

# Integrating Jenkins with Power Automate to call Jenkins API

November 18, 2022 November 20, 2022 Chris [Edit](#)

## **Objectives:**

To call the Internal Jenkins API using Microsoft Power Automate Flows

This article will help you to set up a continuous integration environment using Jenkins for and Microsoft Power Automate.

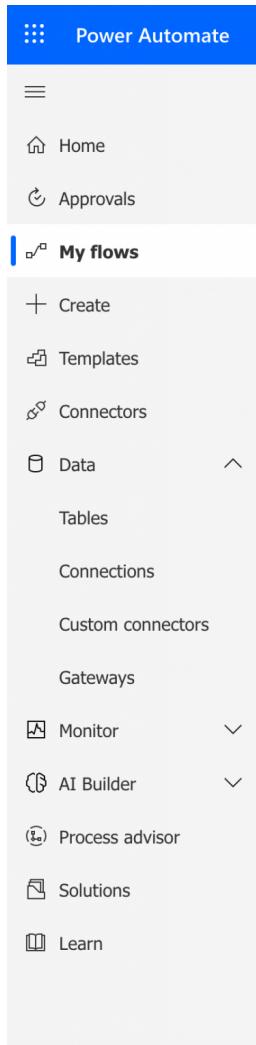
## Building the Jenkins Custom Connector

Let's start with the assumption that you have an In-house Jenkins server sitting somewhere in your Infrastructure stack.

Power Automate has an On-premises gateway you can install on Windows. The Idea is to Install the Gateway on a Windows host that has network access to the Jenkins box. Microsoft has a pretty decent Article on this [here](#)

Once Installed & configured, we can head over to <https://make.powerautomate.com>

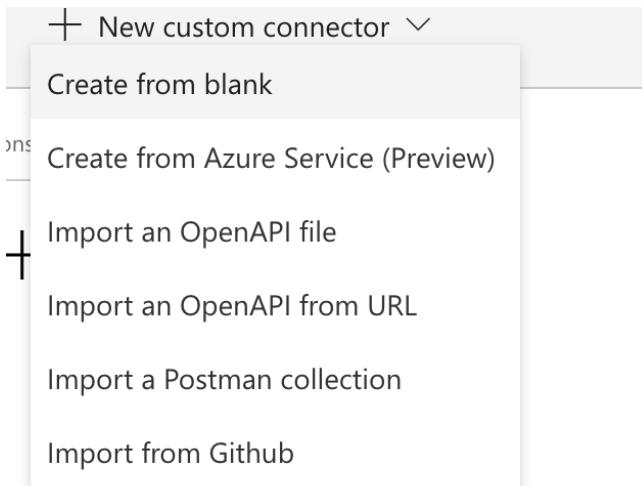
On the Left Menu, expand Data.



Now Click on “Custom connectors”

Custom connectors		<a href="#">+ New custom connector</a>
Icon	Name	Actions

Click “New custom connector”



Choose "Create from blank"

Create from blank

Connector name

Add a name

Continue Cancel

Give your Custom connector a name. I gave my custom connector the name "Jenkins-CustomConnector"  
Once you click Continue, you should see the below.

Connector Name Jenkins-CustomConnector

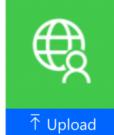
1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test

Swagger Editor  Create connector

### General information

Add an icon and short description to your custom connector. Your host and base URL will be automatically generated from the swagger file.

General information



Upload connector icon  
Supported file formats are PNG and JPG. (< 1MB)

Icon background color  
A color to show behind the icon (e.g., '#007ee5')

Description  
Give your custom connector a short description

Connect via on-premises data gateway [Learn more](#)

Scheme \*  
 HTTPS  HTTP

Host \*  
api.contoso.com !

Base URL  
/

Security →

The First Option allows you to choose an **Icon** for your Custom Connector.

You have the option to choose a **background** color.

Give your custom connector a **description**.

Tick the “**Connect via on-premises data gateway**” option.

Choose your API **scheme**.

Populate the Jenkins API **hostname**. Example: jenkins.domain.com

### General information

Add an icon and short description to your custom connector. Your host and base URL will be automatically generated from the swagger file.



**Upload connector icon**  
 Supported file formats are PNG and JPG. (< 1MB)

Icon background color

Description

Connect via on-premises data gateway [Learn more](#)

Scheme \*  
 HTTPS  HTTP

Host \*

Base URL

[Security →](#)

The Next page allows you to choose the authentication type for the API call to Jenkins from the on-premises gateway to on-premises Jenkins box. We can choose “Basic authentication”

Enter the parameters names. username for the **username** field and password for the **password** field.

### Security

Choose the authentication type and fill in the required fields to set the security for your custom connector.

