

# Efficient DevSecOps with CI/CD Components

Open Source @ Siemens, 2024-05-14

# Michael Friedrich

Senior Developer Advocate  
@dnsmichi

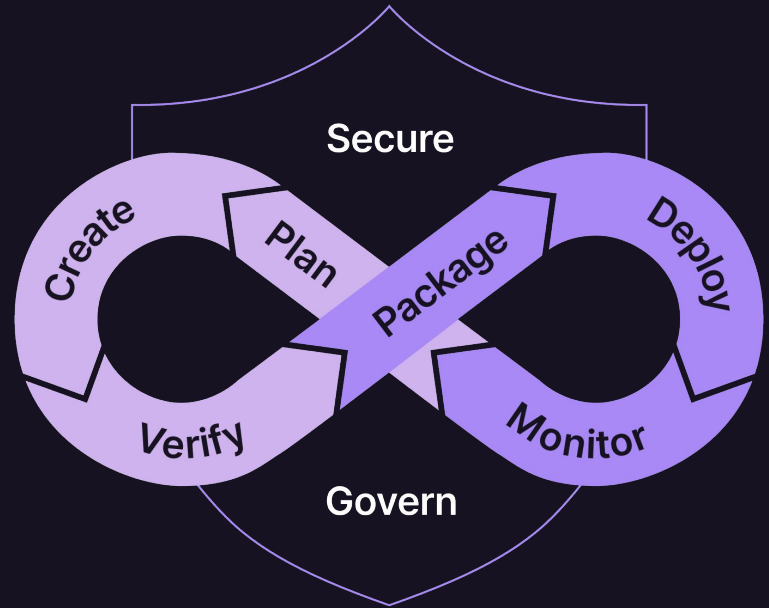


# Efficient DevSecOps?



# 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



# DevSecOps is all about speed



How to discover?

Google / LLM/RAG  
Developer portal  
Ask your peers



How to use?

Documentation  
Wiki  
Read the code

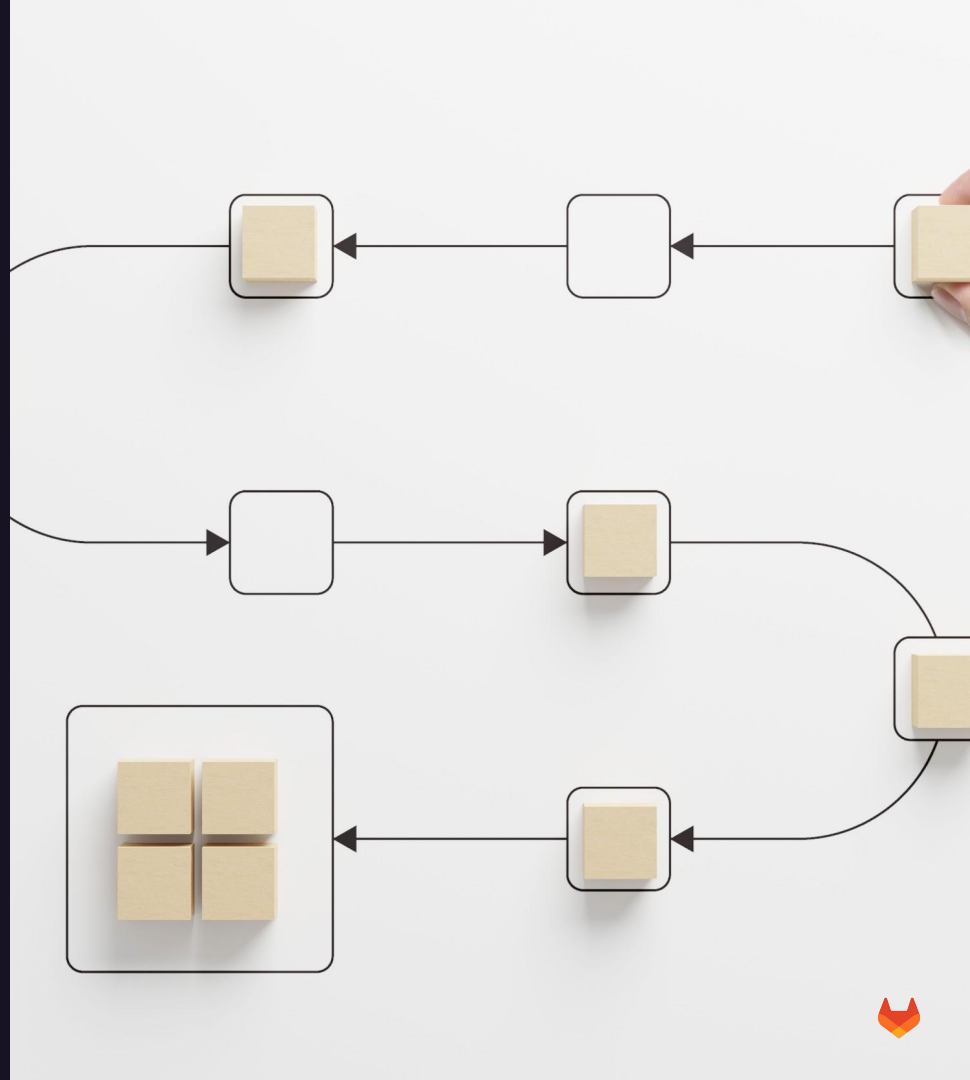


How to share?

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

```
# .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 😞


```
# .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

```
1 rust-docs:
2   stage: docs
3   image: rust:latest
4   script:
5     - 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 🤔

```
# .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

```
1 rust-docs:
2   stage: docs
3   image: rust:latest
4   script:
5     - cargo doc
```

MR:

[https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/migration/inc-lude-stages-rust-gitlab-api/-/merge\\_requests/2](https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/migration/inc-lude-stages-rust-gitlab-api/-/merge_requests/2)

