



DEVELOPER DOCUMENTATION

Groundzero Standard Runtime

The default Python and SQL runtime for Groundzero datasets and jobs.

Developer documentation

Edition 2026-10-02

Runtime environments: Spark, DuckDB, Standard, Lakehouse

Contents

Guide - Standard Runtime

1	Overview	4
1.1	What Standard is	5
1.2	How it runs	5
1.3	Versions	5
1.4	Choose the Standard runtime	6
1.5	The Python (pandas) job runtime	6
1.6	The base runtime and custom runtimes	6
2	Quickstart	7
2.1	1. Pick a connection	7
2.2	2. Write the source query	7
2.3	3. Add Prep code	7
2.4	4. Run and save	8
2.5	5. Read it from code	8
3	How-to guides	11
3.1	Read from a connection	11
3.2	Query lakehouse tables	11
3.3	Work with datasets, outputs and previews	12
3.4	Use parameters and placeholders	12
3.5	Filter without changing the query text	13
3.6	Reshape results with Prep code	13
3.7	Schedule with ETL jobs	14
3.8	Install extra packages (custom runtime)	14
3.9	Tune performance	14
3.10	Logging and errors	14
4	API reference	15
4.1	Prep code contract	15
4.2	Data request	15
4.3	Data response	16
4.4	Structured filters	16
4.5	Error messages	17
5	Installed packages	18
5.1	Python (pandas) job runtime and base runtime	18
5.2	Standard query engine	18
6	Limits and troubleshooting	19
6.1	Limits and quotas	19
6.2	Known issues	19
6.3	Troubleshooting	19
6.4	FAQ	20
7	Examples gallery	21
7.1	1. Shipped orders in the last quarter	21
7.2	2. Revenue by region with a Prep rollout	21

7.3	3. Monthly trend	21
7.4	4. Largest order per region	22
7.5	5. Pivot status counts by region	22
7.6	6. Orders owned by the signed-in user	22
7.7	7. A single row of headline numbers	22
7.8	8. Customers with their segment (join in SQL)	23
7.9	9. Defensive Prep for empty results	23
7.10	10. Test Prep locally before you save it	23

Appendix A - Choosing a Runtime and Shared Concepts

A.1	Choosing a runtime	24
A.1.1	At a glance	25
A.1.2	Decision guide	25
A.1.3	Where the choice is made	26
A.2	How a run starts	27
A.2.1	From Run to results	27
A.2.2	Datasets, jobs and notebooks compared	27
A.2.3	Sessions and warm starts	27
A.2.4	Timeouts and retries	28
A.2.5	Compute sizes, workers and GPU size	28
A.3	What your code can use during a run	30
A.3.1	Run details	30
A.3.2	Job placeholders	30
A.3.3	Imported models	30
A.4	Connections and secrets	31
A.5	The lakehouse (Iceberg)	32
A.6	Custom runtimes	33
A.7	ML variants	35
A.8	Jobs: lifecycle, status and logs	36
A.8.1	Statuses	36
A.8.2	Logs	36
A.8.3	Scheduling	36
A.9	Glossary	37

GUIDE

Standard Runtime

The default Python and SQL runtime for Groundzero datasets and jobs.

-
- 1 Overview
 - 2 Quickstart
 - 3 How-to guides
 - 4 API reference
 - 5 Installed packages
 - 6 Limits and troubleshooting
 - 7 Examples gallery

The Standard runtime is Groundzero's default, zero-infrastructure way to turn a connection into a dataset. You write one read-only query in the connection's own dialect, optionally add a few lines of pandas "Prep" code, and Groundzero returns pages of rows. You size no cluster, start no session and import no helper library.

WHAT \"STANDARD\" MEANS IN GROUNDZERO

The word covers three related things, and this guide keeps them apart.

- **Standard datasets** (the main subject of this guide): the *Standard* choice in the dataset designer. The query runs in Groundzero's managed query engine, not on a runtime you start.
- **The Python (pandas) job runtime**: a runtime for jobs and code datasets, for plain Python and pandas workloads.
- **The base runtime**: the Python 3.12 foundation that the Spark and DuckDB runtimes, and every custom runtime, are built on.

Spark and DuckDB each ship a Groundzero helper library for use in code. Standard has none: its programming surface is SQL plus the Prep contract described in the API reference.

1 Overview

1.1 What Standard is

A Standard dataset has two parts that you write in the dataset designer.

Part	Editor tab	Language	Runs
Source query	Query	SQL (or connector query)	At the source
Prep code	Prep	Python with pandas	In the query engine

When you press Run, or when a chart, dashboard or API client reads the saved dataset, the engine does the following.

1. Checks that you hold Execute permission on the dataset.
2. Substitutes server-side placeholders such as `${USR}` into the query.
3. Applies any structured filters to the query.
4. Looks for a cached answer for the same query and the same access policy.
5. Otherwise runs the query on the connection and fetches the page of rows.
6. Runs your Prep code, if it is more than a pass-through.
7. Returns one page of rows with a schema, and records source lineage.

Because the engine is shared and stateless there is nothing to warm up. A Standard run has no compute size. Startup cost is the connection round trip plus the query itself.

1.2 How it runs

When you run a Standard dataset, Groundzero looks for a cached answer, then runs your query on the connection, applies your Prep code to the fetched rows and returns a page of rows with its schema.

Key properties:

- **One connection per dataset.** The designer offers Standard only when zero or one connection is selected, and only when that connection is a SQL source or the Groundzero lakehouse.
- **Server-side paging.** For SQL connectors the page window is pushed into the source query, so a page view does not fetch the whole table.
- **Shared caching.** Identical requests are answered from a cache that is checked against the caller's access policy on every hit.
- **Lakehouse reads.** Groundzero lakehouse (Iceberg) tables are read directly from object storage.

1.3 Versions

Component	Version	Used for
Python	3.12	Prep code
DuckDB	1.5.5	Lakehouse (Iceberg) reads

Component	Version	Used for
pandas	Installed, version not pinned	Prep code
Python (pandas) job runtime	Python 3.12, NumPy 2.3.5, pandas 2.x	Jobs

1.4 Choose the Standard runtime

Choose Standard when:

- the data lives in one SQL-capable connection and the source can do the heavy work (joins, filters, aggregation);
- you want a dataset for charts, dashboards, pivots, exports or API reads;
- the result is small enough to page through, and any reshaping is a few lines of pandas;
- you want an instant first result with no session start.

Choose another runtime when:

You need	Use
Joins across several connections	DuckDB or Spark
Writes to the lakehouse or other sinks	DuckDB or Spark job
Datasets larger than memory, wide shuffles	Spark
Machine learning with scikit-learn, PyTorch or TensorFlow	A DuckDB or Spark ML variant
Extra Python packages	A custom runtime

1.5 The Python (pandas) job runtime

Jobs and code datasets can also run on a *pandas* runtime. Groundzero starts the runtime you select with the compute size you choose and runs your job code there.

No Groundzero Python helper package is provided for it; connect to sources with the drivers that come with the runtime (see Installed packages). Use the runtime picker to see which pandas runtimes are available to you.

1.6 The base runtime and custom runtimes

Every Groundzero runtime starts from the same base: Python 3.12 with Jupyter, MLflow, the AWS command line and common database client libraries. The Spark and DuckDB runtimes add their engines on top. A *custom runtime* adds your own pip packages on top of any of those runtimes, see the how-to guide Install extra packages.

2 Quickstart

This walk-through builds a Standard dataset of shipped orders by region. It uses a sample `sales.orders` table.

