dash1 (cmdb lambda use case).mp4_96 | Summarize Videos, Audio, PDF & Websites

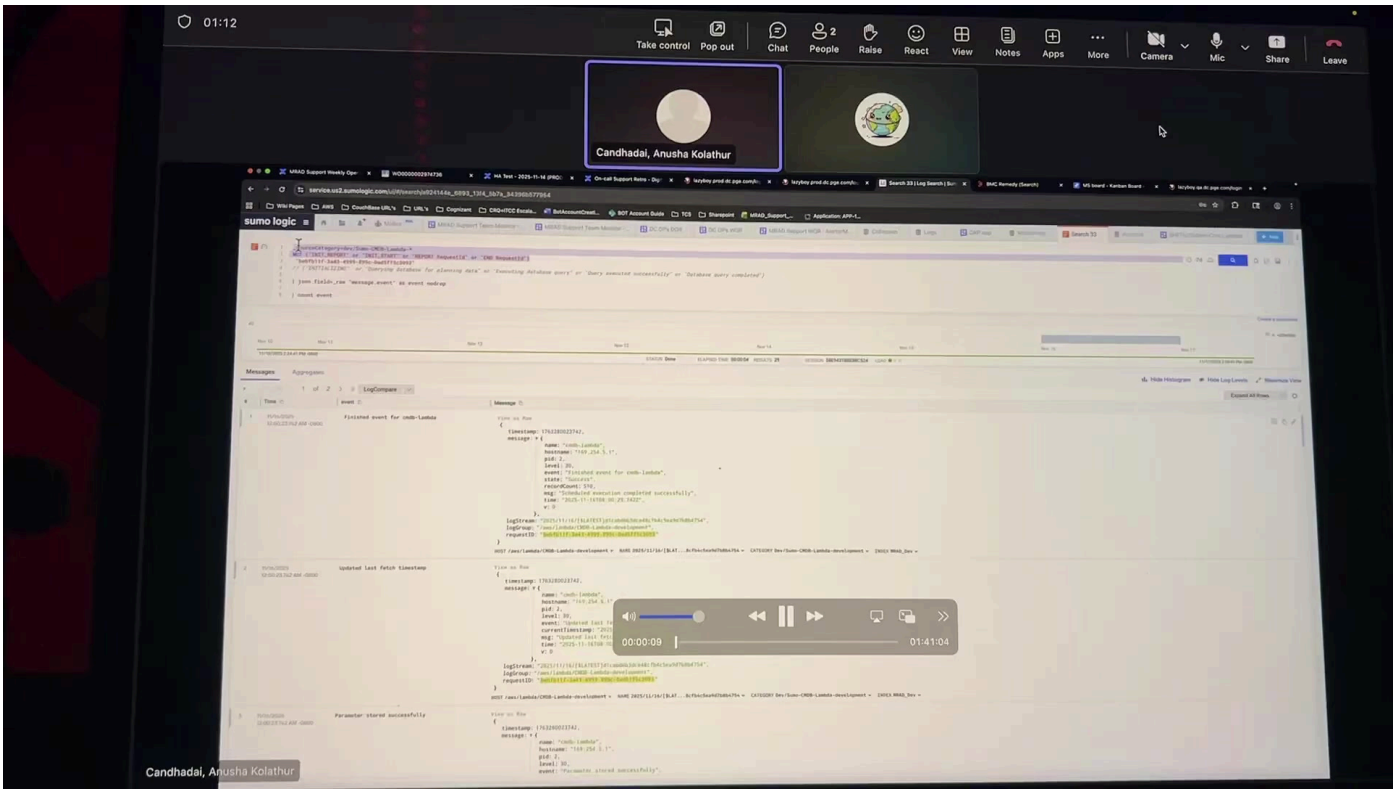


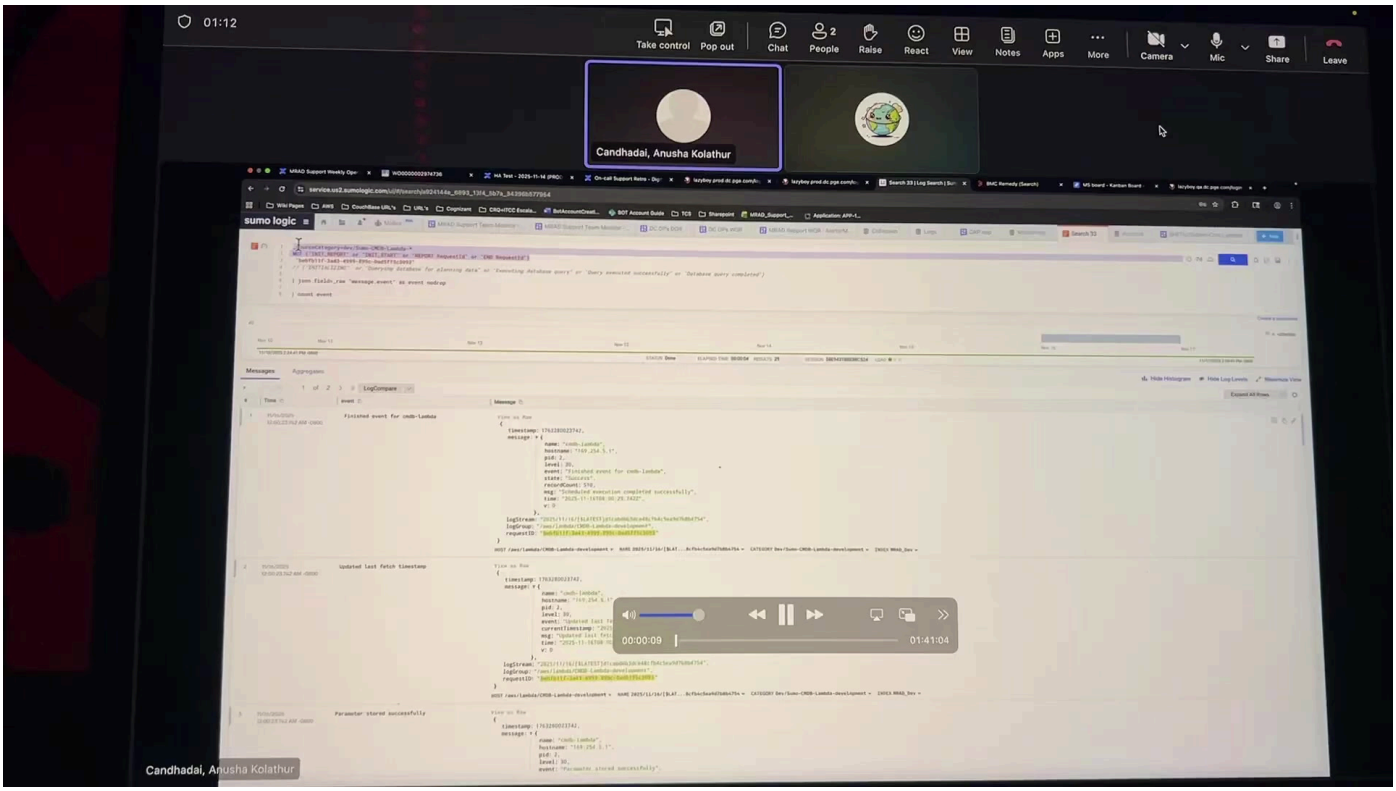
Table of Contents

1. Determining Monitoring Scope from Raw CMDB Lambda Logs
1.3. Reading Logs Chronologically to Understand Flow
1.4. Database Connection and Query Execution Details
2. Parsing Fields for Monitoring Visualization
2.1. Identifying Key Fields for Flow Tracking
2.2. JSON Parsing vs. Raw Parsing
3. Categorizing and Selecting Events for Dashboard Panels
3.1. Organizing Events into Categories
4.1. Determining Dashboard Level of Detail

Unlock the secrets to effective **monitoring** by dissecting raw **CMDB Lambda logs**. Learn the precise steps to filter noise and isolate critical events like **database query execution** and **Couchbase upload** success. This guide shows you how to transform overwhelming logs into actionable dashboard panels focusing only on essential flow and record counts.

