Package 'maestro'

August 27, 2024

Type Package

Title Orchestration of Data Pipelines

Version 0.2.0

Maintainer Will Hipson <will.e.hipson@gmail.com>

Description Framework for creating and orchestrating data pipelines. Organize, orchestrate, and monitor multiple pipelines in a single project. Use tags to decorate functions with scheduling parameters and configuration.

License MIT + file LICENSE

URL https://github.com/whipson/maestro, https://whipson.github.io/maestro/

BugReports https://github.com/whipson/maestro/issues

Encoding UTF-8

LazyData true

Imports cli (>= 3.3.0), dplyr (>= 1.1.0), glue, logger, lubridate (>= 1.9.1), purrr (>= 1.0.0), R.utils, rlang (>= 1.0.0), roxygen2, tictoc, timechange, utils

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Suggests furrr, future, knitr, rmarkdown, rstudioapi, testthat (>= 3.0.0), withr

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Will Hipson [cre, aut, cph] (<https://orcid.org/0000-0002-3931-2189>), Ryan Garnett [aut, ctb, cph]

Repository CRAN

Date/Publication 2024-08-27 13:30:05 UTC

Contents

`build_schedule` *Build a schedule table*

Description

Builds a schedule data.frame for scheduling pipelines in `run_schedule()`.

Usage

```
build_schedule(pipeline_dir = "./pipelines", quiet = FALSE)
```

Arguments

`pipeline_dir`	path to directory containing the pipeline scripts
`quiet`	silence metrics to the console (default = `FALSE`)

Details

This function parses the maestro tags of functions located in `pipeline_dir` which is conventionally called 'pipelines'. An orchestrator requires a schedule table to determine which pipelines are to run and when. Each row in a schedule table is a pipeline name and its scheduling parameters such as its frequency.

The schedule table is mostly intended to be used by `run_schedule()` immediately. In other words, it is not recommended to make changes to it.

Value

data.frame

Examples

```
# Creating a temporary directory for demo purposes! In practice, just
# create a 'pipelines' directory at the project level.

if (interactive()) {
  pipeline_dir <- tempdir()
  create_pipeline("my_new_pipeline", pipeline_dir, open = FALSE)
  build_schedule(pipeline_dir = pipeline_dir)
}
```

`build_schedule_entry` *Create schedule table entry from a script*

Description

Create schedule table entry from a script

Usage

```
build_schedule_entry(script_path)
```

Arguments

`script_path`	path to script

Value

data.frame row

`check_pipelines` *Check which pipelines are scheduled to run and when next pipelines will run*

Description

Check which pipelines are scheduled to run and when next pipelines will run

Usage

```
check_pipelines(
  orch_unit,
  orch_n,
  pipeline_unit,
  pipeline_n,
  check_datetime,
  pipeline_datetime,
  pipeline_hours = 0:23,
  pipeline_days_of_week = 1:7,
  pipeline_days_of_month = 1:31,
  pipeline_months = 1:12
)
```

Arguments

`orch_unit`	unit of time for the orchestrator
`orch_n`	number of units for the orchestrator
`pipeline_unit`	unit for the pipeline frequency
`pipeline_n`	number of units for the pipeline frequency
`check_datetime`	datetime against which to check the running of pipelines (default is current system time in UTC)
`pipeline_datetime`	datetime of the first time the pipeline is to run
`pipeline_hours`	vector of integers [0-23] corresponding to hours of day for the pipeline to run
`pipeline_days_of_week`	vector of integers [1-7] corresponding to days of week for the pipeline to run (1 = Sunday)
`pipeline_days_of_month`	vector of integers [1-31] corresponding to days of month for the pipeline to run
`pipeline_months`	vector of integers [1-12] corresponding to months of year for the pipeline to run

Value

list

convert_to_seconds *Convert a duration string to number of seconds*

Description

Convert a duration string to number of seconds

Usage

```
convert_to_seconds(time_string)
```

Arguments

time_string	string like 1 day, 2 weeks, 12 hours, etc.

