**Introduction**

In the [previous post](https://jozef.io/r106-r-package-gitlab-ci/), we looked at how to easily automate R analysis, modeling, and development work for free using GitLab’s CI/CD. Together with the fantastic [R-hub project](https://builder.r-hub.io/), we can use GitLab CI/CD to do much more.

In this post, we will take it to the next level by using R-hub to test our development work on many different platforms such as multiple Linux setups, MS Windows and MacOS. We will also show how to automate and continuously execute those multiplatform checks using GitLab CI/CD integration and Docker images.

For those too busy to read, we also provide a working example implementation in a public GitLab repository.

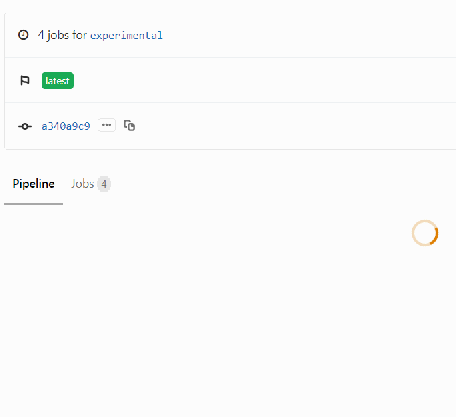
**Contents**

1. [Using R-hub to build, check and test our package on many platforms](https://jozef.io/r107-multiplatform-gitlabci-rhub/#using-r-hub-to-build-check-and-test-our-r-package-on-many-platforms)
2. [Using and evaluating R-hub check results via R scripts](https://jozef.io/r107-multiplatform-gitlabci-rhub/#using-and-evaluating-r-hub-check-results-via-r-scripts)
3. [Preparing a private docker image to use with R-hub](https://jozef.io/r107-multiplatform-gitlabci-rhub/#preparing-a-private-docker-image-to-use-with-r-hub)
4. [Creating a GitLab CI/CD pipeline](https://jozef.io/r107-multiplatform-gitlabci-rhub/#creating-a-gitlab-cicd-pipeline)
5. [TL;DR: Just show it to me in action](https://jozef.io/r107-multiplatform-gitlabci-rhub/#tldr-just-show-it-to-me-in-action)
6. [References](https://jozef.io/r107-multiplatform-gitlabci-rhub/#references)

**Using R-hub to build, check and test our R package on many platforms**

[R-hub](https://builder.r-hub.io/about.html) is a project supported by the [R Consortium](https://www.r-consortium.org/) and offers free R CMD check as a service on different platforms. This enables us to quickly and efficiently check the R package you are developing to make sure it passes all necessary checks on several platforms. As an added bonus, the checks seem to be running in a very short time span, which means we can have your results at hand in a few minutes.

I also recommend that you read the [why should you care about R-hub?](https://blog.r-hub.io/2019/03/26/why-care/) blog post for more info.



CI/CD running checks on multiple platforms with R-hub

**Getting started with R-hub**

Getting started with R-hub is also very simple and can be achieved in 3 lines of code, from a package directory or an RStudio project for a package:

# Install the package

install.packages("rhub")

# Validate your e-mail address

# Provide the email argument if not detected automatically

rhub::validate\_email()

# In an interactive session,

# this will offer a list of platforms to choose from

cr <- rhub::check()

Your validated\_emails.csv should be saved into rappdirs::user\_data\_dir("rhub", "rhub") directory once validate\_email() was run successfully.

For more details on getting started, the [Get started with rhub](https://r-hub.github.io/rhub/articles/rhub.html) post has you covered in detail.

**Using and evaluating R-hub check results via R scripts**

For continuous integration purposes, we may want to evaluate the results of the check based on the number of errors, warnings, and notes that the check gives for each platform. To achieve this goal, we need to tackle 2 issues:

**Getting the results in a non-interactive context**

In a non-interactive session, R-hub will run the check asynchronously and end our process used to request the service to free up resources. This is great but can pose some challenges in the CI context, as we would have to keep around a job to repeatedly query the R-hub job’s status and processing the results once done. Or implement a much smarter reporting solution.

Luckily, since for this purpose maximizing efficiency is not our top concern, the simple workaround is to execute the check as-if in an interactive session via the CI tool. This will provide us with the actual results of the check as soon as done and also write the log into our CI’s run log, at the obvious cost of having the process blocked while waiting for the check to finish on R-hub’s servers.

**Processing the check results**

The public methods for an rhub\_check object currently seem to provide only side-effecting results such as printing them in various levels of detail and returning self, so investigating results via code may be challenging.

