Package 'AnVIL'

October 16, 2020

Title Bioconductor on the AnVIL compute environment

Version 1.0.3

Description The AnVIL is cloud computing resource developed in part by the National Human Genome Research Institute. The AnVIL package provides end-user and devloper functionality. For the end-user, AnVIL provides fast binary package installation, utitlities for working with Terra / AnVIL table and data resources, and convenient functions for file movement to and from Google cloud storage. For developers, AnVIL provides programatic access to the Terra, Leonardo, Dockstore, and Gen3 RESTful programming interface, including helper functions to transform JSON responses to more formats more amenable to manipulation in R.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.0), dplyr

Imports stats, utils, methods, futile.logger, jsonlite, httr, curl, rapiclient (>= 0.1.3), tibble, BiocManager

Suggests knitr, testthat, withr, readr

Collate utilities.R authenticate.R Service.R Services.R Leonardo.R Terra.R Dockstore.R Gen3.R Response.R gcloud_sdk.R gcloud.R gsutil.R localize.R av.R zzz.R

LazyData true

VignetteBuilder knitr

biocViews Infrastructure

RoxygenNote 7.1.0

git_url https://git.bioconductor.org/packages/AnVIL

git_branch RELEASE_3_11

git_last_commit 5e06e75

git_last_commit_date 2020-05-03

Date/Publication 2020-10-16

Author Martin Morgan [aut, cre],
Nitesh Turaga [aut],
BJ Stubbs [ctb],
Vincent Carey [ctb],

Marcel Ramos [ctb],
Sweta Gopaulakrishnan [ctb],
Valerie Obenchain [ctb]

Maintainer Martin Morgan <mtmorgan.bioc@gmail.com>

R topics documented:

`avtables` *Functions for convenient user interaction with AnVIL resources*

Description

'avtables()' describes tables available in a workspace.

'avtable()' returns an AnVIL table.

'avtable_import()' imports a data.frame to an AnVIL table.

'avtable_delete_values()' removes rows from an AnVIL table.

'avdata()' returns the key-value table of 'REFERENCE DATA' and 'OTHER DATA' workspace attributes.

'avbucket()' retrieves the google bucket associated with a workspace.

'avworkspace_namespace()' and 'avworkspace_name()' are utiliity functions to retrieve workspace namespace and name from environment variables or interfaces available in AnVIL. Providing arguments to these functions over-rides AnVIL-determined settings with the provided value. Revert to system settings with arguments 'NA'.

Usage

```
avtables(namespace = avworkspace_namespace(), name = avworkspace_name())

avtable(table, namespace = avworkspace_namespace(), name = avworkspace_name())

avtable_import(
  .data,
  entity = names(.data)[[1]],
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avtable_delete_values(
  table,
```

```
  values,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avdata(namespace = avworkspace_namespace(), name = avworkspace_name())

avbucket(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  as_path = TRUE
)

avworkspace_namespace(namespace = NULL)

avworkspace_name(name = NULL)
```

Arguments

namespace	character(1) AnVIL workspace namespace as returned by, e.g., ‘avworkspace_namespace()‘
name	character(1) AnVIL workspace name as returned by, eg., ‘avworkspace_name()‘.
table	character(1) table name as returned by, e.g., ‘avtables()‘.
.data	A tibble or data.frame for import as an AnVIL table.
entity	‘character(1)‘ column name of ‘.data‘ to be used as imported table name. When the table comes from R, this is usually a column name such as ‘sample‘. The data will be imported into AnVIL as a table ‘sample‘, with the ‘sample‘ column included with suffix ‘_id‘, e.g., ‘sample_id‘. A column in ‘.data‘ with suffix ‘_id‘ can also be used, e.g., ‘entity = "sample_id"‘, creating the table ‘sample‘ with column ‘sample_id‘ in AnVIL. Finally, a value of ‘entity‘ that is not a column in ‘.data‘, e.g., ‘entity = "unknown"‘, will cause a new table with name ‘entity‘ and entity values ‘seq_len(nrow(.data))‘.
values	vector of values in the entity (key) column of ‘table‘ to be deleted. A table ‘sample‘ has an associated entity column with suffix ‘_id‘, e.g., ‘sample_id‘. Rows with entity column entries matching ‘values‘ are deleted.
as_path	logical(1) when TRUE (default) return bucket with prefix ‘gs://‘.