[Learn more](#)

Authentication type

Choose what authentication is implemented by your API \*
   
 Basic authentication

i Do NOT enter secrets here. These fields are used to configure display names for connections.

#### Basic authentication

Users will have to provide a valid user name and password before using this API

Parameter label *	Parameter name
<input type="text" value="username"/>	username
<input type="text" value="password"/>	password

[Edit](#)

[← General](#)
[Definition →](#)

The Next page is called the Definition page. Here you’re able to define your Action and triggers.  
 for the purpose of this article, we will be creating two actions.  
 first 1 for the Jenkins CRUMB request  
 second one to start a Jenkins build.

For the Jenkins CRUMB, click “New Action”

In the **Summary** enter something that makes sense like “Request Jenkins Crumb”

**Description:** Request a Crumb from Signed in Jenkins User

for the **Operation ID** I used: REQCRUMB

The screenshot shows the Jenkins CRUMB configuration interface. On the left, there's a sidebar with sections for Actions (1) and Triggers (0). The Actions section contains a table with one row, showing a checkbox, the number 1, the text 'REQCRUMB', three dots, and a 'New action' button. The Triggers section is empty. The main area has a tab bar with 'General' selected. Under 'General', there are fields for 'Summary' (containing 'Request Jenkins Crumb'), 'Description' (containing 'Request a Crumb from Signed in Jenkins User'), 'Operation ID \*' (containing 'REQCRUMB'), and 'Visibility' (with a radio button selected for 'none').

Scroll down to the “Request” section. You should see the “Import from sample” option. Let’s Click on that.

The screenshot shows the Jenkins Request configuration interface. It has a 'Request' tab selected. There is a blue button labeled '+ Import from sample'. Below it, there is a field for 'Verb \*' with the placeholder text 'The verb describes the operations available on a single path.'

The Import from sample window should appear and you will see an option to select an API verb. Select “GET” and for the URL paste the Internal API call to request a Jenkins Crumb. For me it's <http://jenkins.ctejeda.com/crumbIssuer/api/xml>

My sample setting look like this:

## Import from sample

X

**Verb \***

- GET    DELETE    POST    PUT    HEAD    OPTIONS  
 PATCH

**URL \***

This is the request URL.

**Headers**

Headers separated by a new line, e.g.:  
Content-Type application/json  
Accept application/json

These are custom headers that are part of the request.

**Import****Close**

Click import.

At this point you may choose to test an confirm if you are able to retrieve a response that shows the Jenkins Crumb in the API response.

Click on “Test”

The screenshot shows a web-based configuration interface. On the left, under 'Test operation', there is a descriptive text about testing a specified operation using a selected connection. On the right, under 'Connections', there is a section titled 'Selected connection \*' with a dropdown menu showing 'None'. A 'New connection' button is also present.

Click on “New connection”

You will be redirected to a new page to create a new secure connection.

username \*

password \*

Authentication Type \*

Select gateway \*

If you don't see a gateway or want a new one, you can install one now. [Install gateway](#)

[Cancel](#) [Create connection](#)

Populate the required fields

Username = Jenkins username

Password = Jenkins API Token. **Note:** To Create the API TOKEN, In Jenkins go to: Accounts icon -> configure -> API Token -> Add New token

Authentication Type = Basic

Select Gateway = Select the Gateway you installed earlier

click "Create connection"

The Test page should now have the connection you created as an option.

The screenshot shows two pages side-by-side. On the left, the 'Test operation' page has a 'Test operation' section with instructions and a 'REQCRUMB' operation listed under 'Operations (1)'. On the right, the 'Connections' page shows a dropdown menu with 'Jenkins-CustomConnector' selected. Both pages have a blue header bar.

**Test operation**

Test a specified operation of this custom connector using the selected connection. You must update the custom connector in order to test recent changes.

**Operations (1)**

These are the operations defined by your custom connector. This includes actions and triggers.

1 **REQCRUMB**

**Connections**

Selected connection \*

Jenkins-CustomConnector (Created at 2022-11-18T23:16:25.1291091Z)

+ New connection

**REQCRUMB**

**Test operation**

← Code (Preview)

Click on the Test Operation button to test the Request CRUMB action we built. If successful, you should see a 200 status response, and the body will contain the crumb like below.

## Test operation

Test a specified operation of this custom connector using the selected connection. You must update the custom connector in order to test recent changes.

### Connections

Selected connection \*

Jenkins-CustomConnector (Created at 2022-11-19T18:09:34.3119645Z)

[+ New connection](#)

## Operations (1)

These are the operations defined by your custom connector. This includes actions and triggers.

