Package 'stevedore'

October 22, 2021

Title Docker Client

Version 0.9.4

Description Work with containers over the Docker API. Rather than using system calls to interact with a docker client, using the API directly means that we can receive richer information from docker. The interface in the package is automatically generated using the 'OpenAPI' (a.k.a., 'swagger') specification, and all return values are checked in order to make them type stable.

License MIT + file LICENSE

URL https://github.com/richfitz/stevedore

BugReports https://github.com/richfitz/stevedore/issues

Imports crayon,
curl (>= 2.3.0),
jsonlite,
yaml (>= 2.1.18)

Suggests knitr,
openssl,
redux,
reticulate,
rmarkdown,
testthat,
withr

SystemRequirements docker

RoxygenNote 7.1.2

VignetteBuilder knitr

Encoding UTF-8

ByteCompile TRUE

Language en-GB

R topics documented:

`docker_available` *Test if docker available*

Description

Test if we can construct a docker client and confirm that we can communicate with it. This is intended to help in debug connection issues, and also for use in tests. For example, you might implement a testthat skip test that skips if `stevedore::docker_available()` returns `FALSE` to conditionally use stevedore/docker within tests.

Usage

```
docker_available(..., verbose = FALSE)
```

Arguments

`...`	Passed through to `docker_client` (e.g., `api_version`, `host`).
`verbose`	Logical, indicating if information should be printed about failures to connect. If `FALSE` (the default) the function runs silently.

Details

Reasons for failure to connect might include:

- You do not have a docker daemon running
- You have docker installed but the socket in a non-standard place and have not adjusted environment variables accordingly
- You do not have permission to write to the docker socket
- You are on windows and the required python packages to get everything working there are not present or configured correctly
- There are problems arranging verification over https/tls.

If `versose` is `TRUE` then some diagnostic information will be printed.

Value

Logical scalar, `TRUE` if `docker_client(...)` would succeed.

Examples

```
# Is docker available on your system?
stevedore::docker_available()
```

`docker_client` *Create docker client*

Description

Create a docker client object, which allows you to interact with docker from R. The object has several *methods* that allow interaction with the docker daemon (for this object they are all "system" commands) and *collections*, which contains further methods. The client is structured similarly to the docker command line client, such that `docker container create <args>` in the command line becomes `docker$container$create(...)` in R (if the client is called R).

Usage

```
docker_client(
  ...,
  api_version = NULL,
  host = NULL,
  cert_path = NULL,
  tls_verify = NULL,
  machine = NULL,
  http_client_type = NULL,
  data_frame = NULL,
  quiet = FALSE,
  debug = NULL,
  ignore_environment = FALSE
)
```

Arguments

`...`	Reserved for future use. Passing in any unrecognised argument will throw an error. Part of the role of this argument is to force use of named arguments until the API is stabilised.
`api_version`	Version of the API to use when communicating with the docker daemon. The default value, `NULL`, detects the docker server API version and attempts to match it (this mirrors the default behaviour of the docker command line client). Alternatively, provide an API version number as a string or `numeric_version` object (supported between 1.25 and 1.39). The version 1.29 is the version used in most automated tests, and if problems are encountered, consider forcing this version).
`host`	The URL for the docker daemon. This can be a unix socket (e.g., unix:///var/run/docker.sock) on macOS/Linux, a named pipe (e.g., `npipe:////./pipe/docker_engine`) on Windows, or an http or https url (e.g., `https://localhost:2376`). If not given, we use the environment variable `DOCKER_HOST`, falling back on the default socket or named pipe (for macOS/unix and windows respectively).
`cert_path`	The path to a directory containing certificate files. If using an `https` url this is required. If not given, we use the environment variable `DOCKER_CERT_PATH`. This is ignored without warning if used with a socket or named pipe connection.
`tls_verify`	Logical, indicating if TLS should be verified. This is only used if using an https connection (i.e., host is a tcp/http/https url and`cert_path` is given). If not given, we use the environment variable `DOCKER_TLS_VERIFY`.
`machine`	Scalar character (if provided) indicating the name of a "docker machine" instance to use. If this is provided then `docker-machine` must be installed and the machine must exist and be running. `stevedore` will run `docker-machine env machine` to determine the environment variables to contact this machine and use these values for `host`, `cert_path` and `tls_verify` (silently ignoring any provided values). Carl Boettiger is working on a docker machine package for R that would make managing docker machines from R easier. As an alternative to this option, one can set docker-machine environment variables as described in `docker-machine env` before running R and they would be picked up as described above.
`http_client_type`	HTTP client type to use. The options are (currently) "curl", which uses the `curl` package (works over unix sockets and over TCP) and `httppipe` which works over unix sockets and windows named pipes, using the Docker SDK's pipe code via the `httppipe` package. Not all functionality is supported with the `httppipe` client. This option may eventually be moved into the `...` argument as is not intended for end-user use; it is primarily intended for debugging in development (forcing the `httppipe` client where the `curl` client would ordinarily be preferred).
`data_frame`	Function, used to wrap data.frames returned. This may make output easier to consume. You might use `tibble::as_tibble` to return a `tbl_df` or `datatable::as.data.table` to return `data.table` objects. This will be applied to all data.frames *after* they are constructed, and so must take a single argument (the newly constructed data.frame) and return a new object that is largely compatible with data.frame. Another use for this would be to define a function `data_frame = function(x)`

`structure(x,class = c("foo","data.frame"))` to set the class of all returned data.frame objects to be "foo" as well and then defining a custom S3 print method for "foo" that limited the output.

`quiet` Suppress informational messages.

`debug` Enable http debugging (supported by the curl http driver only). Provide a connection object and http headers and content will be sent to it. Using `debug = TRUE` is equivalent to `code = stdout()`, while `debug = FALSE` is equivalent to `debug = NULL` (the default) which prevents debugging information being printed. This option can be used to write to a file by opening a writeable connection but care must be made not to close this connection because otherwise the curl requests may fail.

`ignore_environment`
Logical, indicating if environment variables (`DOCKER_HOST`, `DOCKER_CERT_PATH`, `DOCKER_TLS_VERIFY` and `DOCKER_API_VERSION`) should be ignored (this has no effect if `machine` is specified).

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Management commands:

`config` Manage config - see `docker_config_collection`

`container` Manage container - see `docker_container_collection`

`image` Manage image - see `docker_image_collection`

`network` Manage network - see `docker_network_collection`

`node` Manage node - see `docker_node_collection`

`plugin` Manage plugin - see `docker_plugin_collection`

`secret` Manage secret - see `docker_secret_collection`

`service` Manage service - see `docker_service_collection`

`swarm` Manage swarm - see `docker_swarm_collection`

`task` Manage task - see `docker_task_collection`

`types` Manage types - see `docker_types`

`volume` Manage volume - see `docker_volume_collection`

Methods:

`api_version` Return the version of the API that the docker client is using to communicate with the server, as a string (e.g., "1.29")

Usage:

```
api_version()
```

`connection_info` Return information about the connection to the server.

Usage:

```
connection_info()
```

`cp` Copy a file or directory between the host filesystem and a running container. Like the command line, exactly one of `from`and `to`must be in the form `<container>:<path>`while the other must be in the form `<path>`. This wrapper is included for compatiblity with the docker command line and is available in a slightly more flexible form as `cp_out`and `cp_in`methods of a `docker_container`object. Similar to the cli command `docker cp`.

Usage:

```
cp(src, dest)
```

Arguments:

- `src`: The source file or directory. If on the host it may be a relative or an absolute path. If on the container then it must be of the form `<container_id>:<path>`; the container id can be an id (hex) or name.
- `dest`: The destination file or directory, with the same pattern as `from`.

`system_df` Get data usage information. Similar to the cli command `docker system df`.

Usage:

```
system_df()
```

`system_events` Monitor events. Stream real-time events from the server.

Various objects within Docker report events when something happens to them.

Containers report these events: `attach,commit,copy,create,destroy,detach,die,exec_create,exec_detach,`

Images report these events: `delete,import,load,pull,push,save,tag,untag`

Volumes report these events: `create,mount,unmount,destroy`

Networks report these events: `create,connect,disconnect,destroy`

The Docker daemon reports these events: `reload`. Similar to the cli command `docker events`.

Usage:

```
system_events(since = NULL, until = NULL, filters = NULL)
```

Arguments:

- `since`: Show events created since this timestamp then stream new events.
- `until`: Show events created until this timestamp then stop streaming.
- `filters`: A named character vector of filters to process on the event list. Available filters:
  - `container=<string>`container name or ID
  - `daemon=<string>`daemon name or ID
  - `event=<string>`event type
  - `image=<string>`image name or ID
  - `label=<string>`image or container label
  - `network=<string>`network name or ID
  - `plugin=<string>` plugin name or ID
  - `type=<string>`object to filter by, one of `container`, `image`, `volume`, `network`, or `daemon`
  - `volume=<string>`volume name or ID

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

system_info Get system information. Similar to the cli command docker info.

Usage:

```
system_info()
```

system_auth Check auth configuration. Validate credentials for a registry and, if available, get an identity token for accessing the registry without password. Similar to the cli command docker login.

Usage:

```
system_auth(username = NULL, password = NULL, email = NULL,
    serveraddress = NULL)
```

Arguments:

- username: For authentication to check
- password: For authentication to check
- email: For authentication to check
- serveraddress: For authentication to check

system_ping Ping. This is a dummy endpoint you can use to test if the server is accessible.

Usage:

```
system_ping()
```

request Interact directly with the Docker HTTP API. This might be useful for debugging or for trying out new/experimental docker features that have not yet made it into stevedore. This is most likely to be useful in conjunction with the Docker API docs; https://docs.docker.com/engine/api Each call will return a list with elements url(the used url), status_code(the HTTP status code), headers(a raw vector containing the returned headers) and content(a raw vector containing the content). The curldriver provides some other entries too.

Usage:

```
request(verb, path, query = NULL, body = NULL,
    headers = NULL, stream = NULL)
```

Arguments:

- verb: A HTTP verb (e.g., GET, POST) as a string. Will be converted to uppercase.
- path: A path as a string, e.g., /versionor /containers/json. Do not provide any hostname, and use an absoluate path.
- query: Optionally, a named list of query parameters (e.g., list(limit = 10)as a query parameter for /containers/json).
- body: Either a raw vector (e.g., a tar-ed image for POST /build) or a string representing json (e.g. for POST /containers/create).
- headers: Headers to send with the request. The API does not use this extensively except for X-Registry-Authfor some endpoints.
- stream: An R function to handle streaming output. If given then the function must take a single argument which will be a raw vector with the docker daemon's output. The function's return output is ignored so needs to do all actions via side effects.

system_version Get version. Returns the version of Docker that is running and various information about the system that Docker is running on. Similar to the cli command docker version.

Usage:

```
system_version()
```

Connection options

stevedore can connect to the docker daemon via a unix socket (this is the default set-up on Linux and macOS), over a named pipe (Windows 10 - see below) and https over a normal tcp connection (this is especially useful with docker-machine.

