



## 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: amazon/nova-micro-v1

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

Duration: 20s

Actual run cost: \$0.000609

Projected execution cost: \$0.4019/mo (30/biz day × 22 days)

Report date: July 14, 2026 - 3:09 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: 18745 chars - Latency: 20s - 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: 20s

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

## Engineering counterpart

7:09 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

7:09 AM - Responded in 4s

Triage Analysis of Failed dbt Production Run:

### ### 1. Likely Root Cause Category

Schema Migration Issue

### ### 2. What Broke and Why

From the log lines, the dbt run for the `analytics.fct\_orders` model encountered an error at line 47 of the `fct\_orders.sql` file while attempting to compile the SQL:

...

```
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
```

...

The error `invalid identifier 'O.REGION\_CODE'` implies that the SQL code in the `fct\_orders.sql` model is trying to reference `REGION\_CODE` from a table alias `O`, but this identifier is unrecognized.

Given the context that the `stg\_orders` table still joins region via `customers`, but the production mart `analytics.fct\_orders` recently changed to read `region\_code` directly from `orders` (as per migration Q2-Schema), the SQL likely hasn't been updated correctly to reflect this schema change.

Why it broke:

- The migration to reading `region\_code` directly from `orders` in the mart did not align correctly with the SQL model's reference which still expects `REGION\_CODE` from a different source.

### ### 3. Immediate Next Steps to Restore the Mart Safely

To safely restore the mart and avoid further issues with other dependent models or data pipelines:

#### #### Step 1: Correct the SQL Model

- Verify the changes in the production schema.
- Ensure the `fct\_orders.sql` model has been updated to correctly reference `region\_code` from the `orders` table directly, without any incorrect join references to `customers` or other tables.

```sql

-- Example SQL snippet for verification, replace based on actual table structure and columns

select

O.order\_id,

O.region\_code, -- Correct column reference as per new schema

...

from

orders O -- Assuming 'orders' is the base table now directly providing region\_code

...

```

#### #### Step 2: Validate Changes

- Run a dry-run or test run of dbt to ensure the corrected model compiles successfully without identifier errors or logical inconsistencies.

```bash

```
dbt run --select fct_orders --full-refresh --no-version-check
---
```

#### Step 3: Re-run the dbt Model

- Deploy the corrected SQL model.

