

# Building Flows to Manage Content and Approvals



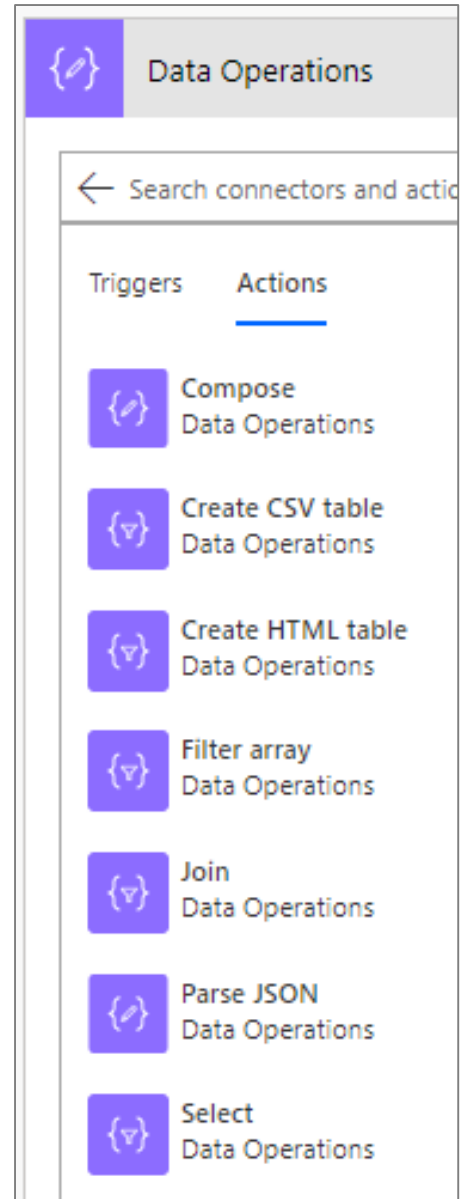
# Agenda

- Converting and Reshaping Data
- Uploading Photos to SharePoint
- Automating Approval Processes
- Integrating Flow with Microsoft Forms
- Handling Runtime Errors
- Understanding Parallel Execution



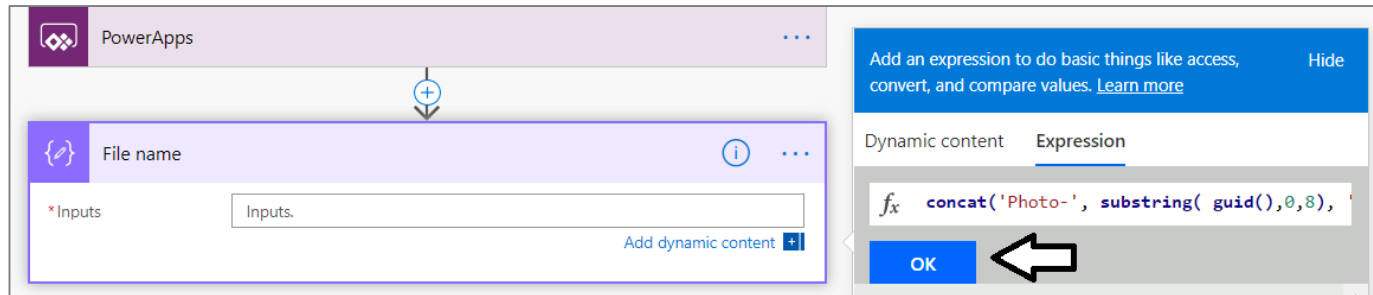
# Actions with Data Operations

- Compose
  - Used to parse together text, HTML, JSON, etc.
- Create CSV Table
  - Used to export data files with CSV format
- Create HTML Table
  - Used to create HTML tables for use in email messages
- Filter Array
  - Filter the contents of an existing array
- Join
  - Parse email address array into single string
- Parse JSON
  - Parse JSON returned from HTTP call into an object
- Select
  - Remap the columns in an array

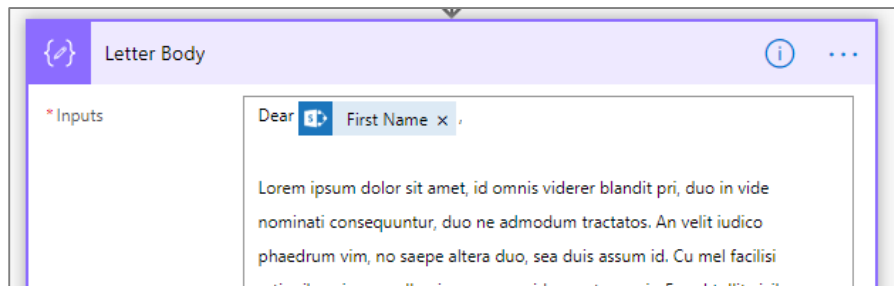


# Making Flows Maintainable with Compose

- Compose actions used to simplify flow building
  - Separates parsing content into its own named action
  - Makes flows easier to read and understand
  - Example 1: use Compose step to create a file name

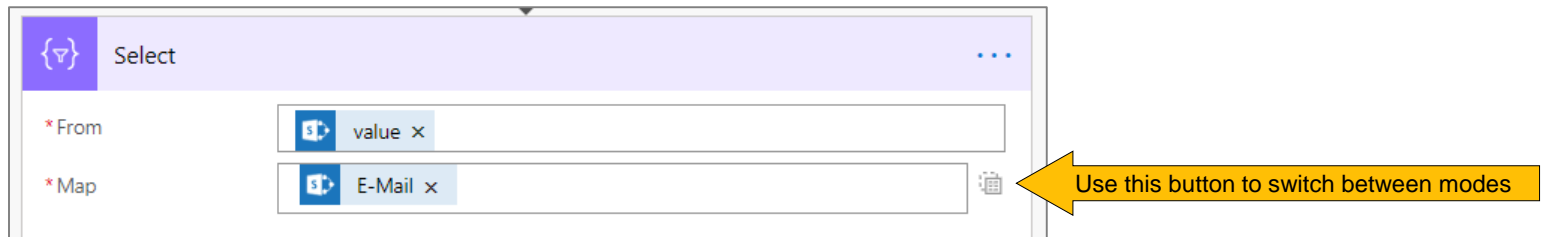


- Example 2: use compose step to generate content for letter

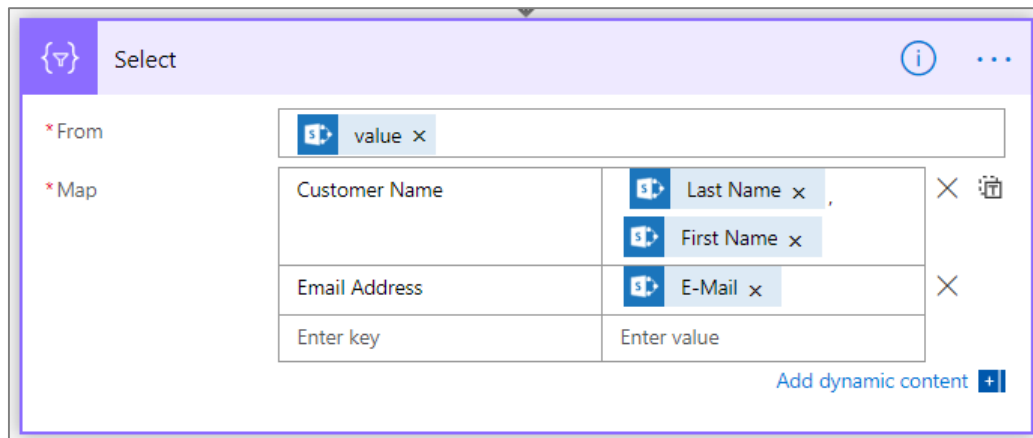


# Transforming Arrays using a Select Action

- The Select action configured to **Text mode** or **Key-Value mode**
- Text mode used to create a simple array
  - Example: use Text mode to create an array of email addresses

