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 CSV file and sending an email notification with the CSV file.

**1. On demand trigger**

* **Type**: Trigger
* **Purpose**: Initiates the workflow manually or via an API.
* **Configuration**:
  + Triggered manually or with an API (as shown in the workflow image).
  + Connects to task-1 to start the data processing flow.

A screenshot of a computer

AI-generated content may be incorrect.

**2. task-1**

* **Type**: "Make use of Dynatrace Grail data in your workflow"
* **Purpose**: Fetches log data using a DQL query.

**DQL Query:  
fetch logs, from:"2025-09-03T15:18:00+05:30", to:"2025-09-03T15:20:00+05:30"**

**| filter log.source == "uxl"**

**| fields clientVersion, hostName, requestId**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Note**: Adjusted the time range to start at 03:20 PM IST (current time) and extend for 2 minutes to capture recent data. If no data is returned, test with a broader range (e.g., from: now-1h, to: now) in Dynatrace Data Explorer.

**3. email\_script**

* **Type**: "Run custom JavaScript code"
* **Purpose**: Processes the DQL results into a CSV, constructs a MIME email with the CSV attachment, and returns the base64-encoded email.
* **JavaScript Code** (based on your provided script with minor updates):

A screenshot of a computer

AI-generated content may be incorrect.

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

export default async function () {

const extractedData = [];

try {

const myResult = await result('task-1'); // dql task name should mention

console.log('The whole result object:', myResult.records);

if (myResult && Array.isArray(myResult.records)) {

myResult.records.forEach(record => {

if (record.clientVersion && record.hostName && record.requestId) {

extractedData.push({

clientVersion: record.clientVersion,

hostName: record.hostName,

requestId: record.requestId,

});

} else {

console.warn('Skipping record with missing properties:', record);

}

});

console.log('Extracted Data:', extractedData);

// Generate CSV content

let csvContent = 'clientVersion,hostName,requestId\n';

extractedData.forEach(record => {

const clientVersion = `"${String(record.clientVersion).replace(/"/g, '""')}"`;

const hostName = `"${String(record.hostName).replace(/"/g, '""')}"`;

const requestId = `"${String(record.requestId).replace(/"/g, '""')}"`;

csvContent += `${clientVersion},${hostName},${requestId}\n`;

});

console.log('Generated CSV:', csvContent);

// Base64 encode the CSV content for email attachment

const base64Csv = btoa(csvContent);

// Generate MIME email with CSV attachment

const boundary = 'boundary\_' + Math.random().toString(16).slice(2);

const mimeMessage = `To: catchme ifyoucan@gmail.com, willcatchyou@gmail.com\n` + // mention the email IDs

`From: sourcegmail@gmail.com\n` + // gmail source id created in the gmail

`Subject:Hello Folks Dynatrace Log Details with CSV Attachment\n` +

`Content-Type: multipart/mixed; boundary="${boundary}"\n\n` +

`--${boundary}\n` +

`Content-Type: text/plain; charset=utf-8\n\n` +

`Attached is the log details CSV file generated from Dynatrace.\n\n` +

`--${boundary}\n` +

`Content-Type: text/csv; name="log\_details.csv"\n` +

`Content-Disposition: attachment; filename="log\_details.csv"\n` +

`Content-Transfer-Encoding: base64\n\n` +

`${base64Csv}\n` +

`--${boundary}--`;

// Base64 encode the entire MIME message for Gmail API

const base64Encoded = btoa(mimeMessage.replace(/\n/g, '\r\n'));

console.log('Generated MIME Email:', mimeMessage);

console.log('Base64 Encoded Email:', base64Encoded);

// Return data for the HTTP Request task

return {

status: 'success',

extractedData,

base64Email: base64Encoded

};

} else {

console.log('No records found in the result.');

return {

status: 'no\_records',

extractedData: [],

base64Email: ''

};

}

} catch (error) {

console.error('Error fetching task result:', error);

return {

status: 'error',

extractedData: [],

base64Email: '',

error: error.message

};

}

}  
  
**Changes**:

* Updated the email subject and body to include the current date and time (03:18 PM IST on September 03, 2025).
* Kept the original recipients (catchme ifyoucan@gmail.com, willcatchyou@gmail.com) and sender (sourcegmail@gmail.com).
* Ensured the MIME message uses \r\n line endings, as required by RFC 2822 for Gmail API compatibility.

