Package 'dockerfiler'

July 9, 2024

Title Easy Dockerfile Creation from R

Version 0.2.3

Description Build a Dockerfile straight from your R session.
'dockerfiler' allows you to create step by step a Dockerfile, and
provide convenient tools to wrap R code inside this Dockerfile.

License MIT + file LICENSE

URL https://github.com/ThinkR-open/dockerfiler

BugReports https://github.com/ThinkR-open/dockerfiler/issues

Imports attempt (>= 0.3.1), cli (>= 2.3.0), desc (>= 1.2.0), fs (>=
1.5.0), glue (>= 1.4.2), jsonlite (>= 1.7.2), memoise, pak (>=
0.6.0), pkgbuild (>= 1.2.0), purrr, R6 (>= 2.5.0), remotes (>=
2.2.0), usethis (>= 2.0.1), utils

Suggests knitr (>= 1.31), rmarkdown (>= 2.6), testthat (>= 3.0.0),
withr

VignetteBuilder knitr

Config/fusen/version 0.6.0

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation no

Author Colin Fay [cre, aut] (<https://orcid.org/0000-0001-7343-1846>),
Vincent Guyader [aut] (<https://orcid.org/0000-0003-0671-9270>),
Josiah Parry [aut] (<https://orcid.org/0000-0001-9910-865X>),
Sébastien Rochette [aut] (<https://orcid.org/0000-0002-1565-9313>)

Maintainer Colin Fay <contact@colinfay.me>

Repository CRAN

Date/Publication 2024-07-09 08:00:02 UTC

Contents

`compact_sysreqs` *Compact Sysreqs*

Description

Compact Sysreqs

Usage

```
compact_sysreqs(
  pkg_installs,
  update_cmd = "apt-get update -y",
  install_cmd = "apt-get install -y",
  clean_cmd = "rm -rf /var/lib/apt/lists/*"
)
```

Arguments

`pkg_installs`	pkg_sysreqs as vector, `pak::pkg_sysreqs` output
`update_cmd`	command used to update packages, "apt-get update -y" by default
`install_cmd`	command used to install packages, "apt-get install -y" by default
`clean_cmd`	command used to clean package folder, "rm -rf /var/lib/apt/lists/*" by default

Value

vector of compacted command to run to install sysreqs

Examples

```
pkg_installs <- list("apt-get install -y htop", "apt-get install -y top")
compact_sysreqs(pkg_installs)
```

Dockerfile *A Dockerfile template*

Description

A Dockerfile template

A Dockerfile template

Public fields

`Dockerfile` The dockerfile content.

Methods

Public methods:

- `Dockerfile$new()`
- `Dockerfile$RUN()`
- `Dockerfile$ADD()`
- `Dockerfile$COPY()`
- `Dockerfile$WORKDIR()`
- `Dockerfile$EXPOSE()`
- `Dockerfile$VOLUME()`
- `Dockerfile$CMD()`
- `Dockerfile$LABEL()`
- `Dockerfile$ENV()`
- `Dockerfile$ENTRYPOINT()`
- `Dockerfile$USER()`
- `Dockerfile$ARG()`
- `Dockerfile$ONBUILD()`
- `Dockerfile$STOPSIGNAL()`
- `Dockerfile$HEALTHCHECK()`
- `Dockerfile$SHELL()`
- `Dockerfile$MAINTAINER()`
- `Dockerfile$custom()`
- `Dockerfile$print()`
- `Dockerfile$write()`
- `Dockerfile$switch_cmd()`
- `Dockerfile$remove_cmd()`
- `Dockerfile$add_after()`
- `Dockerfile$clone()`

Method `new()`**:** Create a new Dockerfile object.

Usage:

```
Dockerfile$new(FROM = "rocker/r-base", AS = NULL)
```

Arguments:

`FROM` The base image.

`AS` The name of the image.

Returns: A Dockerfile object.

Method `RUN()`**:** Add a RUN command.

Usage:

```
Dockerfile$RUN(cmd)
```

Arguments:

`cmd` The command to add.

Returns: the Dockerfile object, invisibly.

Method `ADD()`**:** Add a ADD command.

Usage:

```
Dockerfile$ADD(from, to, force = TRUE)
```

Arguments:

`from` The source file.

`to` The destination file.

`force` If TRUE, overwrite the destination file.

Returns: the Dockerfile object, invisibly.

Method `COPY()`**:** Add a COPY command.

Usage:

```
Dockerfile$COPY(from, to, force = TRUE)
```

Arguments:

`from` The source file.

`to` The destination file.

`force` If TRUE, overwrite the destination file.

Returns: the Dockerfile object, invisibly.

Method `WORKDIR()`**:** Add a WORKDIR command.

Usage:

```
Dockerfile$WORKDIR(where)
```

Arguments:

`where` The working directory.

Returns: the Dockerfile object, invisibly.

