Package 'fabricQueryR'

September 24, 2026

Title Access and Manage 'Microsoft Fabric'

Version 1.0.0

Description Access 'Microsoft Fabric' workspaces, items, and workload endpoints through its web application programming interfaces (APIs). Connect to data in 'OneLake', 'Lakehouse', 'Warehouse', semantic model, and 'Eventhouse' items, with support for 'DBI', 'Arrow', 'GraphQL', and 'Spark'. Manage files, tables, refreshes, jobs, schedules, ingestion, and long-running operations.

License MIT + file LICENSE

Suggests adbcdrivermanager, adbi, arrow (>= 17.0.0), DBI (>= 1.2.0), dplyr, knitr, lifecycle, odbc, processx, purrr, rmarkdown, testthat (>= 3.2.0), waldo, webfakes, withr

VignetteBuilder knitr

Additional_repositories https://r-dbi.r-universe.dev

Config/testthat/edition 3

Config/Needs/ci adbcdrivermanager=url::https://cloud.r-project.org/src/contrib/Archive/adbcdrivermanager/adbcdrivermanager_0.23.0-2.tar.gz, adbi=url::https://cloud.r-project.org/src/contrib/Archive/adbi/adbi_0.1.2.tar.gz

Encoding UTF-8

Imports AzureAuth, bit64, httr2 (>= 1.2.0), R6, vctrs, tibble, jsonlite, methods, cli (>= 3.4.0), nanoarrow (>= 0.8.0), reticulate (>= 1.41), rlang (>= 0.4.10), utils

URL https://github.com/kennispunttwente/fabricQueryR, https://kennispunttwente.github.io/fabricQueryR/

BugReports https://github.com/kennispunttwente/fabricQueryR/issues

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Luka Koning [aut, cre, cph], Kennispunt Twente [fnd]

Maintainer Luka Koning <koningluka@gmail.com>

Repository CRAN

Date/Publication 2026-09-24 09:00:02 UTC

Contents

FabricItem . . . 3
FabricLivyBatch . . . 24
FabricLivySession . . . 27
FabricLivyStatement . . . 31
fabric_catalog_search . . . 33
fabric_delta_config . . . 35
fabric_function_invoke . . . 38
fabric_graphql_collect . . . 41
fabric_graphql_cursor . . . 43
fabric_graphql_paginate . . . 44
fabric_graphql_query . . . 46
fabric_graphql_schema . . . 50
fabric_item . . . 52
fabric_items . . . 54
fabric_job_instances . . . 56
fabric_job_run . . . 58
fabric_job_schedules . . . 63
fabric_job_schedule_config . . . 67
fabric_kql_export . . . 69
fabric_kql_ingest . . . 73
fabric_kql_query . . . 78
fabric_kql_read_table . . . 80
fabric_kql_tables . . . 82
fabric_kql_write_table . . . 83
fabric_lakehouse_read_table . . . 88
fabric_lakehouse_tables . . . 90
fabric_livy_batch_submit . . . 96
fabric_livy_query . . . 100
fabric_livy_session . . . 103
fabric_livy_sessions . . . 106
fabric_mirrored_database_tables . . . 109
fabric_onelake_catalog . . . 112
fabric_onelake_files . . . 115
fabric_onelake_object_files . . . 120
fabric_onelake_read_delta_table . . . 123
fabric_onelake_schema_exists . . . 127
fabric_onelake_shortcuts . . . 129
fabric_operation_status . . . 134
fabric_pbi_dax_query . . . 137
fabric_pbi_refresh . . . 141
fabric_sql_connect . . . 147

fabric_sql_connection_info . . . 150
fabric_sql_query . . . 151
fabric_sql_tables . . . 155
fabric_typed_items . . . 157
fabric_user_data_functions . . . 160
fabric_warehouse_read_table . . . 162
fabric_warehouse_tables . . . 165
fabric_warehouse_write_table . . . 166
fabric_workspaces . . . 170
print.fabric_graphql_rows . . . 172
print.fabric_job . . . 172
print.fabric_job_instance . . . 173
print.fabric_job_schedule . . . 173
print.fabric_kql_export_result . . . 174
print.fabric_kql_ingestion . . . 174
print.fabric_kql_ingestion_status . . . 175
print.fabric_kql_write_result . . . 175
print.fabric_pbi_refresh . . . 176
print.fabric_pbi_refresh_detail . . . 176

Index **177**

`FabricItem` *R6 objects for discovered Microsoft Fabric resources*

Description

These generators back the default `output = "r6"` discovery interface. Users normally receive objects from `fabric_workspaces()`, `fabric_items()`, `fabric_item()`, the typed discovery helpers, or `fabric_catalog_search()` rather than constructing them directly.

Format

An R6::R6Class generator.

Details

Every object includes the complete Fabric API record. Read non-conflicting fields directly with `$`; use `$get()` for collision-safe field access and `$as_list()` or `as.list()` for a plain record. `$get()` is an object-only field helper. Record fields are read-only.

Methods delegate to the corresponding `fabric_*()` function. Their `...` arguments are forwarded unchanged, and the credential used for discovery is reused while the object is in the current R process. An explicitly supplied `token`, `tenant_id`, `client_id`, `auth_args`, or `api_base` takes precedence. Job and refresh lifecycle methods prefer the supplied handle's in-process credential over the discovery credential. Bare IDs and handles whose credentials were removed by serialization use the discovery credential. The Fabric API base used for discovery is also reused, so chained methods stay on the same public, sovereign-cloud, or workspace endpoint.

SQL-capable resources inherit common `sql_*()` methods. Lakehouses, Warehouses, mirrored databases, Eventhouses, KQL databases, GraphQL APIs, semantic models, and runnable job items add workload-specific methods. Other discovered types are returned as `FabricItem` objects with `$details()` (`fabric_item()`) and record access. They do not expose methods that cannot operate from discovery metadata. This generic fallback also applies to typed Environment and User Data Function discovery; a typed helper and workload detail route do not by themselves imply a specialized R6 class.

`FabricEventhouse` and `FabricKqlDatabase` share the KQL methods documented below under their internal `FabricKqlItem` superclass: `$query()`, `$tables()`, `$read_table()`, `$ingest()`, `$write_table()`, `$export()`, `$ingestion_status()`, and `$ingestion_wait()`.

Value

The corresponding R6 generator.

Super class

`FabricRecord` -> `FabricWorkspace`

Methods

Public methods:

- `FabricWorkspace$new()`
- `FabricWorkspace$items()`
- `FabricWorkspace$item()`
- `FabricWorkspace$lakehouses()`
- `FabricWorkspace$warehouses()`
- `FabricWorkspace$warehouse_snapshots()`
- `FabricWorkspace$mirrored_databases()`
- `FabricWorkspace$sql_databases()`
- `FabricWorkspace$semantic_models()`
- `FabricWorkspace$eventhouses()`
- `FabricWorkspace$kql_databases()`
- `FabricWorkspace$notebooks()`
- `FabricWorkspace$data_pipelines()`
- `FabricWorkspace$spark_job_definitions()`
- `FabricWorkspace$environments()`
- `FabricWorkspace$user_data_functions()`
- `FabricWorkspace$graphql_apis()`
- `FabricWorkspace$shortcut_cache_reset()`

`FabricWorkspace$new()`**:** Internal constructor used by discovery factories.

Usage:

```
FabricWorkspace$new(
  record,
  legacy_class = c("fabric_workspace", "list"),
  credential = NULL,
  api_base = NULL
)
```

Arguments:

`record` One named Fabric workspace record.

`legacy_class` Classes assigned by `$as_list()`.

`credential` Optional internal authentication credential.

`api_base` Optional Fabric REST API base inherited from discovery.

`FabricWorkspace$items()`**:** Discover items in this workspace.

Usage:

```
FabricWorkspace$items(...)
```

Arguments:

`...` Arguments forwarded to `fabric_items()`.

Returns: A list of `FabricItem` objects or type-specific subclasses.

`FabricWorkspace$item()`**:** Discover and enrich one item in this workspace.

Usage:

```
FabricWorkspace$item(item, ...)
```

Arguments:

`item` Item GUID, display name, or discovered item.

`...` Arguments forwarded to `fabric_item()`.

Returns: A `FabricItem` object or one of its subclasses.

`FabricWorkspace$lakehouses()`**:** Discover Lakehouses in this workspace.

Usage:

```
FabricWorkspace$lakehouses(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricLakehouse` objects.

`FabricWorkspace$warehouses()`**:** Discover Warehouses in this workspace.

Usage:

```
FabricWorkspace$warehouses(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricWarehouse` objects.

`FabricWorkspace$warehouse_snapshots()`: Discover Warehouse snapshots in this workspace.

Usage:

```
FabricWorkspace$warehouse_snapshots(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricWarehouseSnapshot` objects.

`FabricWorkspace$mirrored_databases()`: Discover mirrored databases in this workspace.

Usage:

```
FabricWorkspace$mirrored_databases(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricMirroredDatabase` objects.

`FabricWorkspace$sql_databases()`: Discover SQL databases in this workspace.

Usage:

```
FabricWorkspace$sql_databases(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricSqlDatabase` objects.

`FabricWorkspace$semantic_models()`: Discover semantic models in this workspace.

Usage:

```
FabricWorkspace$semantic_models(detail = FALSE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricSemanticModel` objects.

`FabricWorkspace$eventhouses()`: Discover Eventhouses in this workspace.

Usage:

```
FabricWorkspace$eventhouses(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricEventhouse` objects.

`FabricWorkspace$kql_databases()`: Discover KQL databases in this workspace.

Usage:

```
FabricWorkspace$kql_databases(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricKqlDatabase` objects.

`FabricWorkspace$notebooks()`**:** Discover notebooks in this workspace.

Usage:

```
FabricWorkspace$notebooks(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricJobItem` objects.

`FabricWorkspace$data_pipelines()`**:** Discover data pipelines in this workspace.

Usage:

```
FabricWorkspace$data_pipelines(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricJobItem` objects.

`FabricWorkspace$spark_job_definitions()`**:** Discover Spark job definitions in this workspace.

Usage:

```
FabricWorkspace$spark_job_definitions(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricJobItem` objects.

`FabricWorkspace$environments()`**:** Discover environments in this workspace.

Usage:

```
FabricWorkspace$environments(detail = TRUE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricItem` objects.

`FabricWorkspace$user_data_functions()`**: [Experimental]**
Discover User Data Functions in this workspace. The service-principal development sandbox can use Core discovery, but it cannot provision and fully inspect disposable User Data Function fixtures.

Usage:

```
FabricWorkspace$user_data_functions(detail = FALSE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricItem` objects.

`FabricWorkspace$graphql_apis()`: Discover GraphQL APIs in this workspace.

Usage:

```
FabricWorkspace$graphql_apis(detail = FALSE, ...)
```

Arguments:

`detail` Whether to retrieve workload details.

`...` Additional discovery arguments.

Returns: A list of `FabricGraphQLApi` objects.

`FabricWorkspace$shortcut_cache_reset()`: Reset this workspace's OneLake shortcut cache.

Usage:

```
FabricWorkspace$shortcut_cache_reset(...)
```

Arguments:

`...` Arguments forwarded to `fabric_onelake_shortcut_cache_reset()`.

Returns: A `fabric_operation` handle.

Super class

`FabricRecord` -> `FabricItem`

Methods

Public methods:

- `FabricItem$new()`
- `FabricItem$details()`

`FabricItem$new()`: Internal constructor used by discovery factories.

Usage:

```
FabricItem$new(
  record,
  legacy_class = c("fabric_item", "list"),
  credential = NULL,
  api_base = NULL
)
```

Arguments:

`record` One named Fabric item record.

`legacy_class` Classes assigned by `$as_list()`.

`credential` Optional internal authentication credential.

`api_base` Optional Fabric REST API base inherited from discovery.

`FabricItem$details()`: Retrieve a fresh item record and supported workload details. User Data Function workload details require `detail = TRUE` and a delegated user identity.

Usage:

```
FabricItem$details(...)
```

Arguments:

`...` Arguments forwarded to `fabric_item()`.

Returns: A new `FabricItem` object or one of its subclasses.

Super classes

`FabricRecord` -> `FabricItem` -> `FabricSqlItem` -> `FabricLakehouse`

Methods

Public methods:

- `FabricLakehouse$schemas()`
- `FabricLakehouse$table()`
- `FabricLakehouse$tables()`
- `FabricLakehouse$read_table()`
- `FabricLakehouse$read_delta_table()`
- `FabricLakehouse$load_table()`
- `FabricLakehouse$write_table()`
- `FabricLakehouse$livy_query()`
- `FabricLakehouse$livy_session()`
- `FabricLakehouse$livy_batch_submit()`
- `FabricLakehouse$onelake_list()`
- `FabricLakehouse$onelake_metadata()`
- `FabricLakehouse$onelake_read_file()`
- `FabricLakehouse$onelake_write_file()`
- `FabricLakehouse$onelake_download()`
- `FabricLakehouse$onelake_upload()`
- `FabricLakehouse$onelake_delete()`
- `FabricLakehouse$schema_exists()`
- `FabricLakehouse$table_exists()`
- `FabricLakehouse$shortcuts()`
- `FabricLakehouse$shortcut()`
- `FabricLakehouse$shortcut_create()`
- `FabricLakehouse$shortcuts_bulk_create()`
- `FabricLakehouse$shortcut_delete()`

`FabricLakehouse$schemas()`: List Lakehouse schemas.

Usage:

```
FabricLakehouse$schemas(...)
```

Arguments:

`...` Arguments forwarded to `fabric_lakehouse_schemas()`.

Returns: A schema inventory tibble.

`FabricLakehouse$table()`: Retrieve one Lakehouse table's metadata.

Usage:

```
FabricLakehouse$table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_lakehouse_table()`.

Returns: A one-row table metadata tibble.

`FabricLakehouse$tables()`: List Lakehouse tables.

Usage:

```
FabricLakehouse$tables(...)
```

Arguments:

`...` Arguments forwarded to `fabric_lakehouse_tables()`.

Returns: A table inventory tibble.

`FabricLakehouse$read_table()`: Read one managed Delta table.

Usage:

```
FabricLakehouse$read_table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_lakehouse_read_table()`.

Returns: A tibble or Arrow stream.

`FabricLakehouse$read_delta_table()`: Read a Delta table directly from OneLake.

Usage:

```
FabricLakehouse$read_delta_table(table_path, ...)
```

Arguments:

`table_path` Table path below the item.

`...` Arguments forwarded to `fabric_onelake_read_delta_table()`.

Returns: A tibble or Arrow stream.

`FabricLakehouse$load_table()`: Load an existing file into a managed table.

Usage:

```
FabricLakehouse$load_table(table, path, ...)
```

Arguments:

`table` Destination table name.

`path` Source path under the Lakehouse.

`...` Arguments forwarded to `fabric_lakehouse_load_table()`.

Returns: A `fabric_operation` or completed operation result.

`FabricLakehouse$write_table()`: Write R or Arrow data to a managed table.

Usage:

```
FabricLakehouse$write_table(table, data, ...)
```

Arguments:

`table` Destination table name.

`data` Data frame or Arrow-compatible source.

`...` Arguments forwarded to `fabric_lakehouse_write_table()`.

Returns: A completed write result.

`FabricLakehouse$livy_query()`: Run one Spark statement through Livy.

Usage:

```
FabricLakehouse$livy_query(code, ...)
```

Arguments:

`code` Spark, PySpark, SparkR, or Spark SQL code.

`...` Arguments forwarded to `fabric_livy_query()`.

Returns: A Livy statement result.

`FabricLakehouse$livy_session()`: Create a reusable Livy session.

Usage:

```
FabricLakehouse$livy_session(...)
```

Arguments:

`...` Arguments forwarded to `fabric_livy_session()`.

Returns: A FabricLivySession.

`FabricLakehouse$livy_batch_submit()`: Submit a standalone Livy batch.

Usage:

```
FabricLakehouse$livy_batch_submit(file, ...)
```

Arguments:

`file` ABFSS application-file URI.

`...` Arguments forwarded to `fabric_livy_batch_submit()`.

Returns: A FabricLivyBatch or its result.

`FabricLakehouse$onelake_list()`: List OneLake files and directories.

Usage:

```
FabricLakehouse$onelake_list(path = "", ...)
```

Arguments:

`path` Path within the Lakehouse.

`...` Arguments forwarded to `fabric_onelake_list()`.

Returns: A file inventory tibble.

`FabricLakehouse$onelake_metadata()`: Retrieve OneLake path metadata.

Usage:

```
FabricLakehouse$onelake_metadata(path = "", ...)
```

Arguments:

`path` Path within the Lakehouse.

`...` Arguments forwarded to `fabric_onelake_metadata()`.

Returns: A one-row metadata tibble.

`FabricLakehouse$onelake_read_file()`: Read a supported file from OneLake.

Usage:

```
FabricLakehouse$onelake_read_file(path, ...)
```

Arguments:

`path` Path within the Lakehouse.

`...` Arguments forwarded to `fabric_onelake_read_file()`.

Returns: A tibble or Arrow stream.

`FabricLakehouse$onelake_write_file()`: Write data to a supported OneLake file.

Usage:

```
FabricLakehouse$onelake_write_file(path, data, ...)
```

Arguments:

`path` Destination path within the Lakehouse.

`data` Data to write.

`...` Arguments forwarded to `fabric_onelake_write_file()`.

Returns: A file-write result.

`FabricLakehouse$onelake_download()`: Download one OneLake file.

Usage:

```
FabricLakehouse$onelake_download(path, ...)
```

Arguments:

`path` Source path within the Lakehouse.

`...` Arguments forwarded to `fabric_onelake_download()`.

Returns: The local destination path.

`FabricLakehouse$onelake_upload()`: Upload a local file or raw vector to OneLake.

Usage:

```
FabricLakehouse$onelake_upload(path, source, ...)
```

Arguments:

path Destination path within the Lakehouse.

source Local path or raw vector.

... Arguments forwarded to fabric_onelake_upload().

Returns: A file-write result.

`FabricLakehouse$onelake_delete()`: Delete a OneLake path.

Usage:

```
FabricLakehouse$onelake_delete(path, ...)
```

Arguments:

path Path within the Lakehouse.

... Arguments forwarded to fabric_onelake_delete().

Returns: TRUE, invisibly, after deletion.

`FabricLakehouse$schema_exists()`: Check whether a schema exists.

Usage:

```
FabricLakehouse$schema_exists(schema, ...)
```

Arguments:

schema Schema name.

... Arguments forwarded to fabric_onelake_schema_exists().

Returns: One logical value.

`FabricLakehouse$table_exists()`: Check whether a table exists.

Usage:

```
FabricLakehouse$table_exists(table, ...)
```

Arguments:

table Table name or discovered table row.

... Arguments forwarded to fabric_onelake_table_exists().

Returns: One logical value.

`FabricLakehouse$shortcuts()`: List OneLake shortcuts.

Usage:

```
FabricLakehouse$shortcuts(...)
```

Arguments:

... Arguments forwarded to fabric_onelake_shortcuts().

Returns: A shortcut inventory tibble.

`FabricLakehouse$shortcut()`: Retrieve one OneLake shortcut.

Usage:

```
FabricLakehouse$shortcut(path, name, ...)
```

Arguments:

path Shortcut parent path.

`name` Shortcut name.

`...` Arguments forwarded to `fabric_onelake_shortcut_get()`.

Returns: A one-row shortcut tibble.

`FabricLakehouse$shortcut_create()`: Create a OneLake shortcut.

Usage:

```
FabricLakehouse$shortcut_create(path, name, target, ...)
```

Arguments:

`path` Shortcut parent path.

`name` Shortcut name.

`target` Shortcut target.

`...` Arguments forwarded to `fabric_onelake_shortcut_create()`.

Returns: A one-row shortcut tibble.

`FabricLakehouse$shortcuts_bulk_create()`: Create multiple OneLake shortcuts.

Usage:

```
FabricLakehouse$shortcuts_bulk_create(shortcuts, ...)
```

Arguments:

`shortcuts` Shortcut request lists.

`...` Arguments forwarded to `fabric_onelake_shortcuts_bulk_create()`.

Returns: A `fabric_operation` handle.

`FabricLakehouse$shortcut_delete()`: Delete one OneLake shortcut.

Usage:

```
FabricLakehouse$shortcut_delete(path, name, ...)
```

Arguments:

`path` Shortcut parent path.

`name` Shortcut name.

`...` Arguments forwarded to `fabric_onelake_shortcut_delete()`.

Returns: `TRUE`, invisibly, after deletion.

Super classes

`FabricRecord` -> `FabricItem` -> `FabricSqlItem` -> `FabricWarehouse`

Methods

Public methods:

- `FabricWarehouse$schemas()`
- `FabricWarehouse$table()`
- `FabricWarehouse$tables()`
- `FabricWarehouse$read_table()`
- `FabricWarehouse$write_table()`

- FabricWarehouse$read_delta_table()

`FabricWarehouse$schemas()`: List Warehouse schemas.

Usage:

```
FabricWarehouse$schemas(...)
```

Arguments:

`...` Arguments forwarded to `fabric_warehouse_schemas()`.

Returns: A schema inventory tibble.

`FabricWarehouse$table()`: Retrieve one Warehouse table's metadata.

Usage:

```
FabricWarehouse$table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_warehouse_table()`.

Returns: A one-row table metadata tibble.

`FabricWarehouse$tables()`: List Warehouse tables.

Usage:

```
FabricWarehouse$tables(...)
```

Arguments:

`...` Arguments forwarded to `fabric_warehouse_tables()`.

Returns: A table inventory tibble.

`FabricWarehouse$read_table()`: Read one Warehouse table.

Usage:

```
FabricWarehouse$read_table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_warehouse_read_table()`.

Returns: A tibble or Arrow stream.

`FabricWarehouse$write_table()`: Write R or Arrow data to a Warehouse table.

Usage:

```
FabricWarehouse$write_table(table, data, staging_lakehouse, ...)
```

Arguments:

`table` Destination table name.

`data` Data frame or Arrow-compatible source.

`staging_lakehouse` Lakehouse used for staged Parquet data.

`...` Arguments forwarded to `fabric_warehouse_write_table()`.

Returns: A completed write result.

`FabricWarehouse$read_delta_table()`: Read a Delta table directly from OneLake.

Usage:

```
FabricWarehouse$read_delta_table(table_path, ...)
```

Arguments:

`table_path` Table path below the item.

`...` Arguments forwarded to `fabric_onelake_read_delta_table()`.

Returns: A tibble or Arrow stream.

Super classes

`FabricRecord` -> `FabricItem` -> `FabricSqlItem` -> `FabricWarehouseSnapshot`

Super classes

`FabricRecord` -> `FabricItem` -> `FabricSqlItem` -> `FabricSqlDatabase`

Super classes

`FabricRecord` -> `FabricItem` -> `FabricSqlItem` -> `FabricMirroredDatabase`

Methods

Public methods:

- `FabricMirroredDatabase$schemas()`
- `FabricMirroredDatabase$table()`
- `FabricMirroredDatabase$tables()`
- `FabricMirroredDatabase$read_table()`
- `FabricMirroredDatabase$read_delta_table()`

`FabricMirroredDatabase$schemas()`: List mirrored database schemas.

Usage:

```
FabricMirroredDatabase$schemas(...)
```

Arguments:

`...` Arguments forwarded to `fabric_mirrored_database_schemas()`.

Returns: A schema inventory tibble.

`FabricMirroredDatabase$table()`: Retrieve one mirrored table's metadata.

Usage:

```
FabricMirroredDatabase$table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_mirrored_database_table()`.

Returns: A one-row table metadata tibble.

`FabricMirroredDatabase$tables()`: List mirrored database tables.

Usage:

```
FabricMirroredDatabase$tables(...)
```

Arguments:

`...` Arguments forwarded to `fabric_mirrored_database_tables()`.

Returns: A table inventory tibble.

`FabricMirroredDatabase$read_table()`: Read one mirrored Delta table.

Usage:

```
FabricMirroredDatabase$read_table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_mirrored_database_read_table()`.

Returns: A tibble or Arrow stream.

`FabricMirroredDatabase$read_delta_table()`: Read a Delta table directly from OneLake.

Usage:

```
FabricMirroredDatabase$read_delta_table(table_path, ...)
```

Arguments:

`table_path` Table path below the item.

`...` Arguments forwarded to `fabric_onelake_read_delta_table()`.

Returns: A tibble or Arrow stream.

Super classes

`FabricRecord` -> `FabricItem` -> `FabricKqlItem`

Methods

Public methods:

- `FabricKqlItem$query()`
- `FabricKqlItem$tables()`
- `FabricKqlItem$read_table()`
- `FabricKqlItem$ingest()`
- `FabricKqlItem$write_table()`
- `FabricKqlItem$export()`
- `FabricKqlItem$ingestion_status()`
- `FabricKqlItem$ingestion_wait()`

`FabricKqlItem$query()`: Run a KQL query.

Usage:

```
FabricKqlItem$query(query, ...)
```

Arguments:

`query` One KQL query.

`...` Arguments forwarded to `fabric_kql_query()`.

Returns: A typed tibble, or a `fabric_kql_tables` list for multiple primary results; see `fabric_kql_query()`.

`FabricKqlItem$tables()`: List KQL tables.

Usage:

```
FabricKqlItem$tables(...)
```

Arguments:

`...` Arguments forwarded to `fabric_kql_tables()`.

Returns: A table inventory tibble.

`FabricKqlItem$read_table()`: Read one KQL table.

Usage:

```
FabricKqlItem$read_table(table, ...)
```

Arguments:

`table` Table name or discovered table row.

`...` Arguments forwarded to `fabric_kql_read_table()`.

Returns: A typed tibble with Kusto metadata attributes.

`FabricKqlItem$ingest()`: Ingest existing sources into a KQL table.

Usage:

```
FabricKqlItem$ingest(table, sources, format, ...)
```

Arguments:

`table` Destination table name.

`sources` Source URLs or source records.

`format` Source data format.

`...` Arguments forwarded to `fabric_kql_ingest()`.

Returns: A `fabric_kql_ingestion` handle.

`FabricKqlItem$write_table()`: Write R or Arrow data to a KQL table.

Usage:

```
FabricKqlItem$write_table(table, data, ...)
```

Arguments:

`table` Destination table name.

`data` Data frame or Arrow-compatible source.

`...` Arguments forwarded to `fabric_kql_write_table()`.

Returns: A `fabric_kql_write_result` with ingestion status and staging disposition.

`FabricKqlItem$export()`: Export a KQL query to Fabric storage.

Usage:

```
FabricKqlItem$export(query, destination, ...)
```

Arguments:

`query` One KQL query.

`destination` Destination Fabric item or OneLake target.

`...` Arguments forwarded to `fabric_kql_export()`.

Returns: A `fabric_kql_export_result` with operation state, artifact paths, and record counts.

`FabricKqlItem$ingestion_status()`: Retrieve one KQL ingestion status snapshot.

Usage:

```
FabricKqlItem$ingestion_status(ingestion, ...)
```

Arguments:

`ingestion` Ingestion handle, status record, or operation ID.

`...` Arguments forwarded to `fabric_kql_ingestion_status()`.

Returns: A `fabric_kql_ingestion_status` record.

`FabricKqlItem$ingestion_wait()`: Wait for a KQL ingestion to finish.

Usage:

```
FabricKqlItem$ingestion_wait(ingestion, ...)
```

Arguments:

`ingestion` Ingestion handle, status record, or operation ID.

`...` Arguments forwarded to `fabric_kql_ingestion_status()`.

Returns: A terminal `fabric_kql_ingestion_status` record.

Super classes

`FabricRecord` -> `FabricItem` -> `FabricKqlItem` -> `FabricEventhouse`

Super classes

`FabricRecord` -> `FabricItem` -> `FabricKqlItem` -> `FabricKqlDatabase`

Super classes

`FabricRecord` -> `FabricItem` -> `FabricGraphQLApi`

Methods

Public methods:

- `FabricGraphQLApi$query()`
- `FabricGraphQLApi$schema()`
- `FabricGraphQLApi$paginate()`

`FabricGraphQLApi$query()`: Run a GraphQL query.

Usage:

```
FabricGraphQLApi$query(query, ...)
```

Arguments:

`query` One GraphQL query.

`...` Arguments forwarded to `fabric_graphql_query()`.

Returns: A `fabric_graphql_result`.

`FabricGraphQLApi$schema()`: Retrieve the GraphQL schema.

Usage:

```
FabricGraphQLApi$schema(...)
```

Arguments:

`...` Arguments forwarded to `fabric_graphql_schema()`.

Returns: A `fabric_graphql_schema`.

`FabricGraphQLApi$paginate()`: Retrieve all pages of a GraphQL cursor query.

Usage:

```
FabricGraphQLApi$paginate(query, next_cursor, ...)
```

Arguments:

`query` One GraphQL query.

`next_cursor` Function extracting the next cursor.

`...` Arguments forwarded to `fabric_graphql_paginate()`.

Returns: A `fabric_graphql_pages` list.

Super classes

`FabricRecord` -> `FabricItem` -> `FabricSemanticModel`

Methods

Public methods:

- `FabricSemanticModel$dax_query()`
- `FabricSemanticModel$refresh()`
- `FabricSemanticModel$refresh_history()`
- `FabricSemanticModel$refresh_status()`
- `FabricSemanticModel$refresh_wait()`
- `FabricSemanticModel$refresh_cancel()`

`FabricSemanticModel$dax_query()`: Run a DAX query against this semantic model.

Usage:

```
FabricSemanticModel$dax_query(dax, ...)
```

Arguments:

`dax` One DAX query.

... Arguments forwarded to fabric_pbi_dax_query().

Returns: A tibble or Arrow stream.

`FabricSemanticModel$refresh()`: Start a refresh of this semantic model.

Usage:

```
FabricSemanticModel$refresh(...)
```

Arguments:

... Arguments forwarded to fabric_pbi_refresh().

Returns: A `fabric_pbi_refresh` handle.

`FabricSemanticModel$refresh_history()`: Retrieve this semantic model's refresh history.

Usage:

```
FabricSemanticModel$refresh_history(...)
```

Arguments:

... Arguments forwarded to fabric_pbi_refresh_history().

Returns: A `fabric_pbi_refresh_history` list.

`FabricSemanticModel$refresh_status()`: Retrieve one refresh status snapshot.

Usage:

```
FabricSemanticModel$refresh_status(refresh, ...)
```

Arguments:

`refresh` Refresh handle, detail record, or request ID.

... Arguments forwarded to fabric_pbi_refresh_status().

Returns: A `fabric_pbi_refresh_detail` record.

`FabricSemanticModel$refresh_wait()`: Wait for a submitted refresh to finish.

Usage:

```
FabricSemanticModel$refresh_wait(refresh, ...)
```

Arguments:

`refresh` Refresh handle or detail record.

... Arguments forwarded to fabric_pbi_refresh_wait().

Returns: A terminal `fabric_pbi_refresh_detail` record.

`FabricSemanticModel$refresh_cancel()`: Cancel a submitted refresh.

Usage:

```
FabricSemanticModel$refresh_cancel(refresh, ...)
```

Arguments:

`refresh` Refresh handle, detail record, or request ID.

... Arguments forwarded to fabric_pbi_refresh_cancel().

Returns: `TRUE`, invisibly, when cancellation is accepted.

Super classes

FabricRecord -> FabricItem -> FabricJobItem

Methods

Public methods:

- FabricJobItem$run()
- FabricJobItem$status()
- FabricJobItem$wait()
- FabricJobItem$cancel()
- FabricJobItem$instances()
- FabricJobItem$schedules()
- FabricJobItem$schedule_create()
- FabricJobItem$schedule_update()
- FabricJobItem$schedule_delete()

`FabricJobItem$run()`: Start an on-demand item job.

Usage:

```
FabricJobItem$run(...)
```

Arguments:

`...` Arguments forwarded to `fabric_job_run()`.

Returns: A `fabric_job` handle.

`FabricJobItem$status()`: Retrieve one job status snapshot.

Usage:

```
FabricJobItem$status(job = NULL, ...)
```

Arguments:

`job` Job handle, instance record, or job instance ID.

`...` Arguments forwarded to `fabric_job_status()`.

Returns: A `fabric_job_instance` record.

`FabricJobItem$wait()`: Wait for a submitted job to finish.

Usage:

```
FabricJobItem$wait(job, ...)
```

Arguments:

`job` Job handle or instance record.

`...` Arguments forwarded to `fabric_job_wait()`.

Returns: A terminal `fabric_job_instance` record.

`FabricJobItem$cancel()`: Cancel a submitted job.

Usage:

```
FabricJobItem$cancel(job = NULL, ...)
```

Arguments:

`job` Job handle, instance record, or job instance ID.

`...` Arguments forwarded to `fabric_job_cancel()`.

Returns: `TRUE`, invisibly, when cancellation is accepted.

`FabricJobItem$instances()`: Retrieve this item's job history.

Usage:

```
FabricJobItem$instances(...)
```

Arguments:

`...` Arguments forwarded to `fabric_job_instances()`.

Returns: A `fabric_job_instance_list`.

`FabricJobItem$schedules()`: List this item's schedules.

Usage:

```
FabricJobItem$schedules(...)
```

Arguments:

`...` Arguments forwarded to `fabric_job_schedules()`.

Returns: A `fabric_job_schedule_list`.

`FabricJobItem$schedule_create()`: Create an item schedule.

Usage:

```
FabricJobItem$schedule_create(configuration, ...)
```

Arguments:

`configuration` A `fabric_job_schedule_config()` record.

`...` Arguments forwarded to `fabric_job_schedule_create()`.

Returns: A `fabric_job_schedule` record.

`FabricJobItem$schedule_update()`: Update an item schedule.

Usage:

```
FabricJobItem$schedule_update(schedule_id, configuration = NULL, ...)
```

Arguments:

`schedule_id` Schedule GUID or schedule record.

`configuration` Optional replacement schedule configuration.

`...` Arguments forwarded to `fabric_job_schedule_update()`.

Returns: A `fabric_job_schedule` record.

`FabricJobItem$schedule_delete()`: Delete an item schedule.

Usage:

```
FabricJobItem$schedule_delete(schedule_id, ...)
```

Arguments:

`schedule_id` Schedule GUID or schedule record.

`...` Arguments forwarded to `fabric_job_schedule_delete()`.

Returns: `TRUE`, invisibly, after deletion.

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
workspace$displayName

# Equivalent function: fabric_lakehouses(workspace)
lakehouse <- workspace$lakehouses()[[1L]]
lakehouse$id
# Equivalent function: fabric_lakehouse_tables(lakehouse)
lakehouse$tables()
# Equivalent function: fabric_lakehouse_read_table(lakehouse, ...)
orders <- lakehouse$read_table("orders", limit = 100L)

# Workload-specific subclasses expose their own lifecycle methods
# Equivalent function: fabric_semantic_models(workspace)
model <- workspace$semantic_models()[[1L]]
# Equivalent function: fabric_pbi_refresh(model)
refresh <- model$refresh()
# Equivalent function: fabric_pbi_refresh_wait(refresh, ...)
model$refresh_wait(refresh, timeout = 1800)

# Equivalent generic: as.list(lakehouse)
lakehouse_record <- lakehouse$as_list()

# Fabric item types outside the typed-helper subset remain usable records
report <- workspace$items(type = "Report")[[1L]]
report$type

## End(Not run)
```

`FabricLivyBatch` *A Microsoft Fabric Livy batch job*

Description

Represents a Spark application submitted with `fabric_livy_batch_submit()` Use `$wait()` to wait for completion, `$result()` or `$logs()` to inspect the outcome, and `$cancel()` to request cancellation. Most users do not need to call this 'R6' class directly. These lifecycle methods do not have separate free-function wrappers

Format

An R6::R6Class generator

Value

The `FabricLivyBatch` 'R6' generator.

