Package 'sixtyfour'

March 31, 2025

Title Humane Interface to Amazon Web Services

Version 0.2.0

Description An opinionated interface to Amazon Web Services <https://aws.amazon.com>, with functions for interacting with 'IAM' (Identity and Access Management), 'S3' (Simple Storage Service), 'RDS' (Relational Data Service), Redshift, and Billing. Lower level functions ('aws_' prefix) are for do it yourself workflows, while higher level functions ('six_' prefix) automate common tasks.

URL https://github.com/getwilds/sixtyfour, https://getwilds.org/sixtyfour/

BugReports https://github.com/getwilds/sixtyfour/issues

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

LazyData true

Depends R (>= 4.1.0)

Imports dplyr, paws (>= 0.9.0), paws.common (>= 0.8.1), purrr, rlang, tibble, fs, s3fs (>= 0.1.5), cli, glue, memoise, uuid, jsonlite, curl, tidyr, clipr, withr, ipaddress

Suggests knitr, rmarkdown, roxyglobals, DBI, RPostgres, RMariaDB, ggplot2, lubridate, testthat (>= 3.0.0), vcr (>= 0.6.0), webmockr

Config/roxyglobals/filename globals.R

Config/roxyglobals/unique FALSE

Config/testthat/edition 3

NeedsCompilation no

Author Sean Kross [aut], Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>), Fred Hutchinson Cancer Center [fnd] (007ps6h72)

Maintainer Scott Chamberlain <sachamber@fredhutch.org>

Repository CRAN

Date/Publication 2025-03-31 16:40:02 UTC

Contents

`as_policy_arn` *Convert a policy name to a policy ARN*

Description

This function simply constructs a string. It only makes an HTTP request if `local=TRUE` and environment variable `AWS_PROFILE` != "localstack"

Usage

```
as_policy_arn(name, local = FALSE, path = NULL)
```

Arguments

`name`	(character) a policy name or arn
`local`	(logical) if `TRUE` use your AWS account for your own managed policies. If `FALSE`, AWS managed policies
`path`	(character) if not `NULL`, we add the path into the ARN before the `name` value

Value

a policy ARN (character)

See Also

Other policies: `aws_policies()`, `aws_policy()`, `aws_policy_attach()`, `aws_policy_create()`, `aws_policy_delete()`, `aws_policy_delete_version()`, `aws_policy_detach()`, `aws_policy_exists()`, `aws_policy_list_entities()`, `aws_policy_list_versions()`, `aws_policy_update()`

Examples

```
as_policy_arn("ReadOnlyAccess")
as_policy_arn("arn:aws:iam::aws:policy/ReadOnlyAccess")
as_policy_arn("AmazonRDSDataFullAccess")

# path = Job function
as_policy_arn("Billing", path = "job-function")

# path = Service role
as_policy_arn("AWSCostAndUsageReportAutomationPolicy",
  path = "service-role"
)


as_policy_arn("MyTestPolicy", local = TRUE)
# returns an arn - and if given an arn returns self
as_policy_arn("MyTestPolicy", local = TRUE) %>%
  as_policy_arn()
```

`aws_billing` *Fetch billing data - with some internal munging for ease of use*

Description

Fetch billing data - with some internal munging for ease of use

Usage

```
aws_billing(date_start, date_end = as.character(Sys.Date()), filter = NULL)
```

Arguments

`date_start`, `date_end`	Start and end date to get billing data for. Date format expected: `yyyy-MM-dd`. required
`filter`	(list) filters costs by different dimensions. optional.

Value

tibble with columns:

- id: "blended", "unblended"
- date: date, in format `yyyy-MM-dd`
- service: AWS service name, spelled out in full
- linked_account: account number
- cost: cost in USD
- acronym: short code for the service; if none known, this row will have the value in `service`

Blended vs. Unblended

- Unblended: Unblended costs represent your usage costs on the day they are charged to you
- Blended: Blended costs are calculated by multiplying each account's service usage against something called a blended rate. A blended rate is the average rate of on-demand usage, as well as Savings Plans- and reservation-related usage, that is consumed by member accounts in an organization for a particular service.

Historical data

If you supply a `date_start` older than 14 months prior to today's date you will likely see an error like "You haven't enabled historical data beyond 14 months". See https://docs.aws.amazon.com/cost-management/latest/userguide/ce-advanced-cost-analysis.html #nolint for help

Filtering

You can optionally pass a list to the `filter` argument to filter AWS costs by different dimensions, tags, or cost categories. This filter expression is passed on to paws. See possible dimensions: https://docs.aws.amazon.com/aws-cost-management/latest/APIReference/API_GetDimensionValues.html) #nolint

This is supplied as a list, with key-value pairs for each criteria. Different filter criteria can be combined in different ways using `AND`, `OR`, and `NOT`. See Examples below and more on Filter expressions at https://docs.aws.amazon.com/aws-cost-management/latest/APIReference/API_Expression.html. #nolint

References

https://www.paws-r-sdk.com/docs/costexplorer/

See Also

Other billing: `aws_billing_raw()`

Examples

```
library(lubridate)
library(dplyr)

start_date <- today() - months(13)
z <- aws_billing(date_start = start_date)
z %>%
  filter(id == "blended") %>%
  group_by(service) %>%
  summarise(sum_cost = sum(cost)) %>%
  filter(sum_cost > 0) %>%
  arrange(desc(sum_cost))

z %>%
  filter(id == "blended") %>%
  filter(cost > 0) %>%
  arrange(service)
```

```
z %>%
  filter(id == "blended") %>%
  group_by(service) %>%
  summarise(sum_cost = sum(cost)) %>%
  filter(service == "Amazon Relational Database Service")

# Simple filter to return only "Usage" costs:
aws_billing(
  date_start = start_date,
  filter = list(
    Dimensions = list(
      Key = "RECORD_TYPE",
      Values = "Usage"
    )
  )
)

# Filter to return "Usage" costs for only m4.xlarge instances:
aws_billing(
  date_start = start_date,
  filter = list(
    And = list(
      list(
        Dimensions = list(
          Key = "RECORD_TYPE",
          Values = list("Usage")
        )
      ),
      list(
        Dimensions = list(
          Key = "INSTANCE_TYPE",
          Values = list("m4.xlarge")
        )
      )
    )
  )
)

# Complex filter example, translated from the AWS Cost Explorer docs:
# <https://docs.aws.amazon.com/aws-cost-management/latest/APIReference/API_Expression.html> #nolint
# Filter for operations within us-west-1 or us-west-2 regions OR have a
# specific Tag value, AND are NOT DataTransfer usage types:
aws_billing(
  date_start = start_date,
  filter = list(
    And = list(
      list(
        Or = list(
          list(
            Dimensions = list(
              Key = "REGION",
              Values = list("us-east-1", "us-west-1")
```

```
            )
          ),
          list(
            Tags = list(
              Key = "TagName",
              Values = list("Value1")
            )
          )
        )
      ),
      list(
        Not = list(
          Dimensions = list(
            Key = "USAGE_TYPE",
            Values = list("DataTransfer")
          )
        )
      )
    )
  )
)
```

---

aws_billing_raw *Fetch billing data - rawest form*

Description

Fetch billing data - rawest form

Usage

```
aws_billing_raw(
  date_start,
  metrics,
  granularity = "daily",
  filter = NULL,
  group_by = NULL,
  date_end = as.character(Sys.Date())
)
```

Arguments

date_start, date_end	Start and end date to get billing data for. Date format expected: yyyy-MM-dd. required
metrics	(character) which metrics to return. required. One of: AmortizedCost, BlendedCost, NetAmortizedCost, NetUnblendedCost, NormalizedUsageAmount, UnblendedCost, and UsageQuantity

granularity	(character) monthly, daily, hourly. required.
filter	(list) filters costs by different dimensions. optional.
group_by	(list) group costs using up to two different groups, either dimensions, tag keys, cost categories, or any two group by types. optional.

Value

list with slots for:

- NextPageToken
- GroupDefinitions
- ResultsByTime
- DimensionValueAttributes

See Also

Other billing: aws_billing()

Examples

```
library(lubridate)
aws_billing_raw(date_start = today() - days(3), metrics = "BlendedCost")
```

---

aws_buckets *List S3 buckets*

Description

List S3 buckets

Usage

```
aws_buckets(...)
```

Arguments

...	named parameters passed on to list_objects

Details

internally uses s3fs::s3_dir_info()

Value

if no objects found, an empty tibble. if tibble has rows each is an S3 bucket, with 8 columns:

- bucket_name (character)
- key (character)
- uri (character)
- size (fs::bytes)
- type (character)
- owner (character)
- etag (character)
- last_modified (dttm)

Note

we set `refresh=TRUE` internally to make sure we return up to date information about your buckets rather than what's cached locally

See Also

Other buckets: `aws_bucket_create()`, `aws_bucket_delete()`, `aws_bucket_download()`, `aws_bucket_exists()`, `aws_bucket_list_objects()`, `aws_bucket_tree()`, `aws_bucket_upload()`, `six_bucket_delete()`, `six_bucket_upload()`

Examples

```
aws_buckets()
```

`aws_bucket_create` *Create an S3 bucket*

Description

Create an S3 bucket

Usage

```
aws_bucket_create(bucket, ...)
```

Arguments

`bucket`	(character) bucket name. required
`...`	named parameters passed on to create_bucket

Value

the bucket path (character)

Note

Requires the env var AWS_REGION

See Also

Other buckets: aws_bucket_delete(), aws_bucket_download(), aws_bucket_exists(), aws_bucket_list_objects(), aws_bucket_tree(), aws_bucket_upload(), aws_buckets(), six_bucket_delete(), six_bucket_upload()

Examples

```
bucket2 <- random_bucket()
aws_bucket_create(bucket2)

# cleanup
six_bucket_delete(bucket2, force = TRUE)
```

---

aws_bucket_delete *Delete an S3 bucket*

Description

Delete an S3 bucket

Usage

```
aws_bucket_delete(bucket, force = FALSE, ...)
```

Arguments

bucket	(character) bucket name. required
force	(logical) force deletion without going through the prompt. default: FALSE. Should only be set to TRUE when required for non-interactive use.
...	named parameters passed on to delete_bucket

Value

NULL, invisibly

Note

Requires the env var AWS_REGION. This function prompts you to make sure that you want to delete the bucket.

See Also

Other buckets: aws_bucket_create(), aws_bucket_download(), aws_bucket_exists(), aws_bucket_list_objects(), aws_bucket_tree(), aws_bucket_upload(), aws_buckets(), six_bucket_delete(), six_bucket_upload()

Examples

```
bucket_name <- random_bucket()
if (!aws_bucket_exists(bucket_name)) {
  aws_bucket_create(bucket = bucket_name)
  aws_buckets()
  aws_bucket_delete(bucket = bucket_name, force = TRUE)
  aws_buckets()
}
```

---

aws_bucket_download *Download an S3 bucket*

Description

Download an S3 bucket

Usage

```
aws_bucket_download(bucket, dest_path, ...)
```

Arguments

bucket	(character) bucket name. required
dest_path	(character) destination directory to store files. required
...	named parameters passed on to s3fs::s3_dir_download()

Value

path (character) to downloaded file(s)/directory

Note

Requires the env var AWS_REGION. This function prompts you to make sure that you want to delete the bucket.

See Also

Other buckets: aws_bucket_create(), aws_bucket_delete(), aws_bucket_exists(), aws_bucket_list_objects(), aws_bucket_tree(), aws_bucket_upload(), aws_buckets(), six_bucket_delete(), six_bucket_upload()

Examples

```
bucket <- random_bucket()
aws_bucket_create(bucket = bucket)
desc_file <- file.path(system.file(), "DESCRIPTION")
aws_file_upload(desc_file, s3_path(bucket, "DESCRIPTION.txt"))
aws_file_upload(desc_file, s3_path(bucket, "d_file.txt"))
temp_dir <- file.path(tempdir(), bucket)
aws_bucket_download(bucket = bucket, dest_path = temp_dir)
fs::dir_ls(temp_dir)

# cleanup
six_bucket_delete(bucket, force = TRUE)
```

---

aws_bucket_exists *Check if an S3 bucket exists*

Description

Check if an S3 bucket exists

Usage

```
aws_bucket_exists(bucket)
```

Arguments

bucket	(character) bucket name; must be length 1. required

Value

a single boolean (logical)

Note

internally uses head_bucket

See Also

Other buckets: aws_bucket_create(), aws_bucket_delete(), aws_bucket_download(), aws_bucket_list_objects(), aws_bucket_tree(), aws_bucket_upload(), aws_buckets(), six_bucket_delete(), six_bucket_upload()

Examples

```
bucket1 <- random_bucket()
aws_bucket_create(bucket1)

# exists
aws_bucket_exists(bucket = bucket1)
# does not exist
aws_bucket_exists(bucket = "no-bucket")

# cleanup
six_bucket_delete(bucket1, force = TRUE)
```

`aws_bucket_list_objects` *List objects in an S3 bucket*

Description

List objects in an S3 bucket

Usage

```
aws_bucket_list_objects(bucket, ...)
```

Arguments

`bucket`	(character) bucket name. required
`...`	named parameters passed on to list_objects

Value

if no objects found, an empty tibble. if tibble has rows each is an S3 bucket, with 8 columns:

- bucket_name (character)
- key (character)
- uri (character)
- size (fs::bytes)
- type (character)
- owner (character)
- etag (character)
- last_modified (dttm)

See Also

Other buckets: `aws_bucket_create()`, `aws_bucket_delete()`, `aws_bucket_download()`, `aws_bucket_exists()`, `aws_bucket_tree()`, `aws_bucket_upload()`, `aws_buckets()`, `six_bucket_delete()`, `six_bucket_upload()`

Examples

```
bucket_name <- random_bucket()
if (!aws_bucket_exists(bucket_name)) aws_bucket_create(bucket_name)
links_file <- file.path(system.file(), "Meta/links.rds")
aws_file_upload(
  links_file,
  s3_path(bucket_name, basename(links_file))
)
aws_bucket_list_objects(bucket = bucket_name)
# cleanup
six_bucket_delete(bucket_name, force = TRUE)
```

