

Table of Contents

Overview

[Logic App overview](#)

[Connectors list](#)

[Enterprise Integration Pack](#)

Get started

[Create a logic app](#)

[Use logic app features](#)

[Templates overview](#)

[Create an app using a template](#)

[Examples and scenarios](#)

[Service Bus scenario](#)

[B2B processing](#)

[XML processing](#)

How To

Build

[Create a connector](#)

[Use File System connector](#)

[Use SAP connector](#)

[Visual Studio tools](#)

[Using Azure functions](#)

[Logic apps as callable endpoints](#)

[Loops, scopes, and debatching](#)

[Using your custom API](#)

[Exception handling](#)

[Exception handling scenario](#)

[Handling content types](#)

[Workflow definition](#)

[Troubleshooting tips](#)

[Limits and config](#)

Use Enterprise Integration Pack (EIP)

Integration accounts overview

Agreements

B2B processing

XML processing

Flat file processing

Add XSLT maps

Transform XML

Validate XML

Using certificates

Add partners

Add schemas

Store metadata

AS2 integration

AS2 encoding

AS2 decoding

EDIFACT integration

EDIFACT encoding

EDIFACT decoding

X12 integration

X12 encoding

X12 decoding

Use Gateway

Connect on-premises

Install

Automate

Automation template

Publish from VS

Manage

Use Visual Studio Cloud Explorer

Secure your logic app

Monitor logic apps

[Monitor B2B messages](#)

[Integration accounts](#)

[Pricing](#)

[Reference](#)

[PowerShell](#)

[Schema History](#)

[GA](#)

[Preview](#)

[REST](#)

[Resources](#)

[Pricing](#)

[MSDN forum](#)

[Stack Overflow](#)

[Service updates](#)

What are Logic Apps?

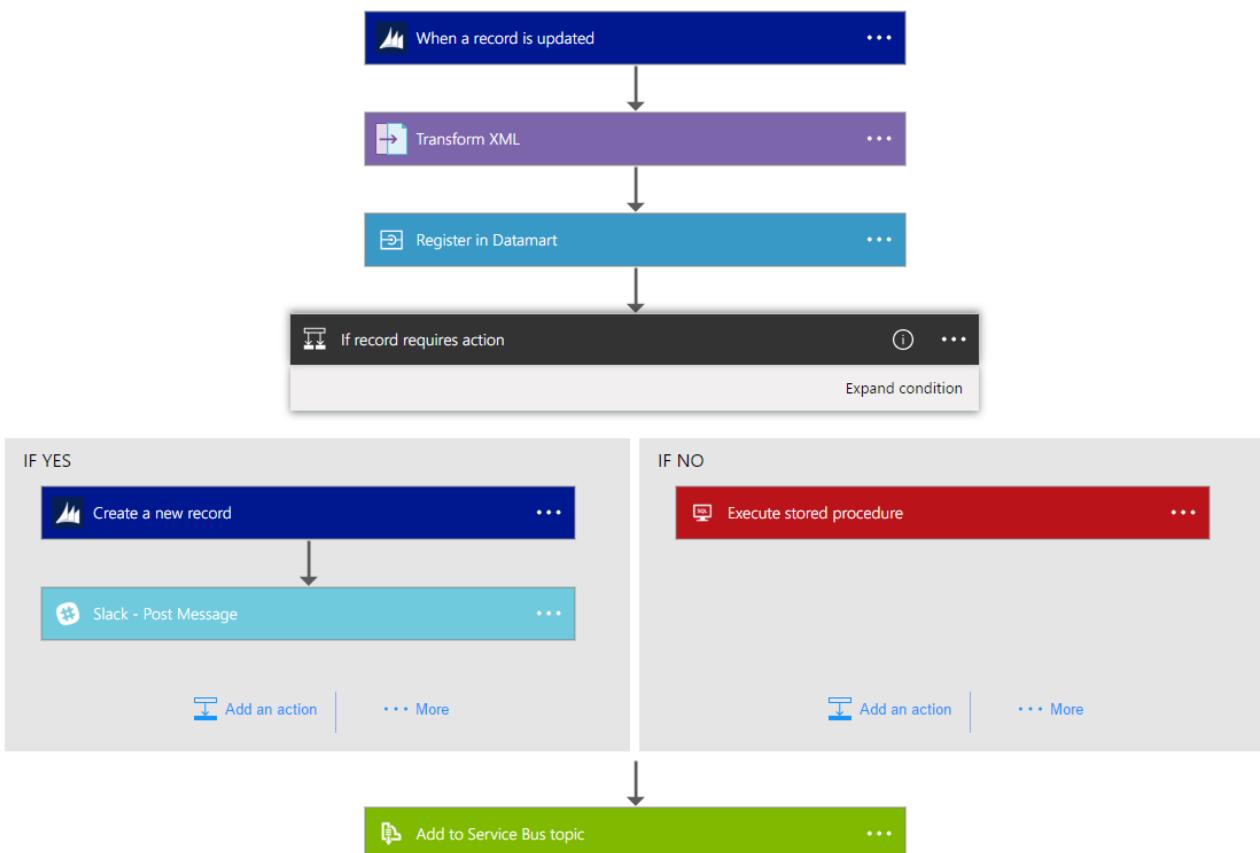
1/25/2017 • 4 min to read • [Edit on GitHub](#)

Logic Apps provide a way to simplify and implement scalable integrations and workflows in the cloud. It provides a visual designer to model and automate your process as a series of steps known as a workflow. There are [many connectors](#) across the cloud and on-premises to quickly integrate across services and protocols. A logic app begins with a trigger (like 'When an account is added to Dynamics CRM') and after firing can begin many combinations actions, conversions, and condition logic.

The advantages of using Logic Apps include the following:

- Saving time by designing complex processes using easy to understand design tools
- Implementing patterns and workflows seamlessly, that would otherwise be difficult to implement in code
- Getting started quickly from templates
- Customizing your logic app with your own custom APIs, code, and actions
- Connect and synchronise disparate systems across on-premises and the cloud
- Build off of BizTalk server, API Management, Azure Functions, and Azure Service Bus with first-class integration support

Logic Apps is a fully managed iPaaS (Integration Platform as a Service) allowing developers not to have to worry about building hosting, scalability, availability and management. Logic Apps will scale up automatically to meet demand.



As mentioned, with Logic Apps, you can automate business processes. Here are a couple examples:

- Move files uploaded to an FTP server into Azure Storage
- Process and route orders across on-premises and cloud systems

- Monitor all tweets about a certain topic, analyze the sentiment, and create alerts and tasks for items needing followup.

Scenarios such as these can be configured all from the visual designer and without writing a single line of code. Get started [building your logic app now](#). Once written - a logic app can be [quickly deployed and reconfigured](#) across multiple environments and regions.

Why Logic Apps?

Logic Apps brings speed and scalability into the enterprise integration space. The ease of use of the designer, variety of available triggers and actions, and powerful management tools make centralizing your APIs simpler than ever. As businesses move towards digitalization, Logic Apps allows you to connect legacy and cutting-edge systems together.

Additionally, with our [Enterprise Integration Account](#) you can scale to mature integration scenarios with the power of a [XML messaging](#), [trading partner management](#), and more.

- **Easy to use design tools** - Logic Apps can be designed end-to-end in the browser or with Visual Studio tools. Start with a trigger - from a simple schedule to when a GitHub issue is created. Then orchestrate any number of actions using the rich gallery of connectors.
- **Connect APIs easily** - Even composition tasks that are easy to describe are difficult to implement in code. Logic Apps makes it easy to connect disparate systems. Want to connect your cloud marketing solution to your on-premises billing system? Want to centralize messaging across APIs and systems with an Enterprise Service Bus? Logic apps are the fastest, most reliable way to deliver solutions to these problems.
- **Get started quickly from templates** - To help you get started we've provided a [gallery of templates](#) that allow you to rapidly create some common solutions. From advanced B2B solutions to simple SaaS connectivity, and even a few that are just 'for fun' - the gallery is the fastest way to get started with the power of Logic Apps.
- **Extensibility baked-in** - Don't see the connector you need? Logic Apps is designed to work with your own APIs and code; you can easily create your own API app to use as a custom connector, or call into an [Azure Function](#) to execute snippets of code on-demand.
- **Real integration horsepower** - Start easy and grow as you need. Logic Apps can easily leverage the power of BizTalk, Microsoft's industry leading integration solution to enable integration professionals to build the solutions they need. Find out more about the [Enterprise Integration Pack](#).

Logic App Concepts

The following are some of the key pieces that comprise the Logic Apps experience.

- **Workflow** - Logic Apps provides a graphical way to model your business processes as a series of steps or a workflow.
- **Managed Connectors** - Your logic apps need access to data and services. Managed connectors are created specifically to aid you when you are connecting to and working with your data. See the list of connectors available now in [managed connectors](#).
- **Triggers** - Some Managed Connectors can also act as a trigger. A trigger starts a new instance of a workflow based on a specific event, like the arrival of an e-mail or a change in your Azure Storage account.
- **Actions** - Each step after the trigger in a workflow is called an action. Each action typically maps to an operation on your managed connector or custom API apps.
- **Enterprise Integration Pack** - For more advanced integration scenarios, Logic Apps includes capabilities from BizTalk. BizTalk is Microsoft's industry leading integration platform. The Enterprise Integration Pack connectors allow you to easily include validation, transformation, and more in to your Logic App workflows.

Getting Started

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- [You can automate business processes with Logic Apps](#)
- [Learn How to Integrate your systems with Logic Apps](#)

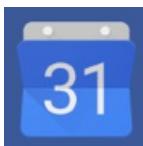
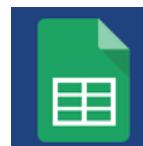
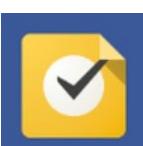
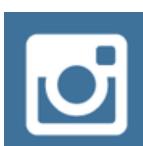
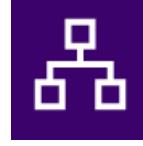
List of connectors

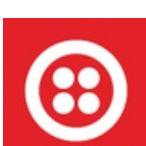
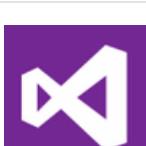
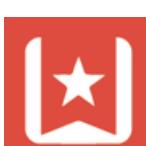
1/20/2017 • 2 min to read • [Edit on GitHub](#)

Select a connector to learn how to build workflows quickly.

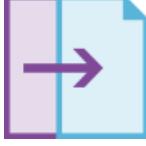
Standard connectors

CONNECTORS			
 API/Web App	 appFigures	 Asana	 Azure DocumentDB
 Azure ML	 Azure Functions	 Azure Blob Storage	 Basecamp 3
 Bitly	 BizTalk Server	 Blogger	 Box
 Campfire	 Cognitive Services Text Analytics	 Common Data Service	 DB2
 Delay	 Dropbox	 Dynamics 365	 Dynamics 365 for Financials
 Dynamics 365 for Operations	 Easy Redmine	 Facebook	 FTP

CONNECTORS			
 GitHub	 Google Calendar	 Google Drive	 Google Sheets
 Google Tasks	 HideKey	 HipChat	 HTTP
 HTTP Swagger	 HTTP Request	 HTTP Response	 Informix
 Insightly	 Instagram	 Instapaper	 JIRA
 MailChimp	 Mandrill	 Microsoft Translator	 Nested Logic App
 Office 365 Outlook	 Office 365 Users	 Office 365 Video	 OneDrive
 Outlook.com	 PagerDuty	 Pinterest	 Project Online
 Query	 Recurrence	 Redmine	 RSS

CONNECTORS			
			
Salesforce	SendGrid	Service Bus	SFTP
			
SharePoint	Slack	Smartsheet	SMTP
			
Todoist	Trello	Twilio	Twitter
			
Vimeo	Visual Studio Team Services	Webhook	WordPress
			
Wunderlist	Yammer	YouTube	File System

Integration account connectors

INTEGRATION ACCOUNT CONNECTORS			
			
XML validation	XML transform	Flat file encode	Flat file decode
			
AS2 decode	AS2 encode	X12 decode	X12 encode

INTEGRATION ACCOUNT CONNECTORS



**EDIFACT
decode**



**EDIFACT
encode**

NOTE

If you want to get started with Azure Logic Apps before signing up for an Azure account, go to [Try Logic App](#). You can immediately create a short-lived starter Logic app in App Service. No credit cards required; no commitments.

Enterprise connectors

Use the enterprise connectors to create Logic apps for B2B scenarios that include EAI and EDI.

ENTERPRISE CONNECTORS



MQ



Connectors can be triggers

Several connectors provide triggers that can notify your app when specific events occur. For example, the FTP connector has the OnUpdatedFile trigger. You can build either a Logic app, PowerApp or Flow that listens to this trigger and takes an action whenever the trigger is fired.

There are two types of triggers:

- Poll Triggers: These triggers poll your service at a specified frequency to check for new data. When new data is available, a new instance of your app runs with the data as input. To prevent the same data from being consumed multiple times, the trigger may clean up data that has been read and passed to your app.
- Push Triggers: These triggers listen for data on an endpoint or for an event to occur, then, triggers a new instance of your app. The Twitter connector is one such example.

Connectors can be actions

Connectors can also be used as actions within your apps. Actions are useful for looking up data which can then be used in the execution of your app. For example, you may need to look up customer data from a SQL database when processing an order. Or, you may need to write, update or delete data in a destination table. You can do this using the actions provided by the connectors. Actions map to operations that are defined in the Swagger metadata.

Next Steps

- [Build a logic app now](#)
- [Create a custom connector](#)
- [Monitor your logic apps](#)

Overview of the Enterprise Integration Pack

1/20/2017 • 2 min to read • [Edit on GitHub](#)

What is the Enterprise Integration Pack?

The Enterprise Integration Pack is Microsoft's cloud-based solution for seamlessly enabling business-to-business (B2B) communications. The pack uses industry standard protocols including [AS2](#), [X12](#), and [EDIFACT](#) to exchange messages between business partners. Messages can be optionally secured using both encryption and digital signatures.

The pack allows organizations that use different protocols and formats to exchange messages electronically by transforming the different formats into a format that both organizations' systems can interpret and take action on.

If you are familiar with BizTalk Server or Microsoft Azure BizTalk Services, you'll find it easy to use the Enterprise Integration features because most of the concepts are similar. One major difference is that Enterprise Integration uses integration accounts to simplify the storage and management of artifacts used in B2B communications.

Architecturally, the Enterprise Integration Pack is based on **integration accounts** that store all the artifacts that can be used to design, deploy, and maintain your B2B apps. An integration account is basically a cloud-based container where you store artifacts such as schemas, partners, certificates, maps, and agreements. These artifacts can then be used in Logic apps to build B2B workflows. Before you can use the artifacts in a Logic app, you need to link your integration account to your Logic app. After linking them, your Logic app will have access to the integration account's artifacts.

Why should you use enterprise integration?

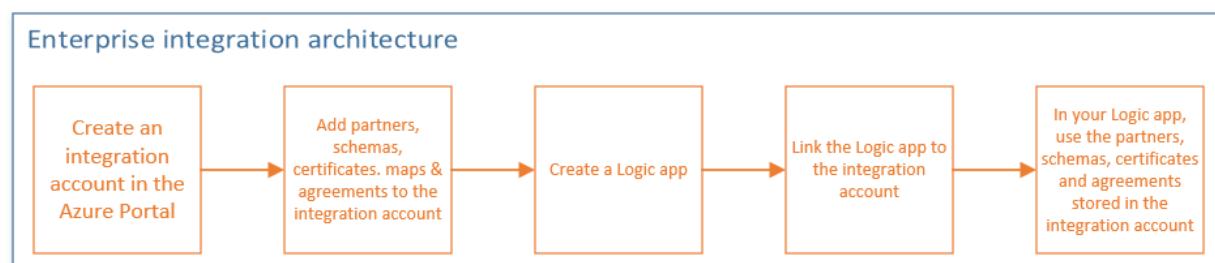
- With enterprise integration, you are able to store all your artifacts in one place, which is your integration account.
- You can leverage the Logic apps engine and all its connectors to build B2B workflows and integrate with 3rd party SaaS applications, on-premises apps as well as custom applications
- You can also leverage Azure functions

How to get started with enterprise integration?

You can build and manage B2B apps using the Enterprise Integration Pack via the Logic apps designer on the [Azure portal](#).

You can also use [PowerShell](#) to manage your Logic apps.

Here is an overview of the steps you need to take before you can create apps in the Azure portal:



What are some common scenarios?

Enterprise Integration supports these industry standards:

- EDI - Electronic Data Interchange
- EAI - Enterprise Application Integration

Here's what you need to get started

- An Azure subscription with an integration account
- Visual Studio 2015 to create maps and schemas
- [Microsoft Azure Logic Apps Enterprise Integration Tools for Visual Studio 2015 2.0](#)

Try it

[Try it now](#) to deploy a fully operational sample AS2 send & receive logic app that uses the B2B features of Logic Apps.

Learn more about:

- [Agreements](#)
- [Business to Business \(B2B\) scenarios](#)
- [Certificates](#)
- [Flat file encoding/decoding](#)
- [Integration accounts](#)
- [Maps](#)
- [Partners](#)
- [Schemas](#)
- [XML message validation](#)
- [XML transform](#)
- [Enterprise Integration Connectors](#)
- [Integration Account Metadata](#)
- [Monitor B2B messages](#)
- [Tracking B2B messages in OMS portal](#)

Create a new logic app connecting SaaS services

1/25/2017 • 2 min to read • [Edit on GitHub](#)

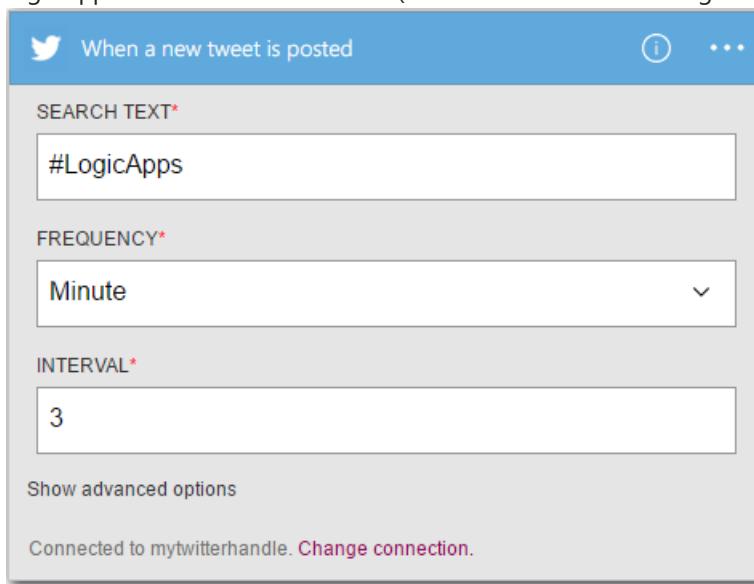
This topic demonstrates how, in just a few minutes, you can get started with [Azure Logic Apps](#). We'll walk through a simple workflow that lets you send interesting tweets to your email.

To use this scenario, you need:

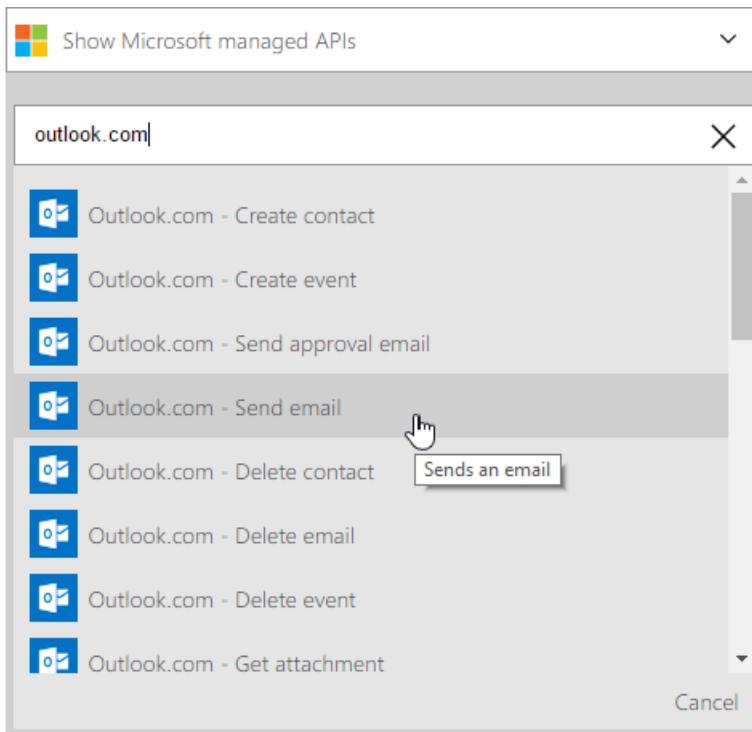
- An Azure subscription
- A Twitter account
- A Outlook.com or hosted Office 365 mailbox

Create a new logic app to email you tweets

1. On the [Azure portal dashboard](#), select **New**.
2. In the search bar, search for 'logic app', and then select **Logic App**. You can also select **New, Web + Mobile**, and select **Logic App**.
3. Enter a name for your logic app, select a location, resource group, and select **Create**. If you select **Pin to Dashboard** the logic app will automatically open once deployed.
4. After opening your logic app for the first time you can select from a template to start. For now click **Blank Logic App** to build this from scratch.
5. The first item you need to create is the trigger. This is the event that will start your logic app. Search for **twitter** in the trigger search box, and select it.
6. Now you'll type in a search term to trigger on. The **Frequency** and **Interval** will determine how often your logic app will check for new tweets (and return all tweets during that time span).



7. Select the **New step** button, and then choose **Add an action** or **Add a condition**
8. When you select **Add an Action**, you can search from the [available connectors](#) to choose an action. For example, you can select **Outlook.com - Send Email** to send mail from an outlook.com address:



9. Now you have to fill out the parameters for the email you want:

10. Finally, you can select **Save** to make your logic app live.

Manage your logic app after creation

Now your logic app is up and running. It will periodically check for tweets with the search term entered. When it finds a matching tweet, it will send you an email. Finally, you'll see how to disable the app, or see how it's doing.

1. Go to the [Azure Portal](#)
2. Click **Browse** on the left side of the screen and select **Logic Apps**.
3. Click the new logic app that you just created to see current status and general information.
4. To edit your new logic app, click **Edit**.
5. To turn off the app, click **Disable** in the command bar.
6. View run and trigger histories to monitor when your logic app is running. You can click **Refresh** to see the latest data.

In less than 5 minutes you were able to set up a simple logic app running in the cloud. To learn more about using Logic Apps features, see [Use logic app features](#). To learn about the Logic App definitions themselves, see

author Logic App definitions.

Use Logic Apps features

1/20/2017 • 4 min to read • [Edit on GitHub](#)

In the [previous topic](#), you created your first logic app. Now we will show you how to build a more complete process using App Services Logic Apps. This topic introduces the following new Logic Apps concepts:

- Conditional logic, which executes an action only when a certain condition is met.
- Code view to edit an existing logic app.
- Options for starting a workflow.

Before you complete this topic, you should complete the steps in [Create a new logic app](#). In the [Azure portal](#), browse to your logic app and click **Triggers and Actions** in the summary to edit the logic app definition.

Reference material

You may find the following documents useful:

- [Management and runtime REST APIs](#) - including how to invoke Logic apps directly
- [Language reference](#) - a comprehensive list of all supported functions/expressions
- [Trigger and action types](#) - the different types of actions and the inputs they take
- [Overview of App Service](#) - description of what components to choose when to build a solution

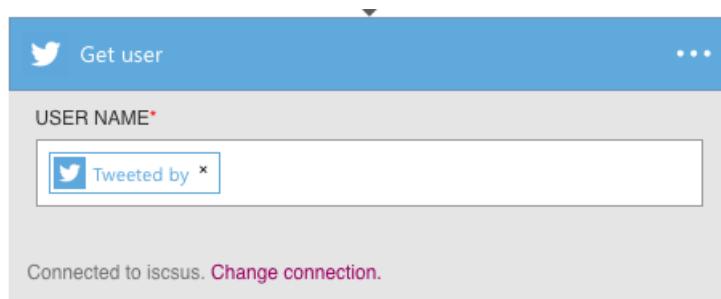
Adding conditional logic

Although the original flow works, there are some areas that could be improved.

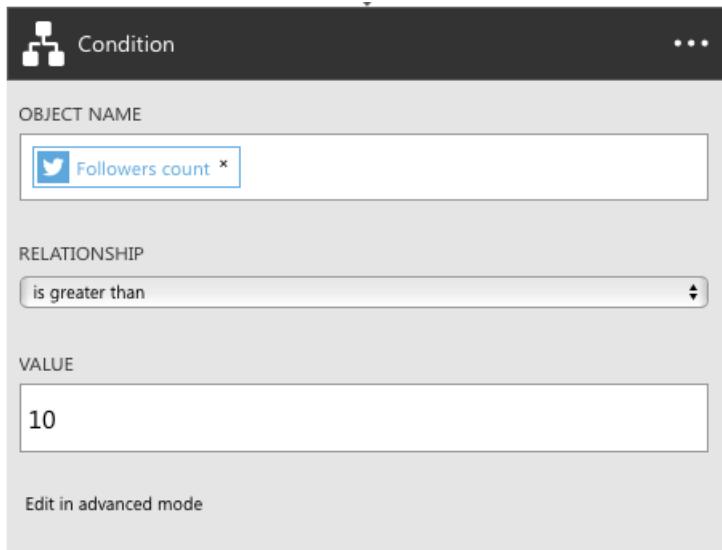
Conditional

This logic app may result in you getting a lot of emails. The following steps add logic to make sure that you only receive an email when the tweet comes from someone with a certain number of followers.

1. Click the plus and find the action *Get User* for Twitter.
2. Pass in the **Tweeted by** field from the trigger to get the information about the Twitter user.



3. Click the plus again, but this time select **Add Condition**
4. In the first box, click the ... underneath **Get User** to find the **Followers count** field.
5. In the dropdown, select **Greater than**
6. In the second box type the number of followers you want users to have.



7. Finally, drag-and-drop the email box into the **If Yes** box. This will mean you'll only get emails when the follower count is met.

Repeating over a list with forEach

The `forEach` loop specifies an array to repeat an action over. If it is not an array the flow fails. As an example, if you have `action1` that outputs an array of messages, and you want to send each message, you can include this `forEach` statement in the properties of your action: `forEach : "@action('action1').outputs.messages"`

Using the code view to edit a Logic App

In addition to the designer, you can directly edit the code that defines a logic app, as follows.

1. Click on the **Code view** button in the command bar.

This opens a full editor that shows the definition you just edited.

The screenshot shows the Microsoft Logic Apps Designer interface. On the left, there's a sidebar with icons for HOME, NOTIFICATIONS (1 unread), BROWSE, ACTIVE, BILLING, and NEW. The main area is titled "Triggers and actions" and shows the JSON definition of a logic app named "TWITTERSEARCH2". The code view contains several sections: "triggers", "actions", and "conditions". The "actions" section includes a "twitterconnector" action with parameters like "host", "operation", "parameters", and "count". Another "twitterconnector" action is shown below it, which repeats the first one. The "parameters" object is located within the "actions" section.

```

17     },
18   },
19   "triggers": {
20     "recurrence": {
21       "recurrence": {
22         "frequency": "Minute",
23         "interval": 1
24       },
25       "type": "Recurrence",
26       "name": "recurrence"
27     }
28   },
29   "actions": {
30     "twitterconnector": {
31       "type": "ApiApp",
32       "version": "2015-01-14",
33       "inputs": {
34         "host": {
35           "id": "/subscriptions/423db32d-4f58-4220-961c-b59f14c962f1/resourcegroups/bpmdemo002/providers/Microsoft.AppService/apiapps/twitterconnector002",
36           "gateway": "https://bpmdemo002proxysite.azurewebsites.net"
37         },
38         "operation": "SearchTweets",
39         "parameters": {
40           "query": "@concat('#',parameters('queryString'))",
41           "count": 3
42         }
43       },
44       "conditions": [],
45       "name": "twitterconnector"
46     },
47     "twitterconnector0": {
48       "repeat": "@actions('twitterconnector').outputs.body",
49       "type": "ApiApp",
50       "version": "2015-01-14",
51       "inputs": {
52         "host": {
53           "id": "/subscriptions/423db32d-4f58-4220-961c-b59f14c962f1/resourcegroups/bpmdemo002/providers/Microsoft.AppService/apiapps/twitterconnector002",
54           "gateway": "https://bpmdemo002proxysite.azurewebsites.net"
55         },
56         "operation": "SearchUser",
57         "parameters": {
58           "user_id": "",
59           "screen_name": "@repeatItem().Tweeted_By"
60         },
61         "authentication": {
62           "type": "Raw",
63           "scheme": "Zumo",
64           "parameter": "@parameters('/subscriptions/423db32d-4f58-4220-961c-b59f14c962f1/resourcegroups/bpmdemo002/providers/Microsoft.AppService/apiapps/twitterco
65         }
66       },
67       "conditions": [

```

By using the text editor, you can copy and paste any number of actions within the same logic app or between logic apps. You can also easily add or remove entire sections from the definition, and you can also share definitions with others.