**3 Step-by-Step Guide to Set Up Gmail API in Google Cloud**

**1. Create a Google Cloud Project**

* **Access Google Cloud Console**:
  + Go to [console.cloud.google.com](https://console.cloud.google.com/).
  + Sign in with the Google account that will send the emails (e.g., yourgmail096@gmail.com).

A screenshot of a computer

AI-generated content may be incorrect.

* **Create a New Project**:
  + Click the project dropdown at the top left and select "New Project".
  + Enter a project name (e.g., Dynatrace-Gmail-Integration).
  + Click "Create". Once created, select the new project from the dropdown.

**2. Enable the Gmail API**

* **Navigate to APIs & Services**:
  + In the Google Cloud Console, go to **APIs & Services** > **Library**.
* **Search for Gmail API**:
  + Type "Gmail API" in the search bar and select it from the results.
* **Enable the API**:
  + Click **Enable**. This allows your project to use the Gmail API for sending emails.

A screenshot of a computer

AI-generated content may be incorrect.

**3. Configure OAuth 2.0 Consent Screen  
A screenshot of a computer

AI-generated content may be incorrect.**

* **Set Up Consent Screen**:
  + Go to **APIs & Services** > **OAuth consent screen**.
  + Choose **clients** (if you want to use it outside your organization) and click **Create**.

A screenshot of a computer

AI-generated content may be incorrect.

* **Fill in Required Information**:
  + **App name**: e.g., Dynatrace Gmail Workflow.
  + **User support email**: Select your email (e.g., yourgmail@gmail.com).
  + **Developer contact information**: Add your email.
  + Leave other fields as default unless specific requirements exist.

A screenshot of a computer

AI-generated content may be incorrect.

* **Clients**:
  + Click **Add or Remove Scopes**.
  + Search for .../auth/gmail.send and add it (this scope allows sending emails).
  + Save and continue.

A screenshot of a computer

AI-generated content may be incorrect.

* **Test Users**:
  + Add test users (e.g., add the sending email id where we are going to send ) if using an external consent screen.
  + Click **Save and Continue**, then **Back to Dashboard**.

A screenshot of a computer

AI-generated content may be incorrect.

**5. Generate an Access Token Using OAuth 2.0 Playground**

* **Access OAuth 2.0 Playground:** 
  + Go to [developers.google.com/oauthplayground](https://developers.google.com/oauthplayground)**.**
* **Configure the Playground:** 
  + Click the gear icon (Settings) and check Use your own OAuth credentials.
  + Enter the Client ID and Client Secret from your Google Cloud project (created as per the previous Gmail API setup instructions).
* **Select Scopes:** 
  + In the list of APIs, expand Gmail API v1 and check <https://www.googleapis.com/auth/gmail.send>.

A screenshot of a computer

AI-generated content may be incorrect.

* **Authorize APIs:** 
  + Click Authorize APIs.
  + Sign in with the Google account that will send the emails (e.g., yourmail096@gmail.com).
  + Grant permission when prompted

A screenshot of a computer

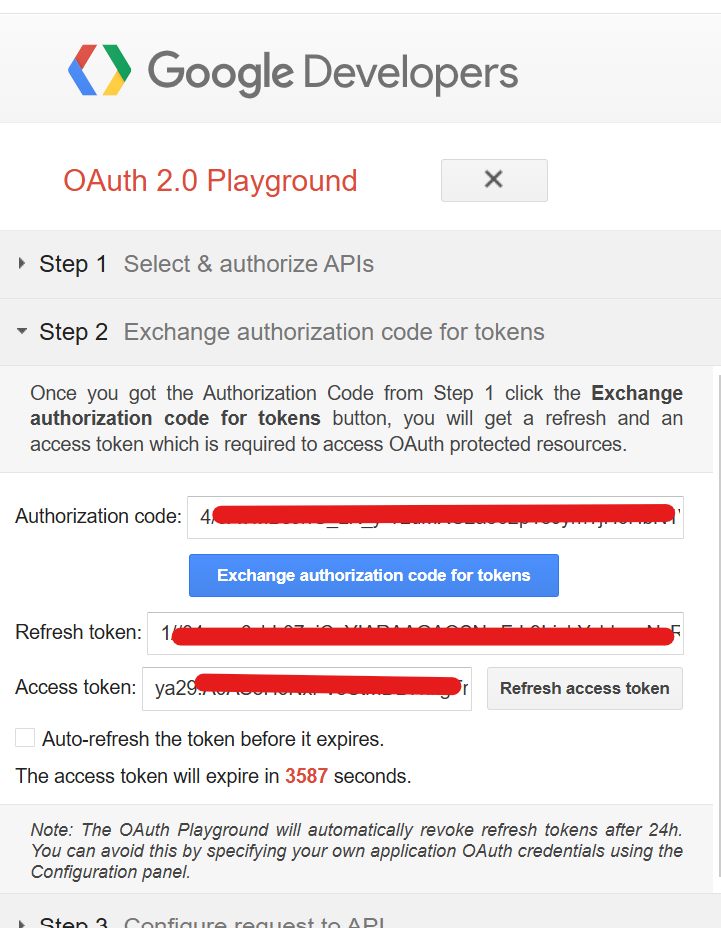
AI-generated content may be incorrect.