Value

‘avtables()‘: A tibble with columns identifying the table, the number of records, and the column names.

‘avtable()‘: a tibble of data corresponding to the AnVIL table ‘table‘ in the specified workspace.

‘avtable_import()‘ returns a ‘character(1)‘ name of the imported AnVIL tibble.

‘avtable_delete_values()‘ returns a ‘tibble‘ representing deleted entities, invisibly.

‘avdata()‘ returns a tibble with five columns: ‘"type"‘ represents the origin of the data from the ’REFERENCE’ or ’OTHER’ data menus. ‘"table"‘ is the table name in the ‘REFERENCE‘ menu, or ’workspace’ for the table in the ’OTHER’ menu, the key used to access the data element, the value label associated with the data element and the value (e.g., google bucket) of the element.

‘avbucket()‘ returns a ‘character(1)‘ bucket identifier, prefixed with ‘gs://‘ if ‘as_path = TRUE‘.

‘avworkspace_namespace()‘, and ‘avworkspace_name()‘ return ‘character(1)‘ identifiers.

Examples

```
if (gcloud_exists() && nzchar(avworkspace_namespace()))
    ## from within AnVIL
    avdata()

if (gcloud_exists() && nzchar(avworkspace_namespace()))
    ## From within AnVIL...
    bucket <- avbucket()                        # discover bucket

## Not run:
path <- file.path(bucket, "mtcars.tab")
gsutil_ls(dirname(path))                    # no 'mtcars.tab'...
write.table(mtcars, gsutil_pipe(path, "w")) # write to bucket
gsutil_stat(path)                           # yep, there!
read.table(gsutil_pipe(path, "r"))          # read from bucket

## End(Not run)
avworkspace_namespace()

avworkspace_name()
```

`gcloud` *Interact with the gcloud command line utility*

Description

These functions invoke the ‘gcloud‘ command line utility. See gsutil for details on how ‘gcloud‘ is located.

‘gcloud_exists()‘ tests whether the ‘gcloud()‘ command can be found on this system. See ’Details’ section of ‘gsutil‘ for where the application is searched.

‘gcloud_account()‘: report the current gcloud account via ‘gcloud config get-value account‘.

‘gcloud_project()‘: report the current gcloud project via ‘gcloud config get-value project‘.

‘gcloud_help()‘: queries ‘gcloud‘ for help for a command or sub-comand via ‘gcloud help ...‘.

‘gcloud_cmd()‘ allows arbitrary ‘gcloud‘ command execution via ‘gcloud ...‘. Use pre-defined functions in preference to this.

Usage

```
gcloud_exists()

gcloud_account(account = NULL)

gcloud_project(project = NULL)

gcloud_help(...)

gcloud_cmd(cmd, ...)
```

Arguments

account	character(1) Google account (e.g., 'user@gmail.com') to use for authentication.
project	character(1) billing project name.
...	additional arguments appended to gcloud commands.
cmd	'character(1)' representing a command used to evaluate 'gcloud cmd ...'.

Value

'gcloud_exists()' returns 'TRUE' when the 'gcloud' application can be found, FALSE otherwise.

'gcloud_account()' returns a 'character(1)' vector containing the active gcloud account, typically a gmail email address.

'gcloud_project()' returns a 'character(1)' vector containing the active gcloud project.

'gcloud_help()' returns an unquoted 'character()' vector representing the text of the help manual page returned by 'gcloud help ...'.

'gcloud_cmd()' returns a 'character()' vector representing the text of the output of 'gcloud cmd ...'

Examples

```
gcloud_exists()

if (gcloud_exists())
    gcloud_account()

if (gcloud_exists())
    gcloud_help()
```

---

gsutil *Interact with the gsutil command line utility*

Description

These functions invoke the 'gsutil' command line utility. See the "Details:" section if you have gsutil installed but the package cannot find it.

'gsutil_requesterpays()': does the google bucket require that the requester pay for access?

'gsutil_ls()': List contents of a google cloud bucket or, if 'source' is missing, all Cloud Storage buckets under your default project ID

'gsutil_exists()': check if the bucket or object exists.

'gsutil_stat()': print, as a side effect, the status of a bucket, directory, or file.