2.1 1. Pick a connection

You need a connection to a SQL source or to the Groundzero lakehouse. The examples assume a sample connection named `SalesWarehouse` that exposes a `sales.orders` table with these columns.

Column	Type	Example
order_id	integer	1001
customer_id	integer	7
customer	text	Northwind Traders
region	text	EMEA
amount	decimal	1250.00
quantity	integer	5
status	text	shipped
order_date	date	2026-03-14
sales_rep	text	alice

2.2 2. Write the source query

SQL

```
SELECT order_id, customer, region, amount, status, order_date
FROM sales.orders
WHERE status = 'shipped'
      AND order_date ≥ DATE '2026-01-01'
```

2.3 3. Add Prep code

Prep receives the fetched rows as `queryData` (a list of dictionaries) and must assign its result to `outputDF`.

PYTHON

```
orders = pd.DataFrame(queryData)
outputDF = (
    orders.groupby("region", as_index=False)
    .agg(orders=("order_id", "count"), revenue=("amount", "sum"))
    .sort_values("revenue", ascending=False)
)
```

LEAVE PREP EMPTY WHEN YOU DO NOT NEED IT

An empty Prep tab, or the single line `outputDF = queryData`, is treated as a pass-through. Groundzero then skips running Prep, keeps server-side paging, and trusts the source schema.

2.4 4. Run and save

Run the dataset. The result grid shows one page of rows. Save the dataset to use it in visualizations, dashboards, pivots and exports.

2.5 5. Read it from code

Any client that can reach the Groundzero data API can run the same request. The body below is what the designer sends.

JSON

```
{
  "ConnectionName": "SalesWarehouse",
  "datasetId": 42,
  "DataSourceQuery": "SELECT region, amount FROM sales.orders",
  "TransformationQuery": "outputDF = queryData",
  "pageSize": 100,
  "pageNumber": 1
}
```

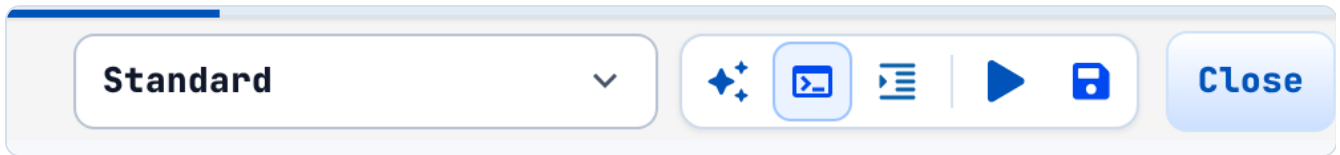
POST it to `/getdata` with your normal `Authorization` header. A successful answer looks like this.

JSON

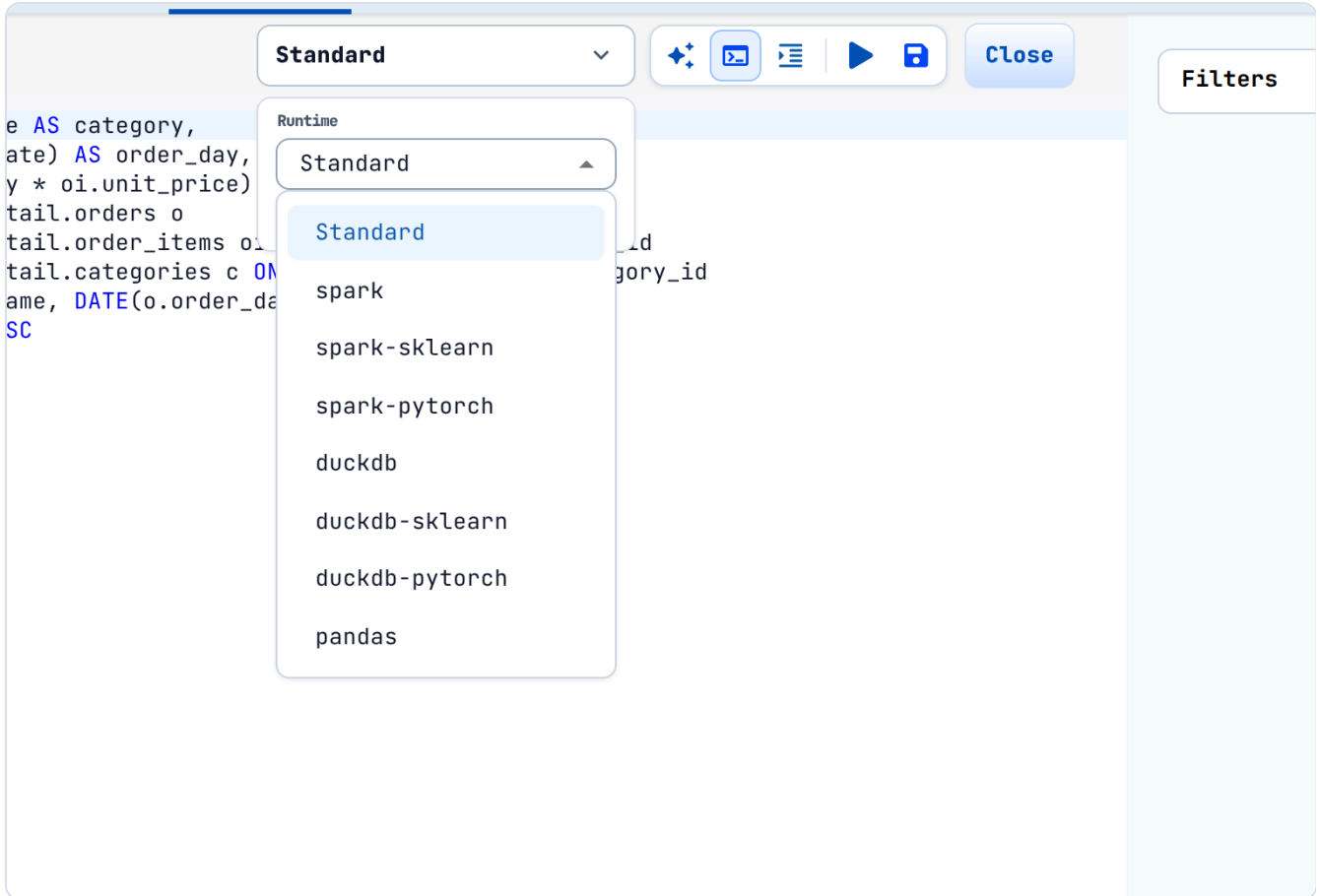
```
{
  "status": "success",
  "data": [{"region": "EMEA", "amount": 1250.0}],
  "totalRecords": 6,
  "hasMore": true,
  "pageStart": 1,
  "pageEnd": 1,
  "pageNumber": 1,
  "schema": [
    {"columnName": "region", "dataType": "text"},
    {"columnName": "amount", "dataType": "decimal"}
  ]
}
```

