# Package 'brickster'

July 24, 2026

**Title** R Toolkit for 'Databricks'

**Version** 0.2.14

**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)

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

**Depends** R (>= 4.1.0)

**Imports** base64enc, cli, curl, DBI, dbplyr, dplyr, fs, glue, httr2 (>= 1.3.0), ini, jsonlite, lifecycle, methods, nanoarrow, processx, purrr, R6, rlang, tibble, utils

**Suggests** arrow, grid, htmltools, huxtable, knitr, magick, rmarkdown, testthat (>= 3.3.0), withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**NeedsCompilation** no

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

**Maintainer** Zac Davies <zac@databricks.com>

**Repository** CRAN

**Date/Publication** 2026-07-24 07:00:02 UTC

# Contents

access_control_request . . . 8
access_control_req_group . . . 9
access_control_req_user . . . 9
add_lib_path . . . 10
aws_attributes . . . 11
azure_attributes . . . 13
close_workspace . . . 14
cluster_autoscale . . . 14
cluster_log_conf . . . 15
condition_task . . . 15
copy_to.DatabricksConnection . . . 16
cron_schedule . . . 17
databricks-dbi . . . 17
databricks-dbplyr . . . 18
DatabricksConnection-class . . . 18
DatabricksDriver-class . . . 18
DatabricksResult-class . . . 18
DatabricksSQL . . . 19
dbAppendTable,DatabricksConnection,character,data.frame-method . . . 19
dbAppendTable,DatabricksConnection,Id,data.frame-method . . . 20
dbBegin,DatabricksConnection-method . . . 20
dbClearResult,DatabricksResult-method . . . 21
dbColumnInfo,DatabricksResult-method . . . 21
dbCommit,DatabricksConnection-method . . . 22
dbConnect,DatabricksDriver-method . . . 22
dbCreateTable,DatabricksConnection,AsIs-method . . . 23
dbCreateTable,DatabricksConnection,character-method . . . 24
dbCreateTable,DatabricksConnection,Id-method . . . 25
dbDataType,DatabricksConnection-method . . . 25
dbDisconnect,DatabricksConnection-method . . . 26
dbExecute,DatabricksConnection,character-method . . . 26
dbExistsTable,DatabricksConnection,AsIs-method . . . 27
dbExistsTable,DatabricksConnection,character-method . . . 27
dbExistsTable,DatabricksConnection,Id-method . . . 28
dbFetch,DatabricksResult-method . . . 28
dbfs_storage_info . . . 29
dbGetInfo,DatabricksConnection-method . . . 29
dbGetQuery,DatabricksConnection,character-method . . . 30
dbGetRowCount,DatabricksResult-method . . . 30
dbGetRowsAffected,DatabricksResult-method . . . 31
dbGetStatement,DatabricksResult-method . . . 31
dbHasCompleted,DatabricksResult-method . . . 32
dbIsValid,DatabricksConnection-method . . . 32
dbListFields,DatabricksConnection,AsIs-method . . . 33
dbListFields,DatabricksConnection,character-method . . . 33
dbListTables,DatabricksConnection-method . . . 34

dbplyr_edition.DatabricksConnection . . . 34
dbQuoteIdentifier,DatabricksConnection,character-method . . . 35
dbQuoteIdentifier,DatabricksConnection,Id-method . . . 35
dbQuoteIdentifier,DatabricksConnection,SQL-method . . . 36
dbReadTable,DatabricksConnection,AsIs-method . . . 36
dbReadTable,DatabricksConnection,character-method . . . 37
dbReadTable,DatabricksConnection,Id-method . . . 37
dbRemoveTable,DatabricksConnection,AsIs-method . . . 38
dbRemoveTable,DatabricksConnection,character-method . . . 38
dbRemoveTable,DatabricksConnection,Id-method . . . 39
dbRollback,DatabricksConnection-method . . . 39
dbSendQuery,DatabricksConnection,character-method . . . 40
dbSendStatement,DatabricksConnection,character-method . . . 40
dbWriteTable,DatabricksConnection,AsIs,data.frame-method . . . 41
dbWriteTable,DatabricksConnection,character,data.frame-method . . . 42
dbWriteTable,DatabricksConnection,Id,data.frame-method . . . 43
db_cluster_action . . . 44
db_cluster_create . . . 45
db_cluster_delete . . . 48
db_cluster_edit . . . 49
db_cluster_events . . . 52
db_cluster_get . . . 53
db_cluster_list . . . 54
db_cluster_list_node_types . . . 55
db_cluster_list_zones . . . 56
db_cluster_perm_delete . . . 57
db_cluster_pin . . . 58
db_cluster_resize . . . 59
db_cluster_restart . . . 60
db_cluster_runtime_versions . . . 61
db_cluster_start . . . 62
db_cluster_terminate . . . 63
db_cluster_unpin . . . 64
db_collect.DatabricksConnection . . . 65
db_context_command_cancel . . . 65
db_context_command_run . . . 66
db_context_command_run_and_wait . . . 67
db_context_command_status . . . 68
db_context_create . . . 69
db_context_destroy . . . 70
db_context_manager . . . 71
db_context_status . . . 72
db_current_cloud . . . 73
db_current_user . . . 73
db_current_workspace_id . . . 74
db_host . . . 75
db_jobs_create . . . 76
db_jobs_delete . . . 77

db_jobs_get . . . 78
db_jobs_list . . . 79
db_jobs_repair_run . . . 80
db_jobs_reset . . . 81
db_jobs_runs_cancel . . . 83
db_jobs_runs_delete . . . 84
db_jobs_runs_export . . . 84
db_jobs_runs_get . . . 85
db_jobs_runs_get_output . . . 86
db_jobs_runs_list . . . 87
db_jobs_runs_submit . . . 88
db_jobs_run_now . . . 89
db_jobs_update . . . 90
db_lakebase_creds_generate . . . 92
db_lakebase_get . . . 93
db_lakebase_get_by_uid . . . 94
db_lakebase_list . . . 94
db_libs_all_cluster_statuses . . . 96
db_libs_cluster_status . . . 97
db_libs_install . . . 97
db_libs_uninstall . . . 99
db_mlflow_model_approve_transition_req . . . 100
db_mlflow_model_delete_transition_req . . . 101
db_mlflow_model_open_transition_reqs . . . 102
db_mlflow_model_reject_transition_req . . . 103
db_mlflow_model_transition_req . . . 104
db_mlflow_model_transition_stage . . . 105
db_mlflow_model_version_comment . . . 106
db_mlflow_model_version_comment_delete . . . 107
db_mlflow_model_version_comment_edit . . . 108
db_mlflow_registered_model_details . . . 109
db_perform_request . . . 110
db_perform_response . . . 110
db_query_create . . . 111
db_query_delete . . . 112
db_query_get . . . 113
db_query_list . . . 114
db_query_update . . . 115
db_read_netrc . . . 116
db_repl . . . 117
db_repo_create . . . 117
db_repo_delete . . . 118
db_repo_get . . . 119
db_repo_get_all . . . 120
db_repo_update . . . 121
db_request . . . 122
db_request_json . . . 122
db_req_error_body . . . 123

db_secrets_delete . . . 123
db_secrets_list . . . 124
db_secrets_put . . . 125
db_secrets_scope_acl_delete . . . 127
db_secrets_scope_acl_get . . . 128
db_secrets_scope_acl_list . . . 129
db_secrets_scope_acl_put . . . 130
db_secrets_scope_create . . . 131
db_secrets_scope_delete . . . 132
db_secrets_scope_list_all . . . 133
db_sql_exec_cancel . . . 134
db_sql_exec_poll_for_success . . . 135
db_sql_exec_query . . . 135
db_sql_exec_result . . . 138
db_sql_exec_status . . . 139
db_sql_global_warehouse_get . . . 140
db_sql_query . . . 140
db_sql_query_history . . . 142
db_sql_warehouse_create . . . 143
db_sql_warehouse_delete . . . 145
db_sql_warehouse_edit . . . 146
db_sql_warehouse_get . . . 147
db_sql_warehouse_list . . . 148
db_sql_warehouse_start . . . 149
db_sql_warehouse_stop . . . 150
db_token . . . 151
db_uc_catalogs_get . . . 152
db_uc_catalogs_list . . . 152
db_uc_schemas_get . . . 153
db_uc_schemas_list . . . 154
db_uc_tables_delete . . . 155
db_uc_tables_exists . . . 156
db_uc_tables_get . . . 157
db_uc_tables_list . . . 158
db_uc_tables_summaries . . . 159
db_uc_volumes_create . . . 160
db_uc_volumes_delete . . . 161
db_uc_volumes_get . . . 162
db_uc_volumes_list . . . 163
db_uc_volumes_update . . . 164
db_volume_delete . . . 165
db_volume_dir_create . . . 166
db_volume_dir_delete . . . 166
db_volume_dir_exists . . . 167
db_volume_download_dir . . . 168
db_volume_file_exists . . . 169
db_volume_list . . . 170
db_volume_read . . . 170

db_volume_upload_dir . . . 171
db_volume_write . . . 172
db_vs_endpoints_create . . . 173
db_vs_endpoints_delete . . . 174
db_vs_endpoints_get . . . 175
db_vs_endpoints_list . . . 176
db_vs_indexes_create . . . 177
db_vs_indexes_delete . . . 178
db_vs_indexes_delete_data . . . 179
db_vs_indexes_get . . . 180
db_vs_indexes_list . . . 181
db_vs_indexes_query . . . 182
db_vs_indexes_query_next_page . . . 184
db_vs_indexes_scan . . . 185
db_vs_indexes_sync . . . 186
db_vs_indexes_upsert_data . . . 187
db_workspace_delete . . . 188
db_workspace_export . . . 189
db_workspace_get_status . . . 190
db_workspace_import . . . 191
db_workspace_list . . . 192
db_workspace_mkdirs . . . 193
db_wsid . . . 194
delta_sync_index_spec . . . 195
direct_access_index_spec . . . 196
docker_image . . . 197
email_notifications . . . 198
embedding_source_column . . . 199
embedding_vector_column . . . 199
file_storage_info . . . 200
for_each_task . . . 200
gcp_attributes . . . 201
get_and_start_cluster . . . 202
get_and_start_warehouse . . . 203
get_latest_dbr . . . 204
git_source . . . 205
init_script_info . . . 205
in_databricks_nb . . . 206
is.access_control_request . . . 206
is.access_control_req_group . . . 207
is.access_control_req_user . . . 207
is.aws_attributes . . . 208
is.azure_attributes . . . 208
is.cluster_autoscale . . . 209
is.cluster_log_conf . . . 209
is.condition_task . . . 210
is.cron_schedule . . . 210
is.dbfs_storage_info . . . 211

is.delta_sync_index . . . 211
is.direct_access_index . . . 212
is.docker_image . . . 212
is.email_notifications . . . 213
is.embedding_source_column . . . 213
is.embedding_vector_column . . . 214
is.file_storage_info . . . 214
is.for_each_task . . . 215
is.gcp_attributes . . . 215
is.git_source . . . 216
is.init_script_info . . . 216
is.job_task . . . 217
is.libraries . . . 217
is.library . . . 218
is.lib_cran . . . 218
is.lib_egg . . . 219
is.lib_jar . . . 219
is.lib_maven . . . 220
is.lib_pypi . . . 220
is.lib_whl . . . 221
is.new_cluster . . . 221
is.notebook_task . . . 222
is.pipeline_task . . . 222
is.python_wheel_task . . . 223
is.run_job_task . . . 223
is.s3_storage_info . . . 224
is.spark_jar_task . . . 224
is.spark_python_task . . . 225
is.spark_submit_task . . . 225
is.sql_file_task . . . 226
is.sql_query_task . . . 226
is.valid_task_type . . . 227
is.vector_search_index_spec . . . 227
job_task . . . 228
job_tasks . . . 229
libraries . . . 230
lib_cran . . . 230
lib_egg . . . 231
lib_jar . . . 231
lib_maven . . . 232
lib_pypi . . . 232
lib_whl . . . 233
new_cluster . . . 233
notebook_task . . . 235
open_workspace . . . 236
pipeline_task . . . 237
python_wheel_task . . . 237
remove_lib_path . . . 238

run_job_task . . . 238
s3_storage_info . . . 239
show,DatabricksConnection-method . . . 240
show,DatabricksDriver-method . . . 240
show,DatabricksResult-method . . . 241
spark_jar_task . . . 241
spark_python_task . . . 242
spark_submit_task . . . 242
sql_file_task . . . 243
sql_query_fields.DatabricksConnection . . . 243
sql_query_save.DatabricksConnection . . . 244
sql_query_task . . . 245
sql_table_analyze.DatabricksConnection . . . 245
wait_for_lib_installs . . . 246

**Index** **247**

---

`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

.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()

## condition_task *Condition Task*

### Description

Condition Task

### Usage

```
condition_task(
  left,
  right,
  op = c("EQUAL_TO", "GREATER_THAN", "GREATER_THAN_OR_EQUAL", "LESS_THAN",
    "LESS_THAN_OR_EQUAL", "NOT_EQUAL")
)
```

### Arguments

| | |
|---|---|
| left | Left operand of the condition task. Either a string value or a job state or parameter reference. |
| right | Right operand of the condition task. Either a string value or a job state or parameter reference. |
| op | Operator, one of "EQUAL_TO", "GREATER_THAN", "GREATER_THAN_OR_EQUAL", "LESS_THAN", "LESS_THAN_OR_EQUAL", "NOT_EQUAL" |

## Details

The task evaluates a condition that can be used to control the execution of other tasks when the condition_task field is present. The condition task does not require a cluster to execute and does not support retries or notifications.

