Package 'brickster'

November 13, 2024

Title R Toolkit for 'Databricks'

Version 0.2.5

Description Collection of utilities that improve using 'Databricks' from R.
Primarily functions that wrap specific 'Databricks' APIs
(<https://docs.databricks.com/api>), 'RStudio' connection pane support, quality
of life functions to make 'Databricks' simpler to use.

License Apache License (>= 2)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports arrow, base64enc, cli, curl, dplyr, glue, httr2, ini,
jsonlite, purrr, reticulate, R6 (>= 2.4.0), rlang, tibble,
utils

Suggests testthat (>= 3.0.0), huxtable, htmltools, knitr, magick,
rmarkdown, rstudioapi, withr

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://github.com/databrickslabs/brickster

Config/testthat/edition 3

NeedsCompilation no

Author Zac Davies [aut, cre],
Rafi Kurlansik [aut],
Databricks [cph, fnd]

Maintainer Zac Davies <zac@databricks.com>

Repository CRAN

Date/Publication 2024-11-13 14:10:06 UTC

Contents

`access_control_request` *Access Control Request*

Description

Access Control Request

Usage

```
access_control_request(...)
```

Arguments

...	Instances of `access_control_req_user()` or `access_control_req_group()`.

See Also

`db_jobs_create()`, `db_jobs_reset()`, `db_jobs_update()`

access_control_req_group

Access Control Request for Group

Description

Access Control Request for Group

Usage

```
access_control_req_group(
  group,
  permission_level = c("CAN_MANAGE", "CAN_MANAGE_RUN", "CAN_VIEW")
)
```

Arguments

group	Group name. There are two built-in groups: `users` for all users, and `admins` for administrators.
permission_level	Permission level to grant. One of `CAN_MANAGE`, `CAN_MANAGE_RUN`, `CAN_VIEW`.

See Also

`access_control_request()`

Other Access Control Request Objects: `access_control_req_user()`

access_control_req_user

Access Control Request For User

Description

Access Control Request For User

Usage

```
access_control_req_user(
  user_name,
  permission_level = c("CAN_MANAGE", "CAN_MANAGE_RUN", "CAN_VIEW", "IS_OWNER")
)
```

Arguments

user_name	Email address for the user.
permission_level	Permission level to grant. One of CAN_MANAGE, CAN_MANAGE_RUN, CAN_VIEW, IS_OWNER.

See Also

access_control_request()

Other Access Control Request Objects: access_control_req_group()

---

add_lib_path *Add Library Path*

Description

Add Library Path

Usage

```
add_lib_path(path, after, version = FALSE)
```

Arguments

path	Directory that will added as location for which packages are searched. Recursively creates the directory if it doesn't exist. On Databricks remember to use /dbfs/ or /Volumes/... as a prefix.
after	Location at which to append the path value after.
version	If TRUE will add the R version string to the end of path. This is recommended if using different R versions and sharing a common path between users.

Details

This functions primary use is when using Databricks notebooks or hosted RStudio, however, it works anywhere.

See Also

base::.libPaths(), remove_lib_path()

aws_attributes *AWS Attributes*

Description

AWS Attributes

Usage

```
aws_attributes(
  first_on_demand = 1,
  availability = c("SPOT_WITH_FALLBACK", "SPOT", "ON_DEMAND"),
  zone_id = NULL,
  instance_profile_arn = NULL,
  spot_bid_price_percent = 100,
  ebs_volume_type = c("GENERAL_PURPOSE_SSD", "THROUGHPUT_OPTIMIZED_HDD"),
  ebs_volume_count = 1,
  ebs_volume_size = NULL,
  ebs_volume_iops = NULL,
  ebs_volume_throughput = NULL
)
```

Arguments

`first_on_demand`
: Number of nodes of the cluster that will be placed on on-demand instances. If this value is greater than 0, the cluster driver node will be placed on an on-demand instance. If this value is greater than or equal to the current cluster size, all nodes will be placed on on-demand instances. If this value is less than the current cluster size, `first_on_demand` nodes will be placed on on-demand instances and the remainder will be placed on availability instances. This value does not affect cluster size and cannot be mutated over the lifetime of a cluster.

`availability`
: One of `SPOT_WITH_FALLBACK`, `SPOT`, `ON_DEMAND`. Type used for all subsequent nodes past the `first_on_demand` ones. If `first_on_demand` is zero, this availability type will be used for the entire cluster.

`zone_id`
: Identifier for the availability zone/datacenter in which the cluster resides. You have three options: availability zone in same region as the Databricks deployment, `auto` which selects based on available IPs, `NULL` which will use the default availability zone.

`instance_profile_arn`
: Nodes for this cluster will only be placed on AWS instances with this instance profile. If omitted, nodes will be placed on instances without an instance profile. The instance profile must have previously been added to the Databricks environment by an account administrator. This feature may only be available to certain customer plans.

`spot_bid_price_percent`

The max price for AWS spot instances, as a percentage of the corresponding instance type's on-demand price. For example, if this field is set to 50, and the cluster needs a new i3.xlarge spot instance, then the max price is half of the price of on-demand i3.xlarge instances. Similarly, if this field is set to 200, the max price is twice the price of on-demand i3.xlarge instances. If not specified, the default value is 100. When spot instances are requested for this cluster, only spot instances whose max price percentage matches this field will be considered. For safety, we enforce this field to be no more than 10000.

`ebs_volume_type`

Either `GENERAL_PURPOSE_SSD` or `THROUGHPUT_OPTIMIZED_HDD`

`ebs_volume_count`

The number of volumes launched for each instance. You can choose up to 10 volumes. This feature is only enabled for supported node types. Legacy node types cannot specify custom EBS volumes. For node types with no instance store, at least one EBS volume needs to be specified; otherwise, cluster creation will fail. These EBS volumes will be mounted at `/ebs0`, `/ebs1`, and etc. Instance store volumes will be mounted at `/local_disk0`, `/local_disk1`, and etc.

If EBS volumes are attached, Databricks will configure Spark to use only the EBS volumes for scratch storage because heterogeneously sized scratch devices can lead to inefficient disk utilization. If no EBS volumes are attached, Databricks will configure Spark to use instance store volumes.

If EBS volumes are specified, then the Spark configuration `spark.local.dir` will be overridden.

`ebs_volume_size`

The size of each EBS volume (in `GiB`) launched for each instance. For general purpose SSD, this value must be within the range `100 - 4096`. For throughput optimized HDD, this value must be within the range `500 - 4096`.

Custom EBS volumes cannot be specified for the legacy node types (memory-optimized and compute-optimized).

`ebs_volume_iops`

The number of IOPS per EBS gp3 volume. This value must be between 3000 and 16000. The value of IOPS and throughput is calculated based on AWS documentation to match the maximum performance of a gp2 volume with the same volume size.

`ebs_volume_throughput`

The throughput per EBS gp3 volume, in `MiB` per second. This value must be between 125 and 1000.

Details

If `ebs_volume_iops`, `ebs_volume_throughput`, or both are not specified, the values will be inferred from the throughput and IOPS of a gp2 volume with the same disk size, by using the following calculation:

Disk size	IOPS	Throughput
Greater than 1000	3 times the disk size up to 16000	250

Between 170 and 1000	3000	250
Below 170	3000	128

See Also

db_cluster_create(), db_cluster_edit()

Other Cloud Attributes: azure_attributes(), gcp_attributes()

---

azure_attributes *Azure Attributes*

Description

Azure Attributes

Usage

```
azure_attributes(
  first_on_demand = 1,
  availability = c("SPOT_WITH_FALLBACK", "SPOT", "ON_DEMAND"),
  spot_bid_max_price = -1
)
```

Arguments

`first_on_demand`
: Number of nodes of the cluster that will be placed on on-demand instances. If this value is greater than 0, the cluster driver node will be placed on an on-demand instance. If this value is greater than or equal to the current cluster size, all nodes will be placed on on-demand instances. If this value is less than the current cluster size, `first_on_demand` nodes will be placed on on-demand instances and the remainder will be placed on availability instances. This value does not affect cluster size and cannot be mutated over the lifetime of a cluster.

`availability`
: One of `SPOT_WITH_FALLBACK`, `SPOT`, `ON_DEMAND`. Type used for all subsequent nodes past the `first_on_demand` ones. If `first_on_demand` is zero, this availability type will be used for the entire cluster.

`spot_bid_max_price`
: The max bid price used for Azure spot instances. You can set this to greater than or equal to the current spot price. You can also set this to -1 (the default), which specifies that the instance cannot be evicted on the basis of price. The price for the instance will be the current price for spot instances or the price for a standard instance. You can view historical pricing and eviction rates in the Azure portal.

See Also

db_cluster_create(), db_cluster_edit()

Other Cloud Attributes: aws_attributes(), gcp_attributes()

close_workspace *Close Databricks Workspace Connection*

Description

Close Databricks Workspace Connection

Usage

```
close_workspace(host = db_host())
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().

Examples

```
## Not run:
close_workspace(host = db_host())

## End(Not run)
```

cluster_autoscale *Cluster Autoscale*

Description

Range defining the min and max number of cluster workers.

Usage

```
cluster_autoscale(min_workers, max_workers)
```

Arguments

min_workers	The minimum number of workers to which the cluster can scale down when underutilized. It is also the initial number of workers the cluster will have after creation.
max_workers	The maximum number of workers to which the cluster can scale up when overloaded. max_workers must be strictly greater than min_workers.

See Also

db_cluster_create(), db_cluster_edit()

cluster_log_conf *Cluster Log Configuration*

Description

Path to cluster log.

Usage

```
cluster_log_conf(dbfs = NULL, s3 = NULL)
```

Arguments

dbfs	Instance of dbfs_storage_info().
s3	Instance of s3_storage_info().

Details

dbfs and s3 are mutually exclusive, logs can only be sent to one destination.

See Also

Other Cluster Log Configuration Objects: dbfs_storage_info(), s3_storage_info()

cron_schedule *Cron Schedule*

Description

Cron Schedule

Usage

```
cron_schedule(
  quartz_cron_expression,
  timezone_id = "Etc/UTC",
  pause_status = c("UNPAUSED", "PAUSED")
)
```

Arguments

quartz_cron_expression	Cron expression using Quartz syntax that describes the schedule for a job. See Cron Trigger for details.
timezone_id	Java timezone ID. The schedule for a job is resolved with respect to this timezone. See Java TimeZone for details.
pause_status	Indicate whether this schedule is paused or not. Either UNPAUSED (default) or PAUSED.

See Also

db_jobs_create(), db_jobs_reset(), db_jobs_update()

DatabricksSqlClient *Databricks SQL Connector*

Description

Wraps the databricks-sql-connector using reticulate.

API reference on Databricks docs

Methods

Public methods:

- DatabricksSqlClient$new()
- DatabricksSqlClient$columns()
- DatabricksSqlClient$catalogs()
- DatabricksSqlClient$schemas()
- DatabricksSqlClient$tables()
- DatabricksSqlClient$execute()
- DatabricksSqlClient$execute_many()
- DatabricksSqlClient$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Note that this object is typically constructed via db_sql_client().

Usage:

```
DatabricksSqlClient$new(
  host,
  token,
  http_path,
  catalog,
  schema,
  use_cloud_fetch,
  session_configuration,
  ...
)
```

Arguments:

`host` (`character(1)`)
: See db_sql_client().

`token` (`character(1)`)
: See db_sql_client().

`http_path` (`character(1)`)
: See db_sql_client().

`catalog (character(1))`
: See db_sql_client().

`schema (character(1))`
: See db_sql_client().

`use_cloud_fetch (logical(1))`
: See db_sql_client().

`session_configuration (list(...))`
: See db_sql_client().

`...` Parameters passed to connection method

Returns: DatabricksSqlClient.

Method `columns()`**:** Execute a metadata query about the columns.

Usage:

```
DatabricksSqlClient$columns(
  catalog_name = NULL,
  schema_name = NULL,
  table_name = NULL,
  column_name = NULL,
  as_tibble = TRUE
)
```

Arguments:

`catalog_name (character(1))`
: A catalog name to retrieve information about. The `%` character is interpreted as a wildcard.

`schema_name (character(1))`
: A schema name to retrieve information about. The `%` character is interpreted as a wildcard.

`table_name (character(1))`
: A table name to retrieve information about. The `%` character is interpreted as a wildcard.

`column_name (character(1))`
: A column name to retrieve information about. The `%` character is interpreted as a wildcard.

`as_tibble (logical(1))`
: If `TRUE` (default) will return tibble::tibble, otherwise returns arrow::Table.

Returns: tibble::tibble or arrow::Table.

Examples:

```
\dontrun{
  client$columns(catalog_name = "defa%")
  client$columns(catalog_name = "default", table_name = "gold_%")
}
```

Method `catalogs()`**:** Execute a metadata query about the catalogs.

Usage:

```
DatabricksSqlClient$catalogs(as_tibble = TRUE)
```

Arguments:

`as_tibble (logical(1))`
: If `TRUE` (default) will return tibble::tibble, otherwise returns arrow::Table.

Returns: tibble::tibble or arrow::Table.

Examples:

```
\dontrun{
  client$catalogs()
}
```

Method `schemas()`**:** Execute a metadata query about the schemas.

Usage:

```
DatabricksSqlClient$schemas(
  catalog_name = NULL,
  schema_name = NULL,
  as_tibble = TRUE
)
```

Arguments:

`catalog_name (character(1))`
: A catalog name to retrieve information about. The `%` character is interpreted as a wildcard.

`schema_name (character(1))`
: A schema name to retrieve information about. The `%` character is interpreted as a wildcard.

`as_tibble (logical(1))`
: If `TRUE` (default) will return tibble::tibble, otherwise returns arrow::Table.

Returns: tibble::tibble or arrow::Table.

Examples:

```
\dontrun{
  client$schemas(catalog_name = "main")
}
```

Method `tables()`**:** Execute a metadata query about tables and views

Usage:

```
DatabricksSqlClient$tables(
  catalog_name = NULL,
  schema_name = NULL,
  table_name = NULL,
  table_types = NULL,
  as_tibble = TRUE
)
```

Arguments:

`catalog_name (character(1))`
: A catalog name to retrieve information about. The `%` character is interpreted as a wildcard.

`schema_name (character(1))`
: A schema name to retrieve information about. The `%` character is interpreted as a wildcard.

`table_name (character(1))`
: A table name to retrieve information about. The `%` character is interpreted as a wildcard.

`table_types (character())`
: A list of table types to match, for example `"TABLE"` or `"VIEW"`.

`as_tibble (logical(1))`
If TRUE (default) will return tibble::tibble, otherwise returns arrow::Table.

Returns: tibble::tibble or arrow::Table.

Method `execute()`**:** Prepares and then runs a database query or command.

Usage:

```
DatabricksSqlClient$execute(operation, parameters = NULL, as_tibble = TRUE)
```

Arguments:

`operation (character(1))`
The query or command to prepare and then run.

`parameters (list())`
Optional. A sequence of parameters to use with the operation parameter.

`as_tibble (logical(1))`
If TRUE (default) will return tibble::tibble, otherwise returns arrow::Table.

Returns: tibble::tibble or arrow::Table.

Examples:

```
\dontrun{
 client$execute("select 1")
 client$execute("select * from x.y.z limit 100")
 client$execute(
   operation = "select * from x.y.z where a < %(threshold)s limit 1000",
   parameters = list(threshold = 100)
 )
}
```

Method `execute_many()`**:** Prepares and then runs a database query or command using all parameter sequences in the seq_of_parameters argument. Only the final result set is retained.

Usage:

```
DatabricksSqlClient$execute_many(
  operation,
  seq_of_parameters = NULL,
  as_tibble = TRUE
)
```

Arguments:

`operation (character(1))`
The query or command to prepare and then run.

`seq_of_parameters (list(list()))`
A sequence of many sets of parameter values to use with the operation parameter.

`as_tibble (logical(1))`
If TRUE (default) will return tibble::tibble, otherwise returns arrow::Table.

Returns: tibble::tibble or arrow::Table.

Examples:

```
\dontrun{
 client$execute_many(
   operation = "select * from x.y.z where a < %(threshold)s limit 1000",
   seq_of_parameters = list(
     list(threshold = 100),
     list(threshold = 200),
     list(threshold = 300)
   )
 )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DatabricksSqlClient$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `DatabricksSqlClient$columns`
## ------------------------------------------------

## Not run:
  client$columns(catalog_name = "defa%")
  client$columns(catalog_name = "default", table_name = "gold_%")

## End(Not run)

## ------------------------------------------------
## Method `DatabricksSqlClient$catalogs`
## ------------------------------------------------

## Not run:
  client$catalogs()

## End(Not run)

## ------------------------------------------------
## Method `DatabricksSqlClient$schemas`
## ------------------------------------------------

## Not run:
  client$schemas(catalog_name = "main")

## End(Not run)

## ------------------------------------------------
## Method `DatabricksSqlClient$execute`
## ------------------------------------------------
```

```
## Not run:
 client$execute("select 1")
 client$execute("select * from x.y.z limit 100")
 client$execute(
   operation = "select * from x.y.z where a < %(threshold)s limit 1000",
   parameters = list(threshold = 100)
 )

## End(Not run)

## ------------------------------------------------
## Method `DatabricksSqlClient$execute_many`
## ------------------------------------------------

## Not run:
 client$execute_many(
   operation = "select * from x.y.z where a < %(threshold)s limit 1000",
   seq_of_parameters = list(
     list(threshold = 100),
     list(threshold = 200),
     list(threshold = 300)
   )
 )

## End(Not run)
```

`dbfs_storage_info` *DBFS Storage Information*

Description

DBFS Storage Information

Usage

```
dbfs_storage_info(destination)
```

Arguments

`destination`	DBFS destination. Example: `dbfs:/my/path`.

See Also

`cluster_log_conf()`, `init_script_info()`

Other Cluster Log Configuration Objects: `cluster_log_conf()`, `s3_storage_info()`

Other Init Script Info Objects: `file_storage_info()`, `s3_storage_info()`

db_cluster_action *Cluster Action Helper Function*

Description

Cluster Action Helper Function

Usage