In the UI

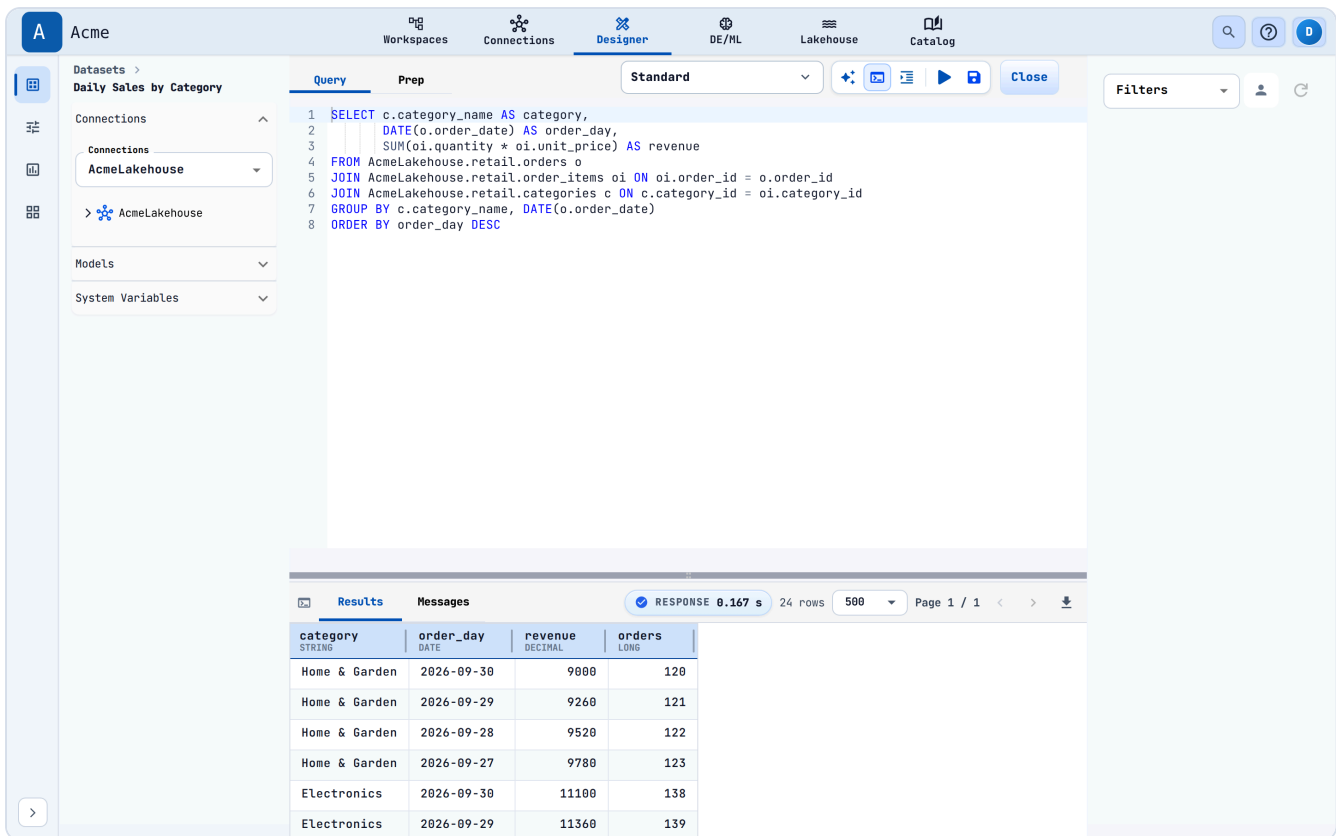
In the dataset designer, the runtime control sits beside the connection picker. **Standard** is the first entry in its Runtime list and is selected by default. When Standard is selected the Compute list is hidden, because Standard has no compute size to choose. Selecting a second connection, or a connection that is not a SQL source or the lakehouse, removes Standard from the list and asks you to choose a runtime and a compute size.



The dataset toolbar with Standard selected.



The Runtime list in the dataset designer: Standard first, then the installed runtimes.



The screenshot displays the Groundzero Standard Runtime interface. The top navigation bar includes tabs for Workspaces, Connections, Designer (active), DE/NL, Lakehouse, and Catalog. The left sidebar shows a tree view with 'Acme' selected, containing 'Daily Sales by Category', 'Connections', 'AcmeLakehouse', 'Models', and 'System Variables'. The main area is split into a 'Query' editor and a 'Results' grid. The query editor shows a SQL query for 'Daily Sales by Category'. The 'Results' grid shows the output of the query, with columns for category, order_day, revenue, and orders. The grid displays 24 rows of data, with the first 6 rows visible. The response time is 0.167 s.

Query:

```
1 SELECT c.category_name AS category,
2       DATE(o.order_date) AS order_day,
3       SUM(o.quantity * oi.unit_price) AS revenue
4 FROM AcmeLakehouse.retail.orders o
5 JOIN AcmeLakehouse.retail.order_items oi ON oi.order_id = o.order_id
6 JOIN AcmeLakehouse.retail.categories c ON c.category_id = oi.category_id
7 GROUP BY c.category_name, DATE(o.order_date)
8 ORDER BY order_day DESC
```

Results:

category	order_day	revenue	orders
STRING	DATE	DECIMAL	LONG
Home & Garden	2026-09-30	9800	120
Home & Garden	2026-09-29	9260	121
Home & Garden	2026-09-28	9520	122
Home & Garden	2026-09-27	9780	123
Electronics	2026-09-30	11100	138
Electronics	2026-09-29	11360	139

A Standard dataset run: the SQL editor, the Results grid, row count and response time.

3 How-to guides

3.1 Read from a connection

Standard supports the SQL sources below. Write the query in the source's own SQL dialect.

Family	Connections
Relational	SQL Server, PostgreSQL, MySQL, Oracle, DB2, SAP HANA, TimescaleDB
Warehouses	Snowflake, Redshift, Google BigQuery, Athena, Teradata, Exasol, Firebolt, SkySQL
Document / search	Cosmos DB, Elasticsearch, Cassandra
Lakehouse	Groundzero lakehouse

SQL

```
SELECT customer, SUM(amount) AS revenue
FROM sales.orders
GROUP BY customer
HAVING SUM(amount) > 1000
ORDER BY revenue DESC
```

Rules of thumb:

- Qualify tables with their schema (`sales.orders`). Lineage uses the qualified name.
- Put filters, joins and aggregations in the SQL. The source is better at them than Prep code.
- Select only the columns you need.

OTHER CONNECTION TYPES

Sources that are not SQL (for example object storage queries) use their own query body. They run in a DuckDB or Spark runtime rather than Standard; the designer will not offer Standard for them.

3.2 Query lakehouse tables

Create a connection of type Groundzero lakehouse, then query tables as `schema.table`.

SQL

```
SELECT region, COUNT(*) AS orders, SUM(amount) AS revenue
FROM sales.orders
WHERE order_date ≥ DATE '2026-01-01'
GROUP BY region
```

Reads are scoped to the caller: the lakehouse access policies that apply to you also apply to your result, and cached rows are only reused after the policy check passes.

STANDARD READS, IT DOES NOT WRITE

A Standard dataset runs a read query. To write to lakehouse tables, use a DuckDB or Spark job.

3.3 Work with datasets, outputs and previews

A run returns one page. The fields to know:

- `totalRecords` : rows available across all pages.
- `hasMore` : true while more pages remain.
- `pageStart` , `pageEnd` , `pageNumber` : the window you received.
- `schema` : one entry per column with `columnName` and `dataType` .

Request the next page by increasing `pageNumber` . Page size defaults to 1000 and is capped at 1000 rows.

JSON

```
{
  "ConnectionName": "SalesWarehouse",
  "DataSourceQuery": "SELECT * FROM sales.orders ORDER BY order_id",
  "TransformationQuery": "outputDF = queryData",
  "pageSize": 500,
  "pageNumber": 3
}
```

ALWAYS ORDER BY WHEN PAGING

Pages are cut from the result in the order the source returns it. Add an explicit `ORDER BY` so that page 2 never repeats rows from page 1.