1. If the machine argument is given then stevedore queries docker-machine for settings. If that command fails (e.g., there is no machine, docker-machine not installed) then that will cause an error. (Note that the docker-machine output does not include API version information so the api_version argument is relevant, but host, cert_path and tls_verify will be silently ignored if provided).
2. The arguments host overrides the environment variable DOCKER_HOST, cert_path overrides DOCKER_CERT_PATH and tls_verify overrides DOCKER_TLS_VERIFY. If ignore_environment is TRUE then the environment variables are not used at all.
3. if code is not provided by any of the above methods (machine, argument or environment variable) it will fall back on the default unix socket (var/run/docker.sock) on Linux/macOS or the default windows named pipe (npipe:////./pipe/docker_engine) on windows.

The API version is set by the api_version argument, which falls back on the environment variable DOCKER_API_VERSION (this is the same as the docker command line client and the python SDK). If neither are provided then stevedore will detect the API version being used by the daemon and match that (provided it falls within the range of versions supported by the package).

Examples

```
if (stevedore::docker_available()) {
  # Create a new client object:
  client <- stevedore::docker_client()

  # Version information for your docker daemon:
  client$version()

  # General information about your daemon:
  client$info()

  # Most of the interesting methods are within the collections.
  # For example, to see a summary of running containers:
  client$container$list()

  # (see ?docker_container) for more information.
}
```

`docker_config_collection` *Management commands for working with swarm configs*

Description

Methods for managing docker swarm configurations. This object is `$config` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`config_create` Create a config. Unimplemented, as 'config_create' (POST /configs/create) requires docker API version at least 1.30 (version 1.29 used)

Usage:

```
config_create(...)
```

Arguments:

- `...`: Ignored in this version

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`config_inspect` Inspect a config. Unimplemented, as 'config_inspect' (GET /configs/id) requires docker API version at least 1.30 (version 1.29 used)

Usage:

```
config_inspect(...)
```

Arguments:

- `...`: Ignored in this version

`config_list` List configs. Unimplemented, as 'config_list' (GET /configs) requires docker API version at least 1.30 (version 1.29 used)

Usage:

```
config_list(...)
```

Arguments:

- `...`: Ignored in this version

`config_delete` Delete a config. Unimplemented, as 'config_delete' (DELETE /configs/id) requires docker API version at least 1.30 (version 1.29 used)

Usage:

```
config_delete(...)
```

Arguments:

- `...`: Ignored in this version

`config_update` Update a Config. Unimplemented, as 'config_update' (POST /configs/id/update) requires docker API version at least 1.30 (version 1.29 used)

Usage:

```
config_update(...)
```

Arguments:

- `...`: Ignored in this version

See Also

`docker_swarm_collection` for management commands for the swarm itself, and `docker_secret_collection` for a similar interface for configuring sensitive configurations.

`docker_container` *Management commands for working with a particular docker container*

Description

Methods for working with a particular docker container. Container objects are returned by creating or running a docker container, or by using `$container$get` to fetch an existing container by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`image_commit` Create a new image from a container. Similar to the cli command `docker commit` or `docker container commit`.

Usage:

```
image_commit(repo = NULL, tag = NULL, author = NULL,
    changes = NULL, comment = NULL, pause = NULL,
    hostname = NULL, domainname = NULL, user = NULL,
    attach_stdin = NULL, attach_stdout = NULL,
    attach_stderr = NULL, exposed_ports = NULL, tty = NULL,
    open_stdin = NULL, stdin_once = NULL, env = NULL,
```

```
cmd = NULL, healthcheck = NULL, args_escaped = NULL,
image = NULL, volumes = NULL, working_dir = NULL,
entrypoint = NULL, network_disabled = NULL,
mac_address = NULL, on_build = NULL, labels = NULL,
stop_signal = NULL, stop_timeout = NULL, shell = NULL)
```

Arguments:

- repo: Repository name for the created image
- tag: Tag name for the create image
- author: Author of the image (e.g., John Hannibal Smith <hannibal@a-team.com>)
- changes: Dockerfileinstructions to apply while committing
- comment: Commit message
- pause: Whether to pause the container before committing
- hostname: The hostname to use for the container, as a valid RFC 1123 hostname.
- domainname: The domain name to use for the container.
- user: The user that commands are run as inside the container.
- attach_stdin: Whether to attach to stdin.
- attach_stdout: Whether to attach to stdout.
- attach_stderr: Whether to attach to stderr.
- exposed_ports: A character vector of port mappings between the container and host, in (1) the form <host>:<container>(e.g., 10080:80to map the container's port 80 to the host's port 10080), <ip>:<host>:<container>to bind a specific host interface as well as a port (e.g., you can use localhostor 127.0.0.1for the first element), (3) form <port>to map the containers port to a random available port on the host s shorthand for <port>:<port>, or (3) a single logical value TRUEindicating to map all container ports to random available ports on the host. You can use the $ports()method in the docker_containerobject to query the port mapping of a running container. Multiple values can be provided to map multiple ports to the host (e.g., c("80","443:443").
- tty: Attach standard streams to a TTY, including stdinif it is not closed.
- open_stdin: Open stdin
- stdin_once: Close stdinafter one attached client disconnects
- env: A list of environment variables to set inside the container in the form ["VAR=value",...]. A variable without =is removed from the environment, rather than to have an empty value.
- cmd: Command to run specified as a string or an array of strings.
- healthcheck: A test to perform to check that the container is healthy.
- args_escaped: Command is already escaped (Windows only)
- image: The name of the image to use when creating the container
- volumes: A character vector of mappings of mount points on the host (or in volumes) to paths on the container. Each element must be of the form <path_host>:<path_container>, possibly followed by :rofor read-only mappings (i.e., the same syntax as the docker command line client). docker_volumeobjects have a $mapmethod to help with generating these paths for volume mappings.
- working_dir: The working directory for commands to run in.
- entrypoint: The entry point for the container as a string or an array of strings.

If the array consists of exactly one empty string (`[""]`) then the entry point is reset to system default (i.e., the entry point used by docker when there is no `ENTRYPOINT`instruction in the `Dockerfile`).

- `network_disabled`: Disable networking for the container.
- `mac_address`: MAC address of the container.
- `on_build`: `ONBUILD`metadata that were defined in the image's `Dockerfile`.
- `labels`: User-defined key/value metadata.
- `stop_signal`: Signal to stop a container as a string or unsigned integer.
- `stop_timeout`: Timeout to stop a container in seconds.
- `shell`: Shell for when `RUN`, `CMD`, and `ENTRYPOINT`uses a shell.

`cp_in` Copy files or directory into the container. This function tries to follow the same semantics as the command line `docker cp`command.

Usage:

```
cp_in(src, dest)
```

Arguments:

- `src`: File or directory to copy into the container.
- `dest`: Destination within the container. This must be a single path.

`cp_out` Copy files or directories from the container. This function tries to follow the same semantics as the command line `docker cp`command.

Usage:

```
cp_out(src, dest)
```

Arguments:

- `src`: File or directory within the container to copy to the host.
- `dest`: Destination on the host. This must be a single path.

`container_changes` Get changes on a container's filesystem. Returns which files in a container's filesystem have been added, deleted, or modified. The `Kind`of modification can be one of:

- `0`: Modified
- `1`: Added
- `2`: Deleted. Similar to the cli command `docker diff`or `docker container diff`.

Usage:

```
container_changes()
```

`exec` Create and run an exec instance. Run a command inside a running container. Similar to the cli command `docker exec`.

Usage:

```
exec(cmd, stdin = NULL, stdout = TRUE, stderr = TRUE,
    detach_keys = NULL, tty = NULL, env = NULL,
    privileged = NULL, user = NULL, detach = FALSE,
    stream = stdout())
```

Arguments:

- `cmd`: Command to run, as a string or array of strings.
- `stdin`: Attach to `stdin`of the exec command.
- `stdout`: Attach to `stdout`of the exec command.
- `stderr`: Attach to `stderr`of the exec command.

- detach_keys: Override the key sequence for detaching a container. Format is a single character [a-Z]or ctrl-<value>where <value>is one of: a-z, @, ^, [, ,or _.
- tty: Allocate a pseudo-TTY.
- env: A list of environment variables in the form ["VAR=value",...].
- privileged: Runs the exec process with extended privileges.
- user: The user, and optionally, group to run the exec process inside the container. Format is one of: user, user:group, uid, or uid:gid.
- detach: Detach from the command.
- stream: The stream to send output to. Options here are (1) a connection object (e.g. stdout(), stderr()or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) FALSEor NULLto disable any output.

exec_create Create an exec instance. Run a command inside a running container. Similar to the cli command docker exec.

Usage:

```
exec_create(cmd, stdin = NULL, stdout = TRUE, stderr = TRUE,
    detach_keys = NULL, tty = NULL, env = NULL,
    privileged = NULL, user = NULL)
```

Arguments:

- cmd: Command to run, as a string or array of strings.
- stdin: Attach to stdinof the exec command.
- stdout: Attach to stdoutof the exec command.
- stderr: Attach to stderrof the exec command.
- detach_keys: Override the key sequence for detaching a container. Format is a single character [a-Z]or ctrl-<value>where <value>is one of: a-z, @, ^, [, ,or _.
- tty: Allocate a pseudo-TTY.
- env: A list of environment variables in the form ["VAR=value",...].
- privileged: Runs the exec process with extended privileges.
- user: The user, and optionally, group to run the exec process inside the container. Format is one of: user, user:group, uid, or uid:gid.

container_export Export a container. Export the contents of a container as a tarball. Similar to the cli command docker exportor docker container export.

Usage:

```
container_export()
```

container_archive Get an archive of a filesystem resource in a container. Get a tar archive of a resource in the filesystem of container id.

Usage:

```
container_archive(path, dest)
```

Arguments:

- path: Resource in the container's filesystem to archive.
- dest: Destination for the archive. Must be either a scalar character (representing a filename) or NULL, in which case a raw vector is returned.

help Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

id Return the container's id

Usage:

```
id()
```

image Return the image for this container, as a docker_imageobject.

Usage:

```
image()
```

inspect Return detailed information about this container. Similar to the cli command docker container inspect.

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a container has been removed.

container_kill Kill a container. Send a POSIX signal to a container, defaulting to killing to the container. Similar to the cli command docker killor docker container kill.