```
db_cluster_action(
  cluster_id,
  action = c("start", "restart", "delete", "permanent-delete", "pin", "unpin"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
action	One of start, restart, delete, permanent-delete, pin, unpin.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

db_cluster_create *Create a Cluster*

Description

Create a Cluster

Usage

```
db_cluster_create(
  name,
  spark_version,
  node_type_id,
  num_workers = NULL,
  autoscale = NULL,
  spark_conf = list(),
  cloud_attrs = aws_attributes(),
```

```
  driver_node_type_id = NULL,
  custom_tags = list(),
  init_scripts = list(),
  spark_env_vars = list(),
  autotermination_minutes = 120,
  log_conf = NULL,
  ssh_public_keys = NULL,
  driver_instance_pool_id = NULL,
  instance_pool_id = NULL,
  idempotency_token = NULL,
  enable_elastic_disk = TRUE,
  apply_policy_default_values = TRUE,
  enable_local_disk_encryption = TRUE,
  docker_image = NULL,
  policy_id = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Cluster name requested by the user. This doesn't have to be unique. If not specified at creation, the cluster name will be an empty string.
spark_version	The runtime version of the cluster. You can retrieve a list of available runtime versions by using db_cluster_runtime_versions().
node_type_id	The node type for the worker nodes. db_cluster_list_node_types() can be used to see available node types.
num_workers	Number of worker nodes that this cluster should have. A cluster has one Spark driver and num_workers executors for a total of num_workers + 1 Spark nodes.
autoscale	Instance of cluster_autoscale().
spark_conf	Named list. An object containing a set of optional, user-specified Spark configuration key-value pairs. You can also pass in a string of extra JVM options to the driver and the executors via spark.driver.extraJavaOptions and spark.executor.extraJavaOptions respectively. E.g. list("spark.speculation" = true, "spark.streaming.ui.retainedBatches" = 5).
cloud_attrs	Attributes related to clusters running on specific cloud provider. Defaults to aws_attributes(). Must be one of aws_attributes(), azure_attributes(), gcp_attributes().
driver_node_type_id	The node type of the Spark driver. This field is optional; if unset, the driver node type will be set as the same value as node_type_id defined above. db_cluster_list_node_types() can be used to see available node types.
custom_tags	Named list. An object containing a set of tags for cluster resources. Databricks tags all cluster resources with these tags in addition to default_tags. Databricks allows at most 45 custom tags.

`init_scripts` Instance of `init_script_info()`.

`spark_env_vars` Named list. User-specified environment variable key-value pairs. In order to specify an additional set of `SPARK_DAEMON_JAVA_OPTS`, we recommend appending them to `$SPARK_DAEMON_JAVA_OPTS` as shown in the following example. This ensures that all default Databricks managed environmental variables are included as well. E.g. `{"SPARK_DAEMON_JAVA_OPTS": "$SPARK_DAEMON_JAVA_OPTS -Dspark.shuffle.service.enabled=true"}`

`autotermination_minutes` Automatically terminates the cluster after it is inactive for this time in minutes. If not set, this cluster will not be automatically terminated. If specified, the threshold must be between 10 and 10000 minutes. You can also set this value to 0 to explicitly disable automatic termination. Defaults to 120.

`log_conf` Instance of `cluster_log_conf()`.

`ssh_public_keys` List. SSH public key contents that will be added to each Spark node in this cluster. The corresponding private keys can be used to login with the user name ubuntu on port 2200. Up to 10 keys can be specified.

`driver_instance_pool_id` ID of the instance pool to use for the driver node. You must also specify `instance_pool_id`. Optional.

`instance_pool_id` ID of the instance pool to use for cluster nodes. If `driver_instance_pool_id` is present, `instance_pool_id` is used for worker nodes only. Otherwise, it is used for both the driver and worker nodes. Optional.

`idempotency_token` An optional token that can be used to guarantee the idempotency of cluster creation requests. If an active cluster with the provided token already exists, the request will not create a new cluster, but it will return the ID of the existing cluster instead. The existence of a cluster with the same token is not checked against terminated clusters. If you specify the idempotency token, upon failure you can retry until the request succeeds. Databricks guarantees that exactly one cluster will be launched with that idempotency token. This token should have at most 64 characters.

`enable_elastic_disk` When enabled, this cluster will dynamically acquire additional disk space when its Spark workers are running low on disk space.

`apply_policy_default_values` Boolean (Default: `TRUE`), whether to use policy default values for missing cluster attributes.

`enable_local_disk_encryption` Boolean (Default: `TRUE`), whether encryption of disks locally attached to the cluster is enabled.

`docker_image` Instance of `docker_image()`.

`policy_id` String, ID of a cluster policy.

`host` Databricks workspace URL, defaults to calling `db_host()`.

`token` Databricks workspace token, defaults to calling `db_token()`.

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Create a new Apache Spark cluster. This method acquires new instances from the cloud provider if necessary. This method is asynchronous; the returned cluster_id can be used to poll the cluster state (db_cluster_get()). When this method returns, the cluster is in a PENDING state. The cluster is usable once it enters a RUNNING state.

Databricks may not be able to acquire some of the requested nodes, due to cloud provider limitations or transient network issues. If Databricks acquires at least 85% of the requested on-demand nodes, cluster creation will succeed. Otherwise the cluster will terminate with an informative error message.

Cannot specify both autoscale and num_workers, must choose one.

More Documentation.

See Also

Other Clusters API: db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_delete *Delete/Terminate a Cluster*

Description

Delete/Terminate a Cluster

Usage

```
db_cluster_delete(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

The cluster must be in the RUNNING state.

---

db_cluster_edit *Edit a Cluster*

Description

Edit the configuration of a cluster to match the provided attributes and size.

Usage

```
db_cluster_edit(
  cluster_id,
  spark_version,
  node_type_id,
  num_workers = NULL,
  autoscale = NULL,
  name = NULL,
  spark_conf = NULL,
  cloud_attrs = NULL,
  driver_node_type_id = NULL,
  custom_tags = NULL,
  init_scripts = NULL,
  spark_env_vars = NULL,
  autotermination_minutes = NULL,
  log_conf = NULL,
  ssh_public_keys = NULL,
  driver_instance_pool_id = NULL,
  instance_pool_id = NULL,
  idempotency_token = NULL,
  enable_elastic_disk = NULL,
  apply_policy_default_values = NULL,
  enable_local_disk_encryption = NULL,
  docker_image = NULL,
  policy_id = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
spark_version	The runtime version of the cluster. You can retrieve a list of available runtime versions by using db_cluster_runtime_versions().

node_type_id The node type for the worker nodes. db_cluster_list_node_types() can be used to see available node types.

num_workers Number of worker nodes that this cluster should have. A cluster has one Spark driver and num_workers executors for a total of num_workers + 1 Spark nodes.

autoscale Instance of cluster_autoscale().

name Cluster name requested by the user. This doesn't have to be unique. If not specified at creation, the cluster name will be an empty string.

spark_conf Named list. An object containing a set of optional, user-specified Spark configuration key-value pairs. You can also pass in a string of extra JVM options to the driver and the executors via spark.driver.extraJavaOptions and spark.executor.extraJavaOptions respectively. E.g. list("spark.speculation" = true, "spark.streaming.ui.retainedBatches" = 5).

cloud_attrs Attributes related to clusters running on specific cloud provider. Defaults to aws_attributes(). Must be one of aws_attributes(), azure_attributes(), gcp_attributes().

driver_node_type_id
The node type of the Spark driver. This field is optional; if unset, the driver node type will be set as the same value as node_type_id defined above. db_cluster_list_node_types() can be used to see available node types.

custom_tags Named list. An object containing a set of tags for cluster resources. Databricks tags all cluster resources with these tags in addition to default_tags. Databricks allows at most 45 custom tags.

init_scripts Instance of init_script_info().

spark_env_vars Named list. User-specified environment variable key-value pairs. In order to specify an additional set of SPARK_DAEMON_JAVA_OPTS, we recommend appending them to $SPARK_DAEMON_JAVA_OPTS as shown in the following example. This ensures that all default Databricks managed environmental variables are included as well. E.g. {"SPARK_DAEMON_JAVA_OPTS": "$SPARK_DAEMON_JAVA_OPTS -Dspark.shuffle.service.enabled=true"}

autotermination_minutes
Automatically terminates the cluster after it is inactive for this time in minutes. If not set, this cluster will not be automatically terminated. If specified, the threshold must be between 10 and 10000 minutes. You can also set this value to 0 to explicitly disable automatic termination. Defaults to 120.

log_conf Instance of cluster_log_conf().

ssh_public_keys
List. SSH public key contents that will be added to each Spark node in this cluster. The corresponding private keys can be used to login with the user name ubuntu on port 2200. Up to 10 keys can be specified.

driver_instance_pool_id
ID of the instance pool to use for the driver node. You must also specify instance_pool_id. Optional.

instance_pool_id
ID of the instance pool to use for cluster nodes. If driver_instance_pool_id is present, instance_pool_id is used for worker nodes only. Otherwise, it is used for both the driver and worker nodes. Optional.

idempotency_token
: An optional token that can be used to guarantee the idempotency of cluster creation requests. If an active cluster with the provided token already exists, the request will not create a new cluster, but it will return the ID of the existing cluster instead. The existence of a cluster with the same token is not checked against terminated clusters. If you specify the idempotency token, upon failure you can retry until the request succeeds. Databricks guarantees that exactly one cluster will be launched with that idempotency token. This token should have at most 64 characters.

enable_elastic_disk
: When enabled, this cluster will dynamically acquire additional disk space when its Spark workers are running low on disk space.

apply_policy_default_values
: Boolean (Default: TRUE), whether to use policy default values for missing cluster attributes.

enable_local_disk_encryption
: Boolean (Default: TRUE), whether encryption of disks locally attached to the cluster is enabled.

docker_image
: Instance of docker_image().

policy_id
: String, ID of a cluster policy.

host
: Databricks workspace URL, defaults to calling db_host().

token
: Databricks workspace token, defaults to calling db_token().

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

You can edit a cluster if it is in a RUNNING or TERMINATED state. If you edit a cluster while it is in a RUNNING state, it will be restarted so that the new attributes can take effect. If you edit a cluster while it is in a TERMINATED state, it will remain TERMINATED. The next time it is started using the clusters/start API, the new attributes will take effect. An attempt to edit a cluster in any other state will be rejected with an INVALID_STATE error code.

Clusters created by the Databricks Jobs service cannot be edited.

See Also

Other Clusters API: db_cluster_create(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

db_cluster_events *List Cluster Activity Events*

Description

List Cluster Activity Events

Usage

```
db_cluster_events(
  cluster_id,
  start_time = NULL,
  end_time = NULL,
  event_types = NULL,
  order = c("DESC", "ASC"),
  offset = 0,
  limit = 50,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to retrieve events about.
start_time	The start time in epoch milliseconds. If empty, returns events starting from the beginning of time.
end_time	The end time in epoch milliseconds. If empty, returns events up to the current time.
event_types	List. Optional set of event types to filter by. Default is to return all events. Event Types.
order	Either DESC (default) or ASC.
offset	The offset in the result set. Defaults to 0 (no offset). When an offset is specified and the results are requested in descending order, the end_time field is required.
limit	Maximum number of events to include in a page of events. Defaults to 50, and maximum allowed value is 500.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Retrieve a list of events about the activity of a cluster. You can retrieve events from active clusters (running, pending, or reconfiguring) and terminated clusters within 30 days of their last termination. This API is paginated. If there are more events to read, the response includes all the parameters necessary to request the next page of events.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_get *Get Details of a Cluster*

Description

Get Details of a Cluster

Usage

```
db_cluster_get(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Retrieve the information for a cluster given its identifier. Clusters can be described while they are running or up to 30 days after they are terminated.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_list *List Clusters*

Description

List Clusters

Usage

```
db_cluster_list(host = db_host(), token = db_token(), perform_request = TRUE)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Return information about all pinned clusters, active clusters, up to 150 of the most recently terminated all-purpose clusters in the past 30 days, and up to 30 of the most recently terminated job clusters in the past 30 days.

For example, if there is 1 pinned cluster, 4 active clusters, 45 terminated all-purpose clusters in the past 30 days, and 50 terminated job clusters in the past 30 days, then this API returns:

- the 1 pinned cluster
- 4 active clusters
- All 45 terminated all-purpose clusters
- The 30 most recently terminated job clusters

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

db_cluster_list_node_types *List Available Cluster Node Types*

Description

List Available Cluster Node Types

Usage

```
db_cluster_list_node_types(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Return a list of supported Spark node types. These node types can be used to launch a cluster.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

db_cluster_list_zones *List Availability Zones (AWS Only)*

Description

List Availability Zones (AWS Only)

Usage

```
db_cluster_list_zones(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Amazon Web Services (AWS) ONLY! Return a list of availability zones where clusters can be created in (ex: us-west-2a). These zones can be used to launch a cluster.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_perm_delete

Permanently Delete a Cluster

Description

Permanently Delete a Cluster

Usage

```
db_cluster_perm_delete(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

If the cluster is running, it is terminated and its resources are asynchronously removed. If the cluster is terminated, then it is immediately removed.

You cannot perform *any action, including retrieve the cluster's permissions, on a permanently deleted cluster. A permanently deleted cluster is also no longer returned in the cluster list.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_pin *Pin a Cluster*

Description

Pin a Cluster

Usage

```
db_cluster_pin(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Ensure that an all-purpose cluster configuration is retained even after a cluster has been terminated for more than 30 days. Pinning ensures that the cluster is always returned by db_cluster_list(). Pinning a cluster that is already pinned has no effect.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_resize *Resize a Cluster*

Description

Resize a Cluster

Usage

```
db_cluster_resize(
  cluster_id,
  num_workers = NULL,
  autoscale = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
num_workers	Number of worker nodes that this cluster should have. A cluster has one Spark driver and num_workers executors for a total of num_workers + 1 Spark nodes.
autoscale	Instance of cluster_autoscale().
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

The cluster must be in the RUNNING state.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

`db_cluster_restart` *Restart a Cluster*

Description

Restart a Cluster

Usage

```
db_cluster_restart(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`cluster_id`	Canonical identifier for the cluster.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

Details

The cluster must be in the `RUNNING` state.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

db_cluster_runtime_versions *List Available Databricks Runtime Versions*

Description

List Available Databricks Runtime Versions

Usage

```
db_cluster_runtime_versions(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Return the list of available runtime versions. These versions can be used to launch a cluster.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

db_cluster_start *Start a Cluster*

Description

Start a Cluster

Usage

```
db_cluster_start(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Start a terminated cluster given its ID.

This is similar to db_cluster_create(), except:

- The terminated cluster ID and attributes are preserved.
- The cluster starts with the last specified cluster size. If the terminated cluster is an autoscaling cluster, the cluster starts with the minimum number of nodes.
- If the cluster is in the RESTARTING state, a 400 error is returned.
- You cannot start a cluster launched to run a job.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_terminate *Delete/Terminate a Cluster*

Description

Delete/Terminate a Cluster

Usage

```
db_cluster_terminate(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

The cluster is removed asynchronously. Once the termination has completed, the cluster will be in the TERMINATED state. If the cluster is already in a TERMINATING or TERMINATED state, nothing will happen.

Unless a cluster is pinned, 30 days after the cluster is terminated, it is permanently deleted.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_unpin(), get_and_start_cluster(), get_latest_dbr()

---

db_cluster_unpin *Unpin a Cluster*

Description

Unpin a Cluster

Usage

```
db_cluster_unpin(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Allows the cluster to eventually be removed from the list returned by db_cluster_list(). Unpinning a cluster that is not pinned has no effect.

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), get_and_start_cluster(), get_latest_dbr()

---

db_context_command_cancel

Cancel a Command

Description

Cancel a Command

Usage

```
db_context_command_cancel(
  cluster_id,
  context_id,
  command_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
context_id	The ID of the execution context.
command_id	The ID of the command to get information about.
host	Databricks workspace URL, defaults to calling db_host().

token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Execution Context API: db_context_command_parse(), db_context_command_run(), db_context_command_run_
db_context_command_status(), db_context_create(), db_context_destroy(), db_context_status()

---

db_context_command_run *Run a Command*

Description

Run a Command

Usage

```
db_context_command_run(
  cluster_id,
  context_id,
  language = c("python", "sql", "scala", "r"),
  command = NULL,
  command_file = NULL,
  options = list(),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
context_id	The ID of the execution context.
language	The language for the context. One of python, sql, scala, r.
command	The command string to run.
command_file	The path to a file containing the command to run.
options	Named list of values used downstream. For example, a 'displayRowLimit' override (used in testing).
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Execution Context API: db_context_command_cancel(), db_context_command_parse(), db_context_command_run_and_wait(), db_context_command_status(), db_context_create(), db_context_destroy(), db_context_status()

---

db_context_command_run_and_wait

Run a Command and Wait For Results

Description

Run a Command and Wait For Results

Usage

```
db_context_command_run_and_wait(
  cluster_id,
  context_id,
  language = c("python", "sql", "scala", "r"),
  command = NULL,
  command_file = NULL,
  options = list(),
  parse_result = TRUE,
  host = db_host(),
  token = db_token()
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
context_id	The ID of the execution context.
language	The language for the context. One of python, sql, scala, r.
command	The command string to run.
command_file	The path to a file containing the command to run.
options	Named list of values used downstream. For example, a 'displayRowLimit' override (used in testing).
parse_result	Boolean, determines if results are parsed automatically.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().

See Also

Other Execution Context API: db_context_command_cancel(), db_context_command_parse(), db_context_command_run(), db_context_command_status(), db_context_create(), db_context_destroy(), db_context_status()

db_context_command_status *Get Information About a Command*

Description

Get Information About a Command

Usage

```
db_context_command_status(
  cluster_id,
  context_id,
  command_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
context_id	The ID of the execution context.
command_id	The ID of the command to get information about.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Execution Context API: db_context_command_cancel(), db_context_command_parse(), db_context_command_run(), db_context_command_run_and_wait(), db_context_create(), db_context_destroy(), db_context_status()

db_context_create *Create an Execution Context*

Description

Create an Execution Context

Usage

```
db_context_create(
  cluster_id,
  language = c("python", "sql", "scala", "r"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
language	The language for the context. One of python, sql, scala, r.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Execution Context API: db_context_command_cancel(), db_context_command_parse(), db_context_command_run(), db_context_command_run_and_wait(), db_context_command_status(), db_context_destroy(), db_context_status()

db_context_destroy *Delete an Execution Context*

Description

Delete an Execution Context

Usage

```
db_context_destroy(
  cluster_id,
  context_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
context_id	The ID of the execution context.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Execution Context API: db_context_command_cancel(), db_context_command_parse(), db_context_command_run(), db_context_command_run_and_wait(), db_context_command_status(), db_context_create(), db_context_status()

---

db_context_manager *Databricks Execution Context Manager (R6 Class)*

Description

Databricks Execution Context Manager (R6 Class)

Databricks Execution Context Manager (R6 Class)

Details

db_context_manager() provides a simple interface to send commands to Databricks cluster and return the results.