`aws_bucket_tree` *Print a tree of the objects in a bucket*

Description

Print a tree of the objects in a bucket

Usage

```
aws_bucket_tree(bucket, recurse = TRUE, ...)
```

Arguments

`bucket`	(character) bucket name; must be length 1. required
`recurse`	(logical) returns all AWS S3 objects in lower sub directories, default: `TRUE`
`...`	Additional arguments passed to `s3fs::s3_dir_tree()`

Value

character vector of objects/files within the bucket, printed as a tree

See Also

Other buckets: `aws_bucket_create()`, `aws_bucket_delete()`, `aws_bucket_download()`, `aws_bucket_exists()`, `aws_bucket_list_objects()`, `aws_bucket_upload()`, `aws_buckets()`, `six_bucket_delete()`, `six_bucket_upload()`

Examples

```
bucket_name <- random_bucket()
if (!aws_bucket_exists(bucket_name)) aws_bucket_create(bucket_name)
links_file <- file.path(system.file(), "Meta/links.rds")
pkgs_file <- file.path(system.file(), "Meta/package.rds")
demo_file <- file.path(system.file(), "Meta/demo.rds")
aws_file_upload(
```

```
  c(links_file, pkgs_file, demo_file),
  s3_path(
    bucket_name,
    c(
      basename(links_file),
      basename(pkgs_file),
      basename(demo_file)
    )
  )
)
aws_bucket_tree(bucket_name)

# cleanup
objs <- aws_bucket_list_objects(bucket_name)
aws_file_delete(objs$uri)
aws_bucket_delete(bucket_name, force = TRUE)
aws_bucket_exists(bucket_name)
```

`aws_bucket_upload` *Upload a folder of files to create an S3 bucket*

Description

Upload a folder of files to create an S3 bucket

Usage

```
aws_bucket_upload(
  path,
  bucket,
  max_batch = fs::fs_bytes("100MB"),
  force = FALSE,
  ...
)
```

Arguments

`path`	(character) local path to a directory. required
`bucket`	(character) bucket name. required
`max_batch`	(fs_bytes) maximum batch size being uploaded with each multipart
`force`	(logical) force deletion without going through the prompt. default: `FALSE`. Should only be set to `TRUE` when required for non-interactive use.
`...`	named parameters passed on to `s3fs::s3_dir_upload()`

Details

To upload individual files see `aws_file_upload()`

Value

the s3 format path of the bucket uploaded to

Note

Requires the env var AWS_REGION. This function prompts you to make sure that you want to delete the bucket.

See Also

Other buckets: aws_bucket_create(), aws_bucket_delete(), aws_bucket_download(), aws_bucket_exists(), aws_bucket_list_objects(), aws_bucket_tree(), aws_buckets(), six_bucket_delete(), six_bucket_upload()

Examples

```
library(fs)
tdir <- path(tempdir(), "apples")
dir.create(tdir)
tfiles <- replicate(n = 10, file_temp(tmp_dir = tdir, ext = ".txt"))
invisible(lapply(tfiles, function(x) write.csv(mtcars, x)))

bucket_name <- random_bucket()
if (!aws_bucket_exists(bucket_name)) aws_bucket_create(bucket_name)
aws_bucket_upload(path = tdir, bucket = bucket_name)
aws_bucket_list_objects(bucket_name)

# cleanup
objs <- aws_bucket_list_objects(bucket_name)
aws_file_delete(objs$uri)
aws_bucket_list_objects(bucket_name)
aws_bucket_delete(bucket_name, force = TRUE)
aws_bucket_exists(bucket_name)
```

---

aws_configure *Configure sixtyfour settings*

Description

Configure sixtyfour settings

Usage

```
aws_configure(redacted = FALSE, redact_str = "*****", verbose = TRUE)
```

Arguments

redacted	(logical) Redact secrets? Default: FALSE. If TRUE, secret values are redacted (replaced with redact_str) in certain messages and output from functions. See *What is Redacted* below.
redact_str	(character) String to use to replace redacted values. Default: "*****"
verbose	(logical) Print verbose output? Default: TRUE. Applies only to cli::cli_alert_info(), cli::cli_alert_warning(), and cli::cli_alert_success() functions that are used throughout this package. There's still a few places where verbose may not be respected.

Value

S3 class aws_settings

What is Redacted

What's redacted is currently hard-coded in the package. There's only certain functions and certain elements in the output of those functions that are redacted. The following is what's redacted with aws_configure(redacted = TRUE) or with_redacted():

- aws_whoami(): AWS Account ID via account_id()
- six_user_creds(): Access Key ID
- groups functions:
  - functions: aws_groups(), aws_group(), aws_group_create()
  - attribute: Arn (includes AWS Account ID)
- roles functions:
  - functions: aws_roles(), aws_role(), aws_role_create()
  - attribute: Arn (includes AWS Account ID)
- user functions:
  - functions: aws_users(), aws_user(), aws_user_create(), aws_user_add_to_group(), aws_user_remove_from_group()
  - attribute: Arn (includes AWS Account ID)
- aws_user_access_key_delete(): Access Key ID

---

aws_db_cluster_status *Get cluster status*

Description

Get cluster status

Usage

```
aws_db_cluster_status(id)
```

Arguments

id	(character) Cluster identifier. Use this identifier to refer to the cluster for any subsequent cluster operations such as deleting or modifying. The identifier also appears in the Amazon Redshift console. Must be unique for all clusters within a Amazon Web Services account.

Value

(character) the status of the cluster, e.g., "creating", "available", "not found"

See Also

Other database: aws_db_instance_status(), aws_db_rds_con(), aws_db_rds_create(), aws_db_rds_list(), aws_db_redshift_con(), aws_db_redshift_create()

Examples

```
## Not run:
aws_db_cluster_status(id = "scotts-test-cluster-456")

## End(Not run)
```

---

aws_db_instance_status

Get instance status

Description

Get instance status

Usage

```
aws_db_instance_status(id)
```

Arguments

id	(character) required. instance identifier. The identifier for this DB instance. This parameter is stored as a lowercase string. Constraints: must contain from 1 to 63 letters, numbers, or hyphens; first character must be a letter; can't end with a hyphen or contain two consecutive hyphens. required.

Value

(character) the status of the instance, e.g., "creating", "available", "not found"

See Also

Other database: aws_db_cluster_status(), aws_db_rds_con(), aws_db_rds_create(), aws_db_rds_list(), aws_db_redshift_con(), aws_db_redshift_create()

Examples

```
## Not run:
aws_db_instance_status(id = "thedbinstance")

## End(Not run)
```

`aws_db_rds_con` *Get a database connection to Amazon RDS*

Description

Supports: MariaDB, MySQL, and Postgres

Usage

```
aws_db_rds_con(
  user = NULL,
  pwd = NULL,
  id = NULL,
  host = NULL,
  port = NULL,
  dbname = NULL,
  engine = NULL,
  ...
)
```

Arguments

`user`, `pwd`, `host`, `port`, `dbname`, `...`
: named parameters passed on to DBI::dbConnect. Note that the `user` and `pwd` are for your AWS IAM account; and the same as those you used to create the cluster with `aws_db_redshift_create()`

`id`
: (character) Cluster identifier. If you supply `id`, we'll fetch `host`, `port`, and `dbname`. If `id` is not supplied. you have to supply `host`, `port`, and `dbname`. Refer to this parameter definition in `aws_db_redshift_create()` for more details.

`engine`
: (character) The engine to use. optional if `user`, `pwd`, and `id` are supplied - otherwise required

Details

RDS supports many databases, but we only provide support for MariaDB, MySQL, and Postgres

If the `engine` you've chosen for your RDS instance is not supported with this function, you can likely connect to it on your own

Value

an S4 object that inherits from `DBIConnection`

See Also

Other database: aws_db_cluster_status(), aws_db_instance_status(), aws_db_rds_create(), aws_db_rds_list(), aws_db_redshift_con(), aws_db_redshift_create()

Examples

```
## Not run:
con_rds <- aws_db_rds_con("<define all params here>")
con_rds

library(DBI)
library(RMariaDB)
dbListTables(con_rds)
dbWriteTable(con_rds, "mtcars", mtcars)
dbListTables(con_rds)
dbReadTable(con_rds, "mtcars")

library(dplyr)
tbl(con_rds, "mtcars")

## End(Not run)
```

---

aws_db_rds_create *Create an RDS cluster*

Description

Create an RDS cluster

Usage

```
aws_db_rds_create(
  id,
  class,
  user = NULL,
  pwd = NULL,
  dbname = "dev",
  engine = "mariadb",
  storage = 20,
  storage_encrypted = TRUE,
  security_group_ids = NULL,
  wait = TRUE,
  verbose = TRUE,
  aws_secrets = TRUE,
  iam_database_auth = FALSE,
  ...
)
```

Arguments

id	(character) required. instance identifier. The identifier for this DB instance. This parameter is stored as a lowercase string. Constraints: must contain from 1 to 63 letters, numbers, or hyphens; first character must be a letter; can't end with a hyphen or contain two consecutive hyphens. required.
class	(character) required. The compute and memory capacity of the DB instance, for example db.m5.large.
user	(character) User name associated with the admin user account for the cluster that is being created. If NULL, we generate a random user name, see random_user()
pwd	(character) Password associated with the admin user account for the cluster that is being created. If NULL, we generate a random password with aws_secrets_pwd() (which uses the AWS Secrets Manager service)
dbname	(character) The name of the first database to be created when the cluster is created. default: "dev". additional databases can be created within the cluster
engine	(character) The engine to use. default: "mariadb". required. one of: mariadb, mysql, or postgres
storage	(character) The amount of storage in gibibytes (GiB) to allocate for the DB instance. default: 20
storage_encrypted	(logical) Whether the DB instance is encrypted. default: TRUE
security_group_ids	(character) VPC security group identifiers; one or more. If none are supplied, you should go into your AWS Redshift dashboard and add the appropriate VPC security group.
wait	(logical) wait for cluster to initialize? default: TRUE. If you don't wait (FALSE) then there's many operations you can not do until the cluster is available. If wait=FALSE use aws_db_instance_status() to check on the cluster status.
verbose	(logical) verbose informational output? default: TRUE
aws_secrets	(logical) should we manage your database credentials in AWS Secrets Manager? default: TRUE
iam_database_auth	(logical) Use IAM database authentication? default: FALSE
...	named parameters passed on to create_db_instance

Details

See above link to create_db_instance docs for details on requirements for each parameter

Note that even though you can use any option for engine in this function, we may not provide the ability to connect to the chosen data source in this package.

Value

returns NULL, this function called for the side effect of creating an RDS instance

Waiting

Note that with `wait = TRUE` this function waits for the instance to be available for returning. That wait can be around 5 - 7 minutes. You can instead set `wait = FALSE` and then check on the status of the instance yourself in the AWS dashboard.

See Also

Other database: `aws_db_cluster_status()`, `aws_db_instance_status()`, `aws_db_rds_con()`, `aws_db_rds_list()`, `aws_db_redshift_con()`, `aws_db_redshift_create()`

`aws_db_rds_list` *Get information for all RDS instances*

Description

Get information for all RDS instances

Usage

```
aws_db_rds_list()
```

Value

a tibble of instance details; see https://www.paws-r-sdk.com/docs/rds_describe_db_instances/ an empty tibble if no instances found

See Also

Other database: `aws_db_cluster_status()`, `aws_db_instance_status()`, `aws_db_rds_con()`, `aws_db_rds_create()`, `aws_db_redshift_con()`, `aws_db_redshift_create()`

Examples

```
aws_db_rds_list()
```

`aws_db_redshift_con` *Get a database connection to Amazon Redshift*

Description

Get a database connection to Amazon Redshift

Usage

```
aws_db_redshift_con(
  user,
  pwd,
  id = NULL,
  host = NULL,
  port = NULL,
  dbname = NULL,
  ...
)
```

Arguments

`user`, `pwd`, `host`, `port`, `dbname`, `...`	named parameters passed on to DBI::dbConnect. Note that the `user` and `pwd` are for your AWS IAM account; and the same as those you used to create the cluster with `aws_db_redshift_create()`
`id`	(character) Cluster identifier. If you supply `id`, we'll fetch `host`, `port`, and `dbname`. If `id` is not supplied. you have to supply `host`, `port`, and `dbname`. Refer to this parameter definition in `aws_db_redshift_create()` for more details.

