Power Automate Cloud flows Best Practices

White paper

Summary: This technical white paper is aimed at Microsoft Power Automate Makers in the enterprise. It contains the best practices for building efficient Power Automate Cloud flows.

Writers: Reza Dorrani, Rasika Chaudhary, Foo Shen Wu

Contents

Introduction

Microsoft Power Automate offers a powerful suite of automation capabilities that empower businesses to streamline processes, automate repetitive tasks, and unlock new opportunities for growth. Microsoft Power Automate, part of the Microsoft Power Platform, offers a comprehensive automation solution that empowers organizations to automate workflows, integrate data and applications, and leverage artificial intelligence to drive efficiency and innovation. Power Automate provides a user-friendly interface and a vast array of connectors to enable automation without the need for extensive coding knowledge. By simply dragging and dropping components and configuring settings, users can quickly create powerful workflows that save time, improve efficiency, and enhance productivity. From simple notification flows to advanced multi-step processes, Power Automate offers the flexibility and scalability to meet a wide range of automation needs, making it an indispensable tool for modern digital workplaces.

Microsoft Power Automate documentation

Purpose of the white paper

This document, released by Power CAT Team, provides essential guidance for developers who design, build, test, deploy, and maintain Power Automate flows in various environments, including small businesses, corporations, and government entities.
Creating a simple cloud flow in Power Automate is straightforward, but as the requirements become complex, maintaining code quality becomes crucial. With abundant information available about Power Automate across the web, it can be challenging to discern best practices. This white paper consolidates insights from expert Power Automate makers worldwide, ensuring that you build your flows/automations correctly & efficiently.

Scope of the whitepaper

Power Automate flows include cloud flows, desktop and business process workflows.
This white paper caters the best practices around **cloud flows**.

Best Practices for Cloud flows

When using Microsoft Power Automate to automate tasks, it's important to make sure everything runs smoothly and quickly. This means following some smart ways of doing things, called "best practices." These practices help make sure your cloud flows are fast, reliable, and efficient. This article categorizes the best practices under design time and runtime. Some of the best design practices include following:

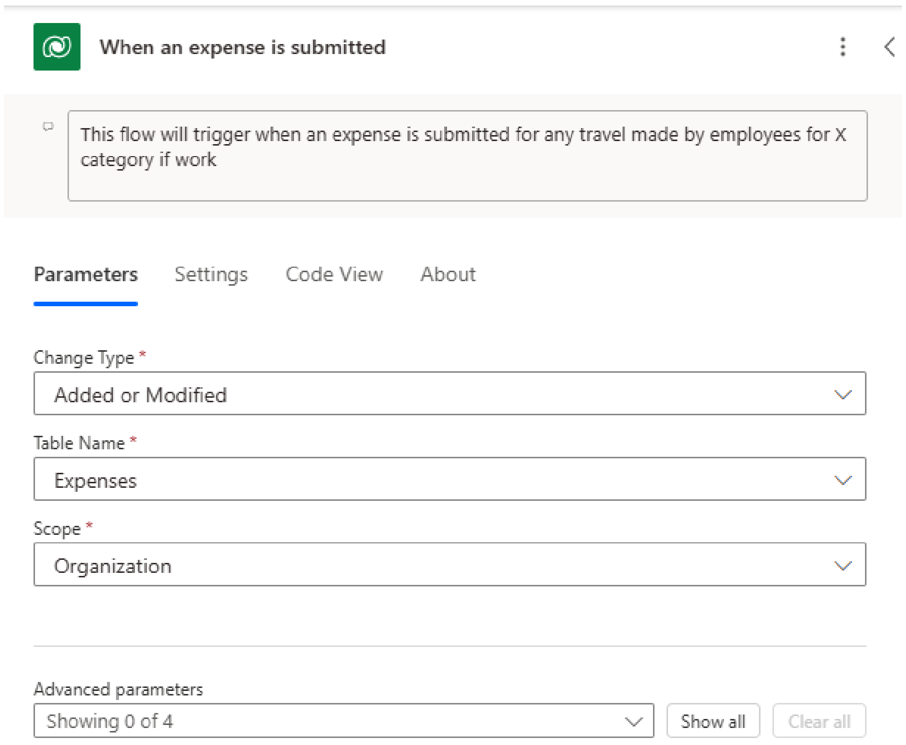
Consistent Naming for Flow Components

Maintaining consistent naming conventions for the components within your Power Automate flows is crucial for ensuring clarity, organization, and ease of management. By adopting a standardized approach to naming, you can enhance collaboration among team members, simplify troubleshooting, and streamline the development and maintenance of your workflows.

Here are some guidelines to follow:
Note: There are recommendations and may change as per individual organization requirements.

1. **Descriptive and Meaningful Names:** Give meaningful names to your flows before saving. Additionally, choose names that accurately describe the purpose or function of each component. Avoid generic or ambiguous names that could lead to confusion. For example, instead of naming a trigger "Trigger1," use a descriptive name like "New Email Received" to clearly indicate its purpose.

2. **Use CamelCase or Underscores:** Use CamelCase (capitalizing the first letter of each word except the first one) or underscores to separate words in your component names. This improves readability and makes it easier to distinguish between different parts of the name. For instance, you might name an action "SendEmailNotification" or "send_email_notification" for consistency.
3. **Prefixes or Tags:** Consider using prefixes or tags to categorize components based on their type or functionality. For example, you could use prefixes like "Trg_" for triggers, "Act_" for actions, or "Var_" for variables. This helps quickly identify the role of each component within the flow.
4. **Consistency Across Flows:** Maintain consistency in naming conventions across all your Power Automate flows to facilitate navigation and standardization. This ensures that team members can easily understand and work with different flows without having to decipher unique naming styles.
5. **Document Naming Conventions:** Document your naming conventions in a style guide or documentation to ensure that all team members are aware of the standards to follow. This promotes uniformity and reduces the likelihood of inconsistencies or misunderstandings.
6. **Add Comments –** Adding comments to the actions make it easy to understand the flow implementation logic.


When an expense is submitted

This flow will trigger when an expense is submitted for any travel made by employees for X category if work

Change Type *: Added or Modified

Table Name *: Expenses

Scope *: Organization

Advanced parameters: Showing 0 of 4

Solution Aware cloud flows vs Non-Solution cloud flows

Cloud flows that are created from/or added within a solution are known as solution-aware cloud flows or solution cloud flows. Solution aware cloud flows become portable, manageable and easy to move from one environment to other enabling strong **ALM** practices. You can add multiple flows to a single solution.

To better understand, how to create flows in a solution refer to the following document. Additionally, there are some known limitations with solution aware flows that users should be aware of.

Use pipelines in Power Platform to deploy solutions: Offers easily deploy solutions to test and production. Once pipelines are in place, makers can initiate in-product deployments with a few clicks. Makers do so directly within their development environments. More information: Overview of pipelines in Power Platform

When flows are added within a solution, flow metadata gets stored in Dataverse in the Process (Workflow) table.

Non solution cloud flows on the other hand are the flows that are created outside the solutions and need to be managed and deployed individually.

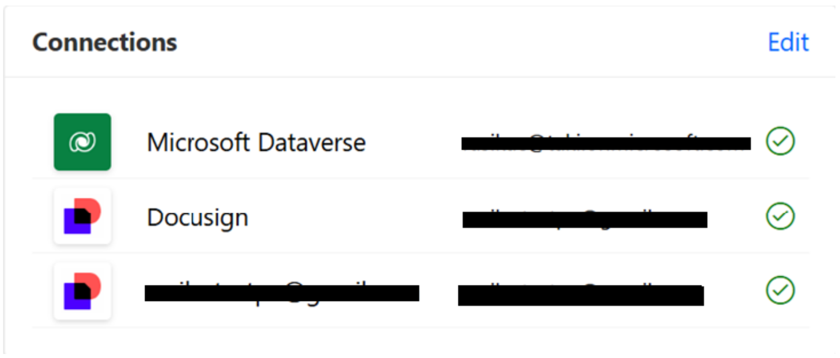
Connection References Vs Connections

In Microsoft Power Automate, the terms "connection references" and "connections" refer to different aspects of how flows interact with external services and data sources. Understanding the distinction between the two is essential for effectively managing and deploying flows, especially within the context of solution-aware cloud flows and non-solution-aware cloud flows.

Connections (non-solution aware cloud flows) - Connections are configurations that establish a link between Power Automate and external services, such as Office 365, SharePoint, Dynamics 365, Twitter, or any other connector. A connection includes authentication details (like API keys, OAuth tokens, or user credentials) required to access the external service.

When you create a flow, you need to create or select a connection to the service you want to interact with. Connections are user-specific, meaning they are tied to the credentials of the user who creates them.

If you were to change a connection for a cloud flow, every action associated with the connection needs to be updated.



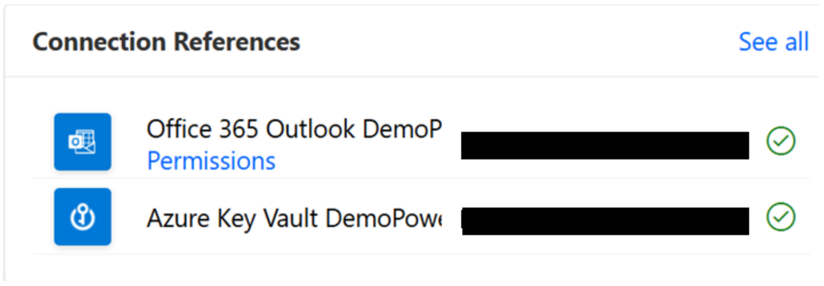
Connections Edit

Microsoft Dataverse	[redacted]	✓
Docusign	[redacted]	✓
[redacted]	[redacted]	✓

Connection References (Solution aware cloud flows) –Connection references are an abstraction layer that simplifies the management of connections within a Power Platform solution. They act as placeholders or pointers to actual connections, making it easier to manage and update connections in solution-aware scenarios.

Connection references are used in solution-aware cloud flows, which are part of a broader Power Platform solution (such as those integrated with Power Apps, Dataverse, or other components). They enable the separation of the flow's design from the specific user connections, facilitating easier deployment and updates across different environments (development, testing, production).

Since connection references are a wrapper for connections, when requiring updating connection for actions within a solution aware cloud flow, only the connection reference needs to be updated.
All the actions within the flow will automatically refer to the updated connection in the connection reference.



Limit for number of flows owned by a user

With non-solution cloud flows, a user can own a max of 600 flows.

This limit does not apply to flows created within solutions.

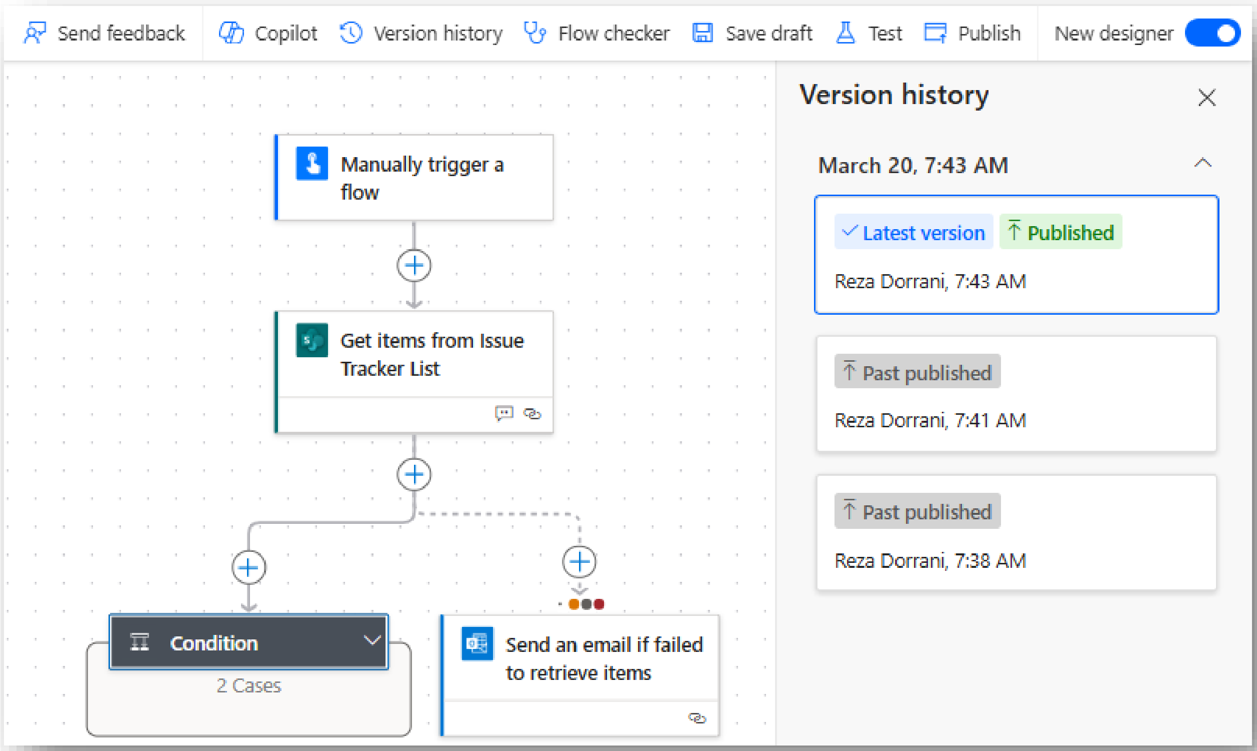
[Documentation reference]

Flow Versioning

Flow versioning is a new feature that is available only to solution-aware cloud flows where users can create draft of flows as they continue to evolve the flow logic and Publish once complete. Versioning information (metadata of flow) gets stored in Dataverse.

It allows flow maker to keep track of changes, revert to previous versions if necessary, and ensure that your flows are stable and reliable.

Refer to the following documentation to learn more about [Flow versioning]



Version history

March 20, 7:43 AM

Latest version · Published
Reza Dorrani, 7:43 AM

Past published
Reza Dorrani, 7:41 AM

Past published
Reza Dorrani, 7:38 AM

Environment Variables

Environment variables enable the basic application lifecycle management (ALM) scenario of moving an application between Power Platform environments. In this scenario, the application stays exactly the same except for a few key external application references (such as tables, connections, and keys) that are different between the source environment and the destination environment. The application requires the structure of the tables or connections to be exactly the same between the source and the destination environments, with some differences. Environment variables allow you to specify which of these different external references should be updated as the application is moved across environments

Only Solution-aware cloud flows (ALM friendly) support the use of environment variables.

Learn more

Manage cloud flow run history in Dataverse

With cloud flow run history in Dataverse, you can apply the extensibility of Dataverse to track the results of your cloud flow executions at scale..
Only solution cloud flows, with their definitions in Dataverse, can have their run history stored in Dataverse.

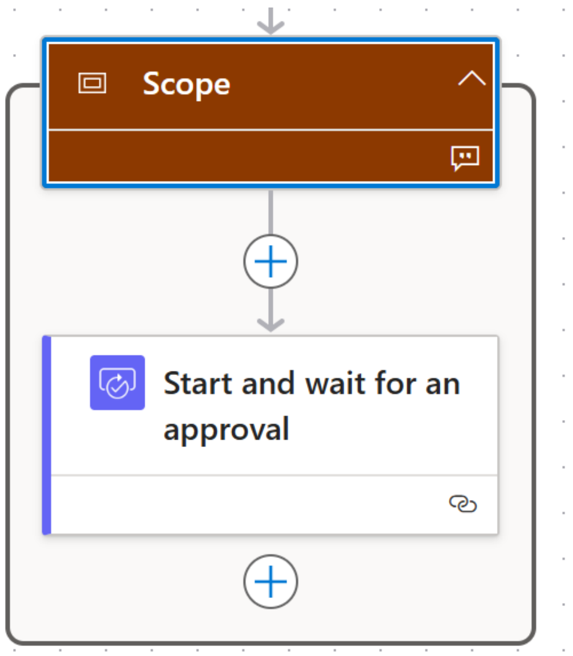
As part of this feature, each cloud flow execution has an entry in the table FlowRun. This feature is using Dataverse's nonrelational database, elastic tables, to store the cloud flow run history. Cloud flow run history in Dataverse is used by the automation center to provide comprehensive monitoring and troubleshooting experiences for automation processes across Power Automate.

By default, flow run data is stored for 28 days (2,419,200 seconds). If you want to modify the duration of how long the executions can be stored, you can update the Time to live (in seconds) for the flow run in the Organization table in an environment backed with Dataverse. Depending on your environment's storage capacity, you can adjust the length of storage for these run records

Learn more

Create Scopes

As workflows get more complicated, it's essential to manage them well to handle problems, test them, and keep them working smoothly. A Scope is like a container that holds a bunch of actions together. This makes it easier to organize and see distinct parts of your workflow. So, when your workflow grows bigger and more complex, using the Scope tool can make it easier to keep things in order and fix any issues that come up.



Here's how scopes work:

1. **Grouping Actions:** You can add multiple actions inside a scope. This is useful when you have a series of actions that are related to each other or need to be executed together.
2. **Organizing Your Flow:** By using scopes, you can create a hierarchical structure within your flow. You can have nested scopes, where one scope contains another scope, allowing you to break down your workflow into smaller, more manageable sections.

3. **Error Handling:** Scopes are also helpful for error handling. You can set up scopes to catch and handle errors that occur within the actions they contain. This makes it easier to troubleshoot and fix issues in your workflow. Error handling has been discussed in more detail, here
4. **Visibility and Collapsibility:** Scopes can be collapsed to hide their contents, making it easier to focus on other parts of your flow. This helps keep your workspace neat and organized, especially when working with large and complex flows.
5. **Avoid using flows for every single action:** Not every action is required to be within a scope. Please be mindful of overusing scopes action if there is no need.

Note: It's advised to add comments or notes in the scope to make the flows more readable.

Leave Complex Business Logic Out

Many times we have seen use cases where millions of records are required to process via flows. In general while flows can be used to automate day to day jobs, for large scale data transformation or integration, power automate may not be the right tool. Here are some recommendations.

If the requirement is to move or process large number of Dataverse records, consider using Dataverse plugins as custom actions. This will enable users to use low code actions in Dataverse for Power Fx usage.

Please refer to the following section to learn more about when to integrate flows with other solutions to reduce complexity

Additionally, use custom connectors for standalone services like Azure Functions, API Management, Azure App services or use custom connectors with custom code.

Create Reusable Code (Child flows)

Power Automate cloud flows can help automate complex solutions, however, they can grow excessively big and complex really quick making it difficult to navigate and maintain the flow. Instead of building a complex single automation solution, leverage child flows which can add complexity, depth and efficiency to these automation processes. They allow you to break down complex processes into manageable chunks, making your flows more modular and easier to manage.

Why Use Child Flows?

- **Modularity**: Child flows promote modularity, allowing you to build reusable components that can be easily plugged into different workflows. This not only streamlines development but also enhances maintainability.
- **Scalability**: By breaking down your automation into smaller, more manageable pieces, you can scale your processes more effectively. Need to make changes or add functionality? With child flows, it's as simple as updating the individual components.

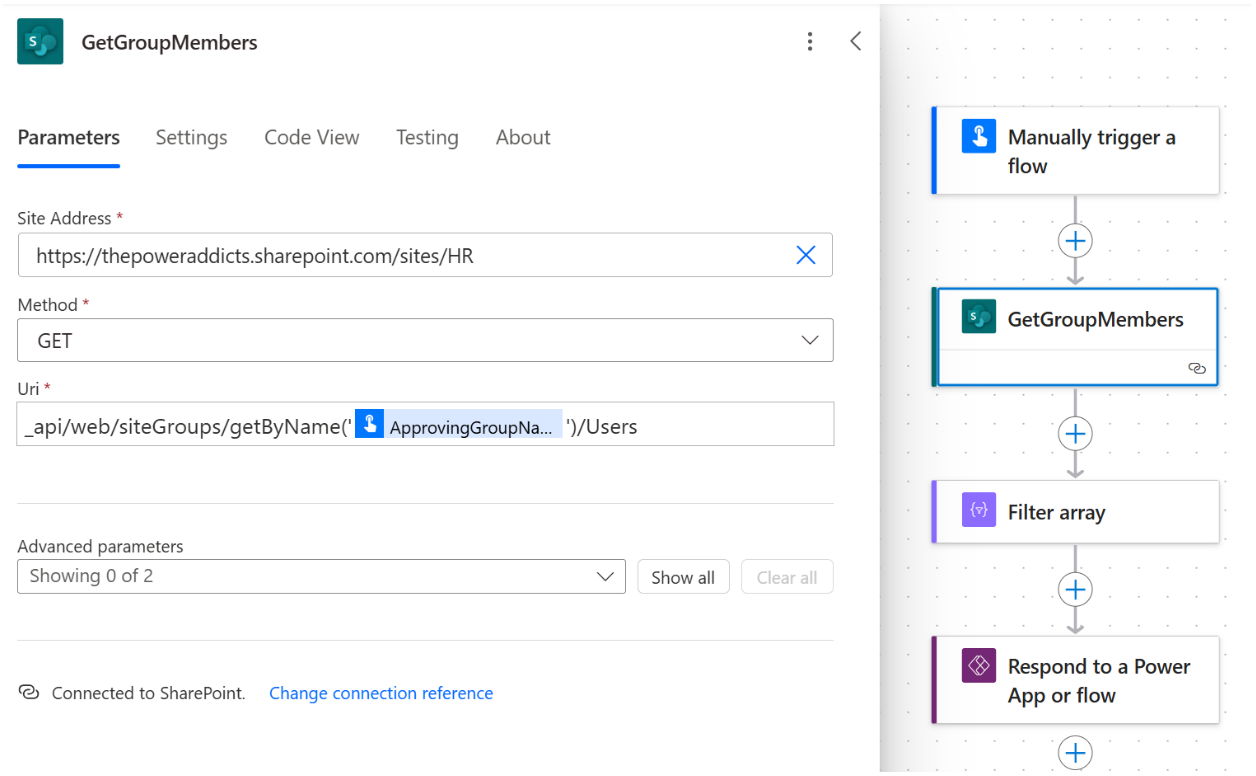
- **Granular Control**: Child flows offer granular control over your automation logic. You can pass data between parent and child flows, enabling dynamic decision-making and personalized experiences.
- **Collaboration**: In a team environment, child flows promote collaboration by allowing team members to work on separate components of a larger automation project simultaneously.

Note: You should create the parent flow and all child flows *directly* in the same solution. Refer to known limitations.

Example Parent/Child Flow Scenario – Contoso organization requires approvals for scenarios like Project Proposal, Leave request and Expense submissions. The approvers are part of different SharePoint groups.

Parent Flows – In this scenario, each parent flow like Project Proposal , Leave request or expense submission can call a single reusable child flow to extract the list of SharePoint approvers from the SP Groups

Child Flow –


GetGroupMembers

Site Address *: https://thepoweraddicts.sharepoint.com/sites/HR

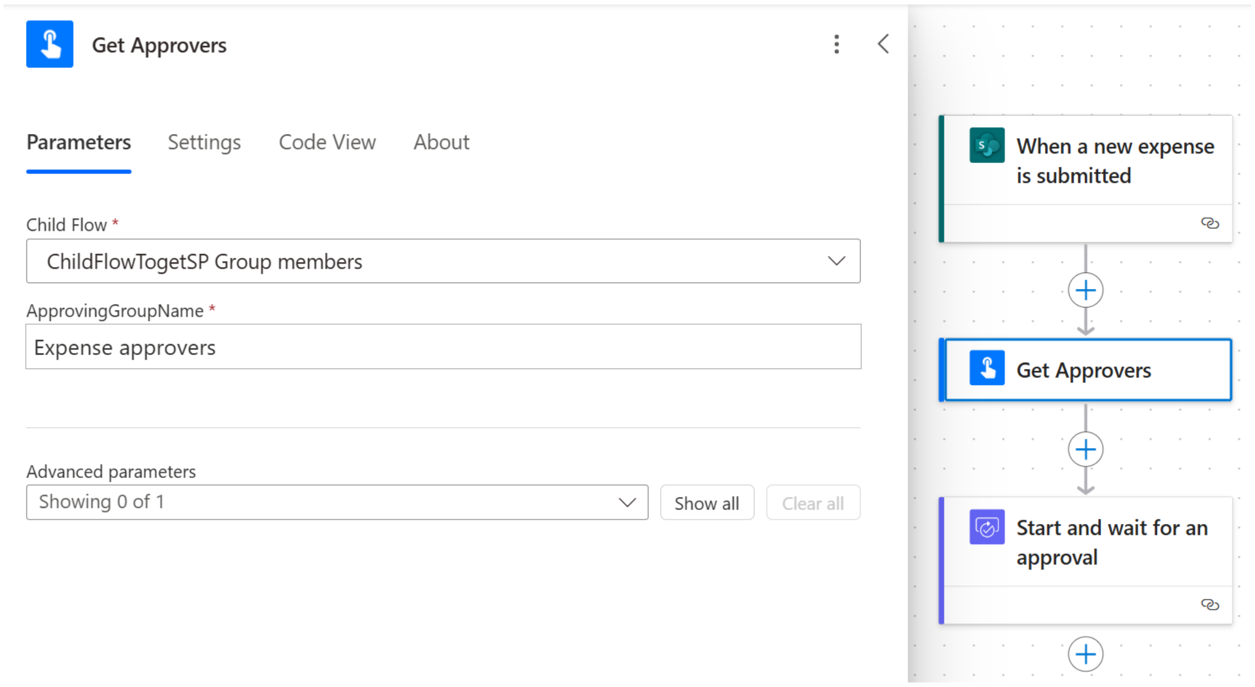
Method *: GET

Uri *: _api/web/siteGroups/getByName(' ApprovingGroupNa... ')/Users

Advanced parameters: Showing 0 of 2

Connected to SharePoint. Change connection reference

Parent Flow -


Get Approvers

Parameters | Settings | Code View | About

Child Flow *
ChildFlowTogetSP Group members

ApprovingGroupName *
Expense approvers

Advanced parameters
Showing 0 of 1

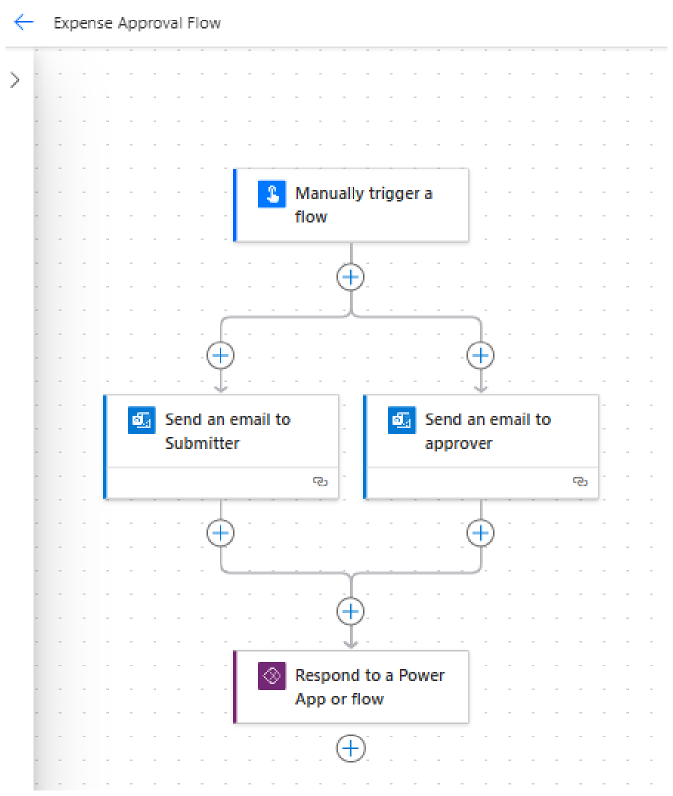
Show all
Clear all
When a new expense is submitted
Get Approvers
Start and wait for an approval

For more details on Child Flows, please refer to the following link – [Child Flows]

Parallel Execution and Concurrency (within For each loop)

Parallel Execution

Power Automate supports parallel execution meaning flows can have two or more steps that run simultaneously, after which the workflow will only priced once all parallel steps have completed.
This enables flow to do more processing at the same time, specifically when the tasks are asynchronous.

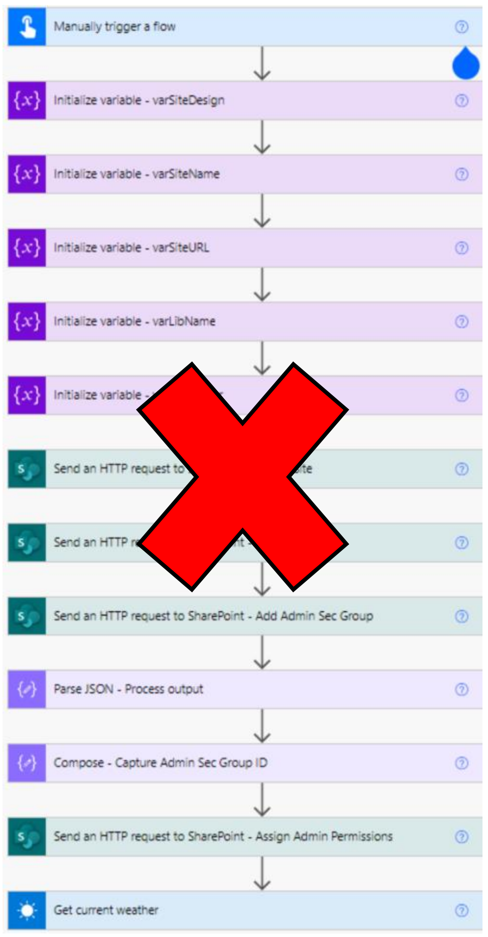

Expense Approval Flow
Manually trigger a flow
Send an email to Submitter
Send an email to approver
Respond to a Power App or flow

A good rule of thumb is to parallelize only when actions **take more than 5 seconds to execute**.
You can use parallel branches to achieve some of the following scenarios:

- Sending non-blocking approval requests (blog post)
- Creating "Quorum" based approvals (blog post)
- Creating or updating records in multiple systems
- Getting data from multiple sources and consolidating them in one

Reducing Scheduling Overhead

Straight line of actions can run sequentially in the engine. Parallel structuring can help organize the flow better and save time. Specially initializing multiple variables can be done in parallel to save time.



Manually trigger a flow

Initialize variable - varSiteDesign

Initialize variable - varSiteName

Initialize variable - varSiteURL

Initialize variable - varLibName

Initialize variable -

Send an HTTP request to te

Send an HTTP r nt

Send an HTTP request to SharePoint - Add Admin Sec Group

Parse JSON - Process output

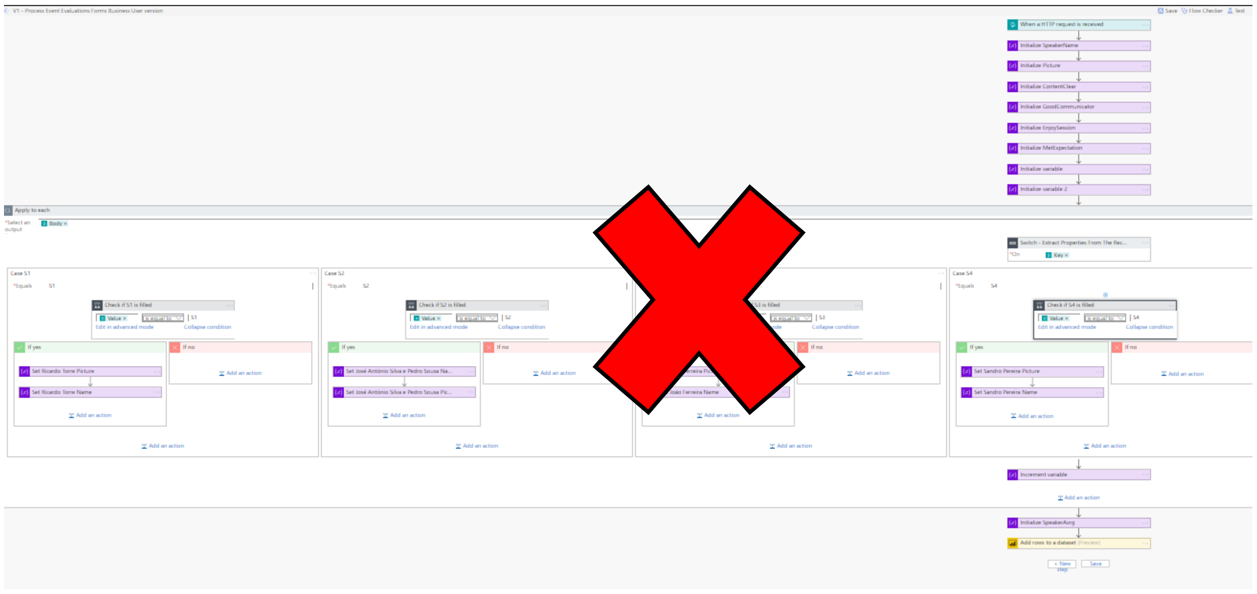
Compose - Capture Admin Sec Group ID

Send an HTTP request to SharePoint - Assign Admin Permissions

Get current weather

Note: One thing to always keep in mind is to avoid huge number of Skipped Actions. Very wide switch statements with lots of actions in each branch or parallel branches where the less used path has significantly more actions and impacts flow readability

Mitigation: Using Child Flows can address this. Instead of having lots of skipped actions in a switch branch, call child flows from the switch branch instead, so we skip the irrelevant child flows instead of a sizeable number of actions. This makes it easier to maintain.