## See Also

Other Task Objects: email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_task()

---

# copy_to.DatabricksConnection

*Copy data frame to Databricks as table or view*

---

## Description

Copy data frame to Databricks as table or view

## Usage

```
## S3 method for class 'DatabricksConnection'
copy_to(
  dest,
  df,
  name = deparse(substitute(df)),
  overwrite = FALSE,
  temporary = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| dest | A DatabricksConnection object |
| df | Data frame to copy |
| name | Name for the table/view |
| overwrite | Whether to overwrite existing table/view |
| temporary | Whether to create as temporary view (default: TRUE, but NOT SUPPORTED - will error) |
| ... | Additional arguments passed to dbWriteTable |

## Details

Note: temporary=TRUE will result in an error as temporary tables are not supported with the SQL Statement Execution API. Use temporary=FALSE to create regular tables.

### Value

dbplyr table reference

---

`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()`

---

`databricks-dbi` *DBI Interface for Databricks SQL Warehouses*

---

### Description

This file implements a standard DBI interface for Databricks SQL warehouses, built on top of the existing `db_sql_query()` infrastructure.

## databricks-dbplyr *dbplyr Backend for Databricks SQL*

### Description

This file implements dbplyr backend support for Databricks SQL warehouses, enabling dplyr syntax to be translated to Databricks SQL.

## DatabricksConnection-class *DBI Connection for Databricks*

### Description

DBI Connection for Databricks

## DatabricksDriver-class *DBI Driver for Databricks*

### Description

DBI Driver for Databricks

## DatabricksResult-class *DBI Result for Databricks*

### Description

DBI Result for Databricks

DatabricksSQL *Create Databricks SQL Driver*

## Description

Create Databricks SQL Driver

## Usage

```
DatabricksSQL()
```

## Value

A DatabricksDriver object

## Examples

```
## Not run:
drv <- DatabricksSQL()
con <- dbConnect(drv, warehouse_id = "your_warehouse_id")

## End(Not run)
```

---

dbAppendTable,DatabricksConnection,character,data.frame-method
*Append rows to an existing Databricks table*

## Description

Append rows to an existing Databricks table

## Usage

```
## S4 method for signature 'DatabricksConnection,character,data.frame'
dbAppendTable(conn, name, value, ..., row.names = FALSE)
```

## Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name (character, Id, or SQL) |
| value | Data frame to append |
| ... | Additional arguments |
| row.names | If TRUE, preserve row names as a column |

## Value

TRUE invisibly on success

---

`dbAppendTable,DatabricksConnection,Id,data.frame-method`
*Append rows to an existing Databricks table (Id method)*

---

## Description

Append rows to an existing Databricks table (Id method)

## Usage

```
## S4 method for signature 'DatabricksConnection,Id,data.frame'
dbAppendTable(conn, name, value, ..., row.names = FALSE)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `name` | Table name as Id object |
| `value` | Data frame to append |
| `...` | Additional arguments |
| `row.names` | If `TRUE`, preserve row names as a column |

## Value

`TRUE` invisibly on success

---

`dbBegin,DatabricksConnection-method`
*Begin transaction (not supported)*

---

## Description

Begin transaction (not supported)

## Usage

```
## S4 method for signature 'DatabricksConnection'
dbBegin(conn, ...)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `...` | Additional arguments (ignored) |

## Value

Always throws an error (transactions not supported)

## dbClearResult,DatabricksResult-method *Clear result set*

### Description

Clear result set

### Usage

```
## S4 method for signature 'DatabricksResult'
dbClearResult(res, ...)
```

### Arguments

| | |
|---|---|
| res | A DatabricksResult object |
| ... | Additional arguments (ignored) |

### Value

TRUE (invisibly)

## dbColumnInfo,DatabricksResult-method *Get column information from result*

### Description

Get column information from result

### Usage

```
## S4 method for signature 'DatabricksResult'
dbColumnInfo(res, ...)
```

### Arguments

| | |
|---|---|
| res | A DatabricksResult object |
| ... | Additional arguments (ignored) |

### Value

A data.frame with column names and types

---

`dbCommit,DatabricksConnection-method`
*Commit transaction (not supported)*

---

## Description

Commit transaction (not supported)

## Usage

```
## S4 method for signature 'DatabricksConnection'
dbCommit(conn, ...)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `...` | Additional arguments (ignored) |

## Value

Always throws an error (transactions not supported)

---

`dbConnect,DatabricksDriver-method`
*Connect to Databricks SQL Warehouse*

---

## Description

Connect to Databricks SQL Warehouse

## Usage

```
## S4 method for signature 'DatabricksDriver'
dbConnect(
  drv,
  warehouse_id = NULL,
  http_path = NULL,
  catalog = NULL,
  schema = NULL,
  staging_volume = NULL,
  disposition = c("EXTERNAL_LINKS", "INLINE"),
  max_active_connections = 30,
  fetch_timeout = 300,
  show_progress = TRUE,
  token = db_token(),
```

```
  host = db_host(),
  ...
)
```

## Arguments

| | |
|---|---|
| drv | A DatabricksDriver object |
| warehouse_id | Optional ID of the SQL warehouse to connect to |
| http_path | Optional HTTP path for the SQL warehouse; if provided, the warehouse ID is extracted from this path |
| catalog | Optional catalog name to use as default |
| schema | Optional schema name to use as default |
| staging_volume | Optional volume path for large dataset staging |
| disposition | Query disposition mode to use by default for DBI query results. Use "EXTERNAL_LINKS" for large results or "INLINE" for smaller results that must avoid direct cloud-storage result downloads. |
| max_active_connections | Maximum number of concurrent download connections when fetching query results (default: 30) |
| fetch_timeout | Timeout in seconds for downloading each result chunk (default: 300) |
| show_progress | If TRUE, show progress updates by default for DBI queries, dbplyr collection, and table writes (default: TRUE) |
| token | Authentication token (defaults to db_token()) |
| host | Databricks workspace host (defaults to db_host()) |
| ... | Additional arguments (ignored) |

## Details

Provide either warehouse_id or http_path. When http_path is supplied, the warehouse ID is extracted from the /warehouses/<id> segment.

## Value

A DatabricksConnection object

---

# dbCreateTable,DatabricksConnection,AsIs-method

*Create an empty Databricks table (AsIs method)*

---

## Description

Create an empty Databricks table (AsIs method)

### Usage

```
## S4 method for signature 'DatabricksConnection,AsIs'
dbCreateTable(conn, name, fields, ..., row.names = NULL, temporary = FALSE)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as AsIs object (from I()) |
| fields | Either a named character vector of types or a data frame |
| ... | Additional arguments (ignored) |
| row.names | Ignored (included for DBI compatibility) |
| temporary | If TRUE, create temporary table (NOT SUPPORTED - will error) |

### Value

TRUE invisibly on success

---

## dbCreateTable,DatabricksConnection,character-method
*Create an empty Databricks table*

---

### Description

Create an empty Databricks table

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbCreateTable(conn, name, fields, ..., row.names = NULL, temporary = FALSE)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name to create |
| fields | Either a named character vector of types or a data frame |
| ... | Additional arguments (ignored) |
| row.names | Ignored (included for DBI compatibility) |
| temporary | If TRUE, create temporary table (NOT SUPPORTED - will error) |

### Value

TRUE invisibly on success

`dbCreateTable,DatabricksConnection,Id-method`
*Create an empty Databricks table (Id method)*

## Description

Create an empty Databricks table (Id method)

## Usage

```
## S4 method for signature 'DatabricksConnection,Id'
dbCreateTable(conn, name, fields, ..., row.names = NULL, temporary = FALSE)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `name` | Table name as Id object |
| `fields` | Either a named character vector of types or a data frame |
| `...` | Additional arguments (ignored) |
| `row.names` | Ignored (included for DBI compatibility) |
| `temporary` | If `TRUE`, create temporary table (NOT SUPPORTED - will error) |

## Value

`TRUE` invisibly on success

---

`dbDataType,DatabricksConnection-method`
*Map R data types to Databricks SQL types*

## Description

Map R data types to Databricks SQL types

## Usage

```
## S4 method for signature 'DatabricksConnection'
dbDataType(dbObj, obj, ...)
```

## Arguments

| | |
|---|---|
| `dbObj` | A DatabricksConnection object |
| `obj` | R object(s) to get SQL types for |
| `...` | Additional arguments (ignored) |

### Value

Character vector of SQL type names

---

## dbDisconnect,DatabricksConnection-method

*Disconnect from Databricks*

---

### Description

Disconnect from Databricks

### Usage

```
## S4 method for signature 'DatabricksConnection'
dbDisconnect(conn, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| ... | Additional arguments (ignored) |

### Value

TRUE (invisibly)

---

## dbExecute,DatabricksConnection,character-method

*Execute statement on Databricks*

---

### Description

Execute statement on Databricks

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbExecute(conn, statement, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| statement | SQL statement |
| ... | Additional arguments (ignored) |

### Value

Number of rows in result set (from metadata, without loading data)

---

`dbExistsTable,DatabricksConnection,AsIs-method`
*Check if table exists (AsIs method)*

---

## Description

Check if table exists (AsIs method)

## Usage

```
## S4 method for signature 'DatabricksConnection,AsIs'
dbExistsTable(conn, name, ...)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `name` | Table name as AsIs object (from I()) |
| `...` | Additional arguments (ignored) |

## Value

`TRUE` if table exists, `FALSE` otherwise

---

`dbExistsTable,DatabricksConnection,character-method`
*Check if table exists in Databricks*

---

## Description

Check if table exists in Databricks

## Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbExistsTable(conn, name, ...)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `name` | Table name to check |
| `...` | Additional arguments (ignored) |

## Value

`TRUE` if table exists, `FALSE` otherwise

---

`dbExistsTable,DatabricksConnection,Id-method`
*Check if table exists (Id method)*

---

## Description

Check if table exists (Id method)

## Usage

```
## S4 method for signature 'DatabricksConnection,Id'
dbExistsTable(conn, name, ...)
```

## Arguments

| | |
|---|---|
| `conn` | A DatabricksConnection object |
| `name` | Table name as Id object |
| `...` | Additional arguments (ignored) |

## Value

`TRUE` if table exists, `FALSE` otherwise

---

`dbFetch,DatabricksResult-method`
*Fetch results from Databricks query*

---

## Description

Fetch results from Databricks query

## Usage

```
## S4 method for signature 'DatabricksResult'
dbFetch(res, n = -1, show_progress = res@connection@show_progress, ...)
```

## Arguments

| | |
|---|---|
| `res` | A DatabricksResult object |
| `n` | Maximum number of rows to fetch (-1 for all rows) |
| `show_progress` | If `TRUE`, show progress updates during result fetching. Defaults to the connection's `show_progress` setting. |
| `...` | Additional arguments (ignored) |

## Value

A data.frame with query results

## 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()

## dbGetInfo,DatabricksConnection-method *Get connection information*

### Description

Get connection information

### Usage

```
## S4 method for signature 'DatabricksConnection'
dbGetInfo(dbObj, ...)
```

### Arguments

| | |
|---|---|
| dbObj | A DatabricksConnection object |
| ... | Additional arguments (ignored) |

### Value

A list with connection details

## dbGetQuery,DatabricksConnection,character-method

*Execute SQL query and return results*

### Description

Execute SQL query and return results

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbGetQuery(
  conn,
  statement,
  disposition = conn@disposition,
  show_progress = conn@show_progress,
  ...
)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| statement | SQL statement to execute |
| disposition | Query disposition mode. Defaults to the connection's disposition setting. |
| show_progress | If TRUE, show progress updates during query execution. Defaults to the connection's show_progress setting. |
| ... | Additional arguments passed to underlying query execution |

### Value

A data.frame with query results

## dbGetRowCount,DatabricksResult-method

*Get number of rows fetched*

### Description

Get number of rows fetched

### Usage

```
## S4 method for signature 'DatabricksResult'
dbGetRowCount(res, ...)
```

### Arguments

| | |
|---|---|
| res | A DatabricksResult object |
| ... | Additional arguments (ignored) |