2. After you make your changes in code view, simply click **Save**.

Parameters

There are some capabilities of Logic Apps that can only be used in the code view. One example of these is parameters. Parameters make it easy to re-use values throughout your logic app. For example, if you have an email address that you want use in several actions, you should define it as a parameter.

The following updates your existing logic app to use parameters for the query term.

1. In the code view, locate the `parameters : {}` object and insert the following topic object:

```

"topic" : {
  "type" : "string",
  "defaultValue" : "MicrosoftAzure"
}

```

2. Scroll to the `twitterconnector` action, locate the `query` value, and replace it with `#@{parameters('topic')}`. You could also use the `concat` function to join together two or more strings, for example:
`@concat('#',parameters('topic'))` is identical to the above.

Parameters are a good way to pull out values that you are likely to change a lot. They are especially useful when you need to override parameters in different environments. For more information on how to override parameters based on environment, see our [REST API documentation](#).

Now, when you click **Save**, every hour you get any new tweets that have more than 5 retweets delivered to a folder called **tweets** in your Dropbox.

To learn more about Logic App definitions, see [author Logic App definitions](#).

Starting a logic app workflow

There are several different options for starting the workflow defined in your logic app. A workflow can always be started on-demand in the [Azure portal](#).

Recurrence triggers

A recurrence trigger runs at an interval that you specify. When the trigger has conditional logic, the trigger determines whether or not the workflow needs to run. A trigger indicates it should run by returning a `200` status code. When it does not need to run, it returns a `202` status code.

Callback using REST APIs

Services can call a logic app endpoint to start a workflow. See [Logic apps as callable endpoints](#) for more information. To start that kind of logic app on-demand, click the **Run now** button on the command bar.

Logic App templates

1/25/2017 • 3 min to read • [Edit on GitHub](#)

What are logic app templates

A logic app template is a pre-built logic app that you can use to quickly get started creating your own workflow.

These templates are a good way to discover various patterns that can be built using logic apps. You can either use these templates as-is or modify them to fit your scenario.

Overview of available templates

There are many available templates currently published in the logic app platform. Some example categories, as well as the type of connectors used in them, are listed below.

Enterprise cloud templates

Templates that integrate Dynamics CRM, Salesforce, Box, Azure Blob, and other connectors for your enterprise cloud needs. Some examples of what can be done with these templates include organizing your leads and backing up your corporate file data.

Enterprise integration pack templates

Configurations of VETER (validate, extract, transform, enrich, route) pipelines, receiving an X12 EDI document over AS2 and transforming it to XML, as well as X12 and AS2 message handling.

Protocol pattern templates

These templates consist of logic apps that contain protocol patterns such as request-response over HTTP as well as integrations across FTP and SFTP. Use these as they exist, or as a basis for creating more complex protocol patterns.

Personal productivity templates

Patterns to help improve personal productivity include templates that set daily reminders, turn important work items into to-do lists, and automate lengthy tasks down to a single user approval step.

Consumer cloud templates

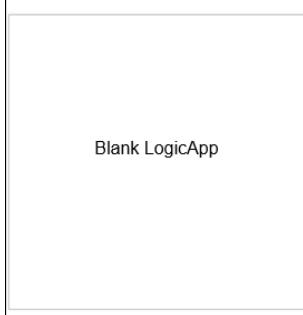
Simple templates that integrate with social media services such as Twitter, Slack, and email, ultimately capable of strengthening social media marketing initiatives. These also include templates such as cloudy copying, which can help increase productivity by saving time spent on traditionally repetitive tasks.

How to create a logic app using a template

To get started using a logic app template, go into the logic app designer. If you're entering the designer by opening an existing logic app, the logic app automatically loads in your designer view. However, if you're creating a new logic app, you see the screen below.

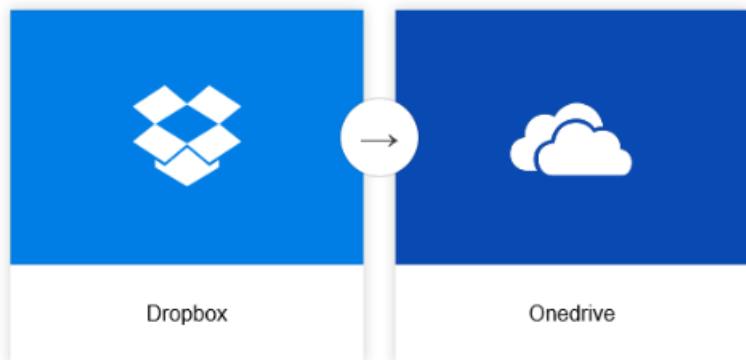
Templates

Choose a template below to create your Logic App.

 Blank LogicApp	 When a new file is created in Dropbox, copy it to OneDrive	 If you approve a new file in SharePoint, move it to a different folder	 Send me an email when a new item is added to a SharePoint Online list
 Send me an email when a new file is added in SharePoint Online	 Save my email attachments to a SharePoint document library	 Post to Slack if a new tweet matches with some hashtag	 Share my Tweets on Facebook

From this screen, you can either choose to start with a blank logic app or a pre-built template. If you select one of the templates, you are provided with additional information. In this example, we use the *When a new file is created in Dropbox, copy it to OneDrive* template.

When a new file is created in Dropbox, copy it to OneDrive



Dropbox Onedrive

Make sure your files end up in both Dropbox and OneDrive. This template will copy all new files that are created in Dropbox to a specific folder in OneDrive.

[Use this template](#)

If you choose to use the template, just select the *use this template* button. You'll be asked to sign in to your accounts based on which connectors the template utilizes. Or, if you've previously established a connection with these connectors, you can select *continue* as seen below.

To use this template:

 Dropbox [Switch account](#)

 OneDrive [Switch account](#)

[Continue](#)

After establishing the connection and selecting *continue*, the logic app opens in designer view.

The screenshot shows the Logic Apps Designer interface with two logic app definitions stacked vertically. The top logic app is triggered by 'When a file is created' and has the following configuration:

- FOLDER***: Select a folder
- FREQUENCY***: Minute
- INTERVAL***: 3
- Show advanced options
- Connected to [redacted] Change connection.

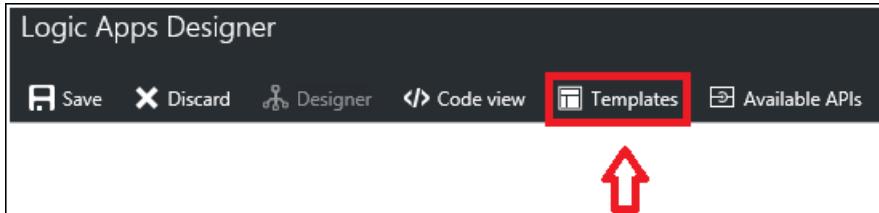
The bottom logic app is triggered by 'Create file' and has the following configuration:

- FOLDER PATH***: /backup from Dropbox
- FILE NAME***: File name
- FILE CONTENT***: File content
- Connected to [redacted] Change connection.

A red arrow points from the 'Create file' section of the bottom template down to the '+ New step' button at the bottom of the designer.

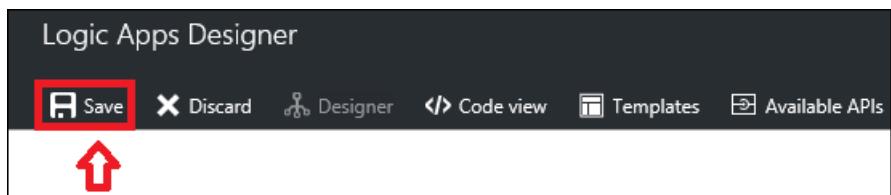
In the example above, as is the case with many templates, some of the mandatory property fields may be filled out within the connectors; however, some might still require a value before being able to properly deploy the logic app. If you try to deploy without entering some of the missing fields, you'll be notified with an error message.

If you wish to return to the template viewer, select the *Templates* button in the top navigation bar. By switching back to the template viewer, you lose any unsaved progress. Prior to switching back into template viewer, you'll see a warning message notifying you of this.



How to deploy a logic app created from a template

Once you have loaded your template and made any desired changes, select the save button in the upper left corner. This saves and publishes your logic app.



If you would like more information on how to add more steps into an existing logic app template, or make edits in general, read more at [Create a logic app](#).

Create a Logic App using a template

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Use an Azure Resource Manager template to create an empty logic app that can be used to define workflows. You can define which resources are deployed and how to define parameters that are specified when the deployment is executed. You can use this template for your own deployments, or customize it to meet your requirements.

For more details on the Logic app properties, see [Logic App Workflow Management API](#).

For examples of the definition itself, see [Author Logic App definitions](#).

For more information about creating templates, see [Authoring Azure Resource Manager Templates](#).

For the complete template, see [Logic App template](#).

What you will deploy

With this template, you deploy a logic app.

To run the deployment automatically, select the following button:



Parameters

With Azure Resource Manager, you define parameters for values you want to specify when the template is deployed. The template includes a section called Parameters that contains all of the parameter values. You should define a parameter for those values that will vary based on the project you are deploying or based on the environment you are deploying to. Do not define parameters for values that will always stay the same. Each parameter value is used in the template to define the resources that are deployed.

When defining parameters, use the **allowedValues** field to specify which values a user can provide during deployment. Use the **defaultValue** field to assign a value to the parameter, if no value is provided during deployment.

We will describe each parameter in the template.

logicAppName

The name of the logic app to create.

```
"logicAppName": {  
    "type": "string"  
}
```

testUri

```
"testUri": {  
    "type": "string",  
    "defaultValue": "http://azure.microsoft.com/en-us/status/feed/"  
}
```

Resources to deploy

Logic app

Creates the logic app.

The template uses a parameter value for the logic app name. It sets the location of the logic app to the same location as the resource group.

This particular definition runs once an hour, and pings the location specified in the **testUri** parameter.

```
{
  "type": "Microsoft.Logic/workflows",
  "apiVersion": "2016-06-01",
  "name": "[parameters('logicAppName')]",
  "location": "[resourceGroup().location]",
  "tags": {
    "displayName": "LogicApp"
  },
  "properties": {
    "definition": {
      "$schema": "http://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
      "contentVersion": "1.0.0.0",
      "parameters": {
        "testURI": {
          "type": "string",
          "defaultValue": "[parameters('testUri')]"
        }
      },
      "triggers": {
        "recurrence": {
          "type": "recurrence",
          "recurrence": {
            "frequency": "Hour",
            "interval": 1
          }
        }
      },
      "actions": {
        "http": {
          "type": "Http",
          "inputs": {
            "method": "GET",
            "uri": "@parameters('testUri')"
          },
          "runAfter": {}
        },
        "outputs": {}
      },
      "parameters": {}
    }
  }
}
```

Commands to run deployment

To deploy the resources to Azure, you must be logged in to your Azure account and you must use the Azure Resource Manager module. To learn about using Azure Resource Manager with either Azure PowerShell or Azure CLI, see:

- [Using Azure PowerShell with Azure Resource Manager](#)
- [Using the Azure CLI for Mac, Linux, and Windows with Azure Resource Management](#).

The following examples assume you already have a resource group in your account with the specified name.

PowerShell

```
New-AzureRmResourceGroupDeployment -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-logic-app-create/azuredeploy.json -ResourceGroupName ExampleDeployGroup
```

Azure CLI

```
azure group deployment create --template-uri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-logic-app-create/azuredeploy.json -g ExampleDeployGroup
```

Logic Apps Examples and Common Scenarios

1/20/2017 • 1 min to read • [Edit on GitHub](#)

This document details common scenarios and examples to help you to understand some of the ways you can use Logic apps to automate business processes.

Custom Triggers and Actions

There are several ways you can trigger a Logic app from another app. Here's a few common examples:

- [Creating a custom trigger or action](#)
- [Long-running actions](#)
- [HTTP request trigger \(POST\)](#)
- [Webhook triggers and actions](#)
- [Polling triggers](#)

Scenarios

- [Request synchronous response](#)
- [Request Response with SMS](#)

Error handling and logging

- [Exception and error handling](#)
- [Configure Azure Alerts and diagnostics](#)

Scenarios

- [Use Case: Error and exception handling](#)

Deploying and managing

- [Create an automated deployment](#)
- [Build and deploy logic apps from Visual Studio](#)
- [Monitor logic apps](#)

Content types, conversions, and transformations

The Logic Apps [workflow definition language](#) contains many functions to allow you to convert and work with different content types. In addition, the engine will do all it can to preserve content-types as data flows through the workflow.

- [Handling of content-types](#) such as application/json, application/xml, and text/plain
- [Authoring workflow definitions](#)
- [Workflow definition language reference](#)

Batches and looping

- [SplitOn](#)
- [ForEach](#)
- [Until](#)

Integrating with Azure Functions

- [Azure Functions integration](#)

Scenarios

- [Azure Function as a Service Bus trigger](#)

HTTP, REST, and SOAP

- [Calling SOAP](#)

We will keep adding examples and scenarios to this document. Use the comments section below to let us know what examples or scenarios you'd like to see here.

Logic app scenario: Create an Azure Service Bus trigger by using Azure Functions

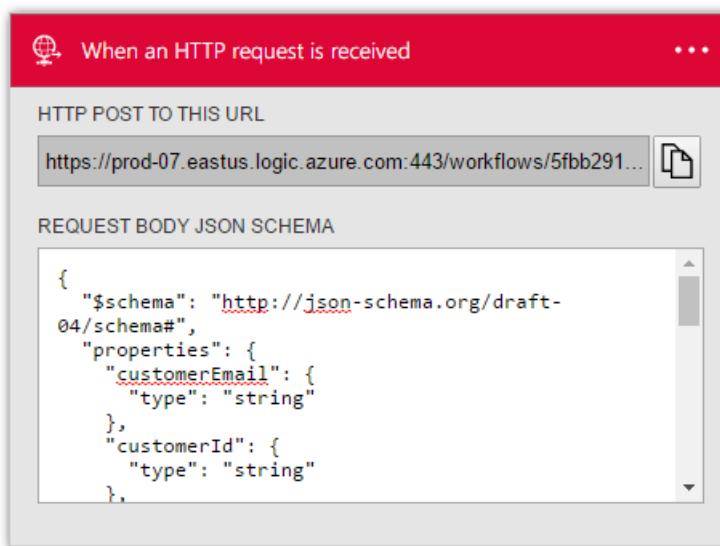
1/20/2017 • 1 min to read • [Edit on GitHub](#)

You can use Azure Functions to create a trigger for a logic app when you need to deploy a long-running listener or task. For example, you can create a function that will listen in on a queue and then immediately fire a logic app as a push trigger.

Build the logic app

In this example, you have a function running for each logic app that needs to be triggered. First, create a logic app that has an HTTP request trigger. The function calls that endpoint whenever a queue message is received.

1. Create a new logic app; select the **Manual - When an HTTP Request is Received** trigger.
Optionally, you can specify a JSON schema to use with the queue message by using a tool like [jsonschema.net](#). Paste the schema in the trigger. This helps the designer understand the shape of the data and more easily flow properties through the workflow.
2. Add any additional steps that you want to occur after a queue message is received. For example, send an email via Office 365.
3. Save the logic app to generate the callback URL for the trigger to this logic app. The URL appears on the trigger card.



Build the function

Next, you need to create a function that will act as the trigger and listen to the queue.

1. In the [Azure Functions portal](#), select **New Function**, and then select the **ServiceBusQueueTrigger - C#** template.

2. Configure the connection to the Service Bus queue (which will use the Azure Service Bus SDK `OnMessageReceive()` listener).
3. Write a simple function to call the logic app endpoint (created earlier) by using the queue message as a trigger. Here's a full example of a function. The example uses an `application/json` message content type, but you can change this if needed.

```

using System;
using System.Threading.Tasks;
using System.Net.Http;
using System.Text;

private static string logicAppUri = @"https://prod-05.westus.logic.azure.com:443/.....";

public static void Run(string myQueueItem, TraceWriter log)
{
    log.Info($"C# ServiceBus queue trigger function processed message: {myQueueItem}");
    using (var client = new HttpClient())
    {
        var response = client.PostAsync(logicAppUri, new StringContent(myQueueItem, Encoding.UTF8,
"application/json")).Result;
    }
}

```

To test, add a queue message via a tool like [Service Bus Explorer](#). See the logic app fire immediately after the function receives the message.

Learn about receiving data using the B2B features of the Enterprise Integration Pack

1/20/2017 • 3 min to read • [Edit on GitHub](#)

Overview

This document is part of the Logic Apps Enterprise Integration Pack. Check out the overview to learn more about the [capabilities of the Enterprise Integration Pack](#).

Prerequisites

To use the AS2 and X12 actions you will need an Enterprise Integration Account

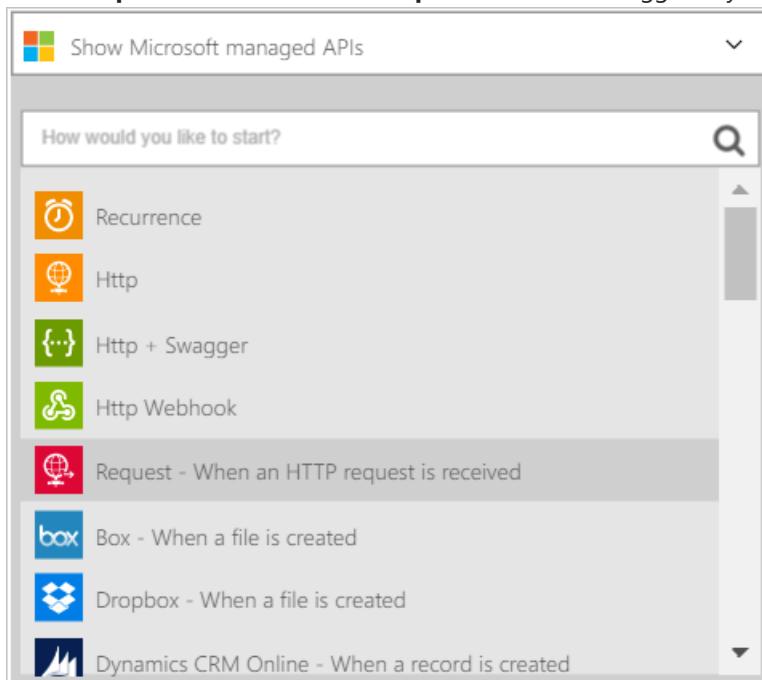
[How to create an Enterprise Integration Account](#)

How to use the Logic Apps B2B connectors

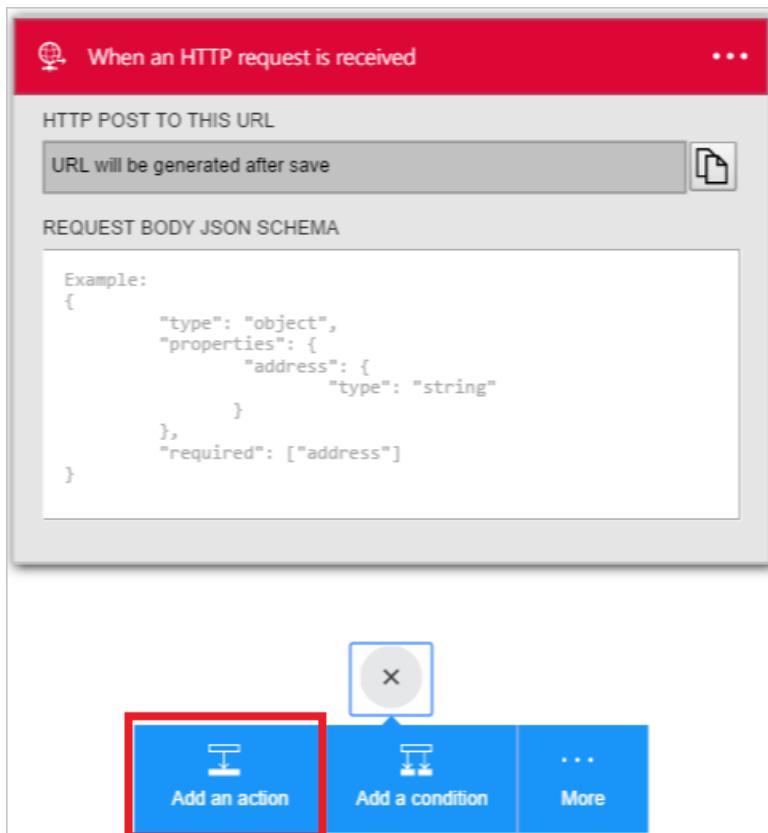
Once you have created an integration account and added partners and agreements to it you are ready to create a Logic App that implements a business to business (B2B) workflow.

In this walkthrough you'll see how to use the AS2 and X12 actions to create a business to business Logic App that receives data from a trading partner.

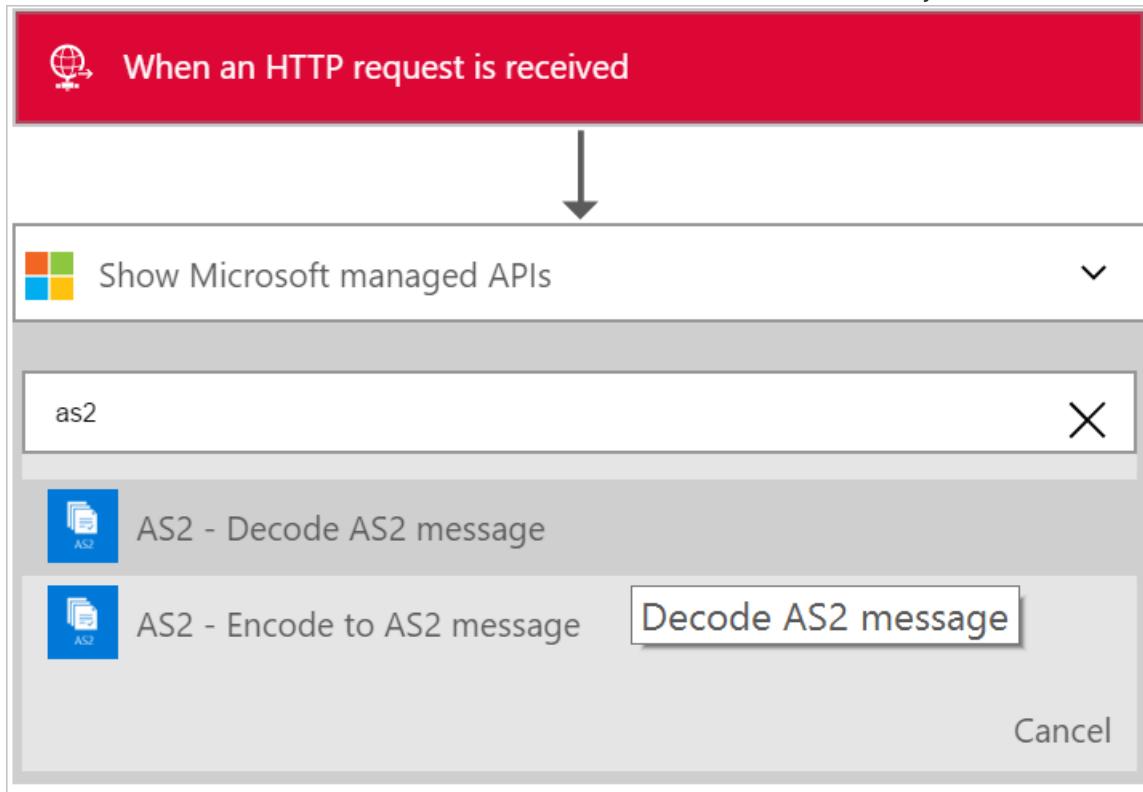
1. Create a new Logic app and [link it to your integration account](#).
2. Add a **Request - When an HTTP request is received** trigger to your Logic app



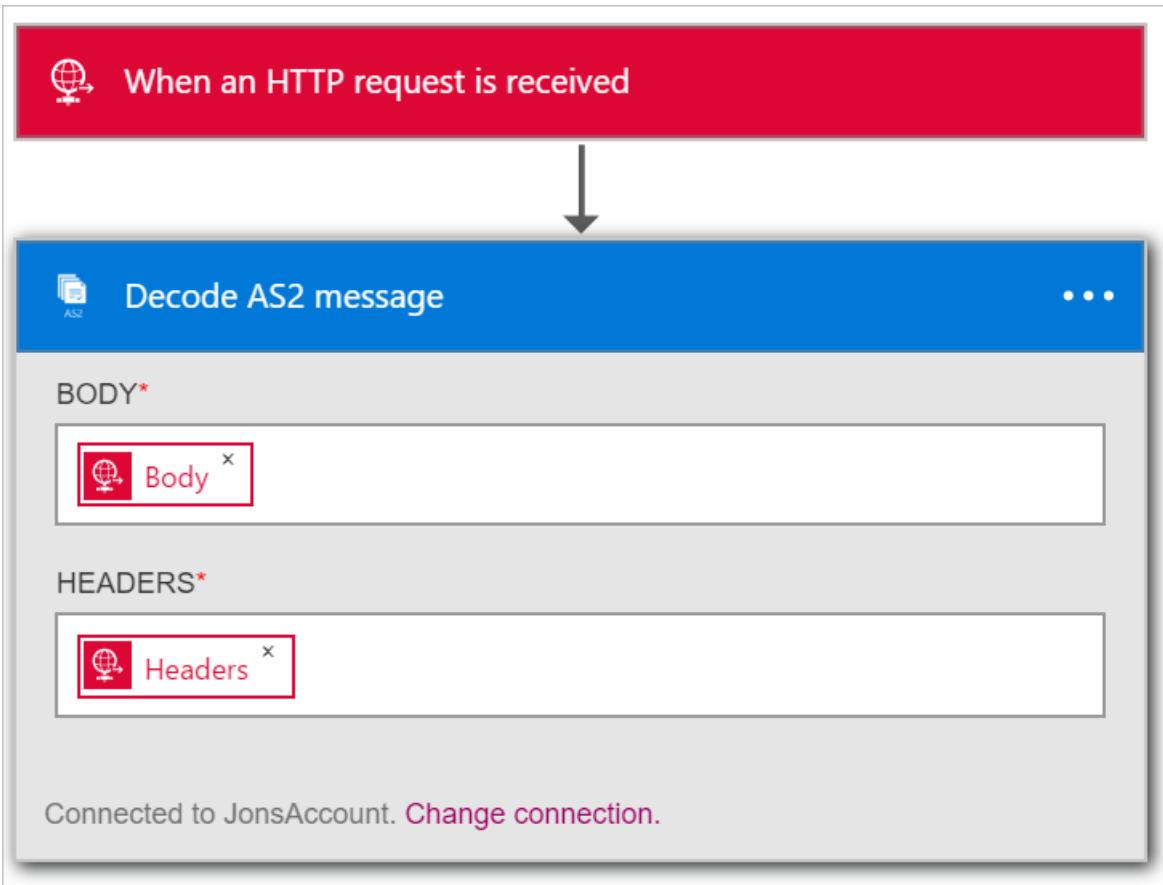
3. Add the **Decode AS2** action by first selecting **Add an action**



4. Enter the word **as2** in the search box in order to filter all the actions to the one that you want to use



5. Select the **AS2 - Decode AS2 message** action



6. As shown, add the **Body** that you will take as input. In this example, select the body of the HTTP request that triggered the Logic app. You can alternatively enter an expression to input the headers in the**HEADERS** field:

```
@triggerOutputs()['headers']
```

7. Add the **Headers** that are required for AS2. These will be in the HTTP request headers. In this example, select the headers of the HTTP request that triggered the Logic app.
8. Now add the Decode X12 message action by again selecting **Add an action**



When an HTTP request is received



Decode AS2 message

...

BODY*



Body ×

HEADERS*



Headers ×

Connected to JonsAccount. Change connection.



