Package 'AnVIL'

July 13, 2025

Title Bioconductor on the AnVIL compute environment

Version 1.21.7

Description The AnVIL is a cloud computing resource developed in part by the National Human Genome Research Institute. The AnVIL package provides programatic access to the Dockstore, Leonardo, Rawls, TDR, and Terra RESTful programming interfaces. For platform-specific user-level functionality, see either the AnVILGCP or AnVILAz package.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.5.0), dplyr, AnVILBase

Imports stats, utils, methods, futile.logger, GCPtools, jsonlite, httr, rapiclient, yaml, tibble, shiny, DT, miniUI, htmltools, BiocBaseUtils

Suggests knitr, rmarkdown, testthat, withr, readr, BiocStyle, devtools, AnVILAz, AnVILGCP, lifecycle

Collate utilities.R authenticate.R api.R AnVIL-package.R Service.R Services.R Leonardo.R Terra.R Rawls.R Dockstore.R TDR.R gadgets.R zzz.R

VignetteBuilder knitr

biocViews Infrastructure

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Date 2025-05-22

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

git_branch devel

git_last_commit c53c254

git_last_commit_date 2025-07-10

Repository Bioconductor 3.22

Date/Publication 2025-07-13

Author Marcel Ramos [aut, cre] (ORCID: <https://orcid.org/0000-0002-3242-0582>), Martin Morgan [aut] (ORCID: <https://orcid.org/0000-0002-5874-8148>), Kayla Interdonato [aut], Yubo Cheng [aut],

Nitesh Turaga [aut],
BJ Stubbs [ctb],
Vincent Carey [ctb],
Sehyun Oh [ctb],
Sweta Gopaulakrishnan [ctb],
Valerie Obenchain [ctb]

Maintainer Marcel Ramos <marcel.ramos@sph.cuny.edu>

Contents

`AnVIL-package` *AnVIL: Bioconductor on the AnVIL compute environment*

Description

The AnVIL is a cloud computing resource developed in part by the National Human Genome Research Institute. The AnVIL package provides end-user and developer 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, Rawls, and Dockstore RESTful programming interface, including helper functions to transform JSON responses to formats more amenable to manipulation in R.

Author(s)

Maintainer: Marcel Ramos <marcel.ramos@sph.cuny.edu> (ORCID)

Authors:

- Martin Morgan (ORCID)
- Kayla Interdonato
- Yubo Cheng
- Nitesh Turaga

Other contributors:

- BJ Stubbs [contributor]
- Vincent Carey [contributor]
- Sehyun Oh [contributor]
- Sweta Gopaulakrishnan [contributor]
- Valerie Obenchain [contributor]

.gadget_run *Functions to implement AnVIL gadget interfaces*

Description

Functions documented on this page are primarily intended for package developers wishing to implement gadgets (graphical interfaces) to navigating AnVIL-generated tables.

`.gadget_run()` presents the user with a tibble-navigating gadget, returning the value of `DONE_FUN` if a row of the tibble is selected, or NULL.

Usage

```
.gadget_run(title, tibble, DONE_FUN)
```

Arguments

`title`	character(1) (required) title to appear at the base of the gadget, e.g., "AnVIL Workspaces".
`tibble`	a `tibble` or `data.frame` to be displayed in the gadget.
`DONE_FUN`	a function of two arguments, `tibble` and `row_selected`. The tibble is the `tibble` provided as an argument to `.gadget_run()`. `row_selected` is the row selected in the gadget by the user. The function is only invoked when the user selects a valid row.

Value

`.gadget_run()` returns the result of `DONE_FUN()` if a row has been selected by the user, or `NULL` if no row is selected (the user presses `Cancel`, or `Done` prior to selecting any row).

Examples

```
## Not run:
tibble <- avworkspaces()
DONE_FUN <- function(tibble, row_selected) {
    selected <- slice(tibble, row_selected)
    with(selected, paste0(namespace, "/", name))
}
.gadget_run("AnVIL Example", tibble, DONE_FUN)

## End(Not run)
```

avworkspace_gadget *Graphical user interfaces for common AnVIL operations*

Description

workspace() allows choice of workspace for subsequent use. It is the equivalent of displaying workspaces with avworkspaces(), and setting the selected workspace with avworkspace().

browse_workspace() uses browseURL() to open a browser window pointing to the Terra workspace.

table() allows choice of table in the current workspace (selected by avworkspace() or workspace()) to be returned as a tibble. It is equivalent to invoking avtables() to show available tables, and avtable() to retrieve the selected table.

workflow() allows choice of workflow for retrieval. It is the equivalent of avworkflows() for listing available workflows, and avworkflow_configuration_get() for retrieving the workflow.

