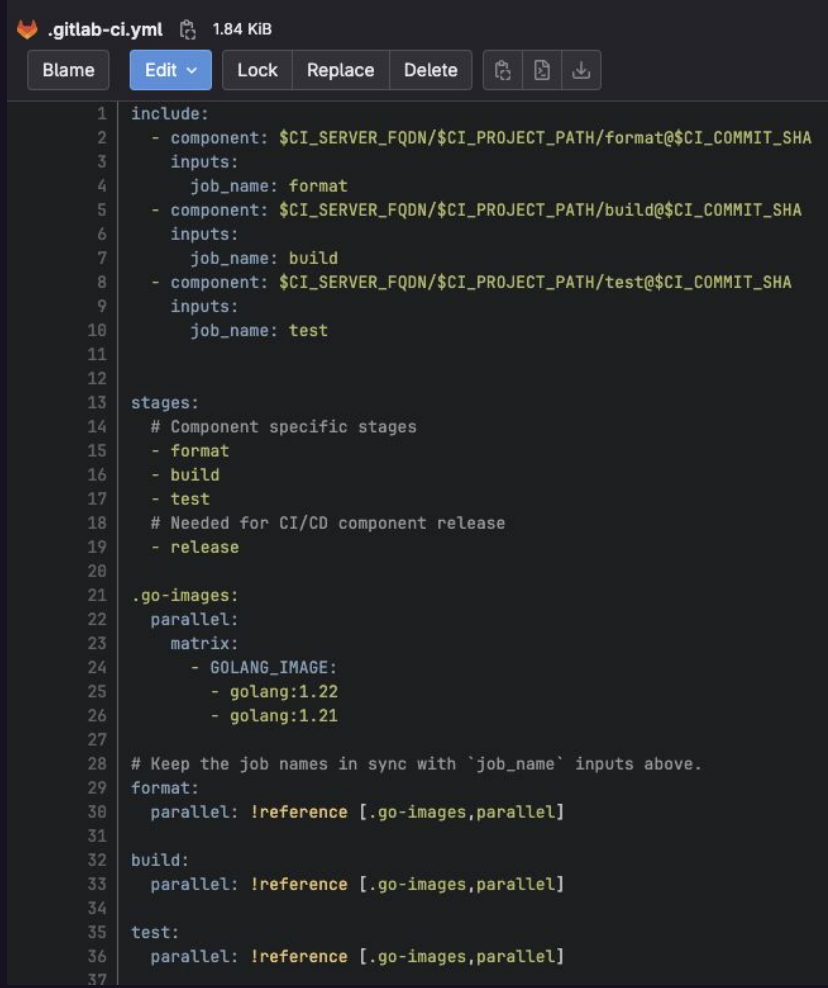
1. Follow the same pattern:
2. Inputs: *job_name*, *stage* and *go_image*
   a. Default values, description
3. Dynamic job name, stage, image



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



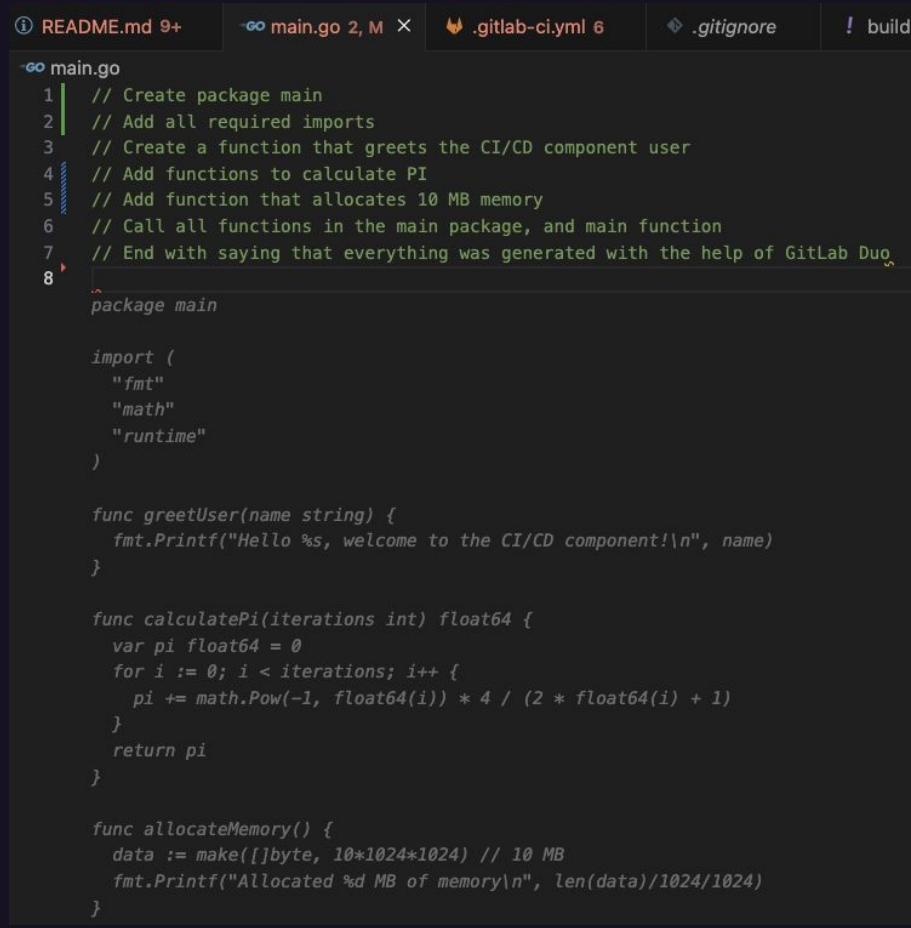
Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