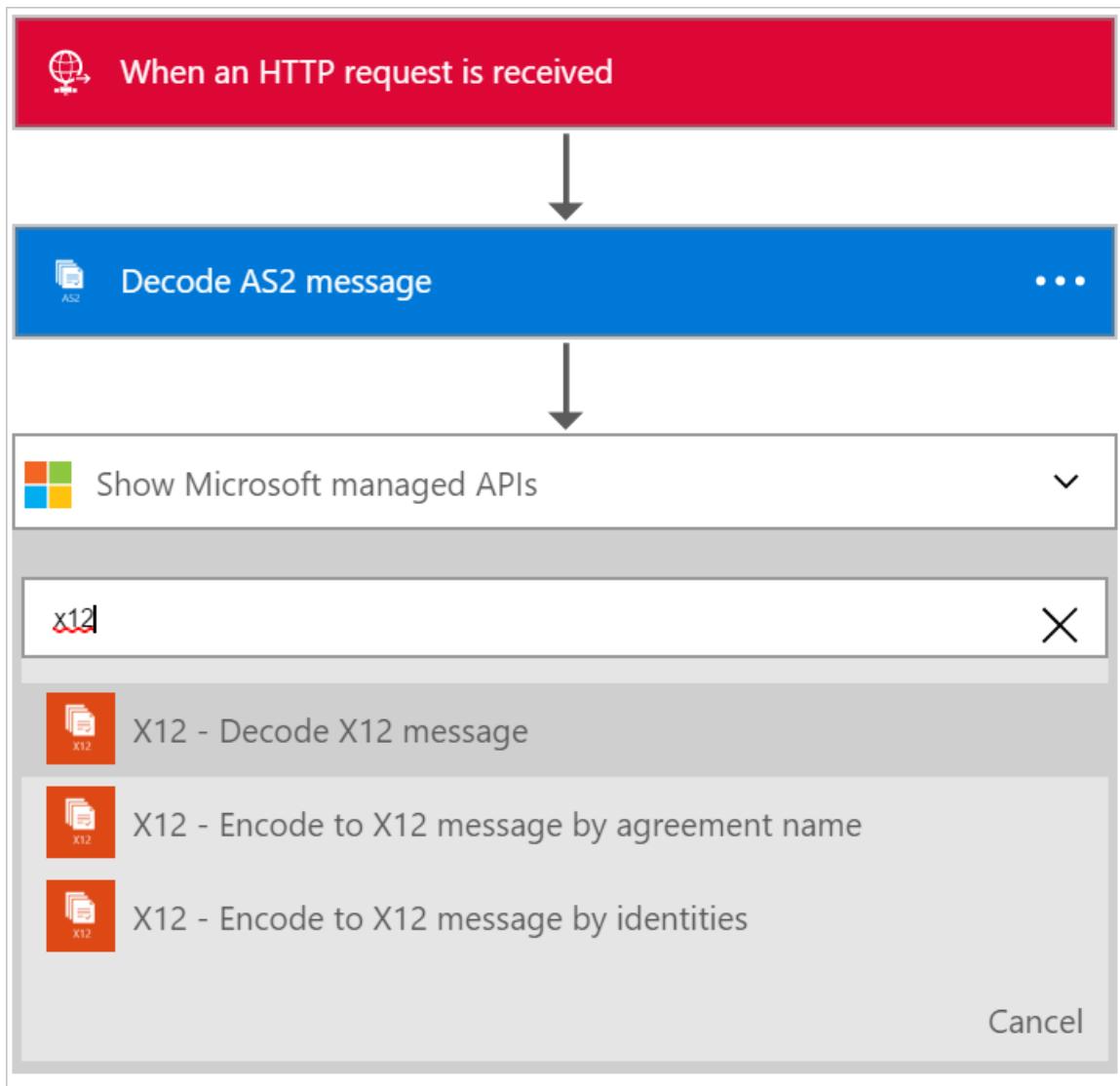
Add an action

Add a condition

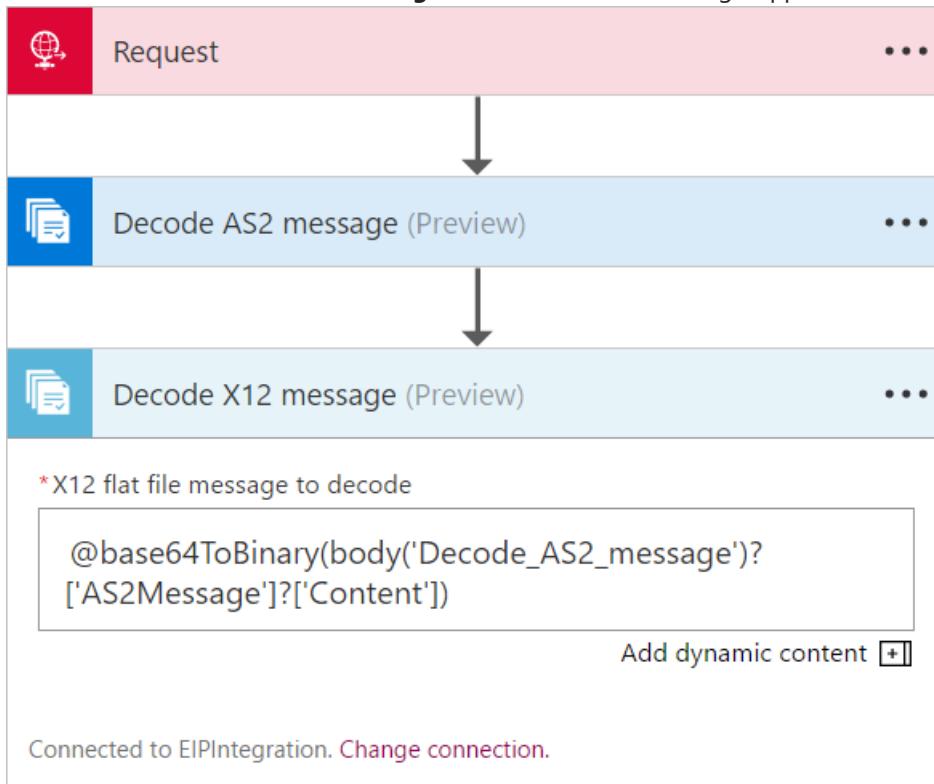
...

More

9. Enter the word **x12** in the search box in order to filter all the actions to the one that you want to use



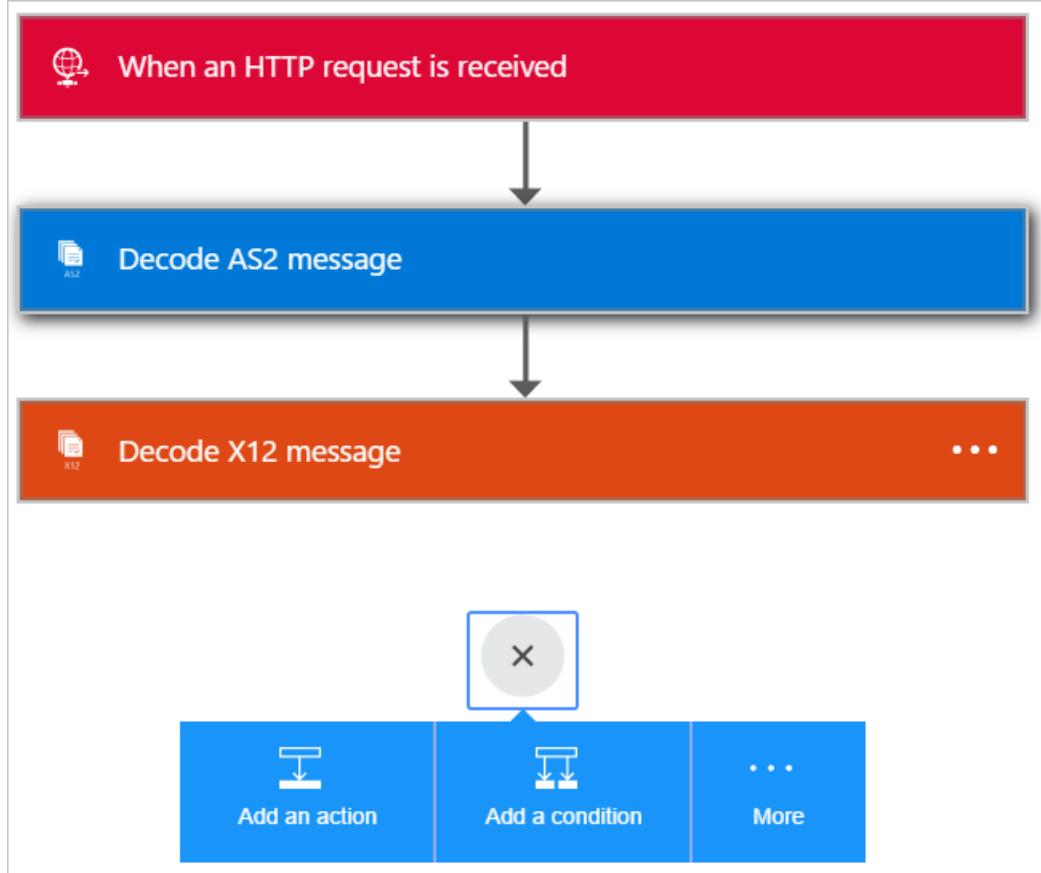
10. Select the **X12 - Decode X12 message** action to add it to the Logic app



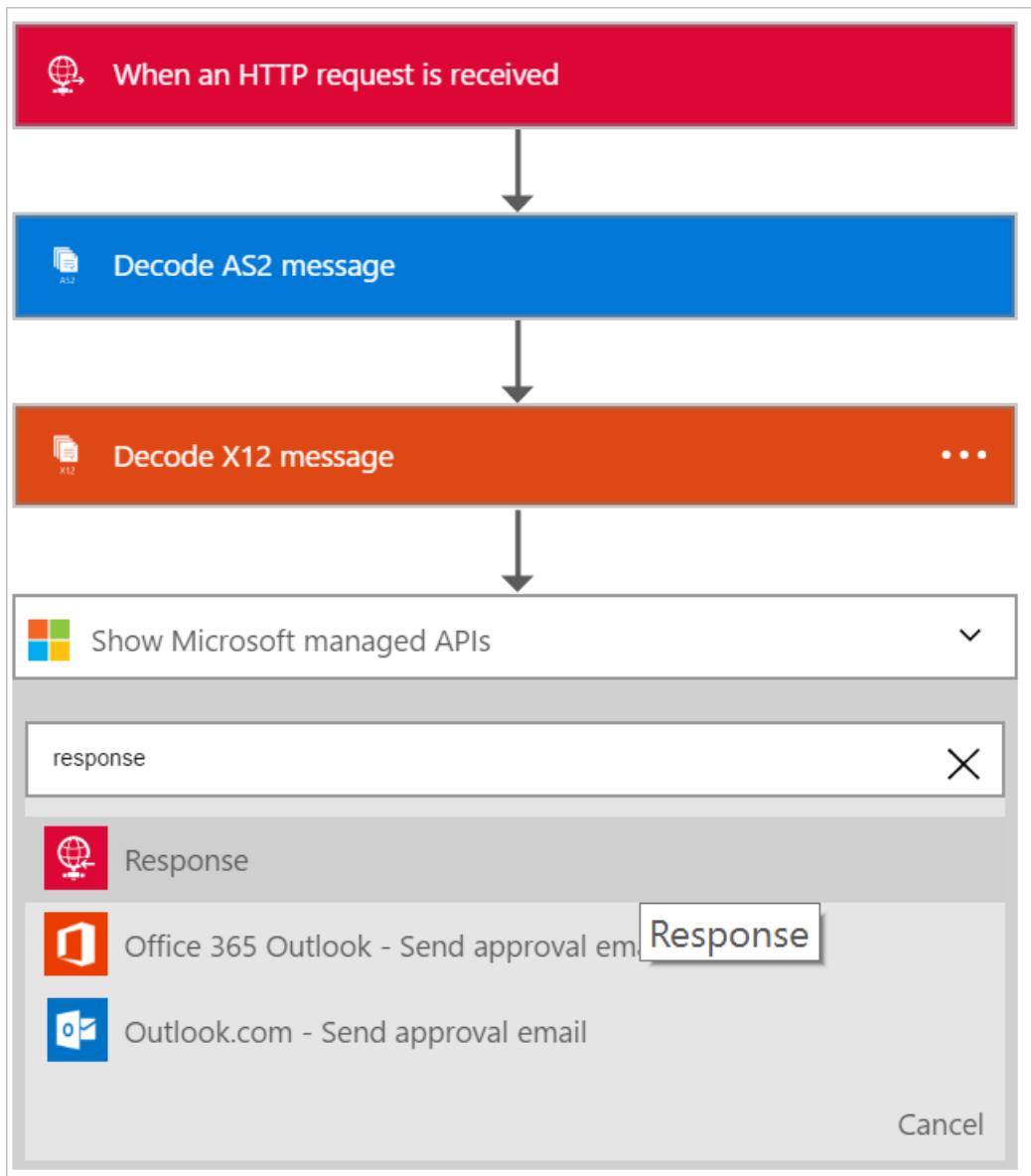
11. You now need to specify the input to this action which will be the output of the AS2 action above. The actual message content is in a JSON object and is base64 encoded. You therefore need to specify an expression as the input so enter the following expression in the **X12 FLAT FILE MESSAGE TO DECODE** input field

```
@base64ToString(body('Decode_AS2_message')?['AS2Message']?['Content'])
```

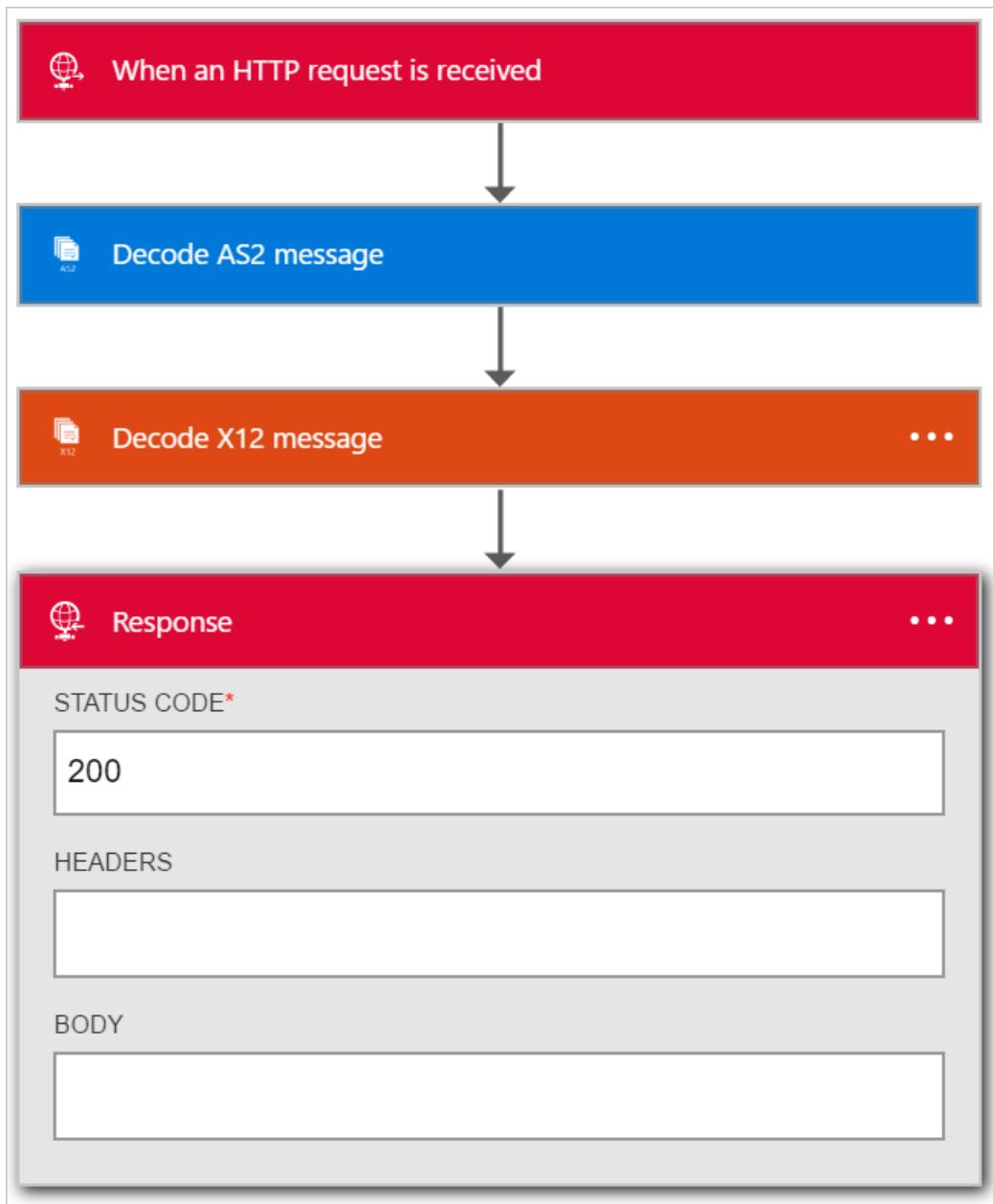
12. This step will decode the X12 data received from the trading partner and will output a number of items in a JSON object. In order to let the partner know of the receipt of the data you can send back a response containing the AS2 Message Disposition Notification (MDN) in an HTTP Response Action
13. Add the **Response** action by selecting **Add an action**



14. Enter the word **response** in the search box in order to filter all the actions to the one that you want to use

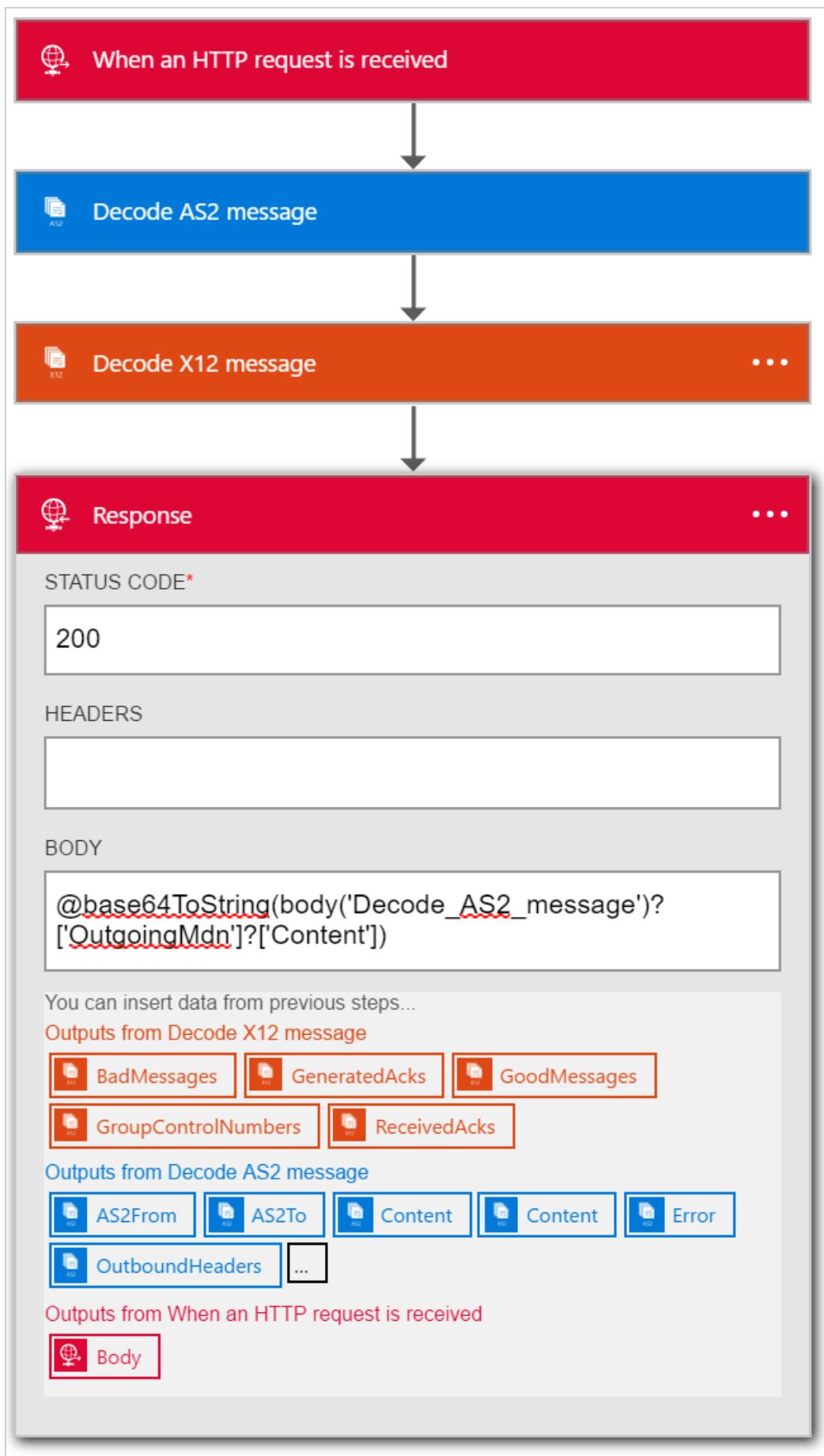


15. Select the **Response** action to add it

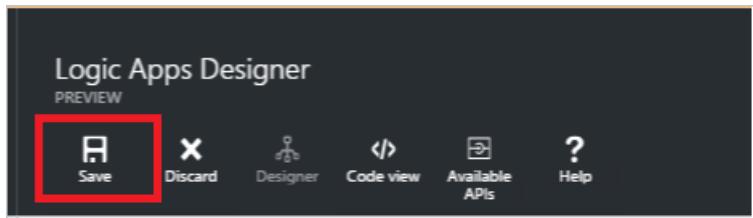


16. Set the response **BODY** field by using the following expression to access the MDN from the output of the **Decode X12 message** action

```
@base64ToString(body('Decode_AS2_message')?['OutgoingMdn']?['Content'])
```



1. Save your work



At this point, you are finished setting up your B2B Logic app. In a real world application, you may want to store the decoded X12 data in an LOB application or data store. You can easily add further actions to do this or write custom APIs to connect to your own LOB applications and use these APIs in your Logic app.

Features and use cases

- The AS2 and X12 decode and encode actions allow you to receive data from and send data to trading partners using industry standard protocols using Logic apps
- You can use AS2 and X12 with or without each other to exchange data with trading partners as required
- The B2B actions make it easy to create partners and agreements in the Integration Account and consume them in a Logic app
- By extending your Logic app with other actions you can send and receive data to and from other applications and services such as SalesForce

Learn more

[Learn more about the Enterprise Integration Pack](#)

XML processing

1/20/2017 • 1 min to read • [Edit on GitHub](#)

The Enterprise Integration Pack makes it easy to validate and process XML documents that you exchange with business partners. Here are the ways you can process these XML messages using Logic apps:

- [XML validation](#) - XML validation provides the ability to validate a message that originates from a source endpoint against a specific schema.
- [XML transform](#) - XML transform provides the ability to convert an XML message based on the requirements of a destination endpoint.
- [Flat file encoding and flat file decoding](#) - Flat file encoding/decoding provides the ability to encode or decode a flat file.
- [XPath](#) - Provides the ability to enrich a message and extract specific properties from the message. The extracted properties can then be used to route the message to a destination or an intermediary endpoint.

Try it for yourself

Why not give it a try. Click [here](#) to deploy a fully operational logic app of your own using the XML features of Logic Apps

Learn more

[Learn more about the Enterprise Integration Pack](#)

Creating a custom API to use with Logic Apps

1/20/2017 • 6 min to read • [Edit on GitHub](#)

If you want to extend the Logic Apps platform, there are many ways you can call into APIs or systems that aren't available as one of our many out-of-the-box connectors. One of those ways to create an API App you can call from within a Logic App workflow.

Helpful Tools

For APIs to work best with Logic Apps, we recommend generating a [swagger](#) doc detailing the supported operations and parameters for your API. There are many libraries (like [Swashbuckle](#)) that will automatically generate the swagger for you. You can also use [TREx](#) to help annotate the swagger to work well with Logic Apps (display names, property types, etc.). For some samples of API Apps built for Logic Apps, be sure to check out our [GitHub repository](#) or [blog](#).

Actions

The basic action for a Logic App is a controller that will accept an HTTP Request and return a response (usually 200). However there are different patterns you can follow to extend actions based on your needs.

By default the Logic App engine will timeout a request after 1 minute. However, you can have your API execute on actions that take longer, and have the engine wait for completion, by following either an async or webhook pattern detailed below.

For standard actions, simply write an HTTP request method in your API which is exposed via swagger. You can see samples of API apps that work with Logic Apps in our [GitHub repository](#). Below are ways to accomplish common patterns with a custom connector.

Long Running Actions - Async Pattern

When running a long step or task, the first thing you need to do is make sure the engine knows you haven't timed out. You also need to communicate with the engine how it will know when you are finished with the task, and finally, you need to return relevant data to the engine so it can continue with the workflow. You can complete that via an API by following the flow below. These steps are from the point-of-view of the custom API:

1. When a request is received, immediately return a response (before work is done). This response will be a `202 ACCEPTED` response, letting the engine know you got the data, accepted the payload, and are now processing. The 202 response should contain the following headers:

- `location` header (required): This is an absolute path to the URL Logic Apps can use to check the status of the job.
- `retry-after` (optional, will default to 20 for actions). This is the number of seconds the engine should wait before polling the location header URL to check status.

2. When a job status is checked, perform the following checks:

- If the job is done: return a `200 OK` response, with the response payload.
- If the job is still processing: return another `202 ACCEPTED` response, with the same headers as the initial response

This pattern allows you to run extremely long tasks within a thread of your custom API, but keep an active connection alive with the Logic Apps engine so it doesn't timeout or continue before work is completed. When adding this into your Logic App, it's important to note you do not need to do anything in your definition for the

Logic App to continue to poll and check the status. As soon as the engine sees a 202 ACCEPTED response with a valid location header, it will honor the async pattern and continue to poll the location header until a non-202 is returned.

You can see a sample of this pattern in GitHub [here](#)

Webhook Actions

During your workflow, you can have the Logic App pause and wait for a "callback" to continue. This callback comes in the form of an HTTP POST. To implement this pattern, you need to provide two endpoints on your controller: subscribe and unsubscribe.

On 'subscribe', the Logic App will create and register a callback URL which your API can store and callback with ready as an HTTP POST. Any content/headers will be passed into the Logic App and can be used within the remainder of the workflow. The Logic App engine will call the subscribe point on execution as soon as it hits that step.

If the run was cancelled, the Logic App engine will make a call to the 'unsubscribe' endpoint. Your API can then unregister the callback URL as needed.

Currently the Logic App Designer doesn't support discovering a webhook endpoint through swagger, so to use this type of action you must add the "Webhook" action and specify the URL, headers, and body of your request.

You can use the `@listCallbackUrl()` workflow function in any of those fields as needed to pass in the callback URL.

You can see a sample of a webhook pattern in GitHub [here](#)

Triggers

In addition to actions, you can have your custom API act as a trigger to a Logic App. There are two patterns you can follow below to trigger a Logic App:

Polling Triggers

Polling triggers act much like the Long Running Async actions above. The Logic App engine will call the trigger endpoint after a certain period of time elapsed (dependent on SKU, 15 seconds for Premium, 1 minute for Standard, and 1 hour for Free).

If there is no data available, the trigger returns a `202 ACCEPTED` response, with a `location` and `retry-after` header. However, for triggers it is recommended the `location` header contains a query parameter of `triggerState`. This is some identifier for your API to know when the last time the Logic App fired. If there is data available, the trigger returns a `200 OK` response with the content payload. This will fire the Logic App.

For example if I was polling to see if a file was available, you could build a polling trigger that would do the following:

- If a request was received with no triggerState the API would return a `202 ACCEPTED` with a `location` header that has a triggerState of the current time and a `retry-after` of 15.
- If a request was received with a triggerState:
 - Check to see if any files were added after the triggerState DateTime.
 - If there is 1 file, return a `200 OK` response with the content payload, increment the triggerState to the DateTime of the file I returned, and set the `retry-after` to 15.
 - If there are multiple files, I can return 1 at a time with a `200 OK`, increment my triggerState in the `location` header, and set `retry-after` to 0. This will let the engine know there is more data available and it will immediately request it at the `location` header specified.
 - If there are no files available, return a `202 ACCEPTED` response, and leave the `location` triggerState the same. Set `retry-after` to 15.

You can see a sample of a polling trigger in GitHub [here](#)

Webhook Triggers

Webhook triggers act much like Webhook Actions above. The Logic App engine will call the 'subscribe' endpoint whenever a webhook trigger is added and saved. Your API can register the webhook URL and call it via HTTP POST whenever data is available. The content payload and headers will be passed into the Logic App run.

If a webhook trigger is ever deleted (either the Logic App entirely, or just the webhook trigger), the engine will make a call to the 'unsubscribe' URL where your API can unregister the callback URL and stop any processes as needed.

Currently the Logic App Designer doesn't support discovering a webhook trigger through swagger, so to use this type of action you must add the "Webhook" trigger and specify the URL, headers, and body of your request. You can use the `@listCallbackUrl()` workflow function in any of those fields as needed to pass in the callback URL.

You can see a sample of a webhook trigger in GitHub [here](#)

Use File System Connector With On-Premises Data Gateway

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Hybrid cloud connectivity is at the heart of Logic Apps. On-premises data gateway enables you to manage data and securely access resources that are on-premises from Logic Apps. In this article, we demonstrate how to connect to an on-premises file system with a simple scenario: copy a file that's uploaded to Dropbox to a file share, then send an email.