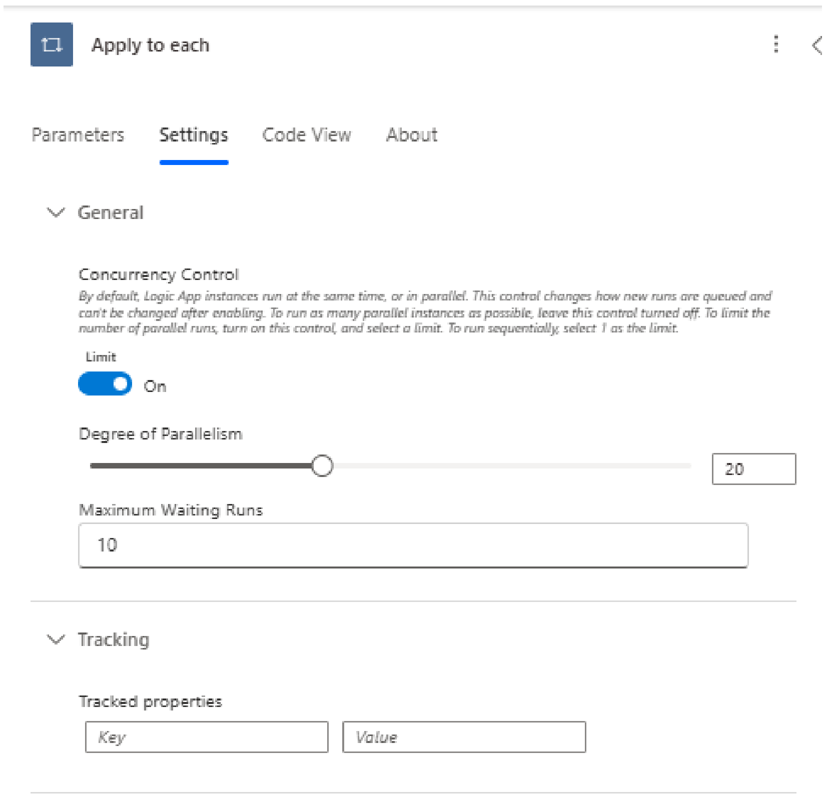
Concurrency Control

Concurrency enables Parallel execution in For Each loop. By default, For each loop executes sequentially. Therefore, when processing large data, it may take a lot of time. If the items in loop are not required to run sequentially, concurrency enables X items to process at once. Currently users are allowed to see the degree of parallelism from 1 to 50.

Consider a scenario where 100 records need to be updated with status as Active, instead of updating one record after the other, Concurrency control enables up to 50 records to be updated simultaneously.

Be mindful of the degree of parallelism to set to

- There is overhead in dividing the work, queueing up extra threads, delays from the endpoint being called, etc.
- High number (i.e. 50) may not necessarily make things go faster



Concurrency Control

By default, Logic App instances run at the same time, or in parallel. This control changes how new runs are queued and can't be changed after enabling. To run as many parallel instances as possible, leave this control turned off. To limit the number of parallel runs, turn on this control, and select a limit. To run sequentially, select 1 as the limit.

Limit: On

Degree of Parallelism: 20

Maximum Waiting Runs: 10

Tracked properties: *Key* *Value*

Some other real-world scenarios where this approach is applicable:

- Sending individual emails to many recipients
- Updating records in Dataverse, SharePoint Lists, SQL
- Bulk creating users in Azure Active directory
- Creating parallelized approvals (more details)
- Fanning out operations in SharePoint (more details)

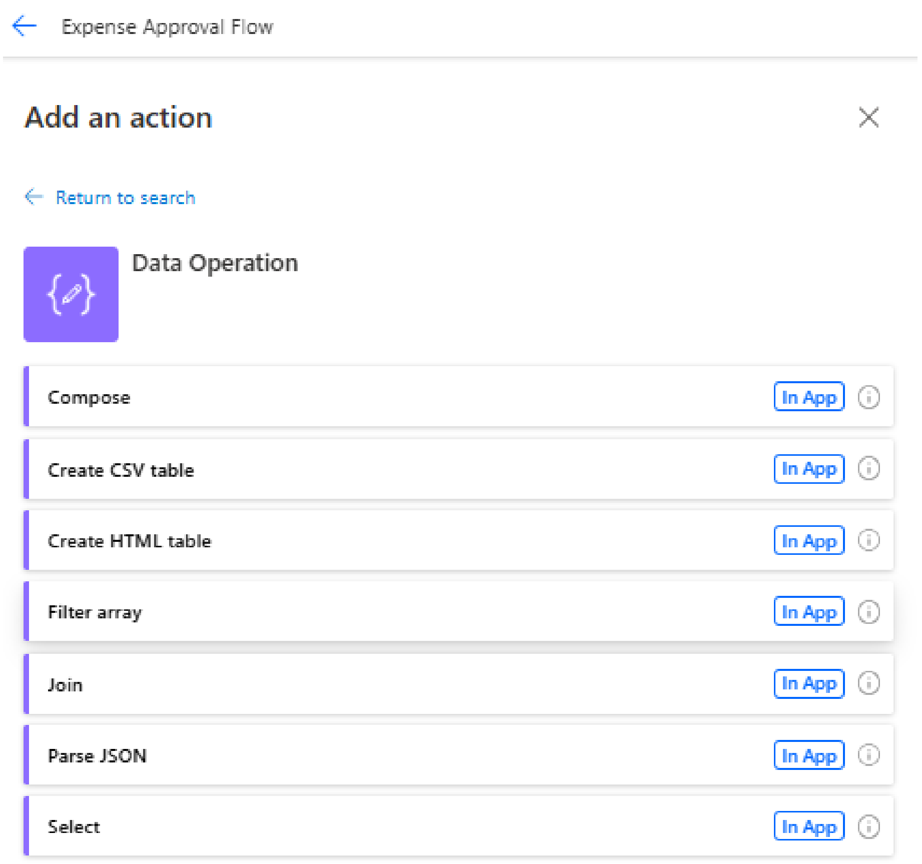
Below is a comparison of the impact of concurrency control for array processing within for each loop.

Array Length	Degree of Parallelism	Time taken to run loop
4	Off	21 seconds
4	2	11 seconds
4	4	6 seconds
4	6	6 seconds

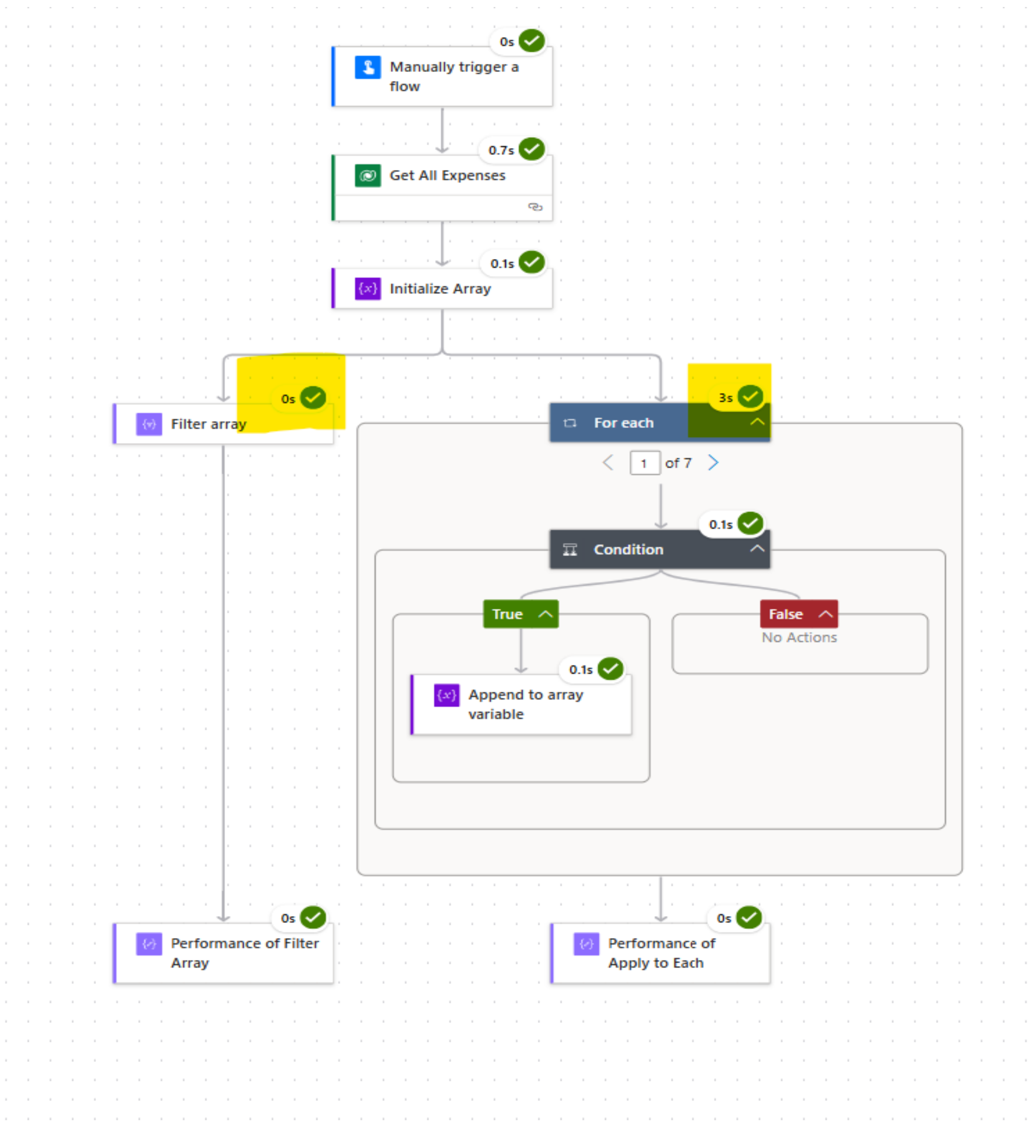
Note that concurrency control for ‘Apply to each’ actions only take effect on the highest level in the cloud flow. When nesting Apply to Each actions, the inner actions will always execute serially.

Working with Data Operations

Data operations in Power Automate refer to the manipulation, transformation, and management of data within your automation flows in an efficient way.
Actions like Filter Array, Select and Join actions can help avoid any unnecessary loops to process data. For example, whenever there is a need to append to variables to make a final array in order to do any filtering, if the element doesn't come from information from external data source, a lot of time they can be simplified by data operations like filter or select.



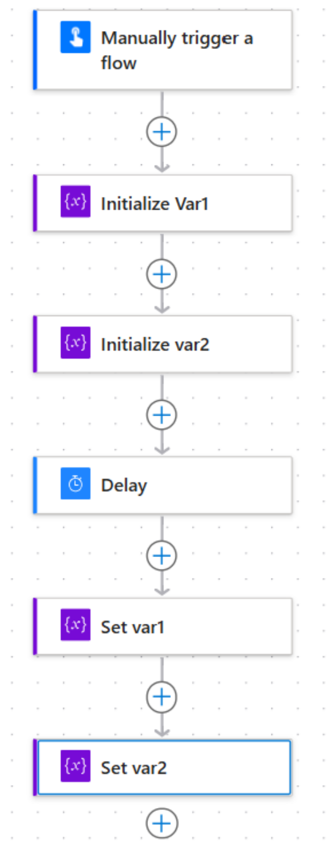
Here is how the flow execution looks like when using append variable within a loop compared to Filter Array. In the example below to filter an array of 100 records, the Filter action took a few milliseconds compared to Append to array within a loop which took 48 seconds and added more actions and complexity to the flow.



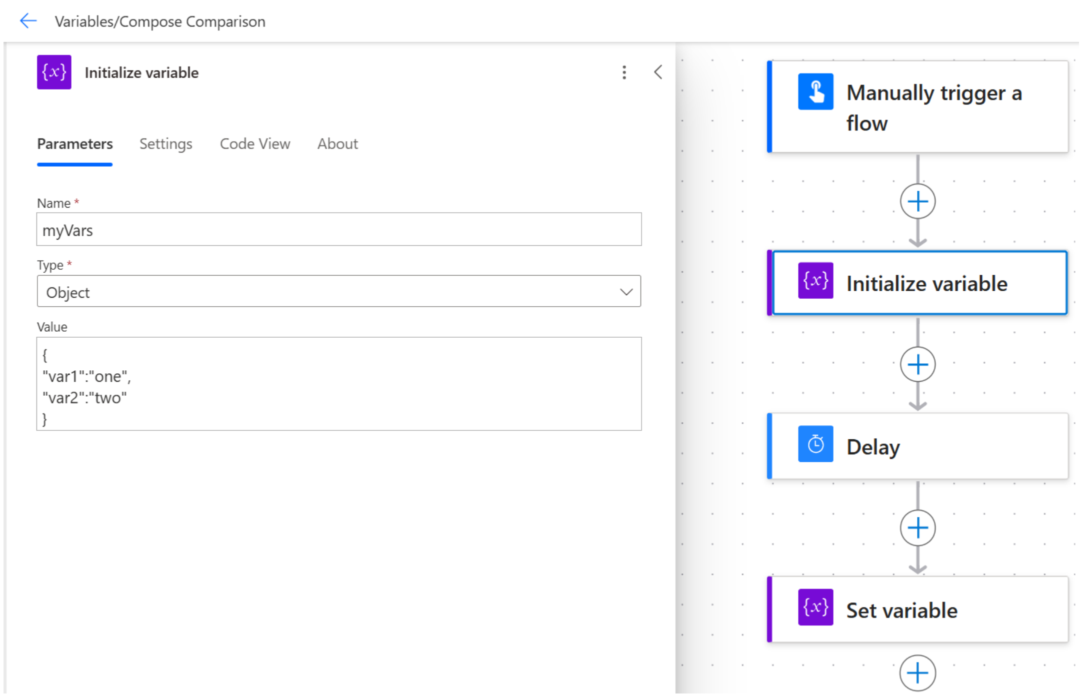
Variables vs Compose

Compose are not update-able at run-time. It is useful for write-once, read-many type of usage. If we need to write or update in many places in the flow, variables will be much more appropriate. Similarly, using compose to create variables can also perform faster than initializing+ Declare variables.

When working with a set of variables that are updated within the same logical blocks in the flow, you can use JSON variables instead of individual variables. This help reduces the number of actions in the flow.



Consider the above: Instead of working with Var1 and Var2 as separate variables, we can use 1 Object Variable to manage both values.



Name *

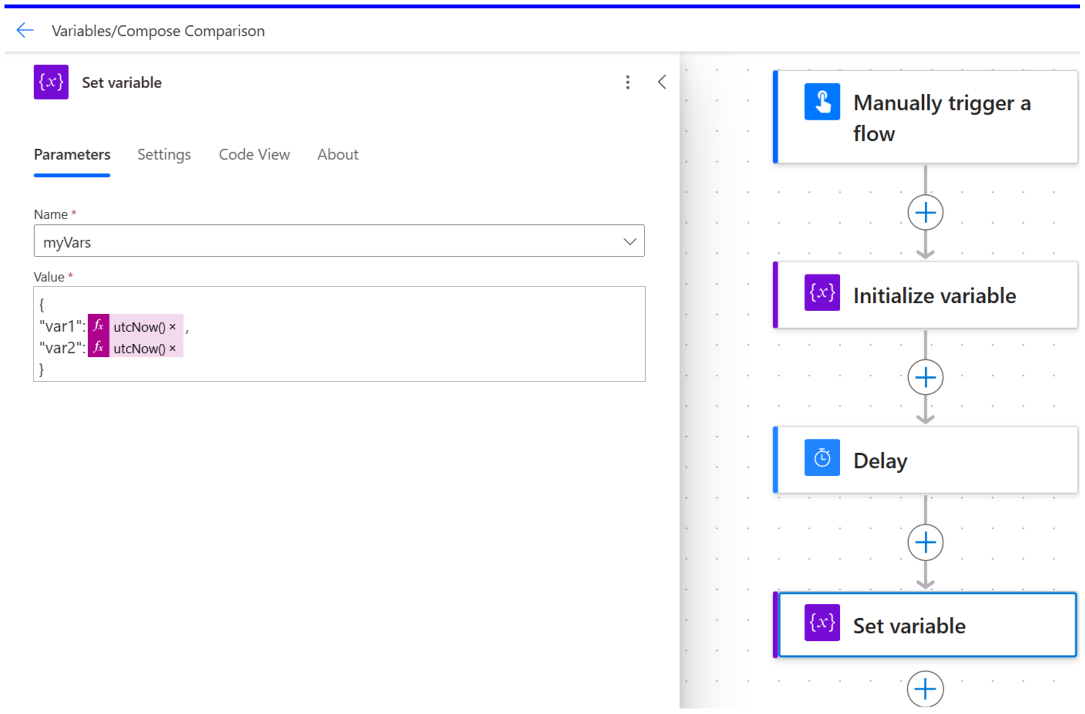
myVars

Type *

Object

Value

```
{
"var1":"one",
"var2":"two"
}
```


Variables/Compose Comparison

Set variable

Parameters Settings Code View About

Name *
myVars

Value *
```
{
"var1": utcNow() ,
"var2": utcNow()
}
```

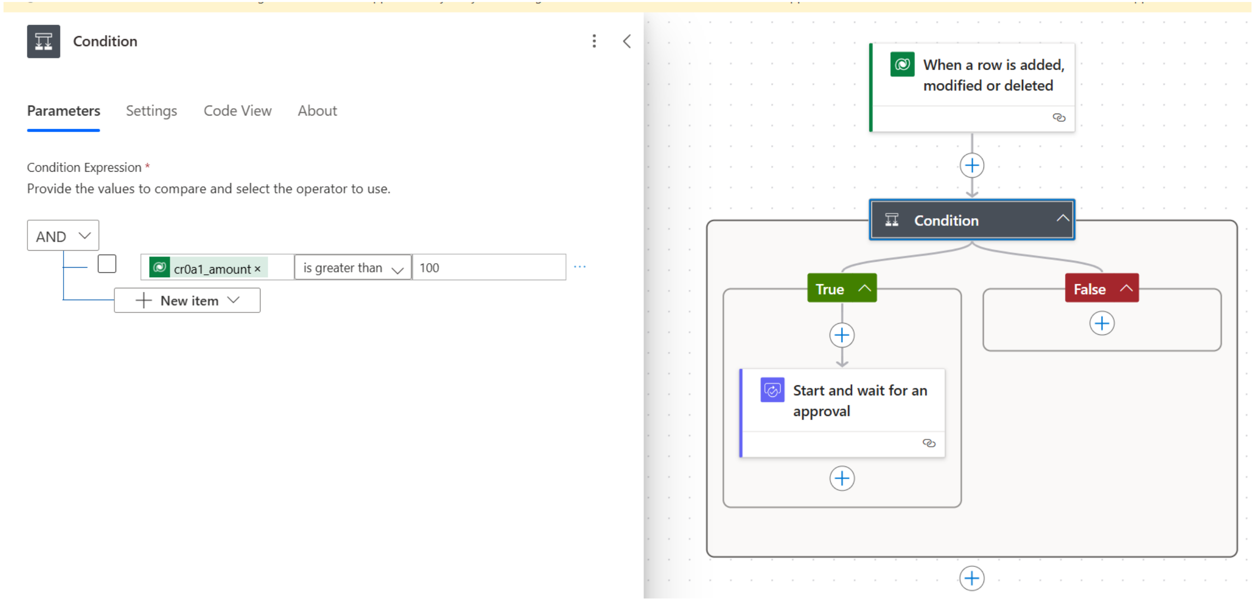
Optimize Power Automate Triggers

Set Trigger Conditions

A common issue for many users who work with Power Automate is that their flow runs whenever a new row or an existing row is modified in the data source. However, we only need the flow to execute when a specific condition is satisfied. Setting triggers correctly can enable your flow to run only when necessary.

Consider a scenario where any user who submits an expense above $5000 need to get approved by their management. If the trigger conditions are not specified, the flow will run for every expense that is submitted and flow author will have to specify additional conditions to filter the expense greater than $5000. With trigger conditions, the flow will only trigger when the expense is above $5000.

Here is the flow for above scenario without trigger conditions:


Condition

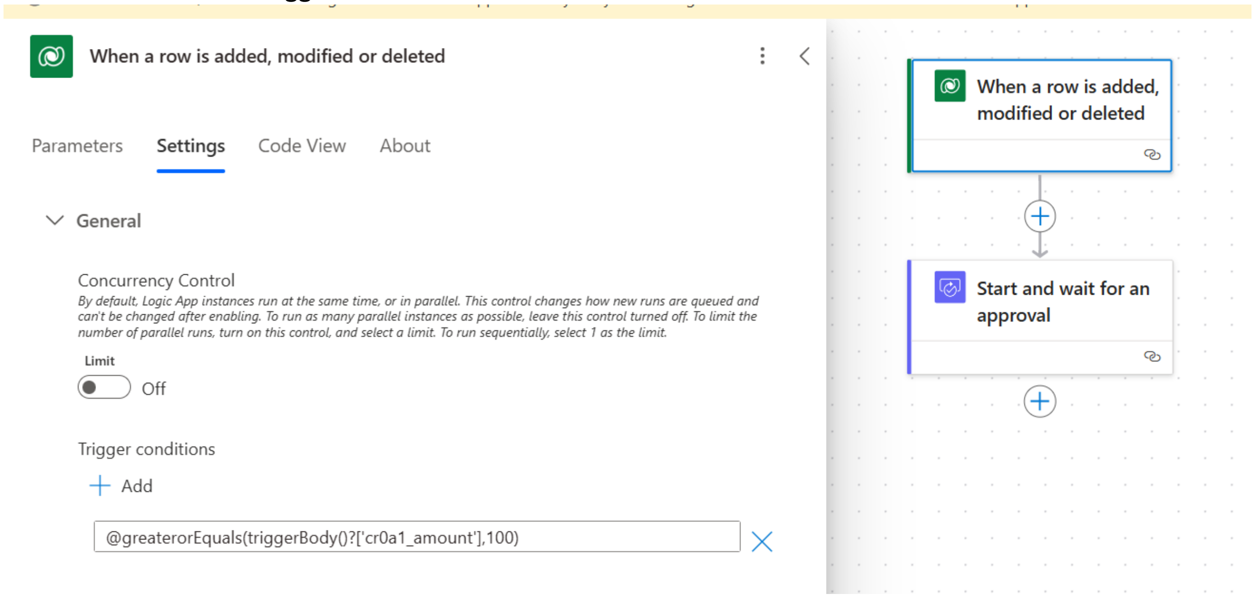
Parameters Settings Code View About

Condition Expression *
Provide the values to compare and select the operator to use.

AND | cr0a1_amount | is greater than | 100

New item
When a row is added, modified or deleted
Condition
True
False
Start and wait for an approval

Here is the flow with trigger conditions:


When a row is added, modified or deleted

Parameters Settings Code View About

General

Concurrency Control
By default, Logic App instances run at the same time, or in parallel. This control changes how new runs are queued and can't be changed after enabling. To run as many parallel instances as possible, leave this control turned off. To limit the number of parallel runs, turn on this control, and select a limit. To run sequentially, select 1 as the limit.

Limit: Off

Trigger conditions

```
@greaterorEquals(triggerBody()?['cr0a1_amount'],100)
```

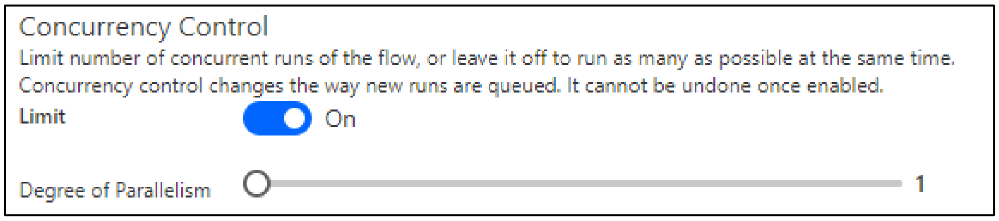
Add
When a row is added, modified or deleted
Start and wait for an approval

Note: In Dataverse triggers, OData filter property also allows the trigger to fire only when the filter conditions are met.
You can also specify the names of the columns on which the flow can trigger.

In some rare scenarios where your flow may work with data sources with very limited through-put, you may consider configuring the trigger's concurrency control. By default, the Cloud Flow trigger will execute as many runs as possible at the same time when its conditions are met. We can change this behavior by changing its Concurrency Control, with the ability to limit from a minimum of 1 up to 100 runs at the same time. Any more runs will be queued automatically.

For example, we may perform an automation that has a dependency on an on-premise resource that does not support parallel executions, or to prevent race-conditions where a dirty-read is possible when there are parallel executions.

Note that once applied, this cannot be undone. To remove the concurrency control, you will need to create a new flow. Therefore, you must do this with caution. The best practice is to leave it as default, but in the event where this is required, do this on a flow that has the least number of actions – for example you may organize your actions that work with data sources that requires such control to be in a dedicated child flow so you can apply this control only in the child flow.

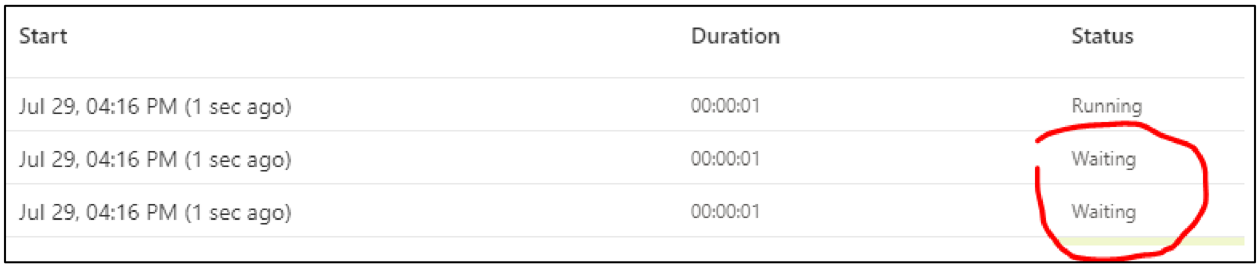


Concurrency Control

Limit number of concurrent runs of the flow, or leave it off to run as many as possible at the same time. Concurrency control changes the way new runs are queued. It cannot be undone once enabled.

Limit: On

Degree of Parallelism: 1



Start	Duration	Status
Jul 29, 04:16 PM (1 sec ago)	00:00:01	Running
Jul 29, 04:16 PM (1 sec ago)	00:00:01	Waiting
Jul 29, 04:16 PM (1 sec ago)	00:00:01	Waiting

Working with relevant data

In Power Automate, working with only relevant data is crucial for efficiency and accuracy in your automated workflows. In the Limits and configuration, we have specified throughput limits as follows:

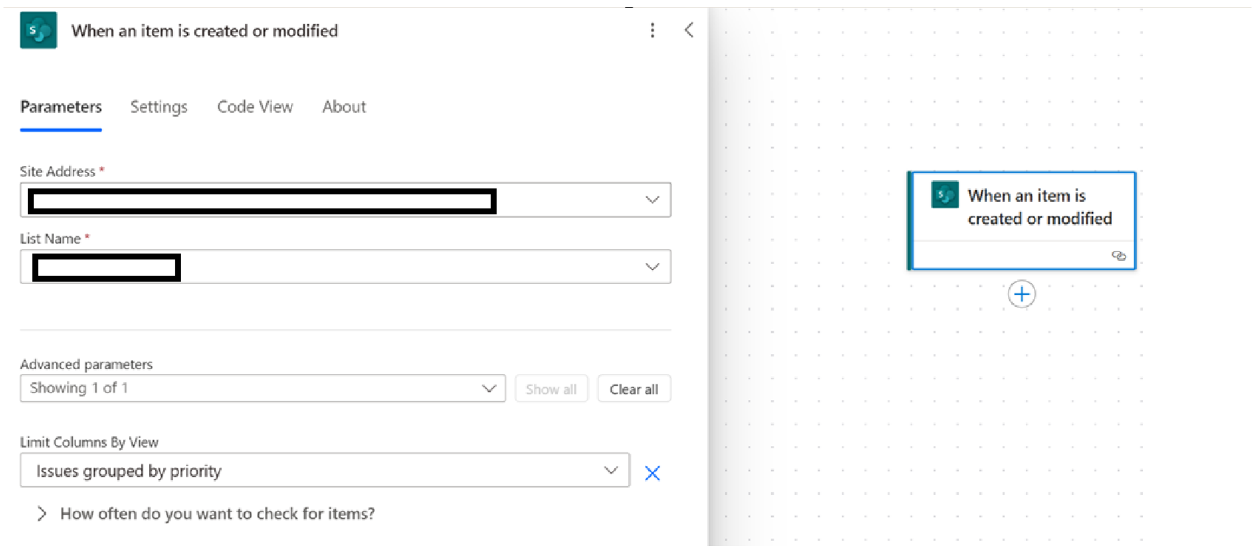
Name	Limit	Notes
Content throughput per 5 minutes	120 MB for Low; 1.2 GB for all others	You can distribute workload across more than one flow as necessary.
Content throughput per 24 hours	200 MB for Low; 2 GB for Medium; 10 GB for High	You can distribute workload across more than one flow as necessary.

Therefore, its important to limit the data that we bring in only the relevant data. This could be done both at the trigger as well as Action levels. In most cases, you can specify Odata expressions, top count and relevant column names at the trigger and action level.

Here are some examples on how to do this:

Triggers –

Data sources like SharePoint allows to specify views to get only relevant columns and filtered records. In trigger, users can specify the view which are relevant to their use case and get only relevant data and selected columns.



When an item is created or modified

Parameters | Settings | Code View | About

Site Address *
[redacted]

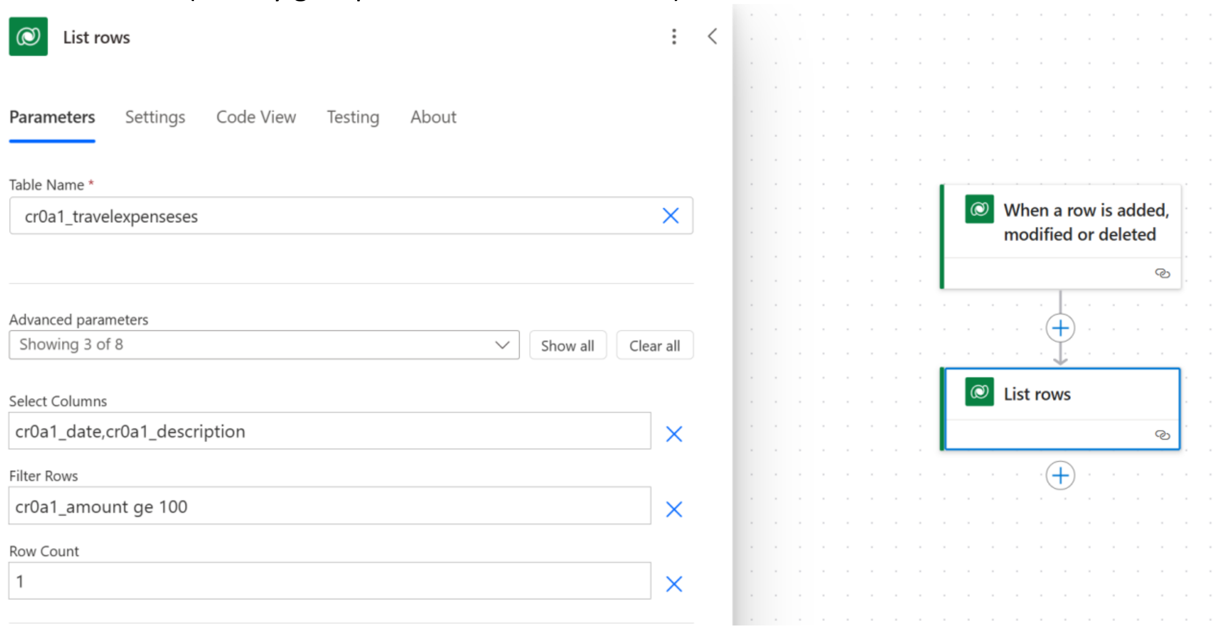
List Name *
[redacted]

Advanced parameters
Showing 1 of 1

Limit Columns By View
Issues grouped by priority

How often do you want to check for items?

