Package 'stevedore'

November 12, 2018

Title Docker Client

Version 0.9.0

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 6.1.1

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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.

(automatic help generation has failed)

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

(automatic help generation has failed)

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

(automatic help generation has failed)

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