**Introduction**

Automating the execution, testing and deployment of R work is a very powerful tool to ensure the reproducibility, quality and overall robustness of the code that we are building, be it for data analysis and modeling purposes, developing R packages or even blogging. Modern tools also provide a free an easy to use way of achieving this goal.

In this post, we will show a quick and simple way to automate R data analysis and package development checking, testing and installation with GitLab CI/CD and provide example files that can be used for testing packages and deploying blogdown-based websites.

**Contents**

1. [A quick overview of CI/CD and GitLab’s approach to it](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/#a-quick-overview-of-cicd-and-gitlabs-approach-to-it)
2. [The simplest example with R use](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/#the-simplest-example-with-r-use)
3. [Example pipeline for R package testing and deployment](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/#example-pipeline-for-r-package-testing-and-deployment)
4. [Docker images for R users and developers](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/#docker-images-for-r-users-and-developers)
5. [TL;DR: Just show it to me in action](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/#tldr-just-show-it-to-me-in-action)
6. [References](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/#references)

**A quick overview of CI/CD and GitLab’s approach to it**

In this paragraph, we will try to introduce GitLab CI/CD and it prerequisites in very *simple and practical terms, at the cost of technical precision*. Terms will be linked to relevant pages for those interested in precise definitions. We will also focus on using the CI/CD provided directly on GitLab. It is also possible to use it on your own infrastructure, but this is out of the scope of this introductory post.

**What is CI/CD and how to use it with Gitlab**

[Continuous integration (CI)](https://en.wikipedia.org/wiki/Continuous_integration) and [Continuous deployment (CD)](https://en.wikipedia.org/wiki/Continuous_deployment) are IT practices that encourage checking and testing code often (e.g. on every change pushed to a repository) and being able to provide the resulting product (e.g. an application) to the users automatically.

For the purpose of this post, we will be focusing on R code and will be happy with the CI/CD

* detecting any code changes that we make in our repository
* automatically running a set of actions that we define when changes are detected

**What is GitLab CI/CD, what can it do**

* [GitLab CI/CD](https://about.gitlab.com/product/continuous-integration/) is a service provided by GitLab that makes using basic CI/CD easy even for non-IT professionals, such as R users
* Free for both public and private repositories hosted on GitLab
* Can execute a [wide variety of tasks](https://docs.gitlab.com/ee/ci/examples/), ranging from executing custom scripts, deployment of Java applications, building Docker images, checking and testing R packages to publishing blogs and more

**What are the prerequisites, how to make it work?**

* To use GitLab CI/CD your project’s code should be hosted on [GitLab](https://gitlab.com/)
* To make it work, you need to create a [yaml file](https://jozefhajnala.gitlab.io/r/r106-r-package-gitlab-ci/) called .gitlab-ci.yml with the instructions and push it to the root directory of your project’s repository
* Once you do that, instructions in that yaml file will be executed by GitLab automatically each time you push a code change to the repository. Different triggers such as specified times, etc. can also be used

**Why GitLab? What if my code is on GitHub?**

This post is by no means supposed to be an advertisement for GitLab, I chose it some time ago for 2 very simple reasons

1. It allowed for free private repositories, which is now also true for GitHub
2. The CI/CD is fully integrated, with no need for other tools

**The simplest example with R use**

To make the post a bit less abstract and more practical, here is an overly simplified example of GitLab CI/CD used with R, which just runs the current version of R and prints the mtcars dataset:

* The [repository](https://gitlab.com/jozefhajnala/simplestrci)
* The [.gitlab-ci.yml](https://gitlab.com/jozefhajnala/simplestrci/blob/master/.gitlab-ci.yml) file
* An overview of [pipeline runs](https://gitlab.com/jozefhajnala/simplestrci/pipelines)
* An example [output of a run](https://gitlab.com/jozefhajnala/simplestrci/-/jobs/194646370)

Let’s have a look at this very simplistic .gitlab-ci.yml:

image: r-base

test:

script:

- R -e 'print(datasets::mtcars)'

We see the following:

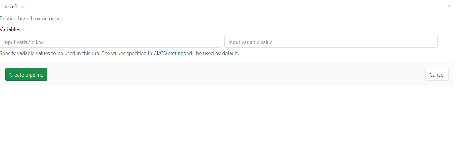
* image: r-base tells GitLab CI/CD to use the r-base Docker image for the run – more on that later
* the rest of the yaml tells GitLab CI/CD to run a job named test, its task is to execute a script defined as R -e 'print(datasets::mtcars)', meaning just to run R and print the mtcars dataset

Now let us take a look at a more useful example for developing R packages.