Actions-
Use Advanced options like Select columns (only allowing flow to get data corresponding to column names specified in the query), Filter rows (to get data based on the Filter condition) and Row count (to only get specified number of rows)



List rows

Parameters | Settings | Code View | Testing | About

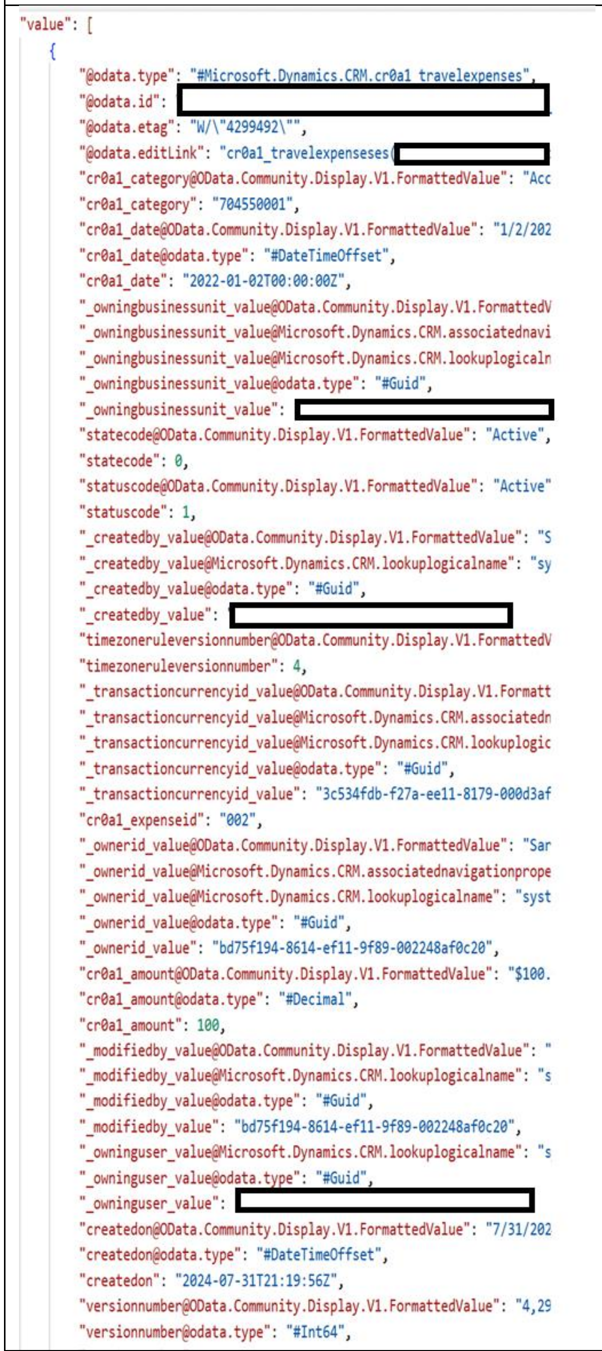
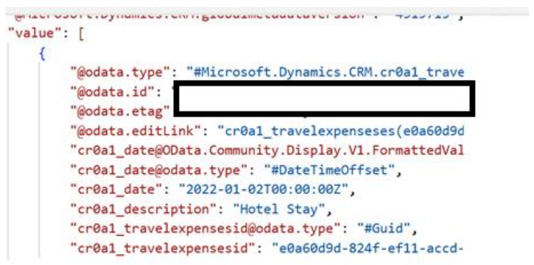
Table Name *
cr0a1_travelexpenseses

Advanced parameters
Showing 3 of 8

Select Columns
cr0a1_date,cr0a1_description

Filter Rows
cr0a1_amount ge 100

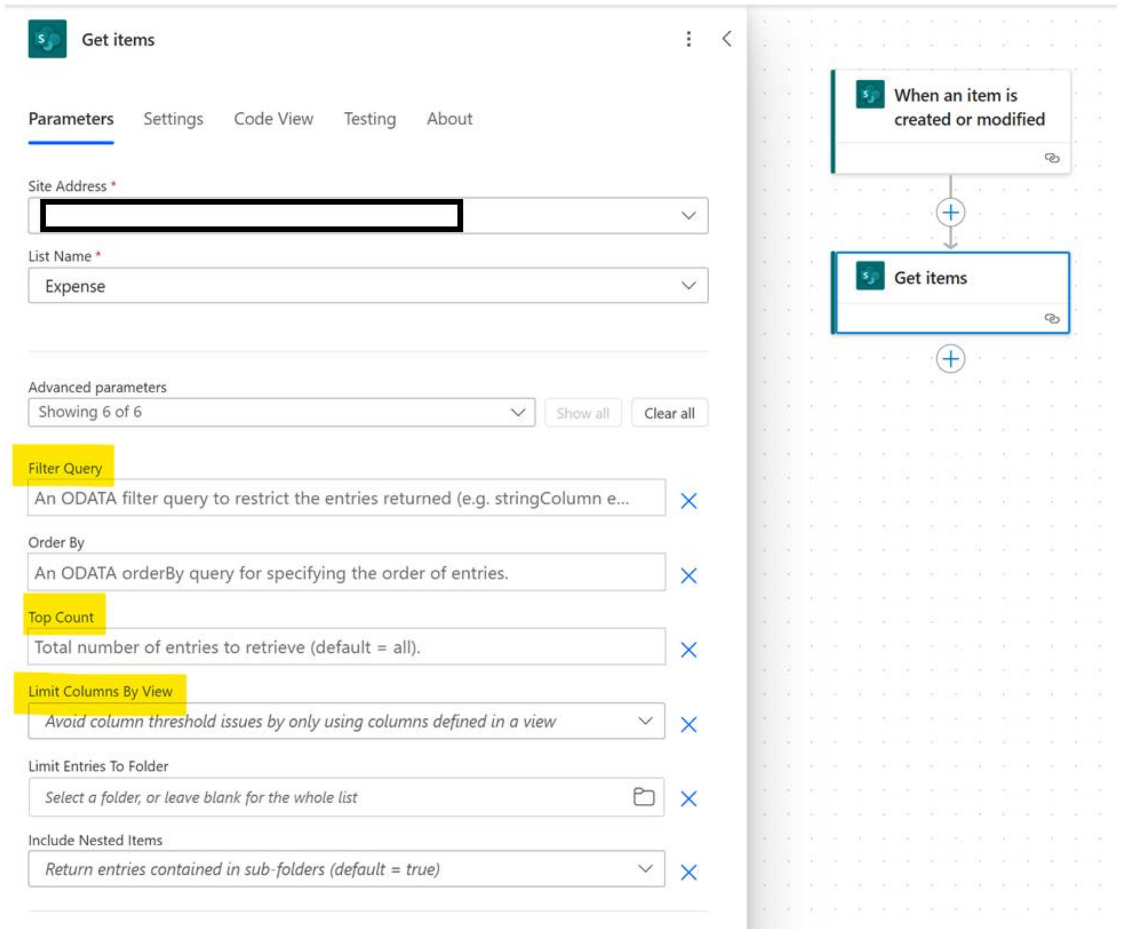
Row Count
1

Result without Select Column	Result with Select Column
	

```
"value": [
    {
        "@odata.type": "#Microsoft.Dynamics.CRM.cr0a1_travelexpenses",
        "@odata.id": [redacted]
        "@odata.etag": "W/\"4299492\"",
        "@odata.editLink": "cr0a1_travelexpenseses([redacted]
        "cr0a1_category@OData.Community.Display.V1.FormattedValue": "Acc
        "cr0a1_category": "704550001",
        "cr0a1_date@OData.Community.Display.V1.FormattedValue": "1/2/202
        "cr0a1_date@odata.type": "#DateTimeOffset",
        "cr0a1_date": "2022-01-02T00:00:00Z",
        "_owningbusinessunit_value@OData.Community.Display.V1.FormattedV
        "_owningbusinessunit_value@Microsoft.Dynamics.CRM.associatednavi
        "_owningbusinessunit_value@Microsoft.Dynamics.CRM.lookuplogicaln
        "_owningbusinessunit_value@odata.type": "#Guid",
        "_owningbusinessunit_value": [redacted]
        "statecode@OData.Community.Display.V1.FormattedValue": "Active",
        "statecode": 0,
        "statuscode@OData.Community.Display.V1.FormattedValue": "Active"
        "statuscode": 1,
        "_createdby_value@OData.Community.Display.V1.FormattedValue": "S
        "_createdby_value@Microsoft.Dynamics.CRM.lookuplogicalname": "sy
        "_createdby_value@odata.type": "#Guid",
        "_createdby_value": [redacted]
        "timezoneruleversionnumber@OData.Community.Display.V1.FormattedV
        "timezoneruleversionnumber": 4,
        "_transactioncurrencyid_value@OData.Community.Display.V1.Formatt
        "_transactioncurrencyid_value@Microsoft.Dynamics.CRM.associatedn
        "_transactioncurrencyid_value@Microsoft.Dynamics.CRM.lookuplogic
        "_transactioncurrencyid_value@odata.type": "#Guid",
        "_transactioncurrencyid_value": "3c534fdb-f27a-ee11-8179-000d3af
        "cr0a1_expenseid": "002",
        "_ownerid_value@OData.Community.Display.V1.FormattedValue": "Sar
        "_ownerid_value@Microsoft.Dynamics.CRM.associatednavigationprope
        "_ownerid_value@Microsoft.Dynamics.CRM.lookuplogicalname": "syst
        "_ownerid_value@odata.type": "#Guid",
        "_ownerid_value": "bd75f194-8614-ef11-9f89-002248af0c20",
        "cr0a1_amount@OData.Community.Display.V1.FormattedValue": "$100.
        "cr0a1_amount@odata.type": "#Decimal",
        "cr0a1_amount": 100,
        "_modifiedby_value@OData.Community.Display.V1.FormattedValue": "
        "_modifiedby_value@Microsoft.Dynamics.CRM.lookuplogicalname": "s
        "_modifiedby_value@odata.type": "#Guid",
        "_modifiedby_value": "bd75f194-8614-ef11-9f89-002248af0c20",
        "_owninguser_value@Microsoft.Dynamics.CRM.lookuplogicalname": "s
        "_owninguser_value@odata.type": "#Guid",
        "_owninguser_value": [redacted]
        "createdon@OData.Community.Display.V1.FormattedValue": "7/31/202
        "createdon@odata.type": "#DateTimeOffset",
        "createdon": "2024-07-31T21:19:56Z",
        "versionnumber@OData.Community.Display.V1.FormattedValue": "4,29
        "versionnumber@odata.type": "#Int64",
```

```
"value": [
    {
        "@odata.type": "#Microsoft.Dynamics.CRM.cr0a1_trave
        "@odata.id": [redacted]
        "@odata.etag"
        "@odata.editLink": "cr0a1_travelexpenseses(e0a60d9d
        "cr0a1_date@OData.Community.Display.V1.FormattedVal
        "cr0a1_date@odata.type": "#DateTimeOffset",
        "cr0a1_date": "2022-01-02T00:00:00Z",
        "cr0a1_description": "Hotel Stay",
        "cr0a1_travelexpensesid@odata.type": "#Guid",
        "cr0a1_travelexpensesid": "e0a60d9d-824f-ef11-accd-
```

In SharePoint, use the following parameters, to work with relevant data.



Get items

Parameters Settings Code View Testing About

Site Address *

[redacted]

List Name *

Expense

Advanced parameters

Showing 6 of 6

Show all Clear all

Filter Query

An ODATA filter query to restrict the entries returned (e.g. stringColumn e...

Order By

An ODATA orderBy query for specifying the order of entries.

Top Count

Total number of entries to retrieve (default = all).

Limit Columns By View

Avoid column threshold issues by only using columns defined in a view

Limit Entries To Folder

Select a folder, or leave blank for the whole list

Include Nested Items

Return entries contained in sub-folders (default = true)

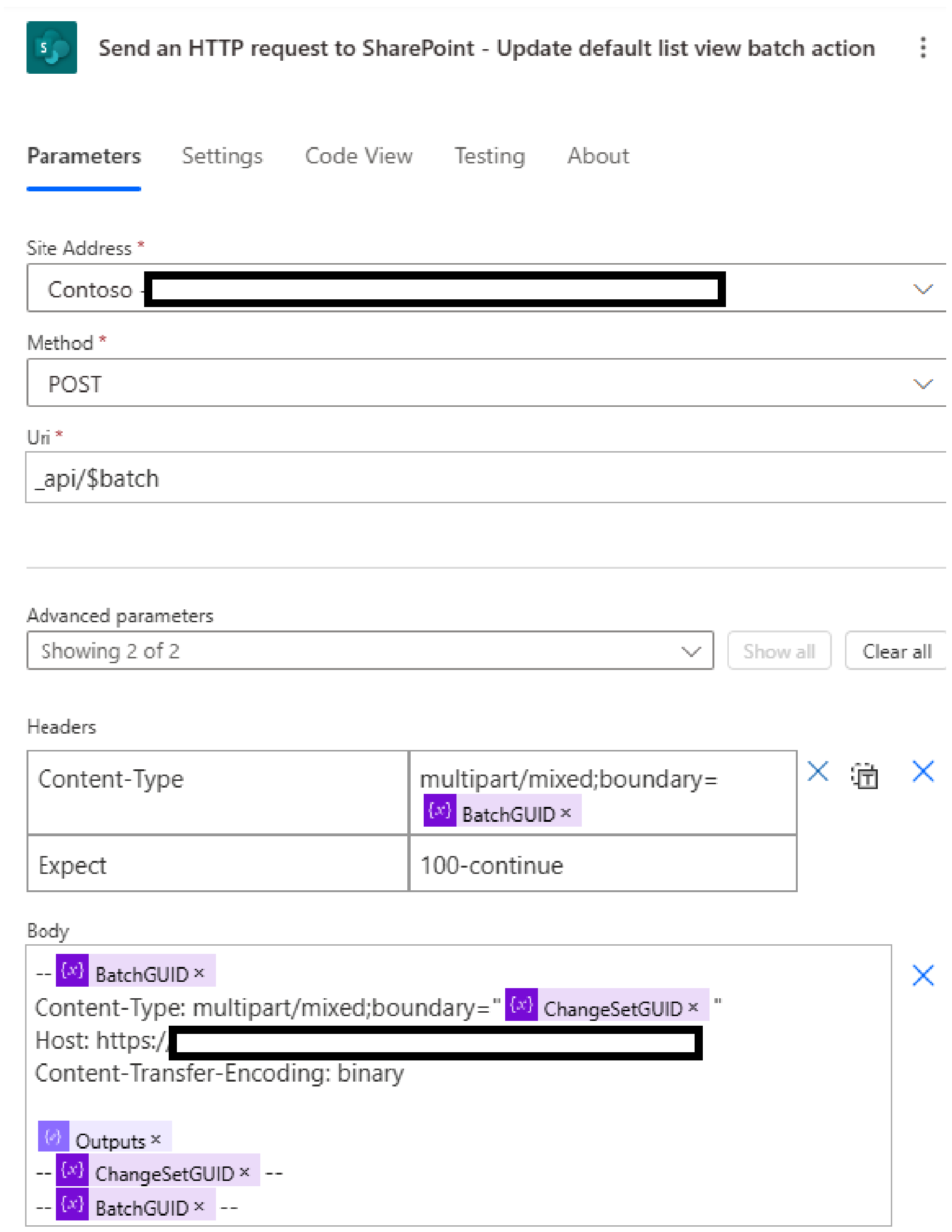
Use Batch where Possible, Max Out Page Size

Often users find using Power Automate to create/updated 1000s of records to the data source when a flow is triggered. Most user would end up using For Each loop which will go through each of these 1000s records, and push them to the data source sequentially causing latency and delays.

To address this, you can try following 2 approaches:

- Create/Patch records to the data source in BATCH. Most of the connectors/services expose API services to post request in BATCH. You can group multiple operations into a single HTTP request using a batch operation. These operations are performed sequentially in the order they're specified. The order of the responses match the order of the requests in the batch operation.
- Parallelism in For Each loop provides up to 50 records to be processed for the services which do not have capability to accept BATCH requests.

Refer to the following links of rest APIs from Sharepoint and Dataverse which shows how to make batch requests.


Send an HTTP request to SharePoint - Update default list view batch action

Site Address *
Contoso [redacted]

Method *
POST

Uri *
_api/$batch

Advanced parameters
Showing 2 of 2

Headers

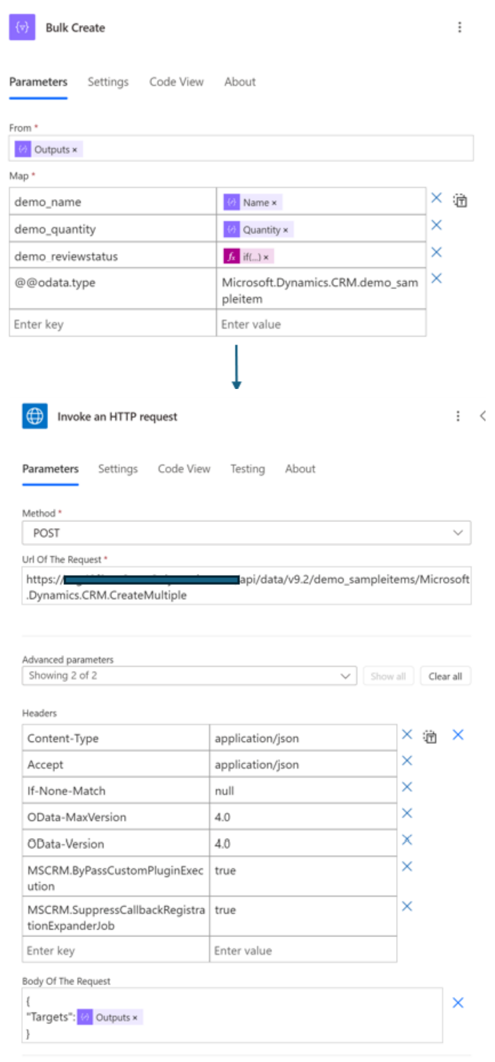
Content-Type	multipart/mixed;boundary= BatchGUID
Expect	100-continue

Body

```
-- BatchGUID
Content-Type: multipart/mixed;boundary=" ChangeSetGUID "
Host: https:/[redacted]
Content-Transfer-Encoding: binary

Outputs
-- ChangeSetGUID --
-- BatchGUID --
```

When working with Dataverse, you can take advantage of the new Bulk Operations Web APIs. Bulk Operations API differs from Batch Operations – while Batch Operations are posted in a single request, it is executed as multiple operations. Bulk Operations are posted in a single request and counted as a single Operation. Bulk Operations Web API can be invoked using HTTP with Entra ID, or with HTTP connector when using with Service Principals. Additionally, Using Dataverse Bulk Operations to reduce number of actions.



Bulk Create

Parameters | Settings | Code View | About

From *: Outputs

Map *

demo_name	Name
demo_quantity	Quantity
demo_reviewstatus	if(...)
@@odata.type	Microsoft.Dynamics.CRM.demo_sampleitem
Enter key	Enter value

Invoke an HTTP request

Parameters | Settings | Code View | Testing | About

Method *: POST

Url Of The Request *: https://[redacted]api/data/v9.2/demo_sampleitems/Microsoft.Dynamics.CRM.CreateMultiple

Advanced parameters: Showing 2 of 2

Headers

Content-Type	application/json
Accept	application/json
If-None-Match	null
OData-MaxVersion	4.0
OData-Version	4.0
MSCRM.ByPassCustomPluginExecution	true
MSCRM.SuppressCallbackRegistrationExpanderJob	true
Enter key	Enter value

Body Of The Request

```
{
"Targets": Outputs
}
```

In the example above, we prepare the records in JSON format using Select action, and use HTTP with Entra ID to post the request using CreateMultiple Web API. If we have 100 records in the JSON output, this will only incur 1 single action instead of 100 Create Row actions in Dataverse.

Avoid Nested For-Each Loops

Nested "For Each" loops can be an expensive operation in Power Automate (and in many other programming contexts) due to their potential impact on performance and resource consumption.

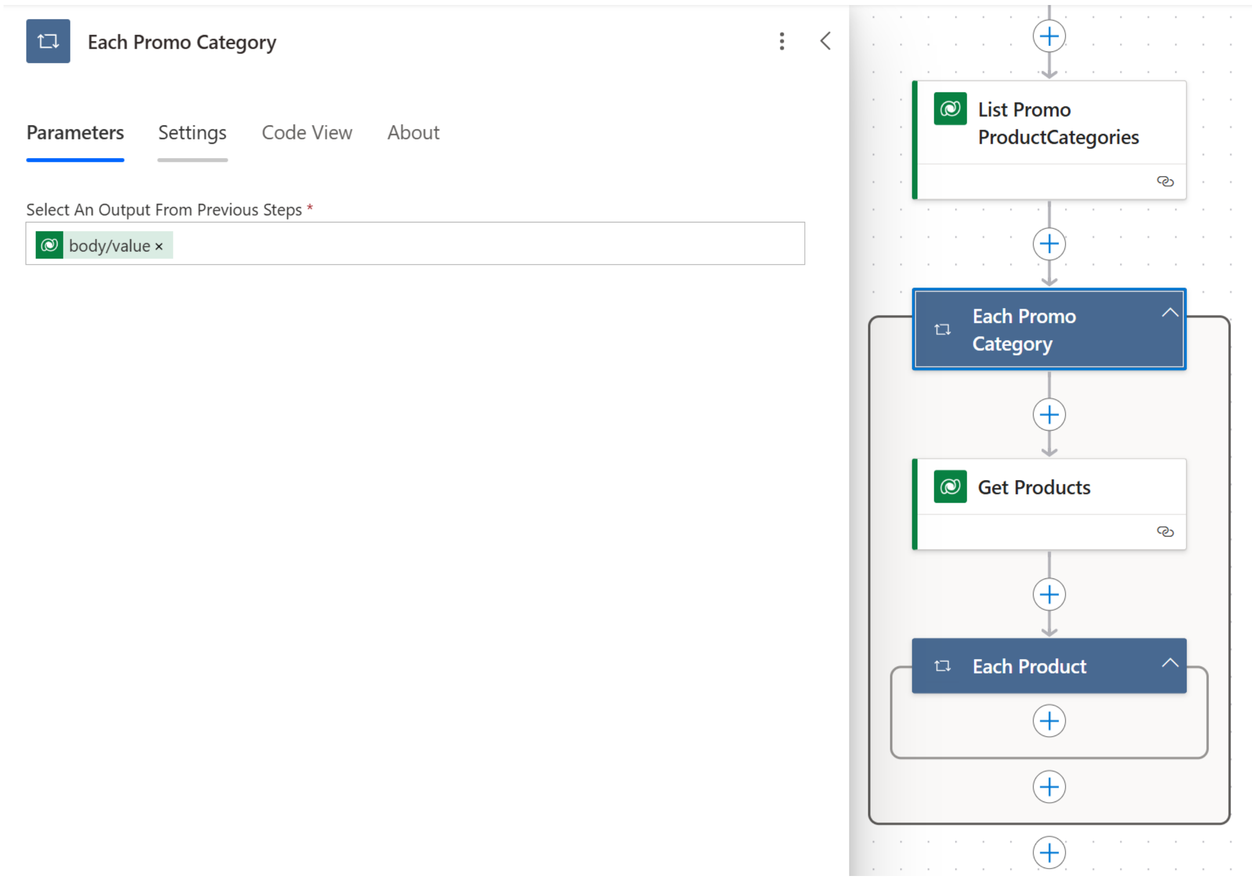
- **Increased Execution Time**: With nested loops, the number of iterations multiply.
  For example, if you have 2 loops, each with 10 iterations, the total number of iterations becomes 10 x 10 = 100. This can significantly increase the execution time of your flow, especially if the loops process large datasets or perform complex operations within each iteration.
- **Limits and Quotas**: Power Automate imposes limits and quotas on various aspects of flow execution, such as the number of iterations allowed within a loop or the maximum execution

time. Nested loops can quickly reach these limits, causing the flow to fail or be throttled, particularly if it operates on large datasets or runs frequently.

Learn more about the limits here.

Depending on your scenarios – you can avoid nested loops with the following alternatives.

- When using nested loops to process related records from a parent table: For example, an outer loop may use List Rows in Dataverse for ProductCategory table to retrieve list of Product Categories where a column IsPromotion = true, and an inner loop is used to loop through the related records in Product table that belongs to a given category from the first loop.


Each Promo Category

Select An Output From Previous Steps *
body/value

A more efficient implementation is to use OData Query Expansion where we only need to work with a single Apply-for-Each loop. This also reduces the total requests against Dataverse to just one RetrieveMultiple call. Use the Expand Query parameter to specify the lookup column name. We can also add ($select=…) to reduce the number of columns returned from the related table. The filter rows parameter can be used to apply conditions on the lookup table’s column.

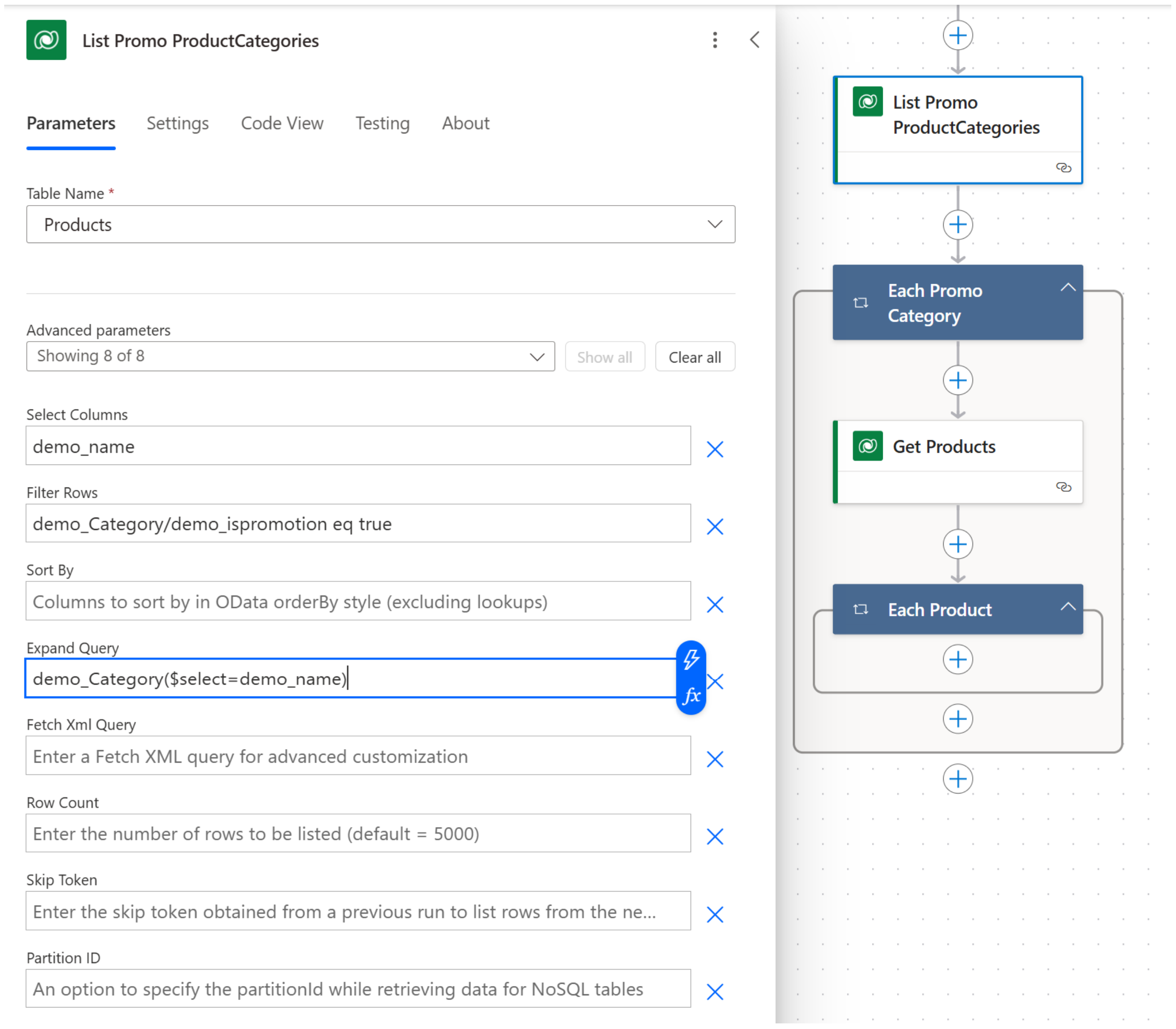

List Promo ProductCategories

Table Name *
Products

Advanced parameters
Showing 8 of 8

Select Columns
demo_name

Filter Rows
demo_Category/demo_ispromotion eq true

Sort By
Columns to sort by in OData orderBy style (excluding lookups)

Expand Query
demo_Category($select=demo_name)

Fetch Xml Query
Enter a Fetch XML query for advanced customization

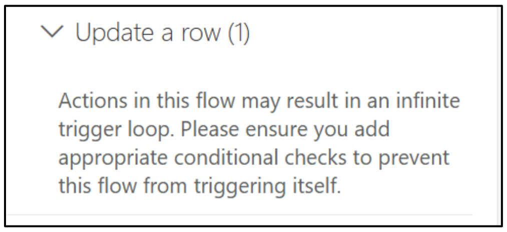
Row Count
Enter the number of rows to be listed (default = 5000)

Skip Token
Enter the skip token obtained from a previous run to list rows from the ne...

Partition ID
An option to specify the partitionId while retrieving data for NoSQL tables

Avoid Infinite Runs

With Power Automate, its very easy to run into a trap where the flows start to trigger infinitely specially when its required to update the same table on which the flow is supposed to be triggered.
Note that the Maker will get a warning when saving a flow that can result in an infinite trigger loop.

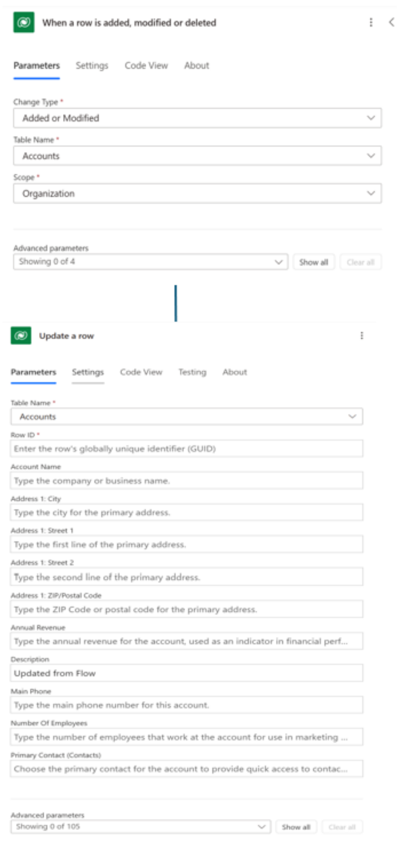

Update a row (1)

Actions in this flow may result in an infinite trigger loop. Please ensure you add appropriate conditional checks to prevent this flow from triggering itself.

To address infinite runs problem, you can follow given steps:

- **Use Trigger Conditions to avoid flow infinite loops -** Incorporate trigger conditions within your flow to check if certain criteria are met before proceeding with subsequent actions. This ensures that the flow only executes when necessary, reducing the likelihood of triggering an infinite loop. Alternative is to use terminate the flow when condition causing the loop is met.

Example of a flow which could run infinitely



When a row is added, modified or deleted

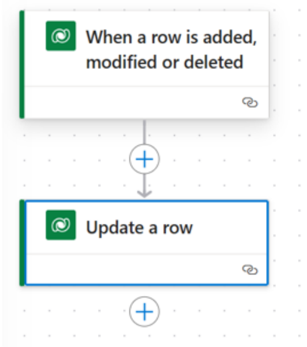
Change Type *: Added or Modified
Table Name *: Accounts
Scope *: Organization

Advanced parameters: Showing 0 of 4

Update a row

Table Name *: Accounts
Row ID *: Enter the row's globally unique identifier (GUID)
Account Name: Type the company or business name.
Address 1: City: Type the city for the primary address.
Address 1: Street 1: Type the first line of the primary address.
Address 1: Street 2: Type the second line of the primary address.
Address 1: ZIP/Postal Code: Type the ZIP Code or postal code for the primary address.
Annual Revenue: Type the annual revenue for the account, used as an indicator in financial perf...
Description: Updated from Flow
Main Phone: Type the main phone number for this account.
Number Of Employees: Type the number of employees that work at the account for use in marketing ...
Primary Contact (Contacts): Choose the primary contact for the account to provide quick access to contac...

