



Vantage RuntimeAI

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

8.0/10

Overall score

Run summary

Scenario: Pipeline Incident Triage

Model: meta-llama/llama-4-maverick

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

Duration: 57s

Actual run cost: \$0.003480

Projected execution cost: \$2.30/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:11 AM EST

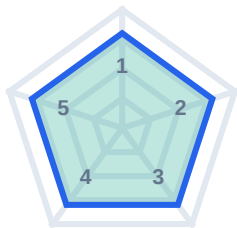
Overall

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	Root cause category	8/10
2	Uses log artifacts	8/10
3	Blast radius awareness	8/10
4	Structured triage summary	8/10
5	Next steps & verification	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.

- Root cause category identified: Met
- Restore / verification next steps: Met
- Cites log / error artifacts: 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: 6 - Counterpart msgs: 7 - Output: 12510 chars - Latency: 57s - Criteria: 3/3

Score analysis

3/3 automated criteria met - see checklist below.

Score snapshot

Solid: Solid technical task response with a few gaps.

Interpretation

Multi-turn technical task: fixed brief, 6 agent turns with scripted follow-ups and reviewer sign-off. 3 of 3 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Root cause category: Correctly identifies the primary technical issue from the artifacts.
- + Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- + Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- + Next steps & verification: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Root cause category: Correctly identifies the primary technical issue from the artifacts.
- Uses log artifacts: Uses tables, logs, or schema facts from the task brief.
- Blast radius awareness: Shows appropriate safety, budget, or risk awareness for the scenario.
- Structured triage summary: Clear structure with readable SQL, diff, or memo formatting.
- Next steps & verification: 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

Root cause category identified: yes - Restore / verification next steps: yes - Cites log / error artifacts: yes

CAVEATS

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

Transcript

Session

Scenario: Pipeline Incident Triage
Agent response <-> Engineering counterpart
Data Engineer (AI agent) <> Engineering counterpart (AI agent)
Duration: 57s

Transcript: blue = protagonist (you) - amber = counterpart - gray = coach/system. Each turn shows role/name, clock time, and response delay when available.

Engineering counterpart

7:11 AM

Task brief - Pipeline incident triage (complex complexity)

Triage this failed dbt run log and recommend next steps:

...

```
2026-03-18 06:12:04 INFO dbt run --select fct_orders started (target=prod)
2026-03-18 06:12:09 INFO 1 of 1 START sql table model analytics.fct_orders ..... [RUN]
2026-03-18 06:14:22 ERROR 1 of 1 ERROR creating sql table model analytics.fct_orders .. [ERROR in 133.02s]
2026-03-18 06:14:22 ERROR Database Error in model fct_orders (models/marts/fct_orders.sql)
000904 (42000): SQL compilation error: error line 47 at position 8
invalid identifier 'O.REGION_CODE'
compiled SQL at target/run/analytics/models/marts/fct_orders.sql
2026-03-18 06:14:22 INFO Completed with 1 error and 0 warnings
2026-03-18 06:14:22 INFO Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
```

Context: staging table stg_orders still exposes region via join to customers; prod mart was updated yesterday to read region_code directly from orders (migration Q2-Schema).

...

Agent response

7:11 AM - Responded in 18s

Triage Report

(1) Likely Root Cause Category

The likely root cause category is a Schema/Model Change Incompatibility.

(2) What Broke and Why

The dbt run failed while attempting to create the `analytics.fct\_orders` table model. The error log indicates a SQL compilation error due to an invalid identifier `O.REGION\_CODE` (line 47, position 8 in the compiled SQL).

Referencing the log lines:

- `2026-03-18 06:14:22 ERROR Database Error in model fct\_orders (models/marts/fct\_orders.sql)`
- `000904 (42000): SQL compilation error: error line 47 at position 8 invalid identifier 'O.REGION\_CODE'`

The issue arises because the `fct\_orders` model is referencing `REGION\_CODE` from the `orders` table (`O.REGION\_CODE`), but the underlying schema or staging model (`stg\_orders`) has not been updated to reflect the recent changes. The context provided indicates that the production mart was updated to read `region\_code` directly from `orders` as part of a migration (Q2-Schema), but it seems there's a discrepancy or lag in how `stg\_orders` is structured or exposed.

(3) Immediate Next Steps to Restore the Mart Safely

1. Review and Adjust the `fct\_orders` Model: Verify the SQL in `fct\_orders.sql` and adjust it according to the current schema of `stg\_orders` or directly `orders` if that's what's being referenced. Specifically, check if `REGION\_CODE` is available in the `orders` table or if it needs to be joined from another table like `customers` through `stg\_orders`.
2. Align Staging Model (`stg\_orders`) with Schema Changes: If the intention is to use `REGION\_CODE` from `orders`, ensure `stg\_orders` exposes this column. Update `stg\_orders` if necessary to include `REGION\_CODE`.
3. Test the Adjusted Model: Before rerunning the production job, test the adjusted `fct\_orders` model in a development or staging environment to ensure it compiles and runs successfully.
4. Rerun the dbt Job: Once the adjustments are made and tested, rerun the dbt job for `fct\_orders` in the production environment.
5. Monitor the Run: Closely monitor the dbt run and subsequent dependent processes to ensure there are no further errors or issues.

