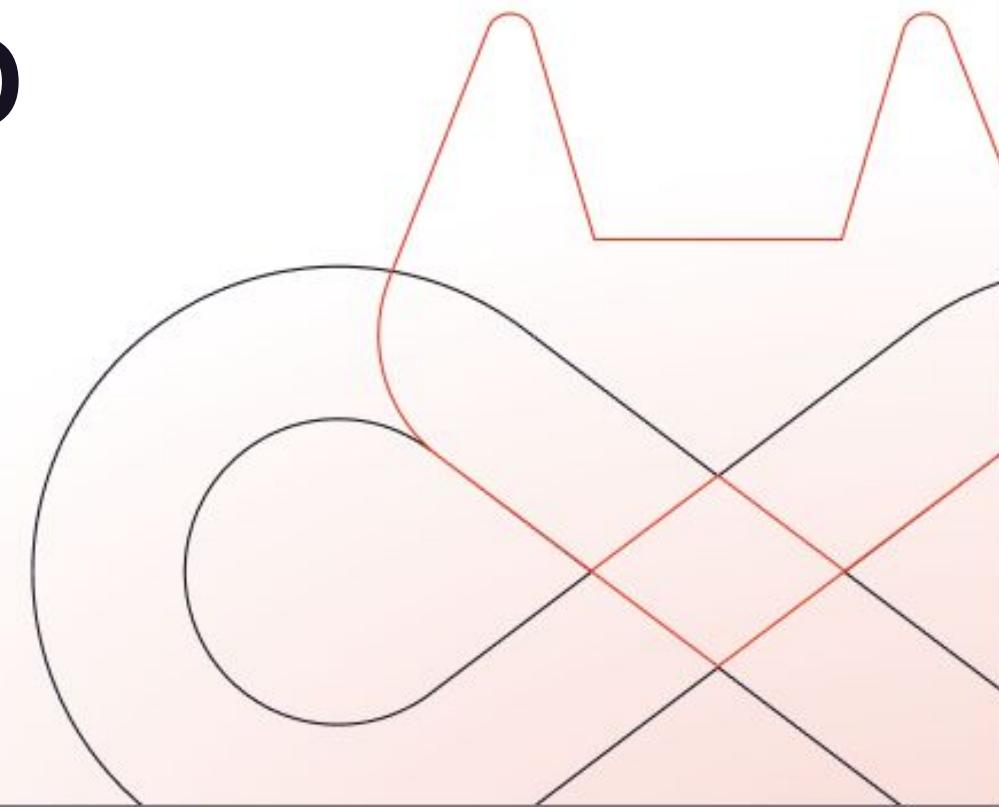




# Advanced GitLab CI/CD



# Agenda



1. **CI/CD Basics: a quick review**
2. **Pipeline structure and ways to hack it**  
Pipeline Architectures
3. **Variables: a foundation before using rules**
4. **Controlling when your jobs run**  
The “rules” label
5. **Holding, securing, and sharing results**  
Job Artifacts
6. **Assembling pipelines from components**  
Using “includes” and “extends” labels

This workshop assumes that participants are familiar with basic GitLab CI/CD.

It presents advanced concepts and techniques for people who are active users of GitLab CI.

We do not cover Runner configuration and any of the CD-specific features such as Environments and Deployments.