Details

The connection returned is created using RPostgres

You can manage Redshift programatically via paws::redshift

Value

an object of class `RedshiftConnection`

See Also

Other database: `aws_db_cluster_status()`, `aws_db_instance_status()`, `aws_db_rds_con()`, `aws_db_rds_create()`, `aws_db_rds_list()`, `aws_db_redshift_create()`

Examples

```
## Not run:
library(DBI)
library(RPostgres)

con_rshift <- aws_db_redshift_con("<define all params here>")
con_rshift
library(RPostgres)
dbListTables(con_rshift)
dbWriteTable(con_rshift, "mtcars", mtcars)
dbListTables(con_rshift)

library(dplyr)
tbl(con_rshift, "mtcars")

## End(Not run)
```

---

aws_db_redshift_create

Create a Redshift cluster

Description

Create a Redshift cluster

Usage

```
aws_db_redshift_create(
  id,
  user,
  pwd,
  dbname = "dev",
  cluster_type = "multi-node",
  node_type = "dc2.large",
  number_nodes = 2,
  security_group_ids = NULL,
  wait = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

id	(character) Cluster identifier. Use this identifier to refer to the cluster for any subsequent cluster operations such as deleting or modifying. The identifier also appears in the Amazon Redshift console. Must be unique for all clusters within a Amazon Web Services account.

user	(character) User name associated with the admin user account for the cluster that is being created. This is the username for your IAM account
pwd	(character) Password associated with the admin user account for the cluster that is being created. This is the password for your IAM account
dbname	(character) The name of the first database to be created when the cluster is created. default: "dev". additional databases can be created within the cluster
cluster_type	(character) The type of the cluster: "single-node" or "multi-node" (default).
node_type	(character) The node type to be provisioned for the cluster. defaul: "dc2.large"
number_nodes	(integer/numeric) number of nodes; for multi-node cluster type, this must be 2 or greater. default: 2
security_group_ids	(character) VPC security group identifiers; one or more. If none are supplied, you should go into your AWS Redshift dashboard and add the appropriate VPC security group.
wait	(logical) wait for cluster to initialize? default: TRUE. If you don't wait (FALSE) then there's many operations you can not do until the cluster is available. If wait=FALSE use aws_db_cluster_status() to check on the cluster status.
verbose	(logical) verbose informational output? default: TRUE
...	named parameters passed on to create_cluster

Value

returns NULL, this function called for the side effect of creating an Redshift instance

Waiting

Note that with wait = TRUE this function waits for the instance to be available for returning. That wait can be around 5 - 7 minutes. You can instead set wait = FALSE and then check on the status of the instance yourself in the AWS dashboard.

Note

See above link to create_cluster docs for details on requirements for each parameter

See Also

Other database: aws_db_cluster_status(), aws_db_instance_status(), aws_db_rds_con(), aws_db_rds_create(), aws_db_rds_list(), aws_db_redshift_con()

`aws_file_attr` *File attributes*

Description

File attributes

Usage

```
aws_file_attr(remote_path)
```

Arguments

`remote_path`	(character) one or more remote S3 paths. required

Value

a tibble with many columns, with number of rows matching length of `remote_path`

Note

uses `s3fs::s3_file_info()` internally

See Also

Other files: `aws_file_copy()`, `aws_file_delete()`, `aws_file_download()`, `aws_file_exists()`, `aws_file_rename()`, `aws_file_upload()`, `six_file_upload()`

Examples

```
library(glue)
bucket <- random_bucket()
if (!aws_bucket_exists(bucket)) {
  aws_bucket_create(bucket)
}

# upload some files
tfiles <- replicate(n = 3, tempfile())
paths <- s3_path(bucket, glue("{basename(tfiles)}.txt"))
for (file in tfiles) cat("Hello saturn!!!!!!\n", file = file)
for (file in tfiles) print(readLines(file))
aws_file_upload(path = tfiles, remote_path = paths)

# files one by one
aws_file_attr(paths[1])
aws_file_attr(paths[2])
aws_file_attr(paths[3])
# or all together
aws_file_attr(paths)
```

```
# Cleanup
six_bucket_delete(bucket, force = TRUE)
```

`aws_file_copy` *Copy files between buckets*

Description

Copy files between buckets

Usage

```
aws_file_copy(remote_path, bucket, force = FALSE, ...)
```

Arguments

`remote_path`	(character) one or more remote S3 paths. required
`bucket`	(character) bucket to copy files to. required. if the bucket does not exist we prompt you asking if you'd like the bucket to be created
`force`	(logical) force bucket creation without going through the prompt. default: `FALSE`. Should only be set to `TRUE` when required for non-interactive use.
`...`	named parameters passed on to `s3fs::s3_file_copy()`

Value

vector of paths, length matches `length(remote_path)`

See Also

Other files: `aws_file_attr()`, `aws_file_delete()`, `aws_file_download()`, `aws_file_exists()`, `aws_file_rename()`, `aws_file_upload()`, `six_file_upload()`

Examples

```
bucket1 <- random_bucket()
aws_bucket_create(bucket1)

# create files in an existing bucket
tfiles <- replicate(n = 3, tempfile())
for (i in tfiles) cat("Hello\nWorld\n", file = i)
paths <- s3_path(bucket1, c("aaa", "bbb", "ccc"), ext = "txt")
aws_file_upload(tfiles, paths)

# create a new bucket
bucket2 <- random_bucket()
new_bucket <- aws_bucket_create(bucket = bucket2)
```

```
# add existing files to the new bucket
aws_file_copy(paths, bucket2)

# or, create a bucket that doesn't exist yet
bucket3 <- random_bucket()
aws_file_copy(paths, bucket3, force = TRUE)

# Cleanup
six_bucket_delete(bucket1, force = TRUE)
six_bucket_delete(bucket2, force = TRUE)
six_bucket_delete(bucket3, force = TRUE)
```

`aws_file_delete` *Delete a file*

Description

Delete a file

Usage

```
aws_file_delete(remote_path, ...)
```

Arguments

`remote_path`	(character) one or more remote S3 paths. required
`...`	named parameters passed on to delete_object

Value

`NULL` invisibly

See Also

Other files: `aws_file_attr()`, `aws_file_copy()`, `aws_file_download()`, `aws_file_exists()`, `aws_file_rename()`, `aws_file_upload()`, `six_file_upload()`

Examples

```
# create a file
bucket <- random_bucket()
aws_bucket_create(bucket)
tfile <- tempfile()
cat("Hello World!\n", file = tfile)
aws_file_upload(path = tfile, remote_path = s3_path(bucket))

# delete the file
```

```
aws_file_delete(s3_path(bucket, basename(tfile)))

# file does not exist - no error is raised
aws_file_delete(s3_path(bucket, "TESTING123"))

# Cleanup
six_bucket_delete(bucket, force = TRUE)
```

`aws_file_download` *Download a file*

Description

Download a file

Usage

```
aws_file_download(remote_path, path, ...)
```

Arguments

`remote_path`	(character) one or more remote S3 paths. required
`path`	(character) one or more file paths to write to. required
`...`	named parameters passed on to `s3fs::s3_file_download()`

Value

(character) a vector of local file paths

See Also

Other files: `aws_file_attr()`, `aws_file_copy()`, `aws_file_delete()`, `aws_file_exists()`, `aws_file_rename()`, `aws_file_upload()`, `six_file_upload()`

Examples

```
library(glue)

# single file
bucket1 <- random_bucket()
aws_bucket_create(bucket1)
tfile1 <- tempfile()
remote1 <- s3_path(bucket1, glue("{basename(tfile1)}.txt"))
cat("Hello World!\n", file = tfile1)
aws_file_upload(path = tfile1, remote_path = remote1)
dfile <- tempfile()
aws_file_download(remote_path = remote1, path = dfile)
readLines(dfile)
```

```
# many files
bucket2 <- random_bucket()
aws_bucket_create(bucket2)
tfiles <- replicate(n = 3, tempfile())
for (file in tfiles) cat("Hello mars!!!!!!\n", file = file)
for (file in tfiles) print(readLines(file))
for (file in tfiles) {
  aws_file_upload(file, s3_path(bucket2, glue("{basename(file)}.txt")))
}
downloadedfiles <- replicate(n = 3, tempfile())
for (file in downloadedfiles) print(file.exists(file))
remotes2 <- s3_path(bucket2, glue("{basename(tfiles)}.txt"))
aws_file_download(remote_path = remotes2, path = downloadedfiles)
for (file in downloadedfiles) print(readLines(file))

# Cleanup
six_bucket_delete(bucket1, force = TRUE)
six_bucket_delete(bucket2, force = TRUE)
```

`aws_file_exists` *Check if a file exists*

Description

Check if a file exists

Usage

```
aws_file_exists(remote_path)
```

Arguments

`remote_path`	(character) one or more remote S3 paths. required

Value

vector of booleans (`TRUE` or `FALSE`), length matches `length(remote_path)`

See Also

Other files: `aws_file_attr()`, `aws_file_copy()`, `aws_file_delete()`, `aws_file_download()`, `aws_file_rename()`, `aws_file_upload()`, `six_file_upload()`

Examples

```
library(glue)
bucket <- random_bucket()
aws_bucket_create(bucket)

# upload some files
tfiles <- replicate(n = 3, tempfile())
paths <- s3_path(bucket, glue("{basename(tfiles)}.txt"))
for (file in tfiles) cat("Hello saturn!!!!!!\n", file = file)
for (file in tfiles) print(readLines(file))
aws_file_upload(path = tfiles, remote_path = paths)

# check that files exist
aws_file_exists(paths[1])
aws_file_exists(paths[2])
aws_file_exists(s3_path(bucket, "doesnotexist.txt"))

# Cleanup
six_bucket_delete(bucket, force = TRUE)
```

`aws_file_rename` *Rename remote files*

Description

Rename remote files

Usage

```
aws_file_rename(remote_path, new_remote_path, ...)
```

Arguments

`remote_path`	(character) one or more remote S3 paths. required
`new_remote_path`	(character) one or more remote S3 paths. required. length must match `remote_path`
`...`	named parameters passed on to `s3fs::s3_file_move()`

Value

vector of paths, length matches `length(remote_path)`

See Also

Other files: `aws_file_attr()`, `aws_file_copy()`, `aws_file_delete()`, `aws_file_download()`, `aws_file_exists()`, `aws_file_upload()`, `six_file_upload()`

Examples

```
bucket <- random_bucket()
aws_bucket_create(bucket)

# rename files
tfiles <- replicate(n = 3, tempfile())
for (i in tfiles) cat("Hello\nWorld\n", file = i)
paths <- s3_path(bucket, c("aaa", "bbb", "ccc"), ext = "txt")
aws_file_upload(tfiles, paths)
new_paths <- s3_path(bucket, c("new_aaa", "new_bbb", "new_ccc"),
  ext = "txt"
)
aws_file_rename(paths, new_paths)

# Cleanup
six_bucket_delete(bucket, force = TRUE)
```

`aws_file_upload` *Upload a file*

Description

Upload a file

Usage

```
aws_file_upload(path, remote_path, ...)
```

Arguments

`path`	(character) a file path to read from. required
`remote_path`	(character) a remote path where the file should go. required
`...`	named parameters passed on to `s3fs::s3_file_copy()`

Details

to upload a folder of files see `aws_bucket_upload()`

Value

(character) a vector of remote s3 paths

See Also

Other files: `aws_file_attr()`, `aws_file_copy()`, `aws_file_delete()`, `aws_file_download()`, `aws_file_exists()`, `aws_file_rename()`, `six_file_upload()`

Examples

```
bucket1 <- random_bucket()
aws_bucket_create(bucket1)
cat(bucket1)
demo_rds_file <- file.path(system.file(), "Meta/demo.rds")
aws_file_upload(
  demo_rds_file,
  s3_path(bucket1, basename(demo_rds_file))
)

## many files at once
bucket2 <- random_bucket()
if (!aws_bucket_exists(bucket2)) {
  aws_bucket_create(bucket2)
}
cat(bucket2)
links_file <- file.path(system.file(), "Meta/links.rds")
aws_file_upload(
  c(demo_rds_file, links_file),
  s3_path(bucket2, c(basename(demo_rds_file), basename(links_file))),
  overwrite = TRUE
)

# set expiration, expire 1 minute from now
aws_file_upload(demo_rds_file, s3_path(bucket2, "ddd.rds"),
  Expires = Sys.time() + 60,
  overwrite = TRUE
)

# bucket doesn't exist
try(aws_file_upload(demo_rds_file, "s3://not-a-bucket/eee.rds"))

# path doesn't exist
try(
  aws_file_upload(
    "file_doesnt_exist.txt",
    s3_path(bucket2, "file_doesnt_exist.txt")
  )
)

# Path's without file extensions behave a little weird
## With extension
bucket3 <- random_bucket()
if (!aws_bucket_exists(bucket3)) {
  aws_bucket_create(bucket3)
}
## Both the next two lines do the same exact thing: make a file in the
## same path in a bucket
pkg_rds_file <- file.path(system.file(), "Meta/package.rds")
aws_file_upload(pkg_rds_file, s3_path(bucket3, "package2.rds"),
 overwrite = TRUE)
aws_file_upload(pkg_rds_file, s3_path(bucket3),
```

```
 overwrite = TRUE)

## Without extension
## However, it's different for a file without an extension
## This makes a file in the bucket at path DESCRIPTION
rd_file <- file.path(system.file(), "Meta/Rd.rds")
desc_file <- system.file("DESCRIPTION", package = "sixtyfour")
aws_file_upload(desc_file, s3_path(bucket3), overwrite = TRUE)

## Whereas this creates a directory called DESCRIPTION with
## a file DESCRIPTION within it
aws_file_upload(desc_file, s3_path(bucket3, "DESCRIPTION"),
  overwrite = TRUE)

# Cleanup
six_bucket_delete(bucket1, force = TRUE)
six_bucket_delete(bucket2, force = TRUE)
six_bucket_delete(bucket3, force = TRUE)
```

`aws_group` *Get a group*

Description

Get a group

Usage

```
aws_group(name)
```

Arguments

`name`	(character) the group name

Details

see docs https://www.paws-r-sdk.com/docs/iam_get_group/

Value

a named list with slots for:

- group: information about the group (tibble)
- users: users in the group (tibble)
- policies (character)
- attached_policies (tibble)

See Also

Other groups: aws_group_create(), aws_group_delete(), aws_group_exists(), aws_groups(), six_group_delete()

Examples

```
# create a group
aws_group_create("testing")
# get the group
aws_group(name = "testing")
# cleanup
aws_group_delete(name = "testing")
```

---

aws_groups *List all groups or groups for a single user*

Description

List all groups or groups for a single user

Usage

```
aws_groups(username = NULL, ...)
```

Arguments

username	(character) a username. optional
...	parameters passed on to paws list_groups_for_user if username is non-NULL, otherwise passed on to list_users

Value

A tibble with information about groups

See Also

Other groups: aws_group(), aws_group_create(), aws_group_delete(), aws_group_exists(), six_group_delete()

Examples

```
aws_groups()
aws_groups(username = aws_user_current())
```

`aws_group_create` *Create a group*

Description

Create a group

Usage