1 **REQCRUMB**

### REQCRUMB

[Test operation](#)

[Request](#) [Response](#)

#### Status

OK (200)

#### Headers

```
{
  "cache-control": "no-cache,no-store",
  "content-encoding": "gzip",
  "content-length": "169",
  "content-type": "application/xml; charset=UTF-8",
  "date": "Sat, 19 Nov 2022 18:09:51 GMT",
}
```

#### Body

```
suer _class='hudson.security.csrf.DefaultCrumbIssuer'><crumb>f02314a42e6ae0cbbed7414c0c6ac5
```

Click on the Definitions tab to create the second Jenkins Action.  
Click "New action"



**New action**

Summary: Jenkins Build Job

Description: Jenkins Build Job

Operation ID: BUILDJENK

### General

Summary [Learn more](#)

Jenkins Build Job

Description [Learn more](#)

Jenkins Build Job

Operation ID \*

This is the unique string used to identify the operation.

BUILDJENK

Visibility [Learn more](#)

none  advanced  internal  important

Click "Import from Sample"

API Verb: POST

URL: <http://jenkins.yourdomain.com/job/{jobname}/job/{build}/build>  
the above values inside of the curly brackets serve as variables

Header: Jenkins-Crumb

The Import from sample should look similar to this:

## Import from sample



## Verb \*

- GET  DELETE  POST  PUT  HEAD  OPTIONS  
 PATCH

## URL \*

```
http://jenkins.ctejeda.com/job/{jobname}/job/{build}/build
```

This is the request URL.

## Headers

```
Jenkins-Crumb
```

These are custom headers that are part of the request.

## Body

```
JSON object with body, e.g.:
{
  "email": "test@test.com",
  "name": "Jane Doe"
}
```

The body is the payload that's appended to the HTTP request. There can only be one body parameter.

**Import**

**Close**

Click Import

The import should generate a Post API similar to this:

**Request**

**Verb \***  
The verb describes the operations available on a single path.

**POST**

**URL \***  
This is the request URL.

`http://jenkins.ctejeda.com/job/{jobname}/job/{build}/build`

**Path**  
Path is used together with Path Templating, where the parameter value is actually part of the operation's URL.

\* jobname▼\* build▼

**Query**  
Query parameters are appended to the URL. For example, in /items?id=####, the query parameter is id.

**Headers**  
These are custom headers that are part of the request.

Jenkins-Crumb▼

**Body**  
The body is the payload that's appended to the HTTP request. There can only be one body parameter.

Now we can call this custom [POST] Jenkins Build API using the CRUMB value we get from the custom [GET] REQCRUMB API. Let's test it. Save your changes by clicking "Update connector"

## ✓ Update connector

Click on the Test Tab.

## > 5. Test

If you've already created a connection, with the REQCRUMB operation selected, click "Test Operation"

### Test operation

Test a specified operation of this custom connector using the selected connection. You must update the custom connector in order to test recent changes.

### Connections

Selected connection \*

Jenkins-CustomConnector (Created at 2022-11-19T18:09:34.3119645Z)

+ New connection

### Operations (2)

These are the operations defined by your custom connector. This includes actions and triggers.

1 REQCRUMB

2 BUILDJENK

### REQCRUMB

[Test operation](#)

[Code \(Preview\)](#)

Copy the Jenkins Crumb from the response. It should be all of the values between <crumb> </crumb>

Now select the BUILDJENK operation.

### Operations (2)

These are the operations defined by your custom connector. This includes actions and triggers.

1 REQCRUMB

2 BUILDJENK

### BUILDJENK

jobname \*

build \*

Jenkins-Crumb

[Test operation](#)

[Code \(Preview\)](#)

**Jobname:** enter your Jenkins Job name. You can usually find this in the address bar when accessing Jenkins Job over the web.

The screenshot shows the Jenkins interface for the 'testjob' project. On the left, a sidebar lists various options: Up, Status (which is selected and highlighted in grey), Configure, New Item, Delete Folder, People, Build History, and Project Relationship. On the right, the main content area displays the 'testjob' folder icon. Below it is a table titled 'All' showing one build entry: 'testbuild'. The table has columns for S (Status), W (Workflow), and Name (sorted by Name). The 'testbuild' row is highlighted with a green checkmark icon and a blue diamond icon. At the bottom of the table, there are icons for S (Status), M (Metrics), and L (Logs). The URL in the browser bar is https://jenkins.ctejeda.com/job/testjob/.

S	W	Name ↓
		testbuild

Icon: S M L

**Build:** enter the build name or number. You can usually find this in the address bar when accessing the Jenkins build over the web.

<https://jenkins.ctejeda.com/job/testjob/job/testbuild/>

**Project testbuild**  
Full project name: testjob/testbuild

**Workspace**

**Recent Changes**

**Permalinks**

- Last build (#5), 1 mo 7 days ago
- Last stable build (#5), 1 mo 7 days ago
- Last successful build (#5), 1 mo 7 days ago
- Last completed build (#5), 1 mo 7 days ago

**Jenkins-Crumb:** Enter the Jenkins Crumb you copied here.  
Your BUILDJENK operation should look similar to this:

**BUILDJENK**

jobname \*

testjob

build \*

testbuild

Jenkins-Crumb

2f0c7c7d697158bbf679e044f235a648ffdaf44005b5cb03f8f0d7e927283f61

Test operation

Click on “Test Operation”

You should get a Created (201) response similar to the one below.