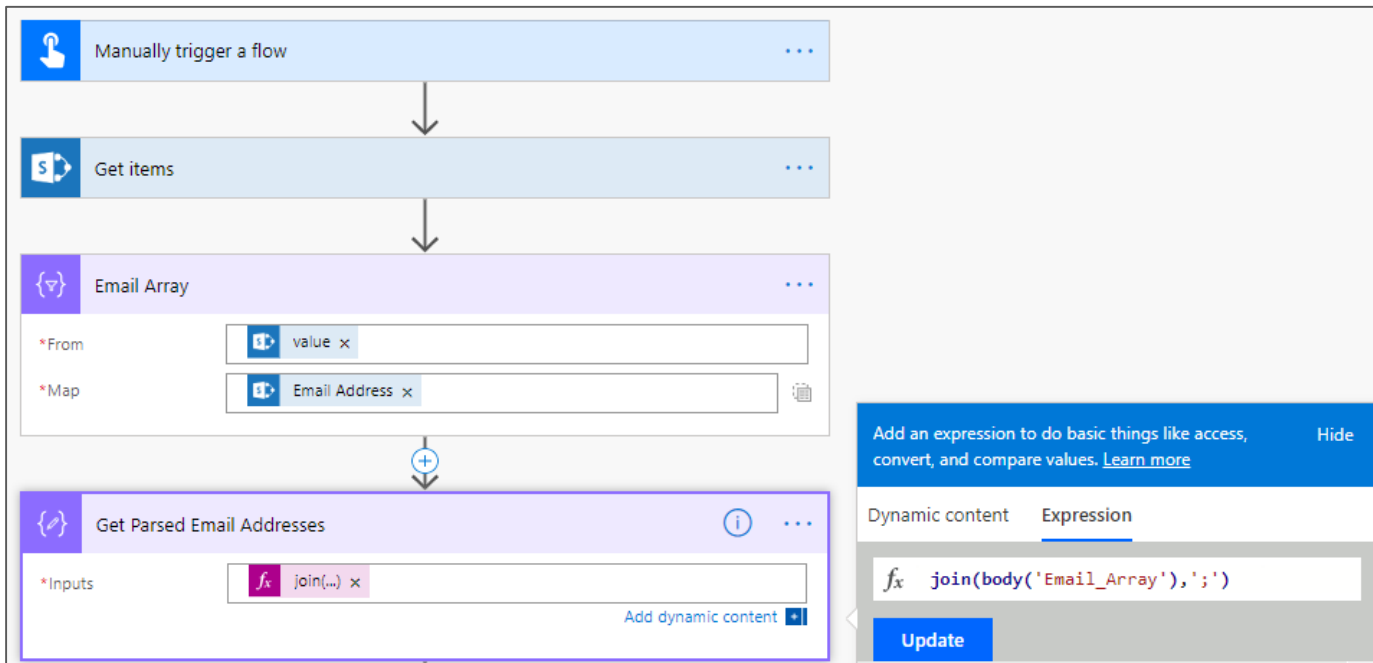


- Key-Value mode allows you to remap columns from table or list
  - Example use Key-Value mode to remap columns from Customers list



# Convert Array of Email Addresses to String

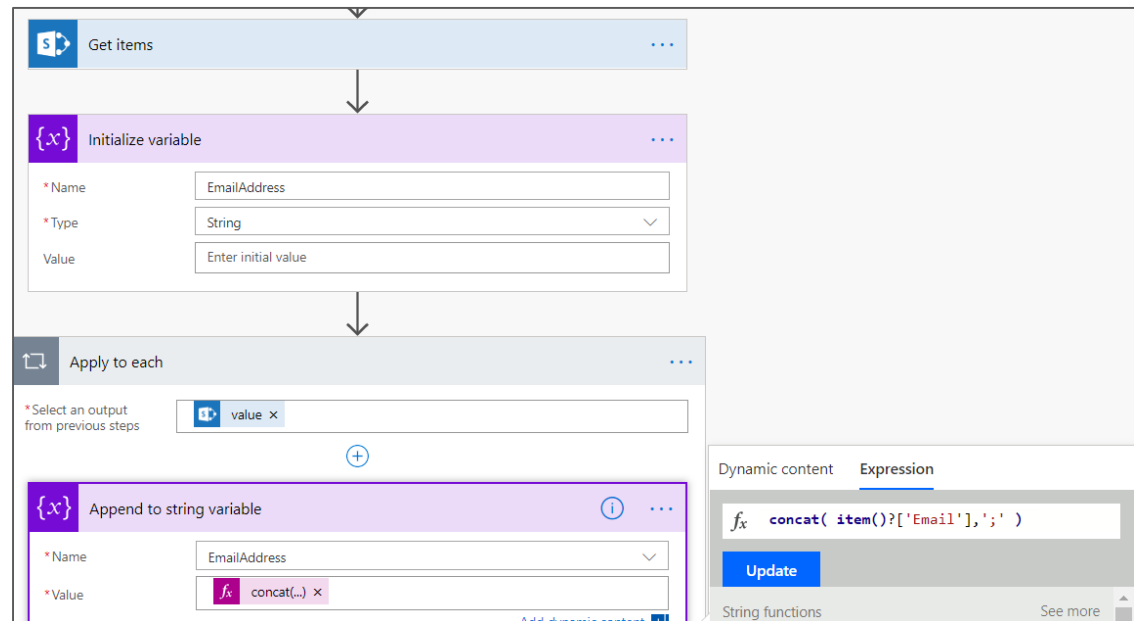
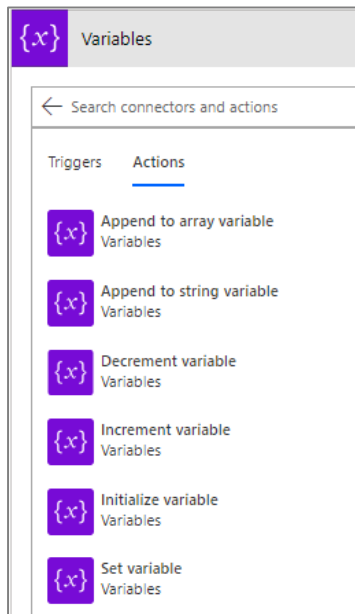
- Steps to create a semicolon-delimited list of emails
  - Use **Get Items** action to get SharePoint Customers list
  - Use **Select** action to create array of email addresses
  - Use **Compose** action to build string from array of email addresses
  - This string can be used in **To** field of **Send an Email** action





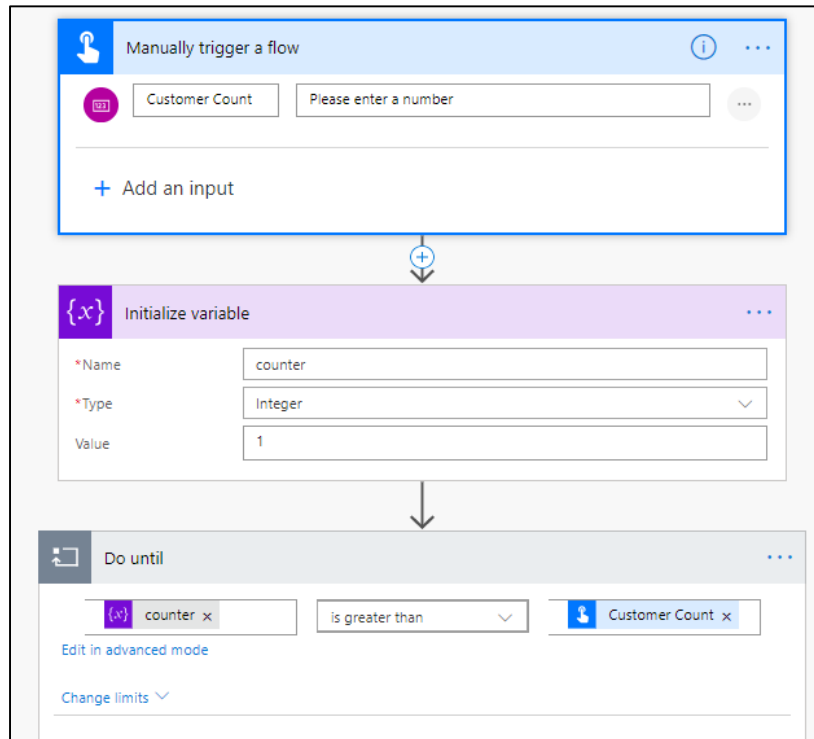
# Tracking State using Variables

- Variables used to track state during flow lifetime
  - **Initialize Variable** action used to create variable instance
  - Other variable actions uses to update variable values
  - By default, variable value persisted for lifetime of flow
  - Variables can be initialized inside **Scope** action to reduce lifetime