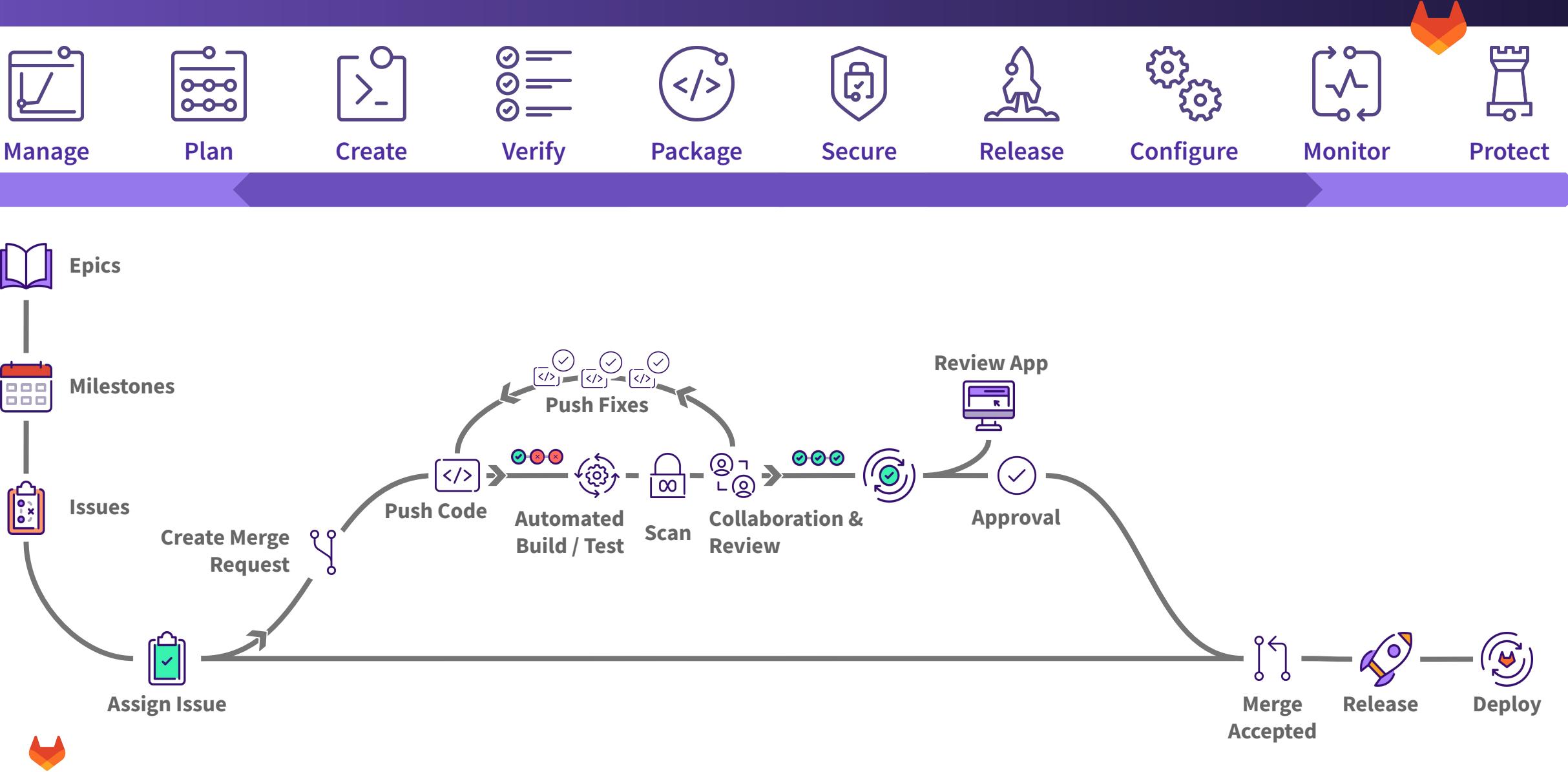




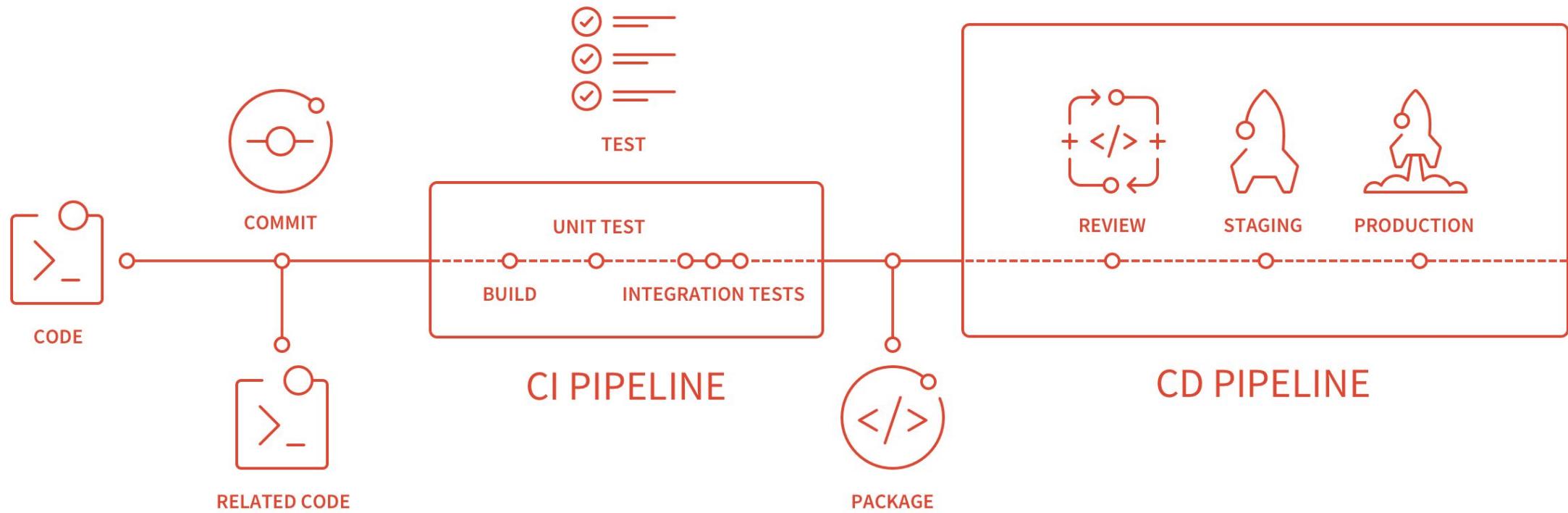
# GitLab CI/CD Basics Review



# GitLab Recommended Process



# GitLab CI/CD



# Anatomy of a GitLab CI/CD build



## Pipeline

- Set of one or more jobs. Optionally organized into stages

## Stages

- Collection of jobs to be run in parallel
- e.g. Build, Test, Deploy

## Jobs

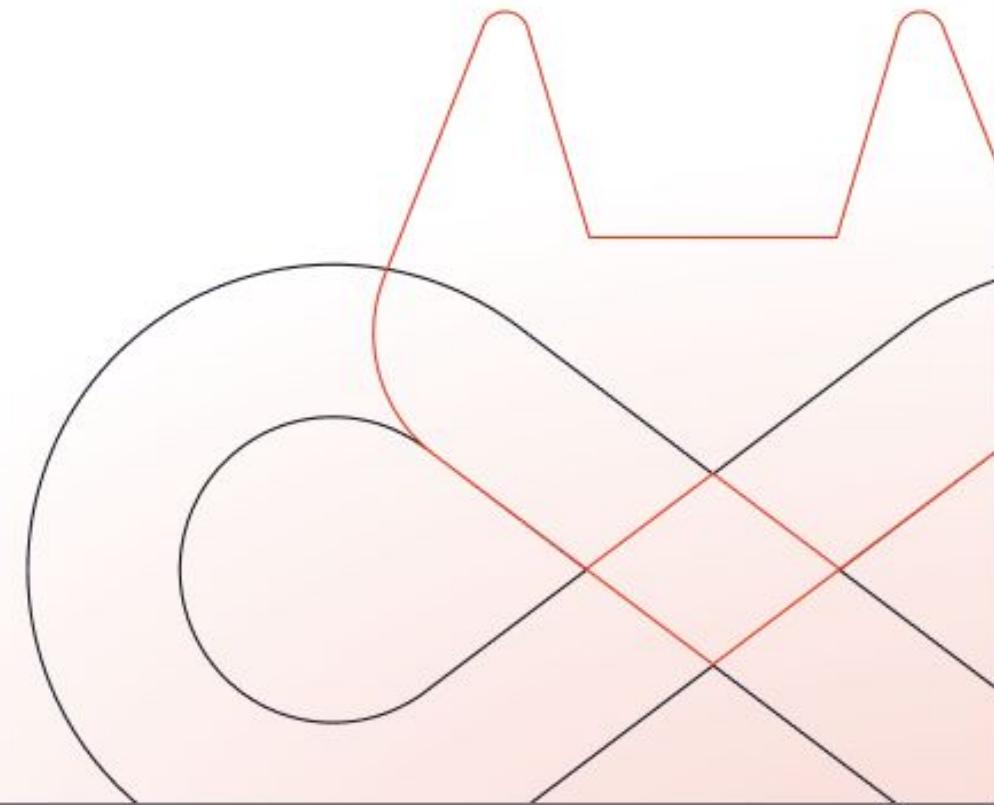
- Scripts that perform tasks
- e.g. npm test; mvn install; etc.

## Environments

- Where we deploy (Test, Review, Staging, Canary, Prod)



# Pipeline Architectures





- The building-blocks of your pipeline.
  - [Basic](#)
  - [Directed Acyclic Graph](#)
  - [Parent/Child Pipelines](#)
  - [Dynamic Child Pipelines](#)
  - [Multi-Project Pipelines](#)

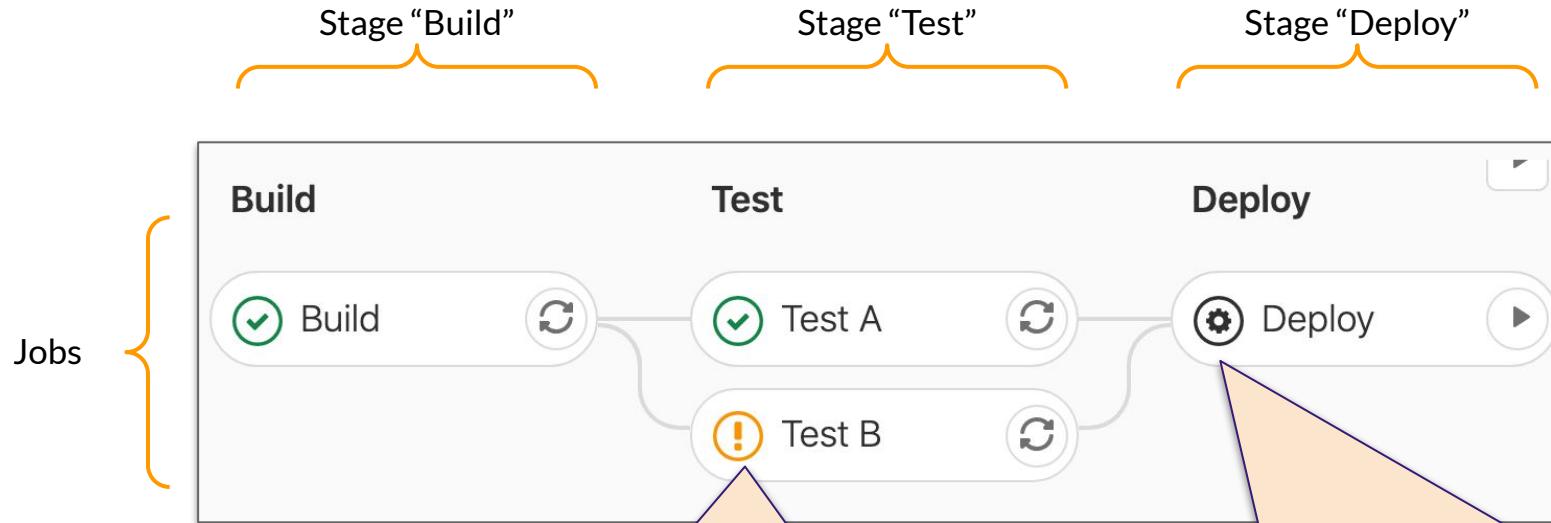




# Basic pipelines

- Simplest pipeline in GitLab
- Jobs run independently, sometimes on different runners
- All Jobs in a Stage must complete successfully before proceeding to the next stage
- Control pipeline with pipeline options

# Pipeline Architectures - Basic pipeline with options



Other options:  
`when: delayed`  
`when: on_failure`  
`when: always`

This job has  
`allow_failure: true`  
...so the pipeline proceeds even though it failed

This job has  
`when: manual`  
...so it waits for someone (with the right permissions) to click a  
“Play” button

