Empowering AI Agents with Beam and Dataflow

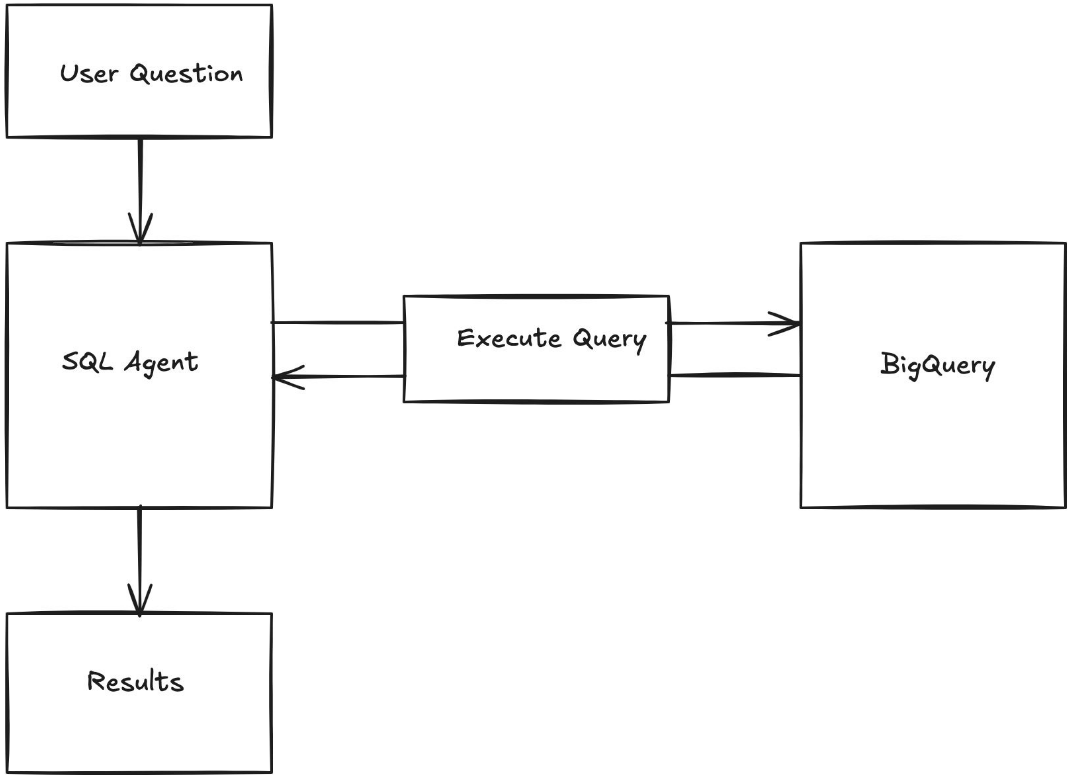
Jasper Van den Bossche
(Software Engineer @ ML6)
Konstantin Buschmeier
(Machine Learning Engineer @ ML6)

What We’re Solving

- Brief description of the problem you're addressing
- Who is affected and why it matters
- What makes your solution different/innovative?

What We’re Solving

What We're Solving



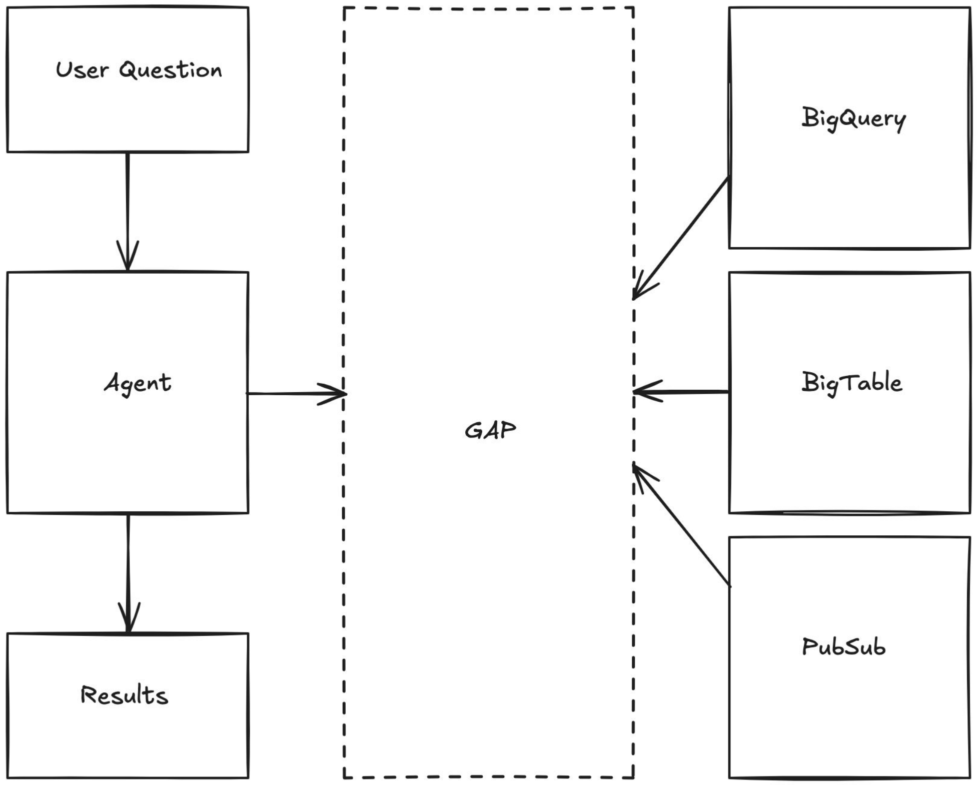
What We're Solving

Complex Multi-Step
Data Transformations

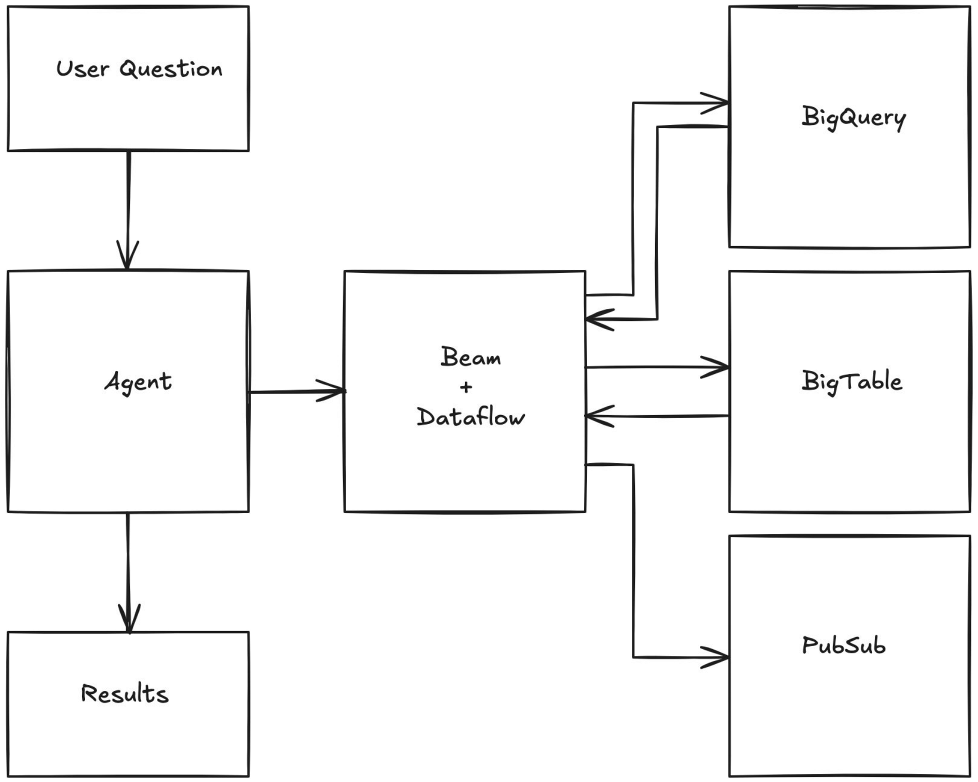
Data Stored
Across Different Sources

Stream Processing

What We’re Solving



What We're Solving

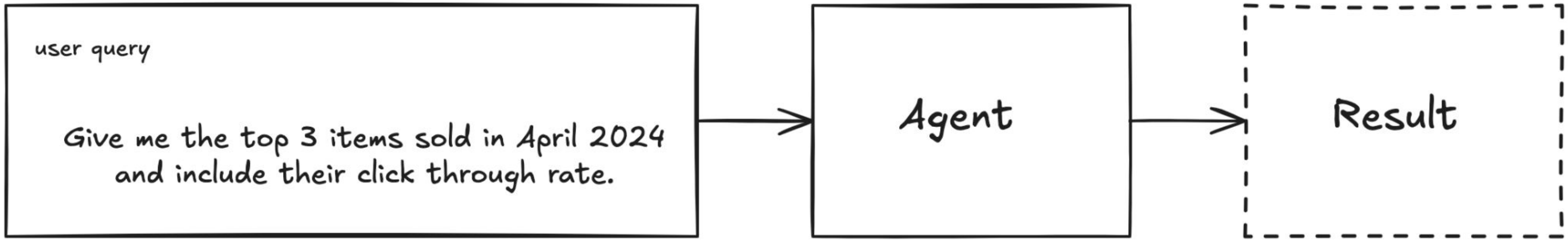


Technical details

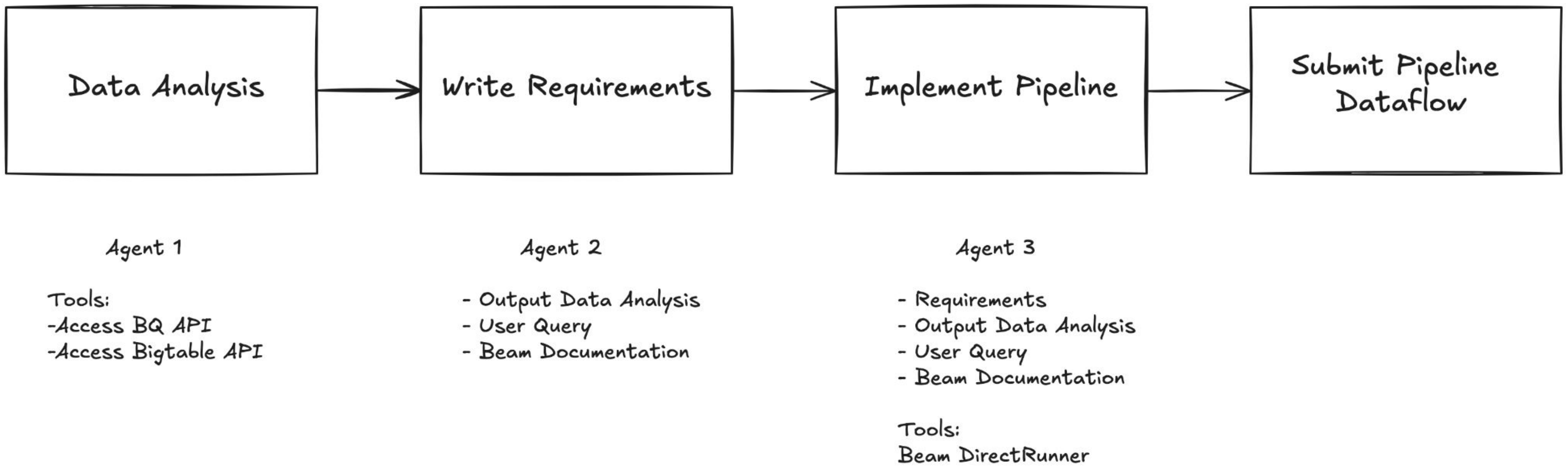
- Tech stack (languages, tools, APIs, platforms, etc.)
- Key challenges you faced (and how you solved/worked around them)
- Screenshots or diagrams of your system/app/notebook

