

Contents

What is Logic App?	2
Insurance claims automation Logic app.....	4
Test Logic app	12

Logic App

What is Logic App?

Azure Logic Apps is a cloud-based platform for creating and running automated [workflows](#) that integrate your apps, data, services, and systems. With this platform, you can quickly develop highly scalable integration solutions for your enterprise and business-to-business (B2B) scenarios. As a member of [Azure Integration Services](#), Azure Logic Apps simplifies the way that you connect legacy, modern, and cutting-edge systems across cloud, on premises, and hybrid environments.

The following list describes just a few example tasks, business processes, and workloads that you can automate using the Azure Logic Apps service:

- Schedule and send email notifications using Office 365 when a specific event happens, for example, a new file is uploaded.
- Route and process customer orders across on-premises systems and cloud services.
- Move uploaded files from an SFTP or FTP server to Azure Storage.
- Monitor tweets, analyze the sentiment, and create alerts or tasks for items that need review.

For more information about the ways workflows can access and work with apps, data, services, and systems, review the following documentation:

- [Connectors for Azure Logic Apps](#)
- [Managed connectors for Azure Logic Apps](#)
- [Built-in triggers and actions for Azure Logic Apps](#)
- [B2B enterprise integration solutions with Azure Logic Apps](#)

Key Terms used in the hackathon :

Logic app

A *logic app* is the Azure resource you create when you want to develop a workflow.

Workflow

A *workflow* is a series of steps that defines a task or process. Each workflow starts with a single trigger, after which you must add one or more actions

Trigger

A *trigger* is always the first step in any workflow and specifies the condition for running any further steps in that workflow. For example, a trigger event might be getting an email in your inbox or detecting a new file in a storage account.

Action

A *trigger* is always the first step in any workflow and specifies the condition for running any further steps in that workflow. For example, a trigger event might be getting an email in your inbox or detecting a new file in a storage account.

Built-in operations

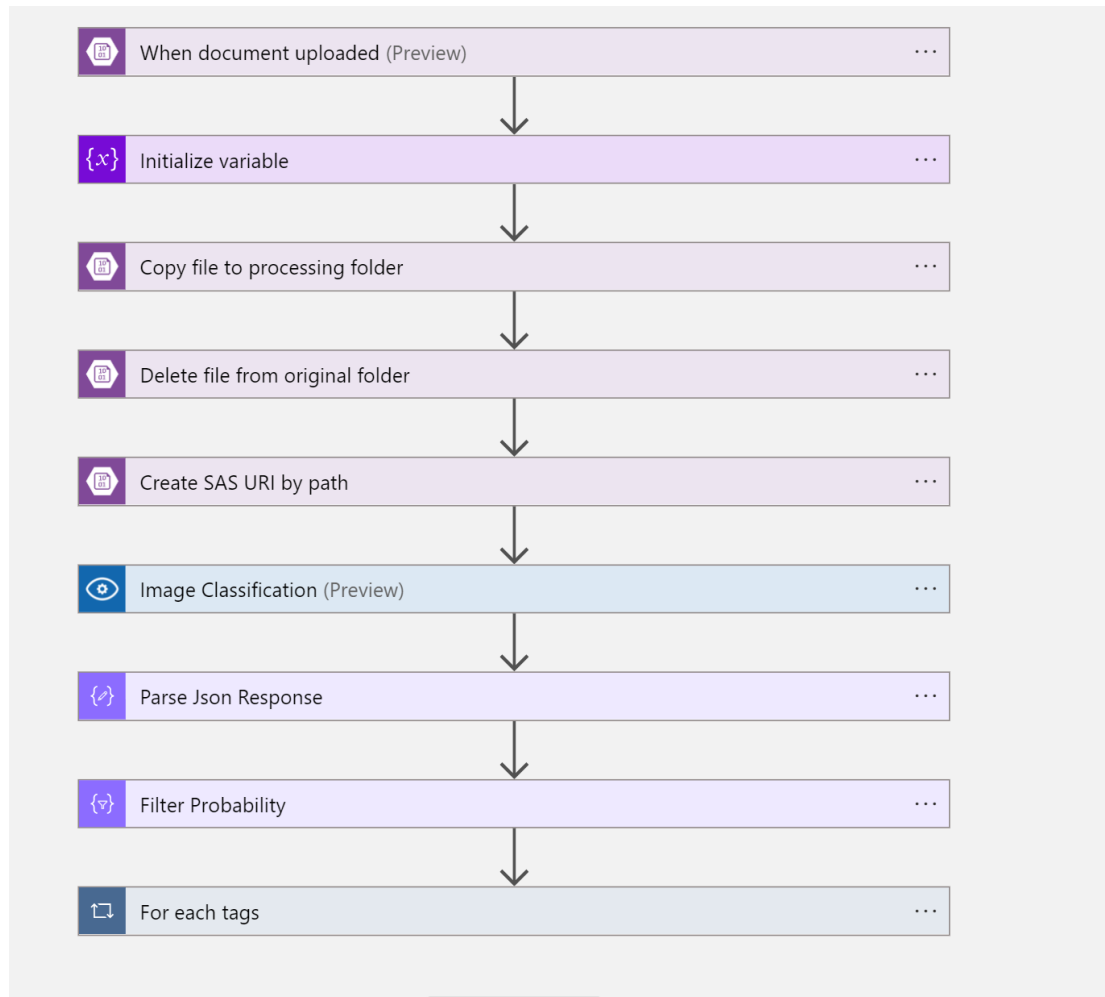
A *built-in* trigger or action is an operation that runs natively in Azure Logic Apps. For example, built-in operations provide ways for you to control your workflow's schedule or structure, run your own code, manage and manipulate data, send or receive requests to an endpoint, and complete other tasks in your workflow.