Value

number of seconds

create_maestro *Creates a new maestro project*

Description

Creates a new maestro project

Usage

```
create_maestro(path, type = "R", overwrite = FALSE, quiet = FALSE, ...)
```

Arguments

path	file path for the orchestrator script
type	file type for the orchestrator (supports R, Quarto, and RMarkdown)
overwrite	whether to overwrite an existing orchestrator or maestro project
quiet	whether to silence messages in the console (default = FALSE)
...	unused

Value

invisible

Examples

```
# Creates a new maestro project with an R orchestrator
if (interactive()) {
  new_proj_dir <- tempdir()
  create_maestro(new_proj_dir, type = "R", overwrite = TRUE)

  create_maestro(new_proj_dir, type = "Quarto", overwrite = TRUE)
}
```

`create_orchestrator` *Create a new orchestrator*

Description

Create a new orchestrator

Usage

```
create_orchestrator(
  path,
  type = c("R", "Quarto", "RMarkdown"),
  open = interactive(),
  quiet = FALSE,
  overwrite = FALSE
)
```

Arguments

`path`	file path for the orchestrator script
`type`	file type for the orchestrator (supports R, Quarto, and RMarkdown)
`open`	whether or not to open the script upon creation
`quiet`	whether to silence messages in the console (default = `FALSE`)
`overwrite`	whether to overwrite an existing orchestrator or maestro project

Value

invisible

create_pipeline *Create a new pipeline in a pipelines directory*

Description

Allows the creation of new pipelines (R scripts) and fills in the maestro tags as specified.

Usage

```
create_pipeline(
  pipe_name,
  pipeline_dir = "pipelines",
  frequency = "1 day",
  start_time = Sys.Date(),
  tz = "UTC",
  log_level = "INFO",
  quiet = FALSE,
  open = interactive(),
  overwrite = FALSE
)
```

Arguments

pipe_name	name of the pipeline and function
pipeline_dir	directory containing the pipeline scripts
frequency	how often the pipeline should run (e.g., 1 day, daily, 3 hours, 4 months). Fills in maestroFrequency tag
start_time	start time of the pipeline schedule. Fills in maestroStartTime tag
tz	timezone that pipeline will be scheduled in. Fills in maestroTz tag
log_level	log level for the pipeline (e.g., INFO, WARN, ERROR). Fills in maestroLogLevel tag
quiet	whether to silence messages in the console (default = FALSE)
open	whether or not to open the script upon creation
overwrite	whether or not to overwrite an existing pipeline of the same name and location.

Value

invisible

Examples

```
if (interactive()) {
  pipeline_dir <- tempdir()
  create_pipeline(
    "extract_data",
```

```
    pipeline_dir = pipeline_dir,
    frequency = "1 hour",
    open = FALSE,
    quiet = TRUE,
    overwrite = TRUE
  )

  create_pipeline(
    "new_job",
    pipeline_dir = pipeline_dir,
    frequency = "20 minutes",
    start_time = as.POSIXct("2024-06-21 12:20:00"),
    log_level = "ERROR",
    open = FALSE,
    quiet = TRUE,
    overwrite = TRUE
  )
}
```

`example_schedule` *Example schedule*

Description

An example of a properly formatted schedule from `build_schedule()`

`get_pipeline_run_sequence`
Generate a sequence of run times for a pipeline

Description

Generate a sequence of run times for a pipeline

Usage

```
get_pipeline_run_sequence(
  pipeline_n,
  pipeline_unit,
  pipeline_datetime,
  check_datetime,
  pipeline_hours = 0:23,
  pipeline_days_of_week = 1:7,
  pipeline_days_of_month = 1:31,
  pipeline_months = 1:12
)
```

Arguments

pipeline_n	number of units for the pipeline frequency
pipeline_unit	unit for the pipeline frequency
pipeline_datetime	datetime of the first time the pipeline is to run
check_datetime	datetime against which to check the running of pipelines (default is current system time in UTC)
pipeline_hours	vector of integers [0-23] corresponding to hours of day for the pipeline to run
pipeline_days_of_week	vector of integers [1-7] corresponding to days of week for the pipeline to run (1 = Sunday)
pipeline_days_of_month	vector of integers [1-31] corresponding to days of month for the pipeline to run
pipeline_months	vector of integers [1-12] corresponding to months of year for the pipeline to run

Value

vector of timestamps or dates

---

last_build_errors *Retrieve latest maestro build errors*

Description

Gets the latest schedule build errors following use of `build_schedule()`. If the build succeeded or `build_schedule()` has not been run it will be `NULL`.