Technical Details

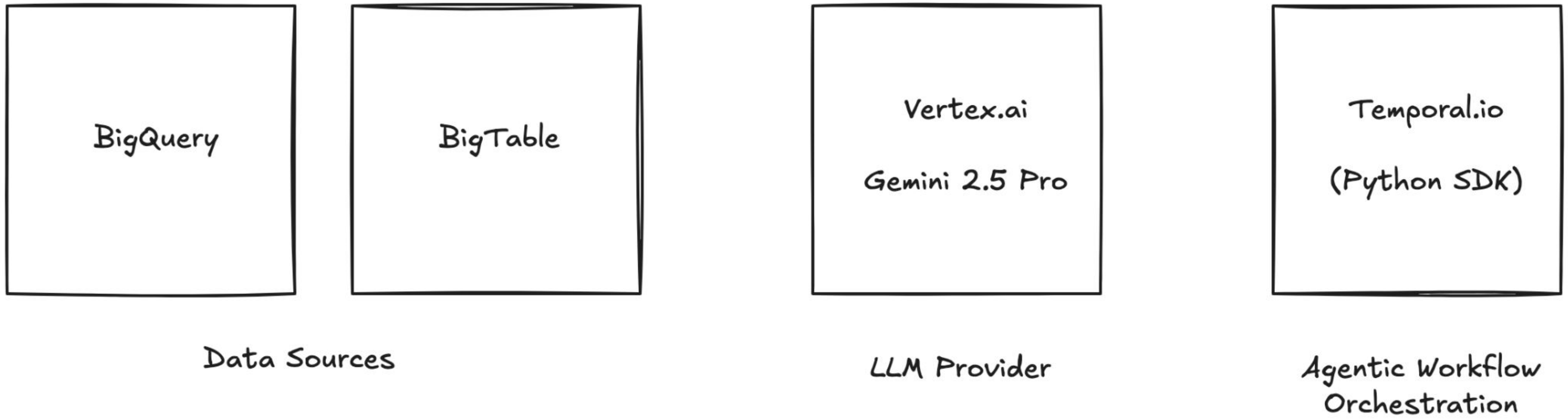
Technical Details


user query
Give me the top 3 items sold in April 2024 and include their click through rate.
Agent
Result

Technical Details

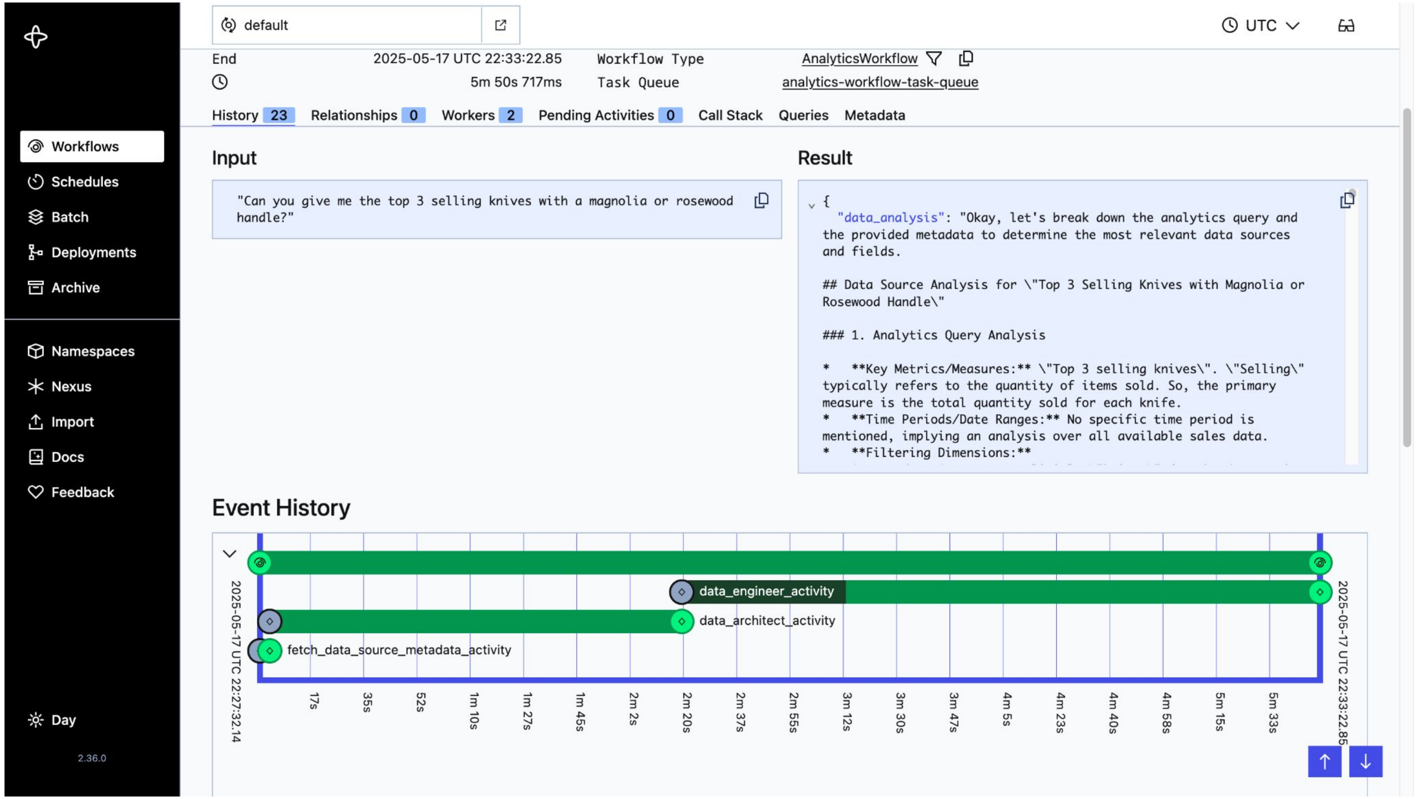


Technical Details



Demo

Start the Workflow with an Analytics Question



End 2025-05-17 UTC 22:33:22.85
5m 50s 717ms
Workflow Type AnalyticsWorkflow
Task Queue analytics-workflow-task-queue

History 23 | Relationships 0 | Workers 2 | Pending Activities 0 | Call Stack | Queries | Metadata

Input

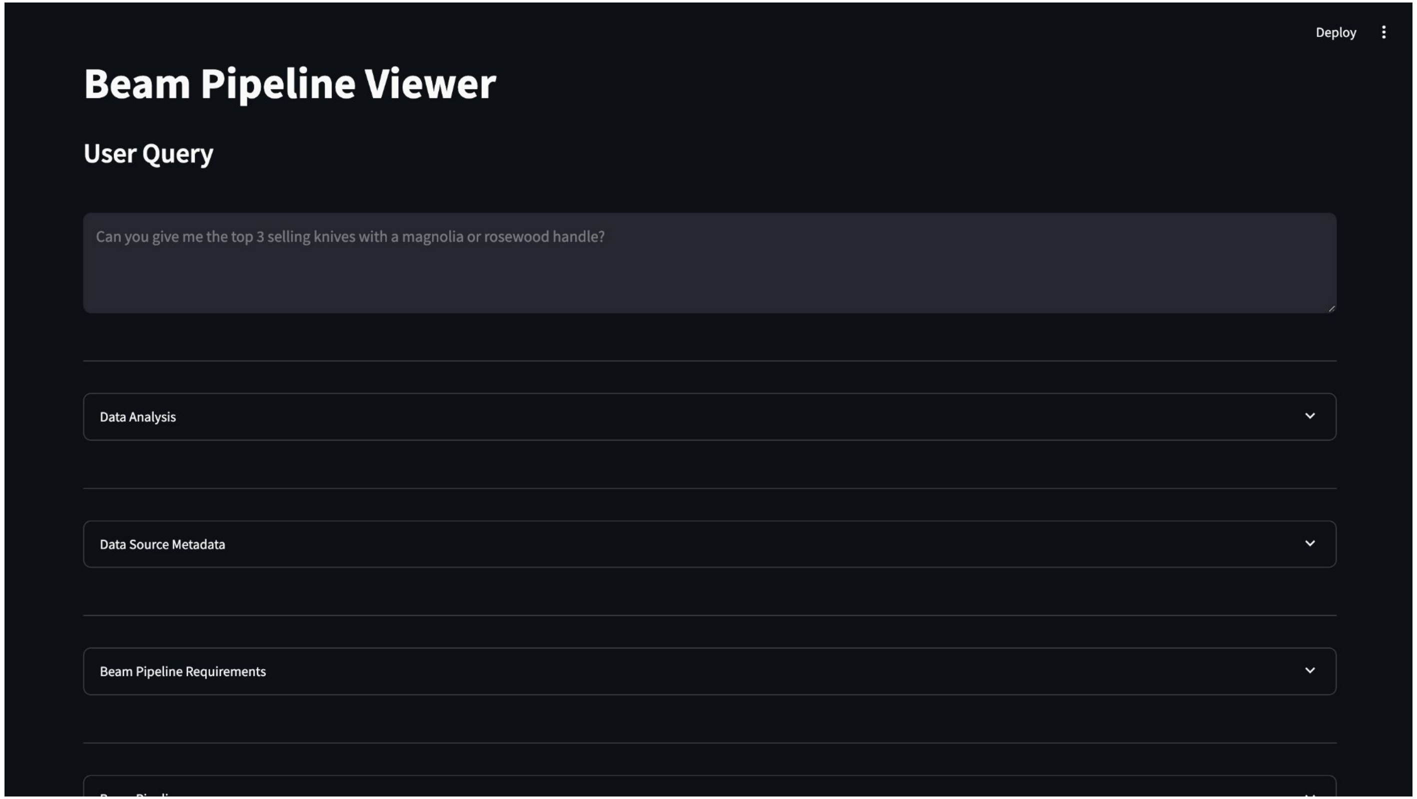
```
"Can you give me the top 3 selling knives with a magnolia or rosewood
handle?"
```

Result

```
{
  "data_analysis": "Okay, let's break down the analytics query and
the provided metadata to determine the most relevant data sources
and fields.

## Data Source Analysis for \"Top 3 Selling Knives with Magnolia or
Rosewood Handle\"

### 1. Analytics Query Analysis

*   **Key Metrics/Measures:** \"Top 3 selling knives\". \"Selling\"
typically refers to the quantity of items sold. So, the primary
measure is the total quantity sold for each knife.
*   **Time Periods/Date Ranges:** No specific time period is
mentioned, implying an analysis over all available sales data.
*   **Filtering Dimensions:**
```