Public fields

`id` Fabric batch ID

`url` Batch lifecycle URL

`state` Latest batch state

`response` Latest raw service response

`cancel_requested` Whether `$cancel()` was called successfully

`submitted_local` Local submission timestamp

`completed_local` Local completion timestamp

`verbose` Whether lifecycle messages are enabled

Methods

Public methods:

- `FabricLivyBatch$new()`
- `FabricLivyBatch$print()`
- `FabricLivyBatch$status()`
- `FabricLivyBatch$wait()`
- `FabricLivyBatch$logs()`
- `FabricLivyBatch$result()`
- `FabricLivyBatch$cancel()`

`FabricLivyBatch$new()`: Internal constructor used by `fabric_livy_batch_submit()`

Usage:

```
FabricLivyBatch$new(response, url, credential, verbose = TRUE)
```

Arguments:

`response` Initial batch response

`url` Batch collection URL

`credential` Internal authentication credential

`verbose` Whether to emit lifecycle messages

Returns: A new batch object

`FabricLivyBatch$print()`: Print a concise batch summary

Usage:

```
FabricLivyBatch$print(...)
```

Arguments:

`...` Unused

Returns: `self`, invisibly

`FabricLivyBatch$status()`: Retrieve current batch metadata

Usage:

```
FabricLivyBatch$status(refresh = TRUE, deadline = NULL)
```

Arguments:

`refresh` Whether to retrieve current state from Fabric

`deadline` Internal wall-clock deadline for the status request

Returns: The raw batch response list

`FabricLivyBatch$wait()`**:** Wait for the batch to reach a terminal state

Usage:

```
FabricLivyBatch$wait(
  timeout = 1200,
  poll_interval = 5,
  error_on_failure = TRUE,
  cancel_on_timeout = FALSE
)
```

Arguments:

`timeout` Maximum wait in seconds

`poll_interval` Polling interval in seconds

`error_on_failure` Raise a structured error for a failed batch

`cancel_on_timeout` Request cancellation before raising a timeout

Returns: `self`, invisibly

`FabricLivyBatch$logs()`**:** Return available Spark driver log lines

Usage:

```
FabricLivyBatch$logs(refresh = TRUE)
```

Arguments:

`refresh` Whether to retrieve current state from Fabric

Returns: A character vector

`FabricLivyBatch$result()`**:** Return structured batch metadata and logs

Usage:

```
FabricLivyBatch$result(refresh = TRUE, error_on_failure = TRUE)
```

Arguments:

`refresh` Whether to retrieve current state from Fabric

`error_on_failure` Raise a structured error for a failed batch

Returns: A `fabric_livy_batch_result` list

`FabricLivyBatch$cancel()`**:** Request batch cancellation

Usage:

```
FabricLivyBatch$cancel(deadline = NULL)
```

Arguments:

`deadline` Internal wall-clock deadline for the cancellation request

Returns: `TRUE`, invisibly, after Fabric accepts the request

Examples

```
## Not run:
# fabric_livy_batch_submit() returns this class for a submitted Spark job
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
scripts <- fabric_onelake_list(workspace, lakehouse, "Files/jobs")
script <- scripts[grepl("[.]py$", scripts$path), ][1L, ]
script_uri <- paste0(
  "abfss://", workspace$id, "@onelake.dfs.fabric.microsoft.com/",
  lakehouse$id, "/", script$path[[1L]]
)
batch <- fabric_livy_batch_submit(
  lakehouse,
  file = script_uri
)
inherits(batch, "FabricLivyBatch")

# Wait for completion, then inspect the application result and logs
batch$wait()
batch$result()
batch$logs()

## End(Not run)
```

FabricLivySession *A Microsoft Fabric Livy session*

Description

A Livy session keeps Spark running while you submit several pieces of code Create one with `fabric_livy_session()`, call `$wait()` once it starts, use `$run()` to execute code, and call `$close()` when finished. Most users do not need to call this 'R6' class directly. These lifecycle methods do not have separate free-function wrappers

Format

An R6::R6Class generator

Value

The `FabricLivySession` 'R6' generator.

Public fields

`id` Fabric session or high-concurrency acquisition ID

`url` Session lifecycle URL

`state` Latest service state

`response` Latest raw service response

`closed` Whether `$close()` completed

`high_concurrency` Whether this is a high-concurrency session

`session_id` Underlying Livy session ID for HC sessions

`repl_id` Isolated REPL ID for HC sessions

`verbose` Whether lifecycle messages are enabled

Methods

Public methods:

- `FabricLivySession$new()`
- `FabricLivySession$print()`
- `FabricLivySession$status()`
- `FabricLivySession$wait()`
- `FabricLivySession$submit()`
- `FabricLivySession$run()`
- `FabricLivySession$statements()`
- `FabricLivySession$reset_timeout()`
- `FabricLivySession$close()`

`FabricLivySession$new()`: Internal constructor used by `fabric_livy_session()`

Usage:

```
FabricLivySession$new(
  livy_url,
  credential,
  payload,
  response = NULL,
  high_concurrency = FALSE,
  verbose = TRUE
)
```

Arguments:

`livy_url` Livy API base or collection URL

`credential` Internal authentication credential

`payload` Session creation request body

`response` Existing session response when attaching

`high_concurrency` Whether to acquire an HC session

`verbose` Whether to emit lifecycle messages

Returns: A new session object

`FabricLivySession$print()`: Print a concise session summary

Usage:

```
FabricLivySession$print(...)
```

Arguments:

`...` Unused

Returns: `self`, invisibly

`FabricLivySession$status()`: Return the latest session response

Usage:

```
FabricLivySession$status(refresh = TRUE, deadline = NULL)
```

Arguments:

`refresh` Whether to retrieve current state from Fabric

`deadline` Internal wall-clock deadline for the status request

Returns: The raw session response list

`FabricLivySession$wait()`: Wait until the session can accept statements

Usage:

```
FabricLivySession$wait(timeout = 600, poll_interval = 3)
```

Arguments:

`timeout` Maximum wait in seconds

`poll_interval` Polling interval in seconds

Returns: `self`, invisibly

`FabricLivySession$submit()`: Submit code without waiting for completion

Usage:

```
FabricLivySession$submit(
  code,
  kind = c("spark", "pyspark", "sparkr", "sql"),
  source_id = NULL
)
```

Arguments:

`code` One string of Spark code

`kind` Statement language

`source_id` Optional caller-defined source identifier

Returns: A FabricLivyStatement

`FabricLivySession$run()`: Submit code, wait, and return its parsed result

Usage:

```
FabricLivySession$run(
  code,
  kind = c("spark", "pyspark", "sparkr", "sql"),
  source_id = NULL,
  timeout = 600,
  poll_interval = 2
)
```

Arguments:

`code` One string of Spark code

`kind` Statement language

`source_id` Optional caller-defined source identifier

`timeout` Maximum wait in seconds

`poll_interval` Polling interval in seconds

Returns: A `fabric_livy_statement_result` list

`FabricLivySession$statements()`: List every statement in this execution context

Usage:

```
FabricLivySession$statements()
```

Returns: The raw Livy statements response

`FabricLivySession$reset_timeout()`: Reset a regular session's inactivity timeout

Usage:

```
FabricLivySession$reset_timeout()
```

Returns: `self`, invisibly

`FabricLivySession$close()`: Release this session or high-concurrency context

Usage:

```
FabricLivySession$close(deadline = NULL)
```

Arguments:

`deadline` Internal wall-clock deadline for the cleanup request

Returns: `TRUE` when closed or `FALSE` when already closed, invisibly

Examples

```
## Not run:
# fabric_livy_session() creates this class for a discovered Lakehouse
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
session <- fabric_livy_session(lakehouse)
inherits(session, "FabricLivySession")

# Wait before running code, and close the Spark session when finished
session$wait()
session$run("print(1 + 1)", kind = "pyspark")
session$close()

## End(Not run)
```

`FabricLivyStatement` *A statement submitted to a Fabric Livy session*

Description

Represents one piece of code submitted to a FabricLivySession. Call `$wait()` and then `$result()` to retrieve its output. For the usual submit-and-wait workflow, use the session's `$run()` method instead. These lifecycle methods do not have separate free-function wrappers

Format

An R6::R6Class generator

Value

The `FabricLivyStatement` 'R6' generator.

Public fields

`id` Numeric Livy statement ID

`url` Statement lifecycle URL

`state` Latest statement state

`response` Latest raw service response

`started_local` Local submission timestamp

`completed_local` Local completion timestamp

`verbose` Whether lifecycle messages are enabled

Methods

Public methods:

- `FabricLivyStatement$new()`
- `FabricLivyStatement$print()`
- `FabricLivyStatement$status()`
- `FabricLivyStatement$wait()`
- `FabricLivyStatement$result()`
- `FabricLivyStatement$cancel()`

`FabricLivyStatement$new()`**:** Internal constructor used by `FabricLivySession$submit()`

Usage:

```
FabricLivyStatement$new(session, response, url, credential, verbose = TRUE)
```

Arguments:

`session` Parent FabricLivySession

`response` Initial statement response

`url` Statement lifecycle URL

`credential` Internal authentication credential

`verbose` Whether to emit lifecycle messages

Returns: A new statement object

`FabricLivyStatement$print()`: Print a concise statement summary

Usage:

```
FabricLivyStatement$print(...)
```

Arguments:

`...` Unused

Returns: `self`, invisibly

`FabricLivyStatement$status()`: Retrieve statement state and available output

Usage:

```
FabricLivyStatement$status(
  refresh = TRUE,
  from = NULL,
  size = NULL,
  deadline = NULL
)
```

Arguments:

`refresh` Whether to retrieve current state from Fabric

`from` Optional zero-based byte offset for returned statement output

`size` Optional maximum number of output bytes to return

`deadline` Internal wall-clock deadline for the status request

Returns: The raw statement response list

`FabricLivyStatement$wait()`: Wait for the statement to reach a terminal state

Usage:

```
FabricLivyStatement$wait(
  timeout = 600,
  poll_interval = 2,
  error_on_failure = TRUE
)
```

Arguments:

`timeout` Maximum wait in seconds

`poll_interval` Polling interval in seconds

`error_on_failure` Raise a structured error for failed statements

Returns: `self`, invisibly

`FabricLivyStatement$result()`: Return parsed output and timing metadata

Usage:

```
FabricLivyStatement$result(
  refresh = TRUE,
  error_on_failure = TRUE,
  from = NULL,
  size = NULL
)
```

Arguments:

`refresh` Whether to retrieve current state from Fabric

`error_on_failure` Raise a structured error for failed statements

`from` Optional zero-based byte offset for returned statement output

`size` Optional maximum number of output bytes to return

Returns: A `fabric_livy_statement_result` list

`FabricLivyStatement$cancel()`**:** Request cancellation of this statement

Usage:

```
FabricLivyStatement$cancel()
```

Returns: The raw cancellation response, invisibly

Examples

```
## Not run:
# Statements are returned by a session; users do not construct them directly
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
session <- fabric_livy_session(lakehouse)
session$wait()

# Submit code, wait for it, and inspect its result
statement <- session$submit("print(40 + 2)", kind = "pyspark")
inherits(statement, "FabricLivyStatement")
statement$wait()
statement$result()
session$close()

## End(Not run)
```

`fabric_catalog_search` *Search the OneLake catalog*

Description

Searches Fabric item metadata across every workspace visible to the caller. Results are lightweight R6 discovery objects that contain the item and workspace identity. Type-specific results expose the methods that can run from that metadata.

Usage

```
fabric_catalog_search(
  search = NULL,
  types = NULL,
  filter = NULL,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  output = c("r6", "list")
)
```

Arguments

search	Optional non-empty text query. Fabric searches display names, workspace display names, and descriptions. Leave NULL to browse visible catalog entries without a text query.
types	Optional unique vector of Fabric item types. This is converted to the catalog API's documented Type eq ... or Type eq ... filter.
filter	Optional raw catalog filter string. Fabric currently supports Type, eq, ne, or, and parentheses. Supply either types or filter, not both.
page_size	Optional number of results requested per page, from 1 to 1000. Leave NULL to use Fabric's service default.
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
api_base	Fabric REST API base URL. Leave unchanged unless using a different Fabric cloud or a test service
output	Discovery record representation. The default "r6" returns R6 objects with type-specific methods. Use "list" when a plain record is specifically required

Details

Catalog search is a preview Fabric API. It is for metadata discovery only and does not grant access to item contents. The caller needs Catalog.Read.All; Fabric returns only entries that the calling user, service principal, or managed identity is authorized to see.

Pagination sends the search parameters only on the first request and sends only the continuation token on later requests, as required by Fabric. A repeated or malformed token raises a fabric_catalog_protocol_error instead of silently returning partial results or looping indefinitely.

Value

With `output = "r6"`, a list of FabricItem objects or type-specific subclasses. With `output = "list"`, a list of `fabric_catalog_entry` records that also inherit from `fabric_item`. Both representations preserve the fields returned by Fabric and add the item workspace identity from the catalog hierarchy. Workspace entries returned by unfiltered browsing become FabricWorkspace objects, or `fabric_catalog_entry` records inheriting `fabric_workspace`.

References

Catalog search REST API

OneLake Catalog REST API overview

Examples

```
## Not run:
# Search all visible workspaces for Lakehouses related to sales
entries <- fabric_catalog_search(
  search = "sales",
  types = "Lakehouse",
  page_size = 100
)

# `$onelake_list()` calls fabric_onelake_list()
lakehouse <- entries[[1L]]
lakehouse$onelake_list(path = "Tables")

## End(Not run)
```

---

fabric_delta_config *Inspect the optional Python Delta runtime*

Description

Shows whether the optional Python tools used for direct Delta reads are ready By default this does not start Python. Set `initialize = TRUE` to prepare the environment, check the minimum Python version and required module imports, and report installed versions. An unusable runtime raises an error with setup instructions. Use `initialize = FALSE` to inspect it without this check

Usage

```
fabric_delta_config(initialize = FALSE)
```

Arguments

`initialize`	Whether to initialize Python

Value

A list describing initialization state, requirements, the selected interpreter, module availability, and installed package versions when initialized. `initialized` only indicates whether Python has started; it does not mean the Delta dependencies are available

Use a managed environment (recommended)

For automatic installation, explicitly select a 'reticulate'-managed environment. First restart R (in RStudio: Session > Restart R), then run:

```
Sys.setenv(RETICULATE_PYTHON = "managed")
library(fabricQueryR)
fabric_delta_config(initialize = TRUE)
```

`"managed"` is a special setting, not an environment name or a Python path. The first line tells 'reticulate' to create or reuse a suitable environment. Loading 'fabricQueryR' declares the required Python version and packages through `reticulate::py_require()`. The final line starts Python, triggering 'uv' to download the runtime and dependencies if needed, and checks setup. 'reticulate' also downloads 'uv' if needed; no separate `py_install()` call is necessary

The restart is required if Python has already started: setting `RETICULATE_PYTHON` cannot switch the interpreter in a running Python session.

To keep this selection for future sessions in this project, add the line `RETICULATE_PYTHON=managed` to the project's `.Renviron` file, then restart R. Otherwise, repeat the `Sys.setenv()` line at the start of each R session

Without this explicit selection, a Python chosen through RStudio, environment variables, `reticulate::use_python()`, or a project virtualenv can take precedence over the managed environment. `py_require()` declares requirements but does not install them into that existing Python, and `initialize = TRUE` does not change that choice. For example, selecting Python 3.9 leaves the Delta requirements unmet even though Python has started. `reticulate::py_config()` reports the selected interpreter and why it was chosen. After the setup above succeeds, both entries in `available` should be `TRUE` and `versions` should report the installed Delta packages

Create a reusable environment with reticulate

For an environment you manage yourself, restart R and create a virtualenv from an installed Python 3.11. Then install the required packages into that named environment and select its interpreter before initializing Python:

```
reticulate::virtualenv_create("fabricQueryR", version = "3.11")
reticulate::py_install(
  c("deltalake==1.6.2", "nanoarrow==0.8.0"),
  envname = "fabricQueryR",
  method = "virtualenv"
)
Sys.setenv(
  RETICULATE_PYTHON = reticulate::virtualenv_python("fabricQueryR")
)
```

```
library(fabricQueryR)
fabric_delta_config(initialize = TRUE)
```

virtualenv_create() needs a compatible installed Python and reuses an existing environment of that name without upgrading its Python. If needed, install Python first with reticulate::install_python("3.11:latest" or use the 'uv' alternative below. install_python() uses 'pyenv' / 'pyenv-win'; it is separate from the automatic 'uv' setup described above

py_install() installs into the named virtualenv using 'pip'. Always pass envname to make the destination explicit. For an existing Conda environment, use method = "conda", pip = TRUE with that environment's name instead

In later R sessions, repeat the Sys.setenv() selection before using Python; installation is needed only when creating or updating the environment. reticulate::use_virtualenv() is another way to select it, but an existing RETICULATE_PYTHON setting takes precedence over that selection

Create a reusable environment with uv

If the 'uv' command-line tool is already installed and available on PATH, it can create a new project environment and download Python if necessary. From the project directory, run in a terminal:

```
uv venv --python 3.11 --seed .venv-fabricQueryR
```

--seed installs 'pip' so that reticulate::py_install() can install into the environment. In a fresh R session in the same project directory:

```
reticulate::py_install(
  c("numpy", "deltalake==1.6.2", "nanoarrow==0.8.0"),
  envname = "./.venv-fabricQueryR",
  method = "virtualenv"
)
Sys.setenv(
  RETICULATE_PYTHON = reticulate::virtualenv_python("./.venv-fabricQueryR")
)
library(fabricQueryR)
fabric_delta_config(initialize = TRUE)
```

The ./ makes this a project path rather than a named environment under the virtualenv directory used by 'reticulate'. 'numpy' is included because 'uv' does not install it automatically

This environment is managed by you; 'reticulate' does not resolve py_require() declarations into it. Use py_install() with that explicit envname, or uv pip install --python .venv-fabricQueryR ..., to update it. See the uv environment guide for details

Install into an existing standalone Python

Select Python 3.10 or newer before initialization. Install the Python dependencies into that exact interpreter, for example from R:

```
system2("/path/to/python", c(
  "-m", "pip", "install", "deltalake==1.6.2", "nanoarrow==0.8.0"
))
```

On Windows, use the full path to `python.exe` with forward slashes. Restart R, select that interpreter with `reticulate::use_python()`, then run `fabric_delta_config(initialize = TRUE)` to verify setup

Examples

```
# Inspect requirements without starting Python or downloading anything
config <- fabric_delta_config()
config[c("initialized", "requirements", "available")]
```

`fabric_function_invoke`

Invoke a published Fabric user data function

Description

[Experimental]

Usage

```
fabric_function_invoke(
  function_url,
  parameters = list(),
  timeout = 110,
  idempotent = FALSE,
  max_response_bytes = .fabric_function_response_limit,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL
)
```

Arguments

`function_url`	Complete public URL copied from the published function's properties in Fabric. A discovered UserDataFunction item is not sufficient because the item API does not return the public function URL.
`parameters`	Named list, data frame, or named atomic vector serialized as the JSON object supplied to the function. Use `list()` for a function with no parameters.
`timeout`	Positive client request timeout in seconds. Fabric currently limits execution through a public function endpoint to 100 seconds.
`idempotent`	Logical. Permit bounded retries after transient failures. Keep `FALSE` for functions whose side effects cannot safely be repeated.

`max_response_bytes`
: Positive whole-number client limit for the complete response body. The default is 32 MiB, slightly above Fabric's documented 30 MB function-output limit.

`tenant_id`
: Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`.

`client_id`
: Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, with the Azure CLI application ID as fallback.

`token`
: Optional access token or token-provider function. Leave `NULL` to let 'fabricQueryR' use its normal sign-in flow for a Microsoft Fabric host. A custom `function_url` requires an explicitly supplied token or provider so an automatically acquired Fabric credential is not forwarded to another host.

`auth_args`
: Additional sign-in options passed to `AzureAuth::get_azure_token()`.

`audience`
: OAuth audience/scope passed to the credential. `NULL` selects `UserDataFunction.Execute.All` for delegated sign-in or the Power BI `.default` audience for client credentials. To use Microsoft's broader alternative, pass `"https://analysis.windows.net/powerbi/api/Item.Ex` explicitly.

Details

Calls the public REST endpoint for one published Microsoft Fabric user data function and returns the service's synchronous execution result. Function definition, publication, and deployment are intentionally outside this helper's scope.

This invocation API is experimental because its request and response handling is covered by offline tests, but the package cannot currently maintain repeatable end-to-end coverage against published functions. The package's development sandbox uses a service principal, while Fabric's User Data Function create, update-definition, and delete APIs currently support delegated user identities only. The sandbox therefore cannot provision and maintain the published public-function fixtures needed for that coverage.

Value

A `fabric_function_result` list with `function_name`, `invocation_id`, `status`, `output`, `errors`, `http_status`, and `response`. Function `output` is not redacted because field names such as `token` can be legitimate domain data. Unsafe whole-number JSON values are exact character text; decimal JSON values use ordinary R doubles. The rest of `response` is redacted and retains unknown future fields. Inspect `status` and `errors`; receiving a result does not by itself mean the function succeeded.

Before you invoke

Publish the user data functions item, switch it to **Run only** mode, enable **Public access** for the function, and copy its **Public URL** from the Fabric portal. Pass that complete URL to `function_url`; item discovery does not currently expose enough information to derive a public function URL safely.

Parameter names and values must match the published Python signature. Fabric supports JSON strings, ISO 8601 datetime strings, booleans, numbers, arrays, and objects as inputs. The top-level `parameters` object therefore needs unique, non-empty camelCase names without underscores. Python keywords and Fabric's reserved `req`, `context`, and `reqInvocationId` names are rejected

before a request is sent. A named atomic vector is converted to a named list; use `I()` around a one-element value when it must remain a JSON array. Supply datetimes as ISO 8601 strings with an explicit timezone and the desired fractional seconds. R `POSIXct` and `POSIXlt` objects are rejected, including inside lists and data frames, to avoid lossy JSON conversion.

Permissions and authentication

Delegated authentication defaults to the narrower Power BI permission `UserDataFunction.Execute.All`. Microsoft also accepts the broader `Item.Execute.All` permission; use it only when the app registration grants that scope, and pass its full scope URL explicitly through `audience`. Either delegated scope still requires Execute permission on the user data functions item. Service-to-service callers can use an application credential with the Power BI `.default` audience and the required tenant and item access.

Application authentication for the public invocation endpoint is distinct from authentication used by connections inside the function. Microsoft currently does not support using a service principal through connections managed by user data functions to access Fabric items or data sources. A service principal can therefore invoke a compatible function while a function that relies on an unsupported managed connection can still fail.

The function URL is a credential boundary. Tokens are sent to the explicitly supplied HTTPS endpoint. URLs containing credentials, query parameters, fragments, or nonstandard ports are rejected. HTTPS and route validation do not prove hostname ownership or token audience. Use a custom host only when your organization controls it, with a token or provider issued for that host's intended audience.

Results, retries, and limits

Fabric reports `Succeeded`, `BadRequest`, `Failed`, `Timeout`, and `ResponseTooLarge` through one response envelope. Valid envelopes remain inspectable as `fabric_function_result` objects even when Fabric uses a non-success HTTP status. Authentication, authorization, throttling, and malformed service responses continue to raise the package's typed HTTP or response errors. The documentation describes an error `name`, while current responses can use `errorCode`; `errors` adds `name` as an alias when needed and `response` retains the original service shape.

Invocations are not retried by default because functions can have arbitrary side effects. Set `idempotent = TRUE` only when repeating the function is safe; this enables the package's bounded retries for transport failures, throttling, and transient HTTP responses.

Fabric limits public-endpoint execution to 100 seconds, request parameters to 4 MB, and a function's return value to 30 MB. The default 110-second client timeout allows the service timeout response to arrive. The 32 MiB client response cap leaves room for Fabric's envelope around a 30 MB output. Secret-named fields and bearer-token text are redacted recursively from errors, response metadata, and conditions. Function `output` is domain data and is not redacted, even when it contains secret-like field names. Unsafe whole-number JSON values are returned as exact character text; decimal JSON values use ordinary R doubles.

References

Invoke user data functions from a Python application

Fabric user data functions service limits

Fabric user data function programming model

Examples

```
## Not run:
# Discover the user data functions item that owns the published function
workspace <- fabric_workspaces()[[1L]]
function_item <- fabric_user_data_functions(workspace)[[1L]]
function_item$displayName

# Discovery cannot expose a function URL yet. Copy the published function's
# complete Invoke URL from this item's Run-only settings into this variable
function_url <- Sys.getenv("FABRIC_FUNCTION_URL")

# Parameter names must match the published Python function signature
result <- fabric_function_invoke(
  function_url,
  parameters = list(
    customerName = "Ada",
    order = list(id = 42L, lines = I(c("A", "B")))
  )
)

# Inspect the output and any function-level errors returned by Fabric
result$status
result$output
result$errors

## End(Not run)
```

---

fabric_graphql_collect

Collect paged GraphQL row objects into a tibble

Description

Combines row objects from a caller-selected field in every result returned by `fabric_graphql_paginate()`. The explicit `path` is relative to each page's `data` field because GraphQL response shapes are schema-defined and cannot be inferred safely

Usage

```
fabric_graphql_collect(pages, path)
```

Arguments

`pages`	A `fabric_graphql_pages` result from `fabric_graphql_paginate()`. Requiring that result, rather than an unverified single response, lets the function enforce completion
`path`	Character path from each page's `data` field to the list of row objects, for example `c("viewer", "products", "items")`

Details

Scalar fields become ordinary tibble columns. Nested objects and arrays stay as list-columns and are never flattened. Fields introduced on later pages are added in first-seen order, with missing or GraphQL `null` scalar values represented by typed `NA` values when their type can be inferred. Exact integer strings returned by `fabric_graphql_query()` remain character data; finite numeric entries in the same field, including fractions, are promoted to character using text that recovers the received R value exactly. Fields containing only numeric values retain ordinary numeric columns. Nullable row elements retain their positions as missing rows; the `null_rows` attribute records their one-based positions, distinguishing them from objects whose fields are all null.

A successful result has class `fabric_graphql_rows` and reports completion, page count, path, and GraphQL errors in its printed header and attributes. Use `attr(rows, "errors")` to inspect partial GraphQL errors. If pagination stopped before `next_cursor` reported completion, including at `max_pages`, the function raises `fabric_graphql_collection_error`; its `partial_data` field contains the rows collected so far and is explicitly marked incomplete

Value

A `fabric_graphql_rows` tibble. Attributes `complete`, `errors`, `page_count`, `path`, and `null_rows` retain collection metadata

References

Fabric API for GraphQL limits

Fabric GraphQL aggregation and pagination shape

Examples

```
## Not run:
# Discover the GraphQL API, then fetch every Products page
workspace <- fabric_workspaces()[[1L]]
api <- fabric_graphql_apis(workspace)[[1L]]
pages <- fabric_graphql_paginate(
  api,
  query = paste(
    "query Products($first: Int!, $after: String) {",
    "  products(first: $first, after: $after) {",
    "    items { id name details { category } }",
    "    hasNextPage endCursor",
    "  }",
    "}"
  ),
  variables = list(first = 100L, after = NULL),
  next_cursor = fabric_graphql_cursor("products")
)

# Combine nested item rows from every page into one tibble
rows <- fabric_graphql_collect(pages, c("products", "items"))
attr(rows, "complete")
attr(rows, "errors")
```

```
## End(Not run)
```

`fabric_graphql_cursor` *Locate pagination information in a GraphQL result*

Description

Creates the `next_cursor` function used by `fabric_graphql_paginate()` for the common `hasNextPage` and `endCursor` pagination fields

Usage

```
fabric_graphql_cursor(path, has_next = "hasNextPage", end_cursor = "endCursor")
```

Arguments

`path`	Character path from the result's `data` field to a connection object, for example `"products"` or `c("viewer", "products")`. This is the parent object that contains the pagination fields, not the `items` field
`has_next`	Name of the logical connection field indicating another page. Fabric commonly uses `"hasNextPage"`
`end_cursor`	Name of the connection field containing the opaque cursor Fabric commonly uses `"endCursor"`

Value

A function suitable for `next_cursor` in `fabric_graphql_paginate()`. For each page it returns the cursor when `has_next` is true, otherwise `NULL`

Examples

```
# Build a reusable extractor for a GraphQL connection named "products"
next_cursor <- fabric_graphql_cursor("products")

# This small local result shows the response shape expected by the extractor
page <- structure(
  list(data = list(products = list(
    hasNextPage = TRUE,
    endCursor = "opaque-cursor"
  ))),
  class = c("fabric_graphql_result", "list")
)

# TRUE plus a cursor tells the paginator to request another page
next_cursor(page)
```

fabric_graphql_paginate

Read all pages from a Fabric GraphQL query

Description

Repeats fabric_graphql_query() until the API reports that no more pages are available. Because every GraphQL schema can store pagination information in a different place, next_cursor tells the function where to find it

Usage

```
fabric_graphql_paginate(
  api,
  query,
  next_cursor,
  variables = list(),
  cursor_variable = "after",
  operation_name = NULL,
  workspace_id = NULL,
  error_policy = c("return", "warn", "error"),
  max_pages = 100L,
  timeout = 110,
  idempotent = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  api_base = .fabric_api_base,
  numeric_policy = c("exact", "double")
)
```

Arguments

api	GraphQL endpoint, API ID, or one discovered GraphQLApi object. An item from fabric_graphql_apis() is usually easiest because it supplies the endpoint and workspace ID
query	One GraphQL document containing a query or mutation. Use variables for changing values instead of pasting values into this string
next_cursor	Function accepting a fabric_graphql_result and returning the next opaque cursor, or NULL when pagination is complete. Use fabric_graphql_cursor() for Fabric's normal connection fields

variables	Named list of values for variables declared in query. Numeric and other R missing values are sent as JSON null; numeric NaN and infinities are rejected because GraphQL JSON has no such numbers. Supply date-times as explicit ISO 8601 strings with a UTC Z or offset, for example "2024-02-29T12:34:56.123456Z". POSIXct and POSIXlt values are rejected, including inside nested inputs, because implicit JSON conversion can discard their time zone and fractional seconds. One-element values are normally sent as scalars. Wrap a one-element list variable in I(), for example list(ids = I("x")), to send it as an array
cursor_variable	Name of the GraphQL variable that receives the next cursor, commonly "after". It must match the variable declared in query
operation_name	Optional operation name. Supply it when the document contains more than one named operation; otherwise leave NULL
workspace_id	Workspace GUID. Required when api is a GraphQL API GUID, and otherwise inferred from a discovered object
error_policy	How GraphQL-level errors are handled. "return" lets the caller inspect partial data and errors; "warn" also makes errors visible immediately; "error" stops and attaches the result to a fabric_graphql_error. HTTP/authentication failures always stop
max_pages	Positive maximum number of requests. This guards against a faulty or unexpectedly large pagination loop
timeout	Maximum time in seconds for the request. The default allows Fabric's own 100-second query timeout response to arrive
idempotent	Logical. Permit retries after transient HTTP failures TRUE is normally suitable for a read-only query, but not for a mutation that could be applied twice
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, with the Azure CLI application ID as fallback
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow for a Microsoft Fabric host. A custom API endpoint, including an API Management gateway, requires an explicitly supplied token or provider so an automatically acquired Fabric credential is not forwarded to another host
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
audience	OAuth audience/scope passed to the credential. NULL selects the documented scope from the authentication flow. Set this only for a custom token provider or unusual identity flow
api_base	Fabric REST API base URL used to derive endpoints from IDs Most users should keep the default
numeric_policy	Numeric response policy. "exact" preserves decimal and exponent JSON numbers in GraphQL data as character source text; "double" decodes them as ordinary R doubles and can lose precision or lexical scale. Whole-number handling is unchanged

Details

Failures while paging raise `fabric_graphql_pagination_error`, retaining incomplete `pages`, the failing `result` (when available), request `variables`, `cursor`, `seen_cursors`, `page_number`, and the original condition as `parent`. Error-only GraphQL responses cannot continue pagination, even with `error_policy = "return"`. Partial data with usable cursors remains supported.