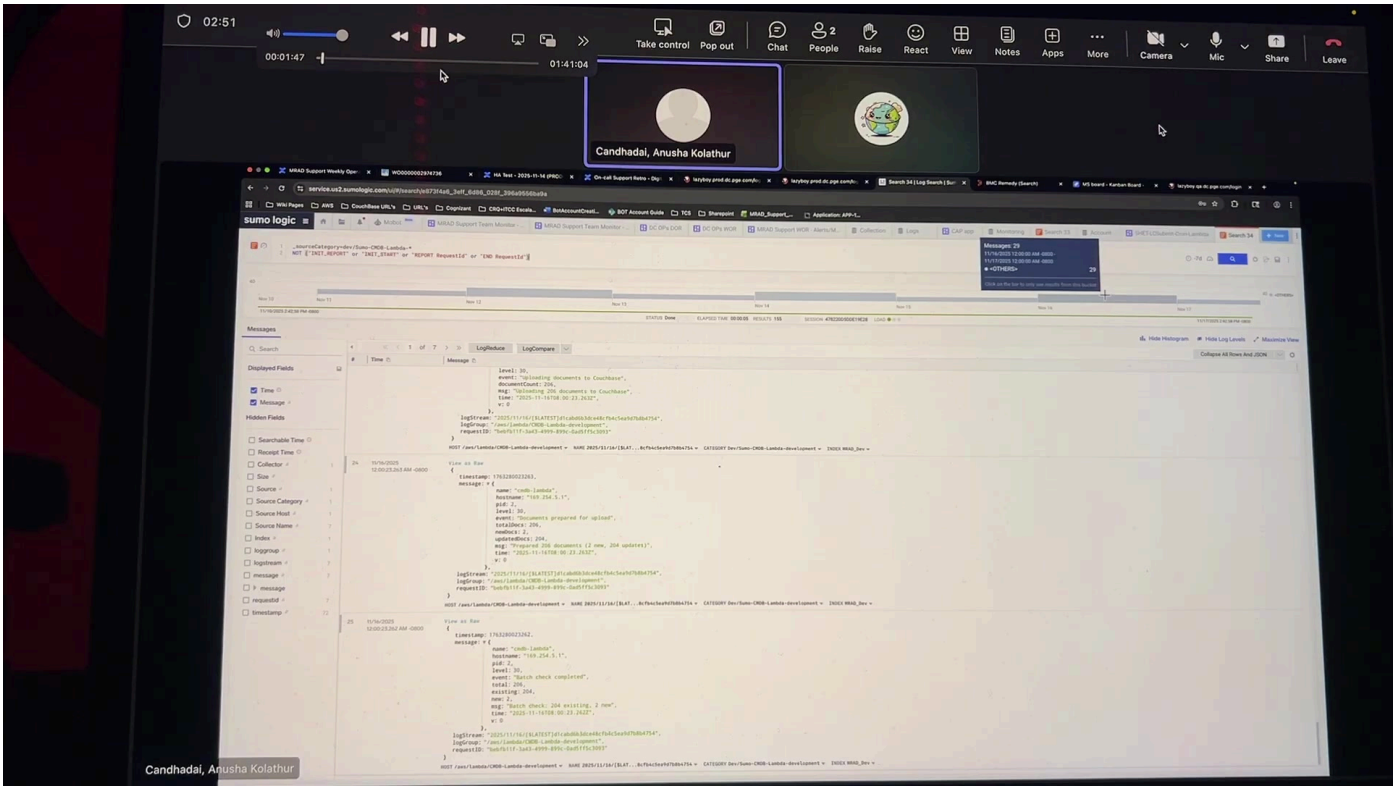
1. Determining Monitoring Scope from Raw CMDB Lambda Logs [1]

00:00:00 (9 min)

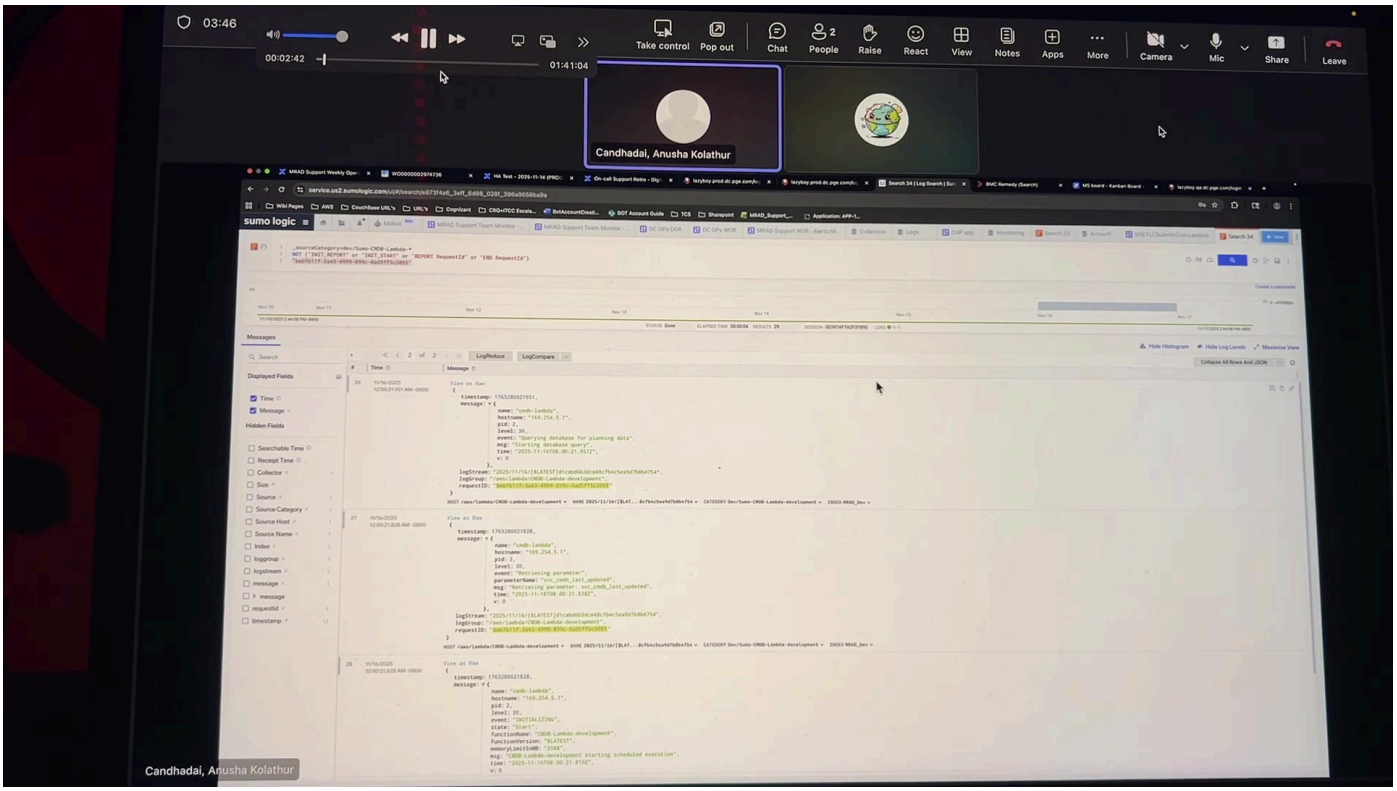


1. **Determine Monitoring Needs**: Define what to monitor, alert on, and ignore from service logs. [1]
2. **Start with Raw Log Search**: Begin by searching raw logs for the specific service in depth. [1]
3. **Observe Initial Log Volume**: A raw search over seven days shows many usual events associated with the lambda. [2]
4. **First Filtering Step**: The immediate action is to start filtering the logs to see only desired events. [3]
5. **Ignore Unnecessary Events**: Exclude events not needed for reports or monitoring visualization. [4]
6. **Ignore Specific Log Patterns**: Use a "not" filter to ignore known unnecessary log patterns for this lambda. [6]
7. **Verify Full Log Visibility**: Confirm that all logs for the service of concern are now visible. [8]

8. **Determine Service Run Frequency**: Check how often the service runs, such as once per day as stated. [14]