Event History

Inspect the Results

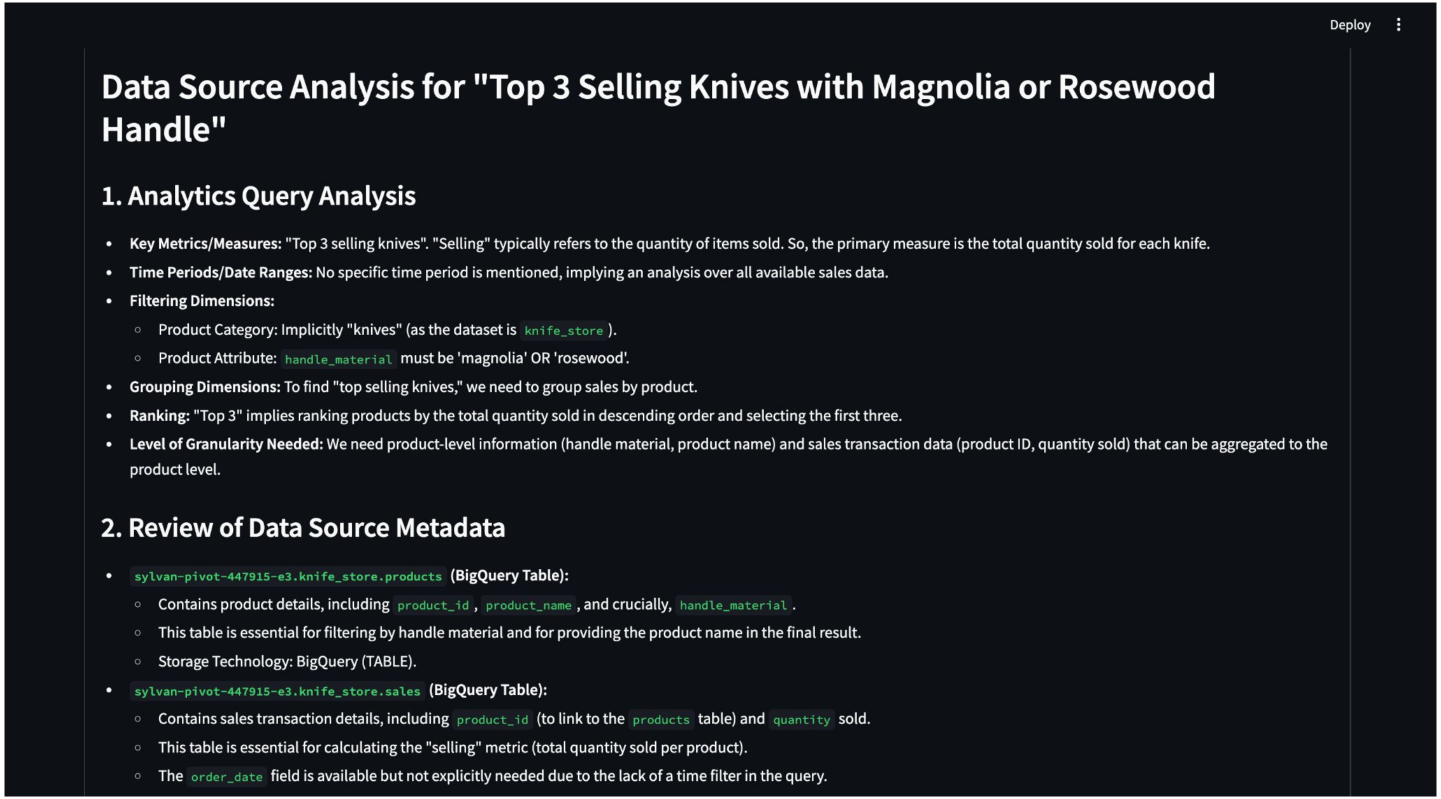


Beam Pipeline Viewer

User Query

Can you give me the top 3 selling knives with a magnolia or rosewood handle?

Data Analysis



Data Source Analysis for "Top 3 Selling Knives with Magnolia or Rosewood Handle"

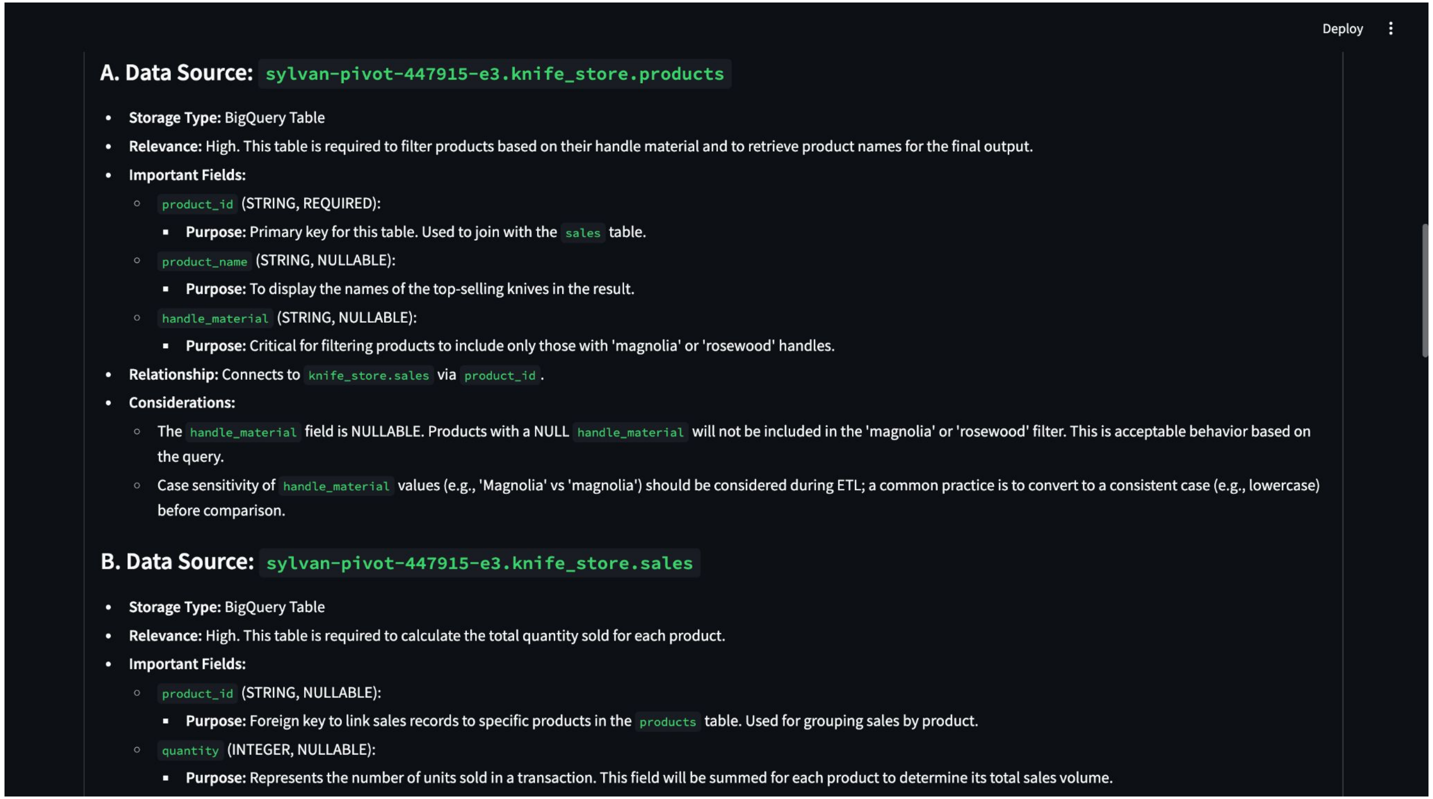
1. Analytics Query Analysis

- **Key Metrics/Measures:** "Top 3 selling knives". "Selling" typically refers to the quantity of items sold. So, the primary measure is the total quantity sold for each knife.
- **Time Periods/Date Ranges:** No specific time period is mentioned, implying an analysis over all available sales data.
- **Filtering Dimensions:**
  - Product Category: Implicitly "knives" (as the dataset is `knife_store`).
  - Product Attribute: `handle_material` must be 'magnolia' OR 'rosewood'.
- **Grouping Dimensions:** To find "top selling knives," we need to group sales by product.
- **Ranking:** "Top 3" implies ranking products by the total quantity sold in descending order and selecting the first three.
- **Level of Granularity Needed:** We need product-level information (handle material, product name) and sales transaction data (product ID, quantity sold) that can be aggregated to the product level.

2. Review of Data Source Metadata

- `sylvan-pivot-447915-e3.knife_store.products` (BigQuery Table):
  - Contains product details, including `product_id`, `product_name`, and crucially, `handle_material`.
  - This table is essential for filtering by handle material and for providing the product name in the final result.
  - Storage Technology: BigQuery (TABLE).
- `sylvan-pivot-447915-e3.knife_store.sales` (BigQuery Table):
  - Contains sales transaction details, including `product_id` (to link to the `products` table) and `quantity` sold.
  - This table is essential for calculating the "selling" metric (total quantity sold per product).
  - The `order_date` field is available but not explicitly needed due to the lack of a time filter in the query.

Data Analysis



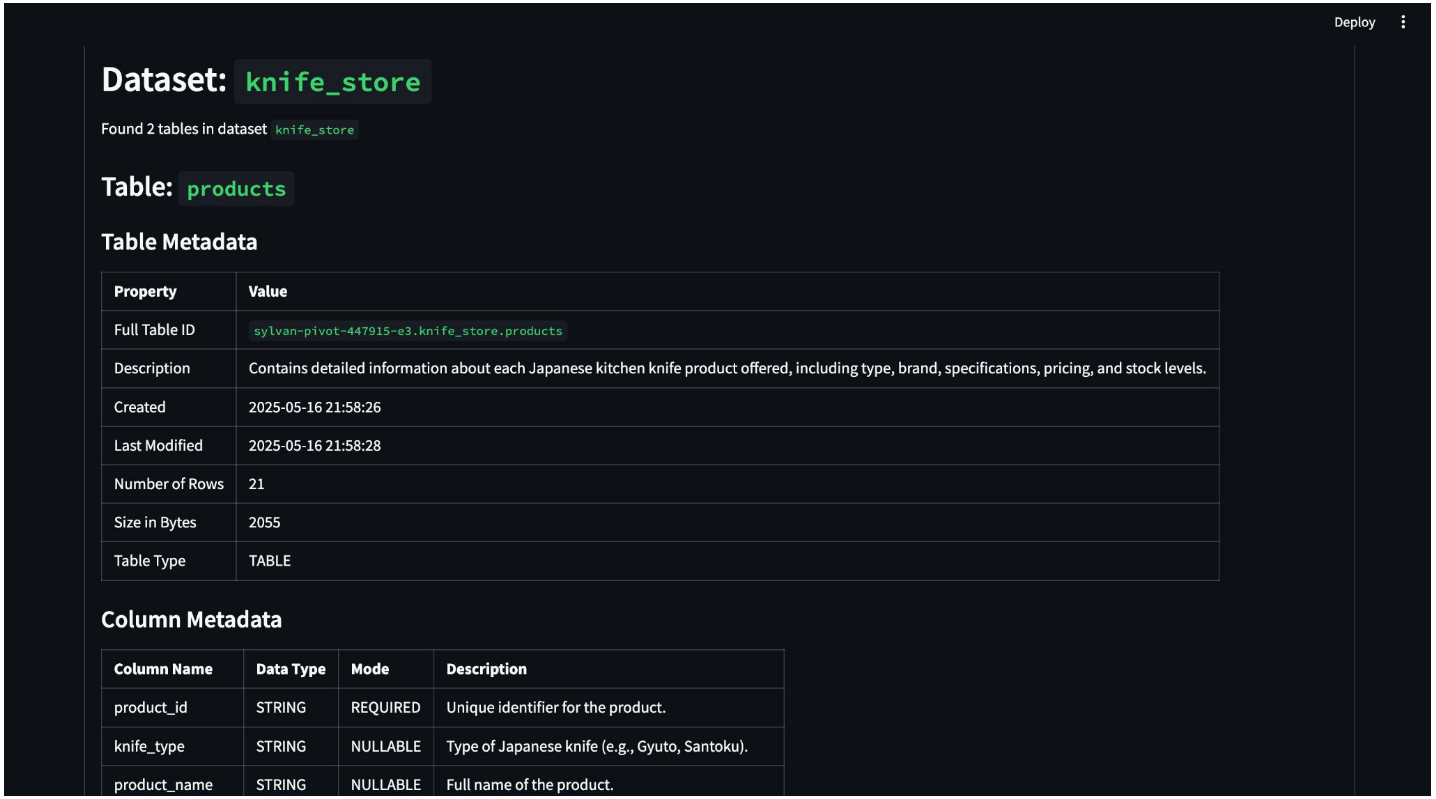
A. Data Source: `sylvan-pivot-447915-e3.knife_store.products`

- **Storage Type:** BigQuery Table
- **Relevance:** High. This table is required to filter products based on their handle material and to retrieve product names for the final output.
- **Important Fields:**
  - `product_id` (STRING, REQUIRED):
    - **Purpose:** Primary key for this table. Used to join with the `sales` table.
  - `product_name` (STRING, NULLABLE):
    - **Purpose:** To display the names of the top-selling knives in the result.
  - `handle_material` (STRING, NULLABLE):
    - **Purpose:** Critical for filtering products to include only those with 'magnolia' or 'rosewood' handles.
- **Relationship:** Connects to `knife_store.sales` via `product_id`.
- **Considerations:**
  - The `handle_material` field is NULLABLE. Products with a NULL `handle_material` will not be included in the 'magnolia' or 'rosewood' filter. This is acceptable behavior based on the query.
  - Case sensitivity of `handle_material` values (e.g., 'Magnolia' vs 'magnolia') should be considered during ETL; a common practice is to convert to a consistent case (e.g., lowercase) before comparison.