Operations (2)

These are the operations defined by your custom connector. This includes actions and triggers.

**1** REQCRUMB  
**2** BUILDJENK

**BUILDJENK**

jobname \*

build \*

Jenkins-Crumb

**Test operation**

**Request**    **Response**

**Status**  
Created (201)

**Headers**

```
"strict-transport-security": "max-age=31536000; includeSubDomains",
"via": "1.1 jenkins.v3locity.com",
"x-content-type-options": "nosniff,nosniff",
"x-frame-options": "DENY",
"Content-Type": "application/json; charset=UTF-8",
"Content-Length": "100",
"Date": "Mon, 20 Nov 2022 04:59:45 GMT",
"Server": "Apache/2.4.41 (Ubuntu)"
```

As a result, You should see the Jenkins Job building.

**Up**

**Status**

**Changes**

**Workspace**

**Build Now**

**Configure**

**Delete Project**

**Move**

**Rename**

**Build History**    **trend**

**Filter builds...**

#18 (pending—Finished waiting)

## Project testbuild

Full project name: testjob/testbuild



Workspace



Recent Changes

### Permalinks

- Last build (#16), 1 min 28 sec ago
- Last stable build (#16), 1 min 28 sec ago
- Last successful build (#16), 1 min 28 sec ago
- Last completed build (#16), 1 min 28 sec ago

The screenshot shows the Jenkins interface for a build named 'testbuild' under the job 'testjob'. The build number is '#18'. On the left, there's a sidebar with links: 'Back to Project', 'Status', 'Changes', 'Console Output' (which is selected and highlighted in blue), and 'View as plain text'. The main content area is titled 'Console Output' with a green checkmark icon. It displays the following information:  
Started by user Christopher Tejeda  
Running as SYSTEM  
Building in workspace /var/lib/jenkins/workspace/testjob/testbuild  
Finished: SUCCESS

## Building the Power Automate Flow

Now we can start to build out our Power Automate Flow.  
Click on My flows in the left Menu.

A screenshot of a user interface showing a menu item labeled 'My flows' with a square icon containing a flow symbol.

Click on “New flow” and choose “Automated cloud flow”

A screenshot of the Power Automate 'My flows' menu. At the top, there are 'New flow' and 'Import' buttons. Below them are two sections:  
**Start from a template**: Contains 'Template' and 'Visio template' options.  
**Build your own from blank**: Contains 'Automated cloud flow', 'Instant cloud flow', 'Scheduled cloud flow', 'You describe it, AI builds it', and 'Desktop flow' options. The 'Automated cloud flow' option is highlighted with a blue background.

the “Build an automated cloud flow” window will appear.

## Build an automated cloud flow



Free yourself from repetitive work just by connecting the apps you already use—automate alerts, reports, and other tasks.

Examples:

- Automatically collect and store data in business solutions
- Generate reports via custom queries on your SQL database

### Flow name

Add a name or we'll generate one

### Choose your flow's trigger \*

Search or select a trigger from the list below to create a flow. (Required)

Search all triggers

When a new response is submitted  
Microsoft Forms (i)

When an item is created  
SharePoint (i)

When an item is created or modified  
SharePoint (i)

When a file is created in a folder (depr...)  
SharePoint (i)

When a file is created  
(i)

Skip

Create

Cancel

click on “skip”

The first screen you will see is an option to choose a trigger for your flow. for the purpose if this article, I will choose the “Flow button for mobile” to test.

Search connectors and triggers

All Built-in Standard Premium Custom My clipboard

  
Flow button  
for mobile

  
PowerApps

  
Power Virtual  
Agents

  
Microsoft  
Forms

  
SharePoint

  
OneDrive for  
Business

  
Planner

  
Microsoft  
Dataverse

  
RSS

  
Gmail

  
Microsoft  
Teams

  
Google  
Calendar

  
Azure  
DevOps

  
Office 365  
Outlook

 Triggers

 Actions

See more

  
Manually trigger a flow  
Flow button for mobile

  
PowerApps  
PowerApps

  
When Power Virtual Agents calls a flow  
Power Virtual Agents

  
When a new response is submitted  
Microsoft Forms

  
When an item is created  
SharePoint

  
When an item is created or modified  
SharePoint

 Search connectors and triggers

Triggers Actions See more

---

 Manually trigger a flow  
Flow button for mobile 

Don't see what you need?  
 Help us decide which connectors and triggers to add next with [UserVoice](#)

 Manually trigger a flow  ...