Method `EXPOSE()`**:** Add a EXPOSE command.

Usage:

```
Dockerfile$EXPOSE(port)
```

Arguments:

`port` The port to expose.

Returns: the Dockerfile object, invisibly.

Method `VOLUME()`**:** Add a VOLUME command.

Usage:

```
Dockerfile$VOLUME(volume)
```

Arguments:

`volume` The volume to add.

Returns: the Dockerfile object, invisibly.

Method `CMD()`**:** Add a CMD command.

Usage:

```
Dockerfile$CMD(cmd)
```

Arguments:

`cmd` The command to add.

Returns: the Dockerfile object, invisibly.

Method `LABEL()`**:** Add a LABEL command.

Usage:

```
Dockerfile$LABEL(key, value)
```

Arguments:

`key, value` The key and value of the label.

Returns: the Dockerfile object, invisibly.

Method `ENV()`**:** Add a ENV command.

Usage:

```
Dockerfile$ENV(key, value)
```

Arguments:

`key, value` The key and value of the label.

Returns: the Dockerfile object, invisibly.

Method `ENTRYPOINT()`**:** Add a ENTRYPOINT command.

Usage:

```
Dockerfile$ENTRYPOINT(cmd)
```

Arguments:

`cmd` The command to add.

Returns: the Dockerfile object, invisibly.

Method `USER()`**:** Add a USER command.

Usage:

```
Dockerfile$USER(user)
```

Arguments:

`user` The user to add.

Returns: the Dockerfile object, invisibly.

Method `ARG()`**:** Add a ARG command.

Usage:

```
Dockerfile$ARG(arg, ahead = FALSE)
```

Arguments:

`arg` The argument to add.

`ahead` If TRUE, add the argument at the beginning of the Dockerfile.

Returns: the Dockerfile object, invisibly.

Method `ONBUILD()`**:** Add a ONBUILD command.

Usage:

```
Dockerfile$ONBUILD(cmd)
```

Arguments:

`cmd` The command to add.

Returns: the Dockerfile object, invisibly.

Method `STOPSIGNAL()`**:** Add a STOPSIGNAL command.

Usage:

```
Dockerfile$STOPSIGNAL(signal)
```

Arguments:

`signal` The signal to add.

Returns: the Dockerfile object, invisibly.

Method `HEALTHCHECK()`**:** Add a HEALTHCHECK command.

Usage:

```
Dockerfile$HEALTHCHECK(check)
```

Arguments:

`check` The check to add.

Returns: the Dockerfile object, invisibly.

Method `SHELL()`**:** Add a SHELL command.

Usage:

```
Dockerfile$SHELL(shell)
```

Arguments:

`shell` The shell to add.

Returns: the Dockerfile object, invisibly.

Method `MAINTAINER()`**:** Add a MAINTAINER command.

Usage:

```
Dockerfile$MAINTAINER(name, email)
```

Arguments:

`name, email` The name and email of the maintainer.

Returns: the Dockerfile object, invisibly.

Method `custom()`**:** Add a custom command.

Usage:

```
Dockerfile$custom(base, cmd)
```

Arguments:

`base, cmd` The base and command to add.

Returns: the Dockerfile object, invisibly.

Method `print()`**:** Print the Dockerfile.

Usage:

```
Dockerfile$print()
```

Returns: used for side effect

Method `write()`**:** Write the Dockerfile to a file.

Usage:

```
Dockerfile$write(as = "Dockerfile")
```

Arguments:

`as` The file to write to.

Returns: used for side effect

Method `switch_cmd()`**:** Switch commands.

Usage:

```
Dockerfile$switch_cmd(a, b)
```

Arguments:

`a, b` The commands to switch.

Returns: the Dockerfile object, invisibly.

Method `remove_cmd()`**:** Remove a command.

Usage:

```
Dockerfile$remove_cmd(where)
```

Arguments:

`where` The commands to remove.

Returns: the Dockerfile object, invisibly.

Method `add_after()`**:** Add a command after another.

Usage:

```
Dockerfile$add_after(cmd, after)
```

Arguments:

`cmd` The command to add.

`after` Where to add the cmd

Returns: the Dockerfile object, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Dockerfile$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
my_dock <- Dockerfile$new()
```

`docker_ignore_add` *Create a dockerignore file*

Description

Create a dockerignore file

Usage

```
docker_ignore_add(path)
```

Arguments

`path`	Where to write the file

Value

The path to the `.dockerignore` file, invisibly.

Examples

```
## Not run:
  docker_ignore_add()

## End(Not run)
```

dock_from_desc *Create a Dockerfile from a DESCRIPTION*

Description

Create a Dockerfile from a DESCRIPTION

Usage