Value

A `fabric_graphql_pages` list with `pages`, combined `errors`, and the final `variables`. `pages` contains one `fabric_graphql_result` per request and `complete` is `TRUE` when the callback reported no next page Results are kept page-by-page because the requested schema shape can vary

Examples

```
## Not run:
# Discover the GraphQL API that exposes the Products query
workspace <- fabric_workspaces()[[1L]]
api <- fabric_graphql_apis(workspace)[[1L]]

# Fetch pages until the helper sees no next cursor
pages <- fabric_graphql_paginate(
  api,
  query = paste(
    "query Products($first: Int!, $after: String) {",
    "  products(first: $first, after: $after) {",
    "    items { id name } hasNextPage endCursor",
    "  }",
    "}"
  ),
  variables = list(first = 100L, after = NULL),
  next_cursor = fabric_graphql_cursor("products")
)
pages$complete

## End(Not run)
```

---

fabric_graphql_query *Run a query against a Fabric GraphQL API*

Description

Sends a GraphQL query or mutation to an **API for GraphQL** item and returns the result as a nested R list. Use this when a Fabric API already exposes the Lakehouse, Warehouse, or SQL Database data you need

Usage

```
fabric_graphql_query(
  api,
  query,
  variables = list(),
  operation_name = NULL,
  workspace_id = NULL,
  error_policy = c("return", "warn", "error"),
  timeout = 110,
  idempotent = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  api_base = .fabric_api_base,
  numeric_policy = c("exact", "double")
)
```

Arguments

api	GraphQL endpoint, API ID, or one discovered GraphQLApi object. An item from fabric_graphql_apis() is usually easiest because it supplies the endpoint and workspace ID
query	One GraphQL document containing a query or mutation. Use variables for changing values instead of pasting values into this string
variables	Named list of values for variables declared in query. Numeric and other R missing values are sent as JSON null; numeric NaN and infinities are rejected because GraphQL JSON has no such numbers. Supply date-times as explicit ISO 8601 strings with a UTC Z or offset, for example "2024-02-29T12:34:56.123456Z". POSIXct and POSIXlt values are rejected, including inside nested inputs, because implicit JSON conversion can discard their time zone and fractional seconds. One-element values are normally sent as scalars. Wrap a one-element list variable in I(), for example list(ids = I("x")), to send it as an array
operation_name	Optional operation name. Supply it when the document contains more than one named operation; otherwise leave NULL
workspace_id	Workspace GUID. Required when api is a GraphQL API GUID, and otherwise inferred from a discovered object
error_policy	How GraphQL-level errors are handled. "return" lets the caller inspect partial data and errors; "warn" also makes errors visible immediately; "error" stops and attaches the result to a fabric_graphql_error. HTTP/authentication failures always stop
timeout	Maximum time in seconds for the request. The default allows Fabric's own 100-second query timeout response to arrive
idempotent	Logical. Permit retries after transient HTTP failures TRUE is normally suitable for a read-only query, but not for a mutation that could be applied twice

tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, with the Azure CLI application ID as fallback
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow for a Microsoft Fabric host. A custom API endpoint, including an API Management gateway, requires an explicitly supplied token or provider so an automatically acquired Fabric credential is not forwarded to another host
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
audience	OAuth audience/scope passed to the credential. NULL selects the documented scope from the authentication flow. Set this only for a custom token provider or unusual identity flow
api_base	Fabric REST API base URL used to derive endpoints from IDs Most users should keep the default
numeric_policy	Numeric response policy. "exact" preserves decimal and exponent JSON numbers in GraphQL data as character source text; "double" decodes them as ordinary R doubles and can lose precision or lexical scale. Whole-number handling is unchanged

Value

A fabric_graphql_result list with data, errors, extensions, and response (the complete parsed response). data follows the nested shape requested in the GraphQL document and is usually a combination of named lists and vectors, not a tibble. Because GraphQL can return partial data, inspect errors even when data is present

Before you query

Before using this function, create an **API for GraphQL** item in a Fabric workspace, connect its data source, and choose which tables, fields, queries, and mutations the API exposes. Fabric's built-in GraphQL editor and schema explorer are the easiest places to design and test a document before copying it to R

Mutation availability depends on the configured source. Fabric Warehouse and SQL Database sources can expose supported mutations, while Lakehouse and mirrored SQL analytics endpoint sources are read-only and expose queries only

The easiest input is an item from fabric_graphql_apis(). You can instead supply the API's endpoint, or its ID together with workspace_id

Permissions and authentication

Interactive authentication requires the Power BI delegated scope GraphQLApi.Execute.All, plus **Run Queries and Mutations** permission on the API. Service principals are also supported by Fabric: request a Fabric API token with auth_args or pass one through token, enable service principals for Fabric APIs in the tenant, and grant the principal API Execute access or a suitable workspace role. With SSO connectivity, the caller also needs the required access to the underlying data source Saved-credential APIs use the configured connection instead

Most users can leave `audience = NULL`; 'fabricQueryR' chooses the documented scope for the sign-in flow. Set it only for a custom identity provider. HTTPS and URL-shape validation do not prove hostname ownership or token audience. Use a custom API Management or gateway host only when your organization controls it, with a token or provider issued for that host's intended audience

Retries and service limits

GraphQL POST requests are not retried by default because a document can contain mutations. Set `idempotent = TRUE` only when the operation is safe to repeat

Fabric returns at most 100 items by default and permits at most 100,000 items across pagination. Each response is limited to 64 MB, each request to 100 seconds, and query nesting to 10 levels. Use smaller pages and filtered query partitions when a result could approach these service limits. One GraphQL API item can have at most 1,000 source objects attached across its data sources; this is not a limit of 1,000 data sources. Split objects from multiple sources across multiple API items, or use stored procedures or another abstraction for a single large source

Large integers outside R's exact numeric range are returned as character values so identifiers and other large integer fields are not rounded. By default, JSON numbers containing a decimal point or exponent are also returned as their exact source text, retaining precision, scale, trailing zeros, and exponent spelling. Set `numeric_policy = "double"` to decode those values as ordinary R doubles instead

References

Fabric API for GraphQL editor

Fabric GraphQL schema explorer

Use service principals with Fabric API for GraphQL

Fabric API for GraphQL limits

Examples

```
## Not run:
# Discover an API for GraphQL item instead of copying its endpoint or ID
workspace <- fabric_workspaces()[[1L]]
api <- fabric_graphql_apis(workspace)[[1L]]

# Keep the filter value in variables rather than inserting it into the query
result <- fabric_graphql_query(
  api,
  query = paste(
    "query Products($category: String!) {",
    "  products(filter: {category: {eq: $category}}) {",
    "    items { id name category }",
    "  }",
    "}"
  ),
  variables = list(category = "A"),
  operation_name = "Products"
)
```

```
# GraphQL can return data and errors in the same response; inspect both
result$data$products$items
result$errors

## End(Not run)
```

`fabric_graphql_schema` *Inspect a Fabric GraphQL schema*

Description

Runs the standard GraphQL introspection query against an **API for GraphQL** item. The returned schema retains the service's nested type references, fields, input values, enum values, and directives so callers can explore the API without assuming how Fabric named its generated objects

Usage

```
fabric_graphql_schema(
  api,
  workspace_id = NULL,
  timeout = 110,
  idempotent = TRUE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  api_base = .fabric_api_base,
  numeric_policy = c("exact", "double")
)
```

Arguments

`api`	GraphQL endpoint, API ID, or one discovered GraphQLApi object. An item from `fabric_graphql_apis()` is usually easiest because it supplies the endpoint and workspace ID
`workspace_id`	Workspace GUID. Required when `api` is a GraphQL API GUID, and otherwise inferred from a discovered object
`timeout`	Maximum time in seconds for the request. The default allows Fabric's own 100-second query timeout response to arrive
`idempotent`	Logical. Permit retries after transient HTTP failures `TRUE` is normally suitable for a read-only query, but not for a mutation that could be applied twice
`tenant_id`	Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`
`client_id`	Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, with the Azure CLI application ID as fallback

token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow for a Microsoft Fabric host. A custom API endpoint, including an API Management gateway, requires an explicitly supplied token or provider so an automatically acquired Fabric credential is not forwarded to another host
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
audience	OAuth audience/scope passed to the credential. NULL selects the documented scope from the authentication flow. Set this only for a custom token provider or unusual identity flow
api_base	Fabric REST API base URL used to derive endpoints from IDs Most users should keep the default
numeric_policy	Numeric response policy. "exact" preserves decimal and exponent JSON numbers in GraphQL data as character source text; "double" decodes them as ordinary R doubles and can lose precision or lexical scale. Whole-number handling is unchanged

Details

Microsoft Fabric disables runtime introspection by default. A workspace administrator must enable it under **API Settings > Introspection**. When it must remain disabled, use **Export schema** in the Fabric portal instead; schema export remains available independently of the runtime setting

Value

A fabric_graphql_schema list containing the standard __schema fields. The original GraphQL response and its (normally empty) errors are available in the response and errors attributes

References

Fabric API for GraphQL introspection and schema export

Examples

```
## Not run:
# Discover the GraphQL API whose schema you want to inspect
workspace <- fabric_workspaces()[[1L]]
api <- fabric_graphql_apis(workspace)[[1L]]

# Request the standard GraphQL introspection schema
schema <- fabric_graphql_schema(api)

# List its named types to learn what can be queried
vapply(schema$types, `[[`, character(1), "name")

## End(Not run)
```

fabric_item *Discover one Microsoft Fabric item*

Description

Finds one item and returns the connection details needed by 'fabricQueryR' Use this when you know the item's name or ID and do not need to list every item in the workspace

Usage

```
fabric_item(
  workspace,
  item,
  type = NULL,
  detail = NULL,
  detail_errors = c("abort", "record"),
  include = NULL,
  personal_workspace_tenant_id = NULL,
  personal_workspace_owner = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  output = c("r6", "list")
)
```

Arguments

workspace	Workspace name, ID, or object returned by fabric_workspaces(). A name is convenient for interactive use; an object avoids an extra lookup
item	Item GUID, exact display name, or an item object returned by a discovery function. A display name must identify exactly one item of the requested type; use a GUID or discovered object when names are duplicated
type	Optional Fabric API item type, for example "Lakehouse", "Warehouse", "SemanticModel", or "Notebook". Matching is done by Fabric, so use the API spelling. Leave NULL to list all item types
detail	Whether to retrieve connection details as well as names and IDs. This takes more requests and may require additional permissions. For fabric_item(), NULL enriches every supported type except User Data Functions, whose detail endpoint does not support application identities. The typed Semantic Model, GraphQL, and User Data Function helpers also default to lightweight records
detail_errors	What to do if some connection details cannot be read "record" returns the available information and stores an error message with the affected item; "abort" stops the call

include — Optional character vector of additional item properties to request. Fabric currently documents "DefaultIdentity"; values are sent as the API's comma-separated include query parameter

personal_workspace_tenant_id — Optional Microsoft Entra tenant ID used to build the XMLA endpoint for a Personal workspace

personal_workspace_owner — Optional owner UPN or Entra object ID used to build the XMLA endpoint for a Personal workspace. Microsoft Fabric's workspace API does not return either personal-workspace identifier, so supply this together with personal_workspace_tenant_id when a dax_connection_string is needed for a semantic model in My Workspace

tenant_id — Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID

client_id — Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID

token — Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow

auth_args — Additional sign-in options passed to AzureAuth::get_azure_token()

api_base — Fabric REST API base URL. When workspace is an object containing apiEndpoint, that workspace-specific endpoint is used unless api_base is supplied explicitly

output — Discovery record representation. The default "r6" returns R6 objects with type-specific methods. Use "list" when a plain record is specifically required

Details

GUID-based lookup requires read access to the item. A workspace GUID is used directly without requesting workspace details, so directly shared items do not require a workspace role. Name lookup requires permission to list the relevant workspaces or items. To reuse workspace-specific endpoints, pass a discovered workspace object; alternatively, supply api_base explicitly. Workload-specific enrichment additionally requires Item.Read.All/Item.ReadWrite.All or the applicable workload-specific read scope and access to the item. Microsoft currently limits User Data Function detail retrieval to delegated user identities, so its automatic default is lightweight. Set detail = TRUE explicitly when using a supported identity

Value

With output = "r6", a FabricItem object or type-specific subclass. With output = "list", one fabric_item record containing the item's name, ID, type, workspace, and available connection details

References

Get item REST API

Examples

```
## Not run:
# Discover a workspace and obtain a lightweight Warehouse object
```

```
workspace <- fabric_workspaces()[[1L]]
warehouses <- workspace$items(type = "Warehouse")

# Enrich that discovered object with connection details
warehouse <- fabric_item(workspace, warehouses[[1L]])

# `$sql_connection_info()` calls fabric_sql_connection_info()
warehouse$sql_connection_info()

## End(Not run)
```

`fabric_items` *Discover Microsoft Fabric items*

Description

Returns the Lakehouses, Warehouses, semantic models, notebooks, and other items stored in a workspace. Every item type returned by Fabric's core list API can be represented. Where the package has a specialized R6 subclass, its methods perform the matching query, connection, file, Spark, or job operations; other types remain complete generic FabricItem records

Usage

```
fabric_items(
  workspace,
  type = NULL,
  detail = FALSE,
  detail_errors = c("record", "abort"),
  recursive = TRUE,
  root_folder_id = NULL,
  include = NULL,
  personal_workspace_tenant_id = NULL,
  personal_workspace_owner = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  output = c("r6", "list")
)
```

Arguments

`workspace`	Workspace name, ID, or object returned by `fabric_workspaces()`. A name is convenient for interactive use; an object avoids an extra lookup

`type` Optional Fabric API item type, for example `"Lakehouse"`, `"Warehouse"`, `"SemanticModel"`, or `"Notebook"`. Matching is done by Fabric, so use the API spelling. Leave `NULL` to list all item types

`detail` Whether to retrieve connection details as well as names and IDs. This takes more requests and may require additional permissions. For `fabric_item()`, `NULL` enriches every supported type except User Data Functions, whose detail endpoint does not support application identities. The typed Semantic Model, GraphQL, and User Data Function helpers also default to lightweight records

`detail_errors` What to do if some connection details cannot be read `"record"` returns the available information and stores an error message with the affected item; `"abort"` stops the call

`recursive` Logical. `TRUE` includes items inside workspace folders; `FALSE` lists only items at the workspace root

`root_folder_id` Optional Fabric folder GUID used as the root of the listing. With `recursive = FALSE`, only direct children are returned; with `TRUE`, nested folders are included

`include` Optional character vector of additional item properties to request. Fabric currently documents `"DefaultIdentity"`; values are sent as the API's comma-separated `include` query parameter

`personal_workspace_tenant_id`
Optional Microsoft Entra tenant ID used to build the XMLA endpoint for a Personal workspace

`personal_workspace_owner`
Optional owner UPN or Entra object ID used to build the XMLA endpoint for a Personal workspace. Microsoft Fabric's workspace API does not return either personal-workspace identifier, so supply this together with `personal_workspace_tenant_id` when a `dax_connection_string` is needed for a semantic model in My Workspace

`tenant_id` Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`

`client_id` Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID

`token` Optional access token or token-provider function. Leave `NULL` to let 'fabricQueryR' use its normal sign-in flow

`auth_args` Additional sign-in options passed to `AzureAuth::get_azure_token()`

`api_base` Fabric REST API base URL. When `workspace` is an object containing `apiEndpoint`, that workspace-specific endpoint is used unless `api_base` is supplied explicitly

`output` Discovery record representation. The default `"r6"` returns R6 objects with type-specific methods. Use `"list"` when a plain record is specifically required

Details

The caller needs at least access to the workspace (the Viewer role is sufficient for the core list operation). Workload enrichment additionally requires `Item.Read.All`/`Item.ReadWrite.All` or the corresponding workload-specific read scope and access to the item Personal-workspace semantic models use Microsoft's v2 XMLA endpoint and require both `personal_workspace_tenant_id` and `personal_workspace_owner`

Value

A list with one item object per match. Every object includes common fields such as `id`, `displayName`, `type`, and `workspaceId`. With `output = "r6"`, results are FabricItem objects or type-specific subclasses. With `output = "list"`, results are `fabric_item` lists. With `detail = TRUE`, both representations include connection details when Fabric makes them available

Generic and typed discovery

`fabric_items()` and `workspace$items()` are the broad, future-compatible discovery interfaces. Their optional `type` filter is passed to Fabric, and item types without package-specific methods are returned as FabricItem objects with all service fields, `$details()`, and `$as_list()`.

The helpers documented in fabric_typed_items are an intentional convenience subset of Fabric's larger and evolving item catalog. A typed helper means that the package knows the item-type spelling and workload Get route; it does not necessarily mean that the result has its own R6 subclass. See fabric_typed_items for the exact support matrix

References

List items REST API

Fabric item management overview

Personal-workspace XMLA endpoints

Examples

```
## Not run:
# Start by discovering a workspace instead of copying its ID
workspaces <- fabric_workspaces()
workspace <- workspaces[[1L]]

# `$items()` is the object interface to fabric_items()
items <- workspace$items()
vapply(items, `[[`, character(1), "displayName")

# `$lakehouses()` calls fabric_lakehouses(); `$tables()` calls
# fabric_lakehouse_tables()
lakehouse <- workspace$lakehouses()[[1L]]
lakehouse$tables()

## End(Not run)
```

---

fabric_job_instances *Inspect Microsoft Fabric job history*

Description

Lists recent and active job instances for a Fabric item. All pages returned by Fabric are collected, and each result can be passed directly to `fabric_job_status()`, `fabric_job_wait()`, or `fabric_job_cancel()`.

Usage

```
fabric_job_instances(
  item,
  workspace = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)
```

Arguments

item	Item GUID, exact display name, or an item object returned by a discovery function. A discovered object is recommended because it includes the item type and workspace ID.
workspace	Workspace GUID, exact display name, or a workspace object. Omit it when item is a discovered object containing workspaceId.
item_type	Optional Fabric item type when item is a GUID. A discovered item supplies this automatically.
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow A fabric_job handle reuses its stored credential unless tenant_id, client_id, token, or non-empty auth_args is supplied explicitly
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token() when no token source is supplied
api_base	Fabric REST API base URL. Most users should keep the default A discovered workspace-specific endpoint is used unless this argument is supplied explicitly

Details

Reading history requires an item read permission. The returned records keep an in-process reference to the supplied credential so they can be refreshed, waited on, or cancelled. That credential is not retained when a record is serialized.

Value

A list of fabric_job_instance records. Fabric usually retains at most 100 recently completed instances per item, plus active instances. Unknown future status and invocation values are returned unchanged.

References

List item job instances

Examples

```
## Not run:
# Discover the Notebook whose run history you want to inspect
workspace <- fabric_workspaces()[[1L]]
notebook <- fabric_notebooks(workspace)[[1L]]

# List runs, then refresh one returned job record
history <- fabric_job_instances(notebook)
history[[1]]$status
fabric_job_status(history[[1]])

## End(Not run)
```

`fabric_job_run` *Run and monitor Microsoft Fabric item jobs*

Description

Start a Notebook, data pipeline, Spark job definition, or another supported Fabric item from R. The related functions check its progress, wait for it to finish, or request cancellation. Use Fabric's scheduler for recurring runs

Usage

```
fabric_job_run(
  item,
  workspace = NULL,
  job_type = NULL,
  item_type = NULL,
  parameters = NULL,
  parameter_types = NULL,
  execution_data = NULL,
  default_lakehouse = NULL,
  default_lakehouse_workspace = NULL,
  compute = NULL,
  session_tag = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  .sleep = Sys.sleep,
  .now = Sys.time
```

```
)

fabric_job_status(
  job = NULL,
  workspace = NULL,
  item = NULL,
  job_instance_id = NULL,
  item_type = NULL,
  job_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  respect_retry_after = TRUE,
  notebook_details = FALSE,
  .sleep = Sys.sleep,
  .now = Sys.time
)

fabric_job_wait(
  job,
  poll_interval = NULL,
  timeout = 600,
  error_on_failure = TRUE,
  cancel_on_timeout = FALSE,
  cancel = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  notebook_details = FALSE,
  .sleep = Sys.sleep,
  .now = Sys.time
)

fabric_job_cancel(
  job = NULL,
  workspace = NULL,
  item = NULL,
  job_instance_id = NULL,
  item_type = NULL,
  job_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
```

```
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)
```

Arguments

item	Item GUID, exact display name, or an item object returned by a discovery function. A discovered object is recommended because it already includes the item type and workspace ID
workspace	Workspace GUID, exact display name, or a discovered object Omit it when item is a discovered object containing workspaceId
job_type	Fabric job type. 'fabricQueryR' uses the current typed "Execute" operation for data pipelines and knows the usual values for notebooks and Spark job definitions, so normally omit this unless running another item type. Set job_type = "Pipeline" for a data pipeline only when explicitly using Fabric's legacy core job endpoint
item_type	Optional Fabric item type when item is a GUID. A discovered item supplies this automatically. Examples are "Notebook", "DataPipeline", and "SparkJobDefinition"
parameters	A named list of values to pass to the job, such as list(run_date = as.Date("2026-01-31"), full_load = FALSE), infers types from R and is appropriate for most runs. Names must match the parameters configured in Fabric. Advanced callers can instead supply records with name, value, and type. The typed DataPipeline Execute endpoint does not accept parameters. bit64::integer64 values infer Number and are accepted only when exactly representable as a double, since Fabric may interpret numeric parameters as floating point. This check also applies to explicit Number and Automatic types, which also require finite numeric values. Fabric's decimal parameter binder has a narrower range and scale than R doubles. Numeric parameters are rejected if parsing their JSON token through that binder would overflow or change the original double. For other exact integers or decimals, pass character values with type Text; for doubles, sprintf("%.17g", value) supplies reversible text. The receiving job must handle these values as text. Fabric normalizes numeric negative zero to zero; pass "-0.0" with type Text when its sign must be retained. R date-times must have whole-second precision. Fractional seconds are rejected because Fabric's DateTime format cannot preserve them; round or truncate explicitly before submission if that loss is acceptable.
parameter_types	Optional named character vector overriding inferred parameter types. Supported values are VariableReference, Integer, Number, Text, Boolean, DateTime, Guid, and Automatic. Use this only when R's inferred type is not the type expected in Fabric
execution_data	Optional advanced job settings in the format documented for the Fabric item type. Use the simpler arguments below for common notebook settings. In custom payload fields, wrap a one-element atomic vector in I() (or use an unnamed list) when it must remain a JSON array. The typed DataPipeline Execute endpoint does not accept a request body

default_lakehouse
: Optional Lakehouse GUID or discovered object used to set the notebook's default Lakehouse for this run. This changes the run context, not the notebook's saved default

default_lakehouse_workspace
: Optional workspace GUID or discovered record for default_lakehouse. When omitted, a discovered Lakehouse's workspace is used when available, otherwise the job workspace. An explicit workspace must match the workspace carried by a discovered Lakehouse

compute
: Notebook compute kind: "Spark", "Jupyter", or "DataWarehouse". Use "Spark" (the default) for Spark notebooks, "Jupyter" for a Jupyter runtime, and "DataWarehouse" for a notebook attached to Warehouse compute. It must match what the notebook code needs

session_tag
: Optional tag that enables Spark high-concurrency mode, so related notebook runs may reuse compute. See Details for its effect on failure reporting

tenant_id
: Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID

client_id
: Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID

token
: Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow A fabric_job handle reuses its stored credential unless tenant_id, client_id, token, or non-empty auth_args is supplied explicitly

auth_args
: Additional sign-in options passed to AzureAuth::get_azure_token() when no token source is supplied

api_base
: Fabric REST API base URL. Most users should keep the default A discovered workspace-specific endpoint is used unless this argument is supplied explicitly

.sleep, .now
: Internal hooks for deterministic tests

job
: A fabric_job returned by fabric_job_run() or a fabric_job_instance returned by a status, wait, or history function. Status and cancellation functions also accept a job instance GUID when workspace, item, and enough type information are supplied

job_instance_id
: Alternative argument for a job instance GUID. Do not supply it together with a handle, instance record, or GUID through job

respect_retry_after
: Whether to wait for Fabric's recommended first status-check time. Keep TRUE for normal use

notebook_details
: For Notebook jobs, whether to opt into the beta Notebook status endpoint for exit values and compute details. The default FALSE uses only the stable Core Job Scheduler status endpoint; exit_value and workload-specific properties may then be unavailable.

poll_interval
: Minimum seconds between status checks. NULL follows Fabric's recommendation, with a two-second fallback

timeout
: Maximum seconds to wait before raising a fabric_job_timeout

`error_on_failure`
: Whether failed, cancelled, or deduplicated jobs raise typed errors. Set to `FALSE` to inspect those terminal results directly

`cancel_on_timeout`
: Ask Fabric to cancel the job when the client-side timeout expires. `FALSE` stops waiting but leaves the Fabric job running

`cancel`
: Optional function checked between status updates. If it returns `TRUE`, 'fabricQueryR' requests cancellation. This can support an application's stop button

Value

`fabric_job_run()` returns a `fabric_job` handle for use with the other job functions `fabric_job_status()` and `fabric_job_wait()` return a `fabric_job_instance` record with status, times, failure information, and a notebook exit value when available. `fabric_job_cancel()` invisibly returns `TRUE` after Fabric accepts or confirms the cancellation

Typical workflow

Start a job with `fabric_job_run()`, then pass the returned handle to `fabric_job_wait()`. The handle keeps the workspace, item, job type, and sign-in context, so later calls do not need those details again Parameterized Core jobs can return a collection `Location` without an instance GUID. In that documented case, `fabric_job_run()` honors `Retry-After` and polls recent job history for one matching manual run. If the accepted instance cannot be resolved safely, it raises a `fabric_job_accepted_unresolved` condition rather than implying that the run request failed or replaying it

High-concurrency notebooks

A `session_tag` lets related notebook runs share Spark compute, but Fabric may report a failed statement as a completed shared session with no exit value Omit the tag when job status must reliably signal notebook failure. Otherwise, have the notebook report its outcome with `notebookutils.notebook.exit()`. The former `mssparkutils` namespace remains backward compatible but Microsoft recommends migrating because it will be retired

Notebook submission uses the released workload-specific route so Fabric applies per-run parameters and compute settings. Status and waiting use the stable Core endpoint by default. Set `notebook_details = TRUE` to opt into the beta Notebook status endpoint when exit values or workload-specific properties are required; the Core endpoint remains its fallback.

Permissions and status handling

Running and cancelling need an item execute permission. Checking or waiting also needs an item read permission, as does resolving a parameterized run's collection `Location`. For a parameterized Notebook, 'fabricQueryR' captures recent history before submission so a collection `Location` cannot be confused with an earlier run. Recovery requires response correlation with the job's root activity ID; history alone cannot establish ownership. Recovery stops with an accepted-but-unresolved error if correlation is absent or ambiguous. 'fabricQueryR' reconciles notebook status information from Fabric before returning it and stops with a typed error if Fabric reports an unfamiliar state instead of waiting indefinitely

References

Core Job Scheduler REST API

Run an on-demand item job

Run an on-demand notebook

Get a Notebook job instance (beta)

Manage and execute notebooks with public APIs

Fabric job scheduler

Examples

```
## Not run:
# Discover the workspace and Notebook that will be run
workspace <- fabric_workspaces()[[1L]]
notebook <- fabric_notebooks(workspace)[[1L]]

# Start the discovered Notebook and keep the returned job handle
job <- fabric_job_run(notebook)

# Refresh the current state without waiting for completion
current <- fabric_job_status(job)
current$status

# Opt into beta Notebook details only when an exit value is required
completed <- fabric_job_wait(
  job,
  timeout = 900,
  notebook_details = TRUE
)
completed$status
completed$exit_value

# A separate active run can be cancelled when it is no longer needed
job_to_cancel <- fabric_job_run(notebook)
fabric_job_cancel(job_to_cancel)

## End(Not run)
```

---

fabric_job_schedules *Manage Microsoft Fabric item schedules*

Description

List, create, update, or delete recurring schedules for a supported Fabric item. Use `fabric_job_schedule_config()` to construct the four schedule types in the current REST contract.

Usage

```
fabric_job_schedules(
  item,
  workspace = NULL,
  job_type = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_job_schedule_create(
  item,
  configuration,
  workspace = NULL,
  job_type = NULL,
  item_type = NULL,
  enabled = TRUE,
  execution_data = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_job_schedule_update(
  item,
  schedule_id,
  configuration = NULL,
  workspace = NULL,
  job_type = NULL,
  item_type = NULL,
  enabled = NULL,
  execution_data = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_job_schedule_delete(
```

```
  item,
  schedule_id,
  workspace = NULL,
  job_type = NULL,
  item_type = NULL,
  confirm = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)
```

Arguments

item	Item GUID, exact display name, or an item object returned by a discovery function. A discovered object is recommended because it includes the item type and workspace ID.
workspace	Workspace GUID, exact display name, or a workspace object. Omit it when item is a discovered object containing workspaceId.
job_type	Schedule job type. Notebooks default to "RunNotebook", Spark job definitions to "SparkJob", and data pipelines, Dataflows, and Data Build Tool Jobs to "Execute". For a Dataflow publish schedule, set job_type = "ApplyChanges" explicitly. Lakehouses require an explicit value because they do not have a safe generic default; use "RefreshMaterializedLakeViews" for the materialized Lake View refresh route and supply its documented execution_data. Microsoft labels the Lakehouse materialized Lake View schedule API as Preview, for evaluation and development only, and does not recommend it for production use. Unknown item types retain the Core Scheduler's "DefaultJob" fallback. Supply an explicit value for another workload-specific schedule job type. When passing one of these item types as a GUID instead of a discovered item, also supply item_type or set the documented job_type explicitly.
item_type	Optional Fabric item type when item is a GUID. A discovered item supplies this automatically.
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow A fabric_job handle reuses its stored credential unless tenant_id, client_id, token, or non-empty auth_args is supplied explicitly
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token() when no token source is supplied
api_base	Fabric REST API base URL. Most users should keep the default A discovered workspace-specific endpoint is used unless this argument is supplied explicitly

configuration	A value returned by fabric_job_schedule_config(). Advanced callers may pass a named list in the documented Fabric ScheduleConfig shape. Known types are validated; an unknown future type is passed through and remains inspectable.
enabled	Whether the schedule is enabled. Fabric can automatically disable schedules after repeated failures; updating one with enabled = TRUE explicitly restarts it.
execution_data	Optional named list of static, workload-specific execution data. Its schema is defined by the item's job type. The package preserves it without assuming that all workloads share one schema.
schedule_id	Schedule GUID, or a fabric_job_schedule record returned by a schedule function.
confirm	Must be explicitly set to TRUE before a schedule is deleted.

Details

Schedule deletion is not replayed after an ambiguous transport failure. An already absent schedule is treated as deleted.

List operations need an item read permission. Create and update require item execute and read-write permissions; delete requires item read-write permission. The current service limit is 20 schedules per item.

fabric_job_schedule_update() accepts partial R input for convenience, but the Fabric PATCH contract requires enabled and a complete configuration. When either is omitted, the function first reads the current schedule and preserves the omitted value. Omitted configuration and omitted or NULL execution_data are replayed from the original response JSON, retaining numeric precision and empty objects or arrays; supply a named list to replace either value. Decoded record fields use ordinary R JSON types and cannot represent arbitrary decimals.

The published REST response currently exposes enabled but no standard auto-disable reason. auto_disabled is therefore NA unless Fabric returns an explicit marker. The complete response stays available in raw.

Semantic-model refresh schedules use the Power BI dataset schedule API, not the Fabric Core Job Scheduler. These functions reject a discovered semantic model unless job_type is supplied explicitly for a future or custom route.

Value

fabric_job_schedules() returns a list of fabric_job_schedule records. Create and update return one such record. Delete invisibly returns TRUE. Records expose normalized common fields and retain the complete service response in raw.

References

Fabric Job Scheduler REST API

Update a semantic-model refresh schedule

Schedule a Data Pipeline

Schedule Dataflow Execute

Schedule Dataflow Apply Changes

Schedule a Lakehouse materialized Lake View refresh

Schedule a Data Build Tool Job

Fabric Data Pipeline REST API capabilities

Fabric job scheduler behavior

Examples

```
## Not run:
# Discover the Notebook instead of copying workspace and item IDs
workspace <- fabric_workspaces()[[1L]]
notebook <- fabric_notebooks(workspace)[[1L]]

# Inspect existing schedules before creating another one
existing <- fabric_job_schedules(notebook)

# Build a weekly configuration using Fabric's Windows time-zone name
configuration <- fabric_job_schedule_config(
  "Weekly",
  start_time = "2026-10-01T00:00:00Z",
  end_time = "2027-10-01T00:00:00Z",
  time_zone = "W. Europe Standard Time",
  times = "07:30",
  weekdays = c("Monday", "Thursday")
)

# Create, disable, and finally delete the schedule returned by Fabric
schedule <- fabric_job_schedule_create(notebook, configuration)
fabric_job_schedule_update(notebook, schedule, enabled = FALSE)
fabric_job_schedule_delete(notebook, schedule, confirm = TRUE)

## End(Not run)
```

---

fabric_job_schedule_config

Build a Microsoft Fabric job schedule configuration

Description

Creates a validated configuration for `fabric_job_schedule_create()` or `fabric_job_schedule_update()`. Recurrence clock times use the supplied Windows time-zone identifier, while schedule boundaries are sent to Fabric in UTC.

Usage

```
fabric_job_schedule_config(
  type = "Cron",
  start_time,
```

```
  end_time,
  time_zone = "UTC",
  interval = NULL,
  times = NULL,
  weekdays = NULL,
  recurrence = NULL,
  day_of_month = NULL,
  week_index = NULL,
  weekday = NULL
)
```

Arguments

`type`	Schedule type: `"Cron"` for a minute interval, `"Daily"`, `"Weekly"`, or `"Monthly"`.
`start_time`, `end_time`	A scalar `POSIXt` value or RFC 3339 string with an explicit `Z` or numeric offset. These boundaries are converted to UTC. Fractional seconds are rejected; round or truncate explicitly before use.
`time_zone`	Windows time-zone identifier used to interpret `times`, such as `"UTC"`, `"W. Europe Standard Time"`, or `"Central Standard Time"`. Fabric validates the identifier.
`interval`	For a `Cron` schedule, a whole-number interval in minutes from 1 through 5,270,400.
`times`	For daily, weekly, and monthly schedules, one or more local clock times in 24-hour `"HH:MM"` form. The REST contract permits up to 100.
`weekdays`	For a weekly schedule, one or more English weekday names.
`recurrence`	For a monthly schedule, the whole-number month interval from 1 through 12.
`day_of_month`	For a monthly schedule, a day from 1 through 31. Invalid dates in a particular month are skipped by Fabric. Supply this or the `week_index` and `weekday` pair.
`week_index`	For an ordinal monthly schedule, one of `"First"` through `"Fifth"`.
`weekday`	For an ordinal monthly schedule, one English weekday name.

Details

A Cron schedule is Fabric's minute-interval schedule; this function does not accept a cron expression because the REST API does not use one. Daylight saving behavior is controlled by Fabric using `time_zone`, not by the R process's local time zone. Arguments that do not belong to the selected documented schedule type are rejected.

Non-UTC firing remains unverified in the package's persistent Fabric sandbox: enabled Amsterdam schedules have not produced the expected run within the test window. The cause has not been established. Monthly recurrence and nonexistent or repeated local times at daylight-saving transitions also lack live execution evidence. Verify a scheduled run through `fabric_job_instances()` in the target workspace before relying on these configurations; successful schedule creation only confirms API acceptance.

Value

A named list using the Fabric `ScheduleConfig` JSON field names.

References

Create item schedule

Windows default time zones

Examples

```
# Describe a schedule in the Windows time zone used by Fabric
daily <- fabric_job_schedule_config(
  "Daily",
  start_time = "2026-10-01T00:00:00Z",
  end_time = "2027-10-01T00:00:00Z",
  time_zone = "W. Europe Standard Time",
  times = c("08:30", "17:00")
)
```

`fabric_kql_export` *Export a KQL query directly to external storage*

Description

Runs Kusto's server-side `.export to storage` command and waits for its asynchronous operation to finish. This avoids returning a large query result through R and the Kusto client-result channel. A discovered Fabric item plus a `Files/` directory is converted to a OneLake connection string using caller impersonation; a complete documented Kusto storage connection string can also be supplied.

Usage

```
fabric_kql_export(
  cluster,
  query,
  destination,
  database = NULL,
  workspace = NULL,
  path = NULL,
  item_type = NULL,
  format = c("parquet", "csv", "tsv", "json"),
  compressed = TRUE,
  include_headers = NULL,
  name_prefix = NULL,
  file_extension = NULL,
  encoding = NULL,
  compression_type = NULL,
  distribution = c("per_shard", "per_node", "single"),
  size_limit = 1e+08,
  parquet_row_group_size = NULL,
  parquet_datetime_precision = NULL,
  timeout = 900,
```

```
  poll_interval = 2,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  numeric_policy = c("exact", "service"),
  .sleep = Sys.sleep,
  .now = Sys.time
)
```

Arguments

cluster	Query URI, or one Eventhouse or KQLDatabase discovery object. A KQLDatabase object also supplies database.
query	One non-empty KQL query. The first result set is exported.
destination	A discovered Fabric item, item name or ID, complete OneLake path, or complete HTTPS/ABFSS/ADL connection string for writable Azure Blob, ADLS Gen1/Gen2, or Amazon S3 storage. Arbitrary HTTPS web resources are not writable export destinations. A character vector of complete paths distributes export work across multiple destinations. For an item, also supply path and optionally workspace.
database	KQL database display name. Omit for a discovered KQLDatabase.
workspace	Workspace containing an item supplied as destination. Omit when the discovered item contains its workspace ID.
path	Destination directory relative to the OneLake item. It must be below Files/. Omit when destination is a complete storage path.
item_type	Optional Fabric item type used to resolve a named item.
format	Storage artifact format.
compressed	Whether the artifacts use compression.
include_headers	For CSV/TSV, one of "none", "all", or "firstFile". NULL uses Kusto's default.
name_prefix	Optional prefix for generated artifact names.
file_extension	Optional artifact extension beginning with a dot.
encoding	For CSV/TSV/JSON text, "UTF8NoBOM" or "UTF8BOM".
compression_type	Optional compression codec. Non-Parquet exports use "gzip"; Parquet also supports "snappy", "lz4_raw", "brotli", and "zstd".
distribution	Kusto export distribution hint.
size_limit	Maximum uncompressed bytes per artifact, from 100 MB to 4 GB (100,000,000 to 4,000,000,000 bytes).
parquet_row_group_size	Optional positive Parquet row-group row count.

parquet_datetime_precision
: Optional "millisecond" or "microsecond" precision for Parquet datetime values. NULL omits the setting and uses Kusto's default of milliseconds, discarding finer precision. Set "microsecond" to retain six fractional digits. Neither setting preserves Kusto's seventh fractional digit (100-nanosecond ticks). For full source precision, project a text column in the KQL query with format_datetime(value, 'yyyy-MM-dd HH:mm:ss.fffffff') before exporting (Kusto datetimes are UTC). numeric_policy governs decimals, not datetime precision.

timeout
: Positive total client-side limit in seconds, shared by schema preflight, submission, status polling, and retrieval of artifact details.

poll_interval
: Positive seconds between operation status requests.

tenant_id
: Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID

client_id
: Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, with the Azure CLI application ID as fallback

token
: Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow

auth_args
: Additional sign-in options passed to AzureAuth::get_azure_token()

numeric_policy
: Decimal Parquet policy. "exact" refuses queries with decimal output before export. "service" explicitly accepts service conversion, which can change decimal values even on successful exports. This option does not change other export formats.