Advanced parameters: Showing 0 of 105



Example of how you can stop the flow from running infinitely

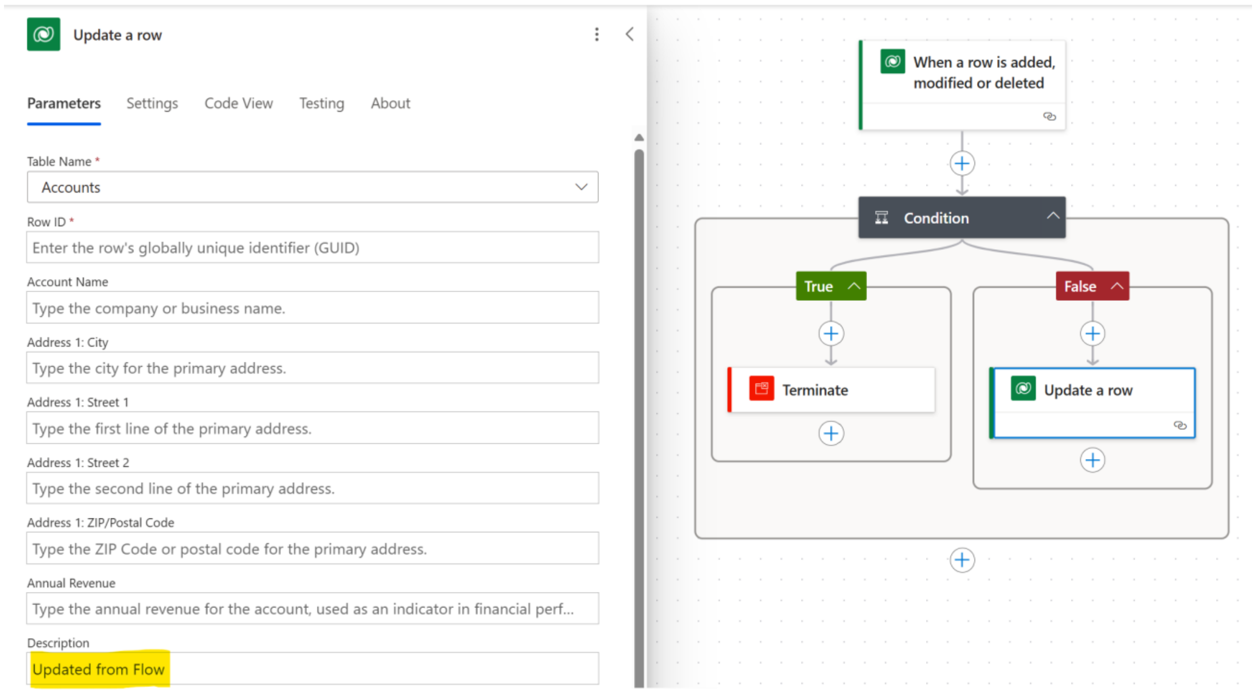

Update a row

Table Name *
Accounts

Row ID *
Enter the row's globally unique identifier (GUID)

Account Name
Type the company or business name.

Address 1: City
Type the city for the primary address.

Address 1: Street 1
Type the first line of the primary address.

Address 1: Street 2
Type the second line of the primary address.

Address 1: ZIP/Postal Code
Type the ZIP Code or postal code for the primary address.

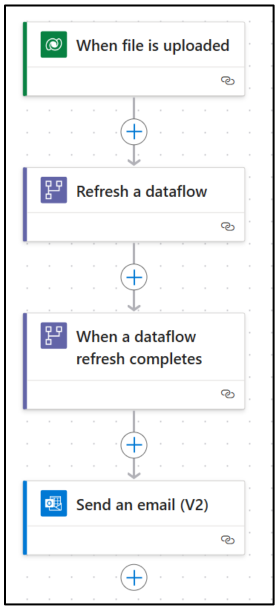
Annual Revenue
Type the annual revenue for the account, used as an indicator in financial perf...

Description
Updated from Flow

Avoid performing large amount of data transformation operations

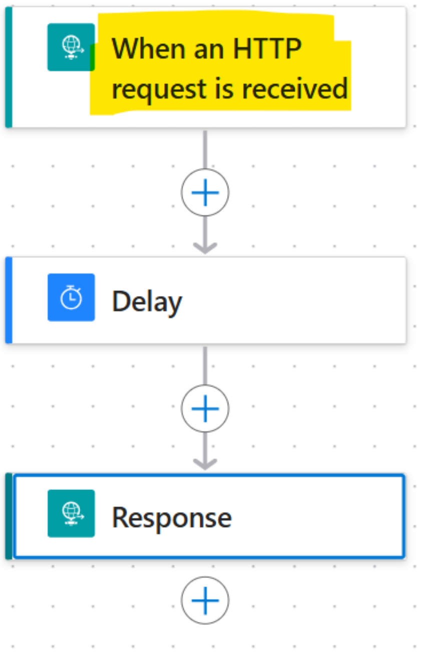
When you need to work with large amount of data transformation, consider if it really should be an Extract-Transform-Load scenario (ETL). For example, using a Power Automate cloud flow to read data off a large Excel spreadsheet, perform some data formatting or validations, and then write it into Dataverse. In such scenario, it is more appropriate to use Power Platform Dataflow or other equivalent ETL tool instead of Cloud Flows.

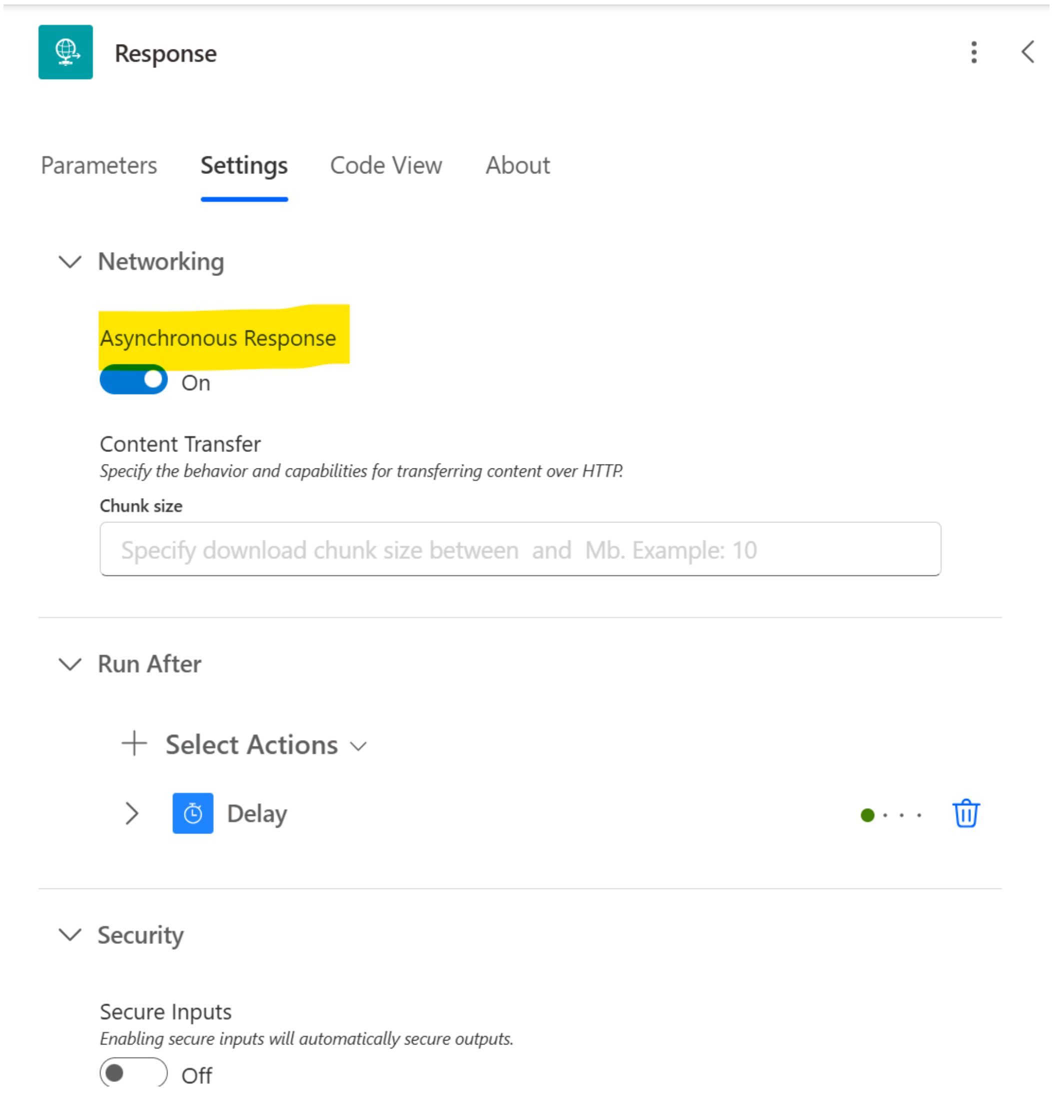
If you need to manage the data load using some form of orchestration logic that you can implement in Cloud Flows, consider combining it with Dataflow. We can use the Dataflow connector to invoke a refresh action, and leverage its trigger “When a dataflow refresh completes” to perform post-ETL actions.



Asynchronous flow pattern

Use asynchronous flow pattern for long running flows. Generally, if a flow is called from a parent flow or a Power App, it has to send a response back to the calling entity within 120 seconds. If the called flow fails to do that, then the calling entity will receive a timeout and will error out. Therefore, in such cases, setting Asynchronous response will allow flow to respond with a 202 to indicate the request has been accepted for processing. Additionally, A location header will be provided to retrieve the final state.




Response

Parameters Settings Code View About

Networking

Asynchronous Response
On

Content Transfer
Specify the behavior and capabilities for transferring content over HTTP.
Chunk size
Specify download chunk size between and Mb. Example: 10

Run After

Select Actions

Delay

Security

Secure Inputs
Enabling secure inputs will automatically secure outputs.
Off

Environment Variables

Environment variables in Power Automate cloud flows are a powerful feature that enhance the flexibility, maintainability, and manageability of your workflows, especially when working with multiple environments like development, testing, and production. Here's an in-depth look at how environment variables are used and managed in Power Automate:

Why use environment variables?

Environment variables are parameters that can be used to store configuration settings and other values that can be easily changed without modifying the actual flow logic. These variables can be referenced within flows, making it easier to manage environment-specific settings, such as URLs, API keys, connection strings, or any other configuration data.

1. **Separation of Configuration and Logic**: By using environment variables, you separate the configuration data from the flow logic. This makes the flows more modular and easier to manage.
2. **Easier Deployment**: Environment variables simplify the deployment process across different environments (development, testing, production) by allowing you to change configurations without editing the flows.
3. **Maintainability**: They make it easier to update and maintain flows. When a configuration value needs to change, you can update the environment variable without needing to edit each individual flow.

Consider following best practices while using Environment Variables

1. **Consistent Naming Conventions**: Use clear and consistent naming conventions for your environment variables to ensure they are easily identifiable and understandable.
2. **Use Default Values Wisely**: Set meaningful default values that can be used as fallbacks in case the current value is not set.
3. **Document Variables**: Document the purpose and usage of each environment variable to ensure that team members understand what each variable is for and how to configure it.
4. **Limit Sensitive Data**: Avoid storing highly sensitive data directly in environment variables. Consider using secure mechanisms for sensitive configurations (e.g., Azure Key Vault).

Example Scenario

Imagine you have a flow that connects to an API endpoint which differs across development, testing, and production environments:

1. **Create Environment Variables**:
   - API_Base_URL with default value https://api.dev.example.com for development.
   - Change the current value to https://api.test.example.com in the testing environment.
   - Change the current value to https://api.example.com in the production environment.
2. **Reference in Flow**:
   - In the HTTP action within your flow, set the URL to @{variables('API_Base_URL')}.

By using environment variables, when you promote your solution from development to testing and production, you only need to update the environment variable values without altering the flow logic. This ensures consistency, reduces errors, and simplifies management.

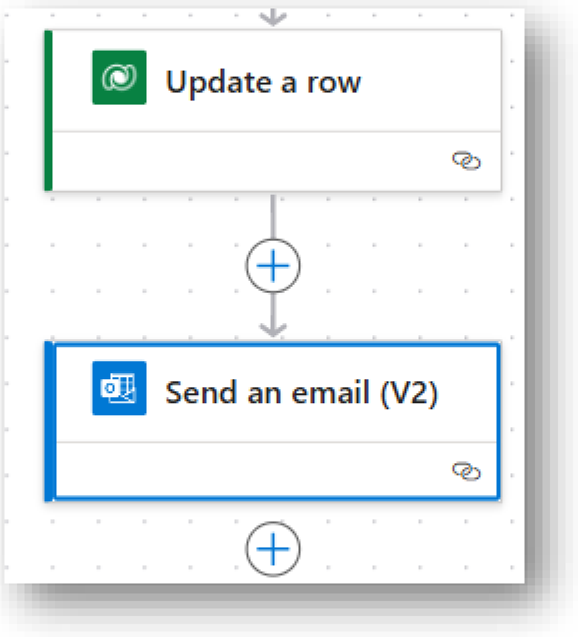
Error Handling

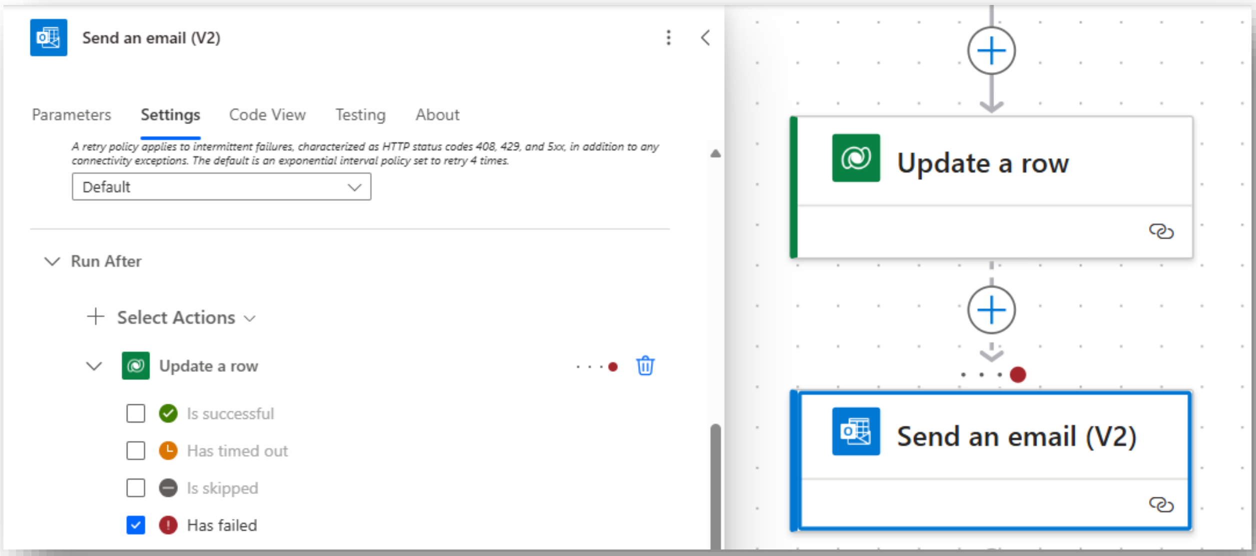
Error handling in Power Automate is essential for ensuring the reliability and robustness of your automated workflows. Here are some key strategies for implementing error handling in Power Automate:

Configure "Run After" Settings

For each action in your flow, you can configure the "Run After" settings to specify whether the action should run based on the outcome of previous actions. This enables conditional execution and error handling. For example, you can configure an action to run only if the previous action failed , skipped or timeout.

In this example a user will get a notification anytime the flow fails at the step “Update a row”





A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy set to retry 4 times.

Default

Run After

Select Actions

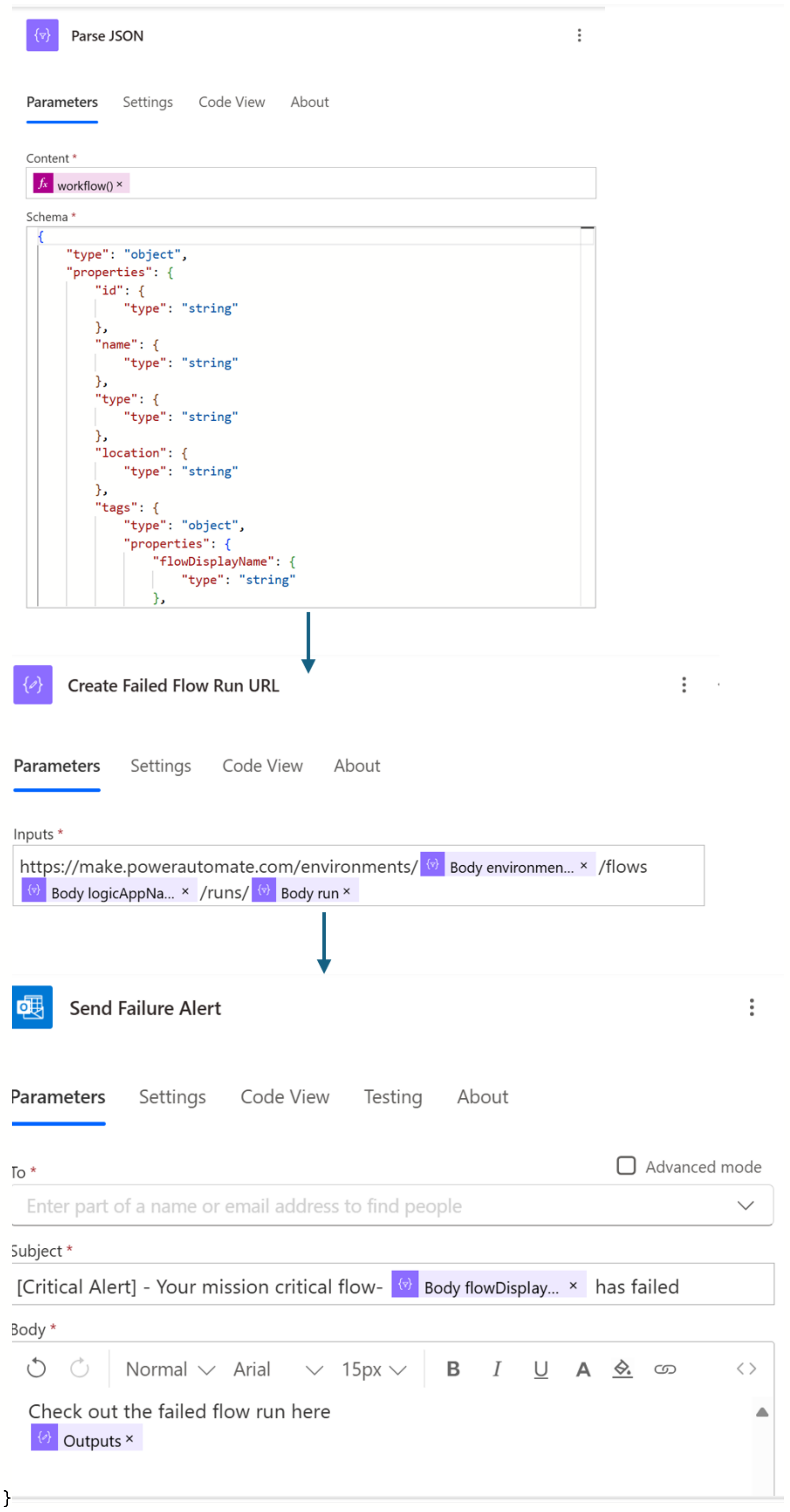
Update a row

- [ ] Is successful
- [ ] Has timed out
- [ ] Is skipped
- [x] Has failed

You can also use the **workflow()** function in Power Automate to get detailed information about the flow run, including the current environment guid, Flow name, Flow guid and run ID.

Workflow() json schema

```
{
    "type": "object",
    "properties": {
        "id": {
            "type": "string"
        },
        "name": {
            "type": "string"
        },
        "type": {
            "type": "string"
        },
        "location": {
            "type": "string"
        },
        "tags": {
            "type": "object",
            "properties": {
                "flowDisplayName": {
                    "type": "string"
                },
                "environmentName": {
                    "type": "string"
            },
            "logicAppName": {
                "type": "string"
            },
            "environmentFlowSuspensionReason": {
                "type": "string"
            },
            "state": {
                "type": "string"
            },
            "createdTime": {
                "type": "string"
            },
            "lastModifiedTime": {
                "type": "string"
            },
            "createdBy": {
                "type": "string"
            },
            "triggerType": {
                "type": "string"
            }
        }
    },
    "run": {
        "type": "object",
        "properties": {
            "id": {
                "type": "string"
            },
            "name": {
                "type": "string"
            },
            "type": {
                "type": "string"
            }
        }
    }
} }
```



Parse JSON

Parameters Settings Code View About

Content *

workflow() ×

Schema *

```
{
    "type": "object",
    "properties": {
        "id": {
            "type": "string"
        },
        "name": {
            "type": "string"
        },
        "type": {
            "type": "string"
        },
        "location": {
            "type": "string"
        },
        "tags": {
            "type": "object",
            "properties": {
                "flowDisplayName": {
                    "type": "string"
                },
```

Create Failed Flow Run URL

Parameters Settings Code View About

Inputs *

https://make.powerautomate.com/environments/ Body environmen... × /flows Body logicAppNa... × /runs/ Body run ×

Send Failure Alert

Parameters Settings Code View Testing About

To * Advanced mode

Enter part of a name or email address to find people

Subject *

[Critical Alert] - Your mission critical flow- Body flowDisplay... × has failed

Body *

Normal Arial 15px

Check out the failed flow run here
Outputs ×

}

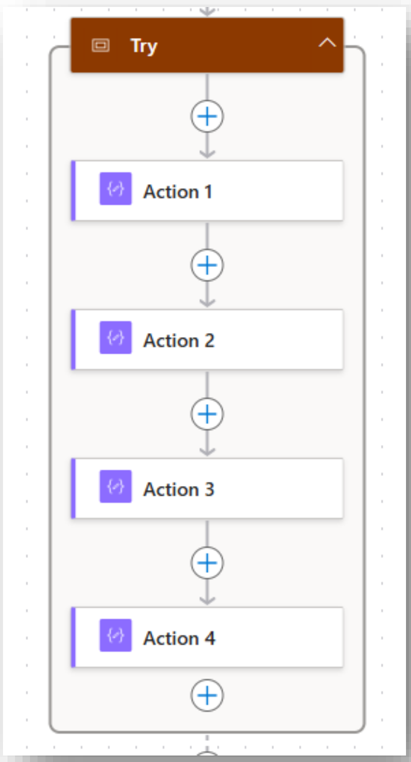
You can then parse that information and build the flow run URL which you can include as a hyperlink in the notification email.

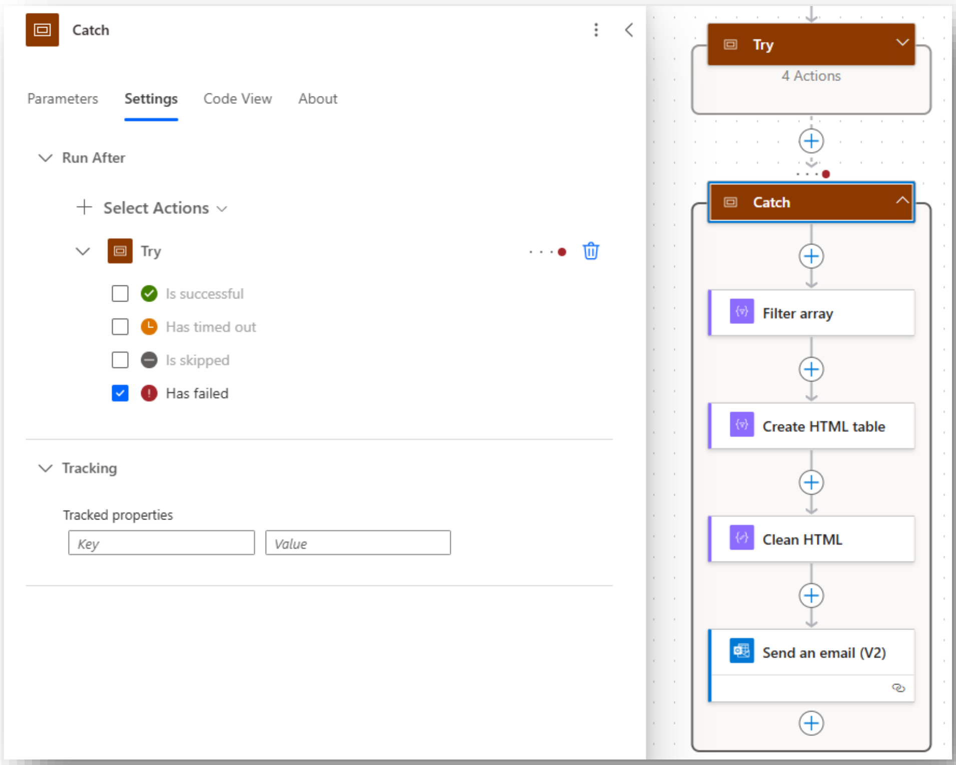
Alert !! - Use this option wisely as it could lead to a lot of custom logging and cause too many actions and impact overall performance. In one way it can spiral into a large anti-pattern if alerted too many times.

Implement Try-Catch Pattern Using Scopes

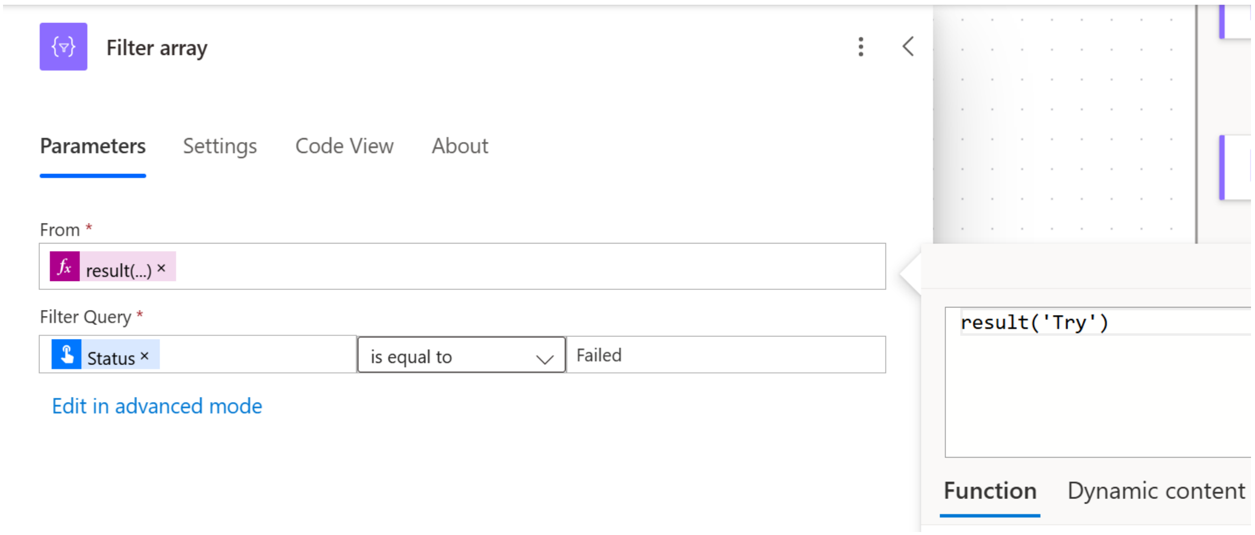
Results of Try will give you all the details of the action within the try block. In the example below, we have 4 Compose Actions within the try block configured as follows

Action1 – test (string)
Action 2 – concat('af','g')
Action 3 – div(1,0)
Action 4 – test (string)





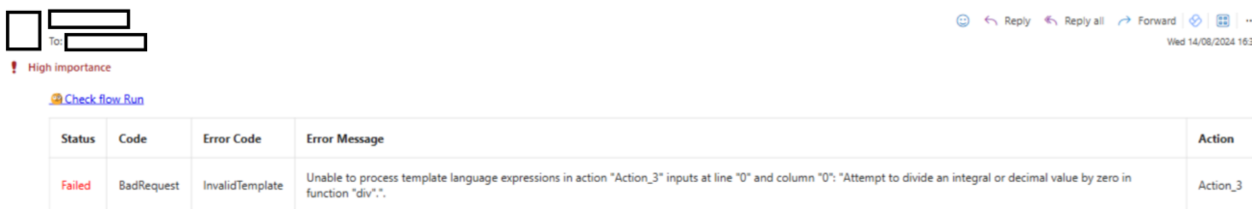
Use Filter array to filter the Result function to get the failed errors. Users can further clean up the records and create an HTML table to send email alerts as shown below



From: result(...)

Filter Query: Status is equal to Failed

```
result('Try')
```



[redacted] [redacted]

To: [redacted]

Wed 14/08/2024 16:

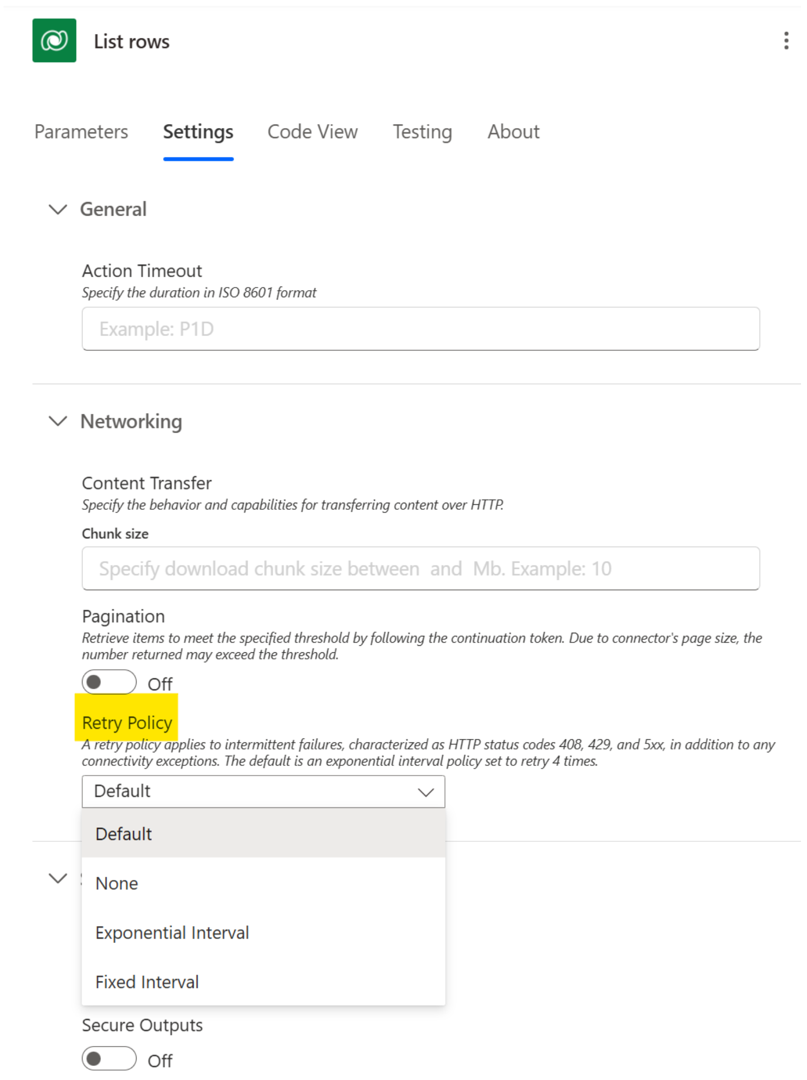
High importance

Check flow Run

Status	Code	Error Code	Error Message	Action
Failed	BadRequest	InvalidTemplate	Unable to process template language expressions in action "Action_3" inputs at line "0" and column "0": "Attempt to divide an integral or decimal value by zero in function "div".".	Action_3

Use Retry Policy

Retry policy under flow settings helps in handling default transient failure. It is recommended to choose exponential policies as they can go for a prolonged period of time.