```
```bash
dbt run --select fct_orders
---
```

#### Step 4: Verify Data Integrity

- After the model runs successfully, verify that data in `analytics.fct\_orders` is accurate and reflects the expected schema.

```
```sql
SELECT FROM analytics.fct_orders
LIMIT 100; -- Or any verification query to assert data correctness
---
```

#### Step 5: Monitor for Potential Dependent Issues

- Since schema changes can impact other models, monitor the dbt run logs for any subsequent errors and address them promptly.

---

Following these steps should help restore the `analytics.fct\_orders` mart and align the model to the new schema.

**Engineering counterpart**  
**7:09 AM - Responded in 0s**

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

## Agent response

7:09 AM - Responded in 4s

Certainly! Let's walk through this step by step, focusing on why and how we arrived at our conclusions, along with any relevant assumptions made.

### ### Context

We have a log of a failed dbt (data build tool) run for the `analytics.fct\_orders` table:

#### 1. Log Entry:

- The run was initiated (`INFO dbt run --select fct\_orders started`).
- The model started (`INFO 1 of 1 START sql table model analytics.fct\_orders`).

#### 2. Failure:

- The dbt run failed with the following error:

...

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

...

### ### Analysis

#### 1. Initial Observations:

- The error `Database Error in model fct\_orders` indicates an issue specifically with the SQL query defined in `fct\_orders.sql`.
- The error message `SQL compilation error: error line 47 at position 8: invalid identifier 'O.REGION\_CODE'` signifies that the SQL statement tried to reference an identifier that doesn't exist in the given context. In this case, `O.REGION\_CODE` doesn't seem to be recognized.
- `compiled SQL at target/run/analytics/models/marts/fct\_orders.sql` points to the location of the problematic SQL file.

#### 2. Contextual Info Provided:

- `stg\_orders` still joins region via `customers`.
- The production mart was recently updated to read `region\_code` directly from `orders` (migration Q2-Schema).

### ### Assumptions and Reasoning

#### - Assumption 1: The Schema Migration is Complete but Incomplete in the SQL Model

- Reason: The production schema has been updated to pull region\_code directly from the `orders` table, but the referenced SQL model (`fct\_orders.sql`) might not have been updated accordingly. Given that it's trying to compile a reference (`O.REGION\_CODE`), it's likely the model was not updated to reflect this schema update.

#### - Assumption 2: The SQL Syntax is Correct Except for the 'Invalid Identifier'

- Reason: If the SQL was compiling correctly apart from this identifier issue, the overall structure of the query should still be intact. The failure is purely due to the unrecognized identifier.

#### - Assumption 3: The Change in Reference is the Sole Issue

- Reason: Given that dbt's error message pointed directly to the `invalid identifier 'O.REGION\_CODE'`, the problem is likely isolated to this issue in the SQL. There are no downstream logical or syntactical issues with the rest of the SQL.

### ### Immediate Steps

1. Correct the SQL Model:

- Action: Identify and revise the SQL code in `fct\_orders.sql` to ensure it references `region\_code` correctly from the newly updated schema.
- Verification: Ensure the SQL now directly references the `region\_code` from the `orders`

**Engineering counterpart**  
**7:09 AM - Responded in 0s**

What would you verify before shipping this to production?

## Agent response

7:09 AM - Responded in 3s

Before shipping the updated SQL model to production, there are several verification steps that should be undertaken to ensure that the changes address the issue and do not introduce new problems. Here's a step-by-step checklist:

### ### 1. Local Dry-Run

Action: Run the dbt model locally to validate SQL syntax and logic in a controlled environment.

```
``bash
dbt run --select fct_orders --full-refresh --no-version-check
``
```

Verification: Confirm that there are no SQL compilation errors or runtime errors. The model should execute successfully, and the output should match expected data patterns.

### ### 2. Data Validation

Action: Examine a sample of the result set from the updated mart to ensure the data is correct.

```
``sql
SELECT * FROM analytics.fct_orders
LIMIT 100;
``
```

Verification: Verify that all columns, including `region\_code`, are correctly populated. Ensure there are no missing values or data anomalies, especially in the `region\_code` field.

### ### 3. Schema Validation

Action: Review the schema of the `analytics.fct\_orders` to confirm the expected structure and data types.

```
``sql
DESC TABLE analytics.fct_orders;
``
```

Verification: Ensure that the schema reflects the expected columns and data types, particularly `region\_code`.

### ### 4. Data Lineage Check

Action: Review dbt documentation or model comments to map the relationship between source tables and the new mart.

Verification: Confirm that the transformations applied in the `fct\_orders` model are aligned with the intended business logic and that data flows as expected from sources to the mart.

### ### 5. Test Environment Deployment

Action: Deploy the model to a staging or test environment that mirrors production as closely as possible.

Verification: Ensure the model performs as expected in the staging environment. Monitor for any dbt run errors or warnings and validate the output against expected results.

### ### 6. Performance Review

Action: Measure the execution time and resource usage of the updated model.

Verification: Confirm that the model runs efficiently within acceptable performance metrics. If performance degrades, investigate the causes and optimize the SQL if necessary.

### ### 7. Backup and Rollback Plan

Action: Document a backup and rollback plan prior to deploying the changes to production.

Verification: Ensure that a recent backup of the current state of `analytics.fct\_orders` is available. Have a rollback strategy ready in case deployment issues arise.

### ### 8. Run Unit Tests

Action: Execute any dbt tests specifically configured for the `fct\_orders` model.

