

a deep dive into the dbt manifest

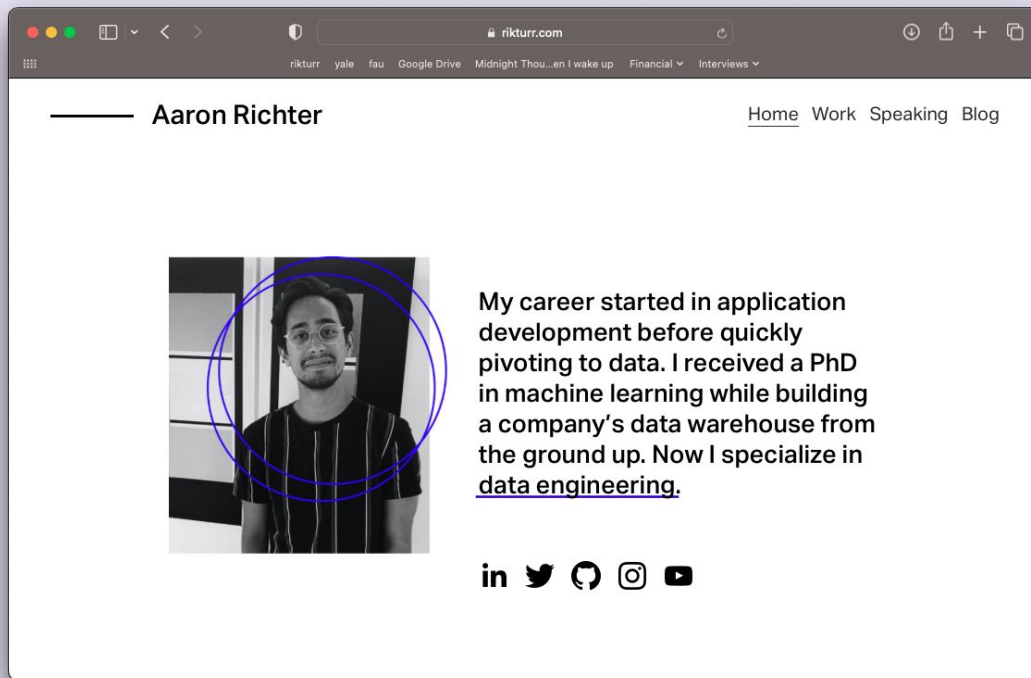
data council austin
march 2023

about me:



rikturr.com

@rikturr



dbt



dbt

Software engineering fundamentals for
SQL-based* data pipelines

- Project structure
- SQL templating and re-use
- Prod/dev separations
- Testing and documentation
- “Analytics engineering”

orders.sql ×

models > orders.sql

```
1  {% set payment_methods = ['credit_card', 'coupon', 'b
2
3  with orders as (
4
5      select * from {{ ref('stg_orders') }}
6
7  ),
8
9  payments as (
10
11     select * from {{ ref('stg_payments') }}
12
13  ),
14
15  order_payments as (
16
17     select
18         order_id,
19
20         {% for payment_method in payment_methods -%}
21         sum(case when payment_method = '{{ payment_me
22         {% endfor -%}
23
24         sum(amount) as total_amount
25
26     from payments
27
28     group by order_id
29
30  ),
```

! schema.yml ×

models > ! schema.yml

```
32  - name: orders
33    description: This table has basic information about orders, as
34
35    columns:
36      - name: order_id
37        tests:
38          - unique
39          - not_null
40        description: This is a unique identifier for an order
41
42      - name: customer_id
43        description: Foreign key to the customers table
44        tests:
45          - not_null
46          - relationships:
47            to: ref('customers')
48            field: customer_id
49
50      - name: order_date
51        description: Date (UTC) that the order was placed
52
53      - name: status
54        description: '{{ doc("orders_status") }}'
55        tests:
56          - accepted_values:
57            values: ['placed', 'shipped', 'completed', 'return_p
58
59      - name: amount
60        description: Total amount (AUD) of the order
61        tests:
```

```
● (dbt) → jaffle_shop dbt build
21:39:44 Running with dbt=1.4.5
21:39:44 Found 6 models, 20 tests, 0 snapshots, 0 analyses, 297 macros, 0 operations, 3 seed files, 0 sources, 0 exposures, 0 metrics
21:39:44
21:39:44 Concurrency: 1 threads (target='dev')
21:39:44
21:39:44 1 of 28 START seed file main.raw_customers ..... [RUN]
21:39:44 1 of 28 OK loaded seed file main.raw_customers ..... [INSERT 100 in 0.14s]
21:39:44 2 of 28 START seed file main.raw_orders ..... [RUN]
21:39:44 2 of 28 OK loaded seed file main.raw_orders ..... [INSERT 99 in 0.11s]
21:39:44 3 of 28 START seed file main.raw_payments ..... [RUN]
21:39:44 3 of 28 OK loaded seed file main.raw_payments ..... [INSERT 113 in 0.12s]
21:39:45 4 of 28 START sql view model main.stg_customers ..... [RUN]
21:39:45 4 of 28 OK created sql view model main.stg_customers ..... [OK in 0.05s]
21:39:45 5 of 28 START sql view model main.stg_orders ..... [RUN]
21:39:45 5 of 28 OK created sql view model main.stg_orders ..... [OK in 0.02s]
21:39:45 6 of 28 START sql view model main.stg_payments ..... [RUN]
21:39:45 6 of 28 OK created sql view model main.stg_payments ..... [OK in 0.03s]
21:39:45 7 of 28 START test not_null_stg_customers_customer_id ..... [RUN]
21:39:45 7 of 28 PASS not_null_stg_customers_customer_id ..... [PASS in 0.04s]
21:39:45 8 of 28 START test unique_stg_customers_customer_id ..... [RUN]
```

```
(dbt) → jaffle_shop dbt build
21:39:44 Running with dbt=1.4.5
21:39:44 Found 6 models, 20 tests, 0 snapshots, 0 analyses, 297 macros, 0 operations, 3 seed files, 0 sources, 0 exposures, 0 metrics
21:39:44 Concurrency: 1 threads (target='dev')
21:39:44 1 of 28 START seed
21:39:44 1 of 28 OK loaded s
21:39:44 2 of 28 START seed
21:39:44 2 of 28 OK loaded s
21:39:44 3 of 28 START seed
21:39:45 3 of 28 OK loaded s
21:39:45 4 of 28 START sql v
21:39:45 4 of 28 OK created
21:39:45 5 of 28 START sql v
21:39:45 5 of 28 OK created
21:39:45 6 of 28 START sql v
21:39:45 6 of 28 OK created
21:39:45 7 of 28 START test
21:39:45 7 of 28 PASS not_n
21:39:45 8 of 28 START test
```

