

Formula Template Examples

Last Modified on 04/28/2020 2:50 pm EDT

On this page


The examples in this section show a selection of common use cases for Formulas. Each example includes a table that identifies the types of triggers, steps, and variables used in the formula. The table also identifies any prerequisites required, like a connector with events. Lastly, each example includes a downloadable JSON file that you can use to create your own version of the example template with the `POST /formulas` endpoint.

CRM to Messages

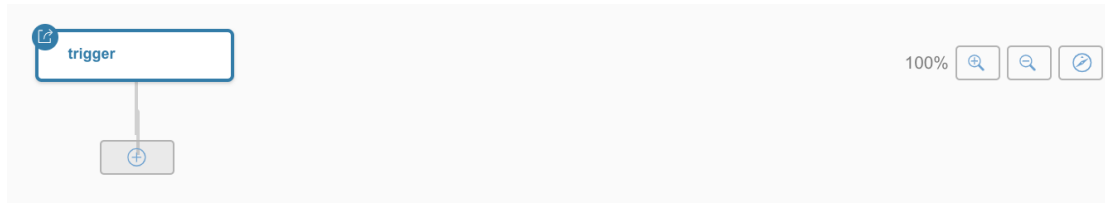
This example listens for an event on a CRM connector and then sends an email with that event information using a messaging connector. This example was tested with the [Salesforce Sales Cloud](#) and [SendGrid](#) connectors.

Trigger	Step Types	Variable Types	Prerequisites	Template JSON
Event	<ul style="list-style-type: none">JS ScriptConnector API Request	Connector Instance	<ul style="list-style-type: none">CRM hub connector instance with eventsMessaging hub authenticated connector instance	Formula JSON

To create a formula that listens for an event and emails a message:

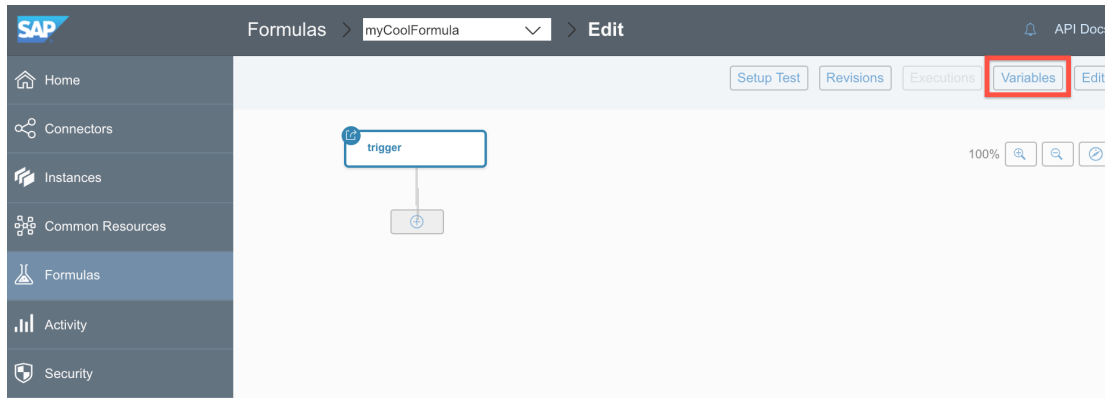
1. [Build a formula template](#) and select **Event** as the trigger.
2. Because the trigger is a change to a CRM connector, add a connector instance variable that refers to a CRM connector.
 1. Click .
 2. Click **Add New Variable**, and then click **Connector Instance**.
 3. Enter a name for your CRM variable. In this example, we'll use `crmElement`.
 4. Click **Save**.
 5. Select the variable that you just created (`crmElement`), and then click **Save** on the Edit event: "trigger" page.

Your formula visualization should look like the following example:



3. Add another connector instance variable for the messaging connector.

1. Click **Variables**.




2. Click **Connector Instance**.

3. Enter a name. For this tutorial we'll call it `messagingElement`.

4. Click **Save**.



4. In the formula visualization, click  to add a step.

5. Create a JS Script step that constructs a message when the trigger happens.

1. Click **JS Script**.

2. Enter a name for the script. We'll call it `constructBody`.

3. Enter a script that constructs a message, such as the example below.

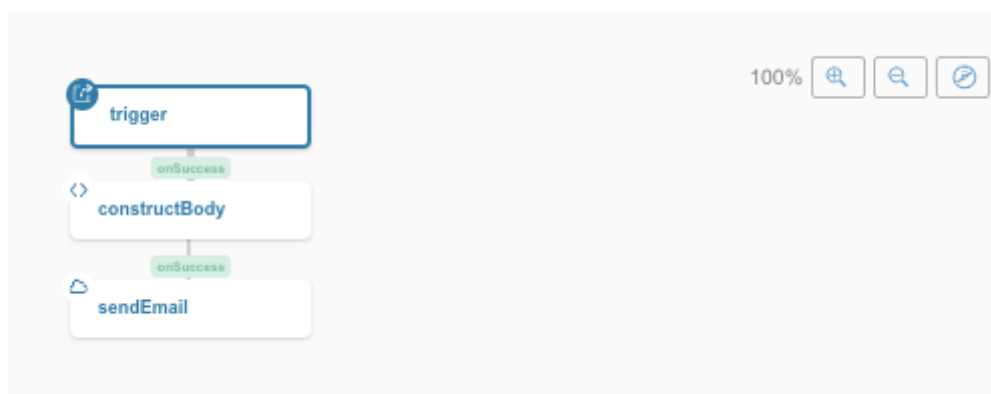
```
done( {  
  "subject": "CRM Event Occurred",  
  "to": "recipient@cloud-elements.com",  
  "from": "sender@cloud-elements.com",  
  "message": `${trigger.event.objectType} with ID ${trigger.event.  
objectId} was ${trigger.event.eventType}`  
});
```

4. Click **Save**.

6. Create a Connector API Request step to send the message that you created in the previous step. Click the `constructBody` step, and then click **Add OnSuccess**.

1. Select **Connector API Request**.
2. Enter a name for the step. We'll call it `sendEmail`.
3. In **Connector Instance Variable**, click **+**, and then select the `messagingElement` variable that we created earlier.
4. In **Method**, select **POST** because the formula will submit a POST request to the messaging hub to send an email.
5. In **API**, enter the API used to send email messages. In this case, enter `/messages`.
6. Click **Show Advanced**.
7. Scroll to **Body** and enter the reference to the email that we constructed earlier. In this case, type `${steps.constructBody}`.
8. Click **Save**.

Your formula should look like the visualization below. It should include a trigger and two steps: the first constructs an email and the second sends a message.



Add New Contact Created in One System to Another

This example listens for a new contact on one connector instance, and then adds the new contact to another connector instance. The trigger for the formula is an Event. When a new contact is created at a connector instance that has events set up, the trigger receives a payload with the raw contact information. Because this raw data cannot be used to create the same contact at a different connector instance, the formula uses the `objectID` from the trigger to get the transformed contact instead. The formula then posts the transformed contact to the target connector instance.

For this example to work, you must [define a common resource](#) to transform the data received from Salesforce.

This example was tested with the [Salesforce Sales Cloud](#) and [HubSpot CRM](#) connectors.

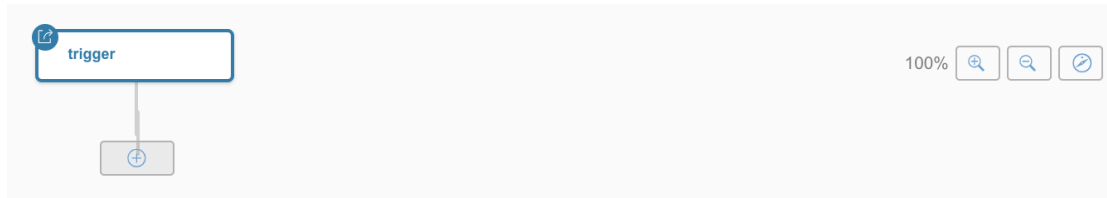
Trigger	Step Types	Variable Types	Prerequisites	Template JSON
Event	<ul style="list-style-type: none"> JS Filter 	Connector	<ul style="list-style-type: none"> CRM hub authenticated 	Formula

Trigger	Step Types	Connector Instance Variable Types	Prerequisites	JSON Template JSON
	<ul style="list-style-type: none"> Connector API Request 		<ul style="list-style-type: none"> connector instance with events CRM hub authenticated connector instance to sync new contact to A common resource that transforms contacts A common resource mapped to the origin and destination connector instances 	

To create a formula that adds new contacts created in one system to another:

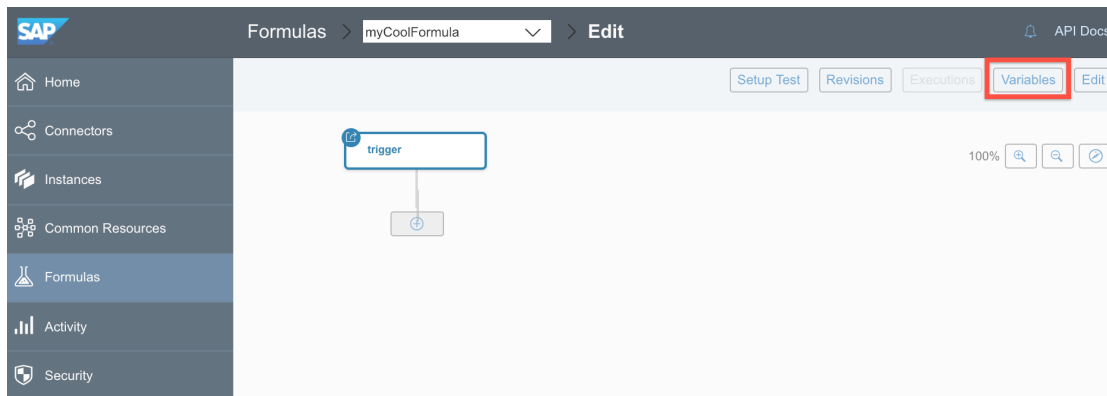
1. [Build a formula template](#) and select **Event** as the trigger.
2. Because the trigger originates from a connector instance configured to listen for events, add a connector instance variable.
 1. Click **+**.
 2. Click **Add New Variable**, and then click **Connector Instance**.
 3. Enter a name for your CRM variable. In this example, we'll use `originInstance`.
 4. Click **Save**.
 5. Select the variable that you just created (`originInstance`), and then click **Save**.

Your formula visualization should look like the following example:




3. Add another Connector Instance variable to represent the system to update after you create a contact at the `originInstance`.

1. Click **Variables**.



2. Click **Connector Instance**.
3. Enter a name. For this tutorial we'll call it `destinationInstance`.
4. Click **Save**.



4. In the formula visualization, click  to add a step.
5. Create a JS Filter step that checks to be sure the event is a created contact, and not an updated or deleted contact.


1. Click **JS Filter (true/false)**.
2. Enter a name for the script. We'll call it `isCreateContact`.
3. Enter a script that checks to be sure the event was caused by a created object, such as the example below.

```
let theEvent = trigger.event.eventType;
let theObject = trigger.event.objectType;

done((theEvent === 'CREATED') && (theObject === 'Contact' || theObject === 'contacts'));
```

7. Create a Connector API Request step to retrieve the transformed version of the newly created object based on the `objectId` in the trigger. Click the `isCreateContact` step, and then click **Add OnSuccess**.

Note: This step uses the `objectId` from the trigger to retrieve the transformed object. If you just retrieved the information about the object from the event payload in the trigger, it would not be transformed and could not sync with another connector.

1. Select **Connector API Request**.
2. Enter a name. For this tutorial we'll call it `retrieveOriginalContact`.
3. In **Connector Instance Variable**, click , and then select the `originInstance` variable that we created earlier.
4. In **Method**, select **GET** because the formula will submit a GET request to a common resource.
5. In **API**, retrieve the transformed newly created contact by entering the endpoint of the common resource and specifying the `objectId` from the trigger. For this tutorial, the common resource is called `myContacts`.

```
/MyContacts/${trigger.event.objectId}
```

