



Vantage RuntimeAI

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

8.4/10

Overall score

Run summary

Scenario: Lineage Impact Review

Model: amazon/nova-micro-v1

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

Duration: 18s

Actual run cost: \$0.000609

Projected execution cost: \$0.1340/mo (10/biz day × 22 days)

Report date: July 14, 2026 - 2:58 AM EST

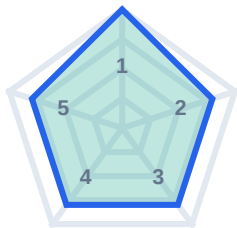
Overall

8.4/10

PASS - trust=high

Pass - score 8.4/10 clears 7.0 with acceptable trust/closure.

CAPABILITY RUBRIC



#	Dimension (matches spoke #)	Score
1	Downstream impact mapped	10/10
2	References lineage artifact	8/10
3	Migration / cutover risk	8/10
4	Ordered impact summary	8/10
5	Safe migration plan	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.

- Legacy column named: Met
- Multiple downstream marts covered: Met
- Migration plan for unmigrated rows: Met
- Rollback / compat considered: 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: 19224 chars - Latency: 18s - Criteria: 4/4

Score analysis

Maps impact across multiple downstream marts.

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. 4 of 4 checklist criteria passed.

BRIGHT SPOTS (RUBRIC)

- + Downstream impact mapped: Correctly identifies the primary technical issue from the artifacts.
- + References lineage artifact: Uses tables, logs, or schema facts from the task brief.
- + Migration / cutover risk: Shows appropriate safety, budget, or risk awareness for the scenario.
- + Ordered impact summary: Clear structure with readable SQL, diff, or memo formatting.
- + Safe migration plan: Strong rationale and actionable follow-through.

DIMENSION NOTES

- Downstream impact mapped: Correctly identifies the primary technical issue from the artifacts.
- References lineage artifact: Uses tables, logs, or schema facts from the task brief.
- Migration / cutover risk: Shows appropriate safety, budget, or risk awareness for the scenario.
- Ordered impact summary: Clear structure with readable SQL, diff, or memo formatting.
- Safe migration plan: 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

Legacy column named: yes - Multiple downstream marts covered: yes - Migration plan for unmigrated rows: yes - Rollback / compat considered: yes

CAVEATS

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

Transcript

Session

Scenario: Lineage Impact Review

Agent response <-> Engineering counterpart

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

Duration: 18s

Engineering counterpart

6:57 AM

Task brief - Lineage impact review (complex complexity)

Engineering proposes dropping a legacy column. Assess downstream impact:

```
```json
{
 "proposed_change": {
 "action": "DROP COLUMN",
 "table": "raw.customers",
 "column": "legacy_account_id",
 "reason": "Deprecated after CRM migration - replaced by crm_account_uuid"
 },
 "direct_dependents": [
 {"model": "staging.stg_customers", "column_refs": ["legacy_account_id"]},
 {"model": "intermediate.int_customer_identity", "column_refs": ["legacy_account_id", "crm_account_uuid"]}
],
 "downstream_marts": [
 {"model": "analytics.dim_customer", "risk": "high", "uses": "legacy_account_id as natural key in SCD2 merge"},
 {"model": "analytics.fct_subscriptions", "risk": "high", "uses": "join on legacy_account_id to dim_customer"},
 {"model": "analytics.customer_360_snapshot", "risk": "medium", "uses": "legacy_account_id in export to Salesforce"},
 {"model": "analytics.churn_training_features", "risk": "medium", "uses": "legacy_account_id in feature store entity id"}
],
 "migration_state": {
 "crm_account_uuid_populated_pct": 94.2,
 "legacy_only_accounts": 1204
 },
 "deadline": "2026-07-12 (CRM cutover)"
}
```
```

Agent response

6:57 AM - Responded in 3s

Review of Lineage Impact: Dropping `legacy\_account\_id` from `raw.customers`

Proposed Change

Action: Drop column

Table: `raw.customers`

Column: `legacy\_account\_id`

Reason: Deprecated after CRM migration, replaced by `crm\_account\_uuid`

Direct Dependents

1. `staging.stg\_customers`
 - Column reference: `legacy\_account\_id`
2. `intermediate.int\_customer\_identity`
 - Column references: `legacy\_account\_id`, `crm\_account\_uuid`