### Value

Number of rows fetched so far

---

## dbGetRowsAffected,DatabricksResult-method

*Get number of rows affected (not applicable for SELECT)*

---

### Description

Get number of rows affected (not applicable for SELECT)

### Usage

```
## S4 method for signature 'DatabricksResult'
dbGetRowsAffected(res, ...)
```

### Arguments

| | |
|---|---|
| res | A DatabricksResult object |
| ... | Additional arguments (ignored) |

### Value

-1 (not applicable for SELECT queries)

---

## dbGetStatement,DatabricksResult-method

*Get SQL statement from result*

---

### Description

Get SQL statement from result

### Usage

```
## S4 method for signature 'DatabricksResult'
dbGetStatement(res, ...)
```

### Arguments

| | |
|---|---|
| res | A DatabricksResult object |
| ... | Additional arguments (ignored) |

### Value

The SQL statement as character

---

## dbHasCompleted,DatabricksResult-method

*Check if query has completed*

---

### Description

Check if query has completed

### Usage

```
## S4 method for signature 'DatabricksResult'
dbHasCompleted(res, ...)
```

### Arguments

| | |
|---|---|
| res | A DatabricksResult object |
| ... | Additional arguments (ignored) |

### Value

TRUE if query is complete, FALSE otherwise

---

## dbIsValid,DatabricksConnection-method

*Check if connection is valid*

---

### Description

Check if connection is valid

### Usage

```
## S4 method for signature 'DatabricksConnection'
dbIsValid(dbObj, ...)
```

### Arguments

| | |
|---|---|
| dbObj | A DatabricksConnection object |
| ... | Additional arguments (ignored) |

### Value

TRUE if connection is valid, FALSE otherwise

---

## dbListFields,DatabricksConnection,AsIs-method

*List column names of a Databricks table (AsIs method)*

---

### Description

List column names of a Databricks table (AsIs method)

### Usage

```
## S4 method for signature 'DatabricksConnection,AsIs'
dbListFields(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as AsIs object (from I()) |
| ... | Additional arguments (ignored) |

### Value

Character vector of column names

---

## dbListFields,DatabricksConnection,character-method

*List column names of a Databricks table*

---

### Description

List column names of a Databricks table

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbListFields(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name to describe |
| ... | Additional arguments (ignored) |

### Value

Character vector of column names

---

## dbListTables,DatabricksConnection-method

*List tables in Databricks catalog/schema*

---

### Description

List tables in Databricks catalog/schema

### Usage

```
## S4 method for signature 'DatabricksConnection'
dbListTables(conn, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| ... | Additional arguments (ignored) |

### Value

Character vector of table names

---

## dbplyr_edition.DatabricksConnection

*Declare dbplyr API version for Databricks connections*

---

### Description

Declare dbplyr API version for Databricks connections

### Usage

```
## S3 method for class 'DatabricksConnection'
dbplyr_edition(con)
```

### Arguments

| | |
|---|---|
| con | A DatabricksConnection object |

### Value

The dbplyr edition number (2L)

---

## dbQuoteIdentifier,DatabricksConnection,character-method
*Quote identifiers for Databricks SQL*

---

### Description

Quote identifiers for Databricks SQL

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbQuoteIdentifier(conn, x, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| x | Character vector of identifiers to quote |
| ... | Additional arguments (ignored) |

### Value

SQL object with quoted identifiers

---

## dbQuoteIdentifier,DatabricksConnection,Id-method
*Quote complex identifiers (schema.table)*

---

### Description

Quote complex identifiers (schema.table)

### Usage

```
## S4 method for signature 'DatabricksConnection,Id'
dbQuoteIdentifier(conn, x, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| x | Id object with catalog/schema/table components |
| ... | Additional arguments (ignored) |

### Value

SQL object with quoted identifier components

---

## dbQuoteIdentifier,DatabricksConnection,SQL-method

*Quote SQL objects (passthrough)*

---

### Description

Quote SQL objects (passthrough)

### Usage

```
## S4 method for signature 'DatabricksConnection,SQL'
dbQuoteIdentifier(conn, x, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| x | SQL object (already quoted) |
| ... | Additional arguments (ignored) |

### Value

The SQL object unchanged

---

## dbReadTable,DatabricksConnection,AsIs-method

*Read a Databricks table (AsIs method)*

---

### Description

Read a Databricks table (AsIs method)

### Usage

```
## S4 method for signature 'DatabricksConnection,AsIs'
dbReadTable(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as AsIs object (from I()) |
| ... | Additional arguments passed to dbGetQuery |

### Value

A data.frame with table contents

---

## dbReadTable,DatabricksConnection,character-method

*Read a Databricks table*

---

### Description

Read a Databricks table

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbReadTable(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name to read |
| ... | Additional arguments passed to dbGetQuery |

### Value

A data.frame with table contents

---

## dbReadTable,DatabricksConnection,Id-method

*Read a Databricks table (Id method)*

---

### Description

Read a Databricks table (Id method)

### Usage

```
## S4 method for signature 'DatabricksConnection,Id'
dbReadTable(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as Id object |
| ... | Additional arguments passed to dbGetQuery |

### Value

A data.frame with table contents

---

## dbRemoveTable,DatabricksConnection,AsIs-method

*Remove a Databricks table (AsIs method)*

---

### Description

Remove a Databricks table (AsIs method)

### Usage

```
## S4 method for signature 'DatabricksConnection,AsIs'
dbRemoveTable(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as AsIs object (from I()) |
| ... | Additional arguments (ignored) |

### Value

TRUE invisibly on success

---

## dbRemoveTable,DatabricksConnection,character-method

*Remove a Databricks table*

---

### Description

Remove a Databricks table

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbRemoveTable(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name to remove |
| ... | Additional arguments (ignored) |

### Value

TRUE invisibly on success

---

## dbRemoveTable,DatabricksConnection,Id-method

*Remove a Databricks table (Id method)*

---

### Description

Remove a Databricks table (Id method)

### Usage

```
## S4 method for signature 'DatabricksConnection,Id'
dbRemoveTable(conn, name, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as Id object |
| ... | Additional arguments (ignored) |

### Value

TRUE invisibly on success

---

## dbRollback,DatabricksConnection-method

*Rollback transaction (not supported)*

---

### Description

Rollback transaction (not supported)

### Usage

```
## S4 method for signature 'DatabricksConnection'
dbRollback(conn, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| ... | Additional arguments (ignored) |

### Value

Always throws an error (transactions not supported)

---

## dbSendQuery,DatabricksConnection,character-method

*Send query to Databricks (asynchronous)*

---

### Description

Send query to Databricks (asynchronous)

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbSendQuery(conn, statement, disposition = conn@disposition, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| statement | SQL statement to execute |
| disposition | Query disposition mode. Defaults to the connection's disposition setting. |
| ... | Additional arguments (ignored) |

### Value

A DatabricksResult object

---

## dbSendStatement,DatabricksConnection,character-method

*Send statement to Databricks*

---

### Description

Send statement to Databricks

### Usage

```
## S4 method for signature 'DatabricksConnection,character'
dbSendStatement(conn, statement, ...)
```

### Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| statement | SQL statement |
| ... | Additional arguments (ignored) |

### Value

A DatabricksResult object

---

## dbWriteTable,DatabricksConnection,AsIs,data.frame-method

*Write table to Databricks (AsIs name signature)*

---

### Description

Write table to Databricks (AsIs name signature)

### Usage

```
## S4 method for signature 'DatabricksConnection,AsIs,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  row.names = FALSE,
  temporary = FALSE,
  field.types = NULL,
  staging_volume = NULL,
  show_progress = conn@show_progress,
  ...
)
```

### Arguments

| | |
|---|---|
| conn | DatabricksConnection object |
| name | Table name as AsIs object (from I()) |
| value | Data frame to write |
| overwrite | If TRUE, overwrite existing table |
| append | If TRUE, append to existing table |
| row.names | If TRUE, preserve row names as a column |
| temporary | If TRUE, create temporary table (NOT SUPPORTED - will error) |
| field.types | Named character vector of SQL types for columns |

| | |
|---|---|
| staging_volume | Optional volume path for large dataset staging |
| show_progress | If TRUE, show progress updates while writing. Defaults to the connection's show_progress setting. |
| ... | Additional arguments. |

## Value

TRUE invisibly on success

---

dbWriteTable,DatabricksConnection,character,data.frame-method
*Write a data frame to Databricks table*

---

## Description

Write a data frame to Databricks table

## Usage

```
## S4 method for signature 'DatabricksConnection,character,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  row.names = FALSE,
  temporary = FALSE,
  field.types = NULL,
  staging_volume = NULL,
  show_progress = conn@show_progress,
  ...
)
```

## Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name (character, Id, or SQL) |
| value | Data frame to write |
| overwrite | If TRUE, overwrite existing table |
| append | If TRUE, append to existing table |
| row.names | If TRUE, preserve row names as a column |
| temporary | If TRUE, create temporary table (NOT SUPPORTED - will error) |
| field.types | Named character vector of SQL types for columns |

| | |
|---|---|
| staging_volume | Optional volume path for large dataset staging |
| show_progress | If TRUE, show progress updates while writing. Defaults to the connection's show_progress setting. |
| ... | Additional arguments. |

## Value

TRUE invisibly on success

---

dbWriteTable,DatabricksConnection,Id,data.frame-method
*Write a data frame to Databricks table (Id method)*

---

## Description

Write a data frame to Databricks table (Id method)

## Usage

```
## S4 method for signature 'DatabricksConnection,Id,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  row.names = FALSE,
  temporary = FALSE,
  field.types = NULL,
  staging_volume = NULL,
  show_progress = conn@show_progress,
  ...
)
```

## Arguments

| | |
|---|---|
| conn | A DatabricksConnection object |
| name | Table name as Id object |
| value | Data frame to write |
| overwrite | If TRUE, overwrite existing table |
| append | If TRUE, append to existing table |
| row.names | If TRUE, preserve row names as a column |
| temporary | If TRUE, create temporary table (NOT SUPPORTED - will error) |
| field.types | Named character vector of SQL types for columns |

| | |
|---|---|
| staging_volume | Optional volume path for large dataset staging |
| show_progress | If TRUE, show progress updates while writing. Defaults to the connection's show_progress setting. |
| ... | Additional arguments. |

## Value

TRUE invisibly on success

---

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,
  kind = c("CLASSIC_PREVIEW"),
 data_security_mode = c("NONE", "SINGLE_USER", "USER_ISOLATION", "LEGACY_TABLE_ACL",
    "LEGACY_PASSTHROUGH", "LEGACY_SINGLE_USER", "LEGACY_SINGLE_USER_STANDARD",
    "DATA_SECURITY_MODE_STANDARD", "DATA_SECURITY_MODE_DEDICATED",
    "DATA_SECURITY_MODE_AUTO"),
  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.

kind
: The kind of compute described by this compute specification.

data_security_mode
: Data security mode decides what data governance model to use when accessing data from a 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

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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

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,
  kind = c("CLASSIC_PREVIEW"),
 data_security_mode = c("NONE", "SINGLE_USER", "USER_ISOLATION", "LEGACY_TABLE_ACL",
    "LEGACY_PASSTHROUGH", "LEGACY_SINGLE_USER", "LEGACY_SINGLE_USER_STANDARD",
    "DATA_SECURITY_MODE_STANDARD", "DATA_SECURITY_MODE_DEDICATED",
    "DATA_SECURITY_MODE_AUTO"),
  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.

kind
: The kind of compute described by this compute specification.

data_security_mode
: Data security mode decides what data governance model to use when accessing data from a 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

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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns a nested list with class `db_cluster`. If `FALSE`, returns an `httr2_request`.

## 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

## Value

If `perform_request = TRUE`, returns a nested list of clusters with class `db_cluster_list`; each element has class `db_cluster`. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_collect.DatabricksConnection

*Collect query results with proper progress timing for Databricks*

### Description

Collect query results with proper progress timing for Databricks

### Usage