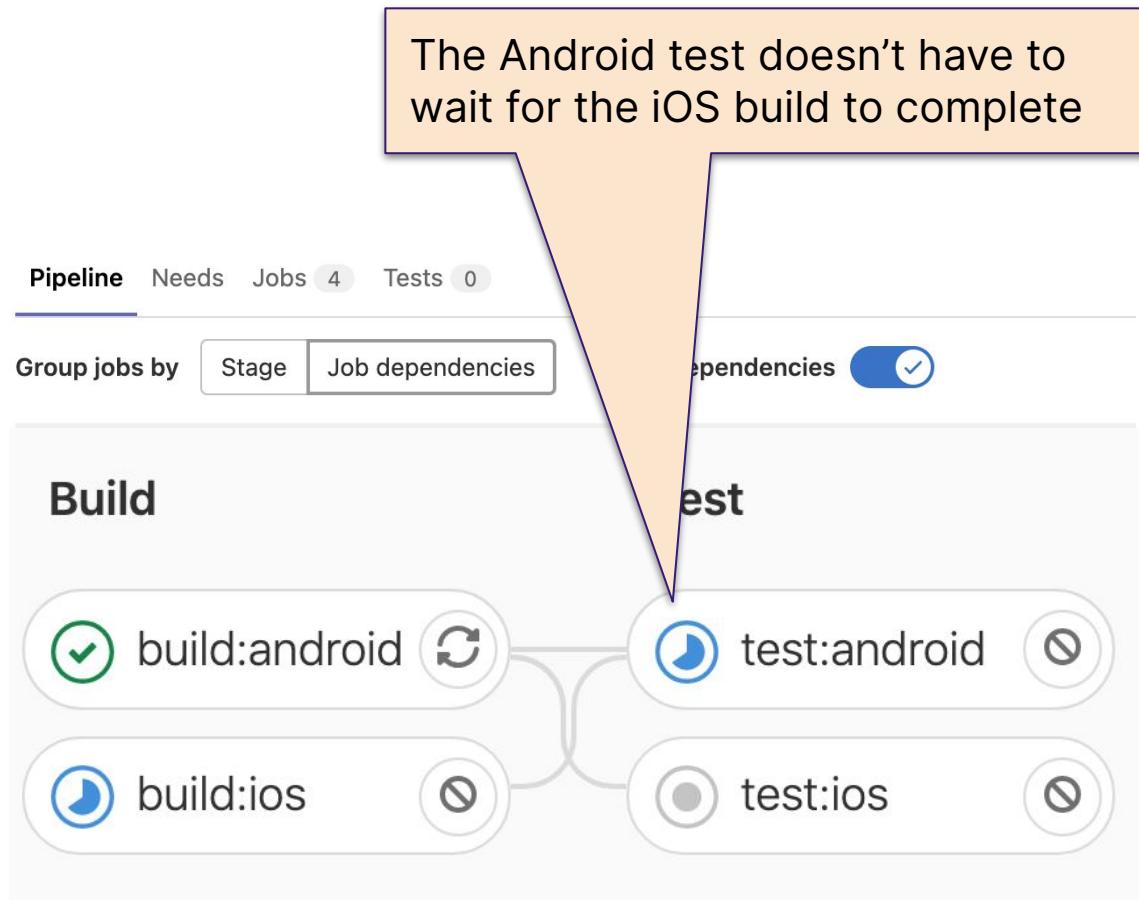




# Needs or Directed Acyclic Graph (DAG)

- Define job dependencies to optimize pipeline flow
- Jobs still run independently, sometimes on different runners
- Dependent jobs can proceed to next stage without waiting for other jobs in stage to finish

# Pipeline Architectures - Directed Acyclic Graph (DAG)

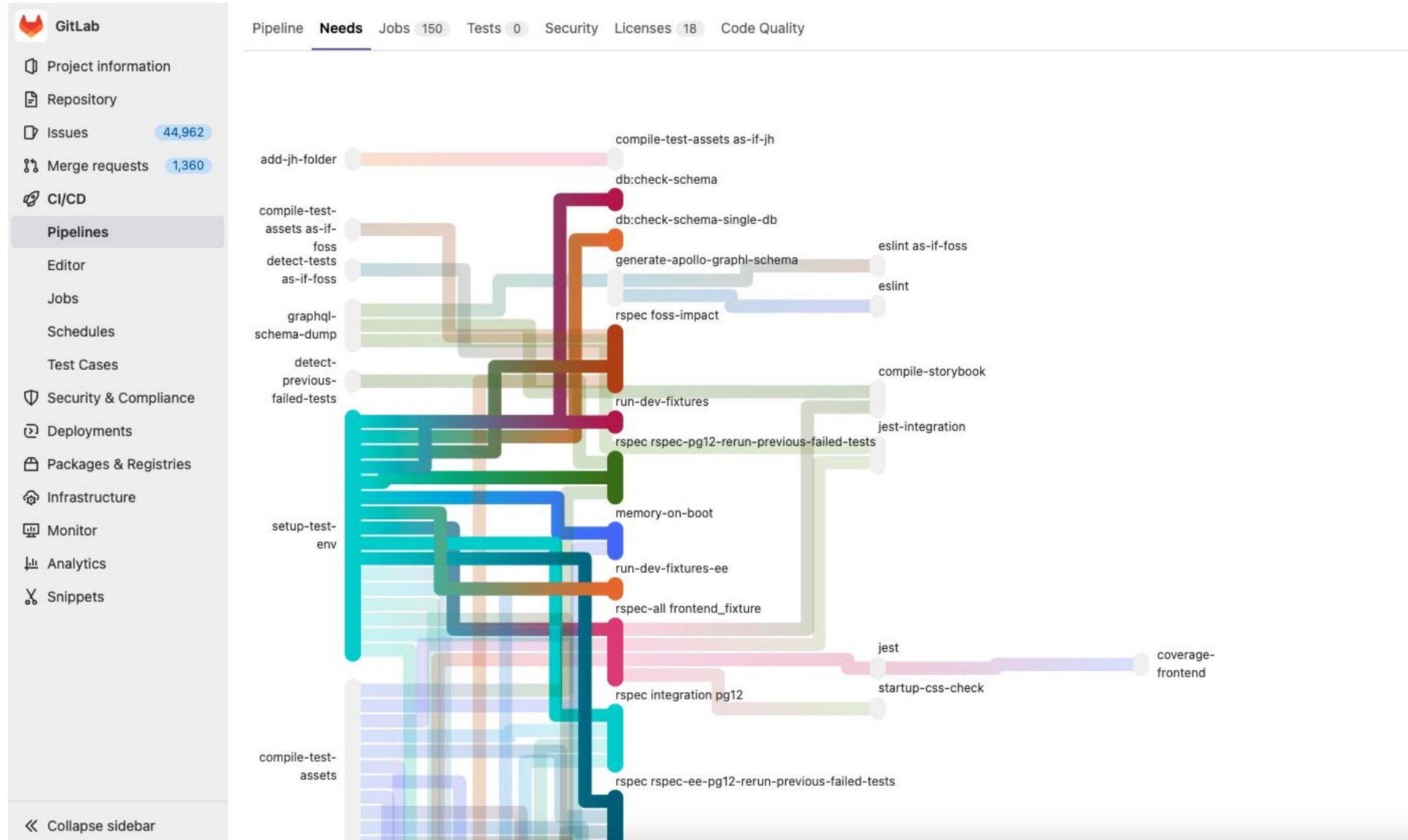


```
build:ios:  
stage: build  
script:  
  - echo "Building for iOS"  
  - sleep 10  
  
build:android:  
stage: build  
script:  
  - echo "Building for Android"  
  
test:ios:  
stage: test  
needs: ["build:ios"]  
script:  
  - echo "Testing for iOS"  
  
test:android:  
stage: test  
needs: ["build:android"]  
script:  
  - echo "Testing for Android"
```

Jobs still runs in stages, but the “needs” keyword overrides the need for the previous stage to complete entirely.



# Directed Acyclic Graph Pipelines



# Example: needs



```
linux:build:  
  stage: build  
  script: echo "Building linux..."  
  
mac:build:  
  stage: build  
  script: echo "Building mac..."  
  
lint:  
  stage: test  
  needs: []  
  script: echo "Linting..."  
  
linux:rspec:  
  stage: test  
  needs: ["linux:build"]  
  script: echo "Running rspec on linux..."  
  
mac:rspec:  
  stage: test  
  needs: ["mac:build"]  
  script: echo "Running rspec on mac..."  
  
production:  
  stage: deploy  
  script: echo "Running production..."
```

Linux path: the `linux:rspec` job will be run as soon as the `linux:build` job finishes without waiting for `mac:build` to finish.

macOS path: the `mac:rspec` job will be run as soon as the `mac:build` job finishes without waiting for `linux:build` to finish.

The `lint` job will run immediately without waiting for `build` jobs.

The `production` job runs as soon as all previous jobs finish.

Source: <https://docs.gitlab.com/ee/ci/yaml/#needs>

# Additional example: needs with artifacts



```
test-job1:  
  stage: test  
  needs:  
    - job: build_job1  
      artifacts: true  
  
test-job2:  
  stage: test  
  needs:  
    - job: build_job2  
      artifacts: false  
  
test-job3:  
  needs:  
    - job: build_job1  
      artifacts: true  
    - job: build_job2  
    - build_job3
```

The **test-job1** job downloads the **build\_job1** artifacts

The **test-job2** job does not download the **build\_job2** artifacts.