Usage:

```
container_kill(signal = NULL)
```

Arguments:

- signal: Signal to send to the container as an integer or string (e.g. SIGINT)

labels Return labels for this container

Usage:

```
labels(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a container has been removed.

container_logs Get container logs. Get stdoutand stderrlogs from a container.
Note: This endpoint works only for containers with the json-fileor journaldlogging driver. Similar to the cli command docker logsor docker container logs.

Usage:

```
container_logs(follow = NULL, stdout = TRUE, stderr = TRUE,
    since = NULL, timestamps = NULL, tail = NULL,
    stream = stdout())
```

Arguments:

- follow: Return the logs as a stream.
  This will return a 101HTTP response with a Connection: upgradeheader, then hijack the HTTP connection to send raw output. For more information about hijacking and the stream format, hrefhttps://docs.docker.com/engine/api/1.29/#operation/ContainerAttachsee the documentation for the attach endpoint.
- stdout: Return logs from stdout
- stderr: Return logs from stderr
- since: Only return logs since this time, as a UNIX timestamp

- timestamps: Add timestamps to every log line
- tail: Only return this number of log lines from the end of the logs. Specify as an integer or allto output all log lines.
- stream: The stream to send output to. Options here are (1) a connection object (e.g. stdout(), stderr()or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) FALSEor NULLto disable any output.

name Return name for this container

Usage:

```
name()
```

container_path_stat Get information about files in a container. A response header X-Docker-Container-Path-Statis return containing a base64 - encoded JSON object with some filesystem header information about the path.

Usage:

```
container_path_stat(path)
```

Arguments:

- path: Resource in the container's filesystem to archive.

container_pause Pause a container. Use the cgroups freezer to suspend all processes in a container.

Traditionally, when suspending a process the SIGSTOPsignal is used, which is observable by the process being suspended. With the cgroups freezer the process is unaware, and unable to capture, that it is being suspended, and subsequently resumed. Similar to the cli command docker pauseor docker container pause.

Usage:

```
container_pause()
```

ports Return a data.frameof information about ports opened by this container. This is particularly useful in a container started with ports = TRUE, where container ports are mapped to random ports on the host. Similar to the cli command docker container port.

Usage:

```
ports(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a container has been removed.

container_import Extract an archive of files or folders to a directory in a container. Upload a tar archive to be extracted to a path in the filesystem of container id.

Usage:

```
container_import(src, path, no_overwrite_dir_non_dir = NULL)
```

Arguments:

- src: The input stream must be a tar archive compressed with one of the following algorithms: identity (no compression), gzip, bzip2, xz.
- path: Path to a directory in the container to extract the archive's contents into.
- no_overwrite_dir_non_dir: If '1', 'true', or 'True' then it will be an error if unpacking the given content would cause an existing directory to be replaced with a non-directory and vice versa.

reload Refresh infomation on the container from the server, returning $inspect()invisibly.

Usage:

```
reload()
```

container_delete Remove a container. Similar to the cli command docker rmor docker container rm.

Usage:

```
container_delete(delete_volumes = NULL, force = NULL,
    link = NULL)
```

Arguments:

- delete_volumes: Remove the volumes associated with the container.
- force: If the container is running, kill it before removing it.
- link: Remove the specified link associated with the container.

container_rename Rename a container. Similar to the cli command docker renameor docker container rename.

Usage:

```
container_rename(name)
```

Arguments:

- name: New name for the container

container_resize Resize a container TTY. Resize the TTY for a container. You must restart the container for the resize to take effect.

Usage:

```
container_resize(h = NULL, w = NULL)
```

Arguments:

- h: Height of the tty session in characters
- w: Width of the tty session in characters

container_restart Restart a container. Similar to the cli command docker restartor docker container restart.

Usage:

```
container_restart(t = NULL)
```

Arguments:

- t: Number of seconds to wait before killing the container

container_start Start a container. Similar to the cli command docker startor docker container start.

Usage:

```
container_start(detach_keys = NULL)
```

Arguments:

- detach_keys: Override the key sequence for detaching a container. Format is a single character [a-Z]or ctrl-<value>where <value>is one of: a-z, @, ^, [, ,or _.

container_stats Get container stats based on resource usage. This endpoint returns a live stream of a container's resource usage statistics.

The precpu_statsis the CPU statistic of last read, which is used for calculating the CPU usage percentage. It is not the same as the cpu_statsfield.

If either precpu_stats.online_cpusor cpu_stats.online_cpusis nil then for compatibility with older daemons the length of the corresponding cpu_usage.percpu_usagearray should be used. Similar to the cli command docker statsor docker container stats.

Usage:

```
container_stats()
```

`status` Short status of the container ("running", "exited", etc).

Usage:

```
status(reload = TRUE)
```

Arguments:

- `reload`: Refresh container data?

`container_stop` Stop a container. Similar to the cli command `docker stop`or `docker container stop`.

Usage:

```
container_stop(t = NULL)
```

Arguments:

- `t`: Number of seconds to wait before killing the container

`container_top` List processes running inside a container. On Unix systems, this is done by running the `ps`command. This endpoint is not supported on Windows. Similar to the cli command `docker top`or `docker container top`.

Usage:

```
container_top(ps_args = NULL)
```

Arguments:

- `ps_args`: The arguments to pass to `ps`. For example, `aux`

`container_unpause` Unpause a container. Resume a container which has been paused. Similar to the cli command `docker unpause`or `docker container unpause`.

Usage:

```
container_unpause()
```

`container_update` Update a container. Change various configuration options of a container without having to recreate it. Similar to the cli command `docker update`or `docker container update`.

Usage:

```
container_update(cpu_shares = NULL, memory = NULL,
    cgroup_parent = NULL, blkio_weight = NULL,
    blkio_weight_device = NULL, blkio_device_read_bps = NULL,
    blkio_device_write_bps = NULL,
    blkio_device_read_iops = NULL,
    blkio_device_write_iops = NULL, cpu_period = NULL,
    cpu_quota = NULL, cpu_realtime_period = NULL,
    cpu_realtime_runtime = NULL, cpuset_cpus = NULL,
    cpuset_mems = NULL, devices = NULL,
    device_cgroup_rules = NULL, disk_quota = NULL,
    kernel_memory = NULL, memory_reservation = NULL,
    memory_swap = NULL, memory_swappiness = NULL,
    nano_cpus = NULL, oom_kill_disable = NULL,
    pids_limit = NULL, ulimits = NULL, cpu_count = NULL,
    cpu_percent = NULL, io_maximum_iops = NULL,
    io_maximum_bandwidth = NULL, restart_policy = NULL)
```

Arguments:

- `cpu_shares`: An integer value representing this container's relative CPU weight versus other containers.
- `memory`: Memory limit in bytes.

- cgroup_parent: Path to cgroupsunder which the container's cgroupis created. If the path is not absolute, the path is considered to be relative to the cgroupspath of the init process. Cgroups are created if they do not already exist.
- blkio_weight: Block IO weight (relative weight).
- blkio_weight_device: Block IO weight (relative device weight) in the form [{"Path": "device_path","Weight": weight}].
- blkio_device_read_bps: Limit read rate (bytes per second) from a device, in the form [{"Path": "device_path","Rate": rate}].
- blkio_device_write_bps: Limit write rate (bytes per second) to a device, in the form [{"Path": "device_path","Rate": rate}].
- blkio_device_read_iops: Limit read rate (IO per second) from a device, in the form [{"Path": "device_path","Rate": rate}].
- blkio_device_write_iops: Limit write rate (IO per second) to a device, in the form [{"Path": "device_path","Rate": rate}].
- cpu_period: The length of a CPU period in microseconds.
- cpu_quota: Microseconds of CPU time that the container can get in a CPU period.
- cpu_realtime_period: The length of a CPU real-time period in microseconds. Set to 0 to allocate no time allocated to real-time tasks.
- cpu_realtime_runtime: The length of a CPU real-time runtime in microseconds. Set to 0 to allocate no time allocated to real-time tasks.
- cpuset_cpus: CPUs in which to allow execution (e.g., 0-3, 0,1)
- cpuset_mems: Memory nodes (MEMs) in which to allow execution (0-3, 0,1). Only effective on NUMA systems.
- devices: A list of devices to add to the container.
- device_cgroup_rules: a list of cgroup rules to apply to the container
- disk_quota: Disk limit (in bytes).
- kernel_memory: Kernel memory limit in bytes.
- memory_reservation: Memory soft limit in bytes.
- memory_swap: Total memory limit (memory + swap). Set as -1to enable unlimited swap.
- memory_swappiness: Tune a container's memory swappiness behavior. Accepts an integer between 0 and 100.
- nano_cpus: CPU quota in units of 10⁻⁹ CPUs.
- oom_kill_disable: Disable OOM Killer for the container.
- pids_limit: Tune a container's pids limit. Set -1 for unlimited.
- ulimits: A list of resource limits to set in the container. For example: {"Name": "nofile","Soft": 1024,"Hard": 2048}"
- cpu_count: The number of usable CPUs (Windows only).
  On Windows Server containers, the processor resource controls are mutually exclusive. The order of precedence is CPUCountfirst, then CPUShares, and CPUPercentlast.
- cpu_percent: The usable percentage of the available CPUs (Windows only).
  On Windows Server containers, the processor resource controls are mutually exclusive. The order of precedence is CPUCountfirst, then CPUShares, and CPUPercentlast.
- io_maximum_iops: Maximum IOps for the container system drive (Windows only)
- io_maximum_bandwidth: Maximum IO in bytes per second for the container system drive (Windows only)

- `restart_policy`: The behavior to apply when the container exits. The default is not to restart.
  An ever increasing delay (double the previous delay, starting at 100ms) is added before each restart to prevent flooding the server.

`container_wait` Wait for a container. Block until a container stops, then returns the exit code. Similar to the cli command `docker wait`or `docker container wait`.

Usage:

```
container_wait()
```

See Also

`docker_container_collection` for other container management methods.

`docker_container_collection` *Management commands for working with docker containers*

Description

Methods for working with docker containers. This object is `$container` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`container_create` Create a container. Similar to the cli command `docker create`or `docker container create`.

Usage:

```
container_create(image, cmd = NULL, hostname = NULL,
    domainname = NULL, user = NULL, attach_stdin = NULL,
    attach_stdout = NULL, attach_stderr = NULL, ports = NULL,
    tty = NULL, open_stdin = NULL, stdin_once = NULL,
    env = NULL, health_check = NULL, args_escaped = NULL,
    volumes = NULL, working_dir = NULL, entrypoint = NULL,
    network_disabled = NULL, mac_address = NULL,
    on_build = NULL, labels = NULL, stop_signal = NULL,
    stop_timeout = NULL, shell = NULL, host_config = NULL,
    network = NULL, name = NULL)
```

Arguments:

- `image`: The name of the image to use when creating the container

- cmd: Command to run specified as a string or an array of strings.
- hostname: The hostname to use for the container, as a valid RFC 1123 hostname.
- domainname: The domain name to use for the container.
- user: The user that commands are run as inside the container.
- attach_stdin: Whether to attach to stdin.
- attach_stdout: Whether to attach to stdout.
- attach_stderr: Whether to attach to stderr.
- ports: A character vector of port mappings between the container and host, in (1) the form <host>:<container>(e.g., 10080:80to map the container's port 80 to the host's port 10080), <ip>:<host>:<container>to bind a specific host interface as well as a port (e.g., you can use localhostor 127.0.0.1for the first element), (3) form <port>to map the containers port to a random available port on the host s shorthand for <port>:<port>, or (3) a single logical value TRUEindicating to map all container ports to random available ports on the host. You can use the $ports()method in the docker_containerobject to query the port mapping of a running container. Multiple values can be provided to map multiple ports to the host (e.g., c("80","443:443").
- tty: Attach standard streams to a TTY, including stdinif it is not closed.
- open_stdin: Open stdin
- stdin_once: Close stdinafter one attached client disconnects
- env: A list of environment variables to set inside the container in the form ["VAR=value",...]. A variable without =is removed from the environment, rather than to have an empty value.
- health_check: A test to perform to check that the container is healthy. Construct with $types$health_config()
- args_escaped: Command is already escaped (Windows only)
- volumes: A character vector of mappings of mount points on the host (or in volumes) to paths on the container. Each element must be of the form <path_host>:<path_container>, possibly followed by :rofor read-only mappings (i.e., the same syntax as the docker command line client). docker_volumeobjects have a $mapmethod to help with generating these paths for volume mappings.
- working_dir: The working directory for commands to run in.
- entrypoint: The entry point for the container as a string or an array of strings.
  If the array consists of exactly one empty string ([""]) then the entry point is reset to system default (i.e., the entry point used by docker when there is no ENTRYPOINTinstruction in the Dockerfile).
- network_disabled: Disable networking for the container.
- mac_address: MAC address of the container.
- on_build: ONBUILDmetadata that were defined in the image's Dockerfile.
- labels: User-defined key/value metadata.
- stop_signal: Signal to stop a container as a string or unsigned integer.
- stop_timeout: Timeout to stop a container in seconds.
- shell: Shell for when RUN, CMD, and ENTRYPOINTuses a shell.
- host_config: Container configuration that depends on the host we are running on
- network: This container's networking configuration.
- name: Assign the specified name to the container. Must match /?[a-zA-Z0-9_-]+.