```
## S3 method for class 'DatabricksConnection'
db_collect(
  con,
  sql,
  n = -1,
  warn_incomplete = TRUE,
  show_progress = con@show_progress,
  ...
)
```

### Arguments

| | |
|---|---|
| con | A DatabricksConnection object |
| sql | SQL query to execute |
| n | Maximum number of rows to collect (-1 for all) |
| warn_incomplete | Whether to warn if results were truncated |
| show_progress | If TRUE, show progress updates during collection. Defaults to the connection's show_progress setting. |
| ... | Additional arguments |

### Value

A data frame with query results

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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(). |

## Value

Endpoint-specific output.

## 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

A new `databricks_context_manager` object.

Command 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().

**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.

**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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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_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,
  parameters = list(),
  email_notifications = NULL,
  timeout_seconds = NULL,
  max_concurrent_runs = 1,
  access_control_list = NULL,
  git_source = NULL,
  queue = TRUE,
  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. |
| parameters | Named list of job level parameters. Values of the list represent default values. |
| 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().

queue
: If true, enable queueing for 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.

## Details

Full Documentation

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_repair_run(), 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

Other Jobs API: db_jobs_create(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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. |

## Value

If `perform_request = TRUE`, returns a nested list with class `db_job`. If `FALSE`, returns an `httr2_request`.

## See Also

Other Jobs API: `db_jobs_create()`, `db_jobs_delete()`, `db_jobs_list()`, `db_jobs_repair_run()`, `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. |

## Value

If `perform_request = TRUE`, returns a nested list of jobs with class `db_job_list`; each element has class `db_job`. If `FALSE`, returns an `httr2_request`.

## See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_repair_run(), 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_repair_run *Repair A Job Run*

---

## Description

Repair A Job Run

## Usage

```
db_jobs_repair_run(
  run_id,
  rerun_tasks = NULL,
  job_parameters = list(),
  latest_repair_id = NULL,
  performance_target = NULL,
  pipeline_full_refresh = NULL,
  rerun_all_failed_tasks = NULL,
  rerun_dependent_tasks = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| run_id | Job run ID of the run to repair. The run must not be in progress. |
| rerun_tasks | Character vector. Task keys of the task runs to repair. |
| job_parameters | Named list of job level parameters used in the run. |
| latest_repair_id | The ID of the latest repair. This parameter is not required when repairing a run for the first time, but must be provided on subsequent requests to repair the same run. |
| performance_target | The performance mode on a serverless job (either 'PERFORMANCE_OPTIMIZED' or 'STANDARD'). The performance target determines the level of compute performance or cost-efficiency for the run. This field overrides the performance target defined on the job level. |
| pipeline_full_refresh | Boolean. Controls whether the pipeline should perform a full refresh. |

rerun_all_failed_tasks
: Boolean. If TRUE, repair all failed tasks. Only one of rerun_tasks or rerun_all_failed_tasks can be used.

rerun_dependent_tasks
: Boolean. If TRUE, repair all tasks that depend on the tasks in rerun_tasks, even if they were previously successful. Can be also used in combination with rerun_all_failed_tasks.

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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_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,
  tasks,
  schedule = NULL,
  job_clusters = NULL,
  parameters = list(),
  email_notifications = NULL,
  timeout_seconds = NULL,
```

```
  max_concurrent_runs = 1,
  access_control_list = NULL,
  git_source = NULL,
  queue = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| job_id | The canonical identifier of the job. |
| 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. |
| parameters | Named list of job level parameters. Values of the list represent default values. |
| 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(). |
| queue | If true, enable queueing for 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## See Also

Other Jobs API: `db_jobs_create()`, `db_jobs_delete()`, `db_jobs_get()`, `db_jobs_list()`, `db_jobs_repair_run()`, `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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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,
  parameters = list(),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| job_id | The canonical identifier of the job. |
| parameters | Named list of job level parameters. |
| 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.

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

Other Jobs API: db_jobs_create(), db_jobs_delete(), db_jobs_get(), db_jobs_list(), db_jobs_repair_run(), 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,
  parameters = NULL,
  email_notifications = NULL,
  timeout_seconds = NULL,
  max_concurrent_runs = NULL,
  access_control_list = NULL,
```

```
  git_source = NULL,
  queue = 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. |
| parameters | Named list of job level parameters. Values of the list represent default values. |
| 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()`. |
| queue | If true, enable queueing for 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. |

## Details

Parameters which are shared with `db_jobs_create()` are optional, only specify those that are changing. Job-level parameters can be updated using the same structure as `db_jobs_create()`.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## See Also

Other Jobs API: `db_jobs_create()`, `db_jobs_delete()`, `db_jobs_get()`, `db_jobs_list()`, `db_jobs_repair_run()`, `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_lakebase_creds_generate

*Generate Database Credential*

---

## Description

Generate Database Credential

## Usage

```
db_lakebase_creds_generate(
  instance_names,
  tables = NULL,
  permission_set = c("READ_ONLY"),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| `instance_names` | Character vector of database instance names to scope the credential to. |
| `tables` | Optional character vector of table names to scope the credential to. |
| `permission_set` | Permission set for the credential request. Currently only `READ_ONLY` is supported. |
| `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

An idempotency token is generated automatically for each request (UUID4-like string).

## Value

List

## See Also

Other Database API: db_lakebase_get(), db_lakebase_get_by_uid(), db_lakebase_list()

---

`db_lakebase_get` *Get Database Instance*

---

## Description

Get Database Instance

## Usage

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

## Arguments

| | |
|---|---|
| `name` | Name of the database instance to retrieve. |
| `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

## See Also

Other Database API: db_lakebase_creds_generate(), db_lakebase_get_by_uid(), db_lakebase_list()

## db_lakebase_get_by_uid *Find Database Instance by UID*

### Description

Find Database Instance by UID

### Usage

```
db_lakebase_get_by_uid(
  uid,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| uid | UID of the database instance to retrieve. |
| 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

### See Also

Other Database API: db_lakebase_creds_generate(), db_lakebase_get(), db_lakebase_list()

## db_lakebase_list *List Database Instances*

### Description

List Database Instances

## Usage

```
db_lakebase_list(
  page_size = 50,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| page_size | Maximum number of instances to return in a single page. |
| page_token | Pagination token to retrieve the next page of 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. |

## Value

List

## See Also

Other Database API: db_lakebase_creds_generate(), db_lakebase_get(), db_lakebase_get_by_uid()

## Examples

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

# list all lakebase instances
dbs <- db_lakebase_list()

# connect to the first instance available using {RPostgres}
# using identity that brickster is running as generate a token
creds <- db_lakebase_creds_generate(instance_names = dbs[[1]]$name)

con <- dbConnect(
  drv = RPostgres::Postgres(),
  host = dbs[[1]]$read_write_dns,
  user = db_current_user()$userName,
  password = creds$token,
  dbname = "databricks_postgres",
  sslmode = "require"
)
```

```
dbListTables(con)

## End(Not run)
```

---

`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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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_
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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_
db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_
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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_s(), db_mlflow_model_version_comment(), db_mlflow_model_version_comment_delete(), db_mlflow_model_version_d(), 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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_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_perform_response(), db_req_error_body(), db_request(), db_request_json()

## db_perform_response *Perform Databricks API Request and Return Response*

### Description

Perform Databricks API Request and Return Response

### Usage

```
db_perform_response(req, ...)
```

### Arguments

| | |
|---|---|
| req | {httr2} request. |
| ... | Parameters passed to httr2::req_perform() |

### See Also

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

db_query_create *Create a SQL Query*

## Description

Create a SQL Query

## Usage

```
db_query_create(
  warehouse_id,
  query_text,
  display_name,
  description = NULL,
  catalog = NULL,
  schema = NULL,
  parent_path = NULL,
  run_as_mode = c("OWNER", "VIEWER"),
  apply_auto_limit = FALSE,
  auto_resolve_display_name = TRUE,
  tags = list(),
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| warehouse_id | description |
| query_text | Text of the query to be run. |
| display_name | Display name of the query that appears in list views, widget headings, and on the query page. |
| description | General description that conveys additional information about this query such as usage notes. |
| catalog | Name of the catalog where this query will be executed. |
| schema | Name of the schema where this query will be executed. |
| parent_path | Workspace path of the workspace folder containing the object. |
| run_as_mode | Sets the "Run as" role for the object. |
| apply_auto_limit | Whether to apply a 1000 row limit to the query result. |
| auto_resolve_display_name | Automatically resolve query display name conflicts. Otherwise, fail the request if the query's display name conflicts with an existing query's display name. |

| | |
|---|---|
| tags | Named list that describes the warehouse. Databricks tags all warehouse resources with these tags. |
| 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

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other SQL Queries API: db_query_delete(), db_query_get(), db_query_list(), db_query_update()

---

# db_query_delete *Delete a SQL Query*

---

## Description

Delete a SQL Query

## Usage

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

## Arguments

| | |
|---|---|
| id | String, ID for the 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. |

## Details

Moves a query to the trash. Trashed queries immediately disappear from searches and list views, and cannot be used for alerts. You can restore a trashed query through the UI. A trashed query is permanently deleted after 30 days.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other SQL Queries API: db_query_create(), db_query_get(), db_query_list(), db_query_update()

---

# db_query_get *Get a SQL Query*

---

## Description

Returns the repo with the given repo ID.

## Usage

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

## Arguments

| | |
|---|---|
| id | String, ID for the 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other SQL Queries API: db_query_create(), db_query_delete(), db_query_list(), db_query_update()

db_query_list *List SQL Queries*

## Description

List SQL Queries

## Usage

```
db_query_list(
  page_size = 20,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| page_size | Integer, number of results to return for each request. |
| 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

Gets a list of queries accessible to the user, ordered by creation time. Warning: Calling this API concurrently 10 or more times could result in throttling, service degradation, or a temporary ban.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other SQL Queries API: db_query_create(), db_query_delete(), db_query_get(), db_query_update()

db_query_update *Update a SQL Query*

## Description

Update a SQL Query

## Usage

```
db_query_update(
  id,
  warehouse_id = NULL,
  query_text = NULL,
  display_name = NULL,
  description = NULL,
  catalog = NULL,
  schema = NULL,
  parent_path = NULL,
  run_as_mode = NULL,
  apply_auto_limit = NULL,
  auto_resolve_display_name = NULL,
  tags = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| id | Query id |
| warehouse_id | description |
| query_text | Text of the query to be run. |
| display_name | Display name of the query that appears in list views, widget headings, and on the query page. |
| description | General description that conveys additional information about this query such as usage notes. |
| catalog | Name of the catalog where this query will be executed. |
| schema | Name of the schema where this query will be executed. |
| parent_path | Workspace path of the workspace folder containing the object. |
| run_as_mode | Sets the "Run as" role for the object. |
| apply_auto_limit | Whether to apply a 1000 row limit to the query result. |
| auto_resolve_display_name | Automatically resolve query display name conflicts. Otherwise, fail the request if the query's display name conflicts with an existing query's display name. |

| | |
|---|---|
| tags | Named list that describes the warehouse. Databricks tags all warehouse resources with these tags. |
| 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

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other SQL Queries API: db_query_create(), db_query_delete(), db_query_get(), db_query_list()

---

# 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.

### Value

Invisibly returns NULL.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_perform_response()`, `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_perform_response(), 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_perform_response(), 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.

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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.

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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_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

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

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

---

# db_sql_exec_poll_for_success *Poll a Query Until Successful*

---

## Description

Poll a Query Until Successful

## Usage

```
db_sql_exec_poll_for_success(
  statement_id,
  interval = 1,
  show_progress = TRUE,
  host = db_host(),
  token = db_token()
)
```

## Arguments

| | |
|---|---|
| statement_id | String, query execution statement_id |
| interval | Number of seconds between status checks. |
| show_progress | If TRUE, show progress updates during polling (default: TRUE) |
| host | Databricks workspace URL, defaults to calling db_host(). |
| token | Databricks workspace token, defaults to calling db_token(). |

---

# 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 = "0s",
  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.<br>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.<br>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

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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 *Execute query with SQL Warehouse*

### Description

Execute query with SQL Warehouse

## Usage

```
db_sql_query(
  warehouse_id,
  statement,
  schema = NULL,
  catalog = NULL,
  parameters = NULL,
  row_limit = NULL,
  byte_limit = NULL,
  wait_timeout = "5s",
  return_arrow = FALSE,
  max_active_connections = 30,
  fetch_timeout = 300,
  disposition = "EXTERNAL_LINKS",
  host = db_host(),
  token = db_token(),
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| warehouse_id | String, ID of warehouse upon which to execute a statement. |
| statement | String, the SQL statement to execute. The statement can optionally be parameterized, see parameters. |
| schema | String, sets default schema for statement execution, similar to USE SCHEMA in SQL. |
| catalog | String, sets default catalog for statement execution, similar to USE CATALOG in SQL. |
| parameters | List of Named Lists, parameters to pass into a SQL statement containing parameter markers.<br>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.<br>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. |
| 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.

| | |
|---|---|
| return_arrow | Boolean, determine if result is tibble::tibble or arrow::Table. |
| max_active_connections | Integer to decide on concurrent downloads. |
| fetch_timeout | Integer, timeout in seconds for downloading each result chunk |
| disposition | Disposition mode ("INLINE" or "EXTERNAL_LINKS") |
| host | Databricks workspace URL, defaults to calling db_host(). |
| token | Databricks workspace token, defaults to calling db_token(). |
| show_progress | If TRUE, show progress updates during query execution (default: TRUE) |

## Value

tibble::tibble or arrow::Table.

---

`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 endpoint_ids are NULL.