orders.sql ×

target > run > jaffle_shop > models > orders.sql

```
4
5 create table
6 | "dbt"."main"."orders_dbt_tmp"
7 as (
8
9
10 with orders as (
11
12     select * from "dbt"."main"."stg_orders"
13
14 ),
15
16 payments as (
17
18     select * from "dbt"."main"."stg_payments"
19
20 ),
```

RT 100 in 0.14s]

RT 99 in 0.11s]

RT 113 in 0.12s]

n 0.05s]

n 0.02s]

n 0.03s]

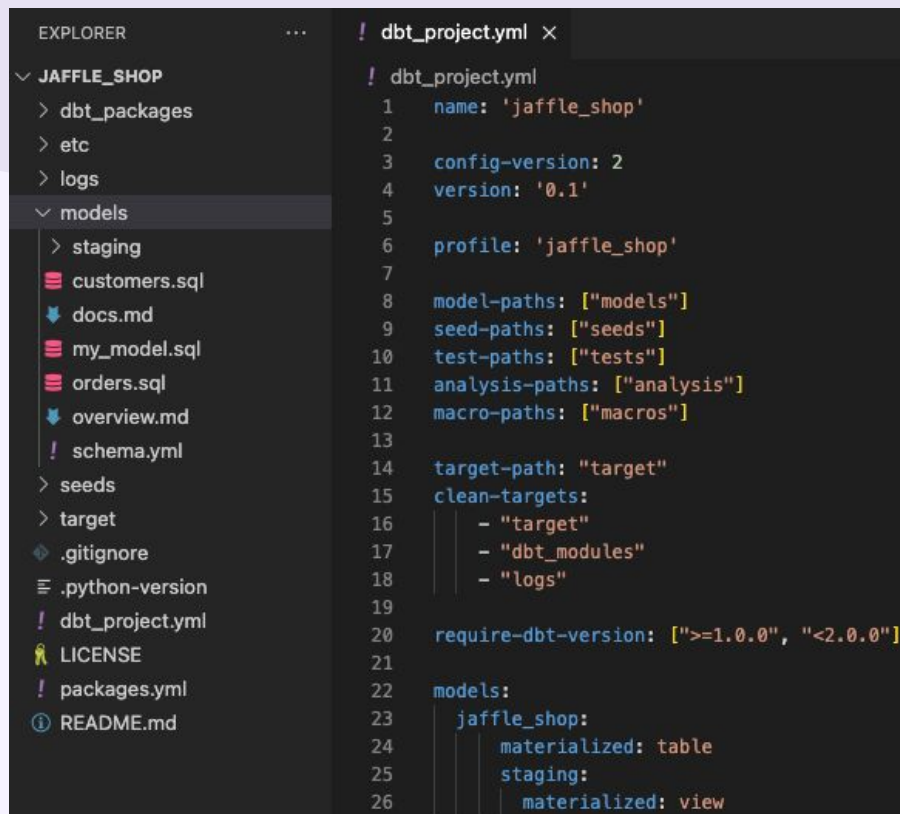
in 0.04s]

