



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: anthropic/claude-haiku-4.5

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

Duration: 33s

Actual run cost: \$0.0130

Projected execution cost: \$14.30/mo (50/biz day × 22 days)

Report date: July 12, 2026 - 11:08 PM 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: 11026 chars - Latency: 33s - 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: 33s

Engineering counterpart

3:07 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

3:08 AM - Responded in 7s

Freshness SLA Breach Triage & Recovery Plan

? Root Cause Analysis

Blocking Model: `intermediate.int\_order\_enriched`

- Status: FAILED (since 09:12 UTC)
- Error: DBT_RUN_TIMEOUT after 45 minutes
- Impact: Prevents `fct\_orders` refresh (388 min stale vs. 120 min SLA)

Why it's blocking: `int\_order\_enriched` is a critical dependency in the DAG. Until it succeeds, downstream mart `fct\_orders` cannot refresh.

? Blast Radius

| Consumer | Severity | Impact |

|-----|-----|-----|

| Looker Revenue Overview | ? Critical | Execs viewing stale revenue metrics (6+ hrs old) |
| exec_daily_orders Slack bot | ? Critical | Automated daily digest showing yesterday's data |

Estimated business impact: Revenue reporting lag, potential decision-making delays.

? Ordered Recovery Steps (SLA Restore Target: <30 min)

Phase 1: Immediate Diagnostics (5 min)

1. SSH into orchestrator / check logs:

```
```sql
-- Query recent DBT run logs
SELECT run_id, model, status, runtime_seconds, error_message
FROM metadata.dbt_run_history
WHERE model = 'intermediate.int_order_enriched'
ORDER BY started_at DESC LIMIT 5;
```
```

2. Identify timeout root cause:

- Is `int\_order\_enriched` performing expensive joins/aggregations?
- Has upstream data volume spiked (check `stg\_orders` row count)?
- Are there missing table indexes?

Phase 2: Quick Win Attempt (10 min)

Option A - Increase timeout & retry:

```
```yaml
```

# dbt\_project.yml for int\_order\_enriched

configs:

```
 timeout_seconds: 1800 # Increase from ~2700s (45m) to 30m
```

```
Re-trigger immediately
dbt run --models intermediate.int_order_enriched --threads 8

```

Option B - Parallel execution optimization:

```
```sql
-- If resource contention: increase warehouse size
ALTER WAREHOUSE analytics_compute SET WAREHOUSE_SIZE = 'XLARGE';
---
```

Phase 3: If Phase 2 Fails -> Escalate (15 min)

3. Investigate query performance:

```
```sql
-- Analyze int_order_enriched compilation
EXPLAIN intermediate.int_order_enriched;

-- Check for missing indexes
SELECT FROM information_schema.statistics
WHERE table_schema = 'intermediate' AND table_name LIKE '%order%';

```

4. Temporary materialization downgrade:

```
```yaml
# dbt_project.yml - reduce to table (vs. incremental/view) if applicable
models:
  intermediate:
    int_order_enriched:
      materialized: table # Simplify from complex incremental logic
---
```