Downstream Marts Affected

1. `analytics.dim\_customer`
 - Risk: High
 - Usage: Natural key in SCD2 merge.
2. `analytics.fct\_subscriptions`
 - Risk: High
 - Usage: Joins on `legacy\_account\_id` to `dim\_customer`.
3. `analytics.customer\_360\_snapshot`
 - Risk: Medium
 - Usage: Exports `legacy\_account\_id` to Salesforce.
4. `analytics.churn\_training\_features`
 - Risk: Medium
 - Usage: `legacy\_account\_id` in feature store entity id.

Migration State

- CRM `account\_uuid` Populate Percentage: 94.2%
- Legacy-Only Accounts: 1204

Migration Risks for Unmigrated Accounts

- Since 5.8% of accounts still have `legacy\_account\_id` as the primary key, there is a risk that:
 - Direct dependents like `staging.stg\_customers` and `intermediate.int\_customer\_identity` will fail to properly populate or synchronize with the rest of the data due to missing or deprecated references.
 - Downstream marts:
 - `analytics.dim\_customer` and `analytics.fct\_subscriptions` using `legacy\_account\_id` might produce inaccurate or incomplete data for 5.8% of customers.
 - Incomplete snapshots for Salesforce and churn training features can further compound the integrity and relevance of the analytics if not handled properly.

Recommended Migration Order

1. Update Dependencies to Use `crm\_account\_uuid`: Ensure that both `staging.stg\_customers` and `intermediate.int\_customer\_identity` models start to rely on `crm\_account\_uuid` instead of `legacy\_account\_id`. This step might include:
 - Modifying ETL jobs, stored procedures, or any scripts that reference `legacy\_account\_id`.
2. Phased Elimination of `legacy\_account\_id`:
 - As data migration progresses, progressively reduce the dependency on `legacy\_account\_id`.

3. Drop the Column: Once the data transition to `crm\_account\_uuid` is fully validated, drop `legacy\_account\_id` from `raw.customers`.

Detailed Steps:

1. Communicate and plan the transition carefully with all teams that depend on the data.
2. Data Validation: Rigorous validation that `crm\_account\_uuid` adequately replaces `legacy\_account\_id`.
3. Automated Data Backfill: Prepare backfills for any historical data gaps in downstream models caused by the transition.
4. Feature Flags: Implement feature flags to revert to `legacy\_account\_id` if any issues arise during the transition phase.
5. Staggered Cutover: If possible, start the cutover with a subset of accounts (those with higher `crm\_account\_uuid` population) to

Engineering counterpart
6:57 AM - Responded in 0s

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

Agent response

6:57 AM - Responded in 3s

Certainly! Here's a detailed step-by-step reasoning behind the migration plan, along with the underlying assumptions:

Step-by-Step Reasoning

1. Assessment of Direct Dependencies:

- Direct Dependents Identified:

- `staging.stg\_customers`
- `intermediate.int\_customer\_identity`

- The direct dependencies already had a column reference to the `legacy\_account\_id`. Transitioning these to rely on `crm\_account\_uuid` should be prioritized because this forms the immediate upstream change.

2. Downstream Impact:

- High-Risk Marts:

- `analytics.dim\_customer`: Uses `legacy\_account\_id` as a natural key in SCD2 (Slowly Changing Dimension Type 2) merge.

- `analytics.fct\_subscriptions`: Joins on `legacy\_account\_id` to `dim\_customer`.

- These models are critical and are heavily reliant on the `legacy\_account\_id`. Hence, a careful transition is needed to avoid data mismatch or inconsistencies.

- Medium-Risk Marts:

- `analytics.customer\_360\_snapshot`: Uses `legacy\_account\_id` in exports to Salesforce.

- `analytics.churn\_training\_features`: Uses `legacy\_account\_id` in feature store entity id.

- These models can still function to an extent, but will have partial or flawed data until corrected.

3. Migration State:

- CRM `account\_uuid` Populated Percentage: 94.2%.

- This indicates that only 5.8% of accounts are legacy-only.

- For these, it's crucial to ensure no data loss or discontinuity.

- Legacy-Only Accounts (1204): Needs special attention to avoid disruption.