dbt artifacts 101

a dbt project

dbt_project.yml

models/



The image shows a code editor with two panes. The left pane, titled 'EXPLORER', displays the file structure of a project named 'JAFFLE_SHOP'. The right pane shows the content of the 'dbt_project.yml' file.

```
EXPLORER
└─ JAFFLE_SHOP
  └─ dbt_packages
  └─ etc
  └─ logs
  └─ models
    └─ staging
    └─ customers.sql
    └─ docs.md
    └─ my_model.sql
    └─ orders.sql
    └─ overview.md
    └─ schema.yml
  └─ seeds
  └─ target
  └─ .gitignore
  └─ .python-version
  └─ dbt_project.yml
  └─ LICENSE
  └─ packages.yml
  └─ README.md

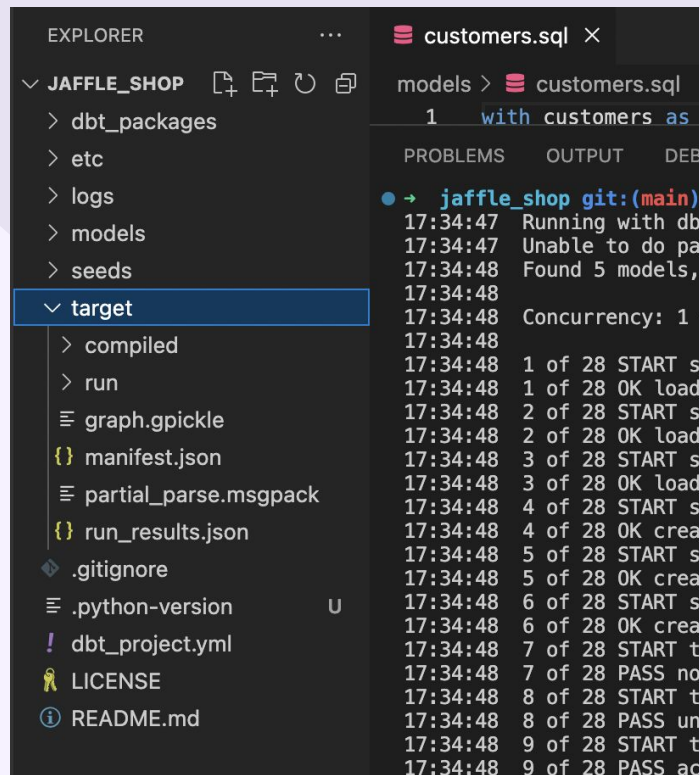
! dbt_project.yml x
! dbt_project.yml
1  name: 'jaffle_shop'
2
3  config-version: 2
4  version: '0.1'
5
6  profile: 'jaffle_shop'
7
8  model-paths: ["models"]
9  seed-paths: ["seeds"]
10 test-paths: ["tests"]
11 analysis-paths: ["analysis"]
12 macro-paths: ["macros"]
13
14 target-path: "target"
15 clean-targets:
16   - "target"
17   - "dbt_modules"
18   - "logs"
19
20 require-dbt-version: [">=1.0.0", "<2.0.0"]
21
22 models:
23   jaffle_shop:
24     materialized: table
25     staging:
26       materialized: view
```

artifacts

Files that describe your dbt project

Information only possible after *invocation*

Typically in target/ folder



```
EXPLORER
  JAFFLE_SHOP
    dbt_packages
    etc
    logs
    models
    seeds
    target
      compiled
      run
      graph.gpickle
      manifest.json
      partial_parse.msgpack
      run_results.json
      .gitignore
      .python-version
      dbt_project.yml
      LICENSE
      README.md
  customers.sql

customers.sql
models > customers.sql
1 with customers as

PROBLEMS OUTPUT DEB
  jaffle_shop git:(main)
17:34:47 Running with db
17:34:47 Unable to do pa
17:34:48 Found 5 models,
17:34:48
17:34:48 Concurrency: 1
17:34:48
17:34:48 1 of 28 START s
17:34:48 1 of 28 OK load
17:34:48 2 of 28 START s
17:34:48 2 of 28 OK load
17:34:48 3 of 28 START s
17:34:48 3 of 28 OK load
17:34:48 4 of 28 START s
17:34:48 4 of 28 OK crea
17:34:48 5 of 28 START s
17:34:48 5 of 28 OK crea
17:34:48 6 of 28 START s
17:34:48 6 of 28 OK crea
17:34:48 7 of 28 START t
17:34:48 7 of 28 PASS no
17:34:48 8 of 28 START t
17:34:48 8 of 28 PASS und
17:34:48 9 of 28 START t
17:34:48 9 of 28 PASS ad
```

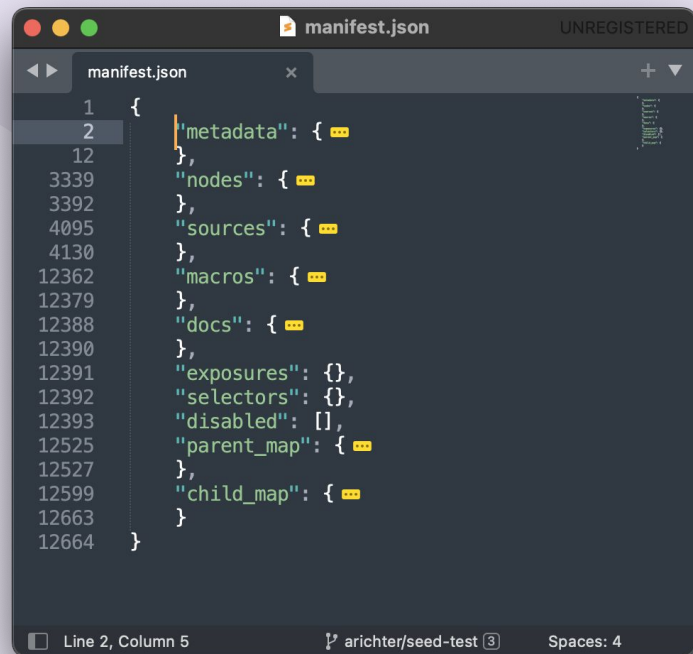
<https://docs.getdbt.com/reference/artifacts/dbt-artifacts>

the manifest

what is the manifest?

“This single file contains a full representation of your dbt project's resources (models, tests, macros, etc), including all node configurations and resource properties.”