.sleep, .now
: Internal hooks for deterministic polling tests.

Value

A fabric_kql_export_result containing the operation state, redacted destination, artifact paths, per-artifact record counts, and aggregate record count.

Tracking and failure safety

The export submission is sent once and is never automatically replayed. The function polls .show operations until Kusto reports a terminal state, then calls .show operation ... details for the authoritative artifact paths and record counts. Kusto does not remove files written before a failed export, so a failure or timeout identifies the destination and operation ID but never reports partial files as a successful result. If Kusto has already reported Completed but the artifact-details request exhausts the client deadline, the resulting details-timeout condition records operation_completed = TRUE; it does not imply that the export is still running or failed.

Storage connection strings are emitted as obfuscated Kusto string literals and are redacted from returned objects and conditions. If a submission fails before its operation ID is received, inspect the destination and Kusto operation history before trying again.

Output properties

format supports Kusto's parquet, csv, tsv, and json exporters. compressed = TRUE enables the selected compression_type, or Kusto's default codec when it is omitted. size_limit is the uncompressed target size of each artifact and must be from 100 MB through 4 GB. Text header and encoding options, and Parquet row-group and datetime-precision options, are accepted only for their applicable formats.

Decimal Parquet safety

By default, Parquet exports first request the query's output schemas without changing its text. Any decimal output raises `fabric_kql_export_decimal_error` before export submission, including an empty decimal result. Kusto can silently replace large decimals with zero and truncate fractional digits when exporting to Parquet; `Completed` only confirms the operation completed. Failed or unrecognized schema responses also stop the export. The schema check does not lock a query's schema against changes between the check and export requests.

To retain decimal values, explicitly project them as strings in your query, for example `| project value_text=tostring(val`
The companion null flag is necessary because `tostring()` turns a null into an empty string. This preserves Kusto's decimal value text, including any canonicalization already applied by the service, rather than its original input precision or scale. Alternatively, explicitly select `numeric_policy = "service"` to accept Kusto's Parquet conversion. The policy is also forwarded by an Eventhouse or KQLDatabase item's `$export()` method. Other export formats retain their service-defined conversion behavior.

Permissions

The caller needs at least Kusto Database Viewer permission. OneLake caller impersonation additionally needs write access equivalent to Storage Blob Data Contributor on the destination.

References

Kusto export to storage

Kusto storage connection strings

Kusto management HTTP request

Show Kusto operations

Kusto request properties

Examples

```
## Not run:
# Discover both the source KQL database and destination Lakehouse
workspace <- fabric_workspaces()[[1L]]
database <- fabric_kql_databases(workspace)[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
table <- Sys.getenv("FABRIC_KQL_TABLE")
table_literal <- jsonlite::toJSON(table, auto_unbox = TRUE)

# Export a bounded query to a new folder in the discovered Lakehouse
exported <- fabric_kql_export(
  database,
  query = paste0("table(", table_literal, ") | take 10000"),
  destination = lakehouse,
  path = "Files/exports/events-weekly",
  format = "parquet",
  parquet_datetime_precision = "microsecond",
  name_prefix = "events"
)
exported$artifacts
```

```
## End(Not run)
```

`fabric_kql_ingest` *Submit and monitor tracked Eventhouse ingestion*

Description

Queue existing blob or OneLake files for ingestion into an existing KQL table, then inspect or wait for the tracked per-file result. These functions use Kusto's queued-ingestion REST API, which is currently in preview

Usage

```
fabric_kql_ingest(
  cluster,
  table,
  sources,
  database = NULL,
  format,
  source_ids = NULL,
  raw_sizes = NULL,
  mapping = NULL,
  tags = character(),
  ingest_if_not_exists = character(),
  ignore_first_record = FALSE,
  skip_batching = FALSE,
  delete_after_download = FALSE,
  creation_time = NULL,
  validation_policy = NULL,
  zip_pattern = NULL,
  timestamp = NULL,
  timeout = 60,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  .deadline = NULL,
  .now = Sys.time
)

fabric_kql_ingestion_status(
  ingestion,
  cluster = NULL,
  database = NULL,
  table = NULL,
```

```
  details = TRUE,
  wait = FALSE,
  timeout = 900,
  poll_interval = 2,
  error_on_failure = TRUE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  .sleep = Sys.sleep,
  .now = Sys.time,
  .deadline = NULL
)
```

Arguments

`cluster`	Ingestion URI, or one Eventhouse or KQLDatabase object from `fabric_eventhouses()`, `fabric_kql_databases()`, or `fabric_item()`. A KQLDatabase object also supplies `database`. Use the **Ingestion URI**, not the Query URI, for direct character input
`table`	One existing target KQL table name
`sources`	Existing blob or OneLake storage connection strings, a data frame of source metadata, or a list of source records. See Sources and storage access
`database`	Target KQL database display name. Omit it when `cluster` is a discovered KQLDatabase object
`format`	Kusto ingestion format. Supported file formats include `csv`, `json`, `multijson`, `parquet`, `avro`, `orc`, and the documented delimited text formats
`source_ids`	Optional GUID per character `sources` entry. Missing IDs are generated. Do not combine with structured source records
`raw_sizes`	Optional uncompressed byte size per character `sources` entry. Use `NA` for an unknown size. Do not combine with structured source records
`mapping`	Optional name of a predefined ingestion mapping whose kind matches `format`. Omit it to use Kusto's identity mapping derived from the existing table schema: ordered text formats map by column position, while JSON, Parquet, Avro, ORC, and W3CLOGFILE map case-sensitive field names
`tags`	Character vector of extent tags to attach
`ingest_if_not_exists`	Stable keys used for idempotent ingestion of one source. The service checks existing `ingest-by:` tags for these values. Cannot be combined with a multi-source request
`ignore_first_record`	Whether to skip the first record in every source, commonly used for CSV headers
`skip_batching`	Whether to bypass normal Kusto ingestion batching. This can reduce latency but should be reserved for latency-critical workloads

`delete_after_download`
: Whether Kusto may delete a source after it has downloaded it. The default preserves source data

`creation_time`
: Optional ISO 8601 extent creation time, `Date`, or `POSIXt`. A `Date` is sent as midnight UTC. Align historical values with the target merge policy lookback

`validation_policy`
: Optional JSON string or named list describing CSV validation behavior

`zip_pattern`
: Optional regular expression selecting files inside ZIP sources

`timestamp`
: Optional ISO 8601 request timestamp, `Date`, or `POSIXt`

`timeout`
: Positive client-side limit in seconds. For a wait, this bounds the complete polling operation; otherwise it bounds the status request

`tenant_id`
: Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`

`client_id`
: Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID

`token`
: Optional access token or token-provider function. Status calls reuse an in-process handle credential unless authentication is overridden

`auth_args`
: Additional sign-in options passed to `AzureAuth::get_azure_token()`

`.deadline`
: Internal absolute POSIX date-time used when a higher-level operation composes submission and status polling under one deadline

`ingestion`
: A `fabric_kql_ingestion` handle or a non-empty operation ID

`details`
: Whether status should include per-source detail records

`wait`
: Whether to poll until all expected sources are terminal

`poll_interval`
: Minimum seconds between status requests while waiting

`error_on_failure`
: Whether a terminal failed or canceled ingestion raises a typed error. Use `FALSE` to inspect the returned status

`.sleep`, `.now`
: Internal hooks for deterministic deadline and polling tests

Value

`fabric_kql_ingest()` returns a `fabric_kql_ingestion` handle with the operation ID and source IDs. `fabric_kql_ingestion_status()` returns a `fabric_kql_ingestion_status` record with normalized counts, state, UTC times, and an optional details tibble

Sources and storage access

`fabric_kql_ingest()` never uploads local data or serializes an R object. Every `sources` value must already identify a file in blob storage or OneLake, and `table` must already exist. Use `fabric_kql_write_table()` when the data is a data frame, tibble, or Arrow object; that function performs staging and can create the target with `create_if_missing = TRUE`.

`sources` can be a character vector of storage connection strings, a data frame with `url`, `source_id`, and optional `raw_size` columns, or a list of records with those fields. The camel-case service names `sourceId` and `rawSize` are also accepted. Character inputs use the parallel `source_ids` and `raw_sizes` arguments

Only existing `https://` or `abfss://` storage sources are accepted. Nonpublic sources must include a Kusto-supported authentication suffix or credential in the storage connection string. For example, append `;impersonate` to a OneLake URL when the caller has permission to read it

Source IDs are generated when omitted and are returned in the ingestion handle. They identify blobs in status details, but they are not by themselves an exactly-once guarantee

Delivery and idempotency

Queued ingestion has at-least-once delivery semantics. Submission is therefore not automatically replayed after throttling, network failure, or an ambiguous response. Retain the returned operation ID before starting unrelated work

For idempotent ingestion, submit one source per call and set `ingest_if_not_exists` to one or more stable keys for that source. The function also attaches the corresponding `ingest-by:` tags unless they are already present. A later submission with a matching key is observable in detailed status instead of silently duplicating a committed extent. The function rejects keys for multi-source requests because Kusto applies the shared properties to every source and ingests tagged sources independently. Idempotency checks can race when the same key is queued concurrently, so serialize submissions that share a key

Tracking and failures

`fabric_kql_ingestion_status()` accepts the handle returned by `fabric_kql_ingest()` or a raw operation ID plus the ingestion target. With `wait = FALSE`, it returns one snapshot. With `wait = TRUE`, it polls until every expected source is terminal or `timeout` is reached

The returned status distinguishes `Succeeded`, `PartiallySucceeded`, `Failed`, `Canceled`, `PartiallyCanceled`, and `InProgress`. Detailed blob failures retain `error_code`, `failure_status`, and `message`. Source URLs and raw service data are redacted so SAS tokens and embedded credentials are not retained in the result. Set `error_on_failure = FALSE` to inspect a terminal failure instead of receiving a typed condition carrying the same status in `last_status`. When a submission handle supplies the expected blob count, completion requires the documented status counts to match it exactly. Unknown nonzero status categories and impossible totals raise a protocol error rather than being misreported as successful completion

Limits and permissions

The preview REST API accepts at most 20 blobs per request and a maximum of 6 GB of uncompressed data. `raw_sizes` are validated and summed when all are known. Supplying sizes also avoids a metadata read by the ingestion service

The caller needs Kusto Table Ingestor permission on the target table and Database User access. Reading nonpublic source files additionally requires storage access through the authentication method in each storage connection string. `delete_after_download = TRUE` also requires delete permission and permanently removes successfully downloaded source blobs

References

Queued ingestion REST API (preview)

Queued ingestion status REST API (preview)

Supported ingestion formats

Storage connection strings

Data ingestion properties

Ingestion mappings and identity mapping

Examples

```
## Not run:
# Discover the KQL database and a Lakehouse containing staged CSV files
workspace <- fabric_workspaces()[[1L]]
database <- fabric_kql_databases(workspace)[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
files <- fabric_onelake_list(
  workspace,
  lakehouse,
  path = "Files/events"
)
csv_file <- files[grepl("[.]csv$", files$path), ][1L, ]

# Build the source URI from discovered IDs and the listed file path
source <- paste0(
  "https://onelake.dfs.fabric.microsoft.com/",
  workspace$id, "/", lakehouse$id, "/", csv_file$path[[1L]],
  ";impersonate"
)

# A named mapping is optional when the source matches the table schema
table <- Sys.getenv("FABRIC_KQL_TABLE")
mapping <- Sys.getenv("FABRIC_KQL_CSV_MAPPING", unset = "")

# Queue the file once using a stable ingest-if-not-exists key
ingestion <- fabric_kql_ingest(
  database,
  table = table,
  sources = source,
  format = "csv",
  mapping = if (nzchar(mapping)) mapping else NULL,
  ignore_first_record = TRUE,
  ingest_if_not_exists = paste0("file:", csv_file$path[[1L]])
)

# Wait for every submitted file to reach a terminal ingestion state
result <- fabric_kql_ingestion_status(
  ingestion,
  wait = TRUE,
  timeout = 900
)
result$state
result$details

## End(Not run)
```

`fabric_kql_query` *Run a KQL query in Microsoft Fabric*

Description

Runs a read-only query against a KQL database and returns the result as a tibble. KQL databases are commonly used for event, log, telemetry, and time-series data in a Fabric Eventhouse

Usage

```
fabric_kql_query(
  cluster,
  query,
  database = NULL,
  parameters = list(),
  request_properties = list(),
  timeout = 60,
  retain_raw_frames = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list()
)
```

Arguments

`cluster`	Query URI, or one Eventhouse or KQLDatabase object returned by `fabric_eventhouses()`, `fabric_kql_databases()`, or `fabric_item()`. A KQLDatabase object also supplies `database`. Despite the argument name, use Fabric's **Query URI** here
`query`	One non-empty, read-only KQL query, for example `"Events | where Severity == 'Error' | take 100"`
`database`	KQL database display name. Supply it with a copied Query URI or an Eventhouse object; omit it when `cluster` is a KQLDatabase object
`parameters`	Named list of values declared with `declare query_parameters(...)` in `query`
`request_properties`	Named list of Kusto client request options, such as `servertimeout = "2m"` or `notruncation = TRUE`. Most users can leave this empty; these are server-side Kusto controls, not query parameters. Fabric does not support `queryconsistency` or `query_weakconsistency_session_id`
`timeout`	Positive client-side HTTP timeout in seconds. This is separate from the Kusto `servertimeout` request property
`retain_raw_frames`	Logical. Attach the complete decoded Kusto frame response as `kusto_raw_frames`. Keep `FALSE` for normal queries to avoid retaining a second copy of large result data, including on partial-error conditions

tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, with the Azure CLI application ID as fallback
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()

Value

A typed tibble for one primary result, a fabric_kql_tables list for multiple primary results (one named element per table), or an empty tibble when there is no primary result. See Details for the KQL-to-R type mapping

Basic use

The easiest input is an item from fabric_kql_databases(), which already contains the database name and its **Query URI**. If you copy a URI from Fabric, choose **Query URI**, not **Ingestion URI**. This function reads existing data; it does not load data or run management commands

Put changing values in parameters and declare them in KQL with declare query_parameters(...). The values are sent separately from the query text, which is safer and easier to quote correctly than using paste(). Scalar R values become KQL scalar values; vectors and lists become dynamic arrays or objects. Nested date/time objects and non-finite numbers are rejected because JSON conversion can change their values or types. Use explicit strings (including timezone and fractional seconds for timestamps) and cast them in KQL, or pass these values as separate scalar parameters

Advanced request options

request_properties controls server behavior such as timeouts and result truncation. Most users can leave it empty Microsoft Fabric does not support the queryconsistency or query_weakconsistency_session_id request properties. Do not include either name in request_properties, even though Azure Data Explorer supports them

Result types

KQL bool, datetime, int, long, real, and timespan columns normally become logical, UTC POSIXct, integer, bit64::integer64, double, and difftime vectors. Base R and 'bit64' reserve the minimum signed int and long values for missing data; a column containing either boundary is returned as character with a warning so the value remains exact. dynamic columns are list-columns, and GUIDs, strings, and decimal values are character vectors. Keeping decimal values in their original lexical form avoids the silent precision loss that conversion to an R double can cause

A query with several result tables returns a named fabric_kql_tables list; a query with no result table returns an empty tibble. Service metadata is retained in kusto_* attributes for troubleshooting

Permissions

The caller needs database access through a Fabric workspace role, Eventhouse sharing, or KQL database sharing. Authentication uses the Kusto query service

References

Access a KQL database and copy its Query URI

Kusto query HTTP request and parameters

Kusto request properties

Kusto role-based access control

Examples

```
## Not run:
# Discover the KQL database and choose one of its existing tables
workspace <- fabric_workspaces()[[1L]]
database <- fabric_kql_databases(workspace)[[1L]]
table <- Sys.getenv("FABRIC_KQL_TABLE")

# Keep the changing table name out of the KQL text by using a parameter
events <- fabric_kql_query(
  database,
  query = paste(
    "declare query_parameters(selected_table:string);",
    "table(selected_table) | take 100"
  ),
  parameters = list(selected_table = table)
)

## End(Not run)
```

---

fabric_kql_read_table *Read a Microsoft Fabric KQL table*

Description

Provides the table-oriented read counterpart to `fabric_kql_write_table()`. It safely resolves the table through Kusto's `table()` function, optionally projects columns and limits rows, then delegates typed result handling to `fabric_kql_query()`. Use that lower-level function for filters, ordering, joins, aggregations, or other KQL expressions.

Usage

```
fabric_kql_read_table(
  cluster,
  table,
  database = NULL,
  columns = NULL,
  limit = NULL,
  request_properties = list(),
  timeout = 60,
  retain_raw_frames = FALSE,
```

```
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list()
)
```

Arguments

`cluster`	Query URI, or one Eventhouse or KQLDatabase object returned by `fabric_eventhouses()`, `fabric_kql_databases()`, or `fabric_item()`. A KQLDatabase object also supplies `database`. Despite the argument name, use Fabric's **Query URI** here
`table`	KQL table name, or a record containing a `name`, `table`, or `displayName` field.
`database`	KQL database display name. Supply it with a copied Query URI or an Eventhouse object; omit it when `cluster` is a KQLDatabase object
`columns`	Optional unique column names to project.
`limit`	Optional non-negative whole-number maximum number of rows to return, no greater than Kusto's signed 32-bit `take` limit.
`request_properties`	Named list of Kusto client request options, such as `servertimeout = "2m"` or `notruncation = TRUE`. Most users can leave this empty; these are server-side Kusto controls, not query parameters. Fabric does not support `queryconsistency` or `query_weakconsistency_session_id`
`timeout`	Positive client-side HTTP timeout in seconds. This is separate from the Kusto `servertimeout` request property
`retain_raw_frames`	Logical. Attach the complete decoded Kusto frame response as `kusto_raw_frames`. Keep `FALSE` for normal queries to avoid retaining a second copy of large result data, including on partial-error conditions
`tenant_id`	Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`
`client_id`	Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, with the Azure CLI application ID as fallback
`token`	Optional access token or token-provider function. Leave `NULL` to let 'fabricQueryR' use its normal sign-in flow
`auth_args`	Additional sign-in options passed to `AzureAuth::get_azure_token()`

Value

A typed tibble containing the selected table rows. Kusto metadata is retained in the same attributes as `fabric_kql_query()`.

Large results

The Kusto query HTTP response is collected and decoded in R. Use `columns` and `limit` to bound an interactive read. For a result too large for client memory, use `fabric_kql_export()` to export it server-side to OneLake or another supported storage destination.

References

Kusto `table()` function

Kusto `take` operator

Kusto entity names

Kusto query HTTP request

Examples

```
## Not run:
# Discover a KQL database instead of copying its Query URI and name
workspace <- fabric_workspaces()[[1L]]
database <- fabric_kql_databases(workspace)[[1L]]

# Choose an existing table shown under Tables in the Fabric KQL explorer
table <- Sys.getenv("FABRIC_KQL_TABLE")

# Read a bounded portion of that table into a tibble
events <- fabric_kql_read_table(
  database,
  table,
  limit = 1000
)

## End(Not run)
```

`fabric_kql_tables` *Discover Microsoft Fabric KQL tables*

Description

Lists tables in a Fabric KQL database through Kusto's management endpoint. With `detail = TRUE`, retrieves the database JSON schema once and exposes each table's ordered columns while retaining its complete metadata.

Usage

```
fabric_kql_tables(
  cluster,
  database = NULL,
  detail = TRUE,
  timeout = 60,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list()
)
```

Arguments

cluster	Query URI, or one Eventhouse or KQLDatabase object returned by fabric_eventhouses(), fabric_kql_databases(), or fabric_item(). A KQLDatabase object also supplies database. Despite the argument name, use Fabric's **Query URI** here
database	KQL database display name. Supply it with a copied Query URI or an Eventhouse object; omit it when cluster is a KQLDatabase object
detail	Whether to retrieve the database JSON schema and map it to every table. Set to FALSE to issue only the table-list command.
timeout	Positive client-side HTTP timeout in seconds. This is separate from the Kusto servertimeout request property
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, with the Azure CLI application ID as fallback
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()

Value

A tibble with table name, database, folder, description, list-column columns, parsed schema_metadata, and the unmodified listing row in raw.

References

Kusto .show tables command

Kusto .show database schema command

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
database <- fabric_kql_databases(workspace)[[1L]]

tables <- fabric_kql_tables(database)
events <- fabric_kql_read_table(database, tables[1L, ], limit = 1000)

## End(Not run)
```

---

fabric_kql_write_table

Write an R or Arrow object to an Eventhouse table

Description

Serializes an R or Arrow object to Parquet, uploads it using the storage container or OneLake folder preferred by the Kusto ingestion service, submits tracked queued ingestion, waits for the terminal per-file result, and manages staging cleanup. With `cleanup = TRUE`, Storage sources may be deleted after download, before ingestion succeeds; OneLake staging is removed only after confirmed success. Use `cleanup = FALSE` to retain Storage sources for recovery.

Usage

```
fabric_kql_write_table(
  cluster,
  table,
  data,
  database = NULL,
  mapping = NULL,
  staging_folder = NULL,
  staging_root = "fabricqueryr-staging",
  cleanup = TRUE,
  keep_staging_on_failure = TRUE,
  compression = "snappy",
  target_file_size = 512 * 1024^2,
  max_rows_per_file = NULL,
  tags = character(),
  ingest_if_not_exists = character(),
  skip_batching = FALSE,
  creation_time = NULL,
  timeout = 900,
  poll_interval = 2,
  error_on_failure = TRUE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  storage_token = NULL,
  auth_args = list(),
  create_if_missing = FALSE,
  column_types = NULL,
  query_cluster = NULL,
  numeric_policy = c("exact", "service"),
  .sleep = Sys.sleep,
  .now = Sys.time
)
```

Arguments

`cluster`	Ingestion URI or Eventhouse/KQLDatabase discovery object; see `fabric_kql_ingest()`.
`table`	Target KQL table name.

`data` Data frame, tibble, Arrow Table/RecordBatch, lazy Arrow Dataset/Scanner/query, Arrow RecordBatchReader, or compatible array stream.

`database` Target KQL database name. Omit for a discovered KQLDatabase.

`mapping` Optional predefined Parquet ingestion mapping name.

`staging_folder` Optional trusted OneLake folder URI beginning below an item's `Files/` area. The ingestion configuration's lake folder is used by default.

`staging_root` Relative directory created below the selected lake folder for package staging.

`cleanup` Remove OneLake staging after confirmed success, or authorize Kusto to delete Storage blobs after download.

`keep_staging_on_failure`
Retain staging after a confirmed terminal Kusto failure. The client never deletes staging after ambiguous failures; Storage may already have deleted downloaded blobs when `cleanup = TRUE`.

`compression` Parquet compression supported by `arrow::write_parquet()`.

`target_file_size`
Soft maximum bytes per staged Parquet file. The service's advertised total-size and blob-count limits are still enforced. Storage-container staging uses block upload when a completed file exceeds Azure Storage's single-request Put Blob limit.

`max_rows_per_file`
Optional exact maximum rows per staged file.

`tags` Extent tags passed to `fabric_kql_ingest()`.

`ingest_if_not_exists`
Stable idempotency keys passed to `fabric_kql_ingest()`. Requires staging to produce exactly one file.

`skip_batching` Whether Kusto should bypass normal ingestion batching.

`creation_time` Optional extent creation time passed to `fabric_kql_ingest()`.

`timeout` Positive number of seconds shared by submission and tracked status waiting after upload. Time spent submitting reduces the time available for polling.

`poll_interval` Minimum seconds between ingestion status requests.

`error_on_failure`
Raise a typed error for a confirmed failed or canceled ingestion. Set `FALSE` to return the failed result and its staging disposition.

`tenant_id` Microsoft Entra tenant ID.

`client_id` Microsoft Entra application/client ID.

`token` Optional access token or audience-aware token-provider function. A fixed token must target Kusto and be paired with `storage_token`.

`storage_token` Optional separate Azure Storage access token or token provider. Required when `token` cannot acquire a different audience.

`auth_args` Additional options passed to `AzureAuth::get_azure_token()`.

`create_if_missing`
Whether to create a missing KQL table from the Arrow schema after local validation and before upload. Existing tables are left unchanged.

`column_types` Optional named character vector giving one Kusto scalar type for every data column when `create_if_missing = TRUE`. Supported canonical types are `bool`, `datetime`, `decimal`, `dynamic`, `guid`, `int`, `long`, `real`, and `string`. `NULL` infers them. Arrow time and duration columns must be converted because Kusto's Parquet mapping cannot ingest them as `timespan`.

`query_cluster` Optional Kusto query-service URI or discovery object used for table creation and identity-schema validation. A discovered `cluster` already carries this URI; a standard Microsoft ingestion URI is converted to its paired query URI. Supply this explicitly for a trusted custom ingestion endpoint.

`numeric_policy` Decimal ingestion policy. `"exact"` checks actual staged decimal values before creation/upload and rejects coefficients requiring more than 34 significant digits. `"service"` explicitly delegates conversion to Kusto, including possible rounding or replacement by null. Neither policy changes the source Parquet values or preserves their precision, scale or trailing-zero spelling in Kusto.

`.sleep`, `.now` Internal deterministic polling hooks.

Value

A `fabric_kql_write_result` containing row/file counts, compressed Parquet `bytes`/`part_bytes`, diagnostic Arrow `buffer_bytes`/`part_buffer_bytes`, normalized ingestion status, tracking handle, source IDs, and staging disposition.

One-call staging workflow

The queued-ingestion REST API accepts storage blobs rather than inline R values. This function provides the higher-level one-call workflow: it reads the ingestion service's preview configuration, honors its preferred upload method, creates a unique `fabricqueryr-staging` path, and uploads bounded Parquet parts. Service-provided Storage containers use their short-lived SAS credentials. OneLake staging uses a Storage-audience access token, so an audience-aware credential obtains both required tokens. When `token` is a fixed bearer token or `AzureToken` and OneLake is selected, supply the separate `storage_token`. `staging_folder` explicitly selects OneLake and overrides the advertised upload preference with a trusted `Files/` URI.

The caller therefore needs Kusto Table Ingestor and Database User access, plus write/delete access when OneLake is selected. Advertised Storage containers carry the service-managed SAS access needed for staging.

R and Arrow inputs

Data frames and tibbles are converted through Arrow. Factors become strings; complex and `difftime` columns require an explicit conversion. Arrow Tables, RecordBatches, Datasets, Scanners, `arrow_dplyr_query` objects, and RecordBatchReaders are accepted, as are Arrow-compatible `nanoarrow_array_stream` objects returned by package query helpers. Lazy inputs are read one record batch at a time and written directly to a temporary Parquet parts, so the complete data set is never collected into R memory. A supplied reader or stream is single-use and is consumed.

Parquet identity mapping matches source fields to existing KQL columns by case-sensitive name. Before staging, the writer verifies that those names and their Kusto scalar types exactly match the target table. Supply `mapping` when the Parquet schema and table need an explicit predefined mapping; a named mapping bypasses this identity-schema check.

`ingest_if_not_exists` requires staging to produce one Parquet file, regardless of `skip_batching`. A shared idempotency tag can suppress later files in the same logical write. Stage one file or omit the idempotency key.

The service's advertised `maxDataSize` and source `rawSize` refer to the uncompressed source representation. Arrow's in-memory buffer size is not an equivalent Parquet measurement, so the writer deliberately omits `rawSize` and lets Kusto inspect the staged Parquet metadata. Compressed file sizes and Arrow buffer sizes remain available separately in the result.

Set `create_if_missing = TRUE` to issue Kusto's idempotent `.create table` command after local validation and before upload. A missing table is created from the Arrow schema; an existing table is returned unchanged, so this option never alters an existing schema. Common Arrow scalar and nested types are inferred as Kusto types. Supply a named `column_types` vector to override every column type.

By default, decimal values are checked in every staged Parquet batch before table creation or upload. Kusto ingestion can replace decimals with more than 34 significant digits by null even when ingestion succeeds. Precision above 34 in an Arrow schema is allowed when the actual values fit; insignificant leading and trailing zeros do not count. The same check applies inside nested data and with explicit `column_types` or a named `mapping`. It does not certify arbitrary transformations in those user-selected mappings. Convert decimal columns explicitly to Arrow strings to transfer their full text, or select `numeric_policy = "service"` to accept service conversion, rounding and nulls. Kusto strings merge missing and empty values; preserve a separate null flag when that distinction matters. The check protects mathematical decimal values within the staged Parquet representation; Kusto can canonicalize their precision, scale and trailing-zero spelling.

Service-owned Storage credentials are reacquired after local serialization. During a multipart upload, the writer honors the advertised configuration refresh interval and retries once with new credentials when Storage reports an expired authorization.

Failure and cleanup safety

A successful tracked ingestion is cleaned up by default. Kusto removes service-owned Storage blobs after download; the client removes OneLake staging after confirmed success. Ambiguous results retain OneLake staging. With Storage and `cleanup = TRUE`, an ambiguous or failed batch reports `staging_retained = NA`: some or all blobs may already have been deleted. Set `cleanup = FALSE` to retain Storage sources for recovery. After a confirmed terminal failure, the client leaves remaining staging alone unless `keep_staging_on_failure = FALSE`. The full staging path is carried by `fabric_kql_write_error` conditions. A transport failure during OneLake's final atomic rename can also leave the unique destination present; upload errors report `staging_retained = NA` and the path to inspect.

References

Queued ingestion configuration REST API (preview)

Queued ingestion REST API (preview)

Create a Kusto table

Kusto scalar data types

Kusto Parquet mappings

OneLake ADLS-compatible access

Arrow RecordBatchReader

Arrow Parquet writer

Examples

```
## Not run:
# Discover the KQL database that will receive the R data
workspace <- fabric_workspaces()[[1L]]
database <- fabric_kql_databases(workspace)[[1L]]

# Create a new table when needed, stage the data, and wait for ingestion
result <- fabric_kql_write_table(
  database,
  table = "EventsFromR",
  data = data.frame(id = 1:3, value = c("a", "b", "c")),
  create_if_missing = TRUE,
  ingest_if_not_exists = "r-batch-2026-08-14"
)
result$status$state

# A local Arrow Dataset is scanned batch by batch rather than collected
dataset <- arrow::open_dataset(Sys.getenv("ARROW_DATASET_PATH"))
fabric_kql_write_table(database, "EventsFromArrow", dataset)

## End(Not run)
```

---

fabric_lakehouse_read_table

Read a Microsoft Fabric Lakehouse table

Description

Provides the symmetric read counterpart to `fabric_lakehouse_write_table()`. It resolves a discovered Lakehouse object and table record, then delegates to the authenticated OneLake Delta reader. Use `result = "arrow_stream"` to keep a larger result out of R memory.

Usage

```
fabric_lakehouse_read_table(
  lakehouse,
  table,
  workspace = NULL,
  schema = NULL,
  columns = NULL,
  limit = NULL,
  version = NULL,
  result = c("tibble", "arrow_stream"),
  verbose = TRUE,
```

```
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com"
)
```

Arguments

lakehouse	Lakehouse GUID, exact display name, or one Lakehouse object returned by fabric_lakehouses(). A discovered object is recommended because it carries its workspace ID and default schema.
table	Table name or one row returned by fabric_lakehouse_tables().
workspace	Workspace GUID, exact display name, or discovered workspace. Omit it when lakehouse is a discovered object containing workspaceId.
schema	Optional schema. A table record supplies its schema when this argument is omitted.
columns	Optional unique column names to project before collection.
limit	Optional non-negative maximum number of rows to return.
version	Optional non-negative Delta table version for time travel.
result	Return a "tibble" or a disk-backed, single-use "arrow_stream". Release a stream with stream[["release"]]() when using 'nanoarrow' directly, or close the 'arrow' reader that takes ownership of it
verbose	Whether to report authentication and read progress.
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID.
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID.
token	Optional access token or audience-aware token-provider function.
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token().
dfs_base	OneLake DFS service address. A private or regional endpoint on a discovered object is preferred when this argument is omitted.

Details

Direct reads use the Python runtime described in fabric_delta_config(). See fabric_onelake_read_delta_table() for runtime setup, supported Delta features, OneLake permissions, and column-type conversion rules.

Value

A tibble, or a disk-backed nanoarrow_array_stream when result = "arrow_stream". Explicit release deletes its temporary file.

References

OneLake table APIs for Delta

Connect to OneLake

Examples

```
## Not run:
# Discover both the Lakehouse and the table to read
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
tables <- fabric_lakehouse_tables(lakehouse)
table <- tables[1L, ]

# Read the discovered table into a tibble
rows <- fabric_lakehouse_read_table(lakehouse, table)

# Count rows in batches when the full table may not fit in R memory
row_count <- local({
  stream <- fabric_lakehouse_read_table(
    lakehouse,
    table,
    result = "arrow_stream"
  )
  on.exit(nanoarrow::nanoarrow_pointer_release(stream), add = TRUE)
  reader <- arrow::as_record_batch_reader(stream)
  on.exit(reader$Close(), add = TRUE, after = FALSE)
  count <- 0
  repeat {
    batch <- reader$read_next_batch()
    if (is.null(batch)) break
    count <- count + batch$num_rows
  }
  count
})

## End(Not run)
```

`fabric_lakehouse_tables` *Discover and load Microsoft Fabric Lakehouse tables*

Description

Use Fabric's table APIs to inspect Delta tables, load staged CSV or Parquet files, or write an R/Arrow object through a failure-aware staging workflow.

- `fabric_lakehouse_tables()` combines Fabric's paginated List Tables API with the read-only OneLake Delta table API. The first supplies managed or external type, format, and location; the second supplies schemas and, with `detail = TRUE`, column metadata.

- `fabric_lakehouse_load_table()` starts the preview Fabric Load Table API for a file or folder that already exists below the Lakehouse `Files/` area. It returns a handle accepted by `fabric_operation_status()`.
- `fabric_lakehouse_write_table()` streams an R or Arrow object to Parquet, uploads it to a unique `Files/` staging path, waits for the Delta load, and removes the staged file after confirmed success by default.

Usage

```
fabric_lakehouse_tables(
  lakehouse,
  workspace = NULL,
  schema = NULL,
  detail = TRUE,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)

fabric_lakehouse_load_table(
  lakehouse,
  table,
  path,
  workspace = NULL,
  schema = NULL,
  path_type = c("File", "Folder"),
  format = NULL,
  mode = c("Overwrite", "Append"),
  recursive = FALSE,
  header = TRUE,
  delimiter = ",",
  file_extension = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_lakehouse_write_table(
  lakehouse,
```