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

---

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If perform_request = TRUE, returns a nested list with class db_sql_warehouse. If FALSE, returns an httr2_request.

## 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. |

### Value

If perform_request = TRUE, returns a nested list of warehouses with class db_sql_warehouse_list; each element has class db_sql_warehouse. If FALSE, returns an httr2_request.

### 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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 db_token() returns NULL.

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_uc_catalogs_get *Get Catalog (Unity Catalog)*

### Description

Get Catalog (Unity Catalog)

### Usage

```
db_uc_catalogs_get(
  catalog,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| catalog | The name of the catalog. |
| 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

### See Also

Other Unity Catalog Management: db_uc_catalogs_list(), db_uc_schemas_get(), db_uc_schemas_list()

## db_uc_catalogs_list *List Catalogs (Unity Catalog)*

### Description

List Catalogs (Unity Catalog)

## Usage

```
db_uc_catalogs_list(
  max_results = 1000,
  include_browse = TRUE,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| max_results | Maximum number of catalogs to return (default: 1000). |
| include_browse | Whether to include catalogs in the response for which the principal can only access selective metadata for. |
| page_token | Opaque token used to get the next page of results. Optional. |
| 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

Full API response list, including next_page_token when present, or an httr2_request when perform_request = FALSE.

## See Also

Other Unity Catalog Management: db_uc_catalogs_get(), db_uc_schemas_get(), db_uc_schemas_list()

---

# db_uc_schemas_get *Get Schema (Unity Catalog)*

---

## Description

Get Schema (Unity Catalog)

## Usage

```
db_uc_schemas_get(
  catalog,
  schema,
  include_browse = TRUE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| catalog | Parent catalog for schema of interest. |
| schema | Schema of interest. |
| include_browse | Whether to include catalogs in the response for which the principal can only access selective metadata for. |
| 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

### See Also

Other Unity Catalog Management: db_uc_catalogs_get(), db_uc_catalogs_list(), db_uc_schemas_list()

---

## db_uc_schemas_list *List Schemas (Unity Catalog)*

---

### Description

List Schemas (Unity Catalog)

### Usage

```
db_uc_schemas_list(
  catalog,
  max_results = 1000,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| catalog | Parent catalog for schemas of interest. |
| max_results | Maximum number of schemas to return (default: 1000). |
| page_token | Opaque token used to get the next page of results. Optional. |
| 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

Full API response list, including `next_page_token` when present, or an `httr2_request` when `perform_request = FALSE`.

## See Also

Other Unity Catalog Management: `db_uc_catalogs_get()`, `db_uc_catalogs_list()`, `db_uc_schemas_get()`

---

`db_uc_tables_delete` *Delete Table (Unity Catalog)*

---

## Description

Delete Table (Unity Catalog)

## Usage

```
db_uc_tables_delete(
  catalog,
  schema,
  table,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| `catalog` | Parent catalog of table. |
| `schema` | Parent schema of table. |
| `table` | Table 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. |

## Value

Boolean

## See Also

Other Unity Catalog Table Management: `db_uc_tables_exists()`, `db_uc_tables_get()`, `db_uc_tables_list()`, `db_uc_tables_summaries()`

db_uc_tables_exists *Check Table Exists (Unity Catalog)*

## Description

Check Table Exists (Unity Catalog)

## Usage

```
db_uc_tables_exists(
  catalog,
  schema,
  table,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| catalog | Parent catalog of table. |
| schema | Parent schema of table. |
| table | Table 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. |

## Value

List with fields table_exists and supports_foreign_metadata_update

## See Also

Other Unity Catalog Table Management: db_uc_tables_delete(), db_uc_tables_get(), db_uc_tables_list(), db_uc_tables_summaries()

db_uc_tables_get *Get Table (Unity Catalog)*

## Description

Get Table (Unity Catalog)

## Usage

```
db_uc_tables_get(
  catalog,
  schema,
  table,
  omit_columns = TRUE,
  omit_properties = TRUE,
  omit_username = TRUE,
  include_browse = TRUE,
  include_delta_metadata = TRUE,
  include_manifest_capabilities = FALSE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| catalog | Parent catalog of table. |
| schema | Parent schema of table. |
| table | Table name. |
| omit_columns | Whether to omit the columns of the table from the response or not. |
| omit_properties | Whether to omit the properties of the table from the response or not. |
| omit_username | Whether to omit the username of the table (e.g. owner, updated_by, created_by) from the response or not. |
| include_browse | Whether to include tables in the response for which the principal can only access selective metadata for. |
| include_delta_metadata | Whether delta metadata should be included in the response. |
| include_manifest_capabilities | Whether to include a manifest containing capabilities the table has. |
| 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

## See Also

Other Unity Catalog Table Management: db_uc_tables_delete(), db_uc_tables_exists(), db_uc_tables_list(), db_uc_tables_summaries()

---

db_uc_tables_list *List Tables (Unity Catalog)*

---

## Description

List Tables (Unity Catalog)

## Usage

```
db_uc_tables_list(
  catalog,
  schema,
  max_results = 50,
  omit_columns = TRUE,
  omit_properties = TRUE,
  omit_username = TRUE,
  include_browse = TRUE,
  include_delta_metadata = FALSE,
  include_manifest_capabilities = FALSE,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| catalog | Name of parent catalog for tables of interest. |
| schema | Parent schema of tables. |
| max_results | Maximum number of tables to return (default: 50, max: 50). |
| omit_columns | Whether to omit the columns of the table from the response or not. |
| omit_properties | Whether to omit the properties of the table from the response or not. |
| omit_username | Whether to omit the username of the table (e.g. owner, updated_by, created_by) from the response or not. |
| include_browse | Whether to include tables in the response for which the principal can only access selective metadata for. |

include_delta_metadata
: Whether delta metadata should be included in the response.

include_manifest_capabilities
: Whether to include a manifest containing capabilities the table has.

page_token
: Opaque token used to get the next page of results. Optional.

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

Full API response list, including next_page_token when present, or an httr2_request when perform_request = FALSE.

## See Also

Other Unity Catalog Table Management: db_uc_tables_delete(), db_uc_tables_exists(), db_uc_tables_get(), db_uc_tables_summaries()

---

# db_uc_tables_summaries

*List Table Summaries (Unity Catalog)*

---

## Description

List Table Summaries (Unity Catalog)

## Usage

```
db_uc_tables_summaries(
  catalog,
  schema_name_pattern = NULL,
  table_name_pattern = NULL,
  max_results = 10000,
  include_manifest_capabilities = FALSE,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| catalog | Name of parent catalog for tables of interest. |
| schema_name_pattern | A sql LIKE pattern (% and _) for schema names. All schemas will be returned if not set or empty. |
| table_name_pattern | A sql LIKE pattern (% and _) for table names. All tables will be returned if not set or empty. |
| max_results | Maximum number of summaries for tables to return (default: 10000, max: 10000). If not set, the page length is set to a server configured value. |
| include_manifest_capabilities | Whether to include a manifest containing capabilities the table has. |
| page_token | Opaque token used to get the next page of results. Optional. |
| 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

## See Also

Other Unity Catalog Table Management: db_uc_tables_delete(), db_uc_tables_exists(), db_uc_tables_get(), db_uc_tables_list()

---

# db_uc_volumes_create *Update Volume (Unity Catalog)*

---

## Description

Update Volume (Unity Catalog)

## Usage

```
db_uc_volumes_create(
  catalog,
  schema,
  volume,
  volume_type = c("MANAGED", "EXTERNAL"),
  storage_location = NULL,
  comment = NULL,
  host = db_host(),
```

```
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| catalog | Parent catalog of volume |
| schema | Parent schema of volume |
| volume | Volume name. |
| volume_type | Either 'MANAGED' or 'EXTERNAL'. |
| storage_location | The storage location on the cloud, only specified when volume_type is 'EXTERNAL'. |
| comment | The comment attached to the volume. |
| 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

## See Also

Other Unity Catalog Volume Management: db_uc_volumes_delete(), db_uc_volumes_get(), db_uc_volumes_list(), db_uc_volumes_update()

---

# db_uc_volumes_delete *Delete Volume (Unity Catalog)*

---

## Description

Delete Volume (Unity Catalog)

## Usage

```
db_uc_volumes_delete(
  catalog,
  schema,
  volume,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| catalog | Parent catalog of volume |
| schema | Parent schema of volume |
| volume | Volume 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. |

### Value

Boolean

### See Also

Other Unity Catalog Volume Management: db_uc_volumes_create(), db_uc_volumes_get(), db_uc_volumes_list(), db_uc_volumes_update()

---

## db_uc_volumes_get *Get Volume (Unity Catalog)*

---

### Description

Get Volume (Unity Catalog)

### Usage

```
db_uc_volumes_get(
  catalog,
  schema,
  volume,
  include_browse = TRUE,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

### Arguments

| | |
|---|---|
| catalog | Parent catalog of volume |
| schema | Parent schema of volume |
| volume | Volume name. |
| include_browse | Whether to include volumes in the response for which the principal can only access selective metadata for. |

| | |
|---|---|
| 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

## See Also

Other Unity Catalog Volume Management: db_uc_volumes_create(), db_uc_volumes_delete(), db_uc_volumes_list(), db_uc_volumes_update()

---

# db_uc_volumes_list *List Volumes (Unity Catalog)*

---

## Description

List Volumes (Unity Catalog)

## Usage

```
db_uc_volumes_list(
  catalog,
  schema,
  max_results = 10000,
  include_browse = TRUE,
  page_token = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| catalog | Parent catalog of volume |
| schema | Parent schema of volume |
| max_results | Maximum number of volumes to return (default: 10000). |
| include_browse | Whether to include volumes in the response for which the principal can only access selective metadata for. |
| page_token | Opaque token used to get the next page of results. Optional. |
| 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

Full API response list, including `next_page_token` when present, or an `httr2_request` when `perform_request = FALSE`.

## See Also

Other Unity Catalog Volume Management: `db_uc_volumes_create()`, `db_uc_volumes_delete()`, `db_uc_volumes_get()`, `db_uc_volumes_update()`

---

`db_uc_volumes_update` *Update Volume (Unity Catalog)*

---

## Description

Update Volume (Unity Catalog)

## Usage

```
db_uc_volumes_update(
  catalog,
  schema,
  volume,
  owner = NULL,
  comment = NULL,
  new_name = NULL,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE
)
```

## Arguments

| | |
|---|---|
| `catalog` | Parent catalog of volume |
| `schema` | Parent schema of volume |
| `volume` | Volume name. |
| `owner` | The identifier of the user who owns the volume (Optional). |
| `comment` | The comment attached to the volume (Optional). |
| `new_name` | New name for the volume (Optional). |
| `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

## See Also

Other Unity Catalog Volume Management: db_uc_volumes_create(), db_uc_volumes_delete(), db_uc_volumes_get(), db_uc_volumes_list()

---

# 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Volumes FileSystem API: db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_download_dir(), db_volume_file_exists(), db_volume_list(), db_volume_read(), db_volume_upload_dir(), 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

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

## db_volume_dir_delete *Volume FileSystem Delete Directory*

### Description

Volume FileSystem Delete Directory

### Usage

```
db_volume_dir_delete(
  path,
  recursive = FALSE,
  verbose = 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. |
| recursive | If TRUE, recursively delete directory contents (default: FALSE) |
| verbose | If TRUE, announce each file/directory deletion (default: FALSE) |
| 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

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_exists(), db_volume_download_dir(), db_volume_file_exists(), db_volume_list(), db_volume_read(), db_volume_upload_dir(), 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

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

---

## db_volume_download_dir

*Download Directory from Volume in Parallel*

---

### Description

Download files from a volume directory to a local directory using parallel requests.

### Usage

```
db_volume_download_dir(
  volume_dir,
  local_dir,
  overwrite = TRUE,
  recursive = TRUE,
  host = db_host(),
  token = db_token()
)
```

### Arguments

| | |
|---|---|
| volume_dir | Volume directory path (must start with /Volumes/) |
| local_dir | Path to local directory where files will be downloaded |
| overwrite | Flag to overwrite existing local files (default: TRUE) |
| recursive | If TRUE, recursively include subdirectories (default: TRUE). If FALSE, only top-level files are transferred. |
| host | Databricks workspace URL, defaults to calling db_host(). |
| token | Databricks workspace token, defaults to calling db_token(). |

## Value

TRUE if all downloads successful

## 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_volume_upload_dir(), 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_download_dir(), db_volume_list(), db_volume_read(), db_volume_upload_dir(), 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

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

## db_volume_read *Volume FileSystem Read*

### Description

Return the contents of a file within a volume (up to 5GiB).