# Do Until Action with Counter Variable

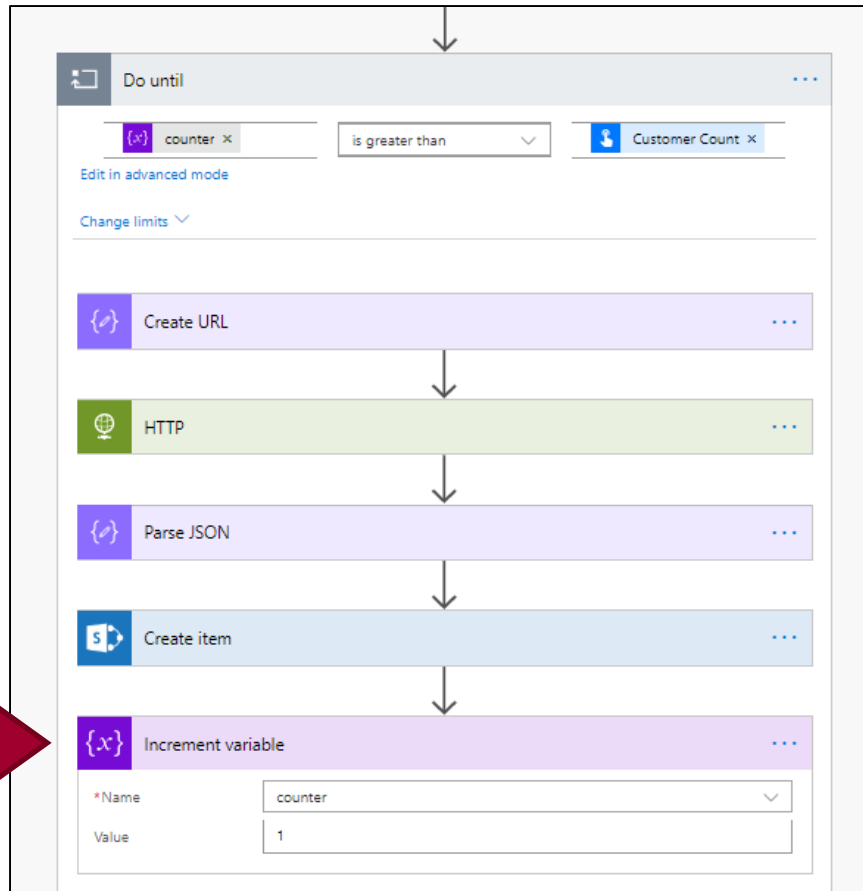
- Using a variable to control looping in **Do until** action
  - Initialize integer variable before **Do until** action to act as counter
  - **Do until** action condition checks variable value
  - **Do until** action body requires action to increment variable value





# Executing Operations inside Do Until Loop

- Implementation of **Do until** action body
  - **Increment variable** action used to change variable value



# Handling Type Conversion

- Some conversion is automatic
  - Sometimes conversions are performed for you
  - In other cases, you must explicitly convert between types



## **string(value)**

Convert the parameter to a string



## **float(value)**

Convert the parameter argument to a floating-point number



## **bool(value)**

Convert the parameter to a Boolean



## **base64(value)**

Returns the base 64 representation of the input string



## **base64ToBinary(value)**

Returns a binary representation of a base 64 encoded string



## **base64ToString(value)**

Returns a string representation of a base 64 encoded string



## **binary(value)**

Returns a binary representation of a value



## **dataUriToBinary(value)**

Returns a binary representation of a data URI



## **dataUriToString(value)**

Returns a string representation of a data URI



## **dataUri(value)**

Returns a data URI of a value



## **uriComponent(value)**

Returns a URI encoded representation of a value



## **uriComponentToBinary(value)**

Returns a binary representation of a URI encoded string



## **uriComponentToString(value)**

Returns a string representation of a URI encoded string



# dataUriToBinary()

- PowerApps photos require conversion
  - Allows you to upload photos to SharePoint
  - Accomplished using **dataUriToBinary()** function

```
dataUriToBinary(triggerBody()['Createfile_FileContent'])
```

The screenshot displays a PowerApps flow with two steps:

- Get File Name**: The input field contains the expression `concat(...)`.
- Create file**: This step includes the following fields:
  - Site Address**: Critical Path Training Labs Team Site - `https://msd0910.sharepoint.com/`
  - Folder Path**: `/My Photos`
  - File Name**: `Output`
  - File Content**: `dataUriToBinary(...)`

A dynamic content panel on the right shows the expression `dataUriToBinary(triggerBody()['Createfile_FileContent'])` and a list of string functions, including `concat(text_1, text_2?, ...)`.



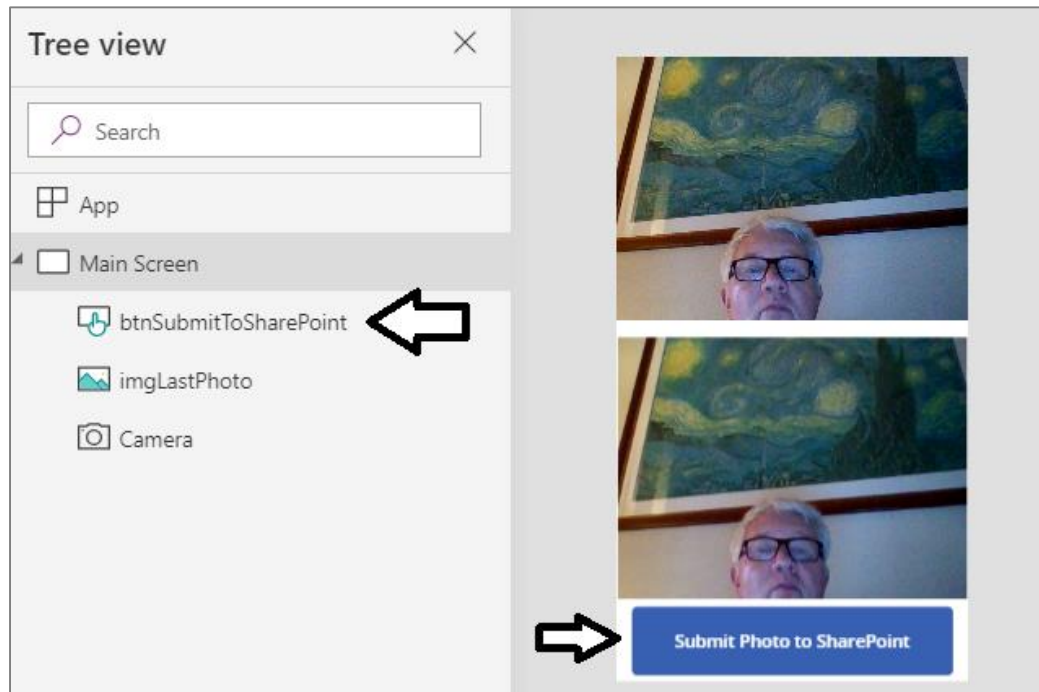
# Agenda

- ✓ Converting and Reshaping Data
- Uploading Photos to SharePoint
- Automating Approval Processes
- Integrating Flow with Microsoft Forms
- Handling Runtime Errors
- Understanding Parallel Execution