```
aws_group_create(name, path = NULL)
```

Arguments

`name`	(character) A group name. required
`path`	(character) The path for the group name. optional. If it is not included, it defaults to a slash (/).

Details

See https://www.paws-r-sdk.com/docs/iam_create_group/ docs for details on the parameters

Value

A tibble with information about the group created

See Also

Other groups: `aws_group()`, `aws_group_delete()`, `aws_group_exists()`, `aws_groups()`, `six_group_delete()`

Examples

```
aws_group_create("testingagroup")
aws_group("testingagroup")
# cleanup
aws_group_delete("testingagroup")
```

aws_group_delete *Delete a group*

Description

Delete a group

Usage

```
aws_group_delete(name)
```

Arguments

name	(character) A group name. required

Details

See https://www.paws-r-sdk.com/docs/iam_delete_group/ docs for more details

Value

NULL invisibly

See Also

Other groups: aws_group(), aws_group_create(), aws_group_exists(), aws_groups(), six_group_delete()

Examples

```
aws_group_create("somegroup")
aws_group_delete("somegroup")
```

aws_group_exists *Check if a group exists*

Description

Check if a group exists

Usage

```
aws_group_exists(name)
```

Arguments

name	(character) the group name

Details

uses aws_group internally. see docs https://www.paws-r-sdk.com/docs/iam_get_group/

Value

a single boolean

See Also

Other groups: aws_group(), aws_group_create(), aws_group_delete(), aws_groups(), six_group_delete()

Examples

```
aws_group_create("apples")
aws_group_exists("apples")
aws_group_exists("doesnotexist")
# cleanup
aws_group_delete("apples")
```

---

aws_has_creds *Check if appropriate AWS credentials are available*

Description

Check if appropriate AWS credentials are available

Usage

```
aws_has_creds()
```

Value

single boolean

Examples

```
aws_has_creds()
```

`aws_policies` *List policies*

Description

List policies

Usage

```
aws_policies(refresh = FALSE, ...)
```

Arguments

`refresh`	(logical) refresh results? default: `FALSE`. to invalidate cache and refresh policy data, set `refresh=TRUE`
`...`	named arguments passed on to list_policies

Details

uses `memoise` internally to cache results to speed up all subsequent calls to the function

Value

A tibble with information about policies. Each row is a policy. Columns:

- PolicyName
- PolicyId
- Path
- Arn
- CreateDate
- UpdateDate
- AttachmentCount
- PermissionsBoundaryUsageCount
- IsAttachable
- Description
- Tags

See Also

Other policies: `as_policy_arn()`, `aws_policy()`, `aws_policy_attach()`, `aws_policy_create()`, `aws_policy_delete()`, `aws_policy_delete_version()`, `aws_policy_detach()`, `aws_policy_exists()`, `aws_policy_list_entities()`, `aws_policy_list_versions()`, `aws_policy_update()`

Examples

```
# takes a while on the first execution in an R session
aws_policies()


# faster because first call memoised the result
aws_policies()
# refresh=TRUE will pull from AWS
aws_policies(refresh = TRUE)
```

`aws_policy` *Get a policy*

Description

Get a policy

Usage

```
aws_policy(name, local = FALSE, path = NULL)
```

Arguments

`name`	(character) a policy name or arn
`local`	(logical) if `TRUE` use your AWS account for your own managed policies. If `FALSE`, AWS managed policies
`path`	(character) if not `NULL`, we add the path into the ARN before the `name` value

Details

see docs https://www.paws-r-sdk.com/docs/iam_get_policy/

Value

a tibble with policy details

See Also

Other policies: `as_policy_arn()`, `aws_policies()`, `aws_policy_attach()`, `aws_policy_create()`, `aws_policy_delete()`, `aws_policy_delete_version()`, `aws_policy_detach()`, `aws_policy_exists()`, `aws_policy_list_entities()`, `aws_policy_list_versions()`, `aws_policy_update()`

Examples

```
# get an AWS managed policy (local = FALSE - the default)
aws_policy("AmazonS3FullAccess")

# get a policy by arn
aws_policy("arn:aws:iam::aws:policy/AmazonS3FullAccess")
```

`aws_policy_attach` *Attach a policy to a user, group, or role*

Description

Attach a policy to a user, group, or role

Usage

```
aws_policy_attach(.x, policy)
```

Arguments

`.x`	result of a call to create or get method for user, group, or role
`policy`	(character) a policy name or ARN

Value

A tibble with information about policies

See Also

Other policies: `as_policy_arn()`, `aws_policies()`, `aws_policy()`, `aws_policy_create()`, `aws_policy_delete()`, `aws_policy_delete_version()`, `aws_policy_detach()`, `aws_policy_exists()`, `aws_policy_list_entities()`, `aws_policy_list_versions()`, `aws_policy_update()`

Examples

```
if (aws_user_exists("user123")) {
  aws_user_delete("user123")
}

aws_user_create("user123")
aws_policy("AmazonRDSDataFullAccess")
aws_user("user123") %>% aws_policy_attach("AmazonRDSDataFullAccess")
aws_user("user123")$attached_policies
# cleanup
six_user_delete("user123")
```

aws_policy_create *Create a policy*

Description

Create a policy

Usage

```
aws_policy_create(name, document, path = NULL, description = NULL, tags = NULL)
```

Arguments

name	(character) a policy name. required
document	(character) the policy document you want to use as the content for the new policy. required.
path	(character) the path for the policy. if not given default is "/". optional
description	(character) a friendly description of the policy. optional. cannot be changed after assigning it
tags	(character) a vector of tags that you want to attach to the new IAM policy. Each tag consists of a key name and an associated value. optional

Details

see docs https://www.paws-r-sdk.com/docs/iam_create_policy/

Value

a tibble with policy details

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_delete(), aws_policy_delete_version(), aws_policy_detach(), aws_policy_exists(), aws_policy_list_entities(), aws_policy_list_versions(), aws_policy_update()

Examples

```
if (aws_policy_exists("MyPolicy123")) {
  aws_policy_delete("MyPolicy123")
}

# Create policy document
st8ment1 <- aws_policy_statement("iam:GetUser", "*")
st8ment2 <- aws_policy_statement("s3:ListAllMyBuckets", "*")
doc <- aws_policy_document_create(st8ment1, st8ment2)
```

```
# Create policy
aws_policy_create("MyPolicy123", document = doc)

# cleanup - delete policy
aws_policy_delete("MyPolicy123")
```

`aws_policy_delete` *Delete a user managed policy*

Description

Delete a user managed policy

Usage

```
aws_policy_delete(name)
```

Arguments

`name`	(character) a policy name. required. within the function we lookup the policy arn which is what's passed to the AWS API

Value

invisibly returns `NULL`

AWS managed policies

You can not delete AWS managed policies.

Deleting process (adapted from `paws` docs)

Before you can delete a managed policy, you must first detach the policy from all users, groups, and roles that it is attached to. In addition, you must delete all the policy's versions. The following steps describe the process for deleting a managed policy:

- Detach the policy from all users, groups, and roles that the policy is attached to using `aws_policy_attach()`. To list all the users, groups, and roles that a policy is attached to use `aws_policy_list_entities()`
- Delete all versions of the policy using `aws_policy_delete_version()`. To list the policy's versions, use `aws_policy_list_versions()`. You cannot use `aws_policy_delete_version()` to delete the version that is marked as the default version. You delete the policy's default version in the next step of the process.
- Delete the policy using this function (this automatically deletes the policy's default version)

References

delete_policy

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete_version(), aws_policy_detach(), aws_policy_exists(), aws_policy_list_entities(), aws_policy_list_versions(), aws_policy_update()

Examples

```
if (aws_policy_exists("RdsAllow456")) {
  aws_policy_delete("RdsAllow456")
}

# Create policy document
doc <- aws_policy_document_create(
  aws_policy_statement(
    action = "rds-db:connect",
    resource = "*"
  )
)

# Create policy
invisible(aws_policy_create("RdsAllow456", document = doc))

# Delete policy
aws_policy_delete("RdsAllow456")
```

---

aws_policy_delete_version

Delete a policy version

Description

Delete a policy version

Usage

```
aws_policy_delete_version(name, version_id)
```

Arguments

name	(character) a policy name. required. within the function we lookup the policy arn which is what's passed to the AWS API
version_id	(character) The policy version to delete. required. Allows (via regex) a string of characters that consists of the lowercase letter 'v' followed by one or two digits, and optionally followed by a period '.' and a string of letters and digits.

Value

invisibly returns NULL

References

https://www.paws-r-sdk.com/docs/iam_delete_policy_version/

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete(), aws_policy_detach(), aws_policy_exists(), aws_policy_list_entities(), aws_policy_list_versions(), aws_policy_update()

Examples

```
if (aws_policy_exists("RdsAllow888")) {
  aws_policy_delete("RdsAllow888")
}

# Create policy document
doc <- aws_policy_document_create(
  aws_policy_statement(
    action = "rds-db:connect",
    resource = "*"
  )
)

# Create policy
invisible(aws_policy_create("RdsAllow888", document = doc))

# Add a new version of the policy
st8ment1 <- aws_policy_statement("iam:GetUser", "*")
new_doc <- aws_policy_document_create(st8ment1)
arn <- as_policy_arn("RdsAllow888", local = TRUE)
aws_policy_update(arn, document = new_doc, defaul = TRUE)

# List versions of the policy
aws_policy_list_versions("RdsAllow888")

# Delete a policy version
aws_policy_delete_version("RdsAllow888", "v1")

# Cleanup - delete  policy
aws_policy_delete("RdsAllow888")
```

---

aws_policy_detach *Detach a policy from a user, group, or role*

Description

Detach a policy from a user, group, or role

Usage

```
aws_policy_detach(.x, policy)
```

Arguments

.x	result of a call to create or get method for user, group, or role
policy	(character) a policy name or ARN

Value

A tibble with information about policies

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete(), aws_policy_delete_version(), aws_policy_exists(), aws_policy_list_entities(), aws_policy_list_versions(), aws_policy_update()

Examples

```
if (aws_user_exists("user456")) {
  aws_user_delete("user456")
}

aws_user_create("user456")
aws_user("user456") %>% aws_policy_attach("AmazonRDSDataFullAccess")
aws_user("user456") %>% aws_policy_detach("AmazonRDSDataFullAccess")
aws_user("user456")$attached_policies
# cleanup
six_user_delete("user456")
```

---

aws_policy_document_create

Create a policy document

Description

Create a policy document

Usage

```
aws_policy_document_create(..., .list = NULL)
```

Arguments

..., .list	policy statements as created by aws_policy_statement() or created manually. Pass in 1 or more statements via ... like statement1, statement2 or pass in as a list like .list = list(statement1, statement2). Each element must be a named list.

Value

a json class string. use as.character() to coerce to a regular string

Actions

Actions documentation appears to be all over the web. Here's a start:

- S3: https://docs.aws.amazon.com/service-authorization/latest/reference/list_amazons3.html # nolint
- EC2: https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_Operations.html # nolint
- IAM: https://docs.aws.amazon.com/IAM/latest/APIReference/API_Operations.html # nolint

Note

a document item is hard-coded:

- Version is set to 2012-10-17"

References

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html # nolint

Examples

```
library(jsonlite)

st8ment1 <- aws_policy_statement("iam:GetUser", "*")
st8ment2 <- aws_policy_statement("s3:ListAllMyBuckets", "*")
st8ment3 <- aws_policy_statement("s3-object-lambda:List*", "*")
aws_policy_document_create(st8ment1, st8ment2) %>% prettify()
aws_policy_document_create(.list = list(st8ment1, st8ment2)) %>% prettify()
aws_policy_document_create(st8ment3, .list = list(st8ment1, st8ment2)) %>%
  prettify()

# Policy document to give a user access to RDS
resource <- "arn:aws:rds-db:us-east-2:1234567890:dbuser:db-ABCDE1212/jane"
st8ment_rds <- aws_policy_statement(
  action = "rds-db:connect",
  resource = resource
)
```

```
aws_policy_document_create(st8ment_rds) %>% prettify()


### DB account = user in a database that has access to it
# all DB instances & DB accounts for a AWS account and AWS Region
aws_policy_document_create(
  aws_policy_statement(
    action = "rds-db:connect",
    resource = resource_rds("*", "*")
  )
) %>% prettify()
# all DB instances for a AWS account and AWS Region, single DB account
aws_policy_document_create(
  aws_policy_statement(
    action = "rds-db:connect",
    resource = resource_rds("jane_doe", "*")
  )
) %>% prettify()
# single DB instasnce, single DB account
aws_policy_document_create(
  aws_policy_statement(
    action = "rds-db:connect",
    resource = resource_rds("jane_doe", "db-ABCDEFGHIJKL01234")
  )
) %>% prettify()
# single DB instance, many users
aws_policy_document_create(
  aws_policy_statement(
    action = "rds-db:connect",
    resource = resource_rds(c("jane_doe", "mary_roe"), "db-ABCDEFGHIJKL01")
  )
) %>% prettify()
```

`aws_policy_exists` *Check if a policy exists*

Description

Checks for both customer managed and AWS managed policies

Usage

```
aws_policy_exists(name)
```

Arguments

`name`	(character) a policy name or arn

Value

single logical, TRUE or FALSE

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete(), aws_policy_delete_version(), aws_policy_detach(), aws_policy_list_entities(), aws_policy_list_versions(), aws_policy_update()

Examples

```
# just the policy name
aws_policy_exists("ReadOnlyAccess")
# as an ARN
aws_policy_exists("arn:aws:iam::aws:policy/ReadOnlyAccess")
# includes job-function in path
aws_policy_exists("Billing")
# includes service-role in path
aws_policy_exists("AWSCostAndUsageReportAutomationPolicy")
```

---

aws_policy_list_entities

List policy entities

Description

List policy entities

Usage