Usage

```
avworkspace_gadget()

browse_workspace(use_avworkspace = TRUE)

avtable_gadget()

avworkflow_gadget()
```

Arguments

use_avworkspace
: logical(1) when TRUE (default), use the selected workspace (via workspace() or avworkspace() if available. If FALSE or no workspace is currently selected, use workspace() to allow the user to select the workspace.

Value

workspace() returns the selected workspace as a character(1) using the format namespace/name, or character(0) if no workspace is selected.

browse_workspace() returns the status of a system() call to launch the browser, invisibly.

table() returns a tibble representing the selected AnVIL table.

workflow() returns an avworkflow_configuration object representing the inputs and outputs of the selected workflow. This can be edited and updated as described in the "Running an AnVIL workflow within R" vigenette.

Examples

```
## Not run:
workspace()
browse_workspace(use_avworkspace = FALSE)
tbl <- table()
wkflw <- avworkflow_gadget()

## End(Not run)
```

`Service` *RESTful service constructor*

Description

RESTful service constructor

Usage

```
Service(
  service,
  host,
  config = httr::config(),
  authenticate = TRUE,
  api_url = character(),
  package = "AnVIL",
  schemes = "https",
  api_reference_url = api_url,
  api_reference_md5sum = character(),
  api_reference_version = character(),
  api_reference_headers = NULL,
  ...
)
```

Arguments

`service`	character(1) The `Service` class name, e.g., `"terra"`.
`host`	character(1) host name that provides the API resource, e.g., `"leonardo.dsde-prod.broadinstitute`
`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.
`api_reference_url`	character(1) path to reference API. See Details.
`api_reference_md5sum`	character(1) the result of `tools::md5sum()` applied to the reference API.
`api_reference_version`	character(1) the version of the reference API. This is used to check that the version of the service matches the version of the reference API. It is usally set by the service generation function,. e.g., `AnVIL::Rawls()`.
`api_reference_headers`	character() header(s) to be used (e.g., `c(Authorization = paste("Bearer", token))`) when retrieving the API reference for validation.
`...`	additional arguments passed to `rapiclient::get_api()`

Details

This function creates a RESTful interface to a service provided by a host, e.g., "leonardo.dsde-prod.broadinstitute.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`.

When provided, the `api_reference_md5sum` is used to check that the file described at `api_reference_url` has the same checksum as an author-validated version.

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, ..., .deprecated = FALSE)

## S4 method for signature 'Service'
operations(x, ..., auto_unbox = FALSE, .deprecated = FALSE)

schemas(x)

tags(x, .tags, .deprecated = FALSE)

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

Leonardo()

Terra()

Rawls()
```

`Dockstore()`

`TDR()`

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.
`.deprecated`	optional logical(1) include deprecated operations?
`auto_unbox`	logical(1) If FALSE (default) do not automatically 'unbox' R scalar values from JSON arrays to JSON scalers.
`.tags`	optional character() of tags to use to filter operations.
`name`	A symbol representing a defined operation, e.g., `leonardo$listRuntimes()`.

Details

Note the services `Terra()`, `Rawls()`, and `Leonardo()` require the `GCPtools` package for authentication to the Google Cloud Platform. See `?GCPtools::gcloud_access_token()` for details.

When using `$` to select a service, some arguments appear in 'body' of the REST request. Specify these using the `.__body__=` argument, as illustrated for `createBillingProjectFull()`, below.

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 Leonardo container deployment service at https://leonardo.dsde-prod.broadinstitute.org/api-docs.yaml.

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

`Rawls()` creates the API of the Rawls cloud computational environemnt at https://rawls.dsde-prod.broadinstitute.org.

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

`TDR()` creates the API of the Terra Data Repository to work with snapshot data in the Terra Data Repository at https://data.terra.bio.

Examples

```
empty_object

library(GCPtools)
if (gcloud_exists()) {
    ## Arguments to be used as the 'body' (`.__body__=`) of a REST query
    Terra()$createBillingProjectFull       # 6 arguments...
    args(Terra()$createBillingProjectFull) # ... passed as `.__body__ = list(...)`
}
library(GCPtools)
if (gcloud_exists())
```

```
    Leonardo()

library(GCPtools)
if (gcloud_exists()) {
    tags(Terra())
    tags(Terra(), "Billing")
}

library(GCPtools)
if (gcloud_exists()) {
    tags(Rawls())
    tags(Rawls(), "billing")
}

Dockstore()

library(GCPtools)
if (gcloud_exists())
    TDR()
```

`utilities` *Utilities for managing library paths*

Description

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

Usage

```
add_libpaths(paths)
```

Arguments

`paths`	`character()`: vector of directories to add to `.libPaths()`. Paths that do not exist will be created.

Value

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

Examples

```
## Not run: add_libpaths("/tmp/host-site-library")
```

Index