5. Full-refresh trigger:

```
```bash
dbt run --models intermediate.int_order_enriched --full-refresh --threads
```

**Engineering counterpart**  
**3:08 AM - Responded in 0s**

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

## Agent response

3:08 AM - Responded in 9s

### # Reasoning Walkthrough & Assumptions

#### ## Step 1: Root Cause Identification

What I did:

I isolated `intermediate.int\_order\_enriched` as the blocker by:

- Cross-referencing the DAG: `fct\_orders` (the mart) depends on `int\_order\_enriched`
- Checking upstream status: `stg\_orders` and `stg\_customers` both succeeded recently (14:58-15:05 UTC)
- Noting the critical signal: `int\_order\_enriched` failed at 09:12 UTC and hasn't retried

Key assumption:

- I assumed a linear DAG dependency: `stg\_` -> `int\_order\_enriched` -> `fct\_orders`
- Reality check: This is fragile. If there's a parallel path (e.g., `int\_order\_enriched` and `int\_customer\_metrics` both feed `fct\_orders`), and only one failed, the mart might still refresh from the other. I should have asked: Is `int\_order\_enriched` the ONLY upstream input to `fct\_orders`?

---

#### ## Step 2: Blast Radius Assessment

What I did:

I listed downstream consumers and labeled them "Critical" based on:

- Looker dashboard = user-facing, executive visibility
- Slack bot = automated dependency, stakeholders expect daily updates

Assumptions I made:

1. Both consumers are actively monitored -> Someone is already noticing the stale data
  - Reality: Maybe the Slack bot runs at 8 AM and no one checks it until 10 AM. The actual pain might be lower.
2. Revenue dashboard = high priority -> Implicit business assumption
  - Reality: I don't know if these dashboards are viewed by C-suite or just analysts. SLA tier says "P1 revenue dashboard," so I borrowed that signal.
3. No fallback/caching -> Consumers have no stale data serving mechanism
  - Reality: Looker might cache the last 6-hour query result. Slack bot might have a manual override. I should have asked.

---

#### ## Step 3: Recovery Steps Ordering

What I did:

I ordered steps by speed of iteration and likelihood of success:

...

Diagnostics (5 min)

? (gather info, don't wait for deep analysis)

Quick wins (10 min)

? (cheap retry + resource bump)

Escalation (15 min)

? (root cause investigation + config changes)

...

Assumptions embedded here:

### Assumption A: Timeout is the most likely cause

- Evidence: Error message explicitly says `DBT\_RUN\_TIMEOUT after 45m`
- Unstated assumption: The timeout didn't change, so something about the query or data got slower
- Missing check: Did a code deploy happen? Did upstream data volume spike? Did a table lose an index?

### Assumption B: Increasing timeout will work

```
``yaml
```

```
timeout_seconds: 1800 # 30 min
```

```
``
```

- I assumed: The query can complete in 30 min with more time
- Reality check: What if it's an infinite loop or a full cartesian join? More time doesn't help.
- Better first step: Ask-did this model ever succeed? If yes,

**Engineering counterpart**  
**3:08 AM - Responded in 0s**

What would you verify before shipping this to production?

## Agent response

3:08 AM - Responded in 8s

# Pre-Production Verification Checklist

## ? Critical Unknowns (Block shipping without answers)

### 1. Confirm the actual blocking dependency

```
```sql
```

```
-- Query the orchestrator DAG, not assumptions
```

```
SELECT
```

```
  downstream_model,
```

```
  upstream_model,
```

```
  dependency_type
```

```
FROM metadata.model_lineage
```

```
WHERE downstream_model = 'analytics.fct_orders';
```

```
```
```

Why: My recovery plan assumes `int\_order\_enriched` is required for `fct\_orders`. If it's actually optional (e.g., a left join that can be null), the mart might refresh without it. This changes everything.

Verification:

- [ ] Confirm `int\_order\_enriched` is in the critical path (not a parallel optional enrichment)

- [ ] Check if `fct\_orders` has retry logic or can skip failed dependencies

---

### 2. Understand why the timeout happened NOW

```
```sql
```

```
-- Compare runtime history
```

```
SELECT
```

```
  run_date,
```

```
  model,
```

```
  runtime_seconds,
```

```
  row_count_input,
```

```
  row_count_output
```

```
FROM metadata.dbt_run_history
```

```
WHERE model = 'intermediate.int_order_enriched'
```

```
ORDER BY run_date DESC LIMIT 30;
```

```
```
```

Why: If I just increase the timeout and re-run without understanding the spike, I'm treating a symptom, not the disease.

Verification:

- [ ] Did runtime spike today or was it always slow? (If always slow, why is it failing now?)

- [ ] Did input data volume spike? (Check `stg\_orders` row count vs. last 7 days)

- [ ] Was there a code deploy in the last 6 hours?

- [ ] Are there competing warehouse queries consuming resources?

---

### 3. Test the recovery steps in a safe environment first

```
```bash
```

```
# DON'T run this on production immediately
```

```
# Instead:
```

```
# Step A: Dry-run in dev/staging
dbt run --models intermediate.int_order_enriched \
  --profiles-dir profiles_staging \
  --target staging \
  --threads 4

# Step B: Check compiled SQL for obvious issues
dbt parse --models intermediate.int_order_enriched
cat target/compiled/project/models/intermediate/int_order_enriched.sql
---
```

Why: The timeout might be due to a query that's fundamentally broken (not just slow).