```
``bash
dbt test --select fct_orders
``
```

Verification: Verify that all tests pass and confirm that the model adheres to defined data quality rules and constraints.

### ### 9. Peer Review

Action: Arrange for a peer or team member to review the SQL changes for any overlooked issues.

Verification: Ensure there are no syntactical errors, logical mistakes, or missed optimizations.

### ### 10. Documentation Update

Action: Update any relevant documentation regarding the model changes, schema updates, or business logic adjustments.

Verification: Ensure that all team members are aware of the changes, expected behavior, and any potential impacts on downstream processes.

After completing these steps, you can be more confident that the correction is appropriate and that the risks of deployment are minimized. Proceed to production with the necessary caution, monitoring closely for any unexpected issues after initial deployment.

**Engineering counterpart**  
**7:09 AM - Responded in 0s**

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

## Agent response

7:09 AM - Responded in 4s

When deploying any changes to a production environment, especially with data models like ``fct_orders``, several risks could arise and a robust rollback plan is crucial. Let's outline the key risks and the corresponding rollback plan:

### ### Top Risks

#### 1. Data Integrity Issues:

- Risk: The data in ``analytics.fct_orders`` could become inaccurate or incomplete due to incorrect schema references or transformations.
- Impact: Downstream analytics and reporting could be negatively affected, leading to incorrect business decisions.

#### 2. Performance Degradation:

- Risk: The new SQL could run slower than expected, affecting the real-time availability of data.
- Impact: Delays in data availability could affect businesses dependent on timely data.

#### 3. Schema Mismatch:

- Risk: Incorrectly updated schema may not match expectations, leading to issues in data structures or relationships that could result in inaccurate reporting.
- Impact: Incorrect reporting metrics could lead to faulty business insights.

#### 4. Interruption of Data Pipeline:

- Risk: Failure in the dbt run could interrupt the overall data pipeline, causing delays in data flow.
- Impact: This could result in missed insights and potential losses due to delayed decision-making.

#### 5. Incomplete Data Transfer:

- Risk: Part of the data may not get transferred due to logical errors in the model, leading to an incomplete dataset.
- Impact: Incomplete datasets could result in biased analyses and decisions.

### ### Rollback Plan

The rollback plan is designed to restore the system to its previous state as quickly as possible in case the deployed changes lead to significant issues.

#### 1. Immediate Suspension of Data Loading:

- Immediate Action: Halt any further data loading to ``analytics.fct_orders`` to prevent the propagation of any incorrect data.

#### 2. Restoration from Backup:

- Action: Restore ``analytics.fct_orders`` from a recent, known-good backup taken before the changes were made.

#### 3. Revert to Previous Schema:

- Action: Revert the schema of ``analytics.fct_orders`` to its previous state that included the correct references necessary for querying regions via the ``customers`` table.

#### 4. Rollback Deployment:

- Action: Use version control to revert the ``fct_orders.sql`` file to the last working version.
- Verification: Confirm the correctness of the reverted model through unit tests and data validation.

#### 5. Notification and Communication:

- Action: Notify relevant stakeholders about the rollback, including the development, data engineering, and business teams.
- Communication: Provide details on what went wrong, what the rollback entailed, and next steps.

#### 6. Detailed Analysis Post-Rollback:

- Action: Conduct a post-mortem analysis to understand the root cause of the original failure and prevent recurrence.

- Follow-Up: Implement the necessary changes identified during the analysis to resolve the root cause before redeploying.

### ### Monitoring and Follow-Up

After a rollback, continuous monitoring of the database and dbt jobs should be implemented to ensure any anomalous behavior is quickly identified and addressed. Regular check-ins with the team to resolve the underlying issue and ensure the system's stability going forward should also be scheduled.

By having a well-defined rollback plan and being prepared to execute it, the impact of any failures due to model changes can be minimized, safeguarding against significant disruptions to your data pipeline and the insights that rely on it.

**Engineering counterpart**  
**7:09 AM - Responded in 0s**

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

## Agent response

7:09 AM - Responded in 3s

Certainly! Here's a more detailed, concrete rollback plan with owners assigned and timings:

### ### Rollback Plan with Timelines and Assignments

#### #### Immediate Actions (Within minutes of detection)

##### 1. Halt Data Loading to `analytics.fct\_orders`:

- Owner: Data Pipeline Manager