B. Data Source: `sylvan-pivot-447915-e3.knife_store.sales`

- **Storage Type:** BigQuery Table
- **Relevance:** High. This table is required to calculate the total quantity sold for each product.
- **Important Fields:**
  - `product_id` (STRING, NULLABLE):
    - **Purpose:** Foreign key to link sales records to specific products in the `products` table. Used for grouping sales by product.
  - `quantity` (INTEGER, NULLABLE):
    - **Purpose:** Represents the number of units sold in a transaction. This field will be summed for each product to determine its total sales volume.

Metadata Fetched via Tool Calling



Dataset: `knife_store`

Found 2 tables in dataset `knife_store`

Table: `products`

Table Metadata

Property	Value
Full Table ID	`sylvan-pivot-447915-e3.knife_store.products`
Description	Contains detailed information about each Japanese kitchen knife product offered, including type, brand, specifications, pricing, and stock levels.
Created	2025-05-16 21:58:26
Last Modified	2025-05-16 21:58:28
Number of Rows	21
Size in Bytes	2055
Table Type	TABLE

Column Metadata

Column Name	Data Type	Mode	Description
product_id	STRING	REQUIRED	Unique identifier for the product.
knife_type	STRING	NULLABLE	Type of Japanese knife (e.g., Gyuto, Santoku).
product_name	STRING	NULLABLE	Full name of the product.

Metadata Fetched via Tool Calling

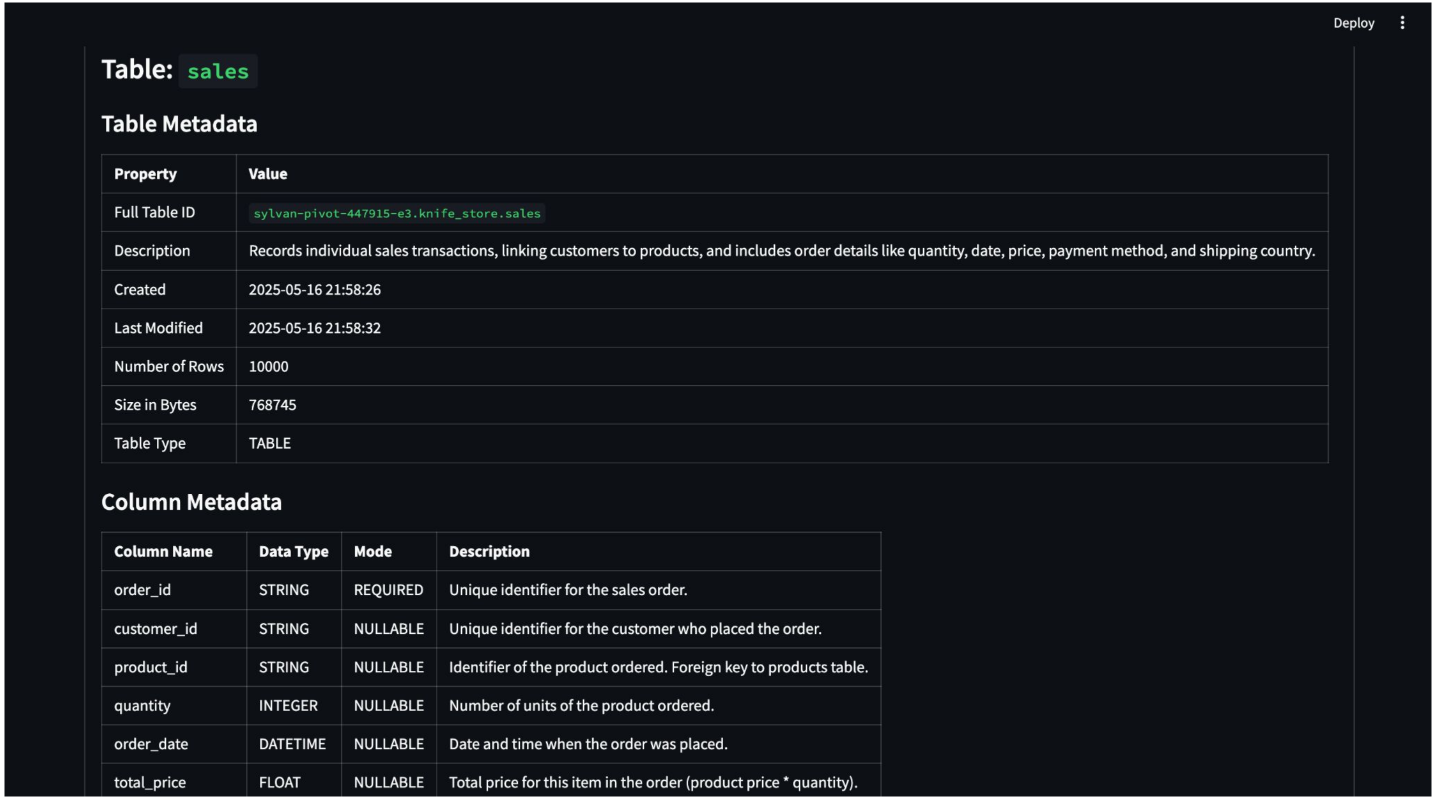


Table: `sales`

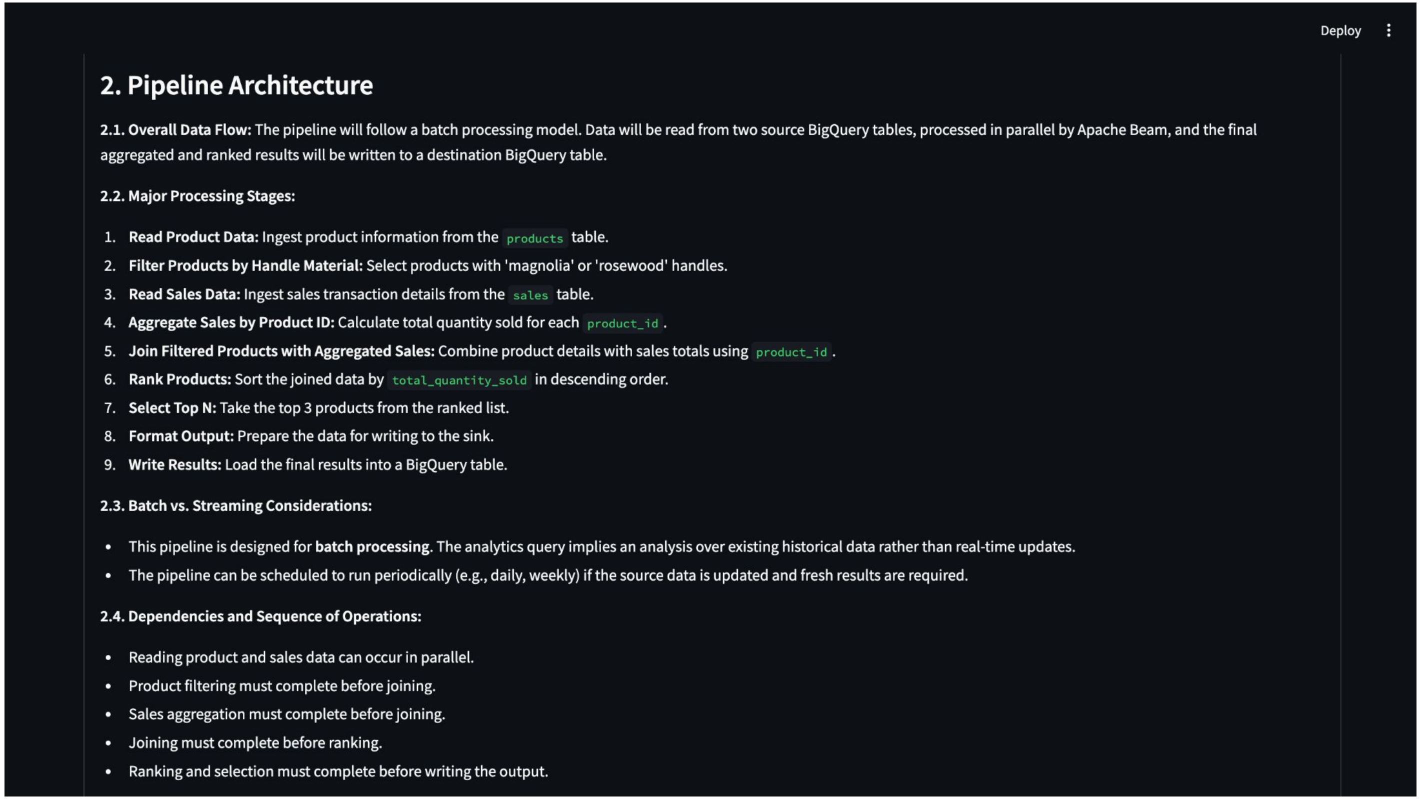
Table Metadata

Property	Value
Full Table ID	`sylvan-pivot-447915-e3.knife_store.sales`
Description	Records individual sales transactions, linking customers to products, and includes order details like quantity, date, price, payment method, and shipping country.
Created	2025-05-16 21:58:26
Last Modified	2025-05-16 21:58:32
Number of Rows	10000
Size in Bytes	768745
Table Type	TABLE

Column Metadata

Column Name	Data Type	Mode	Description
order_id	STRING	REQUIRED	Unique identifier for the sales order.
customer_id	STRING	NULLABLE	Unique identifier for the customer who placed the order.
product_id	STRING	NULLABLE	Identifier of the product ordered. Foreign key to products table.
quantity	INTEGER	NULLABLE	Number of units of the product ordered.
order_date	DATETIME	NULLABLE	Date and time when the order was placed.
total_price	FLOAT	NULLABLE	Total price for this item in the order (product price * quantity).

Write Requirements Beam Pipeline



2. Pipeline Architecture

2.1. Overall Data Flow: The pipeline will follow a batch processing model. Data will be read from two source BigQuery tables, processed in parallel by Apache Beam, and the final aggregated and ranked results will be written to a destination BigQuery table.

2.2. Major Processing Stages:

1. **Read Product Data:** Ingest product information from the `products` table.
2. **Filter Products by Handle Material:** Select products with 'magnolia' or 'rosewood' handles.
3. **Read Sales Data:** Ingest sales transaction details from the `sales` table.
4. **Aggregate Sales by Product ID:** Calculate total quantity sold for each `product_id`.
5. **Join Filtered Products with Aggregated Sales:** Combine product details with sales totals using `product_id`.
6. **Rank Products:** Sort the joined data by `total_quantity_sold` in descending order.
7. **Select Top N:** Take the top 3 products from the ranked list.
8. **Format Output:** Prepare the data for writing to the sink.
9. **Write Results:** Load the final results into a BigQuery table.