```
  table,
  data,
  workspace = NULL,
  schema = NULL,
  mode = c("Overwrite", "Append"),
  staging_root = "Files/fabricqueryr-staging",
  cleanup = TRUE,
  keep_staging_on_failure = TRUE,
  compression = "snappy",
  target_file_size = 512 * 1024^2,
  max_rows_per_file = NULL,
  poll_interval = NULL,
  timeout = 900,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  storage_token = NULL
)
```

Arguments

lakehouse	Lakehouse GUID, exact display name, or one Lakehouse object returned by fabric_lakehouses(). A discovered object is recommended because it includes the workspace and default schema.
workspace	Workspace GUID, exact display name, or discovered workspace. Omit it when lakehouse is a discovered object containing workspaceId.
schema	Optional Lakehouse schema. When omitted from fabric_lakehouse_tables(), every schema is listed. For loading, a discovered schema-enabled Lakehouse supplies its documented default schema; otherwise provide the destination schema explicitly.
detail	Whether table discovery should retrieve per-table column metadata. Detail retrieval enriches the listing snapshot and never removes a listed row if a table disappears concurrently. Set to FALSE to make only schema and table-list requests.
page_size	Optional maximum records requested per table API page, from 1 to the Fabric List Tables maximum of 100. All continuation values are followed regardless of this value.
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID.
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID.
token	Optional access token or audience-aware token-provider function. Table discovery needs both Fabric- and Storage-audience tokens; staging needs Storage and loading needs Fabric.

auth_args	Additional sign-in options passed to AzureAuth::get_azure_token() when no token source is supplied.
api_base	Fabric REST API base URL. Most users should keep the default.
table_api_base	OneLake Delta table API base URL. Most users should keep the default.
storage_token	Optional separate Azure Storage token or token-provider function for fabric_lakehouse_tables() and fabric_lakehouse_write_table(). Supply it when token is a fixed bearer token or AzureToken; automatic and callback credentials obtain both audiences themselves.
table	Destination Delta table name. Fabric's Load Table API permits 1 to 256 ASCII letters, numbers, and underscores and requires at least one letter or underscore.
path	Existing item-relative OneLake source path equal to "Files" or beginning with Files/, for example "Files/incoming/orders.parquet".
path_type	Whether path names one "File" or a "Folder".
format	Source format, "Parquet" or "Csv". For a file, NULL infers the format from its extension. A folder should specify the format.
mode	Load mode, "Overwrite" or "Append". Overwrite and append behavior is performed by Fabric's managed Delta load, never by changing files below Tables/ directly. Fabric documents overwrite as dropping and recreating an existing Delta table; the API does not expose a truncate alternative.
recursive	Whether a folder load should include descendant folders.
header	Whether the first CSV row contains column names.
delimiter	CSV delimiter of 0 to 8 characters. Spaces and tabs are allowed; Fabric excludes parentheses, brackets, braces, and quotes.
file_extension	Optional extension used to filter a folder load, without a leading dot.
data	A data frame, tibble, Arrow Table/RecordBatch, lazy Arrow Dataset/Scanner/query, or Arrow RecordBatchReader to serialize as Parquet. Lazy inputs are consumed batch by batch without collecting the complete object in R memory. Arrow-compatible nanoarrow_array_stream inputs are also accepted. Readers and streams are single-use. The optional 'arrow' package is required.
staging_root	Item-relative directory below Files/ used for unique staging files.
cleanup	Whether to delete the staged Parquet files after Fabric confirms a successful load.
keep_staging_on_failure	Whether to retain a completely uploaded staging directory when the load fails. The raised condition includes staging_path and staging_retained fields.
compression	Parquet compression passed to arrow::write_parquet().
target_file_size	Soft maximum bytes per staged Parquet file. A file rotates after its current Arrow row group reaches this size.
max_rows_per_file	Optional exact maximum rows per staged file. This is useful when row counts are a more predictable boundary than compressed bytes.

poll_interval	Minimum seconds between load-operation status requests. NULL follows Fabric's Retry-After hint with the shared fallback.
timeout	Maximum total seconds to wait for an R/Arrow load.
dfs_base	OneLake DFS service address used for the staging upload. A workspace-specific endpoint from a discovered object is preferred when this argument is not supplied.

Value

fabric_lakehouse_tables() returns a tibble with table name, schema, full_name, type, format, location, timestamps, list-column columns, schema_metadata, the unmodified OneLake raw record, and the matching unmodified Fabric fabric_raw record. Unknown future metadata remains available in those raw list columns.

fabric_lakehouse_load_table() returns a reusable fabric_operation. Pass it to fabric_operation_status(), fabric_operation_wait(), or fabric_operation_result().

fabric_lakehouse_write_table() returns a fabric_lakehouse_write_result containing the destination, row count, terminal operation state, staging path, and whether staging was retained.

Preview status and permissions

Microsoft marks Fabric's List Tables and Load Table routes as preview or beta and does not recommend them for production use. Loading requires write access to the Lakehouse and the Lakehouse.ReadWrite.All delegated scope. Discovery requires Lakehouse.Read.All or Lakehouse.ReadWrite.All for the Fabric list plus table read permission for OneLake metadata.

Fabric currently rejects List Tables for some schema-enabled Lakehouses. In that documented-endpoint/service mismatch, discovery still returns OneLake schema, format, location, and column metadata; type can be missing because OneLake currently returns a null table type for those records.

Service principals and managed identities are supported by the Load Table API. Tenant and item permissions still determine whether those identities can use OneLake and the Lakehouse.

Choose an existing-file load or an R-object write

fabric_lakehouse_load_table() never uploads a local file or serializes an R object. Its path must already exist inside the selected Lakehouse's OneLake Files/ area. Use fabric_onelake_upload() first when intentionally managing that source yourself, or use fabric_lakehouse_write_table() for a single call that accepts a data frame, tibble, or Arrow object, stages it, waits for the load, and cleans up.

Both load functions can create a missing destination Delta table. Fabric infers its schema from the source. No create_if_missing flag is needed.

Data types and names

Arrow determines the Parquet schema before Fabric infers the destination Delta schema. Ordinary R logical, integer, double, character, Date, POSIXct, and bit64::integer64 columns map to their corresponding Parquet logical types. Factors are written as strings. List columns are passed to

Arrow as nested data and can fail if their values do not have one consistent Arrow type. R complex and `difftime` columns are rejected.

R has no native fixed-precision decimal vector. Supply Arrow data with a decimal field when decimal precision and scale must be explicit. Fabric's Load to Tables flow does not accept a caller-defined destination schema, so use Spark or another schema-controlled writer when inference is unsuitable.

To preserve names exactly, `fabric_lakehouse_write_table()` requires unique column names containing only Unicode letters, decimal digits, and underscores, up to Fabric's documented 128-character limit. Use precomposed letters: managed Parquet loads reject decomposed combining marks, connector punctuation other than underscore, and numeric symbols such as superscript digits.

Failure and cleanup behavior

The high-level writer uploads complete Parquet parts atomically to a unique folder and starts the managed folder load only after every upload succeeds. A successful load is a committed Delta operation. On failure, the destination is left to Fabric's transactional load behavior and 'fabricQueryR' never edits `Tables/` files.

Retained staging paths are included in `fabric_lakehouse_write_error` conditions so the source can be inspected or passed to `fabric_lakehouse_load_table()` again. Cleanup failures after a successful load produce a warning and return `staging_retained = TRUE`; they do not make a committed table load appear to have failed. Once Fabric accepts a load, staging is retained if status polling loses access or fails ambiguously; only a confirmed terminal operation failure permits failure cleanup.

References

OneLake table APIs for Delta

Getting started with OneLake Delta table APIs

Arrow RecordBatchReader

List Lakehouse tables

Load a Lakehouse table

Load a schema Lakehouse table (beta)

Load to Delta Lake tables

Examples

```
## Not run:
# Discover a Lakehouse instead of copying its workspace and item IDs
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]

# List its existing Delta tables
tables <- fabric_lakehouse_tables(lakehouse)

# Discover a CSV already stored in this Lakehouse's Files area
files <- fabric_onelake_list(
```

```
  workspace,
  lakehouse,
  path = "Files/incoming"
)
csv_file <- files[grepl("[.]csv$", files$path), ][1L, ]

# Load that discovered CSV into a managed Delta table
operation <- fabric_lakehouse_load_table(
  lakehouse,
  table = "orders_from_csv",
  path = csv_file$path[[1L]],
  format = "Csv",
  header = TRUE,
  delimiter = ","
)
fabric_operation_wait(operation, timeout = 900)

# Or stage an R data frame and write it as a managed Delta table
result <- fabric_lakehouse_write_table(
  lakehouse,
  table = "orders_from_r",
  data = data.frame(id = 1:3, amount = c(10.5, NA, 30))
)
result$operation_status$status

## End(Not run)
```

`fabric_livy_batch_submit` *Submit a Microsoft Fabric Livy batch job*

Description

Runs a complete Python, R, or Java/Scala Spark application stored in OneLake or ADLS. Use this for repeatable scripts and unattended processing; use `fabric_livy_session()` when several interactive statements should share variables and Spark state

Usage

```
fabric_livy_batch_submit(
  livy_url,
  file,
  name = NULL,
  class_name = NULL,
  args = NULL,
  jars = NULL,
  files = NULL,
  py_files = NULL,
  archives = NULL,
```

```
  conf = NULL,
  environment_id = NULL,
  target_lakehouse_id = NULL,
  tags = NULL,
  driver_memory = NULL,
  driver_cores = NULL,
  executor_memory = NULL,
  executor_cores = NULL,
  num_executors = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  verbose = TRUE,
  wait = FALSE,
  timeout = 1200,
  poll_interval = 5,
  cancel_on_timeout = TRUE
)
```

Arguments

livy_url	A copied Livy connection URL, Livy API base URL, or enriched Lakehouse object. Copy the batch-job URL from **Lakehouse settings > Livy endpoint**, or use an item from fabric_lakehouses()
file	Absolute ABFS/ABFSS URI of the main Python, R, or Java/Scala application file. It must contain a filesystem/container, host, and non-root path, without a password, port, query, fragment, backslash, or dot path segment. After uploading a script under a Lakehouse's Files/ area, its **Properties** dialog can copy this path. Spaces in path segments must be percent-encoded as %20; raw spaces and authority whitespace are invalid
name	Optional readable job name shown in Fabric monitoring
class_name	Main class for a Java/Scala application; leave NULL for Python or R scripts
args	Optional character vector of command-line arguments passed to the application
jars	Optional JAR dependency URIs
files	Optional supporting-file URIs copied to the job
py_files	Optional Python dependency URIs, such as .py or .zip files
archives	Optional archive URIs that Spark should unpack
conf	Optional named list of Spark settings or application-specific values
environment_id	Optional GUID of a published Fabric Environment whose libraries and Spark settings should be used
target_lakehouse_id	Optional Lakehouse GUID made available as spark.targetLakehouse. Use this when the application needs an explicit default Lakehouse context

`tags` Optional named list of string labels for monitoring

`driver_memory`, `executor_memory`
Optional Spark memory values such as `"4g"`. Leave `NULL` to use Fabric defaults

`driver_cores`, `executor_cores`, `num_executors`
Optional Spark resource counts. Larger values consume more capacity; leave `NULL` unless the workload has been sized deliberately

`tenant_id` Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`

`client_id` Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID

`token` Optional access token or token-provider function. Leave `NULL` to let 'fabricQueryR' use its normal sign-in flow for a Microsoft Fabric host. A custom `livy_url` requires an explicitly supplied token or provider. HTTPS validation does not prove ownership or token audience; use a custom host only when your organization controls it, with a credential issued for its intended audience

`auth_args` Additional sign-in options passed to `AzureAuth::get_azure_token()`

`audience` Optional sign-in scopes. For delegated sign-in, `NULL` requests the four required Livy scopes listed below. An explicit vector replaces those defaults, so include every required scope plus any optional `Code.Access*` scope the Spark code needs. Client credentials require one `.default` audience

`verbose` Logical. Show submission and lifecycle messages

`wait` Logical. `FALSE` returns immediately so other R work can continue; `TRUE` waits for a terminal state before returning the same object

`timeout` Maximum seconds to wait when `wait = TRUE`

`poll_interval` Seconds between status checks when waiting

`cancel_on_timeout`
Logical. When waiting at submission time, request cancellation if the local timeout expires. Defaults to `TRUE`, so a timed out call does not normally leave Spark compute running unattended. The structured timeout condition contains the live FabricLivyBatch object in `handle`, for status checks or cancellation in the current R process, and stable public metadata in `batch`. A serialized handle intentionally loses its in-process credential

Value

A FabricLivyBatch 'R6' object. Inspect its `$state`, call `$result()` for structured metadata and logs, and call `$wait()` later when submitting with `wait = FALSE`

Before you submit

Fabric needs a workspace on supported capacity and a Lakehouse. The application file must already be accessible through an ABFS/ABFSS URI; this function does not upload a local script. Use `fabric_onelake_upload()` first when needed

Python and Java batch applications have produced their expected output in the package's persistent Fabric sandbox. Standalone R batches remain unverified: attempts have failed during Spark-context

initialization. Treat the R batch path as experimental and validate an application's output in the target runtime before relying on it. Successful `kind = "sparkr"` interactive statements do not establish standalone R batch support.

Delegated sign-in requires `Lakehouse.Execute.All`, `Lakehouse.Read.All`, `Code.AccessFabric.All`, and `Code.AccessStorage.All`. Add `Code.AccessAzureKeyvault.All`, `Code.AccessAzureDataLake.All`, `Code.AccessAzureDataExplorer.All`, or `Code.AccessSQL.All` only when Spark accesses that Azure service at runtime. The signed-in identity also needs an appropriate workspace role

Microsoft's current batch guide is internally inconsistent about service principals: its introduction says SPN is unsupported, while its authentication section provides a certificate-based SPN example. This package can acquire and send a client-credentials token, but cannot make the Fabric service accept that identity. Until Microsoft clarifies the contract, verify unattended batch authentication in the target tenant and use a delegated user when the service rejects an SPN. A Contributor role alone is not a guarantee of batch SPN support

See Also

Microsoft Fabric batch jobs

Examples

```
## Not run:
# Discover the Lakehouse and Python file used by this batch
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
scripts <- fabric_onelake_list(
  workspace,
  lakehouse,
  path = "Files/jobs"
)
script <- scripts[grepl("[.]py$", scripts$path), ][1L, ]
script_uri <- paste0(
  "abfss://", workspace$id, "@onelake.dfs.fabric.microsoft.com/",
  lakehouse$id, "/", script$path[[1L]]
)

# Submit the discovered script and wait for its Spark application to finish
batch <- fabric_livy_batch_submit(
  lakehouse,
  file = script_uri,
  wait = TRUE,
  cancel_on_timeout = TRUE
)
batch$result()

## End(Not run)
```

`fabric_livy_query` *Run Spark code in a temporary Microsoft Fabric Livy session*

Description

Starts Spark, runs one piece of code, returns its output, and closes the Spark session. This is the simplest Livy helper for a one-off operation. For quick reads from a Lakehouse or Warehouse, SQL is often faster to start

Usage

```
fabric_livy_query(
  livy_url,
  code,
  kind = c("spark", "pyspark", "sparkr", "sql"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  environment_id = NULL,
  conf = NULL,
  verbose = TRUE,
  poll_interval = 2,
  timeout = 600,
  ...
)
```

Arguments

`livy_url`	A Livy connection URL copied from the Lakehouse settings, or an enriched Lakehouse object from `fabric_lakehouses()` or `fabric_item()` A discovered object avoids copying workspace and Lakehouse IDs
`code`	One string containing the Spark code to run. Objects created in this temporary session are lost after the function returns, although writes made to Lakehouse storage persist
`kind`	Statement language. Use `"pyspark"` for Python with Spark, `"spark"` for Scala, `"sql"` for Spark SQL, or `"sparkr"` for SparkR. This must match the syntax in `code`. The `sparklyr` package is an R API, not a separate Livy language: experimental code that initializes `sparklyr` through item-scoped Livy still uses `"sparkr"`. SparkR is deprecated upstream in Spark 4.x; see **R on Runtime 2.0** below for the distinction
`tenant_id`	Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`
`client_id`	Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID

token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow. HTTPS validation does not prove ownership or token audience for a custom host; use one only when your organization controls it, with a token or provider issued for its intended audience
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
audience	Optional sign-in scopes. For delegated sign-in, NULL requests the four required Livy scopes listed below. An explicit vector replaces those defaults, so include every required scope plus any optional Code.Access* scope the Spark code needs. Client credentials require one .default audience
environment_id	Optional GUID of a published Fabric Environment whose libraries and Spark settings should be used. Leave NULL to use the Lakehouse/workspace defaults
conf	Optional named list of Spark configuration overrides, for example list("spark.sql.shuffle.partitio = "100"). Most users can leave this NULL and configure shared settings in a Fabric Environment
verbose	Logical. Show session startup, execution, and cleanup progress
poll_interval	Seconds between status checks. Lower values update sooner but make more API calls
timeout	Maximum seconds to wait for session readiness and, separately, statement completion
...	Compatibility arguments. The former named access_token argument is accepted here as a deprecated alias for token; all other arguments are rejected

Value

Invisibly, a fabric_livy_statement_result list. The most useful component is output$parsed: a tibble for tabular output, an R object for JSON, or a character vector for text. The result also keeps status, timing, submitted code, errors, and the original response. A successful statement is still returned when session cleanup fails, with a fabric_livy_cleanup_warning identifying the retained session. When both execution and cleanup fail, a fabric_livy_execution_cleanup_error retains the execution error and safe cleanup diagnostics

Tabular column names

Duplicate SQL aliases and joined column names are repaired with make.unique(names, sep = "..."): for example, id, id becomes id, id...1. Every column retains its positional values. The spark_schema attribute keeps the original header names and types, and the result retains the original response.

Before you run code

Fabric needs a workspace on supported capacity, a Lakehouse, and the tenant admin setting for the Livy API enabled. In the Fabric portal, open the Lakehouse settings, find **Livy endpoint**, and copy the session-job connection string. For several statements that reuse variables and Spark state, use fabric_livy_session(). To run a complete Python, Scala/Java, or R application file, use fabric_livy_batch_submit()

A delegated caller needs the `Lakehouse.Execute.All`, `Lakehouse.Read.All`, `Code.AccessFabric.All`, and `Code.AccessStorage.All` scopes and must be a Contributor in the workspace. For session jobs, Microsoft's current guide also documents service-principal (SPN) tokens. The service principal must be added to the workspace as a Contributor, but that role alone does not override tenant settings or other service-side identity restrictions. Add `Code.AccessAzureKeyvault.All`, `Code.AccessAzureDataLake.All`, `Code.AccessAzureDataExplorer.All`, or `Code.AccessSQL.All` only when the Spark code accesses that Azure service at runtime

Spark long and decimal columns are returned as character values when needed to preserve them exactly. Dates and timestamps with a time zone use R temporal classes; timestamps without a time zone remain wall-clock text Fabric's SQL JSON output represents non-finite floating-point values as `null`, so those values are returned as typed missing values. Binary and nested values use list-columns Nested decimal values retain their JSON spelling. Fabric may round these values before sending SQL JSON output; cast decimal leaves to STRING in Spark when full precision is required across that service boundary. Generic JSON arrays combine compatible numbers into numeric vectors. Mixed scalar types remain lists or tibble list-columns so numbers and exact integer or decimal strings retain their original values.

R on Runtime 2.0

Microsoft Fabric distributes `sparklyr` and documents `sparklyr::spark_connect(method = "synapse")` for Fabric notebooks and Spark job definitions. Microsoft does not currently document that connection from an item-scoped Livy session, and this package's live suite validates the `"sparkr"` interpreter but not a `sparklyr` connection over it. Treat that adaptation as experimental and verify it in the target runtime before use. It still depends on the SparkR JVM bridge, which Spark 4.x deprecates. Prefer PySpark or Spark SQL when the remote workload must be independent of that bridge

See Also

Microsoft Fabric Livy API overview, Livy API setup and authorization, Use sparklyr in Fabric, and Fabric Runtime 2.0

Examples

```
# Livy can run SQL, PySpark, Spark, and SparkR code in Microsoft Fabric
# This function is not called automatically because it requires credentials
fabric_livy_query_example <- function() {
  # Discover a Lakehouse whose record contains its Fabric Livy endpoint
  workspace <- fabric_workspaces()[[1L]]
  lakehouse <- fabric_lakehouses(workspace)[[1L]]
  table <- fabric_lakehouse_tables(lakehouse)[1L, ]

  # Build SQL from the discovered table, then close the temporary session
  sql <- sprintf(
    "SELECT COUNT(*) AS row_count FROM `%s`.`%s`",
    table$schema[[1L]],
    table$name[[1L]]
  )
  sql_result <- fabric_livy_query(
    livy_url = lakehouse,
```

```
  kind = "sql",
  code = sql
)

# PySpark avoids the SparkR bridge. The Livy vignette separately labels the
# item-scoped sparklyr adaptation experimental and not live-validated here
pyspark_result <- fabric_livy_query(
  livy_url = lakehouse,
  kind = "pyspark",
  code = "print(1 + 2)"
)

invisible(list(sql = sql_result, pyspark = pyspark_result))
}
```

`fabric_livy_session` *Create a Microsoft Fabric Livy session*

Description

Starts Spark compute that can run several statements while keeping variables and Spark state between calls. Use `fabric_livy_query()` instead for a single, self-contained operation

Usage

```
fabric_livy_session(
  livy_url,
  high_concurrency = FALSE,
  session_tag = NULL,
  name = NULL,
  tags = NULL,
  conf = NULL,
  environment_id = NULL,
  archives = NULL,
  driver_memory = NULL,
  driver_cores = NULL,
  executor_memory = NULL,
  executor_cores = NULL,
  num_executors = NULL,
  artifact_name = NULL,
  file = NULL,
  class_name = NULL,
  args = NULL,
  jars = NULL,
  files = NULL,
  py_files = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
```

```
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  verbose = TRUE
)
```

Arguments

livy_url	A copied session or batch connection URL, Livy API base URL, or enriched Lakehouse object from fabric_lakehouses() or fabric_item() Copy the session-job URL from **Lakehouse settings > Livy endpoint**, or use a discovered object to avoid handling IDs manually
high_concurrency	Whether to let Fabric share Spark compute between several isolated workloads. Keep FALSE for a typical sequence of calls in one R process
session_tag	Optional high-concurrency packing hint. Related requests with the same tag may share an underlying Livy session while keeping separate REPL state. Each call still returns a distinct HC session
name	Optional readable session name shown in service metadata
tags	Optional named list of string labels for monitoring
conf	Optional named list of Spark settings. Prefer a published Fabric Environment for configuration shared by several jobs
environment_id	Optional GUID of a published Fabric Environment whose libraries and Spark settings should be used
archives	Optional character vector of archive URIs made available to Spark
driver_memory, executor_memory	Optional Spark memory values such as "4g". Leave NULL to use Fabric defaults
driver_cores, executor_cores, num_executors	Optional Spark resource counts. Larger values consume more capacity; leave NULL unless the workload has been sized deliberately
artifact_name	Optional Lakehouse/artifact label used for a high-concurrency job in the Fabric Monitoring hub
file	Optional application file URI for a high-concurrency request
class_name	Optional Java/Scala main class for file
args	Optional character vector of application arguments
jars, files, py_files	Optional character vectors of dependency URIs supplied to Spark
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow for a Microsoft Fabric host. A custom livy_url requires an explicitly supplied token or provider. HTTPS validation does not prove ownership or token audience; use a custom host only when your organization controls it, with a credential issued for its intended audience

auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
audience	Optional sign-in scopes. For delegated sign-in, NULL requests the four required Livy scopes listed below. An explicit vector replaces those defaults, so include every required scope plus any optional Code.Access* scope the Spark code needs. Client credentials require one .default audience
verbose	Logical. Show session lifecycle messages

Value

A newly created FabricLivySession. It may still be starting; call $wait() before $submit()/$run(), and $close() when finished. These handle lifecycle methods do not have separate free-function wrappers

Choosing a session type

Use a standard session for a typical sequence in one R process. High concurrency is for applications that run several independent Spark workloads at the same time; it is not needed for several sequential statements

Cleanup and permissions

No network request is made when an open object is garbage collected. Call $close() explicitly, and use on.exit(session$close()) inside functions Delegated sign-in requires Lakehouse.Execute.All, Lakehouse.Read.All, Code.AccessFabric.All, and Code.AccessStorage.All. Add Code.AccessAzureKeyvault.All Code.AccessAzureDataLake.All, Code.AccessAzureDataExplorer.All, or Code.AccessSQL.All only when Spark accesses that Azure service at runtime. Microsoft's current session guide documents delegated-user and service-principal (SPN) tokens. The signed-in identity also needs an appropriate workspace role, and service-side tenant settings still apply

Timeouts

A fabric_livy_timeout_error contains the exact session or statement object in its handle field, so it can be polled or cancelled in the current R process. The kind-specific session or statement field contains safe, serializable metadata; a serialized handle intentionally loses its in-process credential

See Also

Microsoft session jobs, high-concurrency Livy, and the Apache Livy REST API

Examples

```
## Not run:
# Discover the Lakehouse whose Livy endpoint will host the Spark session
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]

run_shared_state <- function(lakehouse) {
  # Keep one session alive so successive statements share Spark state
  session <- fabric_livy_session(lakehouse)
```

```
  on.exit(session$close(), add = TRUE)
  session$wait()
  session$run("shared_value = 40", kind = "pyspark")
  session$run("print(shared_value + 2)", kind = "pyspark")
}
run_shared_state(lakehouse)

run_high_concurrency <- function(lakehouse) {
  # A session tag lets compatible callers reuse high-concurrency compute
  session <- fabric_livy_session(
    lakehouse,
    high_concurrency = TRUE,
    session_tag = "report-workers"
  )
  on.exit(session$close(), add = TRUE)
  session$wait()
  session$run("SELECT current_timestamp()", kind = "sql")
}
run_high_concurrency(lakehouse)

## End(Not run)
```

`fabric_livy_sessions` *Discover and reattach to Microsoft Fabric Livy work*

Description

Lists existing Livy sessions or batches and creates a newly authenticated handle for work that was started by an earlier R process. Listing never returns authentication credentials. Attaching retrieves current service state and does not create a new session or batch.

Usage

```
fabric_livy_sessions(
  livy_url,
  high_concurrency = FALSE,
  top = 100L,
  skip = 0L,
  count = TRUE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL
)

fabric_livy_batches(
```

```
  livy_url,
  top = 100L,
  skip = 0L,
  count = TRUE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL
)

fabric_livy_session_attach(
  livy_url,
  session_id,
  high_concurrency = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  verbose = TRUE
)

fabric_livy_batch_attach(
  livy_url,
  batch_id,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  audience = NULL,
  verbose = TRUE
)
```

Arguments

`livy_url`	A copied session or batch connection URL, Livy API base URL, or enriched Lakehouse object from `fabric_lakehouses()` or `fabric_item()`
`high_concurrency`	For `fabric_livy_session_attach()`, whether to attach to a high-concurrency session. `fabric_livy_sessions()` only lists regular sessions because the Livy endpoint does not expose a high-concurrency collection-list operation
`top`	Maximum records requested for this page
`skip`	Number of matching records to skip

count	Whether Fabric should include the total matching record count. When Fabric returns only a count, the matching page is retrieved separately; the total and rows can therefore reflect different instants.
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to use the normal sign-in flow for a Microsoft Fabric host. A custom livy_url requires an explicitly supplied token or provider
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
audience	Optional sign-in scopes. Delegated sign-in defaults to the required Fabric Livy scopes; client credentials require one .default audience
session_id, batch_id	Service GUID returned by a list or submit operation
verbose	Logical. Show handle lifecycle messages

Value

fabric_livy_sessions() and fabric_livy_batches() return one page as a tibble with columns id, name, state, result, app_id, service timestamps, and raw. The tibble has total_count, page_size, and skip attributes. The attach functions return a FabricLivySession or FabricLivyBatch with a fresh in-process credential.

Restart recovery

Livy handles intentionally do not serialize their credentials. Store the service ID, then call the corresponding attach function after restarting R. Attaching only reconstructs the local handle; it never submits new Spark work.

High-concurrency recovery

Fabric supports acquiring an HC session and getting or deleting one by its HC session ID, but it does not expose a collection GET for highConcurrencySessions. Store the ID returned by fabric_livy_session() and pass it to fabric_livy_session_attach() with high_concurrency = TRUE. Calling fabric_livy_sessions() with high_concurrency = TRUE fails locally instead of sending an unsupported request.

See Also

Microsoft Fabric Livy API specification and Microsoft's high-concurrency endpoint reference

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]

sessions <- fabric_livy_sessions(lakehouse)
```

```
session <- fabric_livy_session_attach(lakehouse, sessions$id[[1L]])
session$status()

batches <- fabric_livy_batches(lakehouse)
batch <- fabric_livy_batch_attach(lakehouse, batches$id[[1L]])
batch$status()

## End(Not run)
```

`fabric_mirrored_database_tables` *Work with Microsoft Fabric mirrored database tables*

Description

Discover schemas and Delta tables replicated into a Fabric mirrored database, retrieve one table's detailed metadata, or read a table directly from OneLake. The discovery helpers use the read-only OneLake table metadata API; the reader uses the mirrored Delta log.

Usage

```
fabric_mirrored_database_schemas(
  mirrored_database,
  workspace = NULL,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)

fabric_mirrored_database_tables(
  mirrored_database,
  workspace = NULL,
  schema = NULL,
  detail = TRUE,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
```

```
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)

fabric_mirrored_database_table(
  mirrored_database,
  table,
  workspace = NULL,
  schema = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)

fabric_mirrored_database_read_table(
  mirrored_database,
  table,
  workspace = NULL,
  schema = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  version = NULL,
  verbose = TRUE,
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  columns = NULL,
  limit = NULL,
  result = c("tibble", "arrow_stream"),
  api_base = .fabric_api_base,
  storage_token = NULL
)
```

Arguments

`mirrored_database`	Mirrored Database GUID, exact display name, or one object returned by `fabric_mirrored_databases()`. A discovered object is recommended because it contains workspace, OneLake, and SQL details.
`workspace`	Workspace GUID, exact display name, or discovered workspace. Omit it when `mirrored_database` contains `workspaceId`.
`page_size`	Optional maximum records requested per OneLake metadata page, from 1 to

	100. All continuation tokens are followed.
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID.
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID.
token	Optional access token or audience-aware token-provider function.
auth_args	Additional sign-in options passed to fabric_credential().
api_base	Fabric REST API base used when an item name or GUID must be resolved. Most users should keep the default.
table_api_base	OneLake Delta table API base URL. Most users should keep the default.
storage_token	Optional separate Azure Storage token or token-provider function. Supply it when token is fixed and item lookup is needed.
schema	Optional schema filter. The singular metadata and read helpers use a discovered default schema when available, otherwise "dbo".
detail	Whether table discovery should retrieve column metadata for every table.
table	Table name, or a one-row record containing name and optionally schema.
version	Specific Delta table version to read, or NULL for latest.
verbose	Whether to show authentication and read progress.
dfs_base	OneLake service address. Most users should keep the default; a workspace-specific address discovered from Fabric is used when available.
columns	Column names to return, or NULL for all columns.
limit	Maximum number of rows to return, or NULL for all rows.
result	"tibble" or "arrow_stream" for batch processing.

Value

fabric_mirrored_database_schemas() returns the same schema tibble as fabric_lakehouse_schemas(). The table metadata functions return the same table tibble as fabric_warehouse_tables(). The reader returns a tibble or a single-use nanoarrow_array_stream.

SQL alternative

Mirrored databases also expose a read-only SQL analytics endpoint. Pass a discovered mirrored database object to fabric_sql_tables(), fabric_sql_read_table(), or fabric_sql_query() when SQL permissions or SQL views are required.

References

Get Mirrored Database

Mirroring in Microsoft Fabric

OneLake catalog table APIs

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
database <- fabric_mirrored_databases(workspace)[[1L]]

schemas <- fabric_mirrored_database_schemas(database)
tables <- fabric_mirrored_database_tables(database)
rows <- fabric_mirrored_database_read_table(database, tables[1L, ], limit = 1000)

## End(Not run)
```

---

fabric_onelake_catalog

Discover OneLake schemas and individual tables

Description

These helpers expose the read-only OneLake Delta table metadata API for Lakehouses and Warehouses. The schema helpers follow every metadata page. The singular table helpers retrieve one table's full column metadata without listing every table in every schema.

Usage

```
fabric_lakehouse_schemas(
  lakehouse,
  workspace = NULL,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)

fabric_warehouse_schemas(
  warehouse,
  workspace = NULL,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
```

```
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)

fabric_lakehouse_table(
  lakehouse,
  table,
  workspace = NULL,
  schema = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL,
  enrich_fabric = FALSE
)

fabric_warehouse_table(
  warehouse,
  table,
  workspace = NULL,
  schema = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)
```

Arguments

lakehouse	Lakehouse GUID, exact display name, or one Lakehouse object returned by fabric_lakehouses().
workspace	Workspace GUID, exact display name, or discovered workspace. Omit it when the item object contains workspaceId.
page_size	Optional maximum schemas requested per metadata page, from 1 to 100. All continuation tokens are followed.
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID.
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID.
token	Optional access token or audience-aware token-provider function.

auth_args	Additional sign-in options passed to fabric_credential().
api_base	Fabric REST API base used when an item name or GUID must be resolved. Most users should keep the default.
table_api_base	OneLake Delta table API base URL. Most users should keep the default.
storage_token	Optional separate Azure Storage token or token-provider function. Supply it when token is fixed and Fabric item lookup is needed.
warehouse	Warehouse GUID, exact display name, or one Warehouse object returned by fabric_warehouses().
table	Table name, or a record containing a name, table, or displayName field. A record can also supply schema.
schema	Schema containing table. Defaults to the Lakehouse default schema when available, otherwise "dbo".
enrich_fabric	For fabric_lakehouse_table(), also list the Fabric table inventory to enrich the result. Defaults to FALSE, so a complete discovered item needs only a Storage token. Setting TRUE requires both Fabric and Storage audiences; use an audience-aware provider or supply token and storage_token separately.

Value

The schema functions return a tibble with name, catalog, full_name, comment, owner, schema_id, timestamps, and the unmodified metadata record in raw. The table functions return one row with the same columns as fabric_lakehouse_tables() or fabric_warehouse_tables().

Permissions

The OneLake table API uses the Azure Storage token audience and requires permission to read the item's tables through OneLake.

References

Explore tables with OneLake catalog APIs

OneLake table APIs for Delta

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]

schemas <- fabric_lakehouse_schemas(lakehouse)
orders <- fabric_lakehouse_table(lakehouse, "orders", schema = "dbo")

## End(Not run)
```

`fabric_onelake_files` *Work with files in Microsoft Fabric OneLake*

Description

List, inspect, download, upload, and delete ordinary files stored in OneLake These helpers are intended for files such as CSV, JSON, images, and model artifacts in a Fabric item's `Files/` area

- `fabric_onelake_list()` lists paths and follows all continuation tokens
- `fabric_onelake_metadata()` returns file or directory properties
- `fabric_onelake_download()` reads a file into memory or streams it to disk
- `fabric_onelake_upload()` creates or replaces a file
- `fabric_onelake_delete()` explicitly deletes a file or directory

Usage

```
fabric_onelake_list(
  workspace,
  item = NULL,
  path = "",
  recursive = FALSE,
  page_size = 5000L,
  begin_from = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com"
)

fabric_onelake_metadata(
  workspace,
  item = NULL,
  path = "",
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com"
)

fabric_onelake_download(
```

```
  workspace,
  item = NULL,
  path = "",
  dest = NULL,
  range = NULL,
  overwrite = FALSE,
  if_match = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com"
)

fabric_onelake_upload(
  workspace,
  item = NULL,
  path = "",
  source,
  overwrite = FALSE,
  if_match = NULL,
  content_type = NULL,
  create_parents = TRUE,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  allow_managed_tables = FALSE,
  chunk_size = getOption("fabricqueryr.onelake.chunk_size", 8 * 1024^2)
)

fabric_onelake_delete(
  workspace,
  item = NULL,
  path = "",
  recursive = FALSE,
  confirm = FALSE,
  if_match = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
```

```
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  allow_managed_tables = FALSE
)
```