 Add an input

 + New step  Save

Click Next step

Choose an operation

Search connectors and actions

All   Built-in   Standard   Premium   Custom   My clipboard

Control   AI Builder   Desktop flows   Excel Online (Business)   MSN Weather   Mail   Microsoft Dataverse

Triggers   Actions   See more

**Condition**  
Control

**Analyze positive or negative sentiment in text**  
AI Builder

**Classify text into categories with one of your custom models**  
AI Builder

**Classify text into categories with the standard model (preview)**  
AI Builder

**Detect and count objects in images**  
AI Builder

**Detect the language being used in text**  
AI Builder

click the “custom” tab under “Choose an operation”

Here you should see your custom Jenkins connector ( I have a few ). Choose the one you created.

The screenshot shows the Zapier interface for choosing an operation. At the top, there's a search bar with the placeholder "Search connectors and actions". Below it, a navigation bar has tabs for "All", "Built-in", "Standard", "Premium", "Custom" (which is underlined in blue), and "My clipboard". Under the "Custom" tab, three connectors are listed: "Jenkins-CustomCo...", "Jenkins-TokReq", and "Test-Jenkins". Each item has a small icon and a brief description.

A list of Actions should appear.

The screenshot shows the "Jenkins-CustomConnector" action list. It features a header with the connector name and standard navigation icons. A search bar at the top allows users to find specific actions. Below the search bar, two actions are listed under the "Actions" tab: "Jenkins Build Job" (marked as PREMIUM) and "Request Jenkins Crumb" (also marked as PREMIUM). Each action has a small icon, its name, the premium status, and a help icon. A "Don't see what you need?" section at the bottom encourages user feedback via UserVoice.

Choose “Request Jenkins Crumb” (or the description you used for your custom connector)

The screenshot shows a flow editor interface. At the top, there is a blue header bar with a hand icon and the text "Manually trigger a flow". Below this is a green header bar for a step titled "Request Jenkins Crumb". A tooltip "Menu for Request Jenkins Crumb" is visible near the top right of the green bar. In the center, there is a text area stating: "No additional information is needed for this step. You will be able to use the outputs in subsequent steps." At the bottom of the step's container, there are two buttons: "+ New step" and "Save".

Click on the ellipsis on the right of the action and select “Add a connection”

The screenshot shows a configuration dialog for a "Jenkins-CustomConnector". The title bar says "Jenkins-CustomConnector". The form contains the following fields:

- \* Connection name: Enter name for connection
- \* Authentication Type: basic
- \* username: The username for this api
- \* password: The password for this api
- \* gateway: gateway

At the bottom are "Create" and "Cancel" buttons.

**Connection name:** Enter a name for your connection.

**Authentication Type:** Authentication can be left as Basic

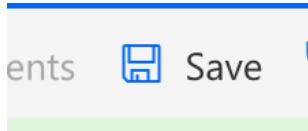
**Username:** enter a username (service account if you have one)

**Password:** enter the API Token we generated

**Gateway:** Select the on-premises gateway we configured earlier

Click Create

click Save



click test to test our flow with our custom connector.

## Test Flow

X

Manually

Perform the starting action to trigger it.

Automatically

There are no runs for this flow.

Test

Cancel

Click Test

## Run flow



**Button -> Request Jenkins Crumb**

Owner: Christopher Tejeda

**Sign in \***

This flow uses the following apps. A green check means you're ready to go.



Jenkins-CustomConnec...



...

---

**Continue**

**Cancel**

Click Continue & Run Flow

 Run flow

Your flow run successfully started. To monitor it,  
go to the [Flow Runs Page](#).

---

[Done](#)

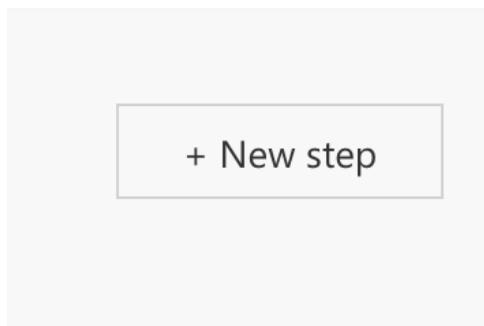
As a result, the “Request Jenkins Crumb” action should show the Jenkins Crumb in the body

The screenshot shows a Jenkins pipeline configuration. It starts with a 'Manually trigger a flow' step, which has a duration of 0s. An arrow points down to the next step, 'Request Jenkins Crumb', which has a duration of 2s. The 'Request Jenkins Crumb' step shows its 'INPUTS' section with a 'Click to download' button. In the 'OUTPUTS' section, there is a 'body' field containing XML code. The XML includes an 'Issuer' element with a class attribute set to 'hudson.security.csrf.DefaultCrumbIssuer'. A crumb value '5a70093...' is also present. Below the step, it says 'Connection: Jenkins-CustomConnector'.