Saved datasets can be used as the source of visualizations and dashboards, and appear in the lineage graph with their source tables.

3.4 Use parameters and placeholders

Two kinds of placeholder exist.

Server-side placeholders are replaced by the engine before the query runs.

Placeholder	Becomes
<code>\${USR}</code>	The signed-in user name, as a quoted string
<code>\${USRGRPS}</code>	The user's groups, comma-joined, as one quoted string
<code>\${SITENM}</code>	The site name, as a quoted string
<code>\${CLNTNM}</code>	The client name, as a quoted string
<code>\${RGN}</code>	The region, as a quoted string
<code>\${STRTTM}</code>	The request time <code>YYYY-MM-DD HH:MM:SS</code> , wrapped for code (see the warning below)

SQL

```
SELECT order_id, customer, amount
FROM sales.orders
WHERE sales_rep = ${USR}
```

WRITE THE PLACEHOLDER WITHOUT QUOTES

`${USR}`, `${SITENM}`, `${CLNTNM}`, `${RGN}` and `${USRGRPS}` already expand to a single-quoted literal. Writing `'${USR}'` produces doubled quotes and a syntax error. `${STRTTM}` expands to the timestamp wrapped in double and single quotes, which is meant for Prep code (a Python string), not for SQL.

SINGLE BRACES ARE REPLACED TOO

The engine also replaces `{USR}`, `{SITENM}`, `{CLNTNM}`, `{RGN}`, `{USRGRPS}`, `{STRTTM}` and `{ERR}` written with single braces. Avoid those exact names in Python format strings in Prep code.

Filter placeholders `{{filter.prop}}` are filled in by the dashboard or visualization that reads your dataset. If a filter has no value, the run fails with `Filter 'name' has no value` before it reaches the source.

3.5 Filter without changing the query text

Dashboards can send filters as data. Groundzero adds them to the query's WHERE clause using parsed expressions, never string concatenation, and keeps cache entries stable across filter changes.

JSON

```
{
  "ConnectionName": "SalesWarehouse",
  "DataSourceQuery": "SELECT o.region, o.amount FROM sales.orders AS o",
  "TransformationQuery": "outputDF = queryData",
  "pageSize": 100,
  "filters": [
    {"column": "region", "table": "o", "op": "in", "values": ["EMEA", "APAC"]},
    {"column": "amount", "table": "o", "op": "gte", "values": [100]}
  ]
}
```

When `filters` is present, send the base query without those filters. Fixed predicates may stay in the query. See the API reference for the operators.

3.6 Reshape results with Prep code

Prep is ordinary Python with pandas available as `pd`.

PYTHON

```
orders = pd.DataFrame(queryData)
orders["order_date"] = pd.to_datetime(orders["order_date"])
orders["month"] = orders["order_date"].dt.strftime("%Y-%m")
outputDF = orders.groupby("month", as_index=False)["amount"].sum()
```

`outputDF` may be a DataFrame, a Series, a list of dictionaries, a list of plain values, a dictionary (one row) or a single value. Anything that is not a DataFrame or list of dictionaries is converted to rows (see the API reference).

3.7 Schedule with ETL jobs

A Standard dataset answers requests on demand; it is not itself scheduled. To run work on a schedule, create an ETL job on the Python (pandas), DuckDB or Spark runtime and attach a schedule. A scheduled run may be given a stop time; the run then ends at that moment. Job runs go through the statuses Queued, Running, Success, Failed and Aborted, and write logs you can read from the job's run page.

3.8 Install extra packages (custom runtime)

Standard datasets cannot import additional packages. For jobs, create a custom runtime.

1. Prepare a `requirements.txt` with one pip requirement per line.
2. Upload it in the custom runtime screen and choose the base runtime to extend and a tag name.
3. Groundzero builds the runtime from that base and installs your packages.
4. Watch the build status move from QUEUED to IN PROGRESS and then SUCCESS or FAILURE.
5. Choose the new runtime when you create the job.

TEXT

```
splink  
sqlglot  
snowflake-connector-python
```

Pin versions (`package=1.2.3`) for repeatable builds.

3.9 Tune performance

Order of effect, biggest first.

1. **Push work to the source.** Filter, join and aggregate in SQL.
2. **Keep Prep a pass-through when possible.** Real Prep must see every source row, so Groundzero fetches the full result before it paginates. A pass-through keeps server-side paging.
3. **Select fewer columns and rows.** Cached payloads larger than 8 MiB are not cached.
4. **Use structured filters** so one cached base serves many filter choices.
5. **Prefer the lakehouse connection** for large tables: reads are columnar and go straight to object storage.
6. For results beyond what fits in a page-by-page workflow, move to DuckDB or Spark.

3.10 Logging and errors

Standard returns errors in the response body with `status: "error"`, an `errorCode` and a `message`. An exception inside Prep code returns the exception text, with `totalRecords` set to the number of source rows and an empty page. Test Prep logic locally with a small list of dictionaries first (see the last example in the gallery). For jobs, open the run's log view.

4 API reference

Standard has no Groundzero Python helper library. Its programming surface is the Prep contract and the data request.

4.1 Prep code contract

Prep is executed as a Python program. It sees these names.

Name	Type	Meaning
<code>queryData</code>	list of dict	Rows fetched by the source query, one dictionary per row
<code>pd</code>	module	pandas
<code>json</code>	module	The standard JSON module
<code>outputDF</code>	any	You must assign this. The result

Normalisation of `outputDF` into rows.

You assign	Rows returned
DataFrame	One row per DataFrame row
Series	A one-column frame named after the Series (or <code>value</code>)
List of dictionaries	Unchanged
List of other values	One row per item with column <code>value</code>
Dictionary	A single row
Any other value	A single row with column <code>value</code>

Raises. If Prep throws, or never assigns `outputDF`, the response is `{"status": "error", "message": "<exception text>"}`. A missing `outputDF` reports `'outputDF'`.

Pass-through. Empty Prep and `outputDF= queryData` (whitespace and a trailing semicolon ignored) skip execution entirely.

Example

PYTHON

```
rows = pd.DataFrame(queryData)
outputDF = rows[rows["amount"] > 100]
```

4.2 Data request

POST `/getdata` with a JSON object body.

Field	Type	Required	Description
ConnectionName	string	yes	Name of a connection you may use
DataSourceQuery	string	yes	Source query, placeholders allowed
TransformationQuery	string	yes	Prep code, <code>outputDF = queryData</code> for none
datasetId	integer	no	Saved dataset id; enables permission check and lineage
pageSize	integer	no	Rows per page, default 1000, maximum 1000
pageNumber	integer	no	1-based page, default 1
filters	array	no	Structured filters, below

4.3 Data response

Field	Type	Description
status	string	success, error or Running
data	array	Rows for the page
totalRecords	integer	Rows across all pages
hasMore	boolean	More pages remain
pageStart, pageEnd	integer	1-based row window
pageNumber	integer	Page returned
schema	array	columnName and dataType per column
errorCode, message		Present when status is error

4.4 Structured filters

Each element of `filters`:

Field	Required	Meaning
column	yes	Column name, or dotted path such as <code>o.region</code> (max 128)
table	no	Table alias used in the query; not together with a dotted column
op	yes	in, eq, between, gte, gt, lte, lt
values	yes	Scalars: string, number or boolean

Rules: `between` takes exactly two values; `eq`, `gte`, `gt`, `lte` and `lt` take one; `in` takes 1 to 5000. Strings are limited to 1024 characters. `null`, objects, arrays and non-finite numbers are rejected. Filters combine with AND after any existing WHERE. A malformed filter fails with 400 and never runs the unfiltered query:

JSON

```
{  
  "status": "error",  
  "errorCode": 400,  
  "message": "Invalid filters: filters[0].op must be a filter op"}  
}
```

4.5 Error messages

Source	Message
Standard	Request body must be an object (400)
Standard	datasetId must be a positive integer (400)
Standard	Invalid filters: ... (400)
Standard	Filter 'region' has no value (400)
Standard	You do not have "All" or "execute" permission on Dataset.
Standard	Connection type not supported. Please check the dataset.