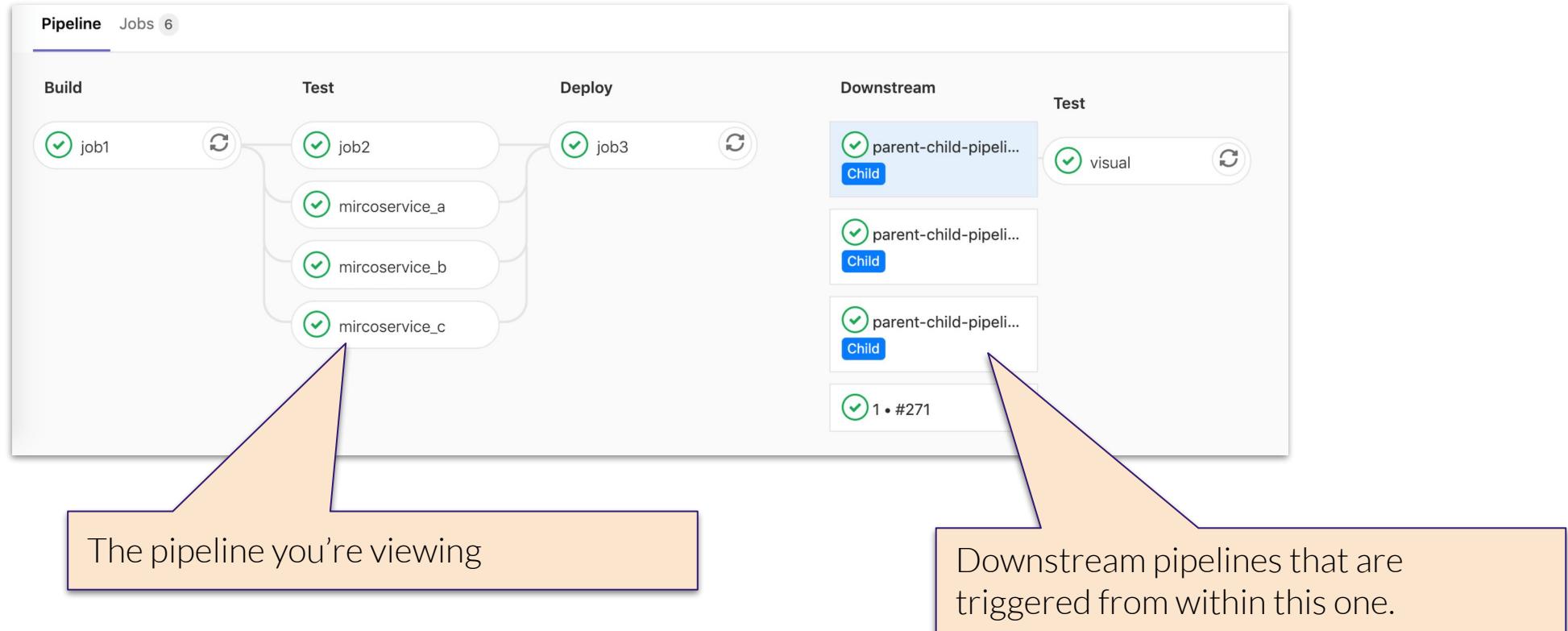
The **test-job3** job downloads the artifacts from all three **build\_jobs**, because **artifacts** is **true**, or defaults to **true**, for all three needed jobs.



# Parent/Child Pipelines

- Run child pipelines independent from each other.
- Separates entire pipeline configuration in multiple files to keep things simple.
- Combine with DAG pipelines to achieve benefits of both.
- Useful to branch out long running tasks into separate pipelines.

# Pipeline Architectures - Parent/Child Pipelines



# Pipeline Architectures - Parent/Child Pipelines for Monorepos



Run the job if there are changes in those files

```
stages:  
  - build  
  - finalize
```

```
build-1:  
  stage: build  
  only:  
    changes:  
      - project-1/*  
  trigger:  
    include: project-1/.gitlab-ci.yml  
    strategy: depend
```

Include files from elsewhere in the project - must reference a YAML file

**strategy: depend**  
means to hold this pipeline until the other pipeline finishes





# Dynamic Pipelines

- Generate pipeline configuration at build time
- Use the generated configuration at a later stage to run as a child pipeline.
- Useful to use a single pipeline configuration with different settings to support a matrix of targets and architectures

# Pipeline Architectures - Dynamic Pipelines



.gitlab-ci.yml

```
stages:
  - setup
  - test
  - finalize

setup:
  stage: setup
  script:
    - sed "s/REPLACEME/DynamicPipeline/g" test.gitlab-ci.yml > generated-config.yml
artifacts:
  paths:
    - generated-config.yml

test:
  stage: test
  trigger:
    include:
      - artifact: generated-config.yml
        job: setup
  strategy: depend

finalize:
  stage: finalize
  script:
    - echo "Finalize job"
```

A large curved orange arrow points from the top right of the .gitlab-ci.yml code block down towards the generated-config.yml artifact entry. This visual cue indicates that the generated configuration file is being stored as an artifact.

Place a generated YAML  
file in the job artifact store

test.gitlab-ci.yml

```
stages:
  - unit tests
  - integration tests

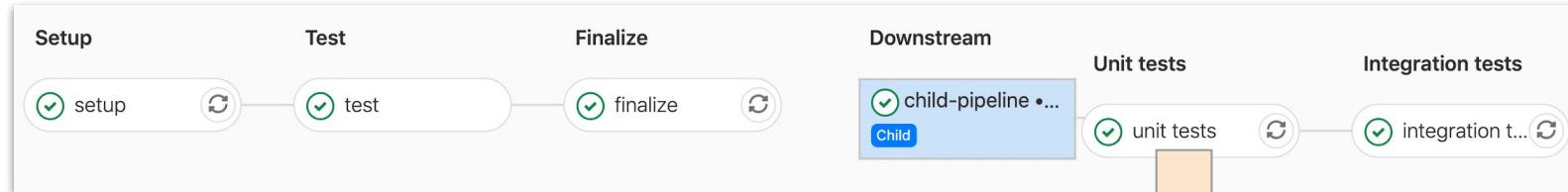
unit tests:
  stage: unit tests
  script:
    - echo "REPLACEME"

integration tests:
  stage: integration tests
  script:
    - echo "REPLACEME"
```

Reference it later to  
actually run the pipeline



# Pipeline Architectures - Dynamic Pipelines



passed Job #673072268 triggered 9 minutes ago by Francis Potter

```
1 Running with gitlab-runner 13.2.2 (a998cacd)
2 on docker-auto-scale fa6cab46
3 Preparing the "docker+machine" executor
4 Using Docker executor with image ruby:2.5 ...
5 Pulling docker image ruby:2.5 ...
6 Using docker image sha256:896afe8ad24a941448575390111af68ddcf195105b8d1f23e60461c0b8c4ed3c for ruby:2.5 ...
7 Preparing environment
8 Preparing environment
9 Running on runner-fa6cab46-project-20364915-concurrent-0 via runner-fa6cab46-srm-1596667784-f3980b85...
10 Getting source from Git repository
11 eval "$CI_PRE_CLONE_SCRIPT"
12 Fetching changes with git depth set to 50...
13 Initialized empty Git repository in /builds/fpotter/examples/dynamic-pipelines/.git/
14 Created fresh repository.
15 Checking out 1bf7ef6f as master...
16 Skipping Git submodules setup
17 Executing "step_script" stage of the job script
18 $ echo "DynamicPipeline"
19 DynamicPipeline
20 Job succeeded
```

The replacement happened  
in the parent pipeline





# Multi-project Pipelines

- A pipeline in one project can trigger a pipeline in another project
- You can specify a specific branch
- You can pass variables to downstream pipelines
- If downstream pipeline fails it will not fail upstream pipeline
- Useful when building / deploying large applications that are made up of different components that have their own project and build pipeline

Documentation link:

[https://docs.gitlab.com/ee/ci/multi\\_project\\_pipelines.html](https://docs.gitlab.com/ee/ci/multi_project_pipelines.html)

# Pipeline Architectures - Multi-Project Pipelines



```
test:  
  stage: test  
  script:  
    - echo "I am building!"  
  
bridge:  
  variables:  
    ENVIRONMENT: staging  
  stage: test  
  trigger:  
    project: my/project  
    branch: master  
    strategy: depend  
  forward:  
    pipeline_variables:
```

