



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-3.3-70b-instruct

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

Duration: 1m 31s

Actual run cost: \$0.001340

Projected execution cost: \$1.47/mo (50/biz day × 22 days)

Report date: July 14, 2026 - 2:55 AM EST

Overall

8.0/10

PASS - trust=high

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

CAPABILITY RUBRIC



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: 13658 chars - Latency: 91s - 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: 1m 31s

Engineering counterpart

6:54 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

6:54 AM - Responded in 8s

Freshness SLA Breach Analysis and Resolution Plan

Summary

The `analytics.fct\_orders` mart has breached its freshness SLA with a staleness of 388 minutes, exceeding the maximum allowed staleness of 120 minutes. The breach is caused by the failure of the `intermediate.int\_order\_enriched` model, which is an upstream dependency of `fct\_orders`.

Upstream Model Blocking Refresh

The upstream model blocking the refresh of `fct\_orders` is `intermediate.int\_order\_enriched`. This model failed with a `DBT\_RUN\_TIMEOUT` error after 45 minutes, which occurred at `2026-07-05T09:12:00Z`.

Blast Radius on Downstream Consumers

The blast radius of this breach affects two downstream consumers:

1. Looker Revenue Overview: This dashboard relies on up-to-date data from `fct\_orders` to provide accurate revenue insights. The stale data may lead to incorrect decisions based on outdated information.
2. exec_daily_orders Slack bot: This bot likely depends on fresh data from `fct\_orders` to send daily order updates. The breach may cause the bot to send outdated or incorrect information, potentially affecting business operations or customer communications.

Ordered Steps to Restore the Mart within SLA

To restore the mart within the SLA, follow these ordered steps:

1. Investigate and Resolve `intermediate.int\_order\_enriched` Failure:
 - Review the logs and metrics related to the `DBT\_RUN\_TIMEOUT` error.
 - Identify the cause of the timeout (e.g., resource constraints, query optimization issues).
 - Apply fixes, which may include optimizing queries, increasing resource allocation, or adjusting timeouts.
2. Rerun `intermediate.int\_order\_enriched` Model:
 - Once the issue is resolved, rerun the `intermediate.int\_order\_enriched` model to ensure it completes successfully.
 - Monitor the run closely to catch any recurring issues promptly.
3. Trigger Refresh of `analytics.fct\_orders` Mart:
 - After the successful completion of `intermediate.int\_order\_enriched`, manually trigger a refresh of the `analytics.fct\_orders` mart if it doesn't automatically rerun due to dependencies.
 - Ensure that all upstream dependencies have successfully completed before expecting `fct\_orders` to refresh.
4. Verify Downstream Consumers:
 - After `fct\_orders` has been successfully refreshed, verify that downstream consumers (`Looker Revenue Overview` and `exec\_daily\_orders` Slack bot) are receiving and processing the updated data correctly.
 - Perform a sanity check on the data visible to end-users to ensure it reflects the expected freshness and accuracy.
5. Review and Implement Preventative Measures:
 - Conduct a post-mortem analysis to identify how such breaches can be prevented in the future.
 - Consider implementing additional monitoring, automated retries for transient failures, or adjustments to model timeouts and resource allocations as needed.

By following these steps, the freshness SLA for the `analytics.fct\_orders` mart should be restored, ensuring that downstream consumers receive accurate and timely data.

Engineering counterpart

6:54 AM - Responded in 0s

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

Agent response

6:54 AM - Responded in 45s

Let's break down the analysis and resolution plan step by step, outlining the reasoning and assumptions made at each stage.

1. Identifying the Breach and Its Cause

- Given Information: The `analytics.fct\_orders` mart has breached its freshness SLA with a staleness of 388 minutes.
- Analysis: The breach indicates that the mart has not been updated within the expected timeframe, which could lead to downstream consumers using outdated data.
- Assumption: The staleness is due to a failure or delay in one of the upstream models or the mart itself.

