



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: openai/gpt-4.1-nano

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

Duration: 1m 17s

Actual run cost: \$0.001740

Projected execution cost: \$1.15/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:13 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: 12455 chars - Latency: 77s - 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: 1m 17s

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

7:12 AM - Responded in 11s

1. Likely Root Cause Category:

Schema mismatch / code breakage due to recent schema migration.

2. What broke and why, referencing log lines:

The error `invalid identifier 'O.REGION_CODE'` indicates that the SQL query attempted to reference the `REGION_CODE` column in the `orders` table (aliased as `O`), but that column does not exist in the current schema. The logs show the failure occurred during the creation of `analytics.fct_orders` (at line 47 of the SQL), which likely contains a reference to `O.REGION_CODE`. Context notes that the production mart was updated yesterday to read `region_code` directly from `orders`, migrating away from relying on joins to the `customers` table. This schema change appears not to have been fully incorporated into the dbt model, leading to the missing column reference.

3. Immediate Next Steps to Restore the Mart Safely:

- Short-term fix:
 - Check the current schema of the `orders` table in the production environment to confirm the presence or absence of the `region_code` column.
 - Update the `models/marts/fct_orders.sql` model with a conditional or versioned logic to refer to `region_code` directly, aligning with the recent schema change.
- Data validation:
 - Verify if the `region_code` column is populated for existing data. If not, determine fallback logic (e.g., nulls or default values).
- Rollback if necessary:
 - If the change in the model is not ready for deployment, consider reverting the last schema change or deploying a quick fix that temporarily reverts to the previous join-based approach until the model is updated accordingly.
- Deployment:
 - After fixing the model, run dbt in a controlled environment to validate the change before a full production run.
 - Monitor logs closely during re-run to confirm resolution.

Engineering counterpart

7:12 AM - Responded in 0s

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

Agent response

7:12 AM - Responded in 20s

Certainly. Here's my reasoning process broken down step by step, including the assumptions I made:

Step 1: Analyzing the Error Message

- Log line:

``SQL compilation error: error line 47 at position 8 invalid identifier 'O.REGION_CODE``

- Inference:

The SQL query tried to reference a column ``REGION_CODE`` aliased as ``O``, likely from the ``orders`` table.

- Assumption:

The model ``fct_orders`` contains a line that references ``O.REGION_CODE``, probably in a ``SELECT`` statement, at or around line 47.

Step 2: Understanding the Context

- Context info:

The brief indicates that the production mart was updated yesterday to read ``region_code`` directly from ``orders``, moving away from reliance on joins with the ``customers`` table (a migration in Q2-Schema).

- Assumption:

Before the update, the ``region_code`` was likely derived via a join with the ``customers`` table; after the update, the ``orders`` table should have a ``region_code`` column directly.

Step 3: Connecting the Error to Schema Changes

- Since the error references an invalid identifier, I assume that the column ``REGION_CODE`` does not exist in the current schema of ``orders``.

- Assumption:

The schema change (adding ``region_code`` directly in ``orders``) was recent and not fully reflected in the code of the ``fct_orders`` model, leading to the mismatch.

Step 4: Cross-Checking the Context

- The context mentions that ``stg_orders`` still exposes ``region`` via a join to ``customers``, implying that the staging layer hasn't fully migrated to the new schema.

- Assumption:

The ``fct_orders`` model was perhaps not updated accordingly, and references ``O.REGION_CODE`` assuming the new schema, which isn't yet in effect or was incorrectly referenced.

Step 5: Making Corrective Actions

- To fix this, I assumed that updating the model to reference the ``region_code`` column directly from ``orders`` is the correct approach, but the code may still have hardcoded references to ``O.REGION_CODE``.

- Immediate action involves verifying the current schema of ``orders`` and then updating the model's SQL code accordingly.

Assumptions Summary:

- The ``fct_orders`` model's SQL contains a reference to ``O.REGION_CODE``.

- The ``orders`` table in production recently migrated to include a ``region_code`` column, but the dbt model has not been fully updated to reference it.

- The ``stg_orders`` still exposes region via a join, indicating incomplete migration in staging models.

- The root cause is a mismatch between the schema change and the model code, leading to the invalid identifier error.

In essence, I assumed the root cause is a schema-model mismatch due to recent migration work that has not yet been fully reflected in the dbt models, resulting in referencing a column that no longer exists.

Engineering counterpart

7:12 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

7:12 AM - Responded in 9s

Before deploying any fix to production, I would verify the following:

1. Current Schema of the `orders` Table

- Confirm whether the `region\_code` column exists in the `orders` table in the production environment. This can be done via schema introspection queries or metadata inspection tools.

2. Updated Model SQL Code

- Ensure that the `models/marts/fct\_orders.sql` has been modified to reference `region\_code` directly from `orders` instead of relying on joins or the old schema, particularly around line 47 where the error occurred.

3. Code correctness and compliance

- Run the SQL query (the part that references `region\_code`) independently against a staging or development environment to verify it compiles successfully without errors.
- Validate that the change aligns with the intended schema migration and data model design.

4. Data Validation

- Check if the `region\_code` column in `orders` is populated for existing data. If it contains nulls or missing values, decide on fallback logic or note implications for downstream analyses.

5. Test Runs

- Execute the dbt run in a controlled environment (staging or dev) to confirm that the fix addresses the issue without introducing new errors.
- Run relevant tests, including any schema tests and data tests, to verify data integrity.

6. Impact Analysis

- Assess if the change affects other models, downstream reports, or dashboards.

7. Version Control & Documentation

- Confirm that the fix is committed, reviewed, and documented appropriately for traceability.