`get` Get a container by name or id

Usage:

```
get(id)
```

Arguments:

- `id`: A scalar character with the container's name or id (abbreviations of the id are allowed and will be resolved by the docker daemon).

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`container_list` List containers. Similar to the cli command `docker ps`or `docker container ls`.

Usage:

```
container_list(all = NULL, limit = NULL, size = NULL,
    filters = NULL)
```

Arguments:

- `all`: Return all containers. By default, only running containers are shown
- `limit`: Return this number of most recently created containers, including non-running ones.
- `size`: Return the size of container as fields `SizeRw`and `SizeRootFs`.
- `filters`: Filters to process on the container list, as a named character vector. For example `c(status = "paused")`will only return paused containers. Available filters:
  - `ancestor=(<image-name>[:<tag>]`, `<image id>`, or `<image@digest>)`
  - `before=(<container id>`or `<container name>)`
  - `expose=(<port>[/<proto>]|<startport-endport>/[<proto>])`
  - `exited=<int>`containers with exit code of `<int>`
  - `health=(starting|healthy|unhealthy|none)`
  - `id=<ID>`a container's ID
  - `isolation=(default|process|hyperv)` (Windows daemon only)
  - `is-task=(true|false)`
  - `label=key`or `label="key=value"`of a container label
  - `name=<name>`a container's name
  - `network=(<network id>`or `<network name>)`
  - `publish=(<port>[/<proto>]|<startport-endport>/[<proto>])`
  - `since=(<container id>`or `<container name>)`
  - `status=(created|restarting|running|removing|paused|exited|dead)`
  - `volume=(<volume name>`or `<mount point destination>)`

`container_prune` Delete stopped containers. Similar to the cli command `docker container prune`.

Usage:

```
container_prune(filters = NULL)
```

Arguments:

- `filters`: Filters to process on the prune list, as a named character vector.
  Available filters:
  - `until=<timestamp>`Prune containers created before this timestamp. The `<timestamp>`can be Unix timestamps, date formatted timestamps, or Go duration strings (e.g. `10m`, `1h30m`) computed relative to the daemon machine's time.

`container_delete` Remove a container. Similar to the cli command `docker rm`or `docker container rm`.

Usage:

```
container_delete(id, delete_volumes = NULL, force = NULL,
    link = NULL)
```

Arguments:

- `id`: ID or name of the container
- `delete_volumes`: Remove the volumes associated with the container.
- `force`: If the container is running, kill it before removing it.
- `link`: Remove the specified link associated with the container.

`run` Run a command in a new container. This method does rather a lot, and wraps several other methods. It aims to mimic the behaviour of `docker run`in the command line tool. It will:

- Try to pull a container if it does not yet exist (see `$pull`in `docker_image_collection`)
- Create the container (see `$create`in `docker_container_collection`)
- Start the container (see `$start`in `docker_container`)
- Optionally stream the logs, if not detached (see `$logs`in `docker_container`
- Wait for the container to finish (see `$wait`in `docker_container`)

It returns a list with a container object as `"container"`and a `"docker_stream"`object containing logs as `"logs"`). If `rm = TRUE`and `detach = TRUE`the container object will be of limited utility and you will need to use `reload = FALSE`on many methods (and some will not work) as the container will have been removed on exit.
Unlike the command line version, interrupting the streaming logs will not necessarily kill the container but may leave it running in the background.
Unlike the command line version, the `attach = TRUE` simply attaches the output of the container and blocks the R session until it is complete. There is currently no way of sending input into the docker container. Similar to the cli command `docker run`.

Usage:

```
run(image, cmd = NULL, ..., detach = FALSE, rm = FALSE,
    stream = stdout(), host_config = NULL)
```

Arguments:

- `image`: Image to run. Can be a name, id, or a `docker_image`object.
- `cmd`: Command to run in the container. Must be a character vector. If not specified then the default `ENTRYPOINT`and `CMD`will be used (see the docker documentation for details)
- `...`: Additional arguments passed through to `$create`(see `docker_container_collection`. There are many possible arguments here.
- `detach`: Detach the container as soon as it has started and return control to R. The container will run in the background. The returned object can be used to interrogate the container afterwards (see `docker_container`).
- `rm`: Remove the container on exit.

- `stream`: The stream to use to send output to, if `detach = FALSE`. The default uses the standard output stream (i.e., where `cat`would send output). Other valid values are an R connection object, a string (interpreted as a filename) or `FALSE`to prevent any output.
- `host_config`: Passed through to `$create`, as with `...`.

See Also

`docker_container` for information on container objects.

`docker_exec` *Commands for working with a docker exec instance*

Description

Methods for working with docker "exec" instances, which are returned by running `exec` on a running container

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`id` Return the exec instance's id

Usage:

```
id()
```

`inspect` Return detailed information about this exec instance

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon.

`reload` Refresh infomation on the exec instance from the server, returning `$inspect()`invisibly.

Usage:

```
reload()
```

`exec_resize` Resize an exec instance. Resize the TTY session used by an exec instance. This endpoint only works if `tty`was specified as part of creating and starting the exec instance.

Usage:

```
exec_resize(h = NULL, w = NULL)
```

Arguments:

- `h`: Height of the TTY session in characters
- `w`: Width of the TTY session in characters

`exec_start` Start an exec instance. Starts a previously set up exec instance. If detach is true, this endpoint returns immediately after starting the command. Otherwise, it sets up an interactive session with the command.

Usage:

```
exec_start(detach = FALSE, tty = NULL, stream = stdout())
```

Arguments:

- `detach`: Detach from the command.
- `tty`: Allocate a pseudo-TTY.
- `stream`: The stream to send output to. Options here are (1) a connection object (e.g. `stdout()`, `stderr()`or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) `FALSE`or `NULL`to disable any output.

See Also

`docker_container`

`docker_image` *Management commands for working with a particular docker image*

Description

Methods for working with a particular docker image. Image objects are returned by building or pulling a docker image, or by using `$image$get` to fetch an existing image by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`image_tarball` Export an image. Get a tarball containing all images and metadata for a repository.

If `name`is a specific name and tag (e.g. `ubuntu:latest`), then only that image (and its parents) are returned. If `name`is an image ID, similarly only that image (and its parents) are returned, but with the exclusion of the `repositories`file in the tarball, as there were no image names referenced.

Image tarball format

An image tarball contains one directory per image layer (named using its long ID), each containing these files:

- VERSION: currently 1.0- the file format version
- json: detailed layer information, similar to docker inspect layer_id
- layer.tar: A tarfile containing the filesystem changes in this layer

The layer.tarfile contains aufsstyle .wh..wh.aufsfiles and directories for storing attribute changes and deletions.

If the tarball defines a repository, the tarball should also include a repositoriesfile at the root that contains a list of repository and tag names mapped to layer IDs.

```
json
{
  "hello-world": {
   "latest": "565a9d68a73f6706862bfe8409a7f659776d4d60a8d096eb4a3cbce6999cc2a1"
  }
}
```

. Similar to the cli command docker image save.

Usage:

```
image_tarball()
```

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

`image_history` Get the history of an image. Return parent layers of an image. Similar to the cli command docker image history.

Usage:

```
image_history()
```

`id` Return the image's id

Usage:

```
id()
```

`inspect` Return detailed information about this image. Similar to the cli command docker image inspect.

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter an image has been removed.

`labels` Return labels for this image

Usage:

```
labels(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter an image has been removed.

`name` Return name for this image

Usage:

```
name()
```

`reload` Refresh infomation on the image from the server, returning `$inspect()`invisibly.

Usage:

```
reload()
```

`image_delete` Remove an image. Remove an image, along with any untagged parent images that were referenced by that image.
Images can't be removed if they have descendant images, are being used by a running container or are being used by a build. Similar to the cli command `docker rmi`or `docker image rm`.

Usage:

```
image_delete(force = NULL, noprune = NULL)
```

Arguments:

- `force`: Remove the image even if it is being used by stopped containers or has other tags
- `noprune`: Do not delete untagged parent images

`short_id` Abbreviated identifier, truncated to 10 characters (perhaps plus the leading `sha256:`

Usage:

```
short_id()
```

`image_tag` Tag an image. Tag an image so that it becomes part of a repository. Similar to the cli command `docker tag`or `docker image tag`.

Usage:

```
image_tag(repo, tag = NULL)
```

Arguments:

- `repo`: The repository to tag in. For example, `someuser/someimage`.
- `tag`: The name of the new tag.

`tags` Return tags for this image

Usage:

```
tags(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after an image has been removed.

`untag` Remove tags from the image. This is a lot like `image$remove`(see `docker_image_collection`) but with `noprune = TRUE`.

Usage:

```
untag(repo_tag)
```

Arguments:

- `repo_tag`: Tags to remove

See Also

`docker_image_collection` for other image management methods.