GitLab Copyright



# Traditional Pipeline Composition in GitLab CI/CD

Solution 2: override the job 🙄

```
# .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

rust-docs:	rust-docs:
2	stage: docs
3	image: rust:latest
4	script:
5	- cargo doc

MR:

[https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/migration/inc-lude-stages-rust-gitlab-api/-/merge\\_requests/3](https://gitlab.com/gitlab-da/use-cases/cicd-components-catalog/migration/inc-lude-stages-rust-gitlab-api/-/merge_requests/3)

GitLab Copyright



# CI/CD Components



# Streamline and automate pipeline creation

Easy to assemble pipeline components

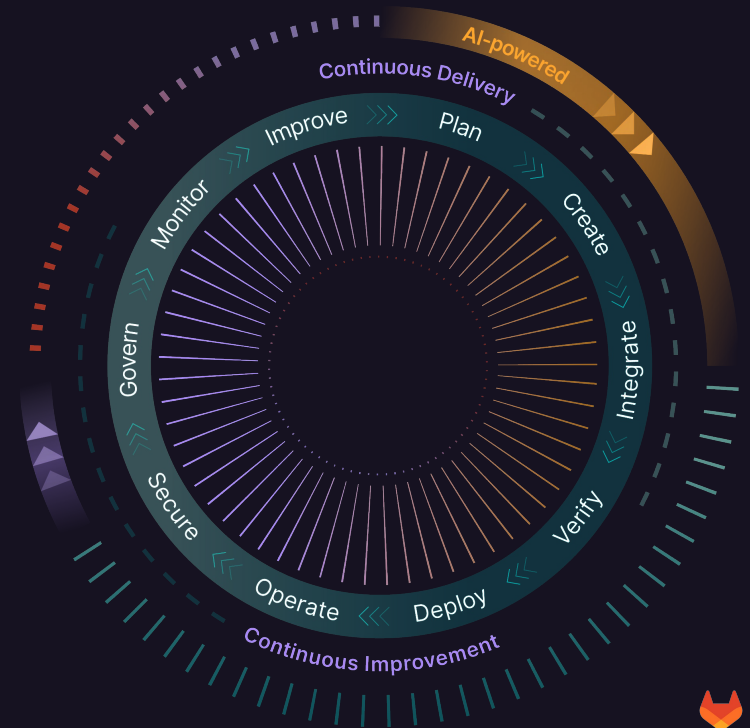
→ consistent and repeatable workflows

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)





# 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





# Inside a component



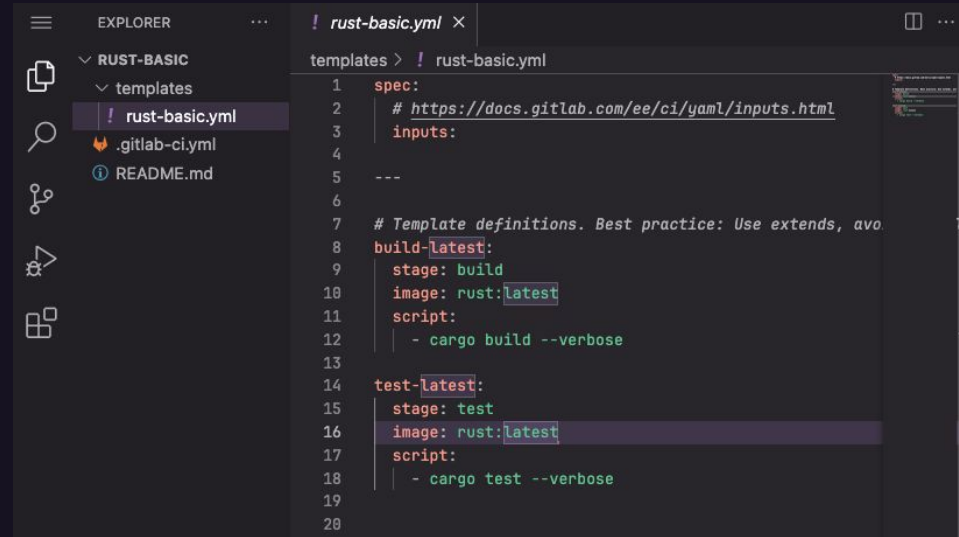
# CI/CD component overview

Directory structure

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

Template definitions

Project in GitLab, release workflows



```
1 spec:
2   # https://docs.gitlab.com/ee/ci/yaml/inputs.html
3   inputs:
4
5   ---
6
7   # Template definitions. Best practice: Use extends, avoid
8   build-latest:
9     stage: build
10    image: rust:latest
11    script:
12      - cargo build --verbose
13
14    test-latest:
15      stage: test
16      image: rust:latest
17      script:
18        - cargo test --verbose
19
20
```





# 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
1 stages:
2   - build
3   - test
4
5 default:
6   image: rust:latest
7   cache:
8     key: ${CI_COMMIT_REF_SLUG}
9     paths:
10    - .cargo/bin
11    - .cargo/registry/index
12    - .cargo/registry/cache
13    - target/debug/deps
14    - target/debug/build
15    policy: pull-push
16
17 # Cargo data needs to be in the project directory for being cached.
18 variables:
19   CARGO_HOME: ${CI_PROJECT_DIR}/.cargo
20
21 build-latest:
22   stage: build
23   script:
24     - cargo build --verbose
25
26 test-latest:
27   stage: build
28   script:
29     - cargo test --verbose
```



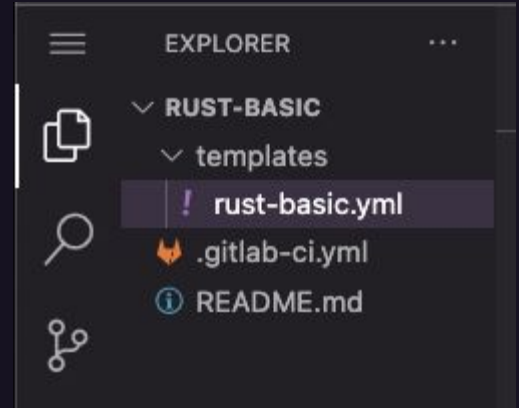
# Basic Rust component

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

Create a GitLab project

Directory tree

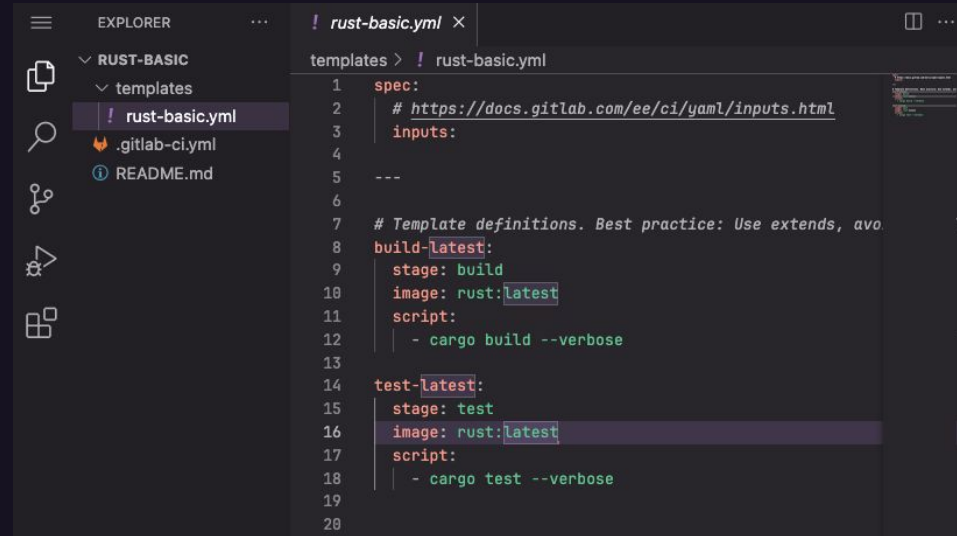
<i>templates/</i>	← component type
<i>rust-basic.yml</i>	← component template name
<i>README.md</i>	← How to use
<i>.gitlab-ci.yml</i>	← testing the component