Verification:

- [] Does the model compile without errors?
- [] Are there obvious N² joins or missing WHERE clauses?
- [] Does it run successfully in dev with smaller data?

? Medium Priority (Verify before executing Phase 2)

4. Check if warehouse/compute is the constraint

```
```sql
-- Query Snowflake query history
SELECT
 query_id,
 query_text,
 execution_time,
 compilation_time,
 warehouse_name,
 warehouse_size
FROM snowflake.account_usage.query_history
WHERE query_text LIKE '%int_order_enriched%'
 AND start_time > CURRENT_TIMESTAMP - INTERVAL '6 hours'
ORDER BY start_time DESC;
```
```

Why: If the warehouse is at 100% CPU, just retrying will fail again.

Verification:

- [] Is

Engineering counterpart
3:08 AM - Responded in 0s

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

Agent response

3:08 AM - Responded in 8s

Top Risks & Rollback Strategy

? Top 5 Risks (Ranked by Likelihood × Impact)

Risk 1: Timeout increase masks a deeper issue (HIGH LIKELIHOOD, HIGH IMPACT)

Scenario:

...

Action: Increase timeout to 30 min, retry

Result: Query still times out OR completes but produces wrong data

Outcome: fct_orders refreshes with corrupted/incomplete rows

Downstream: Revenue dashboard shows artificially inflated/deflated metrics

...

Why this is likely:

- We haven't diagnosed why it's slow (data spike? query plan? regression?)
- Just adding time is a band-aid; the root cause persists
- If it's a query logic bug (e.g., unintended full outer join), more time makes it worse

Detection trigger:

```sql

-- Post-refresh sanity check

SELECT COUNT(), SUM(order\_amount)

FROM analytics.fct\_orders

WHERE date\_trunc('day', order\_date) = CURRENT\_DATE;

-- Compare to yesterday's daily total

-- If off by >10%, ABORT

...

---

### Risk 2: Resource contention cascades (MEDIUM LIKELIHOOD, HIGH IMPACT)

Scenario:

...

Action: Bump warehouse to XLARGE, trigger refresh

Result: Other dependent jobs starve (downstream models timeout)

Outcome: SLA breach spreads to dependent marts (fct\_customers, fct\_products)

...

Why this is likely:

- We don't know if there are other concurrent jobs at 15:40 UTC
- Scaling up one job can starve others if they share the same warehouse

Detection trigger:

```sql

-- Check warehouse queue depth before/after

SELECT

warehouse_name,

queued_queries,

executing_queries

FROM snowflake.account_usage.warehouse_events

WHERE start_time > CURRENT_TIMESTAMP - INTERVAL '10 minutes';

Risk 3: Rollback leaves stale data permanently (MEDIUM LIKELIHOOD, MEDIUM IMPACT)

Scenario:

Action: Rollback to pre-failure state, halt refresh

Result: fct_orders stays at 09:12 UTC forever (or until manual intervention)

Outcome: Stakeholders lose trust in data platform; manual workarounds proliferate

Why this is likely:

- If we can't fix `int\_order\_enriched` quickly, we might decide to "stop the bleeding"
- But we have no automated fallback to serve stale-but-consistent data
- Manual recovery becomes a Slack thread at 3 AM

Risk 4: Partial refresh creates downstream inconsistency (MEDIUM LIKELIHOOD, MEDIUM IMPACT)

Scenario:

Action: Temporarily skip int_order_enriched, refresh fct_orders from stg_only

Result: fct_orders refreshes with missing enrichment columns (e.g., customer_segment)

Outcome: Looker dashboard shows NULLs; Slack bot reports incomplete metrics

Why this is likely:

- Temptation to "just bypass the dependency" under time pressure
- Looks like a quick win but violates data contract

Risk 5: Recovery blocks subsequent scheduled runs (LOW LIKELIHOOD, HIGH IMPACT)

Scenario:

Action: Manual retry at 15:55 UTC

Result

Engineering counterpart**3:08 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.