8 Add Go source code

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

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

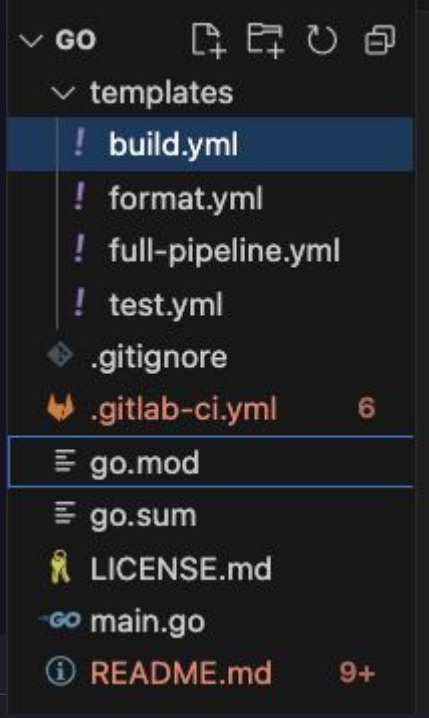
💡 Tip: Use GitLab Duo Code Suggestions and use comments as prompt instructions to generate code.

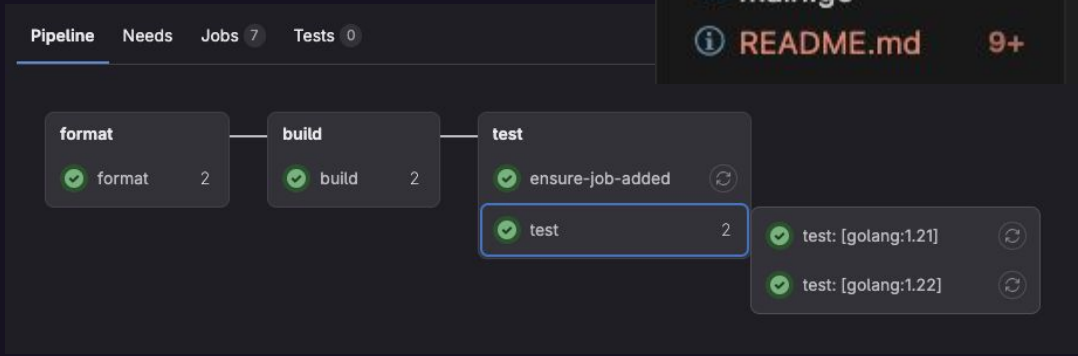


Inspect the Git history in https://gitlab.com/components/go to follow the learning iterations.

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

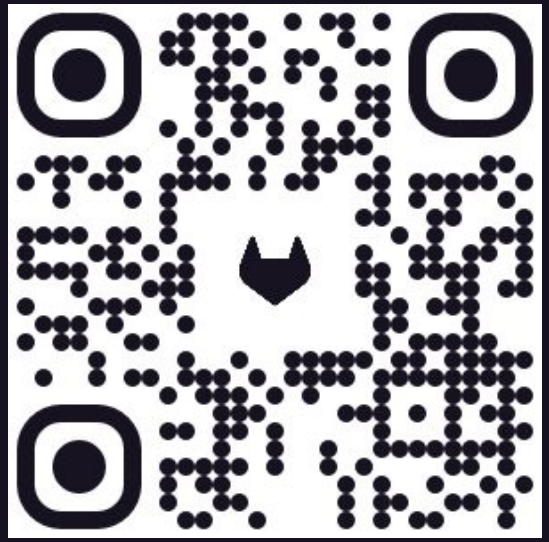
- Reusable, self-contained building blocks for GitLab CI/CD
- Visible and discoverable
- Share “best practice” pipeline jobs
- Formalized, and testable input parameters
- Create and consume
- Maintain, test and release
- Inspire collaboration

Photo by Diego PH on Unsplash

GitLab

Resources

Public slides: https://go.gitlab.com/duA2Fc



GitLab

Thank you