# 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



```
EXPLORER  ...  ! rust-basic.yml x
templates > ! rust-basic.yml
1 spec:
2   # https://docs.gitlab.com/ee/ci/yaml/inputs.html
3   inputs:
4
5   ---
6
7   # Template definitions. Best practice: Use extends, avo
8 build-latest:
9   stage: build
10  image: rust:latest
11  script:
12  | - cargo build --verbose
13
14 test-latest:
15  stage: test
16  image: rust:latest
17  script:
18  | - cargo test --verbose
19
20
```



# Consume Rust CI/CD component

```
.gitlab-ci.yml 563 B
```

Blame Edit Lock Replace Delete

```
1 stages:
2   - build
3   - test
4
5 include:
6   - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
7   inputs:
```

Component project path

Component template in templates/ directory

Version: Git tag

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
1 1 spec:
2 2 # https://docs.gitlab.com/ee/ci/yaml/inputs.html
3 3 inputs:
4 + stage_build:
5 + default: build
6 + description: 'Defines the build stage'
7 + stage_test:
8 + default: test
9 + description: 'Defines the test stage'
4 10
5 11 ---
6 12
↓
@@ -27,13 +33,13 @@ variables:
27 33
28 34 # Template definitions. Best practice: Use extends,
keywords
29 35 build-latest:
30 - stage: build
36 + stage: $$[ inputs.stage_build ]
31 37 extends: [.rust-template]
32 38 script:
33 39 - cargo build --verbose
34 40
35 41 test-latest:
36 - stage: test
42 + stage: $$[ inputs.stage_test ]
37 43 extends: [.rust-template]
38 44 script:
39 45 - cargo test --verbose
↓
```





# Dynamic inputs - validation

Specify the component inputs

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

## Use stage inputs for components

Passed Michael Friedrich created pipeline for commit 8b409547 finished just now

For main

latest 2 Jobs 0 34 seconds, queued for 1 seconds

Pipeline Needs Jobs 2 Tests 0

### build

build-latest

test-latest

.gitlab-ci.yml

```
1  stages:
2    - build
3    - test
4
5  include:
6    component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@main
7    inputs:
8      # set the same stage to show component behavior
9      stage_build: build
10     stage_test: build
11
```



# 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](https://hub.docker.com/_/rust/tags)

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

↑ @@ -7,6 +7,9 @@ spec:
7 7     stage_test:
8 8     default: test
9 9     description: 'Defines the test stage'
10 + rust_version:
11 +     default: latest
12 +     description: 'Specify the Rust version, use values from
    https://hub.docker.com/_/rust/tags Defaults to latest'
10 13
11 14     ---
12 15

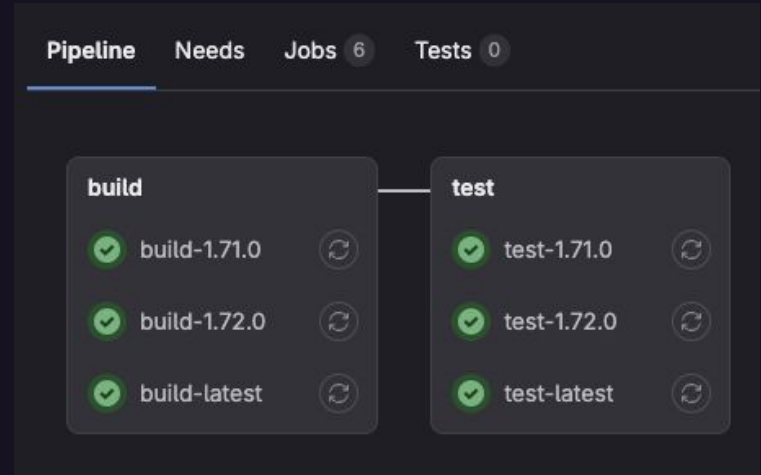
↕ @@ -17,7 +20,8 @@ variables:
17 20
18 21     # Optimize: Job templates
19 22     .rust-template:
20 -     image: rust:latest
21 +     # Optimize: Rust version input for dynamic workflows
22 +     image: rust:${[ inputs.rust_version ]}
23 25     # Add caching (2.)
24 26     cache:
25 27     key: ${CI_COMMIT_REF_SLUG}
26 28

↕ @@ -30,13 +34,14 @@ variables:
30 34     policy: pull-push
31 35
32 36     # Job definitions
33 -     build-latest:
34 +     # Optimize: Rust version input for dynamic workflows
35 +     "build-${[ inputs.rust_version ]}":
36     stage: ${[ inputs.stage_build ]}
37     extends: [.rust-template]
38     script:
39     - cargo build --verbose
40 41
41 -     test-latest:
42 +     "test-${[ inputs.rust_version ]}":
43     stage: ${[ inputs.stage_test ]}
44     extends: [.rust-template]
45     script:
```

# Dynamic job names

⚡ Job names based on input

🌟 Reusable components



```
.gitlab-ci.yml 563 B
```

Blame Edit Lock Replace Delete

```
1 stages:
2   - build
3   - test
4
5 include:
6   - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
7     inputs:
8       stage_build: build
9       stage_test: test
10      rust_version: latest
11  - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
12    inputs:
13      rust_version: 1.72.0
14  - component: gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/rust-basic@0.5
15    inputs:
16      rust_version: 1.71.0
17
```

Component project path

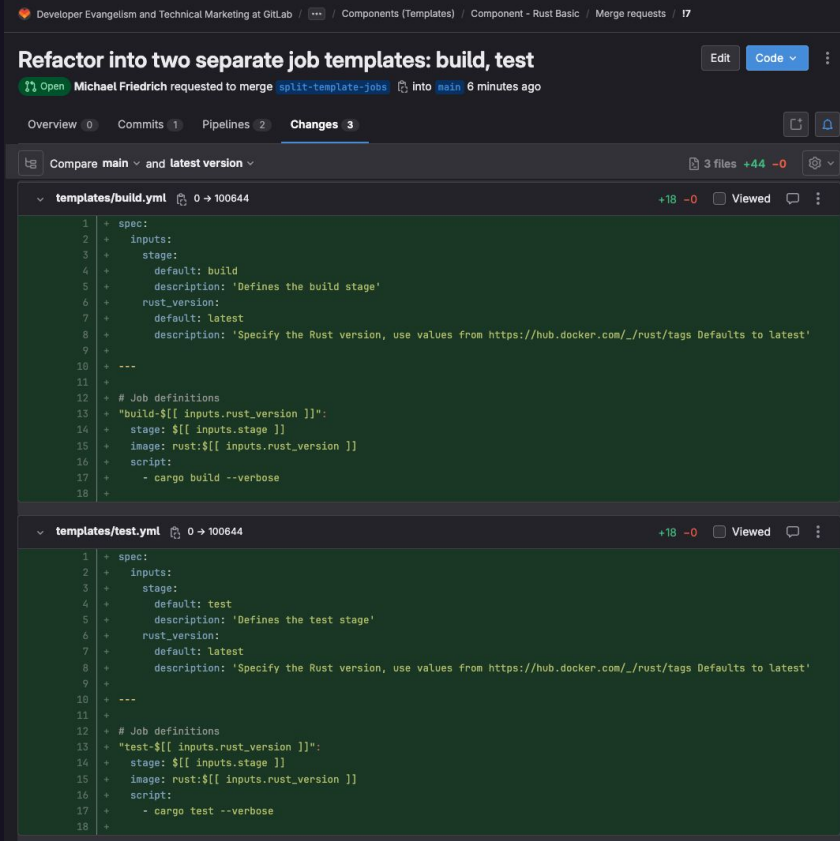
Component template in templates/ directory