6. Click **Save**.
8. Create a Connector API Request step to add the contact to another connector instance.

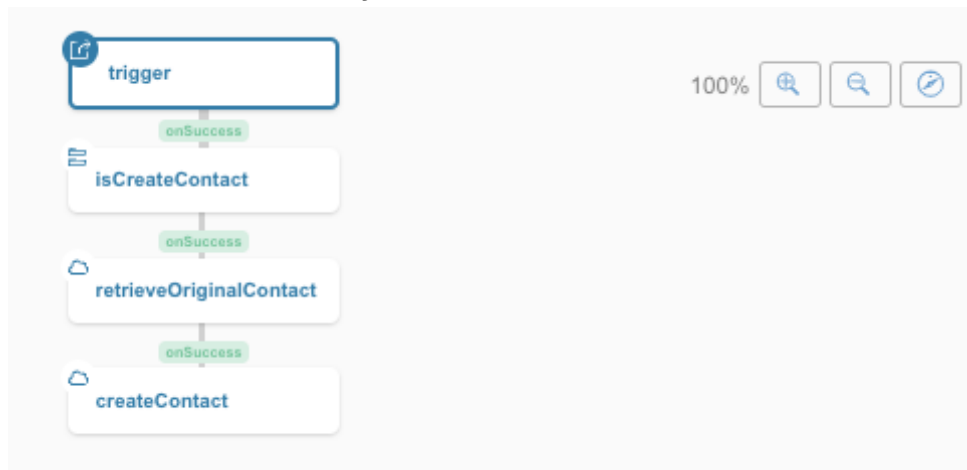
Click the `retrieveOriginalContact` step, and then click **Add OnSuccess**.

1. Select **Connector API Request**.
2. Enter a name. For this tutorial we'll call it `createContact`.
3. In **Connector Instance Variable**, click **+**, and then select the `destinationInstance` variable that we created earlier.
4. In **Method**, select **POST** because the formula will submit a POST request to sync the contact.
5. In **API**, enter the API to the common resource. For this tutorial, the common resource is called `myContacts`.

```
/MyContacts
```

6. Click **Show Advanced**.
7. Scroll to **Body** and enter the reference to the step with the transformed contact data. In this case, type `${steps.retrieveOriginalContact.response.body}` . This inserts the body from the `retrieveOriginalContact` step—the JSON describing the transformed contact—in the POST request to the `destinationInstance`.
8. Click **Save**.

Your formula is finished and should look like the visualization below. It should include a trigger and three steps: the first checks that an event is a created contact, the second gets the transformed contact data, and the third syncs the contact.



Bulk Transfer CRM Data

Bulk data transfer is a common use case. For example, your first sync between CRM systems or maybe you add many accounts or contacts each day and want a single job to run to sync between systems. This example demonstrates how to use two Formulas to complete a bulk transfer.

Trigger	Step Types	Variable Types	Prerequisites	Template JSON
<ul style="list-style-type: none"> Scheduled (Formula 1) Manual (Formula 2) 	<ul style="list-style-type: none"> JS Script Connector API Request JS Filter (Formula 2) Stream File (Formula 2) 	<ul style="list-style-type: none"> Value Connector Instance 	<ul style="list-style-type: none"> CRM hub authenticated connector instance with events CRM hub authenticated connector instance to sync new contact to 	<ul style="list-style-type: none"> Step 1 Formula JSON Step 2 Formula JSON

Formula 1

To create a formula that makes a bulk query and then triggers the second formula that will download and then upload the bulk files:

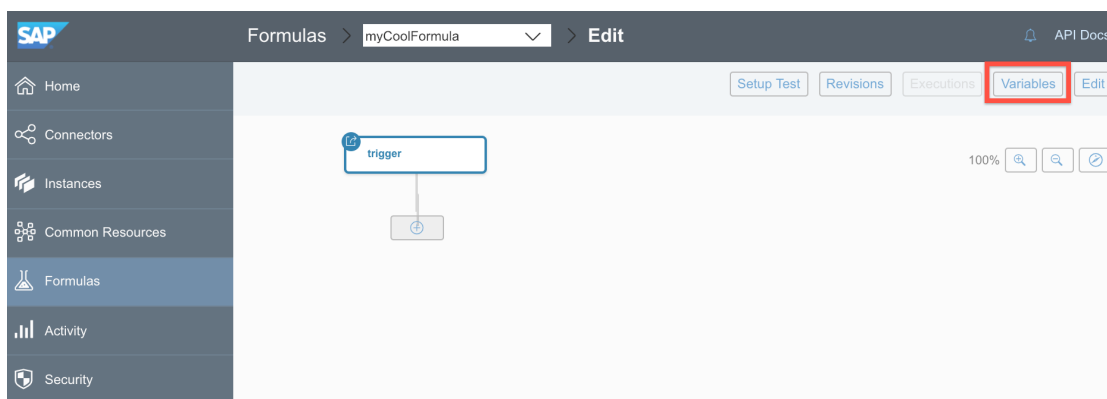
1. [Build a formula template](#) and select **Scheduled** as the trigger.
2. Add a cron string to identify when the sync occurs.