Action Timeout
Specify the duration in ISO 8601 format
Example: P1D

Content Transfer
Specify the behavior and capabilities for transferring content over HTTP.
Chunk size
Specify download chunk size between and Mb. Example: 10

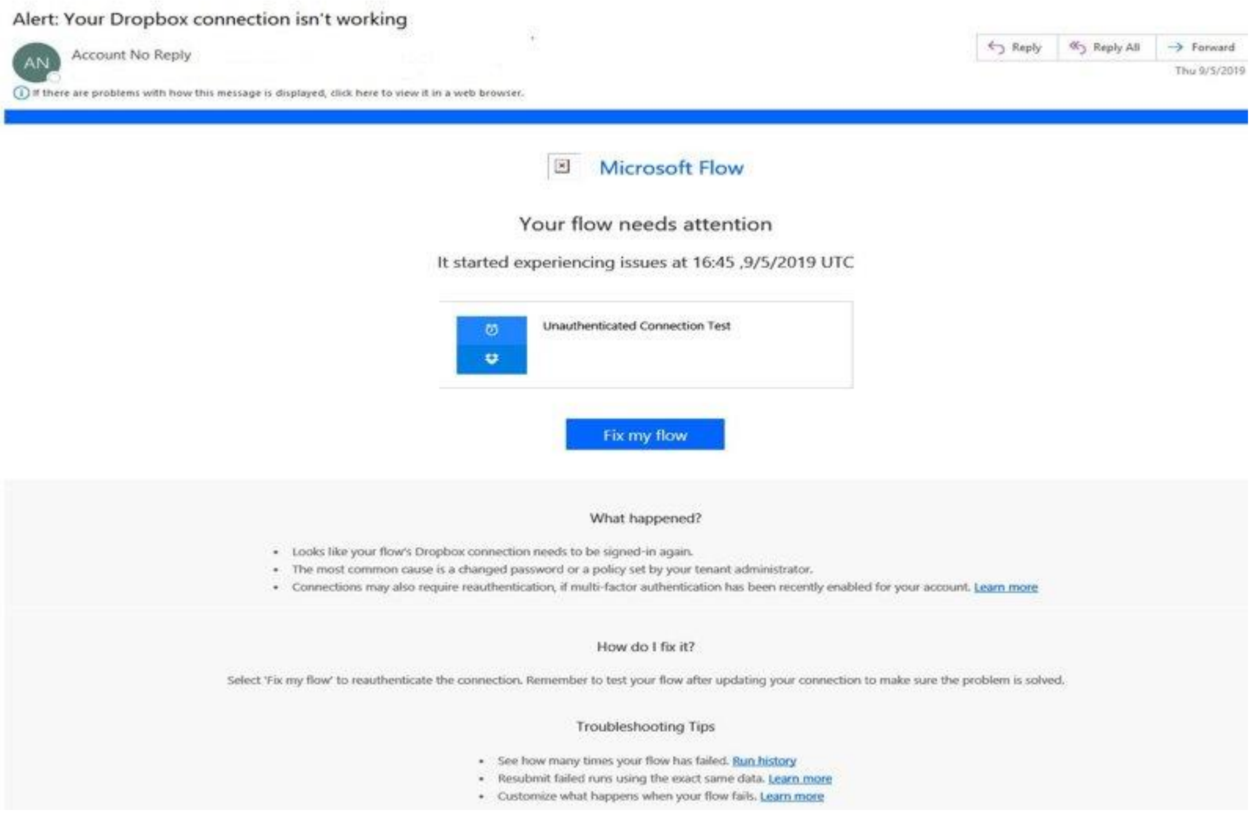
Pagination
Retrieve items to meet the specified threshold by following the continuation token. Due to connector's page size, the number returned may exceed the threshold.
Off

Retry Policy
A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy set to retry 4 times.
Default

Secure Outputs
Off

Use Flow Remediation Emails

Power Automate Flow service will generate email alerts to all flow owners for certain common or critical failures like broken connections or flow turning off due to throttling. These email contain the error information in detail as well as troubleshooting tips on how to rectify the issue.



Alert: Your Dropbox connection isn't working

Account No Reply

If there are problems with how this message is displayed, click here to view it in a web browser.

Your flow needs attention

It started experiencing issues at 16:45 ,9/5/2019 UTC

Unauthenticated Connection Test

What happened?

- Looks like your flow's Dropbox connection needs to be signed-in again.
- The most common cause is a changed password or a policy set by your tenant administrator.
- Connections may also require reauthentication, if multi-factor authentication has been recently enabled for your account. Learn more

How do I fix it?

Select 'Fix my flow' to reauthenticate the connection. Remember to test your flow after updating your connection to make sure the problem is solved.

Troubleshooting Tips

- See how many times your flow has failed. Run history
- Resubmit failed runs using the exact same data. Learn more
- Customize what happens when your flow fails. Learn more

Flow Turn off behavior

A trigger is an event that starts a cloud flow. For example, if you want to get a notification in Microsoft Teams when someone sends you an email, in this case you receiving an email is the trigger that starts this flow.

How do trigger work?

There are 2 types of triggers:

- Polling triggers
- Webhook triggers

Once the flow is created, the trigger will register itself to either poll the service its trying to connect to or listen to the service. By service we mean that if its a SQL trigger it will poll or listen to SQL server it trying to connect. If its an Outlook trigger it will listen to outlook service.

Trigger type	Description
Polling, such as the recurrence trigger	When the flow is turned on again, all unprocessed or pending events are processed. If you don't want to process pending items when you turn your flow back on, delete and then recreate your flow.
Webhook	When the flow is turned on again, it processes new events that are generated after the flow is turned on.

Polling triggers

Once the polling trigger is registered it polls the service every X min to get the details of any records/events that are created based on the filters that has been applied to a trigger, depending on the license that the user owns. To poll the service, in the backend the trigger tracks the timestamp, when it last polled the service and every X minutes based on that time stamp it polls again.

So, say if the trigger is when a SQL record is created, then the trigger will poll the SQL Service say every 1 min and gets information on any records that have been created in a minutes since it last polled. if there are records that are created it will trigger the flow. If there are no records created then it the run will be skipped. You can see this polls in checks section in run history page. Now when the flow is stopped, for example, it stopped on Sept 13th, 12:30 PM. It will note this timestamp as when it last polled the service, when the flow is turned on back it say on Sept 14th 1:30 PM PT, the flow will poll the service to get all the events that are created between its last poll time and current time (meaning, it will poll for all records between sept 13th 12:30 PM – Sept 14th 1:30 PM). This is because turning off the flow doesn't de-register the trigger. It only stops the polling clock. This is by design as sometimes, the flow is stopped erroneously or due to throttling, but we don't want to miss any data to trigger.

The only way to get around this behavior is to create a new copy of the flow as it will re-register the trigger again when you turn on the flow for the first time and delete the existing one.

Webhook triggers

Webhook triggers on the other hand, do not poll the service. Once created they register with the service to let it know that they want to get any notification when a certain event happens on that service. Webhooks are simple HTTP callbacks used to provide event notifications. Power Automate allow you to use webhooks as triggers. A flow listens for this trigger and performs an action whenever the trigger fires.

The idea is that when an event happens, the service will send an event notification to the trigger with all the details of that particular event and then the flow will get triggered. this means that the flow doesnt have to poll the service and reduces a lot of interactions.

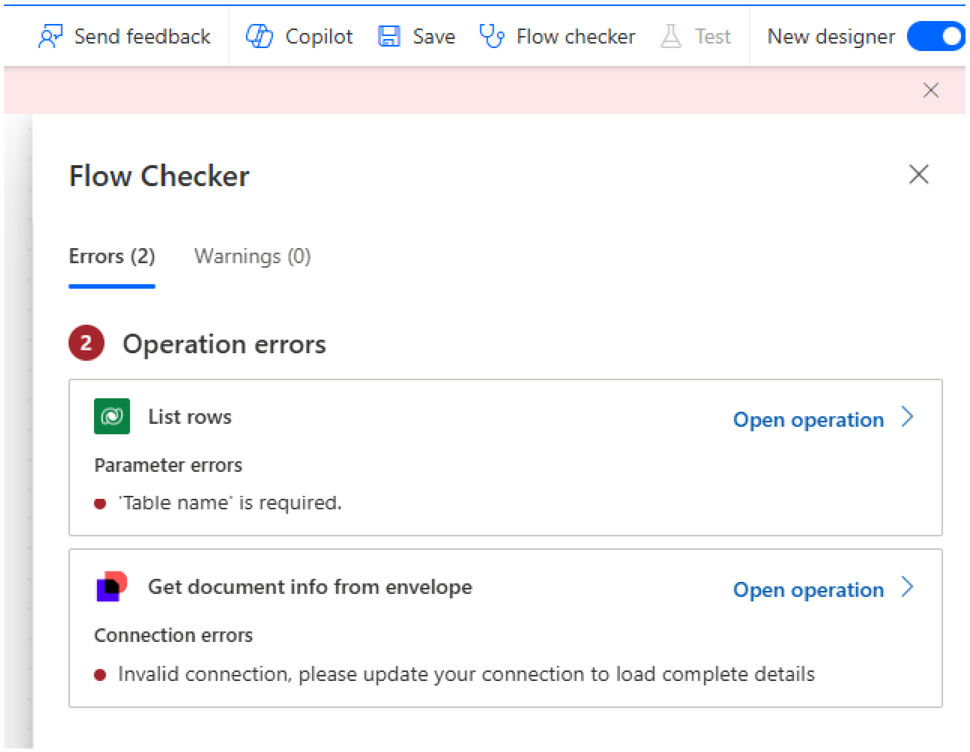
For more information, refer here

Flow Checker

The Flow Checker in Microsoft Power Automate is a built-in tool designed to help users ensure the quality and correctness of their cloud flows. It provides real-time analysis and feedback on potential issues, helping to improve the reliability and functionality of the flows. This tool is essential for both new and experienced users, as it enhances the development process by providing actionable insights.

Here is how Flow Checker can help makers –

1. **Error Detection**: The Flow Checker identifies errors that will prevent the flow from running, such as missing or incorrect inputs, misconfigured actions, and connectivity issues.
2. **Warnings and Suggestions**: It also provides warnings and best practice suggestions to optimize the flow's performance and maintainability. These might include recommendations for improving logic or identifying deprecated actions.
3. **Real-time Feedback**: As you build or modify your flow, the Flow Checker provides real-time feedback, allowing you to address issues immediately.
4. **Detailed Explanations**: Each error or warning comes with a detailed explanation and guidance on how to resolve the issue, making it easier for users to understand and fix problems.



Flow Checker

Errors (2) Warnings (0)

2 Operation errors

List rows — Open operation

Parameter errors

- 'Table name' is required.

Get document info from envelope — Open operation

Connection errors

- Invalid connection, please update your connection to load complete details

You can learn more about Flow Checker here.

Cloud Flow Testing

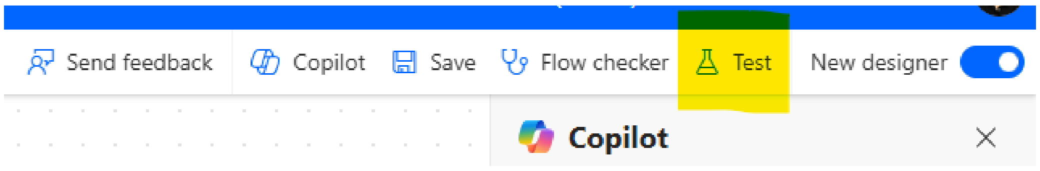
Effective testing of Power Automate cloud flows is essential for ensuring their reliability, performance, and correctness. Here are some of the mechanisms that you could use to test your flows.

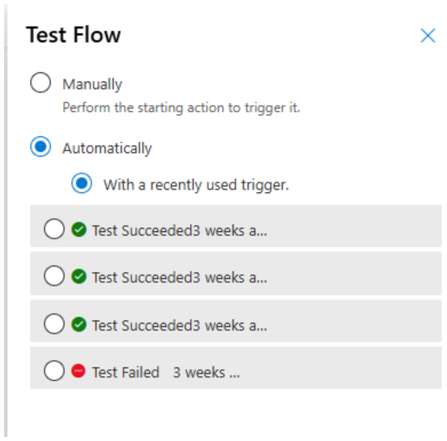
Design Phase testing

Consider using Flow checker and Test Flow tool during this phase. We have covered Flow checker in the above section.

In the Test pane, there are now three options for testing your flow:

- Manually trigger the test yourself by doing the action that triggers the flow. For example, you can go to your inbox and send yourself a test email. Or, you can go to SQL and insert a row.
- Use data from previous runs to perform the test.





Manually
Perform the starting action to trigger it.
Automatically
With a recently used trigger.
Test Succeeded3 weeks a...
Test Succeeded3 weeks a...
Test Succeeded3 weeks a...
Test Failed 3 weeks ...

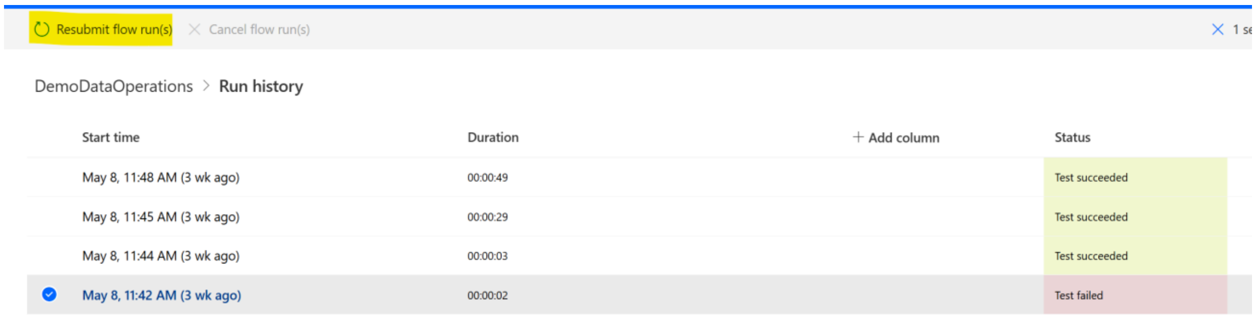
Static result testing (Mock Data)

Use mock data for testing actions that interact with external systems to avoid unintended changes to production data.

Resubmitting cloud flow runs

Sometimes Cloud Flow might stop working unexpectedly or fail due to the Server or an asynchronous process that did not meet some requirements while running. In this case, you need to re-run the Flow, which you can do by choosing the Run History and clicking on Re-submit.

We also have the ability to resubmit or cancel runs in bulk.



Start time	Duration	Status
May 8, 11:48 AM (3 wk ago)	00:00:49	Test succeeded
May 8, 11:45 AM (3 wk ago)	00:00:29	Test succeeded
May 8, 11:44 AM (3 wk ago)	00:00:03	Test succeeded
May 8, 11:42 AM (3 wk ago)	00:00:02	Test failed

Considerations for Resubmitting Runs

1. **Error Handling:** Ensure your flow has proper error handling and retry policies in place. This can help mitigate issues that caused the initial failure and provide better insights into why a run might fail again.

2. **Data Duplication:** Be aware of potential data duplication or other side effects that could result from resubmitting a run. For example, if your flow includes actions that create records or send emails, resubmitting the run might repeat these actions.

3. **Input Data:** Verify that the input data required for the flow run is still valid and available. Changes in the data or the environment (e.g., deleted files or records) can affect the outcome of the resubmitted run.

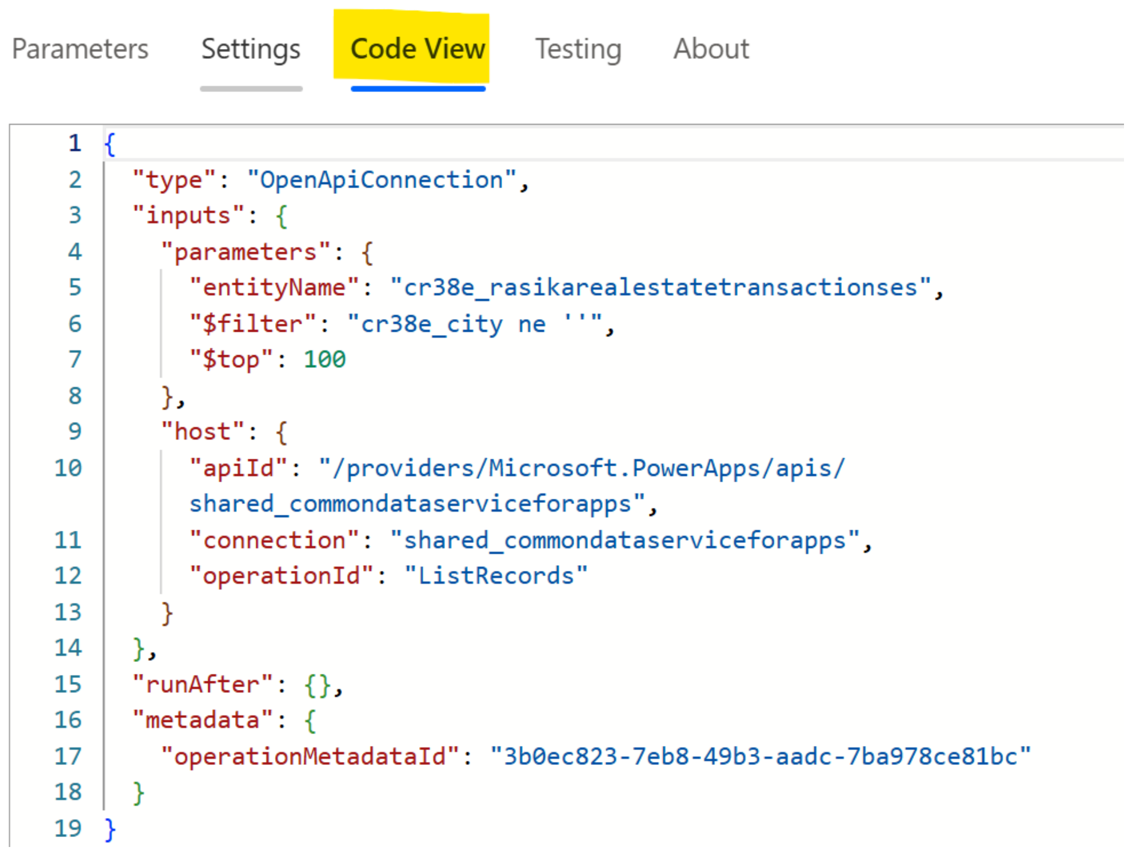
4. **Flow Updates:** If you have made changes to the flow since the original run, consider whether the changes might affect the resubmission. It's usually best to resubmit using the version of the flow that was in place at the time of the original run to ensure consistency.

Pro Tips

Peek code/Code View

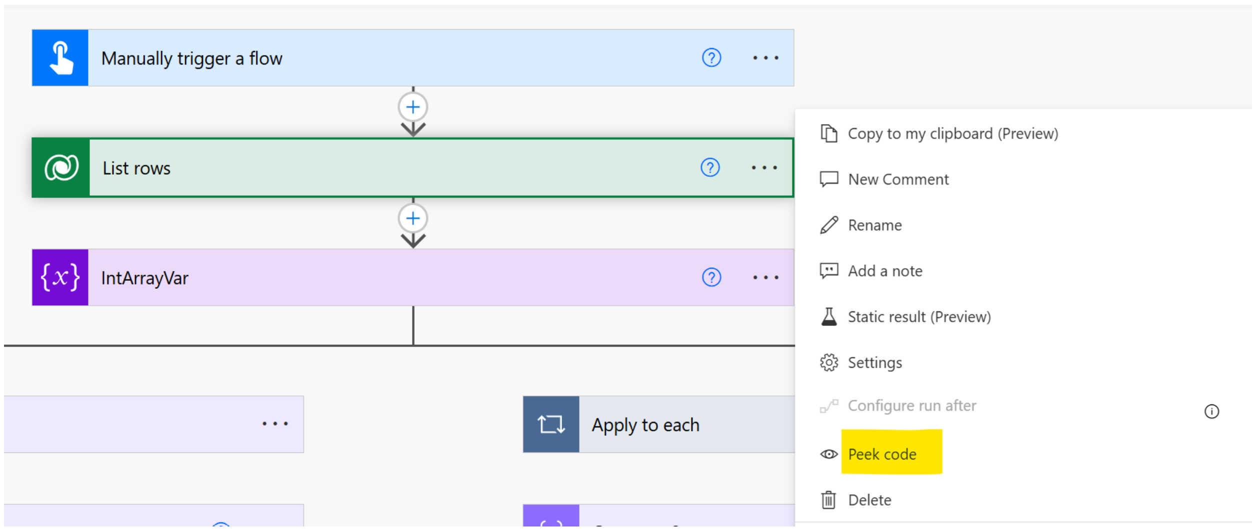
Peek code or Code view is a feature that allows users to view the underlying JSON representation of the actions and triggers within a flow. This feature is particularly useful for advanced users who want to understand the exact structure and configuration of their flows, troubleshoot issues, or manually edit specific details that are not easily accessible through the standard visual interface.

In the new designer use Code view to view trigger/action code

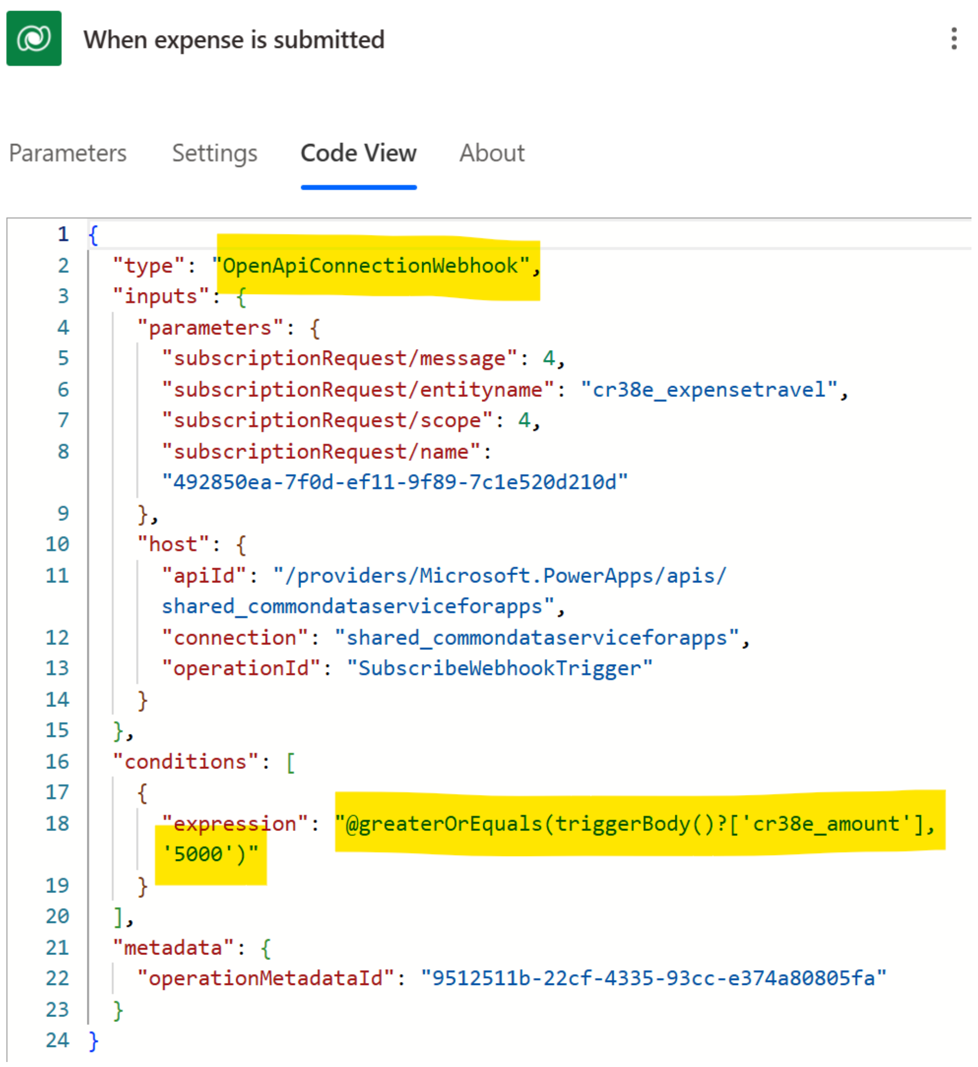


```
{
  "type": "OpenApiConnection",
  "inputs": {
    "parameters": {
      "entityName": "cr38e_rasikarealestatetransactionses",
      "$filter": "cr38e_city ne ''",
      "$top": 100
    },
    "host": {
      "apiId": "/providers/Microsoft.PowerApps/apis/
      shared_commondataserviceforapps",
      "connection": "shared_commondataserviceforapps",
      "operationId": "ListRecords"
    }
  },
  "runAfter": {},
  "metadata": {
    "operationMetadataId": "3b0ec823-7eb8-49b3-aadc-7ba978ce81bc"
  }
}
```

In classic designer, use peek code to view the JSON code



Additionally, code view/peek code can be used to view trigger polling frequency, trigger type, Odata queries even JSON representation of the dynamic content etc.



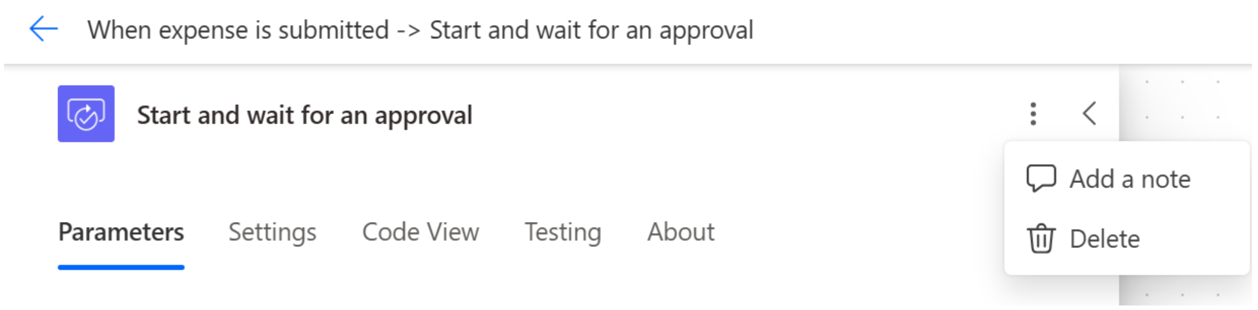
When expense is submitted

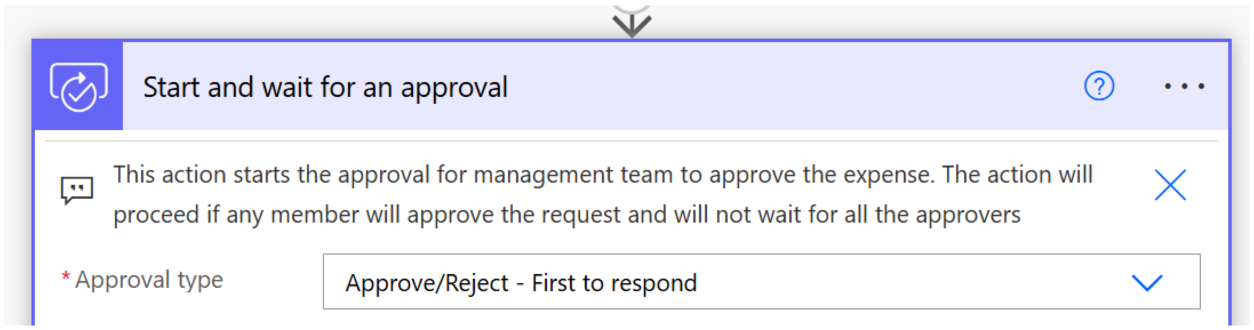
Parameters Settings **Code View** About

```
{
  "type": "OpenApiConnectionWebhook",
  "inputs": {
    "parameters": {
      "subscriptionRequest/message": 4,
      "subscriptionRequest/entityname": "cr38e_expensetravel",
      "subscriptionRequest/scope": 4,
      "subscriptionRequest/name":
      "492850ea-7f0d-ef11-9f89-7c1e520d210d"
    },
    "host": {
      "apiId": "/providers/Microsoft.PowerApps/apis/
      shared_commondataserviceforapps",
      "connection": "shared_commondataserviceforapps",
      "operationId": "SubscribeWebhookTrigger"
    }
  },
  "conditions": [
    {
      "expression": "@greaterOrEquals(triggerBody()?['cr38e_amount'],
      '5000')"
    }
  ],
  "metadata": {
    "operationMetadataId": "9512511b-22cf-4335-93cc-e374a80805fa"
  }
}
```

Add a note

Adding notes or comments in Power Automate cloud flows is a helpful way to document the purpose, logic, and specific details of various actions and triggers within the flow. This practice is beneficial for both individual users and teams, as it improves the readability and maintainability of the flows.


When expense is submitted -> Start and wait for an approval
Start and wait for an approval
Parameters
Settings
Code View
Testing
About
Add a note
Delete

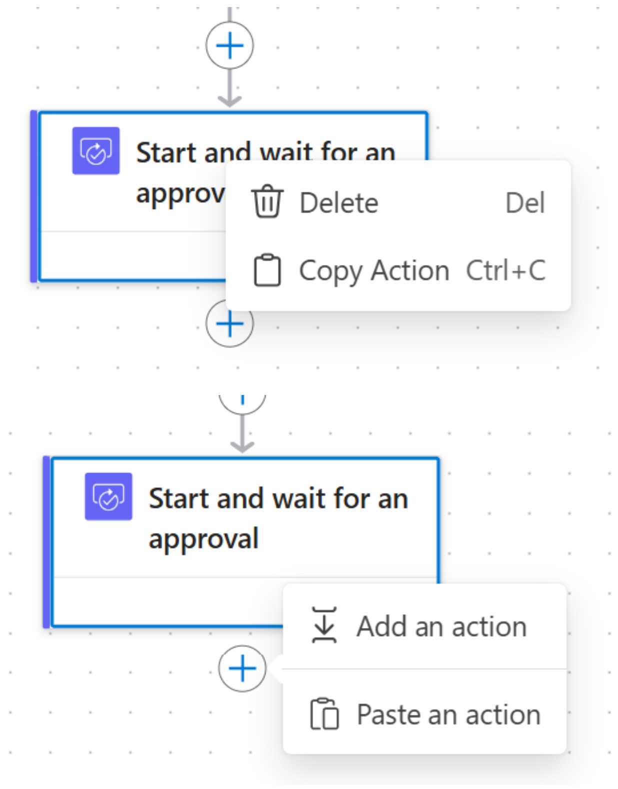

Start and wait for an approval

This action starts the approval for management team to approve the expense. The action will proceed if any member will approve the request and will not wait for all the approvers

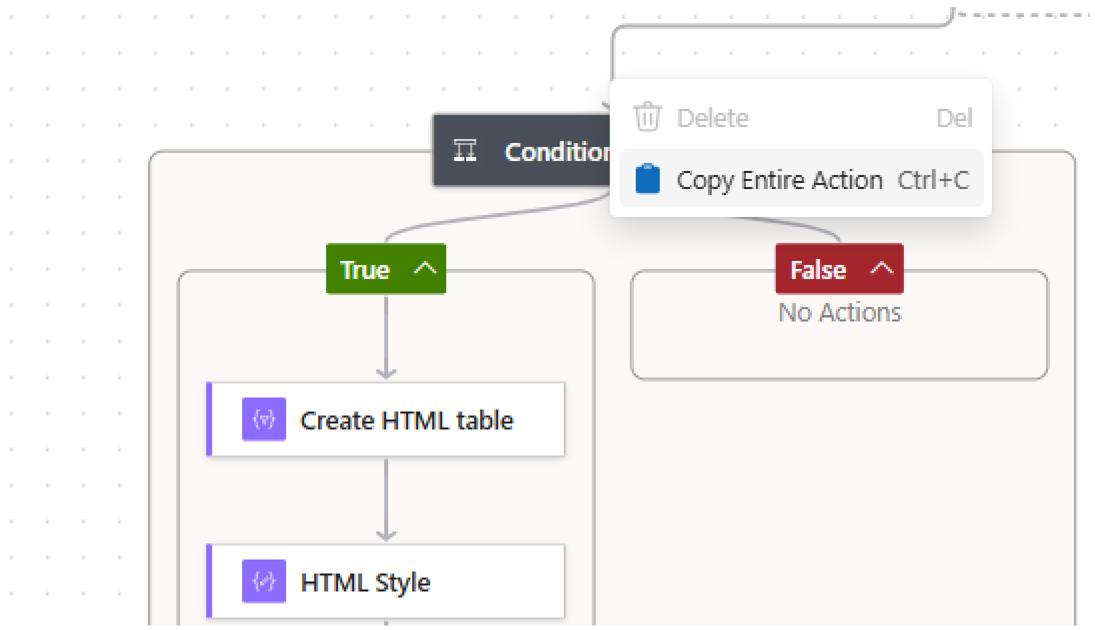
* Approval type: Approve/Reject - First to respond

Copy Paste actions

Copy paste of actions is a useful way to reuse the same action multiple times within a flow or across different flows. This helps in simplifying the process of creating and modifying flows and saves time and multiple clicks.



The new designer in power automate gives the ability to copy and paste entire conditions or scope actions as well.



Troubleshooting cloud flows

Power Automate Cloud flows detailed troubleshooting documents are listed as follows:

Troubleshooting triggers
Troubleshoot Cloud flows

Here are some important handy tips

Understand the error codes

Error codes 400s are something that can be corrected at the user side.

400 – Bad Request – This can happen due to misconfiguration at the trigger or action level
401 – Access Denied – User does not have access to the service
403 – Forbidden – User has access to the service but is forbidden to access a given end point
404 – Not found – Calling resource doesn't exist.

Error codes 500 or 502 are service related and may be transient and rectify by itself. These errors should be reviewed at the service level.