Version: Git tag



# 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



The screenshot shows a GitLab merge request titled "Refactor into two separate job templates: build, test". The merge request is for the "Component - Rust Basic" and is comparing the "main" branch with the "split-template-jobs" branch. The merge request was requested by Michael Friedrich 6 minutes ago. The interface shows two files being compared: "templates/build.yml" and "templates/test.yml".

```
1 + spec:
2 +   inputs:
3 +     stage:
4 +       default: build
5 +       description: 'Defines the build stage'
6 +     rust_version:
7 +       default: latest
8 +       description: 'Specify the Rust version, use values from https://hub.docker.com/_/rust/tags Defaults to latest'
9 +
10 + ---
11 +
12 + # Job definitions
13 + "build-${[ inputs.rust_version ]}":
14 +   stage: $[ inputs.stage ]
15 +   image: rust:${[ inputs.rust_version ]}
16 +   script:
17 +     - cargo build --verbose
18 +
```

```
1 + spec:
2 +   inputs:
3 +     stage:
4 +       default: test
5 +       description: 'Defines the test stage'
6 +     rust_version:
7 +       default: latest
8 +       description: 'Specify the Rust version, use values from https://hub.docker.com/_/rust/tags Defaults to latest'
9 +
10 + ---
11 +
12 + # Job definitions
13 + "test-${[ inputs.rust_version ]}":
14 +   stage: $[ inputs.stage ]
15 +   image: rust:${[ inputs.rust_version ]}
16 +   script:
17 +     - cargo test --verbose
18 +
```

# Maintain CI/CD Components

Documentation, tests



# Documentation

## README.md best practices

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

### Individual jobs

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

```
include:
- component: gitlab.com/components/rust/build@<VERSION>
  inputs:
    stage: build
    rust_version: latest
- component: gitlab.com/components/rust/test@<VERSION>
  inputs:
    stage: test
    rust_version: latest
```

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

### Inputs

#### Build

Input	Default value	Description
<code>stage</code>	<code>build</code>	The build stage name
<code>rust_version</code>	<code>latest</code>	The Rust image tag version from <a href="https://hub.docker.com/_/rust">https://hub.docker.com/_/rust</a>

#### Test

Input	Default value	Description
<code>stage</code>	<code>test</code>	The test stage name
<code>rust_version</code>	<code>latest</code>	The Rust image tag version from <a href="https://hub.docker.com/_/rust">https://hub.docker.com/_/rust</a>

# 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]
1 include:
2   # include the component located in the current project from the current SHA
3   - component: gitlab.com/$CI_PROJECT_PATH/rust-basic@$CI_COMMIT_SHA
4     inputs:
5       stage_build: build
6       stage_test: test
7       rust_version: latest
8   - component: gitlab.com/$CI_PROJECT_PATH/build@$CI_COMMIT_SHA
9     inputs:
10      rust_version: 1.72.0
11      #stage: build
12   - component: gitlab.com/$CI_PROJECT_PATH/test@$CI_COMMIT_SHA
13     inputs:
14      rust_version: 1.72.0
15      # stage: test
16
17 stages:
18   - build
19   - test
20   - release
21
```

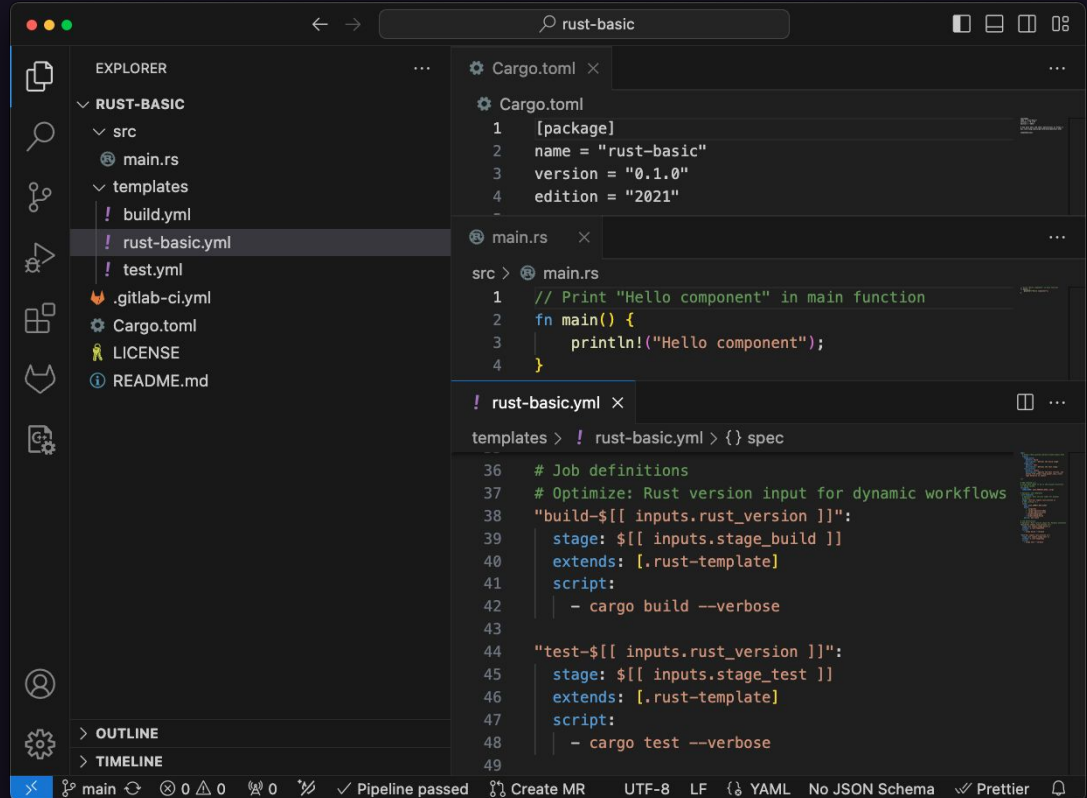


# Testing a CI/CD component

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

Rust (run *cargo init*)

*Cargo.toml* configuration  
*src/main.rs* source code



The screenshot shows a code editor with the following content:

```
EXPLORER
├── RUST-BASIC
│   ├── src
│   │   └── main.rs
│   ├── templates
│   │   ├── build.yml
│   │   └── rust-basic.yml
│   ├── Cargo.toml
│   ├── LICENSE
│   └── README.md
├── .gitlab-ci.yml
├── Cargo.toml
├── LICENSE
└── README.md
```

```
Cargo.toml
1 [package]
2 name = "rust-basic"
3 version = "0.1.0"
4 edition = "2021"
```

```
src > @ main.rs
1 // Print "Hello component" in main function
2 fn main() {
3     println!("Hello component");
4 }
```

```
templates > ! rust-basic.yml > {} spec
36 # Job definitions
37 # Optimize: Rust version input for dynamic workflows
38 "build-${[[ inputs.rust_version ]]}":
39     stage: ${[[ inputs.stage_build ]]
40     extends: [.rust-template]
41     script:
42     - cargo build --verbose
43
44 "test-${[[ inputs.rust_version ]]}":
45     stage: ${[[ inputs.stage_test ]]
46     extends: [.rust-template]
47     script:
48     - cargo test --verbose
49
```

GitLab Copyright

main 0 0 0 Pipeline passed Create MR UTF-8 LF YAML No JSON Schema Prettier





# Visibility

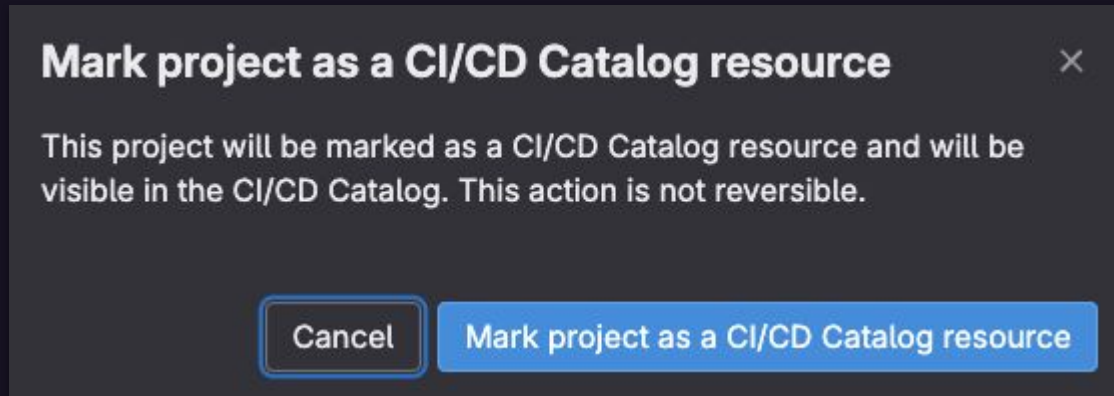
CI/CD Catalog



# Add Rust component to CI/CD catalog

Project > Settings >

1. Add description
2. Visibility, project features, permissions > CI/CD catalog resource



# 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)

The screenshot shows a GitLab CI/CD pipeline named 'create-release' that has successfully completed. The pipeline was started 1 minute ago by Michael Friedrich. The main console output shows the following steps:

- Running with gitlab-runner 16.6.0-beta.105.gd2263193
- on blue-3.saas-linux-small-amd64.runners-manager.gitlab.com/default zwxgkJP, system ID: s\_ds53abddfd9a
- feature flags: FF\_USE\_IMPROVED\_URL\_MASKING:true
- Resolving secrets
- Preparing the "docker+machine" executor
- Using Docker executor with image registry.gitlab.com/gitlab-org/release-cli:latest ...
- Authenticating with credentials from job payload (Gitlab Registry)
- Pulling docker image registry.gitlab.com/gitlab-org/release-cli:latest ...
- Using docker image sha256:a187ed282417311621b2295e2655ab2c2c80fc55a1f9d2a734ec0cf39e66afed for registry.gitlab.com/gitlab-org/release-cli:latest with digest registry.gitlab.com/gitlab-org/release-cli@sha256:5a71acbadc47c1971180f5246b09f88ba09e84e7769e425475dc85245a2bf ...
- Preparing environment
- Running on runner-zwxgkJP-project-51720506-concurrent-0 via runner-zwxgkJP-s-l-s-amd64-1705355053-11c0580b...
- Getting source from Git repository
- Fetching changes with git depth set to 20...
- Initialized empty Git repository in /builds/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/.git/
- Created fresh repository.
- Checking out 4322651d as detached HEAD (ref is 0.7.3)...
- Skipping Git submodules setup
- Executing "step\_script" stage of the job script
- Using docker image sha256:a187ed282417311621b2295e2655ab2c2c80fc55a1f9d2a734ec0cf39e66afed for registry.gitlab.com/gitlab-org/release-cli:latest with digest registry.gitlab.com/gitlab-org/release-cli@sha256:5a71acbadc47c1971180f5246b09f88ba09e84e7769e425475dc85245a2bf ...
- echo "Creating release \$CI\_COMMIT\_TAG"
- Creating release 0.7.3
- Executing "step\_release" stage of the job script
- \$ release-cli create --description "Release 0.7.3 of components repository gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic" --tag-name "0.7.3"
- times:"2024-01-15T21:45:06Z" level:info msg:"Creating Release..." cli=release-cli command:create name= project-id=51720506 ref=4322651db95d978356465f38ee866dbcc8b94440 server-url="https://gitlab.com" tag=message= tag-name=0.7.3 version=0.16.0
- Tag: 0.7.3
- Name: 0.7.3
- Description: Release 0.7.3 of components repository gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic
- Created At: 2024-01-15 21:45:05.639 +0800 UTC
- Released At: 2024-01-15 21:45:05.639 +0800 UTC
- See all available releases here: https://gitlab.com/gitlab-de/use-cases/cicd-components-catalog/components-templates/rust-basic/-/releases
- times:"2024-01-15T21:45:06Z" level:info msg:"release created successfully!" cli=release-cli command:create name= project-id=51720506 ref=4322651db95d978356465f38ee866dbcc8b94440 server-url="https://gitlab.com" tag-messages= tag-name=0.7.3 version=0.16.0
- Cleaning up project directory and file based variables