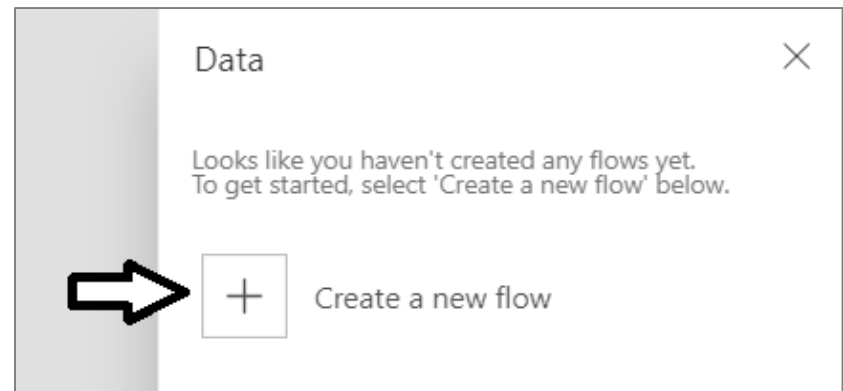
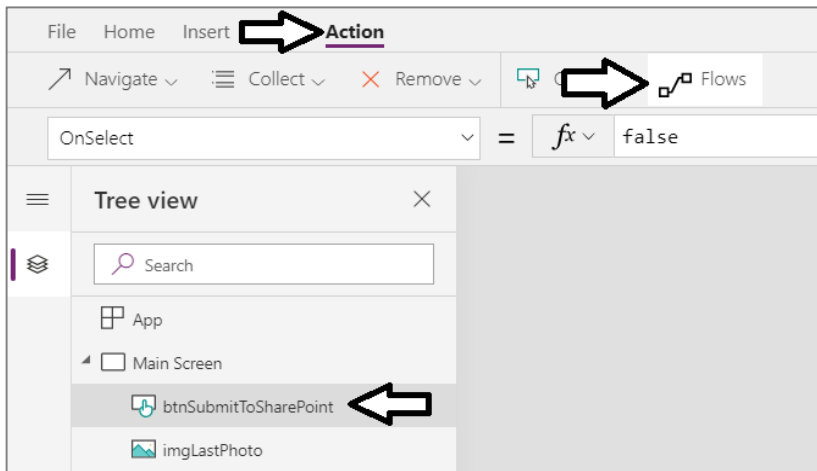
# Creating a Canvas App to Upload Photos

- Begin by creating a blank canvas app
  - Add a Camera control
  - Add an Image control to display camera's last photo
  - Add a button control to trigger a flow to upload photo



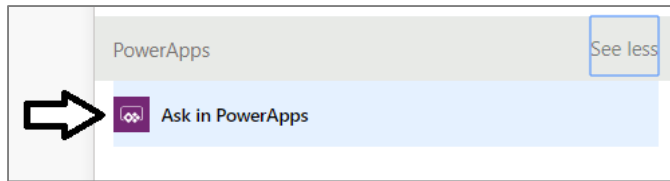
# Creating a Flow with a PowerApps Trigger

- Create a new flow from the Canvas App editor
  - Select the button which should trigger the flow
  - Click Flows button in Action tab to display the Data pane
  - Click the **Create a new flow** button
  - The new flow automatically created with PowerApps button trigger

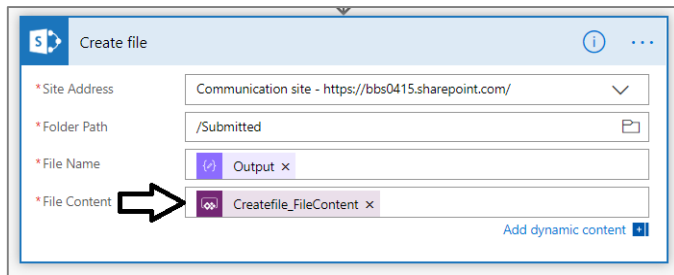


# Adding Parameters with Ask In PowerApps

- PowerApps trigger provides Ask in PowerApps parameter
  - Clicking **Ask in PowerApps** automatically creates new parameter



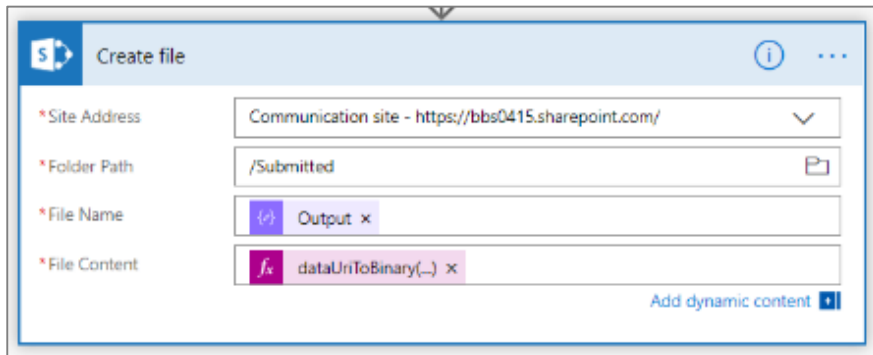
- New parameters used to pass data from canvas app to flow



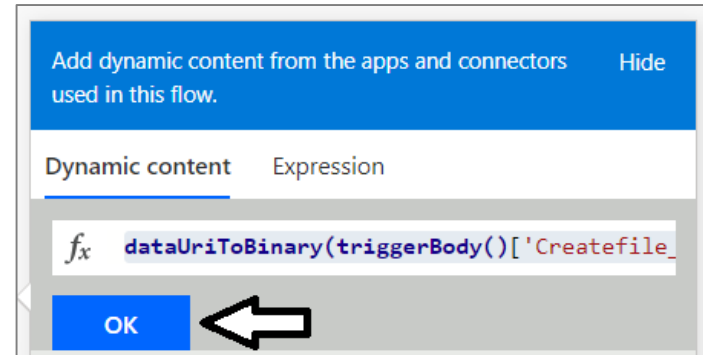


# Converting Photos to Binary Format

- Camera control image not compatible with SharePoint
  - Camera control photo image based on Data Uri format
  - SharePoint document library expects files in binary format
  - You must convert photos using **dataUriToBinary** function



The screenshot shows the 'Create file' dialog in SharePoint. It has four fields: 'Site Address' (Communication site - https://bbs0415.sharepoint.com/), 'Folder Path' (/Submitted), 'File Name' (Output), and 'File Content' (dataUriToBinary(...)). The 'File Content' field is highlighted with a purple background. An 'Add dynamic content' button is at the bottom right.



The screenshot shows the 'Add dynamic content' dialog. It has a blue header with the text 'Add dynamic content from the apps and connectors used in this flow.' and a 'Hide' button. Below the header is a table with two columns: 'Dynamic content' and 'Expression'. The 'Dynamic content' column is selected, showing the expression 'dataUriToBinary(triggerBody()['Createfile\_'])'. An 'OK' button is at the bottom left, with a white arrow pointing to it.