How Logic app works

In a logic app, each workflow always starts with a single trigger. A trigger fires when a condition is met, for example, when a specific event happens or when data meets specific criteria. Many triggers include scheduling capabilities that control how often your workflow runs. Following the trigger, one or more actions run operations that, for example, process, handle, or convert data that travels through the workflow, or that advance the workflow to the next step.

Insurance claims automation Logic app

Logic app workflow to automate the insurance claims is already deployed in your resource group. Open the Logic app that is deployed and the designer should look like below:



Following is the high-level process flow for the workflow:

1. Trigger the logic app when a new document is uploaded into the storage account (/upload container)

When document uploaded (Preview)

* Container: /upload

Number of blobs to return: 1

How often do you want to check for items?
 5 Second

Add new parameter

Connected to azureblob. [Change connection.](#)

2. Initialize the ClaimId from the received blob image. Note we are using custom functions capability within logic app

Initialize variable

* Name: claimId

* Type: String

Value: `substring(...)`

Add dynamic content

Dynamic content Expression
`substring(triggerBody()?['DisplayName'],0,`

Update

Copy file to processing folder

3. Copy the file that is uploaded into processing container to process the document

Copy file to processing folder

* Source url: List of Files Path

* Destination blob path: /processing/ List of Files Name

Overwrite? Yes

Connected to azureblob. [Change connection.](#)

4. To avoid duplication, delete the original file from the /upload container

Delete file from original folder

* Blob: List of Files Id

Connected to azureblob. [Change connection.](#)

5. Dynamically create the SAS URI so that we can use that to call our custom vision model.

Create SAS URI by path

* Blob path: Path

Group Policy Identifier: The string identifying a stored access policy. The Group policy parameters (i

Permissions: Read,Write,List

Add new parameter

Connected to azureblob. [Change connection.](#)

- Use the inbuilt connector to call custom vision model. **Note: Ensure you change the ProjectId to match to what you captured from "Custom Vision" section. Also do notice the published name is "latest" which is what we called our model when training custom model. Image URL is the SAS URI path that was created in Step 5.**

Image Classification (Preview)

* Project ID: 5b4c6533-835e-4248-8ff9-6393d327694a

* Published Name: latest

* Image URL: Web Url

Connected to cognitiveservicescustomvision. [Change connection.](#)

- Because we are getting the JSON response from custom vision, use the capability within Logic app to parse the JSON (from object predictions)

Parse Json Response

* Content: Predictions

* Schema

```
{
  "items": {
    "properties": {
      "probability": {
        "type": "number"
      },
      "tagId": {
        "type": "string"
      }
    }
  }
}
```

[Use sample payload to generate schema](#)

- Let's filter the response so that the probability is > 0.85

Filter Probability

*From

Body x

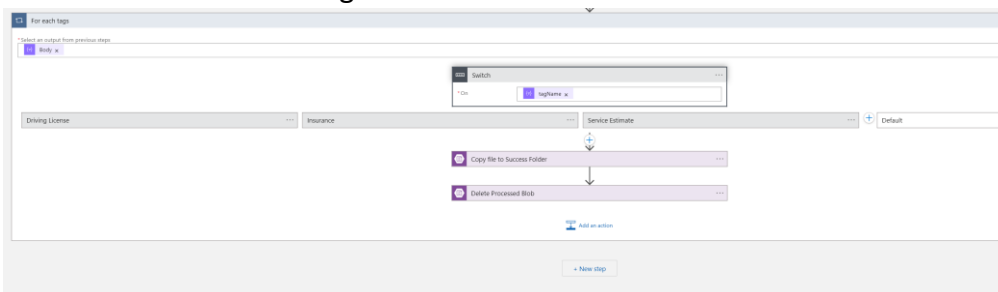
probability x

is greater than

0.85

[Edit in advanced mode](#)