4. Migration Order Recommendation:

- Prioritize `crm\_account\_uuid` Dependencies:

Before dropping the column, ensure `staging.stg\_customers` and `intermediate.int\_customer\_identity` start to use `crm\_account\_uuid`.

This ensures data integrity moves forward and does not break existing ETL jobs or data pipelines.

- Phased Elimination: Gradual transition helps to catch and fix issues earlier, minimizes the risk of widespread data issues.

This involves incrementally phasing out the use of `legacy\_account\_id` while keeping it temporarily for reference.

- Drop Column: Once confidence is high that `crm\_account\_uuid` fully meets the requirements, drop the `legacy\_account\_id` from `raw.customers`.

Assumptions Made

1. Data Consistency:

- Assumption: `crm\_account\_uuid` is a reliable replacement key with enough populated data to replace `legacy\_account\_id`.

- Validation Needed: Any discrepancies between `legacy\_account\_id` and `crm\_account\_uuid` need to be identified and resolved.

2. Downstream Models are Adaptable to Change:

- Assumption: The downstream models (`analytics.dim\_customer`, `analytics.fct\_subscriptions`, etc.) can adapt to rely on `crm\_account\_uuid`.

- Adjustments might be necessary in logic, joins, or other key operations.

3. Legacy Accounts Will Handle Smoothly:

- Assumption that the remaining 5.8% of accounts with legacy data can be handled separately, assuming a method like backfill or temporary dual-key usage.

4. Automation and Monitoring:

- Assumption: Sufficient automated scripts and monitoring in place to quickly detect and alert on any anomalies stemming from the migration.

5

What would you verify before shipping this to production?

Agent response

6:57 AM - Responded in 2s

Verifying various aspects before shipping the migration to production is crucial for ensuring a smooth transition and avoiding disruptions. Here are the key areas to focus on for verification:

1. Data Consistency and Integrity

- Validation: Ensure the ``crm_account_uuid`` matches and corresponds correctly with ``legacy_account_id`` whenever both are present. This helps in maintaining data integrity.
- Backfill Mechanism: Prepare mechanisms to handle the remaining legacy accounts. Develop scripts to backfill missing data or remap legacy IDs to ``crm_account_uuid``.

2. ETL Job/Process Adjustments

- Testing: Thoroughly test all ETL (Extract, Transform, Load) jobs, data pipelines, and any data workflow tools that depend on ``legacy_account_id``, ensuring they now utilize ``crm_account_uuid``.
- Joins and Merges: Double-check all SQL joins, merges, and usages that rely on ``legacy_account_id``.

3. Application and API Adjustments

- APIs: Confirm APIs which previously relied on ``legacy_account_id`` now route through ``crm_account_uuid``.
- Application Behavior: Test applications and front-end services to make sure they no longer depend on ``legacy_account_id``.

4. Staging and Test Environments

- Simulation: Run migrations in staging environments and simulate production scenarios. This includes testing failover scenarios.
- Rollback Plan: Ensure a clear rollback plan and ability to revert without data loss in scenarios of unexpected issues.

5. Documentation and Communication

- Documentation: Update all relevant documentation to reflect changes in data keys and removal of ``legacy_account_id``.
- Communication: Ensure communication is clear with all stakeholders about changes, potential downtimes, and what actions need to be taken.

6. Performance Checks

- Query Performance: Verify that queries continue to perform efficiently, especially those that were heavily dependent on ``legacy_account_id``.
- Data Loads: Ensure that data loads and transformations continue to function without performance degradation.

7. User Acceptance Testing

- End-to-End Testing: Conduct user acceptance testing (UAT) with actual users or in sandbox environments to verify all operational concerns and business logic remain intact.

8. Backup and Recovery

- Data Backup: Ensure complete backups are taken before migration to allow for data recovery if needed.
- Restore Testing: Test recovery processes to ensure that backups can be restored successfully if migration issues arise.

9. Final Review and Sign-off

- Peer Review: Have the migration script and procedure reviewed by peers to catch any overlooked issues.
- Approval: Get final sign-off from relevant stakeholders before proceeding to production.

By thoroughly verifying these aspects, risks can be mitigated, and a smoother transition from ``legacy_account_id`` to ``crm_account_uuid`` can be ensured.