Understanding Platform limits/Avoiding throttling

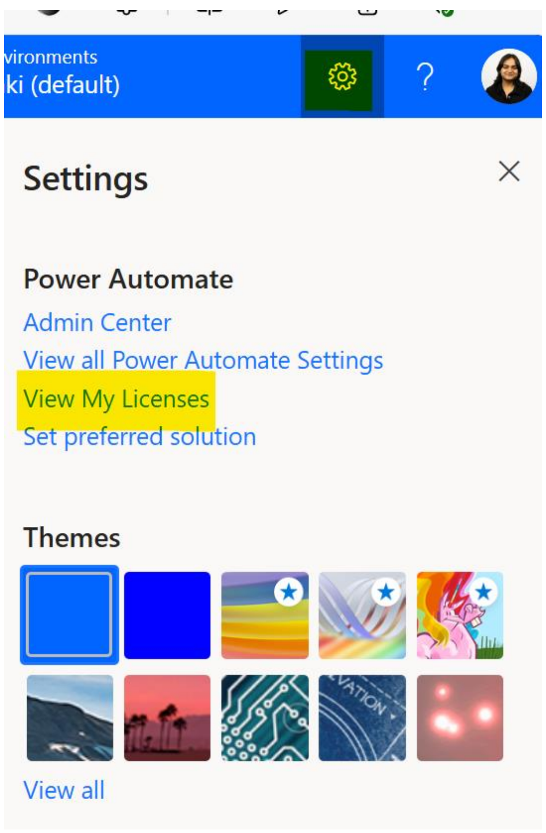
Understanding Power Automate and Power Platform limits can help users design scalable Power Automate Flows.
These limits can determine flow's performance and help avoid throttling (slowing down) or turning off of flows due to request limits violations. Any flow that gets throttled continuously for 14 days gets turned off. These flows can be turned on anytime. Refer to Retention Limits for more details.

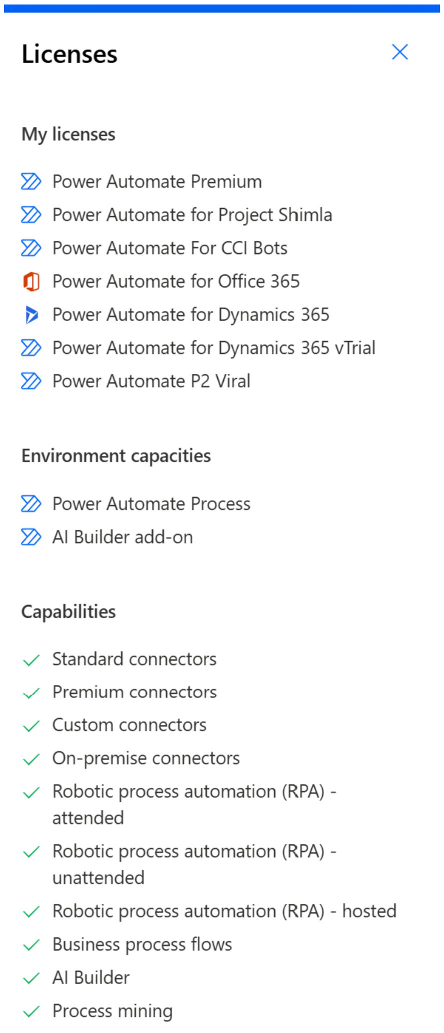
To know more about the platform and power automate limits, refer to the following learn documents:

API requests limits and allocations
Power Automate Limits and configurations

How to check your license plans

Some of the platform or API limits, depend on the license plan that the user has. The easiest way to identify your license is via Settings - > View My Licenses




Licenses

My licenses

- Power Automate Premium
- Power Automate for Project Shimla
- Power Automate For CCI Bots
- Power Automate for Office 365
- Power Automate for Dynamics 365
- Power Automate for Dynamics 365 vTrial
- Power Automate P2 Viral

Environment capacities

- Power Automate Process
- AI Builder add-on

Capabilities

- ✓ Standard connectors
- ✓ Premium connectors
- ✓ Custom connectors
- ✓ On-premise connectors
- ✓ Robotic process automation (RPA) - attended
- ✓ Robotic process automation (RPA) - unattended
- ✓ Robotic process automation (RPA) - hosted
- ✓ Business process flows
- ✓ AI Builder
- ✓ Process mining

An alternative option can be to press CNTRL + ALT+ A from the power automate portal to get deeper plan level details.

API request limits

Requests in Microsoft Power Platform consist of various actions that a user makes across various products. At a high level, below is what constitutes an API request in Power Automate

All API requests to connectors, process advisor analysis, HTTP actions, and built-in actions from initializing variables to a simple compose action. Both succeeded and failed actions count towards these limits. Additionally, retries and other requests from pagination count as action executions as well. For more information, see What counts as Power Platform request?

Based on license plan, there are limits to the number of actions a cloud flow can run in day. These limits are different from connector throttling limits. You can see the number of actions your flow runs by selecting **Analytics** from the flow details page and looking at the **Actions** tab.

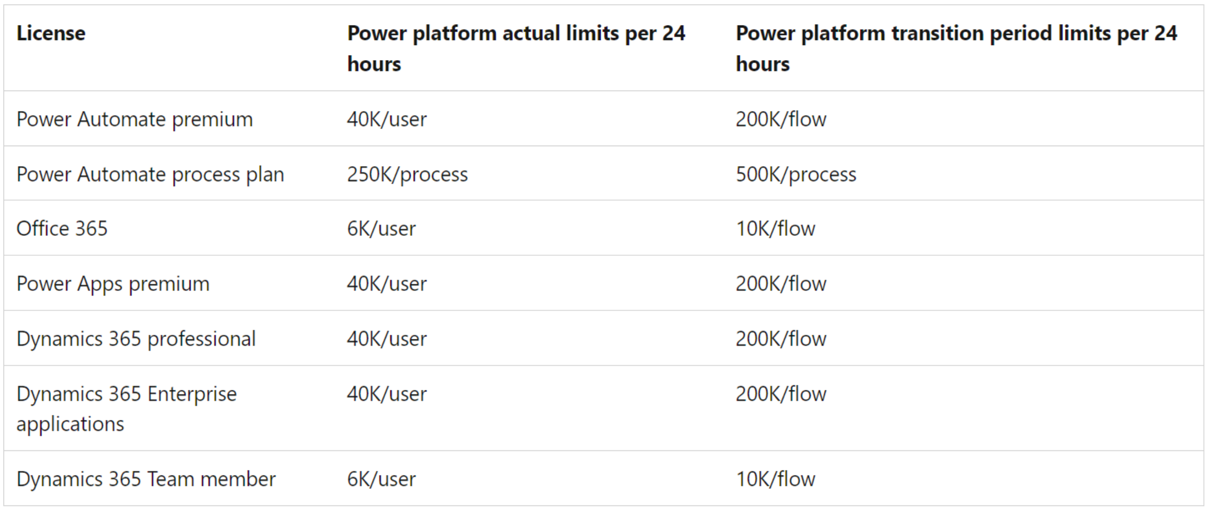
Even when the flow uses fewer Power Platform requests, you can still reach your limits if the flow runs more frequently than you expect. For example, you might create a cloud flow that sends you a push notification whenever your manager sends you an email. That flow must run every time you get an email (from anyone) because the flow must check whether the email came from your manager. The limit applies to all runs across all your flows in a 24-hour period. Here are some guidelines to estimate the request usage of a flow.

- One or more actions run as part of a flow run. A simple flow with one trigger and one action results in two "actions" each time the flow runs, consuming two requests.
- Every trigger/action in the flow generates Power Platform requests. All kinds of actions like connector actions, HTTP actions, built-in actions (from initializing variables, creating scopes to a simple compose action) generate Power Platform requests. For example, a flow that connects SharePoint, Exchange, Twitter, and Dataverse, all those actions are counted towards Power Platform request limits.
- Both succeeded and failed actions count towards these limits. Skipped actions aren't counted towards these limits.
- Each action generates one request. If the action is in an apply to each loop, it generates more Power Platform requests as the loop executes.
- An action can have multiple expressions but it's counted as one API request.
- Retries and extra requests from pagination count as action executions as well.

Here is a quick overview of API Limits at the platform level based on user License.

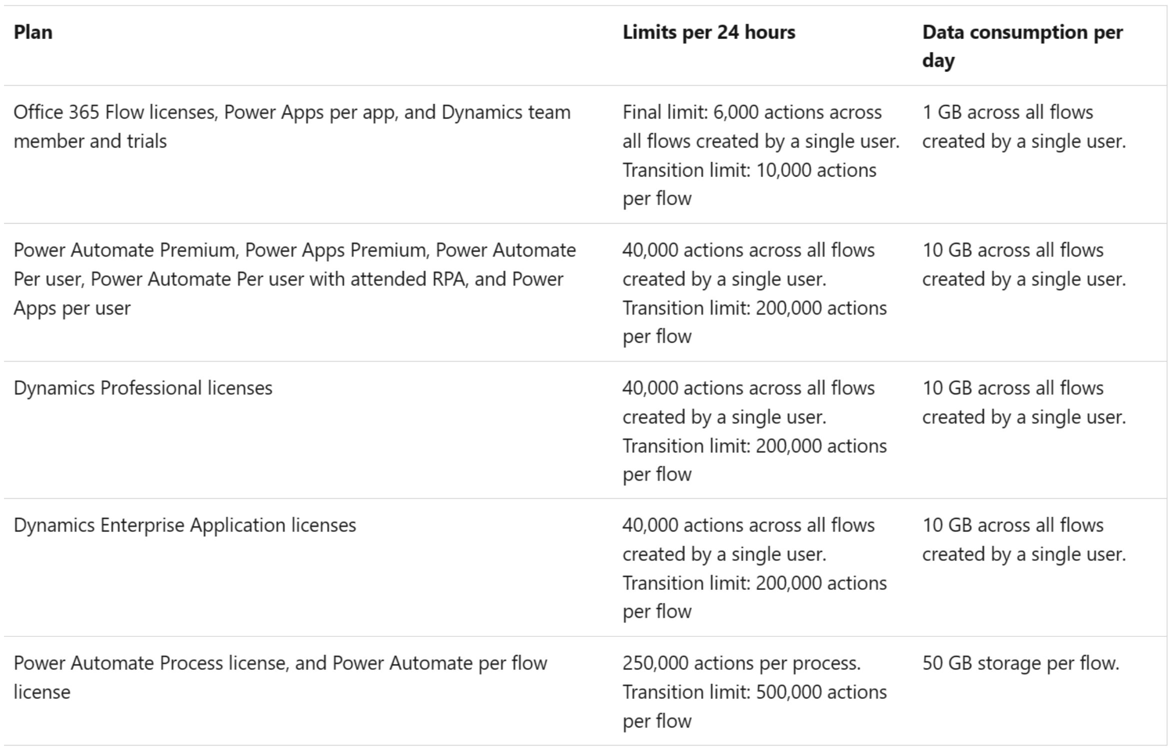
Note: currently all customers are in a Transition period, where API limits are not fully enforced and are higher. This transition period will end after Power Platform admin center reports are GA. Organizations have six months to analyze their usage and purchase licenses that are appropriate before strict enforcement on license limits begins. Learn more about transition period here

Transition period Comparison:



License	Power platform actual limits per 24 hours	Power platform transition period limits per 24 hours
Power Automate premium	40K/user	200K/flow
Power Automate process plan	250K/process	500K/process
Office 365	6K/user	10K/flow
Power Apps premium	40K/user	200K/flow
Dynamics 365 professional	40K/user	200K/flow
Dynamics 365 Enterprise applications	40K/user	200K/flow
Dynamics 365 Team member	6K/user	10K/flow

Additional Data consumption details:



Plan	Limits per 24 hours	Data consumption per day
Office 365 Flow licenses, Power Apps per app, and Dynamics team member and trials	Final limit: 6,000 actions across all flows created by a single user. Transition limit: 10,000 actions per flow	1 GB across all flows created by a single user.
Power Automate Premium, Power Apps Premium, Power Automate Per user, Power Automate Per user with attended RPA, and Power Apps per user	40,000 actions across all flows created by a single user. Transition limit: 200,000 actions per flow	10 GB across all flows created by a single user.
Dynamics Professional licenses	40,000 actions across all flows created by a single user. Transition limit: 200,000 actions per flow	10 GB across all flows created by a single user.
Dynamics Enterprise Application licenses	40,000 actions across all flows created by a single user. Transition limit: 200,000 actions per flow	10 GB across all flows created by a single user.
Power Automate Process license, and Power Automate per flow license	250,000 actions per process. Transition limit: 500,000 actions per flow	50 GB storage per flow.

Key Things to remember

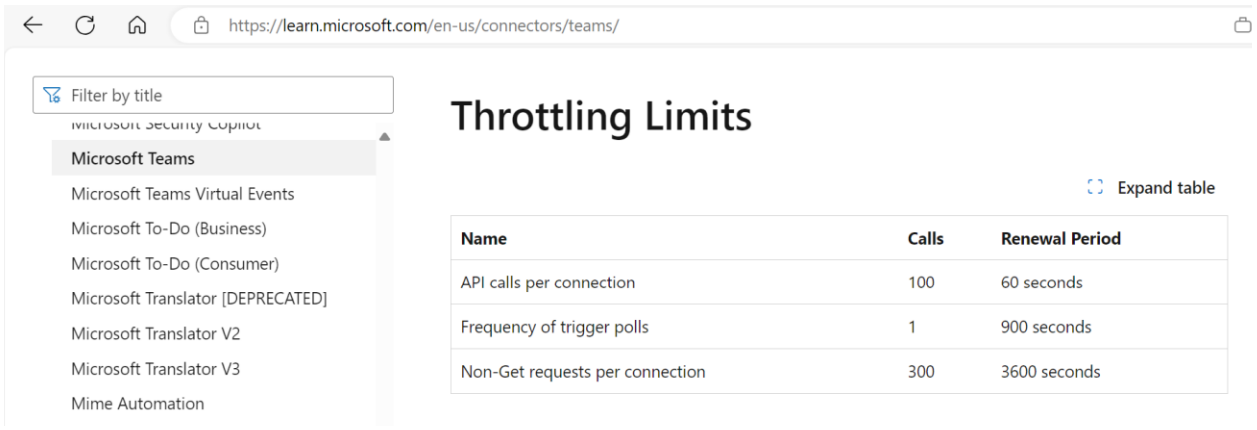
- User limits are applied across all flows
  - Automatic flows which they created
  - Manual flows which they executed
- Flow/Process limits are applied to the particular flows
  - Per-Process can apply to group of related flows
  - Per flow applies to a single flow
- Pay-Go environments
  - Flows don’t have daily quota as you pay for consumption

API throughput limits on connectors

In addition to platform limits, each connector service has its own limits. Connector throttling in Power Automate refers to the mechanism by which connectors enforce rate limits or usage quotas to prevent abuse and ensure fair resource allocation. When a connector is throttled, it restricts the number of requests or operations that can be made within a specific timeframe.

When a flow runs into connector level throttling limits, the service will return a “429 (Too Many Requests)” error code, with error text like “Rate limit is exceeded. Try again in 27 seconds”

Each connector will have its own throttling limit. Here is an example of Teams Connector –


https://learn.microsoft.com/en-us/connectors/teams/

Filter by title
Microsoft Security Copilot
Microsoft Teams
Microsoft Teams Virtual Events
Microsoft To-Do (Business)
Microsoft To-Do (Consumer)
Microsoft Translator [DEPRECATED]
Microsoft Translator V2
Microsoft Translator V3
Mime Automation
Expand table

Throttling Limits

Name	Calls	Renewal Period
API calls per connection	100	60 seconds
Frequency of trigger polls	1	900 seconds
Non-Get requests per connection	300	3600 seconds

Dataverse API Limits

Dataverse as a connector service has defined its own Service protection limits. The service protection API limits are evaluated per user. When called by a flow the "user" is whoever is associated with the action. Usually this is the flow owner but can be the invoking user if using invoking user context in the action.

Measure	Description	Limit per web server
Number of requests	The cumulative number of requests made by the user.	6000 within the 5 minute sliding window
Execution time	The combined execution time of all requests made by the user.	20 minutes (1200 seconds) within the 5 minute sliding window
Number of concurrent requests	The number of concurrent requests made by the user	52 or higher

Flow Concurrency Limits

Designing scalable, efficient flows includes understanding of the concurrency, looping and debatching limits to help avoid unnecessary delays. Here are the limits for a single run:

Name	Limit	Notes
Concurrent runs	- Unlimited for flows with Concurrency Control turned off - 1 to 100 when Concurrency Control is turned on (defaults to 25)	This is the limit for how many runs a flow can have at the same time. **Note**: Concurrency Control is set in the flow's trigger settings and is off by default. Turning on Concurrency Control can't be undone without deleting and re-adding the trigger.
Waiting runs	- Not applicable when Concurrency Control is off - 10 plus the degree of parallelism (1-100) when Concurrency Control is on	This limit describes the highest number of flow runs that can be queued when the flow is at its maximum number of concurrent runs. **Note**: Additional triggers that arrive while the waiting runs limit is met might be re-tried by the connector. However, the retry attempts might not succeed if the maximum waiting limit continues to be met for an extended period of time. To ensure all triggers result in flow runs, leave the Concurrency Control setting off in the flow's trigger.
Apply to each array item	5,000 for Low, 100,000 for all others	This limit describes the highest number of array items that an "apply to each" loop can process. To filter larger arrays, you can use the query action.
Apply to each concurrency	1 is the default limit. You can change the default to a value between 1 and 50 inclusively.	This limit is highest number of "apply to each" loop iterations that can run at the same time, or in parallel.
Split on items	- 5,000 for Low without trigger concurrency - 100,000 for all others without trigger concurrency - 100 with trigger concurrency	For triggers that return an array, you can specify an expression that uses a 'SplitOn' property that splits or debatches array items into multiple workflow instances for processing, rather than use a "Foreach" loop. This expression references the array to use for creating and running a workflow instance for each array item. **Note**: When concurrency is turned on, the Split on limit is reduced to 100 items.
Until iterations	- Default: 60 - Maximum: 5,000	
Paginated items	5,000 for Low, 100,000 for all others	To process more items, trigger multiple flow runs over your data.

Action Burst Limits

Action Burst Limits refer to the maximum number of actions that can be triggered within a specific period, typically measured in a rolling window of time. Currently there is a per-flow cap of 100,000 actions per 5 minutes.
Any bursts of triggering or loops can lead to over exceeding this limit, causing slowing down/throttling of flows.

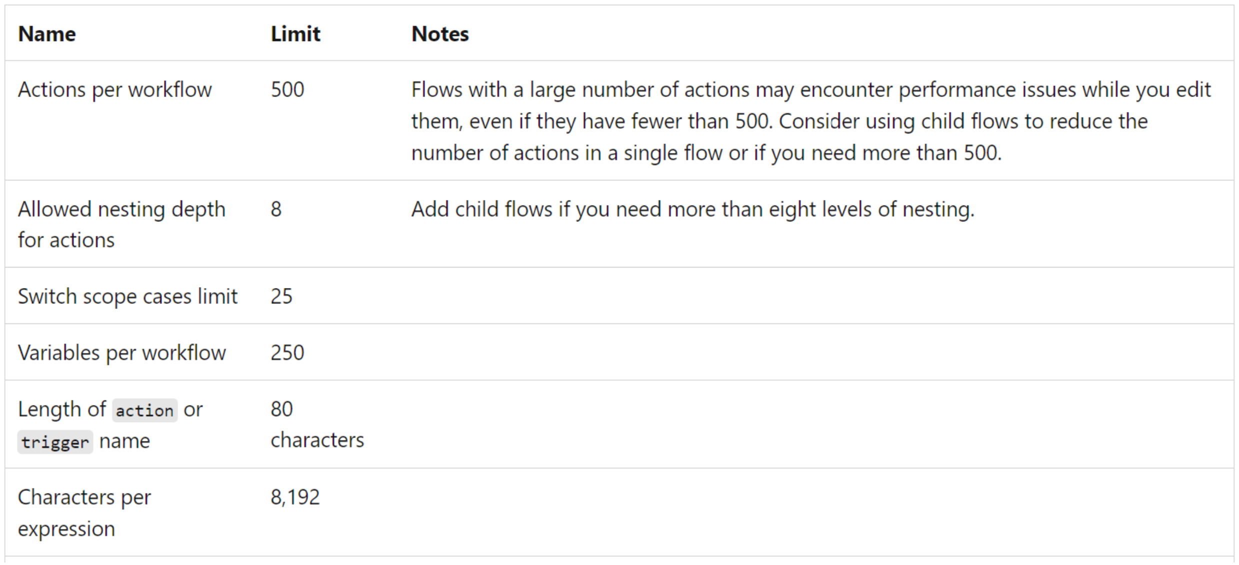
Best way to address this would be to distribute the load between multiple flows.

- Use trigger conditions
- Use best practices documented above in this whitepaper for designing for each loop.

Additional Limits

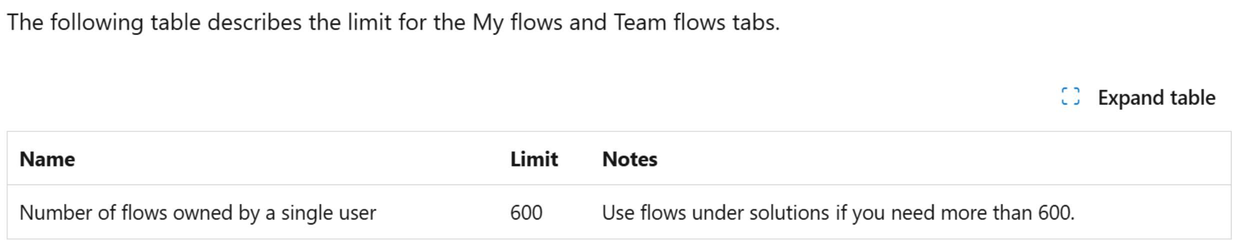
Flow Design limits

While designing flows, users can run into some limits that are defined at the design/definition level. Consider redesigning your flows, if you run into any of these limits:



Name	Limit	Notes
Actions per workflow	500	Flows with a large number of actions may encounter performance issues while you edit them, even if they have fewer than 500. Consider using child flows to reduce the number of actions in a single flow or if you need more than 500.
Allowed nesting depth for actions	8	Add child flows if you need more than eight levels of nesting.
Switch scope cases limit	25	
Variables per workflow	250	
Length of `action` or `trigger` name	80 characters	
Characters per expression	8,192	

Flow Creation limits

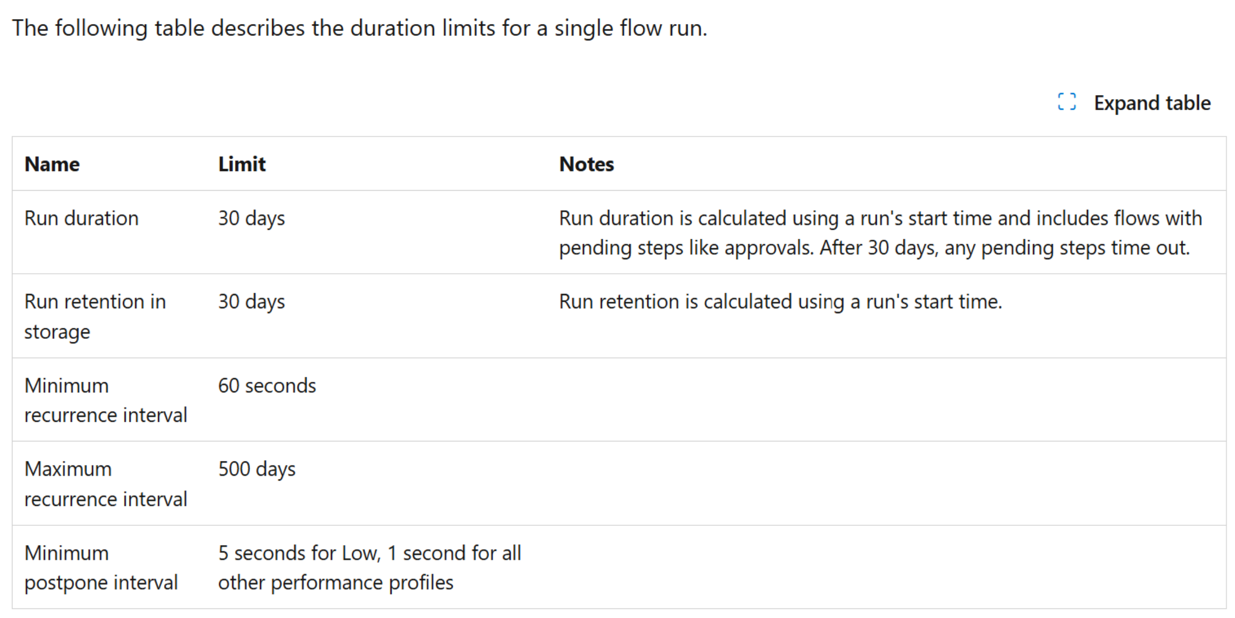


The following table describes the limit for the My flows and Team flows tabs.

Expand table

Name	Limit	Notes
Number of flows owned by a single user	600	Use flows under solutions if you need more than 600.

Flow Timeout limits



The following table describes the duration limits for a single flow run.

Expand table

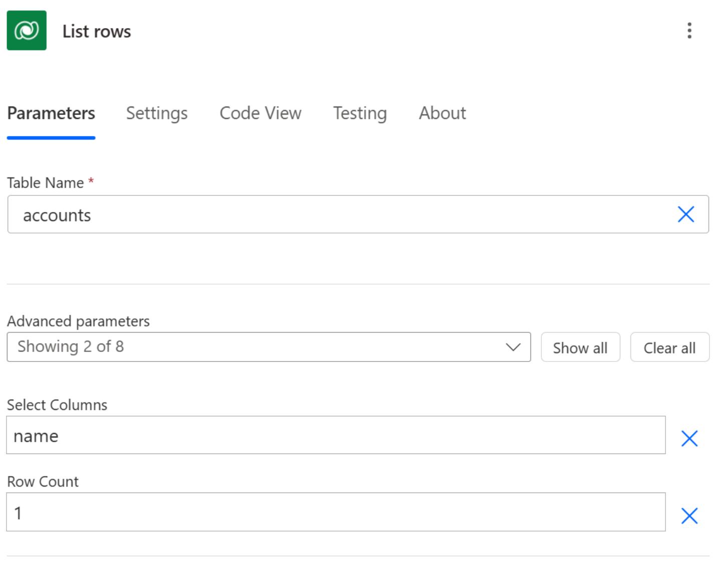
Name	Limit	Notes
Run duration	30 days	Run duration is calculated using a run's start time and includes flows with pending steps like approvals. After 30 days, any pending steps time out.
Run retention in storage	30 days	Run retention is calculated using a run's start time.
Minimum recurrence interval	60 seconds	
Maximum recurrence interval	500 days	
Minimum postpone interval	5 seconds for Low, 1 second for all other performance profiles	

Designing Secure Flows

As much as its required to create a well-designed power automate flow which could run efficiently generating the required results efficiently, its equally important to ensure that the flow are secured and protect any sensitive data, prevent unauthorized access and ensure compliance with security standards.

Using Secure Inputs/Outputs

Currently, Power Automate gives users ability to view flow run history and deep dive into a trigger or action’s inputs and outputs information.


List rows

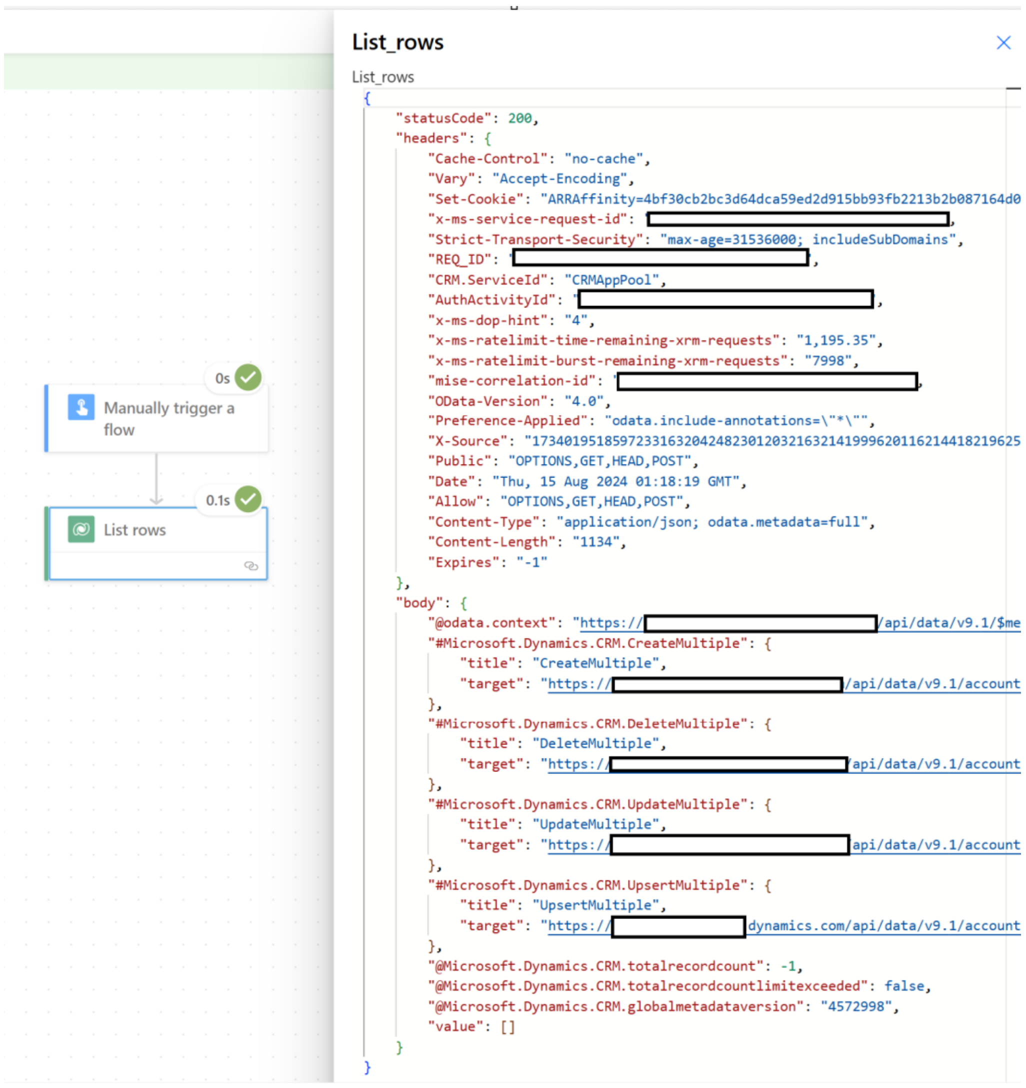
Parameters | Settings | Code View | Testing | About

Table Name *
accounts

Advanced parameters
Showing 2 of 8 | Show all | Clear all

Select Columns
name

Row Count
1

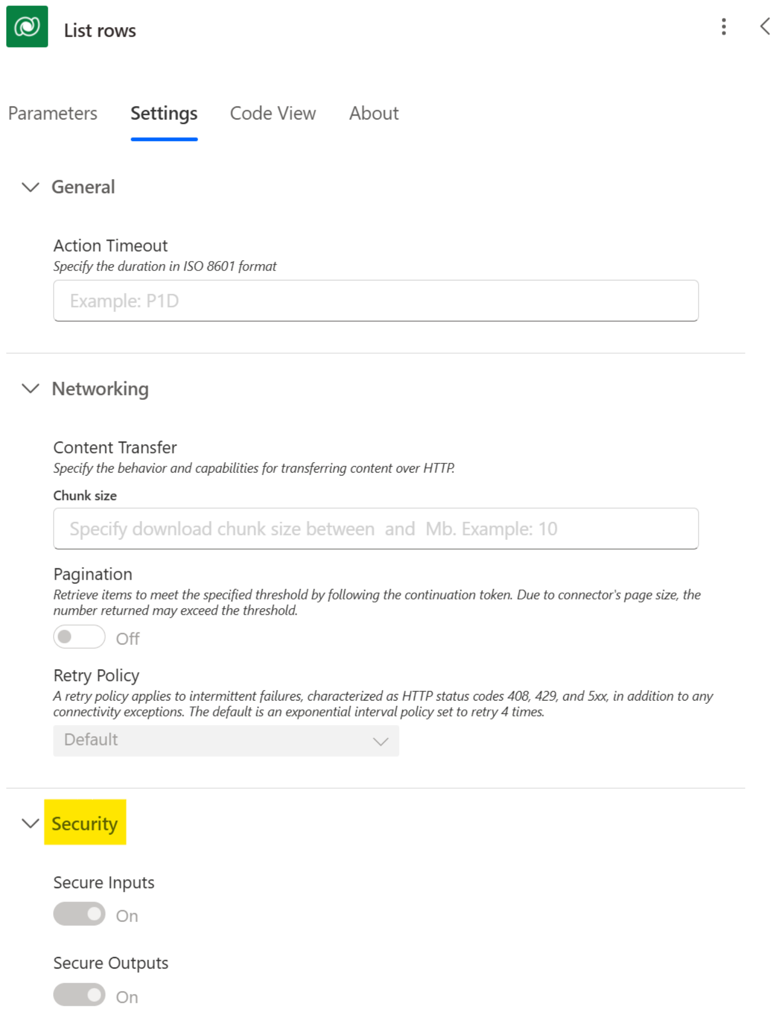


List_rows

List_rows

```
{
    "statusCode": 200,
    "headers": {
        "Cache-Control": "no-cache",
        "Vary": "Accept-Encoding",
        "Set-Cookie": "ARRAffinity=4bf30cb2bc3d64dca59ed2d915bb93fb2213b2b087164d0
        "x-ms-service-request-id": "[redacted]",
        "Strict-Transport-Security": "max-age=31536000; includeSubDomains",
        "REQ_ID": "[redacted]",
        "CRM.ServiceId": "CRMAppPool",
        "AuthActivityId": "[redacted]",
        "x-ms-dop-hint": "4",
        "x-ms-ratelimit-time-remaining-xrm-requests": "1,195.35",
        "x-ms-ratelimit-burst-remaining-xrm-requests": "7998",
        "mise-correlation-id": "[redacted]",
        "OData-Version": "4.0",
        "Preference-Applied": "odata.include-annotations=\"*\"",
        "X-Source": "17340195185972331632042482301203216321419996201162144182196 25
        "Public": "OPTIONS,GET,HEAD,POST",
        "Date": "Thu, 15 Aug 2024 01:18:19 GMT",
        "Allow": "OPTIONS,GET,HEAD,POST",
        "Content-Type": "application/json; odata.metadata=full",
        "Content-Length": "1134",
        "Expires": "-1"
    },
    "body": {
        "@odata.context": "https://[redacted]/api/data/v9.1/$me
        "#Microsoft.Dynamics.CRM.CreateMultiple": {
            "title": "CreateMultiple",
            "target": "https://[redacted]/api/data/v9.1/account
        },
        "#Microsoft.Dynamics.CRM.DeleteMultiple": {
            "title": "DeleteMultiple",
            "target": "https://[redacted]/api/data/v9.1/account
        },
        "#Microsoft.Dynamics.CRM.UpdateMultiple": {
            "title": "UpdateMultiple",
            "target": "https://[redacted]api/data/v9.1/account
        },
        "#Microsoft.Dynamics.CRM.UpsertMultiple": {
            "title": "UpsertMultiple",
            "target": "https://[redacted]dynamics.com/api/data/v9.1/account
        },
        "@Microsoft.Dynamics.CRM.totalrecordcount": -1,
        "@Microsoft.Dynamics.CRM.totalrecordcountlimitexceeded": false,
        "@Microsoft.Dynamics.CRM.globalmetadataversion": "4572998",
        "value": []
    }
}
```

However, with the use of Secure Input and Output settings, you can prevent users from seeing any sensitive information.