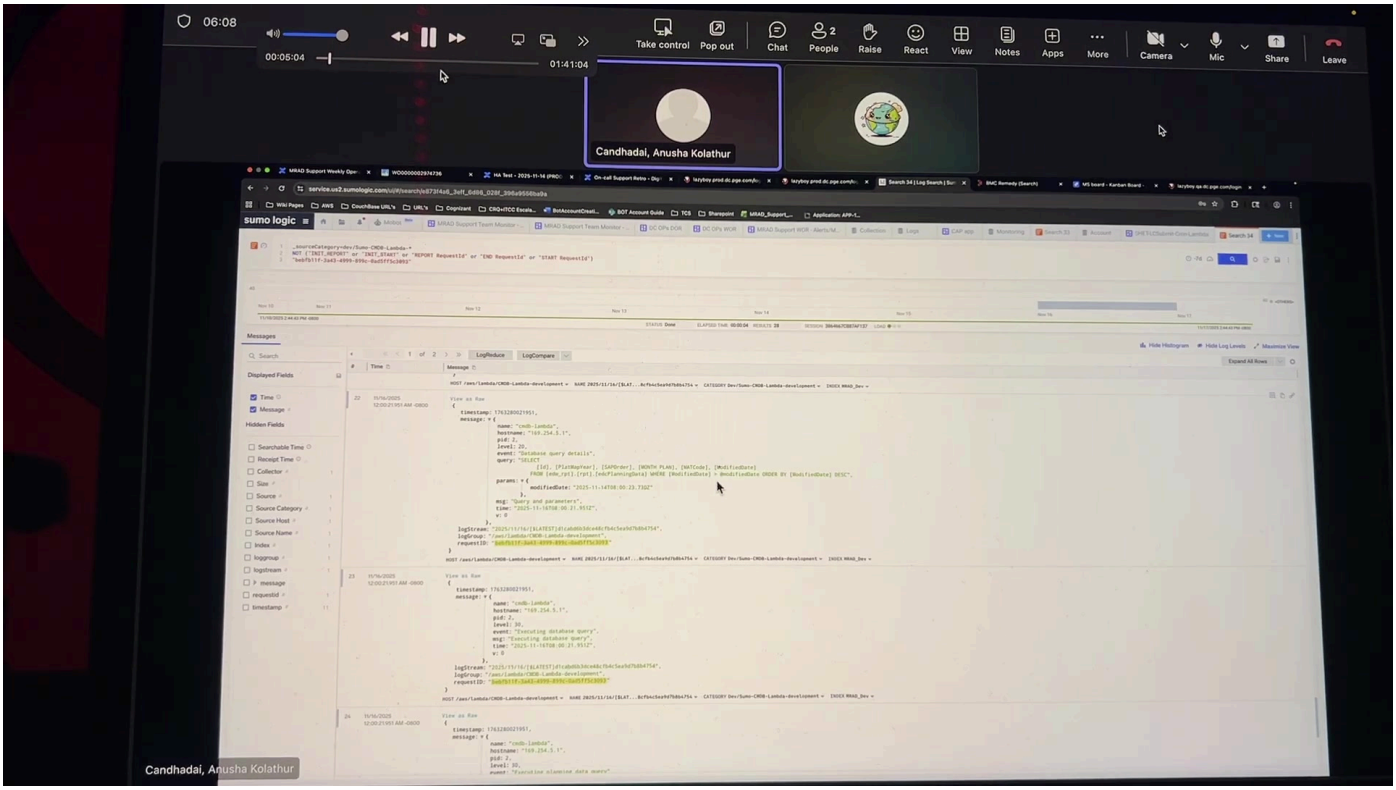


1. **Pull Logs by Single Thread**: For real-time running services, start pulling logs based on a unique identifier. [16]
2. **Identify Unique Transaction Value**: Look for a unique value like a Transaction ID, Document ID, or Request ID. [17]
3. **Identify Available Identifier**: In this log set, there is no Transaction ID visible at a higher level. [18]
4. **Use Request ID**: The available unique identifier is the **Request ID** field. [21]
5. **Request ID Function**: This ID uniquely identifies every transaction the lambda processes. [23]
6. **Filter by Specific Request ID**: Filter the logs for one specific Request ID to isolate its flow. [25]
7. **Count Specific Logs**: For the isolated transaction, there are around 29 specific logs. [26]

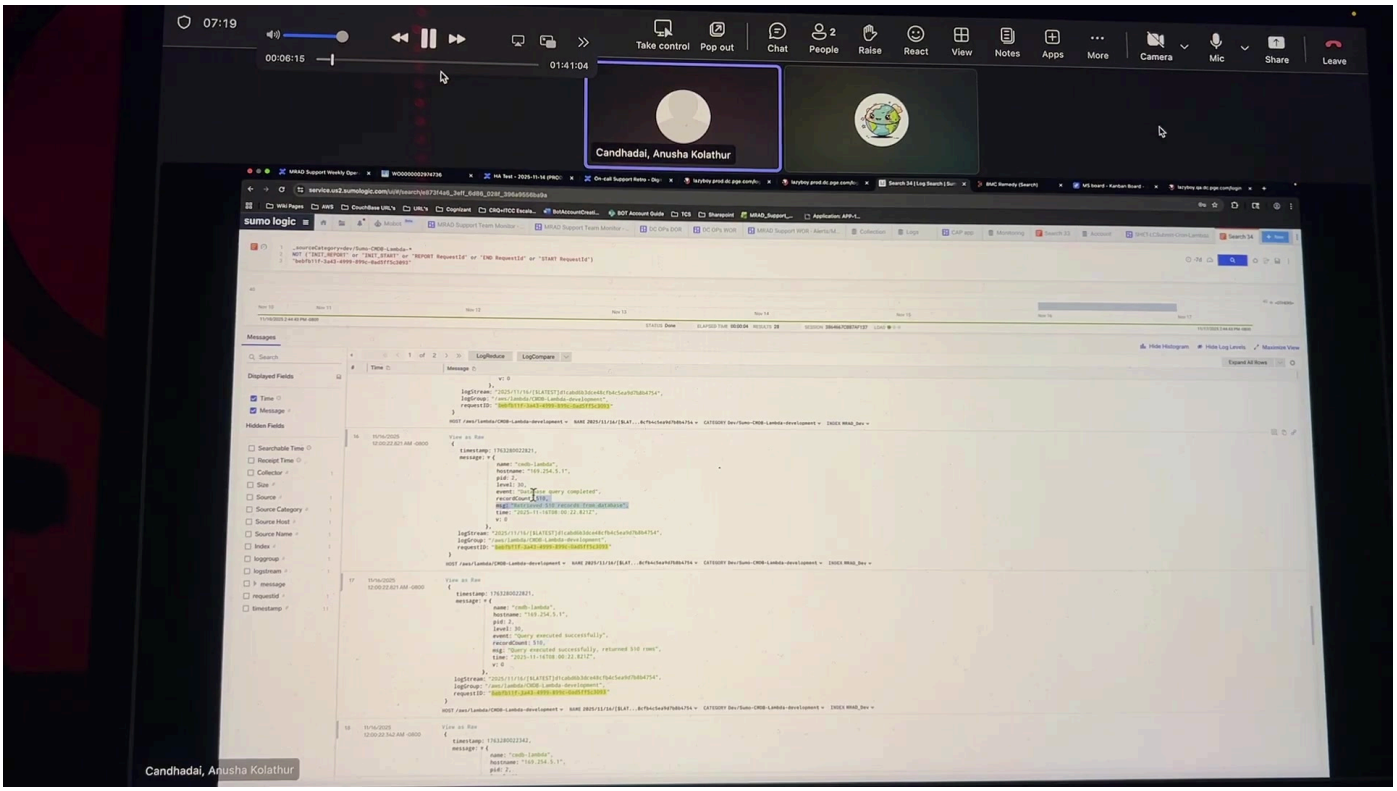


1. **Start Reading Log Messages**: Begin reading through the log messages to understand the service actions. [28]
2. **Filter Start Message**: Filter out the "start request ID" log as it is not needed. [29]
3. **Understand Each Log**: The job is to understand what each log tells you about the service. [31]
4. **Read Logs Backwards**: Always start reading from the end (bottom) of the logs, not the beginning. [34]
5. **Lambda Initialization**: The flow starts at the bottom with the lambda being initialized. [37]
6. **Parameter Retrieval**: The next log shows retrieving a parameter from the parameter store. [39]
7. **Database Query Start**: The service begins querying the database for planning data. [44]
8. **Using Last Updated Timestamp**: Data fetching uses the **last updated timestamp** stored in the database. [47]
9. **Data Fetching Scope**: All data since this last updated timestamp is being fetched. [48]
10. **Unnecessary Query Logs**: Logs showing the query execution details are unnecessary for monitoring. [49]

11. **Summary of Initial Steps**: The service retrieves a parameter (date) and queries the database for records newer than that date. [57]