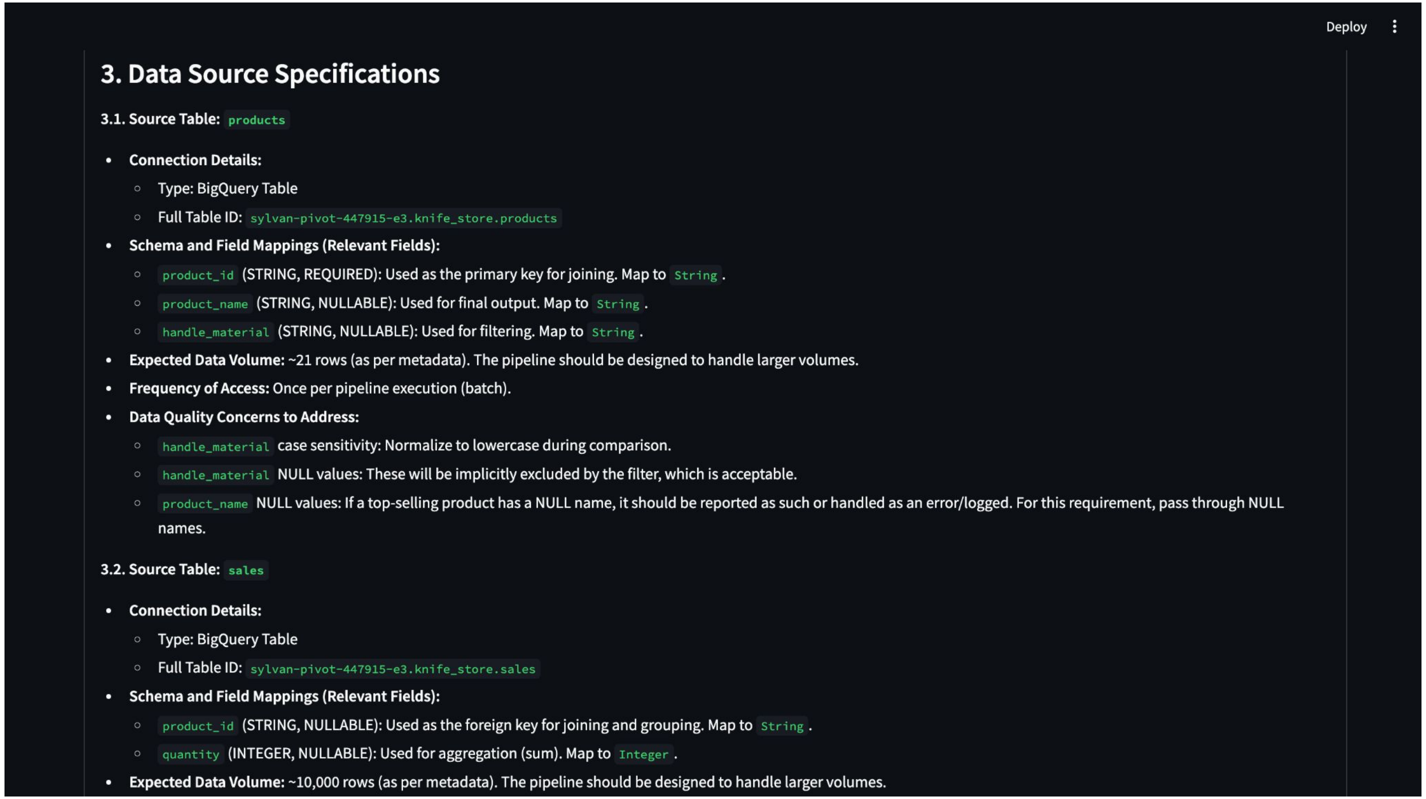
2.3. Batch vs. Streaming Considerations:

- This pipeline is designed for **batch processing**. The analytics query implies an analysis over existing historical data rather than real-time updates.
- The pipeline can be scheduled to run periodically (e.g., daily, weekly) if the source data is updated and fresh results are required.

2.4. Dependencies and Sequence of Operations:

- Reading product and sales data can occur in parallel.
- Product filtering must complete before joining.
- Sales aggregation must complete before joining.
- Joining must complete before ranking.
- Ranking and selection must complete before writing the output.

Write Requirements Beam Pipeline



3. Data Source Specifications

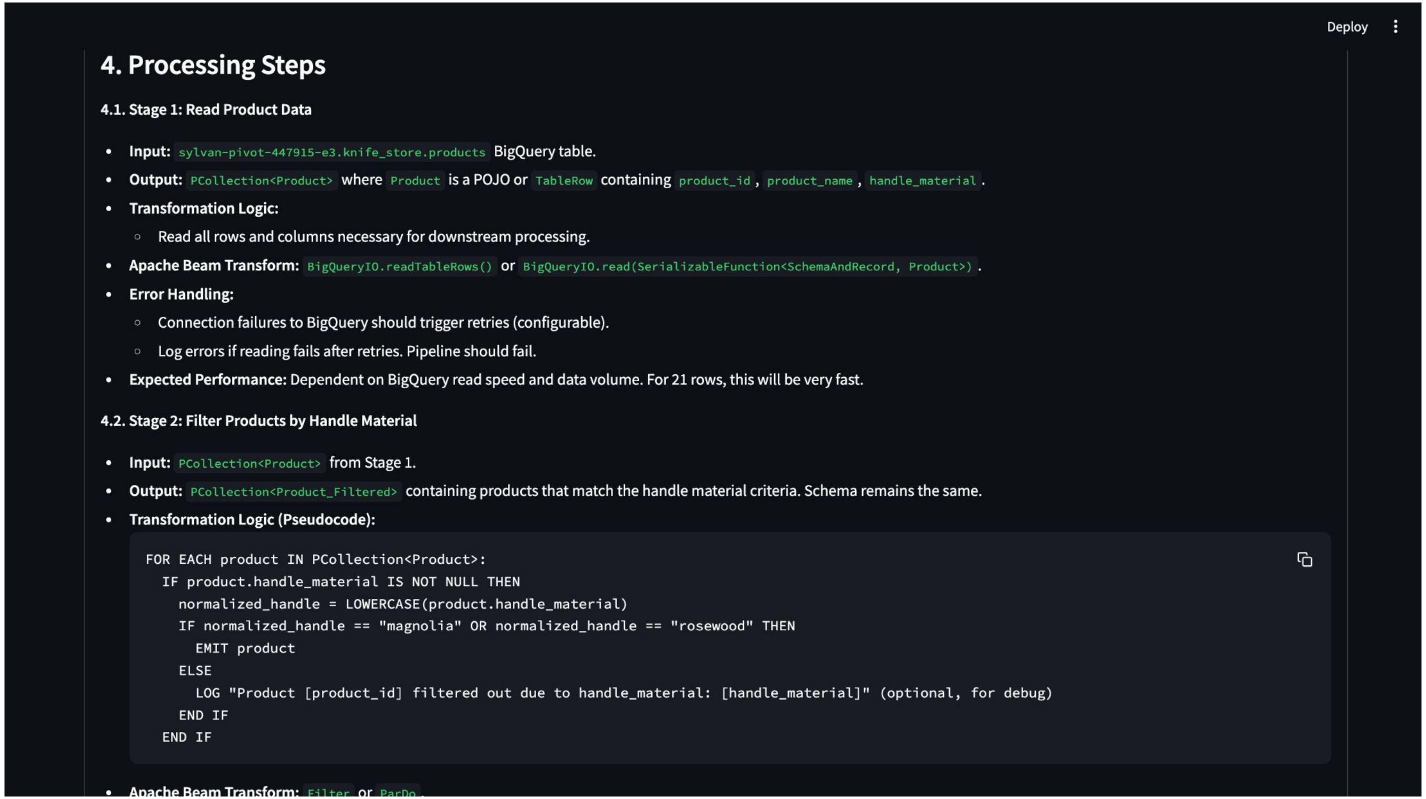
3.1. Source Table: `products`

- **Connection Details:**
  - Type: BigQuery Table
  - Full Table ID: `sylvan-pivot-447915-e3.knife_store.products`
- **Schema and Field Mappings (Relevant Fields):**
  - `product_id` (STRING, REQUIRED): Used as the primary key for joining. Map to `String`.
  - `product_name` (STRING, NULLABLE): Used for final output. Map to `String`.
  - `handle_material` (STRING, NULLABLE): Used for filtering. Map to `String`.
- **Expected Data Volume:** ~21 rows (as per metadata). The pipeline should be designed to handle larger volumes.
- **Frequency of Access:** Once per pipeline execution (batch).
- **Data Quality Concerns to Address:**
  - `handle_material` case sensitivity: Normalize to lowercase during comparison.
  - `handle_material` NULL values: These will be implicitly excluded by the filter, which is acceptable.
  - `product_name` NULL values: If a top-selling product has a NULL name, it should be reported as such or handled as an error/logged. For this requirement, pass through NULL names.

3.2. Source Table: `sales`

- **Connection Details:**
  - Type: BigQuery Table
  - Full Table ID: `sylvan-pivot-447915-e3.knife_store.sales`
- **Schema and Field Mappings (Relevant Fields):**
  - `product_id` (STRING, NULLABLE): Used as the foreign key for joining and grouping. Map to `String`.
  - `quantity` (INTEGER, NULLABLE): Used for aggregation (sum). Map to `Integer`.
- **Expected Data Volume:** ~10,000 rows (as per metadata). The pipeline should be designed to handle larger volumes.

Write Requirements Beam Pipeline



4. Processing Steps

4.1. Stage 1: Read Product Data

- **Input:** `sylvan-pivot-447915-e3.knife_store.products` BigQuery table.
- **Output:** `PCollection<Product>` where `Product` is a POJO or `TableRow` containing `product_id`, `product_name`, `handle_material`.
- **Transformation Logic:**
  - Read all rows and columns necessary for downstream processing.
- **Apache Beam Transform:** `BigQueryIO.readTableRows()` or `BigQueryIO.read(SerializableFunction<SchemaAndRecord, Product>)`.
- **Error Handling:**
  - Connection failures to BigQuery should trigger retries (configurable).
  - Log errors if reading fails after retries. Pipeline should fail.
- **Expected Performance:** Dependent on BigQuery read speed and data volume. For 21 rows, this will be very fast.

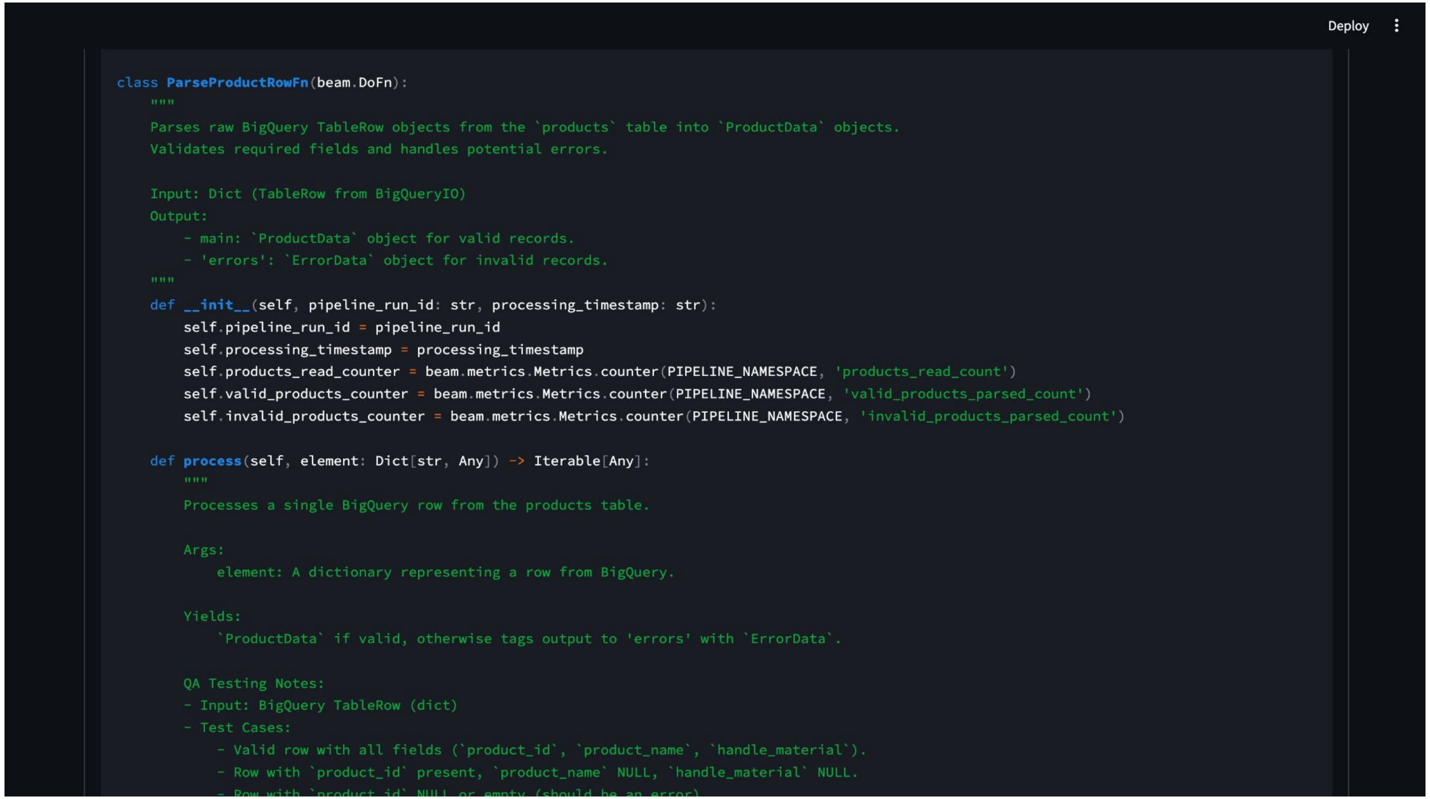
4.2. Stage 2: Filter Products by Handle Material