The simplest current solution is to use the object’s private fields to access the results in the desired format. The below example looks at the status\_ private field and returns a data frame with the number of errors, warnings, and notes for each. For an object containing only 1 check result it can look as follows:

statuses <- cr[[".\_\_enclos\_env\_\_"]][["private"]][["status\_"]]

res <- do.call(rbind, lapply(statuses, function(thisStatus) {

data.frame(

plaform = thisStatus[["platform"]][["name"]],

errors = length(thisStatus[["result"]][["errors"]]),

warnings = length(thisStatus[["result"]][["warnings"]]),

notes = length(thisStatus[["result"]][["notes"]]),

stringsAsFactors = FALSE

)

}))

res

## plaform errors warnings notes

## 1 debian-gcc-release 0 0 0

Now we have a data frame which we can use to signal the CI/CD job to succeed or fail based on our wishes. For example, if we want to fail if the check discovered any notes, warnings or errors, a simple statement like the following will suffice:

if (any(colSums(res[2L:4L]) > 0)) {

stop("Some checks resulted in errors, warnings or notes.")

}

**Putting it together into a script**

Now that we have solved the above challenges, we can put it all together into a script that can be later used in the context of a CI/CD job:

# Retrieve passed command line arguments

args <- commandArgs(trailingOnly = TRUE)

if (length(args) != 1L) {

stop("Incorrect number of args, needs 1: platform (string)")

}

platform <- args[[1L]]

# Check if passed platform is valid

if (!is.element(platform, rhub::platforms()[[1L]])) {

stop(paste(

"Given platform not in rhub::platforms()[[1L]]:",

platform

))

}

# Run the check on the selected platform

# Use show\_status = TRUE to wait for results

cr <- rhub::check(platform = platform, show\_status = TRUE)

# Get the statuses from private field status\_

statuses <- cr[[".\_\_enclos\_env\_\_"]][["private"]][["status\_"]]

# Create and print a data frame with results

res <- do.call(rbind, lapply(statuses, function(thisStatus) {

data.frame(

plaform = thisStatus[["platform"]][["name"]],

errors = length(thisStatus[["result"]][["errors"]]),

warnings = length(thisStatus[["result"]][["warnings"]]),

notes = length(thisStatus[["result"]][["notes"]]),

stringsAsFactors = FALSE

)

}))

print(res)

# Fail if any errors, warnings or notes found

if (any(colSums(res[2L:4L]) > 0)) {

stop("Some checks had errors, warnings or notes. See above for details.")

}

**Preparing a private docker image to use with R-hub**

If you are new to Docker, Colin Fay has you covered with his [Introduction to Docker for R Users](https://colinfay.me/docker-r-reproducibility/) blog post.

**Creating and testing an image**

Thanks to all the hard work done by the maintainers of the [Rocker images](https://www.rocker-project.org/images/), our task with creating an image suitable for use with R hub is very simple. Essentially we only need 2 additions to the [r-base image](https://hub.docker.com/_/r-base):

1. The rhub package and a few system dependencies
2. A validated\_emails.csv file placed into the correct directory, providing R-hub with the information on validated e-mail to use for the checks

The [following Dockerfile](https://gitlab.com/jozefhajnala/dockerfiles/blob/master/rhub/Dockerfile) can be used the create such an image for yourself. Just make sure you have your validated\_emails.csv file present in the resources folder when running docker build.

To test our docker image, we can use a command like the following to create a container and run R within it in an interactive session:

docker run --rm -it /: R

Now we can see the list of validated e-mails in that R session:

rhub::list\_validated\_emails()

## email token

## 1 myemail@somemail.com 00000000000000000000

**Pushing the image into a private repository**

Now that we have our image created, we need to push it to a repository for GitLab CI to be able to use it. Normally this is very simple:

docker push /:

However as we are storing some relatively sensitive data in our image, namely our R-hub token we should probably make this image private. Thanks to Dockerhub, this process is very easy – just click the proper buttons as shown in [this post in the Dockerhub docs](https://docs.docker.com/docker-hub/repos/#private-repositories). Note that for free a Dockerhub user has only 1 private repository available.

**Creating a GitLab CI/CD pipeline**

For an introduction to using GitLab CI/CD for R work, look at the previous post on [How to easily automate R analysis, modeling and development work using CI/CD, with working examples](https://jozef.io/r106-r-package-gitlab-ci/)

**Setting up a pipeline with .gitlab-ci.yml**

Now, we are ready with our private Docker image and the script to run and evaluate our R-hub checks, all that is left is to create and setup a CI/CD pipeline. For GitLab CI/CD, this means creating a .gitlab-ci.yml file in the root of our GitLab repository directory. Without much extra talk, that file can look as follows:

image: index.docker.io/jozefhajnala/rhub:rbase

stages:

- check

variables:

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

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

before\_script:

- apt-get update

check\_ubuntu:

stage: check

script:

- Rscript inst/rhubcheck.R "ubuntu-gcc-release"

check\_fedora:

stage: check

script:

- Rscript inst/rhubcheck.R "fedora-clang-devel"

check\_mswin:

stage: check

script:

- Rscript inst/rhubcheck.R "windows-x86\_64-devel"

check\_macos:

stage: check

script:

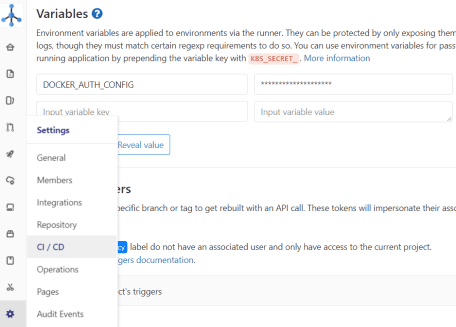
- Rscript inst/rhubcheck.R "macos-elcapitan-release"

This file will make sure that:

1. The CI/CD jobs start from the image we have created
2. Will have one stage named check
3. Set a couple of environment variables for R
4. Run three jobs check\_ubuntu, check\_fedora, check\_mswin, and check\_macos – each of them by using Rscript to execute an R script stored under inst/rhubcheck.R, with different arguments specifying the platform to check on

**Authenticating to use a private repository**

Since we have made our Docker image private, GitLab will not be able to use it out of the box, we need to provide it with information on how to authenticate against Dockerhub to be able to pull the private image. There are a [few ways to reach this](https://docs.gitlab.com/ee/ci/docker/using_docker_images.html#define-an-image-from-a-private-container-registry) goal, I have used the one to setup a variable via the Settings -> CI/CD -> Variables option in GitLab’s web UI:



Creating CI/CD variable with GitLab

The variable name should be DOCKER\_AUTH\_CONFIG and the value:

{

"auths": {

"registry.example.com:5000": {

"auth": "bXlfdXNlcm5hbWU6bXlfcGFzc3dvcmQ="

}

}

}

Where

* "registry.example.com:5000" is replaced by our registry, for example "index.docker.io"
* the value for "auth" is replaced by a base64-encoded version of our ":", which we can retrieve for example using R:

base64enc::base64encode(charToRaw("my\_username:my\_password"))

## [1] "bXlfdXNlcm5hbWU6bXlfcGFzc3dvcmQ="

And that is all! We are now ready to run our checks using a Docker image stored in a private repository. Once we push the .gitlab-ci.yml and inst/rhubcheck.R files to a GitLab repository, the pipeline will be automatically executed every time we push a commit to that repository.

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

In case you are only interested in seeing the CI/CD pipeline with R-hub implemented for an R package, look at:

* The [.gitlab-ci.yml](https://gitlab.com/jozefhajnala/jhaddins/blob/experimental/.gitlab-ci.yml) file for the jhaddins package on branch experimental
* The [Dockerfile](https://gitlab.com/jozefhajnala/dockerfiles/blob/master/rhub/Dockerfile) used to build the image used in the above .gitlab-ci.yml
* An [R script that runs the checks](https://gitlab.com/jozefhajnala/jhaddins/blob/experimental/inst/rhubcheck.R) via R-hub and evaluates the results
* An example [of a successful run](https://gitlab.com/jozefhajnala/jhaddins/pipelines/57477509) with checks on 3 platforms
* An example output of a [check on Windows](https://gitlab.com/jozefhajnala/jhaddins/-/jobs/199152548) provided by R-hub

**References**

**R-hub**

* The [why should you care about R-hub?](https://blog.r-hub.io/2019/03/26/why-care/) blog post
* Get [started with R-Hub](https://r-hub.github.io/rhub/articles/rhub.html)
* R-Hub on the [R Consortium website](https://www.r-consortium.org/projects/r-hub)
* R-Hub’s [reference online](https://r-hub.github.io/rhub/reference/index.html)
* Documentation on [rhub\_check R6 objects](https://r-hub.github.io/rhub/reference/rhub_check.html)

**R work and GitLab**

* Blog post on [automating R analysis, modeling and development using CI/CD, with working examples](https://jozef.io/r106-r-package-gitlab-ci/)
* GitLab [Continuous Integration documentation](https://docs.gitlab.com/ee/ci/README.html)
* GitLab CI/CD [environment variables](https://docs.gitlab.com/ee/ci/variables/)
* Using a [private container registry](https://docs.gitlab.com/ee/ci/docker/using_docker_images.html#define-an-image-from-a-private-container-registry) with GitLab CI/CD

**R work and Docker**

* [Docker images for R](https://www.rocker-project.org/images/) on the Rocker Project
* Colin Fay’s [Introduction to Docker for R Users](https://colinfay.me/docker-r-reproducibility/)
* [Get started with Docker](https://docs.docker.com/get-started/) official documentation
* Using [private repositories](https://docs.docker.com/docker-hub/repos/#private-repositories) in DockerHub