Pass variables to downstream projects

Specify project and branch

Define what variables to forward

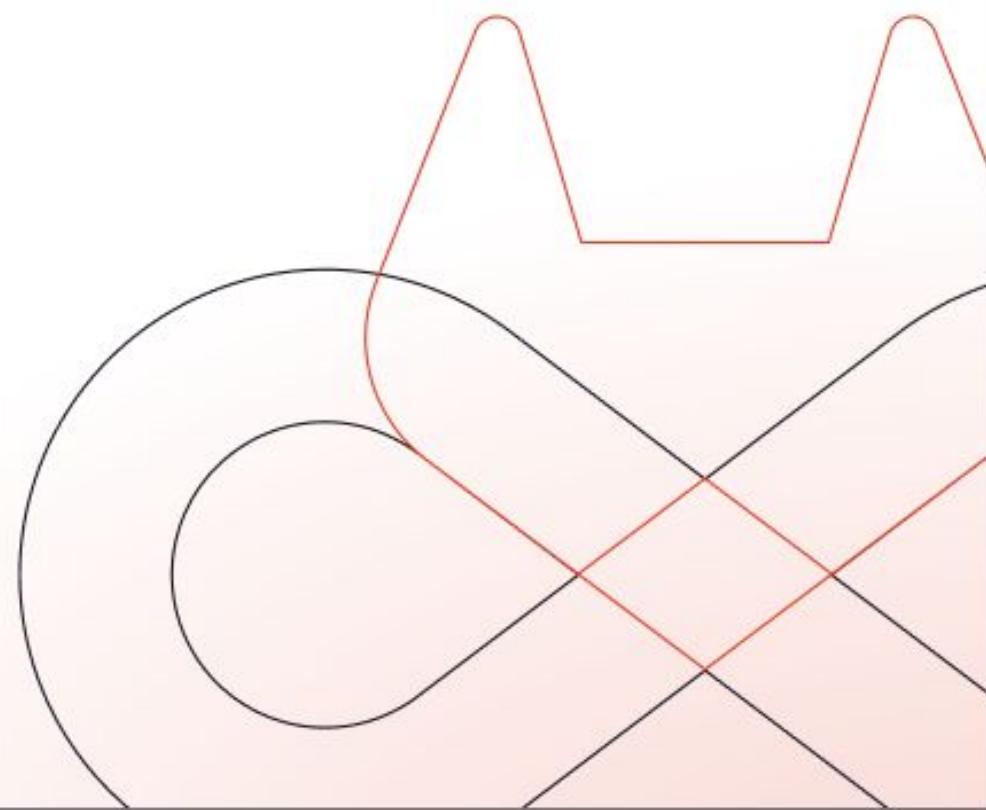
The job with the trigger is referred to as the “bridge” job

Downstream pipeline can be anything

```
staging:  
  stage: deploy  
  script:  
    - echo "Deploying to $ENVIRONMENT!"
```



# Variables



# Variables in UI at project level



The screenshot shows the 'Variables' section of the 'chipmunk' project settings. The left sidebar highlights the 'Settings' and 'CI / CD' sections. The main content area displays environment variables with two entries:

Type	Key	Value	Protected	Masked	Environments
Variable	CI_DEBUG_TRACE	*****	✓	✗	All (default)
Variable	SAST_DISABLED	*****	✓	✗	All (default)

Buttons for 'Reveal values' and 'Add Variable' are visible. A callout box highlights the 'Variables' section with the text: "Variables can also be set at the Group and Instance level". Other collapsed sections shown include 'Group variables (inherited)', 'Pipeline triggers', and 'Cleanup policy for tags'.



# Variables entered for the pipeline run



The screenshot shows the GitLab Pipelines interface for the project 'chipmunk'. The left sidebar is highlighted with a red oval around the 'Pipelines' option. The main area displays a list of pipeline runs, with the first run (#160244003) having its 'Run Pipeline' button circled in red. An orange arrow points from this button to a detailed 'Run Pipeline' modal window on the right. The modal allows setting the branch ('Run for') to 'master' and defining variables ('Variables') using URL parameters. A purple callout box at the bottom right contains a 'Hot tip!' about prepopulating variables via URL parameters, followed by a code snippet.

Run Pipeline

All 85 Finished Branches Tags

Filter pipelines

Status	Pipeline	Triggerer	Commit	Stages
skipped	#160244003	p/test-bran...	ce1aa56c	Add CODEOWNERS
failed	#158950442	p41-chang...	c5fd1d26	Update db/demo.txt
skipped	#158950354	p41-chang...	ce1aa56c	Add CODEOWNERS
skipped	#158947065	p/master	ce1aa56c	Add CODEOWNERS
failed	#153772549	p/master	zeccca75	Merge branch '8-m...
passed	#153052255	p7-test-te...	8980e09d	Est tempora sit sit d...
failed	#153051428	p7-test-te...	zeccca75	Merge branch '8-m...
passed	#152990369	p10-non-d...	zeccca75	Merge branch '8-m...

Run Pipeline

Run for

master

Existing branch name or tag

Variables

Variable

Input variable key

Input variable value

Specify variable values to be used in this run. The values specified in C

Run Pipeline

Hot tip!

Prepopulate the keys and values using URL parameters

.../pipelines/new?ref=<branch>&var[<variable\_key>]=<value>

# Variables defined in the CI configuration



```
variables:  
  GLOBAL_VAR: test  
  
build:  
  variables:  
    ENVIRONMENT: staging  
  script:  
    - echo "Trying $GLOBAL_VAR on $ENVIRONMENT"
```



# Secrets



```
deploy:
  stage: deploy
  secrets:
    DATABASE_PASSWORD:
      vault: # Translates to secret: `ops/data/production/db`, field: `password`
        engine:
          name: kv-v2
          path: ops
          path: production/db
          field: password
          file: false

# Shortened version:
# vault: production/db/password@ops
```



# Inherited variables

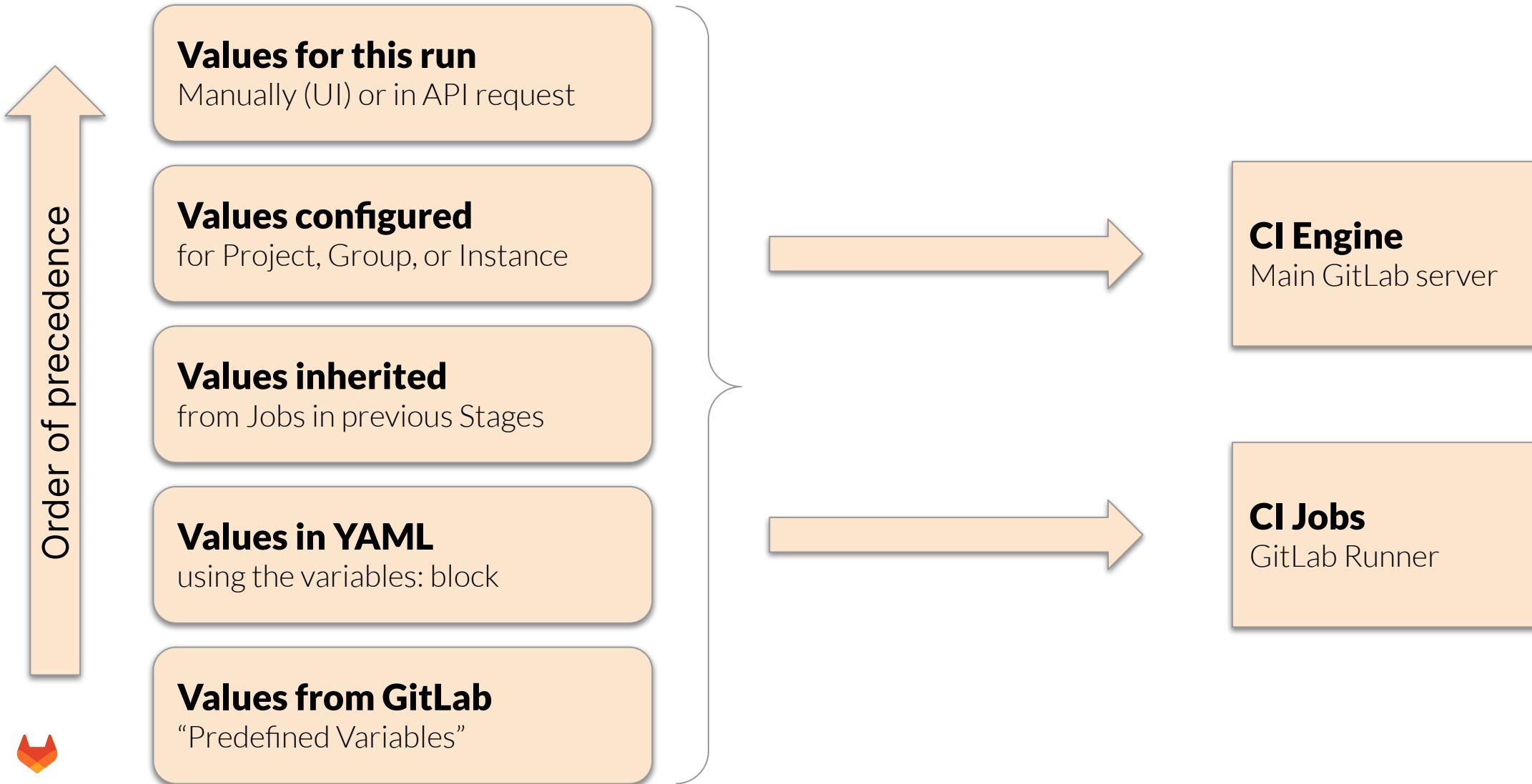


```
build:
  stage: build
  script:
    - echo "BUILD_VARIABLE=value_from_build_job" >> build.env
  artifacts:
  reports:
    dotenv: build.env

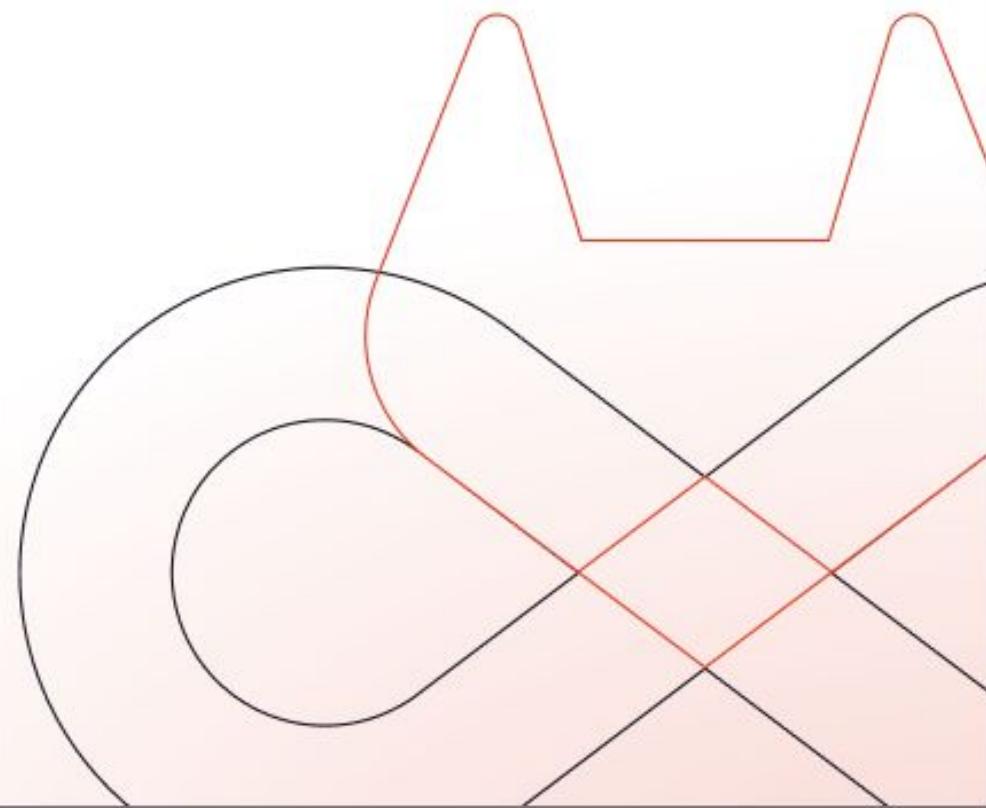
deploy:
  stage: deploy
  variables:
    BUILD_VARIABLE: value_from_deploy_job
  script:
    - echo "$BUILD_VARIABLE" # Output is: 'value_from_build_job' due to precedence
```