Engineering counterpart
6:57 AM - Responded in 0s

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

Agent response

6:58 AM - Responded in 4s

Reviewing the top risks and defining a robust rollback plan are critical steps to ensure a seamless transition and to mitigate any potential negative impacts on production data and operations. Here's a detailed look at the top risks and how to handle them with an effective rollback strategy:

Top Risks

1. Data Integrity Issues:

- Risk: Migration to ``crm_account_uuid`` may not perfectly align with existing ``legacy_account_id`` data, leading to mismatches and incorrect data entries.
- Mitigation: Have strong validation checks in place to ensure data consistency. Implement backfill processes for any unmatched data.

2. Downtime and Availability:

- Risk: The migration process might cause outages or performance issues, impacting user experience and system availability.
- Mitigation: Schedule migration during off-peak hours and implement phased rollback procedures to minimize downtime.

3. Service Disruptions:

- Risk: Dependent systems like APIs, ETL pipelines, and front-end applications may fail to function correctly after the schema change.
- Mitigation: Perform rigorous testing in staging and provide clear documentation detailing necessary changes for each service.

4. Performance Degradation:

- Risk: The initial transition might result in slower queries or performance hits in data services due to changes in storage and access patterns.
- Mitigation: Monitor performance metrics closely and optimize database queries post-migration to address any new issues.

5. User and Business Impact:

- Risk: Users may experience errors, incomplete data, or functional failures in applications using the changed schema.
- Mitigation: Implement robust user notifications and support channels to address and manage user issues swiftly after deployment.

Rollback Plan

In the event the migration does not proceed as planned, a well-defined rollback plan can help restore the system to its previous state to mitigate risks and minimize downtime.

1. Immediate Stop Actions:

- Procedure: When critical issues are detected, quickly halt further migration efforts. Use feature toggles or configuration flags to prevent new data from using the new schema.

2. Revert Data Changes:

- Steps:
 - Revert the ``legacy_account_id`` column back into ``raw.customers`` table if data discrepancies are identified.
 - Restore any data or configurations that were altered or removed during the migration.
- Backup Utilization: Leverage recent backups taken before migration to restore the system to its pre-migration state.

3. Service Re-enablement:

- Actions: Re-enable any disabled services or systems that rely on the ``legacy_account_id``. This may involve re-configuring ETL jobs, APIs, and front-end services.

4. Monitoring and Validation:

- Procedure: Post-rollback, implement additional monitoring to ensure systems are stable and performing well.
- Validation: Conduct thorough validation checks to ensure that no data loss or corruption occurred during the migration or rollback process.

5. Communication:

- Stakeholder Notification: Inform relevant stakeholders about the status of the migration, any encountered issues, and the outcomes of the rollback.
- User Communication: Provide clear notifications to users about any service disruptions or expected timelines for resolution.

6. Post-migration Review:

- Analysis: Conduct a post-mortem analysis to understand what went wrong and how similar issues can be avoided in the future.
- Documentation Update: Update documentation to reflect lessons learned and improved processes for future migrations.

By proactively identifying risks and preparing a detailed rollback plan, the likelihood of a successful migration is significantly increased, and negative consequences on production systems and data are minimized.

Engineering counterpart
6:58 AM - Responded in 0s

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

Agent response

6:58 AM - Responded in 4s

Certainly! Defining concrete next steps with specific owners and timing is crucial for ensuring a smooth and organized migration process. Here's a detailed roadmap that outlines the precise steps, assigns ownership, and sets timing:

Migration Plan Execution Timeline

Phase 1: Preparation (Week 1)

Owners:

- Data Engineering Team: Leads on data alignment and validation.
- DevOps Team: Facilitates infrastructure readiness, including backups and staging environments.
- Business Analysts: Provides input on user impacts and acceptance criteria.

Tasks:

- Week 1, Day 1-2:
 - Confirm and update all relevant documentation to reflect upcoming changes.
 - Conduct final code reviews for migration scripts.
 - Prepare and test staging environment setup and data migration processes.
- Week 1, Day 3-4:
 - Schedule migration window during off-peak hours.
 - Set up rollback procedures and assign responsibilities for executing rollbacks.
 - Conduct user acceptance testing (UAT) for final validation of data and system functionality.