```
dock_from_desc(
  path = "DESCRIPTION",
  FROM = paste0("rocker/r-ver:", R.Version()$major, ".", R.Version()$minor),
  AS = NULL,
  sysreqs = TRUE,
  repos = c(CRAN = "https://cran.rstudio.com/"),
  expand = FALSE,
  update_tar_gz = TRUE,
  build_from_source = TRUE,
  extra_sysreqs = NULL
)
```

Arguments

path	path to the DESCRIPTION file to use as an input.
FROM	The FROM of the Dockerfile. Default is FROM rocker/r-ver:R.Version()$major.R.Version()$minor.
AS	The AS of the Dockerfile. Default it NULL.
sysreqs	boolean. If TRUE, the Dockerfile will contain sysreq installation.
repos	character. The URL(s) of the repositories to use for options("repos").
expand	boolean. If TRUE each system requirement will have its own RUN line.
update_tar_gz	boolean. If TRUE and build_from_source is also TRUE, an updated tar.gz is created.
build_from_source	boolean. If TRUE no tar.gz is created and the Dockerfile directly mount the source folder.
extra_sysreqs	character vector. Extra debian system requirements. Will be installed with apt-get install.

dock_from_renv *Create a Dockerfile from an* renv.lock *file*

Description

Create a Dockerfile from an renv.lock file

Usage

```
dock_from_renv(
  lockfile = "renv.lock",
  distro = NULL,
  FROM = "rocker/r-base",
  AS = NULL,
  sysreqs = TRUE,
  repos = c(CRAN = "https://cran.rstudio.com/"),
  expand = FALSE,
  extra_sysreqs = NULL,
  use_pak = FALSE,
  user = NULL,
  dependencies = NA,
  sysreqs_platform = "ubuntu",
  renv_version
)
```

Arguments

lockfile	Path to an renv.lock file to use as an input..
distro	• deprecated - only debian/ubuntu based images are supported
FROM	Docker image to start FROM Default is FROM rocker/r-base
AS	The AS of the Dockerfile. Default it NULL.
sysreqs	boolean. If TRUE, the Dockerfile will contain sysreq installation.
repos	character. The URL(s) of the repositories to use for options("repos").
expand	boolean. If TRUE each system requirement will have its own RUN line.
extra_sysreqs	character vector. Extra debian system requirements. Will be installed with apt-get install.
use_pak	boolean. If TRUE use pak to deal with dependencies during renv::restore(). FALSE by default
user	Name of the user to specify in the Dockerfile with the USER instruction. Default is NULL, in which case the user from the FROM image is used.
dependencies	What kinds of dependencies to install. Most commonly one of the following values: • NA: only required (hard) dependencies,

- TRUE: required dependencies plus optional and development dependencies,
- FALSE: do not install any dependencies. (You might end up with a non-working package, and/or the installation might fail.)

sysreqs_platform
: System requirements platform.ubuntu by default. If NULL, then the current platform is used. Can be : "ubuntu-22.04" if needed to fit with the FROM Operating System. Only debian or ubuntu based images are supported

renv_version
: character. The renv version to use in the generated Dockerfile. By default, it is set to the version specified in the renv.lock file. If the renv.lock file does not specify a renv version, the version of renv bundled with dockerfiler, specifically 1.0.3, will be used. If you set it to NULL, the latest available version of renv will be used.

Details

System requirements for packages are provided through RStudio Package Manager via the pak package. The install commands provided from pak are added as RUN directives within the Dockerfile.

The R version is taken from the renv.lock file. Packages are installed using renv::restore() which ensures that the proper package version and source is used when installed.

Value

A R6 object of class Dockerfile.

Examples

```
## Not run:
dock <- dock_from_renv("renv.lock", distro = "xenial")
dock$write("Dockerfile")

## End(Not run)
```

---

get_sysreqs *Get system requirements*

Description

This function retrieves information about the system requirements using the pak::pkg_sysreqs().

Usage

```
get_sysreqs(packages, quiet = TRUE, batch_n = 30)
```

Arguments

packages
: character vector. Packages names.

quiet
: Boolean. If TRUE the function is quiet.

batch_n
: numeric. Number of simultaneous packages to ask.

Value

A vector of system requirements.

---

parse_dockerfile *Parse a Dockerfile*

Description

Create a Dockerfile object from a Dockerfile.

Usage

```
parse_dockerfile(path)
```

Arguments

path	path to the Dockerfile

Value

A Dockerfile object

Examples

```
parse_dockerfile(system.file("Dockerfile", package = "dockerfiler"))
```

---

r *Turn an R call into an Unix call*

Description

Turn an R call into an Unix call

Usage

```
r(code)
```

Arguments

code	the function to call

Value

an unix R call

Examples

```
r(print("yeay"))
r(install.packages("plumber", repo = "http://cran.irsn.fr/"))
```

`renv` *Internalised renv*

Description

https://rstudio.github.io/renv/reference/vendor.html?q=vendor

Usage

```
renv
```

Format

An object of class `environment` of length 1446.

Index