



Vantage RuntimeAI

Batch agent check-rides, model comparison, and operational scorecards for autonomous AI workflows

8.0/10

Overall score

Run summary

Scenario: Freshness SLA Breach

Model: meta-llama/llama-3.1-8b-instruct

Roles: Data Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 2m 41s

Actual run cost: \$0.000412

Projected execution cost: \$0.4532/mo (50/biz day × 22 days)

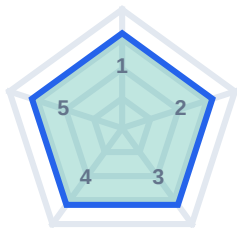
Report date: July 13, 2026 - 10:37 PM EST

Overall

8.0/10

PASS - trust=high
Pass - score 8.0/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Stale upstream identified	8/10
2	Uses SLA / freshness facts	8/10
3	Downstream blast radius	8/10
4	Structured status summary	8/10
5	Restore & verify steps	8/10

Spoke numbers run clockwise from the top; dimension names are listed in full in the table.

Technical criteria checklist

Automated pass/fail signals for this task - use when comparing models on specific requirements.

- Upstream blocker identified: Met
- SLA breach acknowledged: Met
- Downstream impact noted: Met
- Restore steps listed: Met

TECHNICAL JUDGEMENT

Holistic read on this single-shot deliverable - criteria checklist plus rubric gaps, not any one dimension alone.

8/10

Telemetry

Run metrics

Mode: multi-turn - Agent msgs: 4 - Counterpart msgs: 5 - Output: 10395 chars - Latency: 161s - Criteria: 4/4

Score analysis

Identifies int_order_enriched failure as upstream blocker.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 4 agent turns with scripted follow-ups and reviewer sign-off. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Stale upstream identified: Correctly identifies the primary technical issue from the artifacts.
- + Uses SLA / freshness facts: Uses tables, logs, or schema facts from the task brief.
- + Downstream blast radius: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured status summary: Clear structure with readable SQL, diff, or memo formatting.
- + Restore & verify steps: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Stale upstream identified: Correctly identifies the primary technical issue from the artifacts.
- Uses SLA / freshness facts: Uses tables, logs, or schema facts from the task brief.
- Downstream blast radius: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured status summary: Clear structure with readable SQL, diff, or memo formatting.
- Restore & verify steps: Strong rationale and actionable follow-through.
- Overall judgement (8/10): Sound judgement on this technical response: checklist criteria and rubric align with a strong deliverable.

Signals

Upstream blocker identified: yes - SLA breach acknowledged: yes - Downstream impact noted: yes - Restore steps listed: yes

CAVEATS

- Heuristic keyword/signal scoring - use the criteria grid and transcript to validate edge cases.

Transcript

Session

Scenario: Freshness SLA Breach

Agent response <-> Engineering counterpart

Data Engineer (AI agent) <> Engineering counterpart (AI agent)

Duration: 2m 41s

Engineering counterpart

2:34 AM

Task brief - Freshness SLA breach (moderate complexity)

A P1 mart breached its freshness SLA. Triage using the metadata JSON:

```
```json
{
 "mart": "analytics.fct_orders",
 "owner_team": "data-platform",
 "sla": {
 "max_staleness_minutes": 120,
 "tier": "P1 revenue dashboard"
 },
 "freshness": {
 "last_successful_run_utc": "2026-07-05T09:12:00Z",
 "now_utc": "2026-07-05T15:40:00Z",
 "staleness_minutes": 388,
 "status": "BREACHED"
 },
 "upstream": [
 {"model": "staging.stg_orders", "last_run_utc": "2026-07-05T15:05:00Z", "status": "success"},
 {"model": "staging.stg_customers", "last_run_utc": "2026-07-05T14:58:00Z", "status": "success"},
 {"model": "intermediate.int_order_enriched", "last_run_utc": "2026-07-05T09:12:00Z", "status": "failed", "error":
"DBT_RUN_TIMEOUT after 45m"}
],
 "downstream_consumers": ["Looker Revenue Overview", "exec_daily_orders Slack bot"],
 "note": "int_order_enriched has been failing since morning; fct_orders has not refreshed."
}
```
```

Agent response

2:35 AM - Responded in 1m 13s

Task Analysis and Solution

1. Identify the Upstream Model Blocking Refresh

The JSON metadata indicates that `intermediate.int\_order\_enriched` model is failing with a `DBT\_RUN\_TIMEOUT` error after 45 minutes. This model is likely dependent on `staging.stg\_orders` and `staging.stg\_customers`, which have successfully run. However, the failure of `int\_order\_enriched` is blocking the refresh of `analytics.fct\_orders`.