- **Input:** `PCollection<Product>` from Stage 1.
- **Output:** `PCollection<Product_Filtered>` containing products that match the handle material criteria. Schema remains the same.
- **Transformation Logic (Pseudocode):**

```
FOR EACH product IN PCollection<Product>:
  IF product.handle_material IS NOT NULL THEN
    normalized_handle = LOWERCASE(product.handle_material)
    IF normalized_handle == "magnolia" OR normalized_handle == "rosewood" THEN
      EMIT product
    ELSE
      LOG "Product [product_id] filtered out due to handle_material: [handle_material]" (optional, for debug)
    END IF
  END IF
```

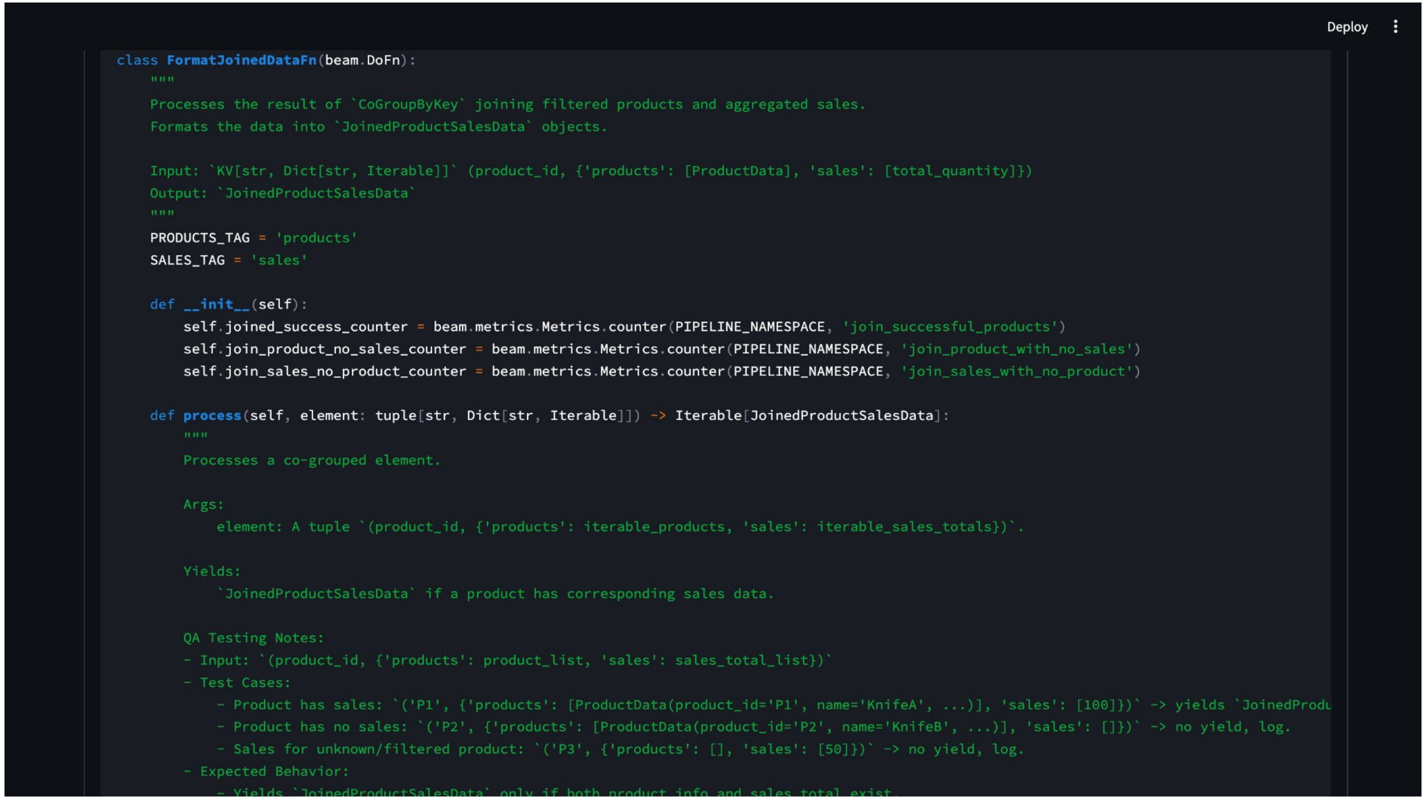
- **Apache Beam Transform:** `Filter` or `ParDo`.

Beam Pipeline Implementation



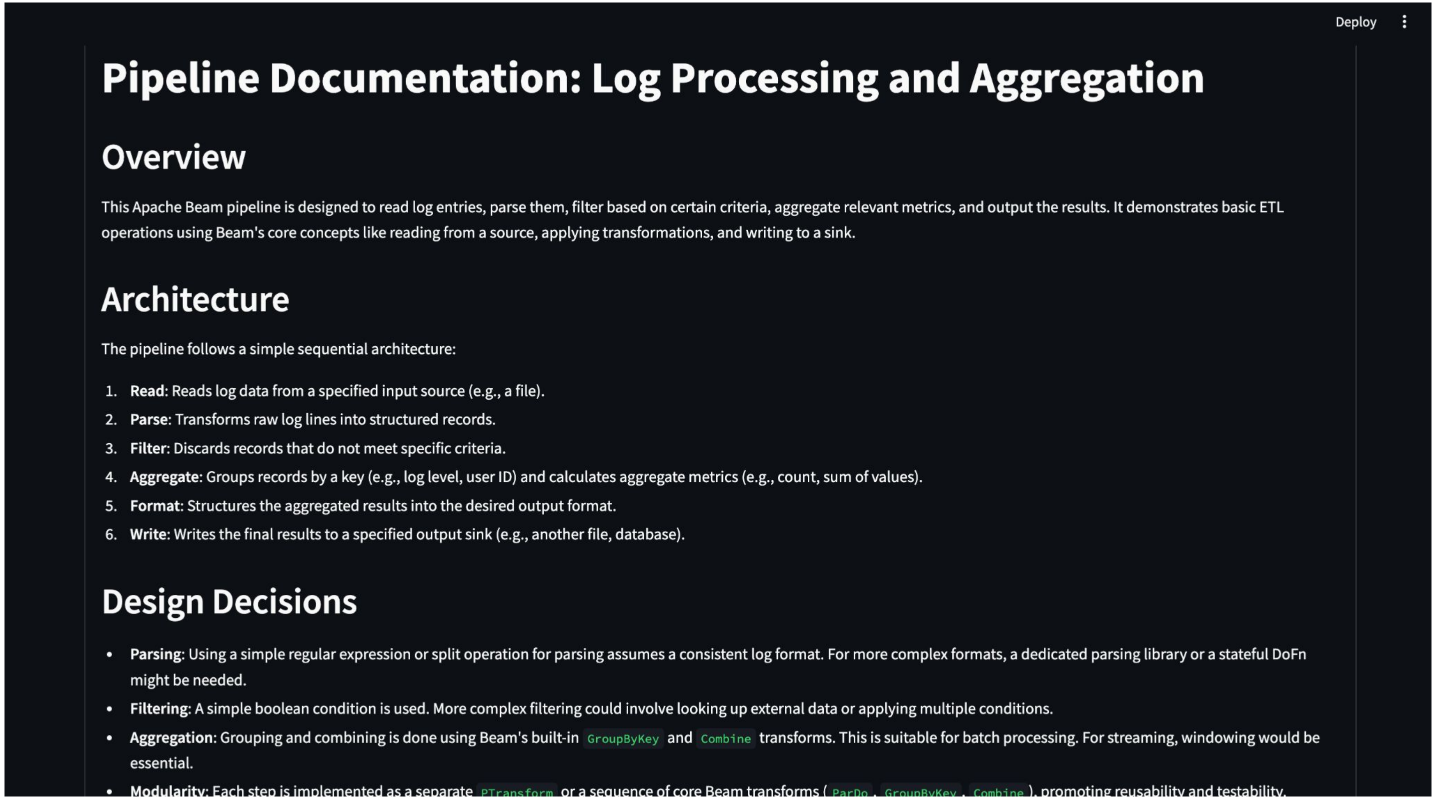
```python
class ParseProductRowFn(beam.DoFn):
    """
    Parses raw BigQuery TableRow objects from the `products` table into `ProductData` objects.
    Validates required fields and handles potential errors.

    Input: Dict (TableRow from BigQueryIO)
    Output:
        - main: `ProductData` object for valid records.
        - 'errors': `ErrorData` object for invalid records.
    """
    def __init__(self, pipeline_run_id: str, processing_timestamp: str):
        self.pipeline_run_id = pipeline_run_id
        self.processing_timestamp = processing_timestamp
        self.products_read_counter = beam.metrics.Metrics.counter(PIPELINE_NAMESPACE, 'products_read_count')
        self.valid_products_counter = beam.metrics.Metrics.counter(PIPELINE_NAMESPACE, 'valid_products_parsed_count')
        self.invalid_products_counter = beam.metrics.Metrics.counter(PIPELINE_NAMESPACE, 'invalid_products_parsed_count')

    def process(self, element: Dict[str, Any]) -> Iterable[Any]:
        """
        Processes a single BigQuery row from the products table.

        Args:
            element: A dictionary representing a row from BigQuery.

        Yields:
            `ProductData` if valid, otherwise tags output to 'errors' with `ErrorData`.

        QA Testing Notes:
        - Input: BigQuery TableRow (dict)
        - Test Cases:
            - Valid row with all fields (`product_id`, `product_name`, `handle_material`).
            - Row with `product_id` present, `product_name` NULL, `handle_material` NULL.
            - Row with `product_id` NULL or empty (should be an error).
```

Beam Pipeline Implementation



```
class FormatJoinedDataFn(beam.DoFn):
    """
    Processes the result of `CoGroupByKey` joining filtered products and aggregated sales.
    Formats the data into `JoinedProductSalesData` objects.

    Input: `KV[str, Dict[str, Iterable]]` (product_id, {'products': [ProductData], 'sales': [total_quantity]})
    Output: `JoinedProductSalesData`
    """
    PRODUCTS_TAG = 'products'
    SALES_TAG = 'sales'

    def __init__(self):
        self.joined_success_counter = beam.metrics.Metrics.counter(PIPELINE_NAMESPACE, 'join_successful_products')
        self.join_product_no_sales_counter = beam.metrics.Metrics.counter(PIPELINE_NAMESPACE, 'join_product_with_no_sales')
        self.join_sales_no_product_counter = beam.metrics.Metrics.counter(PIPELINE_NAMESPACE, 'join_sales_with_no_product')

    def process(self, element: tuple[str, Dict[str, Iterable]]) -> Iterable[JoinedProductSalesData]:
        """
        Processes a co-grouped element.

        Args:
            element: A tuple `(product_id, {'products': iterable_products, 'sales': iterable_sales_totals})`.

        Yields:
            `JoinedProductSalesData` if a product has corresponding sales data.

        QA Testing Notes:
        - Input: `(product_id, {'products': product_list, 'sales': sales_total_list})`
        - Test Cases:
            - Product has sales: `('P1', {'products': [ProductData(product_id='P1', name='KnifeA', ...)], 'sales': [100]})` -> yields `JoinedProdu
            - Product has no sales: `('P2', {'products': [ProductData(product_id='P2', name='KnifeB', ...)], 'sales': []})` -> no yield, log.
            - Sales for unknown/filtered product: `('P3', {'products': [], 'sales': [50]})` -> no yield, log.
        - Expected Behavior:
            - Yields `JoinedProductSalesData` only if both product info and sales total exist.
```

Documentation



Pipeline Documentation: Log Processing and Aggregation

Overview

This Apache Beam pipeline is designed to read log entries, parse them, filter based on certain criteria, aggregate relevant metrics, and output the results. It demonstrates basic ETL operations using Beam's core concepts like reading from a source, applying transformations, and writing to a sink.