Now that we know we can use the Request Jenkins Crumb action, The next step is to parse the body from the outputs of “Request Jenkins Crumb” action.

Click on the Edit button to go back to editing mode

Click Next Step



When the “Choose an operation” window appears, search for Compose.

Select the 3 dots on the right of the Compose action, and rename it to “Parse XML” or something descriptive.

The screenshot shows the 'Parse XML' configuration dialog. At the top, there is a purple header bar with a edit icon, the text 'Parse XML', a help icon, and three dots. Below the header, there is a section labeled '\* Inputs' with a text input field containing 'Inputs'. A red error message 'Inputs' is required. An arrow points down to the next step in the pipeline.

Click on the “Inputs” field. You should see an option to add the dynamic data from the previous step. Select the body from the “Request Jenkins Crumb” action.

The screenshot shows the 'Parse XML' step configuration. The 'Inputs' field is highlighted with a red asterisk and contains the error message "'Inputs' is required.". Below the inputs field are two buttons: '+ New step' and 'Save'. To the right of the main step area is a 'Dynamic content' panel titled 'Add dynamic content from the apps and connectors used in this flow.' It has tabs for 'Dynamic content' (selected) and 'Expression'. A search bar says 'Search dynamic content'. Below it is a list with 'Request Jenkins Crumb' and 'body' selected.

You new Compose operation should look similar to the below:

The screenshot shows the 'Compose' step configuration. The 'Inputs' field contains the value 'body'. Below the inputs field are two buttons: '+ New step' and 'Save'. To the right of the main step area is a 'Dynamic content' panel titled 'Add an expression to do basic things like access, convert, and compare values.' It has tabs for 'Dynamic content' (selected) and 'Expression'. A search bar says 'outputs('Parse\_XML')?['\$content']'. Below it is a button labeled 'OK'.

Click Next step

Search for Compose

Click on the "Inputs" field. This time, click on the "Expression" tab and enter the following regular expression: `outputs('Parse_XML')?['$content']`

The screenshot shows the 'Compose' step configuration. The 'Inputs' field contains the expression 'outputs('Parse\_XML')?['\$content']'. Below the inputs field are two buttons: '+ New step' and 'Save'. To the right of the main step area is a 'Dynamic content' panel titled 'Add an expression to do basic things like access, convert, and compare values.' It has tabs for 'Dynamic content' (selected) and 'Expression'. A search bar says 'fx outputs('Parse\_XML')?['\$content']'. Below it is a button labeled 'OK'.

click OK

You new Compose operation should look similar to this:

The screenshot shows the 'Compose' step configuration. The 'Inputs' field contains the expression 'outputs(...)'.

If you save an run the flow now, the last Compose Operation will show the Jenkins Crumb only. We can now use this to call our Jenkins Job.

Click Next Step and choose the Custom tab when the “Choose an operation” window appears.

Click the Custom Jenkins connector we built.

Click on the “Jenkins Build Job” Action.

The screenshot shows a web-based interface titled "Jenkins-CustomConnector". At the top, there is a search bar with the placeholder text "Search connectors and actions". Below the search bar, there are two tabs: "Triggers" and "Actions", with "Actions" being the active tab. Under the "Actions" tab, there are two items listed:

- Jenkins Build Job** [PREMIUM] - This action is associated with the "Jenkins-CustomConnector". To its right is an ellipsis button (three dots) with a question mark icon.
- Request Jenkins Crumb** [PREMIUM] - This action is also associated with the "Jenkins-CustomConnector". To its right is an ellipsis button (three dots) with a question mark icon.

Below these actions, there is a section titled "Don't see what you need?" followed by a smiley face icon and the text "Help us decide which connectors and triggers to add next with [UserVoice](#)".

Click on the ellipsis on the right of the action and choose the connection we created earlier.

The screenshot shows a configuration dialog for the "Jenkins Build Job" action. The dialog has a header with the action name and a help/question icon. It contains three input fields:

- \* jobname: An input field with no value.
- \* build: An input field with no value.
- Jenkins-Crumb: An input field with no value.

To the right of each input field is a small ellipsis button (three dots) with a question mark icon, which is the target for the user's click.

enter your parameters. for the purpose of this article, I will be starting a Jenkins test job I have created.

For the Jenkins-Crumb parameter, click on the “Jenkins-Crumb” field and select the outputs from the previous “Compose” action.