8. Monitoring Plan

- Prepare monitoring during and after deployment to quickly detect any anomalies or failures.

In sum:

I would verify schema alignment, code correctness, data quality, and perform thorough testing to ensure the fix is safe, effective, and aligns with the recent schema migration before shipping to production.

Engineering counterpart

7:12 AM - Responded in 0s

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

Agent response

7:12 AM - Responded in 15s

Top Risks

1. Schema Mismatch or Inconsistency:

- If the `region\_code` column isn't present in the `orders` table as expected, deploying the fix will fail again or result in incomplete data models.

2. Incomplete Migration or Data Gaps:

- If the `region\_code` column exists but contains nulls or inconsistent data, downstream analyses or dashboards relying on this field could be impacted.

3. Code Drift or Deployment Errors:

- There is a risk of deploying incorrect or partially updated SQL code, leading to runtime errors or data discrepancies.

4. Reactivation of Old Dependencies:

- If the updated model inadvertently reintroduces reliance on outdated joins or schemas, the failure could recur.

5. Performance Impact:

- Changes in the model or schema could affect query performance, especially if the `region\_code` column is large or not indexed properly.

Rollback Plan if the Fix Doesn't Work

1. Revert to the Last Known Good State:

- Re-deploy the previous version of the `fct\_orders` model that was working before the recent changes. This involves restoring the SQL code to the last stable version in version control.

2. Restore Schema if Altered:

- If the migration to include `region\_code` in `orders` was partial or incomplete, coordinate with the schema management team to revert or correct the schema.

3. Verify Data Integrity:

- Run data validation checks to ensure no incomplete or inconsistent data exists from the failed attempt.

4. Disable the Failing Model:

- If immediate restoration is needed, temporarily disable the problematic model or set it to skip during the next run, preventing further failures downstream.

5. Communicate and Log the Rollback:

- Document the rollback action, communicate with stakeholders, and log the incident for post-mortem analysis.

6. Plan for Re-Deployment:

- After verifying the fix, re-apply the correct changes with additional validation, then redeploy once confidence is restored.

In summary:

The primary rollback involves reverting the model code to the last stable version, confirming the schema's state, and ensuring data integrity before attempting re-deployment. This minimizes downtime and prevents propagation of errors.

Engineering counterpart

7:12 AM - Responded in 0s

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

Agent response

7:13 AM - Responded in 17s

Certainly. Here's a set of concrete next steps with assigned owners and recommended timelines:

Immediate Next Steps (Within 1-2 hours)

1. Verify `orders` schema in production
 - Owner: Data Engineer / DBA
 - Action: Run a schema query (e.g., `SHOW COLUMNS IN orders` or equivalent) to confirm whether `region\_code` exists.
 - Timing: Immediately
2. Review and update the dbt model SQL
 - Owner: Data Engineer / Developer
 - Action: Confirm that `models/marts/fct\_orders.sql` references `region\_code` correctly.
 - Timing: Within 1 hour
3. Run a local/staging DBT run with the updated code
 - Owner: Data Engineer / Developer
 - Action: Execute `dbt run --select fct\_orders` in a staging environment to validate compilation and correctness.
 - Timing: Within 1-2 hours
4. Validate data quality in staging
 - Owner: Data Analyst / Data Engineer
 - Action: Query recent `orders` data to verify `region\_code` presence and population.
 - Timing: Within 2 hours

Short-term Remediation Plan (Next 2-4 hours)

5. Prepare a rollback plan
 - Owner: Data Engineering Lead / Release Manager
 - Action: Identify last stable version of `fct\_orders.sql` and prepare deployment steps to revert to it if needed.
 - Timing: Immediately after validation
6. Communicate status & plan to stakeholders
 - Owner: Engineering Manager / Team Lead
 - Action: Notify relevant teams of the current status, validation results, and contingency plan.
 - Timing: Within 2 hours

Deployment & Monitoring (Next 4-6 hours)

7. Deploy the fix to production
 - Owner: Deployment Engineer / Data Engineer
 - Action: Run the fixed dbt model in production environment, monitor logs for errors.
 - Timing: After successful validation in staging
8. Confirm successful run and data quality
 - Owner: Data Analyst / Data Engineer
 - Action: Verify data completeness and correctness post-deploy.
 - Timing: Immediately after deployment

Contingency (If issues persist)

9. Execute rollback if necessary
 - Owner: Deployment Lead / Data Engineering Lead
 - Action: Revert to the last good version of the model and verify data consistency.
 - Timing: Within 1 hour of detecting failure post-deployment

10. Post-mortem & Fix Validation

- Owner: Data Engineering Team
- Action: Investigate root cause, fix the schema or code mismatch, and plan re-deployment with improved validations.
- Timing: Next 1-2 days

Summary:

- Confirm schema correctness (owner: DBA) - immediately.
- Validate model code and run in staging (owners: dev team) - within 1-2 hours.
- Prepare rollback plan (owner: lead) - immediately.
- Deploy after validation, monitor, and verify data - within 4-6 hours.
- If issues occur, rollback swiftly, then investigate root cause, and re-deploy once validated.

Engineering counterpart

7:13 AM - Responded in 0s

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

Agent response

7:13 AM - Responded in 5s

The one thing I'd want a second pair of eyes on is the actual SQL code of the ``fct_orders`` model, specifically around line 47, to ensure that it correctly references the ``region_code`` column from the ``orders`` table and aligns with the recent schema migration.

This review would confirm whether the model code has been properly updated to match the current schema, preventing potential oversights or typos that could cause repeated failures.

Engineering counterpart

7:13 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.