**Workflow for to send email with inline table**

**Experimented and Tested By: Jawahar, Sneha**  
**Documented by: Jawahar, Rajesh** **Date: September 1, 2025**

The following document outlines the step-by-step process for creating a workflow in Dynatrace, a platform used for monitoring and observability. This workflow involves setting up a trigger, defining a task with a Dynatrace Query Language (DQL) query, processing data with JavaScript to generate an in-line table, and sending an email notification with the results.

**Step-by-Step Workflow Creation**

**Step 1: Initiate Workflow Creation**

* **Action**: Navigate to the Dynatrace interface, search for the "Workflow" feature, and click on "Create Workflow."
* **Purpose**: This action initiates the process of building a new workflow to automate tasks within Dynatrace.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2: Name the Workflow and Select a Trigger**

* **Action**: On the workflow creation page, provide a name for the workflow and click on "Select Trigger."
* **Explanation**: Naming the workflow helps identify it within Dynatrace. Selecting a trigger determines what initiates the workflow (e.g., a schedule, event, or manual activation).

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step 3: Configure the Trigger and Add a Task**

* **Action**:
  + Select the trigger type. For this demo, the trigger is set to "On Demand," meaning the workflow runs manually when initiated.
  + Add a task to the workflow and define the action it performs.
* **Explanation**:
  + The trigger type dictates when and how the workflow executes. "On Demand" is suitable for testing or manual execution.
  + A task is a specific action, such as querying data. Here, the task involves running a DQL query.

**Step 4: Provide the DQL Query**

* **Action**: Input the DQL (Dynatrace Query Language) query for the task.
* **Explanation**:
  + DQL is used to query data from Dynatrace, such as metrics, logs, or events.
  + The specific DQL query retrieves data (e.g., client version, hostname, request ID) to be processed in the next step. (Note: The exact query is not provided in the original document.)
  + Example: A DQL query might be fetch logs | fields clientVersion, hostName, requestId.
* **Purpose**: The DQL query defines the data the workflow processes.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 5: Run JavaScript to Generate an In-Line Table**

* **Action**: Select the "Run JavaScript" action and paste the following code to process the DQL query results and format them into a markdown table.

A screenshot of a computer

AI-generated content may be incorrect.

Java Script:  
import { result } from '@dynatrace-sdk/automation-utils';

export default async function () {  
    const dqlResult = await result('dql\_query');// "dql\_query" is the DQL task neme we need to mention here to read the data   
    const records = dqlResult.records || [];

    if (records.length === 0) {  
        return 'No data found for the specified time range.';  
    }

    let table = '| Client Version|       Host Name           | Request ID |\n';  
    table += '| --- -------------|-------------------- | --- |\n';

    records.forEach(row => {  
        table += `| ${row.clientVersion || ''} | ${row.hostName || ''} | ${row.requestId || ''} |\n`;  
    });

    return table;  
}

* **Code Explanation**:
  + Imports the result function to access DQL query output.
  + Retrieves results from the DQL task named dql\_query (replace with the actual task name).
  + Checks if data exists; if not, returns "No data found for the specified time range."
  + Creates a markdown table with headers for Client Version, Host Name, and Request ID.
  + Iterates through records, appending each as a table row, handling missing values with empty strings.
  + Returns the formatted table.
* **Purpose**: Transforms DQL query results into a human-readable markdown table for notifications or reports.

**Step 6: Send Email Notification**

* **Action:** Configure the email notification stage in the Dynatrace workflow to send the processed data from the previous "Run custom JavaScript code" task to the specified recipient.
* **Explanation:** 
  + **Interface Overview:** The image shows the Dynatrace workflow editor with the "send\_email\_1" action selected. This action follows the "Inline\_table" task, which generates a table using JavaScript
  + **Recipient**: The email is addressed to "[trymeifyouwant@gmail.com](mailto:trymeifyouwant@gmail.com)". Multiple email addresses can be added, separated by semicolons (;), and static addresses can be mixed with dynamic expressions.
  + **Cc and Bcc**: Optional fields for carbon copy (Cc) and blind carbon copy (Bcc) are available but left blank in this example.
  + **Subject**: The subject line is set to "Test the work Flow".
  + **Message Content**: The email body includes:
  + A title "# UXL Log Report".
  + A description: "Below are the latest logs from Dynatrace (source: uxl):".
  + A placeholder run\_javascript\_1 that dynamically inserts the table generated by the JavaScript code.
  + A link: "[View in Dynatrace](https://your-dynatrace-url.com)" (replace with the actual Dynatrace URL).

A screenshot of a computer

AI-generated content may be incorrect.

**Jinja Expression**  
The "Input" section of a Dynatrace workflow’s email action, where the message content is defined. Here’s how Jinja expressions are used to read the inline table data  
A screenshot of a computer

AI-generated content may be incorrect.

* **Jinja Expression in the Email Body**:
  + The line {{ result('run\_javascript\_1') }} is a Jinja expression that retrieves the output of the "run\_javascript\_1" JavaScript task.
  + In this case, the JavaScript code (as outlined in your earlier documentation) processes DQL query results and formats them into a markdown table. The result() function fetches this table, which is then inserted into the email body.
  + The email content includes a header ("# UXL Log Report"), a description ("Below are the latest logs from Dynatrace (source: uxl):"), the Jinja expression, and a link. When the workflow executes, {{ result('run\_javascript\_1') }} is replaced with the generated table

**How It Works**:

* The JavaScript task returns a string (the markdown table) as its output. This output is stored and can be accessed using the task’s name ("run\_javascript\_1") within Jinja expressions.
* The result() function in the Jinja expression acts as a placeholder that resolves to the JavaScript task’s output during workflow execution. This ensures the email dynamically includes the latest table data.

**Placement and Syntax**:

* The Jinja expression is embedded directly in the email body’s "Message" field, surrounded by double curly braces {{ }}.
* The task name inside result() must match the exact name of the JavaScript task in the workflow (e.g., "run\_javascript\_1"). This name is set when you configure the task in the workflow editor.