5 Installed packages

5.1 Python (pandas) job runtime and base runtime

Python 3.12.

Package	Version
Python	3.12
NumPy	2.3.5
pandas	2.2 or newer, below 3
SQLAlchemy	unpinned
boto3	unpinned
urllib3	unpinned
matplotlib	unpinned
PyMySQL	unpinned
jupyter_server, ipython, ipykernel, jupyterlab	unpinned
MLflow	2.20.4

System tools: the AWS command line, `git`, `curl`, `wget`, `unzip`, `build-essential`, ODBC (`unixodbc`), PostgreSQL and MySQL client libraries, and `tzdata`.

5.2 Standard query engine

Package	Version
Python	3.12
DuckDB	1.5.5
pandas	unpinned

6 Limits and troubleshooting

6.1 Limits and quotas

Limit	Value
Rows per page	1000 (larger values are reduced to 1000)
Connections per Standard dataset	1
Filter values in one <code>in</code>	5000
String filter value	1024 characters
Largest cached result	8 MiB
Cached result lifetime	About 15 minutes (with random jitter up to 20 percent shorter)
Policy and access cache	60 to 120 seconds
Wait for a code dataset's result	300 seconds

6.2 Known issues

- A Prep that does real work forces a full fetch before paging, so very large results can be slow or run out of memory. Push work into the SQL.
- Oracle and Teradata queries are fetched in full before paging, because their dialects have no safe way to wrap a page window.

6.3 Troubleshooting

Symptom	Cause	Fix
<code>'outputDF'</code>	Prep never assigned <code>outputDF</code>	End Prep with <code>outputDF = ...</code>
Filter <code>'region'</code> has no value	A <code>{{region.prop}}</code> placeholder was left empty	Give the filter a value or remove the placeholder
Syntax error near quotes around a user name	Wrote <code>'\${USR}'</code>	Write <code>\${USR}</code> without quotes
<code>datasetId</code> must be a positive integer	<code>datasetId</code> was text, zero or negative	Send a positive integer or omit it for ad-hoc queries
You do not have "All" or "execute" permission on Dataset.	Missing dataset permission	Ask the dataset owner for Execute
Connection type not supported. Please check the dataset.	The connection type has no Standard connector	Choose DuckDB or Spark
Page 2 repeats rows	No <code>ORDER BY</code>	Add a stable <code>ORDER BY</code>

Symptom	Cause	Fix
Empty frame, <code>KeyError</code> in Prep	Zero rows came back	Guard with <code>if not queryData:</code>
Toast <code>Please select Compute</code>	Chose a DuckDB or Spark runtime without a compute size	Pick a compute size, or choose Standard
Toast <code>Unable to load runtimes</code>	The runtime list request failed	Reload; check your permissions
The dataset session stopped responding before it took the run. Run the dataset again to start a new session.	A dataset session died	Run the dataset again
Query execution timed out.	A DuckDB or Spark run produced no output within 300 seconds	Reduce data, or use a larger compute size

6.4 FAQ

Which library does Standard use? SQL and the Prep contract. The Spark and DuckDB runtimes provide helper libraries.

Why is Standard missing from my runtime list? Standard is offered only for datasets, with at most one connection, and only for SQL sources and the lakehouse.

Does Standard use my compute quota? Standard datasets do not use a compute size.

Can two users see each other's cached rows? Only when the access policy permits sharing, and every hit is re-authorized first.

Can Prep call external services or read files? Treat Prep as a pure transformation of `queryData`. For anything else, use a job runtime.

Which timestamp does `$_STRTTM` use? The time the request reached Groundzero.

7 Examples gallery

All examples use the sample `sales.orders` table from the Quickstart.

7.1 1. Shipped orders in the last quarter

SQL

```
SELECT order_id, customer, region, amount, order_date
FROM sales.orders
WHERE status = 'shipped'
      AND order_date ≥ DATE '2026-01-01'
ORDER BY order_date, order_id
```

7.2 2. Revenue by region with a Prep rollout

SQL

```
SELECT region, amount, status
FROM sales.orders
```

PYTHON

```
orders = pd.DataFrame(queryData)
shipped = orders[orders["status"] == "shipped"]
outputDF = (
    shipped.groupby("region", as_index=False)["amount"]
        .sum()
        .rename(columns={"amount": "revenue"})
        .sort_values("revenue", ascending=False)
)
```

7.3 3. Monthly trend

SQL

```
SELECT order_date, amount
FROM sales.orders
ORDER BY order_date
```

PYTHON

```
orders = pd.DataFrame(queryData)
orders["month"] = pd.to_datetime(orders["order_date"]).dt.strftime("%Y-%m")
outputDF = orders.groupby("month", as_index=False)["amount"].sum()
```

7.4 4. Largest order per region

SQL

```
SELECT order_id, region, customer, amount
FROM (
  SELECT o.*, ROW_NUMBER() OVER (
    PARTITION BY region ORDER BY amount DESC
  ) AS rn
  FROM sales.orders AS o
) ranked
WHERE rn = 1
ORDER BY region
```

7.5 5. Pivot status counts by region

SQL

```
SELECT region, status, order_id
FROM sales.orders
```

PYTHON

```
orders = pd.DataFrame(queryData)
outputDF = (
  orders.pivot_table(
    index="region", columns="status", values="order_id",
    aggfunc="count", fill_value=0,
  )
  .reset_index()
)
outputDF.columns.name = None
```

7.6 6. Orders owned by the signed-in user

SQL

```
SELECT order_id, customer, amount, status
FROM sales.orders
WHERE sales_rep = ${USR}
ORDER BY order_id
```

7.7 7. A single row of headline numbers

SQL

```
SELECT amount, quantity, status
FROM sales.orders
```

PYTHON

```
orders = pd.DataFrame(queryData)
outputDF = {
    "orders": int(len(orders)),
    "revenue": float(orders["amount"].sum()),
    "units": int(orders["quantity"].sum()),
    "shipped_share": float((orders["status"] == "shipped").mean()),
}
```

7.8 8. Customers with their segment (join in SQL)

SQL

```
SELECT o.customer, c.segment, SUM(o.amount) AS revenue
FROM sales.orders AS o
JOIN sales.customers AS c ON c.customer_id = o.customer_id
GROUP BY o.customer, c.segment
ORDER BY revenue DESC
```

7.9 9. Defensive Prep for empty results

PYTHON

```
orders = pd.DataFrame(queryData)
if orders.empty:
    outputDF = []
else:
    orders["amount"] = orders["amount"].astype(float).round(0)
    outputDF = orders.sort_values("amount", ascending=False).head(5)
```

7.10 10. Test Prep locally before you save it

Prep is plain pandas, so you can exercise it with a small list of rows on your own machine.