```
aws_policy_list_entities(name, ...)
```

Arguments

name	(character) a policy name. required. within the function we lookup the policy arn which is what's passed to the AWS API
...	additional named arguments passed on to internal paws method (see link below to its docs)

Value

tibble with columns:

- type: one of Users, Roles, Groups
- name: the user, role or group name
- id: the id for the user, role or group name

Zero row tibble if there are no entities

References

https://www.paws-r-sdk.com/docs/iam_list_entities_for_policy/

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete(), aws_policy_delete_version(), aws_policy_detach(), aws_policy_exists(), aws_policy_list_versions(), aws_policy_update()

Examples

```
aws_policy_list_entities("AdministratorAccess")
aws_policy_list_entities("AmazonRedshiftReadOnlyAccess")
```

---

aws_policy_list_versions

List policy versions

Description

List policy versions

Usage

```
aws_policy_list_versions(name, ...)
```

Arguments

name	(character) a policy name. required. within the function we lookup the policy arn which is what's passed to the AWS API
...	additional named arguments passed on to internal paws method (see link below to its docs)

Value

tibble with columns:

- VersionId
- IsDefaultVersion
- CreateDate

References

https://www.paws-r-sdk.com/docs/iam_list_policy_versions/

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete(), aws_policy_delete_version(), aws_policy_detach(), aws_policy_exists(), aws_policy_list_entities(), aws_policy_update()

Examples

```
aws_policy_list_versions("AmazonS3FullAccess")
aws_policy_list_versions("AmazonAppFlowFullAccess")
aws_policy_list_versions("AmazonRedshiftFullAccess")
```

---

aws_policy_statement *Create a policy statement*

Description

Create a policy statement

Usage

```
aws_policy_statement(action, resource, effect = "Allow", ...)
```

Arguments

action	(character) an action. required. see Actions below.
resource	(character) the object or objects the statement covers; see link below for more information
effect	(character) valid values: "Allow" (default), "Deny". length==1
...	Additional named arguments. See link in Details for options, and examples below

Details

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html
#nolint

Value

a named list

Examples

```
aws_policy_statement("iam:GetUser", "*")
aws_policy_statement("iam:GetUser", "*", Sid = "MyStatementId")
aws_policy_statement("iam:GetUser", "*",
  Condition = list(
    StringEqualsIgnoreCase = list("aws:username" = "johndoe")
  )
)
aws_policy_statement("iam:GetUser", "*",
  Principal = list(Service = "s3.amazonaws.com")
)
```

---

aws_policy_update *Update a policy*

Description

Update a policy

Usage

```
aws_policy_update(arn, document, default = FALSE)
```

Arguments

arn	(character) policy arn. required
document	(character) the policy document you want to use as the content for the new policy. required
default	(character) set this version as the policy's default version? optional. When this parameter is TRUE, the new policy version becomes the operative version. That is, it becomes the version that is in effect for the IAM users, groups, and roles that the policy is attached to. default: FALSE

Details

see docs https://www.paws-r-sdk.com/docs/iam_create_policy_version/

Value

a tibble with policy version details:

- VersionId
- IsDefaultVersion
- CreateDate

See Also

Other policies: as_policy_arn(), aws_policies(), aws_policy(), aws_policy_attach(), aws_policy_create(), aws_policy_delete(), aws_policy_delete_version(), aws_policy_detach(), aws_policy_exists(), aws_policy_list_entities(), aws_policy_list_versions()

Examples

```
if (aws_policy_exists("polisee")) {
  aws_policy_delete("polisee")
}

# Create policy document
st8ment1 <- aws_policy_statement("iam:GetUser", "*")
st8ment2 <- aws_policy_statement("s3:ListAllMyBuckets", "*")
doc <- aws_policy_document_create(st8ment1, st8ment2)

# Create policy
invisible(aws_policy_create("polisee", document = doc))

# Update the same policy
new_doc <- aws_policy_document_create(st8ment1)
arn <- as_policy_arn("polisee", local = TRUE)
aws_policy_update(arn, document = new_doc, default = TRUE)
aws_policy_list_versions("polisee")

# cleanup - delete the policy
aws_policy_delete_version("polisee", "v1")
aws_policy_delete("polisee")
```

---

aws_role *Get a role*

Description

Get a role

Usage

```
aws_role(name)
```

Arguments

name	(character) the role name

Details

see docs https://www.paws-r-sdk.com/docs/iam_get_role/; also includes policies and attached policies by calling list_role_policies and list_attached_role_policies

Value

a named list with slots for:

- role (tibble)
- policies (character)
- attached_policies (tibble)

See Also

Other roles: aws_role_create(), aws_role_delete(), aws_role_exists(), aws_roles()

Examples

```
trust_policy <- list(
  Version = "2012-10-17",
  Statement = list(
    list(
      Effect = "Allow",
      Principal = list(
        Service = "lambda.amazonaws.com"
      ),
      Action = "sts:AssumeRole"
    )
  )
)
doc <- jsonlite::toJSON(trust_policy, auto_unbox = TRUE)
desc <- "Another test role"
z <- aws_role_create("ALittleRole",
  assume_role_policy_document = doc,
  description = desc
)
aws_policy_attach(z, "ReadOnlyAccess")
res <- aws_role(name = "ALittleRole")
res
res$role
res$policies
res$attached_policies

# cleanup
aws_role("ALittleRole") %>%
  aws_policy_detach("ReadOnlyAccess")
aws_role_delete("ALittleRole")
```

---

aws_roles *List roles*

Description

List roles

Usage

```
aws_roles(...)
```

Arguments

...	parameters passed on to the paws list_users method

Value

A tibble with information about roles

See Also

Other roles: `aws_role()`, `aws_role_create()`, `aws_role_delete()`, `aws_role_exists()`

Examples

```
aws_roles()
```

---

aws_role_create *Create a role*

Description

Create a role

Usage

```
aws_role_create(
  name,
  assume_role_policy_document,
  path = NULL,
  description = NULL,
  max_session_duration = NULL,
  permission_boundary = NULL,
  tags = NULL
)
```

Arguments

name	(character) A role name. required
assume_role_policy_document	(character) The trust relationship policy document that grants an entity permission to assume the role. json as string. required
path	(character) The path for the role name. optional. If it is not included, it defaults to a slash (/).

`description` (character) a description fo the role. optional

`max_session_duration`
(character) The maximum session duration (in seconds) that you want to set for the specified role. optional

`permission_boundary`
(character) The ARN of the managed policy that is used to set the permissions boundary for the role. optional

`tags` (list) A list of tags that you want to attach to the new user. optional

Details

See https://www.paws-r-sdk.com/docs/iam_create_role/ docs for details on the parameters

Value

A tibble with information about the role created

See Also

Other roles: `aws_role()`, `aws_role_delete()`, `aws_role_exists()`, `aws_roles()`

Examples

```
role_name <- "AMinorRole"
trust_policy <- list(
  Version = "2012-10-17",
  Statement = list(
    list(
      Effect = "Allow",
      Principal = list(
        Service = "lambda.amazonaws.com"
      ),
      Action = "sts:AssumeRole"
    )
  )
)
doc <- jsonlite::toJSON(trust_policy, auto_unbox = TRUE)
desc <- "My test role"
z <- aws_role_create(role_name,
  assume_role_policy_document = doc,
  description = desc
)
# attach a policy
invisible(z %>% aws_policy_attach("AWSLambdaBasicExecutionRole"))

# cleanup
invisible(z %>% aws_policy_detach("AWSLambdaBasicExecutionRole"))
aws_role_delete(role_name)
```

aws_role_delete *Delete a role*

Description

Delete a role

Usage

```
aws_role_delete(name)
```

Arguments

name	(character) A role name. required

Details

See https://www.paws-r-sdk.com/docs/iam_delete_role/ docs for more details

Value

NULL invisibly

See Also

Other roles: aws_role(), aws_role_create(), aws_role_exists(), aws_roles()

Examples

```
if (aws_role_exists(name = "MyRole")) {
  aws_role_delete(name = "MyRole")
}
```

aws_role_exists *Check if a role exists*

Description

Check if a role exists

Usage

```
aws_role_exists(name)
```

Arguments

name	(character) the role name

Value

a single boolean

See Also

Other roles: aws_role(), aws_role_create(), aws_role_delete(), aws_roles()

Examples

```
aws_role_exists("AWSServiceRoleForRedshift")
aws_role_exists("NotARole")
```

---

aws_s3_policy_doc_create

Create a policy document for an S3 bucket

Description

Create a policy document for an S3 bucket

Usage

```
aws_s3_policy_doc_create(
  bucket,
  action,
  resource,
  effect = "Allow",
  sid = NULL,
  ...
)
```

Arguments

bucket	(character) bucket name. required
action	(character) an action. required. see Actions below.
resource	(character) the object or objects the statement covers; see link below for more information
effect	(character) valid values: "Allow" (default), "Deny". length==1
sid	(character) a statement id. optional
...	Additional named arguments. See link in Details for options, and examples below

Details

There's this separate function for creating policy docs for S3 because buckets are globally unique, so AWS figures out the region and account ID for you.

Value

a policy document as JSON (of class `json`)

Examples

```
bucket <- random_bucket()
aws_s3_policy_doc_create(
  bucket = bucket,
  action = s3_actions_read(),
  resource = c(bucket_arn(bucket), bucket_arn(bucket, objects = "*"))
)
```

---

aws_secrets_all *Get all secret values*

Description

Get all secret values

Usage

```
aws_secrets_all()
```

Value

(tbl) with secrets

Examples

```
aws_secrets_all()
```

`aws_secrets_create` *Create a secret*

Description

This function does not create your database username and/or password. Instead, it creates a "secret", which is typically a combination of credentials (username + password + other metadata)

Usage

```
aws_secrets_create(name, secret, description = NULL, ...)
```

Arguments

`name`	(character) The name of the new secret. required
`secret`	(character/raw) The text or raw data to encrypt and store in this new version of the secret. AWS recommends for text to use a JSON structure of key/value pairs for your secret value (see examples below). required
`description`	(character) The description of the secret. optional
`...`	further named parameters passed on to `create_secret` https://www.paws-r-sdk.com/docs/secretsmanager_create_secret/

Details

Note that we autogenerate a random UUID to pass to the `ClientRequestToken` parameter of the `paws` function `create_secret` used internally in this function.

This function creates a new secret. See `aws_secrets_update()` to update an existing secret. This function fails if you call it with an existing secret with the same name or ARN

Value

(list) with fields:

- ARN
- Name
- VersionId
- ReplicationStatus

Examples

```
try({
# Text secret
secret1 <- random_string("secret-", size = 16)
aws_secrets_create(
  name = secret1,
  secret = '{"username":"david","password":"EXAMPLE-PASSWORD"}',
```

```
  description = "My test database secret as a string"
)
aws_secrets_get(secret1)$SecretString

# Raw secret
secret2 <- random_string("secret-", size = 16)
aws_secrets_create(
  name = secret2,
  secret = charToRaw('{"username":"david","password":"EXAMPLE-PASSWORD"}'),
  description = "My test database secret as raw"
)
aws_secrets_get(secret2)$SecretBinary

# Cleanup
aws_secrets_delete(secret1, ForceDeleteWithoutRecovery = TRUE)
aws_secrets_delete(secret2, ForceDeleteWithoutRecovery = TRUE)
})
```

`aws_secrets_delete` *Delete a secret*

Description

Delete a secret

Usage

```
aws_secrets_delete(id, ...)
```

Arguments

`id`	(character) The name or ARN of the secret. required
`...`	further named parameters passed on to `delete_secret` https://www.paws-r-sdk.com/docs/secretsmanager_delete_secret/

Value

(list) with fields:

- ARN
- Name
- DeletionDate

Examples

```
try({
# Create a secret
secret <- random_string("secret-", size = 16)
aws_secrets_create(
  name = secret,
  secret = '{"username":"jill","password":"cow"}',
  description = "The fox jumped over the cow"
)

# Delete a secret
aws_secrets_delete(id = secret, ForceDeleteWithoutRecovery = TRUE)
})
```

---

aws_secrets_get *Get a secret*

Description

Get a secret

Usage

```
aws_secrets_get(id, ...)
```

Arguments

id	(character) The name or ARN of the secret. required
...	further named parameters passed on to get_secret_value https://www.paws-r-sdk.com/docs/secretsmanager_get_secret_value/

Value

(list) with fields:

- ARN
- Name
- VersionId
- SecretBinary
- SecretString
- VersionStages
- CreatedDate

Examples

```
try({
# Create a secret
secret <- random_string("secret-", size = 16)
aws_secrets_create(
  name = secret,
  secret = '{"username":"jane","password":"cat"}',
  description = "A string"
)

aws_secrets_get(secret)

# Does exist
aws_secrets_get(id = "MyTestDatabaseSecret")

# Does not exist
try(aws_secrets_get(id = "DoesntExist"))

# Cleanup
aws_secrets_delete(secret, ForceDeleteWithoutRecovery = TRUE)
})
```

---

aws_secrets_list *List secrets*

Description

List secrets

Usage

```
aws_secrets_list(...)
```

Arguments

...	parameters passed on to the paws method

Value

(list) list with secrets

Note

see https://www.paws-r-sdk.com/docs/secretsmanager_list_secrets/ for available parameters

Examples

```
aws_secrets_list()
```

`aws_secrets_pwd` *Get a random password*

Description

Get a random password

Usage

```
aws_secrets_pwd(...)
```

Arguments

`...`	named parameters passed on to `get_random_password` https://www.paws-r-sdk.com/docs/secretsmanager_get_random_password/

Details

The parameter `PasswordLength` is hard coded to `40L`

Value

a single string, of length 40

Examples

```
aws_secrets_pwd()
aws_secrets_pwd(ExcludeNumbers = TRUE)
```

`aws_secrets_rotate` *Rotate a secret*

Description

Rotate a secret

Usage

```
aws_secrets_rotate(id, lambda_arn = NULL, rules = NULL, immediately = TRUE)
```

Arguments

id	(character) The name or ARN of the secret. required
lambda_arn	(character) The ARN of the Lambda rotation function. Only supply for secrets that use a Lambda rotation function to rotate
rules	(list) asdfadf
immediately	(logical) whether to rotate the secret immediately or not. default: TRUE

Details

Note that we autogenerate a random UUID to pass to the ClientRequestToken parameter of the paws function used internally

Value

(list) with fields:

- ARN
- Name
- VersionId

References

https://www.paws-r-sdk.com/docs/secretsmanager_rotate_secret/

Examples