`docker_image_collection` *Management commands for working with docker images*

Description

Methods for working with docker images. This object is `$image` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`image_build` Build an image. Build an image from a tar archive with a `Dockerfile`in it.
The `Dockerfile`specifies how the image is built from the tar archive. It is typically in the archive's root, but can be at a different path or have a different name by specifying the `dockerfile`parameter. See the `Dockerfile`reference for more information.
The Docker daemon performs a preliminary validation of the `Dockerfile`before starting the build, and returns an error if the syntax is incorrect. After that, each instruction is run one-by-one until the ID of the new image is output.
The build is canceled if the client drops the connection by quitting or being killed. Similar to the cli command `docker build`or `docker image build`.
Usage:

```
image_build(context, dockerfile = NULL, tag = NULL,
    extrahosts = NULL, remote = NULL, q = NULL,
    nocache = NULL, cachefrom = NULL, pull = NULL, rm = NULL,
    forcerm = NULL, memory = NULL, memswap = NULL,
    cpushares = NULL, cpusetcpus = NULL, cpuperiod = NULL,
    cpuquota = NULL, buildargs = NULL, shmsize = NULL,
    squash = NULL, labels = NULL, networkmode = NULL,
    registry_config = NULL, stream = stdout())
```

Arguments:

- `context`: A tar archive compressed with one of the following algorithms: identity (no compression), gzip, bzip2, xz.
- `dockerfile`: Path within the build context to the `Dockerfile`. This is ignored if `remote`is specified and points to an external `Dockerfile`.
- `tag`: A name and optional tag to apply to the image in the `name:tag`format. If you omit the tag the default `latest`value is assumed. You can provide a vector of mutiple tags here.
- `extrahosts`: Extra hosts to add to /etc/hosts

- remote: A Git repository URI or HTTP/HTTPS context URI. If the URI points to a single text file, the file's contents are placed into a file called Dockerfileand the image is built from that file. If the URI points to a tarball, the file is downloaded by the daemon and the contents therein used as the context for the build. If the URI points to a tarball and the dockerfileparameter is also specified, there must be a file with the corresponding path inside the tarball.
- q: Suppress verbose build output.
- nocache: Do not use the cache when building the image.
- cachefrom: JSON array of images used for build cache resolution.
- pull: Attempt to pull the image even if an older image exists locally.
- rm: Remove intermediate containers after a successful build.
- forcerm: Always remove intermediate containers, even upon failure.
- memory: Set memory limit for build.
- memswap: Total memory (memory + swap). Set as -1to disable swap.
- cpushares: CPU shares (relative weight).
- cpusetcpus: CPUs in which to allow execution (e.g., 0-3, 0,1).
- cpuperiod: The length of a CPU period in microseconds.
- cpuquota: Microseconds of CPU time that the container can get in a CPU period.
- buildargs: JSON map of string pairs for build-time variables. Users pass these values at build-time. Docker uses the buildargs as the environment context for commands run via the DockerfileRUN instruction, or for variable expansion in other Dockerfileinstructions. This is not meant for passing secret values. Read more about the buildargs instruction.
- shmsize: Size of /dev/shmin bytes. The size must be greater than 0. If omitted the system uses 64MB.
- squash: Squash the resulting images layers into a single layer. *(Experimental release only.)*
- labels: Arbitrary key/value labels to set on the image, as a JSON map of string pairs.
- networkmode: Sets the networking mode for the run commands during build. Supported standard values are: bridge, host, none, and container:<name|id>. Any other value is taken as a custom network's name to which this container should connect to.
- registry_config: This is a base64-encoded JSON object with auth configurations for multiple registries that a build may refer to.
  The key is a registry URL, and the value is an auth configuration object, hrefhttps://docs.docker.com/engine/api/1.2 described in the authentication section. For example:

```
{
  "docker.example.com": {
    "username": "janedoe",
    "password": "hunter2"
  },
  "https://index.docker.io/v1/": {
    "username": "mobydock",
    "password": "conta1n3rize14"
  }
}
```

  Only the registry domain name (and port if not the default 443) are required. However, for legacy reasons, the Docker Hub registry must be specified with both a https://prefix and a /v1/suffix even though Docker will prefer to use the v2 registry API.

- stream: The stream to send output to. Options here are (1) a connection object (e.g. stdout(), stderr()or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) FALSEor NULLto disable any output.

image_build_clean Delete builder cache. Unimplemented, as 'image_build_clean' (POST /build/prune) requires docker API version at least 1.31 (version 1.29 used)

Usage:

```
image_build_clean(...)
```

Arguments:

- ...: Ignored in this version

image_export Export several images. Get a tarball containing all images and metadata for several image repositories.

For each value of the namesparameter: if it is a specific name and tag (e.g. ubuntu:latest), then only that image (and its parents) are returned; if it is an image ID, similarly only that image (and its parents) are returned and there would be no names referenced in the 'repositories' file for this image ID.

For details on the format, see hrefhttps://docs.docker.com/engine/api/1.29/#operation/ImageGetthe export image endpoint. Similar to the cli command docker image save.

Usage:

```
image_export(names = NULL)
```

Arguments:

- names: Image names to filter by

get Get an image by name or id

Usage:

```
get(id)
```

Arguments:

- id: A scalar character with the image's name or id (abbreviations of the id are allowed and will be resolved by the docker daemon).

help Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

image_import Import images. Load a set of images and tags into a repository.

For details on the format, see hrefhttps://docs.docker.com/engine/api/1.29/#operation/ImageGetthe export image endpoint. Similar to the cli command docker image import.

Usage:

```
image_import(images_tarball = NULL, quiet = NULL)
```

Arguments:

- images_tarball: Tar archive containing images
- quiet: Suppress progress details during load.

image_list List Images. Returns a list of images on the server. Note that it uses a different, smaller representation of an image than inspecting a single image. Similar to the cli command docker imagesor docker image list.

Usage:

```
image_list(all = NULL, filters = NULL, digests = NULL)
```

Arguments:

- `all`: Show all images. Only images from a final layer (no children) are shown by default.
- `filters`: A named character vector of filters to process on the images list. Available filters:
  - `before=(<image-name>[:<tag>]`, `<image id>`or `<image@digest>`)
  - `dangling=true`
  - `label=key`or `label="key=value"`of an image label
  - `reference=(<image-name>[:<tag>])`
  - `since=(<image-name>[:<tag>]`, `<image id>`or `<image@digest>`)
- `digests`: Show digest information as a `RepoDigests`field on each image.

`image_prune` Delete unused images. Similar to the cli command `docker image prune`.

Usage:

```
image_prune(filters = NULL)
```

Arguments:

- `filters`: Filters to process on the prune list, as a named character vector. Available filters:
  - `dangling=<boolean>`When set to `true`(or `1`), prune only unused *and*untagged images. When set to `false` (or `0`), all unused images are pruned.
  - `until=<string>`Prune images created before this timestamp. The `<timestamp>`can be Unix timestamps, date formatted timestamps, or Go duration strings (e.g. `10m`, `1h30m`) computed relative to the daemon machine's time.

`image_create` Create an image. Create an image by either pulling it from a registry or importing it. Similar to the cli command `docker pull`or `docker image pull`.

Usage:

```
image_create(name, tag = NULL, stream = stdout())
```

Arguments:

- `name`: Name of the image to pull. The name may include a tag or digest. This parameter may only be used when pulling an image. The pull is cancelled if the HTTP connection is closed.
- `tag`: Tag or digest. If empty when pulling an image, this causes all tags for the given image to be pulled.
- `stream`: The stream to send output to. Options here are (1) a connection object (e.g. `stdout()`, `stderr()`or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) `FALSE`or `NULL`to disable any output.

`image_push` Push an image. Push an image to a registry.

If you wish to push an image on to a private registry, that image must already have a tag which references the registry. For example, `registry.example.com/myimage:latest`.

The push is cancelled if the HTTP connection is closed. Similar to the cli command `docker push`or `docker image push`.

Usage:

```
image_push(name, stream = stdout())
```

Arguments:

- `name`: Image name or ID.

- `stream`: The stream to send output to. Options here are (1) a connection object (e.g. `stdout()`, `stderr()`or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) `FALSE`or `NULL`to disable any output.

`image_delete` Remove an image. Remove an image, along with any untagged parent images that were referenced by that image.
Images can't be removed if they have descendant images, are being used by a running container or are being used by a build. Similar to the cli command `docker rmi`or `docker image rm`.
Usage:

```
image_delete(name, force = NULL, noprune = NULL)
```

Arguments:

- `name`: Image name or ID
- `force`: Remove the image even if it is being used by stopped containers or has other tags
- `noprune`: Do not delete untagged parent images

`image_search` Search images. Search for an image on Docker Hub. Similar to the cli command `docker search`.
Usage:

```
image_search(term, limit = NULL, filters = NULL)
```

Arguments:

- `term`: Term to search
- `limit`: Maximum number of results to return
- `filters`: A named character vector of filters to process on the images list. Available filters:
  - `is-automated=(true|false)`
  - `is-official=(true|false)`
  - `stars=<number>`Matches images that has at least 'number' stars.

See Also

`docker_image` for information on image objects.

`docker_network` *Management commands for working with a particular docker network*

Description

Methods for working with a particular docker network. Network objects are returned by creating a docker network, or by using `$network$get` to fetch an existing network by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`network_connect` Connect a container to a network. Similar to the cli command `docker network connect`.

Usage:

```
network_connect(container = NULL, endpoint_config = NULL)
```

Arguments:

- `container`: The ID or name of the container to connect to the network.
- `endpoint_config`: Configuration for a network endpoint.

`containers` Return containers connected to this network

Usage:

```
containers(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon (usually what you want).

`network_disconnect` Disconnect a container from a network. Similar to the cli command `docker network disconnect`.

Usage:

```
network_disconnect(container = NULL, force = NULL)
```

Arguments:

- `container`: The ID or name of the container to disconnect from the network.
- `force`: Force the container to disconnect from the network.

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`id` Return the network's id

Usage:

```
id()
```

`inspect` Return detailed information about this network. Similar to the cli command `docker network inspect`.

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after an network has been removed.

`name` Return name for this network

Usage:

```
name(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after a network has been removed.

`reload` Refresh infomation on the network from the server, returning `$inspect()`invisibly.

Usage:

```
reload()
```

`network_delete` Remove a network. Similar to the cli command `docker network rm`.

Usage:

```
network_delete()
```

See Also

`docker_network_collection` for other network management methods.

`docker_network_collection`

Management commands for working with docker networks

Description

Methods for working with docker networks. This object is `$network` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`network_create` Create a network. Similar to the cli command `docker network create`.

Usage:

```
network_create(name = NULL, check_duplicate = TRUE,
    driver = NULL, internal = NULL, attachable = NULL,
    ingress = NULL, ipam = NULL, enable_ipv6 = NULL,
    options = NULL, labels = NULL)
```

Arguments:

- `name`: The network's name.
- `check_duplicate`: Check for networks with duplicate names. Since Network is primarily keyed based on a random ID and not on the name, and network name is strictly a user-friendly alias to the network which is uniquely identified using ID, there is no guaranteed way to check for duplicates. CheckDuplicate is there to provide a best effort checking of any networks which has the same name but it is not guaranteed to catch all name collisions.
- `driver`: Name of the network driver plugin to use.
- `internal`: Restrict external access to the network.

- attachable: Globally scoped network is manually attachable by regular containers from workers in swarm mode.
- ingress: Ingress network is the network which provides the routing-mesh in swarm mode.
- ipam: Optional custom IP scheme for the network.
- enable_ipv6: Enable IPv6 on the network.
- options: Network specific options to be used by the drivers.
- labels: User-defined key/value metadata.

get Get a network by name or id

Usage:

```
get(id)
```

Arguments:

- id: A scalar character with the network's name or id (abbreviations of the id are allowed and will be resolved by the docker daemon).

help Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

network_list List networks. Similar to the cli command docker network ls.

Usage:

```
network_list(filters = NULL)
```

Arguments:

- filters: A named character vector of filters to process on the networks list. Available filters:
  - driver=<driver-name>Matches a network's driver.
  - id=<network-id>Matches all or part of a network ID.
  - label=<key>or label=<key>=<value>of a network label.
  - name=<network-name>Matches all or part of a network name.
  - scope=["swarm"|"global"|"local"]Filters networks by scope (swarm, global, or local).
  - type=["custom"|"builtin"]Filters networks by type. The customkeyword returns all user-defined networks.

network_prune Delete unused networks. Similar to the cli command docker network prune.