- Job succeeded

The right sidebar shows pipeline details: Duration: 10 seconds, Finished: 1 minute ago, Queued: 0 seconds, Timeout: 1h (from project), Runner: #12270835 (zwxgkJP) 3-blue.saas-linux-small-amd64.runners-manager.gitlab.com/default, Commit: 4322651d, Add missing example for individual jobs usage, Pipeline #1139116358 Passed for 0.7.3, and Related jobs: create-release.

# CI/CD catalog

Search or go to  
> Explore >  
CI/CD catalog

<https://gitlab.com/explore/catalog>

Explore / CI/CD Catalog

52 6 36

Search or go to...

Explore

- Projects
- Groups
- CI/CD Catalog**
- Topics
- Snippets

CI/CD Catalog

Discover CI/CD components that can improve your pipeline with additional functionality. [Learn more](#)

All 165 Your groups 20

Search

Released at

- xrow-public/ci-tools  
**ci-tools** 4.13.1  
Kubernetes Components  
• **Components:** spellcheck, semantic-release, security-npm, s2i, s2i-php, review, review-ansible, renovate, operator, mkdocs, lint-yaml, lint-style, lint-markdown, lint-json, lint-javascript, lint-ansible, ibexa, helm, by Björn Dieding  
git-push, deploy-helm, database-dump, cypress, container, component, common, bash, ansible-runner, ansible-ee, ansible-collection
- to-be-continuous/semantic-release  
**semantic-release** 3.9.0  
Release management template for semantic-release  
• **Components:** gitlab-ci-semrel, gitlab-ci-semrel-vault
- gitlab-data/ds-component-pipeline  
**Data Science ML Component Pipeline** 1.0.3  
Data Science / Machine Learning Pipeline component for training and deploying ML models using CI  
• **Components:** ds-pipeline
- components/ruby  
**Ruby** 1.1.0  
A collection of components to build, lint, test and deploy Ruby projects.  
• **Components:** test, lint, base, all-jobs
- Linaro/components/tuxsuite  
**TuxSuite** v0.1.2  
Linux Kernel Testing CI/CD Component. Build and Test pipeline for the Linux Kernel Tree across Multiple architectures using TuxSuite Tools.  
• **Components:** boot
- audacix/cyber-chief-security-scanner  
**cyber-chief-security-scanner** 1.1.9  
Run scans on Cyber Chief from your own pipeline  
• **Components:** scanner
- guided-explorations/ci-components/gitlab-profile  
**Darwins CI Component Builders Guide** 0.1.23  
This is a best practices guide readme. Checklist icons created by Prosymbols - FlatIcon Part of the DarwinJS Builder Component Library.  
• **Components:** na
- guided-explorations/ci-components/aws/amazon-codeguru-secure-sast  
**Amazon CodeGuru Secure SAST** 0.1.1  
GitLab CI Component for AWS CodeGuru Secure SAST Scanning. Part of the DarwinJS Builder Component Library.  
• **Components:** codeguru-scan
- guided-explorations/ci-components/checkov-iac-sast  
**Checkov IaC SAST** 0.9.24  
Checkov IaC SAST Scanner. Integrates with Security Dashboards, MR Findings and Security Policy Merge Approvals. Part of the DarwinJS Builder Component Library.  
• **Components:** checkov-iac-sast

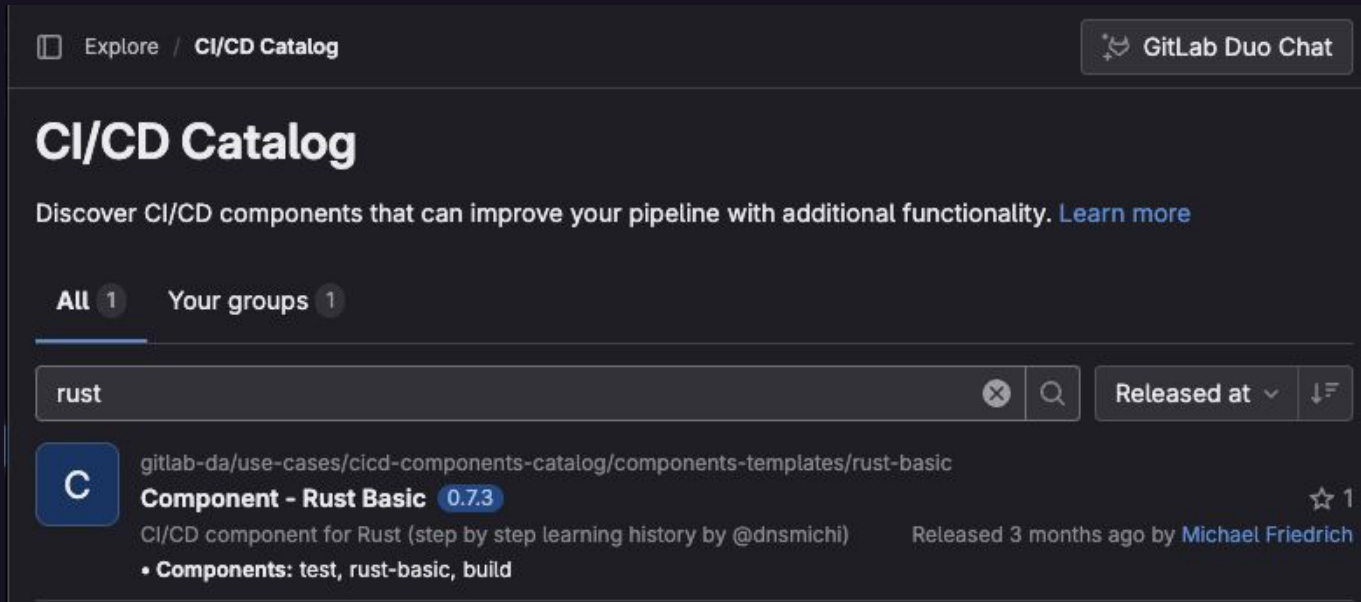
Help



# CI/CD catalog

Search for the  
component  
name

<https://gitlab.com/explore/catalog>



Explore / CI/CD Catalog GitLab Duo Chat

## CI/CD Catalog

Discover CI/CD components that can improve your pipeline with additional functionality. [Learn more](#)

All 1 Your groups 1

rust Released at

**C** gitlab-da/use-cases/cicd-components-catalog/components-templates/rust-basic  
**Component - Rust Basic** 0.7.3 ☆ 1  
CI/CD component for Rust (step by step learning history by @dnsmichi) Released 3 months ago by [Michael Friedrich](#)  
• **Components:** test, rust-basic, build



# Component Types



# 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/](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](https://about.gitlab.com/direction/verify/component_catalog/#expanding-catalog-resource-types)

## Glossary

[https://docs.gitlab.com/ee/architecture/blueprints/ci\\_pipeline\\_components/#glossary](https://docs.gitlab.com/ee/architecture/blueprints/ci_pipeline_components/#glossary)



# 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](https://gitlab.com/gitlab-org/ci-cd/runner-tools/echo-step/-/blob/master/step.yml?ref_type=heads)

```
# (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.

```
# 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 interchangeably.