**DEMO**

## **Using a Flow to Upload a Photo to a SharePoint Document Library**

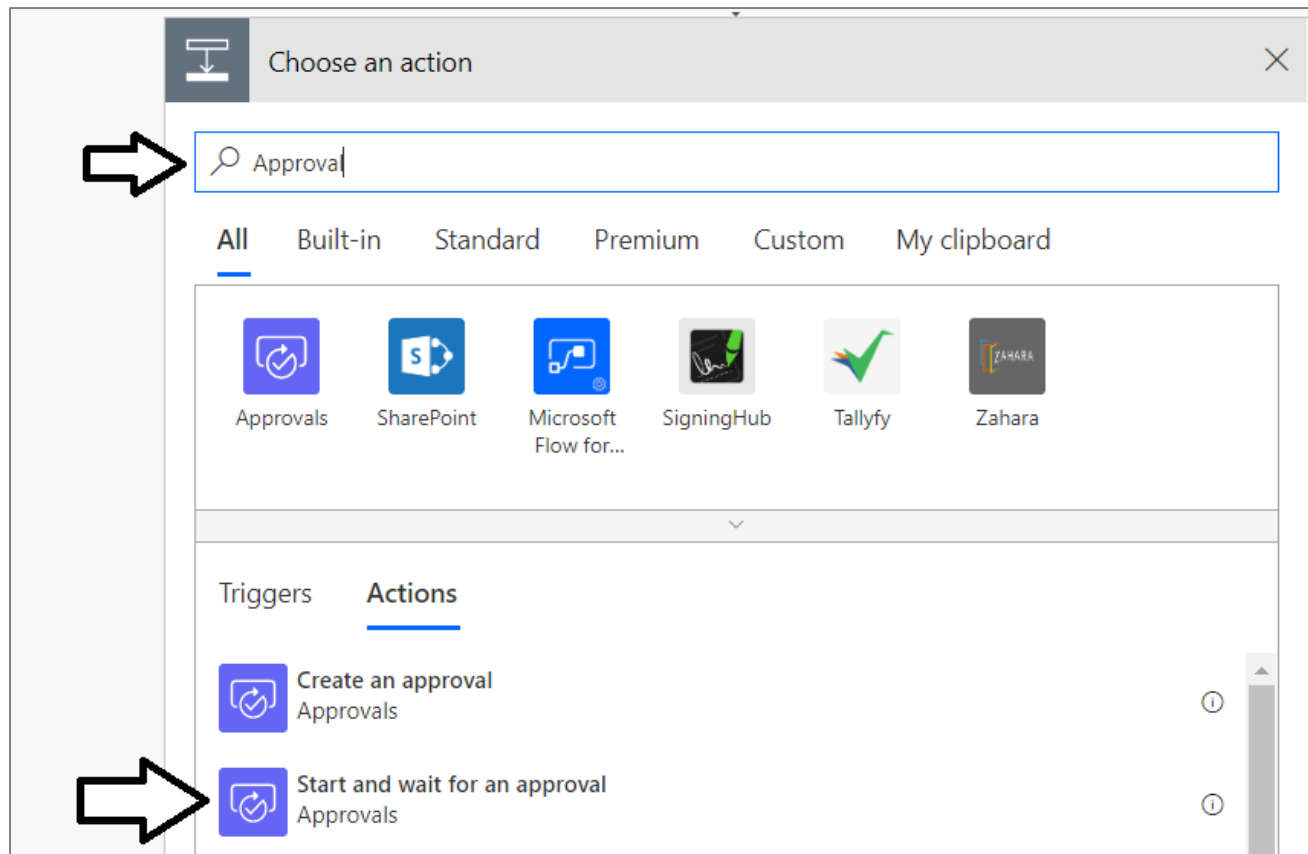
# Agenda

- ✓ Converting and Reshaping Data
- ✓ Uploading Photos to SharePoint
- Automating Approval Processes
  - Integrating Flow with Microsoft Forms
  - Handling Runtime Errors
  - Understanding Parallel Execution



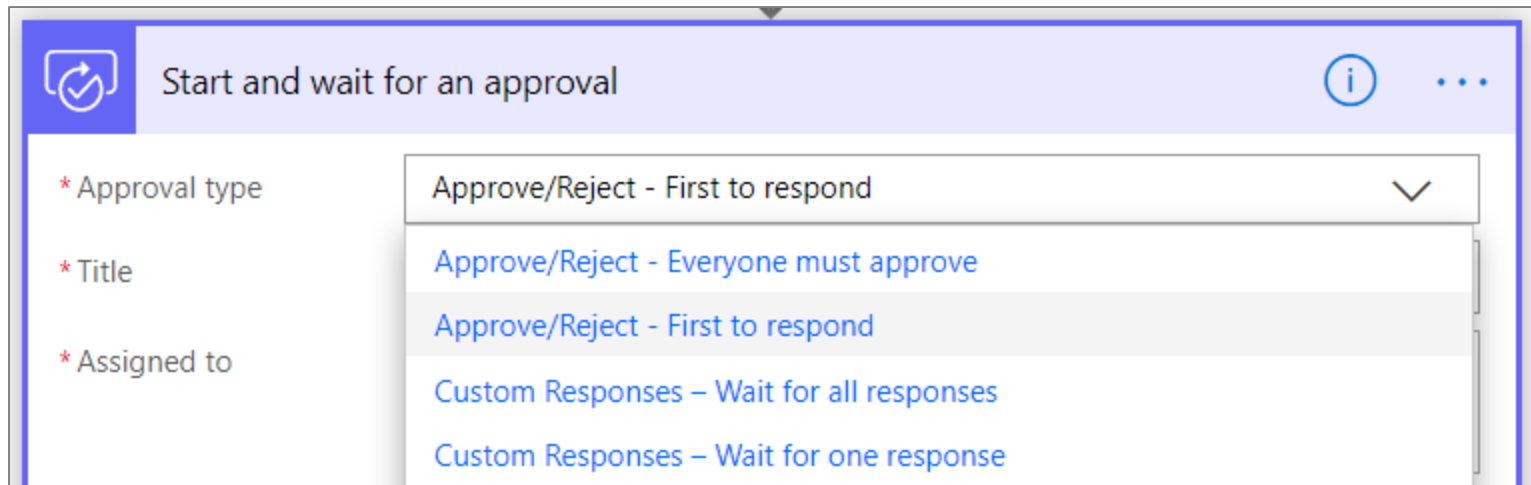
# Add the Start an Approval Action

- An **Approval** process is added as an action
  - Select the Approvals action named **Start an approval**



# Approval Types

- There are several types of approvals
  - Approve/Reject – Everyone must approve
  - Approve/Reject – First to respond
  - Custom Responses – Wait for all responses
  - Custom Responses – Wait for one response



The screenshot shows a dialog box titled "Start and wait for an approval" with a checkmark icon on the left and an information icon and a three-dot menu on the right. The dialog contains three fields: "Approval type", "Title", and "Assigned to". The "Approval type" field is currently set to "Approve/Reject - First to respond" and is open, showing a dropdown menu with five options: "Approve/Reject - Everyone must approve", "Approve/Reject - First to respond" (highlighted), "Custom Responses – Wait for all responses", and "Custom Responses – Wait for one response".