Methods

Public methods:

- db_context_manager$new()
- db_context_manager$close()
- db_context_manager$cmd_run()
- db_context_manager$clone()

Method `new()`**:** Create a new context manager object.

Usage:

```
db_context_manager$new(
  cluster_id,
  language = c("r", "py", "scala", "sql", "sh"),
  host = db_host(),
  token = db_token()
)
```

Arguments:

`cluster_id` The ID of the cluster to execute command on.

`language` One of `r`, `py`, `scala`, `sql`, or `sh`.

`host` Databricks workspace URL, defaults to calling `db_host()`.

`token` Databricks workspace token, defaults to calling `db_token()`.

Returns: A new `databricks_context_manager` object.

Method `close()`**:** Destroy the execution context

Usage:

```
db_context_manager$close()
```

Method `cmd_run()`**:** Execute a command against a Databricks cluster

Usage:

```
db_context_manager$cmd_run(cmd, language = c("r", "py", "scala", "sql", "sh"))
```

Arguments:

`cmd` code to execute against Databricks cluster

`language` One of `r`, `py`, `scala`, `sql`, or `sh`.

Returns: Command results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
db_context_manager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`db_context_status` *Get Information About an Execution Context*

Description

Get Information About an Execution Context

Usage

```
db_context_status(
  cluster_id,
  context_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	The ID of the cluster to create the context for.
context_id	The ID of the execution context.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Execution Context API: db_context_command_cancel(), db_context_command_parse(), db_context_command_run(), db_context_command_run_and_wait(), db_context_command_status(), db_context_create(), db_context_destroy()

---

db_current_cloud *Detect Current Workspaces Cloud*

Description

Detect Current Workspaces Cloud

Usage

```
db_current_cloud(host = db_host(), token = db_token(), perform_request = TRUE)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Value

String

db_current_user *Get Current User Info*

Description

Get Current User Info

Usage

```
db_current_user(host = db_host(), token = db_token(), perform_request = TRUE)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Value

list of user metadata

db_current_workspace_id *Detect Current Workspace ID*

Description

Detect Current Workspace ID

Usage

```
db_current_workspace_id(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Value

String

`db_dbfs_add_block` *DBFS Add Block*

Description

Append a block of data to the stream specified by the input handle.

Usage

```
db_dbfs_add_block(
  handle,
  data,
  convert_to_raw = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`handle`	Handle on an open stream.
`data`	Either a path for file on local system or a character/raw vector that will be base64-encoded. This has a limit of 1 MB.
`convert_to_raw`	Boolean (Default: `FALSE`), if `TRUE` will convert character vector to raw via `base::as.raw()`.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

Details

- If the handle does not exist, this call will throw an exception with `RESOURCE_DOES_NOT_EXIST`.
- If the block of data exceeds 1 MB, this call will throw an exception with `MAX_BLOCK_SIZE_EXCEEDED`.

Typical File Upload Flow

- Call create and get a handle via `db_dbfs_create()`
- Make one or more `db_dbfs_add_block()` calls with the handle you have
- Call `db_dbfs_close()` with the handle you have

See Also

Other DBFS API: db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

---

db_dbfs_close *DBFS Close*

Description

Close the stream specified by the input handle.

Usage

```
db_dbfs_close(
  handle,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

handle	The handle on an open stream. This field is required.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

If the handle does not exist, this call throws an exception with RESOURCE_DOES_NOT_EXIST.

Value

HTTP Response

Typical File Upload Flow

- Call create and get a handle via db_dbfs_create()
- Make one or more db_dbfs_add_block() calls with the handle you have
- Call db_dbfs_close() with the handle you have

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

db_dbfs_create *DBFS Create*

Description

Open a stream to write to a file and returns a handle to this stream.

Usage

```
db_dbfs_create(
  path,
  overwrite = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
overwrite	Boolean, specifies whether to overwrite existing file or files.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

There is a 10 minute idle timeout on this handle. If a file or directory already exists on the given path and overwrite is set to FALSE, this call throws an exception with RESOURCE_ALREADY_EXISTS.

Value

Handle which should subsequently be passed into db_dbfs_add_block() and db_dbfs_close() when writing to a file through a stream.

Typical File Upload Flow

- Call create and get a handle via db_dbfs_create()
- Make one or more db_dbfs_add_block() calls with the handle you have
- Call db_dbfs_close() with the handle you have

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

---

db_dbfs_delete *DBFS Delete*

Description

DBFS Delete

Usage

```
db_dbfs_delete(
  path,
  recursive = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
recursive	Whether or not to recursively delete the directory's contents. Deleting empty directories can be done without providing the recursive flag.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

db_dbfs_get_status *DBFS Get Status*

Description

Get the file information of a file or directory.

Usage

```
db_dbfs_get_status(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

- If the file or directory does not exist, this call throws an exception with RESOURCE_DOES_NOT_EXIST.

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

db_dbfs_list *DBFS List*

Description

List the contents of a directory, or details of the file.

Usage

```
db_dbfs_list(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

When calling list on a large directory, the list operation will time out after approximately 60 seconds.

We **strongly** recommend using list only on directories containing less than 10K files and discourage using the DBFS REST API for operations that list more than 10K files. Instead, we recommend that you perform such operations in the context of a cluster, using the File system utility (dbutils.fs), which provides the same functionality without timing out.

- If the file or directory does not exist, this call throws an exception with RESOURCE_DOES_NOT_EXIST.

Value

data.frame

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

---

db_dbfs_mkdirs *DBFS mkdirs*

Description

Create the given directory and necessary parent directories if they do not exist.

Usage

```
db_dbfs_mkdirs(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

- If there exists a file (not a directory) at any prefix of the input path, this call throws an exception with RESOURCE_ALREADY_EXISTS.
- If this operation fails it may have succeeded in creating some of the necessary parent directories.

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_move(), db_dbfs_put(), db_dbfs_read()

---

db_dbfs_move *DBFS Move*

Description

Move a file from one location to another location within DBFS.

Usage

```
db_dbfs_move(
  source_path,
  destination_path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

source_path	The source path of the file or directory. The path should be the absolute DBFS path (for example, /mnt/my-source-folder/).
destination_path	The destination path of the file or directory. The path should be the absolute DBFS path (for example, /mnt/my-destination-folder/).
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

If the given source path is a directory, this call always recursively moves all files.

When moving a large number of files, the API call will time out after approximately 60 seconds, potentially resulting in partially moved data. Therefore, for operations that move more than 10K files, we **strongly** discourage using the DBFS REST API. Instead, we recommend that you perform such operations in the context of a cluster, using the File system utility (dbutils.fs) from a notebook, which provides the same functionality without timing out.

- If the source file does not exist, this call throws an exception with RESOURCE_DOES_NOT_EXIST.
- If there already exists a file in the destination path, this call throws an exception with RESOURCE_ALREADY_EXISTS.

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_put(), db_dbfs_read()

---

db_dbfs_put *DBFS Put*

Description

Upload a file through the use of multipart form post.

Usage

```
db_dbfs_put(
  path,
  file = NULL,
  contents = NULL,
  overwrite = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
file	Path to a file on local system, takes precedent over path.
contents	String that is base64 encoded.
overwrite	Flag (Default: FALSE) that specifies whether to overwrite existing files.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Either contents or file must be specified. file takes precedent over contents if both are specified.

Mainly used for streaming uploads, but can also be used as a convenient single call for data upload.

The amount of data that can be passed using the contents parameter is limited to 1 MB if specified as a string (MAX_BLOCK_SIZE_EXCEEDED is thrown if exceeded) and 2 GB as a file.

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_read()

---

db_dbfs_read *DBFS Read*

Description

Return the contents of a file.

Usage

```
db_dbfs_read(
  path,
  offset = 0,
  length = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	The path of the new file. The path should be the absolute DBFS path (for example /mnt/my-file.txt).
offset	Offset to read from in bytes.
length	Number of bytes to read starting from the offset. This has a limit of 1 MB, and a default value of 0.5 MB.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

If offset + length exceeds the number of bytes in a file, reads contents until the end of file.

- If the file does not exist, this call throws an exception with RESOURCE_DOES_NOT_EXIST.
- If the path is a directory, the read length is negative, or if the offset is negative, this call throws an exception with INVALID_PARAMETER_VALUE.
- If the read length exceeds 1 MB, this call throws an exception with MAX_READ_SIZE_EXCEEDED.

See Also

Other DBFS API: db_dbfs_add_block(), db_dbfs_close(), db_dbfs_create(), db_dbfs_delete(), db_dbfs_get_status(), db_dbfs_list(), db_dbfs_mkdirs(), db_dbfs_move(), db_dbfs_put()

---

db_host *Generate/Fetch Databricks Host*

Description

If both id and prefix are NULL then the function will check for the DATABRICKS_HOST environment variable. .databrickscfg will be searched if db_profile and use_databrickscfg are set or if Posit Workbench managed OAuth credentials are detected.

When defining id and prefix you do not need to specify the whole URL. E.g. https://<prefix>.<id>.cloud.databricks is the form to follow.

Usage

```
db_host(id = NULL, prefix = NULL, profile = default_config_profile())
```

Arguments

id	The workspace string
prefix	Workspace prefix
profile	Profile to use when fetching from environment variable (e.g. .Renviron) or .databricksfg file

Details

The behaviour is subject to change depending if `db_profile` and `use_databrickscfg` options are set.

- `use_databrickscfg`: Boolean (default: `FALSE`), determines if credentials are fetched from profile of `.databrickscfg` or `.Renviron`
- `db_profile`: String (default: `NULL`), determines profile used. `.databrickscfg` will automatically be used when Posit Workbench managed OAuth credentials are detected.

See vignette on authentication for more details.

Value

workspace URL

See Also

Other Databricks Authentication Helpers: `db_read_netrc()`, `db_token()`, `db_wsid()`

`db_jobs_create` *Create Job*

Description

Create Job

Usage

```
db_jobs_create(
  name,
  tasks,
  schedule = NULL,
  job_clusters = NULL,
  email_notifications = NULL,
  timeout_seconds = NULL,
  max_concurrent_runs = 1,
  access_control_list = NULL,
  git_source = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name for the job.
tasks	Task specifications to be executed by this job. Use job_tasks().
schedule	Instance of cron_schedule().
job_clusters	Named list of job cluster specifications (using new_cluster()) that can be shared and reused by tasks of this job. Libraries cannot be declared in a shared job cluster. You must declare dependent libraries in task settings.
email_notifications	Instance of email_notifications().
timeout_seconds	An optional timeout applied to each run of this job. The default behavior is to have no timeout.
max_concurrent_runs	Maximum allowed number of concurrent runs of the job. Set this value if you want to be able to execute multiple runs of the same job concurrently. This setting affects only new runs. This value cannot exceed 1000. Setting this value to 0 causes all new runs to be skipped. The default behavior is to allow only 1 concurrent run.
access_control_list	Instance of access_control_request().
git_source	Optional specification for a remote repository containing the notebooks used by this job's notebook tasks. Instance of git_source().
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Full Documentation

See Also

job_tasks(), job_task(), email_notifications(), cron_schedule(), access_control_request(), access_control_req_user(), access_control_req_group(), git_source()

Other Jobs API: db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

db_jobs_delete *Delete a Job*

Description

Delete a Job

Usage

```
db_jobs_delete(
  job_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

job_id	The canonical identifier of the job.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

db_jobs_get *Get Job Details*

Description

Get Job Details

Usage

```
db_jobs_get(
  job_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

job_id	The canonical identifier of the job.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

---

db_jobs_list *List Jobs*

Description

List Jobs

Usage

```
db_jobs_list(
  limit = 25,
  offset = 0,
  expand_tasks = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

limit	Number of jobs to return. This value must be greater than 0 and less or equal to 25. The default value is 25. If a request specifies a limit of 0, the service instead uses the maximum limit.
offset	The offset of the first job to return, relative to the most recently created job.
expand_tasks	Whether to include task and cluster details in the response.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

---

db_jobs_reset *Overwrite All Settings For A Job*

Description

Overwrite All Settings For A Job

Usage

```
db_jobs_reset(
  job_id,
  name,
  schedule,
  tasks,
  job_clusters = NULL,
  email_notifications = NULL,
  timeout_seconds = NULL,
  max_concurrent_runs = 1,
  access_control_list = NULL,
  git_source = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

job_id	The canonical identifier of the job.
name	Name for the job.
schedule	Instance of cron_schedule().
tasks	Task specifications to be executed by this job. Use job_tasks().
job_clusters	Named list of job cluster specifications (using new_cluster()) that can be shared and reused by tasks of this job. Libraries cannot be declared in a shared job cluster. You must declare dependent libraries in task settings.
email_notifications	Instance of email_notifications().
timeout_seconds	An optional timeout applied to each run of this job. The default behavior is to have no timeout.

max_concurrent_runs
: Maximum allowed number of concurrent runs of the job. Set this value if you want to be able to execute multiple runs of the same job concurrently. This setting affects only new runs. This value cannot exceed 1000. Setting this value to 0 causes all new runs to be skipped. The default behavior is to allow only 1 concurrent run.

access_control_list
: Instance of access_control_request().

git_source
: Optional specification for a remote repository containing the notebooks used by this job's notebook tasks. Instance of git_source().

host
: Databricks workspace URL, defaults to calling db_host().

token
: Databricks workspace token, defaults to calling db_token().

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

---

db_jobs_runs_cancel *Cancel Job Run*

Description

Cancels a run.

Usage

```
db_jobs_runs_cancel(
  run_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

run_id
: The canonical identifier of the run.

host
: Databricks workspace URL, defaults to calling db_host().

token
: Databricks workspace token, defaults to calling db_token().

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

The run is canceled asynchronously, so when this request completes, the run may still be running. The run are terminated shortly. If the run is already in a terminal `life_cycle_state`, this method is a no-op.

See Also

Other Jobs API: `db_jobs_create()`, `db_jobs_delete()`, `db_jobs_get()`, `db_jobs_list()`, `db_jobs_reset()`, `db_jobs_run_now()`, `db_jobs_runs_delete()`, `db_jobs_runs_export()`, `db_jobs_runs_get()`, `db_jobs_runs_get_output()`, `db_jobs_runs_list()`, `db_jobs_runs_submit()`, `db_jobs_update()`

---

db_jobs_runs_delete *Delete Job Run*

Description

Delete Job Run

Usage

```
db_jobs_runs_delete(
  run_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`run_id`	The canonical identifier of the run.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

See Also

Other Jobs API: `db_jobs_create()`, `db_jobs_delete()`, `db_jobs_get()`, `db_jobs_list()`, `db_jobs_reset()`, `db_jobs_run_now()`, `db_jobs_runs_cancel()`, `db_jobs_runs_export()`, `db_jobs_runs_get()`, `db_jobs_runs_get_output()`, `db_jobs_runs_list()`, `db_jobs_runs_submit()`, `db_jobs_update()`

db_jobs_runs_export *Export Job Run Output*

Description

Export and retrieve the job run task.

Usage

```
db_jobs_runs_export(
  run_id,
  views_to_export = c("CODE", "DASHBOARDS", "ALL"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

run_id	The canonical identifier of the run.
views_to_export	Which views to export. One of CODE, DASHBOARDS, ALL. Defaults to CODE.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

db_jobs_runs_get *Get Job Run Details*

Description

Retrieve the metadata of a run.

Usage

```
db_jobs_runs_get(
  run_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

run_id	The canonical identifier of the run.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

---

db_jobs_runs_get_output

Get Job Run Output

Description

Get Job Run Output

Usage

```
db_jobs_runs_get_output(
  run_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

run_id	The canonical identifier of the run.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_list(), db_jobs_runs_submit(), db_jobs_update()

---

db_jobs_runs_list *List Job Runs*

Description

List runs in descending order by start time.