Usage:

```
network_prune(filters = NULL)
```

Arguments:

- filters: Filters to process on the prune list, as a named character vector.
  Available filters:
  - until=<timestamp>Prune networks created before this timestamp. The <timestamp>can be Unix timestamps, date formatted timestamps, or Go duration strings (e.g. 10m, 1h30m) computed relative to the daemon machine's time.

`network_delete` Remove a network. Similar to the cli command `docker network rm`.

Usage:

```
network_delete(id)
```

Arguments:

- `id`: Network ID or name

See Also

`docker_network` for information on network objects.

`docker_node` *Management commands for working with a particular docker node*

Description

Methods for working with a particular docker node. Node objects are by using `$node$get` to fetch an existing node by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`availability` Return the node's availability (active, pause, drain)

Usage:

```
availability(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after a node has been removed.

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`hostname` Return the node's hostname

Usage:

```
hostname(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a node has been removed.

id Return the node's id

Usage:

```
id()
```

inspect Return detailed information about this node. Similar to the cli command docker node inspect.

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a node has been removed.

reload Refresh infomation on the node from the server, returning $inspect()invisibly.

Usage:

```
reload()
```

role Return the node's role (worker, manager))

Usage:

```
role(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a node has been removed.

status Return the node's status (

Usage:

```
status(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a node has been removed.

node_update Update a node. Similar to the cli command docker node update.

Usage:

```
node_update(version, name = NULL, labels = NULL, role = NULL,
    availability = NULL)
```

Arguments:

- version: The version number of the node object being updated. This is required to avoid conflicting writes.
- name: Name for the node.
- labels: User-defined key/value metadata.
- role: Role of the node.
- availability: Availability of the node.

version Return the node's version. This is used by some other methods to avoid conflicting writes.

Usage:

```
version(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a node has been removed.

See Also

docker_node_collection for other node management methods.

`docker_node_collection`

Management commands for working with swarm nodes

Description

Methods for managing docker swarm nodes. This object is `$node` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`node_delete` Delete a node. Similar to the cli command `docker node rm`.

Usage:

```
node_delete(id, force = NULL)
```

Arguments:

- `id`: The ID or name of the node
- `force`: Force remove a node from the swarm

`get` Get a service by name or id

Usage:

```
get(id)
```

Arguments:

- `id`: A scalar character with the network's name or id (abbreviations of the id are allowed and will be resolved by the docker daemon).

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`node_list` List nodes. Similar to the cli command `docker node list`.

Usage:

```
node_list(filters = NULL)
```

Arguments:

- `filters`: Filters to process on the nodes list, as a named character vector.
  Available filters:
  - `id=<node id>`
  - `label=<engine label>`
  - `membership=(accepted|pending)`
  - `name=<node name>`
  - `role=(manager|worker)`

See Also

`docker_swarm_collection` for management commands for the swarm itself.

`docker_plugin` *Management commands for working with a particular docker plugin*

Description

Methods for working with a particular docker plugin. Plugin objects are returned by installing or building a docker plugin, or by using `$plugin$get` to fetch an existing plugin by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`plugin_configure` Configure a plugin. Similar to the cli command `docker plugin set`.

Usage:

```
plugin_configure(body = NULL)
```

Arguments:

- `body`: NA

`plugin_disable` Disable a plugin. Similar to the cli command `docker plugin disable`.

Usage:

```
plugin_disable()
```

`plugin_enable` Enable a plugin. Similar to the cli command `docker plugin enable`.

Usage:

```
plugin_enable(timeout = 0L)
```

Arguments:

- `timeout`: Set the HTTP client timeout (in seconds)

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

id Return the plugins' id

Usage:

```
id()
```

inspect Return detailed information about this plugin

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a plugin has been removed.

is_enabled Test if plugin is enabled

Usage:

```
is_enabled(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a plugin has been removed.

name Return the plugin's name

Usage:

```
name()
```

reload Refresh infomation on the plugin from the server, returning $inspect()invisibly.

Usage:

```
reload()
```

plugin_remove Remove a plugin. Similar to the cli command docker plugin rm.

Usage:

```
plugin_remove(force = NULL)
```

Arguments:

- force: Disable the plugin before removing. This may result in issues if the plugin is in use by a container.

See Also

docker_plugin_collection for other plugin management methods.

---

docker_plugin_collection

Management commands for working with docker plugins

Description

Methods for working with docker plugins. This object is $plugin within a docker_client object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`plugin_create` Create a plugin. Similar to the cli command `docker plugin create`.

Usage:

```
plugin_create(name, plugin_data_dir)
```

Arguments:

- `name`: The name of the plugin. The `:latest`tag is optional, and is the default if omitted.
- `plugin_data_dir`: Path to tar containing plugin rootfs and manifest

`get` Get a plugin by name or id

Usage:

```
get(name)
```

Arguments:

- `name`: A scalar character with the plugins' name.

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`plugin_install` Install a plugin. Pulls and installs a plugin. After the plugin is installed, it can be enabled using the hrefhttps://docs.docker.com/engine/api/1.29/#operation/PostPluginsEnable`POST /plugins/{name}/enable`endpoint. Similar to the cli command `docker plugin install`.

Usage:

```
plugin_install(remote, alias = NULL, registry_auth = NULL,
    disable = FALSE, grant_all = NULL, stream = stdout())
```

Arguments:

- `remote`: Remote reference for plugin to install.
  The `:latest`tag is optional, and is used as the default if omitted.
- `alias`: Local name for the pulled plugin.
  The `:latest`tag is optional, and is used as the default if omitted.
- `registry_auth`: A base64-encoded auth configuration to use when pulling a plugin from a registry. hrefhttps://docs.docker.com/engine/api/1.29/#section/AuthenticationSee the authentication section for details.
- `disable`: Do not enable the plugin on install
- `grant_all`: Logical, indicating if all requested permissions should be granted. If `NULL`, this will prompt interactively for permission. Setting this to `FALSE`will cause the installation to fail.

- `stream`: The stream to send output to. Options here are (1) a connection object (e.g. `stdout()`, `stderr()`or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) `FALSE`or `NULL`to disable any output.

`plugin_list` List plugins. Returns information about installed plugins. Similar to the cli command `docker plugin ls`.

Usage:

```
plugin_list(filters = NULL)
```

Arguments:

- `filters`: A JSON encoded value of the filters (a `map[string][]string`) to process on the plugin list. Available filters:
  - `capability=<capability name>`
  - `enable=<true>|<false>`

`plugin_privileges` Get plugin privileges

Usage:

```
plugin_privileges(remote)
```

Arguments:

- `remote`: The name of the plugin. The `:latest`tag is optional, and is the default if omitted.

See Also

`docker_plugin` for information on plugin objects.

`docker_secret_collection`

Management commands for working with swarm secrets

Description

Methods for managing docker swarm secrets. This object is `$secret` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`secret_create` Create a secret. Similar to the cli command `docker secret create`.

Usage:

```
secret_create(name, data, labels = NULL)
```

Arguments:

- name: User-defined name of the secret.
- data: Secret data, as a raw vector or character string
- labels: User-defined key/value metadata.

help Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

secret_inspect Inspect a secret. Similar to the cli command docker secret inspect.

Usage:

```
secret_inspect(id)
```

Arguments:

- id: ID of the secret

secret_list List secrets. Similar to the cli command docker secret ls.

Usage:

```
secret_list(filters = NULL)
```

Arguments:

- filters: A named character vector of filters to process on the secrets list. Available filters:
  - id=<secret id>
  - label=<key> or label=<key>=value
  - name=<secret name>
  - names=<secret name>

secret_delete Delete a secret. Similar to the cli command docker secret rm.

Usage:

```
secret_delete(id)
```

Arguments:

- id: ID of the secret

secret_update Update a Secret

Usage:

```
secret_update(id, version, name = NULL, labels = NULL,
    data = NULL)
```

Arguments:

- id: The ID or name of the secret
- version: The version number of the secret object being updated. This is required to avoid conflicting writes.
- name: User-defined name of the secret.
- labels: User-defined key/value metadata.
- data: Secret data, as a raw vector or character string

See Also

docker_swarm_collection for management commands for the swarm itself, and docker_config_collection for a similar interface for configuring non-sensitive configurations.

`docker_service` *Management commands for working with a particular docker service*

Description

Methods for working with a particular docker service. Service objects are returned by creating a docker service, or by using `$service$get` to fetch an existing service by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`id` Return the services's id

Usage:

```
id()
```

`inspect` Return detailed information about this service. Similar to the cli command `docker service inspect`.

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after a service has been removed.

`name` Return the service's name

Usage:

```
name(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after a service has been removed.

`ps` Get summary information about a service. Similar to the cli command `docker service ps`.

Usage:

```
ps(resolve_names = TRUE, filters = NULL)
```

Arguments:

- `resolve_names`: Resolve task and node names, from their ids? If `FALSE`, rather than do this instead only print ids. Task "names" are constructed from a combination of service name (or id) and the replicate number.
- `filters`: A named character vector of filters to process on the tasks list. Available filters are the same as `$task$list()`, except that `service`is not accepted (it is used internally by this method). Useful filters include `desired-state`, `label`, `node`, `name`and `id`.

`reload` Refresh infomation on the service from the server, returning `$inspect()`invisibly.

Usage:

```
reload()
```

`service_delete` Delete a service. Similar to the cli command `docker service rm`.

Usage:

```
service_delete()
```

`tasks` Fetch tasks associated with this service. Returns a list of `docker_task`objects

Usage:

```
tasks(filters = NULL)
```

Arguments:

- `filters`: A named character vector of filters to process on the tasks list. Available filters are the same as `$task$list()`, except that `service`is not accepted (it is used internally by this method). Useful filters include `desired-state`, `label`, `node`, `name`and `id`.

`version` Return the service's version. This is used by some other methods to avoid conflicting writes.

Usage:

```
version(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after a service has been removed.

See Also

`docker_service_collection` for other service management methods.

`docker_service_collection` *Management commands for working with docker services*

Description

Methods for working with docker services. This object is `$service` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`service_create` Create a service. Similar to the cli command `docker service create`.