```
try({
# Create a secret
secret <- random_string("secret-", size = 16)
aws_secrets_create(
  name = secret,
  secret = '{"username":"billy","password":"willy"}',
  description = "A string"
)

# Rotate
try(aws_secrets_rotate(id = secret))

# Cleanup
aws_secrets_delete(secret, ForceDeleteWithoutRecovery = TRUE)
})
```

`aws_secrets_update` *Update a secret*

Description

Update a secret

Usage

```
aws_secrets_update(id, secret, ...)
```

Arguments

`id`	(character) The name or ARN of the secret. required
`secret`	(character/raw) The text or raw data to encrypt and store in this new version of the secret. AWS recommends for text to use a JSON structure of key/value pairs for your secret value (see examples below). required
`...`	further named parameters passed on to `put_secret_value` https://www.paws-r-sdk.com/docs/secretsmanager_put_secret_value/

Details

Note that we autogenerate a random UUID to pass to the `ClientRequestToken` parameter of the `paws` function used internally

Value

(list) with fields:

- ARN
- Name
- VersionId
- VersionStages

Examples

```
try({
# Create a secret
secret <- random_string("secret-", size = 16)
aws_secrets_create(
  name = secret,
  secret = '{"username":"debby","password":"kitty"}',
  description = "A string"
)

aws_secrets_get(secret)
```

```
# Update the secret
aws_secrets_update(
  id = secret,
  secret = '{"username":"debby","password":"kitten"}'
)

aws_secrets_get(secret)

# Cleanup
aws_secrets_delete(secret, ForceDeleteWithoutRecovery = TRUE)
})
```

`aws_user` *Get a user*

Description

Gets user information, including policies, groups, and attached policies

Usage

```
aws_user(username = NULL)
```

Arguments

`username`	(character) A user name. required

Details

See the following docs links for details

- https://www.paws-r-sdk.com/docs/iam_get_user/
- https://www.paws-r-sdk.com/docs/iam_list_user_policies/
- https://www.paws-r-sdk.com/docs/iam_list_groups_for_user/
- https://www.paws-r-sdk.com/docs/iam_list_attached_user_policies/

Value

a named list with slots for:

- user (tibble)
- policies (list)
- attached_policies (list)
- groups (list)

Note

if username not supplied, gets logged in user

See Also

Other users: aws_user_access_key(), aws_user_access_key_delete(), aws_user_add_to_group(), aws_user_create(), aws_user_current(), aws_user_delete(), aws_user_exists(), aws_users(), six_user_create(), six_user_delete()

Examples

```
## Not run:
# if username not supplied, gets the logged in user
aws_user()

## End(Not run)


if (aws_user_exists("testBlueBird")) {
  aws_user_delete("testBlueBird")
}
aws_user_create("testBlueBird")
aws_user("testBlueBird")

# cleanup
aws_user_delete("testBlueBird")
```

---

aws_users *List Users*

Description

List Users

Usage

```
aws_users(...)
```

Arguments

...	parameters passed on to the paws list_users method

Value

A tibble with information about user accounts, with columns:

- UserName
- UserId
- Path
- Arn
- CreateDate
- PasswordLastUsed

See Also

Other users: aws_user(), aws_user_access_key(), aws_user_access_key_delete(), aws_user_add_to_group(), aws_user_create(), aws_user_current(), aws_user_delete(), aws_user_exists(), six_user_create(), six_user_delete()

Examples

```
aws_users()
```

---

aws_user_access_key *Get AWS Access Key for a user*

Description

IMPORTANT: the secret access key is only accessible during key and user creation

Usage

```
aws_user_access_key(username = NULL, ...)
```

Arguments

username	(character) A user name. required
...	further named args passed on to list_access_keys

Details

See https://www.paws-r-sdk.com/docs/iam_list_access_keys/ docs for more details

Value

a tibble with key details

See Also

Other users: aws_user(), aws_user_access_key_delete(), aws_user_add_to_group(), aws_user_create(), aws_user_current(), aws_user_delete(), aws_user_exists(), aws_users(), six_user_create(), six_user_delete()

`aws_user_access_key_delete`

Delete current user's AWS Access Key

Description

Delete current user's AWS Access Key

Usage

```
aws_user_access_key_delete(access_key_id, username = NULL)
```

Arguments

`access_key_id`	(character) The access key ID for the access key ID and secret access key you want to delete. required.
`username`	(character) A user name. optional. however, if you do not supply a username, paws will likely use the current user, and so may not be the user the access key id is associated - and then you'll get an error like `NoSuchEntity (HTTP 404). The Access Key with id xx`

Details

See https://www.paws-r-sdk.com/docs/iam_delete_access_key/ docs for more details

Value

NULL, invisibly

See Also

Other users: aws_user(), aws_user_access_key(), aws_user_add_to_group(), aws_user_create(), aws_user_current(), aws_user_delete(), aws_user_exists(), aws_users(), six_user_create(), six_user_delete()

`aws_user_add_to_group` *Add or remove a user to/from a group*

Description

Add or remove a user to/from a group

Usage

```
aws_user_add_to_group(username, groupname)

aws_user_remove_from_group(username, groupname)
```

Arguments

`username`	(character) A user name. required
`groupname`	(character) a group name. required

Details

See https://www.paws-r-sdk.com/docs/iam_add_user_to_group/ https://www.paws-r-sdk.com/docs/iam_remove_user_from_group/ docs for more details

Value

a named list with slots for:

- user (tibble)
- policies (list)
- attached_policies (list)
- groups (list)

See Also

Other users: `aws_user()`, `aws_user_access_key()`, `aws_user_access_key_delete()`, `aws_user_create()`, `aws_user_current()`, `aws_user_delete()`, `aws_user_exists()`, `aws_users()`, `six_user_create()`, `six_user_delete()`

Examples

```
group1 <- random_string("group")
if (!aws_group_exists(group1)) {
  aws_group_create(group1)
}
name1 <- random_user()
if (!aws_user_exists(name1)) {
  aws_user_create(name1)
}
```

```
aws_user_add_to_group(name1, group1)
aws_group(group1) # has user name1
aws_user_remove_from_group(name1, group1)
aws_group(group1) # does not have user name1
```

`aws_user_create` *Create a user*

Description

Create a user

Usage

```
aws_user_create(username, path = NULL, permission_boundary = NULL, tags = NULL)
```

Arguments

`username`	(character) A user name. required
`path`	(character) The path for the user name. optional. If it is not included, it defaults to a slash (/).
`permission_boundary`	(character) The ARN of the managed policy that is used to set the permissions boundary for the user. optional
`tags`	(list) A list of tags that you want to attach to the new user. optional

Details

See https://www.paws-r-sdk.com/docs/iam_create_user/ docs for details on the parameters

Value

A tibble with information about the user created

See Also

Other users: `aws_user()`, `aws_user_access_key()`, `aws_user_access_key_delete()`, `aws_user_add_to_group()`, `aws_user_current()`, `aws_user_delete()`, `aws_user_exists()`, `aws_users()`, `six_user_create()`, `six_user_delete()`

Examples

```
user1 <- random_user()
if (aws_user_exists(user1)) {
  aws_user_delete(user1)
}
aws_user_create(user1)

# cleanup
aws_user_delete(user1)
```

`aws_user_current` *Get the current logged-in username as a string*

Description

Get the current logged-in username as a string

Usage

```
aws_user_current()
```

Value

username as character, scalar

See Also

Other users: `aws_user()`, `aws_user_access_key()`, `aws_user_access_key_delete()`, `aws_user_add_to_group()`, `aws_user_create()`, `aws_user_delete()`, `aws_user_exists()`, `aws_users()`, `six_user_create()`, `six_user_delete()`

`aws_user_delete` *Delete a user*

Description

Delete a user

Usage

```
aws_user_delete(username)
```

Arguments

`username`	(character) A user name. required

Details

See https://www.paws-r-sdk.com/docs/iam_delete_user/ docs for more details

Value

NULL invisibly

See Also

Other users: aws_user(), aws_user_access_key(), aws_user_access_key_delete(), aws_user_add_to_group(), aws_user_create(), aws_user_current(), aws_user_exists(), aws_users(), six_user_create(), six_user_delete()

Examples

```
user_name <- random_user()
aws_user_create(user_name)
aws_user_delete(user_name)
aws_user_exists(user_name)
```

---

aws_user_exists *Check if a user exists*

Description

Check if a user exists

Usage

```
aws_user_exists(username)
```

Arguments

username	(character) the user name

Details

uses aws_user() internally. see docs https://www.paws-r-sdk.com/docs/iam_get_user/

Value

a single boolean

See Also

Other users: aws_user(), aws_user_access_key(), aws_user_access_key_delete(), aws_user_add_to_group(), aws_user_create(), aws_user_current(), aws_user_delete(), aws_users(), six_user_create(), six_user_delete()

Examples

```
aws_user_exists(aws_user_current())
aws_user_exists("doesnotexist")
```

`aws_vpc` *Get a VPC by id*

Description

Get a VPC by id

Usage

```
aws_vpc(id, ...)
```

Arguments

`id`	(character) The id of the VPC. required
`...`	parameters passed on to describe_vpcs

Value

(list) with fields:

- Vpcs (list) each VPC group
- NextToken (character) token for paginating

Each element of Vpcs is a list with slots:

- CidrBlock
- DhcpOptionsId
- State
- VpcId
- OwnerId
- InstanceTenancy
- Ipv6CidrBlockAssociationSet
- CidrBlockAssociationSet
- IsDefault
- Tags

aws_vpcs *List VPCs*

Description

List VPCs

Usage

```
aws_vpcs(...)
```

Arguments

...	parameters passed on to describe_vpcs

Value

(list) list with VPCs, see `aws_vpc()` for details

Examples

```
aws_vpcs()
aws_vpcs(MaxResults = 6)
```

aws_vpc_security_group *Get a security group by ID*

Description

Get a security group by ID

Usage

```
aws_vpc_security_group(id, ...)
```

Arguments

id	(character) The id of the security group. required
...	named parameters passed on to describe_security_groups

Value

(list) with fields:

- SecurityGroups (list) each security group
  - Description
  - GroupName
  - IpPermissions
  - OwnerId
  - GroupId
  - IpPermissionsEgress
  - Tags
  - VpcId
- NextToken (character) token for paginating

See Also

Other security groups: aws_vpc_sec_group_rules_mod(), aws_vpc_security_group_create(), aws_vpc_security_group_ingress(), aws_vpc_security_groups(), aws_vpc_sg_with_ingress()

`aws_vpc_security_groups` *List VPC security groups*

Description

List VPC security groups

Usage

```
aws_vpc_security_groups(...)
```

Arguments

...	named parameters passed on to describe_security_groups

Value

(list) list with security groups, see aws_vpc_security_group() for details

See Also

Other security groups: aws_vpc_sec_group_rules_mod(), aws_vpc_security_group(), aws_vpc_security_group_cre
aws_vpc_security_group_ingress(), aws_vpc_sg_with_ingress()

Examples

```
aws_vpc_security_groups()
aws_vpc_security_groups(MaxResults = 6)
```

`aws_vpc_security_group_create` *Create a security group*

Description

Create a security group

Usage

```
aws_vpc_security_group_create(
  name,
  engine = "mariadb",
  description = NULL,
  vpc_id = NULL,
  tags = NULL,
  ...
)

aws_vpc_security_group_delete(id = NULL, name = NULL, ...)
```

Arguments

`name`	(character) The name of the new secret. required for `*_create` and optional for `*_delete`
`engine`	(character) The engine to use. default: "mariadb". required. one of: mariadb, mysql, or postgres
`description`	(character) The description of the secret. optional
`vpc_id`	(character) a VPC id. optional. if not supplied your default VPC is used. To get your VPCs, see `aws_vpcs()`
`tags`	(character) The tags to assign to the security group. optional
`...`	named parameters passed on to create_security_group
`id`	(character) The id of the security group. optional. provide `id` or `name`

Value

(list) with fields:

- GroupId (character)
- Tags (list)

See Also

Other security groups: aws_vpc_sec_group_rules_mod(), aws_vpc_security_group(), aws_vpc_security_group_ing
aws_vpc_security_groups(), aws_vpc_sg_with_ingress()

Examples

```
## Not run:
# create security group
grp_name1 <- random_string("vpcsecgroup")
x <- aws_vpc_security_group_create(
  name = grp_name1,
  description = "Testing security group creation"
)

grp_name2 <- random_string("vpcsecgroup")
aws_vpc_security_group_create(name = grp_name2)

grp_name3 <- random_string("vpcsecgroup")
aws_vpc_security_group_create(
  name = grp_name3,
  tags = list(
    list(
      ResourceType = "security-group",
      Tags = list(
        list(
          Key = "sky",
          Value = "blue"
        )
      )
    )
  )
)

# add ingress
aws_vpc_security_group_ingress(
  id = x$GroupId,
  ip_permissions = ip_permissions_generator("mariadb")
)

# cleanup
aws_vpc_security_group_delete(name = grp_name1)
aws_vpc_security_group_delete(name = grp_name2)
aws_vpc_security_group_delete(name = grp_name3)

## End(Not run)
```

---

aws_vpc_security_group_ingress

Authorize Security Group Ingress

Description

Authorize Security Group Ingress

Usage