Usage

```
db_jobs_runs_list(
  job_id,
  active_only = FALSE,
  completed_only = FALSE,
  offset = 0,
  limit = 25,
  run_type = c("JOB_RUN", "WORKFLOW_RUN", "SUBMIT_RUN"),
  expand_tasks = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

job_id	The canonical identifier of the job.
active_only	Boolean (Default: FALSE). If TRUE only active runs are included in the results; otherwise, lists both active and completed runs. An active run is a run in the PENDING, RUNNING, or TERMINATING. This field cannot be true when completed_only is TRUE.
completed_only	Boolean (Default: FALSE). If TRUE, only completed runs are included in the results; otherwise, lists both active and completed runs. This field cannot be true when active_only is TRUE.
offset	The offset of the first job to return, relative to the most recently created job.
limit	Number of jobs to return. This value must be greater than 0 and less or equal to 25. The default value is 25. If a request specifies a limit of 0, the service instead uses the maximum limit.
run_type	The type of runs to return. One of JOB_RUN, WORKFLOW_RUN, SUBMIT_RUN.
expand_tasks	Whether to include task and cluster details in the response.
host	Databricks workspace URL, defaults to calling db_host().

token Databricks workspace token, defaults to calling db_token().

perform_request
If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_submit(), db_jobs_update()

---

db_jobs_runs_submit *Create And Trigger A One-Time Run*

Description

Create And Trigger A One-Time Run

Usage

```
db_jobs_runs_submit(
  tasks,
  run_name,
  timeout_seconds = NULL,
  idempotency_token = NULL,
  access_control_list = NULL,
  git_source = NULL,
  job_clusters = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

tasks Task specifications to be executed by this job. Use job_tasks().

run_name Name for the run.

timeout_seconds
An optional timeout applied to each run of this job. The default behavior is to have no timeout.

idempotency_token
An optional token that can be used to guarantee the idempotency of job run requests. If an active run with the provided token already exists, the request does not create a new run, but returns the ID of the existing run instead. If you specify the idempotency token, upon failure you can retry until the request succeeds. Databricks guarantees that exactly one run is launched with that idempotency token. This token must have at most 64 characters.

access_control_list
: Instance of access_control_request().

git_source
: Optional specification for a remote repository containing the notebooks used by this job's notebook tasks. Instance of git_source().

job_clusters
: Named list of job cluster specifications (using new_cluster()) that can be shared and reused by tasks of this job. Libraries cannot be declared in a shared job cluster. You must declare dependent libraries in task settings.

host
: Databricks workspace URL, defaults to calling db_host().

token
: Databricks workspace token, defaults to calling db_token().

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_update()

---

db_jobs_run_now *Trigger A New Job Run*

Description

Trigger A New Job Run

Usage

```
db_jobs_run_now(
  job_id,
  jar_params = list(),
  notebook_params = list(),
  python_params = list(),
  spark_submit_params = list(),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

job_id
: The canonical identifier of the job.

jar_params
: Named list. Parameters are used to invoke the main function of the main class specified in the Spark JAR task. If not specified upon run-now, it defaults to an empty list. jar_params cannot be specified in conjunction with notebook_params.

notebook_params
: Named list. Parameters is passed to the notebook and is accessible through the `dbutils.widgets.get` function. If not specified upon run-now, the triggered run uses the job's base parameters.

python_params
: Named list. Parameters are passed to Python file as command-line parameters. If specified upon run-now, it would overwrite the parameters specified in job setting.

spark_submit_params
: Named list. Parameters are passed to spark-submit script as command-line parameters. If specified upon run-now, it would overwrite the parameters specified in job setting.

host
: Databricks workspace URL, defaults to calling `db_host()`.

token
: Databricks workspace token, defaults to calling `db_token()`.

perform_request
: If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

Details

- `*_params` parameters cannot exceed 10,000 bytes when serialized to JSON.
- `jar_params` and `notebook_params` are mutually exclusive.

See Also

Other Jobs API: `db_jobs_create()`, `db_jobs_delete()`, `db_jobs_get()`, `db_jobs_list()`, `db_jobs_reset()`, `db_jobs_runs_cancel()`, `db_jobs_runs_delete()`, `db_jobs_runs_export()`, `db_jobs_runs_get()`, `db_jobs_runs_get_output()`, `db_jobs_runs_list()`, `db_jobs_runs_submit()`, `db_jobs_update()`

`db_jobs_update` *Partially Update A Job*

Description

Partially Update A Job

Usage

```
db_jobs_update(
  job_id,
  fields_to_remove = list(),
  name = NULL,
  schedule = NULL,
  tasks = NULL,
  job_clusters = NULL,
  email_notifications = NULL,
  timeout_seconds = NULL,
```

```
  max_concurrent_runs = NULL,
  access_control_list = NULL,
  git_source = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

job_id	The canonical identifier of the job.
fields_to_remove	Remove top-level fields in the job settings. Removing nested fields is not supported. This field is optional. Must be a `list()`.
name	Name for the job.
schedule	Instance of `cron_schedule()`.
tasks	Task specifications to be executed by this job. Use `job_tasks()`.
job_clusters	Named list of job cluster specifications (using `new_cluster()`) that can be shared and reused by tasks of this job. Libraries cannot be declared in a shared job cluster. You must declare dependent libraries in task settings.
email_notifications	Instance of `email_notifications()`.
timeout_seconds	An optional timeout applied to each run of this job. The default behavior is to have no timeout.
max_concurrent_runs	Maximum allowed number of concurrent runs of the job. Set this value if you want to be able to execute multiple runs of the same job concurrently. This setting affects only new runs. This value cannot exceed 1000. Setting this value to 0 causes all new runs to be skipped. The default behavior is to allow only 1 concurrent run.
access_control_list	Instance of `access_control_request()`.
git_source	Optional specification for a remote repository containing the notebooks used by this job's notebook tasks. Instance of `git_source()`.
host	Databricks workspace URL, defaults to calling `db_host()`.
token	Databricks workspace token, defaults to calling `db_token()`.
perform_request	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

Details

Parameters which are shared with `db_jobs_create()` are optional, only specify those that are changing.

See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_reset(), db_jobs_run_now(), db_jobs_runs_cancel(), db_jobs_runs_delete(), db_jobs_runs_export(), db_jobs_runs_get(), db_jobs_runs_get_output(), db_jobs_runs_list(), db_jobs_runs_submit()

`db_libs_all_cluster_statuses`
Get Status of All Libraries on All Clusters

Description

Get Status of All Libraries on All Clusters

Usage

```
db_libs_all_cluster_statuses(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`host`	Databricks workspace URL, defaults to calling db_host().
`token`	Databricks workspace token, defaults to calling db_token().
`perform_request`	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

A status will be available for all libraries installed on clusters via the API or the libraries UI as well as libraries set to be installed on all clusters via the libraries UI.

If a library has been set to be installed on all clusters, `is_library_for_all_clusters` will be true, even if the library was also installed on this specific cluster.

See Also

Other Libraries API: db_libs_cluster_status(), db_libs_install(), db_libs_uninstall()

db_libs_cluster_status *Get Status of Libraries on Cluster*

Description

Get Status of Libraries on Cluster

Usage

```
db_libs_cluster_status(
  cluster_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Unique identifier of a Databricks cluster.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

wait_for_lib_installs()

Other Libraries API: db_libs_all_cluster_statuses(), db_libs_install(), db_libs_uninstall()

db_libs_install *Install Library on Cluster*

Description

Install Library on Cluster

Usage

```
db_libs_install(
  cluster_id,
  libraries,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Unique identifier of a Databricks cluster.
libraries	An object created by libraries() and the appropriate lib_*() functions.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Installation is asynchronous - it completes in the background after the request.

This call will fail if the cluster is terminated. Installing a wheel library on a cluster is like running the pip command against the wheel file directly on driver and executors.

Installing a wheel library on a cluster is like running the pip command against the wheel file directly on driver and executors. All the dependencies specified in the library setup.py file are installed and this requires the library name to satisfy the wheel file name convention.

The installation on the executors happens only when a new task is launched. With Databricks Runtime 7.1 and below, the installation order of libraries is nondeterministic. For wheel libraries, you can ensure a deterministic installation order by creating a zip file with suffix .wheelhouse.zip that includes all the wheel files.

See Also

lib_egg(), lib_cran(), lib_jar(), lib_maven(), lib_pypi(), lib_whl()

Other Libraries API: db_libs_all_cluster_statuses(), db_libs_cluster_status(), db_libs_uninstall()

---

db_libs_uninstall *Uninstall Library on Cluster*

Description

Uninstall Library on Cluster

Usage

```
db_libs_uninstall(
  cluster_id,
  libraries,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

cluster_id	Unique identifier of a Databricks cluster.
libraries	An object created by libraries() and the appropriate lib_*() functions.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

The libraries aren't uninstalled until the cluster is restarted.

Uninstalling libraries that are not installed on the cluster has no impact but is not an error.

See Also

Other Libraries API: db_libs_all_cluster_statuses(), db_libs_cluster_status(), db_libs_install()

---

db_mlflow_model_approve_transition_req

Approve Model Version Stage Transition Request

Description

Approve Model Version Stage Transition Request

Usage

```
db_mlflow_model_approve_transition_req(
  name,
  version,
  stage = c("None", "Staging", "Production", "Archived"),
  archive_existing_versions = TRUE,
  comment = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name of the model.
version	Version of the model.
stage	Target stage of the transition. Valid values are: None, Staging, Production, Archived.

archive_existing_versions
: Boolean (Default: TRUE). Specifies whether to archive all current model versions in the target stage.

comment
: User-provided comment on the action.

host
: Databricks workspace URL, defaults to calling db_host().

token
: Databricks workspace token, defaults to calling db_token().

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_delete_transition_req(), db_mlflow_model_open_transition_reqs(), db_mlflow_model_reject_transition_req(), db_mlflow_model_transition_req(), db_mlflow_model_transition_, db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_, db_mlflow_registered_model_details()

---

db_mlflow_model_delete_transition_req

Delete a Model Version Stage Transition Request

Description

Delete a Model Version Stage Transition Request

Usage

```
db_mlflow_model_delete_transition_req(
  name,
  version,
  stage = c("None", "Staging", "Production", "Archived"),
  creator,
  comment = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name
: Name of the model.

version
: Version of the model.

stage
: Target stage of the transition. Valid values are: None, Staging, Production, Archived.

creator	Username of the user who created this request. Of the transition requests matching the specified details, only the one transition created by this user will be deleted.
comment	User-provided comment on the action.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_open_transition_reqs(
db_mlflow_model_reject_transition_req(), db_mlflow_model_transition_req(), db_mlflow_model_transition_
db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_c
db_mlflow_registered_model_details()

---

db_mlflow_model_open_transition_reqs

Get All Open Stage Transition Requests for the Model Version

Description

Get All Open Stage Transition Requests for the Model Version

Usage

```
db_mlflow_model_open_transition_reqs(
  name,
  version,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name of the model.
version	Version of the model.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req(), db_mlflow_model_reject_transition_req(), db_mlflow_model_transition_req(), db_mlflow_model_transition_req(), db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_comment_delete(), db_mlflow_registered_model_details()

`db_mlflow_model_reject_transition_req`

Reject Model Version Stage Transition Request

Description

Reject Model Version Stage Transition Request

Usage

```
db_mlflow_model_reject_transition_req(
  name,
  version,
  stage = c("None", "Staging", "Production", "Archived"),
  comment = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`name`	Name of the model.
`version`	Version of the model.
`stage`	Target stage of the transition. Valid values are: `None`, `Staging`, `Production`, `Archived`.
`comment`	User-provided comment on the action.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req(), db_mlflow_model_open_transition_reqs(), db_mlflow_model_transition_req(), db_mlflow_model_transition_req(), db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_comment_delete(), db_mlflow_registered_model_details()

`db_mlflow_model_transition_req` *Make a Model Version Stage Transition Request*

Description

Make a Model Version Stage Transition Request

Usage

```
db_mlflow_model_transition_req(
  name,
  version,
  stage = c("None", "Staging", "Production", "Archived"),
  comment = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`name`	Name of the model.
`version`	Version of the model.
`stage`	Target stage of the transition. Valid values are: `None`, `Staging`, `Production`, `Archived`.
`comment`	User-provided comment on the action.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: `db_mlflow_model_approve_transition_req()`, `db_mlflow_model_delete_transition_req()`, `db_mlflow_model_open_transition_reqs()`, `db_mlflow_model_reject_transition_req()`, `db_mlflow_model_transition_req()`, `db_mlflow_model_version_comment()`, `db_mlflow_model_version_comment_delete()`, `db_mlflow_model_version_details()`, `db_mlflow_registered_model_details()`

`db_mlflow_model_transition_stage` *Transition a Model Version's Stage*

Description

Transition a Model Version's Stage

Usage

```
db_mlflow_model_transition_stage(
  name,
  version,
  stage = c("None", "Staging", "Production", "Archived"),
  archive_existing_versions = TRUE,
  comment = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`name`	Name of the model.
`version`	Version of the model.
`stage`	Target stage of the transition. Valid values are: `None`, `Staging`, `Production`, `Archived`.
`archive_existing_versions`	Boolean (Default: `TRUE`). Specifies whether to archive all current model versions in the target stage.
`comment`	User-provided comment on the action.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

Details

This is a Databricks version of the MLflow endpoint that also accepts a comment associated with the transition to be recorded.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req
db_mlflow_model_open_transition_reqs(), db_mlflow_model_reject_transition_req(), db_mlflow_model_trans
db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_
db_mlflow_registered_model_details()

`db_mlflow_model_version_comment`

Make a Comment on a Model Version

Description

Make a Comment on a Model Version

Usage

```
db_mlflow_model_version_comment(
  name,
  version,
  comment,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

`name`	Name of the model.
`version`	Version of the model.
`comment`	User-provided comment on the action.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.
`perform_request`	If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req
db_mlflow_model_open_transition_reqs(), db_mlflow_model_reject_transition_req(), db_mlflow_model_trans
db_mlflow_model_transition_stage(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_
db_mlflow_registered_model_details()

db_mlflow_model_version_comment_delete
Delete a Comment on a Model Version

Description

Delete a Comment on a Model Version

Usage

```
db_mlflow_model_version_comment_delete(
  id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	Unique identifier of an activity.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req(), db_mlflow_model_open_transition_reqs(), db_mlflow_model_reject_transition_req(), db_mlflow_model_transition_stage(), db_mlflow_model_version_comment(), db_mlflow_model_version_comment(), db_mlflow_registered_model_details()

db_mlflow_model_version_comment_edit
Edit a Comment on a Model Version

Description

Edit a Comment on a Model Version

Usage

```
db_mlflow_model_version_comment_edit(
  id,
  comment,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	Unique identifier of an activity.
comment	User-provided comment on the action.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req(), db_mlflow_model_open_transition_reqs(), db_mlflow_model_reject_transition_req(), db_mlflow_model_transition_stage(), db_mlflow_model_version_comment(), db_mlflow_model_version_comment(), db_mlflow_registered_model_details()

---

db_mlflow_registered_model_details

Get Registered Model Details

Description

Get Registered Model Details

Usage

```
db_mlflow_registered_model_details(
  name,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name of the model.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Model Registry API: db_mlflow_model_approve_transition_req(), db_mlflow_model_delete_transition_req
db_mlflow_model_open_transition_reqs(), db_mlflow_model_reject_transition_req(), db_mlflow_model_trans
db_mlflow_model_transition_stage(), db_mlflow_model_version_comment(), db_mlflow_model_version_comment
db_mlflow_model_version_comment_edit()

---

db_perform_request *Perform Databricks API Request*

Description

Perform Databricks API Request

Usage

```
db_perform_request(req, ...)
```

Arguments

req	{httr2} request.
...	Parameters passed to httr2::resp_body_json()

See Also

Other Request Helpers: db_req_error_body(), db_request(), db_request_json()

db_read_netrc *Read .netrc File*

Description

Read .netrc File

Usage

```
db_read_netrc(path = "~/.netrc")
```

Arguments

path	path of .netrc file, default is ~/.netrc.

Value

named list of .netrc entries

See Also

Other Databricks Authentication Helpers: db_host(), db_token(), db_wsid()

db_repl *Remote REPL to Databricks Cluster*

Description

Remote REPL to Databricks Cluster

Usage

```
db_repl(
  cluster_id,
  language = c("r", "py", "scala", "sql", "sh"),
  host = db_host(),
  token = db_token()
)
```

Arguments

cluster_id	Cluster Id to create REPL context against.
language	for REPL ('r', 'py', 'scala', 'sql', 'sh') are supported.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().

Details

db_repl() will take over the existing console and allow execution of commands against a Databricks cluster. For RStudio users there are Addins which can be bound to keyboard shortcuts to improve usability.

---

db_repo_create *Create Repo*

Description

Creates a repo in the workspace and links it to the remote Git repo specified.

Usage

```
db_repo_create(
  url,
  provider,
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

url	URL of the Git repository to be linked.
provider	Git provider. This field is case-insensitive. The available Git providers are gitHub, bitbucketCloud, gitLab, azureDevOpsServices, gitHubEnterprise, bitbucketServer and gitLabEnterpriseEdition.
path	Desired path for the repo in the workspace. Must be in the format /Repos/{folder}/{repo-name}.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Repos API: db_repo_delete(), db_repo_get(), db_repo_get_all(), db_repo_update()

db_repo_delete *Delete Repo*

Description

Deletes the specified repo

Usage

```
db_repo_delete(
  repo_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

repo_id	The ID for the corresponding repo to access.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Repos API: db_repo_create(), db_repo_get(), db_repo_get_all(), db_repo_update()

db_repo_get *Get Repo*

Description

Returns the repo with the given repo ID.

Usage

```
db_repo_get(
  repo_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

repo_id	The ID for the corresponding repo to access.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Repos API: db_repo_create(), db_repo_delete(), db_repo_get_all(), db_repo_update()

---

db_repo_get_all *Get All Repos*

Description

Get All Repos

Usage

```
db_repo_get_all(
  path_prefix,
  next_page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path_prefix	Filters repos that have paths starting with the given path prefix.
next_page_token	Token used to get the next page of results. If not specified, returns the first page of results as well as a next page token if there are more results.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Returns repos that the calling user has Manage permissions on. Results are paginated with each page containing twenty repos.

See Also

Other Repos API: db_repo_create(), db_repo_delete(), db_repo_get(), db_repo_update()

---

db_repo_update *Update Repo*

Description

Updates the repo to the given branch or tag.