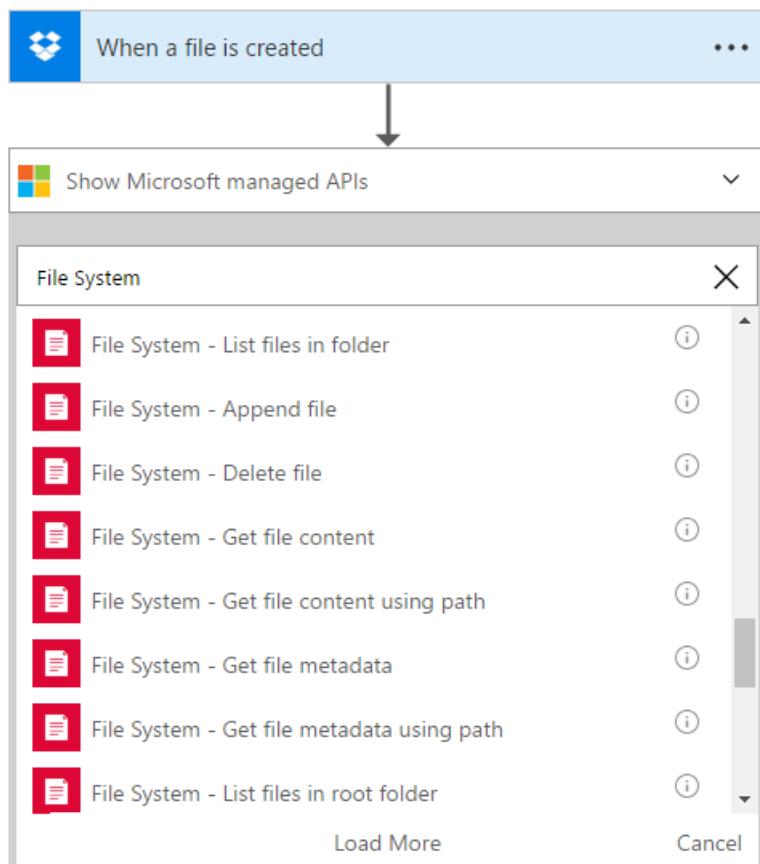
Prerequisites

- Install and configure the latest [on-premises data gateway](#).

Install the latest on-premises data gateway, version 1.15.6150.1 or above, if you haven't already. Instructions can be found in [this article](#). The gateway must be installed on an on-premises machine before you can continue with the rest of the steps.

Use File System Connector

1. Let's create a Dropbox "When a file is created" trigger, then, get a glance of all the supported file connector action is as simple as typing "File System" in search.



2. Choose "Create file" and create a connection for it.

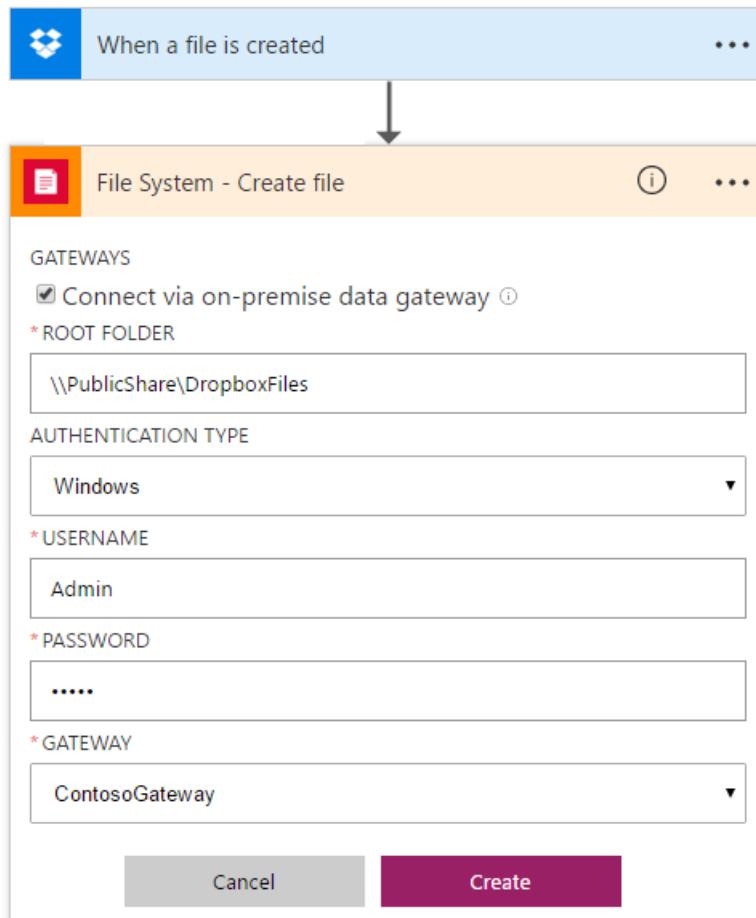
- If you don't have an existing connection, you will be prompted to create one.
- Check "Connect via on-premises data gateway" option, you will see additional fields shows up once the

checkbox is selected.

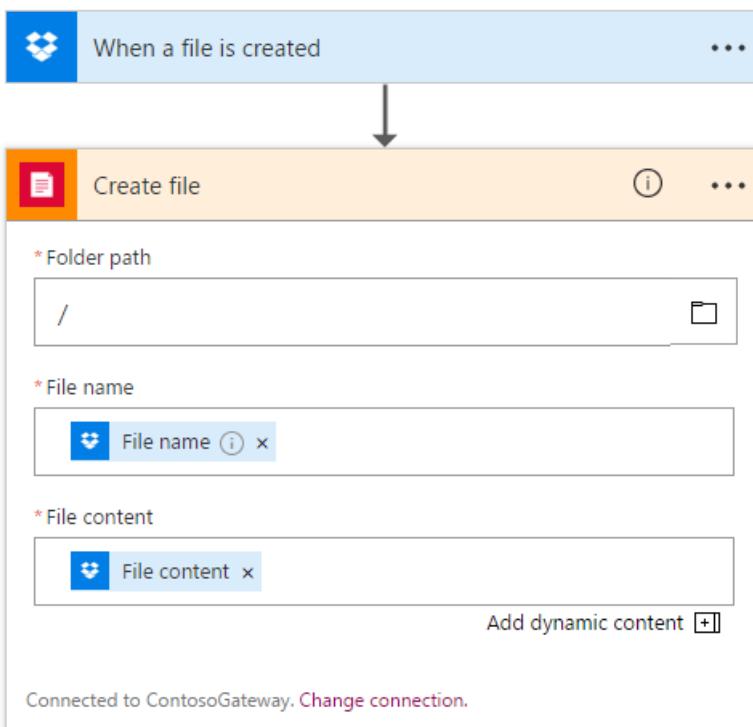
- Specify the root folder, it can be a local folder on the machine in which on-premises data gateway is installed, or a network share in which the machine has access to.
- Enter the username and password to the gateway.
- Select the gateway you installed from previous step.

NOTE

Root Folder is the main parent folder, which will be used for relative paths for all file-related actions.

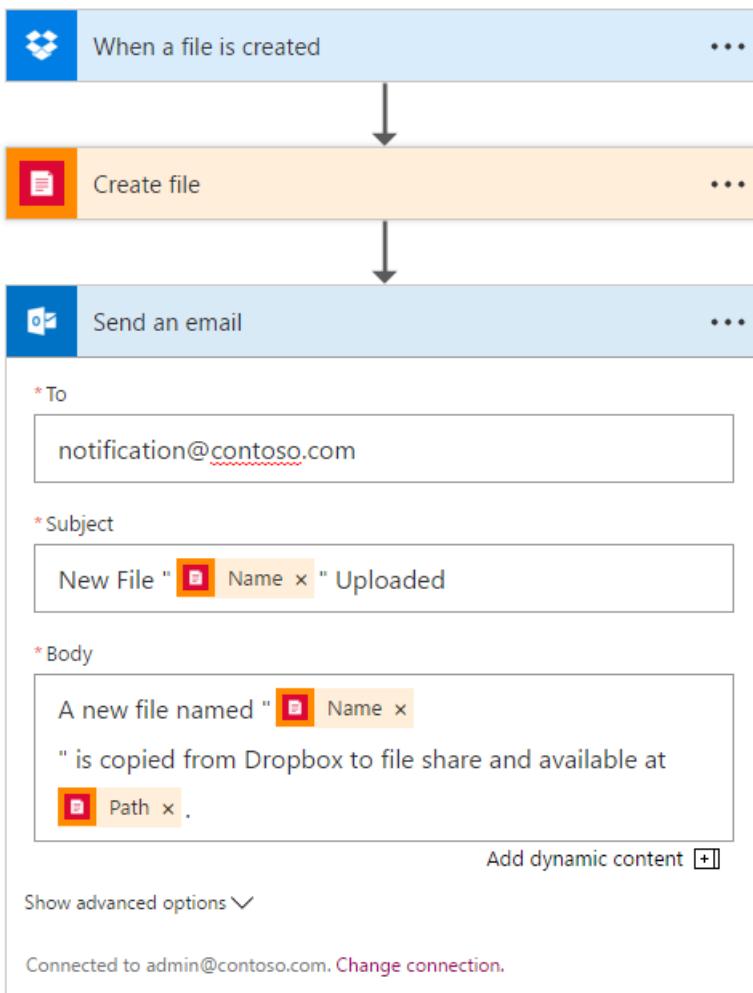


3. Once you have provided all the details, click "Create". Logic Apps will configure and test the connection to make sure it's working properly. If everything checks out, you will see options for the card you selected previously, and the file connector is now ready for use.
4. Take the file uploaded to Dropbox, and copy it to the root folder of the file share located on-premises.



5. Once the file is copied, let's send an e-mail so relevant users know about it. Like other connectors, output from previous actions will be available in the "dynamic content" selector.

- Specify the recipients, title, and body of the email. Use "dynamic content" selector to pick the output from file connector to make the email richer.



6. Save your Logic App, and test it by uploading a file to Dropbox. You should see the file being copied to the on-premises file share, and receive an email notification on the operation.

TIP

Check out how to [monitor your Logic Apps](#).

7. All done, now you have a working Logic App using the file system connector. You can start exploring other functionalities it offers:

- Create file
- List files in folder
- Append file
- Delete file
- Get file content
- Get file content using path
- Get file metadata
- Get file metadata using path
- List files in root folder
- Update file

Next steps

- Learn about [Enterprise Integration Pack](#).
- Create an [on-premises connection](#) to Logic Apps.

Get started with the SAP Connector

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Hybrid cloud connectivity is at the heart of Logic Apps. On-premises data gateway enables you to manage data and securely access resources that are on-premises from Logic Apps. In this article, we demonstrate how to connect to an on-premises SAP system with a simple scenario: Request an IDOC over HTTP and send the response back.

NOTE

This SAP Connector supports the following SAP Systems There is currently a limitation to time out in Logic Apps that block requests over 90 seconds There is currently a limitation to how many fields to show in the file picker (paths may be added manually)

Prerequisites

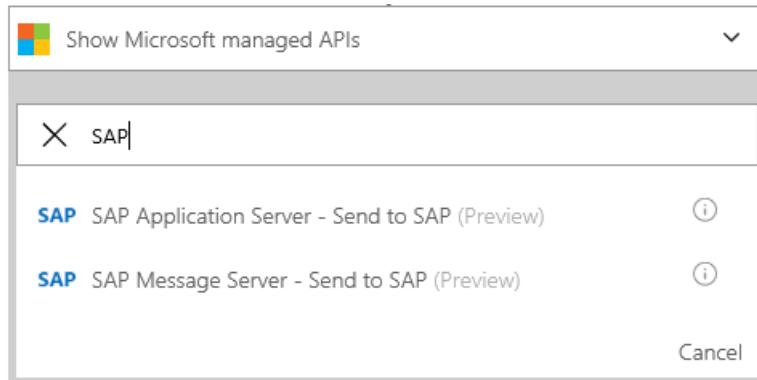
- Install and configure the latest [on-premises data gateway](#).

Install the latest on-premises data gateway, version 1.15.6150.1 or above, if you haven't already. Instructions can be found in [this article](#). The gateway must be installed on an on-premises machine before you can continue with the rest of the steps.

- Download and install the latest SAP Client library on the same machine where you installed the universal gateway

Use SAP Connector

1. Let's create an HTTP Request trigger, then, select the SAP connector action by typing "SAP" in the search field.



2. Choose "SAP" (ApplicationHost or MessagingHost based on your SAP setup) and create a connection for it by using the universal gateway.

- If you don't have an existing connection, you are prompted to create one.
- Check "Connect via on-premises data gateway" option, you see additional fields shows up once the checkbox is selected.
- Specify the connection name string
- Select the gateway you installed from previous step or select "Install Gateway" to install a new gateway.

SAP SAP Application Server - Send to SAP

GATEWAYS

Connect via on-premise data gateway ⓘ

* SAP CLIENT NUMBER

1001

* APPLICATION SERVER: HOST ADDRESS

mySAPInstallation.domain.com

* SYSTEM NUMBER

1000

SAP GATEWAY HOST

8088

SAP GATEWAY SERVER

Gateway Server

AUTHENTICATION TYPE

Basic

* SAP USERNAME

myUsername

* SAP PASSWORD

Password credential of SAP System

* GATEWAY

+ Install gateway

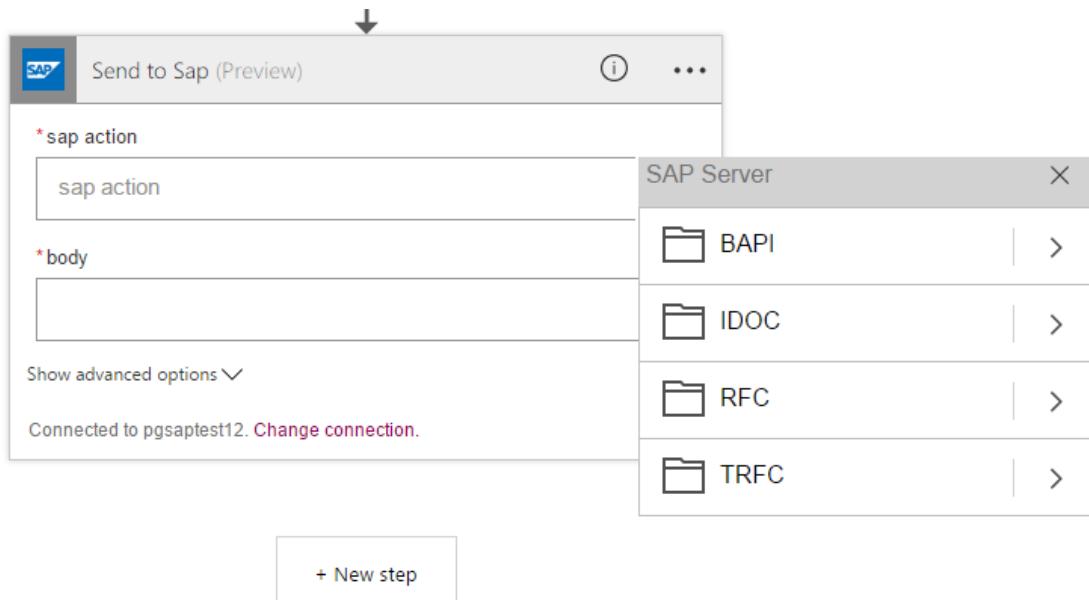
Create

The screenshot shows the configuration of an SAP connection in the Logic Apps designer. The connection is named "SAP Application Server - Send to SAP". The "GATEWAYS" section is expanded, showing the "Connect via on-premise data gateway" checkbox is checked. Other fields include SAP CLIENT NUMBER (1001), APPLICATION SERVER: HOST ADDRESS (mySAPInstallation.domain.com), and SYSTEM NUMBER (1000). The SAP GATEWAY HOST is set to 8088, and the SAP GATEWAY SERVER is set to Gateway Server. Under AUTHENTICATION TYPE, Basic is selected. The SAP USERNAME is myUsername, and the SAP PASSWORD is a placeholder for a password credential of the SAP System. The GATEWAY section shows an option to "+ Install gateway". At the bottom is a large "Create" button.

NOTE

There are two different SAP Connections, one for [Application Host](#) and another for [Messaging host](#)

- Once you have provided all the details, click "Create". Logic Apps configure and test the connection to make sure it's working properly. If everything checks out, you see options for the card you selected previously, use the file picker to find the right IDOC category or manually type in the path and select the HTTP response in the body field.



4. Create an HTTP Response by adding a new action, the response message should be from the SAP output
5. Save your Logic App and test it by sending an IDOC through the HTTP trigger URL
6. Once the IDOC is sent, wait for the response from the Logic App

TIP

Check out how to [monitor your Logic Apps](#).

7. All done, now you have a working Logic App using the SAP connector. You can start exploring other functionalities it offers:

- BAPI
- RFC

Next steps

- Learn about [Enterprise Integration Pack](#).
- Create an [on-premises connection](#) to Logic Apps.

Build and Deploy Logic Apps in Visual Studio

1/20/2017 • 4 min to read • [Edit on GitHub](#)

Although the [Azure portal](#) gives you a great way to design and manage your Logic apps, you may also want to design and deploy your logic app from Visual Studio instead. Logic Apps comes with a rich Visual Studio toolset, which allows you to build a logic app using the designer, configure any deployment and automation templates, and deploy into any environment.

Installation steps

Below are the steps to install and configure the Visual Studio tools for Logic Apps.

Prerequisites

- [Visual Studio 2015](#)
- [Latest Azure SDK](#) (2.9.1 or greater)
- [Azure PowerShell](#)
- Access to the web when using the embedded designer

Install Visual Studio tools for Logic Apps

Once you have the prerequisites installed,

1. Open Visual Studio 2015 to the **Tools** menu and select **Extensions and Updates**
2. Select the **Online** category to search online
3. Search for **Logic Apps** to display the **Azure Logic Apps Tools for Visual Studio**
4. Click the **Download** button to download and install the extension
5. Restart Visual Studio after installation

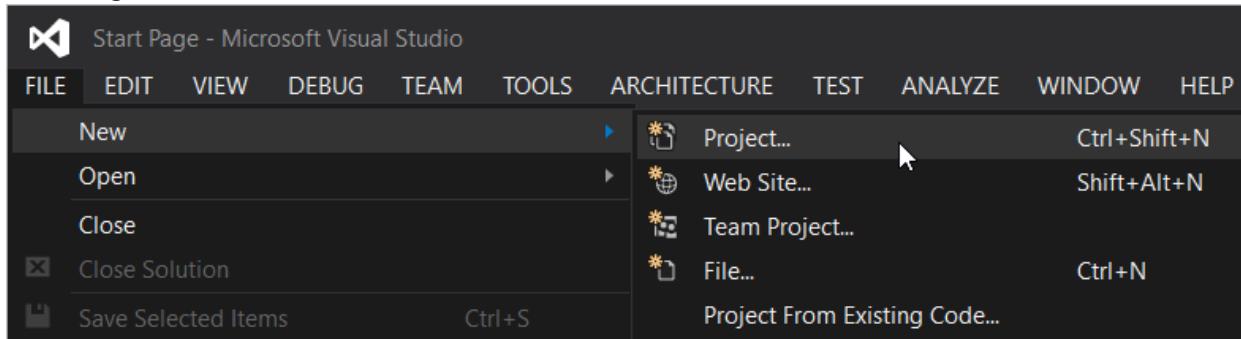
NOTE

You can also download the extension directly from [this link](#)

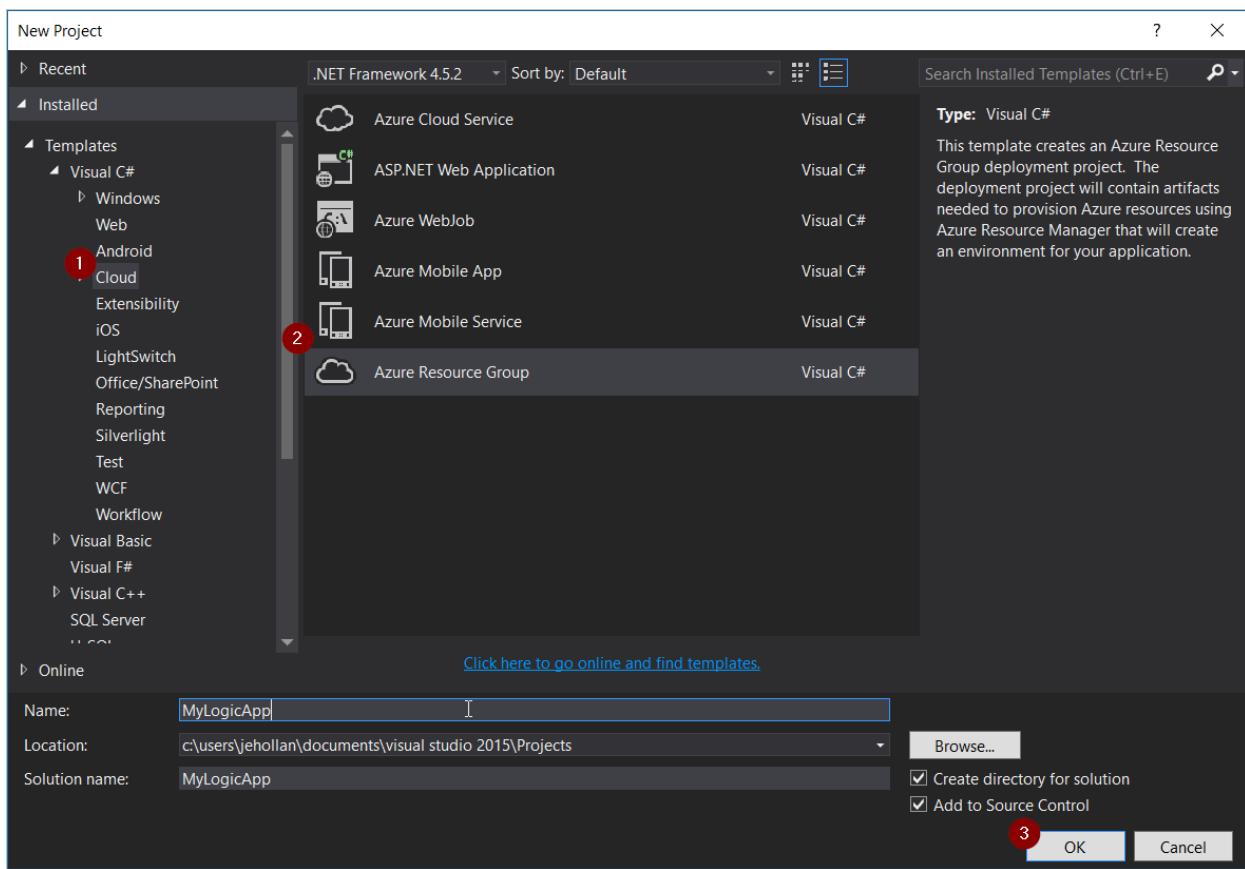
Once installed you are able to use the Azure Resource Group project with the Logic App Designer.

Create a project

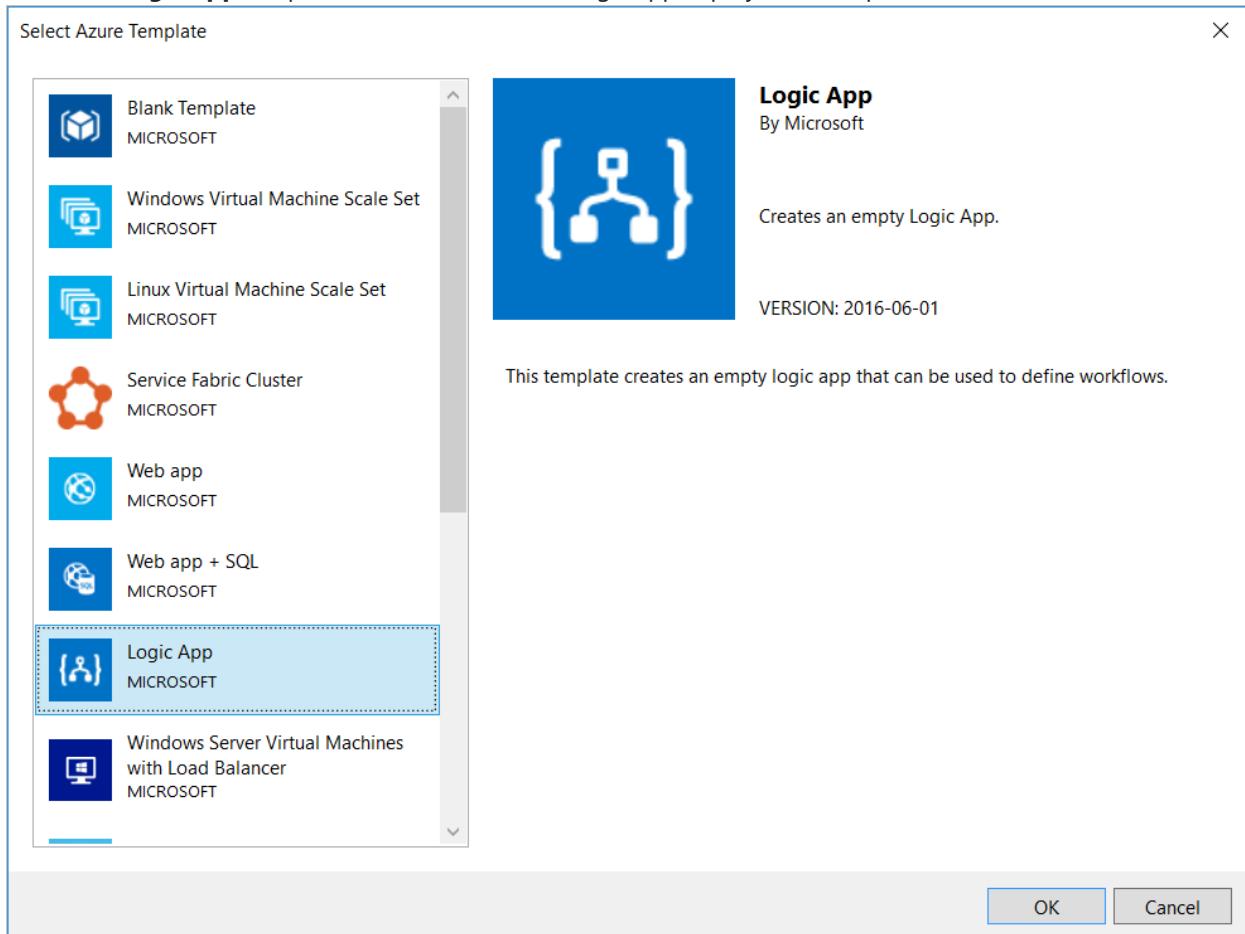
1. Go to the **File** menu and select **New > Project** (or, you can go to **Add** and then select **New project** to add it to an existing solution):



2. In the dialog, find **Cloud**, and then select **Azure Resource Group**. Type a **Name** and then click **OK**.

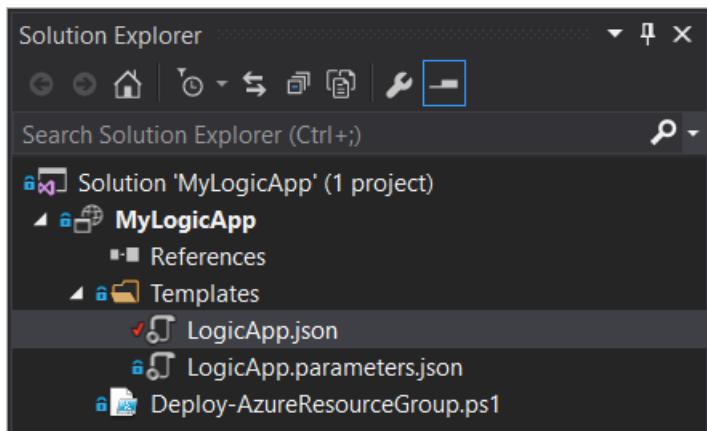


3. Select the **Logic app** template. This creates a blank logic app deployment template to start with.



4. Once you have selected your **Template**, hit **OK**.

Now your Logic app project is added to your solution. You should see the deployment file in the Solution Explorer:



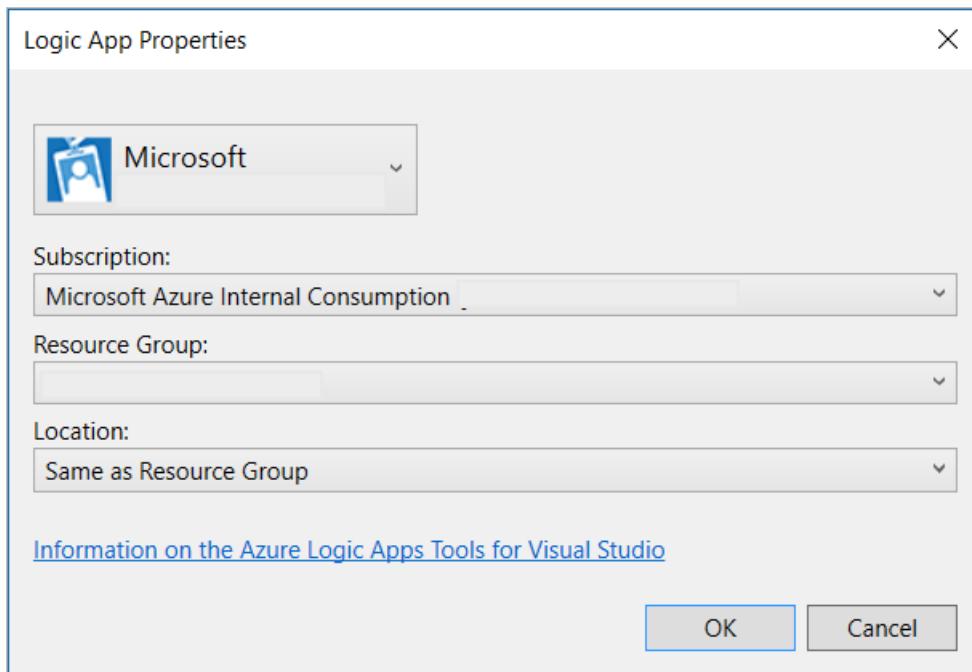
Using the Logic App Designer

Once you have an Azure Resource Group project that contains a logic app, you can open the designer within Visual Studio to assist you in creating the workflow. The designer requires an internet connection to query the connectors for available properties and data (for example, if using the Dynamics CRM Online connector, the designer queries your CRM instance to list available custom and default properties).