Usage

```
last_build_errors()
```

Value

error messages

Examples

```
last_build_errors()
```

last_run_errors *Retrieve latest maestro pipeline errors*

Description

Gets the latest pipeline errors following use of `run_schedule()`. If the all runs succeeded or `run_schedule()` has not been run it will be `NULL`.

Usage

```
last_run_errors()
```

Value

error messages

Examples

```
last_run_errors()
```

last_run_messages *Retrieve latest maestro pipeline messages*

Description

Gets the latest pipeline messages following use of `run_schedule()`. If there are no messages or `run_schedule()` has not been run it will be `NULL`.

Usage

```
last_run_messages()
```

Details

Note that setting `maestroLogLevel` to something greater than `INFO` will ignore messages.

Value

messages

Examples

```
last_run_messages()
```

last_run_warnings *Retrieve latest maestro pipeline warnings*

Description

Gets the latest pipeline warnings following use of `run_schedule()`. If there are no warnings or `run_schedule()` has not been run it will be `NULL`.

Usage

```
last_run_warnings()
```

Details

Note that setting `maestroLogLevel` to something greater than `WARN` will ignore warnings.

Value

warning messages

Examples

```
last_run_warnings()
```

maestro `maestro` *package*

Description

Lightweight pipeline orchestration in R

Details

Documentation: GitHub

Author(s)

Maintainer: Will Hipson <will.e.hipson@gmail.com> (ORCID) [copyright holder]

Authors:

- Ryan Garnett <ryangarnett78@gmail.com> [contributor, copyright holder]

See Also

Useful links:

- https://github.com/whipson/maestro
- https://whipson.github.io/maestro/
- Report bugs at https://github.com/whipson/maestro/issues

`maestro_parse_cli` *cli output for generate schedule table*

Description

cli output for generate schedule table

Usage

```
maestro_parse_cli(parse_succeeds, parse_errors)
```

Arguments

`parse_succeeds`	list of parse results (i.e., succeeded)
`parse_errors`	list of parse errors

Value

cli output

`maestro_tags` *Maestro Tags*

Description

maestro tags are roxygen2 comments for configuring the scheduling and execution of pipelines.

Details

maestro tags follow the format `#' @maestroTagName`

Tag List:

tagName	description	value	examples (con
maestroFrequency	Time unit for scheduling	string	1 hour, daily, 3
maestroLogLevel	Threshold for logging when logging is requested	string	INFO, WARN
maestroSkip	Skips the pipeline when running (presence of tag indicates to skip)	n/a	
maestroStartTime	Start time of the pipeline; sets the point in time for recurrence	date or timestamp	1970-01-01 00

maestroTz	Timezone of the start time	string	UTC, America
maestroHours	Hours of day to run pipeline	ints	0 12 23
maestroDays	Days of week or days of month to run pipeline	ints or strings	1 14 30, Mon
maestroMonths	Months of year to run pipeline	ints	1 3 9 12

`parse_rounding_unit` *Parse a time string*

Description

Parse a time string

Usage

```
parse_rounding_unit(time_string)
```

Arguments

`time_string`	string like 1 day, daily, 2 weeks, 12 hours, etc.

Value

nunit list

`run_schedule` *Run a schedule*

Description

Given a schedule in a `maestro` project, runs the pipelines that are scheduled to execute based on the current time.