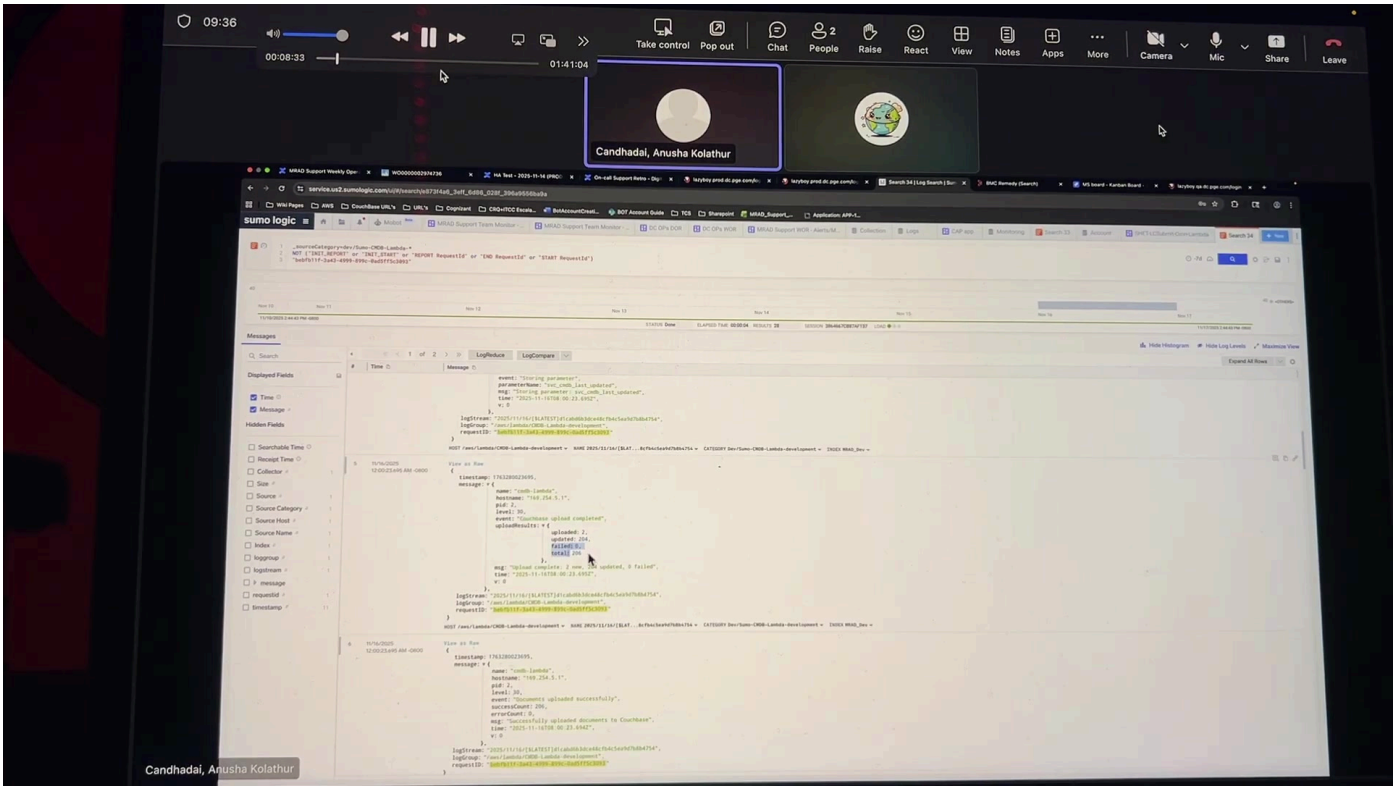
1. **Database Connection Pool Creation**: The next important log shows the creation of the database connection pool. [58]
2. **Fetching Credentials**: Another parameter retrieval occurs for the database **username and password** from the parameter store. [59]
3. **Connection Success**: "Database pool creation successful" confirms the database connection is established. [62]
4. **Executing the Query**: The service executes the query using the fetched date range. [63]
5. **Record Count Retrieved**: The query execution successfully returns a **record count** (510 records this time). [66]
6. **Query Completion Logs**: Logs confirm query execution completion, though some are duplicates. [68]



1. **Starting Couchbase Upload**: The next phase involves starting the upload to Couchbase. [72]
2. **Lambda's Two Aspects**: The lambda fetches data from SQL Server and updates/uploads documents in Couchbase. [73]
3. **Update or Create Logic**: If data exists, documents are updated; if not, they are created. [76]
4. **Records Uploaded**: The log confirms uploading 510 records to Couchbase. [79]
5. **Data Formatting**: Logs show *processing planning data* before upload, indicating data formatting. [80]
6. **Duplicate Removal**: Duplicate removal occurs, showing original count, unique count, and duplicate count. [81]
7. **Batch Processing**: The service processes records in batches, with a maximum size of 500. [84]
8. **Ignoring HTTP Client Log**: The "HTTP client" log is unnecessary and should be ignored. [89]
9. **Batch Check Results**: The batch check shows 204 existing records and 2 new records to be created. [92]

10. **Document Preparation**: Logs confirm documents are prepared for upload, showing total, new, and updated docs counts. [95]

11. **Successful Upload**: Logs confirm documents uploaded successfully, and **Couchbase upload completed**. [99]



1. **Updating Parameter Store**: Upon successful upload, the last fetch timestamp must be updated in the parameter store. [100]

2. **Next Run Preparation**: Updating the timestamp ensures the next lambda run picks up records from this specific time. [101]

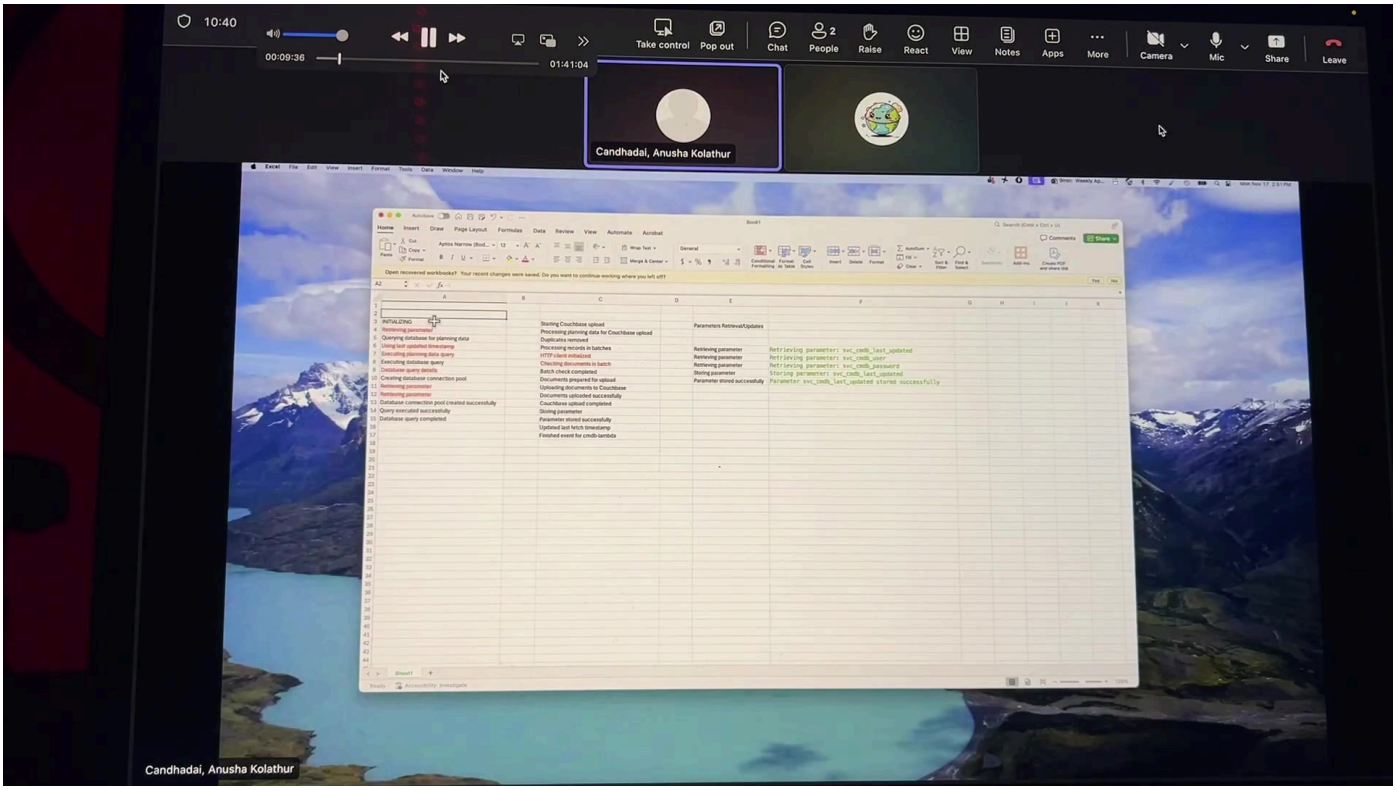
3. **Lambda Finished**: The final event confirms the lambda finished its execution. [101]