[dbt Artifacts - Manifest](#)

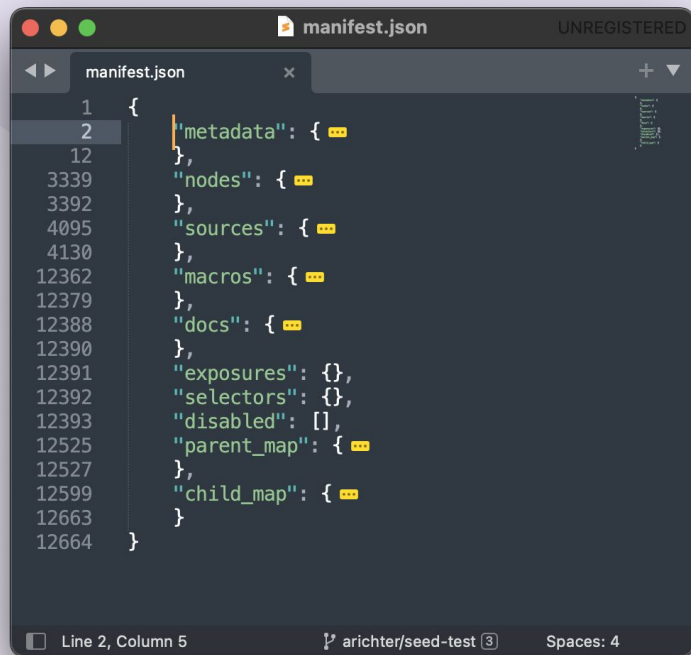


```
1 {
2   "metadata": { ...
12 },
3339 "nodes": { ...
3392 },
4095 "sources": { ...
4130 },
12362 "macros": { ...
12379 },
12388 "docs": { ...
12390 },
12391 "exposures": {},
12392 "selectors": {},
12393 "disabled": [],
12525 "parent_map": { ...
12527 },
12599 "child_map": { ...
12663 }
12664 }
```

what is the manifest?

TL;DR:

Machine-readable file with everything you need to know about the dbt project



```
manifest.json
UNREGISTERED

manifest.json x +
1 {
2   "metadata": { ...
12 },
3339 "nodes": { ...
3392 },
4095 "sources": { ...
4130 },
12362 "macros": { ...
12379 },
12388 "docs": { ...
12390 },
12391 "exposures": {},
12392 "selectors": {},
12393 "disabled": [],
12525 "parent_map": { ...
12527 },
12599 "child_map": { ...
12663 }
12664 }
```

Line 2, Column 5 | arichter/seed-test 3 | Spaces: 4

what is the manifest?

Written to:

`<target-path>/manifest.json`
(default `target/manifest.json`)

Updated on nearly all dbt invocations

All enabled resources are updated, regardless of selection
(i.e. `--select`)

Tip: before you read the manifest, run a dbt command to make sure it's up to date!

! dbt_project.yml

customers.sql

{} manifest.json

models > customers.sql

target > {} manifest.json > {} nodes

```

1 > with customers as (
5   ),
6
7   orders as (
8     select * from {{ ref('stg_orders') }}
9   ),
10
11  ),
12
13  payments as (
14     select * from {{ ref('stg_payments') }}
15  ),
16
17  ),
18
19  customer_orders as (
20
21     select
22     customer_id,
23
24     min(order_date) as first_order,
25     max(order_date) as most_recent_order,
26     count(order_id) as number_of_orders
27   from orders
28
29   group by customer_id
30
31  ),
32
33  customer_payments as (
34
35     select
36     orders.customer_id

```

```

13   "nodes": {
14     "model.jaffle_shop.customers": {
15       "raw_sql": "with customers as (\n\n  select * from {{ ref('stg_customers') }}\n\n)",
16       "compiled": true,
17       "resource_type": "model",
18       "depends_on": {
19         "macros": [
20           "macro.dbt.run_hooks",
21           "macro.dbt.statement",
22           "macro.dbt.persist_docs"
23         ],
24         "nodes": [
25           "model.jaffle_shop.stg_customers",
26           "model.jaffle_shop.stg_orders",
27           "model.jaffle_shop.stg_payments"
28         ]
29       },
30     },
31     "config": {
32       "enabled": true,
33       "alias": null,
34       "schema": null,
35       "database": null,
36       "tags": [],
37       "meta": {},
38       "materialized": "table",
39       "persist_docs": {},
40       "quoting": {},
41       "column_types": {},
42       "full_refresh": null,
43       "unique_key": null,
44       "on_schema_change": "ignore",
45       "post-hook": [],
46       "pre-hook": []