This example fires every Monday through Friday at 1:00 a.m..

```
0 0 1 ? * MON,TUE,WED,THU,FRI *
```


3. Add three variables for the 1) The resource that you want to sync (like `account` or `contact`), 2) The connector instance that includes the resources that you want to sync and, 3) The formula instance id associated with the second formula ([Formula 2](#)) in this process.

1. Click **Variables**.



2. Click **Value**.
3. Enter a name for the variable that represents the resource that you want to sync. For this tutorial we'll call it `resourceName`.
4. Click **Save**.
5. Repeat to create a Value variable called `stepTwoId`.
6. Create a Connector Instance variable named `originInstance`.




4. In the formula visualization, click  to add a step.
5. Create a JS Script step that builds the metadata for the bulk query, including the OCNQL query that requests a specific resource and the callback URL that will be the formula execution endpoint that executes [Formula 2](#).

1. Click **JS Script**.
2. Enter a name for the script. We'll call it `buildMetaData`.

```
done ({
  "query":{
    "q":"select * from " + config.resourceName
  },
  "headers":{
    "Elements-Async-Callback-Url":"/formulas/instances/" + config.stepTwoId + "/executions"
  }
});
```

7. Create a Connector API Request step to make a bulk download query, referencing the query and callback URL created in `buildMetaData`. Click the `buildMetaData` step, and the click **Add OnSuccess**.

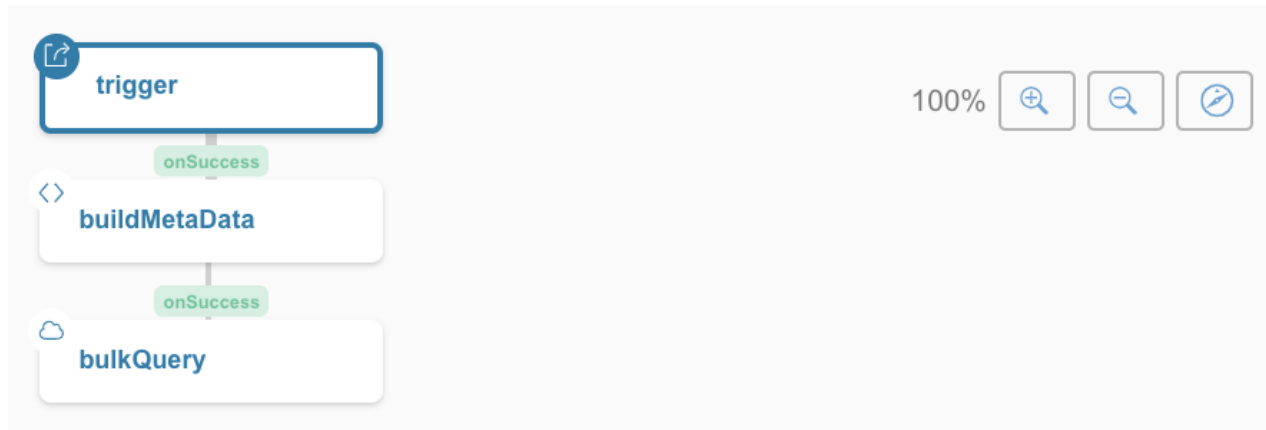
1. Select **Connector API Request**.
2. Enter a name. For this tutorial we'll call it `bulkQuery`.
3. In **Connector Instance Variable**, click , and then select the `originInstance` variable that we created earlier.
4. In **Method**, select **POST** because the formula will submit a POST request to the resource.
5. In **API**, enter the endpoint to make a bulk query.

```
/bulk/query
```

6. Click **Show Advanced**.
7. In **Headers**, enter the reference to the headers that you built in the script in the

- `buildMetaData` step. In this case, type `${steps.buildMetaData.headers}`.
8. In **Query**, enter the reference to the query that you built in the script in the `buildMetaData` step. In this case, type `${steps.buildMetaData.query}`.
 9. Click **Save**.

The first formula should look like the visualization below. It should include a trigger and two steps: the first builds the metadata for a bulk query, and the second makes the bulk query, which includes a callback to the formula execution endpoint of the next formula.