Architecture

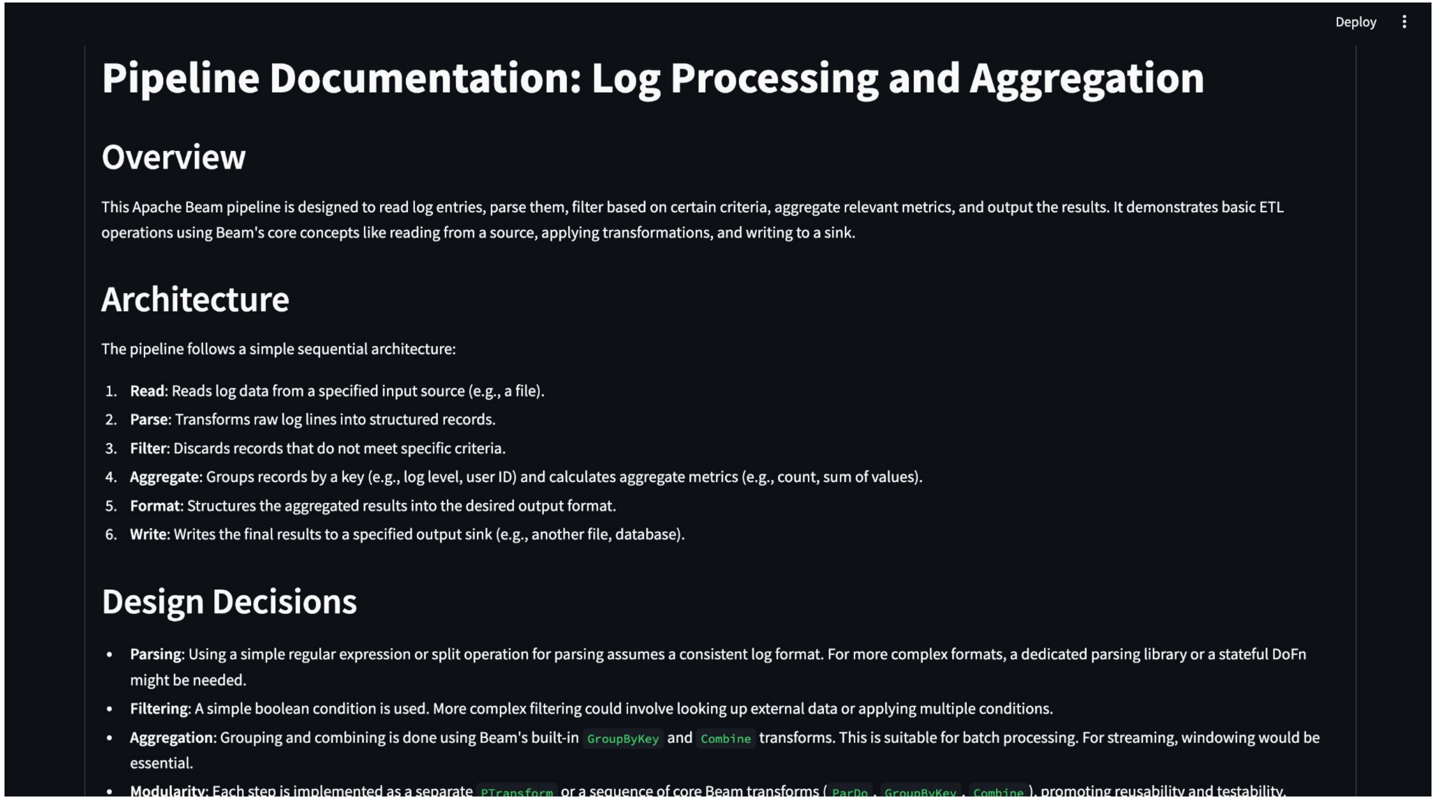
The pipeline follows a simple sequential architecture:

1. **Read**: Reads log data from a specified input source (e.g., a file).
2. **Parse**: Transforms raw log lines into structured records.
3. **Filter**: Discards records that do not meet specific criteria.
4. **Aggregate**: Groups records by a key (e.g., log level, user ID) and calculates aggregate metrics (e.g., count, sum of values).
5. **Format**: Structures the aggregated results into the desired output format.
6. **Write**: Writes the final results to a specified output sink (e.g., another file, database).

Design Decisions

- **Parsing**: Using a simple regular expression or split operation for parsing assumes a consistent log format. For more complex formats, a dedicated parsing library or a stateful DoFn might be needed.
- **Filtering**: A simple boolean condition is used. More complex filtering could involve looking up external data or applying multiple conditions.
- **Aggregation**: Grouping and combining is done using Beam's built-in `GroupByKey` and `Combine` transforms. This is suitable for batch processing. For streaming, windowing would be essential.
- **Modularity**: Each step is implemented as a separate `PTransform` or a sequence of core Beam transforms (`ParDo`, `GroupByKey`, `Combine`), promoting reusability and testability.

Documentation



Pipeline Documentation: Log Processing and Aggregation

Overview

This Apache Beam pipeline is designed to read log entries, parse them, filter based on certain criteria, aggregate relevant metrics, and output the results. It demonstrates basic ETL operations using Beam's core concepts like reading from a source, applying transformations, and writing to a sink.

Architecture

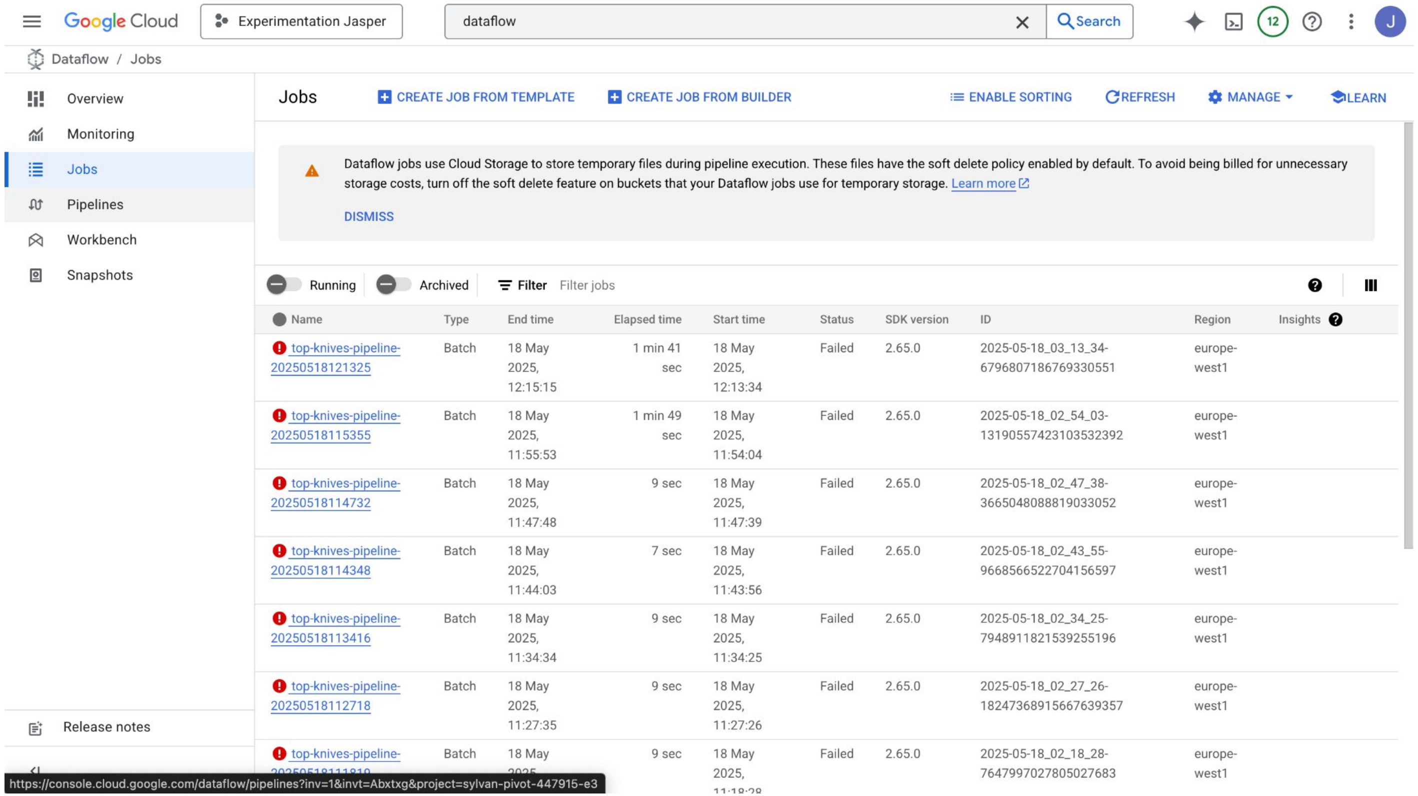
The pipeline follows a simple sequential architecture:

1. **Read**: Reads log data from a specified input source (e.g., a file).
2. **Parse**: Transforms raw log lines into structured records.
3. **Filter**: Discards records that do not meet specific criteria.
4. **Aggregate**: Groups records by a key (e.g., log level, user ID) and calculates aggregate metrics (e.g., count, sum of values).
5. **Format**: Structures the aggregated results into the desired output format.
6. **Write**: Writes the final results to a specified output sink (e.g., another file, database).

Design Decisions

- **Parsing**: Using a simple regular expression or split operation for parsing assumes a consistent log format. For more complex formats, a dedicated parsing library or a stateful DoFn might be needed.
- **Filtering**: A simple boolean condition is used. More complex filtering could involve looking up external data or applying multiple conditions.
- **Aggregation**: Grouping and combining is done using Beam's built-in `GroupByKey` and `Combine` transforms. This is suitable for batch processing. For streaming, windowing would be essential.
- **Modularity**: Each step is implemented as a separate `PTransform` or a sequence of core Beam transforms (`ParDo`, `GroupByKey`, `Combine`), promoting reusability and testability.