PYTHON

```
import pandas as pd

queryData = [
    {"order_id": 1, "region": "EMEA", "amount": 120.0},
    {"order_id": 2, "region": "APAC", "amount": 80.0},
    {"order_id": 3, "region": "EMEA", "amount": 40.0},
]

prep = '''
rows = pd.DataFrame(queryData)
outputDF = rows.groupby("region", as_index=False)["amount"].sum()
'''

scope = {"pd": pd, "queryData": queryData}
exec(compile(prepare, "<prep>", "exec"), scope)
print(scope["outputDF"])
```

APPENDIX A

Choosing a Runtime and Shared Concepts

Compare the runtimes and learn the concepts they share.

- A.1 Choosing a runtime
- A.2 How a run starts
- A.3 What your code can use during a run
- A.4 Connections and secrets
- A.5 The lakehouse (Iceberg)
- A.6 Custom runtimes
- A.7 ML variants
- A.8 Jobs: lifecycle, status and logs
- A.9 Glossary

Groundzero offers four ways to run your data work: the **Standard** runtime, the **DuckDB** runtime, the **Spark** runtime and the **Lakehouse** SQL engine. This chapter covers what they have in common, so the per-runtime guides can stay focused. It explains how to choose a runtime, how a run starts, how to size compute, what your code can use during a run, how connections and the lakehouse work, how custom and machine-learning runtimes are added, and how to follow a run to the end.

A.1 Choosing a runtime

A.1.1 At a glance

	Standard	DuckDB	Spark	Lakehouse
Engine	Managed query engine, SQL plus pandas Prep	DuckDB 1.5.5 on one machine	Apache Spark 3.5.6 cluster	Spark SQL 4.0 with Iceberg 1.10
Compute size	None	You choose	You choose, plus minimum workers	You choose, plus minimum workers
Start	Nothing to start	Quick start, then kept warm for datasets	Cluster start, then kept warm for datasets	Start a session, then run many queries
Connections	One SQL source or the lakehouse	Many, in one query	Many, in one job	The warehouse you open
Writes	No (read queries)	Yes, with the helper library	Yes, with the helper library	Yes, with SQL on Iceberg tables
Scale	Pages of rows	Fits in one machine's memory, or streams from storage	Distributed across workers	Distributed across workers, results paged
Languages	SQL and Python (pandas)	Python and SQL	Python (PySpark) and SQL	SQL (Spark dialect)
Helper library	None	<code>GZ.DuckDBUtils</code>	<code>GZ.SparkUtils</code>	None
ML variants	None	scikit-learn, PyTorch, TensorFlow	scikit-learn, PyTorch, TensorFlow	None
Best for	Charts, dashboards, quick lookups	Fast analytics, ETL, joins across sources	Large data, wide shuffles, distributed ETL	Exploring and maintaining Iceberg tables with SQL

A.1.2 Decision guide

Work down this list and stop at the first match.

1. Is it one query on one SQL source or the lakehouse, feeding a chart, dashboard or API read? Use **Standard**.
2. Is the work interactive SQL on Iceberg tables: exploring, creating or altering tables, `MERGE`, time travel, table maintenance, policies? Use the **Lakehouse**: open a warehouse, start a session and write SQL.
3. Does it join several connections, write results, run on a schedule, or run Python beyond a few lines of pandas? Continue.
4. Does the data fit comfortably in the memory of one machine, or can DuckDB stream it from object storage (Parquet, Iceberg)? Use **DuckDB**. It starts quickly and needs no cluster.

5. **Is the data too large for one machine, or does the job need distributed shuffles, joins of very large tables or Spark-native libraries?** Use **Spark**.
6. **Do you need scikit-learn, PyTorch or TensorFlow?** Pick the matching ML variant of DuckDB or Spark.
7. **Do you need packages that are not installed?** Ask an administrator for a custom runtime built on the runtime you picked.

START SMALL

Prototype on Standard or DuckDB. Move to Spark when a job outgrows one machine, not before. The Groundzero helper libraries of DuckDB and Spark follow the same shape, which keeps a port short.

A.1.3 Where the choice is made

You choose a runtime in three places: when you create a dataset, when you create or schedule an ETL job, and when you start a notebook. The runtime control lists the runtimes available to you and, except for Standard, a compute size. Standard appears only in the dataset designer. You choose the Lakehouse by opening a warehouse in **Lakehouse**, where you pick a compute size and start a session.

A.2 How a run starts

A.2.1 From Run to results

The steps are the same everywhere you can start a run.

1. You press Run, a schedule fires, or an API client submits a job.
2. Groundzero checks that you may use the chosen compute size and the dataset or job.
3. It prepares the connections you selected for the run.
4. It records the run with a unique identifier and starts lineage tracking.
5. It starts your runtime with the compute size you chose, or reuses a warm session (see Sessions and warm starts).
6. Your code runs, and its status, output and logs appear as it goes.
7. For a dataset run, the first page of rows is returned to the caller. For a job, the run page follows status and logs until the run ends.

A.2.2 Datasets, jobs and notebooks compared

	Dataset run	ETL job run	Notebook
Starts from	Dataset designer, chart, dashboard	Job page, schedule, API	Notebook page
Output	A page of rows	Whatever the code writes, plus logs	Interactive
Session	Kept warm and shared (DuckDB, Spark)	Ends when the run ends	Stays up until stopped
Default page size	1000 rows	not applicable	not applicable

In a Spark or DuckDB notebook the Groundzero helper library is already imported (`sparkSession` and `executeJob` for Spark; `duckSession` for DuckDB), together with NumPy, pandas and boto3. Notebooks are saved to your notebook storage each time you save.

A.2.3 Sessions and warm starts

Starting a runtime takes a moment, so Groundzero keeps a **warm session** after a dataset run on DuckDB or Spark. Your next run starts faster if it arrives within a few minutes.

- A session handles one run at a time. A new run takes an idle session that matches its dataset, compute size and connections, and prefers the one you used last.
- If every matching session is busy, Groundzero starts another one. Runs never wait in a queue behind each other.
- A warm session stays available for about 280 seconds after a run and ends after about 300 seconds without a run. After that, the next run starts a new session.
- Lakehouse permissions follow the **user who runs**, not the session, so sharing a session never widens anyone's access.

- Do not keep state in memory between runs. Any run can start in a new session.

Jobs do not share sessions. Each job run gets its own session and ends it when the job finishes. A schedule can also set a stop time, at which the run ends.

A.2.4 Timeouts and retries

Setting	Value
Wait for a dataset run's output	300 seconds by default
Wait for a new session to become ready	60 seconds
Spark workers	1 minimum by default, 10 maximum

If a run request fails before it reaches the runtime, Groundzero retries it automatically. If it may already have reached the runtime, Groundzero does not send it again, so a run never executes twice by accident.

A.2.5 Compute sizes, workers and GPU size

A **compute size** is a named CPU and memory pair that your administrators define. The runtime control lists each as `Name | memory | cpu`. The size you choose is the CPU and memory your code gets. You need Describe permission on a compute size to use it.

For Spark:

- The compute size applies to the driver and to each worker.
- **Minimum workers** sets how many workers start. If you leave it empty, one starts. Workers are added, up to a maximum of 10, when average CPU or memory use passes 70 percent.
- Each executor gets about 65 percent of its worker's memory, and never more than the worker's memory minus 1 GiB.
- Spark starts with one executor and releases idle executors after 120 seconds.

The runtime control also shows a **GPU Size** field for every runtime except Standard.

CHOOSING MEMORY

DuckDB and the Standard runtime hold working data in the memory of one machine. Spark spreads it over workers and spills to disk. When a run stops because it ran out of memory, pick a larger compute size first, then reduce the data you read.