Usage

```
db_repo_update(
  repo_id,
  branch = NULL,
  tag = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

repo_id	The ID for the corresponding repo to access.
branch	Branch that the local version of the repo is checked out to.
tag	Tag that the local version of the repo is checked out to.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Specify either branch or tag, not both.

Updating the repo to a tag puts the repo in a detached HEAD state. Before committing new changes, you must update the repo to a branch instead of the detached HEAD.

See Also

Other Repos API: db_repo_create(), db_repo_delete(), db_repo_get(), db_repo_get_all()

db_request *Databricks Request Helper*

Description

Databricks Request Helper

Usage

```
db_request(endpoint, method, version = NULL, body = NULL, host, token, ...)
```

Arguments

endpoint	Databricks REST API Endpoint
method	Passed to `httr2::req_method()`
version	String, API version of endpoint. E.g. `2.0`.
body	Named list, passed to `httr2::req_body_json()`.
host	Databricks host, defaults to `db_host()`.
token	Databricks token, defaults to `db_token()`.
...	Parameters passed on to `httr2::req_body_json()` when body is not `NULL`.

Value

request

See Also

Other Request Helpers: `db_perform_request()`, `db_req_error_body()`, `db_request_json()`

db_request_json *Generate Request JSON*

Description

Generate Request JSON

Usage

```
db_request_json(req)
```

Arguments

req	a httr2 request, ideally from `db_request()`.

Value

JSON string

See Also

Other Request Helpers: db_perform_request(), db_req_error_body(), db_request()

`db_req_error_body` *Propagate Databricks API Errors*

Description

Propagate Databricks API Errors

Usage

```
db_req_error_body(resp)
```

Arguments

resp	Object with class httr2_response.

See Also

Other Request Helpers: db_perform_request(), db_request(), db_request_json()

`db_secrets_delete` *Delete Secret in Secret Scope*

Description

Delete Secret in Secret Scope

Usage

```
db_secrets_delete(
  scope,
  key,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope that contains the secret to delete.
key	Name of the secret to delete.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

You must have WRITE or MANAGE permission on the secret scope.

- Throws RESOURCE_DOES_NOT_EXIST if no such secret scope or secret exists.
- Throws PERMISSION_DENIED if you do not have permission to make this API call.

See Also

Other Secrets API: db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_get(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_delete(), db_secrets_scope_list_all()

---

db_secrets_list *List Secrets in Secret Scope*

Description

List Secrets in Secret Scope

Usage

```
db_secrets_list(
  scope,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope whose secrets you want to list
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

This is a metadata-only operation; you cannot retrieve secret data using this API. You must have READ permission to make this call.

The last_updated_timestamp returned is in milliseconds since epoch.

- Throws RESOURCE_DOES_NOT_EXIST if no such secret scope exists.
- Throws PERMISSION_DENIED if you do not have permission to make this API call.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_get(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_delete(), db_secrets_scope_list_all()

---

db_secrets_put *Put Secret in Secret Scope*

Description

Insert a secret under the provided scope with the given name.

Usage

```
db_secrets_put(
  scope,
  key,
  value,
  as_bytes = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope to which the secret will be associated with
key	Unique name to identify the secret.
value	Contents of the secret to store, must be a string.
as_bytes	Boolean (default: FALSE). Determines if value is stored as bytes.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

If a secret already exists with the same name, this command overwrites the existing secret's value.

The server encrypts the secret using the secret scope's encryption settings before storing it. You must have `WRITE` or `MANAGE` permission on the secret scope.

The secret key must consist of alphanumeric characters, dashes, underscores, and periods, and cannot exceed 128 characters. The maximum allowed secret value size is 128 KB. The maximum number of secrets in a given scope is 1000.

You can read a secret value only from within a command on a cluster (for example, through a notebook); there is no API to read a secret value outside of a cluster. The permission applied is based on who is invoking the command and you must have at least `READ` permission.

The input fields `string_value` or `bytes_value` specify the type of the secret, which will determine the value returned when the secret value is requested. Exactly one must be specified, this function interfaces these parameters via `as_bytes` which defaults to `FALSE`.

- Throws `RESOURCE_DOES_NOT_EXIST` if no such secret scope exists.
- Throws `RESOURCE_LIMIT_EXCEEDED` if maximum number of secrets in scope is exceeded.
- Throws `INVALID_PARAMETER_VALUE` if the key name or value length is invalid.
- Throws `PERMISSION_DENIED` if the user does not have permission to make this API call.

See Also

Other Secrets API: `db_secrets_delete()`, `db_secrets_list()`, `db_secrets_scope_acl_delete()`, `db_secrets_scope_acl_get()`, `db_secrets_scope_acl_list()`, `db_secrets_scope_acl_put()`, `db_secrets_scope_create()`, `db_secrets_scope_delete()`, `db_secrets_scope_list_all()`

`db_secrets_scope_acl_delete` *Delete Secret Scope ACL*

Description

Delete the given ACL on the given scope.

Usage

```
db_secrets_scope_acl_delete(
  scope,
  principal,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope to remove permissions.
principal	Principal to remove an existing ACL.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

You must have the MANAGE permission to invoke this API.

- Throws RESOURCE_DOES_NOT_EXIST if no such secret scope, principal, or ACL exists.
- Throws PERMISSION_DENIED if you do not have permission to make this API call.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_get(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_delete(), db_secrets_scope_list_all()

---

db_secrets_scope_acl_get

Get Secret Scope ACL

Description

Get Secret Scope ACL

Usage

```
db_secrets_scope_acl_get(
  scope,
  principal,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope to fetch ACL information from.
principal	Principal to fetch ACL information from.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

You must have the MANAGE permission to invoke this

- Throws RESOURCE_DOES_NOT_EXIST if no such secret scope exists.
- Throws PERMISSION_DENIED if you do not have permission to make this API call.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_delete(), db_secrets_scope_list_all()

---

db_secrets_scope_acl_list *List Secret Scope ACL's*

Description

List Secret Scope ACL's

Usage

```
db_secrets_scope_acl_list(
  scope,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope to fetch ACL information from.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

You must have the MANAGE permission to invoke this API.

- Throws RESOURCE_DOES_NOT_EXIST if no such secret scope exists.
- Throws PERMISSION_DENIED if you do not have permission to make this API call.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_get(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_delete(), db_secrets_scope_list_all()

---

db_secrets_scope_acl_put

Put ACL on Secret Scope

Description

Put ACL on Secret Scope

Usage

```
db_secrets_scope_acl_put(
  scope,
  principal,
  permission = c("READ", "WRITE", "MANAGE"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope to apply permissions.
principal	Principal to which the permission is applied
permission	Permission level applied to the principal. One of READ, WRITE, MANAGE.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Create or overwrite the ACL associated with the given principal (user or group) on the specified scope point. In general, a user or group will use the most powerful permission available to them, and permissions are ordered as follows:

- `MANAGE` - Allowed to change ACLs, and read and write to this secret scope.
- `WRITE` - Allowed to read and write to this secret scope.
- `READ` - Allowed to read this secret scope and list what secrets are available.

You must have the `MANAGE` permission to invoke this API.

The principal is a user or group name corresponding to an existing Databricks principal to be granted or revoked access.

- Throws `RESOURCE_DOES_NOT_EXIST` if no such secret scope exists.
- Throws `RESOURCE_ALREADY_EXISTS` if a permission for the principal already exists.
- Throws `INVALID_PARAMETER_VALUE` if the permission is invalid.
- Throws `PERMISSION_DENIED` if you do not have permission to make this API call.

See Also

Other Secrets API: `db_secrets_delete()`, `db_secrets_list()`, `db_secrets_put()`, `db_secrets_scope_acl_delete()`, `db_secrets_scope_acl_get()`, `db_secrets_scope_acl_list()`, `db_secrets_scope_create()`, `db_secrets_scope_delete()`, `db_secrets_scope_list_all()`

---

db_secrets_scope_create

Create Secret Scope

Description

Create Secret Scope

Usage

```
db_secrets_scope_create(
  scope,
  initial_manage_principal = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Scope name requested by the user. Scope names are unique.
initial_manage_principal	The principal that is initially granted MANAGE permission to the created scope.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Create a Databricks-backed secret scope in which secrets are stored in Databricks-managed storage and encrypted with a cloud-based specific encryption key.

The scope name:

- Must be unique within a workspace.
- Must consist of alphanumeric characters, dashes, underscores, and periods, and may not exceed 128 characters.

The names are considered non-sensitive and are readable by all users in the workspace. A workspace is limited to a maximum of 100 secret scopes.

If initial_manage_principal is specified, the initial ACL applied to the scope is applied to the supplied principal (user or group) with MANAGE permissions. The only supported principal for this option is the group users, which contains all users in the workspace. If initial_manage_principal is not specified, the initial ACL with MANAGE permission applied to the scope is assigned to the API request issuer's user identity.

- Throws RESOURCE_ALREADY_EXISTS if a scope with the given name already exists.
- Throws RESOURCE_LIMIT_EXCEEDED if maximum number of scopes in the workspace is exceeded.
- Throws INVALID_PARAMETER_VALUE if the scope name is invalid.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_get(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_delete(), db_secrets_scope_list_all()

db_secrets_scope_delete *Delete Secret Scope*

Description

Delete Secret Scope

Usage

```
db_secrets_scope_delete(
  scope,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

scope	Name of the scope to delete.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

- Throws RESOURCE_DOES_NOT_EXIST if the scope does not exist.
- Throws PERMISSION_DENIED if the user does not have permission to make this API call.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_get(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_list_all()

db_secrets_scope_list_all *List Secret Scopes*

Description

List Secret Scopes

Usage

```
db_secrets_scope_list_all(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

- Throws PERMISSION_DENIED if you do not have permission to make this API call.

See Also

Other Secrets API: db_secrets_delete(), db_secrets_list(), db_secrets_put(), db_secrets_scope_acl_delete(), db_secrets_scope_acl_get(), db_secrets_scope_acl_list(), db_secrets_scope_acl_put(), db_secrets_scope_create(), db_secrets_scope_delete()

db_sql_client *Create Databricks SQL Connector Client*

Description

Create Databricks SQL Connector Client

Usage

```
db_sql_client(
  id,
  catalog = NULL,
  schema = NULL,
  compute_type = c("warehouse", "cluster"),
  use_cloud_fetch = FALSE,
  session_configuration = list(),
  host = db_host(),
  token = db_token(),
  workspace_id = db_current_workspace_id(),
  ...
)
```

Arguments

id	String, ID of either the SQL warehouse or all purpose cluster. Important to set compute_type to the associated type of id.
catalog	Initial catalog to use for the connection. Defaults to NULL in which case the default catalog will be used.
schema	Initial schema to use for the connection. Defaults to NULL in which case the default catalog will be used.
compute_type	One of "warehouse" (default) or "cluster", corresponding to associated compute type of the resource specified in id.
use_cloud_fetch	Boolean (default is FALSE). TRUE to send fetch requests directly to the cloud object store to download chunks of data. FALSE to send fetch requests directly to Databricks. If use_cloud_fetch is set to TRUE but network access is blocked, then the fetch requests will fail.
session_configuration	A optional named list of Spark session configuration parameters. Setting a configuration is equivalent to using the SET key=val SQL command. Run the SQL command SET -v to get a full list of available configurations.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
workspace_id	String, workspace Id used to build the http path for the connection. This defaults to using db_wsid() to get DATABRICKS_WSID environment variable. Not required if compute_type is "cluster".
...	passed onto DatabricksSqlClient().

Details

Create client using Databricks SQL Connector.

Value

DatabricksSqlClient()

Examples

```
## Not run:
  client <- db_sql_client(id = "<warehouse_id>", use_cloud_fetch = TRUE)

## End(Not run)
```

---

db_sql_exec_cancel *Cancel SQL Query*

Description

Cancel SQL Query

Usage

```
db_sql_exec_cancel(
  statement_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

statement_id	String, query execution statement_id
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Requests that an executing statement be canceled. Callers must poll for status to see the terminal state.

Read more on Databricks API docs

See Also

Other SQL Execution APIs: db_sql_exec_query(), db_sql_exec_result(), db_sql_exec_status()

db_sql_exec_query *Execute SQL Query*

Description

Execute SQL Query

Usage

```
db_sql_exec_query(
  statement,
  warehouse_id,
  catalog = NULL,
  schema = NULL,
  parameters = NULL,
  row_limit = NULL,
  byte_limit = NULL,
  disposition = c("INLINE", "EXTERNAL_LINKS"),
  format = c("JSON_ARRAY", "ARROW_STREAM", "CSV"),
  wait_timeout = "10s",
  on_wait_timeout = c("CONTINUE", "CANCEL"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

statement	String, the SQL statement to execute. The statement can optionally be parameterized, see parameters.
warehouse_id	String, ID of warehouse upon which to execute a statement.
catalog	String, sets default catalog for statement execution, similar to USE CATALOG in SQL.
schema	String, sets default schema for statement execution, similar to USE SCHEMA in SQL.
parameters	List of Named Lists, parameters to pass into a SQL statement containing parameter markers. A parameter consists of a name, a value, and *optionally* a type. To represent a NULL value, the value field may be omitted or set to NULL explicitly. See docs for more details.
row_limit	Integer, applies the given row limit to the statement's result set, but unlike the LIMIT clause in SQL, it also sets the truncated field in the response to indicate whether the result was trimmed due to the limit or not.

byte_limit
: Integer, applies the given byte limit to the statement's result size. Byte counts are based on internal data representations and might not match the final size in the requested format. If the result was truncated due to the byte limit, then `truncated` in the response is set to true. When using `EXTERNAL_LINKS` disposition, a default byte_limit of 100 GiB is applied if `byte_limit` is not explicitly set.

disposition
: One of `"INLINE"` (default) or `"EXTERNAL_LINKS"`. See docs for details.

format
: One of `"JSON_ARRAY"` (default), `"ARROW_STREAM"`, or `"CSV"`. See docs for details.

wait_timeout
: String, default is `"10s"`. The time in seconds the call will wait for the statement's result set as `Ns`, where `N` can be set to `0` or to a value between `5` and `50`. When set to `0s`, the statement will execute in asynchronous mode and the call will not wait for the execution to finish. In this case, the call returns directly with `PENDING` state and a statement ID which can be used for polling with `db_sql_exec_status()`.

  When set between `5` and `50` seconds, the call will behave synchronously up to this timeout and wait for the statement execution to finish. If the execution finishes within this time, the call returns immediately with a manifest and result data (or a `FAILED` state in case of an execution error).

  If the statement takes longer to execute, `on_wait_timeout` determines what should happen after the timeout is reached.

on_wait_timeout
: One of `"CONTINUE"` (default) or `"CANCEL"`. When `wait_timeout > 0s`, the call will block up to the specified time. If the statement execution doesn't finish within this time, `on_wait_timeout` determines whether the execution should continue or be canceled.

  When set to `CONTINUE`, the statement execution continues asynchronously and the call returns a statement ID which can be used for polling with `db_sql_exec_status()`.

  When set to `CANCEL`, the statement execution is canceled and the call returns with a `CANCELED` state.

host
: Databricks workspace URL, defaults to calling `db_host()`.

token
: Databricks workspace token, defaults to calling `db_token()`.

perform_request
: If `TRUE` (default) the request is performed, if `FALSE` the httr2 request is returned *without* being performed.

Details

Refer to the web documentation for detailed material on interaction of the various parameters and general recommendations

See Also

Other SQL Execution APIs: `db_sql_exec_cancel()`, `db_sql_exec_result()`, `db_sql_exec_status()`

db_sql_exec_result *Get SQL Query Results*

Description

Get SQL Query Results

Usage

```
db_sql_exec_result(
  statement_id,
  chunk_index,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

statement_id	String, query execution statement_id
chunk_index	Integer, chunk index to fetch result. Starts from 0.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

After the statement execution has SUCCEEDED, this request can be used to fetch any chunk by index.

Whereas the first chunk with chunk_index = 0 is typically fetched with db_sql_exec_result() or db_sql_exec_status(), this request can be used to fetch subsequent chunks

The response structure is identical to the nested result element described in the db_sql_exec_result() request, and similarly includes the next_chunk_index and next_chunk_internal_link fields for simple iteration through the result set.

Read more on Databricks API docs

See Also

Other SQL Execution APIs: db_sql_exec_cancel(), db_sql_exec_query(), db_sql_exec_status()

db_sql_exec_status *Get SQL Query Status*

Description

Get SQL Query Status

Usage

```
db_sql_exec_status(
  statement_id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

statement_id	String, query execution statement_id
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

This request can be used to poll for the statement's status. When the status.state field is SUCCEEDED it will also return the result manifest and the first chunk of the result data.

When the statement is in the terminal states CANCELED, CLOSED or FAILED, it returns HTTP 200 with the state set.

After at least 12 hours in terminal state, the statement is removed from the warehouse and further calls will receive an HTTP 404 response.

Read more on Databricks API docs

See Also

Other SQL Execution APIs: db_sql_exec_cancel(), db_sql_exec_query(), db_sql_exec_result()

db_sql_global_warehouse_get *Get Global Warehouse Config*

Description

Get Global Warehouse Config

Usage

```
db_sql_global_warehouse_get(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_warehouse_create(), db_sql_warehouse_delete(), db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_start(), db_sql_warehouse_stop(), get_and_start_warehouse()

db_sql_query_history *List Warehouse Query History*

Description

For more details refer to the query history documentation. This function elevates the sub-components of filter_by parameter to the R function directly.