Usage:

```
service_create(name = NULL, labels = NULL, image = NULL,
    container_spec_labels = NULL, command = NULL,
    args = NULL, hostname = NULL, env = NULL, dir = NULL,
    user = NULL, groups = NULL, tty = NULL,
    open_stdin = NULL, read_only = NULL, mounts = NULL,
    stop_signal = NULL, stop_grace_period = NULL,
    health_check = NULL, hosts = NULL, dns_config = NULL,
    secrets = NULL, resources = NULL, restart_policy = NULL,
    placement = NULL, force_update = NULL,
    task_spec_networks = NULL, log_driver = NULL,
    update_config = NULL, rollback_config = NULL,
    networks = NULL, endpoint_spec = NULL,
    registry_auth = NULL, stream = stdout(), replicas = NULL,
    global = FALSE, detach = FALSE, timeout = 60,
    time_wait_stable = 5, task_template = NULL,
    container_spec = NULL)
```

Arguments:

- `name`: Name of the service.
- `labels`: User-defined key/value metadata.
- `image`: The image name to use for the container.
- `container_spec_labels`: User-defined key/value data.
- `command`: The command to be run in the image.
- `args`: Arguments to the command.
- `hostname`: The hostname to use for the container, as a valid RFC 1123 hostname.
- `env`: A list of environment variables in the form `VAR=value`.
- `dir`: The working directory for commands to run in.
- `user`: The user inside the container.
- `groups`: A list of additional groups that the container process will run as.
- `tty`: Whether a pseudo-TTY should be allocated.
- `open_stdin`: Open `stdin`
- `read_only`: Mount the container's root filesystem as read only.
- `mounts`: Specification for mounts to be added to containers created as part of the service.
- `stop_signal`: Signal to stop the container.
- `stop_grace_period`: Amount of time to wait for the container to terminate before forcefully killing it.
- `health_check`: A test to perform to check that the container is healthy. Construct with `$types$health_config()`
- `hosts`: A list of hostnames/IP mappings to add to the container's `/etc/hosts`file. The format of extra hosts on swarmkit is specified in: http://man7.org/linux/man-pages/man5/hosts.5.html IP_address canonical_hostname [aliases...]
- `dns_config`: Specification for DNS related configurations in resolver configuration file (`resolv.conf`). Construct with `$types$dns_config()`

- `secrets`: Secrets contains references to zero or more secrets that will be exposed to the service.
- `resources`: Resource requirements which apply to each individual container created as part of the service. Construct with `$types$task_resources()`
- `restart_policy`: Specification for the restart policy which applies to containers created as part of this service. Construct with `$types$task_restart_policy()`
- `placement`: Construct with `$types$task_placement()`
- `force_update`: A counter that triggers an update even if no relevant parameters have been changed.
- `task_spec_networks`: NA
- `log_driver`: Specifies the log driver to use for tasks created from this spec. If not present, the default one for the swarm will be used, finally falling back to the engine default if not specified.
- `update_config`: Specification for the update strategy of the service.
- `rollback_config`: Specification for the rollback strategy of the service.
- `networks`: Array of network names or IDs to attach the service to.
- `endpoint_spec`: Properties that can be configured to access and load balance a service.
- `registry_auth`: A base64-encoded auth configuration for pulling from private registries. hrefhttps://docs.docker.com/engine/api/1.29/#section/AuthenticationSee the authentication section for details.
- `stream`: The stream to send output to. Options here are (1) a connection object (e.g. `stdout()`, `stderr()`or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) `FALSE`or `NULL`to disable any output.
- `replicas`: Number of replicas to use. If `NULL`(the default) then a single replica is used (the same behaviour as the command line client).
- `global`: Logical, indicating if this should be a "global" service (running one instance on each node) rather than a "replicated" service (running across nodes - the default).
- `detach`: Logical, indicating if we should exit immediately, rather than waiting for the service to converge. This option is named for consistency with `docker service create`- this command never "attaches" in the sense of a single container.
- `timeout`: Time, in seconds, to wait for the service to converge when `detach = FALSE`. If the service has not converged by this time an error will be thrown, but the docker daemon will continue trying to start the service.
- `time_wait_stable`: Time, in seconds, to wait for the service to stay up after it has started. This is needed to detect services that fail shortly after starting. The default is 5 seconds, which matches the command line client.
- `task_template`: User modifiable task configuration. Construct with `$types$task_spec()`. If this is given then 'container_spec', 'resources', 'restart_policy', 'placement', 'force_update', 'task_spec_networks' and 'log_driver' must all be NULL.
- `container_spec`: Construct with `$types$container_spec()`. If this is given then 'image', 'container_spec_labels', 'command', 'args', 'hostname', 'env', 'dir', 'user', 'groups', 'tty', 'open_stdin', 'read_only', 'mounts', 'stop_signal', 'stop_grace_period', 'health_check', 'hosts', 'dns_config' and 'secrets' must all be NULL.

`get` Get a service by name or id

Usage:

```
get(id)
```

Arguments:

- `id`: A scalar character with the network's name or id (abbreviations of the id are allowed and will be resolved by the docker daemon).

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`service_list` List services. Similar to the cli command `docker service list`.

Usage:

```
service_list(filters = NULL)
```

Arguments:

- `filters`: A named character vector of filters to process on the services list. Available filters:
  - `id=<service id>`
  - `label=<service label>`
  - `mode=["replicated"|"global"]`
  - `name=<service name>`

`service_delete` Delete a service. Similar to the cli command `docker service rm`.

Usage:

```
service_delete(id)
```

Arguments:

- `id`: ID or name of service.

See Also

`docker_service` for information on service objects.

`docker_swarm_collection`

Management commands for working with docker swarm

Description

Methods for managing the docker swarm. This object is `$swarm` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`swarm_init` Initialize a new swarm. Similar to the cli command `docker swarm init`.

Usage:

```
swarm_init(listen_addr = "0.0.0.0:2377",
    advertise_addr = NULL, force_new_cluster = NULL,
    spec = NULL)
```

Arguments:

- `listen_addr`: Listen address used for inter-manager communication, as well as determining the networking interface used for the VXLAN Tunnel Endpoint (VTEP). This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the default swarm listening port is used.
- `advertise_addr`: Externally reachable address advertised to other nodes. This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the port number from the listen address is used. If `AdvertiseAddr`is not specified, it will be automatically detected when possible.
- `force_new_cluster`: Force creation of a new swarm.
- `spec`: User modifiable swarm configuration.

`swarm_inspect` Inspect swarm

Usage:

```
swarm_inspect()
```

`swarm_join` Join an existing swarm. Similar to the cli command `docker swarm join`.

Usage:

```
swarm_join(listen_addr = NULL, advertise_addr = NULL,
    remote_addrs = NULL, join_token = NULL)
```

Arguments:

- `listen_addr`: Listen address used for inter-manager communication if the node gets promoted to manager, as well as determining the networking interface used for the VXLAN Tunnel Endpoint (VTEP).

- advertise_addr: Externally reachable address advertised to other nodes. This can either be an address/port combination in the form 192.168.1.1:4567, or an interface followed by a port number, like eth0:4567. If the port number is omitted, the port number from the listen address is used. If AdvertiseAddris not specified, it will be automatically detected when possible.
- remote_addrs: Addresses of manager nodes already participating in the swarm.
- join_token: Secret token for joining this swarm.

swarm_leave Leave a swarm. Similar to the cli command docker swarm leave.

Usage:

```
swarm_leave(force = NULL)
```

Arguments:

- force: Force leave swarm, even if this is the last manager or that it will break the cluster.

swarm_unlock Unlock a locked manager. Similar to the cli command docker swarm unlock.

Usage:

```
swarm_unlock(unlock_key = NULL)
```

Arguments:

- unlock_key: The swarm's unlock key.

swarm_unlock_key Get the unlock key. Similar to the cli command docker swarm unlock-key.

Usage:

```
swarm_unlock_key()
```

swarm_update Update a swarm. Similar to the cli command docker swarm update.

Usage:

```
swarm_update(version, name = NULL, labels = NULL,
    orchestration = NULL, raft = NULL, dispatcher = NULL,
    ca_config = NULL, encryption_config = NULL,
    task_defaults = NULL, rotate_worker_token = NULL,
    rotate_manager_token = NULL,
    rotate_manager_unlock_key = NULL)
```

Arguments:

- version: The version number of the swarm object being updated. This is required to avoid conflicting writes.
- name: Name of the swarm.
- labels: User-defined key/value metadata.
- orchestration: Orchestration configuration.
- raft: Raft configuration.
- dispatcher: Dispatcher configuration.
- ca_config: CA configuration.
- encryption_config: Parameters related to encryption-at-rest.
- task_defaults: Defaults for creating tasks in this cluster.
- rotate_worker_token: Rotate the worker join token.
- rotate_manager_token: Rotate the manager join token.
- rotate_manager_unlock_key: Rotate the manager unlock key.

`docker_task` *Management commands for working with a particular docker task*

Description

Methods for working with a particular docker task. Task objects are returned by using `$task$get` to fetch an existing task by name or id, or `$tasks` from a `docker_service` object representing a docker service.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`id` Return the tasks's id

Usage:

```
id()
```

`inspect` Return detailed information about this task

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after a task has been removed.

`task_logs` Get task logs. Get `stdout`and `stderr`logs from a task.

Note: This endpoint works only for services with the `json-file`or `journald`logging drivers.

Usage:

```
task_logs(details = NULL, follow = NULL, stdout = TRUE,
    stderr = TRUE, since = NULL, timestamps = NULL,
    tail = NULL, stream = stdout())
```

Arguments:

- `details`: Show task context and extra details provided to logs.

- follow: Return the logs as a stream.
  This will return a 101HTTP response with a Connection: upgradeheader, then hijack the HTTP connection to send raw output. For more information about hijacking and the stream format, hrefhttps://docs.docker.com/engine/api/1.29/#operation/ContainerAttachsee the documentation for the attach endpoint.
- stdout: Return logs from stdout
- stderr: Return logs from stderr
- since: Only return logs since this time, as a UNIX timestamp
- timestamps: Add timestamps to every log line
- tail: Only return this number of log lines from the end of the logs. Specify as an integer or allto output all log lines.
- stream: The stream to send output to. Options here are (1) a connection object (e.g. stdout(), stderr()or a writable open file connection object, (2) a scalar character indicating a filename to write to, or (3) FALSEor NULLto disable any output.

reload Refresh infomation on the task from the server, returning $inspect()invisibly.

Usage:

```
reload()
```

service Return parent service

Usage:

```
service()
```

state Retrieve task state (running, shutdown, etc). Richer information about task status is available as $inspect()$status

Usage:

```
state(reload = TRUE)
```

Arguments:

- reload: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to FALSEafter a task has been removed.

See Also

docker_task_collection for other task management methods.

---

docker_task_collection

Management commands for working with docker tasks

Description

Methods for working with docker tasks. This object is $task within a docker_client object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`get` Get a task by name or id

Usage:

```
get(id)
```

Arguments:

- `id`: A scalar character with the network's name or id (abbreviations of the id are allowed and will be resolved by the docker daemon).

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`task_list` List tasks

Usage:

```
task_list(filters = NULL)
```

Arguments:

- `filters`: A named character vector of filters to process on the tasks list. Available filters:
  - `desired-state=(running | shutdown | accepted)`
  - `id=<task id>`
  - `label=keyor label="key=value"`
  - `name=<task name>`
  - `node=<node id or name>`
  - `service=<service name>`

See Also

`docker_task` for information on task objects.

`docker_types` *Constructors for complex types*

Description

Methods for building complex docker types. This is most objects more complicated than R's atomic types. Most functions will indicate if they require one of these objects in their help. None of these functions do anything interesting in their own regard - they just validate inputs.

Details

The functions here will all depend on the API versions - some of the most fluid parts of the docker API are the different options that are supported via things like `host_config`.

These functions are needed because `stevedore` aims to be a fairly direct wrapping around the docker API. For most of the single host methods the types here are not really used (with the notable exception of `host_config` which is used by `$container$create` and `$container$update`). But for the swarm endpoints the function definitions would be impossibly complex if we did not reflect the types. So rather than one function call with a hundred arguments, we can build up the required types.