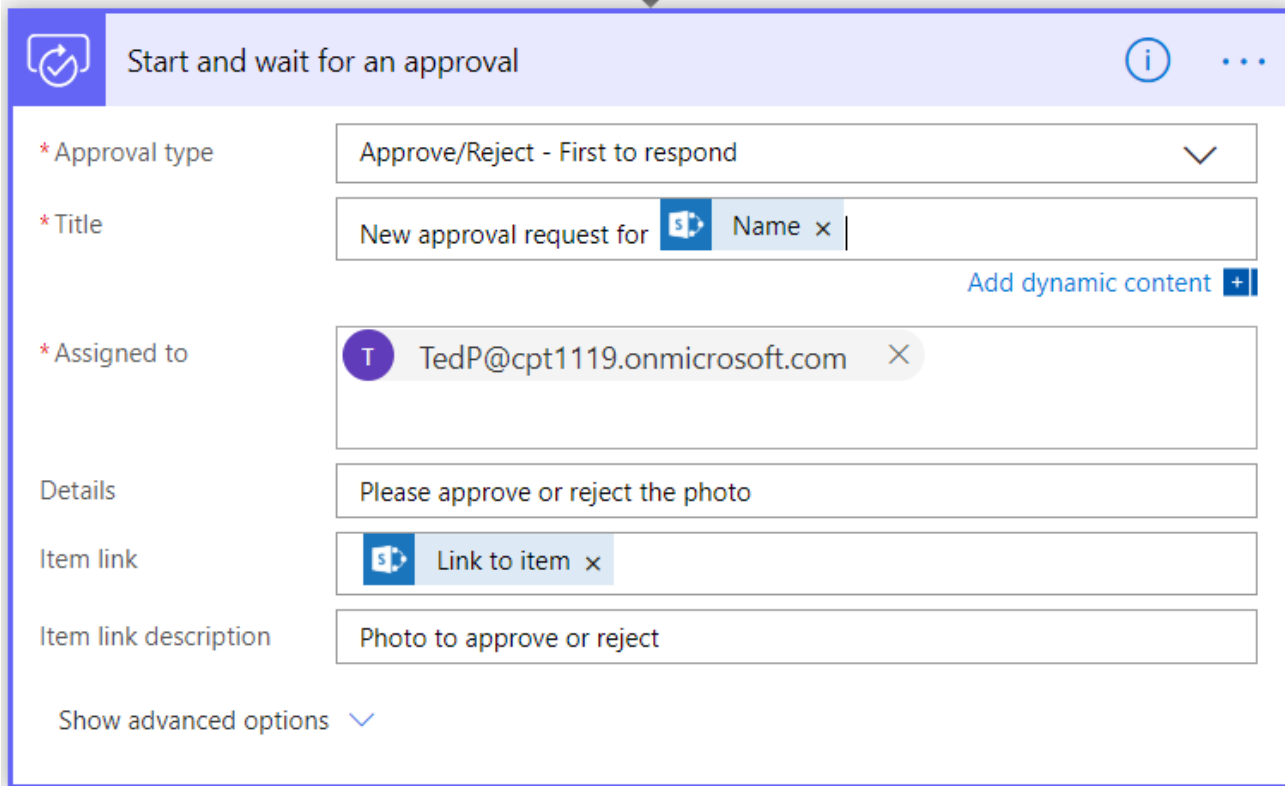
Field	Value
* Approval type	Approve/Reject - First to respond
* Title	
* Assigned to	





# Building Out The Start an Approval Action

- You provide data which is sent to approver



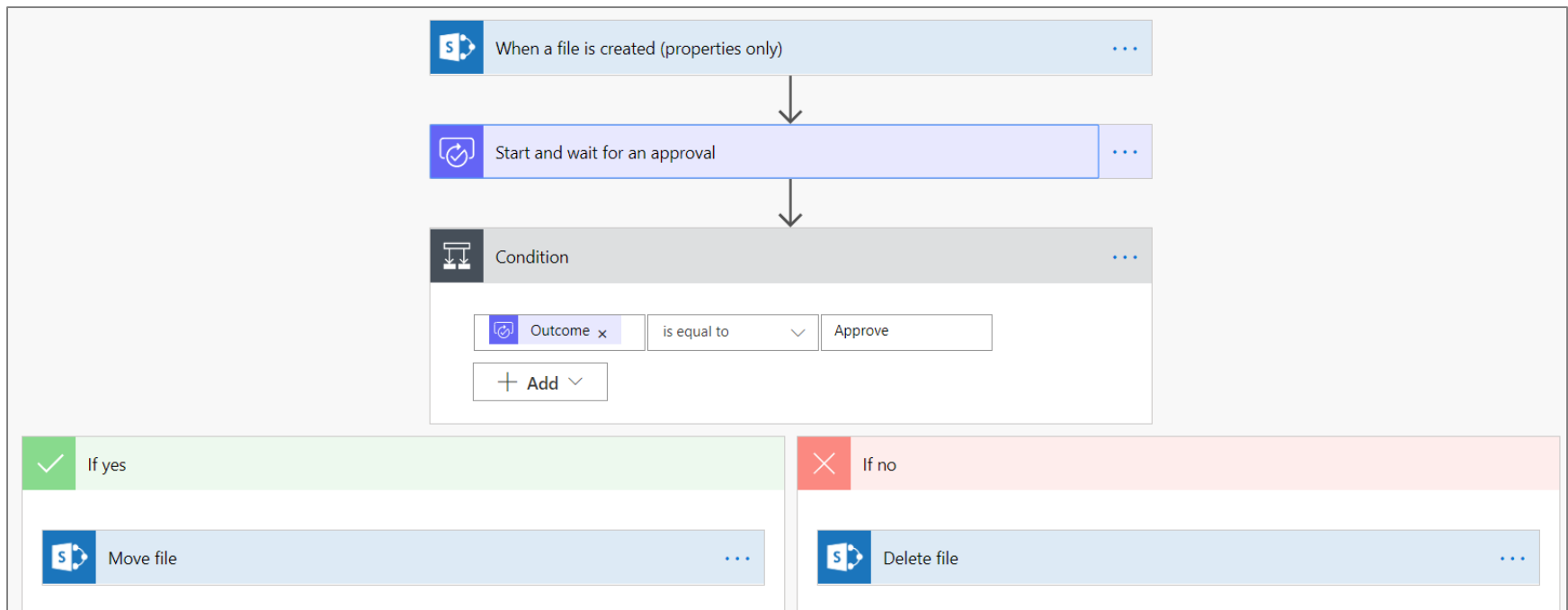
The screenshot shows a dialog box titled "Start and wait for an approval" with a checkmark icon and an information icon. The dialog contains several fields for configuring an approval request:

- \* Approval type:** A dropdown menu set to "Approve/Reject - First to respond".
- \* Title:** A text field containing "New approval request for" followed by a Teams icon, a blue box with "Name", and a close button (x). To the right is a link "Add dynamic content" with a plus icon.
- \* Assigned to:** A field showing a purple circle with "T" and the email "TedP@cpt1119.onmicrosoft.com" with a close button (x).
- Details:** A text field containing "Please approve or reject the photo".
- Item link:** A field showing a Teams icon, a blue box with "Link to item", and a close button (x).
- Item link description:** A text field containing "Photo to approve or reject".
- Show advanced options:** A link with a downward arrow.



# Responding to the Approval Response

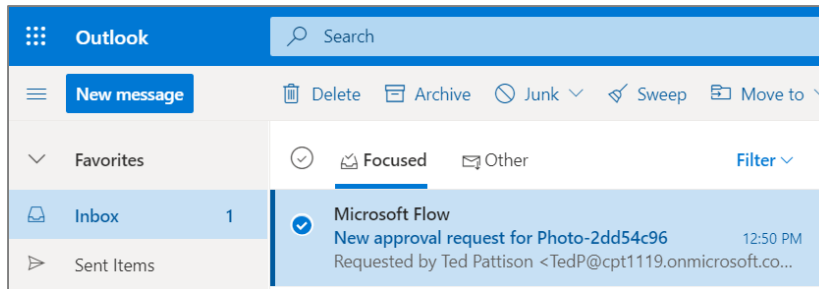
- Start an Approval action followed by a condition
  - Allows flow to determine if approval was accepted or rejected
  - Condition provides If yes and If no branches for both outcomes



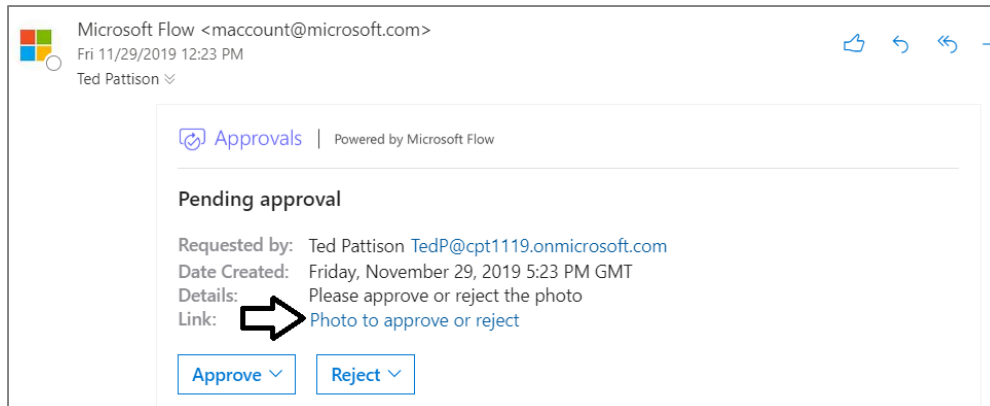


# Sending Email Notification to an Approver

- The flow sends notification email to the approver
  - Flow execution currently paused inside **Start an Approval** action