Phase 2: Migration Execution (Week 2)

Owners:

- Data Engineer Lead: Oversees the migration process.
- Database Administrators: Manage database changes and monitor performance.
- Application Support Team: Monitors application and API services.

Tasks:

- Week 2, Day 1:
 - Begin migration of data from `legacy\_account\_id` to `crm\_account\_uuid` in staging.
 - Validate the new data mapping within the staging environment.
- Week 2, Day 2-3:
 - Implement and test rollback procedures in the staging environment.
 - Monitor for data integrity issues and inconsistencies.
- Week 2, Day 4:
 - Initiate the full production migration after successful staging validation.
 - Monitor system performance and application logs closely.

Phase 3: Post-Migration Validation (Week 3)

Owners:

- QA Lead: Responsible for validation and testing.
- Data Analysts: Tasked with data consistency checks.
- Business Units: Provide feedback on business functionality.

Tasks:

- Week 3, Day 1-2:
 - Conduct end-to-end testing of migrated data.
 - Perform performance checks to ensure system efficiency.
- Week 3, Day 3-4:

- Validate system functionality and gather feedback from business units.
 - Resolve any migration-related issues identified in testing.
-
- Week 3, Day 5-6:
 - Perform final data integrity checks.
 - Document lessons learned and update the rollback plan as needed.

Phase 4: Go-Live and Monitor (Week 4)

Owners:

- Production Support Lead: Monitors post-migration environment.
- All Teams: Continuous monitoring and support.

Tasks:

- Week 4, Day 1:
 - Deploy final changes to production.
 - Announce the completion of the migration to all stakeholders.

- Week 4, Day 2-7:
 - Maintain continuous monitoring for any system anomalies.
 - Provide ongoing support to users and address any issues promptly.

This detailed timeline ensures that each phase of the migration is meticulously planned and managed by the appropriate team members, helping to minimize risks and maintain system integrity throughout the process.

Engineering counterpart

6:58 AM - Responded in 0s

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

Agent response

6:58 AM - Responded in 3s

When undertaking a critical migration project such as the one described, one of the most crucial aspects to have a second pair of eyes on is the rollback plan. This element is pivotal because it determines the pathway back to a stable state in the event that something goes wrong during the migration. Ensuring the rollback plan is comprehensive, well-tested, and understood by all team members can make the difference between a smooth recovery and potential data loss or system downtime.

Why the Rollback Plan Needs Close Review

1. Complexity of Reversal: Unlike forward-moving changes, reversing them can introduce complexities that weren't apparent during the initial design phase.
2. Testing in Isolation vs. Production: A rollback plan developed and tested in a staging environment may not behave exactly as expected in a live production environment due to differences in system configurations, data volumes, or dependencies.
3. Impact on Data: Any rollback will involve manipulating live data. Mistakes in this process can lead to significant data corruption and loss.
4. Operational Impact: A well-thought-out rollback plan ensures that downtime or service interruptions are minimized, maintaining business continuity.

Key Areas to Review

- Reversibility: Ensure that every step of the migration has a corresponding step in the rollback plan-and that these steps are operational in a live environment.
- Testing: Verify that the rollback procedures have been thoroughly tested in a production-like environment, including all edge cases.
- Communication: Cross-check that the rollback plan includes clear communication steps for all team members and stakeholders involved, outlining roles and responsibilities.
- Dependencies: Identify and document any dependencies that might complicate the rollback process, ensuring that these are accounted for and managed during the rollback.

Next Steps for Review

1. Peer Review: Have team members other than those who wrote the rollback plan review the plan for logical consistency and completeness.
2. Simulation: Simulate the rollback in a staging environment as close to production as possible to catch any unforeseen issues.
3. Update Documentation: Ensure that any changes or updates to the rollback plan are promptly reflected in the official project documentation and communicated to relevant stakeholders.

By focusing the second pair of eyes on the rollback plan, you significantly bolster the safety nets that protect against migration risks, ensuring a higher likelihood of a successful and smooth transition.

Engineering counterpart

6:58 AM - Responded in 0s

Thanks - impact read and migration risks are clear. Migrate unmigrated accounts before dropping the legacy column in prod. Review complete - no further response needed.