1. Right-click on the `<template>.json` file and select **Open with Logic App Designer** (or `Ctrl+L`)

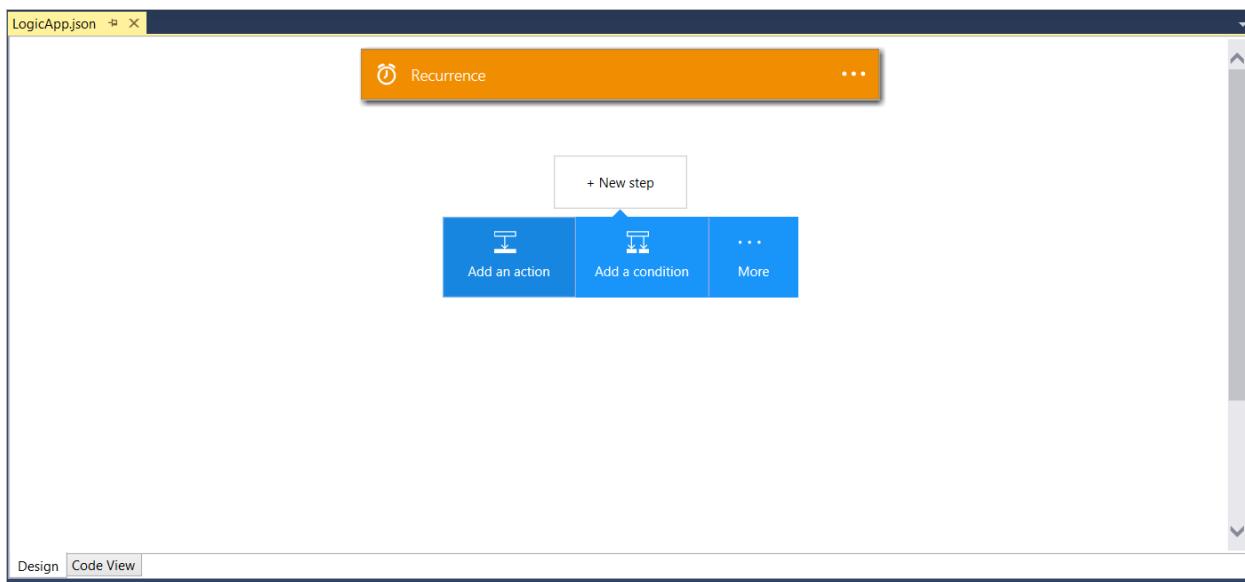
2. Choose the subscription, resource group, and location for the deployment template

- It's important to note that designing a logic app will create **API Connection** resources to query for properties during design. The resource group selected is used to create those connections during design-time. You can view or modify any API Connections by going to the Azure portal and browsing for **API Connections**.



3. The designer should render based on the definition in the `<template>.json` file.

4. You can now create and design your logic app, and changes are updated in the deployment template.



`Microsoft.Web/connections` resources are added to your resource file for any connections needed for the logic app to function. These connection properties can be set when you deploy, and managed after you deploy in **API Connections** in the Azure portal.

Switching to the JSON code-view

You can select the **Code View** tab on the bottom of the designer to switch to the JSON representation of the logic app. To switch back to the full resource JSON, right-click the `<template>.json` file and select **Open**.

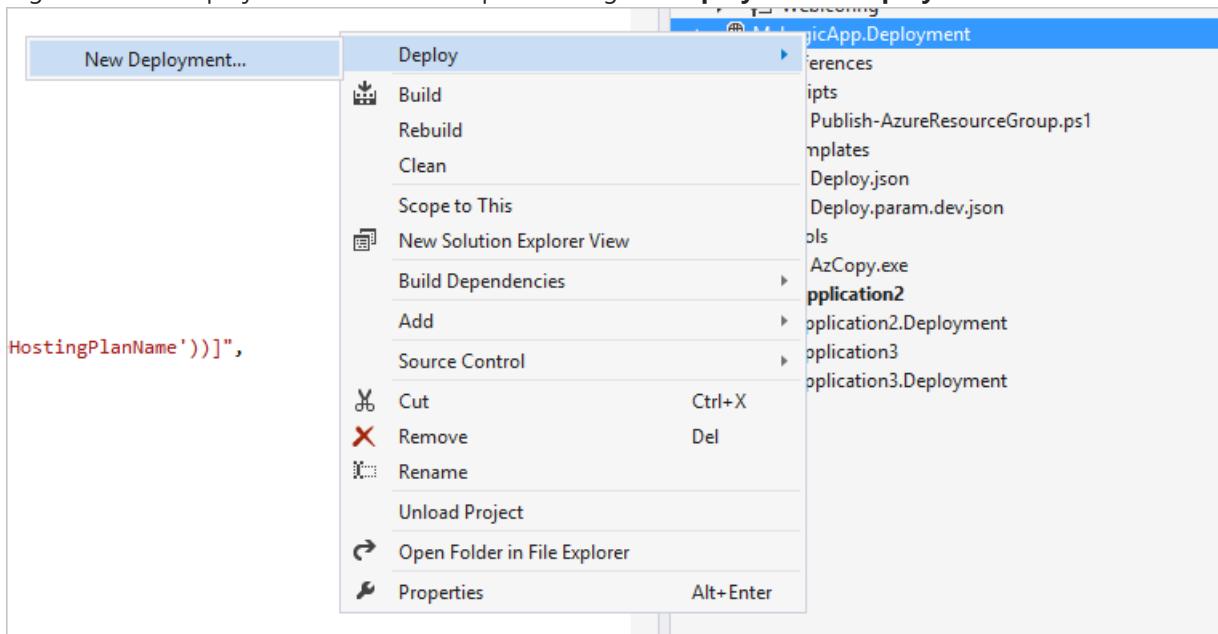
Saving the logic app

You can save the logic app at anytime via the **Save** button or `Ctrl+S`. If there are any errors with your logic app at the time you save, they are displayed in the **Outputs** window of Visual Studio.

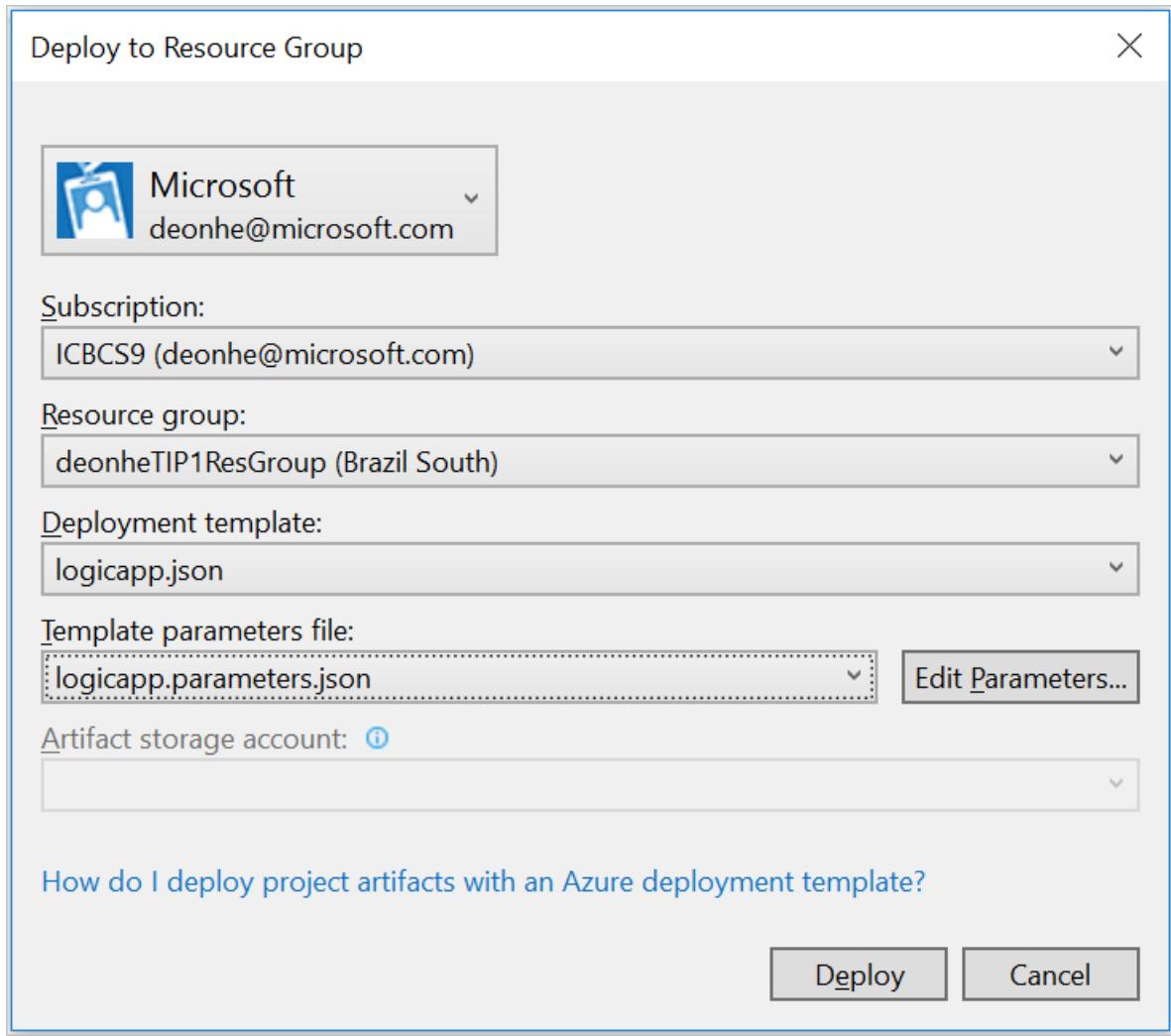
Deploying your Logic app

Finally, after you have configured your app, you can deploy directly from Visual Studio in just a couple steps.

1. Right-click on the project in the Solution Explorer and go to **Deploy > New Deployment...**



2. You are prompted to sign in to your Azure subscription(s).
3. Now you need to choose the details of the resource group that you want to deploy the Logic app to.



NOTE

Be sure to select the right template and parameters files for the resource group (for example if you are deploying to a production environment you'll want to choose the production parameters file).

4. Select the Deploy button
5. The status of the deployment appears in the **Output** window (you may need to choose **Azure Provisioning**.

```
Output: Azure Provisioning
Show output from: Azure Provisioning
05:43:57 - Transfer summary:
05:43:57 - -----
05:43:57 - Total files transferred: 3
05:43:57 - Transfer successfully: 3
05:43:57 - Transfer failed: 0
05:43:57 - Elapsed time: 00:00:00:02
05:43:57 - Done building target "UploadDrop" in project "MyLogicApp.Deployment.deployproj".
05:43:57 - Done building project "MyLogicApp.Deployment.deployproj".
05:43:57 - Build succeeded.
05:43:57 - The following parameter values will be used for this deployment:
05:43:57 -   dropLocation: https://deploymentlogs.blob.core.windows.net/webapplication2
05:43:57 -   dropLocationSasToken: <securestring>
05:43:57 -   webSitePackage: MyLogicApp/package.zip
05:43:57 -   webSiteName: blogordns
05:43:57 -   webSiteHostingPlanName: noentuhoneuth
05:43:57 -   webSiteLocation: westus
05:43:57 -   webSiteHostingPlanSKU: Free
05:43:57 -   webSiteHostingPlanWorkerSize: 0
05:43:57 - Starting deployment. This may take a while...
05:43:57 - Uploading template to Azure storage account 'deploymentlogs'.
05:43:58 - Creating resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10' in location 'westus'.
05:44:12 - Creating deployment 'VS_deploy' in resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10'.
```

In the future, you can revise your Logic app in source control and use Visual Studio to deploy new versions.

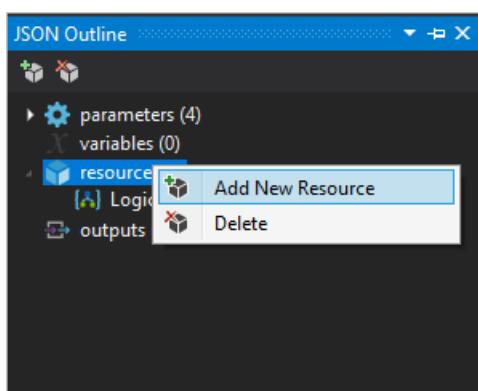
NOTE

If you modify the definition in the Azure portal directly, then the next time you deploy from Visual Studio those changes will be overwritten.

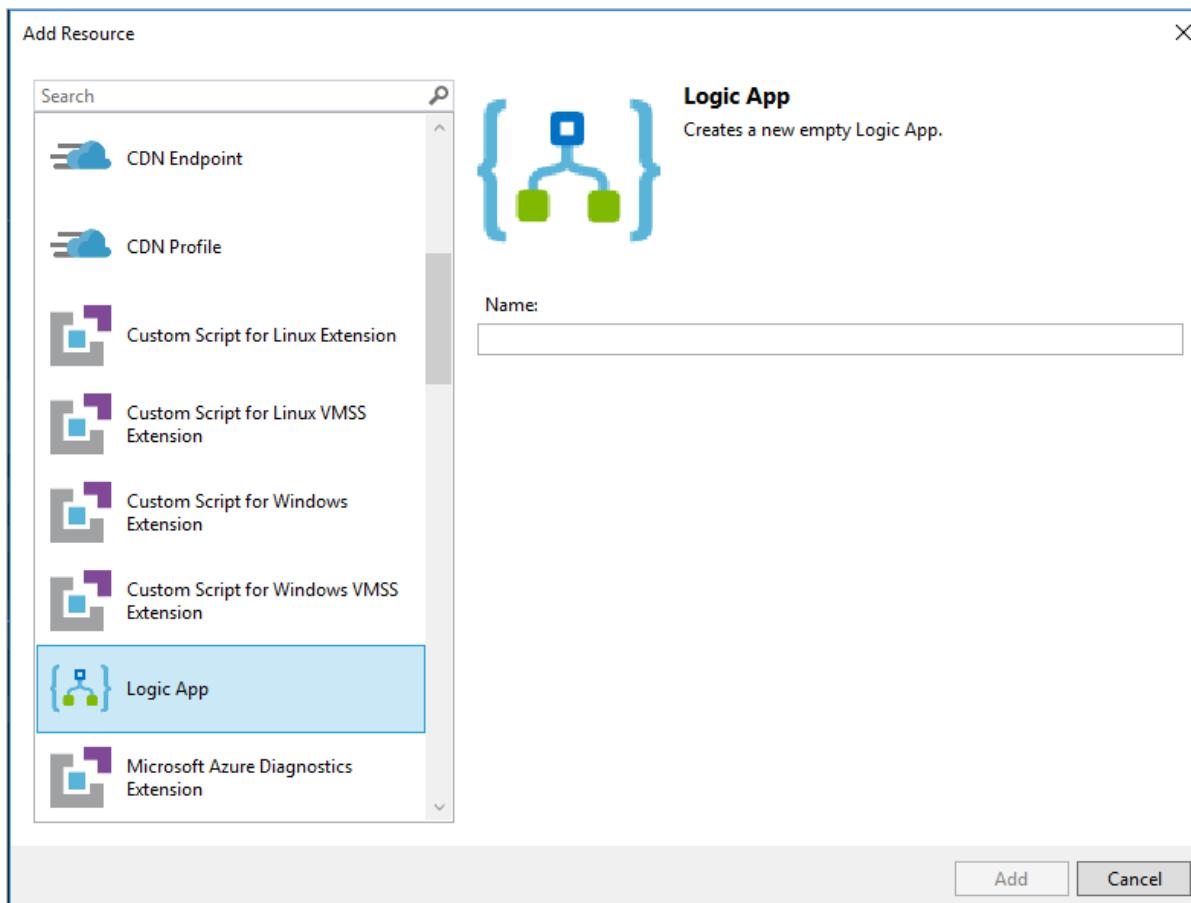
Adding a Logic App to an existing Resource Group project

If you have an existing Resource Group project, then adding a logic app to it, or adding another logic app along side the one you previously created, can be done through the JSON Outline window.

1. Open the `<template>.json` file.
2. Open the JSON Outline window. The JSON Outline window can be found under **View > Other Windows > JSON Outline**.
3. To add a resource to the template file, either click the Add Resource button on the top of the JSON Outline window or right-click on **resources** and select **Add New Resource**.



4. In the **Add Resource** dialog box browse and select **Logic App**, give it a name and select **Add**.



Next Steps

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- [You can automate business processes with Logic Apps](#)
- [Learn How to Integrate your systems with Logic Apps](#)

Using Azure Functions with Logic Apps

1/20/2017 • 2 min to read • [Edit on GitHub](#)

You can run custom snippets of C# or node.js by using Azure Functions from within a logic app. [Azure Functions](#) offers server-free computing in Microsoft Azure. This is useful for performing the following tasks:

- Advanced formatting or compute of fields within a Logic App
- Performing calculations within a workflow
- Extending the functionality of Logic Apps with functions that are supported in C# or node.js

Create a function for Logic Apps

We recommend that you create a new function in the Azure Functions portal by using the **Generic Webhook - Node** or **Generic Webhook - C#** templates. This auto-populates a template that accepts `application/json` from a logic app. If you select the **Integrate** tab in Azure Functions it should have **Mode** set to **Webhook** and **Webhook type** of **Generic JSON**. Functions that use these templates are automatically discovered and listed in the Logic Apps designer under **Azure Functions in my region**.

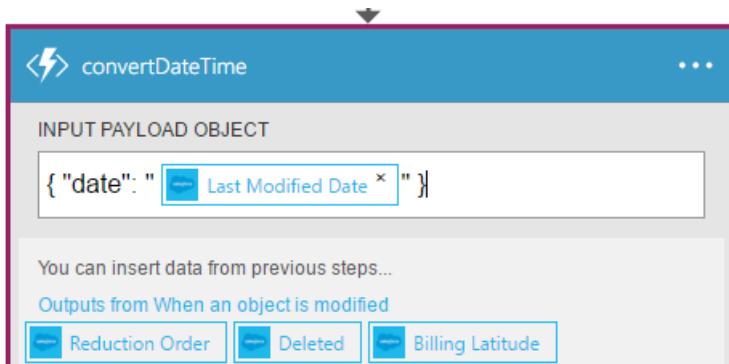
Webhook functions accept a request and pass it into the method via a `data` variable. You can access the properties of your payload by using dot notation like `data.foo`. For example, a simple JavaScript function that converts a DateTime value into a date string looks like the following example:

```
function start(req, res){  
    var data = req.body;  
    res = {  
        body: data.date.ToString();  
    }  
}
```

Call Azure Functions from a logic app

In the designer, if you click the **Actions** menu, you can select **Azure Functions in my Region**. This lists the containers in your subscription and enables you to choose the function that you want to call.

After you select the function, you are prompted to specify an input payload object. This is the message that the logic app sends to the function, and it must be a JSON object. For example, if you want to pass in the **Last Modified** date from a Salesforce trigger, the function payload might look like this:



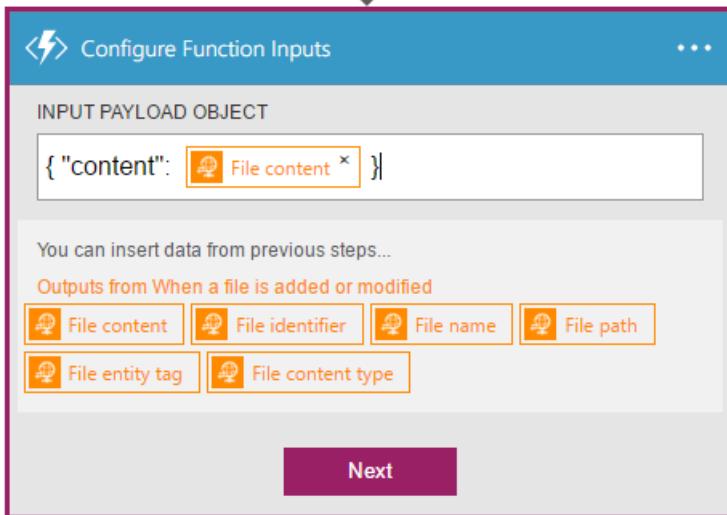
Trigger logic apps from a function

It's also possible to trigger a logic app from within a function. To do this, simply create a logic app with a manual trigger. For more information, see [Logic apps as callable endpoints](#). Then, within your function, generate an HTTP POST to the manual trigger URL with the payload that you want to send to the logic app.

Create a function from the designer

You can also create a nodejs webhook function from within the designer. First, select **Azure Functions in my Region**, and then choose a container for your function. If you don't yet have a container, you need to create one from the [Azure Functions portal](#). Then select **Create New**.

To generate a template based on the data that you want to compute, specify the context object that you plan to pass into a function. This must be a JSON object. For example, if you pass in the file content from an FTP action, the context payload will look like this:



NOTE

Because this object wasn't cast as a string, the content will be added directly to the JSON payload. However, it will error out if it is not a JSON token (that is, a string or a JSON object/array). To cast it as a string, simply add quotes as shown in the first illustration in this article.

The designer then generates a function template that you can create inline. Variables are pre-created based on the context that you plan to pass into the function.

Logic apps as callable endpoints

1/20/2017 • 4 min to read • [Edit on GitHub](#)

Logic Apps natively can expose a synchronous HTTP endpoint as a trigger. You can also use the pattern of callable endpoints to invoke Logic Apps as a nested workflow through the "workflow" action in a Logic App.

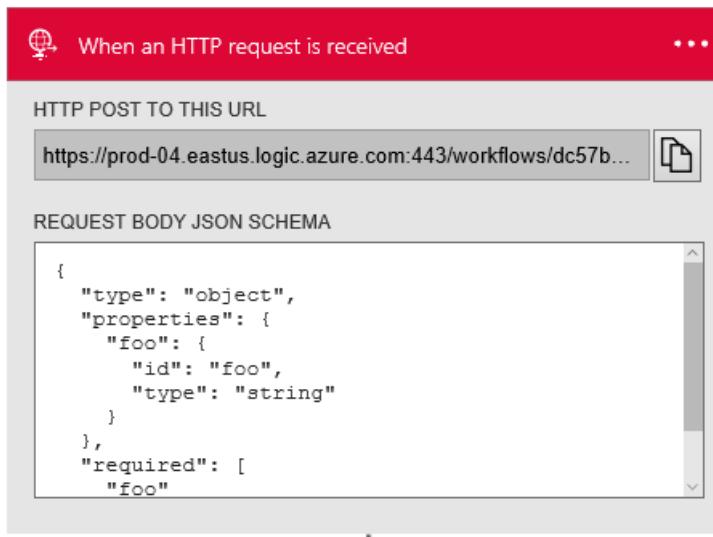
There are 3 types of triggers that can receive requests:

- Request
- ApiConnectionWebhook
- HttpWebhook

For the remainder of the article, we will use **request** as the example, but all of the principles apply identically to the other 2 types of triggers.

Adding a trigger to your definition

The first step is to add a trigger to your Logic app definition that can receive incoming requests. You can search in the designer for "HTTP Request" to add the trigger card. You can define a request body JSON Schema and the designer will generate tokens to help you parse and pass data from the manual trigger through the workflow. I recommend using a tool like [jsonschema.net](#) to generate a JSON schema from a sample body payload.



After you save your Logic App definition, a callback URL will be generated similar to this one:

```
https://prod-
03.eastus.logic.azure.com:443/workflows/080cb66c52ea4e9cabe0abf4e197deff/triggers/myendpointtrigger?
*signature*...
```

This URL contains a SAS key in the query parameters used for authentication.

You can also get this endpoint in the Azure portal:

Fired	Start Time	Trigger Name
Fired	4/5/2016, 10:55 AM	manual
Fired	4/5/2016, 10:19 AM	manual
Fired	4/5/2016, 8:35 AM	manual
Fired	4/4/2016, 7:33 PM	manual
Fired	4/4/2016, 5:11 PM	manual
Fired	4/4/2016, 3:42 PM	manual

Or, by calling:

```
POST https://management.azure.com/{resourceID of your logic app}/triggers/myendpointtrigger/listCallbackURL?
api-version=2015-08-01-preview
```

Security for the trigger URL

Logic App callback URLs are generated securely using a Shared Access Signature. The signature is passed through as a query parameter, and must be validated before the logic app will fire. It is generated through a unique combination of a secret key per logic app, the trigger name, and the operation being performed. Unless someone has access to the secret logic app key, they would not be able to generate a valid signature.

Calling the Logic app trigger's endpoint

Once you have created the endpoint for your trigger, you can trigger it via a `POST` to the full URL. You can include additional headers, and any content in the body.

If the content-type is `application/json` then you will be able to reference properties from inside the request. Otherwise, it will be treated as a single binary unit that can be passed to other APIs but cannot be referenced inside the workflow without converting the content. For example, if you pass `application/xml` content you could use `@xpath()` to do an xpath extraction, or `@json()` to convert from XML to JSON. More information on working with content types [can be found here](#)

In addition, you can specify a JSON schema in the definition. This causes the designer to generate tokens that you can then pass into steps. For example the following will make a `title` and `name` token available in the designer:

```
{
  "properties": {
    "title": {
      "type": "string"
    },
    "name": {
      "type": "string"
    }
  },
  "required": [
    "title",
    "name"
  ],
  "type": "object"
}
```

Referencing the content of the incoming request

The `@triggerOutputs()` function will output the contents of the incoming request. For example, it would look like:

```
{
  "headers" : {
    "content-type" : "application/json"
  },
  "body" : {
    "myprop" : "a value"
  }
}
```

You can use the `@triggerBody()` shortcut to access the `body` property specifically.

Responding to the request

For some requests that start a Logic app, you may want to respond with some content to the caller. There is a new action type called **response** that can be used to construct the status code, body and headers for your response. Note that if no **response** shape is present, the Logic app endpoint will *immediately* respond with **202 Accepted**.

The screenshot shows the configuration pane for a 'Response' action. It includes fields for STATUS CODE*, HEADERS, and BODY.

- STATUS CODE***: A dropdown menu currently set to '200'.
- HEADERS**: A JSON object containing:


```
{
        "content-type": "application/json"
      }
```
- BODY**: A JSON object containing two properties: 'name' and 'title'. Both properties have their values highlighted with red boxes and the placeholder text 'name' and 'title' respectively.

```

"Response": {
    "conditions": [],
    "inputs": {
        "body": {
            "name": "@{triggerBody()['name']}",
            "title": "@{triggerBody()['title']}"
        },
        "headers": {
            "content-type": "application/json"
        },
        "statusCode": 200
    },
    "type": "Response"
}

```

Responses have the following:

PROPERTY	DESCRIPTION
statusCode	The HTTP status code to respond to the incoming request. It can be any valid status code that starts with 2xx, 4xx, or 5xx. 3xx status codes are not permitted.
body	A body object that can be a string, a JSON object, or even binary content referenced from a previous step.
headers	You can define any number of headers to be included in the response

All of the steps in the Logic app that are required for the response must complete within *60 seconds* for the original request to receive the response **unless the workflow is being called as a nested Logic App**. If no response action is reached within 60 seconds then the incoming request will time out and receive a **408 Client timeout** HTTP response. For nested Logic Apps, the parent Logic App will continue to wait for a response until completed, regardless of the amount of time it takes.

Advanced endpoint configuration

Logic apps have built in support for the direct access endpoint and always use the **POST** method to start a run of the Logic app. The **HTTP Listener** API app previously also supported changing the URL segments and the HTTP method. You could even set up additional security or a custom domain by adding it to the API app host (the Web app that hosted the API app).

This functionality is available through **API management**:

- [Change the method of the request](#)
- [Change the URL segments of the request](#)
- Set up your API management domains on the **Configure** tab in the classic Azure portal
- Set up policy to check for Basic authentication (**link needed**)

Summary of migration from 2014-12-01-preview

2014-12-01-PREVIEW	2016-06-01
Click on HTTP Listener API app	Click on Manual trigger (no API app required)

2014-12-01-PREVIEW	2016-06-01
HTTP Listener setting " <i>Sends response automatically</i> "	Either include a response action or not in the workflow definition
Configure basic or OAuth authentication	via API management
Configure HTTP method	via API management
Configure relative path	via API management
Reference the incoming body via <code>@triggerOutputs().body.Content</code>	Reference via <code>@triggerOutputs().body</code>
Send HTTP response action on the HTTP Listener	Click on Respond to HTTP request (no API app required)

Logic Apps Loops, Scopes, and Debatching

1/20/2017 • 3 min to read • [Edit on GitHub](#)

Logic Apps provides a number of ways to work with arrays, collections, batches, and loops within a workflow.