* **Exchange Authorization Code for Tokens:** 
  + After authorization, click Exchange authorization code for tokens.

A screenshot of a computer

AI-generated content may be incorrect.

* + Copy the Access token (valid for 1 hour, expiring around 04:58 PM IST on September 03, 2025) and the Refresh token (for generating new access tokens).



* **Note:** The access token will be used in Dynatrace, and the refresh token can be stored securely for future use.

**2. Store Access Token in Dynatrace Credential Vault**

* **Navigate to Credential Vault:** 
  + In Dynatrace, go to **Settings > Integration > Credential Vault.**
* **Create a New Credential:** 
  + Click Add credential.
  + Name it gmail\_access\_token.
  + Paste the access token obtained from the OAuth 2.0 Playground.
  + Click Save.

**4. gmail\_apirequest**

* **Type**: "Issue an HTTP request to any API"
* **Purpose**: Sends the base64-encoded email to the Gmail API.
* **Configuration**:
  + **Endpoint**: https://gmail.googleapis.com/gmail/v1/users/me/messages/send
  + **Method**: POST
  + **Authentication**-> Turn on and selevct type as Token
  + **Prefix:** as **Bearer** and select the gmail\_access\_token
  + **Headers**:
    - Authorization: Bearer {{ secret('gmail\_access\_token') }}
    - Content-Type: application/json
  + **Body** (using Jinja):

{

"raw": "{{ result('email\_script').base64Email }}"

}

A screenshot of a computer

AI-generated content may be incorrect.

* + **Notes**: Replace email\_script with the actual task ID if different.

**Workflow Flow**

* **On demand trigger** → task-1 → email\_script → gmail\_apirequest
  + The trigger starts the workflow.
  + task-1 fetches the log data.
  + email\_script processes the data and prepares the base64-encoded email.
  + gmail\_apirequest sends the email using the access token from the Vault.

**Setup and Validation**

* **OAuth 2.0 Playground**:
  + Revisit the Playground before the token expires (around 04:58 PM IST) to generate a new access token if needed. Use the refresh token with the token endpoint if automation is desired.
* **Credential Vault**:
  + Ensure gmail\_access\_token is updated with the latest token from the Playground.
* **Testing**:
  + Trigger the workflow manually via the "Run" button.
  + Check logs for task-1 (data), email\_script (MIME generation), and gmail\_apirequest (HTTP response).
  + Verify the recipients’ inboxes (snehathamban24@gmail.com, jawahar1096@gmail.com) for the log\_details.csv attachment.
* **Troubleshooting**:
  + **No Data**: Adjust the DQL time range if task-1 returns no records.
  + **401 Unauthorized**: Refresh the access token if expired.
  + **400 Bad Request**: Log base64Encoded in email\_script to debug MIME issues.
  + **Network**: Ensure gmail.googleapis.com is allowlisted.

**Notes**

* The access token expires in 1 hour (around 04:58 PM IST). Plan to refresh it manually or automate with the refresh token.
* The script uses btoa, which is compatible with Dynatrace’s runtime. If issues arise, use Buffer.from().toString('base64').

This setup leverages the Vault and Playground effectively. Let me know if you need help with token refresh automation or encounter issues!