```

```
customers.sql - jaffle_shop

models > customers.sql
1 > with customers as (...
5   ),
6
7   orders as (
8     select * from {{ ref('stg_orders') }}
9   ),
10
11  payments as (
12    select * from {{ ref('stg_payments') }}
13  ),
14
15  customer_orders as (
16    select
17      customer_id,
18      min(order_date) as first_order,
19      max(order_date) as most_recent_order,
20      count(order_id) as number_of_orders
21    from orders
22  group by customer_id
23  ),
24
25  customer_payments as (
26    select
27      orders.customer_id,
28      payments.customer_id,
29      payments.order_id,
30      payments.amount
31    from orders
32    join payments on orders.customer_id = payments.customer_id
33  )
34
35  select
36    customers.customer_id,
37    customers.first_order,
38    customers.most_recent_order,
39    customers.number_of_orders,
40    customer_payments.first_order,
41    customer_payments.most_recent_order,
42    customer_payments.number_of_orders,
43    customer_payments.first_order - customers.first_order as first_order_delta,
44    customer_payments.most_recent_order - customers.most_recent_order as most_recent_order_delta,
45    customer_payments.number_of_orders - customers.number_of_orders as number_of_orders_delta
46  from customers
47  join customer_payments on customers.customer_id = customer_payments.customer_id
48  join customer_orders on customers.customer_id = customer_orders.customer_id
49  join customer_payments on customer_orders.customer_id = customer_payments.customer_id
50  join payments on customer_payments.order_id = payments.order_id
51  join orders on customer_payments.order_id = orders.order_id
52  join orders on customer_orders.order_id = orders.order_id
53  join orders on customer_payments.order_id = orders.order_id
54  join orders on customer_payments.order_id = orders.order_id
55  join orders on customer_payments.order_id = orders.order_id
56  join orders on customer_payments.order_id = orders.order_id
57  join orders on customer_payments.order_id = orders.order_id
58  join orders on customer_payments.order_id = orders.order_id
59  join orders on customer_payments.order_id = orders.order_id
60  join orders on customer_payments.order_id = orders.order_id
61  join orders on customer_payments.order_id = orders.order_id
62  join orders on customer_payments.order_id = orders.order_id
63  join orders on customer_payments.order_id = orders.order_id
64  join orders on customer_payments.order_id = orders.order_id
65  join orders on customer_payments.order_id = orders.order_id
66  join orders on customer_payments.order_id = orders.order_id
67  join orders on customer_payments.order_id = orders.order_id
68  join orders on customer_payments.order_id = orders.order_id
69  join orders on customer_payments.order_id = orders.order_id
70  join orders on customer_payments.order_id = orders.order_id
71  join orders on customer_payments.order_id = orders.order_id
72  join orders on customer_payments.order_id = orders.order_id
73  join orders on customer_payments.order_id = orders.order_id
74  join orders on customer_payments.order_id = orders.order_id
75  join orders on customer_payments.order_id = orders.order_id
76  join orders on customer_payments.order_id = orders.order_id
77  join orders on customer_payments.order_id = orders.order_id
78  join orders on customer_payments.order_id = orders.order_id
79  join orders on customer_payments.order_id = orders.order_id
80  join orders on customer_payments.order_id = orders.order_id
81  join orders on customer_payments.order_id = orders.order_id
82  join orders on customer_payments.order_id = orders.order_id
83  join orders on customer_payments.order_id = orders.order_id
84  join orders on customer_payments.order_id = orders.order_id
85  join orders on customer_payments.order_id = orders.order_id
86  join orders on customer_payments.order_id = orders.order_id
87  join orders on customer_payments.order_id = orders.order_id
88  join orders on customer_payments.order_id = orders.order_id
89  join orders on customer_payments.order_id = orders.order_id
90  join orders on customer_payments.order_id = orders.order_id
91  join orders on customer_payments.order_id = orders.order_id
92  join orders on customer_payments.order_id = orders.order_id
93  join orders on customer_payments.order_id = orders.order_id
94  join orders on customer_payments.order_id = orders.order_id
95  join orders on customer_payments.order_id = orders.order_id
96  join orders on customer_payments.order_id = orders.order_id
97  join orders on customer_payments.order_id = orders.order_id
98  join orders on customer_payments.order_id = orders.order_id
99  join orders on customer_payments.order_id = orders.order_id
100 join orders on customer_payments.order_id = orders.order_id

target > {} manifest.json > {} nodes
13 "nodes": {
14   "model.jaffle_shop.customers": {
15     "raw_sql": "with customers as (\n\n  select * from {{ ref('stg_customers') }}\n\n)",
16     "compiled": true,
17     "resource_type": "model",
18     "depends_on": {
19       "macros": [
20         "macro.dbt.run_hooks",
21         "macro.dbt.statement",
22         "macro.dbt.persist_docs"
23       ]
24     },
25     "nodes": [
26       "model.jaffle_shop.stg_customers",
27       "model.jaffle_shop.stg_orders",
28       "model.jaffle_shop.stg_payments"
29     ]
30   },
31   "config": {
32     "enabled": true,
33     "alias": null,
34     "schema": null,
35     "database": null,
36     "tags": [],
37     "meta": {},
38     "materialized": "table",
39     "persist_docs": {},
40     "quoting": {},
41     "column_types": {},
42     "full_refresh": null,
43     "unique_key": null,
44     "on_schema_change": "ignore",
45     "post-hook": [],
46     "pre-hook": []
47   }
48 }
```


using the manifest

Program can read the manifest JSON file and perform actions based on its information:

- Python script
- CI/CD tool
- External application



load_manifest.py > ...

```

1 from pathlib import Path
2 import subprocess
3 import json
4
5 def run_cmd(cmd):
6     subprocess.check_call(cmd, shell=True)
7
8 def load_manifest(target_path: Path=Path("target")):
9     """
10    Load the dbt project's manifest file
11    :return: dict of manifest contents
12    """
13    manifest_path = target_path / "manifest.json"
14
15    if not manifest_path.exists():
16        run_cmd("dbt deps")
17        run_cmd("dbt compile")
18    return json.loads(manifest_path.read_text())

```

```

(jaffle-shop) → jaffle_shop git:(main) x ipython
Python 3.8.8 (default, Jul 30 2021, 12:04:10)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 — An enhanced Interactive Python. Type '?' for help.