ForEach loop and arrays

Logic Apps allows you to loop over a set of data and perform an action for each item. This is possible via the `foreach` action. In the designer, you can specify to add a for each loop. After selecting the array you wish to iterate over, you can begin adding actions. Currently you are limited to only one action per foreach loop, but this restriction will be lifted in the coming weeks. Once within the loop you can begin to specify what should occur at each value of the array.

If using code-view, you can specify a for each loop like below. This is an example of a for each loop that sends an email for each email address that contains 'microsoft.com':

```
{
  "email_filter": {
    "type": "query",
    "inputs": {
      "from": "@triggerBody()['emails']",
      "where": "@contains(item()['email'], 'microsoft.com')"
    }
  },
  "forEach_email": {
    "type": "foreach",
    "foreach": "@body('email_filter')",
    "actions": {
      "send_email": {
        "type": "ApiConnection",
        "inputs": {
          "body": {
            "to": "@item()",
            "from": "me@contoso.com",
            "message": "Hello, thank you for ordering"
          },
          "host": {
            "connection": {
              "id": "@parameters('$connections')['office365']['connection']['id']"
            }
          }
        }
      }
    }
  },
  "runAfter": {
    "email_filter": [ "Succeeded" ]
  }
}
```

A `foreach` action can iterate over arrays up to 5,000 rows. Each iteration will execute in parallel by default.

Sequential ForEach loops

To enable a foreach loop to execute sequentially, the `Sequential` operation option should be added.

```

"forEach_email": {
    "type": "foreach",
    "foreach": "@body('email_filter')",
    "operationOptions": "Sequential",
    "..."
}

```

Until loop

You can perform an action or series of actions until a condition is met. The most common scenario for this is calling an endpoint until you get the response you are looking for. In the designer, you can specify to add an until loop. After adding actions inside the loop, you can set the exit condition, as well as the loop limits. There is a 1 minute delay between loop cycles.

If using code-view, you can specify an until loop like below. This is an example of calling an HTTP endpoint until the response body has the value 'Completed'. It will complete when either

- HTTP Response has status of 'Completed'
- It has tried for 1 hour
- It has looped 100 times

```

{
    "until_successful": {
        "type": "until",
        "expression": "@equals(actions('http')['status'], 'Completed')",
        "limit": {
            "count": 100,
            "timeout": "PT1H"
        },
        "actions": {
            "create_resource": {
                "type": "http",
                "inputs": {
                    "url": "http://provisionRseource.com",
                    "body": {
                        "resourceId": "@triggerBody()"
                    }
                }
            }
        }
    }
}

```

SplitOn and debatching

Sometimes a trigger may receive an array of items that you want to debatch and start a workflow per item. This can be accomplished via the `spliton` command. By default, if your trigger swagger specifies a payload that is an array, a `spliton` will be added and start a run per item. SplitOn can only be added to a trigger. This can be manually configured or overridden in definition code-view. Currently SplitOn can debatch arrays up to 5,000 items. You cannot have a `spliton` and also implement the synchronous response pattern. Any workflow called that has a `response` action in addition to `spliton` will run asynchronously and send an immediate `202 Accepted` response.

SplitOn can be specified in code-view as the following example. This receives an array of items and debatches on each row.

```
{  
    "myDebatchTrigger": {  
        "type": "Http",  
        "inputs": {  
            "url": "http://getNewCustomers",  
        },  
        "recurrence": {  
            "frequency": "Second",  
            "interval": 15  
        },  
        "splitOn": "@triggerBody()['rows']"  
    }  
}
```

Scopes

It is possible to group a series of actions together using a scope. This is particularly useful for implementing exception handling. In the designer you can add a new scope, and begin adding any actions inside of it. You can define scopes in code-view like the following:

```
{  
    "myScope": {  
        "type": "scope",  
        "actions": {  
            "call_bing": {  
                "type": "http",  
                "inputs": {  
                    "url": "http://www.bing.com"  
                }  
            }  
        }  
    }  
}
```

Using your custom API hosted on App Service with Logic apps

1/20/2017 • 7 min to read • [Edit on GitHub](#)

Although Logic apps has a rich set of 40+ connectors for a variety of services, you may want to call into your own custom API that can run your own code. One of the easiest and most scalable ways to host your own custom web API's is to use App Service. This article covers how to call into any web API hosted in an App Service API app, web app or mobile app.

For information on building APIs as a trigger or action within Logic apps, check out [this article](#).

Deploy your Web App

First, you'll need to deploy your API as a Web App in App Service. The instructions here cover basic deployment: [Create an ASP.NET web app](#). While you can call into any API from a Logic app, for the best experience we recommend you add Swagger metadata to integrate easily with Logic apps actions. You can find details on [adding swagger](#).

API Settings

In order for the Logic apps designer to parse your Swagger, it's important that you enable CORS and set the APIDefinition properties of your web app. This is very easy to set within the Azure Portal. Simply open the settings blade of your Web App, and under the API section set the 'API Definition' to the URL of your swagger.json file (this is usually [https://\(name\).azurewebsites.net/swagger/docs/v1](https://(name).azurewebsites.net/swagger/docs/v1)), and add a CORS policy for '*' to allow for requests from the Logic apps Designer.

Calling into the API

When within the Logic apps portal, if you have set CORS and the API Definition properties you should be able to easily add Custom API actions within your flow. In the designer you can select to browse your subscription websites to list the websites with a swagger URL defined. You can also use the HTTP + Swagger action to point to a swagger and list available actions and inputs. Finally, you can always create a request using the HTTP action to call any API, even those that do not have or expose a swagger doc.

If you want to secure your API, then there are a couple different ways to do that:

1. No code change required - Azure Active Directory can be used to protect your API without requiring any code changes or redeployment.
2. Enforce Basic Auth, AAD Auth, or Certificate Auth in the code of your API.

Securing calls to your API without a code change

In this section, you'll create two Azure Active Directory applications – one for your Logic app and one for your Web App. You'll authenticate calls to your Web App using the service principal (client id and secret) associated with the AAD application for your Logic app. Finally, you'll include the application ID's in your Logic app definition.

Part 1: Setting up an Application identity for your Logic app

This is what the Logic app uses to authenticate against active directory. You only *need* to do this once for your directory. For example, you can choose to use the same identity for all of your Logic apps, although you may also create unique identities per Logic app if you wish. You can either do this in the UI or use PowerShell.

[Create the application identity using the Azure classic portal](#)

1. Navigate to [Active directory in the Azure classic portal](#) and select the directory that you use for your Web App
2. Click the **Applications** tab
3. Click **Add** in the command bar at the bottom of the page
4. Give your identity a Name to use, click the next arrow
5. Put in a unique string formatted as a domain in the two text boxes, and click the checkmark
6. Click the **Configure** tab for this application
7. Copy the **Client ID**, this will be used in your Logic app
8. In the **Keys** section click **Select duration** and choose either 1 year or 2 years
9. Click the **Save** button at the bottom of the screen (you may have to wait a few seconds)
10. Now be sure to copy the key in the box. This will also be used in your Logic app

Create the application identity using PowerShell

```
1. Switch-AzureMode AzureResourceManager
2. Add-AzureAccount
New-AzureADApplication -DisplayName "MyLogicAppID" -HomePage "http://somerandomdomain.tld" -IdentifierUris
3. "http://somerandomdomain.tld" -Password "Pass@word1!"
```

4. Be sure to copy the **Tenant ID**, the **Application ID** and the password you used

Part 2: Protect your Web App with an AAD app identity

If your Web app is already deployed you can just enable it in the portal. Otherwise, you can make enabling Authorization part of your Azure Resource manager deployment.

Enable Authorization in the Azure Portal

1. Navigate to the Web app and click the **Settings** in the command bar.
2. Click **Authorization/Authentication**.
3. Turn it **On**.

At this point, an Application is automatically created for you. You need this Application's Client ID for Part 3, so you'll need to:

1. Go to [Active directory in the Azure classic portal](#) and select your directory.
2. Search for the app in the search box
3. Click on it in the list
4. Click on the **Configure** tab
5. You should see the **Client ID**

Deploying your Web App using an ARM template

First, you need to create an application for your Web app. This should be different from the application that is used for your Logic app. Start by following the steps above in Part 1, but now for the **HomePage** and **IdentifierUris** use the actual <https://URL> of your Web app.

NOTE

When you create the Application for your Web app, you must use the [Azure classic portal approach](#), as the PowerShell commandlet does not set up the required permissions to sign users into a website.

Once you have the client ID and tenant ID, include the following as a sub resource of the Web app in your deployment template:

```

"resources" : [
    {
        "apiVersion" : "2015-08-01",
        "name" : "web",
        "type" : "config",
        "dependsOn" : [
            "[concat('Microsoft.Web/sites/', 'parameters('webAppName'))]"
        ],
        "properties" : {
            "siteAuthEnabled": true,
            "siteAuthSettings": {
                "clientId": "<<clientID>>",
                "issuer": "https://sts.windows.net/<<tenantID>>/",
            }
        }
    }
]

```

To run a deployment automatically that deploys a blank Web app and Logic app together that use AAD, click the following button:



For the complete template, see [Logic app calls into a Custom API hosted on App Service and protected by AAD](#).

Part 3: Populate the Authorization section in the Logic app

In the **Authorization** section of the **HTTP** action:

```
{"tenant": "<<tenantId>>", "audience": "<<clientID from Part 2>>", "clientId": "<<clientID from Part 1>>","secret": "<<Password or Key from Part 1>>","type": "ActiveDirectoryOAuth" }
```

ELEMENT	DESCRIPTION
type	Type of authentication. For ActiveDirectoryOAuth authentication, the value is ActiveDirectoryOAuth.
tenant	The tenant identifier used to identify the AD tenant.
audience	Required. The resource you are connecting to.
clientID	The client identifier for the Azure AD application.
secret	Required. Secret of the client that is requesting the token.

The above template already has this set up, but if you are authoring the Logic app directly, you'll need to include the full authorization section.

Securing your API in code

Certificate auth

You can use Client certificates to validate the incoming requests to your Web app. See [How To Configure TLS Mutual Authentication for Web App](#) for how to set up your code.

In the *Authorization* section you should provide:

```
{"type": "clientcertificate", "password": "test", "pfx": "long-pfx-key"} .
```

ELEMENT	DESCRIPTION
type	Required. Type of authentication. For SSL client certificates, the value must be ClientCertificate.
pfx	Required. Base64-encoded contents of the PFX file.
password	Required. Password to access the PFX file.

Basic auth

You can use Basic authentication (e.g. username and password) to validate the incoming requests. Basic auth is a common pattern and you can do it in any language you build your app in.

In the *Authorization* section, you should provide: `{"type": "basic", "username": "test", "password": "test"}`.

ELEMENT	DESCRIPTION
type	Required. Type of authentication. For Basic authentication, the value must be Basic.
username	Required. Username to authenticate.
password	Required. Password to authenticate.

Handle AAD auth in code

By default, the Azure Active Directory authentication that you enable in the Portal does not do fine-grained authorization. For example, it does not lock your API to a specific user or app, but just to a particular tenant.

If you want to restrict the API to just the Logic app, for example, in code, you can extract the header which contains the JWT and check who the caller is, rejecting any requests that do not match.

Going further, if you want to implement it entirely in your own code, and not leverage the Portal feature, you can read this article: [Use Active Directory for authentication in Azure App Service](#).

You still need to follow the above steps to create an Application identity for your Logic app and use that to call the API.

Logic Apps Error and Exception Handling

1/20/2017 • 6 min to read • [Edit on GitHub](#)

Logic Apps provides a rich set of tools and patterns to help ensure your integrations are robust and resilient against failures. One of the challenges with any integration architecture is ensuring that downtime or issues of dependent systems are handled appropriately. Logic Apps makes handling errors a first class experience, giving you the tools you need to act on exceptions and errors within your workflows.

Retry policies

The most basic type of exception and error handling is a retry-policy. This policy defines if the action should retry if initial request timed out or failed (any request that resulted in a 429 or 5xx response). By default, all actions retry 4 additional times over 20-second intervals. So if the first request received a `500 Internal Server Error` response, the workflow engine pauses for 20 seconds, and attempt the request again. If after all retries the response is still an exception or failure, the workflow will continue and mark the action status as `Failed`.

You can configure retry policies in the **inputs** of a particular action. A retry-policy can be configured to try as many as 4 times over 1 hour intervals. Full details on the input properties can be [found on MSDN](#).

```
"retryPolicy" : {  
    "type": "<type-of-retry-policy>",  
    "interval": <retry-interval>,  
    "count": <number-of-retry-attempts>  
}
```

If you wanted your HTTP action to retry 4 times and wait 10 minutes between each attempt you would have the following definition:

```
"HTTP":  
{  
    "inputs": {  
        "method": "GET",  
        "uri": "http://myAPIendpoint/api/action",  
        "retryPolicy" : {  
            "type": "fixed",  
            "interval": "PT10M",  
            "count": 4  
        }  
    },  
    "runAfter": {},  
    "type": "Http"  
}
```

For more details on supported syntax, view the [retry-policy section on MSDN](#).

RunAfter property to catch failures

Each logic app action declares which actions need to complete before the action will start. You can think of this as the ordering of steps in your workflow. This ordering is known as the `runAfter` property in the action definition. It is an object that describes which actions and action statuses would execute the action. By default, all actions added through the designer are set to `runAfter` the previous step if the previous step was `Succeeded`. However, you can customize this value to fire actions when previous actions are `Failed`, `Skipped`, or a possible set of these values. If

you wanted to add an item to a designated Service Bus topic after a specific action `Insert_Row` fails, you would use the following `runAfter` configuration:

```
"Send_message": {
    "inputs": {
        "body": {
            "ContentData": "@{encodeBase64(body('Insert_Row'))}",
            "ContentType": "{ \"content-type\" : \"application/json\" }"
        },
        "host": {
            "api": {
                "runtimeUrl": "https://logic-apis-westus.azure-apim.net/apim/servicebus"
            },
            "connection": {
                "name": "@parameters('$connections')['servicebus']['connectionId']"
            }
        },
        "method": "post",
        "path": "/@{encodeURIComponent('failures')}/messages"
    },
    "runAfter": {
        "Insert_Row": [
            "Failed"
        ]
    }
}
```

Notice the `runAfter` property is set to fire if the `Insert_Row` action is `Failed`. To run the action if the action status is `Succeeded`, `Failed`, or `Skipped` the syntax would be:

```
"runAfter": {
    "Insert_Row": [
        "Failed", "Succeeded", "Skipped"
    ]
}
```

TIP

Actions that run after a preceding action have failed, and complete successfully, will be marked as `Succeeded`. This behavior means if you successfully catch all failures in a workflow the run itself is marked as `Succeeded`.

Scopes and results to evaluate actions

Similar to how you can run after individual actions, you can also group actions together inside a `scope` - which act as a logical grouping of actions. Scopes are useful both for organizing your logic app actions, and for performing aggregate evaluations on the status of a scope. The scope itself will receive a status after all the actions within a scope have completed. The scope status is determined with the same criteria as a run -- if the final action in an execution branch is `Failed` or `Aborted` the status is `Failed`.

You can `runAfter` a scope has been marked `Failed` to fire specific actions for any failures that occurred within the scope. Running after a scope fails allows you to create a single action to catch failures if *any* actions within the scope fail.

Getting the context of failures with results

Catching failures from a scope is very useful, but you may also want the context to understand exactly which actions failed, and any errors or status codes that were returned. The `@result()` workflow function provides context into the result of all actions within a scope.

`@result()` takes a single parameter, scope name, and returns an array of all the action results from within that scope. These action objects include the same attributes as the `@actions()` object, including action start time, action end time, action status, action inputs, action correlation IDs, and action outputs. You can easily pair an `@result()` function with a `runAfter` to send context of any actions that failed within a scope.

If you want to execute an action *for each* action in a scope that `Failed`, you can pair `@result()` with a **Filter Array** action and a **ForEach** loop. This allows you to filter the array of results to actions that failed. You can take the filtered result array and perform an action for each failure using the **ForEach** loop. Here's an example below, followed by a detailed explanation. This example will send an HTTP POST request with the response body of any actions that failed within the scope `My_Scope`.

```
"Filter_array": {
    "inputs": {
        "from": "@result('My_Scope')",
        "where": "@equals(item()['status'], 'Failed')"
    },
    "runAfter": {
        "My_Scope": [
            "Failed"
        ]
    },
    "type": "Query"
},
"For_each": {
    "actions": {
        "Log_Exception": {
            "inputs": {
                "body": "@item()['outputs']['body']",
                "method": "POST",
                "headers": {
                    "x-failed-action-name": "@item()['name']",
                    "x-failed-tracking-id": "@item()['clientTrackingId']"
                },
                "uri": "http://requestb.in/"
            },
            "runAfter": {},
            "type": "Http"
        }
    },
    "foreach": "@body('Filter_array')",
    "runAfter": {
        "Filter_array": [
            "Succeeded"
        ]
    },
    "type": "Foreach"
}
```

Here's a detailed walkthrough of what's happening:

1. **Filter Array** action to filter the `@result('My_Scope')` to get the result of all actions within `My_Scope`
2. Condition of the **Filter Array** is any `@result()` item with the status equal to `Failed`. This will filter the array of all action results from `My_Scope` to only an array of failed action results.
3. Perform a **For Each** action on the **Filtered Array** outputs. This will perform an action *for each* failed action result we filtered above.
 - If there was a single action in the scope that failed, the actions in the `foreach` would only run once. Many failed actions would cause one action per failure.
4. Send an HTTP POST on the `foreach` item response body, or `@item()['outputs']['body']`. The `@result()` item shape is the same as the `@actions()` shape, and can be parsed the same way.
5. Also included two custom headers with the failed action name `@item()['name']` and the failed run client

tracking ID `@item()['clientTrackingId']`.

For reference, here is an example of a single `@result()` item. You can see the `name`, `body`, and `clientTrackingId` properties parsed in the example above. It should be noted that outside of a `foreach`, `@result()` returns an array of these objects.

```
{  
    "name": "Example_Action_That_Failed",  
    "inputs": {  
        "uri": "https://myfailedaction.azurewebsites.net",  
        "method": "POST"  
    },  
    "outputs": {  
        "statusCode": 404,  
        "headers": {  
            "Date": "Thu, 11 Aug 2016 03:18:18 GMT",  
            "Server": "Microsoft-IIS/8.0",  
            "X-Powered-By": "ASP.NET",  
            "Content-Length": "68",  
            "Content-Type": "application/json"  
        },  
        "body": {  
            "code": "ResourceNotFound",  
            "message": "/docs/foo/bar does not exist"  
        }  
    },  
    "startTime": "2016-08-11T03:18:19.7755341Z",  
    "endTime": "2016-08-11T03:18:20.2598835Z",  
    "trackingId": "bdd82e28-ba2c-4160-a700-e3a8f1a38e22",  
    "clientTrackingId": "08587307213861835591296330354",  
    "code": "NotFound",  
    "status": "Failed"  
}
```

You can use the expressions above to perform different exception handling patterns. You may choose to execute a single exception handling action outside the scope that accepts the entire filtered array of failures and remove the `foreach`. You can also include other useful properties from the `@result()` response shown above.

Azure Diagnostics and telemetry

The patterns above are great way to handle errors and exceptions within a run, but you can also identify and respond to errors independent of the run itself. [Azure Diagnostics](#) provides a simple way to send all workflow events (including all run and action statuses) to an Azure Storage account or an Azure Event Hub. You can monitor the logs and metrics, or publish them into any monitoring tool you prefer, to evaluate run statuses. One potential option is to stream all the events through Azure Event Hub into [Stream Analytics](#). In Stream Analytics you can write live queries off of any anomalies, averages, or failures from the diagnostic logs. Stream Analytics can easily output to other data sources like queues, topics, SQL, DocumentDB, and Power BI.

Next Steps

- [See how one customer built robust error handling with Logic Apps](#)
- [Find more Logic Apps examples and scenarios](#)
- [Learn how to create automated deployments of logic apps](#)
- [Design and deploy logic apps from Visual Studio](#)

Logging and error handling in Logic Apps

1/20/2017 • 7 min to read • [Edit on GitHub](#)

This article describes how you can extend a logic app to better support exception handling. It is a real-life use case and our answer to the question of, "Does Logic Apps support exception and error handling?"

NOTE

The current Logic Apps schema provides a standard template for action responses. This includes both internal validation and error responses returned from an API app.

Overview of the use case and scenario

The following story is the use case for this article. A well-known healthcare organization engaged us to develop an Azure solution that would create a patient portal by using Microsoft Dynamics CRM Online. They needed to send appointment records between the Dynamics CRM Online patient portal and Salesforce. We were asked to use the [HL7 FHIR](#) standard for all patient records.

The project had two major requirements:

- A method to log records sent from the Dynamics CRM Online portal
- A way to view any errors that occurred within the workflow

How we solved the problem

TIP

You can view a high-level video of the project at the [Integration User Group](#).

We chose [Azure DocumentDB](#) as a repository for the log and error records (DocumentDB refers to records as documents). Because Logic Apps has a standard template for all responses, we would not have to create a custom schema. We could create an API app to **Insert** and **Query** for both error and log records. We could also define a schema for each within the API app.

Another requirement was to purge records after a certain date. DocumentDB has a property called [Time to Live](#) (TTL), which allowed us to set a **Time to Live** value for each record or collection. This eliminated the need to manually delete records in DocumentDB.

Creation of the logic app

The first step is to create the logic app and load it in the designer. In this example, we are using parent-child logic apps. Let's assume that we have already created the parent and are going to create one child logic app.

Because we are going to be logging the record coming out of Dynamics CRM Online, let's start at the top. We need to use a Request trigger because the parent logic app triggers this child.

IMPORTANT

To complete this tutorial, you will need to create a DocumentDB database and two collections (Logging and Errors).

Logic app trigger

We are using a Request trigger as shown in the following example.

```
"triggers": {  
    "request": {  
        "type": "request",  
        "kind": "http",  
        "inputs": {  
            "schema": {  
                "properties": {  
                    "CRMId": {  
                        "type": "string"  
                    },  
                    "recordType": {  
                        "type": "string"  
                    },  
                    "salesforceID": {  
                        "type": "string"  
                    },  
                    "update": {  
                        "type": "boolean"  
                    }  
                },  
                "required": [  
                    "CRMId",  
                    "recordType",  
                    "salesforceID",  
                    "update"  
                ],  
                "type": "object"  
            }  
        }  
    }  
},
```

Steps

We need to log the source (request) of the patient record from the Dynamics CRM Online portal.

1. We need to get a new appointment record from Dynamics CRM Online. The trigger coming from CRM provides us with the **CRM PatientId**, **record type**, **New or Updated Record** (new or update Boolean value), and **SalesforceId**. The **SalesforceId** can be null because it's only used for an update. We will get the CRM record by using the CRM **PatientID** and the **Record Type**.
2. Next, we need to add our DocumentDB API app **InsertLogEntry** operation as shown in the following figures.

Insert log entry designer view

InsertLogEntry

PRESCRIBERID
CRMID

OPERATION
New_Patient

You can insert data from previous steps...

Outputs from manual

Body CRMID recordType salesforceID
update

SOURCE
Headers

SALESFORCEID
salesforceID

DATE
Date

...

[Insert error entry designer view](#)

CreateErrorRecord

Enter a valid integer

STATUSCODE

```
actions('Create_Appointment')['outputs']['statusCode']
```

MESSAGE

```
actions('Create_Appointment')['outputs']['body']['message']
```

SOURCE

```
@{concat(triggerBody()['description'],
  ' START: ', triggerBody()['start'],
  ' END: ', triggerBody()['end'],
  ' COMMENT: ', triggerBody()['comment']) }
```

ACTION

```
Create_Appointment
```

ERRORS

RESOLVED

0

NOTES

ISERROR

true

PATIENTID

```
@{replace(triggerBody()['participant'][0]['actor']['display'],' ','')}
```

SEVERITY

4

...

Check for create record failure

Condition

CONDITION

```
Equals(actions('Create_ErrorRecord')['status'], 'Failed')
```

If yes

Add an action

If no, do nothing

Add an action

CreateErrorRecord

Logic app source code

NOTE

The following are samples only. Because this tutorial is based on an implementation currently in production, the value of a **Source Node** might not display properties that are related to scheduling an appointment.

Logging

The following logic app code sample shows how to handle logging.

Log entry

This is the logic app source code for inserting a log entry.

```
"InsertLogEntry": {
    "metadata": {
        "apiDefinitionUrl": "https://.../swagger/docs/v1",
        "swaggerSource": "website"
    },
    "type": "Http",
    "inputs": {
        "body": {
            "date": "@{outputs('Gets_NewPatientRecord')['headers']['Date']}",
            "operation": "New Patient",
            "patientId": "@{triggerBody()['CRMID']}",
            "providerId": "@{triggerBody()['providerID']}",
            "source": "@{outputs('Gets_NewPatientRecord')['headers']}"
        },
        "method": "post",
        "uri": "https://.../api/Log"
    },
    "runAfter": {
        "Gets_NewPatientRecord": ["Succeeded"]
    }
}
```

Log request

This is the log request message posted to the API app.

```
{
    "uri": "https://.../api/Log",
    "method": "post",
    "body": {
        "date": "Fri, 10 Jun 2016 22:31:56 GMT",
        "operation": "New Patient",
        "patientId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
        "providerId": "",
        "source": "{\"Pragma\":\"no-cache\", \"x-ms-request-id\": \"e750c9a9-bd48-44c4-bbba-1688b6f8a132\", \"OData-Version\": \"4.0\", \"Cache-Control\": \"no-cache\", \"Date\": \"Fri, 10 Jun 2016 22:31:56 GMT\", \"Set-Cookie\": \"ARRAffinity=785f4334b5e64d2db0b84edcc1b84f1bf37319679aefce206b51510e56fd9770; Path=/; Domain=127.0.0.1\", \"Server\": \"Microsoft-IIS/8.0, Microsoft-HTTPAPI/2.0\", \"X-AspNet-Version\": \"4.0.30319\", \"X-Powered-By\": \"ASP.NET\", \"Content-Length\": \"1935\", \"Content-Type\": \"application/json; odata.metadata=minimal; odata.streaming=true\", \"Expires\": \"-1\"}"
    }
}
```

Log response

This is the log response message from the API app.

```
{
  "statusCode": 200,
  "headers": {
    "Pragma": "no-cache",
    "Cache-Control": "no-cache",
    "Date": "Fri, 10 Jun 2016 22:32:17 GMT",
    "Server": "Microsoft-IIS/8.0",
    "X-AspNet-Version": "4.0.30319",
    "X-Powered-By": "ASP.NET",
    "Content-Length": "964",
    "Content-Type": "application/json; charset=utf-8",
    "Expires": "-1"
  },
  "body": {
    "ttl": 2592000,
    "id": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0_1465597937",
    "_rid": "XngRAOT6IQEHAAAAAAA==",
    "_self": "dbs/XngRAA==/colls/XngRAOT6IQE=/docs/XngRAOT6IQEHAAAAAAA==/",
    "_ts": 1465597936,
    "_etag": "\"0400fc2f-0000-0000-0000-575b3ff00000\"",
    "patientID": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
    "timestamp": "2016-06-10T22:31:56Z",
    "source": "{\"Pragma\":\"no-cache\",\"x-ms-request-id\":\"e750c9a9-bd48-44c4-bbba-1688b6f8a132\",\"OData-Version\":\"4.0\",\"Cache-Control\":\"no-cache\",\"Date\":\"Fri, 10 Jun 2016 22:31:56 GMT\",\"Set-Cookie\":\"ARRAffinity=785f4334b5e64d2db0b84edcc1b84f1bf37319679aefce206b51510e56fd9770;Path=/;Domain=127.0.0.1\",\"Server\":\"Microsoft-IIS/8.0,Microsoft-HTTPAPI/2.0\",\"X-AspNet-Version\":\"4.0.30319\",\"X-Powered-By\":\"ASP.NET\",\"Content-Length\":\"1935\",\"Content-Type\":\"application/json; odata.metadata=minimal;odata.streaming=true\",\"Expires\":\"-1\"}",
    "operation": "New Patient",
    "salesforceId": "",
    "expired": false
  }
}
```

Now let's look at the error handling steps.

Error handling

The following Logic Apps code sample shows how you can implement error handling.