```
aws_vpc_security_group_ingress(id, ip_permissions = NULL, ...)
```

Arguments

id	(character) security group id. required
ip_permissions	(list) list of persmissions. see link to paws docs below or use ip_permissions_generator() to generate the list for this parameter
...	named parameters passed on to authorize_security_group_ingress

Value

list with slots:

- Return (boolean)
- SecurityGroupRules (list)
  - SecurityGroupRuleId
  - GroupId
  - GroupOwnerId
  - IsEgress
  - IpProtocol
  - FromPort
  - ToPort
  - CidrIpv4
  - CidrIpv6
  - PrefixListId
  - ReferencedGroupInfo
  - Description
  - Tags

See Also

Other security groups: aws_vpc_sec_group_rules_mod(), aws_vpc_security_group(), aws_vpc_security_group_cre
aws_vpc_security_groups(), aws_vpc_sg_with_ingress()

`aws_vpc_sec_group_rules_mod` *Modify security group rules*

Description

Modify security group rules

Usage

```
aws_vpc_sec_group_rules_mod(id, rules, ...)
```

Arguments

`id`	(character) security group id. required
`rules`	list of rules to add/modify on the security group `id`. required
`...`	named parameters passed on to modify_security_group_rules

Value

list. if successful then `list(Return=TRUE)`

See Also

Other security groups: `aws_vpc_security_group()`, `aws_vpc_security_group_create()`, `aws_vpc_security_group_i`
`aws_vpc_security_groups()`, `aws_vpc_sg_with_ingress()`

Examples

```
# create a security group
a_grp_name <- random_string("vpcsecgroup")
x <- aws_vpc_security_group_create(name = a_grp_name)
x

# add an inbound rule
my_rule <- aws_vpc_security_group_ingress(
  id = x$GroupId,
  ip_permissions = ip_permissions_generator("mariadb")
)
my_rule

# modify the rule
rule_id <- my_rule$SecurityGroupRules[[1]]$SecurityGroupRuleId
fields_to_keep <- c(
  "IpProtocol", "FromPort", "ToPort", "CidrIpv4",
  "CidrIpv6", "PrefixListId", "Description"
)
rule_old <- my_rule$SecurityGroupRules[[1]]
```

```
rule_new <- rule_old[fields_to_keep]
rule_new$Description <- "Modified description"

aws_vpc_sec_group_rules_mod(
  id = x$GroupId,
  rules = list(
    SecurityGroupRuleId = rule_id,
    SecurityGroupRule = rule_new
  )
)

# cleanup
aws_vpc_security_group_delete(name = a_grp_name)
```

`aws_vpc_sg_with_ingress` *Get a security group with one ingress rule based on the engine*

Description

Get a security group with one ingress rule based on the engine

Usage

```
aws_vpc_sg_with_ingress(engine)
```

Arguments

`engine`	(character) The engine to use. default: "mariadb". required. one of: mariadb, mysql, postgres, or redshift

Details

Adds an ingress rule specific to the `engine` supplied (port changes based on the engine), and your IP address. To create your own security group and ingress rules see `aws_vpc_security_group_create()` and `aws_vpc_security_group_ingress()`

Value

(character) security group ID

See Also

Other security groups: `aws_vpc_sec_group_rules_mod()`, `aws_vpc_security_group()`, `aws_vpc_security_group_cre`
`aws_vpc_security_group_ingress()`, `aws_vpc_security_groups()`

bucket_arn *Get bucket ARN*

Description

Get bucket ARN

Usage

```
bucket_arn(bucket, objects = "")
```

Arguments

bucket	(character) a bucket name. required.
objects	(character) path for object(s). default: ""

Value

character string of bucket arn

Examples

```
bucket_arn("somebucket")
bucket_arn("somebucket", objects = "*")
bucket_arn("somebucket", objects = "data.csv")
bucket_arn("somebucket", objects = "myfolder/subset/data.csv")
bucket_arn("somebucket", objects = "myfolder/subset/*")
```

con_iam *Get a* paws *client for a service*

Description

Get a paws client for a service

Usage

```
con_iam()

con_s3()

con_sm()

con_ec2()

con_rds()
```

```
con_redshift()

con_ce()
```

Details

Toggles the credentials used based on the environment variable `AWS_PROFILE` for one of: minio, localstack, aws.

If `AWS_PROFILE` is "minio" then we set the following in the credentials for the connection:

- `access_key_id` uses env var `MINIO_USER`, with default "minioadmin"
- `secret_access_key` uses env var `MINIO_PWD`, with default "minioadmin"
- `endpoint` uses env var `MINIO_ENDPOINT`, with default "http://127.0.0.1:9000"

If `AWS_PROFILE` is "localstack" then we set the following in the credentials for the connection:

- `access_key_id` uses env var `LOCALSTACK_KEY`, with a default string which is essentially ignored. you do not need to set the `LOCALSTACK_KEY` env var. However, if you want to set an account ID for your Localstack you can set the env var and it will be used. see https://docs.localstack.cloud/references/credentials/
- `secret_access_key` uses env var `LOCALSTACK_SECRET`, with a default string which is ignored; and any value you set for `LOCALSTACK_SECRET` will be ignored by Localstack as well. see https://docs.localstack.cloud/references/credentials/
- `endpoint` uses env var `LOCALSTACK_ENDPOINT`. You can set this to the URL for where your Localstack is running at. Default is `http://localhost.localstack.cloud:4566`

If `AWS_PROFILE` is not set, set to "aws", or anything else (other than "localstack") then we don't set any credentials internally, but paws will gather any credentials you've set via env vars, config files, etc.-

Value

- `con_s3`: a list with methods for interfacing with S3; https://www.paws-r-sdk.com/docs/s3/
- `con_iam`: a list with methods for interfacing with IAM; https://www.paws-r-sdk.com/docs/iam/
- `con_sm`: a list with methods for interfacing with Secrets Manager; https://www.paws-r-sdk.com/docs/secretsmanager/
- `con_ec2`: a list with methods for interfacing with EC2; https://www.paws-r-sdk.com/docs/ec2/
- `con_rds`: a list with methods for interfacing with RDS; https://www.paws-r-sdk.com/docs/rds/
- `con_redshift`: a list with methods for interfacing with Redshift; https://www.paws-r-sdk.com/docs/redshift/
- `con_ce`: a list with methods for interfacing with Cost Explorer; https://www.paws-r-sdk.com/docs/costexplorer/

See Also

con_s3fs()

Examples

```
z <- con_iam()
z

withr::with_envvar(
  c("AWS_PROFILE" = "localstack"),
  con_iam()
)
withr::with_envvar(
  c("AWS_PROFILE" = "minio"),
  con_s3()
)
```

---

con_s3fs *s3fs connection*

Description

s3fs connection

Usage

```
con_s3fs()
```

Details

we set `refresh=TRUE` on `s3fs::s3_file_system()` so that you can change the s3 interface within an R session

You can toggle the interface set for one of minio, localstack, aws. See connections for more information.

Value

An S3 list with class 'sixtyfour_client'

See Also

paws_clients

Examples

```
con <- con_s3fs()
con
con_s3fs()$file_copy
```

figure_out_policy_arn *Figure out policy Arn from a name*

Description

Figure out policy Arn from a name

Usage

```
figure_out_policy_arn(name)
```

Arguments

name	(character) a policy name. required.

Value

NULL when not found; otherwise an ARN string

Examples

```
# aws managed
figure_out_policy_arn("AmazonS3ReadOnlyAccess")
# aws managed, job function
figure_out_policy_arn("Billing")
figure_out_policy_arn("DataScientist")
# doesn't exist
figure_out_policy_arn("DoesNotExist")
```

group_policies *Preset group policies*

Description

Preset group policies

Usage

```
group_policies(group)
```

Arguments

group	(character)

Value

character vector of policy names

Admin group policies

- AdministratorAccess
- Billing
- CostOptimizationHubAdminAccess
- AWSBillingReadOnlyAccess
- AWSCostAndUsageReportAutomationPolicy

User group policies

- AmazonRDSReadOnlyAccess
- AmazonRedshiftReadOnlyAccess
- AmazonS3ReadOnlyAccess
- AWSBillingReadOnlyAccess
- IAMReadOnlyAccess

Examples

```
group_policies("admin")
group_policies("users")
```

---

ip_permissions_generator

Ip Permissions generator

Description

Ip Permissions generator

Usage

```
ip_permissions_generator(engine, port = NULL, description = NULL)
```

Arguments

`engine`	(character) one of mariadb, mysql, or postgres
`port`	(character) port number. port determined from `engine` if `port` not given. default: `NULL`
`description`	(character) description. if not given, autogenerated depending on value of `engine`

Value

a list with slots: FromPort, ToPort, IpProtocol, and IpRanges

`random_string` *Get a random string, bucket name, user name or role name*

Description

Get a random string, bucket name, user name or role name

Usage

```
random_string(prefix, size = 8)

random_bucket(prefix = "bucket-", size = 16)

random_user()

random_role()
```

Arguments

`prefix`	(character) any string. required.
`size`	(character) length of the random part (not including `prefix`)

Value

- `random_string`: (character) a string with `prefix` at beginning
- `random_bucket`: (character) a bucket name prefixed with `prefix` (default: "bucket-")
- `random_user`/`random_role`: (character) a user or role name with a random adjective plus a random noun combined into one string, shortened to no longer than 16 characters, if longer than 16

Examples

```
random_string("group-")
replicate(10, random_string("group-"))
random_bucket()
replicate(10, random_bucket())
random_user()
replicate(10, random_user())
random_role()
replicate(10, random_role())
```

resource_rds *Create a resource string for a policy statement for RDS*

Description

Create a resource string for a policy statement for RDS

Usage

```
resource_rds(
  user,
  resource_id,
  region = Sys.getenv("AWS_REGION"),
  account = account_id()
)
```

Arguments

user	(character) a user name that has an IAM account. length>=1. required
resource_id	(character) the identifier for the DB instance. length==1. required
region	(character) the AWS Region for the DB instance. length==1
account	(character) the AWS account number for the DB instance. length==1. The user must be in the same account as the account for the DB instance. by default calls account_id()

Value

a resource ARN (scalar, character)

s3_actions_full *S3 actions for full access (read and write), from the AWS managed policy* AmazonS3FullAccess

Description

S3 actions for full access (read and write), from the AWS managed policy AmazonS3FullAccess

Usage

```
s3_actions_full()
```

Value

character vector of actions

Examples

```
s3_actions_full()
```

`s3_actions_read` *S3 actions for reading, from the AWS managed policy* `AmazonS3ReadOnlyAccess`

Description

S3 actions for reading, from the AWS managed policy `AmazonS3ReadOnlyAccess`

Usage

```
s3_actions_read()
```

Value

character vector of actions

Examples

```
s3_actions_read()
```

`service_map` *Mapping of full names of AWS services to acronyms*

Description

Mapping of full names of AWS services to acronyms

Usage

```
service_map
```

Format

`service_map`**:**
A data frame with 178 rows and 2 columns:

service Service name in full

acronym The acronym, from 2 to 5 characters in length ...

Source

https://tommymaynard.com/aws-service-acronyms/

six_admin_setup *AWS account setup for administrators*

Description

AWS account setup for administrators

Usage

```
six_admin_setup(users_group = "users", admin_group = "admin")
```

Arguments

users_group	(character) name for the users group. default: "users"
admin_group	(character) name for the admin group. default: "admin"

Value

NULL invisibly

What is magical

- Setup a users IAM group: users that do not require admin persmissions
- Add policies to the users group
- Setup an admin IAM group: users that require admin permissions
- Add policies to the admin group

See Also

Other magicians: `six_bucket_delete()`, `six_bucket_upload()`, `six_file_upload()`, `six_user_create()`, `six_user_delete()`

six_bucket_add_user *Add a user to a bucket*

Description

Add a user to a bucket

Usage

```
six_bucket_add_user(bucket, username, permissions)
```

Arguments

bucket	(character) bucket name. required
username	(character) A user name. required
permissions	(character) user permissions, one of read or write. write includes read

Value

invisibly returns nothing

Permissions

- read: read only; not allowed to write or do admin tasks
- write: write (in addition to read); includes deleting files; does not include deleting buckets
- admin: change user permissions (in addition to read and write); includes deleting buckets (THIS OPTION NOT ACCEPTED YET!)

What is magical

- Exits early if permissions is not length 1
- Exits early if permissions is not in allowed set
- Exits early if bucket does not exist
- Creates bucket policy if not created yet
- If user not in bucket already, attach policy to user (which adds them to the bucket)

Examples

```
# create a bucket
bucket <- random_bucket()
if (!aws_bucket_exists(bucket)) {
  aws_bucket_create(bucket)
}

# create a user
user <- random_user()
if (!aws_user_exists(user)) {
  aws_user_create(user)
}

six_bucket_add_user(
  bucket = bucket,
  username = user,
  permissions = "read"
)

# cleanup
six_user_delete(user)
aws_bucket_delete(bucket, force = TRUE)
```

```
## Not run:
# not a valid permissions string
six_bucket_add_user(
  bucket = "mybucket",
  username = "userdmgziqpt",
  permissions = "notavalidpermission"
)

## End(Not run)
```

---

six_bucket_change_user

Change user permissions for a bucket

Description

Change user permissions for a bucket

Usage

```
six_bucket_change_user(bucket, username, permissions)
```

Arguments

bucket	(character) bucket name. required
username	(character) A user name. required
permissions	(character) user permissions, one of read or write. write includes read

Value

invisibly returns nothing

Important

This function is built around policies named by this package. If you use your own policies that you name this function may not work.

Examples

```
# create a bucket
bucket <- random_bucket()
if (!aws_bucket_exists(bucket)) {
  aws_bucket_create(bucket)
}

# create user
user <- random_user()
if (!aws_user_exists(user)) {
```

```
  aws_user_create(user)
}

# user doesn't have any permissions for the bucket
# - use six_bucket_add_user to add permissions
six_bucket_change_user(
  bucket = bucket,
  username = user, permissions = "read"
)
six_bucket_add_user(
  bucket = bucket, username = user,
  permissions = "read"
)

# want to change to read to write, makes the change
six_bucket_change_user(
  bucket = bucket, username = user,
  permissions = "write"
)

# want to change to write - but already has write
six_bucket_change_user(
  bucket = bucket, username = user,
  permissions = "write"
)