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`container_spec` container_spec

Usage:

```
container_spec(image = NULL, labels = NULL, command = NULL,
    args = NULL, hostname = NULL, env = NULL, dir = NULL,
    user = NULL, groups = NULL, tty = NULL,
    open_stdin = NULL, read_only = NULL, mounts = NULL,
    stop_signal = NULL, stop_grace_period = NULL,
    health_check = NULL, hosts = NULL, dns_config = NULL,
    secrets = NULL)
```

Arguments:

- `image`: The image name to use for the container.
- `labels`: User-defined key/value data.
- `command`: The command to be run in the image.
- `args`: Arguments to the command.
- `hostname`: The hostname to use for the container, as a valid RFC 1123 hostname.
- `env`: A list of environment variables in the form `VAR=value`.
- `dir`: The working directory for commands to run in.
- `user`: The user inside the container.
- `groups`: A list of additional groups that the container process will run as.
- `tty`: Whether a pseudo-TTY should be allocated.
- `open_stdin`: Open `stdin`
- `read_only`: Mount the container's root filesystem as read only.
- `mounts`: Specification for mounts to be added to containers created as part of the service.
- `stop_signal`: Signal to stop the container.
- `stop_grace_period`: Amount of time to wait for the container to terminate before forcefully killing it.
- `health_check`: A test to perform to check that the container is healthy. Construct with `$types$health_config()`

- hosts: A list of hostnames/IP mappings to add to the container's /etc/hostsfile. The format of extra hosts on swarmkit is specified in: http://man7.org/linux/man-pages/man5/hosts.5.html IP_address canonical_hostname [aliases...]
- dns_config: Specification for DNS related configurations in resolver configuration file (resolv.conf). Construct with $types$dns_config()
- secrets: Secrets contains references to zero or more secrets that will be exposed to the service.

dns_config Specification for DNS related configurations in resolver configuration file (resolv.conf).

Usage:

```
dns_config(nameservers = NULL, search = NULL,
    options = NULL)
```

Arguments:

- nameservers: The IP addresses of the name servers.
- search: A search list for host-name lookup.
- options: A list of internal resolver variables to be modified (e.g., debug, ndots:3, etc.).

health_config A test to perform to check that the container is healthy.

Usage:

```
health_config(test = NULL, interval = NULL, timeout = NULL,
    retries = NULL, start_period = NULL)
```

Arguments:

- test: The test to perform. Possible values are:
  - []inherit healthcheck from image or parent image
  - ["NONE"]disable healthcheck
  - ["CMD",args...]exec arguments directly
  - ["CMD-SHELL",command]run command with system's default shell
- interval: The time to wait between checks in nanoseconds. It should be 0 or not less than 1000000000(1s). 0 means inherit.
- timeout: The time to wait before considering the check to have hung. It should be 0 or not less than 1000000000(1s). 0 means inherit.
- retries: The number of consecutive failures needed to consider a container as unhealthy. 0 means inherit.
- start_period: Start period for the container to initialize before starting health-retries countdown in nanoseconds. 0 means inherit.

help Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- help_type: Passed to utils::help, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value getOption("help_type")and should follow the same behaviour as other R help (e.g., using "?")

mount mount

Usage:

```
mount(target = NULL, source = NULL, type = NULL,
    read_only = NULL, consistency = NULL,
    bind_options = NULL, volume_options = NULL,
    tmpfs_options = NULL)
```

Arguments:

- `target`: Container path.
- `source`: Mount source (e.g. a volume name, a host path).
- `type`: The mount type. Available types:
  - `bind`Mounts a file or directory from the host into the container. Must exist prior to creating the container.
  - `volume`Creates a volume with the given name and options (or uses a pre-existing volume with the same name and options). These are **not**removed when the container is removed.
  - `tmpfs`Create a tmpfs with the given options. The mount source cannot be specified for tmpfs.
- `read_only`: Whether the mount should be read-only.
- `consistency`: The consistency requirement for the mount: `default`, `consistent`, `cached`, or `delegated`.
- `bind_options`: Optional configuration for the `bind`type.
- `volume_options`: Optional configuration for the `volume`type.
- `tmpfs_options`: Optional configuration for the `tmpfs`type.

`task_placement` task_placement

Usage:

```
task_placement(constraints = NULL, preferences = NULL)
```

Arguments:

- `constraints`: An array of constraints.
- `preferences`: Preferences provide a way to make the scheduler aware of factors such as topology. They are provided in order from highest to lowest precedence.

`task_resources` Resource requirements which apply to each individual container created as part of the service.

Usage:

```
task_resources(limits = NULL, reservation = NULL)
```

Arguments:

- `limits`: Define resources limits.
- `reservation`: Define resources reservation.

`task_restart_policy` Specification for the restart policy which applies to containers created as part of this service.

Usage:

```
task_restart_policy(condition = NULL, delay = NULL,
    max_attempts = NULL, window = NULL)
```

Arguments:

- `condition`: Condition for restart.
- `delay`: Delay between restart attempts.
- `max_attempts`: Maximum attempts to restart a given container before giving up (default value is 0, which is ignored).
- `window`: Windows is the time window used to evaluate the restart policy (default value is 0, which is unbounded).

`task_spec` User modifiable task configuration.

Usage:

```
task_spec(container_spec = NULL, resources = NULL,
    restart_policy = NULL, placement = NULL,
    force_update = NULL, networks = NULL, log_driver = NULL)
```

Arguments:

- `container_spec`: Construct with `$types$container_spec()`
- `resources`: Resource requirements which apply to each individual container created as part of the service. Construct with `$types$task_resources()`
- `restart_policy`: Specification for the restart policy which applies to containers created as part of this service. Construct with `$types$task_restart_policy()`
- `placement`: Construct with `$types$task_placement()`
- `force_update`: A counter that triggers an update even if no relevant parameters have been changed.
- `networks`: NA
- `log_driver`: Specifies the log driver to use for tasks created from this spec. If not present, the default one for the swarm will be used, finally falling back to the engine default if not specified.

`docker_volume` *Management commands for working with a particular docker volume*

Description

Methods for working with a particular docker volume. Volume objects are returned by creating a docker volume, or by using `$volume$get` to fetch an existing volume by name or id.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`inspect` Return detailed information about this volume. Similar to the cli command `docker volume inspect`.

Usage:

```
inspect(reload = TRUE)
```

Arguments:

- `reload`: Logical scalar, indicating if the information should be refreshed from the daemon. This is useful to set to `FALSE`after an volume has been removed.

`map` Generate a path suitable to be passed in to `docker$container$run`or `docker$container$create`

Usage:

```
map(path, readonly = FALSE)
```

Arguments:

- `path`: Path in the container to map the volume to
- `readonly`: Logical, indicating if this should be a readonly mount

`name` Return name for this volume

Usage:

```
name()
```

`reload` Refresh infomation on the volume from the server, returning `$inspect()`invisibly.

Usage:

```
reload()
```

`volume_delete` Remove a volume. Instruct the driver to remove the volume. Similar to the cli command `docker volume rm`.

Usage:

```
volume_delete(force = NULL)
```

Arguments:

- `force`: Force the removal of the volume

See Also

`docker_volume_collection` for other volume management methods.

`docker_volume_collection` *Management commands for working with docker volumes*

Description

Methods for working with docker volumes. This object is `$volume` within a `docker_client` object.

Details

Below is reference documentation for all methods for version '1.29' of the docker API - other versions are available. This documentation is automatically generated from docker's API schema, and so inaccuracies may exist between it and stevedore's interface (especially references to JSON objects). Please report any documentation that might be improved at https://github.com/richfitz/stevedore/issues

Methods:

`volume_create` Create a volume. Similar to the cli command `docker volume create`.

Usage:

```
volume_create(name = NULL, driver = NULL, driver_opts = NULL,
    labels = NULL)
```

Arguments:

- `name`: The new volume's name. If not specified, Docker generates a name.
- `driver`: Name of the volume driver to use.
- `driver_opts`: A mapping of driver options and values. These options are passed directly to the driver and are driver specific.
- `labels`: User-defined key/value metadata.

`get` Get a volume by name or id

Usage:

```
get(name)
```

Arguments:

- `name`: A scalar character with the volume's name.

`help` Display help for this object

Usage:

```
help(help_type = getOption("help_type"))
```

Arguments:

- `help_type`: Passed to `utils::help`, can be one of "text", "html" or "pdf" (or an abbreviation). By default it uses the value `getOption("help_type")`and should follow the same behaviour as other R help (e.g., using "?")

`volume_list` List volumes. Similar to the cli command `docker volume ls`.

Usage:

```
volume_list(filters = NULL)
```

Arguments:

- `filters`: A named character vector of filters to process on the volumes list. Available filters:
  - `dangling=<boolean>`When set to `true`(or `1`), returns all volumes that are not in use by a container. When set to `false` (or `0`), only volumes that are in use by one or more containers are returned.
  - `driver=<volume-driver-name>`Matches volumes based on their driver.
  - `label=<key>`or `label=<key>:<value>`Matches volumes based on the presence of a `label`alone or a `label`and a value.
  - `name=<volume-name>`Matches all or part of a volume name.

`volume_prune` Delete unused volumes. Similar to the cli command `docker volume prune`.

Usage:

```
volume_prune(filters = NULL)
```

Arguments:

- `filters`: Filters to process on the prune list, as a named character vector.
  Available filters:

`volume_delete` Remove a volume. Instruct the driver to remove the volume. Similar to the cli command `docker volume rm`.

Usage:

```
volume_delete(name, force = NULL)
```

Arguments:

- `name`: Volume name or ID
- `force`: Force the removal of the volume

See Also

docker_volume for information on volume objects.

---

stevedore *Docker Client For R*

Description

stevedore implements a docker client for R. Docker is a framework for "containerisation" - abstracting the details of how software is installed and run. It is conceptually similar to virtualisation but much lighter weight.

Details

Within the R space containers have been discussed for:

Reproducible research: collecting all dependencies for an analysis into an image that can be run by other people without installation headaches.

Testing packages: Collect all the requirements of a package together and run your tests in an isolated environment.

Containers can also be used to construct larger applications with multiple processes that need to talk to each other - for example a database, API server and proxy server. One might also implement something like a set of shiny servers that are load balanced through a proxy!

This package provides a complete interface to docker allowing you to basically everything that can be done from the command line from within R. All communication happens over docker's HTTP API and does not use system commands. As a result, the information returned back to R is richer and the interface is likely to be reliable than parsing the command line output. stevedore's interface is largely automatically generated so will track new features available in the docker daemon closely.

The interface is designed to be similar to docker's command link API - the command for creating a network on the command line is

```
docker network create mynetwork
```

and in stevedore can be done as

```
docker <- stevedore::docker_client()
docker$network$create("mynetwork")
```

Familiarity with the command line interface will be helpful but probably as much because of the concepts as the details.

To get started, please see the package vignette - running vignette("stevedore") will work if the package was installed with the vignettes, or see https://richfitz.github.io/stevedore/. A good place to get started with the reference documentation is the docker_client function.

Index