Usage

```
db_sql_query_history(
  statuses = NULL,
  user_ids = NULL,
  endpoint_ids = NULL,
  start_time_ms = NULL,
  end_time_ms = NULL,
  max_results = 100,
  page_token = NULL,
  include_metrics = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

statuses	Allows filtering by query status. Possible values are: QUEUED, RUNNING, CANCELED, FAILED, FINISHED. Multiple permitted.
user_ids	Allows filtering by user ID's. Multiple permitted.
endpoint_ids	Allows filtering by endpoint ID's. Multiple permitted.
start_time_ms	Integer, limit results to queries that started after this time.
end_time_ms	Integer, limit results to queries that started before this time.
max_results	Limit the number of results returned in one page. Default is 100.
page_token	Opaque token used to get the next page of results. Optional.
include_metrics	Whether to include metrics about query execution.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

By default the filter parameters statuses, user_ids, and endpoints_ids are NULL.

---

db_sql_warehouse_create

Create Warehouse

Description

Create Warehouse

Usage

```
db_sql_warehouse_create(
  name,
  cluster_size,
  min_num_clusters = 1,
  max_num_clusters = 1,
  auto_stop_mins = 30,
  tags = list(),
  spot_instance_policy = c("COST_OPTIMIZED", "RELIABILITY_OPTIMIZED"),
  enable_photon = TRUE,
  warehouse_type = c("CLASSIC", "PRO"),
  enable_serverless_compute = NULL,
  disable_uc = FALSE,
  channel = c("CHANNEL_NAME_CURRENT", "CHANNEL_NAME_PREVIEW"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name of the SQL warehouse. Must be unique.
cluster_size	Size of the clusters allocated to the warehouse. One of 2X-Small, X-Small, Small, Medium, Large, X-Large, 2X-Large, 3X-Large, 4X-Large.
min_num_clusters	Minimum number of clusters available when a SQL warehouse is running. The default is 1.
max_num_clusters	Maximum number of clusters available when a SQL warehouse is running. If multi-cluster load balancing is not enabled, this is limited to 1.
auto_stop_mins	Time in minutes until an idle SQL warehouse terminates all clusters and stops. Defaults to 30. For Serverless SQL warehouses (enable_serverless_compute = TRUE), set this to 10.
tags	Named list that describes the warehouse. Databricks tags all warehouse resources with these tags.
spot_instance_policy	The spot policy to use for allocating instances to clusters. This field is not used if the SQL warehouse is a Serverless SQL warehouse.
enable_photon	Whether queries are executed on a native vectorized engine that speeds up query execution. The default is TRUE.
warehouse_type	Either "CLASSIC" (default), or "PRO"
enable_serverless_compute	Whether this SQL warehouse is a Serverless warehouse. To use a Serverless SQL warehouse, you must enable Serverless SQL warehouses for the workspace. If Serverless SQL warehouses are disabled for the workspace, the default is FALSE If Serverless SQL warehouses are enabled for the workspace, the default is TRUE.

disable_uc	If TRUE will use Hive Metastore (HMS). If FALSE (default), then it will be enabled for Unity Catalog (UC).
channel	Whether to use the current SQL warehouse compute version or the preview version. Databricks does not recommend using preview versions for production workloads. The default is CHANNEL_NAME_CURRENT.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_delete(), db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_start(), db_sql_warehouse_stop(), get_and_start_warehouse()

---

db_sql_warehouse_delete

Delete Warehouse

Description

Delete Warehouse

Usage

```
db_sql_warehouse_delete(
  id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	ID of the SQL warehouse.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_start(), db_sql_warehouse_stop(), get_and_start_warehouse()

db_sql_warehouse_edit *Edit Warehouse*

Description

Edit Warehouse

Usage

```
db_sql_warehouse_edit(
  id,
  name = NULL,
  cluster_size = NULL,
  min_num_clusters = NULL,
  max_num_clusters = NULL,
  auto_stop_mins = NULL,
  tags = NULL,
  spot_instance_policy = NULL,
  enable_photon = NULL,
  warehouse_type = NULL,
  enable_serverless_compute = NULL,
  channel = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	ID of the SQL warehouse.
name	Name of the SQL warehouse. Must be unique.
cluster_size	Size of the clusters allocated to the warehouse. One of 2X-Small, X-Small, Small, Medium, Large, X-Large, 2X-Large, 3X-Large, 4X-Large.
min_num_clusters	Minimum number of clusters available when a SQL warehouse is running. The default is 1.
max_num_clusters	Maximum number of clusters available when a SQL warehouse is running. If multi-cluster load balancing is not enabled, this is limited to 1.
auto_stop_mins	Time in minutes until an idle SQL warehouse terminates all clusters and stops. Defaults to 30. For Serverless SQL warehouses (enable_serverless_compute = TRUE), set this to 10.
tags	Named list that describes the warehouse. Databricks tags all warehouse resources with these tags.

spot_instance_policy
: The spot policy to use for allocating instances to clusters. This field is not used if the SQL warehouse is a Serverless SQL warehouse.

enable_photon
: Whether queries are executed on a native vectorized engine that speeds up query execution. The default is TRUE.

warehouse_type
: Either "CLASSIC" (default), or "PRO"

enable_serverless_compute
: Whether this SQL warehouse is a Serverless warehouse. To use a Serverless SQL warehouse, you must enable Serverless SQL warehouses for the workspace. If Serverless SQL warehouses are disabled for the workspace, the default is FALSE If Serverless SQL warehouses are enabled for the workspace, the default is TRUE.

channel
: Whether to use the current SQL warehouse compute version or the preview version. Databricks does not recommend using preview versions for production workloads. The default is CHANNEL_NAME_CURRENT.

host
: Databricks workspace URL, defaults to calling db_host().

token
: Databricks workspace token, defaults to calling db_token().

perform_request
: If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Modify a SQL warehouse. All fields are optional. Missing fields default to the current values.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_delete(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_start(), db_sql_warehouse_stop(), get_and_start_warehouse()

---

db_sql_warehouse_get *Get Warehouse*

Description

Get Warehouse

Usage

```
db_sql_warehouse_get(
  id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	ID of the SQL warehouse.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_delete() db_sql_warehouse_edit(), db_sql_warehouse_list(), db_sql_warehouse_start(), db_sql_warehouse_stop(), get_and_start_warehouse()

---

db_sql_warehouse_list *List Warehouses*

Description

List Warehouses

Usage

```
db_sql_warehouse_list(
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_delete() db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_start(), db_sql_warehouse_stop(), get_and_start_warehouse()

db_sql_warehouse_start *Start Warehouse*

Description

Start Warehouse

Usage

```
db_sql_warehouse_start(
  id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	ID of the SQL warehouse.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_delete(), db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_stop(), get_and_start_warehouse()

db_sql_warehouse_stop *Stop Warehouse*

Description

Stop Warehouse

Usage

```
db_sql_warehouse_stop(
  id,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

id	ID of the SQL warehouse.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_delete(), db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_start(), get_and_start_warehouse()

---

db_token *Fetch Databricks Token*

Description

The function will check for a token in the DATABRICKS_HOST environment variable. .databrickscfg will be searched if db_profile and use_databrickscfg are set or if Posit Workbench managed OAuth credentials are detected. If none of the above are found then will default to using OAuth U2M flow.

Refer to api authentication docs

Usage

```
db_token(profile = default_config_profile())
```

Arguments

profile	Profile to use when fetching from environment variable (e.g. .Renviron) or .databricksfg file

Details

The behaviour is subject to change depending if db_profile and use_databrickscfg options are set.

- use_databrickscfg: Boolean (default: FALSE), determines if credentials are fetched from profile of .databrickscfg or .Renviron
- db_profile: String (default: NULL), determines profile used. .databrickscfg will automatically be used when Posit Workbench managed OAuth credentials are detected.

See vignette on authentication for more details.

Value

databricks token

See Also

Other Databricks Authentication Helpers: db_host(), db_read_netrc(), db_wsid()

---

db_volume_delete *Volume FileSystem Delete*

Description

Volume FileSystem Delete

Usage

```
db_volume_delete(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_file_exists(), db_volume_list(), db_volume_read(), db_volume_write()

db_volume_dir_create *Volume FileSystem Create Directory*

Description

Volume FileSystem Create Directory

Usage

```
db_volume_dir_create(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_file_exists(), db_volume_list(), db_volume_read(), db_volume_write()

db_volume_dir_delete *Volume FileSystem Delete Directory*

Description

Volume FileSystem Delete Directory

Usage

```
db_volume_dir_delete(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_exists(), db_volume_file_exists(), db_volume_list(), db_volume_read(), db_volume_write()

---

db_volume_dir_exists *Volume FileSystem Check Directory Exists*

Description

Volume FileSystem Check Directory Exists

Usage

```
db_volume_dir_exists(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_file_exists(), db_volume_list(), db_volume_read(), db_volume_write()

db_volume_file_exists *Volume FileSystem File Status*

Description

Volume FileSystem File Status

Usage

```
db_volume_file_exists(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_list(), db_volume_read(), db_volume_write()

db_volume_list *Volume FileSystem List Directory Contents*

Description

Volume FileSystem List Directory Contents

Usage

```
db_volume_list(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_file_exists(), db_volume_read(), db_volume_write()

---

db_volume_read *Volume FileSystem Read*

Description

Return the contents of a file within a volume (up to 2GiB).

Usage

```
db_volume_read(
  path,
  destination,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
destination	Path to write downloaded file to.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_file_exists(), db_volume_list(), db_volume_write()

db_volume_write *Volume FileSystem Write*

Description

Upload a file to volume filesystem.

Usage

```
db_volume_write(
  path,
  file = NULL,
  overwrite = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the file in the Files API, omitting the initial slash.
file	Path to a file on local system, takes precedent over path.
overwrite	Flag (Default: FALSE) that specifies whether to overwrite existing files.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Uploads a file of up to 5 GiB.

See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_file_exists(), db_volume_list(), db_volume_read()

db_vs_endpoints_create *Create a Vector Search Endpoint*

Description

Create a Vector Search Endpoint

Usage

```
db_vs_endpoints_create(
  name,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name of vector search endpoint
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

This function can take a few moments to run.

See Also

Other Vector Search API: db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_endpoints_delete *Delete a Vector Search Endpoint*

Description

Delete a Vector Search Endpoint

Usage

```
db_vs_endpoints_delete(
  endpoint,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

endpoint	Name of vector search endpoint
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_endpoints_get *Get a Vector Search Endpoint*

Description

Get a Vector Search Endpoint

Usage

```
db_vs_endpoints_get(
  endpoint,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

endpoint	Name of vector search endpoint
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

---

db_vs_endpoints_list *List Vector Search Endpoints*

Description

List Vector Search Endpoints

Usage

```
db_vs_endpoints_list(
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

page_token	Token for pagination
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

---

db_vs_indexes_create *Create a Vector Search Index*

Description

Create a Vector Search Index

Usage

```
db_vs_indexes_create(
  name,
  endpoint,
  primary_key,
  spec,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

name	Name of vector search index
endpoint	Name of vector search endpoint
primary_key	Vector search primary key column name
spec	Either delta_sync_index_spec() or direct_access_index_spec().
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

---

db_vs_indexes_delete *Delete a Vector Search Index*

Description

Delete a Vector Search Index

Usage

```
db_vs_indexes_delete(
  index,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_indexes_delete_data *Delete Data from a Vector Search Index*

Description

Delete Data from a Vector Search Index

Usage

```
db_vs_indexes_delete_data(
  index,
  primary_keys,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
primary_keys	primary keys to be deleted from index
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_indexes_get *Get a Vector Search Index*

Description

Get a Vector Search Index

Usage

```
db_vs_indexes_get(
  index,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

---

db_vs_indexes_list *List Vector Search Indexes*

Description

List Vector Search Indexes

Usage

```
db_vs_indexes_list(
  endpoint,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

endpoint	Name of vector search endpoint
page_token	page_token returned from prior query
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

---

db_vs_indexes_query *Query a Vector Search Index*

Description

Query a Vector Search Index

Usage

```
db_vs_indexes_query(
  index,
  columns,
  filters_json,
  query_vector = NULL,
  query_text = NULL,
  score_threshold = 0,
  query_type = c("ANN", "HYBRID"),
  num_results = 10,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
columns	Column names to include in response
filters_json	JSON string representing query filters, see details.

query_vector	Numeric vector. Required for direct vector access index and delta sync index using self managed vectors.
query_text	Required for delta sync index using model endpoint.
score_threshold	Numeric score threshold for the approximate nearest neighbour (ANN) search. Defaults to 0.0.
query_type	One of ANN (default) or HYBRID
num_results	Number of returns to return (default: 10).
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

You cannot specify both query_vector and query_text at the same time.

filter_jsons examples:

- '{"id <": 5}': Filter for id less than 5
- '{"id >": 5}': Filter for id greater than 5
- '{"id <=": 5}': Filter for id less than equal to 5
- '{"id >=": 5}': Filter for id greater than equal to 5
- '{"id": 5}': Filter for id equal to 5
- '{"id": 5, "age >=": 18}': Filter for id equal to 5 and age greater than equal to 18

filter_jsons will convert attempt to use jsonlite::toJSON on any non character vectors.

Refer to docs for Vector Search.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

Examples

```
## Not run:
db_vs_indexes_sync(
  index = "myindex",
  columns = c("id", "text"),
  query_vector = c(1, 2, 3)
)

## End(Not run)
```

db_vs_indexes_query_next_page
Query Vector Search Next Page

Description

Query Vector Search Next Page

Usage

```
db_vs_indexes_query_next_page(
  index,
  endpoint,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
endpoint	Name of vector search endpoint
page_token	page_token returned from prior query
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_indexes_scan *Scan a Vector Search Index*

Description

Scan a Vector Search Index

Usage

```
db_vs_indexes_scan(
  endpoint,
  index,
  last_primary_key,
  num_results = 10,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

endpoint	Name of vector search endpoint to scan
index	Name of vector search index to scan
last_primary_key	Primary key of the last entry returned in previous scan
num_results	Number of returns to return (default: 10)
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Scan the specified vector index and return the first num_results entries after the exclusive primary_key.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_indexes_sync *Synchronize a Vector Search Index*

Description

Synchronize a Vector Search Index

Usage

```
db_vs_indexes_sync(
  index,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Triggers a synchronization process for a specified vector index. The index must be a 'Delta Sync' index.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_vs_indexes_upsert_data *Upsert Data into a Vector Search Index*

Description

Upsert Data into a Vector Search Index

Usage

```
db_vs_indexes_upsert_data(
  index,
  df,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

index	Name of vector search index
df	data.frame containing data to upsert
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

db_workspace_delete *Delete Object/Directory (Workspaces)*

Description

Delete Object/Directory (Workspaces)

Usage

```
db_workspace_delete(
  path,
  recursive = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the notebook or directory.
recursive	Flag that specifies whether to delete the object recursively. False by default.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Delete an object or a directory (and optionally recursively deletes all objects in the directory). If path does not exist, this call returns an error RESOURCE_DOES_NOT_EXIST. If path is a non-empty directory and recursive is set to false, this call returns an error DIRECTORY_NOT_EMPTY.

Object deletion cannot be undone and deleting a directory recursively is not atomic.

See Also

Other Workspace API: db_workspace_export(), db_workspace_get_status(), db_workspace_import(), db_workspace_list(), db_workspace_mkdirs()

---

db_workspace_export *Export Notebook or Directory (Workspaces)*

Description

Export Notebook or Directory (Workspaces)

Usage

```
db_workspace_export(
  path,
  format = c("AUTO", "SOURCE", "HTML", "JUPYTER", "DBC", "R_MARKDOWN"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the notebook or directory.
format	One of AUTO, SOURCE, HTML, JUPYTER, DBC, R_MARKDOWN. Default is SOURCE.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Export a notebook or contents of an entire directory. If path does not exist, this call returns an error RESOURCE_DOES_NOT_EXIST.

You can export a directory only in DBC format. If the exported data exceeds the size limit, this call returns an error MAX_NOTEBOOK_SIZE_EXCEEDED. This API does not support exporting a library.

At this time we do not support the direct_download parameter and returns a base64 encoded string.

See More.

Value

base64 encoded string

See Also

Other Workspace API: db_workspace_delete(), db_workspace_get_status(), db_workspace_import(), db_workspace_list(), db_workspace_mkdirs()

---

db_workspace_get_status

Get Object Status (Workspaces)

Description

Gets the status of an object or a directory.

Usage

```
db_workspace_get_status(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the notebook or directory.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

If path does not exist, this call returns an error RESOURCE_DOES_NOT_EXIST.

See Also

Other Workspace API: db_workspace_delete(), db_workspace_export(), db_workspace_import(), db_workspace_list(), db_workspace_mkdirs()

---

db_workspace_import *Import Notebook/Directory (Workspaces)*

Description

Import a notebook or the contents of an entire directory.

Usage

```
db_workspace_import(
  path,
  file = NULL,
  content = NULL,
  format = c("AUTO", "SOURCE", "HTML", "JUPYTER", "DBC", "R_MARKDOWN"),
  language = NULL,
  overwrite = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the notebook or directory.
file	Path of local file to upload. See formats parameter.
content	Content to upload, this will be base64-encoded and has a limit of 10MB.
format	One of AUTO, SOURCE, HTML, JUPYTER, DBC, R_MARKDOWN. Default is SOURCE.

language	One of R, PYTHON, SCALA, SQL. Required when format is SOURCE otherwise ignored.
overwrite	Flag that specifies whether to overwrite existing object. FALSE by default. For DBC overwrite is not supported since it may contain a directory.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

file and content are mutually exclusive. If both are specified content will be ignored.

If path already exists and overwrite is set to FALSE, this call returns an error RESOURCE_ALREADY_EXISTS. You can use only DBC format to import a directory.