9. For each (ideally there should be only one with probability > 0.85) tag, execute the conditional statement. Conditional statement will (based on the document type) will call either the out of the box model (ID or invoice) or the custom model (insurance) that we created in “Form Recognizer” Section



10. For out of the box model, we call Analyze method of the form recognizer model, passing on the URL

Driving License

*Equals

Driving License

Analyze Driving License (Preview)

Document/Image File Content: A PDF document or image (JPG or PNG) file to analyze.

Document/Image URL: Web Url x

Include Text Details: Yes

Add new parameter

Connected to formrecognizer. [Change connection.](#)

For each Id

[Add an action](#)

Service Estimate ...

* Equals


Service Estimate

Analyze Service Estimate (Preview) i ...

Document/Image File Content

A PDF document or image (JPG or PNG) file to analyze.

Document/Image URL

 Web Url x

Include Text Details

Yes ✓ ✕


Add new parameter

▼

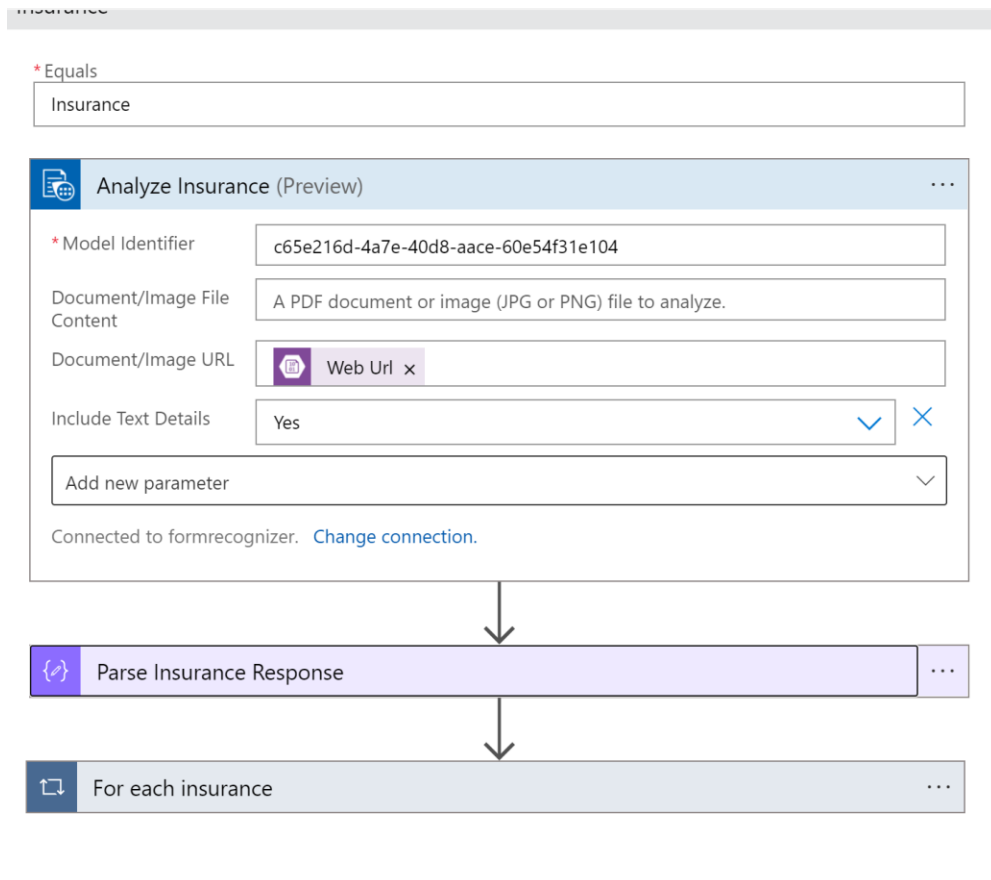
Connected to formrecognizer. [Change connection.](#)

+ ↓

↻ For each Invoice ...

 [Add an action](#)

11. For custom model (insurance card), we are calling Analyze method on the custom model API. Since it's custom model, we need the "ModelId" that we captured in "Form Recognizer" Section. Copy the Form Recognizer modelId



12. Once the output from the form recognizer is parsed, we call CosmosDb connector to insert the data in CosmosDb collection

For each insurance

...