```

In [1]: from load_manifest import *

In [2]: manifest = load_manifest()

In [3]: manifest.keys()

```
Out[3]: dict_keys(['metadata', 'nodes', 'sources', 'macros', 'docs', 'exp
osures', 'metrics', 'selectors', 'disabled', 'parent_map', 'child_map'])
```

In [4]: len(manifest["nodes"])

Out[4]: 28

In [5]: manifest["nodes"]["model.jaffle_shop.customers"]["depends_on"]

```
Out[5]:
{'macros': ['macro.dbt.run_hooks',
'macro.dbt.statement',
'macro.dbt.persist_docs'],
'nodes': ['model.jaffle_shop.stg_customers',
'model.jaffle_shop.stg_orders',
'model.jaffle_shop.stg_payments']}
```

In [6]:

! schema.yml load_manifest.py U x {} manifest.json

```
load_manifest.py > ...
1 from pathlib import Path
2 import subprocess
3 import json
4
5 def run_cmd(cmd):
6     subprocess.check_call(cmd, shell=True)
7
8 def load_manifest(target_path: Path=Path("target")):
9     """
10    Load the dbt project's manifest file
11    :return: dict of manifest contents
12    """
13    manifest_path = target_path / "manifest.json"
14
15    if not manifest_path.exists():
16        run_cmd("dbt deps")
17        run_cmd("dbt compile")
18    return json.loads(manifest_path.read_text())
```

```
(jaffle-shop) → jaffle_shop git:(main) x ipython
Python 3.8.8 (default, Jul 30 2021, 12:04:10)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: from load_manifest import *
```

```
In [2]: manifest = load_manifest()
```

```
In [3]: manifest.keys()
```

```
Out[3]: dict_keys(['metadata', 'nodes', 'sources', 'macros', 'docs', 'exposures', 'metrics', 'selectors', 'disabled', 'parent_map', 'child_map'])
```

```
In [4]: len(manifest["nodes"])
```

```
Out[4]: 28
```

```
In [5]: manifest["nodes"]["model.jaffle_shop.customers"]["depends_on"]
```

```
Out[5]:
{'macros': ['macro.dbt.run hooks',
            'macro.dbt.statement',
            'macro.dbt.persist_docs'],
 'nodes': ['model.jaffle_shop.stg_customers',
            'model.jaffle_shop.stg_orders',
            'model.jaffle_shop.stg_payments']}
```

```
In [6]:
```

manifest-powered applications

airflow orchestration

developer experience

external configuration

observability

dbt + airflow

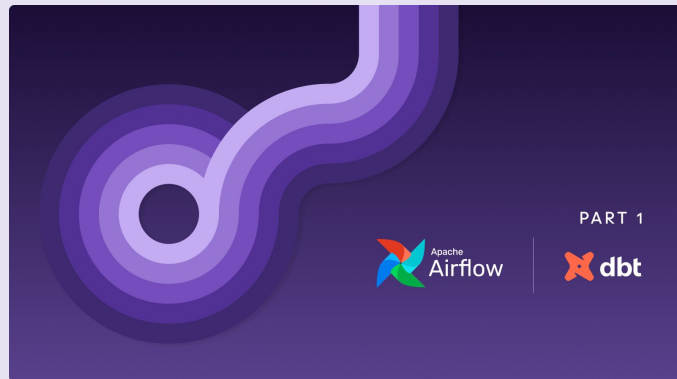
Courtesy of Astronomer, Updater, and Sam Bail

Create model-level Airflow DAGs that mirror the dbt project DAG

One task per dbt model

Manifest contents

- Nodes → depends_on
 - To build graph



Building a Scalable Analytics Architecture With Airflow and dbt

```

data = load_manifest()

dbt_tasks = {}
for node in data["nodes"].keys():
    if node.split(".")[0] == "model":
        node_test = node.replace("model", "test")

        dbt_tasks[node] = make_dbt_task(node, "run")
        dbt_tasks[node_test] = make_dbt_task(node, "test")

for node in data["nodes"].keys():
    if node.split(".")[0] == "model":

        # Set dependency to run tests on a model after model runs finishes
        node_test = node.replace("model", "test")
        dbt_tasks[node] >> dbt_tasks[node_test]

        # Set all model -> model dependencies
        for upstream_node in data["nodes"][node]["depends_on"]["nodes"]:

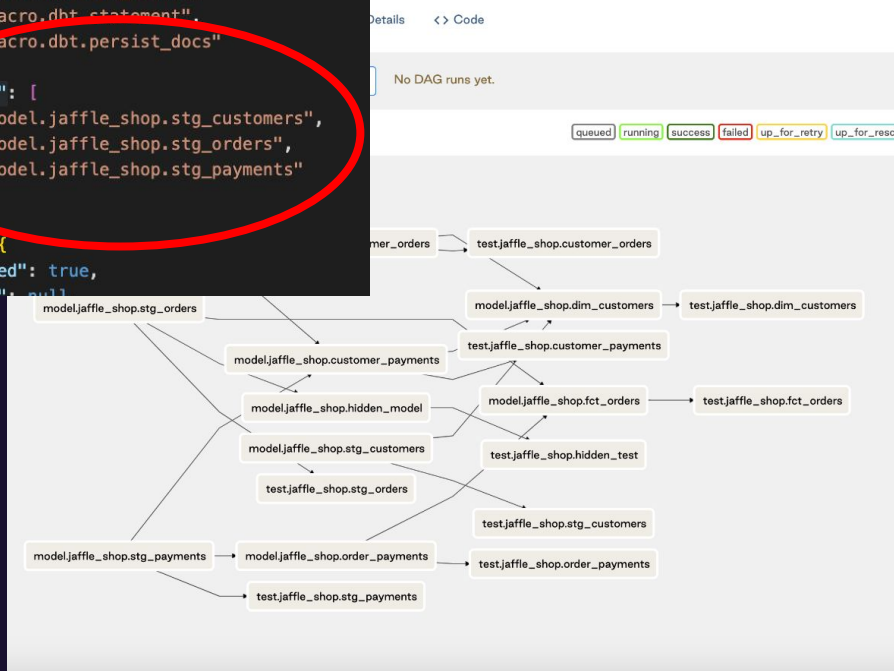
            upstream_node_type = upstream_node.split(".")[0]
            if upstream_node_type == "model":
                dbt_tasks[upstream_node] >> dbt_tasks[node]