See Also

Other Workspace API: db_workspace_delete(), db_workspace_export(), db_workspace_get_status(), db_workspace_list(), db_workspace_mkdirs()

---

db_workspace_list *List Directory Contents (Workspaces)*

Description

List Directory Contents (Workspaces)

Usage

```
db_workspace_list(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the notebook or directory.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

List the contents of a directory, or the object if it is not a directory. If the input path does not exist, this call returns an error RESOURCE_DOES_NOT_EXIST.

See Also

Other Workspace API: db_workspace_delete(), db_workspace_export(), db_workspace_get_status(), db_workspace_import(), db_workspace_mkdirs()

---

db_workspace_mkdirs *Make a Directory (Workspaces)*

Description

Make a Directory (Workspaces)

Usage

```
db_workspace_mkdirs(
  path,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

Arguments

path	Absolute path of the directory.
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
perform_request	If TRUE (default) the request is performed, if FALSE the httr2 request is returned *without* being performed.

Details

Create the given directory and necessary parent directories if they do not exists. If there exists an object (not a directory) at any prefix of the input path, this call returns an error RESOURCE_ALREADY_EXISTS. If this operation fails it may have succeeded in creating some of the necessary parent directories.

See Also

Other Workspace API: db_workspace_delete(), db_workspace_export(), db_workspace_get_status(), db_workspace_import(), db_workspace_list()

db_wsid *Fetch Databricks Workspace ID*

Description

Workspace ID, optionally specified to make connections pane more powerful. Specified as an environment variable `DATABRICKS_WSID`. `.databrickscfg` will be searched if `db_profile` and `use_databrickscfg` are set or if Posit Workbench managed OAuth credentials are detected.

Refer to api authentication docs

Usage

```
db_wsid(profile = default_config_profile())
```

Arguments

`profile`	Profile to use when fetching from environment variable (e.g. `.Renviron`) or `.databricksfg` file

Details

The behaviour is subject to change depending if `db_profile` and `use_databrickscfg` options are set.

- `use_databrickscfg`: Boolean (default: `FALSE`), determines if credentials are fetched from profile of `.databrickscfg` or `.Renviron`
- `db_profile`: String (default: `NULL`), determines profile used. `.databrickscfg` will automatically be used when Posit Workbench managed OAuth credentials are detected.

See vignette on authentication for more details.

Value

databricks workspace ID

See Also

Other Databricks Authentication Helpers: `db_host()`, `db_read_netrc()`, `db_token()`

`delta_sync_index_spec` *Delta Sync Vector Search Index Specification*

Description

Delta Sync Vector Search Index Specification

Usage

```
delta_sync_index_spec(
  source_table,
  embedding_writeback_table = NULL,
  embedding_source_columns = NULL,
  embedding_vector_columns = NULL,
  pipeline_type = c("TRIGGERED", "CONTINUOUS")
)
```

Arguments

`source_table`	The name of the source table.
`embedding_writeback_table`	Name of table to sync index contents and computed embeddings back to delta table, see details.
`embedding_source_columns`	The columns that contain the embedding source, must be one or list of `embedding_source_column()`
`embedding_vector_columns`	The columns that contain the embedding, must be one or list of `embedding_vector_column()`
`pipeline_type`	Pipeline execution mode, see details.

Details

`pipeline_type` is either:

- `"TRIGGERED"`: If the pipeline uses the triggered execution mode, the system stops processing after successfully refreshing the source table in the pipeline once, ensuring the table is updated based on the data available when the update started.
- `"CONTINUOUS"` If the pipeline uses continuous execution, the pipeline processes new data as it arrives in the source table to keep vector index fresh.

The only supported naming convention for `embedding_writeback_table` is `"<index_name>_writeback_table"`.

See Also

`db_vs_indexes_create()`

Other Vector Search API: `db_vs_endpoints_create()`, `db_vs_endpoints_delete()`, `db_vs_endpoints_get()`, `db_vs_endpoints_list()`, `db_vs_indexes_create()`, `db_vs_indexes_delete()`, `db_vs_indexes_delete_data()`, `db_vs_indexes_get()`, `db_vs_indexes_list()`, `db_vs_indexes_query()`, `db_vs_indexes_query_next_page()`,

db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), direct_access_index_spec(), embedding_source_column(), embedding_vector_column()

---

direct_access_index_spec *Delta Sync Vector Search Index Specification*

Description

Delta Sync Vector Search Index Specification

Usage

```
direct_access_index_spec(
  embedding_source_columns = NULL,
  embedding_vector_columns = NULL,
  schema
)
```

Arguments

embedding_source_columns	The columns that contain the embedding source, must be one or list of embedding_source_column()
embedding_vector_columns	The columns that contain the embedding, must be one or list of embedding_vector_column() vectors.
schema	Named list, names are column names, values are types. See details.

Details

The supported types are:

- "integer"
- "long"
- "float"
- "double"
- "boolean"
- "string"
- "date"
- "timestamp"
- "array<float>": supported for vector columns
- "array<double>": supported for vector columns

See Also

db_vs_indexes_create()

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), embedding_source_column(), embedding_vector_column()

---

docker_image *Docker Image*

Description

Docker image connection information.

Usage

```
docker_image(url, username, password)
```

Arguments

url	URL for the Docker image.
username	User name for the Docker repository.
password	Password for the Docker repository.

Details

Uses basic authentication, **strongly** recommended that credentials are not stored in any scripts and environment variables should be used.

See Also

db_cluster_create(), db_cluster_edit()

email_notifications *Email Notifications*

Description

Email Notifications

Usage

```
email_notifications(
  on_start = NULL,
  on_success = NULL,
  on_failure = NULL,
  no_alert_for_skipped_runs = TRUE
)
```

Arguments

on_start	List of email addresses to be notified when a run begins. If not specified on job creation, reset, or update, the list is empty, and notifications are not sent.
on_success	List of email addresses to be notified when a run successfully completes. A run is considered to have completed successfully if it ends with a TERMINATED life_cycle_state and a SUCCESSFUL result_state. If not specified on job creation, reset, or update, the list is empty, and notifications are not sent.
on_failure	List of email addresses to be notified when a run unsuccessfully completes. A run is considered to have completed unsuccessfully if it ends with an INTERNAL_ERROR life_cycle_state or a SKIPPED, FAILED, or TIMED_OUT result_state. If this is not specified on job creation, reset, or update the list is empty, and notifications are not sent.
no_alert_for_skipped_runs	If TRUE (default), do not send email to recipients specified in on_failure if the run is skipped.

See Also

job_task()

Other Task Objects: libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_python_task(), spark_submit_task()

embedding_source_column *Embedding Source Column*

Description

Embedding Source Column

Usage

```
embedding_source_column(name, model_endpoint_name)
```

Arguments

name	Name of the column
model_endpoint_name	Name of the embedding model endpoint

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_vector_column()

embedding_vector_column *Embedding Vector Column*

Description

Embedding Vector Column

Usage

```
embedding_vector_column(name, dimension)
```

Arguments

name	Name of the column
dimension	dimension of the embedding vector

See Also

Other Vector Search API: db_vs_endpoints_create(), db_vs_endpoints_delete(), db_vs_endpoints_get(), db_vs_endpoints_list(), db_vs_indexes_create(), db_vs_indexes_delete(), db_vs_indexes_delete_data(), db_vs_indexes_get(), db_vs_indexes_list(), db_vs_indexes_query(), db_vs_indexes_query_next_page(), db_vs_indexes_scan(), db_vs_indexes_sync(), db_vs_indexes_upsert_data(), delta_sync_index_spec(), direct_access_index_spec(), embedding_source_column()

---

file_storage_info *File Storage Information*

Description

File Storage Information

Usage

```
file_storage_info(destination)
```

Arguments

destination	File destination. Example: file:/my/file.sh.

Details

The file storage type is only available for clusters set up using Databricks Container Services.

See Also

init_script_info()

Other Init Script Info Objects: dbfs_storage_info(), s3_storage_info()

---

gcp_attributes *GCP Attributes*

Description

GCP Attributes

Usage

```
gcp_attributes(use_preemptible_executors = TRUE, google_service_account = NULL)
```

Arguments

use_preemptible_executors
: Boolean (Default: TRUE). If TRUE Uses preemptible executors

google_service_account
: Google service account email address that the cluster uses to authenticate with Google Identity. This field is used for authentication with the GCS and BigQuery data sources.

Details

For use with GCS and BigQuery, your Google service account that you use to access the data source must be in the same project as the SA that you specified when setting up your Databricks account.

See Also

db_cluster_create(), db_cluster_edit()

Other Cloud Attributes: aws_attributes(), azure_attributes()

---

get_and_start_cluster *Get and Start Cluster*

Description

Get and Start Cluster

Usage

```
get_and_start_cluster(
  cluster_id,
  polling_interval = 5,
  host = db_host(),
  token = db_token(),
  silent = FALSE
)
```

Arguments

cluster_id	Canonical identifier for the cluster.
polling_interval	Number of seconds to wait between status checks
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
silent	Boolean (default: FALSE), will emit cluster state progress if TRUE.

Details

Get information regarding a Databricks cluster. If the cluster is inactive it will be started and wait until the cluster is active.

Value

`db_cluster_get()`

See Also

`db_cluster_get()` and `db_cluster_start()`.

Other Clusters API: `db_cluster_create()`, `db_cluster_edit()`, `db_cluster_events()`, `db_cluster_get()`, `db_cluster_list()`, `db_cluster_list_node_types()`, `db_cluster_list_zones()`, `db_cluster_perm_delete()`, `db_cluster_pin()`, `db_cluster_resize()`, `db_cluster_restart()`, `db_cluster_runtime_versions()`, `db_cluster_start()`, `db_cluster_terminate()`, `db_cluster_unpin()`, `get_latest_dbr()`

Other Cluster Helpers: `get_latest_dbr()`

---

get_and_start_warehouse *Get and Start Warehouse*

Description

Get and Start Warehouse

Usage

```
get_and_start_warehouse(
  id,
  polling_interval = 5,
  host = db_host(),
  token = db_token()
)
```

Arguments

`id`	ID of the SQL warehouse.
`polling_interval`	Number of seconds to wait between status checks
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.

Details

Get information regarding a Databricks cluster. If the cluster is inactive it will be started and wait until the cluster is active.

Value

db_sql_warehouse_get()

See Also

db_sql_warehouse_get() and db_sql_warehouse_start().

Other Warehouse API: db_sql_global_warehouse_get(), db_sql_warehouse_create(), db_sql_warehouse_delete(), db_sql_warehouse_edit(), db_sql_warehouse_get(), db_sql_warehouse_list(), db_sql_warehouse_start(), db_sql_warehouse_stop()

---

get_latest_dbr *Get Latest Databricks Runtime (DBR)*

Description

Get Latest Databricks Runtime (DBR)

Usage

```
get_latest_dbr(lts, ml, gpu, photon, host = db_host(), token = db_token())
```

Arguments

lts	Boolean, if TRUE returns only LTS runtimes
ml	Boolean, if TRUE returns only ML runtimes
gpu	Boolean, if TRUE returns only ML GPU runtimes
photon	Boolean, if TRUE returns only photon runtimes
host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().

Details

There are runtime combinations that are not possible, such as GPU/ML and photon. This function does not permit invalid combinations.

Value

Named list

See Also

Other Clusters API: db_cluster_create(), db_cluster_edit(), db_cluster_events(), db_cluster_get(), db_cluster_list(), db_cluster_list_node_types(), db_cluster_list_zones(), db_cluster_perm_delete(), db_cluster_pin(), db_cluster_resize(), db_cluster_restart(), db_cluster_runtime_versions(), db_cluster_start(), db_cluster_terminate(), db_cluster_unpin(), get_and_start_cluster()

Other Cluster Helpers: get_and_start_cluster()

git_source *Git Source for Job Notebook Tasks*

Description

Git Source for Job Notebook Tasks

Usage

```
git_source(
  git_url,
  git_provider,
  reference,
  type = c("branch", "tag", "commit")
)
```

Arguments

git_url	URL of the repository to be cloned by this job. The maximum length is 300 characters.
git_provider	Unique identifier of the service used to host the Git repository. Must be one of: `github`, `bitbucketcloud`, `azuredevopsservices`, `githubenterprise`, `bitbucketserver`, `gitlab`, `gitlabenterpriseedition`, `awscodecommit`.
reference	Branch, tag, or commit to be checked out and used by this job.
type	Type of reference being used, one of: `branch`, `tag`, `commit`.

init_script_info *Init Script Info*

Description

Init Script Info

Usage

```
init_script_info(...)
```

Arguments

...	Accepts multiple instances `s3_storage_info()`, `file_storage_info()`, or `dbfs_storage_info()`.

Details

`file_storage_info()` is only available for clusters set up using Databricks Container Services.

For instructions on using init scripts with Databricks Container Services, see Use an init script.

See Also

db_cluster_create(), db_cluster_edit()

---

install_db_sql_connector

Install Databricks SQL Connector (Python)

Description

Install Databricks SQL Connector (Python)

Usage

```
install_db_sql_connector(
  envname = determine_brickster_venv(),
  method = "auto",
  ...
)
```

Arguments

envname	The name, or full path, of the environment in which Python packages are to be installed. When NULL (the default), the active environment as set by the RETICULATE_PYTHON_ENV variable will be used; if that is unset, then the r-reticulate environment will be used.
method	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.
...	Additional arguments passed to conda_install() or virtualenv_install().

Details

Installs databricks-sql-connector. Environemnt is resolved by determine_brickster_venv() which defaults to r-brickster virtualenv.

When running within Databricks it will use the existing python environment.

Examples

```
## Not run: install_db_sql_connector()
```

in_databricks_nb *Detect if running within Databricks Notebook*

Description

Detect if running within Databricks Notebook

Usage

```
in_databricks_nb()
```

Details

R sessions on Databricks can be detected via various environment variables and directories.

Value

Boolean

is.access_control_request *Test if object is of class AccessControlRequest*

Description

Test if object is of class AccessControlRequest

Usage

```
is.access_control_request(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the AccessControlRequest class.

is.access_control_req_group

Test if object is of class AccessControlRequestForGroup

Description

Test if object is of class AccessControlRequestForGroup

Usage

```
is.access_control_req_group(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the AccessControlRequestForGroup class.

is.access_control_req_user

Test if object is of class AccessControlRequestForUser

Description

Test if object is of class AccessControlRequestForUser

Usage

```
is.access_control_req_user(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the AccessControlRequestForUser class.

is.aws_attributes *Test if object is of class AwsAttributes*

Description

Test if object is of class AwsAttributes

Usage

```
is.aws_attributes(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the AwsAttributes class.

is.azure_attributes *Test if object is of class AzureAttributes*

Description

Test if object is of class AzureAttributes

Usage

```
is.azure_attributes(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the AzureAttributes class.

is.cluster_autoscale *Test if object is of class AutoScale*

Description

Test if object is of class AutoScale

Usage

```
is.cluster_autoscale(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the AutoScale class.

is.cluster_log_conf *Test if object is of class ClusterLogConf*

Description

Test if object is of class ClusterLogConf

Usage

```
is.cluster_log_conf(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the ClusterLogConf class.

is.cron_schedule *Test if object is of class CronSchedule*

Description

Test if object is of class CronSchedule

Usage

```
is.cron_schedule(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the CronSchedule class.

is.dbfs_storage_info *Test if object is of class DbfsStorageInfo*

Description

Test if object is of class DbfsStorageInfo

Usage

```
is.dbfs_storage_info(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the DbfsStorageInfo class.

is.delta_sync_index *Test if object is of class DeltaSyncIndex*

Description

Test if object is of class DeltaSyncIndex

Usage

```
is.delta_sync_index(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the DeltaSyncIndex class.

is.direct_access_index *Test if object is of class DirectAccessIndex*

Description

Test if object is of class DirectAccessIndex

Usage

```
is.direct_access_index(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the DirectAccessIndex class.

is.docker_image *Test if object is of class DockerImage*

Description

Test if object is of class DockerImage

Usage

```
is.docker_image(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the DockerImage class.

is.email_notifications *Test if object is of class JobEmailNotifications*

Description

Test if object is of class JobEmailNotifications

Usage

```
is.email_notifications(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the JobEmailNotifications class.

is.embedding_source_column *Test if object is of class EmbeddingSourceColumn*

Description

Test if object is of class EmbeddingSourceColumn

Usage

```
is.embedding_source_column(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the EmbeddingSourceColumn class.

is.embedding_vector_column *Test if object is of class EmbeddingVectorColumn*

Description

Test if object is of class EmbeddingVectorColumn

Usage

```
is.embedding_vector_column(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the EmbeddingVectorColumn class.

is.file_storage_info *Test if object is of class FileStorageInfo*

Description

Test if object is of class FileStorageInfo

Usage

```
is.file_storage_info(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the FileStorageInfo class.

is.gcp_attributes *Test if object is of class GcpAttributes*

Description

Test if object is of class GcpAttributes

Usage

```
is.gcp_attributes(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the GcpAttributes class.

is.git_source *Test if object is of class GitSource*

Description

Test if object is of class GitSource

Usage

```
is.git_source(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the GitSource class.

is.init_script_info *Test if object is of class InitScriptInfo*

Description

Test if object is of class InitScriptInfo

Usage

```
is.init_script_info(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the InitScriptInfo class.

is.job_task *Test if object is of class JobTaskSettings*

Description

Test if object is of class JobTaskSettings

Usage

```
is.job_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the JobTaskSettings class.

is.libraries *Test if object is of class Libraries*

Description

Test if object is of class Libraries

Usage

```
is.libraries(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the Libraries class.

is.library *Test if object is of class Library*

Description

Test if object is of class Library

Usage

```
is.library(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the Library class.

is.lib_cran *Test if object is of class CranLibrary*

Description

Test if object is of class CranLibrary

Usage

```
is.lib_cran(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the CranLibrary class.

is.lib_egg *Test if object is of class EggLibrary*

Description

Test if object is of class EggLibrary

Usage

```
is.lib_egg(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the EggLibrary class.

is.lib_jar *Test if object is of class JarLibrary*

Description

Test if object is of class JarLibrary

Usage

```
is.lib_jar(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the JarLibrary class.

is.lib_maven *Test if object is of class MavenLibrary*

Description

Test if object is of class MavenLibrary

Usage