Formula 2

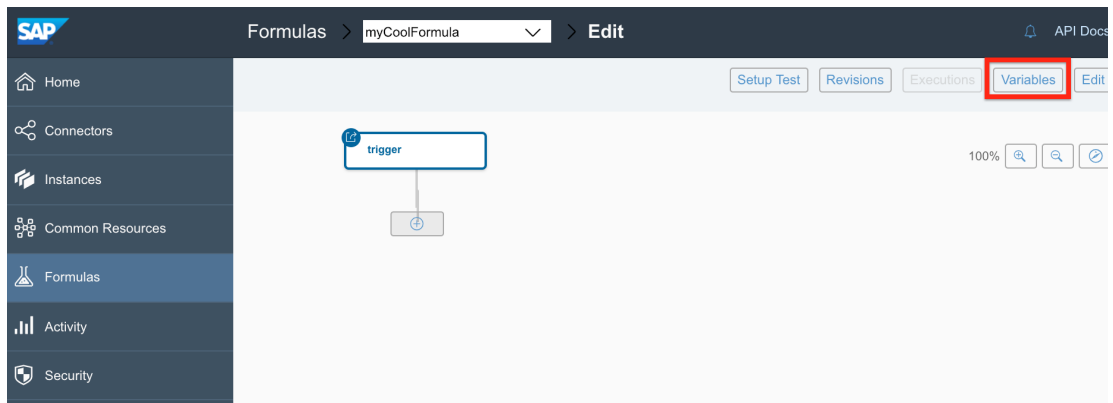
To create a formula that receives the notification that the job completes, downloads the file from the original connector, and posts to the destination:

1. [Build a formula template](#) and select **Manual** as the trigger, and then click **Save**.

Note: You do not need to configure anything for the manual trigger, but take note of the endpoint that you will need to trigger the formula:

```
POST
/formulas/instances/:id/executions
```

2. Add two connector instance variables to represent the connector that you are downloading from and the connector that you are uploading to, and a variable to represent the resource that you are syncing.
 1. Click **Variables**.



2. Click **Value**.
3. Enter a name for the variable that represents the resource that you want to sync. For this tutorial we'll call it `resourceName`.
4. Click **Save**.
5. Create Connector Instance variables to represent the source and target systems to sync. For this example, use `originInstance` and `destinationInstance`.



3. In the formula visualization, click  to add a step.

4. Create a JS Filter step that makes sure that the bulk query is completed.
 1. Click **JS Filter (true/false)**.
 2. Enter a name for the script. We'll call it `isSuccessful`.
 3. Enter a script such as the example below.

```

let status = trigger.args.status;

if (status && status === "COMPLETED") {
  done(true);
} else {
  done(false);
}

```

6. Create a JS Script step that defines an identifier field, which is the unique key for an upsert operation. It also specifies the content type as `csv`. Click the `isSuccessful` step, and then click **Add OnSuccess**.

1. Click **JS Script**.
2. Enter a name for the script. We'll call it `buildMetaData`.
3. Enter a script like the following example:

```

const metaData = {
  "identifierFieldName": "email"
}

const downloadHeaders = {
  "Accept": "text/csv"
};

done({
  "metaData": metaData,
  "downloadHeaders": downloadHeaders
});

```

8. Create a Connector Stream step to move the files downloaded from the origin instance to the destination instance. Click the **buildMetaData** step, and then click **Add OnSuccess** .

1. Select **Stream File**.
2. Enter a name. For this example we'll call it `bulkStream` .
3. In **Download Connector Instance Variable**, click **+** , and then select the **originInstance** variable that we created earlier.
4. In **Download Method**, enter `GET` .
5. In **Download API**, enter `/bulk/${trigger.args.id}/${config.resourceName}` .
`${trigger.args.id}` gets the id from the payload sent to the trigger by [Formula](#)
 1. `${config.resourceName}` refers to the resourceName variable that identifies the resource that you want to sync.
6. In **Upload Connector Instance Variable**, click **+** , and then select the **destinationInstance** variable that we created earlier.
7. In **Upload Method**, enter `POST` .
8. In **Upload API**, enter `/bulk/${config.resourceName}/${trigger.args.id}` .
9. Click **Show Advanced**.
10. In **Download Headers**, enter the reference to the download headers that you built in the script in the `buildMetaData` step. In this case, type `${steps.buildMetaData.downloadHeaders}` .
11. In **Upload Query**, enter the reference to the upload query that you built in the script in the `buildMetaData` step. In this case, type `${steps.buildMetaData.metaData}` .
12. Click **Save**.

The second formula should look like the visualization below.