Usage

```
run_schedule(
  schedule,
  orch_frequency = "1 day",
  check_datetime = lubridate::now(tzone = "UTC"),
  resources = list(),
  run_all = FALSE,
  n_show_next = 5,
  cores = 1,
  logging = FALSE,
  log_file = NULL,
  log_file_max_bytes = 1e+06,
  quiet = FALSE
)
```

Arguments

schedule	a table of scheduled pipelines generated from build_schedule()
orch_frequency	of the orchestrator, a single string formatted like "1 day", "2 weeks", "hourly", etc.
check_datetime	datetime against which to check the running of pipelines (default is current system time in UTC)
resources	named list of shared resources made available to pipelines as needed
run_all	run all pipelines regardless of the schedule (default is FALSE) - useful for testing. Does not apply to pipes with a maestroSkip tag.
n_show_next	show the next n scheduled pipes
cores	number of cpu cores to run if running in parallel. If > 1, furrr is used and a multisession plan must be executed in the orchestrator (see details)
logging	whether or not to write the logs to a file (default = FALSE)
log_file	path to the log file (ignored if logging == FALSE)
log_file_max_bytes	numeric specifying the maximum number of bytes allowed in the log file before purging the log (within a margin of error)
quiet	silence metrics to the console (default = FALSE)

Details

Pipeline schedule logic:

The function run_schedule() examines each pipeline in the schedule table and determines whether it is scheduled to run at the current time using some simple time arithmetic. We assume run_schedule(schedule, check_datetime = Sys.time()), but this need not be the case.

Output:

run_schedule() returns a list with two elements: status and artifacts. Status is a data.frame where each row is a pipeline and the columns are information about the pipeline status, execution time, etc. Artifacts are any values returned from pipelines.

Pipelines with arguments (resources):

If a pipeline takes an argument that doesn't include a default value, these can be supplied in the orchestrator via run_schedule(resources = list(arg1 = val)). The name of the argument used by the pipeline must match the name of the argument in the list. Currently, each named resource must refer to a single object. In other words, you can't have two pipes using the same argument but requiring different values.

Running in parallel:

Pipelines can be run in parallel using the cores argument. First, you must run future::plan(future::multisession) in the orchestrator. Then, supply the desired number of cores to the cores argument. Note that console output appears different in multicore mode.

Value

list with named elements status and artifacts

Examples

```
# Runs the schedule every 1 day
run_schedule(
  example_schedule,
  orch_frequency = "1 day",
  quiet = TRUE
)

# Runs the schedule every 15 minutes
run_schedule(
  example_schedule,
  orch_frequency = "15 minutes",
  quiet = TRUE
)
```

---

run_schedule_entry *Runs a single pipeline*

Description

Runs a single pipeline

Usage

```
run_schedule_entry(
  script_path,
  pipe_name,
  resources = list(),
  log_file = NULL,
  log_level = "INFO",
  log_file_max_bytes
)
```

Arguments

script_path	path to the script containing the pipeline
pipe_name	name of the pipeline
resources	list of resources for the pipeline
log_file	path to log file
log_level	log level
log_file_max_bytes	numeric specifying the maximum number of bytes allowed in the log file before purging the log (within a margin of error)

Value

invisible

schedule_validity_check

Checks the validity of a schedule

Description

Checks the validity of a schedule

Usage

```
schedule_validity_check(schedule)
```

Arguments

schedule	a schedule table returned from build_schedule

Value

invisible or error

suggest_orch_frequency

Suggest orchestrator frequency based on a schedule

Description

Suggests a frequency to run the orchestrator based on the frequencies of the pipelines in a schedule.

Usage

```
suggest_orch_frequency(
  schedule,
  check_datetime = lubridate::now(tzone = "UTC")
)
```

Arguments

schedule	schedule data.frame created by build_schedule()
check_datetime	datetime against which to check the running of pipelines (default is current system time in UTC)

Details

This function attempts to find the smallest interval of time between all pipelines. If the smallest interval is less than 15 minutes, it just uses the smallest interval.

Note this function is intended to be used interactively when deciding how often to schedule the orchestrator. Programmatic use is not recommended.

Value

frequency string

Examples

```
suggest_orch_frequency(example_schedule)
```

Index