```
# 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]
```



# 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>

```
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 ]]
```

```
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](https://docs.gitlab.com/ee/ci/runners/hosted_runners/linux.html)

## String: Image as inputs

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

```
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
```



# Benefits



# Building blocks

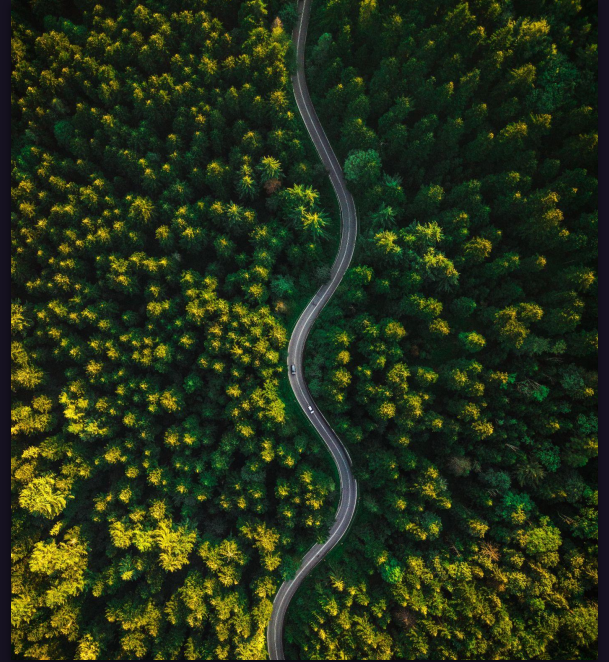
Programming languages - lint, build, test best practices

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>



to-be-continuous/python

Python 6.11.1

Build template for Python

Readme Components

## 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`:

```
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
```

#### Use as a CI/CD template (legacy)

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

```
include:
  # 1: include the template
  - project: 'to-be-continuous/python'
    ref: '6.11.1'
    file: '/templates/gitlab-ci-python.yml'

variables:
  # 2: set/override template variables
  PYTHON_IMAGE: registry.hub.docker.com/library/python:3.10
  PYTEST_ENABLED: "true"
```



# 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>



google-gitlab-components/gke

GKE 0.1.0

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

[Readme](#) [Components](#)

## deploy-gke

Status: [Beta](#)

- Requires `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:

```
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"
```

### Authentication

- The component authenticates to Google Cloud services by workload Identity Federation. Please follow [GitLab - Google Cloud integration onboarding process](#) to complete the workload identity Federation setup.
- Requires `google_cloud_support_feature_flag` (Beta) flag need to be enabled.

### Inputs

# Maintained components

Highlight: GitLab - Ruby

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

Explore / CI/CD Catalog



components/ruby  
Ruby 1.1.0

A collection of components to build, lint, test and deploy Ruby projects.

[Readme](#) [Components](#)

## Ruby

The Ruby CI components provide a way to [lint](#) and [test](#) Ruby code.

### All Jobs

The [all-jobs](#) component includes both the [lint](#) and [test](#) components.

You can add it to your `.gitlab-ci.yml` by adding a block along the lines of:

```
include:
  - component: gitlab.com/components/ruby/all-jobs@~latest
  inputs:
    project_path: example_project
    ruby_image: ruby:3.2
```

### Lint

The [lint](#) component runs the static code analyzer [Rubocop](#) against your Ruby code.

GitLab components / Ruby / Jobs / #6119868481

Search job log

```
85 Installing rubocop 1.60.2
86 Bundle complete! 5 Gemfile dependencies, 30 gems now installed.
87 Bundled gems are installed into `./vendor`
88 $ bundle exec rubocop
89 Inspecting 6 files
90 .CC...
91 Offenses:
92 Lib/example.rb:3:1: C: Style/Documentation: Missing top-level documentation comment for class Example.
93 class Example
94   ^^^^^^^^^^^^^
95 Lib/example.rb:7:5: C: [Correctable] Style/GuardClause: Use a guard clause (return unless 1 == 2) instead of wrapping the code inside a conditional expression.
96   if 1 == 2
97     ^^
98 Lib/example.rb:7:5: C: [Correctable] Style/IfUnlessModifier: Favor modifier if usage when having a single-line body. Another good alternative is the usage of control flow &&|||
```

Duration: 37 seconds  
Finished: 14 hours ago  
Queued: 0 seconds  
Timeout: 1h (from project)

Runner: #12270840 (-AzERasQr) 5-blue.sas-linux-small-amd64.runners-manager.gitlab.com/default

Commit 8ab4b61b in !1  
Fix spec (example block)

Pipeline #1168298692 Failed for initial-templates-and-test-pipelines






# Maintained components

Highlight: GitLab - OpenTofu

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



Explore CI/CD Catalog GitLab Duo Chat

 components/opentofu  
**OpenTofu** 0.18.0-rc1

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.

[Readme](#) [Components](#)

## OpenTofu CI/CD Component

 **NOTE** 


The `src/gitlab-tofu.sh` script is still merely a copy from `gitlab-terraform`. Therefore, lots of things in this script and in the templates are still Terraform-related and haven't been changed to their OpenTofu equivalents. Have a look at the [Migrating from the Terraform CI/CD templates](#) section when migrating from Terraform CI/CD templates.

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.

Read more:

- [CI/CD components](#)
- [Development guide for GitLab CI/CD components](#)
- [CI/CD Catalog](#)

**Note:** Please make sure to use a released version of this CI/CD component. You find all releases on the [Releases Overview Page](#).

 **Migrating from the Terraform CI/CD templates?** Check [this](#) out.

- OpenTofu CI/CD Component
  - Usage
    - Opinionated Templates
    - Job Templates
    - Inputs
    - Available OpenTofu Versions
    - Variables
    - Install additional tools
  - Releases & Versioning
    - Component Versions
    - Image Versions
  - Usage on self-managed
  - Migrating from the Terraform CI/CD templates
    - OpenTofu component inputs vs. Terraform template variables
  - Can I use this component with Terraform?
  - Contributing

## Usage

```
include:
- component: gitlab.com/components/opentofu/full-pipeline@<VERSION>
  inputs:
    # The version must currently be specified explicitly as an input,
    # to find the correctly associated images. # This can be removed
```



# 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



# Everyone can contribute

Create



# 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>



# 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.

```
9 image: goLang:latest
10
11 stages:
12   - test
13   - build
14   - deploy
15
16 format:
17   stage: test
18   script:
19     - go fmt $(go list ./... | grep -v /vendor/)
20     - go vet $(go list ./... | grep -v /vendor/)
21     - go test -race $(go list ./... | grep -v /vendor/)
22
23 compile:
24   stage: build
25   script:
26     - mkdir -p mybinaries
27     - go build -o mybinaries ./...
28 artifacts:
29   paths:
30     - mybinaries
31
```