4. **Overall Lambda Job**: The lambda fetches data from the database based on a date and uploads it to Couchbase. [102]

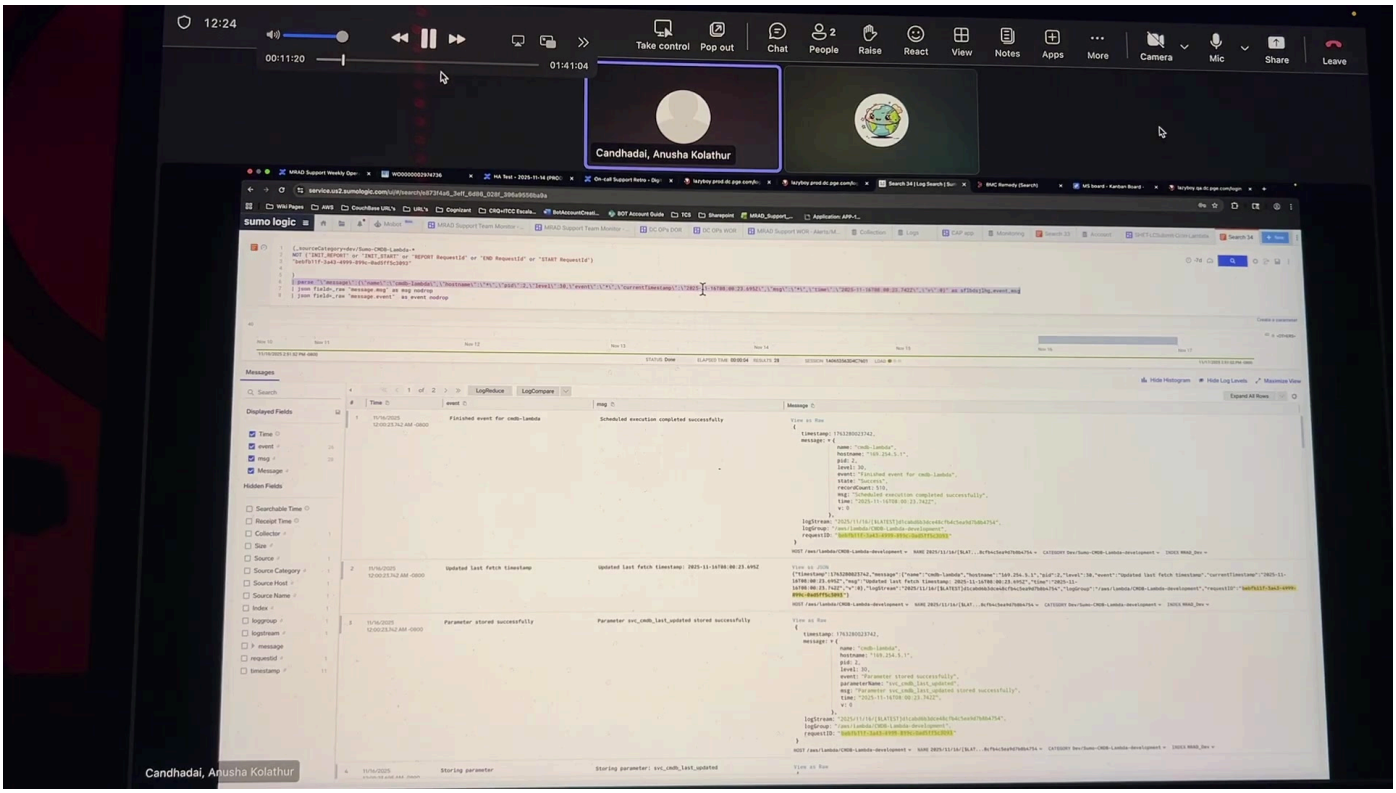
5. **Focus on Key Aspects**: The job involves two main aspects: the **database side** and the **Couchbase side**. [104]

2. Parsing Fields for Monitoring Visualization [106]

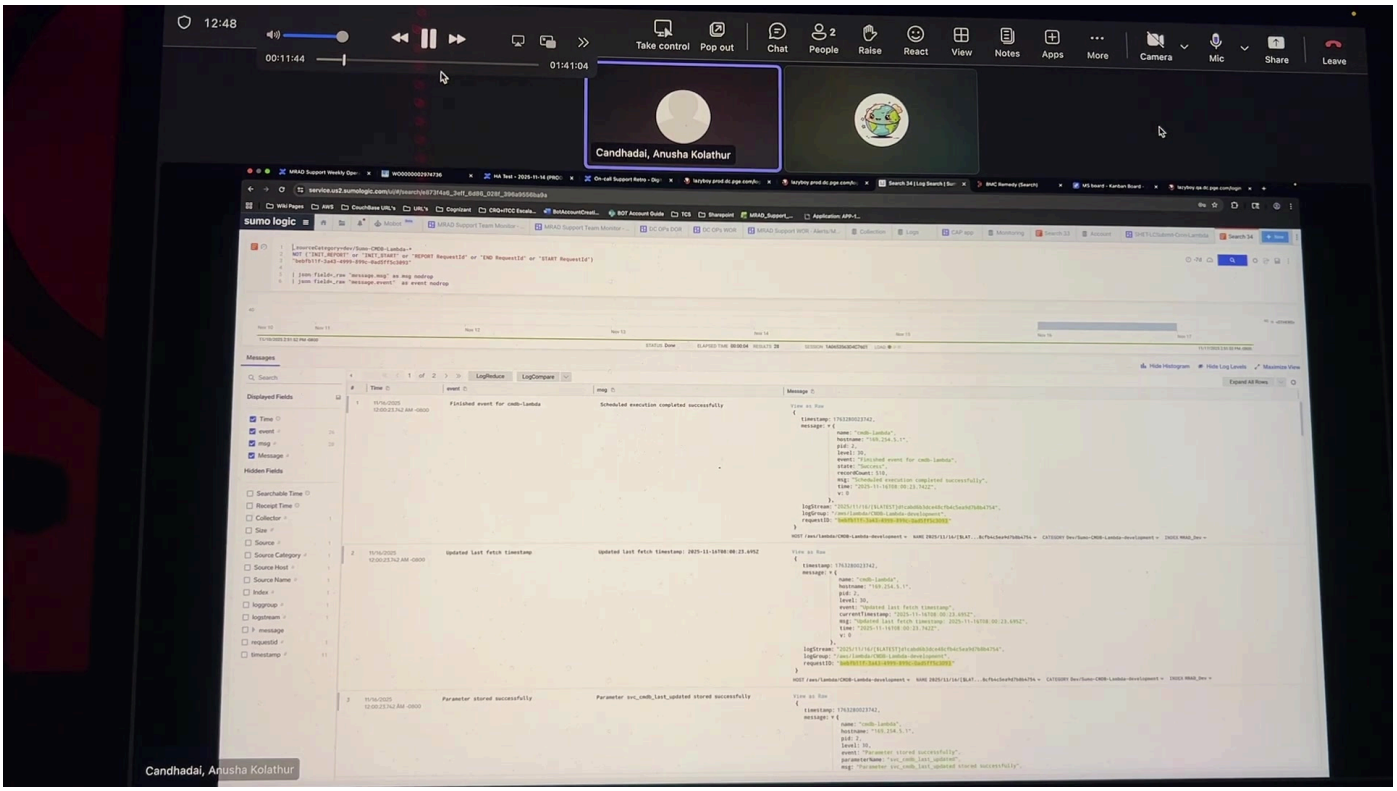
00:09:28 (3 min)