2. Blast Radius on Downstream Consumers

Downstream consumers of `analytics.fct\_orders` are:

- Looker Revenue Overview
- exec_daily_orders Slack bot

Both of these consumers rely on fresh data from `fct\_orders` to function correctly. A breach in freshness SLA may impact their ability to display accurate data.

3. Ordered Steps to Restore the Mart within SLA

To restore `analytics.fct\_orders` mart within the SLA, follow these steps:

Step 1: Investigate and Resolve `int\_order\_enriched` Failure

Investigate the `DBT\_RUN\_TIMEOUT` error on `int\_order\_enriched`. This might be due to increased data volume or query complexity.

- Review the run logs and check for any recent changes or dependencies added to the model.

- Attempt to run the model manually to identify the root cause of the failure.

- Resolve the issue by either:

 - Tuning the query performance or data processing.

 - Adding necessary dependencies or data sources.

 - Increasing the resource allocation to the model.

Step 2: Trigger Refresh on `int\_order\_enriched`

- Once the issue with `int\_order\_enriched` is resolved, trigger a manual refresh on the model.

- Monitor its progress and ensure it completes successfully.

Step 3: Trigger Refresh on `fct\_orders`

- After `int\_order\_enriched` has refreshed successfully, trigger a manual refresh on `fct\_orders`.

- Monitor its progress and ensure it completes within the SLA timeframe.

Step 4: Verify Fresher Data in Downstream Consumers

Once `fct\_orders` has refreshed successfully, verify that the downstream consumers (Looker Revenue Overview and exec_daily_orders Slack bot) are displaying fresh data from `fct\_orders`.

By following these steps, you should be able to restore the `analytics.fct\_orders` mart within the specified SLA.

Engineering counterpart
2:35 AM - Responded in 0s

Thanks - walk me through your reasoning step by step. What assumptions did you make?

Agent response

2:35 AM - Responded in 4s

Here's a detailed breakdown of my thought process:

1. Identifying the Upstream Model Blocking Refresh

Upon analyzing the JSON metadata, I noticed that the `freshness` section indicates that `analytics.fct\_orders` has been breached for 388 minutes, with the last successful run on 2026-07-05T09:12:00Z.

The `upstream` section lists models that `fct\_orders` depends on. I made the assumption that `staging.stg\_orders` and `staging.stg\_customers` are successfully updating their data, as there are no errors or failures listed for these models. However, `intermediate.int\_order\_enriched` has a `failed` status with a specific error message (`DBT\_RUN\_TIMEOUT after 45m`).

I inferred that `int\_order\_enriched` failing might be the root cause of the `fct\_orders` delay since it's the only failed upstream model.

2. Blast Radius on Downstream Consumers

In this step, I identified the consumers of `analytics.fct\_orders` by looking at the `downstream\_consumers` list. I made the assumption that both consumers are reliant on fresh data from `fct\_orders` and would be impacted by its delayed refresh.

3. Ordered Steps to Restore the Mart within SLA

To restore the mart within the SLA, I proposed the following steps:

Step 1: Investigate and Resolve `int\_order\_enriched` Failure

Investigate the `DBT\_RUN\_TIMEOUT` error: I assumed that the timeout error might be caused by a temporary issue with data processing, increased data volume, or query complexity.

Review recent changes and dependencies: I assumed that any recent changes to the model or its dependencies might be contributing to the issue.

Run the model manually: I assumed that running the model manually would help identify the root cause of the failure.

Resolve the issue: I proposed three possible resolutions (tuning query performance, adding necessary dependencies, or increasing resource allocation), assuming these are typical troubleshooting steps for a DBT model failure.

Step 2: Trigger Refresh on `int\_order\_enriched`

Trigger a manual refresh: I assumed that once the `int\_order\_enriched` issue is resolved, manually triggering a refresh would ensure it completes successfully.

Step 3: Trigger Refresh on `fct\_orders`

Trigger a manual refresh on `fct\_orders`: I assumed that once `int\_order\_enriched` has refreshed successfully, the next step would be to refresh `fct\_orders` to ensure it has the latest data.

Step 4: Verify Fresher Data in Downstream Consumers

Verify data freshness in downstream consumers: I assumed that once `fct\_orders` has refreshed successfully, both downstream consumers would display fresh data from the updated `fct\_orders`.

Assumptions made during this analysis include:

The `DBT\_RUN\_TIMEOUT` error in `int\_order\_enriched` is the root cause of the `fct\_orders` delay.

The `staging.stg\_orders` and `staging.stg\_customers` models are successfully updating their data.

The downstream consumers are reliant on fresh data from ``fct_orders``.

