dash5 (cmdb lambda use case).mp4_18 | Summarize Videos, Audio, PDF & Websites

Table of Contents

Master **CMDB lambda visualization** using the **funneling way** approach for clear data integrity checks. Learn to configure dashboards with **stacked column charts** and **override line graphs** to instantly verify record counts against total inputs. This guide shows you how to **parametrize** dashboards for easy environment toggling, ensuring robust troubleshooting across Dev, QA, and Prod.

This summary details the creation of a CMDB lambda visualization dashboard. The approach uses a **funneling way** for data integrity checks. [1]

Extracting capture image...

1. **Visualization Components** [2]
    1. Total records are shown as a **line graph** (override line graph). [2]
    2. Other data points are shown as **stacked column charts**. [4]

2. **The Funneling Way Definition** [5]

    1. The top of the funnel must be the **total number of records** input. [6]
    2. For yesterday's sample, the total records were 510. [7]
    3. Unique, duplicate, and failed records must sum up to the top count. [9]
    4. The combined record count should **not exceed** the total records. [10]
    5. This confirms the lambda code is functioning correctly. [13]
    6. This representation ensures all counts are enclosed within the top of the funnel. [13]

3. **Attempt, Success, and Error Events Panel** [17]

    1. Another dashboard panel categorizes events into three types. [18]
    2. These categories are: **attempt event**, **success event**, and **error event**. [20]
    3. *Attempt* is the starting point of the transaction. [21]
    4. Any attempt should result in either a success or a failure. [23]

4. **Interpreting the Funnel Graph** [55]

    1. Attempt events are shown in green dots (the top). [55]
    2. Success and error counts must lie within the attempt count. [59]
    3. Ten attempt logs mean ten parallel transactions occurred. [61]
    4. The tip of the graph should be *tight* with no gaps. [63]
    5. Gaps indicate that some transactions might be missed. [66]
    6. Missing data is identified when gaps appear in the visualization. [70]
    7. Code troubleshooting is the second level after identifying gaps. [71]

5. **Handling Retry Logic** [80]
    1. If retry logic exists, the graph might show extra data above total records. [80]
    2. Example: 10 total records, 5 unique and 5 duplicate counts (5+5=10). [84]
    3. Retries occur when uploading unique counts fails repeatedly. [88]
    4. Currently, there is no retry logic implemented. [92]
    5. This representation proves data sent was sent to Couchbase. [93]

Extracting capture image...

1. **Query Execution** [98]
    1. A query parses all counts and performs a sum across counts by time slice. [98]
    2. A 30-minute time slice was used for this specific example. [99]
    3. The query executes over the past seven days when viewed. [100]

2. **Time Slice Functionality** [102]
    1. The time slice chops data into defined chunks for viewing. [103]
    2. This specific service runs once daily, so one-day slice is used. [104]
    3. For highly sensitive services like *polar* or *router*, slices are shorter (e.g., five minutes). [107]
    4. A time slice groups and splits data for intervals like five minutes, one hour, or one day. [112]

3. **Filling Missing Time Slices** [114]
    1. The command *"fill missing time slice"* is used for busy services. [114]
    2. Without this, the graph appears *erratic* if long gaps exist. [116]
    3. This line improves visualization by filling gaps where no logs exist. [118]

Extracting capture image...

1. **Naming and General Settings** [125]
    1. A name must be provided for the panel in the General section. [125]

2. **Display Settings: Chart Type** [128]
    1. Cosmetic changes are made in the **Display** section. [129]
    2. Initially, everything is set to a **column chart** (time series). [130]
    3. With default settings, all charts stack on top of each other. [134]
    4. The stacking is controlled by the **display type as stacked** setting. [137]
    5. Stacked charts are used for the funneling method. [139]

3. **Creating the Funnel Override** [140]
    1. The **total record** must be set at the top of the funnel. [140]
    2. The **overwrite column** is used to separate the total record line. [144]
    3. Select *total records* and choose **another line graph** as the chart type. [147]
    4. Cosmetic line adjustments include *dotted line*, *marker type*, and *line thickness*. [149]
    5. Increasing line thickness makes the line appear higher on the graph. [152]
    6. This override creates the desired funneling effect. [154]

4. **Finalizing the Override** [155]
    1. You can choose any chart type that works for the override. [155]
    2. The effect is applied by running the query and updating it with the override. [156]

Extracting capture image...

1. **Implementing Environmental Variables** [161]
    1. The next step is to **parametrize** the dashboard using environmental variables. [162]
    2. Add variables via the **filter** section. [164]
    3. For the environment variable (*env*), use a **custom list** for specified values. [165]
    4. Possible values are *dev*, *product*, and *QA*. [167]
    5. The default value is set to *dev* as it has the most data. [167]

2. **Passing Parameters in Queries** [168]
    1. Parameters must be passed in *every panel* for toggling to work. [168]
    2. Parameterization uses **two curly braces** and is *case-sensitive*. [168]
    3. This allows easy toggling between environments for debugging issues. [171]

3. **Parameterizing Time Slice** [179]
    1. The time slice parameter is added similarly using a **custom list**. [181]
    2. The default time slice is set to *one day* for this daily running service. [183]
    3. The parameter is inserted into the query where the time slice is defined. [185]
    4. Using *one day* shows the graph in a better, expanded form. [187]

4. **Verification** [192]
    1. After parameterization, you can switch between *dev*, *QA*, and *prod*. [192]
    2. In an ideal case, the panels should load data for the selected environment. [195]
    3. Every shift in environment should update the displayed data correctly. [198]

Extracting capture image...

1. **Alerts and Monitoring** [202]
    1. The next step after dashboard creation is adding **alerts or monitors**. [203]
    2. Alert scheduling depends on whether **retry logic** is implemented. [204]
    3. Alerts must be scheduled based on the lambda's actual run frequency. [204]
    4. For a service running once daily at 6:00 AM, an alert can check for errors in the past hour at 6:30 AM. [207]

2. **Dashboard Completion Status** [209]
    1. The team will inform Jonathan that the high-level dashboard is complete. [209]
    2. Final work depends on business confirmation regarding lambda run times or changes. [219]
    3. This dashboard is a basic, parametrized visualization for easy toggling. [221]
    4. Graphs make events and errors easier to interpret than tables. [223]
    5. Graphs and events/errors are part of every dashboard created. [226]
    6. The visual appeal helps confirm infrastructure is moving fine quickly. [228]

3. **On-Call Tips** [234]
    1. For any on-call alert, open the alert to see the query. [234]
    2. Understanding the query explains why the alert is complaining. [235]
    3. Most on-call alerts require opening the query for investigation. [235]

>