List rows

Parameters | Settings | Code View | About

General

Action Timeout
Specify the duration in ISO 8601 format
Example: P1D

Networking

Content Transfer
Specify the behavior and capabilities for transferring content over HTTP.
Chunk size
Specify download chunk size between and Mb. Example: 10

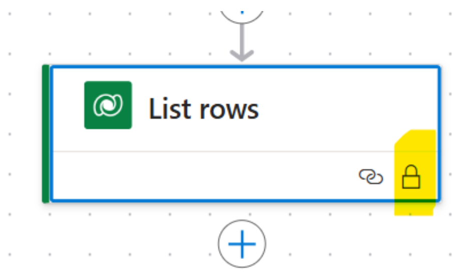
Pagination
Retrieve items to meet the specified threshold by following the continuation token. Due to connector's page size, the number returned may exceed the threshold.
Off

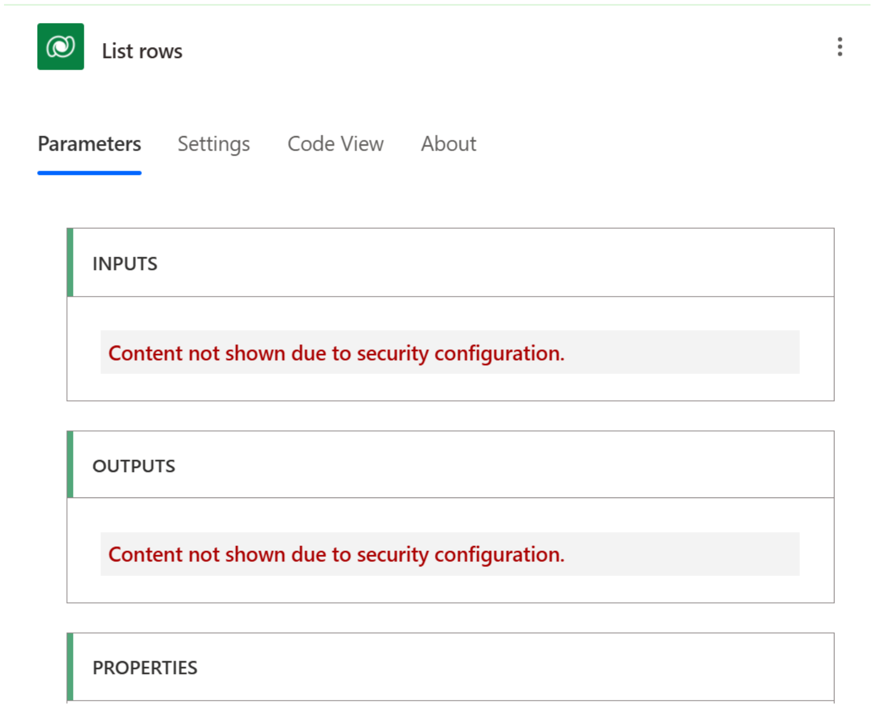
Retry Policy
A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy set to retry 4 times.
Default

Security

Secure Inputs
On

Secure Outputs
On




List rows

Parameters Settings Code View About

INPUTS

Content not shown due to security configuration.

OUTPUTS

Content not shown due to security configuration.

PROPERTIES

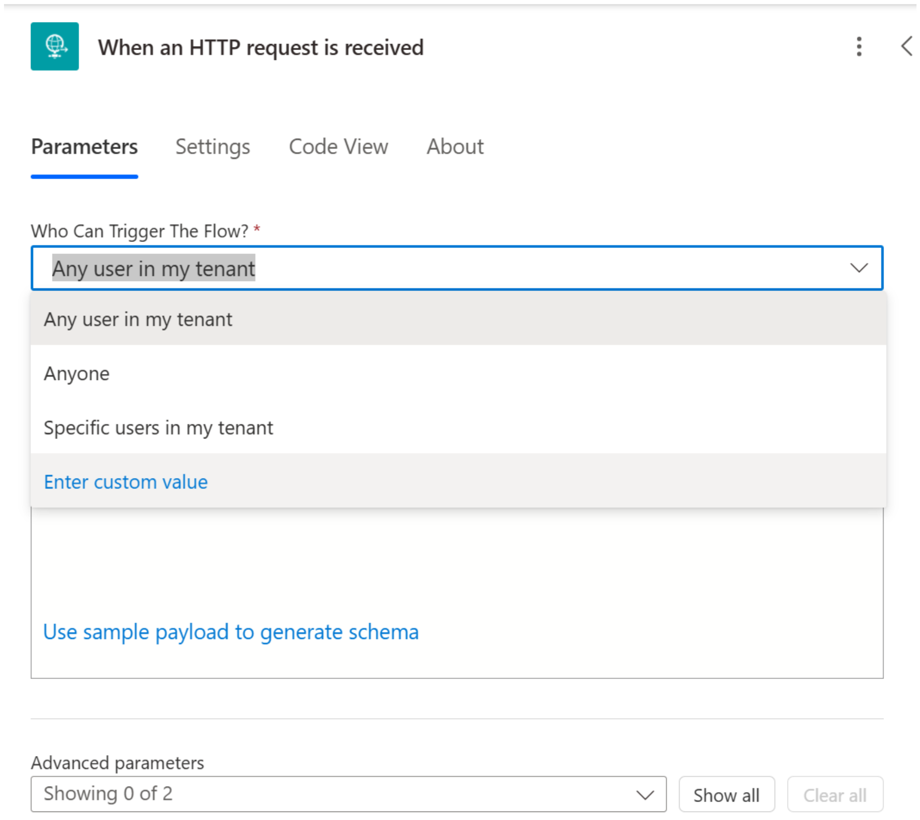
Overall, some of the best practices to secure your data within Power Automate, include:

- Avoid hardcoding sensitive information directly into your flows or variables. Instead, use Secure Inputs/Outputs to store and retrieve this data securely.
- Consider integrating Power Automate with Azure Key Vault for enhanced security and centralized management of secrets.
- Azure Key Vault allows you to store and manage sensitive information securely, and Power Automate can retrieve secrets from Key Vault at runtime using managed identities or service principals.

Securing HTTP Request Trigger

You can use the **When an HTTP request is received** trigger to trigger workflows by sending a request to an HTTP request to the endpoint generated from the flow. You can restrict what users can trigger in this workflow by ensuring that only authenticated users can trigger this workflow.

One of the approach will be to use AAD token. This token can be defined to restrict specific users/principals in a tenant, or any user within a tenant. More information can be find here


When an HTTP request is received

Who Can Trigger The Flow? *

Any user in my tenant

- Any user in my tenant
- Anyone
- Specific users in my tenant
- Enter custom value

The second approach will be to use IP-Pinning. Environment admins can configure IP set or range that are allowed to interact with Power Platform resources.
This is currently in preview. More information can be found here – IP firewall in Power Platform environments

Keeping Flow configuration generic

Environment Variables

When a Power Automate flow is exported via a solution or as a zip package, its very easy to open the flow definition and any hard coded values like passwords or secrets can be exposed. To avoid this, use Environment variables.

Service Principals

A service principal is a non-human security identity that represents an application or service that can own and manage resources within Azure and the Power Platform. To use a service principal in the Power Platform, a service principal application user needs to be created that represents the service principal through the portal or through API. An application user can have connections shared with them and own resources such as flows.

Use service principals for flow ownership as well as creating connections wherever possible. To understand more about service principals refer here.

Configure AAD/Entra conditional access policies

Power Automate conditional access policies can be created in addition to DLP, HTTP OAuth and IP-pinning to prevent data exfiltration.

The minimum audience to include in conditional access policies would include:

- Your unique Dataverse org audience (e.g. https://{your-org}.crm.dynamics.com/)
- Power Platform: https://api.powerplatform.com
- Power Automate: https://service.flow.microsoft.com/
- Power Apps: https://service.powerapps.com/
- Connections: https://apihub.azure.com

For more details on the impact of conditional access policies on flows, refer here

Understanding Access to flows

Ownership of Flows (SPN vs User)

When managing Power Automate flows, the choice between using a Service Principal Name (SPN) or a user account as the flow owner is critical. Using an SPN for flow ownership offers consistency and security, as SPNs are not tied to individual users, ensuring that flows continue running smoothly even if personnel changes occur. This setup reduces the risk of disruptions and allows for stricter control over permissions, as SPNs can be granted only the necessary access.

Service Principal Name (SPN) Ownership

Advantages:

- **Consistency**: SPNs are not tied to an individual user, which ensures that flows continue to run regardless of personnel changes (e.g., if a user leaves the organization).
- **Security**: SPNs can be configured with the minimum required permissions, reducing the risk of unauthorized access.
- **Scalability**: SPNs are better suited for managing automated tasks across large environments, especially in scenarios involving multiple flows or environments.
- **Compliance**: Using SPNs can help maintain compliance, as they provide a clearer audit trail of actions taken by the flow, independent of specific users.

User Account Ownership

Advantages:

- Ease of Setup: Flows can be created and owned by users with minimal setup, making it easier for users to develop and manage their own flows.
- Human Interaction: If the flow involves tasks that require human approval, decision-making, or interaction, using a user account might be more appropriate.
- Personalized Context: Flows owned by users can run in the context of that user, utilizing their permissions and settings

Best Practices

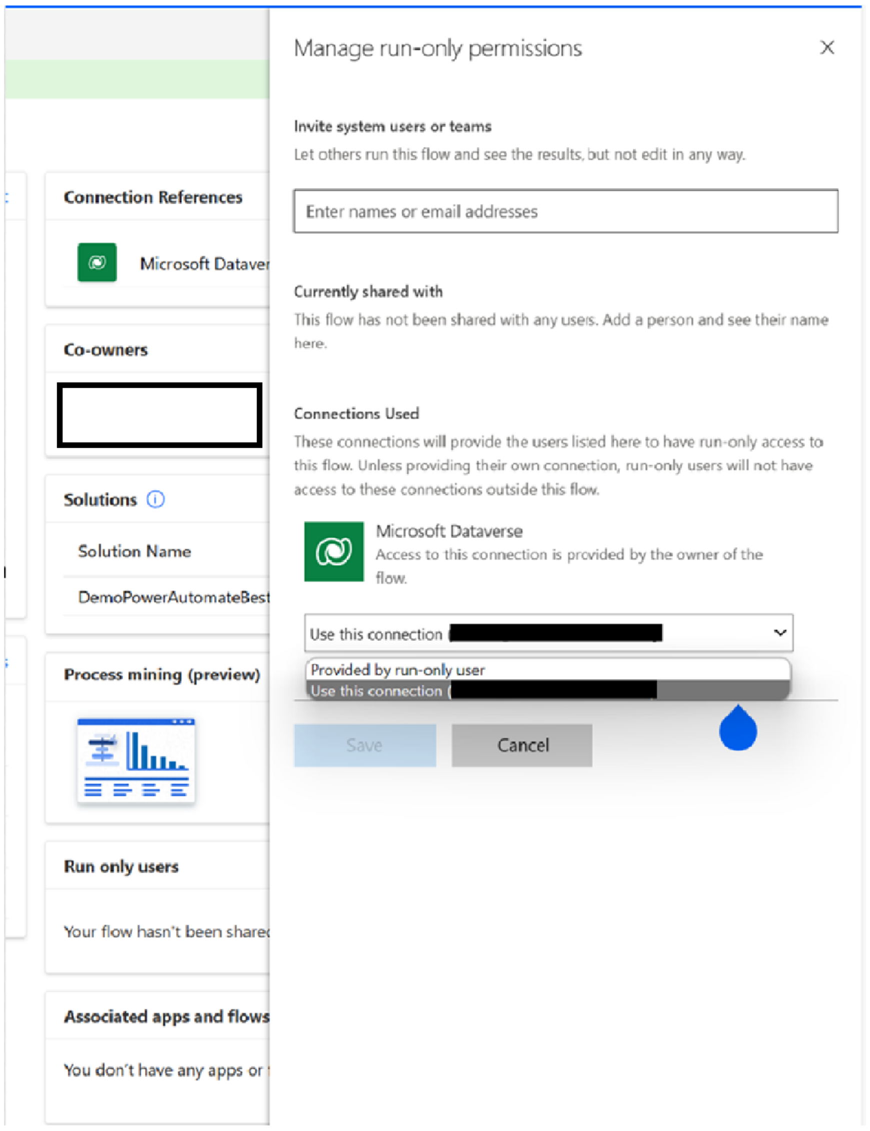
- Critical or Long-Running Flows: Use an SPN to ensure the flow's stability and independence from specific users.
- User-Specific or Interactive Flows: Use a user account when the flow requires the specific context of a user or involves user interaction.

Co-Ownership of Flows

Adding a co-owner to a cloud flow is the most common way to share a cloud flow. Any owner of a cloud flow can perform these actions:

- View the run history.
- Manage the properties of the flow (for example, start or stop the flow, add owners, or update credentials for a connection).
- Edit the definition of the flow (for example, add or remove an action or condition).
- Add or remove other owners (but not the flow's creator), including guest users.
- Delete the flow.

Therefore, only add co-owners for flow collaborations as needed. In most cases, if flow needs to be shared, share the flow with run-only permissions, which will restrict the users to view flow run history or make any changes to the flows. This will also allow users to specify whether the flow will use the connections of the user invoking the flow or the inbuilt connections of the user /account creating the flow. For more information, refer here



Connection References

Microsoft Datavei

Co-owners

Solutions

Solution Name

DemoPowerAutomateBest

Process mining (preview)

Run only users

Your flow hasn't been sharec

Associated apps and flows

You don't have any apps or

Manage run-only permissions

Invite system users or teams

Let others run this flow and see the results, but not edit in any way.

Enter names or email addresses

Currently shared with

This flow has not been shared with any users. Add a person and see their name here.

Connections Used

These connections will provide the users listed here to have run-only access to this flow. Unless providing their own connection, run-only users will not have access to these connections outside this flow.

Microsoft Dataverse

Access to this connection is provided by the owner of the flow.

Use this connection ([redacted]

Provided by run-only user

Use this connection ([redacted]

Save

Cancel

Configuring security roles for appropriate access

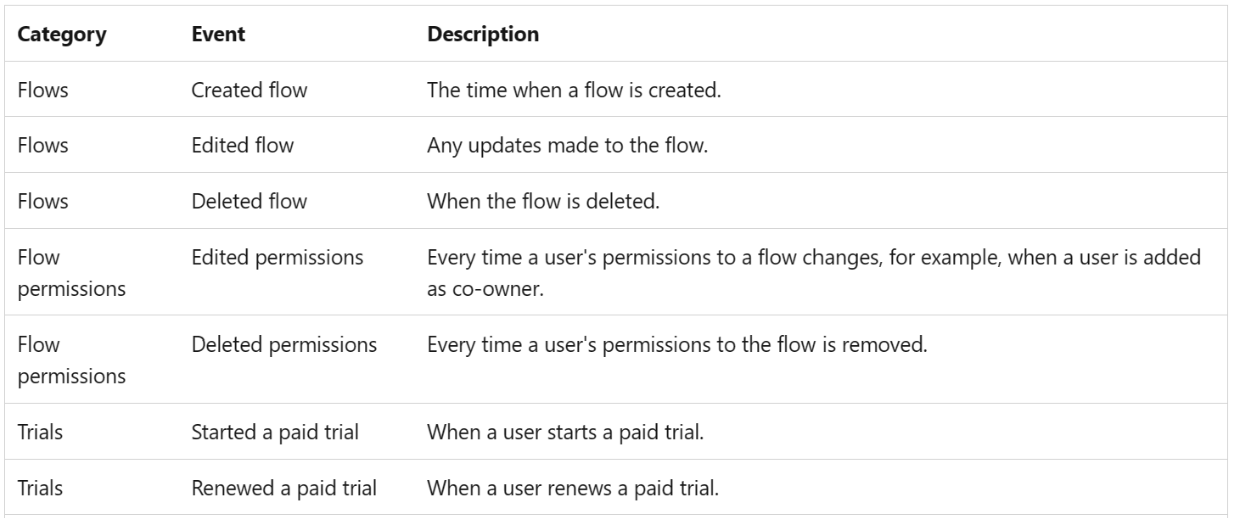
Apart from making users as co-owners, certain users can get access to flows by virtue of their security roles on Workflow tables or their role as environment admins.
In general, users with full access to Workflow tables, can edit any flow and view any run history. Similarly, environment admins will always have full access to edit workflows and view all the data that flows through them.

Sharing flows vs Send a copy

Using Auditing in Dataverse and Purview

Enabling auditing on Workflows and Connection Reference tables can give more visibility on when and who creates/changes/deleted or shares a Power Automate flows (applicable to all the flows created within solutions).
Additionally, Microsoft Purview can be used further to track cloud flow operations and who performed them.

For more information, refer [View Power Automate audit logs - Power Platform | Microsoft Learn]



Category	Event	Description
Flows	Created flow	The time when a flow is created.
Flows	Edited flow	Any updates made to the flow.
Flows	Deleted flow	When the flow is deleted.
Flow permissions	Edited permissions	Every time a user's permissions to a flow changes, for example, when a user is added as co-owner.
Flow permissions	Deleted permissions	Every time a user's permissions to the flow is removed.
Trials	Started a paid trial	When a user starts a paid trial.
Trials	Renewed a paid trial	When a user renews a paid trial.

Using Data Loss Policies

"Data Loss Prevention" (DLP) policies are relevant for managing and controlling data sharing and movement across Power Apps, Power Automate, and other Power Platform components. These policies help organizations prevent data loss and ensure compliance with regulatory requirements and internal policies.

Some of the best practices to consider while creating DLP are-

- Block/isolate non-business connectors, especially in default environments.
- Consider blocking high risk operations such as HTTP, HTTP with AAD and SharePoint URL etc. in environments with many makers.
- Lastly, leverage endpoint filtering to make sure only expected endpoints are reached.

For more details, refer [Manage data loss prevention (DLP) policies - Power Platform | Microsoft Learn]

CMK (Customer Managed Keys) for Cloud flows

Customer Managed Keys (CMK) for cloud flows in Power Automate is a security feature that allows organizations to use their own encryption keys to encrypt and protect data within their flows. This provides enhanced control over the encryption keys and meets specific compliance requirements for data security and privacy. Here's an overview of how CMK works in Power Automate and how to implement it:

What is Customer Managed Keys (CMK)?
Customer Managed Keys (CMK) enable organizations to manage their own encryption keys, typically using a key management service such as Azure Key Vault. By using CMK, organizations can:

- Gain greater control over the encryption keys used to protect their data.
- Meet regulatory and compliance requirements that mandate customer-controlled encryption.
- Revoke access to data by disabling or deleting keys.

How CMK Works in Power Automate

1. **Encryption at Rest**: CMK allows you to encrypt the data stored by Power Automate at rest using your own keys.
2. **Azure Key Vault**: The keys are typically stored and managed in Azure Key Vault, which provides secure key management capabilities.
3. **Key Rotation and Management**: You have the ability to rotate, revoke, and audit the keys, providing enhanced control over the encryption lifecycle.

For more information, refer to the following learn document
More details on Managing CMK

ALM best practices for Power Automate

ALM (Application Lifecycle Management) enables organizations to safely move artifacts within a solution across environments such as Dev, Test and Prod

- **Unmanaged solutions** are fully editable and allow developers to modify and customize the components within a solution.
- **Managed solutions** on the other hand are sealed and do not allow any modifications once deployed to target environments.

While moving solutions from one environment to another, here are some of the key deployment practices that should be followed:

Solutions best practices

- Solutions can be used as a logical boundary to keep related flows together. Consider keeping parent and child flows together in one solutions who may require to use same connection reference.

- Environment variables in PowerApps allow you to dynamically change configurations across different environments (e.g., development, staging, production) without modifying the app's code. These variables act as placeholders for data that can be set or updated externally, making it easier to manage and deploy the application.

- Azure Key Vault, a secure storage service, complements this by securely storing sensitive information like secrets, keys, and certificates. Integrating Azure Key Vault with Power Automate ensures that sensitive data is stored and retrieved securely. This integration allows Power Automate to access the secrets directly from Azure Key Vault, reducing the risk of exposure and enhancing the overall security posture of the application.

Deployment best practices

- To ensure safe ALM, make sure to not allow any unmanaged customizations in the Test and Production environments by deploying only managed solutions.
- Any mission critical flows should only be edited in a dev environment and then moved to test and then production environments. Within the production environments, ensure that flows should be owned by service principals.

Additionally, any changes to production environments should always go through ALM processes.

Use Pipelines for deployment

Pipelines provides an easy way to move solutions from Dev to Stage to Prod environments. Pipelines also gives ability to deploy solutions via service principal and allow solutions rollback. Additionally, pipelines can be extended for approvals, to github etc.

Understanding when to integrate Flows with Power Apps

There are a range of capabilities available through Power Platform that enable makers to create business logic for their Power Apps. Below are the primary methods along with guidance to determine which one to use:

Power Apps with Power Fx

Power Fx is the low code functional programming language shared by Excel and Power Platform. Combined with Power Platform connectors, makers can access data as well as create custom app logic. Power Fx supports the following characteristics:

- **Live** – Canvas Power Apps "recalc" just as Excel spreadsheets do. As the end user interacts with the app, Power Fx is in the background making data requests and keeping the state of the app up to date with Dataverse.
- **Delegation** – Power Fx automatically "delegates" what it can to the server. Functions such as Filter(), Lookup(), and Search() can enable an app to access to data using server side filtering, so that only sufficient data is brought into the app to support the experience and functional logic. Where delegation is not possible, functions execute in the local JavaScript context (i.e. in the local browser).
- **Optimized for Dataverse** – Dataverse is the native data storage service for Power Apps and as such there are fewer layers involved in using Power Fx in conjunction with Dataverse (which means low latency access). In addition, Power Fx supports the various many to one and one to many relationships available in Dataverse.
- **Offline** – Dataverse data can be taken offline with a mobile player for use in the field. Changes on both sides are synced when reconnected. Business logic defined in the app is run even when offline.

Power Automate

A low code workflow service built on top of the Power Platform connector ecosystem. Power Automate adds the following capabilities when building Power Apps:

- **Asynchronous** – A Power Automate cloud flow is inherently asynchronous. This means that when a flow is initiated, it is leveraging a queuing system to manage the various subtasks. The asynchronous nature of Power Automate means that it is well suited for longer running complex sequences of logic.
- **Detailed logging** – All flows create a record of what happened when they executed, called the "Run History". This provides traceability and ensures there's an auditing record for what happened and why.
- **Multi-connector** – Although it is possible to create multi-connector Power Apps logic, due to the live nature of Power Apps, as you increase the number of connectors performance will degrade. These complex multi-connector scenarios are a great place to leverage Power Automate which can offload these cases from live execution in the app.

Dataverse Plugins with Power Fx

Written in C#, Dataverse plugins are a common way that professional developers have expressed business logic for years. As a new low code option, Power Fx can now be used to create plugins, in preview now and will GA within a few months. It brings delegation and Dataverse optimizations as well as these characteristics:

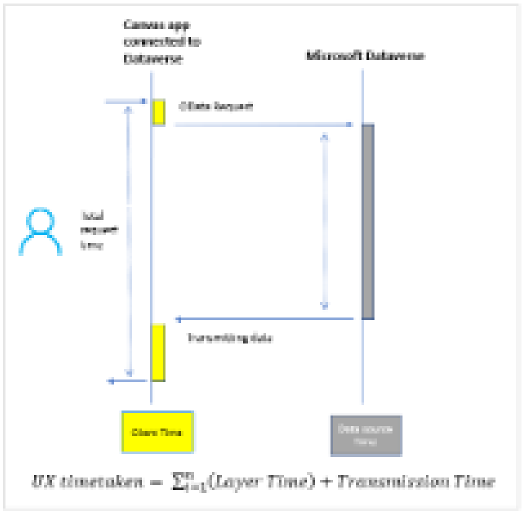
- **In the transaction** – Power Fx plugins run in the Dataverse transaction that is updating the database. Even if there is a problem updating a secondary table, all the changes are rolled back, always keeping the database in a consistent state.
- **Error reporting** – Reporting an error all the way back to the end user in an app or flow is as simple as calling the Error() function. The end user will see the error as a result of their action, they can take immediate corrective action, and retry the operation.
- **Common choke point** – There is no getting around the plugin. Common update business logic can be written and maintained in one place for all apps, flows, and other Power Platform end points.
- **Efficiency** – Plugins can make multiple connector and database calls in succession without the storage and networking overhead of each step if they were in a Power Automate flow.
- **Increased security –** Dataverse virtual network support allows you to protect your outbound connections to resources within your private network. In this way, you can securely manage your egress traffic from the Power Platform according to your network policy. In addition, data can be insulated and protected from clients by wrapping access in a server-side plugin.

When you should use Power Fx Vs Power Automate in Power Apps

Power Fx in the app should be the default mechanism used for building Power App business logic - Like any tool however there are sweet spots and break points where it makes sense to leverage other tools in the toolbox.

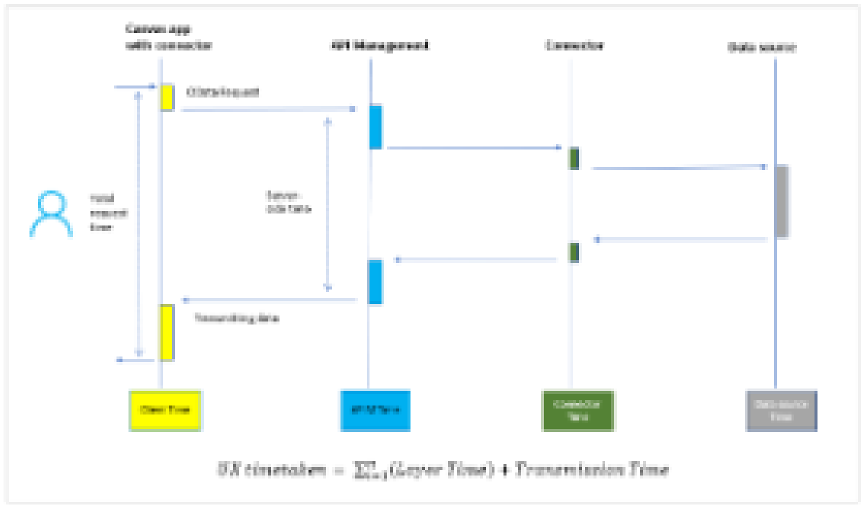
Low Latency Use Cases

If **low latency** in a Power App is the focus, the live nature of a Power Fx function means that you have the **best ability to deliver low latency business logic via Power Fx**. Achieving low latency depends significantly on the complexity of the task and size of the dataset involved. It is therefore important to highlight, that although you can achieve low latency through Power Fx, what you're trying to do and how it's been designed will have a significant bearing on the performance realized. Additional guidance available here.
For common data access scenarios, Dataverse combined with Power Fx is going to be the fastest approach.



Complex sequences & multi-connector

For **complex sequences** of actions, across multiple connectors, **then Power Automate is an asynchronous mechanism to offload processing from the Power App**. In addition, the asynchronous nature of Power Automate allows the Power App to initiate a workflow and enable the experience to move on without waiting for a response.



<more to add here>

Centralized business logic

If the business logic is gating changes to the database, for example validation before adding a record, then **Dataverse plugins with Power Fx is the best answer**. No other solution is a part of the Dataverse transaction and can return an error all the way back to the end user who initiated the original change, allowing them to make corrections, and retry, while keeping the database consistent throughout. Plugins provide a unified choke point to ensure the same business logic is enforced for all apps (Canvas and Model-driven), flows, and other end points. Similar to Power Automate, Dataverse plugins with Power Fx also provide a central location for defining custom actions that don't necessarily change the database. This facilitates sharing of Power Fx based business logic across all Power Platform end points.

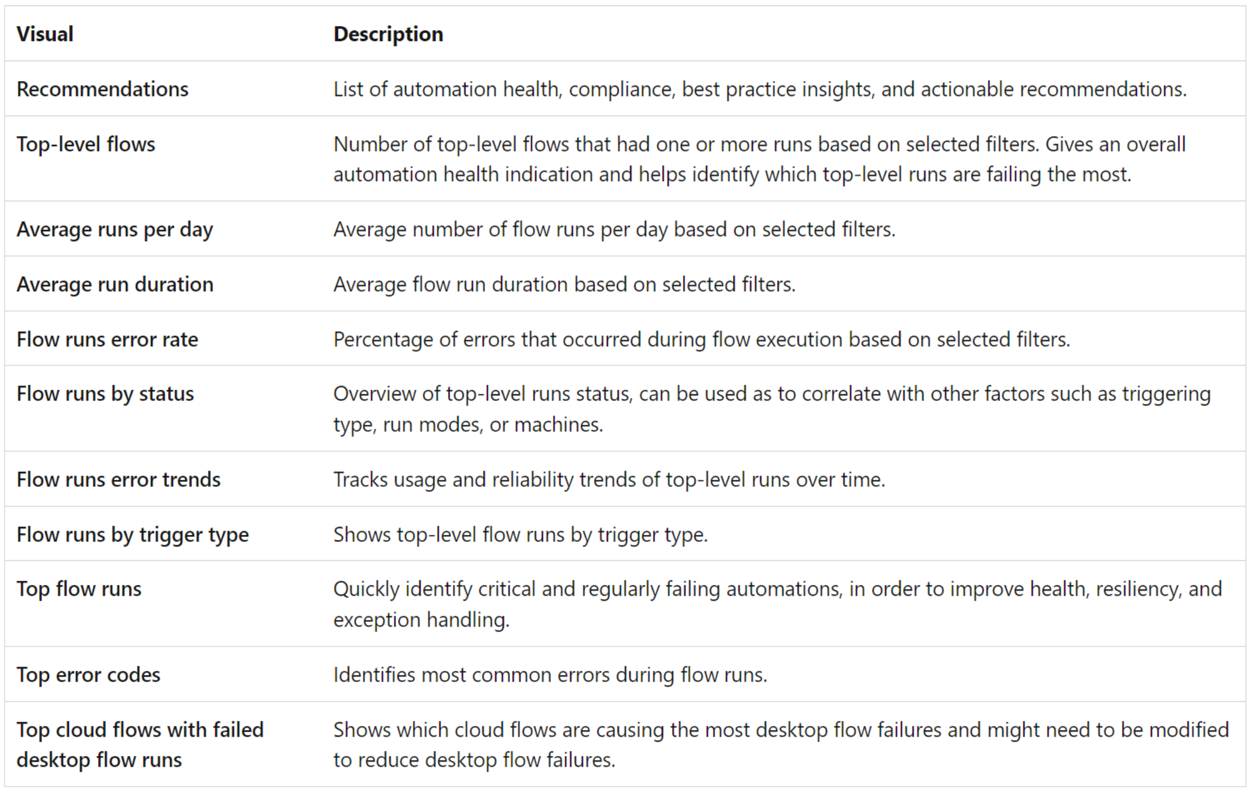
Monitoring and Alerting

Regularly monitoring Power Automate flows is vital for businesses to ensure the smooth and efficient operation of their workflows. This practice enables organizations to promptly identify and resolve any emerging issues or bottlenecks that could hinder the automation process. Through proactive monitoring, businesses can implement necessary measures to enhance overall performance. Moreover, monitoring aids in the detection and mitigation of potential security vulnerabilities or compliance concerns, safeguarding the secure management of sensitive data.
Furthermore, monitoring offers valuable insights into usage patterns and performance metrics, empowering businesses to optimize their workflows for heightened productivity and cost-effectiveness.