Arguments

workspace	Workspace name, ID, object from fabric_workspaces(), or a complete OneLake HTTPS/ABFSS path When item contains a workspace ID, a supplied workspace name must match its recorded workspaceDisplayName. If that name is unavailable, supply the workspace ID or a discovered workspace object instead.
item	Item name, GUID, or discovered Fabric item. Use NULL when workspace is a complete OneLake path. An item from fabric_lakehouses() is the least ambiguous input
path	Path relative to the item, usually beginning with Files/ or Tables/, for example "Files/incoming/data.csv". Use forward slashes A complete OneLake path already contains this value
recursive	For listing, whether to include all descendants. For deletion, whether a non-empty directory may be removed
page_size	Maximum paths requested from OneLake per API call, from 1 to 5000. Smaller values reduce each response size but require more requests
begin_from	Optional path at which to begin a listing. Use this to resume a long, alphabetically ordered scan. Non-recursive listings accept only a single path level
item_type	Optional Fabric item type appended to an item name unless that name already ends in the same suffix, for example "Lakehouse" Usually unnecessary for a discovered item or a name such as "Sales.Lakehouse"
tenant_id	Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
dfs_base	OneLake service address. Most users should keep the default; a workspace-specific address discovered from Fabric is used when available
dest	Optional local destination. When NULL, download returns a raw vector held in R memory. Supply a path to stream large files to disk. A destination download is staged before it replaces an existing file
range	Optional inclusive zero-based byte range. Supply one value for all bytes from that offset onward, or two values for start through end Leave NULL to download the entire file. A ranged request must receive a matching HTTP 206 Content-Range response whose body has the claimed size
overwrite	Whether an existing local or OneLake file may be replaced Existing files are protected by default
if_match	Optional file version (etag) returned by fabric_onelake_metadata(). The operation proceeds only if the file still has that version

source	Local file path or raw vector to upload. A path is streamed; a raw vector is already held in memory
content_type	Optional MIME type stored with an uploaded file, for example "text/csv"
create_parents	Logical. Create missing parent directories below the Fabric-managed first-level folder. Keep TRUE for normal uploads
allow_managed_tables	Whether to allow direct changes below Tables/ Keep FALSE for normal use: changing Delta files directly can corrupt a managed table. This guard checks only the supplied path and does not resolve shortcuts. A path below Files/ can reach a managed table through a shortcut; writes or deletion of descendants then change the shortcut target's data.
chunk_size	Upload chunk size in bytes. The default suits most files; larger values make fewer requests but use more memory
confirm	Safety switch that must be explicitly set to TRUE before deletion is attempted

Value

fabric_onelake_list() returns one row per path, including its item-relative path, file name, is_directory, content_length, etag, and modification/permission fields fabric_onelake_metadata() and fabric_onelake_upload() return a one-row tibble with the resolved path and available HTTP metadata fabric_onelake_download() returns a raw vector when dest = NULL, or invisibly returns the destination path after writing to disk fabric_onelake_delete() invisibly returns TRUE

Choosing a target

The easiest inputs are a workspace plus an item returned by fabric_lakehouses(). You can also use names, IDs, or a complete OneLake HTTPS/ABFSS path. When using an item name, include its type suffix, such as "Sales.Lakehouse", or supply item_type

A Lakehouse's Tables/ area is managed as Delta tables. Use fabric_onelake_read_delta_table() to read those tables, and use SQL, Spark, or another Delta-aware tool to change them. Uploading or deleting individual files below Tables/ can damage a table and is blocked by default

Permissions

The signed-in user or application needs access through a workspace role or the item's **OneLake security roles**, configured under **Manage OneLake security**. Uploading and deleting need write permission. Your Fabric administrator must also allow external apps to access OneLake. If a call returns HTTP 403 after sign-in succeeds, check both that tenant setting and the item's data permissions

Listing integrity

Directory listing validates every JSON page and path record before returning data. Malformed envelopes, invalid metadata values, and paths outside the requested item directory raise fabric_onelake_protocol_error; they are not silently converted to empty or partial results

Storage API version

Requests use OneLake's currently documented ADLS API version, `2021-06-08`. For controlled compatibility testing with a later service version, set option `fabricqueryr.onelake.api_version` to another date in `YYYY-MM-DD` form

Safe file replacement

Existing files are protected unless `overwrite = TRUE`. Uploads and downloads are staged before replacing their destination, so an interrupted transfer does not normally leave a partial file. Local downloads are published with an atomic same-directory rename or hard link and fail closed when the filesystem cannot provide the required primitive. Use `if_match` when a OneLake file should be replaced only if it has not changed since you inspected it

A response failure during an upload's final rename can leave the server-side outcome unknown. In that case a `fabric_onelake_commit_ambiguous` error reports absolute target and staging URLs plus their relative paths. Automatic cleanup is not attempted, but a committed rename may already have consumed the staging path, so the condition reports its presence as unknown

References

Connect to OneLake with ADLS APIs

ADLS Gen2 List Paths

Create and manage OneLake security roles

OneLake security best practices

OneLake tenant settings

Examples

```
## Not run:
# Discover the OneLake target instead of typing workspace and item names
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]

# Create a small local CSV and upload it to the discovered Lakehouse
local_csv <- tempfile(fileext = ".csv")
write.csv(data.frame(id = 1:3), local_csv, row.names = FALSE)
fabric_onelake_upload(
  workspace,
  lakehouse,
  "Files/incoming/example.csv",
  source = local_csv
)

# List the folder and inspect metadata for the uploaded file
files <- fabric_onelake_list(
  workspace = workspace,
  item = lakehouse,
  path = "Files/incoming",
  recursive = TRUE
)
```

```
metadata <- fabric_onelake_metadata(
  workspace,
  lakehouse,
  "Files/incoming/example.csv"
)

# Download the first 100 bytes when only a file sample is needed
bytes <- fabric_onelake_download(
  workspace,
  lakehouse,
  "Files/incoming/example.csv",
  range = c(0, 99)
)

# Deletion is explicit and requires confirm = TRUE
fabric_onelake_delete(
  workspace,
  lakehouse,
  "Files/incoming/example.csv",
  confirm = TRUE
)

## End(Not run)
```

`fabric_onelake_object_files` *Read and write R or Arrow objects in OneLake Files*

Description

These object-aware helpers sit above `fabric_onelake_download()` and `fabric_onelake_upload()`. They serialize data frames, tibbles, and lazy Arrow inputs without collecting the complete object in R memory, and decode supported OneLake files directly to a tibble or Arrow stream.

Usage

```
fabric_onelake_read_file(
  workspace,
  item = NULL,
  path = "",
  format = c("auto", "parquet", "csv", "arrow"),
  result = c("tibble", "arrow_stream"),
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
```

```
  col_names = TRUE,
  na = "NA",
  col_types = NULL,
  csv_numeric = c("exact", "infer")
)

fabric_onelake_write_file(
  workspace,
  item = NULL,
  path = "",
  data,
  format = c("auto", "parquet", "csv", "arrow"),
  overwrite = FALSE,
  if_match = NULL,
  compression = "snappy",
  include_header = TRUE,
  na = "NA",
  create_parents = TRUE,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  allow_managed_tables = FALSE,
  chunk_size = getOption("fabricqueryr.onelake.chunk_size", 8 * 1024^2)
)
```

Arguments

workspace	Workspace name, ID, object from fabric_workspaces(), or a complete OneLake HTTPS/ABFSS path.
item	Item name, GUID, or discovered Fabric item. Use NULL when workspace is a complete OneLake path.
path	Item-relative path, normally below Files/.
format	File format. "auto" infers "parquet", "csv", or "arrow" from the path extension.
result	Return a "tibble" or a disk-backed, single-use "arrow_stream". Tibbles preserve decimals as character strings. Signed 64-bit integers use bit64::integer64, or character if the column contains the minimum signed value (reserved for missing values by bit64). Int32 columns containing -2147483648 use exact R doubles. Nested lists retain character 64-bit integers and decimals, and double 32-bit integers. Structs with null parents require result = "arrow_stream": tibble collection cannot distinguish them from valid structs with all-null fields and raises fabric_arrow_null_struct_error before discarding that distinction.

item_type — Optional Fabric item type used to resolve a named item.

tenant_id — Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID.

client_id — Entra application ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID.

token — Optional access token or audience-aware token-provider function.

auth_args — Additional sign-in options passed to AzureAuth::get_azure_token().

dfs_base — OneLake DFS service address. A private or regional endpoint on a discovered object is preferred when this argument is omitted.

col_names — Whether a CSV has a header, or a character vector of column names. Use FALSE for files written with include_header = FALSE.

na — Text used for missing values in a written CSV, or character values interpreted as missing when reading CSV. The default, "NA", preserves empty strings separately from missing values and keeps missing-only rows visible in CSV files. Blank records are retained when reading. For legacy files whose empty fields mean missing, read with na = c("", "NA"). Any literal text matching a chosen missing marker is also read as missing; choose the same custom marker for writing and reading when needed.

col_types — Optional Arrow Schema specifying CSV column types, such as arrow::schema(amount = arrow::decimal128(38, 15), id = arrow::uint64()). Columns omitted from the schema follow csv_numeric. Use arrow::utf8() to preserve all spelling, including leading zeros in identifiers. Arrow infers omitted column types from an initial block, not the entire file. Later values incompatible with that type raise an error, including oversized integers after small integers, or text after an all-null block. Supply explicit types for such columns, for example arrow::schema(id = arrow::utf8(), label = arrow::utf8()), to read all blocks consistently without losing the original text.

csv_numeric — CSV numeric inference policy. The default, "exact", retains inferred floating-point columns as character strings to avoid rounding decimals or oversized integers. Inferred integer columns retain their exact values but canonicalize spellings such as leading zeros. "infer" opts into Arrow's ordinary inference, including approximate floating-point values. Explicit col_types entries override this policy. These options apply equally to tibbles and Arrow streams. CSV does not store type metadata; use col_types to recover known types after writing, or Parquet/Arrow IPC to retain numeric types automatically.

data — A data frame, tibble, Arrow Table/RecordBatch, lazy Arrow Dataset/Scanner/query, RecordBatchReader, or Arrow-compatible array stream.

overwrite — Whether an existing OneLake file may be replaced.

if_match — Optional destination ETag for conditional replacement.

compression — Parquet compression codec passed to Arrow.

include_header — Whether a written CSV includes column names.

create_parents — Whether missing parent directories are created.

allow_managed_tables — Whether direct writes below Tables/ are permitted. Keep the safe default, FALSE, for managed Delta tables. This guard checks only the supplied path

and does not resolve shortcuts. A `Files/` shortcut can lead to a managed table, where writes change the target's data even with `allow_managed_tables = FALSE`.

`chunk_size` Upload chunk size in bytes.

Value

`fabric_onelake_read_file()` returns a tibble or a disk-backed `nanoarrow_array_stream`. `fabric_onelake_write_fil` returns OneLake metadata with additional `format`, `rows`, and `columns` fields.

References

Connect to OneLake with ADLS APIs

Get data into OneLake

Examples

```
## Not run:
# Discover the Lakehouse that will store the Parquet file
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]

# Serialize the R data frame directly to OneLake as Parquet
fabric_onelake_write_file(
  workspace,
  lakehouse,
  "Files/exports/orders.parquet",
  data.frame(id = 1:3, amount = c(10.5, NA, 30))
)

# Read the same file back as a tibble
orders <- fabric_onelake_read_file(
  workspace,
  lakehouse,
  "Files/exports/orders.parquet"
)

## End(Not run)
```

`fabric_onelake_read_delta_table` *Read a Delta table from OneLake*

Description

Loads a Lakehouse or compatible Warehouse table into R. By default the result is a tibble; you can select columns, preview a limited number of rows, read an earlier table version, or return an Arrow stream for larger results

Usage

```
fabric_onelake_read_delta_table(
  table_path,
  workspace_name,
  lakehouse_name,
  schema = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  version = NULL,
  verbose = TRUE,
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  columns = NULL,
  limit = NULL,
  result = c("tibble", "arrow_stream")
)
```

Arguments

table_path	Table name. Supply its schema separately when needed
workspace_name	Workspace name, ID, or an object returned by fabric_workspaces()
lakehouse_name	Lakehouse name, ID, or discovery object. Compatible Warehouse and mirrored database items are also accepted
schema	Schema containing the table, or NULL. Warehouses and mirrored databases default to "dbo" when discovery provides no default. Use "" for a physical table directly below Tables/ without a schema directory
item_type	"Lakehouse", "Warehouse", "MirroredDatabase", or NULL. Usually inferred; specify it only when using an item name without a type suffix
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Most users can leave this as NULL and let 'fabricQueryR' sign in
auth_args	Extra sign-in options passed to AzureAuth::get_azure_token()
version	Specific table version to read, or NULL for the latest
verbose	Whether to show authentication and read progress
dfs_base	OneLake service address. Most users should keep the default; a workspace-specific address discovered from Fabric is used when available
columns	Column names to return, or NULL for all columns
limit	Maximum number of rows to return, or NULL for all rows
result	"tibble" (the default) or "arrow_stream" for batch processing

Value

A tibble, or a disk-backed, lazy, single-use Arrow stream when `result = "arrow_stream"`. Explicitly release that stream, or close an 'arrow' reader that takes ownership of it, to delete its temporary file

Basic use

Supply the table name, workspace, and Lakehouse. Names, IDs, and discovery records are accepted. If the Lakehouse uses schemas, pass the schema name separately. The function otherwise reads the latest version and all columns and rows into a tibble

Use `columns` to keep only the fields you need, `limit` for a quick preview, and `version` to read an earlier version. A row limit does not guarantee which rows are selected

Large and nested results

For a large table, or one containing nested data, set `result = "arrow_stream"` to process rows in batches instead of collecting them all into R memory. The stream is disk-backed and can be read only once, so enough temporary disk space must be available for the selected data. Release the stream deterministically when finished: call `stream[["release"]]()` when using 'nanoarrow' directly, or call `reader$Close()` after `arrow::as_record_batch_reader(stream)`. Do not rely on garbage collection to delete the staged file, particularly on Windows

A refreshable credential retries the entire read once after an authentication failure, including a failure while spooling an Arrow stream. Partial local output is discarded before retrying. Each attempt uses one token for its complete scan; credentials are not continuously replaced during long scans.

Column types

Common dates, timestamps, numbers, text, and logical values are converted to practical R types. Values that R cannot represent exactly, including decimal and 64-bit integer values, are returned as character data when collecting a tibble. Nested columns require an Arrow stream. The complete mapping is:

Delta/Arrow source	Arrow stream result	Tibl
Decimal (any precision/scale)	UTF-8 text	char
Large UTF-8 / large binary	original large-offset type	char
UTF-8 / binary views	UTF-8 / binary with 32-bit offsets	char
List views / large-list views	list / large list	reje
Large list	original large-offset list	reje
Signed/unsigned 64-bit integer	original integer type	exac
Signed 32-bit integer	original integer type	dou
Timestamp without timezone	original Arrow timestamp	char
Timestamp with timezone	original Arrow timestamp	UTC
Date, Boolean, floating point, smaller integers, UTF-8, binary	corresponding Arrow scalar	corr
Struct, map, list, extension/Variant	corresponding normalized Arrow type when supported	reje

Decimal text retains its scale and digits. Arrow view types are normalized for R compatibility; ordinary large-offset types retain their offsets

Permissions and supported tables

Direct reads require OneLake data access; item `Read` permission by itself is not enough. The caller needs `ReadAll` or a suitable OneLake security role, and the tenant setting for external OneLake apps must be enabled. Callers restricted by row- or column-level security must use a supported Fabric engine instead. See the Fabric permission model and OneLake tenant settings

This function uses the Python deltalake reader through 'reticulate'. Python 3.10 or newer and compatible Python 'deltalake' and 'nanoarrow' packages are required. Inspect the exact requirements with `fabric_delta_config()`, or call `fabric_delta_config(initialize = TRUE)` to initialize the runtime. 'reticulate' can download a managed environment on first use; an already initialized custom environment must provide the required packages itself.

Some newer Delta features, including Type Widening, V2 Checkpoints, and shredded Fabric Variant, are not supported by that reader. The reader can query an unshredded Variant table only when `columns` explicitly excludes every top-level column containing Variant values. It otherwise returns Variant's physical binary storage instead of decoded logical values, so this function rejects that projection. Use SQL or Spark (Livy) for Variant values or when the function reports another unsupported table feature

Compatible Warehouse tables can also be read through their published Delta logs. If the reader cannot open a Warehouse table, use `fabric_sql_query()`

Examples

```
## Not run:
# Discover a Lakehouse and one of its Delta tables
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
tables <- fabric_lakehouse_tables(lakehouse)
table <- tables[1L, ]

# Read the discovered table into a tibble
rows <- fabric_lakehouse_read_table(
  lakehouse = lakehouse,
  table = table
)

# Stream the same table when it may not fit in R memory
row_count <- local({
  stream <- fabric_lakehouse_read_table(
    lakehouse = lakehouse,
    table = table,
    result = "arrow_stream"
  )
  on.exit(nanoarrow::nanoarrow_pointer_release(stream), add = TRUE)
  reader <- arrow::as_record_batch_reader(stream)
  on.exit(reader$Close(), add = TRUE, after = FALSE)
  count <- 0
  repeat {
    batch <- reader$read_next_batch()
    if (is.null(batch)) break
    count <- count + batch$num_rows
  }
```

```
  count
})

## End(Not run)
```

---

fabric_onelake_schema_exists *Check whether a OneLake schema or table exists*

Description

Searches the paginated Delta metadata collections or retrieves one namespace or table record through the Iceberg REST Catalog API. These helpers avoid downloading table data when only existence is needed.

Usage

```
fabric_onelake_schema_exists(
  item,
  schema,
  workspace = NULL,
  item_type = NULL,
  protocol = c("delta", "iceberg"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_origin,
  storage_token = NULL
)

fabric_onelake_table_exists(
  item,
  table,
  workspace = NULL,
  schema = NULL,
  item_type = NULL,
  protocol = c("delta", "iceberg"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_origin,
  storage_token = NULL
)
```

Arguments

item	Fabric data item GUID, exact display name, or a discovered item object. An object containing workspaceId avoids workspace discovery.
schema	Schema or Iceberg namespace name. For a table, NULL uses the item's discovered default schema and otherwise falls back to "dbo".
workspace	Workspace GUID, exact display name, or discovered workspace. Omit it when item contains workspaceId.
item_type	Optional item type used to disambiguate an item supplied by name.
protocol	OneLake table metadata protocol: "delta" or "iceberg".
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
api_base	Fabric REST API base URL. Leave unchanged unless using a different Fabric cloud or a test service
table_api_base	OneLake table API HTTPS origin, or a protocol-specific base ending in /delta or /iceberg. Most users should keep the default.
storage_token	Optional separate Azure Storage token or token-provider function. Supply it when token is fixed and Fabric item lookup is needed.
table	Table name or a record containing name, table, or displayName. A record can also supply its schema.

Details

The table APIs use the Azure Storage token audience and require permission to read the item's tables through OneLake. If name-based item discovery is necessary, use the package's normal audience-aware sign-in or token provider because the Fabric Core and Storage audiences are both involved.

Iceberg requests first call GET /iceberg/v1/config with the item's workspace/item warehouse identity and validate the returned prefix before retrieving the namespace or table record. Delta requests follow all metadata collection pages because OneLake currently rejects its documented schema and table HEAD routes.

Value

One logical value. Delta returns TRUE when the paginated metadata inventory contains the requested name. Iceberg returns TRUE when the metadata GET succeeds and FALSE for HTTP 404. Authentication, permission, throttling, and service errors are not converted to FALSE.

References

OneLake table APIs for Delta

OneLake table APIs for Iceberg

Examples

```
## Not run:
lakehouse <- fabric_lakehouses(fabric_workspaces()[[1L]])[[1L]]

fabric_onelake_schema_exists(lakehouse, "dbo")
fabric_onelake_table_exists(lakehouse, "orders", schema = "dbo")
fabric_onelake_table_exists(
  lakehouse,
  "orders",
  schema = "dbo",
  protocol = "iceberg"
)

## End(Not run)
```

---

fabric_onelake_shortcuts

Manage OneLake shortcuts

Description

Lists, inspects, creates or updates, and deletes shortcuts on a Fabric item. Discovered Fabric items can be used directly as OneLake targets. A validated raw target list supports connection-backed shortcut types already configured in Fabric without copying data into R.

Usage

```
fabric_onelake_shortcuts(
  item,
  workspace = NULL,
  item_type = NULL,
  parent_path = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_onelake_shortcut_get(
  item,
  path,
  name,
  workspace = NULL,
  item_type = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
```

```
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_onelake_shortcut_create(
  item,
  path,
  name,
  target,
  workspace = NULL,
  item_type = NULL,
  target_workspace = NULL,
  target_path = NULL,
  target_item_type = NULL,
  conflict_policy = c("Abort", "GenerateUniqueName", "CreateOrOverwrite",
    "OverwriteOnly"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_onelake_shortcuts_bulk_create(
  item,
  shortcuts,
  workspace = NULL,
  item_type = NULL,
  conflict_policy = c("Abort", "GenerateUniqueName", "CreateOrOverwrite",
    "OverwriteOnly"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_onelake_shortcut_cache_reset(
  workspace,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
```

```
  auth_args = list(),
  api_base = .fabric_api_base
)

fabric_onelake_shortcut_delete(
  item,
  path,
  name,
  workspace = NULL,
  item_type = NULL,
  confirm = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base
)
```

Arguments

item	Destination Fabric item name, GUID, or object returned by a discovery function.
workspace	Workspace name, GUID, or discovery object containing item. May be omitted when item contains workspaceId.
item_type	Optional item type used to disambiguate a destination item supplied by name.
parent_path	Optional Files or Tables path from which listing starts. Fabric still returns shortcuts below that path exhaustively.
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
api_base	Fabric REST API base URL. Leave unchanged unless using a different Fabric cloud or a test service
path	Parent Files or Tables path where the shortcut exists or will be created. Local checks validate path syntax, while Fabric applies item- and workload-specific placement rules. For a table shortcut in a Lakehouse without schemas, use Tables. In a schema-enabled Lakehouse, use a schema path such as Tables/dbo. A schema shortcut instead lives under Tables and targets a folder containing multiple Delta tables.
name	Shortcut name.
target	A discovered Fabric item, its name or GUID, or a raw named shortcut target list. A raw target must contain exactly one documented key such as oneLake, adlsGen2, amazonS3, azureBlobStorage, googleCloudStorage, oneDriveSharePoint,

s3Compatible, or dataverse. Connection-backed targets must include the documented required fields, use a Fabric connection ID, and must not embed credentials. Local checks cover structure, required fields, identifier shape, and generic URL safety; they do not verify source-specific host/path rules or that a connection refers to the supplied location. A raw oneLake target may also include its documented optional connectionId. Fabric service validation is authoritative.

target_workspace
: Workspace containing a OneLake target. May be omitted when a discovered target contains workspaceId.

target_path
: Item-relative Files or Tables path for a OneLake target. Required when target is an item and unused for a raw target list.

target_item_type
: Optional Fabric item type used to disambiguate a OneLake target supplied by name.

conflict_policy
: "Abort" preserves an existing shortcut with the same path and name. "GenerateUniqueName" creates a uniquely named shortcut, "CreateOrOverwrite" creates or updates it, and "OverwriteOnly" updates an existing shortcut without creating one.

shortcuts
: For bulk creation, a non-empty list of shortcut request lists. Each request requires path, name, and target, accepts the same target companion fields as fabric_onelake_shortcut_create(), and may include a transform list. The only currently documented transform is csvToDelta; see Details.

confirm
: Logical. Deletion is disabled unless explicitly set to TRUE. Deleting a shortcut does not delete its destination data.

Details

Shortcut names, parent paths, and OneLake target paths follow Fabric's current shortcut limits: %, +, and non-ASCII characters are rejected. Other source- and destination-specific restrictions are intentionally left to Fabric so that newly supported connection types and rules remain usable.

For example, a table shortcut named orders uses path = "Tables" in a Lakehouse without schemas, or path = "Tables/dbo" in a Lakehouse with schemas; its target_path identifies one Delta table. A schema shortcut named sales uses path = "Tables" and a target such as Tables/sales containing multiple Delta tables. File shortcuts use Files or a folder beneath it and do not register tables.

Listing follows Fabric continuation links and tokens until every shortcut below parent_path is returned. Unknown target details and transform fields are preserved in list columns for forward compatibility.

Create is deliberately not replayed automatically because its POST outcome can be ambiguous after a transport failure. The default conflict policy is Fabric's non-destructive Abort; overwrite must be requested explicitly. Deletion is also not replayed automatically, and a 404 confirms that the shortcut link is already absent.

Bulk creation is a preview Fabric API. Its optional csvToDelta transform accepts includeSubfolders and a properties list containing delimiter, skipFilesWithErrors, and useFirstRowAsHeader. Supported delimiters are comma, space, tab, |, &, and ;. A bulk request returns a fabric_operation;

use fabric_operation_result() to retrieve the per-request statuses, created shortcuts, and errors after completion.

These Core REST APIs require OneLake.Read.All or OneLake.ReadWrite.All for reads, and OneLake.ReadWrite.All for create and delete. Fabric documents support for users, service principals, and managed identities. API scope is not sufficient by itself: listing or reading also requires item Read permission or OneLake Read permission on the destination path. Creating requires item Write or OneLake ReadWrite on the destination, plus Read access to the target path. Updating or deleting likewise requires item Write or destination-path OneLake ReadWrite permission.

Value

fabric_onelake_shortcuts() returns a tibble with one row per shortcut. fabric_onelake_shortcut_get() and fabric_onelake_shortcut_create() return the same one-row shape. fabric_onelake_shortcut_delete() returns TRUE invisibly after success. fabric_onelake_shortcut_cache_reset() returns a fabric_operation handle for either immediate or asynchronous completion.

References

OneLake shortcuts REST API

OneLake shortcut placement and limitations

Create table and schema shortcuts

OneLake shortcut security and path permissions

Create shortcuts in bulk

Reset shortcut cache

Examples

```
## Not run:
# Discover two Lakehouses in the same workspace
workspace <- fabric_workspaces()[[1L]]
lakehouses <- fabric_lakehouses(workspace)
destination <- lakehouses[[1L]]
source <- lakehouses[[2L]]
source_paths <- fabric_onelake_list(workspace, source, path = "Tables")
source_table <- source_paths[source_paths$is_directory, ][1L, ]

# Create a shortcut whose target came from the source Lakehouse listing
created <- fabric_onelake_shortcut_create(
  destination,
  path = "Files",
  name = "shared-orders",
  target = source,
  target_path = source_table$path[[1L]]
)

# List the folder, then fetch the created shortcut by its returned identity
fabric_onelake_shortcuts(destination, parent_path = "Files")
shortcut <- fabric_onelake_shortcut_get(
  destination,
```

```
  path = created$path[[1L]],
  name = created$name[[1L]]
)

# Delete that same discovered shortcut explicitly
fabric_onelake_shortcut_delete(
  destination,
  path = created$path[[1L]],
  name = created$name[[1L]],
  confirm = TRUE
)

## End(Not run)
```

`fabric_operation_status` *Monitor Microsoft Fabric long-running operations*

Description

Check, wait for, and retrieve the result of a Fabric operation that continues after its initiating request returns. Pass the operation handle returned by a 'fabricQueryR' function when possible. To resume work later, save the complete `Location` URL returned by Fabric. A bare operation ID can reconstruct only the core `/operations/{id}` route, not workload-scoped routes

Usage

```
fabric_operation_status(
  operation,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  respect_retry_after = TRUE,
  .sleep = Sys.sleep,
  .now = Sys.time
)

fabric_operation_wait(
  operation,
  poll_interval = NULL,
  timeout = 300,
  error_on_failure = TRUE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
```

```
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  .sleep = Sys.sleep,
  .now = Sys.time
)

fabric_operation_result(
  operation,
  wait = TRUE,
  poll_interval = NULL,
  timeout = 300,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  .sleep = Sys.sleep,
  .now = Sys.time
)
```

Arguments

`operation`	A `fabric_operation` handle, Fabric operation GUID, or operation state/result URL returned in a `Location` header
`tenant_id`	Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`
`client_id`	Entra application ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID
`token`	Optional access token or token-provider function. Leave `NULL` to let 'fabricQueryR' use its normal sign-in flow. A `fabric_operation` handle reuses its stored credential unless authentication arguments are supplied explicitly
`auth_args`	Additional sign-in options passed to `AzureAuth::get_azure_token()` when no token source is supplied
`api_base`	Fabric REST API base URL. Most users should keep the default
`respect_retry_after`	Whether to wait until Fabric's recommended next status-check time before making the request
`.sleep, .now`	Internal hooks for deterministic tests
`poll_interval`	Minimum seconds between status requests. `NULL` honors Fabric's `Retry-After` value and otherwise uses a two-second fallback
`timeout`	Positive maximum total seconds to wait, including status requests
`error_on_failure`	Whether a failed operation should raise a `fabric_operation_failed` condition. Set to `FALSE` to inspect the returned failed state directly

`wait` Whether to wait for a running operation. When `FALSE`, one state request is made immediately without honoring a stored future polling hint; a non-terminal operation raises `fabric_operation_not_ready`.

Value

`fabric_operation_status()` and `fabric_operation_wait()` return a `fabric_operation_state` record. `fabric_operation_result()` returns a `fabric_operation_result` with `value`, `content_type`, `empty`, HTTP and request identifiers, and the reusable operation handle. JSON results are decoded as lists, binary results are raw vectors, and empty results have a `NULL` value

Typical workflow

A package function that starts asynchronous work may return a `fabric_operation` handle. Use `fabric_operation_wait()` to wait for it to finish and `fabric_operation_result()` to retrieve its output. Result retrieval waits by default, so it is enough for the common case

If the R process restarts, save the service-provided `location` and pass it with fresh authentication arguments. A bare ID is sufficient only for core operations; after success, core operations are completed by reading the documented `/operations/{id}/result` resource

Results and failures

`fabric_operation_status()` preserves Fabric's status, progress, timestamps, request identifiers, and structured error. Status values added by Fabric in the future remain inspectable, but `fabric_operation_wait()` stops with a typed error instead of polling an unfamiliar value indefinitely. The documented `Undefined`, `NotStarted`, and `Running` values remain pending

Some workload APIs, including Lakehouse table loading, expose completion in their state response and do not provide a separate `/result` resource. For those operations, `fabric_operation_result()` returns the terminal state payload as its `value`

A failed operation raises `fabric_operation_failed` by default. A timeout raises `fabric_operation_timeout`; neither condition repeats the request that originally started the operation

Regional operation endpoints

Fabric can return a `Location` on a regional `*.analysis.windows.net` cluster. 'fabricQueryR' recognizes those Microsoft endpoints and automatically uses the Power BI token audience they require. Normal automatic sign-in or an audience-aware token-provider function handles both audiences. A single static Fabric bearer token cannot authenticate a regional operation URL

References

Get operation state

Get operation result

Regional Fabric LRO authentication example

Examples

```
## Not run:
# Discover a Lakehouse and a CSV file that Fabric can load as a table
workspace <- fabric_workspaces()[[1L]]
lakehouse <- fabric_lakehouses(workspace)[[1L]]
files <- fabric_onelake_list(
  workspace,
  lakehouse,
  path = "Files/incoming"
)
csv_file <- files[grepl("[.]csv$", files$path), ][1L, ]

# The load call returns the long-running operation handle used below
operation <- fabric_lakehouse_load_table(
  lakehouse,
  table = "orders_imported",
  path = csv_file$path[[1L]],
  format = "Csv",
  header = TRUE
)

# Check once, wait for completion, then retrieve the operation result
state <- fabric_operation_status(operation)
completed <- fabric_operation_wait(state$operation, timeout = 900)
result <- fabric_operation_result(completed$operation)
result$value

## End(Not run)
```

---

fabric_pbi_dax_query *Query a Microsoft Fabric/Power BI semantic model with DAX*

Description

Runs a DAX query against a published semantic model and returns the result as a tibble. A semantic model is the report-ready data behind Power BI reports, including tables, relationships, measures, and business calculations

Usage

```
fabric_pbi_dax_query(
  connstr = NULL,
  dax,
  workspace_id = NULL,
  dataset_id = NULL,
  my_workspace = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
```

```
  token = NULL,
  auth_args = list(),
  include_nulls = TRUE,
  api_base = "https://api.powerbi.com/v1.0/myorg",
  impersonated_user = NULL,
  api = c("json", "arrow"),
  result = c("tibble", "arrow_stream"),
  arrow_options = list(),
  timeout = 300
)
```

Arguments

connstr	Optional semantic model object from fabric_semantic_models() or fabric_item(), or a Power BI connection string. A character connection string can be, for example, "Data Source=powerbi://api.powerbi.com/v1.0/myorg/Workspace;Initial Catalog=Dataset;" It may contain Data Source= and Initial Catalog= parts, or a bare powerbi://... source plus a Dataset=, Catalog=, or Initial Catalog= key. Omit it when dataset_id is supplied Identity properties EffectiveUserName, Roles, and CustomData are rejected in connection strings. Supply impersonated_user or, with api = "arrow", the corresponding arrow_options instead
dax	One DAX query, normally beginning with EVALUATE. DAX table expressions determine which rows and columns are returned
workspace_id	Optional shared-workspace GUID. Use with dataset_id to avoid name-based discovery. For a model in My Workspace, omit this and set my_workspace = TRUE explicitly
dataset_id	Optional semantic model/dataset GUID. When supplied, no connection-string name lookup is performed
my_workspace	Whether dataset_id belongs to the signed-in user's My Workspace. Leave FALSE for shared workspaces
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
include_nulls	Logical. With TRUE, Power BI includes properties whose value is blank/null. With FALSE, those properties can be absent from a returned row; retaining TRUE usually gives a more consistent tibble Used only by api = "json"; Arrow has a schema and always represents nulls explicitly
api_base	Power BI REST API base URL. The default "https://api.powerbi.com/v1.0/myorg" is correct for the commercial cloud; override it only for a test service that implements the same endpoint and authentication contract. Sovereign Microsoft clouds are not currently supported by this helper

`impersonated_user`
: Optional user principal name, such as `"analyst@example.com"`, sent as `impersonatedUserName` for supported JSON row-level-security scenarios or as `effectiveUsername` for Arrow Leave `NULL` for the normal identity context

`api`
: Response format provided by Power BI. Use `"json"` for ordinary queries or `"arrow"` for richer types and multiple result tables

`result`
: Return a `"tibble"`, or with `api = "arrow"`, a single-use `"arrow_stream"` for batch processing without first collecting all rows in R memory

`arrow_options`
: Named list of optional `executeDaxQueries` request properties. Supported names are `applicationContext`, `culture`, `customData`, `effectiveUsername`, `executionMetrics`, `memoryLimit`, `queryTimeout`, `resultSetRowCountLimit`, `roles`, and `schemaOnly`. The required `query` property is supplied from `dax`. Used only by `api = "arrow"`

`timeout`
: Positive finite client-side timeout in seconds for the DAX execution HTTP request. This is distinct from the Arrow API's server-side `arrow_options$queryTimeout` property

Value

A tibble for one result table. Multiple Arrow result tables are returned as a `fabric_pbi_dax_rowsets` list of tibbles or Arrow streams Power BI column names are preserved. JSON result tables with no rows have no column metadata and return a zero-row, zero-column tibble. Arrow results preserve the column schema even when there are no rows. Missing results and service-reported errors or truncation raise an error. Arrow results respect `arrow_options$resultSetRowCountLimit`; the service default is 1,000,000 rows. Intentional row limits do not raise an error. Reaching the cap does not establish whether more rows exist. For complete extraction, verify expected row counts or query bounded partitions

Choosing a model

The easiest input is an item from `fabric_semantic_models()`. You can instead supply workspace and dataset IDs, or a Power BI connection string copied from the semantic model settings. IDs are the most reliable choice for scheduled code. For a model in My Workspace, supply `dataset_id` and set `my_workspace = TRUE` Tenant-qualified XMLA connection strings cannot be safely resolved by name through the tenant-relative REST API. For B2B access, omit `connstr`, supply `workspace_id` and `dataset_id`, and authenticate to the target tenant.

Choosing a response format

Keep `api = "json"` for ordinary queries and broad compatibility. It returns one result table and is available to Pro, PPU, and capacity-backed models Results are limited by Power BI; 'fabricQueryR' raises an error instead of silently returning a partial result. Very large whole numbers are returned as character values so they are not rounded. Mixed JSON scalar types form list columns. In those columns, an oversized number uses a `fabric_pbi_variant` cell with `type = "integer"` and an exact character `value`, distinguishing it from literal text with the same digits.