## Usage

```
db_volume_read(
  path,
  destination,
  host = db_host(),
  token = db_token(),
  perform_request = TRUE,
  progress = 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. |
| progress | If TRUE, show progress bar for file operations (default: TRUE for uploads/downloads, FALSE for other operations) |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## See Also

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

---

# db_volume_upload_dir *Upload Directory to Volume in Parallel*

---

## Description

Upload all files from a local directory to a volume directory using parallel requests.

### Usage

```
db_volume_upload_dir(
  local_dir,
  volume_dir,
  overwrite = TRUE,
  recursive = TRUE,
  host = db_host(),
  token = db_token()
)
```

### Arguments

| | |
|---|---|
| local_dir | Path to local directory containing files to upload |
| volume_dir | Volume directory path (must start with /Volumes/) |
| overwrite | Flag to overwrite existing files (default: TRUE) |
| recursive | If TRUE, recursively include subdirectories (default: TRUE). If FALSE, only top-level files are transferred. |
| host | Databricks workspace URL, defaults to calling db_host(). |
| token | Databricks workspace token, defaults to calling db_token(). |

### Value

TRUE if all uploads successful

### See Also

Other Volumes FileSystem API: db_volume_delete(), db_volume_dir_create(), db_volume_dir_delete(), db_volume_dir_exists(), db_volume_download_dir(), db_volume_file_exists(), db_volume_list(), db_volume_read(), 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,
  progress = 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. |
| progress | If TRUE, show progress bar for file operations (default: TRUE for uploads/downloads, FALSE for other operations) |

### Details

Uploads a file of up to 5 GiB.

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### See Also

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

---

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

### Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

### 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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_query(
  index = "myindex",
  columns = c("id", "text"),
  filters_json = '{"id >=": 5}',
  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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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. |

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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(),
  output_path = NULL,
  direct_download = FALSE,
  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(). |
| output_path | Path to export file to, ensure to include correct suffix. |
| direct_download | Boolean (default: FALSE), if TRUE download file contents directly to file. Must also specify output_path. |
| 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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`.

## Value

If `perform_request = TRUE`, returns endpoint-specific API output. If `FALSE`, returns an `httr2_request`.

## 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.

## Value

If perform_request = TRUE, returns endpoint-specific API output. If FALSE, returns an httr2_request.

## 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: `condition_task()`, `for_each_task()`, `libraries()`, `new_cluster()`, `notebook_task()`, `pipeline_task()`, `python_wheel_task()`, `run_job_task()`, `spark_jar_task()`, `spark_python_task()`, `spark_submit_task()`, `sql_file_task()`, `sql_query_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()

---

# for_each_task *For Each Task*

---

## Description

For Each Task

## Usage

```
for_each_task(inputs, task, concurrency = 1)
```

### Arguments

| | |
|---|---|
| inputs | Array for task to iterate on. This can be a JSON string or a reference to an array parameter. |
| task | Must be a job_task(). |
| concurrency | Maximum allowed number of concurrent runs of the task. |

### See Also

Other Task Objects: condition_task(), email_notifications(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_task()

---

## 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()

---

# 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.condition_task *Test if object is of class ConditionTask*

### Description

Test if object is of class ConditionTask

### Usage

```
is.condition_task(x)
```

### Arguments

| | |
|---|---|
| x | An object |

### Value

TRUE if the object inherits from the ConditionTask 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.for_each_task *Test if object is of class ForEachTask*

### Description

Test if object is of class ForEachTask

### Usage

```
is.for_each_task(x)
```

### Arguments

| | |
|---|---|
| x | An object |

### Value

TRUE if the object inherits from the ForEachTask 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.run_job_task *Test if object is of class RunJobTask*

### Description

Test if object is of class RunJobTask

### Usage

```
is.run_job_task(x)
```

### Arguments

| | |
|---|---|
| x | An object |

### Value

TRUE if the object inherits from the RunJobTask 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.sql_file_task *Test if object is of class SqlFileTask*

### Description

Test if object is of class SqlFileTask

### Usage

```
is.sql_file_task(x)
```

### Arguments

| | |
|---|---|
| x | An object |

### Value

TRUE if the object inherits from the SqlFileTask class.

## is.sql_query_task *Test if object is of class SqlQueryTask*

### Description

Test if object is of class SqlQueryTask

### Usage

```
is.sql_query_task(x)
```

### Arguments

| | |
|---|---|
| x | An object |

### Value

TRUE if the object inherits from the SqlQueryTask 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,
  run_if = c("ALL_SUCCESS", "ALL_DONE", "NONE_FAILED", "AT_LEAST_ONE_SUCCESS",
    "ALL_FAILED", "AT_LEAST_ONE_FAILED")
)
```

## 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.

`run_if` The condition determining whether the task is run once its dependencies have been completed.

---

`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: condition_task(), email_notifications(), for_each_task(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_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,
  kind = c("CLASSIC_PREVIEW"),
 data_security_mode = c("NONE", "SINGLE_USER", "USER_ISOLATION", "LEGACY_TABLE_ACL",
    "LEGACY_PASSTHROUGH", "LEGACY_SINGLE_USER", "LEGACY_SINGLE_USER_STANDARD",
    "DATA_SECURITY_MODE_STANDARD", "DATA_SECURITY_MODE_DEDICATED",
    "DATA_SECURITY_MODE_AUTO")
)
```

## 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.

kind The kind of compute described by this compute specification.

data_security_mode Data security mode decides what data governance model to use when accessing data from a cluster.

## See Also

job_task()

Other Task Objects: condition_task(), email_notifications(), for_each_task(), libraries(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_task()

---

# notebook_task *Notebook Task*

---

## Description

Notebook Task

## Usage

```
notebook_task(notebook_path, base_parameters = NULL)
```

## 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: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_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: `condition_task()`, `email_notifications()`, `for_each_task()`, `libraries()`, `new_cluster()`, `notebook_task()`, `python_wheel_task()`, `run_job_task()`, `spark_jar_task()`, `spark_python_task()`, `spark_submit_task()`, `sql_file_task()`, `sql_query_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: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_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

.libPaths(), remove_lib_path()

---

`run_job_task` *Run Job Task*

---

## Description

Run Job Task

## Usage

```
run_job_task(job_id, job_parameters, full_refresh = FALSE)
```

## Arguments

| | |
|---|---|
| job_id | ID of the job to trigger. |
| job_parameters | Named list, job-level parameters used to trigger job. |
| full_refresh | If the pipeline should perform a full refresh. |

## See Also

Other Task Objects: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_task()

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()`

---

show,DatabricksConnection-method
*Show method for DatabricksConnection*

---

## Description

Show method for DatabricksConnection

## Usage

```
## S4 method for signature 'DatabricksConnection'
show(object)
```

## Arguments

| | |
|---|---|
| object | A DatabricksConnection object |

---

show,DatabricksDriver-method
*Show method for DatabricksDriver*

---

## Description

Show method for DatabricksDriver

## Usage

```
## S4 method for signature 'DatabricksDriver'
show(object)
```

## Arguments

| | |
|---|---|
| object | A DatabricksDriver object |

## show,DatabricksResult-method *Show method for DatabricksResult*

### Description

Show method for DatabricksResult

### Usage

```
## S4 method for signature 'DatabricksResult'
show(object)
```

### Arguments

| | |
|---|---|
| object | A DatabricksResult object |

## 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: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_python_task(), spark_submit_task(), sql_file_task(), sql_query_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: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_submit_task(), sql_file_task(), sql_query_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: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), sql_file_task(), sql_query_task()

## sql_file_task *SQL File Task*

### Description

SQL File Task

### Usage

```
sql_file_task(path, warehouse_id, source = NULL, parameters = NULL)
```

### Arguments

| | |
|---|---|
| path | Path of the SQL file. Must be relative if the source is a remote Git repository and absolute for workspace paths. |
| warehouse_id | The canonical identifier of the SQL warehouse. |
| source | Optional location type of the SQL file. When set to WORKSPACE, the SQL file will be retrieved from the local Databricks workspace. When set to GIT, the SQL file will be retrieved from a Git repository defined in git_source() If the value is empty, the task will use GIT if git_source() is defined and WORKSPACE otherwise. |
| parameters | Named list of paramters to be used for each run of this job. |

### See Also

Other Task Objects: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_query_task()

## sql_query_fields.DatabricksConnection *SQL Query Fields for Databricks connections*

### Description

Generate SQL for field discovery queries optimized for Databricks. This method generates appropriate SQL for discovering table fields.

### Usage

```
## S3 method for class 'DatabricksConnection'
sql_query_fields(con, sql, ...)
```

### Arguments

| | |
|---|---|
| con | DatabricksConnection object |
| sql | SQL query to discover fields for |
| ... | Additional arguments passed to other methods |

### Value

SQL object for field discovery

---

## sql_query_save.DatabricksConnection

*Create temporary views and tables in Databricks*

---

### Description

Create temporary views and tables in Databricks

### Usage

```
## S3 method for class 'DatabricksConnection'
sql_query_save(con, sql, name, temporary = TRUE, ...)
```

### Arguments

| | |
|---|---|
| con | A DatabricksConnection object |
| sql | SQL query to save as table/view |
| name | Name for the temporary view or table |
| temporary | Whether the object should be temporary (default: TRUE) |
| ... | Additional arguments (ignored) |

### Value

The table/view name (invisibly)

## sql_query_task *SQL Query Task*

### Description

SQL Query Task

### Usage

```
sql_query_task(query_id, warehouse_id, parameters = NULL)
```

### Arguments

| | |
|---|---|
| query_id | The canonical identifier of the SQL query. |
| warehouse_id | The canonical identifier of the SQL warehouse. |
| parameters | Named list of paramters to be used for each run of this job. |

### See Also

Other Task Objects: condition_task(), email_notifications(), for_each_task(), libraries(), new_cluster(), notebook_task(), pipeline_task(), python_wheel_task(), run_job_task(), spark_jar_task(), spark_python_task(), spark_submit_task(), sql_file_task()

## sql_table_analyze.DatabricksConnection *Handle table analysis for Databricks*

### Description

Handle table analysis for Databricks

### Usage

