Package 'jetty'

January 9, 2025

Type Package

Title Execute R in a 'Docker' Context

Version 0.1.0

Description The goal of 'jetty' is to execute R functions and code snippets in an isolated R subprocess within a 'Docker' container and return the evaluated results to the local R session. 'jetty' can install necessary packages at runtime and seamlessly propagates errors and outputs from the 'Docker' subprocess back to the main session. 'jetty' is primarily designed for sandboxed testing and quick execution of example code.

License GPL (>= 3)

Encoding UTF-8

Imports renv (>= 1.0.0), rlang

Suggests here, Matrix, ggplot2, testthat (>= 3.0.0)

RoxygenNote 7.3.2

Config/testthat/edition 3

URL https://github.com/dmolitor/jetty, http://www.dmolitor.com/jetty/

BugReports https://github.com/dmolitor/jetty/issues

NeedsCompilation no

Author Daniel Molitor [aut, cph, cre]

Maintainer Daniel Molitor <molitdj97@gmail.com>

Repository CRAN

Date/Publication 2025-01-09 14:50:02 UTC

Contents

run *Execute an R expression inside a Docker container*

Description

This function is somewhat similar in spirit to `callr::r()` in that the user can pass a function (or a code block) to be evaluated. This code will be executed within the context of a Docker container and the result will be returned within the local R session.

Usage

```
run(
  func,
  args = list(),
  image = paste0("r-base:", r_version()),
  stdout = "",
  stderr = "",
  install_dependencies = FALSE,
  r_profile = file.path(getwd(), ".Rprofile"),
  r_environ = file.path(getwd(), ".Renviron"),
  debug = FALSE
)
```

Arguments

`func`	Function object or code block to be executed in the R session within the Docker container. It is best to reference package functions using the `::` notation, and only packages installed in the Docker container are accessible.
`args`	Arguments to pass to the function. Must be a list.
`image`	A string in the `image:tag` format specifying either a local Docker image or an image available on DockerHub. Default image is `r-base:{jetty:::r_version()}` where your R version is determined from your local R session.
`stdout, stderr`	Where output to 'stdout' or 'stderr' should be sent. Possible values are "" (send to the R console; the default), NULL or FALSE (discard output), TRUE (capture the output in a character vector) or a character string naming a file. See `system2` for more details.
`install_dependencies`	A boolean indicating whether jetty should discover packages used in your code and try to install them in the Docker container prior to executing the provided function/expression. In general, things will work better if the Docker image already has all necessary packages installed.
`r_profile, r_environ`	Paths specifying where jetty should search for the .Rprofile and .Renviron files to transfer to the Docker sub-process. By default jetty will look for files called ".Rprofile" and ".Renviron" in the current working directory. If either file is found, they will be transferred to the Docker sub-process and loaded before executing any R commands.

debug — A boolean indicating whether to print out the commands that are being executed via the shell. This is mostly helpful to see what is happening when things start to error.

Value

Value of the evaluated expression.

Side effects

It is important to note that some side effects, e.g. plotting, may be lost since these are being screamed into the void of the Docker container. However, the user has full control over the 'stderr' and 'stdout' of the R sub-process running in the Docker container, and so side effects such as messages, warnings, and printed output can be directed to the console or captured by the user.

It is also crucial to note that actions on the local file system will not work with jetty sub-processes. For example, reading or writing files to/from the local file system will fail since the R sub-process within the Docker container does not have access to the local file system.

Error handling

jetty will propagate errors from the Docker process to the main process. That is, if an error is thrown in the Docker process, an error with the same message will be thrown in the main process. However, because of the somewhat isolated nature of the local process and the Docker process, calling functions such as `base::traceback()` and `rlang::last_trace()` will not print the callstack of the uncaught error as that has (in its current form) been lost in the Docker void.

Examples

```
## Not run:
run(
  {
    mtcars <- mtcars |>
      transform(cyl = as.factor(cyl))
    model <- lm(mpg ~ ., data = mtcars)
    model
  }
)

# A code snippet that requires packages to be installed
run(
  {
    mtcars <- mtcars |>
      dplyr::mutate(cyl = as.factor(cyl))
    model <- lm(mpg ~ ., data = mtcars)
    model
  },
  install_dependencies = TRUE
)

# This will error since the `praise` package is not installed
run(function() praise::praise())
```

```
## End(Not run)
```

Index