Use `api = "arrow"` when exact semantic-model types matter, when a query has several `EVALUATE` statements, or when you want an Arrow stream. It requires the optional 'arrow' package and a model on Premium or Fabric capacity. Decimal128 and Decimal256 columns are returned as exact character values in a tibble. Power BI Variant columns are returned as list-columns whose cells

contain `type` and `value` fields and inherit from `fabric_pbi_variant`, so mixed scalar types remain distinguishable. Variant Currency values are exact character scalars. Variant whole numbers are `bit64::integer64` scalars, except the minimum signed 64-bit value, which is character because 'bit64' reserves that bit pattern for missing values. `result = "arrow_stream"` retains native Arrow decimal and dense-union types. Null struct parents require `result = "arrow_stream"`; tibble collection raises `fabric_arrow_null_struct_error` to preserve their distinction from valid structs with all-null fields. The Power BI administrator must enable both **Dataset Execute Queries REST API** under Developer settings and **Allow XMLA endpoints and Analyze in Excel with on-premises semantic models** under Integration settings. Multiple result tables are returned in statement order as a `fabric_pbi_dax_rowsets` list

Permissions and tenant settings

The signed-in identity needs Read and Build permission on the semantic model Your Power BI administrator must enable **Dataset Execute Queries REST API**; service principals also need the relevant service-principal tenant setting. The Arrow endpoint has the additional XMLA tenant setting and capacity prerequisites described above The APIs use the Power BI scope and require `Dataset.Read.All` (or `Dataset.ReadWrite.All`). Name lookup also requires workspace read access Row-level security, SSO, user impersonation, and the Arrow endpoint have additional Power BI restrictions; see the linked Microsoft documentation

References

Power BI JSON Execute Queries REST API

Power BI Arrow Execute DAX Queries REST API

Power BI Arrow API overview and capacity requirements

Semantic model permissions

Semantic Model Execute Queries tenant setting

Examples

```
## Not run:
# Discover the semantic model instead of copying workspace and model IDs
workspace <- fabric_workspaces()[[1L]]
model <- fabric_semantic_models(workspace)[[1L]]

# Supply a query tested in the model's DAX query view
dax <- Sys.getenv("FABRIC_DAX_QUERY")

# Evaluate the DAX query and collect the result as a tibble
df <- fabric_pbi_dax_query(
  model,
  dax = dax
)
dplyr::glimpse(df)

# Keep a larger result out of R memory with an Arrow stream
stream <- fabric_pbi_dax_query(
  model,
```

```
  dax = dax,
  api = "arrow",
  result = "arrow_stream"
)
reader <- arrow::as_record_batch_reader(stream)

## End(Not run)
```

---

fabric_pbi_refresh *Refresh and monitor a Power BI semantic model*

Description

Start a semantic-model refresh, inspect recent refreshes and execution details, wait for completion, or cancel an enhanced refresh. The easiest target is an object returned by fabric_semantic_models()

Usage

```
fabric_pbi_refresh(
  connstr = NULL,
  workspace_id = NULL,
  dataset_id = NULL,
  my_workspace = FALSE,
  mode = c("automatic", "standard", "enhanced"),
  notify_option = NULL,
  type = NULL,
  commit_mode = NULL,
  objects = NULL,
  apply_refresh_policy = NULL,
  effective_date = NULL,
  max_parallelism = NULL,
  retry_count = NULL,
  timeout = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = "https://api.powerbi.com/v1.0/myorg",
  principal_type = c("auto", "delegated", "service_principal")
)

fabric_pbi_refresh_history(
  connstr = NULL,
  workspace_id = NULL,
  dataset_id = NULL,
  my_workspace = FALSE,
```

```
  top = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = "https://api.powerbi.com/v1.0/myorg"
)

fabric_pbi_refresh_status(
  refresh = NULL,
  connstr = NULL,
  workspace_id = NULL,
  dataset_id = NULL,
  my_workspace = FALSE,
  refresh_id = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = "https://api.powerbi.com/v1.0/myorg",
  .sleep = Sys.sleep,
  .now = Sys.time
)

fabric_pbi_refresh_wait(
  refresh,
  poll_interval = NULL,
  timeout = 1800,
  error_on_failure = TRUE,
  cancel_on_timeout = FALSE,
  cancel = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = "https://api.powerbi.com/v1.0/myorg",
  .sleep = Sys.sleep,
  .now = Sys.time
)

fabric_pbi_refresh_cancel(
  refresh = NULL,
  connstr = NULL,
  workspace_id = NULL,
  dataset_id = NULL,
```

```
  my_workspace = FALSE,
  refresh_id = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = "https://api.powerbi.com/v1.0/myorg"
)
```

Arguments

connstr	Optional semantic-model object from fabric_semantic_models() or fabric_item(), or a Power BI connection string. Omit it when dataset_id is supplied
workspace_id	Optional shared-workspace GUID. For a semantic model in My Workspace, omit this and set my_workspace = TRUE
dataset_id	Optional semantic-model/dataset GUID
my_workspace	Whether dataset_id belongs to the signed-in user's My Workspace. Leave FALSE for shared workspaces
mode	Refresh request kind. "automatic" chooses enhanced refresh when an enhanced option is supplied and standard refresh otherwise "standard" supports only notify_option; "enhanced" exposes processing controls and requires Premium, PPU, Embedded, or Fabric capacity
notify_option	Standard-refresh email behavior for delegated calls: "NoNotification", "MailOnFailure", or "MailOnCompletion". When omitted, automatic delegated 'AzureAuth' uses "NoNotification" because Power BI requires this field. Service-principal calls omit the field. With an opaque token or provider, supply this or identify the principal through principal_type. Omit this for enhanced refreshes
type	Enhanced processing type: "Full", "ClearValues", "Calculate", "DataOnly", "Automatic", or "Defragment"
commit_mode	Enhanced commit behavior. "Transactional" preserves the previous model if processing fails. "PartialBatch" commits commands separately and can leave partially refreshed or empty tables after failure
objects	Optional enhanced-refresh table or partition selection. Supply table names as a character vector, or records such as list(list(table = "Sales", partition = "2026"))
apply_refresh_policy	Whether an incremental refresh policy should be applied. TRUE is incompatible with commit_mode = "PartialBatch"
effective_date	Optional date-time used instead of the current date by an incremental refresh policy. Accepts a Date, POSIXt, or ISO 8601 string
max_parallelism	Optional positive whole number of parallel processing threads for an enhanced refresh
retry_count	Optional non-negative number of additional enhanced refresh attempts

timeout — In fabric_pbi_refresh(), an optional HH:MM:SS limit for each enhanced attempt; Power BI defaults to five hours per attempt and stops retries after 24 hours of elapsed runtime from the first attempt. In fabric_pbi_refresh_wait(), the maximum number of seconds to wait on the client before raising a separate client-side timeout

tenant_id — Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID

client_id — Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID

token — Optional access token or token-provider function. Leave NULL to use the package's normal sign-in flow. Refresh handles reuse their in-process credential unless new authentication arguments are supplied

auth_args — Additional sign-in options passed to AzureAuth::get_azure_token()

api_base — Power BI REST API base URL. The commercial-cloud default is normally correct

principal_type — Identity used for a standard refresh. "auto" distinguishes the package's automatic delegated and client-credential flows. A supplied token or provider is opaque, so either supply notify_option for a delegated call or set this to "service_principal". Enhanced refreshes do not use this setting

top — Maximum history entries to return. Power BI retains 20 to 60 recent entries, depending on their age

refresh — A fabric_pbi_refresh handle returned by fabric_pbi_refresh() or a fabric_pbi_refresh_detail Status and cancellation functions also accept a refresh GUID when the semantic-model target arguments are supplied; fabric_pbi_refresh_wait() requires a handle or detail because it has no separate target arguments

refresh_id — Alternative refresh GUID. Do not combine it with a handle or GUID supplied through refresh

.sleep, .now — Internal hooks for deterministic polling tests

poll_interval — Minimum seconds between checks. NULL honors the service retry hint and otherwise checks every two seconds

error_on_failure — Whether failed, timed-out, cancelled, or disabled refreshes raise a typed error. Use FALSE to inspect the returned detail

cancel_on_timeout — Whether a client-side wait timeout should request cancellation before raising its timeout error. Cancellation is available only for enhanced refreshes

cancel — Optional function checked between status updates. If it returns TRUE, 'fabricQueryR' requests cancellation and stops waiting. Cancellation is available only for enhanced refreshes

Value

fabric_pbi_refresh() returns a fabric_pbi_refresh handle Status and wait return a fabric_pbi_refresh_detail; history returns a fabric_pbi_refresh_history list. Cancel invisibly returns TRUE

Standard and enhanced refresh

A standard refresh processes the complete model with Power BI defaults and works on shared capacity, subject to the shared-capacity request quota. An enhanced refresh is selected when any processing option is supplied. It can target tables or partitions, retry, change commit behavior, and set an attempt timeout, but requires a capacity-backed model. Only one refresh can run for a semantic model at a time

Standard and service-principal refresh responses can expose the accepted refresh ID through `RequestId` rather than `x-ms-request-id` or `Location`. 'fabricQueryR' recognizes either response form. Standard-refresh status and waiting fall back to refresh history when request-specific execution details are unavailable. For a raw refresh ID, history also determines whether cancellation is supported before a DELETE request is sent. Cancellation is available only for enhanced refreshes. Cancellation DELETE requests are not replayed after ambiguous transport failures; a 404 confirms that the request is already absent.

`Transactional` is the safe commit default. `PartialBatch` can expose a partially refreshed model after failure and cannot apply an incremental refresh policy. Each retry receives its own attempt timeout, while Power BI limits the entire refresh including retries to 24 hours

Results and diagnosis

`fabric_pbi_refresh()` returns a reusable handle `fabric_pbi_refresh_status()` and `fabric_pbi_refresh_wait()` return a `fabric_pbi_refresh_detail` with `state`, service status fields, UTC times, processing objects, attempts, engine messages, parsed service errors, a browser `details_url`, and the untouched response in `raw`. When a standard refresh falls back to history, details are limited to the fields available there `fabric_pbi_refresh_history()` returns a list of the same detail records

Power BI can report a successful refresh with warnings, but Microsoft notes that the history and execution-detail REST APIs do not always include those warnings. When warning messages are returned, the normalized state is `CompletedWithWarnings`; otherwise use `details_url` to inspect the Fabric refresh-detail page

Permissions and service limits

Starting any refresh and cancelling an enhanced refresh require `Dataset.ReadWrite.All` and semantic-model Write permission. History and status accept `Dataset.Read.All` or `Dataset.ReadWrite.All`, but history callers still need model Write permission. A service principal may call the APIs when the tenant allows it and the principal has sufficient workspace/model access; email notification options do not apply to service-principal requests

Shared capacity permits at most eight scheduled and API refresh requests per day and does not support enhanced refresh. Capacity-backed models have no fixed API-refresh count but can queue or throttle under load. Enhanced-refresh cancellation is supported for Import and Composite models in Premium, PPU, Embedded, or Fabric capacity and requires Contributor, Member, or Admin workspace access

Direct Lake refresh is a usually short metadata framing operation, not an import of OneLake data. Automatic Direct Lake updates are enabled by default, so an explicit refresh can be unnecessary unless automatic updates are disabled or a controlled point-in-time frame is required

References

Refresh Dataset API

Enhanced refresh

Refresh history

Refresh execution details

Data refresh and capacity limits

How Direct Lake refresh works

Examples

```
## Not run:
# Discover the semantic model instead of copying workspace and model IDs
workspace <- fabric_workspaces()[[1L]]
model <- fabric_semantic_models(workspace)[[1L]]

# Start a refresh, inspect it once, then wait for completion
refresh <- fabric_pbi_refresh(model)
current <- fabric_pbi_refresh_status(refresh)
current$state
result <- fabric_pbi_refresh_wait(refresh, timeout = 1800)
result$state
result$details_url

# An active enhanced refresh can be cancelled when it is no longer needed
refresh_to_cancel <- fabric_pbi_refresh(
  model,
  mode = "enhanced",
  type = "Full"
)
fabric_pbi_refresh_cancel(refresh_to_cancel)

# Choose a table shown in the model, then refresh only that table
refresh_table <- Sys.getenv("FABRIC_PBI_TABLE")
sales_only <- fabric_pbi_refresh(
  model,
  mode = "enhanced",
  type = "Full",
  objects = refresh_table,
  retry_count = 1L,
  timeout = "02:00:00"
)
fabric_pbi_refresh_wait(sales_only)

# Finally, inspect recent refreshes for the same discovered model
history <- fabric_pbi_refresh_history(model, top = 10L)
history[[1]]$attempts

## End(Not run)
```

`fabric_sql_connect` *Connect to a Microsoft Fabric SQL target*

Description

Opens a 'DBI' connection to a Fabric Warehouse, Warehouse snapshot, Lakehouse, mirrored database, or SQL Database. Use the connection with familiar 'DBI' functions such as `DBI::dbListTables()` and `DBI::dbGetQuery()`

Usage

```
fabric_sql_connect(
  server,
  database = NULL,
  target_type = c("auto", "lakehouse", "warehouse", "sql_database",
    "sql_analytics_endpoint"),
  backend = c("odbc", "adbc"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
 odbc_driver = getOption("fabricqueryr.sql.driver", "ODBC Driver 18 for SQL Server"),
  adbc_driver = getOption("fabricqueryr.sql.adbc_driver", "mssql"),
  port = NULL,
  encrypt = "yes",
  trust_server_certificate = "no",
  timeout = 30L,
  read_only = FALSE,
  verbose = TRUE,
  max_tries = 3L,
  retry_delay = 5,
  ...
)
```

Arguments

`server`	A Fabric SQL server name, a complete connection string copied from the Fabric portal, or one Lakehouse, Warehouse, Warehouse snapshot, or SQL Database object returned by a discovery function. A discovered object is usually simplest because it also supplies the database name
`database`	Optional catalog/database. An explicit value overrides a database found in `server`. For a bare endpoint, supply the item database shown with its connection string in Fabric. If omitted, Warehouse and SQL analytics endpoints open Fabric's `master` context, which is useful for discovery but does not select the item's tables

`target_type` Kind of Fabric SQL item. Keep `"auto"` unless a custom hostname prevents 'fabricQueryR' from identifying it

`backend` Connection driver. Use `"odbc"` for ordinary 'DBI' work or `"adbc"` for a native Arrow path after installing its `mssql` driver

`tenant_id` Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`

`client_id` Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID

`token` Optional access token or token-provider function. Leave `NULL` to let 'fabricQueryR' use its normal sign-in flow

`auth_args` Additional sign-in options passed to `AzureAuth::get_azure_token()`

`odbc_driver` ODBC driver name. ODBC Driver 18 for SQL Server is the default

`adbc_driver` ADBC driver name or shared-library path. The separately installed ADBC Driver Foundry `mssql` driver version 1.5.0 or newer is the default requirement

`port` Optional TCP port. An explicit value overrides a port in `server`; otherwise the standard SQL port, 1433, is used

`encrypt` Whether the driver encrypts the connection. Keep the secure default, `"yes"`, for Fabric

`trust_server_certificate`
Whether to accept a server certificate without validating its trust chain. Keep the secure default, `"no"`, unless diagnosing a controlled test environment

`timeout` Non-negative whole-number login/connect timeout in seconds; `0` lets the driver use an unlimited or driver-specific timeout

`read_only` Whether to ask the driver for a read-only connection. This is a connection hint, not a replacement for Fabric or SQL permissions

`verbose` Logical. Show authentication, retry, and connection progress

`max_tries` Maximum attempts after temporary Fabric SQL failures

`retry_delay` Initial delay in seconds before retrying. Later retries wait progressively longer, up to 60 seconds

`...` Additional arguments forwarded to `DBI::dbConnect()`. The former named `access_token` argument is consumed here as a deprecated alias for `token` and is not forwarded. For ODBC, a caller-supplied `attributes` named list is merged with the package-managed `azure_token`; that protected attribute cannot be overridden. ODBC authentication, target, driver, and TLS options cannot be supplied through `...` because the package validates and constructs those settings before attaching the access token. This also excludes raw `.connection_string`, `DSN`, and `FileDSN` arguments ADBC defaults to `bigint = "integer64"`, so ordinary BIGINT values do not have to fit an R 32-bit integer. Supply another `bigint` policy explicitly through `...` if needed. Direct DBI reads with `integer64` cannot represent the minimum signed BIGINT because 'bit64' reserves that value for `NA`. Direct ODBC binding can misinterpret `integer64` parameters as doubles. Use `fabric_sql_query()` for its exact parameter handling, use ADBC, or supply character parameters with explicit SQL `bigint` casts.

Details

The easiest input is an item returned by `fabric_warehouses()`, `fabric_lakehouses()`, `fabric_mirrored_databases()`, or `fabric_sql_databases()`. You can also paste a SQL connection string from Fabric. Lakehouse and mirrored database SQL endpoints are read-only; use the source system, Spark, or another appropriate writer to change their data

Value

A live `DBIConnection`. Close it with `DBI::dbDisconnect()` when finished. For an ADBC connection with child results still registered, use `DBI::dbDisconnect(con, force = TRUE)` to release them immediately

Choosing a backend

`backend = "odbc"` is the default and works well for ordinary 'DBI' use. It requires Microsoft ODBC Driver 18 or newer. Use `backend = "adbc"` when you want a native Arrow result path, typically for larger analytical results.

Install the R packages 'DBI' and 'odbc' for ODBC, or 'DBI', 'adbi', and 'adbcdrivermanager' for ADBC. 'adbi' is archived on CRAN and is available from `https://r-dbi.r-universe.dev`; see `vignette("reading-data", package = "fabricQueryR")` for installation.

ADBC requires version 1.5.0 or newer of the external `mssql` driver, where Fabric Data Warehouse support was introduced. Install or update it separately with `dbc install mssql`. The connected driver must report its version through the standard ADBC information API

Connection and permissions

Discovery records and complete portal connection strings normally include the database. A bare server can omit `database` to open Fabric's `master` context. Transient connection failures are retried automatically. The user or application must have access through a workspace role or the item's **Manage permissions** settings; SQL permissions may further restrict data

References

Connect to a Fabric Warehouse or SQL analytics endpoint

Microsoft Entra authentication in Fabric Data Warehouse

Lakehouse SQL analytics endpoint

Download Microsoft ODBC Driver 18 for SQL Server

ADBC `mssql` driver changelog

Examples

```
## Not run:
# Discover a Warehouse so no server name or database ID is copied by hand
workspace <- fabric_workspaces()[[1L]]
warehouse <- fabric_warehouses(workspace)[[1L]]

# Open a 'DBI' connection, use it, and always disconnect when finished
con <- fabric_sql_connect(warehouse)
```

```
table <- DBI::dbListTables(con)[[1L]]
table <- DBI::dbQuoteIdentifier(con, table)
DBI::dbGetQuery(con, paste("SELECT TOP 10 * FROM", table))
DBI::dbDisconnect(con)

# The ADBC backend can return Arrow-native results when installed
adbc_con <- fabric_sql_connect(warehouse, backend = "adbc")
DBI::dbDisconnect(adbc_con)

## End(Not run)
```

`fabric_sql_connection_info` *Get connection details for a Fabric SQL item*

Description

Shows the server, database, port, and item type that 'fabricQueryR' will use for a Fabric SQL connection. Most users can pass a discovered item directly to `fabric_sql_connect()` and do not need to call this helper

Usage

```
fabric_sql_connection_info(
  server,
  database = NULL,
  target_type = c("auto", "lakehouse", "warehouse", "sql_database",
    "sql_analytics_endpoint"),
  port = NULL
)
```

Arguments

`server`	A Fabric SQL server name, a complete connection string copied from the Fabric portal, or one Lakehouse, Warehouse, Warehouse snapshot, or SQL Database object returned by a discovery function. A discovered object is usually simplest because it also supplies the database name
`database`	Optional catalog/database. An explicit value overrides a database found in `server`. For a bare endpoint, supply the item database shown with its connection string in Fabric. If omitted, Warehouse and SQL analytics endpoints open Fabric's `master` context, which is useful for discovery but does not select the item's tables
`target_type`	Kind of Fabric SQL item. Keep `"auto"` unless a custom hostname prevents 'fabricQueryR' from identifying it
`port`	Optional TCP port. An explicit value overrides a port in `server`; otherwise the standard SQL port, 1433, is used

Value

A fabric_sql_connection_info list with server, database, port, target_type, and source (whether the input was text or a discovery object). No connection is opened

Examples

```
## Not run:
# Discover a Warehouse object that already contains its SQL endpoint
workspace <- fabric_workspaces()[[1L]]
# `$warehouses()` calls fabric_warehouses()
warehouse <- workspace$warehouses()[[1L]]

# Inspect connection details without opening a database connection
info <- fabric_sql_connection_info(warehouse)
info[c("server", "database", "port", "target_type")]

## End(Not run)
```

---

fabric_sql_query *Run a parameterized query against Microsoft Fabric SQL*

Description

Runs one SQL query and returns its rows, opening and closing the connection automatically. Use fabric_sql_connect() instead when several operations should share a connection. Supply changing values through params rather than pasting them into the SQL text

Usage

```
fabric_sql_query(
  server,
  sql,
  params = NULL,
  result = c("tibble", "arrow_stream"),
  database = NULL,
  target_type = c("auto", "lakehouse", "warehouse", "sql_database",
    "sql_analytics_endpoint"),
  backend = c("odbc", "adbc"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
 odbc_driver = getOption("fabricqueryr.sql.driver", "ODBC Driver 18 for SQL Server"),
  adbc_driver = getOption("fabricqueryr.sql.adbc_driver", "mssql"),
  port = NULL,
  encrypt = "yes",
  trust_server_certificate = "no",
```

```
  timeout = 30L,
  read_only = FALSE,
  verbose = TRUE,
  max_tries = 3L,
  retry_delay = 5,
  idempotent = FALSE,
  numeric_policy = c("auto", "exact", "driver"),
  ...
)
```

Arguments

server	A Fabric SQL server name, a complete connection string copied from the Fabric portal, or one Lakehouse, Warehouse, Warehouse snapshot, or SQL Database object returned by a discovery function. A discovered object is usually simplest because it also supplies the database name
sql	One result-producing T-SQL `SELECT` statement, optionally beginning with a common-table-expression `WITH` clause. For DDL or DML, open a connection with `fabric_sql_connect()` and call `DBI::dbExecute()`. A Lakehouse SQL analytics endpoint is read-only and does not support `INSERT`, `UPDATE`, or `DELETE`
params	Optional list of values for ? placeholders in `sql`. Values are sent separately from the SQL text, which is safer and easier to quote correctly than building a query with `paste()`. Factors are bound as their character labels on both backends. For ODBC, `bit64::integer64` parameters are sent as exact decimal text and their placeholders are cast to `bigint` in SQL, preserving numeric operations and missing values. ADBC binds them natively. This normalization applies to this query helper; direct DBI calls on `fabric_sql_connect()` use the driver's parameter conversion.
result	Return a `"tibble"` for ordinary R analysis, or a single-use `"arrow_stream"`. ADBC streams retain native Arrow types. ODBC streams are converted from R data frames and cannot recover values lost by the driver. The 'adbi' driver may fetch the complete result before returning the stream, so this option does not guarantee bounded-memory retrieval. An Arrow stream owns its DBI result and connection until the stream is released; consume it promptly or release it explicitly with `nanoarrow::nanoarrow_pointer_release()`
database	Optional catalog/database. An explicit value overrides a database found in `server`. For a bare endpoint, supply the item database shown with its connection string in Fabric. If omitted, Warehouse and SQL analytics endpoints open Fabric's `master` context, which is useful for discovery but does not select the item's tables
target_type	Kind of Fabric SQL item. Keep `"auto"` unless a custom hostname prevents 'fabricQueryR' from identifying it
backend	Connection driver. Use `"odbc"` for ordinary 'DBI' work or `"adbc"` for a native Arrow path after installing its `mssql` driver
tenant_id	Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`
client_id	Microsoft Entra application/client ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID

token — Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow

auth_args — Additional sign-in options passed to AzureAuth::get_azure_token()

odbc_driver — ODBC driver name. ODBC Driver 18 for SQL Server is the default

adbc_driver — ADBC driver name or shared-library path. The separately installed ADBC Driver Foundry mssql driver version 1.5.0 or newer is the default requirement

port — Optional TCP port. An explicit value overrides a port in server; otherwise the standard SQL port, 1433, is used

encrypt — Whether the driver encrypts the connection. Keep the secure default, "yes", for Fabric

trust_server_certificate — Whether to accept a server certificate without validating its trust chain. Keep the secure default, "no", unless diagnosing a controlled test environment

timeout — Non-negative whole-number login/connect timeout in seconds; 0 lets the driver use an unlimited or driver-specific timeout

read_only — Whether to ask the driver for a read-only connection. This is a connection hint, not a replacement for Fabric or SQL permissions

verbose — Logical. Show authentication, retry, and connection progress

max_tries — Maximum attempts after temporary Fabric SQL failures

retry_delay — Initial delay in seconds before retrying. Later retries wait progressively longer, up to 60 seconds

idempotent — Logical. Set to TRUE only if running the entire statement a second time has no unwanted effect (usually a plain SELECT). This permits a retry when it is unclear whether Fabric executed the first attempt

numeric_policy — "auto" (default) uses "driver" for ODBC and "exact" for ADBC. Automatic ODBC conversion warns once per R session about possible numeric precision loss. Set "driver" explicitly to accept the driver's conversions without this warning, or use "exact" to reject unsafe ODBC results before fetching

"exact" preserves ADBC decimals as character and BIGINT as bit64::integer64, using character for columns containing the minimum BIGINT. INT columns containing -2147483648 use exact doubles. Nested lists retain character decimals and 64-bit integers, and double 32-bit integers. Null struct parents require result = "arrow_stream"; exact tibble collection raises fabric_arrow_null_struct_error to preserve their distinction from valid structs with all-null fields. ODBC rejects DECIMAL, NUMERIC, INT and BIGINT columns before fetching: its conversion can round or truncate values or turn valid integer boundaries into missing values. Cast these columns to varchar in SQL or use ADBC. "driver" explicitly accepts the backend's conversions, including possible rounding and missing values, for either output format. This policy applies to this query helper; direct DBI calls on fabric_sql_connect() use the selected driver's conversion settings

... — Additional arguments forwarded to DBI::dbConnect(). The former named access_token argument is consumed here as a deprecated alias for token and is not forwarded. For ODBC, a caller-supplied attributes named list is merged

with the package-managed azure_token; that protected attribute cannot be overridden. ODBC authentication, target, driver, and TLS options cannot be supplied through ... because the package validates and constructs those settings before attaching the access token. This also excludes raw .connection_string, DSN, and FileDSN arguments ADBC defaults to bigint = "integer64", so ordinary BIGINT values do not have to fit an R 32-bit integer. Supply another bigint policy explicitly through ... if needed. Direct DBI reads with integer64 cannot represent the minimum signed BIGINT because 'bit64' reserves that value for NA. Direct ODBC binding can misinterpret integer64 parameters as doubles. Use fabric_sql_query() for its exact parameter handling, use ADBC, or supply character parameters with explicit SQL bigint casts.

Value

With result = "tibble", a tibble containing the returned rows and column types determined by numeric_policy. With result = "arrow_stream", a single-use nanoarrow_array_stream for Arrow-compatible tools

Examples

```
## Not run:
# Discover the Warehouse that will receive the query
workspace <- fabric_workspaces()[[1L]]
warehouse <- fabric_warehouses(workspace)[[1L]]

# Discover and quote a table name through a short 'DBI' connection
con <- fabric_sql_connect(warehouse)
table <- DBI::dbListTables(con)[[1L]]
table <- DBI::dbQuoteIdentifier(con, table)
DBI::dbDisconnect(con)
sql <- paste("SELECT TOP 100 * FROM", table)

# Run the resulting read-only query and collect a tibble
result <- fabric_sql_query(warehouse, sql, backend = "adbc")

# Return Arrow-native batches instead of converting to a data frame
stream <- fabric_sql_query(
  warehouse,
  sql,
  backend = "adbc",
  result = "arrow_stream"
)
reader <- arrow::as_record_batch_reader(stream)
table <- reader$read_table()

## End(Not run)
```

fabric_sql_tables *Discover and read tables through a Fabric SQL endpoint*

Description

These helpers provide a target-independent metadata and table-read layer for Fabric SQL endpoints. They accept Lakehouse, Warehouse, Warehouse snapshot, and SQL Database objects, or the same direct server inputs as `fabric_sql_query()`. Discovery uses SQL catalog metadata views and is limited by the caller's SQL metadata permissions. Each discovery call uses one query, including column metadata when `detail = TRUE`, on one connection per attempt. The connection closes before the result is returned.

Usage

```
fabric_sql_tables(
  server,
  schema = NULL,
  detail = TRUE,
  database = NULL,
  target_type = c("auto", "lakehouse", "warehouse", "sql_database",
    "sql_analytics_endpoint"),
  backend = c("odbc", "adbc"),
  token = NULL,
  ...
)

fabric_sql_views(
  server,
  schema = NULL,
  detail = TRUE,
  database = NULL,
  target_type = c("auto", "lakehouse", "warehouse", "sql_database",
    "sql_analytics_endpoint"),
  backend = c("odbc", "adbc"),
  token = NULL,
  ...
)

fabric_sql_read_table(
  server,
  table,
  schema = NULL,
  columns = NULL,
  limit = NULL,
  result = c("tibble", "arrow_stream"),
  database = NULL,
  target_type = c("auto", "lakehouse", "warehouse", "sql_database",
```

```
    "sql_analytics_endpoint"),
  backend = c("odbc", "adbc"),
  token = NULL,
  ...
)
```

Arguments

server	Fabric SQL endpoint, portal connection string, or discovered SQL-capable item object.
schema	Optional schema filter. fabric_sql_read_table() defaults to the schema in a discovered table row, otherwise "dbo".
detail	Whether table or view discovery should retrieve column metadata.
database	Optional catalog/database. An explicit value overrides one discovered from server.
target_type	Kind of Fabric SQL target. Keep "auto" unless a custom hostname prevents automatic identification.
backend	SQL driver backend, either "odbc" or "adbc".
token	Optional SQL access token or audience-aware token-provider function.
...	Additional authentication, driver, endpoint, timeout, verbosity, and retry options passed to fabric_sql_query(). Query text, parameters, read-only status, and idempotency are controlled by these helpers and cannot be supplied here.
table	Table or view name, or a one-row data frame or named list containing name and optionally schema.
columns	Optional unique column names to project.
limit	Optional non-negative maximum number of rows to return.
result	Result representation for fabric_sql_read_table(); either a tibble or a single-use Arrow stream.

Value

fabric_sql_tables() and fabric_sql_views() return a tibble with object name, schema, full_name, type, optional view definition, list-column columns, and the unmodified discovery row in raw. fabric_sql_read_table() returns a tibble or nanoarrow_array_stream.

References

System information schema views

SQL module definitions

Fabric SQL analytics endpoints

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
warehouse <- fabric_warehouses(workspace)[[1L]]

tables <- fabric_sql_tables(warehouse, schema = "dbo")
rows <- fabric_sql_read_table(warehouse, tables[1L, ], limit = 1000)
views <- fabric_sql_views(warehouse)

## End(Not run)
```

`fabric_typed_items` *Typed Microsoft Fabric item discovery*

Description

These shortcuts cover an intentional subset of Microsoft Fabric item types; they are not an exhaustive list of the items that `fabric_items()` can discover. Each helper requests one exact type and has a corresponding FabricWorkspace method. Most retrieve workload connection details by default. Semantic Model and GraphQL helpers default to lightweight discovery because their executable targets are derived from list-level IDs and workspace fields. User Data Functions default to lightweight discovery because Microsoft limits detail retrieval to delegated user identities. Set `detail = TRUE` when the workload and identity support it

Usage

```
fabric_lakehouses(workspace, detail = TRUE, ...)

fabric_warehouses(workspace, detail = TRUE, ...)

fabric_warehouse_snapshots(workspace, detail = TRUE, ...)

fabric_mirrored_databases(workspace, detail = TRUE, ...)

fabric_sql_databases(workspace, detail = TRUE, ...)

fabric_semantic_models(workspace, detail = FALSE, ...)

fabric_eventhouses(workspace, detail = TRUE, ...)

fabric_kql_databases(workspace, detail = TRUE, ...)

fabric_notebooks(workspace, detail = TRUE, ...)

fabric_data_pipelines(workspace, detail = TRUE, ...)

fabric_spark_job_definitions(workspace, detail = TRUE, ...)
```

```
fabric_environments(workspace, detail = TRUE, ...)

fabric_graphql_apis(workspace, detail = FALSE, ...)
```

Arguments

workspace	Workspace name, ID, or object returned by fabric_workspaces(). A name is convenient for interactive use; an object avoids an extra lookup
detail	Whether to retrieve connection details as well as names and IDs. This takes more requests and may require additional permissions. For fabric_item(), NULL enriches every supported type except User Data Functions, whose detail endpoint does not support application identities. The typed Semantic Model, GraphQL, and User Data Function helpers also default to lightweight records
...	Authentication and API arguments forwarded to fabric_items() Do not supply type; each helper sets that value

Value

A list with one FabricItem object or type-specific R6 subclass per matching item. Each object contains common item metadata, applicable connection fields, and workload methods. See fabric_items() for details

Typed support matrix

Default detail is the value used when detail is omitted. FabricItem in the final column means that the typed helper and workload Get route are supported but no workload-specific R6 subclass is currently provided.

Helper	Fabric type	Default detail	R6 class
fabric_lakehouses()	Lakehouse	TRUE	FabricLakehouse
fabric_warehouses()	Warehouse	TRUE	FabricWarehouse
fabric_warehouse_snapshots()	WarehouseSnapshot	TRUE	FabricWarehouseSnapshot
fabric_mirrored_databases()	MirroredDatabase	TRUE	FabricMirroredDatabase
fabric_sql_databases()	SQLDatabase	TRUE	FabricSqlDatabase
fabric_semantic_models()	SemanticModel	FALSE	FabricSemanticModel
fabric_eventhouses()	Eventhouse	TRUE	FabricEventhouse
fabric_kql_databases()	KQLDatabase	TRUE	FabricKqlDatabase
fabric_notebooks()	Notebook	TRUE	FabricJobItem
fabric_data_pipelines()	DataPipeline	TRUE	FabricJobItem
fabric_spark_job_definitions()	SparkJobDefinition	TRUE	FabricJobItem
fabric_environments()	Environment	TRUE	FabricItem
fabric_user_data_functions()	UserDataFunction	FALSE	FabricItem
fabric_graphql_apis()	GraphQLApi	FALSE	FabricGraphQLApi

Choosing a helper

- fabric_lakehouses(), fabric_warehouses(), fabric_warehouse_snapshots(), and fabric_mirrored_databas

find data stores with `$sql_query()` (`fabric_sql_query()`) and other SQL methods; Lakehouses and mirrored databases can also be accessed through OneLake

- `fabric_sql_databases()` finds transactional Fabric SQL databases
- `fabric_semantic_models()` finds business models with `$dax_query()` (`fabric_pbi_dax_query()`) and refresh lifecycle methods
- `fabric_eventhouses()` and `fabric_kql_databases()` find real-time data stores with `$query()` (`fabric_kql_query()`) and `$read_table()` (`fabric_kql_read_table()`), plus ingestion and export methods
- `fabric_notebooks()` finds notebooks with job lifecycle and schedule methods
- `fabric_data_pipelines()` and `fabric_spark_job_definitions()` find the other executable items with the same job methods
- `fabric_environments()` finds reusable Spark runtime configurations
- `fabric_user_data_functions()` finds serverless Python function items
- `fabric_graphql_apis()` finds APIs configured in Fabric with `$query()` (`fabric_graphql_query()`), `$schema()` (`fabric_graphql_schema()`), and `$paginate()` (`fabric_graphql_paginate()`)

Filtering and returned fields

Each helper requests its exact Fabric item type and verifies that every returned object has that type. The objects otherwise keep all fields returned by Fabric, including fields added by the service in the future

Folder recursion, workspace-specific private-link routing, authentication, and `detail_errors` have the same behavior as in `fabric_items()`. With `detail = TRUE`, each helper calls its documented workload-specific Get API and preserves fields such as Spark job and Environment properties. The User Data Function detail endpoint supports delegated users but not service principals or managed identities; those callers can use `detail = FALSE`

References

List items REST API

List data pipelines

List Spark job definitions

List environments

List User Data Functions

Get data pipeline

Get Spark job definition

Get environment

Get User Data Function

Examples

```
## Not run:
# Discover a workspace once, then reuse its object for typed discovery
workspace <- fabric_workspaces()[[1]]

# Discover data items that feed the package's query and storage helpers
lakehouses <- fabric_lakehouses(workspace)
warehouses <- fabric_warehouses(workspace)
snapshots <- fabric_warehouse_snapshots(workspace)
mirrored_databases <- fabric_mirrored_databases(workspace)
sql_databases <- fabric_sql_databases(workspace)
semantic_models <- fabric_semantic_models(workspace)
eventhouses <- fabric_eventhouses(workspace)
kql_databases <- fabric_kql_databases(workspace)
graphql_apis <- fabric_graphql_apis(workspace)

# Each method calls the corresponding exported function
# fabric_lakehouse_tables()
lakehouses[[1L]]$tables()
# fabric_sql_connection_info()
warehouses[[1L]]$sql_connection_info()
# fabric_pbi_dax_query()
semantic_models[[1L]]$dax_query(
  dax = Sys.getenv("FABRIC_DAX_QUERY")
)

# Runnable methods call fabric_job_run() and fabric_job_wait()
notebook <- fabric_notebooks(workspace)[[1]]
pipeline <- fabric_data_pipelines(workspace)[[1]]
spark_job <- fabric_spark_job_definitions(workspace)[[1]]

notebook$wait(notebook$run(), timeout = 900)
pipeline$wait(pipeline$run(), timeout = 900)
spark_job$wait(spark_job$run(), timeout = 900)

# Discover supporting Spark and serverless-function items as well
environments <- fabric_environments(workspace)
functions <- fabric_user_data_functions(workspace)

## End(Not run)
```

`fabric_user_data_functions` *Discover Fabric User Data Functions*

Description

[Experimental]

Usage

```
fabric_user_data_functions(workspace, detail = FALSE, ...)
```

Arguments

workspace	Workspace name, ID, or object returned by fabric_workspaces(). A name is convenient for interactive use; an object avoids an extra lookup
detail	Whether to retrieve workload-specific details. Defaults to FALSE so service-principal and managed-identity callers can use Core item discovery.
...	Authentication and API arguments forwarded to fabric_items(). Do not supply type; this helper fixes it to "UserDataFunction".

Details

Finds User Data Function items in a workspace. The default detail = FALSE path uses Core item discovery and works with delegated users, service principals, and managed identities. Set detail = TRUE to call the workload-specific Get API, which currently supports delegated users only.

This helper is experimental because the package can verify only lightweight Core discovery through its service-principal development sandbox. Fabric's User Data Function create, update-definition, detailed Get, and delete APIs do not currently support service principals or managed identities, so the sandbox cannot provision and fully inspect a disposable User Data Function fixture for repeatable end-to-end coverage.

Value

A list of FabricItem objects for matching User Data Function items.

References

List User Data Functions

Get User Data Function

Create User Data Function

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
functions <- fabric_user_data_functions(workspace)

## End(Not run)
```

`fabric_warehouse_read_table` *Read a Microsoft Fabric Warehouse table*

Description

Provides the table-oriented read counterpart to `fabric_warehouse_write_table()`. It resolves the Warehouse like the writer, safely quotes the schema, table, and projected columns, and delegates query execution and type conversion to `fabric_sql_query()`. Use that lower-level function for filters, ordering, joins, aggregations, or other T-SQL.