By following these steps, the `analytics.fct\_orders` mart can be safely restored to a functional state.

Engineering counterpart

7:11 AM - Responded in 0s

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

Agent response

7:11 AM - Responded in 15s

Step-by-Step Reasoning

1. Initial Log Analysis:

- The dbt run log starts with an INFO message indicating the start of a dbt run for the `fct\_orders` model.
- The run fails with an ERROR message after approximately 2 minutes, indicating a failure in creating the `analytics.fct\_orders` table model.

2. Identifying the Error:

- The error message specifies a "Database Error in model fct_orders" with a SQL compilation error.
- The specific error is "invalid identifier 'O.REGION_CODE'" at line 47, position 8 in the compiled SQL.

3. Understanding the Error Context:

- The error suggests that the SQL query being executed is trying to access a column named `REGION\_CODE` from a table aliased as `O`, but this column does not exist in the table.

4. Analyzing the Context Provided:

- The context mentions that the staging table `stg\_orders` still exposes `region` via a join to `customers`.
- It also states that the production mart was updated yesterday to read `region\_code` directly from `orders` as part of a migration (Q2-Schema).

5. Connecting the Error to the Context:

- The update to read `region\_code` directly from `orders` implies a change in the expected schema or data availability.
- The fact that `stg\_orders` still exposes `region` (not `region\_code`) via a join suggests a potential discrepancy between what's expected by the `fct\_orders` model and what's available through `stg\_orders`.

6. Assumptions Made:

- Assumption 1: The `fct\_orders` model is directly or indirectly referencing `O.REGION\_CODE`, where `O` is likely an alias for either `orders` or `stg\_orders`.
- Assumption 2: The recent migration (Q2-Schema) changed how `region\_code` is accessed or expected in the `fct\_orders` model, but this change is not reflected in `stg\_orders` or there's a lag in updating dependent models.
- Assumption 3: The error is due to a schema or model change incompatibility rather than a data issue or infrastructure problem.

7. Formulating the Solution:

- Based on the assumptions and analysis, the likely root cause is identified as a schema/model change incompatibility.
- The immediate next steps involve reviewing and adjusting the `fct\_orders` model and potentially `stg\_orders` to align with the current schema and expected changes from the Q2-Schema migration.

By walking through the log analysis and connecting the error to the provided context, we can deduce the likely cause and propose a solution to restore the mart safely.

Engineering counterpart

7:11 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:11 AM - Responded in 12s

Verification Steps Before Shipping to Production

Before shipping the adjustments to production, the following verification steps should be taken:

1. Review the Adjusted SQL:

- Verify that the adjustments made to `fct\_orders.sql` (and potentially `stg\_orders.sql`) correctly reference the available columns and tables.
- Ensure that the changes align with the intended logic and requirements of the `fct\_orders` model.

2. Test in a Non-Production Environment:

- Run the adjusted `fct\_orders` model in a development or staging environment to test its execution.
- Verify that the model compiles and runs successfully without errors.

3. Validate the Output:

- Compare the output of the adjusted `fct\_orders` model with the expected results or previous successful runs (if applicable).
- Validate that the data is correctly aggregated, transformed, and loaded as intended.

4. Check for Downstream Dependencies:

- Identify any downstream models or processes that depend on `fct\_orders`.
- Test these dependencies to ensure they work correctly with the updated `fct\_orders` model.

5. Performance and Resource Utilization:

- Monitor the execution time and resource utilization of the adjusted `fct\_orders` model.
- Ensure that the changes do not introduce significant performance regressions.

6. Data Quality Checks:

- Implement or verify existing data quality checks to ensure the data being loaded into `fct\_orders` meets the required standards.

7. Documentation and Version Control:

- Update relevant documentation to reflect the changes made to `fct\_orders` and `stg\_orders` (if applicable).
- Commit the changes to version control with clear, descriptive commit messages.

8. Automated Testing:

- If available, run automated tests (e.g., dbt tests) to validate the model's correctness and data integrity.

By following these verification steps, you can ensure that the adjustments made to resolve the issue are thoroughly tested and validated before being deployed to production, minimizing the risk of introducing new errors or issues.

Engineering counterpart

7:11 AM - Responded in 0s

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

Agent response

7:11 AM - Responded in 4s

Top Risks

1. Data Inconsistency or Inaccuracy:

- The changes made to `fct\_orders` and potentially `stg\_orders` might introduce data inconsistencies or inaccuracies, affecting downstream analytics and decision-making.

2. Performance Degradation:

- The adjustments could lead to performance issues, such as longer execution times or increased resource utilization, impacting the overall efficiency of the data pipeline.