```

```

"nodes": {
  "model.jaffle_shop.customers": {
    "raw_sql": "with customers as (\n\n  select *\n  from {{ source('jaffle_shop', 'customers') }}\n)",
    "compiled": true,
    "resource_type": "model",
    "depends_on": {
      "macros": [
        "macro.dbt.run_hooks",
        "macro.dbt.statement",
        "macro.dbt.persist_docs"
      ]
    },
    "nodes": [
      "model.jaffle_shop.stg_customers",
      "model.jaffle_shop.stg_orders",
      "model.jaffle_shop.stg_payments"
    ]
  },
  "config": {
    "enabled": true,
    "tags": ["jaffle_shop"]
  }
}

```



Building a Scalable Analytics Architecture With Airflow and dbt

airflow dag builder

Squarespace - Aaron Richter

Why?

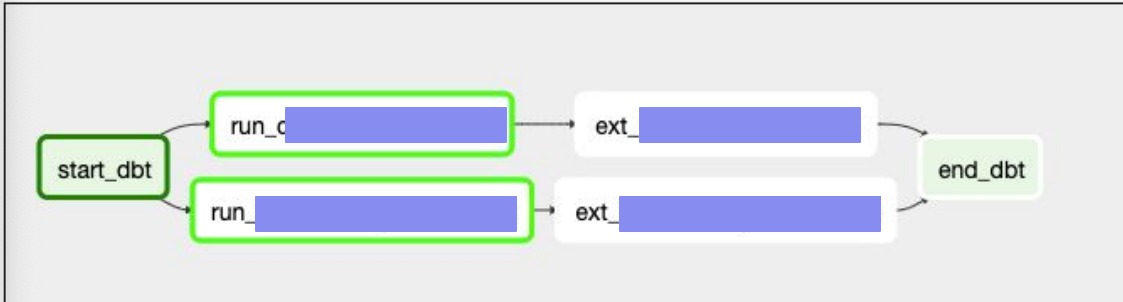
- 30+ analysts, remove need for them to write Airflow Python code
- Support extended dbt operations for every model: test, external tables, etc
- Custom alerting for individual models
- Connecting dependencies between multiple dbt projects (Trino -> BigQuery)




```
dag_builder_dag.py UNREGISTERED
dag_builder_dag.py x
1 from sqsp_dbt import DbtAirflowDagBuilder, ConnType
2
3 dag_builder = DbtAirflowDagBuilder(
4     dag=dag,
5     dbt_project_path=PROJECT_PATH,
6     dbt_target=TARGET,
7     conn_id=CONN_ID,
8     conn_type=ConnType.BIGQUERY,
9 )
10 start_dbt, end_dbt = dag_builder.add_dbt_tasks()
11
12 some_previous_task = ...
13 some_later_task = ...
14
15 some_previous_task >> start_dbt
16 end_dbt >> some_later_task
17
```

Line 17, Column 1 Spaces: 4 Python

DBTRunOperator DummyOperator WriteExternalTablesOperator



airflow dag builder

How to keep DAG up to date?

Option 1: compile during DAG parse

- Airflow DAG structure dynamically updated during DAG parsing
- Compile when project files change
- Execute dbt in isolated virtualenv*

Option 2: upload artifacts at build time

- After main branch merge -> upload artifacts to cloud storage
- Con: select/exclude



airflow orchestration

developer experience

external configuration

observability



dry runs

Squarespace - Aaron Richter

Execute dry run queries for all compiled models

- Ensures tables/columns exist
- Validates SQL syntax
- `**Force all models to be ephemeral to dry runs`

Manifest contents

- Compiled SQL / file path

```
dry_run.py — jaffle_shop
python3.8 + v
dry_run.py U x
dry_run.py > compiled_sql
1 import json
2 import os
3 from pathlib import Path
4 from google.cloud import bigquery
5
6 from load_manifest import load_manifest
7
8 def compiled_sql(model, target):
9     manifest = load_manifest()
10    contents = manifest["nodes"][model]
11    compiled_sql_path = (
12        target / "compiled" /
13        contents["package_name"] /
14        contents["original_file_path"]
15    )
16    return compiled_sql_path.read_text()
17
18 def dry_run(model, target=Path("target")):
19     sql = compiled_sql(model, target=target)
20
21     client = bigquery.Client(project=os.getenv("PROJECT_ID"))
22     job_config = bigquery.QueryJobConfig(dry_run=True, use_query_cache=False)
23
24     # will throw exception if dry run fails
25     client.query(sql, job_config=job_config)
26     print("Dry run succeeded. Hooray!")
27
28
(jaffle-shop) + jaffle_shop git:(main) x ipython
Python 3.8.8 (default, Jul 30 2021, 12:04:10)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from dry_run import dry_run

In [2]: dry_run("model.jaffle_shop.customers")
18:38:12 Running with dbt=1.1.0
18:38:12 Warning: No packages were found in packages.yml
18:38:15 Running with dbt=1.1.0
18:38:15 Partial parse save file not found. Starting full parse.
18:38:16 Found 5 models, 20 tests, 0 snapshots, 0 analyses, 191 macros, 0 operations, 3
seed files, 0 sources, 0 exposures, 0 metrics
18:38:16
18:38:17 Concurrency: 2 threads (target='dev')
18:38:17
18:38:17 Done.
Dry run succeeded. Hooray!

In [3]:
```

enforce org policies

Auto Trader - Darren Haken (coalesce 2021)