'gsutil_cp()': copy contents of 'source' to 'destination'. At least one of 'source' or 'destination' must be Google cloud bucket; 'source' can be a character vector with length greater than 1. Use 'gsutil_help("cp")' for 'gsutil' help.

'gsutil_rm()': remove contents of a google cloud bucket.

'gsutil_rsync()': synchronize a source and a destination.

'gsutil_cat()': concatenate bucket objects to standard output

'gsutil_help()': print 'man' page for the 'gsutil' command or subcommand. Note that only commandes documented on this R help page are supported.

'gsutil_pipe()': create a pipe to read from or write to a gooogle bucket object.

Usage

```
gsutil_requesterpays(source)

gsutil_ls(source = character(), ..., recursive = FALSE)

gsutil_exists(source)

gsutil_stat(source)

gsutil_cp(source, destination, ..., recursive = FALSE, parallel = TRUE)

gsutil_rm(source, ..., force = FALSE, recursive = FALSE, parallel = TRUE)

gsutil_rsync(
  source,
  destination,
  ...,
  dry = TRUE,
  delete = FALSE,
  recursive = FALSE,
  parallel = TRUE
)

gsutil_cat(source, ..., header = FALSE, range = integer())

gsutil_help(cmd = character(0))

gsutil_pipe(source, open = "r", ...)
```

Arguments

source	'character(1)', ('character()' for 'gsutil_requesterpays()', 'gsutil_ls()', 'gsutil_exists()', 'gsutil_cp()') paths to a google storage bucket, possibly with wild-cards for file-level pattern matching.
...	additional arguments passed as-is to the 'gsutil' subcommand.
recursive	'logical(1)'; perform operation recursively from 'source'?. Default: 'FALSE'.
destination	'character(1)', google cloud bucket or local file system destination path.
parallel	'logical(1)', perform parallel multi-threaded / multi-processing (default is 'TRUE').
force	'logical(1)': continue silently despite errors when removing multiple objects. Default: 'FALSE'.
dry	'logical(1)', when 'TRUE' (default), return the consequences of the operation without actually performing the operation.
delete	'logical(1)', when 'TRUE', remove files in 'destination' that are not in 'source'. Exercise caution when you use this option: it's possible to delete large amounts of data accidentally if, for example, you erroneously reverse source and destination.
header	'logical(1)' when 'TRUE' annotate each
range	(optional) 'integer(2)' vector used to form a range from-to of bytes to concatenate. 'NA' values signify concatenation from the start (first position) or to the end (second position) of the file.

cmd	‘character()‘ (optional) command name, e.g., ‘"ls"‘ for help.
open	‘character(1)‘ either ‘"r"‘ (read) or ‘"w"‘ (write) from the bucket.

Details

The ‘gsutil‘ system command is required. The search for ‘gsutil‘ starts with environment variable ‘GCLOUD_SDK_PATH‘ providing a path to a directory containing a ‘bin‘ directory containingin ‘gsutil‘, ‘gcloud‘, etc. The path variable is searched for first as an ‘option()‘ and then system variable. If no option or global variable is found, ‘Sys.which()‘ is tried. If that fails, ‘gsutil‘ is searched for on defined paths. On Windows, the search tries to find ‘Google\Cloud SDK\google-cloud-sdk\bin\gsutil.cmd‘ in the ‘LOCAL APP DATA‘, ‘Program Files‘, and ‘Program Files (x86)‘ directories. On linux / macOS, the search continues with ‘~/google-cloud-sdk‘.

‘gsutil_rsync()’: To make ‘"gs://mybucket/data"‘ match the contents of the local directory ‘"data"‘ you could do:

`gsutil_rsync("data","gs://mybucket/data",delete = TRUE)`

To make the local directory "data" the same as the contents of gs://mybucket/data:

`gsutil_rsync("gs://mybucket/data","data",delete = TRUE)`

If ‘destination‘ is a local path and does not exist, it will be created.

Value

‘gsutil_requesterpays()‘: named ‘logical()‘ vector TRUE when requester-pays is enabled.

‘gsutil_ls()‘: ‘character()‘ listing of ‘source‘ content.

‘gsutil_exists()‘: logical(1) TRUE if bucket or object exists.

‘gsutil_stat()‘: ‘character()‘ description of status of objects matching ‘source‘.