Create error record

This is the Logic Apps source code for creating an error record.

```

"actions": {
    "CreateErrorRecord": {
        "metadata": {
            "apiDefinitionUrl": "https://.../swagger/docs/v1",
            "swaggerSource": "website"
        },
        "type": "Http",
        "inputs": {
            "body": {
                "action": "New_Patient",
                "isError": true,
                "crmId": "@{triggerBody()['CRMID']}",
                "patientID": "@{triggerBody()['CRMID']}",
                "message": "@{body('Create_NewPatientRecord')['message']}",
                "providerId": "@{triggerBody()['providerId']}",
                "severity": 4,
                "source": "@{actions('Create_NewPatientRecord')['inputs']['body']}",
                "statusCode": "@{int(outputs('Create_NewPatientRecord')['statusCode'])}",
                "salesforceId": "",
                "update": false
            },
            "method": "post",
            "uri": "https://.../api/CrMtoSfError"
        },
        "runAfter": {
            {
                "Create_NewPatientRecord": ["Failed"]
            }
        }
    }
}

```

Insert error into DocumentDB--request

```
{
    "uri": "https://.../api/CrMtoSfError",
    "method": "post",
    "body": {
        "action": "New_Patient",
        "isError": true,
        "crmId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
        "patientId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
        "message": "Salesforce failed to complete task: Message: duplicate value found: Account_ID_MED__c duplicates value on record with id: 001U0000001c83gK",
        "providerId": "",
        "severity": 4,
        "salesforceId": "",
        "update": false,
        "source": ""

        {"/Account_Class_vod_c/": "/PRAC/", "/Account_Status_MED_c/": "/I/", "/CRM_HUB_ID_c/": "/6b115f6d-a7ee-e511-80f5-3863bb2eb2d0/", "/Credentials_vod_c/", "/DTC_ID_MED_c/": "", "/Fax/": "", "/FirstName/": "/A/", "/Gender_vod_c/": "", "/IMS_ID_c/": "", "/LastName/": "/BAILEY/", "/MasterID_mp_c/": "", "/C_ID_MED_c/": "/851588/", "/Middle_vod_c/": "", "/NPI_vod_c/": "", "/PDRP_MED_c/": false, "/PersonDoNotCall/": false, "/PersonEmail/": "", "/PersonHasOptedOutOfEmail/": false, "/PersonHasOptedOutOfFax/": false, "/PersonMobilePhone/": "", "/Phone/": "", "/Practicing_Specialty_c/": "/FM - FAMILY MEDICINE/", "/Primary_City_c/": "", "/Primary_State_c/": "", "/Primary_Street_Line2_c/": "", "/Primary_Street_c/": "", "/Primary_Zip_c/": "", "/RecordTypeId/": "/012U0000000JaPWIA0/", "/Request_Date_c/": "/2016-06-10T22:31:55.9647467Z/", "/ONY_ID_c/": "", "/Specialty_1_vod_c/": "", "/Suffix_vod_c/": "", "/Website/": ""},
        "statusCode": "400"
    }
}
```

Insert error into DocumentDB--response

```
{
  "statusCode": 200,
  "headers": {
    "Pragma": "no-cache",
    "Cache-Control": "no-cache",
    "Date": "Fri, 10 Jun 2016 22:31:57 GMT",
    "Server": "Microsoft-IIS/8.0",
    "X-AspNet-Version": "4.0.30319",
    "X-Powered-By": "ASP.NET",
    "Content-Length": "1561",
    "Content-Type": "application/json; charset=utf-8",
    "Expires": "-1"
  },
  "body": {
    "id": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0-1465597917",
    "_rid": "sQx2APhVzAA8AAAAAAA==",
    "_self": "dbs/sQx2AA=/colls/sQx2APhVzAA=/docs/sQx2APhVzAA8AAAAAAA==/",
    "_ts": 1465597912,
    "_etag": "\"0c00eaac-0000-0000-0000-575b3fdc0000\"",
    "prescriberId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
    "timestamp": "2016-06-10T22:31:57.3651027Z",
    "action": "New_Patient",
    "salesforceId": "",
    "update": false,
    "body": "CRM failed to complete task: Message: duplicate value found: CRM_HUB_ID__c duplicates value on record with id: 001U000001c83gK",
    "source": "
{/\"Account_Class_vod_c/":"/PRAC/", /\"Account_Status_MED_c/":"/I/", /\"CRM_HUB_ID_c/":"/6b115f6d-a7ee-e511-80f5-3863bb2eb2d0/", /\"Credentials_vod_c/":"/DO - Degree level is DO", /\"DTC_ID_MED_c/":"/", /\"Fax/":"/", /\"FirstName/":"/A/", /\"Gender_vod_c/":"/", /\"IMS_ID_c/":"/", /\"LastName/":"/BAILEY", /\"MterID_mp_c/":"/", /\"Medicis_ID_MED_c/":"/851588", /\"Middle_vod_c/":"/", /\"NPI_vod_c/": "/", /\"PDRP_MED_c/":false, /\"PersonDoNotCall/":false, /\"PersonEmail/":"/", /\"PersonHasOptedOutOfEmail/":false, /\"PersonHasOptedOutOffax/":false, /\"PersonMobilePhone/":"/", /\"Phone/":"/", /\"Practicing_Specialty_c/":"/FM - FAMILY MEDICINE", /\"Primary_City_c/":"/", /\"Primary_State_c/":"/", /\"Primary_Street_Line2_c/":"/", /\"Primary_Street_c/":"/", /\"Primary_Zip_c/":"/", /\"RecordTypeId/":"/012U0000000JaPWIA0", /\"Request_Date_c/":"/2016-06-10T22:31:55.9647467Z", /\"XXXXXX/":"/", /\"Specialty_1_vod_c/":"/", /\"Suffix_vod_c/":"/", /\"Website/":"/"}",
    "code": 400,
    "errors": null,
    "isError": true,
    "severity": 4,
    "notes": null,
    "resolved": 0
  }
}
```

Salesforce error response

```
{
  "statusCode": 400,
  "headers": {
    "Pragma": "no-cache",
    "x-ms-request-id": "3e8e4884-288e-4633-972c-8271b2cc912c",
    "X-Content-Type-Options": "nosniff",
    "Cache-Control": "no-cache",
    "Date": "Fri, 10 Jun 2016 22:31:56 GMT",
    "Set-Cookie":
      "ARRAffinity=785f4334b5e64d2db0b84edcc1b84f1bf37319679aefce206b51510e56fd9770;Path=/;Domain=127.0.0.1",
    "Server": "Microsoft-IIS/8.0,Microsoft-HTTPAPI/2.0",
    "X-AspNet-Version": "4.0.30319",
    "X-Powered-By": "ASP.NET",
    "Content-Length": "205",
    "Content-Type": "application/json; charset=utf-8",
    "Expires": "-1"
  },
  "body": {
    "status": 400,
    "message": "Salesforce failed to complete task: Message: duplicate value found: Account_ID_MED__C duplicates value on record with id: 001U0000001c83gK",
    "source": "Salesforce.Common",
    "errors": []
  }
}
```

Returning the response back to the parent logic app

After you have the response, you can pass it back to the parent logic app.

Return success response to the parent logic app

```
"SuccessResponse": {
  "runAfter": {
    "UpdateNew_CRMPatientResponse": ["Succeeded"]
  },
  "inputs": {
    "body": {
      "status": "Success"
    },
    "headers": {
      "Content-type": "application/json",
      "x-ms-date": "@utcnow()"
    },
    "statusCode": 200
  },
  "type": "Response"
}
```

Return error response to the parent logic app

```

"ErrorResponse": {
    "runAfter": {
        "Create_NewPatientRecord": [
            "Failed"
        ],
        "inputs": {
            "body": {
                "status": "BadRequest"
            },
            "headers": {
                "Content-type": "application/json",
                "x-ms-date": "@utcnow()"
            },
            "statusCode": 400
        },
        "type": "Response"
    }
}

```

DocumentDB repository and portal

Our solution added additional capabilities with [DocumentDB](#).

Error management portal

To view the errors, you can create an MVC web app to display the error records from DocumentDB. **List**, **Details**, **Edit**, and **Delete** operations are included in the current version.

NOTE

Edit operation: DocumentDB does a replace of the entire document. The records shown in the **List** and **Detail** views are samples only. They are not actual patient appointment records.

Following are examples of our MVC app details created with the previously described approach.

Error management list

The screenshot shows a browser window with the title 'CRM to SF Error Management'. The page has a dark header bar with the title. Below it is a table titled 'List of Errors' with columns: PrescriberId, Action, TimeStamp, Body, and Resolved. There are seven rows of data listed in the table.

PrescriberId	Action	TimeStamp	Body	Resolved
ce1820b0-ee2c-e611-80e7-5065f38a5ba1	Create_SFaddress	6/8/2016 4:16:50 PM	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu	No
ce1820b0-ee2c-e611-80e7-5065f38a5ba1	Create_SFaddress	6/8/2016 4:16:58 PM	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu	No
4433c660-ac2d-e611-80e7-c4346bdc0271	Create_SFcallPlan	6/8/2016 7:23:00 PM	Salesforce failed to complete task: Message: duplicate value found: External_ID_MED__c duplicates value on record with id: a2pm0000000TFWg	No
12c3f133-b02d-e611-80e7-c4346bdc0271	Update Decile	6/8/2016 7:36:48 PM	Salesforce failed to complete task: Message: Unable to create/update fields: Account_MED__c. Please check the security settings of this field and verify that it is read/write for your profile or permission set.	No
8ea10937-fe2e-e611-80e9-5065f38a3ba1	Update Prescriber	6/10/2016 11:32:30 AM	Salesforce failed to complete task: Message: Provided external ID field does not exist or is not accessible: 001m000000X8lmoAAF123	No
8ea10937-fe2e-e611-80e9-5065f38a3ba1	Update_ATL	6/10/2016 11:32:32 AM	Salesforce failed to complete task: Message: Account: id value of incorrect type: 001m000000X8lmoAAF123	No

List of Errors

PrescriberId	Action	TimeStamp	Body	Resolved	
ce1820b0-ee2c-e611-80e7-5065f38a5ba1	Create_SFaddress	6/8/2016 4:16:50 PM	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu	No	Details
ce1820b0-ee2c-e611-80e7-5065f38a5ba1	Create_SFaddress	6/8/2016 4:16:58 PM	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu	No	Details
4433c660-ac2d-e611-80e7-c4346bdc0271	Create_SFcallPlan	6/8/2016 7:23:00 PM	Salesforce failed to complete task: Message: duplicate value found: External_ID_MED__c duplicates value on record with id: a2pm0000000TFWg	No	Details
12c3f133-b02d-e611-80e7-c4346bdc0271	Update Decile	6/8/2016 7:36:48 PM	Salesforce failed to complete task: Message: Unable to create/update fields: Account_MED__c. Please check the security settings of this field and verify that it is read/write for your profile or permission set.	No	Details
8ea10937-fe2e-e611-80e9-5065f38a3ba1	Update Prescriber	6/10/2016 11:32:30 AM	Salesforce failed to complete task: Message: Provided external ID field does not exist or is not accessible: 001m000000X8lmoAAF123	No	Details
8ea10937-fe2e-e611-80e9-5065f38a3ba1	Update_ATL	6/10/2016 11:32:32 AM	Salesforce failed to complete task: Message: Account: id value of incorrect type: 001m000000X8lmoAAF123	No	Details

Error management detail view

View

Error Response

TimeStamp	6/8/2016 4:16:50 PM
Code	400
Body	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu
Source	{"Account_vod__c":"001m000000X74NAAAZ","Address_Type_MED__c":"Mailing","Address_line_2_vod__c":"test str 2","CRM_Hub_Id__c":"ce1820b0-ee2c-e611-80e7-5065f38a5ba1","City_vod__c":"edison","Country_vod__c":"United States","Fax_vod__c":"","License_Expiration_Date_vod__c":"2016-06-30","License_State_MED__c":"NJ","License_Status_vod__c":"Valid_vod","License_vod__c":"342198","Name":"test str 1","Phone_vod__c":"","Primary_vod__c":"false","SLX_Address_Line_3__c":"","State_vod__c":"NJ","Zip_vod__c":"435465"}
Action	Create_SFaddress
Notes	
IsError	<input checked="" type="checkbox"/>
PrescriberId	ce1820b0-ee2c-e611-80e7-5065f38a5ba1

[Back to List](#)

© 2016 - VNBConsulting, Inc

Log management portal

To view the logs, we also created an MVC web app. Following are examples of our MVC app details created with the previously described approach.

Sample log detail view

Properties	
PeterJamesChalmers_1466191912	
_RID	Uw4fAJrEEwEqAAAAAAA==
_TS	Fri, 17 Jun 2016 19:31:50 GMT
_SELF	dbs/Uw4fAA==/colls/Uw4fAJrEEwE=/docs/Uw4fAJrEEwEqAAAAAAA==/_self
_ETAG	"0000dc00-0000-0000-0000-576450290C
_ATTACHMENTS	attachments/

API app details

Logic Apps exception management API

Our open-source Logic Apps exception management API app provides the following functionality.

There are two controllers:

- **ErrorController** inserts an error record (document) in a DocumentDB collection.
- **LogController** Inserts a log record (document) in a DocumentDB collection.

TIP

Both controllers use `async Task<dynamic>` operations. This allows operations to be resolved at runtime, so we can create the DocumentDB schema in the body of the operation.

Every document in DocumentDB must have a unique ID. We are using `PatientId` and adding a timestamp that is converted to a Unix timestamp value (double). We truncate it to remove the fractional value.

You can view the source code of our error controller API [from GitHub](#).

We call the API from a logic app by using the following syntax.

```
"actions": {
    "CreateErrorRecord": {
        "metadata": {
            "apiDefinitionUrl": "https://.../swagger/docs/v1",
            "swaggerSource": "website"
        },
        "type": "Http",
        "inputs": {
            "body": {
                "action": "New_Patient",
                "isError": true,
                "crmId": "@{triggerBody()['CRMID']}",
                "prescriberId": "@{triggerBody()['CRMID']}",
                "message": "@{body('Create_NewPatientRecord')['message']}",
                "salesforceId": "@{triggerBody()['salesforceID']}",
                "severity": 4,
                "source": "@{actions('Create_NewPatientRecord')['inputs']['body']}",
                "statusCode": "@{int(outputs('Create_NewPatientRecord')['statusCode'])}",
                "update": false
            },
            "method": "post",
            "uri": "https://.../api/CrMtoSFError"
        },
        "runAfter": {
            "Create_NewPatientRecord": ["Failed"]
        }
    }
}
```

The expression in the preceding code sample is checking for the *Create_NewPatientRecord* status of **Failed**.

Summary

- You can easily implement logging and error handling in a logic app.
- You can use DocumentDB as the repository for log and error records (documents).
- You can use MVC to create a portal to display log and error records.

Source code

The source code for the Logic Apps exception management API application is available in this [GitHub repository](#).

Next steps

- [View more Logic Apps examples and scenarios](#)
- [Learn about Logic Apps monitoring tools](#)
- [Create a Logic App automated deployment template](#)

Logic Apps Content Type Handling

1/20/2017 • 3 min to read • [Edit on GitHub](#)

There are many different types of content that can flow through a Logic App - including JSON, XML, flat files, and binary data. While all content-types are supported, some are natively understood by the Logic Apps Engine, and others may require casting or conversions as needed. The following article will describe how the engine handles different content-types and how they can be correctly handled as needed.

Content-Type Header

To start simple, let's look at the two `Content-Types` that don't require any conversion or casting to use within a Logic App - `application/json` and `text/plain`.

Application/json

The workflow engine relies on the `Content-Type` header from HTTP calls to determine the appropriate handling. Any request with the content type `application/json` will be stored and handled as a JSON Object. In addition, JSON content can be parsed by default without needing any casting. So a request that has the content-type header `application/json` like this:

```
{  
    "data": "a",  
    "foo": [  
        "bar"  
    ]  
}
```

could be parsed in a workflow with an expression like `@body('myAction')['foo'][0]` to get a value (in this case, `bar`). No additional casting is needed. If you are working with data that is JSON but didn't have a header specified, you can manually cast it to JSON using the `@json()` function (for example: `@json(triggerBody())['foo']`).

Text/plain

Similar to `application/json`, HTTP messages received with the `Content-Type` header of `text/plain` will be stored in its raw form. In addition, if included in a subsequent actions without any casting the request will go out with a `Content-Type : text/plain` header. For example, if working with a flat file you may receive the following HTTP content:

```
Date,Name,Address  
Oct-1,Frank,123 Ave.
```

as `text/plain`. If in the next action you sent it as the body of another request (`@body('flatfile')`), the request would have a `text/plain` Content-Type header. If you are working with data that is plain text but didn't have a header specified, you can manually cast it to text using the `@string()` function (for example:
`@string(triggerBody())`)

Application/xml and Application/octet-stream and Converter Functions

The Logic App Engine will always preserve the `Content-Type` that was received on the HTTP request or response. What this means is if a content is received with `Content-Type` of `application/octet-stream`, including that in a subsequent action with no casting will result in an outgoing request with `Content-Type : application/octet-stream`. In this way the engine can guarantee data will not be lost as it moves throughout the workflow. However, the

action state (inputs and outputs) are stored in a JSON object as it flows throughout the workflow. This means in order to preserve some data-types, the engine will convert the content to a binary base64 encoded string with appropriate metadata that preserves both `$content` and `$content-type` - which will automatically be converted. You can also manually convert between content-types using built in converter functions:

- `@json()` - casts data to `application/json`
- `@xml()` - casts data to `application/xml`
- `@binary()` - casts data to `application/octet-stream`
- `@string()` - casts data to `text/plain`
- `@base64()` - converts content to a base64 string
- `@base64ToString()` - converts a base64 encoded string to `text/plain`
- `@base64toBinary()` - converts a base64 encoded string to `application/octet-stream`
- `@encodeDataUri()` - encodes a string as a dataUri byte array
- `@decodeDataUri()` - decodes a dataUri into a byte array

For example, if you received an HTTP request with `Content-Type : application/xml` of:

```
<?xml version="1.0" encoding="UTF-8" ?>
<CustomerName>Frank</CustomerName>
```

I could cast and use later with something like `@xml(triggerBody())`, or within a function like `@xpath(xml(triggerBody()), '/CustomerName')`.

Other-Content Types

Other content types are supported and will work with a Logic App, but may require manually retrieving the message body by decoding the `$content`. For example, if I were triggering off of a `application/x-www-url-formencoded` request that looked like the following:

```
CustomerName=Frank&Address=123+Avenue
```

since this is not plain text or JSON it will be stored in the action as:

```
...
"body": {
    "$content-type": "application/x-www-url-formencoded",
    "$content": "AAB1241BACDFA=="
}
```

Where `$content` is the payload encoded as a base64 string to preserve all data. Since there currently isn't a native function for form-data, I could still use this data within a workflow by manually accessing the data with a function like `@string(body('formdataAction'))`. If I wanted my outgoing request to also have the `application/x-www-url-formencoded` content-type header, I could just add it to the action body without any casting like `@body('formdataAction')`. However, this will only work if body is the only parameter in the `body` input. If you try to do `@body('formdataAction')` inside of an `application/json` request you will get a runtime error as it will send the encoded body.

Author Logic App definitions

1/20/2017 • 6 min to read • [Edit on GitHub](#)

This topic demonstrates how to use [Azure Logic Apps](#) definitions, which is a simple, declarative JSON language. If you haven't done so yet, check out [how to Create a new Logic app](#) first. You can also read the [full reference material of the definition language on MSDN](#).

Several steps that repeat over a list

You can leverage the [foreach type](#) to repeat over an array of up to 10k items and perform an action for each.

A failure-handling step if something goes wrong

You commonly want to be able to write a *remediation step* — some logic that executes, if, **and only if**, one or more of your calls failed. In this example, we are getting data from a variety of places, but if the call fails, I want to POST a message somewhere so I can track down that failure later:

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
    },
    "triggers": {
        "manual": {
            "type": "manual"
        }
    },
    "actions": {
        "readData": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            }
        },
        "postToErrorMessageQueue": {
            "type": "ApiConnection",
            "inputs": "...",
            "runAfter": {
                "readData": ["Failed"]
            }
        }
    },
    "outputs": {}
}
```

You can make use of the `runAfter` property to specify the `postToErrorMessageQueue` should only run after `readData` is **Failed**. This could also be a list of possible values, so `runAfter` could be `["Succeeded", "Failed"]`.

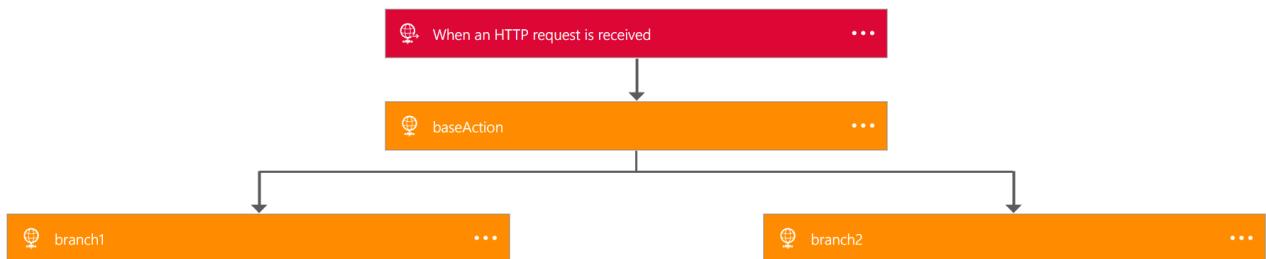
Finally, because you have now handled the error, we no longer mark the run as **Failed**. As you can see here, this run is **Succeeded** even though one step Failed, because I wrote the step to handle this failure.

Two (or more) steps that execute in parallel

To have multiple actions execution in parallel, the `runAfter` property must be equivalent at runtime.

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "triggers": {
        "manual": {
            "type": "manual"
        }
    },
    "actions": {
        "readData": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            }
        },
        "branch1": {
            "type": "Http",
            "inputs": "...",
            "runAfter": {
                "readData": ["Succeeded"]
            }
        },
        "branch2": {
            "type": "Http",
            "inputs": "...",
            "runAfter": {
                "readData": ["Succeeded"]
            }
        }
    },
    "outputs": {}
}
```

As you can see in the example above, both `branch1` and `branch2` are set to run after `readData`. As a result, both of these branches will run in parallel:

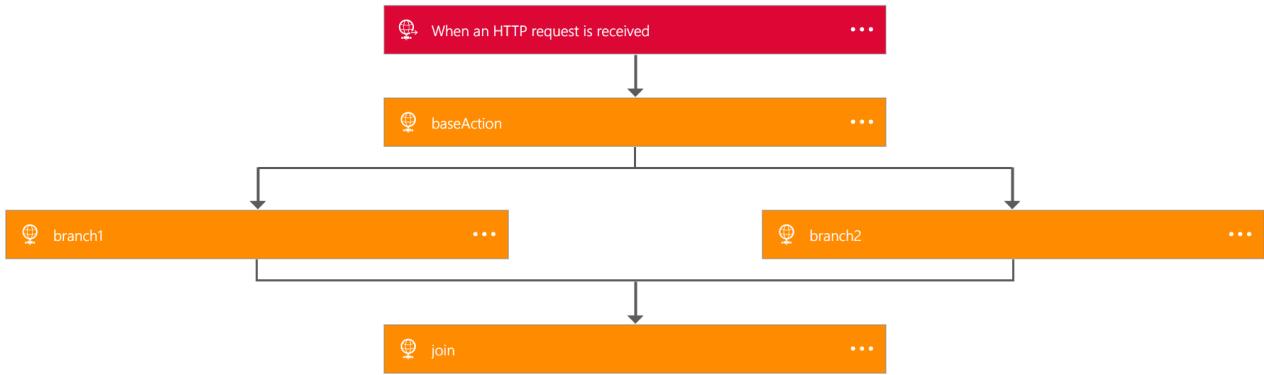


You can see the timestamp for both branches is identical.

Join two parallel branches

You can join two actions that were set to execute in parallel by adding items to the `runAfter` property similar to above.

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-04-01-preview/workflowdefinition.json#",
    "actions": {
        "readData": {
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {},
            "type": "Http"
        },
        "branch1": {
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {
                "readData": [
                    "Succeeded"
                ]
            },
            "type": "Http"
        },
        "branch2": {
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {
                "readData": [
                    "Succeeded"
                ]
            },
            "type": "Http"
        },
        "join": {
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {
                "branch1": [
                    "Succeeded"
                ],
                "branch2": [
                    "Succeeded"
                ]
            },
            "type": "Http"
        }
    },
    "contentVersion": "1.0.0.0",
    "outputs": {},
    "parameters": {},
    "triggers": {
        "manual": {
            "inputs": {
                "schema": {}
            },
            "kind": "Http",
            "type": "Request"
        }
    }
}
```



Mapping items in a list to some different configuration

Next, let's say that we want to get completely different content depending on a value of a property. We can create a map of values to destinations as a parameter:

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "specialCategories": {
            "defaultValue": ["science", "google", "microsoft", "robots", "NSA"],
            "type": "Array"
        },
        "destinationMap": {
            "defaultValue": {
                "science": "http://www.nasa.gov",
                "microsoft": "https://www.microsoft.com/en-us/default.aspx",
                "google": "https://www.google.com",
                "robots": "https://en.wikipedia.org/wiki/Robot",
                "NSA": "https://www.nsa.gov/"
            },
            "type": "Object"
        }
    },
    "triggers": {
        "manual": {
            "type": "manual"
        }
    },
    "actions": {
        "getArticles": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "https://ajax.googleapis.com/ajax/services/feed/load?v=1.0&q=http://feeds.wired.com/wired/index"
            },
            "conditions": []
        },
        "getSpecialPage": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "@parameters('destinationMap')[first(intersection(item().categories, parameters('specialCategories')))]"
            },
            "conditions": [
                {
                    "expression": "@greater(length(intersection(item().categories, parameters('specialCategories'))), 0)"
                }
            ],
            "forEach": "@body('getArticles').responseData.feed.entries"
        }
    }
}
```

In this case, we first get a list of articles, and then the second step looks up in a map, based on the category that was defined as a parameter, which URL to get the content from.

Two items to pay attention here: the `intersection()` function is used to check to see if the category matches one of the known categories defined. Second, once we get the category, we can pull the item of the map using square brackets: `parameters[...]`.

Working with Strings

There are variety of functions that can be used to manipulate string. Let's take an example where we have a string that we want to pass to a system, but we are not confident that character encoding will be handled properly. One option is to base64 encode this string. However, to avoid escaping in a URL we are going to replace a few

characters.

We also want a substring of the the order's name because the first 5 characters are not used.

```
{  
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-  
01/workflowdefinition.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "order": {  
            "defaultValue": {  
                "quantity": 10,  
                "id": "myorder1",  
                "orderer": "NAME=Stèphén_Šicilianö"  
            },  
            "type": "Object"  
        }  
    },  
    "triggers": {  
        "manual": {  
            "type": "manual"  
        }  
    },  
    "actions": {  
        "order": {  
            "type": "Http",  
            "inputs": {  
                "method": "GET",  
                "uri": "http://www.example.com/?  
id=@{replace(replace(base64(substring(parameters('order').orderer,5,sub(length(parameters('order').orderer),  
5 )),'+','-') ,'/','_')?}  
            }  
        },  
        "outputs": {}  
    }  
}
```

Working from the inside out:

1. Get the `length()` of the orderer's name, this returns back the total number of characters
2. Subtract 5 (because we'll want a shorter string)
3. Actually take the `substring()`. We start at index `5` and go the remainder of the string.
4. Convert this substring to a `base64()` string
5. `replace()` all of the `+` characters with `-`
6. `replace()` all of the `/` characters with `_`

Working with Date Times

Date Times can be useful, particularly when you are trying to pull data from a data source that doesn't naturally support **Triggers**. You can also use Date Times to figure out how long various steps are taking.