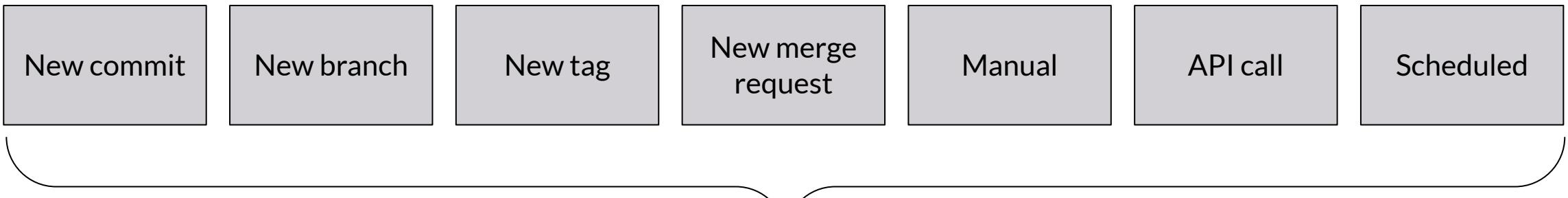
# How variables are processed



# Rules



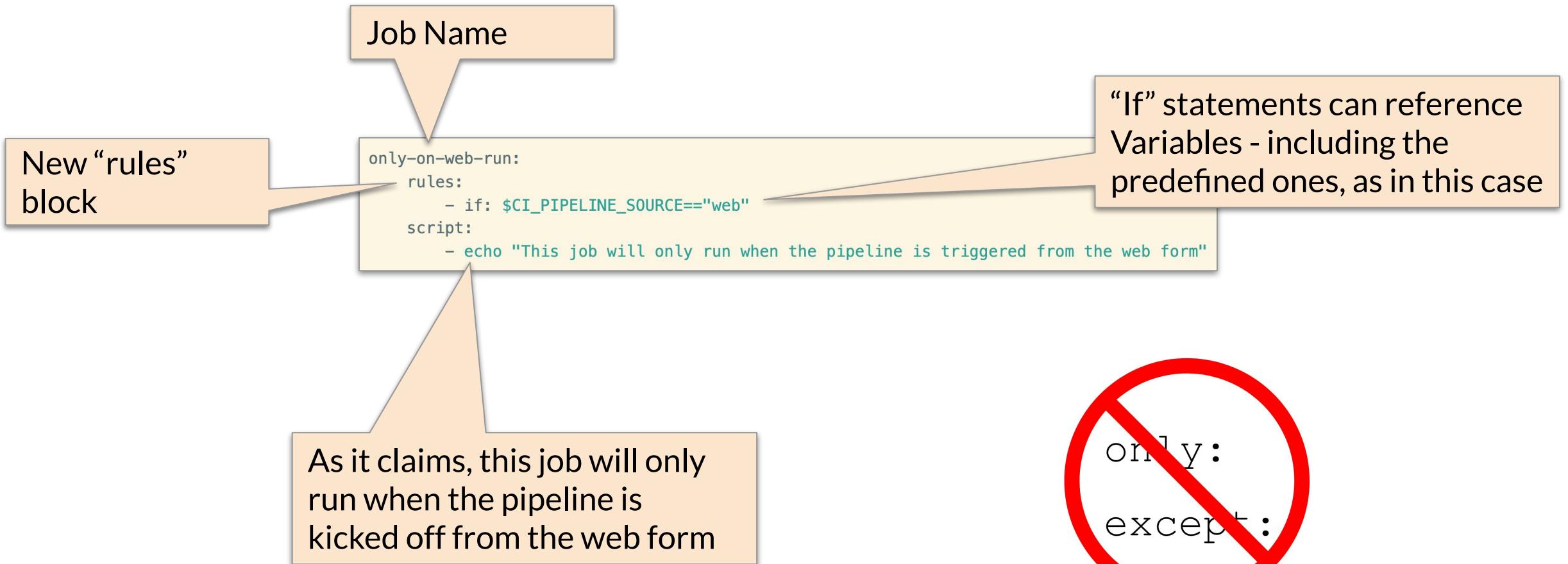
# When are pipelines run?



The screenshot shows the GitLab interface for a pipeline. At the top, it displays a message: "Pipeline #66796601 triggered 1 minute ago by Francis Potter". Below this, the pipeline name is listed as "Labore tempora sed voluptatem eius est ut". It shows 11 jobs queued for 36 seconds. The pipeline stages are: Build, Test, Review, Dast, and Performance. The Build stage has one job named "build" which has passed. The Test stage contains several jobs: "code\_quality", "container\_scan...", "dependency\_sc...", "license\_manag...", "sast", and "test", all of which have passed. The Review stage has one job named "review" which has passed. The Dast stage has one job named "dast" which has passed. The Performance stage has one job named "performance" which has passed. The sidebar on the left shows navigation links for Projects, Groups, Activity, Milestones, Snippets, and a search bar.