## 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.

```
9 image: goLang:latest
10
11 stages:
12   - test
13   - build
14   - deploy
15
16 format:
17   stage: test
18   script:
19     - go fmt $(go list ./... | grep -v /vendor/)
20     - go vet $(go list ./... | grep -v /vendor/)
21     - go test -race $(go list ./... | grep -v /vendor/)
22
23 compile:
24   stage: build
25   script:
26     - mkdir -p mybinaries
27     - go build -o mybinaries ./...
28 artifacts:
29   paths:
30     - mybinaries
31
```





### 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*

```
build.yml 761 B  Blame  Edit  Lock  Replace  Delete    
1 spec:  
2   inputs:  
3     job_name:  
4       default: 'build'  
5       description: 'Defines the job name'  
6     stage:  
7       default: 'build'  
8       description: 'Defines the stage'  
9     go_image:  
10    default: 'golang:latest'  
11    description: 'Defines the Go container image'  
12    binary_directory:  
13    default: 'mybinaries'  
14    description: 'Output directory for created binary artifacts'  
15    ---  
16  
17    "$[[ inputs.job_name ]]:"  
18    stage: $[[ inputs.stage ]]  
19    image: $[[ inputs.go_image ]]  
20    script:  
21      - mkdir -p $[[ inputs.binary_directory ]]  
22      - |  
23        if [ -n "$[[ inputs.binary_directory ]]" ]; then  
24          go build -o $[[ inputs.binary_directory ] ] ./...  
25        else  
26          go build ./...  
27        fi  
28    artifacts:  
29      paths:  
30      - $[[ inputs.binary_directory ]]  
31
```



## 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
1 spec:
2   inputs:
3     job_name:
4       default: 'format'
5       description: 'Defines the job name'
6     stage:
7       default: 'format'
8       description: 'Defines the stage'
9     go_image:
10      default: 'golang:latest'
11      description: 'Defines the Go container image'
12 ---
13
14 "$[[ inputs.job_name ]]":
15   stage: $[[ inputs.stage ]]
16   image: $[[ inputs.go_image ]]
17   script:
18     - go fmt $(go list ./... | grep -v /vendor/)
19     - go vet $(go list ./... | grep -v /vendor/)
20
```



## 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

```
test.yml 427 B [Blame] [Edit] [Lock] [Replace] [De]
1 spec:
2   inputs:
3     job_name:
4       default: 'test'
5       description: 'Defines the job name'
6     stage:
7       default: 'test'
8       description: 'Defines the stage'
9     go_image:
10      default: 'golang:latest'
11      description: 'Defines the Go container image'
12 ---
13
14 "$[[ inputs.job_name ]]":
15   stage: $[[ inputs.stage ]]
16   image: $[[ inputs.go_image ]]
17   script:
18     - go test -race $(go list ./... | grep -v /vendor/)
```



## 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
Blame Edit Lock Replace Delete
1 include:
2   - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/format@$CI_COMMIT_SHA
3     inputs:
4       job_name: format
5   - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/build@$CI_COMMIT_SHA
6     inputs:
7       job_name: build
8   - component: $CI_SERVER_FQDN/$CI_PROJECT_PATH/test@$CI_COMMIT_SHA
9     inputs:
10      job_name: test
11
12
13 stages:
14   # Component specific stages
15   - format
16   - build
17   - test
18   # Needed for CI/CD component release
19   - release
20
21 .go-images:
22   parallel:
23     matrix:
24       - GOLANG_IMAGE:
25         - golang:1.22
26         - golang:1.21
27
28   # Keep the job names in sync with `job_name` inputs above.
29   format:
30     parallel: !reference [.go-images,parallel]
31
32   build:
33     parallel: !reference [.go-images,parallel]
34
35   test:
36     parallel: !reference [.go-images,parallel]
37
```



## 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.

```
README.md 9+  main.go 2, M X  .gitlab-ci.yml 6  .gitignore  ! build

main.go
1 // Create package main
2 // Add all required imports
3 // Create a function that greets the CI/CD component user
4 // Add functions to calculate PI
5 // Add function that allocates 10 MB memory
6 // Call all functions in the main package, and main function
7 // End with saying that everything was generated with the help of GitLab Duo
8
package main

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

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

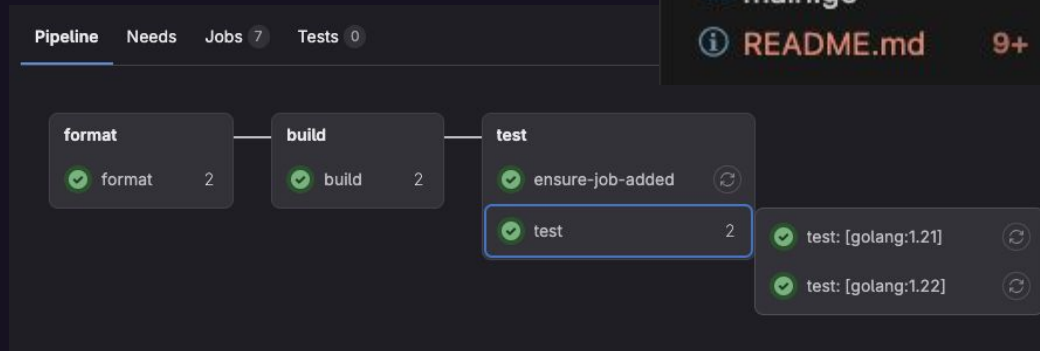
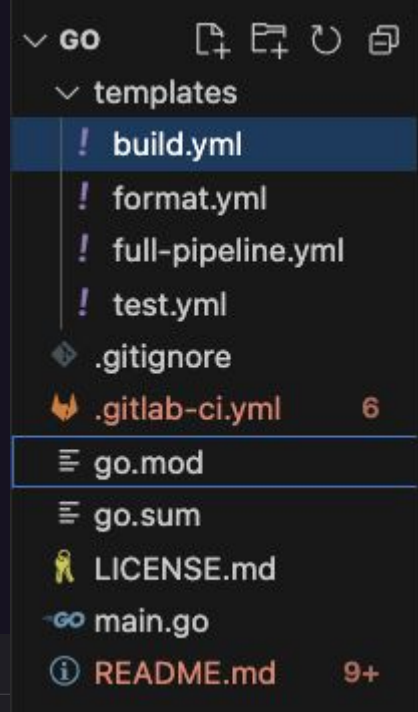
func calculatePi(iterations int) float64 {
    var pi float64 = 0
    for i := 0; i < iterations; i++ {
        pi += math.Pow(-1, float64(i)) * 4 / (2 * float64(i) + 1)
    }
    return pi
}

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



## 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







# 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



# Resources

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



# Thank you