```
## S3 method for class 'DatabricksConnection'
sql_table_analyze(con, table, ...)
```

### Arguments

| | |
|---|---|
| con | A DatabricksConnection object |
| table | Table name to analyze |
| ... | Additional arguments (ignored) |

### Value

SQL statement for table analysis

`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


∗ **Access Control Request Objects**
access_control_req_group, 9
access_control_req_user, 9
∗ **Cloud Attributes**
aws_attributes, 11
azure_attributes, 13
gcp_attributes, 201
∗ **Cluster Helpers**
get_and_start_cluster, 202
get_latest_dbr, 204
∗ **Cluster Log Configuration Objects**
cluster_log_conf, 15
dbfs_storage_info, 29
s3_storage_info, 239
∗ **Cluster Objects**
cluster_autoscale, 14
∗ **Clusters API**
db_cluster_create, 45
db_cluster_edit, 49
db_cluster_events, 52
db_cluster_get, 53
db_cluster_list, 54
db_cluster_list_node_types, 55
db_cluster_list_zones, 56
db_cluster_perm_delete, 57
db_cluster_pin, 58
db_cluster_resize, 59
db_cluster_restart, 60
db_cluster_runtime_versions, 61
db_cluster_start, 62
db_cluster_terminate, 63
db_cluster_unpin, 64
get_and_start_cluster, 202
get_latest_dbr, 204
∗ **Database API**
db_lakebase_creds_generate, 92
db_lakebase_get, 93
db_lakebase_get_by_uid, 94
db_lakebase_list, 94
∗ **Databricks Authentication Helpers**
db_host, 75
db_read_netrc, 116
db_token, 151
db_wsid, 194
∗ **Execution Context API**
db_context_command_cancel, 65
db_context_command_run, 66
db_context_command_run_and_wait, 67
db_context_command_status, 68
db_context_create, 69
db_context_destroy, 70
db_context_status, 72
∗ **Init Script Info Objects**
dbfs_storage_info, 29
file_storage_info, 200
s3_storage_info, 239
∗ **Jobs API**
db_jobs_create, 76
db_jobs_delete, 77
db_jobs_get, 78
db_jobs_list, 79
db_jobs_repair_run, 80
db_jobs_reset, 81
db_jobs_run_now, 89
db_jobs_runs_cancel, 83
db_jobs_runs_delete, 84
db_jobs_runs_export, 84
db_jobs_runs_get, 85
db_jobs_runs_get_output, 86
db_jobs_runs_list, 87
db_jobs_runs_submit, 88
db_jobs_update, 90
∗ **Libraries API**
db_libs_all_cluster_statuses, 96
db_libs_cluster_status, 97
db_libs_install, 97
db_libs_uninstall, 99

∗ **Library Objects**
lib_cran, 230
lib_egg, 231
lib_jar, 231
lib_maven, 232
lib_pypi, 232
lib_whl, 233
libraries, 230
∗ **Model Registry API**
db_mlflow_model_approve_transition_req, 100
db_mlflow_model_delete_transition_req, 101
db_mlflow_model_open_transition_reqs, 102
db_mlflow_model_reject_transition_req, 103
db_mlflow_model_transition_req, 104
db_mlflow_model_transition_stage, 105
db_mlflow_model_version_comment, 106
db_mlflow_model_version_comment_delete, 107
db_mlflow_model_version_comment_edit, 108
db_mlflow_registered_model_details, 109
∗ **Repos API**
db_repo_create, 117
db_repo_delete, 118
db_repo_get, 119
db_repo_get_all, 120
db_repo_update, 121
∗ **Request Helpers**
db_perform_request, 110
db_perform_response, 110
db_req_error_body, 123
db_request, 122
db_request_json, 122
∗ **SQL Execution APIs**
db_sql_exec_cancel, 134
db_sql_exec_query, 135
db_sql_exec_result, 138
db_sql_exec_status, 139
∗ **SQL Queries API**
db_query_create, 111
db_query_delete, 112
db_query_get, 113
db_query_list, 114
db_query_update, 115
∗ **SQL Query History API**
db_sql_query_history, 142
∗ **Secrets API**
db_secrets_delete, 123
db_secrets_list, 124
db_secrets_put, 125
db_secrets_scope_acl_delete, 127
db_secrets_scope_acl_get, 128
db_secrets_scope_acl_list, 129
db_secrets_scope_acl_put, 130
db_secrets_scope_create, 131
db_secrets_scope_delete, 132
db_secrets_scope_list_all, 133
∗ **Task Objects**
condition_task, 15
email_notifications, 198
for_each_task, 200
libraries, 230
new_cluster, 233
notebook_task, 235
pipeline_task, 237
python_wheel_task, 237
run_job_task, 238
spark_jar_task, 241
spark_python_task, 242
spark_submit_task, 242
sql_file_task, 243
sql_query_task, 245
∗ **Unity Catalog Management**
db_uc_catalogs_get, 152
db_uc_catalogs_list, 152
db_uc_schemas_get, 153
db_uc_schemas_list, 154
∗ **Unity Catalog Table Management**
db_uc_tables_delete, 155
db_uc_tables_exists, 156
db_uc_tables_get, 157
db_uc_tables_list, 158
db_uc_tables_summaries, 159
∗ **Unity Catalog Volume Management**
db_uc_volumes_create, 160
db_uc_volumes_delete, 161
db_uc_volumes_get, 162
db_uc_volumes_list, 163

db_uc_volumes_update, 164
∗ **Vector Search API**
db_vs_endpoints_create, 173
db_vs_endpoints_delete, 174
db_vs_endpoints_get, 175
db_vs_endpoints_list, 176
db_vs_indexes_create, 177
db_vs_indexes_delete, 178
db_vs_indexes_delete_data, 179
db_vs_indexes_get, 180
db_vs_indexes_list, 181
db_vs_indexes_query, 182
db_vs_indexes_query_next_page, 184
db_vs_indexes_scan, 185
db_vs_indexes_sync, 186
db_vs_indexes_upsert_data, 187
delta_sync_index_spec, 195
direct_access_index_spec, 196
embedding_source_column, 199
embedding_vector_column, 199
∗ **Volumes FileSystem API**
db_volume_delete, 165
db_volume_dir_create, 166
db_volume_dir_delete, 166
db_volume_dir_exists, 167
db_volume_download_dir, 168
db_volume_file_exists, 169
db_volume_list, 170
db_volume_read, 170
db_volume_upload_dir, 171
db_volume_write, 172
∗ **Warehouse API**
db_sql_global_warehouse_get, 140
db_sql_warehouse_create, 143
db_sql_warehouse_delete, 145
db_sql_warehouse_edit, 146
db_sql_warehouse_get, 147
db_sql_warehouse_list, 148
db_sql_warehouse_start, 149
db_sql_warehouse_stop, 150
get_and_start_warehouse, 203
∗ **Warehouse Helpers**
get_and_start_warehouse, 203
∗ **Workspace API**
db_workspace_delete, 188
db_workspace_export, 189
db_workspace_get_status, 190
db_workspace_import, 191
db_workspace_list, 192
db_workspace_mkdirs, 193
.libPaths(), *10*, *238*

access_control_req_group, 9, *10*
access_control_req_group(), *8*, *77*
access_control_req_user, *9*, 9
access_control_req_user(), *8*, *77*
access_control_request, 8
access_control_request(), *9, 10*, *77*, *82*, *89*, *91*
add_lib_path, 10
arrow::Table, *142*
aws_attributes, 11, *13*, *201*
aws_attributes(), *46*, *50*, *234*
azure_attributes, *13*, 13, *201*
azure_attributes(), *46*, *50*, *234*

close_workspace, 14
cluster_autoscale, 14
cluster_autoscale(), *46*, *50*, *59*, *234*
cluster_log_conf, 15, *29*, *239*
cluster_log_conf(), *29*, *46*, *50*, *235*, *239*
condition_task, 15, *198*, *201*, *230*, *235–238*, *241–243*, *245*
copy_to.DatabricksConnection, 16
cron_schedule, 17
cron_schedule(), *76, 77*, *82*, *91*

databricks-dbi, 17
databricks-dbplyr, 18
DatabricksConnection-class, 18
DatabricksDriver-class, 18
DatabricksResult-class, 18
DatabricksSQL, 19
db_cluster_action, 44
db_cluster_create, 45, *52–64*, *202*, *204*
db_cluster_create(), *13, 14*, *62*, *197*, *201*, *206*
db_cluster_delete, 48
db_cluster_edit, *48*, 49, *53–64*, *202*, *204*
db_cluster_edit(), *13, 14*, *197*, *201*, *206*
db_cluster_events, *48*, *52*, 52, *54–64*, *202*, *204*
db_cluster_get, *48*, *52, 53*, 53, *55–64*, *202*, *204*
db_cluster_get(), *47*, *202*
db_cluster_list, *48*, *52–54*, 54, *56–64*, *202*, *204*

db_cluster_list(), *58*, *64*
db_cluster_list_node_types, *48*, *52–55*, 55, *56–64*, *202*, *204*
db_cluster_list_node_types(), *46*, *50*, *234*
db_cluster_list_zones, *48*, *52–56*, 56, *57–64*, *202*, *204*
db_cluster_perm_delete, *48*, *52–56*, 57, *58–64*, *202*, *204*
db_cluster_pin, *48*, *52–57*, 58, *59–64*, *202*, *204*
db_cluster_resize, *48*, *52–58*, 59, *60–64*, *202*, *204*
db_cluster_restart, *48*, *52–59*, 60, *61–64*, *202*, *204*
db_cluster_runtime_versions, *48*, *52–60*, 61, *62–64*, *202*, *204*
db_cluster_runtime_versions(), *46*, *50*, *234*
db_cluster_start, *48*, *52–61*, 62, *63, 64*, *202*, *204*
db_cluster_start(), *202*
db_cluster_terminate, *48*, *52–62*, 63, *64*, *202*, *204*
db_cluster_unpin, *48*, *52–63*, 64, *202*, *204*
db_collect.DatabricksConnection, 65
db_context_command_cancel, 65, *67–71*, *73*
db_context_command_parse, *66–71*, *73*
db_context_command_run, *66*, 66, *68–71*, *73*
db_context_command_run_and_wait, *66, 67*, 67, *69–71*, *73*
db_context_command_status, *66–68*, 68, *70, 71*, *73*
db_context_create, *66–69*, 69, *71*, *73*
db_context_destroy, *66–70*, 70, *73*
db_context_manager, 71
db_context_status, *66–71*, 72
db_current_cloud, 73
db_current_user, 73
db_current_workspace_id, 74
db_host, 75, *116*, *151*, *194*
db_host(), *14*, *44*, *47, 48*, *51–64*, *66–70*, *72–74*, *77–79*, *81–86*, *88–109*, *112–114*, *116–122*, *124–131*, *133–135*, *137–140*, *142–145*, *147–150*, *152–157*, *159–182*, *184–193*, *202–204*, *236*, *246*
db_jobs_create, 76, *78–81*, *83–90*, *92*
db_jobs_create(), *8*, *17*, *81*, *91*, *228, 229*
db_jobs_delete, *77*, 77, *79–81*, *83–90*, *92*
db_jobs_get, *77, 78*, 78, *80, 81*, *83–90*, *92*
db_jobs_list, *77–79*, 79, *81*, *83–90*, *92*
db_jobs_repair_run, *77–80*, 80, *83–90*, *92*
db_jobs_reset, *77–81*, 81, *83–90*, *92*
db_jobs_reset(), *8*, *17*, *228, 229*
db_jobs_run_now, *77–81*, *83–89*, 89, *92*
db_jobs_run_now(), *236*
db_jobs_runs_cancel, *77–81*, *83*, 83, *84–90*, *92*
db_jobs_runs_delete, *77–81*, *83*, 84, *85–90*, *92*
db_jobs_runs_export, *77–81*, *83, 84*, 84, *86–90*, *92*
db_jobs_runs_get, *77–81*, *83–85*, 85, *87–90*, *92*
db_jobs_runs_get_output, *77–81*, *83–86*, 86, *88–90*, *92*
db_jobs_runs_list, *77–81*, *83–87*, 87, *89, 90*, *92*
db_jobs_runs_submit, *77–81*, *83–88*, 88, *90*, *92*
db_jobs_update, *77–81*, *83–90*, 90
db_jobs_update(), *8*, *17*, *228, 229*
db_lakebase_creds_generate, 92, *93–95*
db_lakebase_get, *93*, 93, *94, 95*
db_lakebase_get_by_uid, *93*, 94, *95*
db_lakebase_list, *93, 94*, 94
db_libs_all_cluster_statuses, 96, *97–99*
db_libs_cluster_status, *96*, 97, *98, 99*
db_libs_cluster_status(), *246*
db_libs_install, *96, 97*, 97, *99*
db_libs_uninstall, *96–98*, 99
db_mlflow_model_approve_transition_req, 100, *102–104*, *106–109*
db_mlflow_model_delete_transition_req, *100*, 101, *102–104*, *106–109*
db_mlflow_model_open_transition_reqs, *100*, *102*, 102, *103, 104*, *106–109*
db_mlflow_model_reject_transition_req, *100*, *102*, 103, *104*, *106–109*
db_mlflow_model_transition_req, *100*, *102, 103*, 104, *106–109*
db_mlflow_model_transition_stage, *100*, *102–104*, 105, *106–109*
db_mlflow_model_version_comment, *100*, *102–104*, *106*, 106, *107–109*

db_mlflow_model_version_comment_delete, *100*, *102–104*, *106*, 107, *108, 109*
db_mlflow_model_version_comment_edit, *100*, *102–104*, *106, 107*, 108, *109*
db_mlflow_registered_model_details, *100*, *102–104*, *106–108*, 109
db_perform_request, *110*, 110, *122, 123*
db_perform_response, *110*, 110, *122, 123*
db_query_create, 111, *113, 114*, *116*
db_query_delete, *112*, 112, *113, 114*, *116*
db_query_get, *112, 113*, 113, *114*, *116*
db_query_list, *112, 113*, 114, *116*
db_query_update, *112–114*, 115
db_read_netrc, *75*, 116, *151*, *194*
db_repl, 117
db_repo_create, 117, *119–121*
db_repo_delete, *118*, 118, *119–121*
db_repo_get, *118, 119*, 119, *120, 121*
db_repo_get_all, *118, 119*, 120, *121*