# Rules: The basics



# Rules Quick Reference



## Clauses

if  
changes  
exists  
(... or none)

## Operators

(just the variable)

==  
!=  
=~  
!~  
&&  
||

## Results

when  
allow\_failure  
start\_in

## “when” options

always  
never  
on\_success  
on\_failure  
manual  
Delayed  
(... or none)





## Default values

when: on\_success

allow\_failure: false

## Job is added when:

A rule matches and has either:

- when: on\_success
- when: delayed
- when: always

No match, but last clause is:  
**(As standalone attribute)**

- when: on\_success
- when: delayed
- when: always

## Job is not added when:

No rules matches and no standalone:

- when: on\_success
- when: delayed
- when: always

A rule matches with:

- when: never



# Rules Example 01



```
job:  
  Script: "echo Hello, Rules!"  
  rules:  
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'  
    - if: '$CI_PIPELINE_SOURCE == "push"'
```

- Control when merge request pipelines run.
  - if: '\$CI\_PIPELINE\_SOURCE == "merge\_request\_event"'
- Control when both branch pipelines and tag pipelines run.
  - if: '\$CI\_PIPELINE\_SOURCE == "push"'



# Rules Example 02



```
job:  
  script: "echo Hello, Rules!"  
  rules:  
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'  
      when: never  
    - if: '$CI_PIPELINE_SOURCE == "schedule"'  
      when: never  
    - when: on_success
```

- This job will not run if this was triggered from a merge request
  - if: '\$CI\_PIPELINE\_SOURCE == "merge\_request\_event"'
- This job will not run if pipeline was scheduled
  - if: '\$CI\_PIPELINE\_SOURCE == "schedule"'
- Otherwise, this job will run if the previous stage was successful



# Rules Example 03



```
job:  
  script: "echo This creates double pipelines!"  
  rules:  
    - if: '$CUSTOM_VARIABLE == "false"'  
      when: never  
    - when: always
```

- This job will not run if `CUSTOM_VARIABLE` is equal to false
  - `if: '$CUSTOM_VARIABLE == "false"'`
- Otherwise, this job will run even if the previous stage had a failure



# Rules Example 04



```
docker build:  
script: docker build -t my-image:$CI_COMMIT_REF_SLUG .  
rules:  
- if: '$VAR == "string value"'  
  changes: # Will include the job and set to when:manual if any of the follow paths match a modified file  
  - Dockerfile  
  - docker/scripts/*  
  when: manual  
# - when: never would be redundant here, this is implied any time rules are listed.
```

- This job will be in the pipeline and will be a manual job
  - if: '\$VAR == "string value"' and there was a change to Dockerfile or any file in docker/scripts
- Otherwise, this job will not run



# Rules Example 05



```
docker build:  
  script: docker build -t my-image:$CI_COMMIT_REF_SLUG .  
  rules:  
    - if: '$CI_COMMIT_BRANCH == "main"'  
      when: delayed  
      start_in: '3 hours'  
      allow_failure: true
```

- This job will run 3 hours after triggered and will be allowed to fail (will not prevent further stages from firing)
  - if: '\$CI\_COMMIT\_BRANCH == "main"'
- Otherwise, this job will not run



# Workflow Rules



```
workflow:  
  rules:  
    - if: $CI_COMMIT_MESSAGE =~ /-wip$/  
      when: never  
    - if: $CI_COMMIT_TAG  
      when: never  
    - when: always
```

- This pipeline will not run if the commit message ends with “-wip”
  - if: '\$CI\_COMMIT\_MESSAGE =~ /-wip\$/'
- This pipeline will not run if this was triggered by a TAG being applied
  - if: '\$CI\_COMMIT\_TAG'
- Otherwise, this pipeline will run



# Workflow - Variables depending on



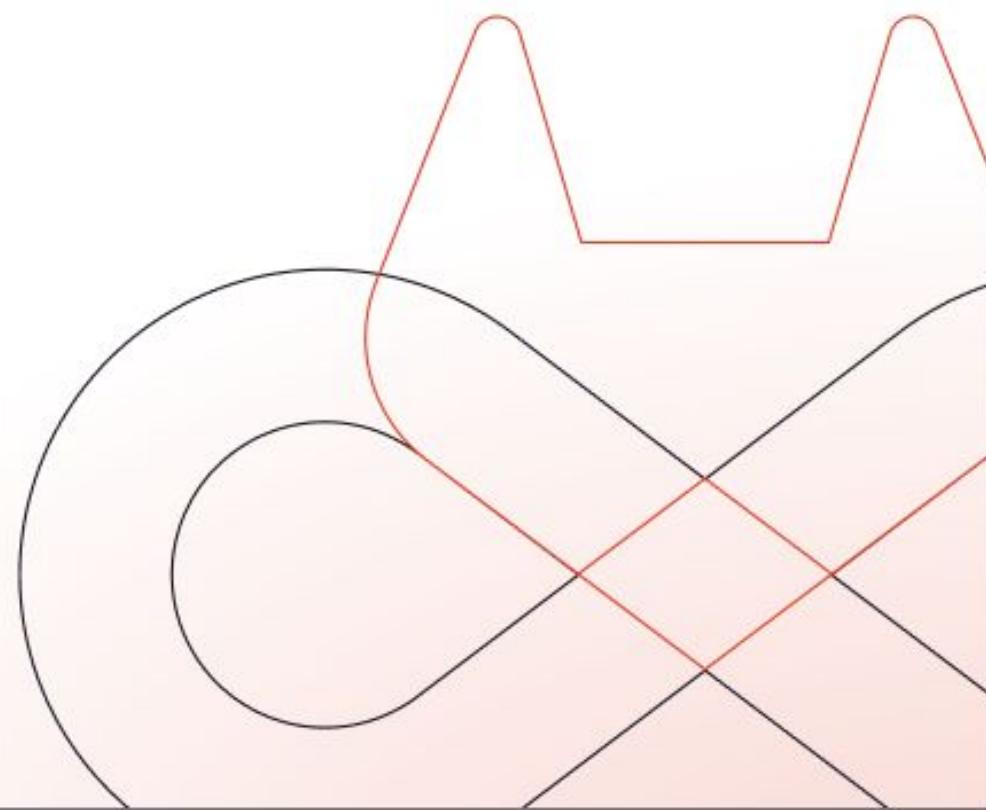
```
workflow:  
  rules:  
    - if: $CI_COMMIT_REF_NAME =~ /-wip$/  
      variables:  
        DOCKER_FILE: test.dockerfile  
    - if: $CI_COMMIT_REF_NAME == $CI_DEFAULT_BRANCH  
      variables:  
        DOCKER_FILE: main.dockerfile  
    - when: always
```

- The variable `DOCKER_FILE` will be set differently based on the rules.
- This pipeline will always run





# Artifacts



# CI configuration



- Allows for **saving of build artifacts** and/or the output of any job
- Are available for use by **subsequent jobs**
- Can pull in **any combination of paths and/or files**
- Use **exclude** to **limit what is added**
- Use **depends** to **limit what gets downloaded** on subsequent jobs
- Use **when** to determine **if artifacts will be stored** or not
- Use **expire\_in** to determine **when artifacts will be destroyed**

```
artifacts:  
  when: on_success  
  paths:  
    - bin/target  
    - .exe  
  exclude:  
    - throw-away.exe  
  expire_in: 3mos
```



# Artifact download by URL



Three ways of  
accessing Artifacts  
via secure HTTPS

**Build product for a specific CI pipeline run:**

```
https://gitlab.com/fpotter/examples/c-with-artifact/-/jobs/207917559/artifacts/file/helloworld
```

**Build product for the latest CI pipeline run on a specific branch (main):**

```
https://gitlab.com/fpotter/examples/c-with-artifact/-/jobs/artifacts/main/raw/helloworld?job=build
```

**All build products (as a Zip) for the latest CI pipeline run on a specific branch:**

```
https://gitlab.com/fpotter/examples/c-with-artifact/-/jobs/artifacts/main/download?job=build
```



# Artifact download in GitLab UI



## On the Pipelines page

Pipelines Jobs Environments Cycle Analytics

All 832 Running 0 Branches Tags

Status Pipeline Commit Stages

#6484362 by API latest ↗ 57a5b7d5 Merge branch 'revert-caa47ca... 00:13:35 5 hours ago Download compile artifacts

#6484314 by API latest ↗ 57a5b7d5 Merge branch 'revert-caa47ca... 00:14:53 5 hours ago

## On the Jobs page

Pipelines Jobs Environments Cycle Analytics

All 3,150 Pending 0 Running 0 Finished 2,989

Status Job Pipeline Stage Name Coverage

#10500942 by master ↗ 57a5b7d5 docker triggered #6484362 by API build compile 05:04 about 5 hours ago

#10500872 by master ↗ 57a5b7d5 docker triggered #6484314 by API deploy pages 05:08 about 5 hours ago

## On a specific Job

Job artifacts  
The artifacts will be removed in 6 days

Keep Download Browse

Job details  
Duration: 5 minutes 4 seconds  
Finished: about 5 hours ago  
Runner: #44028

Raw Erase

## Artifact browser (for a Job)

Artifacts / selenium

Name Size

Name	Size
..	
selenium	
some-log.json	17 Bytes
some-page.html	4 Bytes
some-text.txt	4 Bytes

Download artifacts archive



# Artifact administration



In a self-managed GitLab instance, job artifacts may be stored in **local storage or object storage**

Artifact **expiration times** can be configured at the instance level

Artifact downloads fall under GitLab's access control

## Project members permissions

>Note: In GitLab 11.0, the Master role was renamed to Maintainer.