# cleanup
six_user_delete(user)
aws_bucket_delete(bucket, force = TRUE)
```

`six_bucket_delete` *Delete an S3 bucket*

Description

Takes care of deleting bucket objects, so that the bucket itself can be deleted cleanly

Usage

```
six_bucket_delete(bucket, force = FALSE, ...)
```

Arguments

`bucket`	(character) bucket name. required
`force`	(logical) force deletion without going through the prompt. default: `FALSE`. Should only be set to `TRUE` when required for non-interactive use.
`...`	named parameters passed on to delete_bucket

Value

NULL, invisibly

What is magical

- Exits early if bucket does not exist
- Checks for any objects in the bucket and deletes any present
- Deletes bucket after deleting objects

See Also

Other buckets: aws_bucket_create(), aws_bucket_delete(), aws_bucket_download(), aws_bucket_exists(), aws_bucket_list_objects(), aws_bucket_tree(), aws_bucket_upload(), aws_buckets(), six_bucket_upload()

Other magicians: six_admin_setup(), six_bucket_upload(), six_file_upload(), six_user_create(), six_user_delete()

Examples

```
# bucket does not exist
six_bucket_delete("notabucket")

# bucket exists w/o objects
bucket <- random_bucket()
aws_bucket_create(bucket)
six_bucket_delete(bucket, force = TRUE)

# bucket exists w/ objects (files and directories with files)
bucket <- random_bucket()
aws_bucket_create(bucket)
demo_rds_file <- file.path(system.file(), "Meta/demo.rds")
links_file <- file.path(system.file(), "Meta/links.rds")
aws_file_upload(
  c(demo_rds_file, links_file),
  s3_path(bucket, c(basename(demo_rds_file), basename(links_file)))
)
aws_file_upload(
  c(demo_rds_file, links_file),
  s3_path(
    bucket, "newfolder",
    c(basename(demo_rds_file), basename(links_file))
  )
)
aws_bucket_list_objects(bucket)
six_bucket_delete(bucket, force = TRUE)
```

six_bucket_permissions *Get permissions for a bucket*

Description

Get permissions for a bucket

Usage

```
six_bucket_permissions(bucket)
```

Arguments

bucket	(character) bucket name. required

Value

tibble with a row for each user, with columns:

- user (always present)
- permissions (always present)
- policy_read (optionally present) the policy name behind the "read" permission (if present)
- policy_admin (optionally present) the policy name behind the "admin" permission (if present)

Note that users with no persmissions are not shown; see aws_users()

Examples

```
# create a bucket
bucket <- random_bucket()
if (!aws_bucket_exists(bucket)) aws_bucket_create(bucket)

# create user
user <- random_user()
if (!aws_user_exists(user)) aws_user_create(user)

six_bucket_permissions(bucket)
six_bucket_add_user(bucket, user, permissions = "read")
six_bucket_permissions(bucket)
six_bucket_remove_user(bucket, user)
six_bucket_permissions(bucket)

# cleanup
six_user_delete(user)
aws_bucket_delete(bucket, force = TRUE)
```

six_bucket_remove_user *Remove a user from a bucket*

Description

Remove a user from a bucket

Usage

```
six_bucket_remove_user(bucket, username)
```

Arguments

bucket	(character) bucket name. required
username	(character) A user name. required

Details

This function detaches a policy from a user for accessing the bucket; the policy itself is untouched

Value

invisibly returns nothing

Examples

```
# create a bucket
bucket <- random_bucket()
if (!aws_bucket_exists(bucket)) aws_bucket_create(bucket)

# create user
user <- random_user()
if (!aws_user_exists(user)) aws_user_create(user)

six_bucket_add_user(bucket, user, permissions = "read")
six_bucket_remove_user(bucket, user)

# cleanup
six_user_delete(user)
aws_bucket_delete(bucket, force = TRUE)
```

six_bucket_upload *Magically upload a mix of files and directories into a bucket*

Description

Magically upload a mix of files and directories into a bucket

Usage

```
six_bucket_upload(path, remote, force = FALSE, ...)
```

Arguments

path	(character) one or more file paths to add to the bucket. required. can include directories or files
remote	(character/scalar) a character string to use to upload files in path. the first component of the path will be used as the bucket name. any subsequent path components will be used as a key prefix for all objects created in the bucket
force	(logical) force bucket creation without going through the prompt. default: FALSE. Should only be set to TRUE when required for non-interactive use.
...	named params passed on to put_object

Value

(character) a vector of remote s3 paths where your files are located

What is magical

- Exits early if folder or files do not exist
- Creates the bucket if it does not exist
- Adds files to the bucket at the top level with key as the file name
- Adds directories to the bucket, reconstructing the exact directory structure in the S3 bucket

See Also

Other buckets: aws_bucket_create(), aws_bucket_delete(), aws_bucket_download(), aws_bucket_exists(), aws_bucket_list_objects(), aws_bucket_tree(), aws_bucket_upload(), aws_buckets(), six_bucket_delete()

Other magicians: six_admin_setup(), six_bucket_delete(), six_file_upload(), six_user_create(), six_user_delete()

Examples

```
# single file, single remote path
bucket1 <- random_bucket()
demo_rds_file <- file.path(system.file(), "Meta/demo.rds")
six_bucket_upload(path = demo_rds_file, remote = bucket1, force = TRUE)

## a file and a directory - with a single remote path
bucket2 <- random_bucket()
library(fs)
tdir <- path(path_temp(), "mytmp")
dir_create(tdir)
invisible(purrr::map(letters, \(l) file_create(path(tdir, l))))
dir_tree(tdir)
six_bucket_upload(path = c(demo_rds_file, tdir), remote = bucket2,
force = TRUE)

## a directory with nested dirs - with a single remote path
bucket3 <- random_bucket()
library(fs)
tdir <- path(path_temp(), "apples")
dir_create(tdir)
dir_create(path(tdir, "mcintosh"))
dir_create(path(tdir, "pink-lady"))
cat("Some text in a readme", file = path(tdir, "README.md"))
write.csv(Orange, file = path(tdir, "mcintosh", "orange.csv"))
write.csv(iris, file = path(tdir, "pink-lady", "iris.csv"))
dir_tree(tdir)
six_bucket_upload(path = tdir, remote = path(bucket3, "fruit/basket"),
force = TRUE)

# cleanup
six_bucket_delete(bucket1, force = TRUE)
six_bucket_delete(bucket2, force = TRUE)
six_bucket_delete(bucket3, force = TRUE)
```

---

six_file_upload *Magically upload a file*

Description

Magically upload a file

Usage

```
six_file_upload(path, bucket, force = FALSE, ...)
```

Arguments

path	(character) one or more file paths to add to the bucket. required. cannot include directories
bucket	(character) bucket to copy files to. required. if the bucket does not exist we prompt you asking if you'd like the bucket to be created
force	(logical) force bucket creation without going through the prompt. default: FALSE. Should only be set to TRUE when required for non-interactive use.
...	named params passed on to put_object

Value

(character) a vector of remote s3 paths where your files are located

What is magical

- Exits early if files do not exist
- Exits early if any path values are directories
- Creates the bucket if it does not exist
- Adds files to the bucket, figuring out the key to use from the supplied path
- Function is vectoried for the path argument; you can pass in many file paths

See Also

Other files: aws_file_attr(), aws_file_copy(), aws_file_delete(), aws_file_download(), aws_file_exists(), aws_file_rename(), aws_file_upload()

Other magicians: six_admin_setup(), six_bucket_delete(), six_bucket_upload(), six_user_create(), six_user_delete()

Examples

```
bucket1 <- random_bucket()
demo_rds_file <- file.path(system.file(), "Meta/demo.rds")
six_file_upload(demo_rds_file, bucket1, force = TRUE)

# path doesn't exist, error
try(
  six_file_upload("file_doesnt_exist.txt", random_bucket())
)

# directories not supported, error
mydir <- tempdir()
try(
  six_file_upload(mydir, random_bucket())
)

# Cleanup
six_bucket_delete(bucket1, force = TRUE)
```

```
# requires user interaction with prompts ...
bucket2 <- random_bucket()
demo_rds_file <- file.path(system.file(), "Meta/demo.rds")
six_file_upload(demo_rds_file, bucket2)

## many files at once
links_file <- file.path(system.file(), "Meta/links.rds")
six_file_upload(c(demo_rds_file, links_file), bucket2)

# set expiration, expire 1 minute from now
six_file_upload(demo_rds_file, bucket2, Expires = Sys.time() + 60)

# bucket doesn't exist, ask if you want to create it
not_a_bucket <- random_string("not-a-bucket-")
six_file_upload(demo_rds_file, not_a_bucket)

# Cleanup
six_bucket_delete(bucket2, force = TRUE)
six_bucket_delete(not_a_bucket, force = TRUE)
```

`six_group_delete` *Delete a group, magically*

Description

Delete a group, magically

Usage

```
six_group_delete(name)
```

Arguments

`name`	(character) A group name. required

Details

See https://www.paws-r-sdk.com/docs/iam_delete_group/ docs for more details

Value

`NULL` invisibly

See Also

Other groups: `aws_group()`, `aws_group_create()`, `aws_group_delete()`, `aws_group_exists()`, `aws_groups()`

Examples

```
group <- random_string("group")
aws_group_create(group)
six_group_delete(group)
```

`six_user_create` *Create a user, magically*

Description

Create a user, magically

Usage

```
six_user_create(
  username,
  path = NULL,
  permission_boundary = NULL,
  tags = NULL,
  copy_to_cb = TRUE
)
```

Arguments

`username`	(character) A user name. required
`path`	(character) The path for the user name. optional. If it is not included, it defaults to a slash (/).
`permission_boundary`	(character) The ARN of the managed policy that is used to set the permissions boundary for the user. optional
`tags`	(list) A list of tags that you want to attach to the new user. optional
`copy_to_cb`	(logical) Copy to clipboard. Default: `FALSE`. See section "Clipboard" below for more details.

Details

See `aws_user_create()` for more details. This function creates a user, adds policies so the user can access their own account, and grants them an access key. Add more policies using `aws_polic*` functions

Value

NULL invisibly. A draft email is copied to your clipboard

What is magical

- Adds a `UserInfo` policy to your account if doesn't exist yet
- Attaches `UserInfo` policy to the user created
- Grants an access key, copying an email template to your clipboard

See Also

Other users: `aws_user()`, `aws_user_access_key()`, `aws_user_access_key_delete()`, `aws_user_add_to_group()`, `aws_user_create()`, `aws_user_current()`, `aws_user_delete()`, `aws_user_exists()`, `aws_users()`, `six_user_delete()`

Other magicians: `six_admin_setup()`, `six_bucket_delete()`, `six_bucket_upload()`, `six_file_upload()`, `six_user_delete()`

Examples

```
name <- random_user()
six_user_create(name)

# cleanup
six_user_delete(name)
```

---

six_user_creds *Create access keys for a user*

Description

Creates a new Amazon Web Services secret access key and corresponding Amazon Web Services access key ID

Usage

```
six_user_creds(username, copy_to_cb = FALSE)
```

Arguments

`username`	(character) A user name. required
`copy_to_cb`	(logical) Copy to clipboard. Default: `FALSE`. See section "Clipboard" below for more details.

Details

A user can have more than one pair of access keys. By default a user can have up to 2 pairs of access keys. Using this function will not replace an existing set of keys; but instead adds an additional set of keys.

See https://rstats.wtf/r-startup.html for help on bringing in secrets to an R session.

Note that although we return the AWS Region in the output of this function IAM does not have regional resources. You can however use IAM to manage regions an account has access to, etc. See https://docs.aws.amazon.com/accounts/latest/reference/manage-acct-regions.html #nolint

Value

invisibly returns named list with slots:

- UserName (character)
- AccessKeyId (character)
- Status (character)
- SecretAccessKey (character)
- CreateDate (POSIXct)

Important

Save the secret key after running this function as it can not be viewed again.

Clipboard

If you set `copy_to_cb=TRUE` we'll copy to your clipboard an email template with the credentials and a small amount of instructions. Please do edit that email with information tailored to your group and how you'd like to store secrets

Known error behaviors

- `LimitExceeded (HTTP 409). Cannot exceed quota for AccessKeysPerUser: 2`
- `NoSuchEntity (HTTP 404). The user with name xxx cannot be found.`

See Also

`aws_user_access_key()`, `aws_user_access_key_delete()`

Examples

```
user <- random_user()
if (!aws_user_exists(user)) aws_user_create(user)
six_user_creds(user)
aws_user_access_key(user)
six_user_creds(user, copy_to_cb = TRUE)
aws_user_access_key(user)
# cleanup
```

```
six_user_delete(user)
```

`six_user_delete` *Delete a user*

Description

Delete a user

Usage

```
six_user_delete(username)
```

Arguments

`username`	(character) A user name. required

Details

See https://www.paws-r-sdk.com/docs/iam_delete_user/ docs for more details

Value

an empty list

What is magical

- Detaches any attached policies
- Deletes any access keys
- Then deletes the user

See Also

Other users: `aws_user()`, `aws_user_access_key()`, `aws_user_access_key_delete()`, `aws_user_add_to_group()`, `aws_user_create()`, `aws_user_current()`, `aws_user_delete()`, `aws_user_exists()`, `aws_users()`, `six_user_create()`

Other magicians: `six_admin_setup()`, `six_bucket_delete()`, `six_bucket_upload()`, `six_file_upload()`, `six_user_create()`

Examples

```
name <- random_user()
six_user_create(name)
six_user_delete(name)
```

without_verbose *Without verbose output*

Description

Without verbose output

Usage

```
without_verbose(code)
```

Arguments

code	(expression) Code to run without verbose output.

Value

The results of the evaluation of the code argument

with_redacted *With secrets redacted*

Description

With secrets redacted

Usage

```
with_redacted(code)
```

Arguments

code	(expression) Code to run with secrets redacted

Value

The results of the evaluation of the code argument

Index