Compute ▲

COMPUTE	MEMORY	VCPU
SMALL	4Gi	2
MEDIUM	8Gi	4
LARGE	16Gi	8
XLARGE	32Gi	16

Minimum Workers

-2+

▶

Query 1 × +

1

MEDIUM

2 workers ▼

▶

Query 1 × +

1

Lakehouse >

> 📁 Schedules

> 📁 Backups

A.3 What your code can use during a run

A.3.1 Run details

Every run gives your code a fixed set of values. You can read them from `os.environ` in notebooks and, where listed, in job code.

Name	Meaning
<code>USERNAME</code>	User who started the run
<code>USERGROUPS</code>	The user's groups, comma-separated
<code>SITE_NAME</code>	Site (tenant) name
<code>REGION</code>	Cloud region
<code>JOB_IDENTIFIER</code>	Id of this run
<code>SESSION_ID</code>	Id of the session
<code>WORKLOAD</code>	<code>ETL_JOB</code> or <code>NOTEBOOK</code>
<code>PAGE_NUMBER</code> , <code>PAGE_SIZE</code>	Paging of a dataset run

Do not rely on any other values or try to override these.

A.3.2 Job placeholders

Text in a job can contain `${NAME}` placeholders that the runtime fills in when the job runs.

Placeholder	Meaning
<code>\${STRTTM}</code>	Run start time
<code>\${ENDTM}</code>	Run end time
<code>\${CLNTNM}</code> , <code>\${SITENM}</code>	Client and site name
<code>\${PRJNM}</code> , <code>\${JOBNM}</code>	Project and job name
<code>\${STS}</code> , <code>\${ERR}</code>	Run status and error text
<code>\${RTE}</code> , <code>\${RGN}</code>	Runtime name and region
<code>\${USR}</code> , <code>\${USRGRPS}</code>	User and groups

The Standard runtime uses a smaller set; see the Standard chapter.

A.3.3 Imported models

A run can be given registered ML models. Groundzero finds each model's latest stored version and makes it available to your code.

A.4 Connections and secrets

A **connection** stores how to reach a source (host, database, credentials or role) under a name. Your code refers to the connection by name and never contains credentials.

- Groundzero prepares your connections when a run starts.
- Only the connections you selected for the job or dataset are available to its code.
- Lakehouse connections use short-lived access that Groundzero refreshes for you, so long-running sessions keep working.
- Do not print connection details or credentials to run logs. Logs are stored and shown on the run page.

NEVER HARD-CODE SECRETS

Put credentials in a connection. Do not paste keys or passwords into job code, parameters or notebooks.

A.5 The Lakehouse (Iceberg)

The Groundzero lakehouse stores tables in Apache Iceberg format on object storage, with one warehouse per site. The lakehouse console and its SQL editor run lakehouse SQL for you. Runtimes read and write the table data directly, with temporary access that applies your Groundzero roles and policies.

Runtime	Engine versions	How it reaches the lakehouse
Standard	DuckDB 1.5.5	Reads Iceberg tables (read-only)
DuckDB	DuckDB 1.5.5, iceberg extension	Reads through the helper library (read-only)
Spark	Spark 3.5.6, Iceberg 1.11.0	Reads and writes through the helper library
Lakehouse	Spark SQL 4.0.0, Iceberg 1.10.0	Your SQL session in the Lakehouse pages (see the Lakehouse guide)

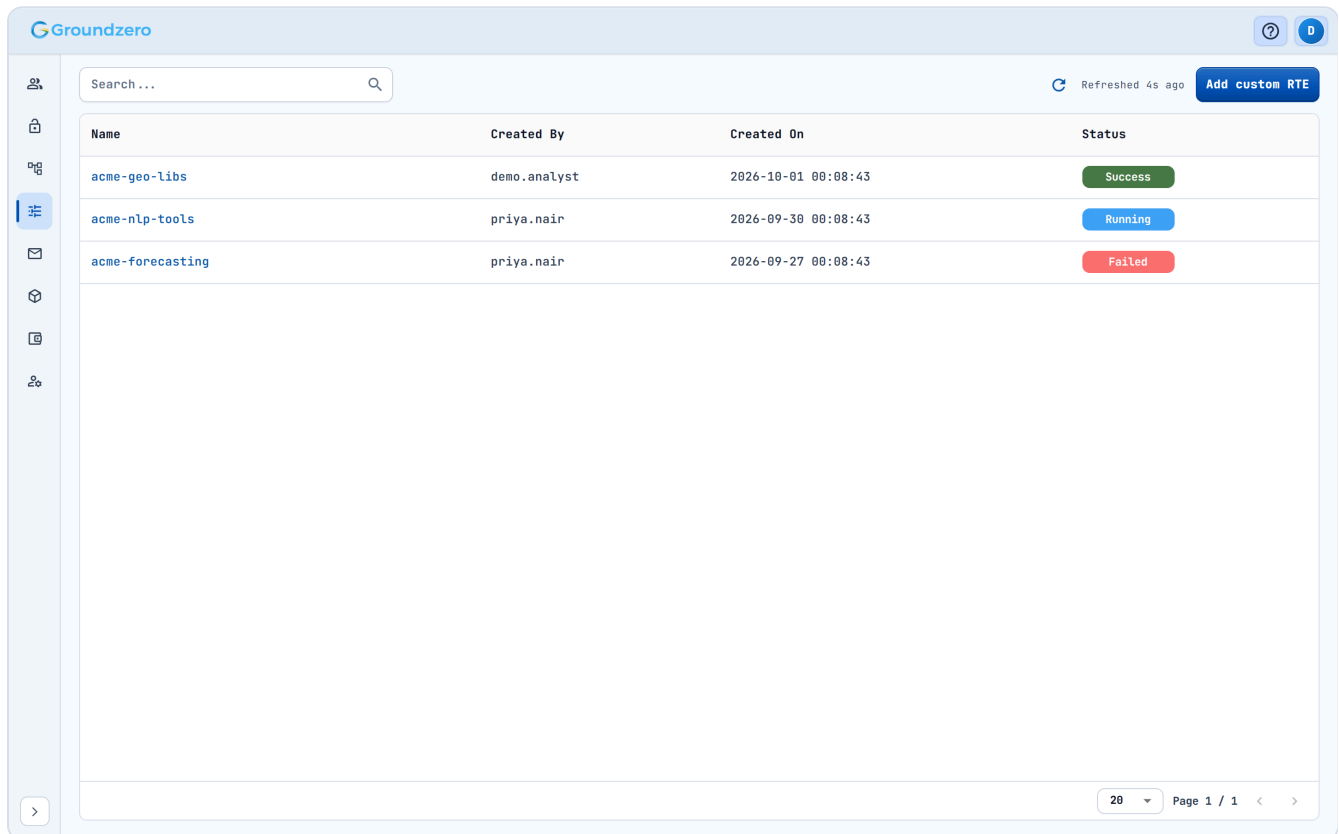
A.6 Custom runtimes

A custom runtime is an existing runtime plus your own pip packages. Administrators create it on the **Custom RTE** page.

1. Upload a `requirements.txt` and choose the **Base Image** (the runtime to extend) and a name.
2. Groundzero builds the runtime with the packages from `requirements.txt`.
3. The build status shows as QUEUED, IN PROGRESS, then SUCCESS or FAILURE.
4. When the build succeeds, the runtime appears in the runtime list.