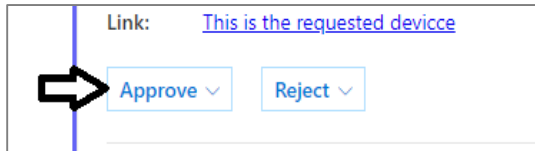


- Email allows approver to approve or reject approval request
- Email should also allow approver to see item for approval

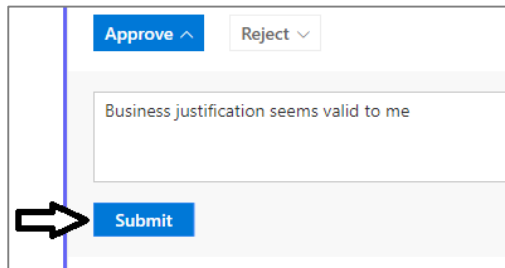


# Approving an Approval Request

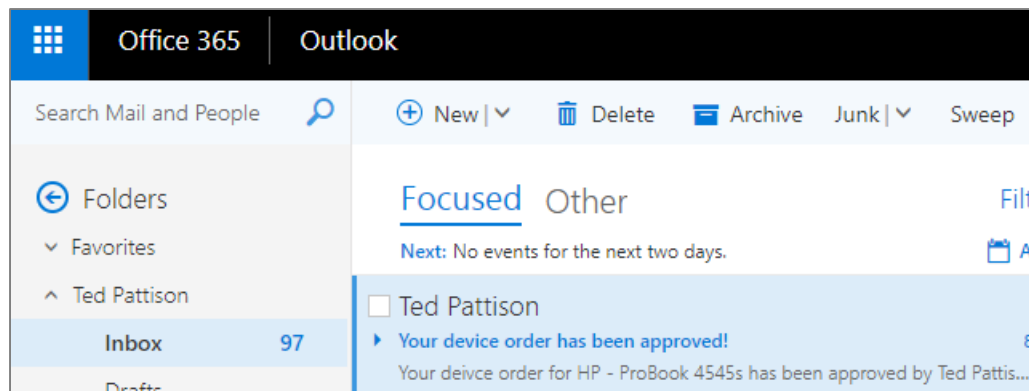
- Notification email provides button to approve or reject request



- Approver can enter comment and submit approval (or rejection)



- Approval or rejection unblocks flow which continue down appropriate branch
  - Approval response determines whether to send approval email or rejection email




# Rejecting an Approval Request


- In the case of a rejected request...

Details: A new device has been requested  
Toshiba - Portege Z935-ST4N02  
\$979.99  
Comments: I already have a great laptop but I want another one for no good reason.

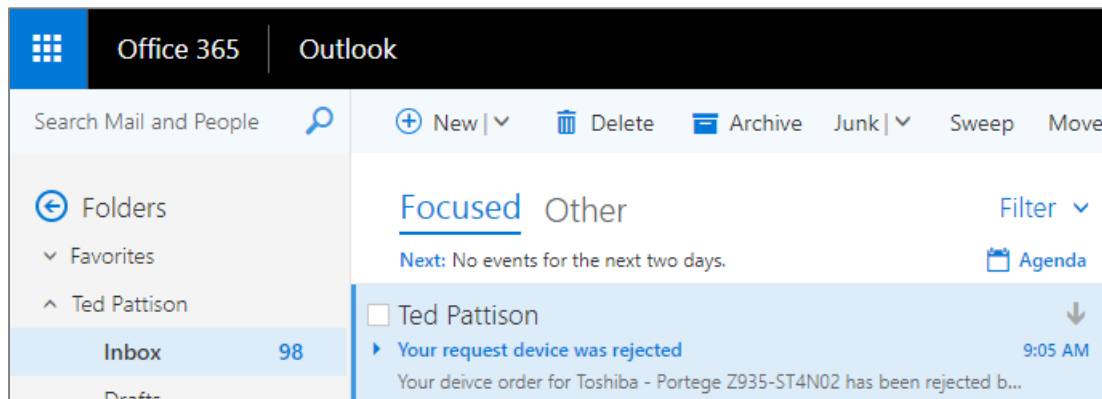
Link: [This is the requested device](#)

Approve  Reject ^

You only need one laptop!

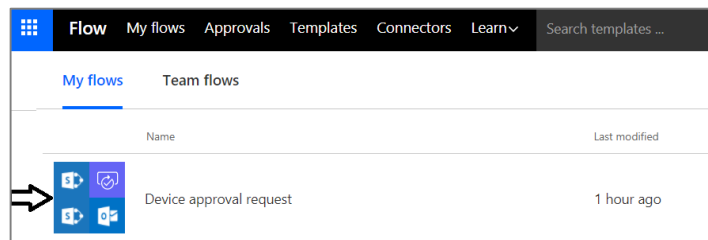
 Submit

- Approval flow sends a notification about rejection

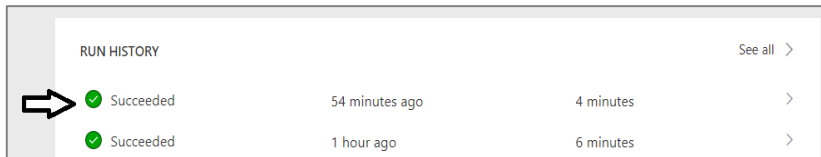


# Run History

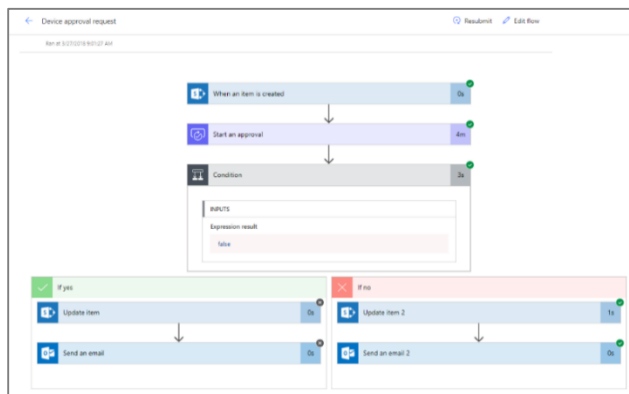
- Flow tracks run history of flow that have started
  - Click on a flow to see its RUN HISTORY list