The following table depicts the various user permission levels in a project.

Action	Guest	Reporter	Developer	Maintainer	Owner
Download project	✓ (1)	✓	✓	✓	✓
Leave comments	✓ (1)	✓	✓	✓	✓
View Insights charts <small>?</small>	✓	✓	✓	✓	✓
View approved/blacklisted licenses <small>?</small>	✓	✓	✓	✓	✓
View license management reports <small>?</small>	✓ (1)	✓	✓	✓	✓
View Security reports <small>?</small>	✓ (1)	✓	✓	✓	✓
View project code	✓ (1)	✓	✓	✓	✓
Pull project code	✓ (1)	✓	✓	✓	✓
View GitLab Pages protected by access control	✓	✓	✓	✓	✓
View wiki pages	✓ (1)	✓	✓	✓	✓
See a list of jobs	✓ (3)	✓	✓	✓	✓
See a job log	~ (3)	✓	✓	✓	✓
Download and browse job artifacts	✓ (3)	✓	✓	✓	✓
Create new issue	~ (1)	✓	✓	✓	✓
See related issues	✓	✓	✓	✓	✓
Create confidential issue	✓ (1)	✓	✓	✓	✓
View confidential issues	(2)	✓	✓	✓	✓
Assign issues	✓	✓	✓	✓	✓



# Container and language-specific package registries



Docker  
registry

Software repository	Description	Available in GitLab version
Container Registry	The GitLab Container Registry enables every project in GitLab to have its own space to store Docker <a href="#">images</a> .	8.8+
Dependency Proxy <small>i</small>	The GitLab Dependency Proxy sets up a local proxy for frequently used upstream images/packages.	11.11+
Conan Repository <small>i</small>	The GitLab Conan Repository enables every project in GitLab to have its own space to store Conan <a href="#">packages</a> .	12.6+
Maven Repository <small>i</small>	The GitLab Maven Repository enables every project in GitLab to have its own space to store Maven <a href="#">packages</a> .	11.3+
NPM Registry <small>i</small>	The GitLab NPM Registry enables every project in GitLab to have its own space to store NPM <a href="#">packages</a> .	11.7+
NuGet Repository <small>i</small>	The GitLab NuGet Repository will enable every project in GitLab to have its own space to store NuGet <a href="#">packages</a> .	12.8+
PyPi Repository <small>i</small>	The GitLab PyPi Repository will enable every project in GitLab to have its own space to store PyPi <a href="#">packages</a> .	12.10+
Go Proxy <small>i</small>	The Go proxy for GitLab enables every project in GitLab to be fetched with the Go proxy protocol <a href="#">.</a>	13.1+
Composer Repository <small>i</small>	The GitLab Composer Repository will enable every project in GitLab to have its own space to store Composer <a href="#">packages</a> .	13.2+

Dependency  
Proxy

Language-specific  
package registries



# dependencies



```
build:osx:
  stage: build
  script: make build:osx
  artifacts:
    paths:
      - binaries/

build:linux:
  stage: build
  script: make build:linux
  artifacts:
    paths:
      - binaries/

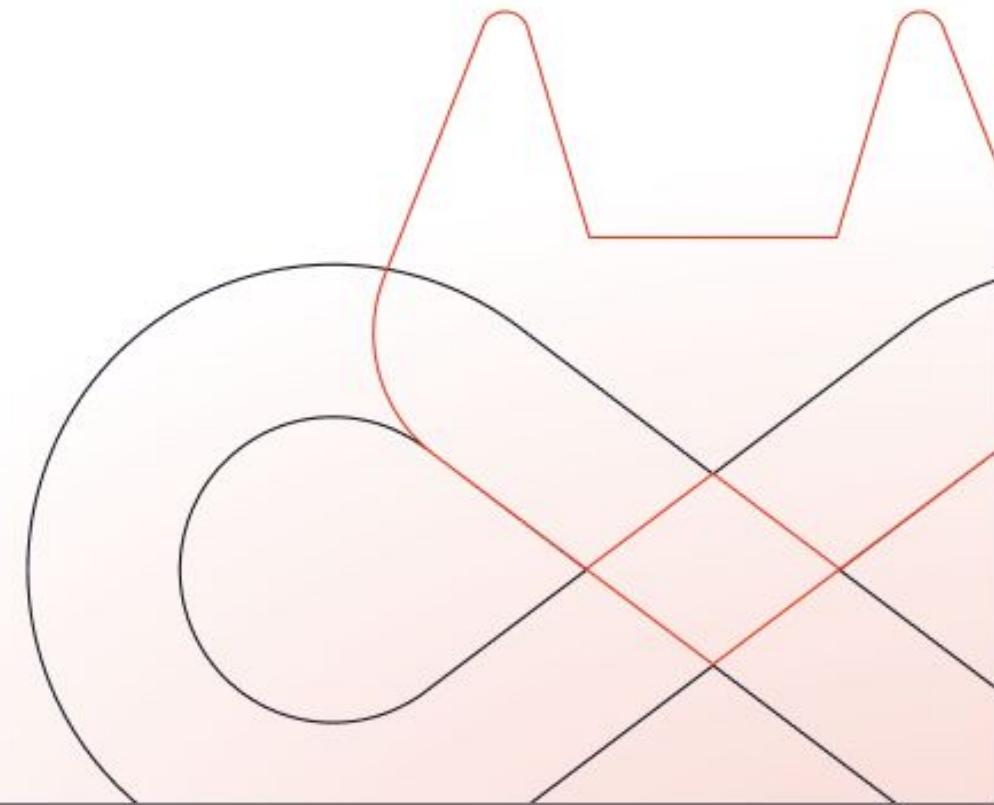
test:osx:
  stage: test
  script: make test:osx
  dependencies:
    - build:osx

test:linux:
  stage: test
  script: make test:linux
  dependencies:
    - build:linux

deploy:
  stage: deploy
  script: make deploy
```

- Reduce number of artifacts passed from previous jobs
- Improve job performance by preventing large artifacts to download for each job
- Empty array will skip downloading artifacts
- Only from jobs from previous stages

# Include and Extends



# Include



- Allows the inclusion of external YAML files
- Helps to break down the CI/CD configuration into multiple files and increases readability for long configuration files
- Possible to have template files stored in a central repository and projects include their configuration files
- Helps avoid duplicated configuration, for example, global default variables for all projects
- Can be nested (up to 100 includes)

```
include: '.gitlab-ci-production.yml'
```

```
include:  
  - local: '/templates/.gitlab-ci-template.yml'
```

```
include:  
  - project: 'my-group/my-project'  
    file: '/templates/.gitlab-ci-template.yml'
```

```
include:  
  - remote: 'https://gitlab.com/awesome-project/raw/master/.gitlab-ci-template.yml'
```

```
include:  
  - template: Android-Fastlane.gitlab-ci.yml  
  - template: Auto-DevOps.gitlab-ci.yml
```

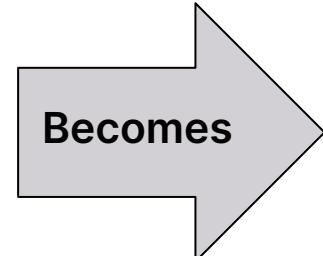
Method	Description
local	Include a file from the local project repository.
file	Include a file from a different project repository.
remote	Include a file from a remote URL. Must be publicly accessible.
template	Include templates which are provided by GitLab.

# Extends



- Defines entry names that a job uses extends is going to inherit from
- Alternative to YML anchors, and is more flexible and readable
- Performs a merge
- Supports multilevel inheritance (but more than 3 is not recommended)
- Can merge hidden jobs (.tests) or from regular jobs

```
.tests:  
  script: rake test  
  stage: test  
  only:  
    refs:  
      - branches  
  
rspec:  
  extends: .tests  
  script: rake rspec  
  only:  
    variables:  
      - $RSPEC
```



```
rspec:  
  script: rake rspec  
  stage: test  
  only:  
    refs:  
      - branches  
    variables:  
      - $RSPEC
```



# Using Includes and Extends together



- Extends works across configuration files with include

If you have a local included.yml:

```
.template:  
  script:  
    - echo Hello!
```

You can do the following:

```
include: included.yml  
  
useTemplate:  
  image: alpine  
  extends: .template
```

This will run a job called `useTemplate` that runs `echo Hello!` as defined in the `.template` job, and uses the `alpine` Docker image as defined in the local job.



# Q&A