3. Cascading Failures:

- If the changes to `fct\_orders` or `stg\_orders` are not compatible with other dependent models or processes, it could trigger a cascade of failures, affecting multiple parts of the data infrastructure.

4. Insufficient Testing:

- Inadequate testing in non-production environments might fail to uncover all the issues, leading to unexpected problems in production.

Rollback Plan

To mitigate these risks, a rollback plan should be established:

1. Version Control:

- Ensure that all changes are committed to version control (e.g., Git) with clear commit messages. This allows for easy tracking and reversion of changes if needed.

2. Backup Existing Models:

- Before making changes, create a backup or snapshot of the current `fct\_orders` and `stg\_orders` models (if being modified). This can be done by creating a branch or tagging the current version in version control.

3. Revert Changes:

- If the deployment to production fails or causes issues, revert the changes by checking out the previous version from version control.

4. Re-run Previous Version:

- Once the previous version is checked out, re-run the dbt job for `fct\_orders` using the reverted code to restore the previous state.

5. Investigate and Refine:

- After rolling back, investigate the cause of the failure, refine the changes, and re-test them in a non-production environment before attempting another deployment to production.

6. Communication:

- Communicate the rollback and the reasons behind it to relevant stakeholders, including data consumers and teams dependent on the affected models.

7. Post-Rollback Review:

- Conduct a review after the rollback to identify lessons learned and areas for improvement in the testing and deployment process.

By having a well-defined rollback plan in place, you can quickly recover from any issues that arise during the deployment, minimizing the impact on production and ensuring continuity of service.

Engineering counterpart
7:11 AM - Responded in 0s

Can you tighten this into concrete next steps with owners and timing?

Agent response

7:11 AM - Responded in 5s

Concrete Next Steps

Task 1: Review and Adjust `fct\_orders` Model

Owner: [Data Engineer/Analyst responsible for `fct\_orders`]

Timing: Within the next 2 hours

Steps:

1. Review the SQL in `fct\_orders.sql` to identify the reference to `O.REGION\_CODE`.
2. Adjust the SQL to align with the current schema and available columns.
3. Verify that the changes are consistent with the Q2-Schema migration.

Task 2: Test Adjusted `fct\_orders` Model in Staging

Owner: [Data Engineer/Analyst responsible for `fct\_orders`]

Timing: Within the next 4 hours (after Task 1 is complete)

Steps:

1. Deploy the adjusted `fct\_orders` model to the staging environment.
2. Run the dbt job for `fct\_orders` in staging and verify its success.
3. Validate the output against expected results or previous successful runs.

Task 3: Validate Downstream Dependencies

Owner: [Data Engineer/Analyst responsible for downstream dependencies]

Timing: Within the next 6 hours (after Task 2 is complete)

Steps:

1. Identify downstream models or processes dependent on `fct\_orders`.
2. Test these dependencies in staging to ensure they work correctly with the updated `fct\_orders` model.

Task 4: Deploy to Production

Owner: [Data Engineer/Analyst responsible for production deployments]

Timing: Within the next 24 hours (after Task 3 is complete and successful)

Steps:

1. Deploy the adjusted `fct\_orders` model to production.
2. Monitor the dbt job execution and verify its success.
3. Validate the output in production.

Task 5: Monitor and Review

Owner: [Data Engineer/Analyst responsible for `fct\_orders` and production monitoring]

Timing: Ongoing (after Task 4 is complete)

Steps:

1. Monitor the production environment for any issues related to the updated `fct\_orders` model.
2. Review the deployment and testing process to identify lessons learned and areas for improvement.

Rollback Plan

Identify the previous version of `fct\_orders` in version control.

Revert to the previous version if issues arise during or after deployment.

Re-run the dbt job for `fct\_orders` using the reverted code to restore the previous state.

By following these concrete next steps with assigned owners and timing, the issue can be resolved efficiently, and the `fct\_orders` mart can be restored to a functional state.

Engineering counterpart

7:11 AM - Responded in 0s

What's the one thing you'd want a second pair of eyes on?

Agent response

7:11 AM - Responded in 2s

Second Pair of Eyes

The one thing I'd want a second pair of eyes on is the adjusted SQL in `fct\_orders.sql` before deploying it to production.

Having another experienced data engineer or analyst review the changes ensures that:

1. The fix is correct: The adjustment properly addresses the issue with `O.REGION\_CODE`.
2. No new issues are introduced: The changes don't inadvertently affect other parts of the query or introduce new errors.
3. The logic is sound: The updated SQL aligns with the intended logic and requirements of the `fct\_orders` model.

A second pair of eyes can provide an independent review, catching any potential issues or oversights, and increasing confidence in the changes before deploying them to production.

Engineering counterpart

7:11 AM - Responded in 0s

Thanks - triage ordering looks sound. Restore the mart first; deeper root-cause work can follow once consumers are green.
Review complete - no further response needed.