db_repo_update, *118–120*, 121
db_req_error_body, *110*, *122, 123*, 123
db_request, *110*, 122, *123*
db_request(), *122*
db_request_json, *110*, *122*, 122, *123*
db_secrets_delete, 123, *125–129*, *131–134*
db_secrets_list, *124*, 124, *126–129*, *131–134*
db_secrets_put, *124, 125*, 125, *127–129*, *131–134*
db_secrets_scope_acl_delete, *124–126*, 127, *128, 129*, *131–134*
db_secrets_scope_acl_get, *124–127*, 128, *129*, *131–134*
db_secrets_scope_acl_list, *124–128*, 129, *131–134*
db_secrets_scope_acl_put, *124–129*, 130, *132–134*
db_secrets_scope_create, *124–129*, *131*, 131, *133, 134*
db_secrets_scope_delete, *124–129*, *131, 132*, 132, *134*
db_secrets_scope_list_all, *124–129*, *131–133*, 133
db_sql_exec_cancel, 134, *137–139*
db_sql_exec_poll_for_success, 135
db_sql_exec_query, *135*, 135, *138, 139*
db_sql_exec_result, *135*, *137*, 138, *139*
db_sql_exec_result(), *138*
db_sql_exec_status, *135*, *137, 138*, 139
db_sql_exec_status(), *137, 138*, *142*
db_sql_global_warehouse_get, 140, *145*, *147–150*, *203*
db_sql_query, 140
db_sql_query_history, 142
db_sql_warehouse_create, *140*, 143, *145*, *147–150*, *203*
db_sql_warehouse_delete, *140*, *145*, 145, *147–150*, *203*
db_sql_warehouse_edit, *140*, *145*, 146, *148–150*, *203*
db_sql_warehouse_get, *140*, *145*, *147*, 147, *149, 150*, *203*
db_sql_warehouse_get(), *203*
db_sql_warehouse_list, *140*, *145*, *147, 148*, 148, *150*, *203*
db_sql_warehouse_start, *140*, *145*, *147–149*, 149, *150*, *203*
db_sql_warehouse_start(), *203*
db_sql_warehouse_stop, *140*, *145*, *147–150*, 150, *203*
db_token, *75*, *116*, 151, *194*
db_token(), *44*, *47, 48*, *51–64*, *66–70*, *72–74*, *77–79*, *81–86*, *88–109*, *112–114*, *116–122*, *124–131*, *133–135*, *137–140*, *142–145*, *147–150*, *152–157*, *159–182*, *184–193*, *202–204*, *236*, *246*
db_uc_catalogs_get, 152, *153–155*
db_uc_catalogs_list, *152*, 152, *154, 155*
db_uc_schemas_get, *152, 153*, 153, *155*
db_uc_schemas_list, *152–154*, 154
db_uc_tables_delete, 155, *156*, *158–160*
db_uc_tables_exists, *155*, 156, *158–160*
db_uc_tables_get, *155, 156*, 157, *159, 160*
db_uc_tables_list, *155, 156*, *158*, 158, *160*
db_uc_tables_summaries, *155, 156*, *158, 159*, 159
db_uc_volumes_create, 160, *162–165*
db_uc_volumes_delete, *161*, 161, *163–165*
db_uc_volumes_get, *161, 162*, 162, *164, 165*
db_uc_volumes_list, *161–163*, 163, *165*
db_uc_volumes_update, *161–164*, 164
db_volume_delete, 165, *166–173*
db_volume_dir_create, *165*, 166, *167–173*
db_volume_dir_delete, *165, 166*, 166, *168–173*

db_volume_dir_exists, *165–167*, 167, *169–173*
db_volume_download_dir, *165–168*, 168, *169–173*
db_volume_file_exists, *165–169*, 169, *170–173*
db_volume_list, *165–169*, 170, *171–173*
db_volume_read, *165–170*, 170, *172, 173*
db_volume_upload_dir, *165–171*, 171, *173*
db_volume_write, *165–172*, 172
db_vs_endpoints_create, 173, *175–181*, *183–187*, *195*, *197*, *199, 200*
db_vs_endpoints_delete, *174*, 174, *176–181*, *183–187*, *195*, *197*, *199, 200*
db_vs_endpoints_get, *174, 175*, 175, *176–181*, *183–187*, *195*, *197*, *199, 200*
db_vs_endpoints_list, *174–176*, 176, *177–181*, *183–187*, *195*, *197*, *199, 200*
db_vs_indexes_create, *174–176*, 177, *178–181*, *183–187*, *195*, *197*, *199, 200*
db_vs_indexes_create(), *195*, *197*
db_vs_indexes_delete, *174–177*, 178, *179–181*, *183–187*, *195*, *197*, *199, 200*
db_vs_indexes_delete_data, *174–178*, 179, *180, 181*, *183–187*, *195*, *197*, *199, 200*
db_vs_indexes_get, *174–179*, 180, *181*, *183–187*, *195*, *197*, *199, 200*
db_vs_indexes_list, *174–180*, 181, *183–187*, *195*, *197*, *199, 200*
db_vs_indexes_query, *174–181*, 182, *184–187*, *195*, *197*, *199, 200*
db_vs_indexes_query_next_page, *174–181*, *183*, 184, *185–187*, *195*, *197*, *199, 200*
db_vs_indexes_scan, *174–181*, *183, 184*, 185, *186, 187*, *196, 197*, *199, 200*
db_vs_indexes_sync, *174–181*, *183–185*, 186, *187*, *196, 197*, *199, 200*
db_vs_indexes_upsert_data, *174–181*, *183–186*, 187, *196, 197*, *199, 200*
db_workspace_delete, 188, *190*, *192, 193*
db_workspace_export, *188*, 189, *190*, *192, 193*
db_workspace_get_status, *188*, *190*, 190, *192, 193*
db_workspace_import, *188*, *190*, 191, *192, 193*
db_workspace_list, *188*, *190*, *192*, 192, *193*
db_workspace_mkdirs, *188*, *190*, *192*, 193
db_wsid, *75*, *116*, *151*, 194
dbAppendTable,DatabricksConnection,character,data.frame-me 19
dbAppendTable,DatabricksConnection,Id,data.frame-method, 20
dbBegin,DatabricksConnection-method, 20
dbClearResult,DatabricksResult-method, 21
dbColumnInfo,DatabricksResult-method, 21
dbCommit,DatabricksConnection-method, 22
dbConnect,DatabricksDriver-method, 22
dbCreateTable,DatabricksConnection,AsIs-method, 23
dbCreateTable,DatabricksConnection,character-method, 24
dbCreateTable,DatabricksConnection,Id-method, 25
dbDataType,DatabricksConnection-method, 25
dbDisconnect,DatabricksConnection-method, 26
dbExecute,DatabricksConnection,character-method, 26
dbExistsTable,DatabricksConnection,AsIs-method, 27
dbExistsTable,DatabricksConnection,character-method, 27
dbExistsTable,DatabricksConnection,Id-method, 28
dbFetch,DatabricksResult-method, 28
dbfs_storage_info, *15*, 29, *200*, *239*
dbfs_storage_info(), *15*, *205*
dbGetInfo,DatabricksConnection-method, 29
dbGetQuery,DatabricksConnection,character-method, 30
dbGetRowCount,DatabricksResult-method, 30

dbGetRowsAffected,DatabricksResult-method, 31
dbGetStatement,DatabricksResult-method, 31
dbHasCompleted,DatabricksResult-method, 32
dbIsValid,DatabricksConnection-method, 32
dbListFields,DatabricksConnection,AsIs-method, 33
dbListFields,DatabricksConnection,character-method, 33
dbListTables,DatabricksConnection-method, 34
dbplyr_edition.DatabricksConnection, 34
dbQuoteIdentifier,DatabricksConnection,character-method, 35
dbQuoteIdentifier,DatabricksConnection,Id-method, 35
dbQuoteIdentifier,DatabricksConnection,SQL-method, 36
dbReadTable,DatabricksConnection,AsIs-method, 36
dbReadTable,DatabricksConnection,character-method, 37
dbReadTable,DatabricksConnection,Id-method, 37
dbRemoveTable,DatabricksConnection,AsIs-method, 38
dbRemoveTable,DatabricksConnection,character-method, 38
dbRemoveTable,DatabricksConnection,Id-method, 39
dbRollback,DatabricksConnection-method, 39
dbSendQuery,DatabricksConnection,character-method, 40
dbSendStatement,DatabricksConnection,character-method, 40
dbWriteTable,DatabricksConnection,AsIs,data.frame-method, 41
dbWriteTable,DatabricksConnection,character,data.frame-method, 42
dbWriteTable,DatabricksConnection,Id,data.frame-method, 43
delta_sync_index_spec, *174–181*, *183–187*, 195, *197*, *199, 200*
delta_sync_index_spec(), *177*
direct_access_index_spec, *174–181*, *183–187*, *196*, 196, *199, 200*
direct_access_index_spec(), *177*
docker_image, 197
docker_image(), *47*, *51*

email_notifications, *16*, 198, *201*, *229, 230*, *235–238*, *241–243*, *245*
email_notifications(), *76, 77*, *82*, *91*
embedding_source_column, *174–181*, *183–187*, *196, 197*, 199, *200*
embedding_source_column(), *195, 196*
embedding_vector_column, *174–181*, *183–187*, *196, 197*, *199*, 199
embedding_vector_column(), *195, 196*

file_storage_info, *29*, 200, *239*
file_storage_info(), *205*
for_each_task, *16*, *198*, 200, *230*, *235–238*, *241–243*, *245*

gcp_attributes, *13*, 201
gcp_attributes(), *46*, *50*, *234*
get_and_start_cluster, *48*, *52–64*, 202, *204*
get_and_start_warehouse, *140*, *145*, *147–150*, 203
get_latest_dbr, *48*, *52–64*, *202*, 204
git_source, 205
git_source(), *77*, *82*, *89*, *91*, *243*

httr2::req_body_json(), *122*
httr2::req_method(), *122*
httr2::req_perform(), *110*
httr2::resp_body_json(), *110*

in_databricks_nb, 206
init_script_info, 205
init_script_info(), *29*, *46*, *50*, *200*, *235*, *239*
is.access_control_req_group, 207
is.access_control_req_user, 207
is.access_control_request, 206
is.aws_attributes, 208
is.azure_attributes, 208
is.cluster_autoscale, 209
is.cluster_log_conf, 209
is.condition_task, 210

is.cron_schedule, 210
is.dbfs_storage_info, 211
is.delta_sync_index, 211
is.direct_access_index, 212
is.docker_image, 212
is.email_notifications, 213
is.embedding_source_column, 213
is.embedding_vector_column, 214
is.file_storage_info, 214
is.for_each_task, 215
is.gcp_attributes, 215
is.git_source, 216
is.init_script_info, 216
is.job_task, 217
is.lib_cran, 218
is.lib_egg, 219
is.lib_jar, 219
is.lib_maven, 220
is.lib_pypi, 220
is.lib_whl, 221
is.libraries, 217
is.library, 218
is.new_cluster, 221
is.notebook_task, 222
is.pipeline_task, 222
is.python_wheel_task, 223
is.run_job_task, 223
is.s3_storage_info, 224
is.spark_jar_task, 224
is.spark_python_task, 225
is.spark_submit_task, 225
is.sql_file_task, 226
is.sql_query_task, 226
is.valid_task_type, 227
is.vector_search_index_spec, 227

job_task, 228
job_task(), *77*, *198*, *201*, *229*, *230*, *235*
job_tasks, 229
job_tasks(), *76*, *77*, *82*, *88*, *91*

lib_cran, *230*, 230, *231–233*
lib_cran(), *98*, *230*
lib_egg, *230*, *231*, 231, *232*, *233*
lib_egg(), *98*, *230*
lib_jar, *230*, *231*, 231, *232*, *233*
lib_jar(), *98*, *230*
lib_maven, *230–232*, 232, *233*
lib_maven(), *98*, *230*
lib_pypi, *230–232*, 232, *233*
lib_pypi(), *98*, *230*
lib_whl, *230–233*, 233
lib_whl(), *98*, *230*
libraries, *16*, *198*, *201*, 230, *231–233*, *235–238*, *241–243*, *245*
libraries(), *98*, *99*, *229*, *231–233*

new_cluster, *16*, *198*, *201*, *230*, 233, *236–238*, *241–243*, *245*
new_cluster(), *76*, *82*, *89*, *91*, *228*
notebook_task, *16*, *198*, *201*, *230*, *235*, 235, *237*, *238*, *241–243*, *245*
notebook_task(), *228*

open_workspace, 236

pipeline_task, *16*, *198*, *201*, *230*, *235*, *236*, 237, *238*, *241–243*, *245*
pipeline_task(), *228*
python_wheel_task, *16*, *198*, *201*, *230*, *235–237*, 237, *238*, *241–243*, *245*
python_wheel_task(), *228*

remove_lib_path, 238
remove_lib_path(), *10*, *238*
run_job_task, *16*, *198*, *201*, *230*, *235–238*, 238, *241–243*, *245*

s3_storage_info, *15*, *29*, *200*, 239
s3_storage_info(), *15*, *205*
show,DatabricksConnection-method, 240
show,DatabricksDriver-method, 240
show,DatabricksResult-method, 241
spark_jar_task, *16*, *198*, *201*, *230*, *235–238*, 241, *242*, *243*, *245*
spark_jar_task(), *228*
spark_python_task, *16*, *198*, *201*, *230*, *235–238*, *241*, *242*, 242, *243*, *245*
spark_python_task(), *228*
spark_submit_task, *16*, *198*, *201*, *230*, *235–238*, *241*, *242*, 242, *243*, *245*
spark_submit_task(), *228*
sql_file_task, *16*, *198*, *201*, *230*, *235–238*, *241*, *242*, 243, *245*
sql_query_fields.DatabricksConnection, 243
sql_query_save.DatabricksConnection, 244

sql_query_task, *16*, *198*, *201*, *230*, *235–238*, *241–243*, 245
sql_table_analyze.DatabricksConnection, 245

tibble::tibble, *142*

wait_for_lib_installs, 246
wait_for_lib_installs(), *97*