Running the Pipeline on Dataflow

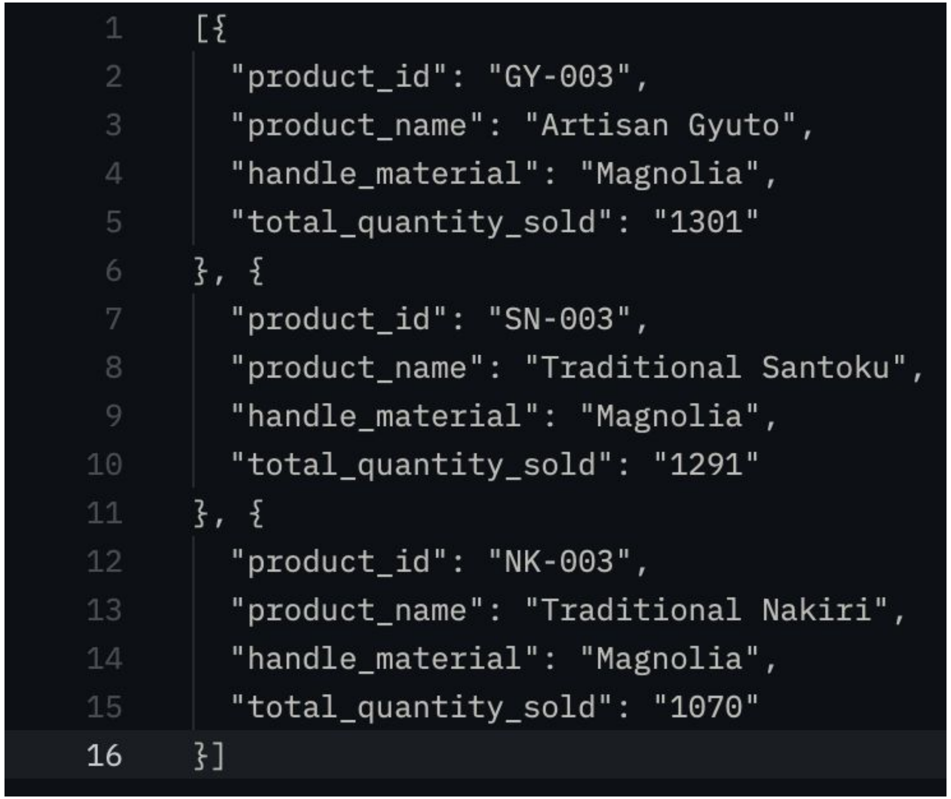


Dataflow jobs use Cloud Storage to store temporary files during pipeline execution. These files have the soft delete policy enabled by default. To avoid being billed for unnecessary storage costs, turn off the soft delete feature on buckets that your Dataflow jobs use for temporary storage. Learn more

Name	Type	End time	Elapsed time	Start time	Status	SDK version	ID	Region	Insights
top-knives-pipeline-20250518121325	Batch	18 May 2025, 12:15:15	1 min 41 sec	18 May 2025, 12:13:34	Failed	2.65.0	2025-05-18_03_13_34-6796807186769330551	europe-west1	
top-knives-pipeline-20250518115355	Batch	18 May 2025, 11:55:53	1 min 49 sec	18 May 2025, 11:54:04	Failed	2.65.0	2025-05-18_02_54_03-13190557423103532392	europe-west1	
top-knives-pipeline-20250518114732	Batch	18 May 2025, 11:47:48	9 sec	18 May 2025, 11:47:39	Failed	2.65.0	2025-05-18_02_47_38-3665048088819033052	europe-west1	
top-knives-pipeline-20250518114348	Batch	18 May 2025, 11:44:03	7 sec	18 May 2025, 11:43:56	Failed	2.65.0	2025-05-18_02_43_55-9668566522704156597	europe-west1	
top-knives-pipeline-20250518113416	Batch	18 May 2025, 11:34:34	9 sec	18 May 2025, 11:34:25	Failed	2.65.0	2025-05-18_02_34_25-7948911821539255196	europe-west1	
top-knives-pipeline-20250518112718	Batch	18 May 2025, 11:27:35	9 sec	18 May 2025, 11:27:26	Failed	2.65.0	2025-05-18_02_27_26-18247368915667639357	europe-west1	
top-knives-pipeline-20250518111810	Batch	18 May 2025,	9 sec	18 May 2025, 11:18:28	Failed	2.65.0	2025-05-18_02_18_28-7647997027805027683	europe-west1	

https://console.cloud.google.com/dataflow/pipelines?inv=1&invt=Abxtxg&project=sylvan-pivot-447915-e3

Results Stored in a JSON File

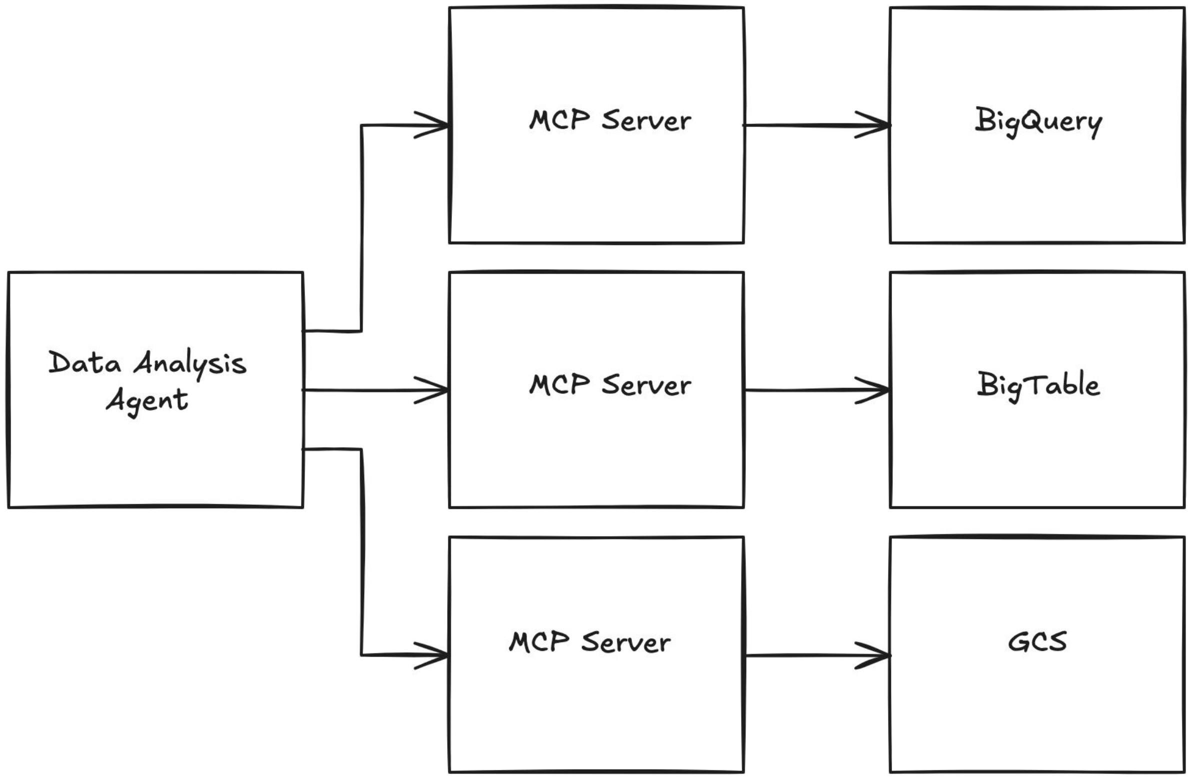


```json
[{
  "product_id": "GY-003",
  "product_name": "Artisan Gyuto",
  "handle_material": "Magnolia",
  "total_quantity_sold": "1301"
}, {
  "product_id": "SN-003",
  "product_name": "Traditional Santoku",
  "handle_material": "Magnolia",
  "total_quantity_sold": "1291"
}, {
  "product_id": "NK-003",
  "product_name": "Traditional Nakiri",
  "handle_material": "Magnolia",
  "total_quantity_sold": "1070"
}]
```

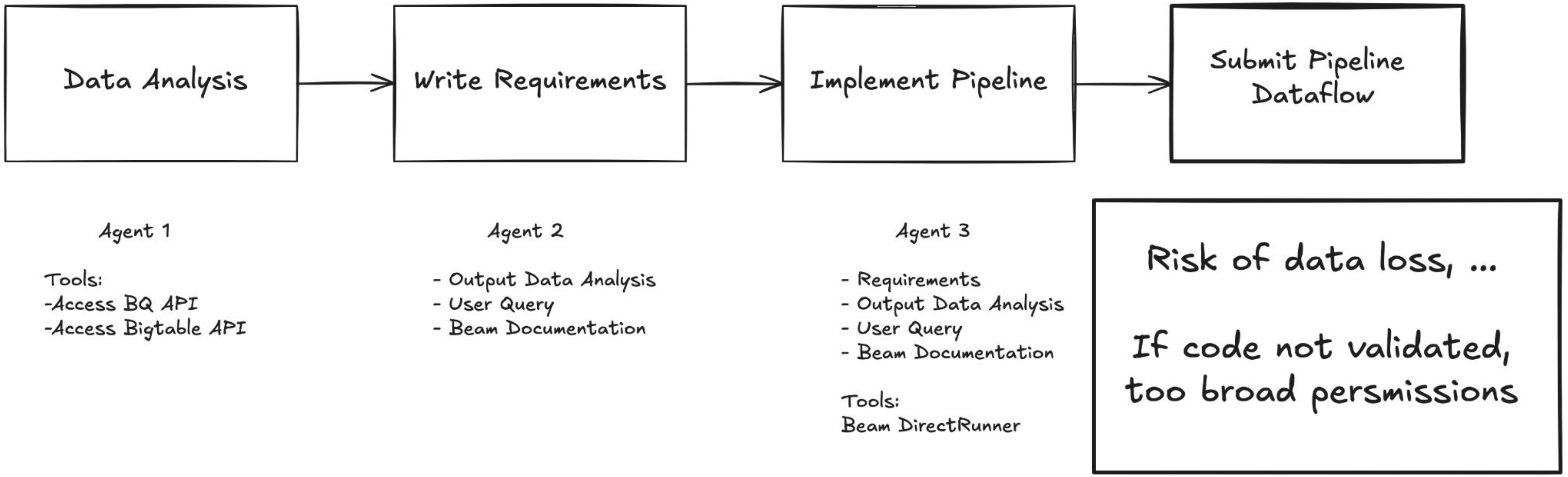
Outcomes & Future Vision

- What you achieved during the hackathon (metrics, MVP features, etc.)
- Next steps: what would you build or improve if given more time?
- Optional: ask or call-to-action (e.g., "We’d love feedback on...")

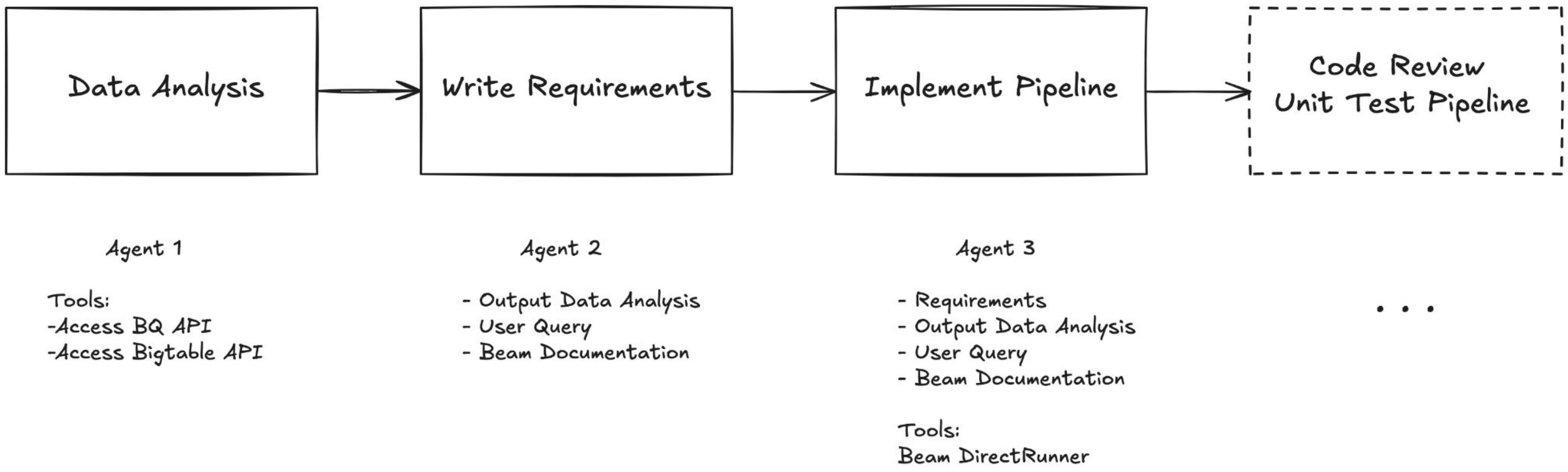
Outcomes & Future Vision: Advanced Agent Tools



Outcomes & Future Vision: Address Security Risks



Outcomes & Future Vision: Improve Agentic Workflow



Thank you!

Jasper Van den Bossche

jasper.van.den.bossche@ml6.eu

Konstantin Buschmeier

konstantin.buschmeier@ml6.eu