Automation Center

The Automation Center in Power Automate is a central hub for managing and overseeing all your automation efforts. It provides a comprehensive interface to create, monitor, and optimize your automated workflows across the entire organization. This feature helps streamline operations, enhance productivity, and ensure that all automation tasks are running efficiently.

Here are some of the visuals that Automation center will show:

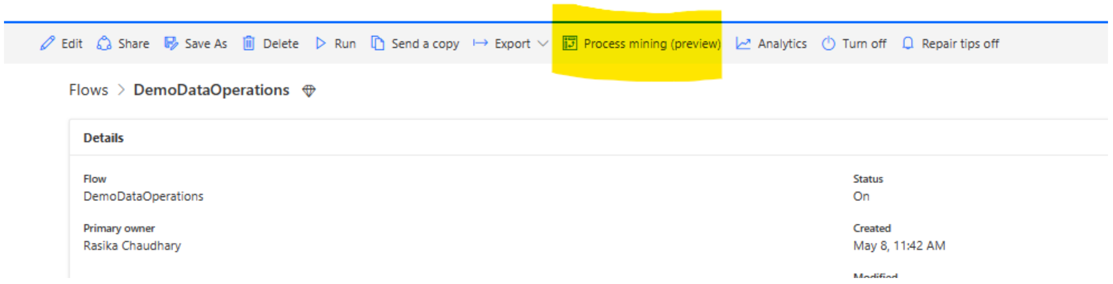


Visual	Description
Recommendations	List of automation health, compliance, best practice insights, and actionable recommendations.
Top-level flows	Number of top-level flows that had one or more runs based on selected filters. Gives an overall automation health indication and helps identify which top-level runs are failing the most.
Average runs per day	Average number of flow runs per day based on selected filters.
Average run duration	Average flow run duration based on selected filters.
Flow runs error rate	Percentage of errors that occurred during flow execution based on selected filters.
Flow runs by status	Overview of top-level runs status, can be used as to correlate with other factors such as triggering type, run modes, or machines.
Flow runs error trends	Tracks usage and reliability trends of top-level runs over time.
Flow runs by trigger type	Shows top-level flow runs by trigger type.
Top flow runs	Quickly identify critical and regularly failing automations, in order to improve health, resiliency, and exception handling.
Top error codes	Identifies most common errors during flow runs.
Top cloud flows with failed desktop flow runs	Shows which cloud flows are causing the most desktop flow failures and might need to be modified to reduce desktop flow failures.

For Automation center to work, Flow run history need to be stored in Dataverse. For more details on Automation center, refer here.

Leverage Process Mining (Preview)

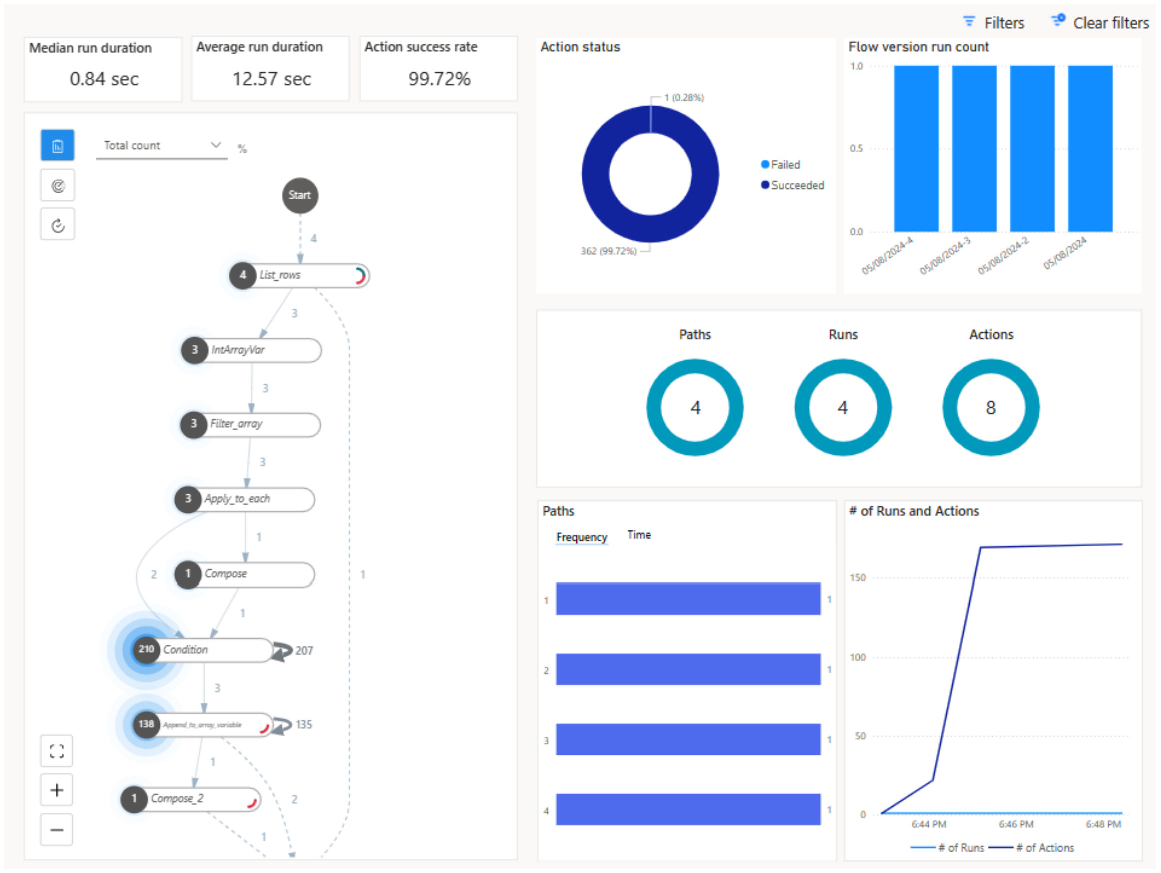
The process mining capability enables you to obtain useful insights and enhance your cloud flows. Users can see how your flow performs, find out where it slows down or can be improved, and track any changes in performance. By using the process mining capability to examine your flow's run history with process mining methods, you can produce these insights from the flow details page directly.
To get more information on the prerequisites and known limitations, please visit here



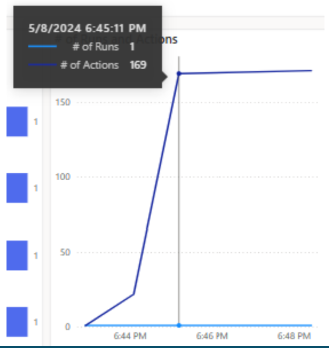
Flows > DemoDataOperations

Details

Flow DemoDataOperations	Status On
Primary owner Rasika Chaudhary	Created May 8, 11:42 AM



The Runs and actions section also gives information about the total action count that a flow run took. This is important to understand if your flow has a possibility of running into daily action bursts throttling limits.



Power Automate Analytics

Power Automate Analytics refers to the suite of tools and features available for analyzing and gaining insights into the performance, usage, and effectiveness of your Power Automate workflows.
Power Automate analytics reports are available at the tenant level as well as individual cloud flows level.

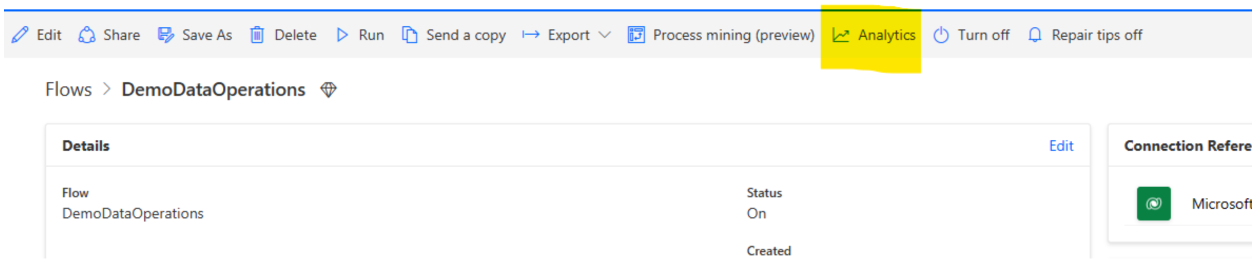
Cloud Flow Analytics

Power Automate offers built-in analytics capabilities that allow you to monitor flow runs, track execution history, and analyze flow performance. Leverage analytics dashboards and reports to gain insights into flow execution patterns and identify bottlenecks.

Microsoft Cloud Flow Analytics is a service offered by Microsoft that provides insights and analytics into the usage and performance of Microsoft Power Automate. You can access Cloud Flow Analytics using the menu button “Analytics“ from each Microsoft Flow detail page.

Following details will be shown under this:

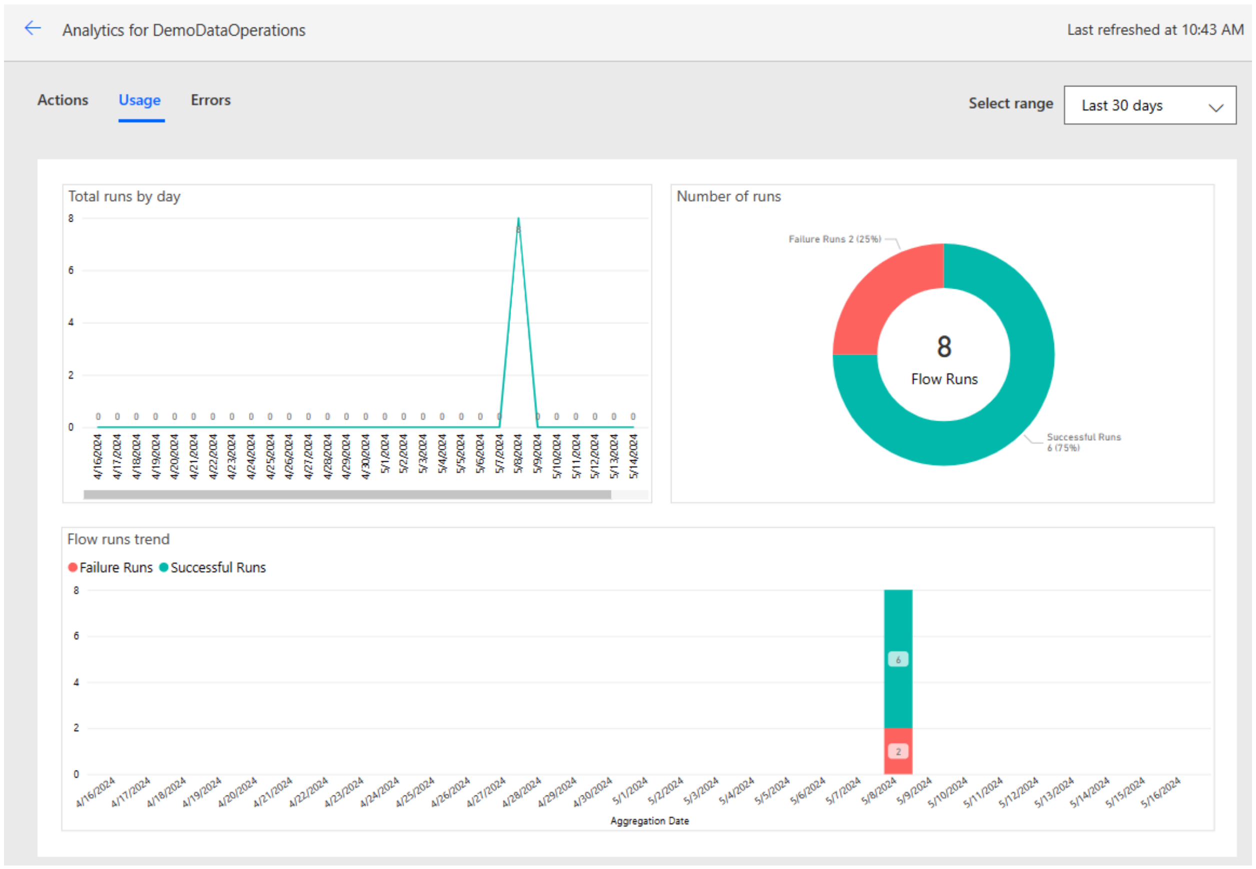
- Flow usage (including number of actions requests from the cloud flow runs)
- Detailed flow error analysis (type of error, count and last occurrence)
- History of 30 days



Flow
DemoDataOperations

Status
On

Created



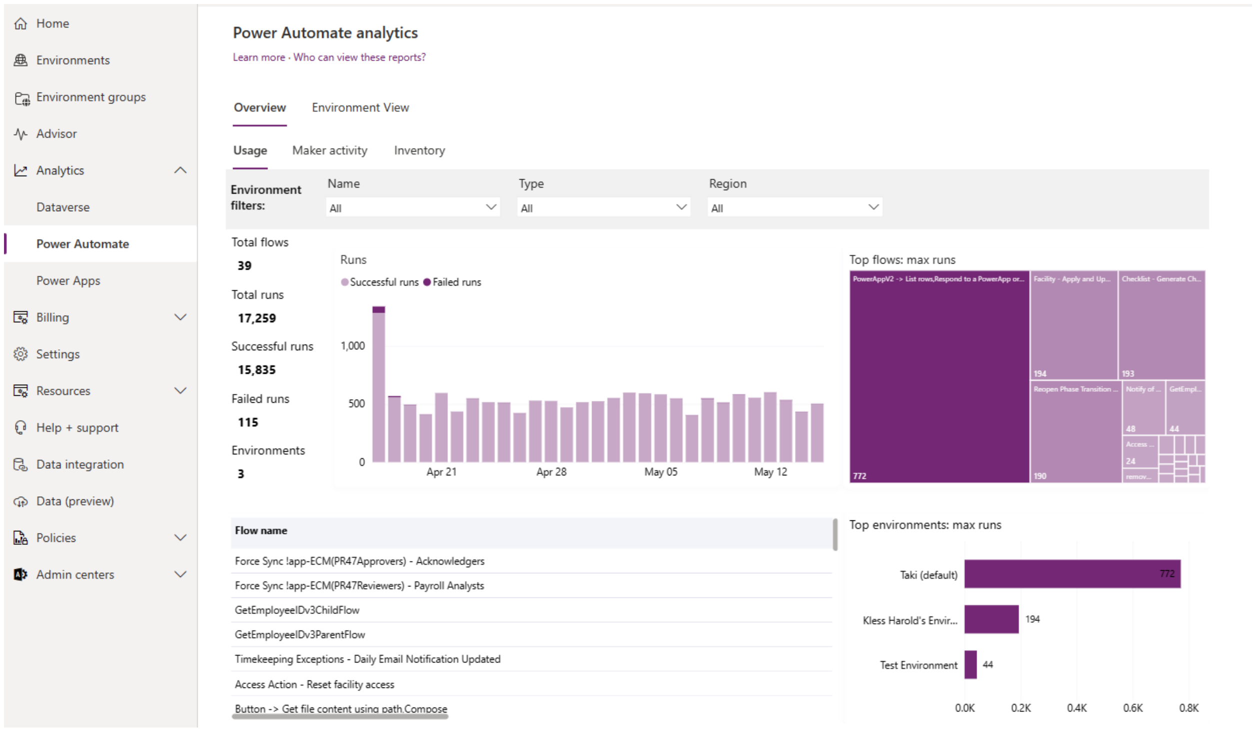
Power Automate Admin Analytics

Environment admins can access analytics for Power Automate in the Microsoft Power Platform admin center. The reports provide insights into runs, usage, errors, types of flows created, shared flows, and details on connectors associated with all the different flow types like automated flows, button flows, scheduled flows, approval flows, business process flows.

Covers following details :

- Reports of flow usage based on environments, environment type and region
- Flow inventory, including flow state
- Maker activity (but only the user ID)
- Number of failed runs (no information which flow failed)
- 28 days history

For more details on Admin analytics refer here



Flow name
Force Sync !app-ECM(PR47Approvers) - Acknowledgers
Force Sync !app-ECM(PR47Reviewers) - Payroll Analysts
GetEmployeeIDv3ChildFlow
GetEmployeeIDv3ParentFlow
Timekeeping Exceptions - Daily Email Notification Updated
Access Action - Reset facility access
Button -> Get file content using path,Compose

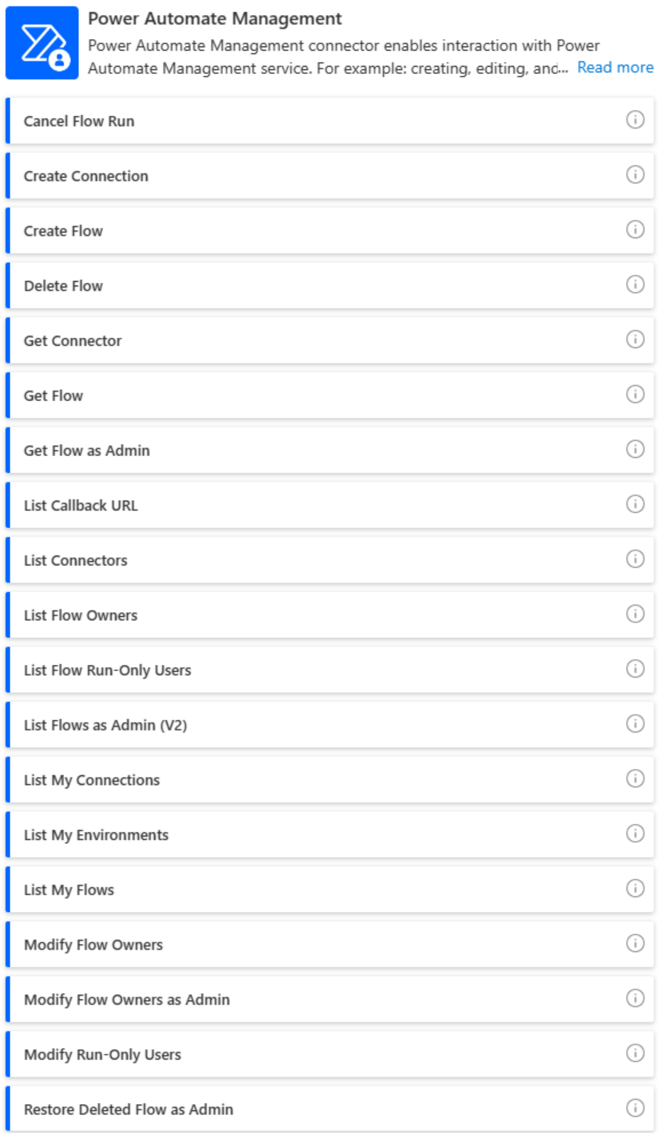
Power Automate Management connectors

Power Automate service also provides ways to automate some of the management activities for Flows via 2 connectors

- Power Automate Management connector
- Power Automate for Admins

Power Automate Management connector

This connector provides actions for managing flows, such as creating, updating, and deleting flows, as well as retrieving flow metadata and run history. It allows you to programmatically interact with flows within your Power Automate environment. Admins can run some of these actions which are suffixed with “As Admin”. More details on this connector can be found here



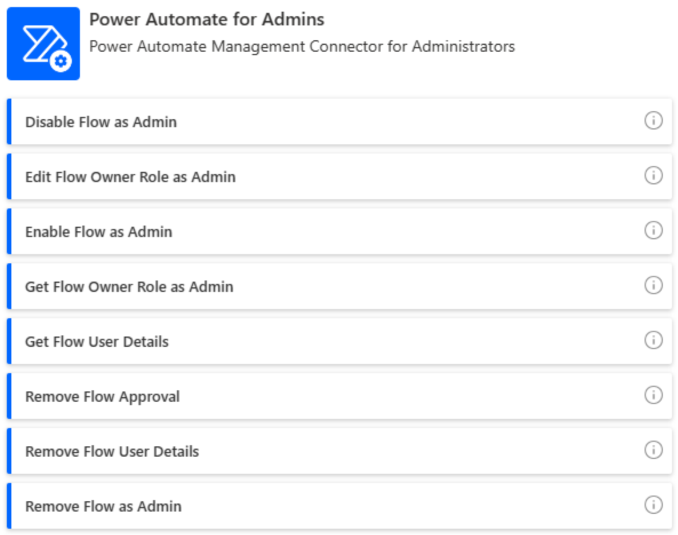
Power Automate Management

Power Automate Management connector enables interaction with Power Automate Management service. For example: creating, editing, and... Read more

- Cancel Flow Run
- Create Connection
- Create Flow
- Delete Flow
- Get Connector
- Get Flow
- Get Flow as Admin
- List Callback URL
- List Connectors
- List Flow Owners
- List Flow Run-Only Users
- List Flows as Admin (V2)
- List My Connections
- List My Environments
- List My Flows
- Modify Flow Owners
- Modify Flow Owners as Admin
- Modify Run-Only Users
- Restore Deleted Flow as Admin

Power Automate for Admins

Power Automate for admins connector on the other hand are used mostly by administrators giving them the ability to enable/disable flows, change flow owners etc. More details on this connector could be found here


Power Automate for Admins
Power Automate Management Connector for Administrators

Disable Flow as Admin
Edit Flow Owner Role as Admin
Enable Flow as Admin
Get Flow Owner Role as Admin
Get Flow User Details
Remove Flow Approval
Remove Flow User Details
Remove Flow as Admin

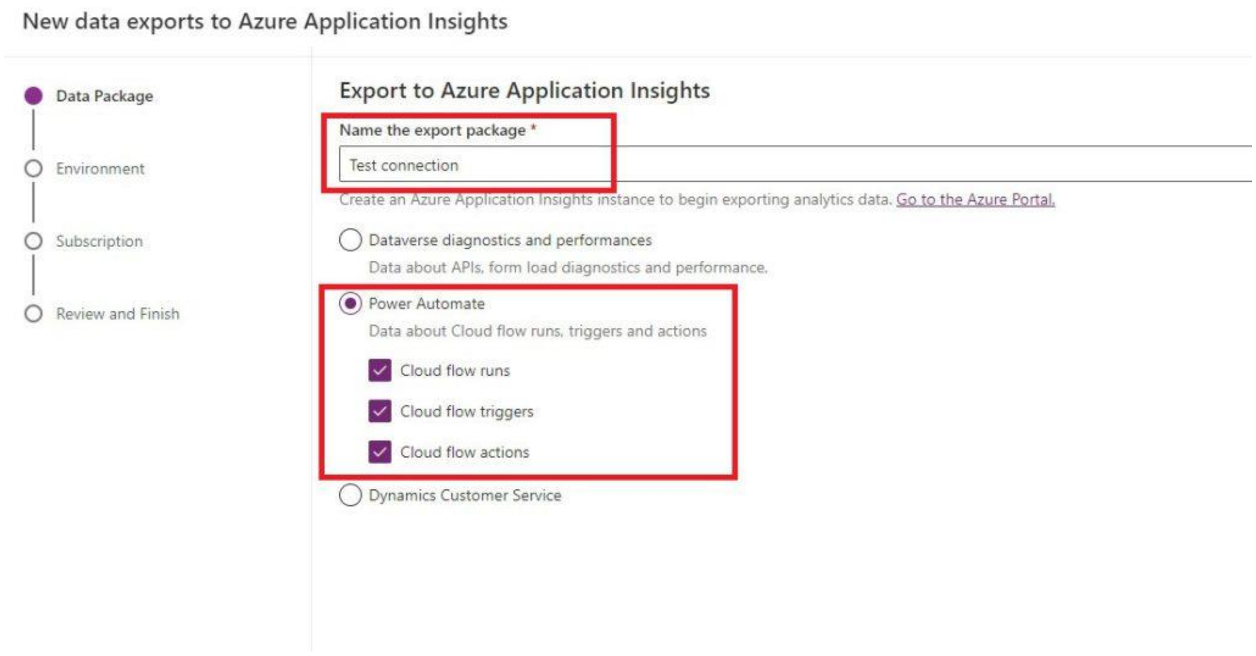
Power Automate App Insights

Power Automate App Insights has been one of the most recent additions to telemetry data that can be accessible to users. Customers can now leverage Azure App Insights to diagnose and monitor their workflows running on Power Automate. In addition to this, users can use this telemetry to build dashboards, out-of-the-box and custom alerts, performance diagnostics and custom analysis via log analytics.

For Power Automate the flow runs, triggers and action-;eve; data can be lined with app insights at an environment level. However, data from multiple environments can log into same app insights resource.

Cloud flow runs telemetry get stored in **requests** table and trigger and action level data gets stored in the **dependencies** table.
For more details, refer to this article

To transfer data to app insights, select Data Export under Analytics and configure as follows:


New data exports to Azure Application Insights

- Data Package
- Environment
- Subscription
- Review and Finish

Export to Azure Application Insights

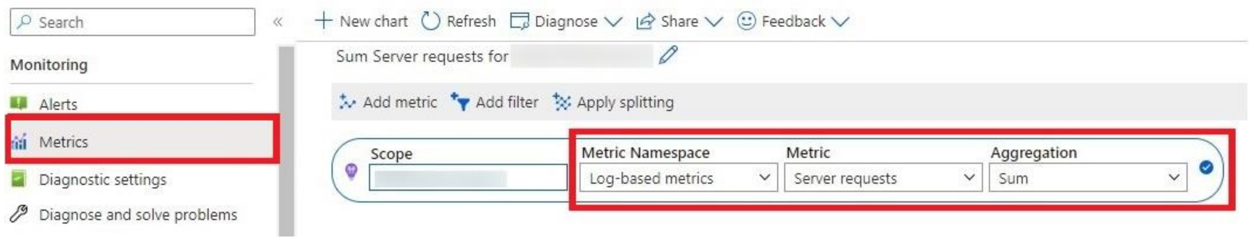
Name the export package *

Test connection

Create an Azure Application Insights instance to begin exporting analytics data. Go to the Azure Portal.

- ◯ Dataverse diagnostics and performances
  Data about APIs, form load diagnostics and performance.
- ● Power Automate
  Data about Cloud flow runs, triggers and actions
  - ✓ Cloud flow runs
  - ✓ Cloud flow triggers
  - ✓ Cloud flow actions
- ◯ Dynamics Customer Service

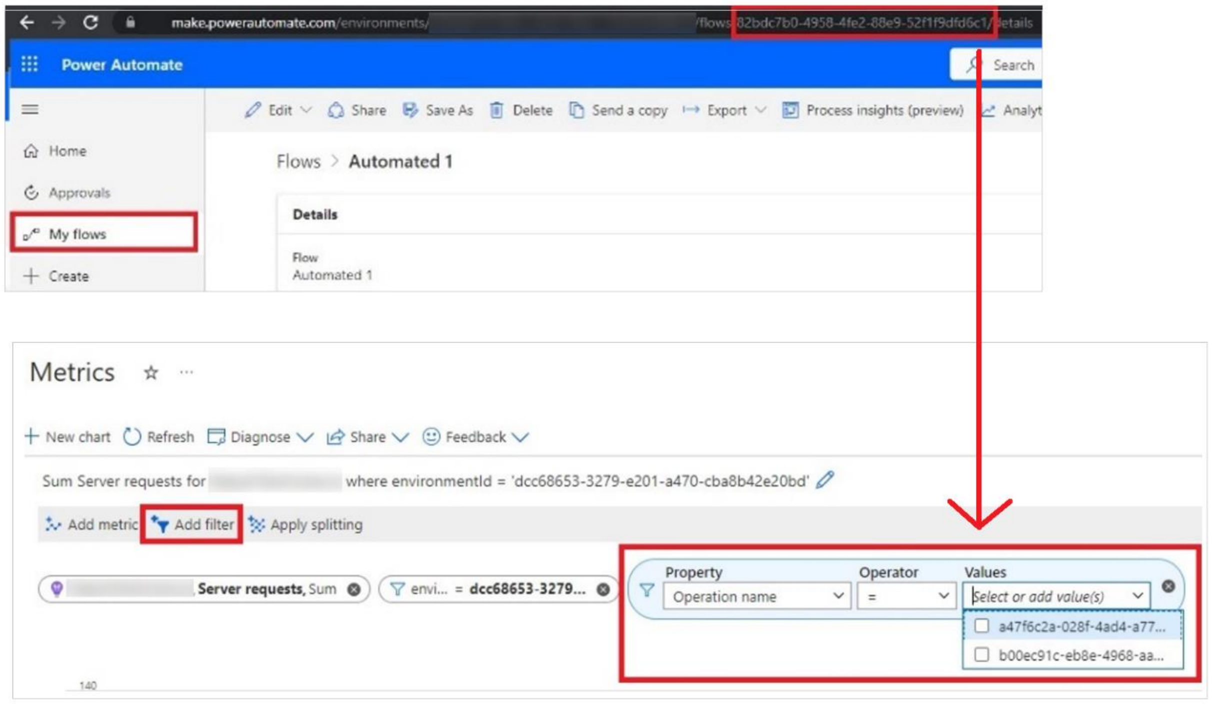
To monitor cloud flow executions, review the Metrics section and apply the following:



Sum Server requests for

Scope	Metric Namespace	Metric	Aggregation
	Log-based metrics	Server requests	Sum

You can further add environment and flow level filters


make.powerautomate.com/environments/[redacted]/flows/82bdc7b0-4958-4fe2-88e9-52f1f9dfd6c1/details

Power Automate

Edit | Share | Save As | Delete | Send a copy | Export | Process insights (preview) | Analyt

Home | Approvals | My flows | Create

Flows > Automated 1

Details

Flow: Automated 1

Metrics

New chart | Refresh | Diagnose | Share | Feedback

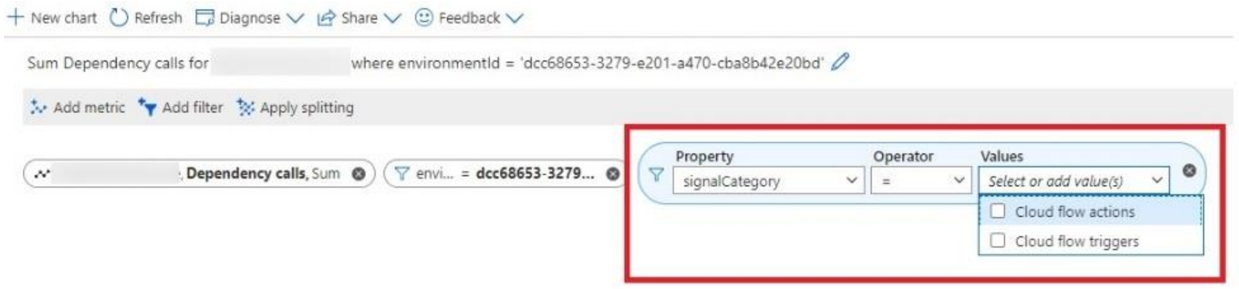
Sum Server requests for [redacted] where environmentId = 'dcc68653-3279-e201-a470-cba8b42e20bd'

Add metric | Add filter | Apply splitting

[redacted] Server requests, Sum | envi... = dcc68653-3279...

Property	Operator	Values
Operation name	=	Select or add value(s)
		a47f6c2a-028f-4ad4-a77...
		b00ec91c-eb8e-4968-aa...

For monitoring trigger and actions, under Metrics, filter for environment and then add Signal category for Property filter and select Cloud flow triggers and Cloud flow actions


New chart | Refresh | Diagnose | Share | Feedback

Sum Dependency calls for [redacted] where environmentId = 'dcc68653-3279-e201-a470-cba8b42e20bd'

Add metric | Add filter | Apply splitting

[redacted] Dependency calls, Sum | envi... = dcc68653-3279...

Property	Operator	Values
signalCategory	=	Select or add value(s)
		Cloud flow actions
		Cloud flow triggers

Flow Run history in Dataverse

This preview feature (as of June 2024) will allow the flow run details to be stored in Dataverse within the FlowRun table. This will empower Automation center to generate meaningful data about cloud flows execution.

By default, the execution data would be stored for 28 days period but the default time range can be changed from the admin center.

Here is some additional information for this feature:
[Known limitations for retaining flow run history in dataverse]
[FAQ]