*Select an output from previous steps

{}

Body x

Create or Update Insurance (Preview)

① ...

*Database ID

fsihack

▼

*Collection ID

claims

▼

*Document

```

{
  "Agency": "",
  "Agency Address": "{ } text x",
  "Company": "{ } text x",
  "Effective Date": "{ } text x",
  "Expiration Date": "{ } text x",
  "FileName": "{ } List of Files Name x",
  "Insured": "{ } text x",
  "Insured Address": "{ } text x",
  "Make": "{ } text x",
  "Model": "{ } text x",
  "PolicyNumber": "{ } text x",
  "State": "{ } fields.State.text x",
  "VIN": "{ } text x",
  "Year": "{ } text x",
  "claimId": "{x} claimId x",
  "formtype": "Insurance",
  "id": "{fx} guid() x"
}

```


Add new parameter

▼


Connected to fsihack.com

Change connection


13. After successfully processing the workflow, copy the file to "Success" container and delete the file from "Processing" container




Copy file to Success Folder




* Storage account name

Use connection settings (od478734sa) 


* Source url




Path 



* Destination blob path

/succeeded/




Name 

Overwrite?



Yes  

Connected to azureblob. [Change connection.](#)







Delete Processed Blob


 


* Storage account name

Use connection settings (od478734sa) 

* Blob



Path 

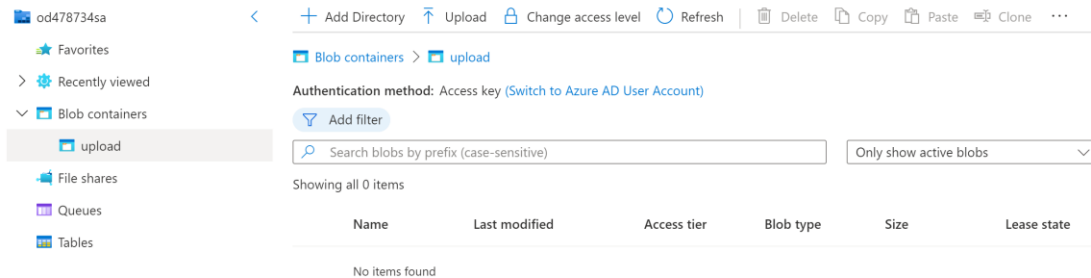


Connected to azureblob. [Change connection.](#)

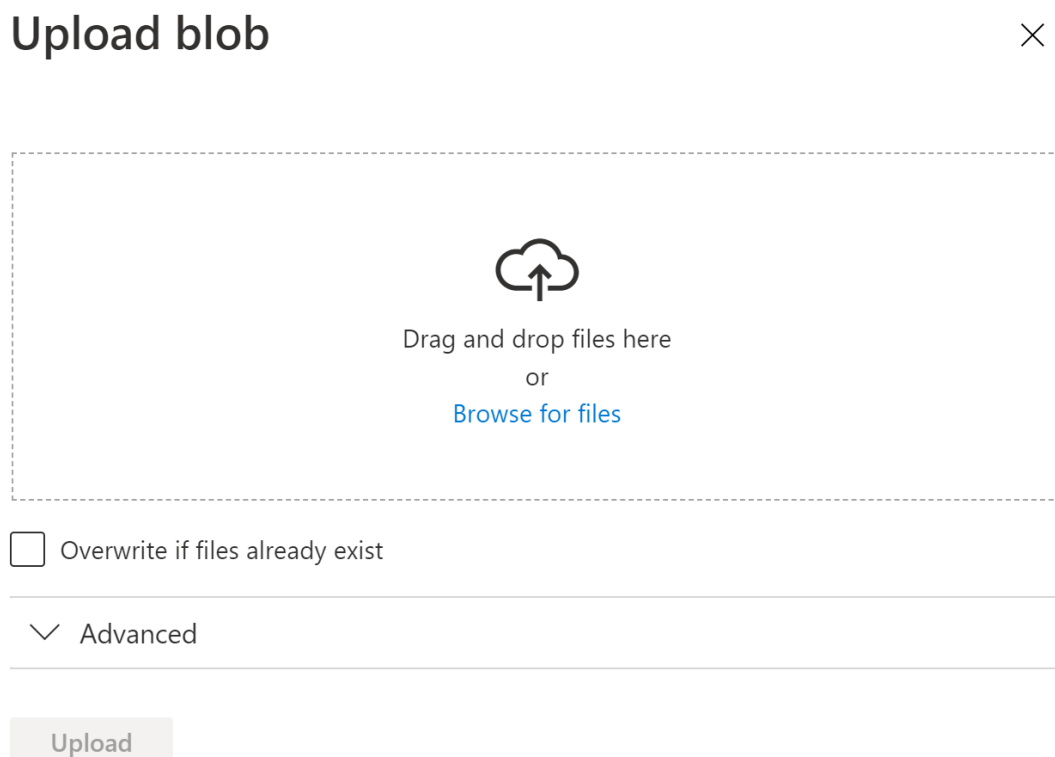
Test Logic app

Since our logic trigger is when the document is uploaded into storage account, we can test our logic app to invoke that trigger.