2. Determining the Upstream Model Blocking Refresh

- Given Information: The `intermediate.int\_order\_enriched` model failed with a `DBT\_RUN\_TIMEOUT` error after 45 minutes, and this model is an upstream dependency of `fct\_orders`.
- Analysis: The failure of `intermediate.int\_order\_enriched` is the most direct cause of the breach since it's an upstream dependency and its failure would prevent `fct\_orders` from refreshing.
- Assumption: The failure of `intermediate.int\_order\_enriched` is the primary blocker for the refresh of `fct\_orders`, and resolving this issue will allow `fct\_orders` to update.

3. Assessing the Blast Radius

- Given Information: Downstream consumers include `Looker Revenue Overview` and `exec\_daily\_orders` Slack bot.
- Analysis: These consumers rely on the freshness of data from `fct\_orders`. Stale data could lead to incorrect insights or actions based on outdated information.
- Assumption: The freshness of `fct\_orders` data is critical for the accuracy and reliability of these downstream applications.

4. Planning the Resolution

- Step 1: Investigate and Resolve `intermediate.int\_order\_enriched` Failure
 - Reasoning: Since `intermediate.int\_order\_enriched` is the blocker, resolving its failure is the first priority.
 - Assumption: The cause of the failure can be identified and fixed within a reasonable timeframe.
- Step 2: Rerun `intermediate.int\_order\_enriched` Model
 - Reasoning: After fixing the issue, rerunning the model ensures that it completes successfully and provides the necessary data for `fct\_orders`.
 - Assumption: The fix will be effective, and the model will run to completion without further issues.
- Step 3: Trigger Refresh of `analytics.fct\_orders` Mart
 - Reasoning: With upstream dependencies resolved, triggering a refresh of `fct\_orders` ensures it updates with the latest data.
 - Assumption: The mart's refresh mechanism is dependent on the successful completion of its upstream models.
- Step 4: Verify Downstream Consumers
 - Reasoning: Ensuring downstream applications are working with the updated data confirms the resolution of the breach.
 - Assumption: These applications automatically update with new data from `fct\_orders` or can be easily verified for correctness.
- Step 5: Review and Implement Preventative Measures
 - Reasoning: To prevent future breaches, understanding the root cause and implementing preventative measures is essential.
 - Assumption: There are potential improvements or adjustments (e.g., in monitoring, resource allocation, or retry mechanisms) that can be made to prevent similar breaches.

Key Assumptions

1. Technical Assumptions: The architecture and dependencies between models and marts are as described, and resolving the `intermediate.int\_order\_enriched` issue will allow `fct\_orders` to refresh.
2. Operational Assumptions: The team has the capability to investigate, fix, and rerun models within a reasonable timeframe, and downstream applications update automatically or can be easily verified.
3. Data Ass

Engineering counterpart
6:54 AM - Responded in 0s

What would you verify before shipping this to production?

Agent response

6:54 AM - Responded in 3s

Before shipping the resolution to production, several aspects should be verified to ensure the fix is effective, stable, and does not introduce new issues. Here's a checklist of items to verify:

1. Functional Verification

- Upstream Model Success: Verify that the ``intermediate.int_order_enriched`` model runs successfully after the applied fix.
- Mart Refresh: Confirm that the ``analytics.fct_orders`` mart refreshes correctly after the upstream model's successful run.
- Data Accuracy: Check the data in ``fct_orders`` for accuracy and completeness, ensuring it reflects the expected changes and updates.

2. Performance and Timing

- Execution Time: Verify that the execution time of ``intermediate.int_order_enriched`` and the subsequent refresh of ``fct_orders`` are within acceptable limits, ensuring they can meet the SLA requirements.
- Resource Utilization: Monitor resource utilization (e.g., CPU, memory, network) during the model runs to ensure they are within expected bounds and do not negatively impact other processes.

3. Dependency and Downstream Impact

- Downstream Consumers: Validate that downstream applications (``Looker Revenue Overview`` and ``exec_daily_orders`` Slack bot) receive and process the updated data correctly, without errors or data inconsistencies.
- Dependency Chain: Confirm that the fix does not break or negatively affect any other part of the dependency chain, including models or applications not directly related to ``fct_orders``.

4. Error Handling and Recovery

- Retry Mechanisms: If implemented, test retry mechanisms for transient failures to ensure they work as expected.
- Alerting and Monitoring: Verify that appropriate alerting and monitoring are in place to quickly detect any future issues with ``intermediate.int_order_enriched`` or ``fct_orders``, allowing for prompt action.