- RUN HISTORY list has entry for each flow that has started

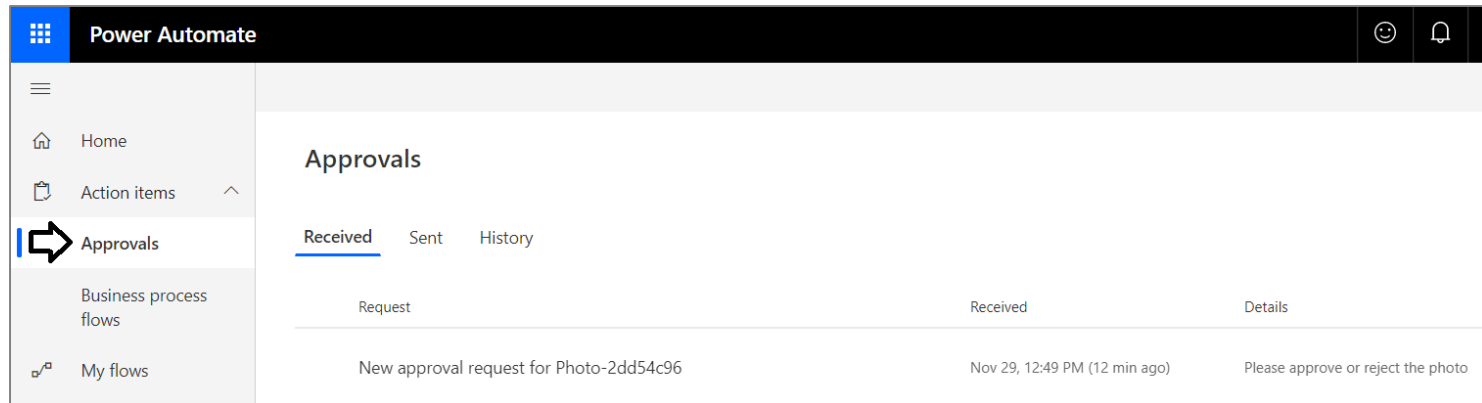


- Drilling into flow run history shows execution path and data



# Approvals Center

- Microsoft Flow provides Approvals Center
  - Provides alternative to email for approve/reject processing
  - Accessible through browser
  - Provides monitoring of completed approvals and pending approvals



# Examining Received Requests

- Approvers can see list of all their approval requests
- Approver can approve or reject approval request

Respond: Approve

Overview

Approval

New approval request for Photo-2dd54c96

Requester

Ted Pattison

Received

Nov 29, 12:49 PM (15 min ago)

Link

[Photo to approve or reject](#)

Details

Please approve or reject the photo

Approve

Add a comment (optional)

Great photo. Love it!

Confirm

Cancel



# Agenda

- ✓ Converting and Reshaping Data
- ✓ Uploading Photos to SharePoint
- ✓ Automating Approval Processes
- Integrating Flow with Microsoft Forms
  - Handling Runtime Errors
  - Understanding Parallel Execution







**DEMO**

## **Creating a Flow to Process Forms Created using Microsoft Forms**

# Agenda

- ✓ Converting and Reshaping Data
- ✓ Uploading Photos to SharePoint
- ✓ Automating Approval Processes
- ✓ Integrating Flow with Microsoft Forms
- Handling Runtime Errors
- Understanding Parallel Execution



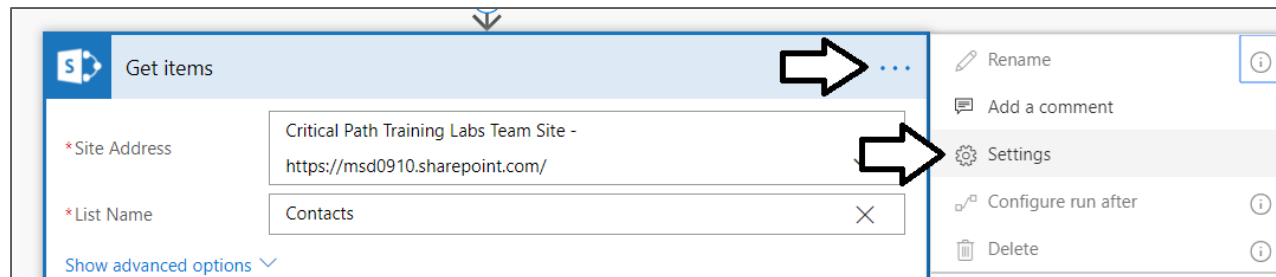
# Normal action execution

- Standard behavior of a flow
  - Action steps execute in sequential order
  - Flow terminates if error occurs (failure or timeout)
- After flow runs, every action left in 1 of 4 possible states



# Action Settings

- Settings let you configure
  - Async Actions
  - Timeouts
  - Retry Policy
  - Sequential Behavior
  - And more!







# Error Handling

- Select the **Run after** option from action menu
  - Choose which error conditions, the arrow will turn dotted red
  - Use parallels for errors that are not at end of flow
  - Retry policy by default handles transient failures
  - Recommended to select exponential as they last a long time

'Handle duplicate file names' should run after:

 This step will only run if the flow couldn't create the file because there's already another one with the same name. It will add some numbers to the end of the file name to make it unique.

 **Create file**  
Failed

☐ is successful  
☒ has failed  
☐ is skipped  
☐ has timed out

**Done** **Cancel**

**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 retry policy is to retry 4 times with a 20 second delay between each attempt.

Retry policy type:

Interval ⓘ:

Count:

**Done** **Cancel**



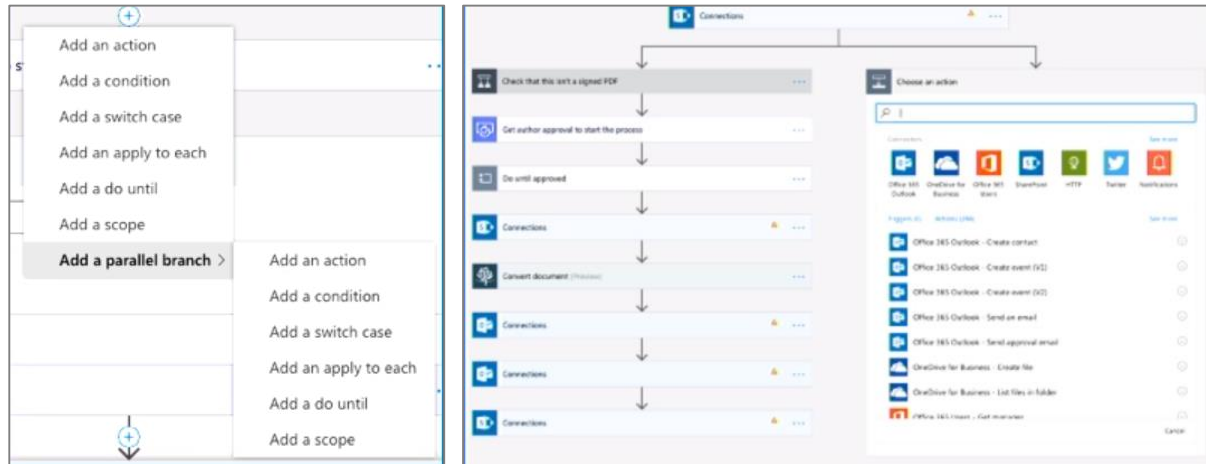
# Agenda

- ✓ Converting and Reshaping Data
- ✓ Uploading Photos to SharePoint
- ✓ Automating Approval Processes
- ✓ Integrating Flow with Microsoft Forms
- ✓ Handling Runtime Errors
- Understanding Parallel Execution

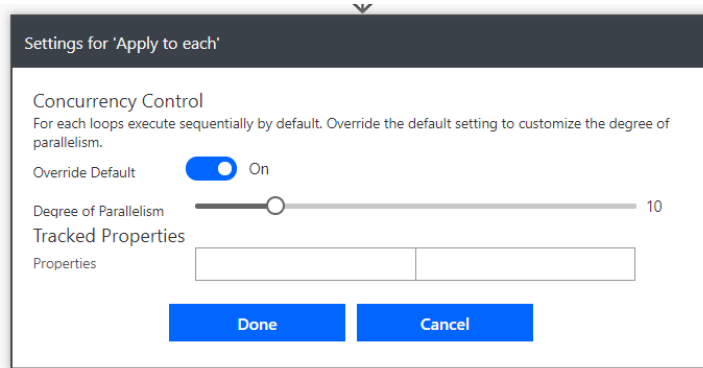


# Parallel Execution

- Add parallel branch from above using ⊕



- Apply to each is sequential by default
  - Adding parallel execute to Apply to each





# Summary

- ✓ Converting and Reshaping Data
- ✓ Uploading Photos to SharePoint
- ✓ Automating Approval Processes
- ✓ Integrating Flow with Microsoft Forms
- ✓ Handling Runtime Errors
- ✓ Understanding Parallel Execution