Metadata enforcement

- Documents contracts between teams
- CI/CD builds check for required metadata

Building a metadata ecosystem with dbt

09 December 2021, 11:10 AM



Darren Haken

Head of Engineering - Platform & Data,
Auto Trader

DBT supports metadata

Metadata as code


Version controlled

Encapsulate model development with
metadata

models/schema.yml

```
version: 2

models:
  - name: users
    meta:
      owner: "@alice"
      model_maturity: in dev
```



airflow orchestration
developer experience
external configuration
observability

external dependencies

Community.com - Teghan Nightengale (coalesce 2021)

Read metadata from manifest to configure external systems

- Example: Slack alerts

Building On Top of dbt: Managing External Dependencies

08 December 2021, 01:55 PM



Teghan Nightengale
Community.com, Data Engineer

Example: Client Requests

Key to note is the JSONPath Syntax:

```
$.nodes.*.config.meta'
```

This path allows a dependent client to fetch exactly what they need from within an artifact, or across artifacts.

```
example_clients -- zsh -- 80x29
~/github/meta/sub/meta/sub/examples/example_clients -- zsh
tnightengale@C02DK2G3MD6T example_clients % curl -X GET -G http://localhost:8080/artifacts/query/ -d sql_json_path='$.nodes.*.config.meta' | jq .
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 315 100 315 0 0 35800 0 --:--:-- --:--:-- --:--:-- 39375
[
  {
    "slack_test_alert": {
      "message": "\n\nSeveral subscriptions in Netsuite are inconsistent with our MRR reporting. Please refer to\nthe docs here for instructions on how to resolve the issue:\nhttps://docs.google.com/document/d/1Vh-gD3c#HWLIGFWRp0JzaLD_ERen064JDeHrvgduhK/edit\n",
      "recipients": [
        "@tnightengale"
      ]
    }
  }
]
```


airflow orchestration

developer experience

external configuration

observability

dbt-artifacts package

Use dbt on your dbt!

Models that include manifest/artifact data*

Saves tables to your warehouse on every dbt execution

*technically not using manifest, but same data, see [graph](#)



```
{% macro upload_models(graph) -%}  
  {% set models = [] %}  
  {% for node in graph.nodes.values() | selectattr("resource_type", "equalto", "model") %}  
    {% do models.append(node) %}  
  {% endfor %}  
  {{ return(adapter.dispatch('get_models_dml_sql', 'dbt_artifacts')(models)) }}  
{%- endmacro %}
```

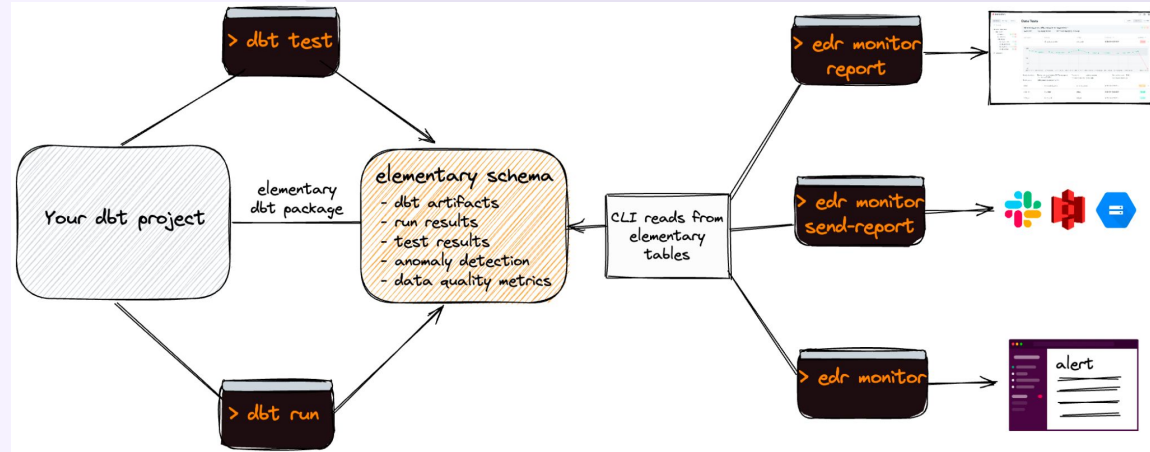
https://github.com/brooklyn-data/dbt_artifacts

elementary data

Open -source and cloud options

Similar data capture as dbt-artifacts

Oob visualization, monitoring, and alerting



honorable mentions

too many good applications to count!

Observability - [Re_data](#)

VSCode extension - [dbt Power user](#)

Automated code refactoring - [dbt-osmosis](#)

recap

recap

The manifest is your friend

Use Python or other processes to read manifest file

Use cases

- Airflow orchestration (nodes → depends_on)
- Project testing (nodes → compiled sql)
- External configuration (nodes → meta)
- Observability

Contact me

Aaron Richter

rikturr@gmail.com

@rikturr

Thanks!

Shut up i'm manifesting