1. **Identify Important Fields**: The most necessary fields for monitoring flow are **event** and **message**. [106]
2. **Information Provided**: These two fields provide all necessary information about the lambda's flow steps. [107]
3. **Parsing Selected Key**: Use the "click and parse selected key" function on a field like *event*. [108]
4. **Raw Parsing Method**: Raw parsing involves clicking "view as raw" and then using "extract value" for specific patterns. [109]
5. **Parsing Event and Message**: Parse both the *event* field and the *message* field separately. [114]
6. **Raw Log Parsing Output**: Clicking submit performs raw log parsing based on the specific pattern. [116]
7. **Raw Parsing Applicability**: This raw parsing method is generally used for sync gateway logs, not this JSON-configured log. [118]



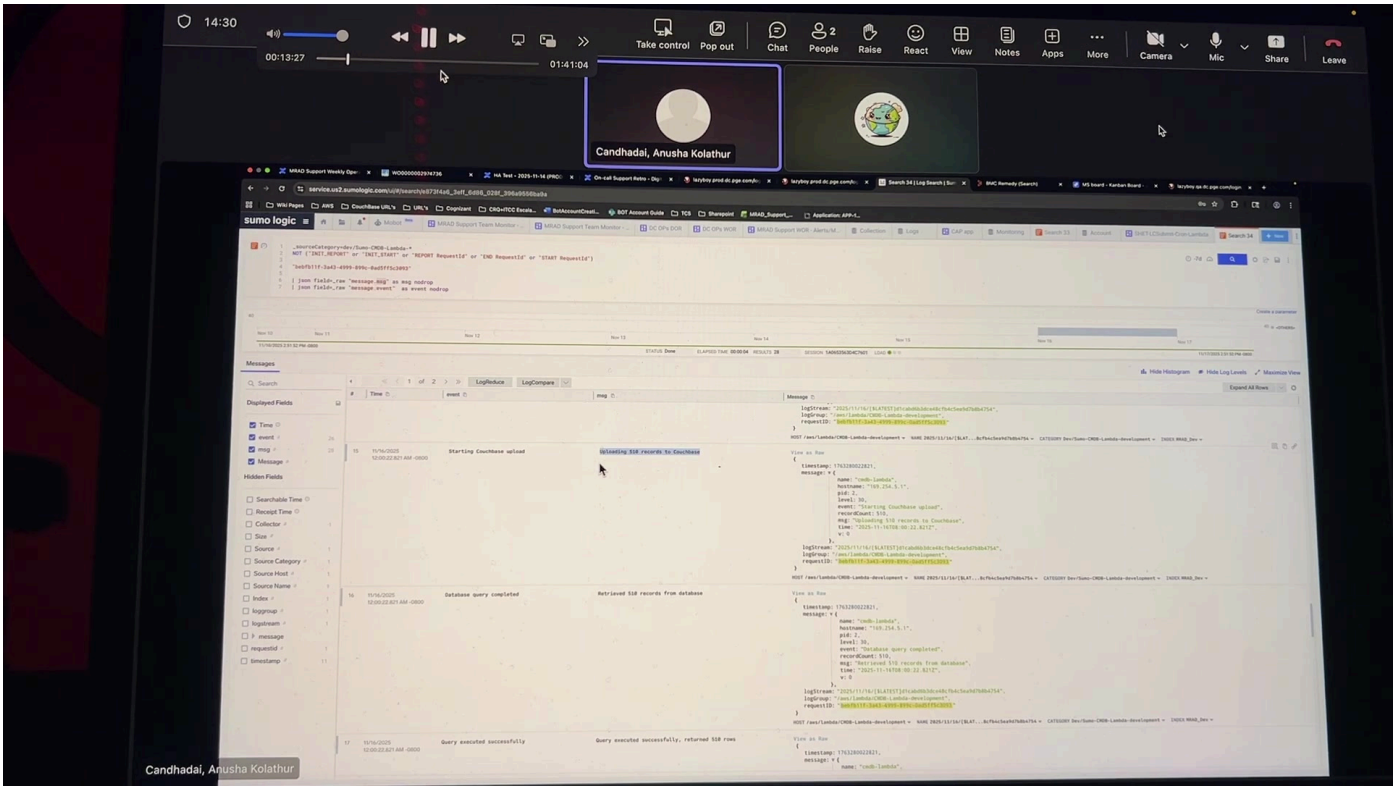
1. **Raw Log Format Definition**: Raw log parsing means making logs available in a raw format (`_raw`) for parsing. [122]
2. **Current Log Format**: This specific service's logs are *beautifully configured as JSON*. [123]
3. **JSON Parsing Preferred**: Therefore, JSON parsing will be used instead of the raw parsing method shown. [123]



1. **Reiterate Filtering**: Filter out logs that are not necessary to see. [126]
2. **Use "Not" Filter**: Place "not" in front of patterns to exclude them from the consumer view. [128]
3. **Query by Document ID**: If available, query based on Document ID or Doc ID. [129]
4. **Doc Type Context**: There is no mention of a new doc type for querying in this log set. [131]
5. **Focus on Available Info**: The current focus is figuring out what to do with the logs available for this Lambda. [136]
6. **Main Informative Fields**: The two main events providing initial information are **event** and **message**. [137]
7. **Message Field Purpose**: The message field explains the event if the event name alone is unclear. [140]
8. **Contextual Understanding**: The message provides context, like explaining that "starting Couchbase upload" means uploading specific records. [144]
9. **Initial Level Analysis**: At an initial level, both event and message fields are parsed for full context. [148]

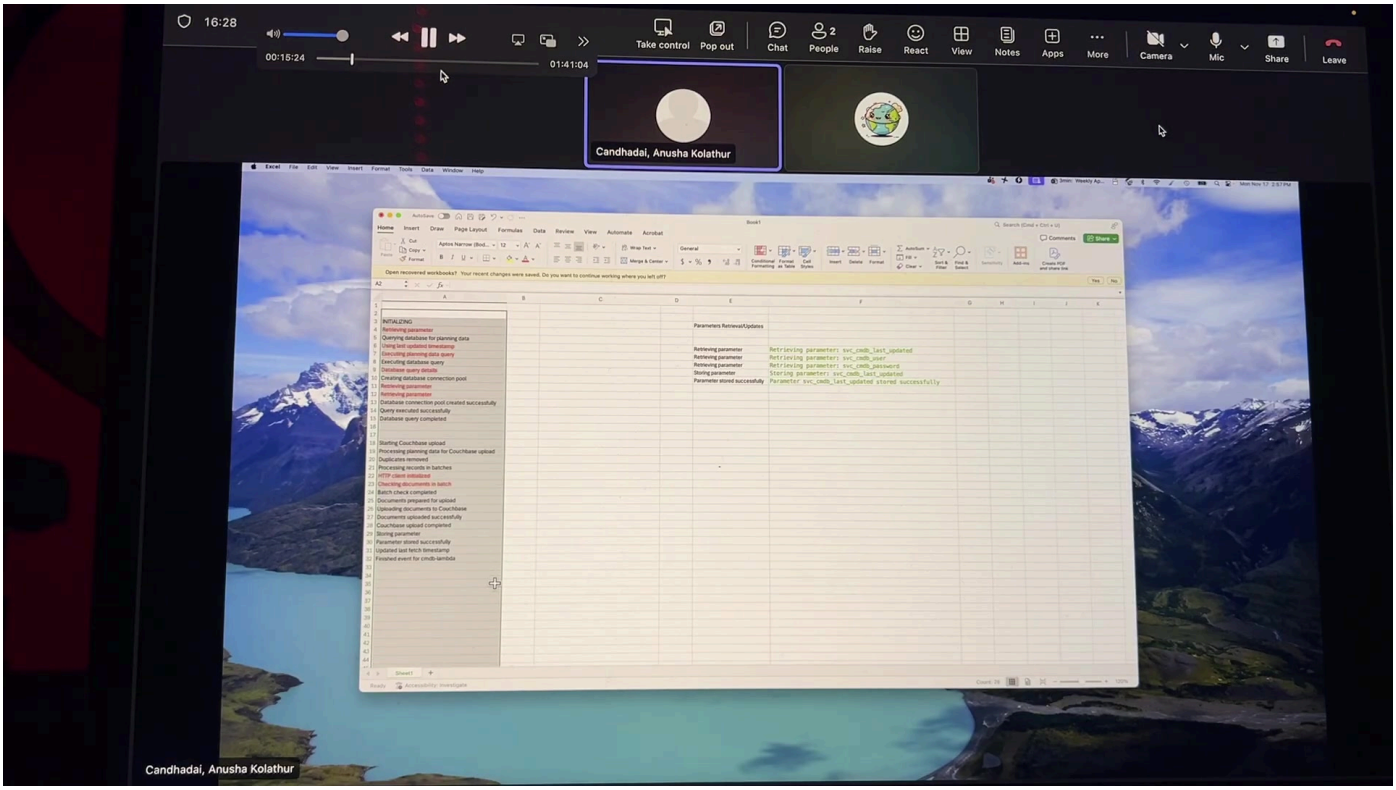
3. Categorizing and Selecting Events for Dashboard Panels [149]