‘gsutil_cp()‘: exit status of ‘gsutil_cp()‘, invisibly.

‘gsutil_rm()‘: exit status of ‘gsutil_rm()‘, invisibly.

‘gsutil_rsync()‘: exit status of ‘gsutil_rsync()‘, invisbly.

‘gsutil_cat()‘ returns the content as a character vector.

‘gsutil_help()‘: ‘character()‘ help text for subcommand ‘cmd‘.

‘gsutil_pipe()‘ an unopened R ‘pipe()‘; the mode is *not* specified, and the pipe must be used in the appropriate context (e.g., a pipe created with ‘open = "r"‘ for input as ‘read.csv()‘)

Examples

```
    src <- "gs://genomics-public-data/1000-genomes/other/sample_info/sample_info.csv"
if (gcloud_exists())
    gsutil_requesterpays(src) # FALSE -- no cost download

if (gcloud_exists()) {
    gsutil_exists(src)
    gsutil_stat(src)
    gsutil_ls(dirname(src))
}

if (gcloud_exists())
   gsutil_cp(src, tempdir())

if (gcloud_exists())
    gsutil_help("ls")
```

```
if (gcloud_exists()) {
    df <- read.csv(gsutil_pipe(src), 5L)
    class(df)
    dim(df)
    head(df)
}
```

`localize` *Copy packages, folders, or files to or from google buckets.*

Description

'localize()': recursively synchronizes files from a Google storage bucket ('source') to the local file system ('destination'). This command acts recursively on the 'source' directory, and does not delete files in 'destination' that are not in 'source.

'delocalize()': synchronize files from a local file system ('source') to a Google storage bucket ('destination'). This command acts recursively on the 'source' directory, and does not delete files in 'destination' that are not in 'source'.

'add_libpaths()': Add local library paths to '.libPaths()'.

'install()': install R / Bioconductor packages, using fast pre-built 'binary' libraries if available.

Usage

```
localize(source, destination, dry = TRUE)

delocalize(source, destination, unlink = FALSE, dry = TRUE)

add_libpaths(paths)

install(
  pkgs,
  lib = .libPaths()[1],
  ...,
  binary_base_url = "https://storage.googleapis.com",
  verbose = getOption("verbose")
)
```

Arguments

`source`	'character(1)', a google storage bucket or local file system directory location.
`destination`	'character(1)', a google storage bucket or local file system directory location.
`dry`	'logical(1)', when 'TRUE' (default), return the consequences of the operation without actually performing the operation.
`unlink`	'logical(1)' remove (unlink) the file or directory in 'source'. Default: 'FALSE'.
`paths`	'character()': vector of directories to add to '.libPaths()'. Paths that do not exist will be created.
`pkgs`	'character()' packages to install from binary repository.

lib	'character(1)' library path (directory) in which to install 'pkgs'; defaults to '.libPaths()[1]'.
...	additional arguments, passed to 'install.packages()'.
binary_base_url	'character(1)' host and base path for binary package 'CRAN-style' repository; not usually required by the end-user.
verbose	'logical(1)' report on package installation progress?

Details

'install()' prepends an additional repository URI to 'BiocManager::repositories()'. The URI is formed by concatenating 'binary_base_url', the environment variables 'TERRA_R_PLATFORM' and the 'major' and 'minor' components of 'TERRA_R_PLATFORM_BINARY_VERSION' and 'BiocManager::version()'. The URI is only prepended if a CRAN-style repostiory exists at that location, with binary package tar.gz content described by 'src/contrib/PACKAGES'.

Value

'localize()': exit status of function 'gsutil_rsync()'.

'delocalize()': exit status of function 'gsutil_rsync()'

'add_libpaths()': updated .libPaths(), invisibly.

'install()': return value of 'install.packages()'.

Examples

```
## Not run:
add_libpaths("/tmp/host-site-library")
install(packages = c('BiocParallel', 'BiocGenerics'))

## End(Not run)
```

Response *Process service responses to tibble and other data structures.*

Description

Process service responses to tibble and other data structures.

Usage

```
flatten(x)

## S4 method for signature 'response'
str(object)

## S3 method for class 'response'
as.list(x, ..., as = c("text", "raw", "parsed"))
```

Arguments

x	A ‘response‘ object returned by the service.
object	A ‘response‘ object returned by the service.
...	not currently used
as	character(1); one of ’raw’, ’text’, ’parsed’

Value

‘flatten()‘ returns a ‘tibble‘ where each row correseponds to a top-level list element of the return value, and columns are the unlisted second and more-nested elements.