Guidelines:

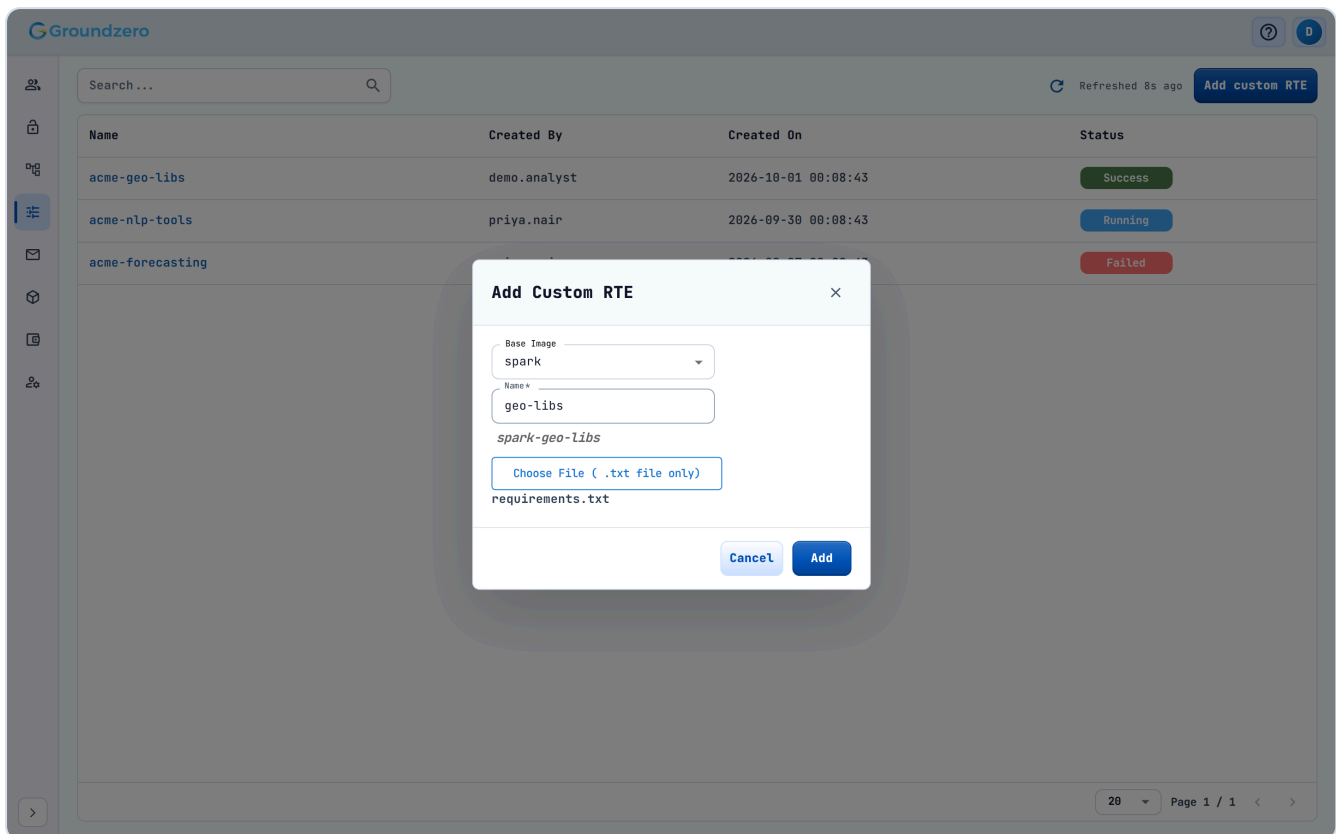
- Extend the runtime you actually run on: a Spark base for Spark jobs, a DuckDB base for DuckDB jobs.
- Pin versions. Do not change packages the base already includes (NumPy, pandas, scikit-learn) unless you need a different version.
- Pure-Python packages install most reliably. A package that needs extra system libraries may fail to build. Check the FAILURE status and the package's requirements.



The screenshot shows the Groundzero admin console interface. At the top, there's a search bar and a refresh button labeled 'Refreshed 4s ago'. A blue button 'Add custom RTE' is in the top right. Below is a table with columns: Name, Created By, Created On, and Status. The table lists three custom runtimes: 'acme-geo-lib' (Success), 'acme-nlp-tools' (Running), and 'acme-forecasting' (Failed). The status is indicated by colored buttons: green for Success, blue for Running, and red for Failed. At the bottom right, there's a pagination control showing '20' items per page and 'Page 1 / 1'.

Name	Created By	Created On	Status
acme-geo-lib	demo.analyst	2026-10-01 00:08:43	Success
acme-nlp-tools	priya.nair	2026-09-30 00:08:43	Running
acme-forecasting	priya.nair	2026-09-27 00:08:43	Failed

The Custom RTE page in the admin console lists each custom runtime and its build status.



Groundzero

Search...

Refreshed 8s ago Add custom RTE

Name	Created By	Created On	Status
acme-geo-libs	demo.analyst	2026-10-01 00:08:43	Success
acme-nlp-tools	priya.nair	2026-09-30 00:08:43	Running
acme-forecasting			Failed

Add Custom RTE

Base Image
spark

Name
geo-libs

spark-geo-libs

Choose File (.txt file only)

requirements.txt

Cancel Add

20 Page 1 / 1

Add custom RTE: choose a base image, name the runtime and upload a requirements file.

A.7 ML variants

Spark and DuckDB each come as a base runtime and three ML variants. A variant is the base plus extra libraries.

Variant	Adds
Base	Spark or DuckDB, Groundzero helper libraries, scikit-learn 1.5.2, CPU PyTorch 2.14.0 and torchvision 0.29.0, document parsing
sklearn	OpenCV (headless), XGBoost, LightGBM, CatBoost
pytorch	sklearn extras, PyTorch 2.5.1 and torchvision 0.20.1 (GPU build), SHAP, LIME, Albumentations, Hugging Face Hub and Transformers, simsimd, fastai
tensorflow	TensorFlow, OpenCV, XGBoost, LightGBM, CatBoost, SHAP, LIME, NLTK, spaCy, h5py

Notes:

- The pytorch variant uses the GPU build of PyTorch, so GPU-sized runs can train on GPUs.
- Variants appear in the runtime list with names such as `gz-rte-duckdb-sklearn` and a version tag such as `1.0.0`.
- The Standard runtime has no ML variants.

A.8 Jobs: lifecycle, status and logs

A.8.1 Statuses

Status	Meaning
Queued	Accepted, waiting to start
Running	Your code is executing
Success	Finished without error
Failed	The code or Groundzero reported an error
Aborted	Stopped before it could finish

Queued and Running are active. Success, Failed and Aborted are final.

A.8.2 Logs

Run logs stream to the run page while the run works.

- Lines never repeat when the page reloads the stream.
- If a run stops without reporting a final state, Groundzero marks it Failed rather than leaving it Running.
- If the page is asked for logs too often, it retries after a short delay.

Print and logging output from your code appears in the run log as the run works.

A.8.3 Scheduling

A schedule names the job, its runtime and its compute size. It may also set a start and stop time.

A.9 Glossary

Term	Meaning
Runtime (RTE)	The environment and engine your code runs in
Compute size	A named CPU and memory pair for a runtime
Session	A running runtime that can accept runs
Warm session	A dataset session kept ready for a few minutes and shared between runs
Connection	A named, stored way of reaching a source
Dataset	A saved query (and code) that returns rows
Prep	The pandas code that reshapes a Standard dataset's rows
Lakehouse	Groundzero's Iceberg tables on object storage
Warehouse	A site's lakehouse storage location
Custom runtime	A runtime extended with your pip packages
ML variant	A runtime with machine-learning libraries added
Lineage	The record of which assets a run read and wrote