5. Security and Access Control

- Permissions: Ensure that the fix does not alter or bypass existing security measures, such as data access controls or encryption.
- Compliance: Verify that all changes comply with relevant data governance policies and regulatory requirements.

6. Documentation and Knowledge Sharing

- Update Documentation: Ensure that all relevant documentation (e.g., technical guides, runbooks) is updated to reflect the changes made.
- Knowledge Sharing: Share knowledge about the issue, its resolution, and any lessons learned with the team to improve future troubleshooting and prevent similar issues.

7. Testing in Staging Environment

Before deploying to production, it's crucial to test the fixes in a staging environment that closely mimics production. This step helps identify and address any environment-specific issues without risking production stability.

8. Rollback Plan

Have a clear rollback plan in place in case the deployment to production introduces unforeseen issues. This plan should outline the steps to quickly revert changes and restore the previous state until the issues can be addressed.

By meticulously verifying these aspects, you can ensure that the resolution is thoroughly tested, validated, and ready for production, minimizing the risk of introducing new problems or failing to meet service level agreements.

Engineering counterpart
6:54 AM - Responded in 0s

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

Agent response

6:55 AM - Responded in 34s

Top Risks

1. Data Inconsistency or Loss: Changes to the `intermediate.int\_order\_enriched` model or the `analytics.fct\_orders` mart could lead to data inconsistencies or loss, affecting downstream applications and business decisions.
2. Dependency Chain Disruption: Altering the upstream model could have unforeseen effects on other dependent models or applications, potentially causing a cascade of failures or delays.
3. Performance Degradation: The fixes applied could inadvertently degrade the performance of the `intermediate.int\_order\_enriched` model or the `analytics.fct\_orders` mart, leading to increased latency or resource utilization that might not meet SLA requirements.
4. Security or Compliance Issues: Changes might introduce security vulnerabilities or compliance issues, especially if they involve altering data access controls, encryption, or other security measures.
5. Monitoring and Alerting Gaps: The resolution might overlook existing monitoring and alerting configurations, leading to undetected issues or delayed responses to future problems.

Rollback Plan

In case the deployed changes do not work as expected, a rollback plan should be executed promptly to minimize disruption. Here's a step-by-step rollback strategy:

Preparation

- Backup Current State: Before applying any changes, ensure that the current state of the `intermediate.int\_order\_enriched` model, the `analytics.fct\_orders` mart, and all relevant dependencies are backed up.
- Identify Rollback Triggers: Clearly define the conditions under which a rollback will be initiated, such as failure to meet SLA, data inconsistencies, or performance degradation.

Rollback Steps

1. Immediate Reversion of Changes: Revert all changes made to the `intermediate.int\_order\_enriched` model and the `analytics.fct\_orders` mart to their pre-change state.
2. Restore from Backups: If necessary, restore the models and mart from the backups taken before applying the changes.
3. Revert Dependency Changes: If any changes were made to dependencies (e.g., updating other models or applications), revert those changes as well.
4. Validate State: After rollback, validate the state of the `intermediate.int\_order\_enriched` model, the `analytics.fct\_orders` mart, and dependent applications to ensure they return to their expected functionality and performance.
5. Notify Stakeholders: Inform all relevant stakeholders about the rollback, including the reasons for it and any next steps or plans for reattempting the fix.

Post-Rollback Analysis

- Root Cause Analysis: Conduct a thorough analysis to identify the root cause of the failure.
- Lessons Learned: Document the lessons learned from the attempt, including what went wrong and how it can be improved in the future.
- Revised Approach: Based on the analysis, devise a revised approach to addressing the original issue, incorporating the insights gained from the failed attempt.

Communication Plan

- Stakeholder Notification: Keep stakeholders informed throughout the process, including the decision to rollback, the progress of the rollback, and the outcome.
- Transparent Reporting: Provide transparent reporting on what happened, why the rollback was necessary, and the plans for moving forward.

By having a well-planned rollback strategy in place, the potential risks associated with deploying changes can be mitigated, ensuring that the system can be quickly restored to a stable state if issues arise.

Engineering counterpart

6:55 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.