‘str()‘ displays a compact representation of the list-like JSON response; it returns ‘NULL‘.

‘as.list()‘ retruns the content of the web service request as a list.

Examples

```
if (gcloud_exists()) {
    leonardo <- Leonardo()
    leonardo$listClusters() %>% flatten()
}

if (gcloud_exists())
   leonardo$getSystemStatus() %>% str()

if (gcloud_exists())
    leonardo$getSystemStatus() %>% as.list()
```

Service *RESTful service constructor*

Description

RESTful service constructor

Usage

```
Service(
  service,
  host,
  config = httr::config(),
  authenticate = TRUE,
  api_url = character(),
  package = "AnVIL",
  schemes = "https"
)
```

Arguments

service	character(1) The ‘Service‘ class name, e.g., ‘"terra"‘.
host	character(1) host name that provides the API resource, e.g., ‘"api.firecloud.org"‘.
config	httr::config() curl options
authenticate	logical(1) use credentials from authentication service file 'auth.json' in the specified package?
api_url	optional character(1) url location of OpenAPI ‘.json‘ or ‘.yaml‘ service definition.
package	character(1) (default ‘AnVIL‘) The package where 'api.json' yaml and (optionally) 'auth.json' files are located
schemes	character(1) (default 'https') Specifies the transfer protocol supported by the API service

Details

This function creates a RESTful interface to a service provided by a host, e.g., "api.firecloud.org". The function requires an OpenAPI ‘.json‘ or ‘.yaml‘ specifcation as well as an (optional) ‘.json‘ authentication token. These files are located in the source directory of a pacakge, at ‘<package>/inst/service/<service>/api.json‘ and ‘<package>/inst/service/<service>/auth.json‘, or at ‘api_url‘.

The service is usually a singleton, created at the package level during ‘.onLoad()‘.

Value

An object of class `Service`.

Examples

```
.MyService <- setClass("MyService", contains = "Service")

MyService <- function() {
    .MyService(Service("my_service", host="my.api.org"))
}
```

Services *RESTful services useful for AnVIL developers*

Description

RESTful services useful for AnVIL developers

Usage

```
empty_object

operations(x, ...)

schemas(x)
```

```
tags(x, .tags)

## S4 method for signature 'Service'
x$name

Leonardo()

Terra()

Dockstore()

Gen3Fence()

Gen3Indexd()

Gen3Sheepdog()

Gen3Peregrine()
```

Arguments

x	A 'Service' instance, usually a singleton provided by the package and documented on this page, e.g., 'leonardo' or 'terra'.
...	additional arguments passed to methods or, for 'operations,Service-method', to the internal 'get_operation()' function.
.tags	optional character() of tags to use to filter operations.
name	A symbol representing a defined operation, e.g., 'leonardo$listClusters()'.

Value

'empty_object' returns a representation to be used as arguments in function calls expecting the empty json object '{}'.

'Leonardo()' creates the API of the Leonard container deployment service at https://leonardo.dev.anvilproject.org/.

'Terra()' creates the API of the Terra cloud computational environemnt at https://api.firecloud.org/.

'Dockstore()' represents the API of the Dockstore platform to share Docker-based tools in CWL or WDL or Nextflow at https://dockstore.org

'gen3_*' APIs are not fully implemented, because a service endpoint has not been identifiied.

'Gen3Fence()' returns the authentication API at https://raw.githubusercontent.com/uc-cdis/fence/master/openapis/swagg

'Gen3Indexd()' returns the indexing service API documented at https://raw.githubusercontent.com/uc-cdis/indexd/master/openapis/swagger.yaml

'Gen3Sheepdog' returns the submission services API at https://raw.githubusercontent.com/uc-cdis/sheepdog/master/oper

'Gen3Peregrine' returns the graphQL query services API at https://raw.githubusercontent.com/uc-cdis/peregrine/master/openapis/swagger.yaml

Examples

```
empty_object

if (gcloud_exists()) {
    terra <- Terra()
```

```
    tags(terra)
    tags(terra, "Billing")
}
```

Index