- Action: Use service management software or manual intervention to stop any new data being loaded into `analytics.fct\_orders`.
- Timing: Immediate, starts the moment the issue is identified.

##### 2. Initiate Notification Procedure:

- Owner: Communication Lead
- Action: Notify key stakeholders including Engineering, Data Engineering, and Business Analysts via preferred communication channels (e.g., Slack, email).
- Timing: Begin once the issue is identified and the data loading has been stopped.

#### #### Quick Restoration (Within 30 minutes after issue identification)

##### 3. Restore `analytics.fct\_orders` from Backup:

- Owner: Database Administrator
- Action: Restore the table from the most recent backup taken before the issue was introduced.
- Timing: Restore operation should be completed within 30 minutes of starting the process.

##### 4. Reversion to Previous Database Schema:

- Owner: Database Administrator
- Action: Re-apply schema changes that were in place prior to the current update, specifically the part where region was joined through the `customers` table.
- Timing: Begin reversion immediately following the backup restoration, ensuring minimal downtime for business operations.

##### 5. Revert `fct\_orders.sql` to Last Working Version:

- Owner: Data Engineer
- Action: Use version control (e.g., Git) to revert `fct\_orders.sql` file to the last functionally verified version.
- Timing: Complete within 30 minutes of initial detection.

##### 6. Test `fct\_orders.sql` Locally:

- Owner: Data Engineer
- Action: Execute the reverted SQL model locally using dbt to check for any compilation errors or logical issues.
- Timing: Within 1 hour from decision to revert.

#### #### Validation and Deployment (Within 4 hours post issue identification)

##### 7. Run Unit Tests for `fct\_orders.sql`:

- Owner: QA Engineer
- Action: Run all available unit tests specific to the `fct\_orders` model.
- Timing: Complete tests within 1 hour following local testing.

##### 8. Deploy Revert to Staging Environment:

- Owner: QA Engineer
- Action: Deploy the reverted and validated model to the staging environment for additional scrutiny.
- Timing: Staging deployment should be completed within 2 hours following unit tests.

9. Conduct Data Validation in Staging:

- Owner: Data Analyst
- Action: In conjunction with the QA Engineer, pull a representative sample of data from the staging `analytics.fct\_orders` and perform thorough validation against expected results.
- Timing: Validation should be done within 1 hour after staging deployment.

#### Final Rollback Execution and Monitoring (Within 6 hours post issue identification)

10. Deploy Revert to Production:

- Owner: Deployment Lead
- Action: Once validations in staging are complete and approved, deploy the reverted model to production.
- Timing: Deploy within 1 hour after staging validation approval.

11. Continuous Monitoring Post-Rollback:

- Owner: Monitoring Engineer
- Action: Monitor the system post-deployment for any anomalies in data or performance issues.
- Timing: Continuous monitoring for 24 hours post-rollback.

### Follow-

**Engineering counterpart**  
**7:09 AM - Responded in 0s**

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

**Agent response**

**7:09 AM - Responded in 2s**

When rolling back changes and particularly in a high-stakes environment like production, the one critical aspect that should always receive a second pair of eyes is the final deployment of the reverted SQL model to production.

### Why This Is Critical:

1. Verification of Deployment: The last step before the changes go live in production is the ultimate verification. An additional set of eyes can catch any discrepancies in the deployment process.
2. Correctness of Reversion: Ensure that exactly the right version has been redeployed and no additional changes were mistakenly included or omitted.
3. Validation of Deployment Accuracy: Confirm that the deployment process completed successfully without errors, and that `analytics.fct\_orders` is now using the correct schema and SQL logic.
4. Final Review of Rollback Plan Execution: An extra check helps validate that all steps of the rollback plan have been executed as intended.

### Process for Second Pair of Eyes:

- Assignment: Assign responsibility for the final verification to a team member not involved in the initial rollback operations. Ideally, this should be another Data Engineer or QA Engineer who has the technical expertise and a fresh perspective.
- Pre-Deployment Review: Before deployment, this second individual should review the updated model and deployment scripts to affirm that everything is correct.
- Post-Deployment Verification: After the deployment, the second reviewer should participate in the initial verification step to ensure that the system behaves as expected and that no unintended issues have arisen.

This final verification step is the safeguard that ensures all else has been done correctly, thus safeguarding against errors that could lead to data integrity problems or operational disruptions.

**Engineering counterpart**  
**7:09 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.