Jenkins Build Job

\*jobname: testjob

\*build: testbuild

Jenkins-Crumb: |

Add dynamic content +

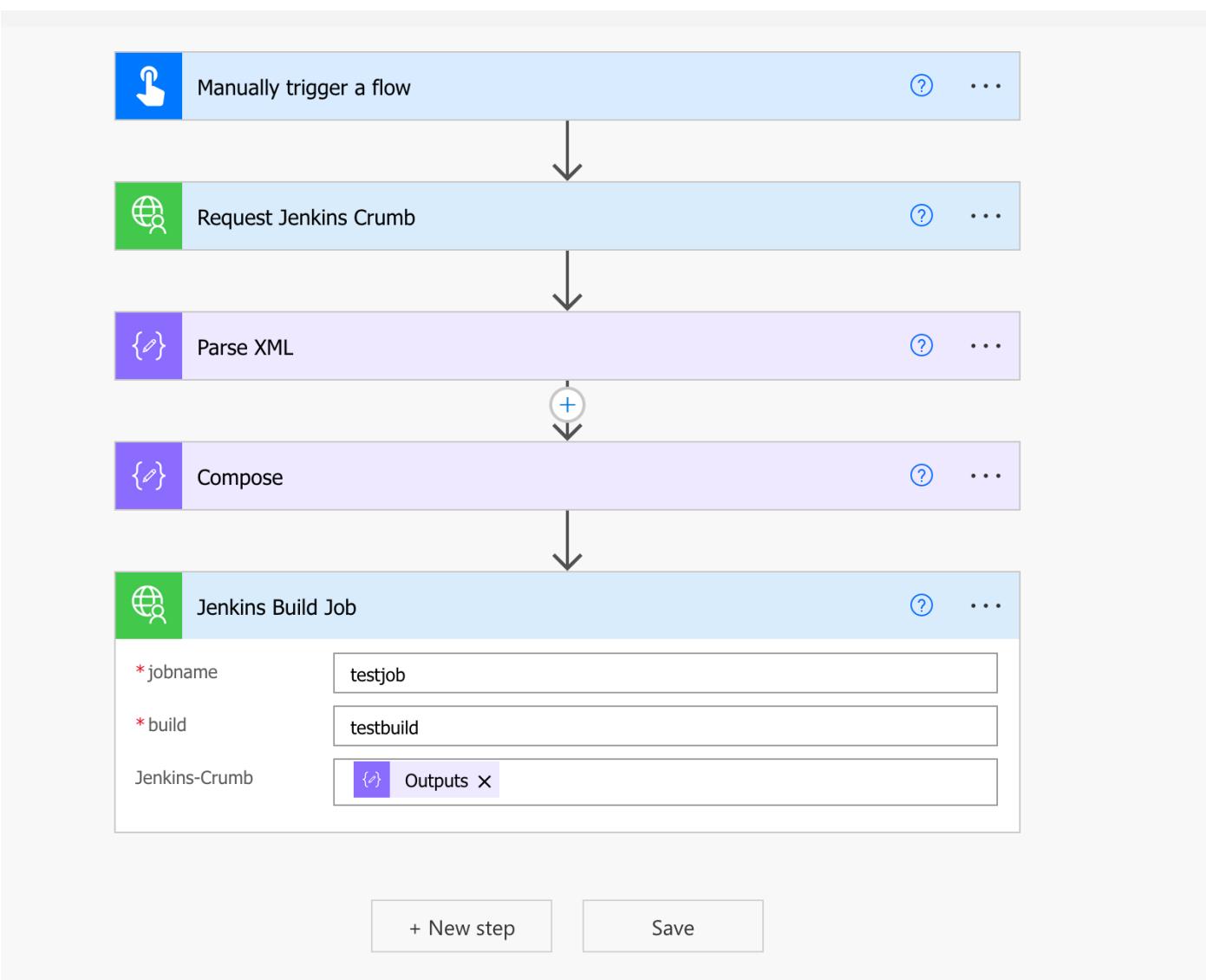
+ New step Save

Dynamic content Expression

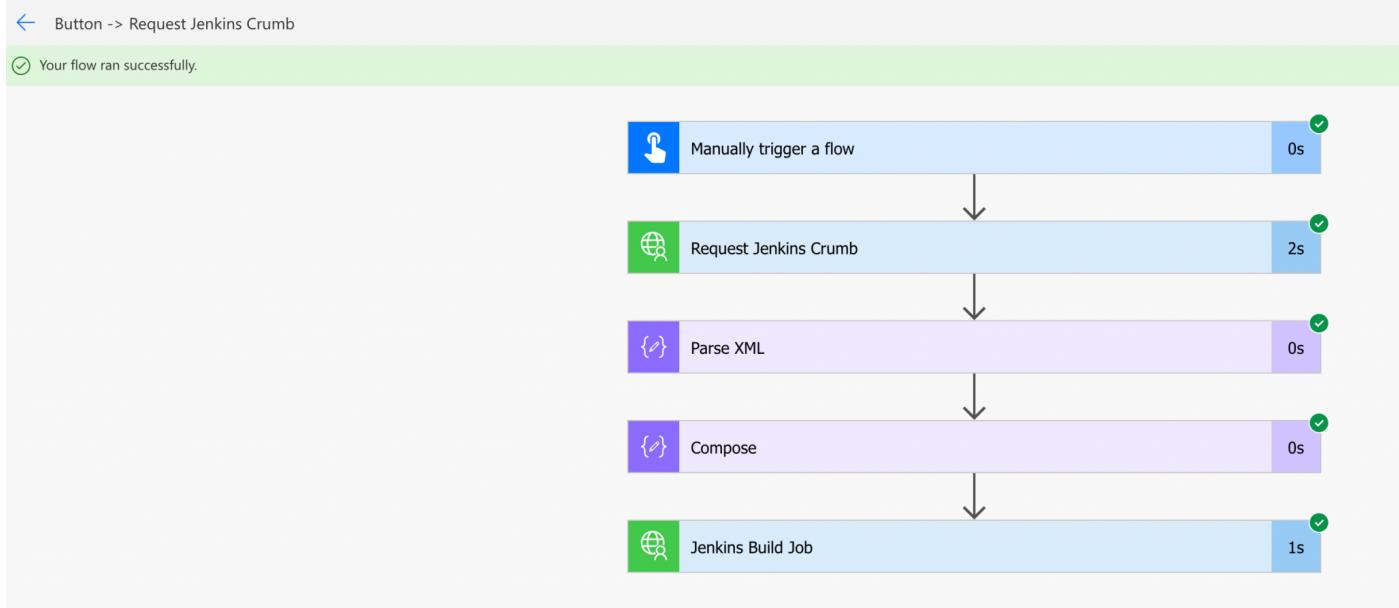
Search dynamic content

Compose Outputs

You flow should now look similar to the below:



Finally, you can save and test your flow.



As a result, your Jenkins Job should be Running.

**Jenkins**

Dashboard > testjob > testbuild >

**Project testbuild**  
Full project name: testjob/testbuild

**Workspace**

**Recent Changes**

**Permalinks**

- Last build (#18), 2 hr 3 min ago
- Last stable build (#18), 2 hr 3 min ago
- Last successful build (#18), 2 hr 3 min ago
- Last completed build (#18), 2 hr 3 min ago

**Build History**

Filter builds... #22 (pending—Finished waiting)

The screenshot shows the Jenkins interface for a build named 'testbuild' of job 'testjob'. The build number is '#22'. The status is 'SUCCESS'. The console output section has a green checkmark icon and the title 'Console Output'. The output text is as follows:

```

Started by user Christopher Tejeda
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/testjob/testbuild
Finished: SUCCESS

```

The left sidebar contains links: 'Back to Project', 'Status', 'Changes', 'Console Output' (which is selected), 'View as plain text', 'Edit Build Information', 'Delete build #22', and 'Previous Build'.

## Conclusion

We can now call Jenkins Build using the Internal Jenkins API, and Microsoft Power Automate (Flows)

as an option, you can upload the custom connector by using the below JSON

```
{
  "swagger": "2.0",