00:13:18 (8 min)



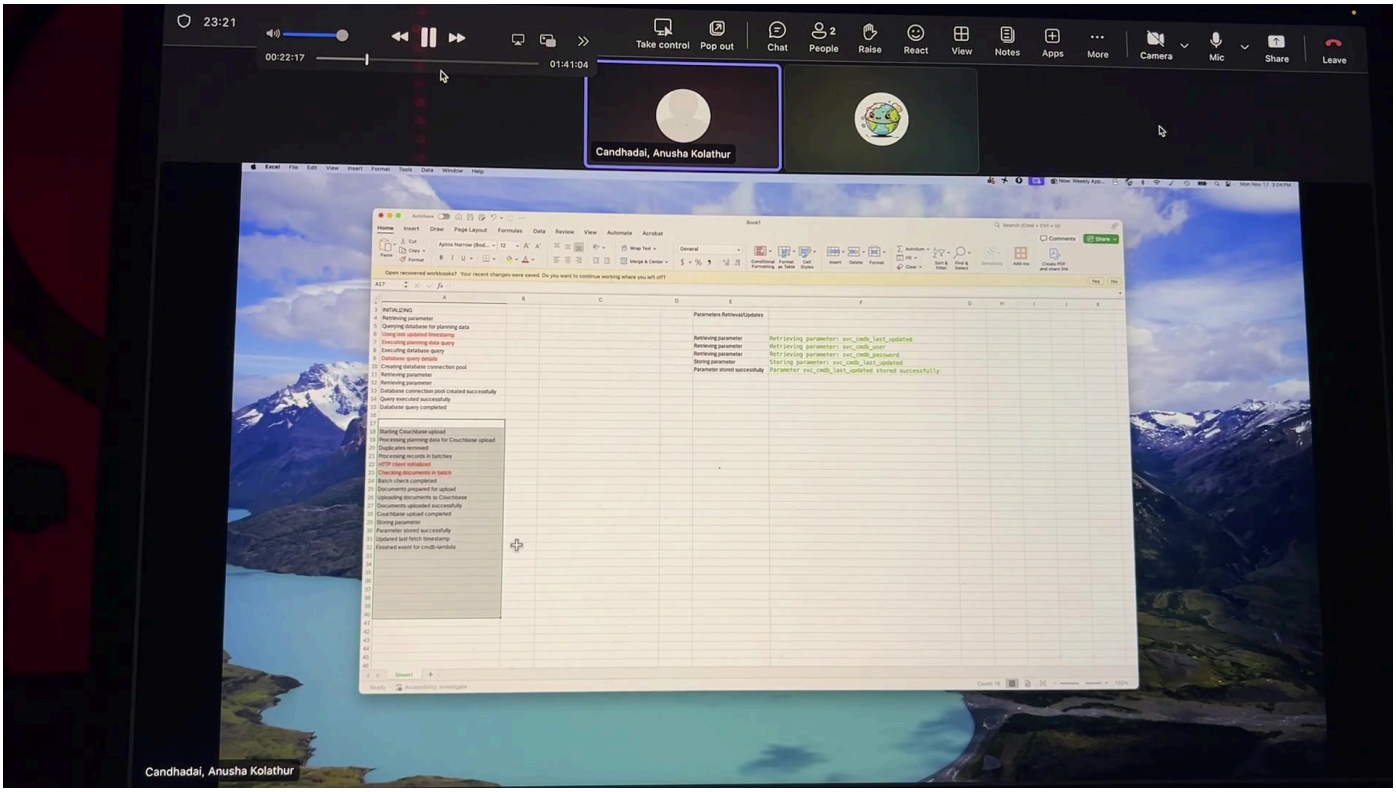
1. **Personal Approach**: The speaker's practice is to put all events into a sheet for easier review. [150]
2. **Two Categories**: Events are split into two categories: **database side** and **Couchbase side**. [152]
3. **Visualization Comfort**: This method is preferred over scrolling up and down in Sumo for a whole picture. [154]
4. **Event Source**: All listed items (initializing, retrieving parameters) are derived from the *event* field in Sumo. [158]
5. **Separation Rationale**: The lambda handles two tasks concurrently: fetching from the database and uploading to Couchbase. [174]
6. **Sorting Purpose**: The separation allows for potentially creating separate panels for the database and Couchbase work. [179]
7. **Event Importance for Monitoring**: The **event** field is crucial because panels are framed around it. [181]

8. **Initial Visualization**: The first step is visualizing *all* events without exclusions to decide what to keep. [277]



1. **Dashboard Goal**: A dashboard must allow quick assessment (in one minute) of whether the service is good or not. [183]
2. **Exclusion Necessity**: Logging all 29 events would make the panel look clumsy, requiring filtering. [185]
3. **Initializing**: **Keep** this event; it confirms the lambda has started. [186]
4. **Retrieving Parameter (Date)**: **Keep** this; main values are stored in parameters. [188]
5. **Querying Database for Planning Data**: **Keep** this; it shows the query building process has started. [190]
6. **Using Last Updated Timestamp**: **Remove** this; intricate details are not needed on the panel. [193]
7. **Executing Planning Data Query**: **Remove** this; it shows the query and modified date, which is not panel-worthy. [196]
8. **Executing Database Query**: **Keep** this; it makes sense when matched with the successful execution log. [209]

9. **Database Query Details**: **Ignore** this; the query details are not needed on the panel. [212]
10. **Creating Database Connection Pool**: **Keep** this; establishing the connection is a very important event to show. [215]
11. **Retrieving Parameter (Username/Password)**: **Keep** this; these are credentials fetched from the parameter store. [219]
12. **Database Pool Creation Successful**: **Keep** this; it confirms the connection was successful. [220]
13. **Query Executed Successfully**: **Keep** this; it confirms database query execution success. [222]
14. **Database Query Completed**: Keep this, although it is similar to the previous success log. [225]
15. **Database Section Conclusion**: The database part concludes after "database query completed." [229]