1. Go to Azure Portal -> Storage Account -> Storage Browser (Preview)
2. Click on Blob Container -> Upload



3. Click on Upload -> Browse for Files



4. Select one of the document type (Insurance, Service Estimate, ID)
5. If you go back to Logic app and view the run history, you will see trigger executed the workflow

Run Trigger Refresh Edit Delete Disable Update Schema Clone Open in mobile Export Feedback

Introducing the new portable Logic Apps runtime that supports local development and debugging. Click to learn more. →

Essentials [JSON View](#)

Resource group (Move) : FSIUC3-478734 Definition : 1 trigger, 21 actions
 Location : East US Status : Enabled
 Subscription (Move) : Shared Sub - 156 Runs last 24 hours : 0 successful, 0 failed
 Subscription ID : a96b8f1a-f683-4f26-918c-f083e93873fc Integration Account : -- --

Get started **Runs history** Trigger history Metrics

All Start time earlier than Pick a date Pick a time

Specify the run identifier to open monitor view directly

Status	Start time	Identifier	Duration	Static Results
✓ Succeeded	10/30/2021, 7:12 PM	08585659677412681125007492663CU45	14.42 Seconds	

6. Clicking on the workflow, it will show you the run history for both triggers and all actions

Microsoft Azure Search resources, services, and docs (G+/i) odl_user_478734@clou... CLOUD EVENTS

Home > od478734lapp > Runs history >

Runs history od478734lapp

Refresh

All Start time earlier than Pick a date Pick a time Search to filter items by identifier

Start time	Duration
✓ 10/30/2021, 7:12 PM	14.42 Seconds
✓ 10/28/2021, 9:53 PM	12.51 Seconds
✗ 10/28/2021, 9:48 PM	1.15 Seconds
✗ 10/28/2021, 9:42 PM	1 Second
✓ 10/28/2021, 9:33 PM	9.1 Seconds
✓ 10/28/2021, 9:30 PM	2.86 Seconds
✗ 10/28/2021, 9:23 PM	3.96 Seconds
✓ 10/28/2021, 9:22 PM	10.41 Seconds
✓ 10/28/2021, 9:16 PM	10.91 Seconds
✗ 10/28/2021, 9:12 PM	9.41 Seconds
✗ 10/28/2021, 9:09 PM	8.74 Seconds
✗ 10/28/2021, 9:05 PM	1.16 Seconds
✗ 10/28/2021, 9:04 PM	4.76 Seconds
✗ 10/28/2021, 9:00 PM	1.58 Seconds

Logic app run 08585659677412681125007492663CU45

Run Details Resubmit Cancel Run Refresh Info

```

graph TD
    A[When document uploaded 0s] --> B[Initialize variable 0s]
    B --> C[Copy file to processing folder 0s]
    C --> D[Delete file from original folder 0s]
    D --> E[Create SAS URI by path 0s]
    E --> F[Image Classification 2s]
    F --> G[Parse Json Response 0s]
    G --> H[Filter Probability 0s]
  
```

7. Feel free to click on each action to view the “Input”, “Output” and additional details and understand the workflow execution and logic.

Image Classification

2s

INPUTS

Show raw inputs >

Project ID

5b4c6533-835e-4248-8ff9-6393d327694a

Published Name

latest

Image URL

https://od478734sa.blob.core.windows.net/processing/Alaska_Insuranc

OUTPUTS

Show raw outputs >

Id

7303fdae-b338-4140-a86a-7900fd0bbe1b

Project

5b4c6533-835e-4248-8ff9-6393d327694a

Iteration

ed2c6c1d-3d8c-4330-b642-32f977478de6

Created

2021-10-31T00:12:26.962Z

Predictions

[
 {
 "probability": 0.97981524,
 "tagId": "bcf30cff-113e-41c7-ae08-a423d24d7679",
 "tagName": "Insurance"
 },
]

8. You can also go to cosmosdb collection to view the data that was inserted.