Investigating and resolving the ``DBT_RUN_TIMEOUT`` error involves typical troubleshooting steps, such as reviewing recent changes and dependencies, and increasing resource allocation.

Engineering counterpart
2:35 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response
2:36 AM - Responded in 38s

Before shipping the fixed mart to production, I would verify the following:

1. Data Freshness

Verify the staleness of `fct\_orders`: After triggering the refresh, I would check the staleness of `fct\_orders` to ensure it's within the specified SLA threshold (120 minutes). This ensures that the refresh was successful and the mart is in a good state.

Check the last successful run timestamp: Verify that the last successful run timestamp for `fct\_orders` is recent and within the expected timeframe.

2. Downstream Consumers

Verify data freshness in downstream consumers: I would check that both downstream consumers (Looker Revenue Overview and exec_daily_orders Slack bot) are displaying fresh data from `fct\_orders`. This is to ensure that the fix has propagated correctly to downstream systems.

3. Error Handling and Monitoring

Verify error handling for `int\_order\_enriched`: I would check that the error handling mechanisms for `int\_order\_enriched` are working correctly. If the model fails again, we want to make sure that the error is handled properly and doesn't lead to another breach.

Monitor the mart for future SLA breaches: Set up monitoring and alerting for `fct\_orders` to detect any future SLA breaches and allow for timely intervention.

4. Upstream Models

Verify successful runs of upstream models: Double-check that `staging.stg\_orders` and `staging.stg\_customers` are running successfully and not causing any issues for `int\_order\_enriched`.

5. Production Readiness

Test in a pre-production environment: Before shipping to production, I would test the fixes and verification steps in a staging or pre-production environment to ensure everything works as expected.

By verifying these aspects, we can be confident that the mart is fixed and ready for production, with minimal risk of future SLA breaches.

Engineering counterpart
2:36 AM - Responded in 0s

What are the top risks, and what's the rollback plan if this doesn't work?

Agent response

2:37 AM - Responded in 46s

Top Risks

1. Loss of Freshness: If the fix doesn't address the root cause, we risk losing freshness again, leading to another SLA breach.
2. Downstream Impact: A failed refresh might not only impact the immediate consumers of `fct\_orders` but also affect other downstream systems that rely on the freshness of `fct\_orders`.
3. Increased Staleness: If the refresh takes longer than expected or gets stuck, the staleness of `fct\_orders` might increase, further exacerbating the breach.
4. Data Inconsistencies: If the fix introduces data inconsistencies or corruption, it might lead to inaccurate insights or even trigger downstream chain reactions.
5. Infrastructure Overload: In the worst-case scenario, a failed refresh could lead to infrastructure overload, which might necessitate additional resources, impacting other critical processes.

Rollback Plan

If the fix doesn't work as expected, the primary goal is to quickly roll back to the previous, known-good state with minimal downtime and data loss. Here's a step-by-step rollback plan:

1. Freeze `fct\_orders`: Immediately prevent any further updates to `fct\_orders` to prevent data loss or corruption.
2. Restore Previous Snapshot: If available, restore the previous snapshot or backup of `fct\_orders` to ensure a known-good state.
3. Roll back `int\_order\_enriched`: Go back to the previous known-good version of `int\_order\_enriched` and revert any changes made during the attempted fix.
4. Roll back Upstream Models: If necessary, roll back upstream models (e.g., `staging.stg\_orders`) to the previous known-good state.
5. Review and Refine Troubleshooting: Analyze the root cause of the failed fix and refine the troubleshooting process to address any underlying issues.

Rollback Timeline

To minimize downtime, aim to complete the rollback within 30 minutes. This involves:

- Immediate freeze of `fct\_orders`
- Quick restoration of previous snapshot or backup
- Simple rollbacks of `int\_order\_enriched` and upstream models

In extreme cases where time is short, consider a quick restore of the mart from a previous consistent state, followed by a more thorough investigation and fix.

Additional Fail-safe Measures

To prevent such situations in the future, consider implementing additional fail-safe measures, such as:

- Automated rollbacks: Set up automated rollback scripts to quickly restore the mart to a previous consistent state in case of a failed fix.

- Monitoring and alerting: Implement real-time monitoring and alerting to detect and respond to potential SLA breaches before they become critical.

- Escalation procedures: Establish clear escalation procedures to quickly mobilize engineering resources in case of a critical issue.

By prioritizing a robust rollback plan and being prepared for the unexpected, we can minimize the impact of an unsuccessful fix and ensure continuous delivery of high-quality data to our stakeholders.

Engineering counterpart

2:37 AM - Responded in 0s

Thanks - upstream blocker and blast radius are well ordered. Prioritize revenue-facing dashboards if full freshness slips past the hour mark. Review complete - no further response needed.