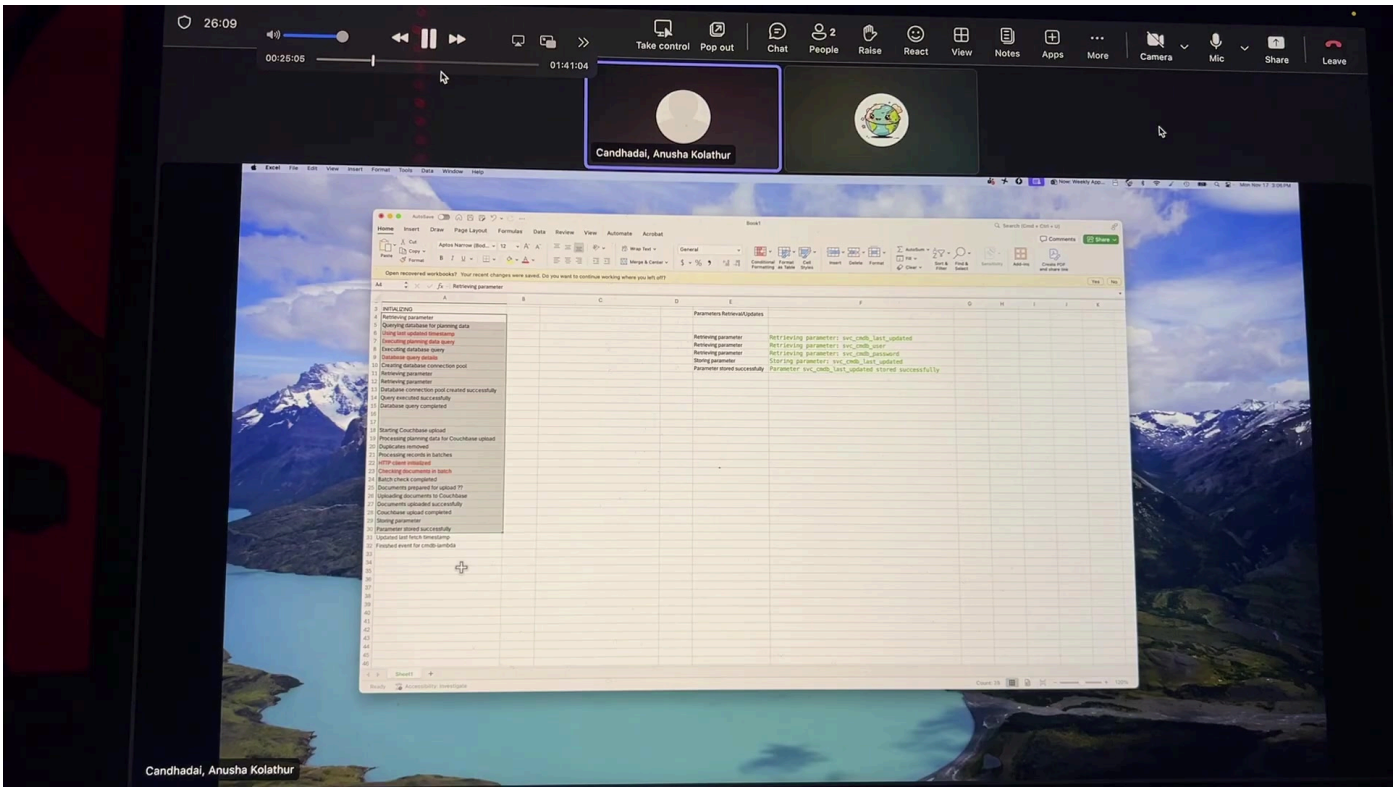


1. **Starting Couchbase Upload**: **Keep** this event as it marks the start of the upload phase. [279]

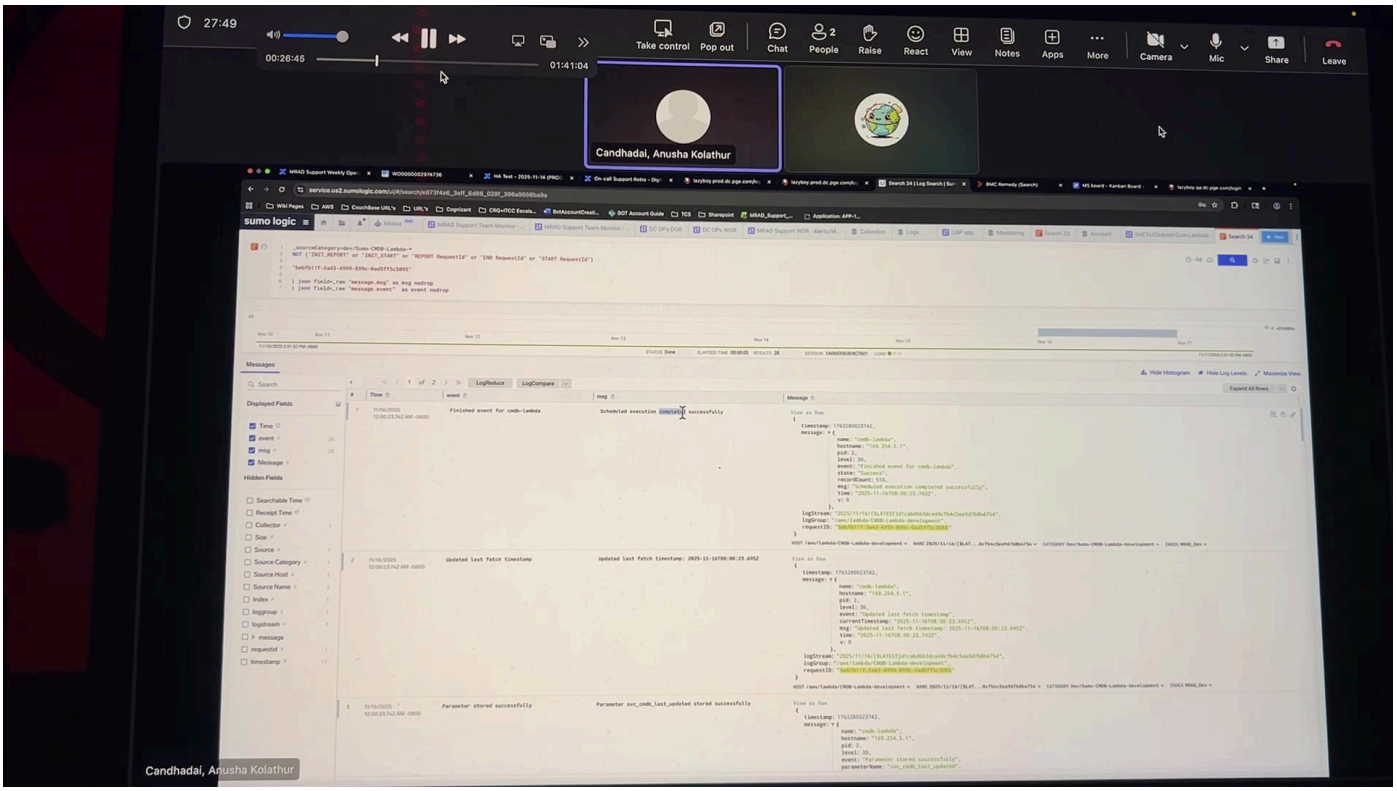
2. **Processing Planning Data**: **Keep** this; it shows checking how many records have come in. [281]
3. **Duplicates Removed**: **Keep** this; duplicate removal is a necessary step. [284]
4. **Processing Records in Batches**: **Keep** this; it reminds that processing uses 500 records per batch. [287]
5. **HTTP Client Initialized**: **Ignore** this; it relates to the backend call and is unnecessary. [290]
6. **Checking Documents in Batch**: **Ignore** this; it is an extra log showing the count (206 documents) already known. [295]
7. **Batch Check Completed**: **Keep** this; it confirms 204 existing and 2 new records were found. [300]
8. **Documents Prepared/Uploading**: **Keep** these events as they are very important. [302]
9. **New/Updated Doc Counts**: Logs show 2 new docs and 204 updated docs. [305]
10. **Uploading/Documents Uploaded Successfully**: **Keep** these logs confirming the upload action. [310]
11. **Couchbase Upload Successful**: **Keep** this final wrapper log for the upload phase. [314]
12. **Storing Parameter Successful**: **Keep** this; it confirms the last fetch timestamp was updated successfully. [318]
13. **Finished Event for CMDB Lambda**: This is the final wrapper event. [325]
14. **Most Important Event**: Ideally, **scheduled execution completed successfully with record count state success** is the only event that matters. [329]
15. **Exclusion Summary**: Events marked in red during analysis are excluded from the primary monitoring panel. [334]

4. Designing the Final Monitoring Dashboard Panel [338]

00:24:57 (5 min)



1. **Visualization Goal**: Decide how the dashboard should present the information. [338]
2. **Two Dashboard Approaches**: Dashboards can show every detail (success/fail) or summarize the core function. [339]
3. **Lambda Core Function**: This lambda fetches data from the database and updates Couchbase. [341]
4. **Minimalist Success View**: The job is done if we show only the **event finished** log as successful, including the record count. [344]
5. **Run Frequency Consideration**: Since it runs only once a day for now, a simple success indicator is sufficient. [346]
6. **Internal Retries**: Internal retries for failures (like 404s) are handled by the lambda itself and are not the focus here. [349]
7. **Sufficient Monitoring**: Showing only whether the Couchbase update was successful is enough. [354]



1. **Start Querying**: Begin writing queries to form the panel for the dashboard. [356]
2. **Focus Event**: Start by querying only for the **finished event for CMDB Lambda**. [358]
3. **Handling Failure Message**: The "finished" part cannot be used yet, as failure logging format is unknown. [363]
4. **Selected Fields for Panel**: Use the **event** field, the **message** field, and the **record count**. [364]
5. **Formatting the Count**: The count field will be renamed for better display (e.g., `_count` becomes a descriptive name). [366]
6. **Display Format**: Display the results on a daily basis using a specific date format. [368]
7. **Date Formatting**: The required date format is **MMDD** (Month/Day). [377]
8. **Example Successful Run**: On 11/16, the event was successful, processing around **510 records**. [380]
9. **Daily Status Check**: This view allows quick confirmation if the CMDB Lambda ran successfully today. [381]
10. **Viewing Multiple Runs**: Filtering for seven days shows multiple runs, including successful runs with **zero records to process**. [384]

11. **Final Presentation**: The final step is formatting this data to look clean on the dashboard. [389]