Usage

```
fabric_warehouse_read_table(
  warehouse,
  table,
  workspace = NULL,
  schema = "dbo",
  columns = NULL,
  limit = NULL,
  result = c("tibble", "arrow_stream"),
  backend = c("odbc", "adbc"),
  numeric_policy = c("auto", "exact", "driver"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  verbose = TRUE,
  timeout = 30L,
  max_tries = 3L,
  retry_delay = 5,
  sql_token = NULL
)
```

Arguments

`warehouse`	A Warehouse object returned by `fabric_warehouses()` or `fabric_item()`, or its name or GUID when `workspace` is supplied.
`table`	Warehouse table name, or a record containing a `name`, `table`, or `displayName` field.
`workspace`	Workspace name, GUID, or discovery object containing `warehouse`. May be omitted when `warehouse` is a discovery object.
`schema`	Warehouse schema. Defaults to `"dbo"`; a table record can supply its schema when this argument is omitted.

columns	Optional unique column names to project.
limit	Optional non-negative maximum number of rows to return.
result	Return a "tibble" for ordinary R analysis, or a single-use "arrow_stream". ADBC streams retain native Arrow types. ODBC streams are converted from R data frames and cannot recover values lost by the driver. The 'adbi' driver may fetch the complete result before returning the stream, so this option does not guarantee bounded-memory retrieval. An Arrow stream owns its DBI result and connection until the stream is released; consume it promptly or release it explicitly with nanoarrow::nanoarrow_pointer_release()
backend	SQL connection backend, "odbc" or "adbc".
numeric_policy	"auto" (default) uses "driver" for ODBC and "exact" for ADBC. Automatic ODBC conversion warns once per R session about possible numeric precision loss. Set "driver" explicitly to accept the driver's conversions without this warning, or use "exact" to reject unsafe ODBC results before fetching "exact" preserves ADBC decimals as character and BIGINT as bit64::integer64, using character for columns containing the minimum BIGINT. INT columns containing -2147483648 use exact doubles. Nested lists retain character decimals and 64-bit integers, and double 32-bit integers. Null struct parents require result = "arrow_stream"; exact tibble collection raises fabric_arrow_null_struct_error to preserve their distinction from valid structs with all-null fields. ODBC rejects DECIMAL, NUMERIC, INT and BIGINT columns before fetching: its conversion can round or truncate values or turn valid integer boundaries into missing values. Cast these columns to varchar in SQL or use ADBC. "driver" explicitly accepts the backend's conversions, including possible rounding and missing values, for either output format. This policy applies to this query helper; direct DBI calls on fabric_sql_connect() use the selected driver's conversion settings
tenant_id	Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID
client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
api_base	Fabric REST API base used when a Warehouse name or GUID must be discovered.
verbose	Whether to report SQL connection progress.
timeout	Non-negative whole-number login/connect timeout in seconds; 0 lets the driver use an unlimited or driver-specific timeout
max_tries	Maximum attempts after temporary Fabric SQL failures
retry_delay	Initial delay in seconds before retrying. Later retries wait progressively longer, up to 60 seconds
sql_token	Optional separate Azure SQL token or token-provider function. Supply it when token is fixed rather than audience-aware.

Value

A tibble, or a single-use nanoarrow_array_stream when result = "arrow_stream".

Large results

Use backend = "adbc" with result = "arrow_stream" for a native Arrow result path that avoids conversion to an R data frame. The current result path through 'DBI' and 'adbi' may fetch the complete result before returning the stream, so use a selective query or limit when the result may exceed memory. The external ADBC mssql driver must be installed.

limit uses T-SQL TOP and does not define row order. Use fabric_sql_query() with an explicit ORDER BY when deterministic row selection matters.

References

Query a Fabric Warehouse

Fabric Warehouse connectivity

Examples

```
## Not run:
# Discover the Warehouse instead of copying its SQL connection details
workspace <- fabric_workspaces()[[1L]]
warehouse <- fabric_warehouses(workspace)[[1L]]

# Use 'DBI' metadata to discover an existing table in that Warehouse
con <- fabric_sql_connect(warehouse)
tables <- DBI::dbListTables(con)
DBI::dbDisconnect(con)
table <- tables[[1L]]

# Read a bounded selection into a tibble
orders <- fabric_warehouse_read_table(
  warehouse,
  table,
  backend = "adbc",
  limit = 1000
)

# Keep the result Arrow-native rather than converting it to a data frame
stream <- fabric_warehouse_read_table(
  warehouse,
  table,
  backend = "adbc",
  result = "arrow_stream"
)
reader <- arrow::as_record_batch_reader(stream)

## End(Not run)
```

fabric_warehouse_tables *Discover Microsoft Fabric Warehouse tables*

Description

Lists schemas and Delta-backed tables in a Fabric Warehouse through the read-only OneLake table metadata API. Set `detail = TRUE` to retrieve column metadata for every table. A returned row can be passed directly to `fabric_warehouse_read_table()`.

Usage

```
fabric_warehouse_tables(
  warehouse,
  workspace = NULL,
  schema = NULL,
  detail = TRUE,
  page_size = NULL,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  table_api_base = .fabric_onelake_table_base,
  storage_token = NULL
)
```

Arguments

`warehouse`	Warehouse GUID, exact display name, or one Warehouse object returned by `fabric_warehouses()`. A discovered object is recommended because it contains the workspace and item IDs.
`workspace`	Workspace GUID, exact display name, or discovered workspace. Omit it when `warehouse` is an object containing `workspaceId`.
`schema`	Optional Warehouse schema. When omitted, every schema is listed.
`detail`	Whether to retrieve per-table column metadata.
`page_size`	Optional maximum records requested per OneLake metadata page, from 1 to 100. All continuation tokens are followed.
`tenant_id`	Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`.
`client_id`	Entra application ID. Defaults to `FABRICQUERYR_CLIENT_ID`, then the Azure CLI application ID.
`token`	Optional access token or audience-aware token-provider function. Warehouse lookup can require a Fabric-audience token; table metadata uses a Storage-audience token.

auth_args	Additional sign-in options passed to AzureAuth::get_azure_token() when no token source is supplied.
api_base	Fabric REST API base URL used when a Warehouse name or GUID must be resolved. Most users should keep the default.
table_api_base	OneLake Delta table API base URL. Most users should keep the default.
storage_token	Optional separate Azure Storage token or token-provider function. Supply it when token is fixed and Warehouse lookup is needed.

Value

A tibble with table name, schema, full_name, type, format, location, timestamps, list-column columns, schema_metadata, and the unmodified OneLake raw record. fabric_raw is an empty list-column because Fabric does not expose a Warehouse counterpart to the Lakehouse List Tables REST route. Unknown future OneLake metadata remains available in raw.

Permissions

The OneLake table API uses the Azure Storage token audience and requires the calling identity to have permission to read tables in the Warehouse through OneLake. This permission is separate from Warehouse T-SQL ReadData permission.

References

Explore tables with OneLake catalog APIs

OneLake table APIs for Delta

Warehouse permissions

Examples

```
## Not run:
workspace <- fabric_workspaces()[[1L]]
warehouse <- fabric_warehouses(workspace)[[1L]]

tables <- fabric_warehouse_tables(warehouse)
orders <- fabric_warehouse_read_table(warehouse, tables[1L, ])

## End(Not run)
```

---

fabric_warehouse_write_table

Write an R or Arrow object to a Fabric Warehouse table

Description

Serializes a data frame, tibble, or Arrow object to bounded Parquet parts, stages them in a Lakehouse, and loads them into a Fabric Warehouse table. Existing tables use the Warehouse COPY INTO command. When creation or drop-based replacement is requested, CREATE TABLE AS SELECT (CTAS) creates and loads the table directly from the staged Parquet schema. Lazy Arrow inputs are consumed as record batches and are not first collected into an R data frame.

Usage

```
fabric_warehouse_write_table(
  warehouse,
  table,
  data,
  staging_lakehouse,
  workspace = NULL,
  staging_workspace = NULL,
  schema = "dbo",
  mode = c("Append", "Overwrite"),
  overwrite_method = c("Truncate", "Drop"),
  create_if_missing = FALSE,
  staging_root = "Files/fabricqueryr-staging",
  cleanup = TRUE,
  keep_staging_on_failure = TRUE,
  compression = "snappy",
  target_file_size = 512 * 1024^2,
  max_rows_per_file = NULL,
  backend = c("odbc", "adbc"),
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  dfs_base = "https://onelake.dfs.fabric.microsoft.com",
  verbose = TRUE,
  storage_token = NULL,
  sql_token = NULL
)
```

Arguments

warehouse	A Warehouse object returned by fabric_warehouses() or fabric_item(), or its name or GUID when workspace is supplied.
table	Destination table name.
data	A data frame, tibble, Arrow Table, RecordBatch, Dataset, Scanner, RecordBatchReader, Arrow 'dplyr' query, or Arrow-compatible array stream. Timestamp columns are staged at microsecond resolution in UTC, so Fabric infers datetime2. Timezone-free timestamps retain their wall-clock values and are interpreted as UTC; timezone-aware timestamps retain their instant. Nanosecond timestamps are rejected before upload: explicitly cast them to microseconds first, choosing how to handle any sub-microsecond precision.
staging_lakehouse	A Lakehouse object returned by fabric_lakehouses() or fabric_item(), or its name or GUID. Fabric does not support a Warehouse item as the OneLake source of COPY INTO, so a Lakehouse staging item is required.

workspace — Workspace name, GUID, or discovery object containing warehouse. May be omitted when warehouse is a discovery object.

staging_workspace — Workspace containing staging_lakehouse. Defaults to the Warehouse workspace. May be omitted when staging_lakehouse is a discovery object.

schema — Destination schema. Defaults to "dbo".

mode — "Append" adds rows. "Overwrite" replaces the table contents using overwrite_method.

overwrite_method — For mode = "Overwrite", "Truncate" preserves the existing table definition and loads it with COPY INTO; "Drop" drops and recreates the table from the staged Parquet schema with CTAS. Drop replacement also removes table-specific metadata such as constraints and grants. Ignored for append mode.

create_if_missing — Whether to create and load a missing destination with CTAS. The default preserves the previous requirement that append and truncate-overwrite targets already exist. Drop-overwrite recreates an existing table; set this argument to TRUE if it may be absent.

staging_root — Lakehouse path below Files/ used for temporary Parquet directories.

cleanup — Whether to remove remote staging after confirmed success.

keep_staging_on_failure — Whether to retain staged files after a confirmed pre-load failure. Staging is always retained when SQL execution might have reached the Warehouse.

compression — Parquet compression codec passed to Arrow.

target_file_size — Soft maximum size in bytes for each staged Parquet part. Fabric recommends files between 100 MB and 1 GB for Warehouse loads.

max_rows_per_file — Optional exact maximum rows per staged part.

backend — SQL connection backend, "odbc" or "adbc".

tenant_id — Microsoft Entra tenant ID. Defaults to FABRICQUERYR_TENANT_ID

client_id — Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID

token — Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow

auth_args — Additional sign-in options passed to AzureAuth::get_azure_token()

api_base — Fabric REST API base used when a Warehouse or staging Lakehouse name or GUID must be discovered.

dfs_base — OneLake service address. Most users should keep the default; a workspace-specific address discovered from Fabric is used when available

verbose — Whether to report SQL connection progress.

storage_token — Optional separate Azure Storage token or token-provider function. Supply it when token is fixed rather than audience-aware.

sql_token — Optional separate Azure SQL token or token-provider function. Supply it when token is fixed rather than audience-aware.

Details

Existing-table writes map input fields by ordinal position to quoted destination columns whose names must exactly match the names in `data`, including letter case. The writer checks the Warehouse catalog before any destructive SQL is issued. Decimal inputs require a decimal destination with at least the source scale and integer-digit capacity; timestamp and time inputs require matching temporal types with sufficient fractional precision. Integer and floating-point inputs require destinations that can represent their full source range and precision. In particular, `int64` to SQL `float` and Arrow `double` to SQL `real`, integer, or decimal types are rejected. Cast the input explicitly when a lossy conversion is intended. These schema checks do not validate every possible SQL conversion or individual value. With `create_if_missing = TRUE`, a missing table is created and populated by a single CTAS statement; Fabric infers its names and types from the staged Parquet files.

Truncate overwrite preserves the table definition. Drop overwrite recreates the table and therefore intentionally discards its previous constraints, indexes, permissions, and other table-level metadata. Both overwrite paths run in an explicit Warehouse transaction and roll back on a confirmed SQL failure.

`COPY INTO` authenticates to OneLake as the identity executing the SQL statement. That identity needs read access to the staged Lakehouse files and the Warehouse T-SQL permissions required by the selected mode, including the applicable bulk-load, DML, and DDL permissions. The identity used to stage and clean up files also needs OneLake write access to the staging folder. Microsoft requires Contributor or higher on both the source Lakehouse workspace and the target Warehouse workspace for OneLake COPY using the executing identity. Granular item or SQL grants alone do not satisfy this documented contract. Workspace Identity has a separate permission model; this writer does not select Workspace Identity credentials.

Local staging is always removed. Remote staging is removed only after a confirmed successful load unless `keep_staging_on_failure = FALSE` and the failure occurred before SQL execution. Retaining files after an ambiguous SQL error makes a retry or investigation possible without changing the source while `COPY INTO` might still be completing.

Value

A `fabric_warehouse_write_result` list containing destination and staging identifiers, row and byte counts, part paths, and cleanup state.

References

COPY INTO in Fabric Warehouse

Warehouse ingestion performance guidance

Transactions in Fabric Warehouse

Create tables in Fabric Warehouse

Query Parquet files in Fabric Warehouse

OneLake security access-control model

Warehouse permissions

Examples

```
## Not run:
```

```
# Discover both the destination Warehouse and staging Lakehouse
workspace <- fabric_workspaces()[[1L]]
warehouse <- fabric_warehouses(workspace)[[1L]]
staging <- fabric_lakehouses(workspace)[[1L]]

# Upload through OneLake staging and create a new Warehouse table
fabric_warehouse_write_table(
  warehouse,
  "orders_from_r",
  data.frame(id = 1:3, amount = c(10, 20, 30)),
  staging_lakehouse = staging,
  create_if_missing = TRUE
)

## End(Not run)
```

`fabric_workspaces` *Discover Microsoft Fabric workspaces*

Description

Returns the Fabric workspaces available to the signed-in user or application Use the result to choose a workspace for `fabric_items()` or one of the typed discovery helpers

Usage

```
fabric_workspaces(
  roles = NULL,
  prefer_workspace_endpoints = FALSE,
  tenant_id = Sys.getenv("FABRICQUERYR_TENANT_ID"),
  client_id = Sys.getenv("FABRICQUERYR_CLIENT_ID", unset =
    "04b07795-8ddb-461a-bbee-02f9e1bf7b46"),
  token = NULL,
  auth_args = list(),
  api_base = .fabric_api_base,
  output = c("r6", "list")
)
```

Arguments

`roles`	Optional workspace roles to include, such as `"Viewer"`, `"Contributor"`, `"Member"`, or `"Admin"`. Leave `NULL` to return every visible workspace
`prefer_workspace_endpoints`	Whether to request workspace-specific API and OneLake endpoints. When `TRUE`, each listed workspace is hydrated with Get Workspace because List Workspaces returns only the API endpoint. Keep `FALSE` unless your organization uses workspace-level private links
`tenant_id`	Microsoft Entra tenant ID. Defaults to `FABRICQUERYR_TENANT_ID`

client_id	Microsoft Entra application/client ID. Defaults to FABRICQUERYR_CLIENT_ID, then the Azure CLI application ID
token	Optional access token or token-provider function. Leave NULL to let 'fabricQueryR' use its normal sign-in flow
auth_args	Additional sign-in options passed to AzureAuth::get_azure_token()
api_base	Fabric REST API base URL. Leave unchanged unless using a different Fabric cloud or a test service
output	Discovery record representation. The default "r6" returns R6 objects with type-specific methods. Use "list" when a plain record is specifically required

Details

The caller needs permission to read Fabric workspaces. Discovery uses the Fabric API and requires Workspace.Read.All or Workspace.ReadWrite.All

Value

A list with one workspace object per visible workspace. With output = "r6", each object is a FabricWorkspace. With output = "list", each object is a fabric_workspace list. Both representations preserve all fields returned by Fabric

References

List workspaces REST API

Get workspace REST API

Workspace roles

Examples

```
## Not run:
# Sign in and list every Fabric workspace you can access
workspaces <- fabric_workspaces()

# Inspect a field before choosing a workspace
vapply(workspaces, `[[`, character(1), "displayName")
workspace <- workspaces[[1L]]
workspace$displayName

# Object methods call the corresponding exported functions
# workspace$items() -> fabric_items(workspace)
items <- workspace$items()
# workspace$lakehouses() -> fabric_lakehouses(workspace)
lakehouse <- workspace$lakehouses()[[1L]]
# lakehouse$tables() -> fabric_lakehouse_tables(lakehouse)
lakehouse$tables()

## End(Not run)
```

print.fabric_graphql_rows *Print collected GraphQL rows*

Description

Print collected GraphQL rows

Usage

```
## S3 method for class 'fabric_graphql_rows'
print(x, ...)
```

Arguments

x	A fabric_graphql_rows tibble returned by fabric_graphql_collect()
...	Additional arguments passed to the tibble print method

Value

x, invisibly

print.fabric_job *Print a submitted Fabric job*

Description

Print a submitted Fabric job

Usage

```
## S3 method for class 'fabric_job'
print(x, ...)
```

Arguments

x	A fabric_job handle returned by fabric_job_run()
...	Reserved for the print method

Value

x, invisibly

print.fabric_job_instance *Print Fabric job status*

Description

Print Fabric job status

Usage

```
## S3 method for class 'fabric_job_instance'
print(x, ...)
```

Arguments

x	A fabric_job_instance returned by fabric_job_status() or fabric_job_wait()
...	Reserved for the print method

Value

x, invisibly

print.fabric_job_schedule *Print a Fabric job schedule*

Description

Print a Fabric job schedule

Usage

```
## S3 method for class 'fabric_job_schedule'
print(x, ...)
```

Arguments

x	A fabric_job_schedule returned by a schedule function.
...	Reserved for the print method.

Value

x, invisibly.

print.fabric_kql_export_result

Print a KQL storage export result

Description

Print a KQL storage export result

Usage

```
## S3 method for class 'fabric_kql_export_result'
print(x, ...)
```

Arguments

x	A fabric_kql_export_result.
...	Unused.

Value

x, invisibly.

print.fabric_kql_ingestion

Print a tracked Kusto ingestion handle

Description

Print a tracked Kusto ingestion handle

Usage

```
## S3 method for class 'fabric_kql_ingestion'
print(x, ...)
```

Arguments

x	A fabric_kql_ingestion handle
...	Unused

Value

x, invisibly

print.fabric_kql_ingestion_status

Print tracked Kusto ingestion status

Description

Print tracked Kusto ingestion status

Usage

```
## S3 method for class 'fabric_kql_ingestion_status'
print(x, ...)
```

Arguments

x	A fabric_kql_ingestion_status record
...	Unused

Value

x, invisibly

print.fabric_kql_write_result

Print an Eventhouse R/Arrow write result

Description

Print an Eventhouse R/Arrow write result

Usage

```
## S3 method for class 'fabric_kql_write_result'
print(x, ...)
```

Arguments

x	A fabric_kql_write_result.
...	Unused.

Value

x, invisibly.

`print.fabric_pbi_refresh`

Print a submitted Power BI refresh

Description

Print a submitted Power BI refresh

Usage

```
## S3 method for class 'fabric_pbi_refresh'
print(x, ...)
```

Arguments

`x`	A `fabric_pbi_refresh` returned by `fabric_pbi_refresh()`
`...`	Reserved for the print method

Value

`x`, invisibly

`print.fabric_pbi_refresh_detail`

Print Power BI refresh details

Description

Print Power BI refresh details

Usage

```
## S3 method for class 'fabric_pbi_refresh_detail'
print(x, ...)
```

Arguments

`x`	A `fabric_pbi_refresh_detail` returned by a refresh status, wait, or history function
`...`	Reserved for the print method

Value

`x`, invisibly

Index

arrow::write_parquet(), *85*, *93*
as.list(), *3*
AzureAuth::get_azure_token(), *34*, *39*, *45*, *48*, *51*, *53*, *55*, *57*, *61*, *65*, *71*, *75*, *79*, *81*, *83*, *85*, *89*, *93*, *98*, *101*, *105*, *108*, *117*, *122*, *124*, *128*, *131*, *135*, *138*, *144*, *148*, *153*, *163*, *166*, *168*, *171*

DBI::dbConnect(), *148*, *153*
DBI::dbDisconnect(), *149*
DBI::dbExecute(), *152*
DBI::dbGetQuery(), *147*
DBI::dbListTables(), *147*

fabric_catalog_search, 33
fabric_catalog_search(), *3*
fabric_data_pipelines *(*fabric_typed_items*)*, 157
fabric_delta_config, 35
fabric_delta_config(), *89*, *126*
fabric_environments *(*fabric_typed_items*)*, 157
fabric_eventhouses *(*fabric_typed_items*)*, 157
fabric_eventhouses(), *74*, *78*, *81*, *83*
fabric_function_invoke, 38
fabric_graphql_apis *(*fabric_typed_items*)*, 157
fabric_graphql_apis(), *44*, *47*, *48*, *50*
fabric_graphql_collect, 41
fabric_graphql_collect(), *172*
fabric_graphql_cursor, 43
fabric_graphql_cursor(), *44*
fabric_graphql_paginate, 44
fabric_graphql_paginate(), *20*, *41*, *43*, *159*
fabric_graphql_query, 46
fabric_graphql_query(), *20*, *42*, *44*, *159*
fabric_graphql_schema, 50
fabric_graphql_schema(), *20*, *159*
fabric_item, 52
fabric_item(), *3–5*, *9*, *74*, *78*, *81*, *83*, *100*, *104*, *107*, *138*, *143*, *162*, *167*
fabric_items, 54
fabric_items(), *3*, *5*, *157–159*, *161*, *170*
fabric_job_cancel *(*fabric_job_run*)*, 58
fabric_job_cancel(), *23*, *56*
fabric_job_instances, 56
fabric_job_instances(), *23*, *68*
fabric_job_run, 58
fabric_job_run(), *22*, *61*, *172*
fabric_job_schedule_config, 67
fabric_job_schedule_config(), *23*, *63*, *66*
fabric_job_schedule_create *(*fabric_job_schedules*)*, 63
fabric_job_schedule_create(), *23*, *67*
fabric_job_schedule_delete *(*fabric_job_schedules*)*, 63
fabric_job_schedule_delete(), *23*
fabric_job_schedule_update *(*fabric_job_schedules*)*, 63
fabric_job_schedule_update(), *23*, *67*
fabric_job_schedules, 63
fabric_job_schedules(), *23*
fabric_job_status *(*fabric_job_run*)*, 58
fabric_job_status(), *22*, *56*, *173*
fabric_job_wait *(*fabric_job_run*)*, 58
fabric_job_wait(), *22*, *56*, *173*
fabric_kql_databases *(*fabric_typed_items*)*, 157
fabric_kql_databases(), *74*, *78*, *79*, *81*, *83*
fabric_kql_export, 69
fabric_kql_export(), *19*, *81*
fabric_kql_ingest, 73
fabric_kql_ingest(), *18*, *84*, *85*
fabric_kql_ingestion_status *(*fabric_kql_ingest*)*, 73
fabric_kql_ingestion_status(), *19*
fabric_kql_query, 78

fabric_kql_query(), *18*, *80, 81*, *159*
fabric_kql_read_table, 80
fabric_kql_read_table(), *18*, *159*
fabric_kql_tables, 82
fabric_kql_tables(), *18*
fabric_kql_write_table, 83
fabric_kql_write_table(), *18*, *75*, *80*
fabric_lakehouse_load_table *(fabric_lakehouse_tables)*, 90
fabric_lakehouse_load_table(), *11*
fabric_lakehouse_read_table, 88
fabric_lakehouse_read_table(), *10*
fabric_lakehouse_schemas *(fabric_onelake_catalog)*, 112
fabric_lakehouse_schemas(), *10*, *111*
fabric_lakehouse_table *(fabric_onelake_catalog)*, 112
fabric_lakehouse_table(), *10*
fabric_lakehouse_tables, 90
fabric_lakehouse_tables(), *10*, *89*, *114*
fabric_lakehouse_write_table *(fabric_lakehouse_tables)*, 90
fabric_lakehouse_write_table(), *11*, *88*, *94*
fabric_lakehouses *(fabric_typed_items)*, 157
fabric_lakehouses(), *89*, *92*, *97*, *100*, *104*, *107*, *113*, *117, 118*, *149*, *167*
fabric_livy_batch_attach *(fabric_livy_sessions)*, 106
fabric_livy_batch_submit, 96
fabric_livy_batch_submit(), *11*, *24, 25*, *101*
fabric_livy_batches *(fabric_livy_sessions)*, 106
fabric_livy_query, 100
fabric_livy_query(), *11*, *103*
fabric_livy_session, 103
fabric_livy_session(), *11*, *27, 28*, *96*, *101*, *108*
fabric_livy_session_attach *(fabric_livy_sessions)*, 106
fabric_livy_sessions, 106
fabric_mirrored_database_read_table *(fabric_mirrored_database_tables)*, 109
fabric_mirrored_database_read_table(), *17*
fabric_mirrored_database_schemas *(fabric_mirrored_database_tables)*, 109
fabric_mirrored_database_schemas(), *16*
fabric_mirrored_database_table *(fabric_mirrored_database_tables)*, 109
fabric_mirrored_database_table(), *16*
fabric_mirrored_database_tables, 109
fabric_mirrored_database_tables(), *17*
fabric_mirrored_databases *(fabric_typed_items)*, 157
fabric_mirrored_databases(), *110*, *149*
fabric_notebooks *(fabric_typed_items)*, 157
fabric_onelake_catalog, 112
fabric_onelake_delete *(fabric_onelake_files)*, 115
fabric_onelake_delete(), *13*
fabric_onelake_download *(fabric_onelake_files)*, 115
fabric_onelake_download(), *12*, *120*
fabric_onelake_files, 115
fabric_onelake_list *(fabric_onelake_files)*, 115
fabric_onelake_list(), *12*
fabric_onelake_metadata *(fabric_onelake_files)*, 115
fabric_onelake_metadata(), *12*, *117*
fabric_onelake_object_files, 120
fabric_onelake_read_delta_table, 123
fabric_onelake_read_delta_table(), *10*, *16, 17*, *89*, *118*
fabric_onelake_read_file *(fabric_onelake_object_files)*, 120
fabric_onelake_read_file(), *12*
fabric_onelake_schema_exists, 127
fabric_onelake_schema_exists(), *13*
fabric_onelake_shortcut_cache_reset *(fabric_onelake_shortcuts)*, 129
fabric_onelake_shortcut_cache_reset(), *8*
fabric_onelake_shortcut_create *(fabric_onelake_shortcuts)*, 129
fabric_onelake_shortcut_create(), *14*
fabric_onelake_shortcut_delete *(fabric_onelake_shortcuts)*, 129

fabric_onelake_shortcut_delete(), *14*
fabric_onelake_shortcut_get (fabric_onelake_shortcuts), 129
fabric_onelake_shortcut_get(), *14*
fabric_onelake_shortcuts, 129
fabric_onelake_shortcuts(), *13*
fabric_onelake_shortcuts_bulk_create (fabric_onelake_shortcuts), 129
fabric_onelake_shortcuts_bulk_create(), *14*
fabric_onelake_table_exists (fabric_onelake_schema_exists), 127
fabric_onelake_table_exists(), *13*
fabric_onelake_upload (fabric_onelake_files), 115
fabric_onelake_upload(), *13*, *94*, *98*, *120*
fabric_onelake_write_file (fabric_onelake_object_files), 120
fabric_onelake_write_file(), *12*
fabric_operation_result (fabric_operation_status), 134
fabric_operation_result(), *94*, *133*
fabric_operation_status, 134
fabric_operation_status(), *91*, *94*
fabric_operation_wait (fabric_operation_status), 134
fabric_operation_wait(), *94*
fabric_pbi_dax_query, 137
fabric_pbi_dax_query(), *21*, *159*
fabric_pbi_refresh, 141
fabric_pbi_refresh(), *21*, *176*
fabric_pbi_refresh_cancel (fabric_pbi_refresh), 141
fabric_pbi_refresh_cancel(), *21*
fabric_pbi_refresh_history (fabric_pbi_refresh), 141
fabric_pbi_refresh_history(), *21*
fabric_pbi_refresh_status (fabric_pbi_refresh), 141
fabric_pbi_refresh_status(), *21*
fabric_pbi_refresh_wait (fabric_pbi_refresh), 141
fabric_pbi_refresh_wait(), *21*
fabric_semantic_models (fabric_typed_items), 157
fabric_semantic_models(), *138*, *139*, *141*, *143*
fabric_spark_job_definitions (fabric_typed_items), 157
fabric_sql_connect, 147
fabric_sql_connect(), *150–153*, *163*
fabric_sql_connection_info, 150
fabric_sql_databases (fabric_typed_items), 157
fabric_sql_databases(), *149*
fabric_sql_query, 151
fabric_sql_query(), *111*, *126*, *148*, *154–156*, *159*, *162*, *164*
fabric_sql_read_table (fabric_sql_tables), 155
fabric_sql_read_table(), *111*
fabric_sql_tables, 155
fabric_sql_tables(), *111*
fabric_sql_views (fabric_sql_tables), 155
fabric_typed_items, *56*, 157
fabric_user_data_functions, 160
fabric_warehouse_read_table, 162
fabric_warehouse_read_table(), *15*, *165*
fabric_warehouse_schemas (fabric_onelake_catalog), 112
fabric_warehouse_schemas(), *15*
fabric_warehouse_snapshots (fabric_typed_items), 157
fabric_warehouse_table (fabric_onelake_catalog), 112
fabric_warehouse_table(), *15*
fabric_warehouse_tables, 165
fabric_warehouse_tables(), *15*, *111*, *114*
fabric_warehouse_write_table, 166
fabric_warehouse_write_table(), *15*, *162*
fabric_warehouses (fabric_typed_items), 157
fabric_warehouses(), *114*, *149*, *162*, *165*, *167*
fabric_workspaces, 170
fabric_workspaces(), *3*, *52*, *54*, *117*, *121*, *124*, *158*, *161*
FabricEventhouse (FabricItem), 3
FabricGraphQLApi (FabricItem), 3
FabricItem, 3, *9*, *14*, *16*, *17*, *19*, *20*, *22*, *35*, *53*, *54*, *56*, *158*, *161*
FabricJobItem (FabricItem), 3
FabricKqlDatabase (FabricItem), 3

FabricKqlItem, *19*
FabricKqlItem *(*FabricItem*)*, 3
FabricLakehouse *(*FabricItem*)*, 3
FabricLivyBatch, *11*, 24, *98*, *108*
FabricLivySession, *11*, 27, *31*, *105*, *108*
FabricLivyStatement, *29*, 31
FabricMirroredDatabase *(*FabricItem*)*, 3
FabricSemanticModel *(*FabricItem*)*, 3
FabricSqlDatabase *(*FabricItem*)*, 3
FabricWarehouse *(*FabricItem*)*, 3
FabricWarehouseSnapshot *(*FabricItem*)*, 3
FabricWorkspace, *35*, *157*, *171*
FabricWorkspace *(*FabricItem*)*, 3

I(), *40*, *45*, *47*, *60*

nanoarrow::nanoarrow_pointer_release(), *152*, *163*

print.fabric_graphql_rows, 172
print.fabric_job, 172
print.fabric_job_instance, 173
print.fabric_job_schedule, 173
print.fabric_kql_export_result, 174
print.fabric_kql_ingestion, 174
print.fabric_kql_ingestion_status, 175
print.fabric_kql_write_result, 175
print.fabric_pbi_refresh, 176
print.fabric_pbi_refresh_detail, 176

R6::R6Class, *3*, *24*, *27*, *31*
reticulate::py_require(), *36*
reticulate::use_virtualenv(), *37*