```
is.lib_maven(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the MavenLibrary class.

is.lib_pypi *Test if object is of class PyPiLibrary*

Description

Test if object is of class PyPiLibrary

Usage

```
is.lib_pypi(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the PyPiLibrary class.

is.lib_whl *Test if object is of class WhlLibrary*

Description

Test if object is of class WhlLibrary

Usage

```
is.lib_whl(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the WhlLibrary class.

is.new_cluster *Test if object is of class NewCluster*

Description

Test if object is of class NewCluster

Usage

```
is.new_cluster(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the NewCluster class.

is.notebook_task *Test if object is of class NotebookTask*

Description

Test if object is of class NotebookTask

Usage

```
is.notebook_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the NotebookTask class.

is.pipeline_task *Test if object is of class PipelineTask*

Description

Test if object is of class PipelineTask

Usage

```
is.pipeline_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the PipelineTask class.

is.python_wheel_task *Test if object is of class PythonWheelTask*

Description

Test if object is of class PythonWheelTask

Usage

```
is.python_wheel_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the PythonWheelTask class.

is.s3_storage_info *Test if object is of class S3StorageInfo*

Description

Test if object is of class S3StorageInfo

Usage

```
is.s3_storage_info(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the S3StorageInfo class.

is.spark_jar_task *Test if object is of class SparkJarTask*

Description

Test if object is of class SparkJarTask

Usage

```
is.spark_jar_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the SparkJarTask class.

is.spark_python_task *Test if object is of class SparkPythonTask*

Description

Test if object is of class SparkPythonTask

Usage

```
is.spark_python_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the SparkPythonTask class.

is.spark_submit_task *Test if object is of class SparkSubmitTask*

Description

Test if object is of class SparkSubmitTask

Usage

```
is.spark_submit_task(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the SparkSubmitTask class.

is.valid_task_type *Test if object is of class JobTask*

Description

Test if object is of class JobTask

Usage

```
is.valid_task_type(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the JobTask class.

is.vector_search_index_spec *Test if object is of class VectorSearchIndexSpec*

Description

Test if object is of class VectorSearchIndexSpec

Usage

```
is.vector_search_index_spec(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the VectorSearchIndexSpec class.

job_task *Job Task*

Description

Job Task

Usage

```
job_task(
  task_key,
  description = NULL,
  depends_on = c(),
  existing_cluster_id = NULL,
  new_cluster = NULL,
  job_cluster_key = NULL,
  task,
  libraries = NULL,
  email_notifications = NULL,
  timeout_seconds = NULL,
  max_retries = 0,
  min_retry_interval_millis = 0,
  retry_on_timeout = FALSE
)
```

Arguments

task_key	A unique name for the task. This field is used to refer to this task from other tasks. This field is required and must be unique within its parent job. On db_jobs_update() or db_jobs_reset(), this field is used to reference the tasks to be updated or reset. The maximum length is 100 characters.
description	An optional description for this task. The maximum length is 4096 bytes.
depends_on	Vector of task_key's specifying the dependency graph of the task. All task_key's specified in this field must complete successfully before executing this task. This field is required when a job consists of more than one task.
existing_cluster_id	ID of an existing cluster that is used for all runs of this task.
new_cluster	Instance of new_cluster().
job_cluster_key	Task is executed reusing the cluster specified in db_jobs_create() with job_clusters parameter.
task	One of notebook_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), pipeline_task(), python_wheel_task().
libraries	Instance of libraries().
email_notifications	Instance of email_notifications.
timeout_seconds	An optional timeout applied to each run of this job task. The default behavior is to have no timeout.
max_retries	An optional maximum number of times to retry an unsuccessful run. A run is considered to be unsuccessful if it completes with the FAILED result_state or INTERNAL_ERROR life_cycle_state. The value -1 means to retry indefinitely and the value 0 means to never retry. The default behavior is to never retry.
min_retry_interval_millis	Optional minimal interval in milliseconds between the start of the failed run and the subsequent retry run. The default behavior is that unsuccessful runs are immediately retried.
retry_on_timeout	Optional policy to specify whether to retry a task when it times out. The default behavior is to not retry on timeout.

---

job_tasks *Job Tasks*

Description

Job Tasks

Usage

```
job_tasks(...)
```

Arguments

...	Multiple Instance of tasks job_task().

See Also

db_jobs_create(), db_jobs_reset(), db_jobs_update()

---

libraries *Libraries*

Description

Libraries

Usage

```
libraries(...)
```

Arguments

...	Accepts multiple instances of lib_jar(), lib_cran(), lib_maven(), lib_pypi(), lib_whl(), lib_egg().

Details

Optional list of libraries to be installed on the cluster that executes the task.

See Also

job_task(), lib_jar(), lib_cran(), lib_maven(), lib_pypi(), lib_whl(), lib_egg()

Other Task Objects: email_notifications(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_python_task(), spark_submit_task()

Other Library Objects: lib_cran(), lib_egg(), lib_jar(), lib_maven(), lib_pypi(), lib_whl()

lib_cran *Cran Library (R)*

Description

Cran Library (R)

Usage

```
lib_cran(package, repo = NULL)
```

Arguments

package	The name of the CRAN package to install.
repo	The repository where the package can be found. If not specified, the default CRAN repo is used.

See Also

libraries()

Other Library Objects: lib_egg(), lib_jar(), lib_maven(), lib_pypi(), lib_whl(), libraries()

lib_egg *Egg Library (Python)*

Description

Egg Library (Python)

Usage

```
lib_egg(egg)
```

Arguments

egg	URI of the egg to be installed. DBFS and S3 URIs are supported. For example: dbfs:/my/egg or s3://my-bucket/egg. If S3 is used, make sure the cluster has read access on the library. You may need to launch the cluster with an instance profile to access the S3 URI.

See Also

libraries()

Other Library Objects: lib_cran(), lib_jar(), lib_maven(), lib_pypi(), lib_whl(), libraries()

lib_jar *Jar Library (Scala)*

Description

Jar Library (Scala)

Usage

```
lib_jar(jar)
```

Arguments

jar	URI of the JAR to be installed. DBFS and S3 URIs are supported. For example: `dbfs:/mnt/databricks/library.jar` or `s3://my-bucket/library.jar`. If S3 is used, make sure the cluster has read access on the library. You may need to launch the cluster with an instance profile to access the S3 URI.

See Also

`libraries()`

Other Library Objects: `lib_cran()`, `lib_egg()`, `lib_maven()`, `lib_pypi()`, `lib_whl()`, `libraries()`

lib_maven *Maven Library (Scala)*

Description

Maven Library (Scala)

Usage

```
lib_maven(coordinates, repo = NULL, exclusions = NULL)
```

Arguments

coordinates	Gradle-style Maven coordinates. For example: `org.jsoup:jsoup:1.7.2`.
repo	Maven repo to install the Maven package from. If omitted, both Maven Central Repository and Spark Packages are searched.
exclusions	List of dependencies to exclude. For example: `list("slf4j:slf4j", "*:hadoop-client")`. Maven dependency exclusions.

See Also

`libraries()`

Other Library Objects: `lib_cran()`, `lib_egg()`, `lib_jar()`, `lib_pypi()`, `lib_whl()`, `libraries()`

lib_pypi *PyPi Library (Python)*

Description

PyPi Library (Python)

Usage

```
lib_pypi(package, repo = NULL)
```

Arguments

package	The name of the PyPI package to install. An optional exact version specification is also supported. Examples: `simplejson` and `simplejson==3.8.0`.
repo	The repository where the package can be found. If not specified, the default pip index is used.

See Also

`libraries()`

Other Library Objects: `lib_cran()`, `lib_egg()`, `lib_jar()`, `lib_maven()`, `lib_whl()`, `libraries()`

lib_whl *Wheel Library (Python)*

Description

Wheel Library (Python)

Usage

```
lib_whl(whl)
```

Arguments

whl	URI of the wheel or zipped wheels to be installed. DBFS and S3 URIs are supported. For example: `dbfs:/my/whl` or `s3://my-bucket/whl`. If S3 is used, make sure the cluster has read access on the library. You may need to launch the cluster with an instance profile to access the S3 URI. Also the wheel file name needs to use the correct convention. If zipped wheels are to be installed, the file name suffix should be `.wheelhouse.zip`.

See Also

`libraries()`

Other Library Objects: `lib_cran()`, `lib_egg()`, `lib_jar()`, `lib_maven()`, `lib_pypi()`, `libraries()`

new_cluster *New Cluster*

Description

New Cluster

Usage

```
new_cluster(
  num_workers,
  spark_version,
  node_type_id,
  driver_node_type_id = NULL,
  autoscale = NULL,
  cloud_attrs = NULL,
  spark_conf = NULL,
  spark_env_vars = NULL,
  custom_tags = NULL,
  ssh_public_keys = NULL,
  log_conf = NULL,
  init_scripts = NULL,
  enable_elastic_disk = TRUE,
  driver_instance_pool_id = NULL,
  instance_pool_id = NULL
)
```

Arguments

num_workers	Number of worker nodes that this cluster should have. A cluster has one Spark driver and num_workers executors for a total of num_workers + 1 Spark nodes.
spark_version	The runtime version of the cluster. You can retrieve a list of available runtime versions by using db_cluster_runtime_versions().
node_type_id	The node type for the worker nodes. db_cluster_list_node_types() can be used to see available node types.
driver_node_type_id	The node type of the Spark driver. This field is optional; if unset, the driver node type will be set as the same value as node_type_id defined above. db_cluster_list_node_types() can be used to see available node types.
autoscale	Instance of cluster_autoscale().
cloud_attrs	Attributes related to clusters running on specific cloud provider. Defaults to aws_attributes(). Must be one of aws_attributes(), azure_attributes(), gcp_attributes().
spark_conf	Named list. An object containing a set of optional, user-specified Spark configuration key-value pairs. You can also pass in a string of extra JVM options to the driver and the executors via spark.driver.extraJavaOptions and

spark.executor.extraJavaOptions respectively. E.g. list("spark.speculation" = true, "spark.streaming.ui.retainedBatches" = 5).

spark_env_vars Named list. User-specified environment variable key-value pairs. In order to specify an additional set of SPARK_DAEMON_JAVA_OPTS, we recommend appending them to $SPARK_DAEMON_JAVA_OPTS as shown in the following example. This ensures that all default Databricks managed environmental variables are included as well. E.g. {"SPARK_DAEMON_JAVA_OPTS": "$SPARK_DAEMON_JAVA_OPTS -Dspark.shuffle.service.enabled=true"}

custom_tags Named list. An object containing a set of tags for cluster resources. Databricks tags all cluster resources with these tags in addition to default_tags. Databricks allows at most 45 custom tags.

ssh_public_keys
List. SSH public key contents that will be added to each Spark node in this cluster. The corresponding private keys can be used to login with the user name ubuntu on port 2200. Up to 10 keys can be specified.

log_conf Instance of cluster_log_conf().

init_scripts Instance of init_script_info().

enable_elastic_disk
When enabled, this cluster will dynamically acquire additional disk space when its Spark workers are running low on disk space.

driver_instance_pool_id
ID of the instance pool to use for the driver node. You must also specify instance_pool_id. Optional.

instance_pool_id
ID of the instance pool to use for cluster nodes. If driver_instance_pool_id is present, instance_pool_id is used for worker nodes only. Otherwise, it is used for both the driver and worker nodes. Optional.

See Also

job_task()

Other Task Objects: email_notifications(), libraries(), notebook_task(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_python_task(), spark_submit_task()

---

notebook_task *Notebook Task*

Description

Notebook Task

Usage

```
notebook_task(notebook_path, base_parameters = list())
```

Arguments

notebook_path	The absolute path of the notebook to be run in the Databricks workspace. This path must begin with a slash.
base_parameters	Named list of base parameters to be used for each run of this job.

Details

If the run is initiated by a call to db_jobs_run_now() with parameters specified, the two parameters maps are merged. If the same key is specified in base_parameters and in run-now, the value from run-now is used.

Use Task parameter variables to set parameters containing information about job runs.

If the notebook takes a parameter that is not specified in the job's base_parameters or the run-now override parameters, the default value from the notebook is used.

Retrieve these parameters in a notebook using dbutils.widgets.get.

See Also

Other Task Objects: email_notifications(), libraries(), new_cluster(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_python_task(), spark_submit_task()

---

open_workspace *Connect to Databricks Workspace*

Description

Connect to Databricks Workspace

Usage

```
open_workspace(host = db_host(), token = db_token(), name = NULL)
```

Arguments

host	Databricks workspace URL, defaults to calling db_host().
token	Databricks workspace token, defaults to calling db_token().
name	Desired name to assign the connection

Examples

```
## Not run:
open_workspace(host = db_host(), token = db_token, name = "MyWorkspace")

## End(Not run)
```

pipeline_task *Pipeline Task*

Description

Pipeline Task

Usage

```
pipeline_task(pipeline_id)
```

Arguments

pipeline_id	The full name of the pipeline task to execute.

See Also

Other Task Objects: email_notifications(), libraries(), new_cluster(), notebook_task(), python_wheel_task(), spark_jar_task(), spark_python_task(), spark_submit_task()

python_wheel_task *Python Wheel Task*

Description

Python Wheel Task

Usage

```
python_wheel_task(package_name, entry_point = NULL, parameters = list())
```

Arguments

package_name	Name of the package to execute.
entry_point	Named entry point to use, if it does not exist in the metadata of the package it executes the function from the package directly using $packageName.$entryPoint().
parameters	Command-line parameters passed to python wheel task.

See Also

Other Task Objects: email_notifications(), libraries(), new_cluster(), notebook_task(), pipeline_task(), spark_jar_task(), spark_python_task(), spark_submit_task()

remove_lib_path *Remove Library Path*

Description

Remove Library Path

Usage

```
remove_lib_path(path, version = FALSE)
```

Arguments

path	Directory to remove from .libPaths().
version	If TRUE will add the R version string to the end of path before removal.

See Also

base::.libPaths(), remove_lib_path()

s3_storage_info *S3 Storage Info*

Description

S3 Storage Info

Usage

```
s3_storage_info(
  destination,
  region = NULL,
  endpoint = NULL,
  enable_encryption = FALSE,
  encryption_type = c("sse-s3", "sse-kms"),
  kms_key = NULL,
  canned_acl = NULL
)
```

Arguments

destination	S3 destination. For example: `s3://my-bucket/some-prefix`. You must configure the cluster with an instance profile and the instance profile must have write access to the destination. **You cannot use AWS keys**.
region	S3 region. For example: `us-west-2`. Either region or endpoint must be set. If both are set, endpoint is used.
endpoint	S3 endpoint. For example: `https://s3-us-west-2.amazonaws.com`. Either region or endpoint must be set. If both are set, endpoint is used.
enable_encryption	Boolean (Default: `FALSE`). If `TRUE` Enable server side encryption.
encryption_type	Encryption type, it could be `sse-s3` or `sse-kms`. It is used only when encryption is enabled and the default type is `sse-s3`.
kms_key	KMS key used if encryption is enabled and encryption type is set to `sse-kms`.
canned_acl	Set canned access control list. For example: `bucket-owner-full-control`. If `canned_acl` is set, the cluster instance profile must have `s3:PutObjectAcl` permission on the destination bucket and prefix. The full list of possible canned ACLs can be found in docs. By default only the object owner gets full control. If you are using cross account role for writing data, you may want to set `bucket-owner-full-control` to make bucket owner able to read the logs.

See Also

`cluster_log_conf()`, `init_script_info()`

Other Cluster Log Configuration Objects: `cluster_log_conf()`, `dbfs_storage_info()`

Other Init Script Info Objects: `dbfs_storage_info()`, `file_storage_info()`

---

spark_jar_task *Spark Jar Task*

Description

Spark Jar Task

Usage

```
spark_jar_task(main_class_name, parameters = list())
```

Arguments

main_class_name	The full name of the class containing the main method to be executed. This class must be contained in a JAR provided as a library. The code must use `SparkContext.getOrCreate` to obtain a Spark context; otherwise, runs of the job fail.
parameters	Named list. Parameters passed to the main method. Use Task parameter variables to set parameters containing information about job runs.

See Also

Other Task Objects: email_notifications(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), spark_python_task(), spark_submit_task()

`spark_python_task` *Spark Python Task*

Description

Spark Python Task

Usage

```
spark_python_task(python_file, parameters = list())
```

Arguments

python_file	The URI of the Python file to be executed. DBFS and S3 paths are supported.
parameters	List. Command line parameters passed to the Python file. Use Task parameter variables to set parameters containing information about job runs.

See Also

Other Task Objects: email_notifications(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_submit_task()

`spark_submit_task` *Spark Submit Task*

Description

Spark Submit Task

Usage

```
spark_submit_task(parameters)
```

Arguments

parameters	List. Command-line parameters passed to spark submit. Use Task parameter variables to set parameters containing information about job runs.

See Also

Other Task Objects: email_notifications(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_python_task()

`wait_for_lib_installs` *Wait for Libraries to Install on Databricks Cluster*

Description

Wait for Libraries to Install on Databricks Cluster

Usage

```
wait_for_lib_installs(
  cluster_id,
  polling_interval = 5,
  allow_failures = FALSE,
  host = db_host(),
  token = db_token()
)
```

Arguments

`cluster_id`	Unique identifier of a Databricks cluster.
`polling_interval`	Number of seconds to wait between status checks
`allow_failures`	If `FALSE` (default) will error if any libraries status is `FAILED`. When `TRUE` any `FAILED` installs will be presented as a warning.
`host`	Databricks workspace URL, defaults to calling `db_host()`.
`token`	Databricks workspace token, defaults to calling `db_token()`.

Details

Library installs on Databricks clusters are asynchronous, this function allows you to repeatedly check installation status of each library.

Can be used to block any scripts until required dependencies are installed.

See Also

`db_libs_cluster_status()`

Index