**Example pipeline for R package testing and deployment**

We can use GitLab CI/CD to automatically

1. Build our package
2. Perform the R CMD check and investigate if we have any errors, warnings or notes
3. Run our unit tests
4. Check our testing coverage and finally
5. Install the package and potentially use it to perform more actions



GitLab CI/CD Pipeline for an R package

An example .gitlab-ci.yml with a pipeline based on a Docker image to test an R package can look as follows. Note that this is most likely overkill and too spacious, one could have a pipeline that is way shorter for this purpose:

image: jozefhajnala/rdev:3.4.4

stages:

- build

- document

- check

- test

- deploy

variables:

\_R\_CHECK\_CRAN\_INCOMING\_: "false"

\_R\_CHECK\_FORCE\_SUGGESTS\_: "true"

CODECOV\_TOKEN: "2329aed3-de38-468c-9a06-95564363211c"

before\_script:

- apt-get update

buildbinary:

stage: build

script:

- r -e 'devtools::build(binary = TRUE)'

documentation:

stage: document

script:

- r inst/ci/document.R

checkerrors:

stage: check

script:

- r -e 'if (!identical(devtools::check(document = FALSE, args = "--no-tests")[["errors"]], character(0))) stop("Check with Errors")'

checkwarnings:

stage: check

script:

- r -e 'if (!identical(devtools::check(document = FALSE, args = "--no-tests")[["warnings"]], character(0))) stop("Check with Warnings")'

checknotes:

stage: check

script:

- r -e 'if (!identical(devtools::check(document = FALSE, args = "--no-tests")[["notes"]], character(0))) stop("Check with Notes")'

unittests:

stage: test

script:

- r -e 'if (any(as.data.frame(devtools::test())[["failed"]] > 0)) stop("Some tests failed.")'

codecov:

stage: test

script:

- r -e 'covr::codecov()'

install:

stage: deploy

script:

- r -e 'devtools::install()'

Now let’s again take a look at the content:

* image – a docker image to use for the pipeline
* stages – defines the ordering of job execution, jobs of the same stage are run in parallel, jobs of the next stage are run after the jobs from the previous stage complete successfully. Stages are the “columns” of the chart below.
* variables – used to pass environment variables to the jobs
* before\_script – used to define commands that should be run before all jobs. An example is installing a needed R package that is not contained in our Docker image.

The rest are jobs definitions. Each job has a stage, which defines in which stage it is ran, where multiple jobs can be included in one stage. A script essentially defines what to do. For R uses, this can usually be:

* Rscript -e '' to execute R commands specified between the quotes
* Rscript pathtoscript.R to execute a script stored in a file
* for [littler](http://dirk.eddelbuettel.com/code/littler.html) users, we can replace Rscript with r for similar purposes as above

**Docker images for R users and developers**

As we have seen, most of the pipelines start with image: . This tells GitLab to use the specified Docker image for the run, which is extremely useful because a suitable Docker image will include all the software that we need to execute our analyses, modeling or other tasks, without us having to install that software within the .yaml file. Example of such software available in an image for R use is, obviously, R and other dependencies such as additional packages.

One of the great things about Docker images is that they are easy to share and adapt.

My personal favorites from the Rocker project are

* [r-ver](https://hub.docker.com/r/rocker/r-ver) images – providing an environment fixed in time, including using a specifically dated MRAN repository. The image used by my CI pipeline for testing packages is [adapted from r-ver:3.4.4](https://gitlab.com/jozefhajnala/dockerfiles/blob/master/r3.4.4/Dockerfile).
* [r-base](https://hub.docker.com/_/r-base) for the current version of base R.

**TL;DR: Just show it to me in action**

In case you are only interested in seeing the CI/CD pipeline work in action for some R uses, you can look at:

**The simplest example using R**

* The [repository](https://gitlab.com/jozefhajnala/simplestrci)
* The [.gitlab-ci.yml](https://gitlab.com/jozefhajnala/simplestrci/blob/master/.gitlab-ci.yml) file
* An overview of the [pipeline runs](https://gitlab.com/jozefhajnala/simplestrci/pipelines)
* An example [output of a run](https://gitlab.com/jozefhajnala/simplestrci/-/jobs/194646370)

**The mentioned R package testing**

* The [.gitlab-ci.yml](https://gitlab.com/jozefhajnala/jhaddins/blob/develop/.gitlab-ci.yml) file for the jhaddins package on branch develop
* An overview of the [pipeline runs](https://gitlab.com/jozefhajnala/jhaddins/pipelines)
* An example [output of a successful run](https://gitlab.com/jozefhajnala/jhaddins/pipelines/55954699)
* An example output of a run that [discovered a NOTE](https://gitlab.com/jozefhajnala/jhaddins/-/jobs/139893165) in the check process

**Building a Docker image**

The Docker image jozefhajnala/rdev:3.4.4 used above

* is based on the following [Dockerfile](https://gitlab.com/jozefhajnala/dockerfiles/blob/master/r3.4.4/Dockerfile)
* is also built with GitLab CI/CD, look at the [.gitlab-ci.yml file](https://gitlab.com/jozefhajnala/dockerfiles/blob/master/.gitlab-ci.yml)
* the build [pipeline in action](https://gitlab.com/jozefhajnala/dockerfiles/-/jobs/194667135)

**Publishing a Hugo-based blogdown blog**

Also, this blog itself is deployed on a schedule via GitLab CI/CD, using a file very similar to the following:

* [Example Hugo site](https://pages.gitlab.io/hugo/) deployed with GitLab CI/CD
* The [.gitlab-ci.yml](https://gitlab.com/pages/hugo/blob/master/.gitlab-ci.yml)

**References**

* GitLab [Continuous Integration documentation](https://docs.gitlab.com/ee/ci/README.html)
* GitLab CI/CD [Pipeline Configuration Reference](https://docs.gitlab.com/ee/ci/yaml/README.html)
* [Building an R Project](https://docs.travis-ci.com/user/languages/r) with Travis CI
* [Docker images for R](https://www.rocker-project.org/images/) on the Rocker Project