  "info": {
    "title": "Jenkins-CustomConnector",
    "description": "A custom Jenkins connector which utilizes the Jenkins API to interact with the Jenkins system. \nAuthor: Chris Tejeda",
    "version": "1.0"
  },
  "host": "jenkins.ctejeda.com",
  "basePath": "/",
  "schemes": [
    "http"
  ],
  "consumes": [],
  "produces": [],
  "paths": {
    "/crumbIssuer/api/xml": {
      "get": {
        "responses": {
          "default": {
            "description": "default",
            "schema": {}
          }
        }
      },
      "summary": "Request Jenkins Crumb",
      "description": "Request a Crumb from Signed in Jenkins User",
      "operationId": "REQCRUMB",
      "parameters": []
    }
  },
  "/job/{jobname}/job/{build}/build": {
    "post": {
      "responses": {
        "default": {
          "description": "default",
          "schema": {}
        }
      },
      "summary": "Jenkins Build Job",
      "description": "Jenkins Build Job",
      "operationId": "BUILDJENK",
      "parameters": [
        {
          "name": "jobname",
          "in": "path",
          "required": true,
          "type": "string"
        }
      ]
    }
  }
}
```

```
        "name": "build",
        "in": "path",
        "required": true,
        "type": "string"
    },
    {
        "name": "Jenkins-Crumb",
        "in": "header",
        "required": false,
        "type": "string"
    }
}
},
"definitions": {},
"parameters": {},
"responses": {},
"securityDefinitions": {
    "basic-auth": {
        "type": "basic"
    }
},
"security": [
    {
        "basic-auth": []
    }
],
"tags": []
}
```