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "order": {
            "defaultValue": {
                "quantity": 10,
                "id": "myorder1"
            },
            "type": "Object"
        }
    },
    "triggers": {
        "Request": {
            "type": "request",
            "kind": "http"
        }
    },
    "actions": {
        "order": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://www.example.com/?id=@{parameters('order').id}"
            }
        },
        "ifTimingWarning": {
            "type": "If",
            "expression": "@less(actions('order').startTime,addseconds(utcNow(),-1))",
            "actions": {
                "timingWarning": {
                    "type": "Http",
                    "inputs": {
                        "method": "GET",
                        "uri": "http://www.example.com/?recordLongOrderTime=@{parameters('order').id}&currentTime=@{utcNow('r')}"
                    }
                }
            }
        },
        "runAfter": {
            "order": [
                "Succeeded"
            ]
        }
    },
    "outputs": {}
}
```

In this example, we are extracting the `startTime` of the previous step. Then we are getting the current time and subtracting one second : `addseconds(..., -1)` (you could use other units of time such as `minutes` or `hours`). Finally, we can compare these two values. If the first is less than the second, then that means more than one second has elapsed since the order was first placed.

Also note that we can use string formatters to format dates: in the query string I use `utcnow('r')` to get the RFC1123. All date formatting [is documented on MSDN](#).

Using deployment-time parameters for different environments

It is common to have a deployment lifecycle where you have a development environment, a staging environment, and then a production environment. In all of these you may want the same definition, but use different databases, for example. Likewise, you may want to use the same definition across many different regions for high availability,

but want each Logic app instance to talk to that region's database.

Note that this is different from taking different parameters at *runtime*, for that you should use the `trigger()` function as called out above.

You can start with a very simplistic definition like this one:

```
{  
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "uri": {  
            "type": "string"  
        }  
    },  
    "triggers": {  
        "manual": {  
            "type": "manual"  
        }  
    },  
    "actions": {  
        "readData": {  
            "type": "Http",  
            "inputs": {  
                "method": "GET",  
                "uri": "@parameters('uri')"  
            }  
        }  
    },  
    "outputs": {}  
}
```

Then, in the actual `PUT` request for the Logic app you can provide the parameter `uri`. Note, as there is no longer a default value this parameter is required in the Logic app payload:

```
{  
    "properties": {},  
    "definition": {  
        // Use the definition from above here  
    },  
    "parameters": {  
        "connection": {  
            "value": "https://my.connection.that.is.per.enviornment"  
        }  
    },  
    "location": "westus"  
}
```

In each environment you can then provide a different value for the `connection` parameter.

See the [REST API documentation](#) for all of the options you have for creating and managing Logic apps.

Diagnosing logic app failures

1/20/2017 • 3 min to read • [Edit on GitHub](#)

If you experience issues or failures with your logic apps, there are a few approaches can help you better understand where the failures are coming from.

Azure portal tools

The Azure portal provides many tools to diagnose each logic app at each step.

Trigger history

Each logic app has at least one trigger. If you notice that apps aren't firing, the first place to look for additional information is the trigger history. You can access the trigger history on the logic app main blade.

The screenshot shows the Azure Logic App main blade for a logic app named "SalesforceToYammer". On the left, under the "Essentials" section, there's a "Summary" card showing "All runs" and a "Monitoring" card showing "Runs succeeded, failed in the past 3 hours". A red circle labeled "1" points to the "All triggers" link in the summary card. On the right, a modal window titled "When_an_object_is_modified" (SalesforceToYammer - PREVIEW) displays the "Trigger histories" table. A red circle labeled "2" points to the first row in the table, which shows a status of "Fired". The table has columns: FIRED, START TIME, TRIGGER NAME, and CALL STATUS. The data rows are:

FIRED	START TIME	TRIGGER NAME	CALL STATUS
	5/18/2016, 3:48 PM	When_an_object_is...	Skipped
	5/18/2016, 3:48 PM	When_an_object_is...	Skipped
	5/18/2016, 9:53 AM	When_an_object_is...	Skipped
2	Fired	When_an_object_is...	Succeeded
	5/18/2016, 9:53 AM	When_an_object_is...	Succeeded

This lists all of the trigger attempts that your logic app has made. You can click each trigger attempt to get the next level of detail (specifically, any inputs or outputs that the trigger attempt generated). If you see any failed triggers, click the trigger attempt and drill into the **Outputs** link to see any error messages that might have been generated (for example, for invalid FTP credentials).

The different statuses you might see are:

- **Skipped**. It polled the endpoint to check for data and received a response that no data was available.
- **Succeeded**. The trigger received a response that data was available. This could be from a manual trigger, a recurrence trigger, or a polling trigger. This likely will be accompanied with a status of **Fired**, but it might not if you have a condition or SplitOn command in code view that wasn't satisfied.
- **Failed**. An error was generated.

Starting a trigger manually

If you want the logic app to check for an available trigger immediately (without waiting for the next recurrence), you can click **Select Trigger** on the main blade to force a check. For example, clicking this link with a Dropbox trigger will cause the workflow to immediately poll Dropbox for new files.

Run history

Every trigger that is fired results in a run. You can access run information from the main blade, which contains a lot of information that can be helpful in understanding what happened during the workflow.

The screenshot shows two side-by-side views of a Logic App run history. The left view is titled 'All runs' and lists 12 runs from April 18, 2016, to April 25, 2016. The right view is titled 'Logic app run' and shows a detailed timeline from 09:53:37 to 09:53:42. The timeline indicates the status of each action: SUCCEEDED (green), FAILED (red), RUNNING (blue), SKIPPED (grey), and ABORTED (yellow). Below the timeline, specific actions are listed with their status and duration. At the bottom, trigger details are shown.

Start Time	Identifier	Duration
5/18/2016, 9:53 AM	08587380164663239625	2.39 Seconds
4/26/2016, 11:54 AM	08587399100336872469	3 Seconds
4/26/2016, 10:33 AM	08587399148808314289	3.16 Seconds
4/26/2016, 10:30 AM	08587399150394887399	23.89 Seconds
4/26/2016, 10:26 AM	08587399152747876681	2.76 Seconds
4/25/2016, 6:23 PM	08587399730805219007	904 Milliseconds
4/25/2016, 6:17 PM	08587399734133570879	360 Milliseconds
4/25/2016, 6:15 PM	08587399735695368752	2.16 Seconds
4/25/2016, 5:15 PM	08587399771353016691	1.01 Minutes
4/25/2016, 5:15 PM	08587399771511205641	1.04 Minutes

Action	Status	Duration
Execute_stored_procedure	Succeeded	1.49 Seconds
Post_message	Succeeded	687 Milliseconds

TRIGGER

NAME: When_an_object_is_modified (08587380164663239625)
OUTPUTS LINK: <https://flowprod00bl01.blob.core.windows.net/flow...>

A run displays one of the following statuses:

- **Succeeded.** All actions succeeded, or, if there was a failure, it was handled by an action that occurred later in the workflow. That is, it was handled by an action that was set to run after a failed action.
- **Failed.** At least one action had a failure that was not handled by an action later in the workflow.
- **Cancelled.** The workflow was running but received a cancel request.
- **Running.** The workflow is currently running. This may occur for workflows that are being throttled, or because of the current pricing plan. Please see action limits on the [pricing page](#) for details. Configuring diagnostics (the charts listed below the run history) also can provide information about any throttle events that are occurring.

When you are looking at a run history, you can drill in for more details.

Trigger outputs

Trigger outputs show the data that was received from the trigger. This can help you determine whether all properties returned as expected.

NOTE

It might be helpful to understand how the Logic Apps feature [handles different content types](#) if you see any content that you don't understand.

Action	Status	Duration
Execute_stored_procedure	Succeeded	1.49 Seconds
Post_message	Succeeded	687 Milliseconds

Trigger

NAME	When_an_object_is_modified (08587380164663239625)
OUTPUTS LINK	https://flowprod00bl01.blob.core.windows.net/flow...

Action inputs and outputs

You can drill into the inputs and outputs that an action received. This is useful for understanding the size and shape of the outputs, as well as to see any error messages that may have been generated.

The screenshot shows the run history of a Logic App. It includes a table of actions, details about the trigger, and links to inputs and outputs.

ACTION	STATUS	DURATION
Execute_stored_procedure	Succeeded	1.49 Seconds
Post_message	Succeeded	687 Milliseconds

TRIGGER

NAME: When_an_object_is_modified (08587380164663239625)
OUTPUTS LINK: <https://flowprodcu00bl01.blob.core.windows.net/flow...>

DETAILS

START DATE: Wednesday, May 18, 2016, 9:53:39 AM
END DATE: Wednesday, May 18, 2016, 9:53:42 AM

INPUTS

88abf944-f349-4498-918d-4d5b171808a... [\[copy\]](#)

INPUTS LINK
<https://flowprodcu00bl01.blob.core.wind..>

SIZE
360 bytes

OUTPUTS

2 OUTPUTS LINK
<https://flowprodcu00bl01.blob.core.wind..>

Debugging workflow runtime

In addition to monitoring the inputs, outputs, and triggers of a run, it could be useful to add some steps within a workflow to help with debugging. [RequestBin](#) is a powerful tool that you can add as a step in a workflow. By using RequestBin, you can set up an HTTP request inspector to determine the exact size, shape, and format of an HTTP request. You can create a new RequestBin and paste the URL in a logic app HTTP POST action along with body content you want to test (for example, an expression or another step output). After you run the logic app, you can refresh your RequestBin to see how the request was formed as it was generated from the Logic Apps engine.

Logic App limits and configuration

1/20/2017 • 3 min to read • [Edit on GitHub](#)

Below are information on the current limits and configuration details for Azure Logic Apps.

Limits

HTTP request limits

These are limits for a single HTTP request and/or connector call

Timeout

NAME	LIMIT	NOTES
Request Timeout	90 Seconds	An async pattern or until loop can compensate as needed

Message size

NAME	LIMIT	NOTES
Message size	50 MB	Some connectors and APIs may not support 50MB
Expression evaluation limit	131,072 characters	<code>@concat()</code> , <code>@base64()</code> , <code>string</code> cannot be longer than this

Retry policy

NAME	LIMIT	NOTES
Retry attempts	4	Can configure with the retry policy parameter
Retry max delay	1 hour	Can configure with the retry policy parameter
Retry min delay	20 sec	Can configure with the retry policy parameter

Run duration and retention

These are the limits for a single logic app run.

NAME	LIMIT	NOTES
Run duration	90 days	
Storage retention	90 days	This is from the run start time
Min recurrence interval	1 sec	
Max recurrence interval	500 days	

NAME	LIMIT	NOTES
------	-------	-------

Looping and debatching limits

These are limits for a single logic app run.

NAME	LIMIT	NOTES
ForEach items	5,000	You can use the query action to filter larger arrays as needed
Until iterations	5,000	
SplitOn items	5,000	
ForEach Parallelism	20	You can set to a sequential foreach by adding "operationOptions": "Sequential" to the <code>foreach</code> action

Throughput limits

These are limits for a single logic app instance.

NAME	LIMIT	NOTES
Triggers per second	100	Can distribute workflows across multiple apps as needed

Definition limits

These are limits for a single logic app definition.

NAME	LIMIT	NOTES
Actions per workflow	250	You can add nested workflows to extend this as needed
Allowed action nesting depth	5	You can add nested workflows to extend this as needed
Flows per region per subscription	1000	
Triggers per workflow	10	
Max characters per expression	8,192	
Max <code>trackedProperties</code> size in characters	16,000	
<code>action</code> / <code>trigger</code> name limit	80	
<code>description</code> length limit	256	

NAME	LIMIT	NOTES
parameters limit	50	
outputs limit	10	

Integration Account limits

These are limits for artifacts added to integration Account

NAME	LIMIT	NOTES
Schema	8MB	You can use blob URI to upload files larger than 2 MB
Map (XSLT file)	2MB	

B2B protocols (AS2, X12, EDIFACT) message size

These are the limits for B2B protocols

NAME	LIMIT	NOTES
AS2	50MB	Applicable to decode and encode
X12	50MB	Applicable to decode and encode
EDIFACT	50MB	Applicable to decode and encode

Configuration

IP Address

Logic App Service

Calls made from a logic app directly (i.e. via [HTTP](#) or [HTTP + Swagger](#)) or other HTTP requests will come from the IP Address specified below:

LOGIC APP REGION	OUTBOUND IP
Australia East	13.75.153.66, 104.210.89.222, 104.210.89.244, 13.75.149.4, 104.210.91.55, 104.210.90.241
Australia Southeast	13.73.115.153, 40.115.78.70, 40.115.78.237, 13.73.114.207, 13.77.3.139, 13.70.159.205
Brazil South	191.235.86.199, 191.235.95.229, 191.235.94.220, 191.235.82.221, 191.235.91.7, 191.234.182.26
Central India	52.172.157.194, 52.172.184.192, 52.172.191.194, 52.172.154.168, 52.172.186.159, 52.172.185.79
Central US	13.67.236.76, 40.77.111.254, 40.77.31.87, 13.67.236.125, 104.208.25.27, 40.122.170.198

LOGIC APP REGION	OUTBOUND IP
East Asia	168.63.200.173, 13.75.89.159, 23.97.68.172, 13.75.94.173, 40.83.127.19, 52.175.33.254
East US	137.135.106.54, 40.117.99.79, 40.117.100.228, 13.92.98.111, 40.121.91.41, 40.114.82.191
East US 2	40.84.25.234, 40.79.44.7, 40.84.59.136, 40.84.30.147, 104.208.155.200, 104.208.158.174
Japan East	13.71.146.140, 13.78.84.187, 13.78.62.130, 13.71.158.3, 13.73.4.207, 13.71.158.120
Japan West	40.74.140.173, 40.74.81.13, 40.74.85.215, 40.74.140.4, 104.214.137.243, 138.91.26.45
North Central US	168.62.249.81, 157.56.12.202, 65.52.211.164, 168.62.248.37, 157.55.210.61, 157.55.212.238
North Europe	13.79.173.49, 52.169.218.253, 52.169.220.174, 40.113.12.95, 52.178.165.215, 52.178.166.21
South Central US	13.65.98.39, 13.84.41.46, 13.84.43.45, 104.210.144.48, 13.65.82.17, 13.66.52.232
Southeast Asia	52.163.93.214, 52.187.65.81, 52.187.65.155, 13.76.133.155, 52.163.228.93, 52.163.230.166
South India	52.172.9.47, 52.172.49.43, 52.172.51.140, 52.172.50.24, 52.172.55.231, 52.172.52.0
West Europe	13.95.155.53, 52.174.54.218, 52.174.49.6, 40.68.222.65, 40.68.209.23, 13.95.147.65
West India	104.211.164.112, 104.211.165.81, 104.211.164.25, 104.211.164.80, 104.211.162.205, 104.211.164.136
West US	52.160.90.237, 138.91.188.137, 13.91.252.184, 52.160.92.112, 40.118.244.241, 40.118.241.243

Connectors

Calls made from a [connector](#) will come from the IP Address specified below:

LOGIC APP REGION	OUTBOUND IP
Australia East	40.126.251.213
Australia Southeast	40.127.80.34
Brazil South	191.232.38.129
Central India	104.211.98.164
Central US	40.122.49.51

LOGIC APP REGION	OUTBOUND IP
East Asia	23.99.116.181
East US	191.237.41.52
East US 2	104.208.233.100
Japan East	40.115.186.96
Japan West	40.74.130.77
North Central US	65.52.218.230
North Europe	104.45.93.9
South Central US	104.214.70.191
Southeast Asia	13.76.231.68
South India	104.211.227.225
West Europe	40.115.50.13
West India	104.211.161.203
West US	104.40.51.248

Next Steps

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- [You can automate business processes with Logic Apps](#)
- [Learn How to Integrate your systems with Logic Apps](#)

Overview of integration accounts

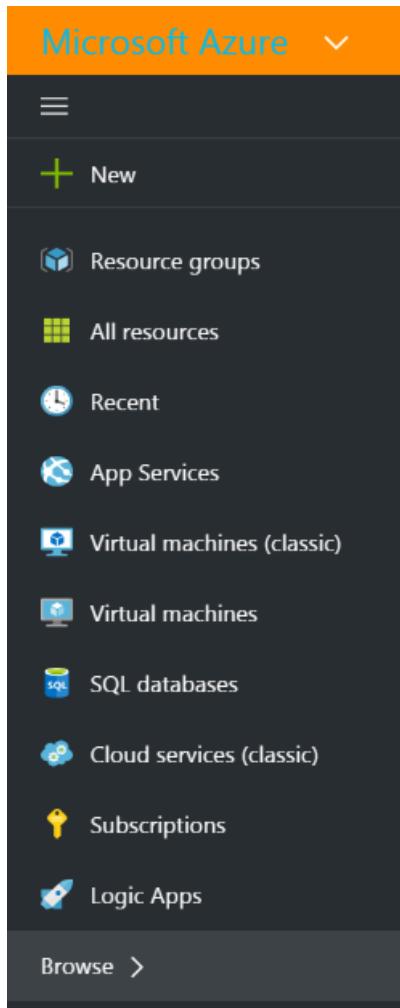
1/20/2017 • 2 min to read • [Edit on GitHub](#)

What is an integration account?

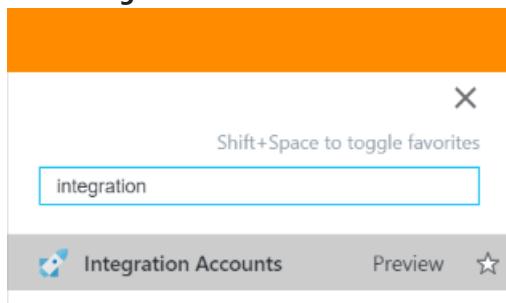
An integration account is an Azure account that allows Enterprise Integration apps to manage artifacts including schemas, maps, certificates, partners and agreements. Any integration app you create will need to use an integration account in order to access a schema, map or certificate, for example.

Create an integration account

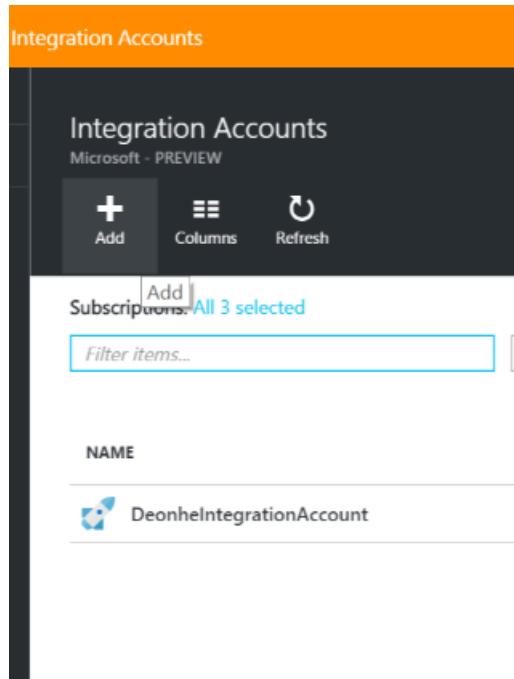
1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list



3. Select *Add* button from the menu at the top of the page



4. Enter the **Name**, select the **Subscription** you want to use, either create a new **Resource group** or select an existing resource group, select a **Location** where your integration account will be hosted, select a **Pricing tier**, then select the **Create** button.

At this point the integration account will be provisioned in the location you selected. This should complete within 1 minute.

Integration Account

PREVIEW

* Name
MyNewIntegrationAccount ✓

* Subscription
ICBCS9

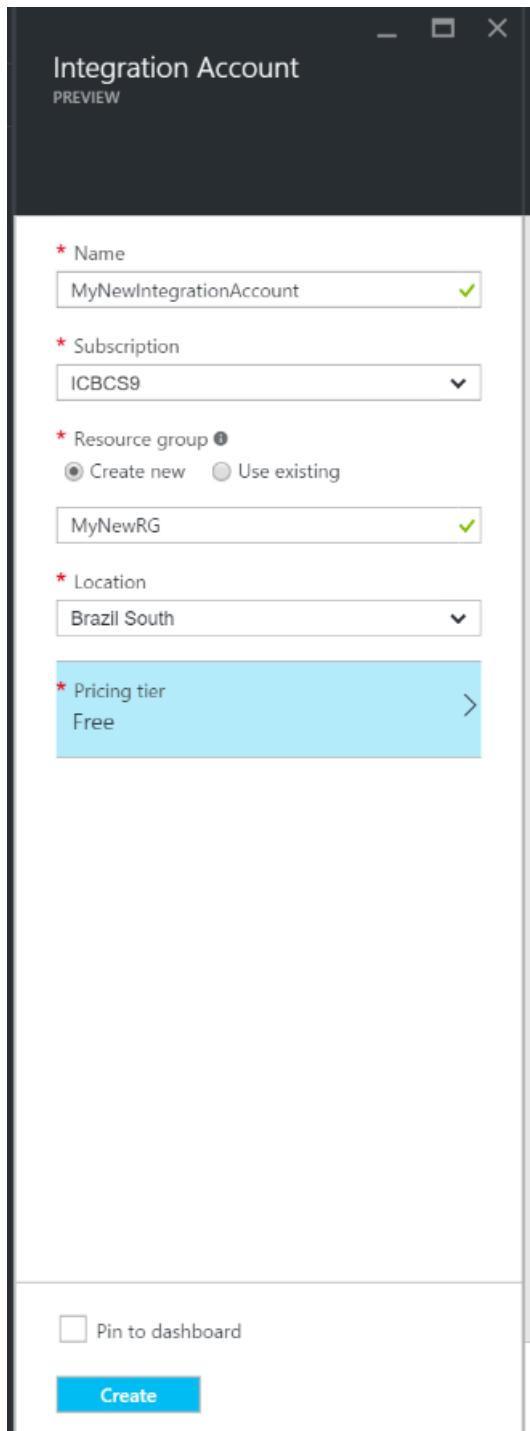
* Resource group ⓘ
Create new Use existing
MyNewRG ✓

* Location
Brazil South

* Pricing tier
Free >

Pin to dashboard

Create



5. Refresh the page. You will see your new integration account listed. Congratulations!

Integration Accounts

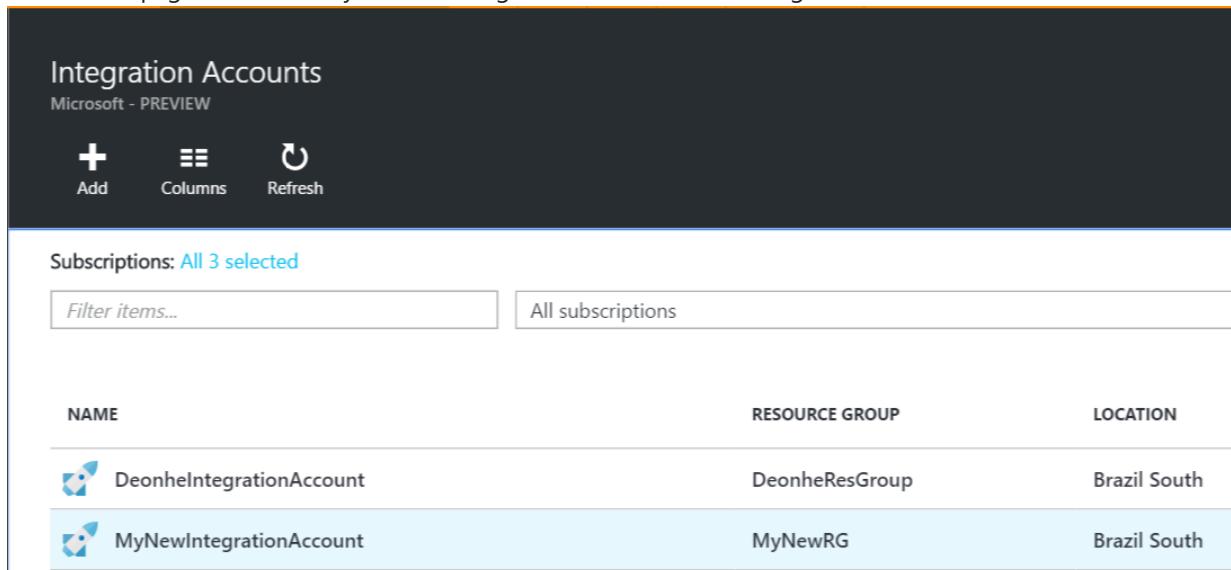
Microsoft - PREVIEW

Add Columns Refresh

Subscriptions: All 3 selected

Filter items... All subscriptions

NAME	RESOURCE GROUP	LOCATION
DeonhelIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South



How to link an integration account to a Logic app

In order for your Logic apps to access to maps, schemas, agreements and other artifacts that are located in your integration account, you must first link the integration account to your Logic app.

Here are the steps to link an integration account to a Logic app

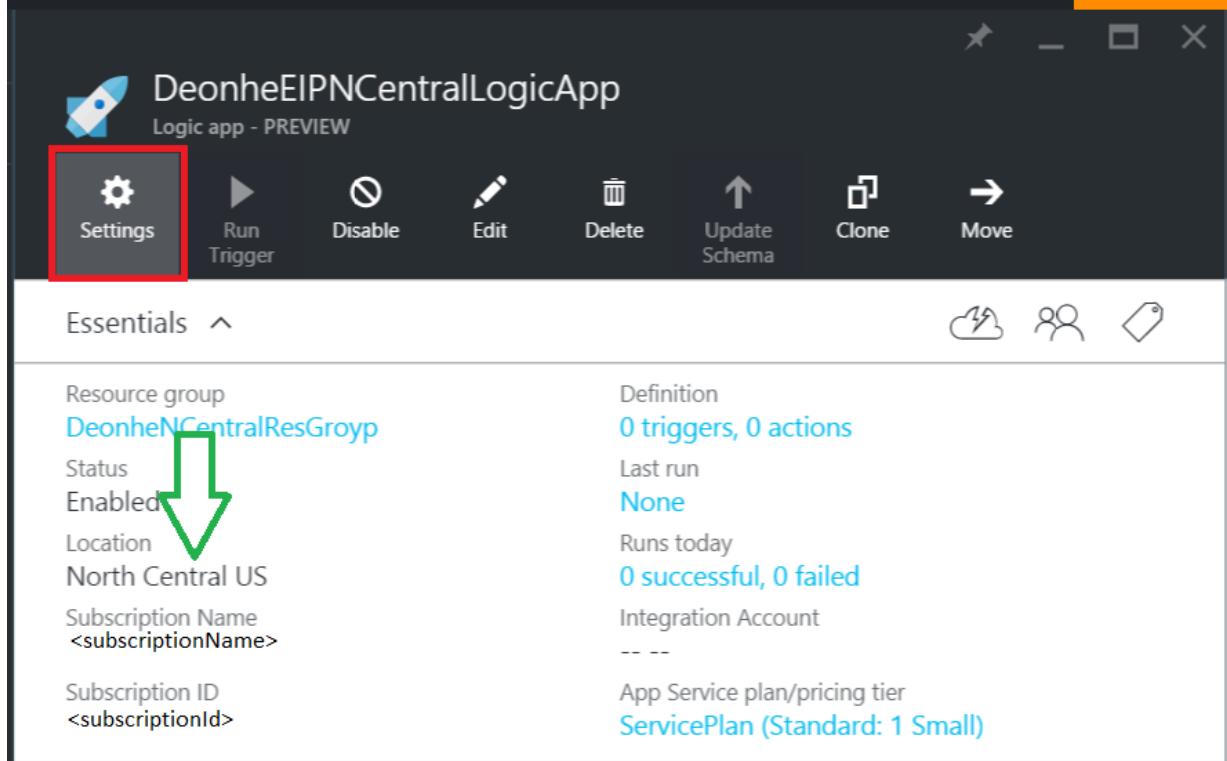
Prerequisites

- An integration account
- A Logic app

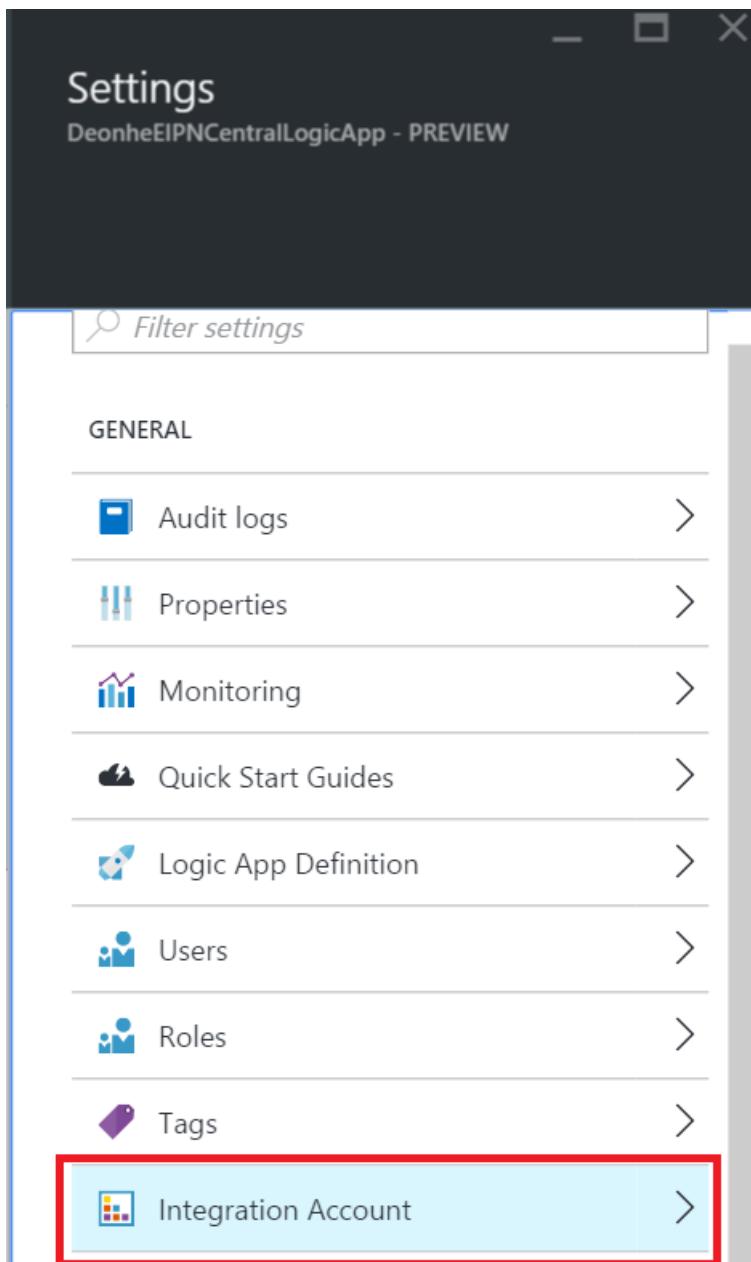
NOTE

Be sure your integration account and Logic app are in the **same Azure location** before you begin

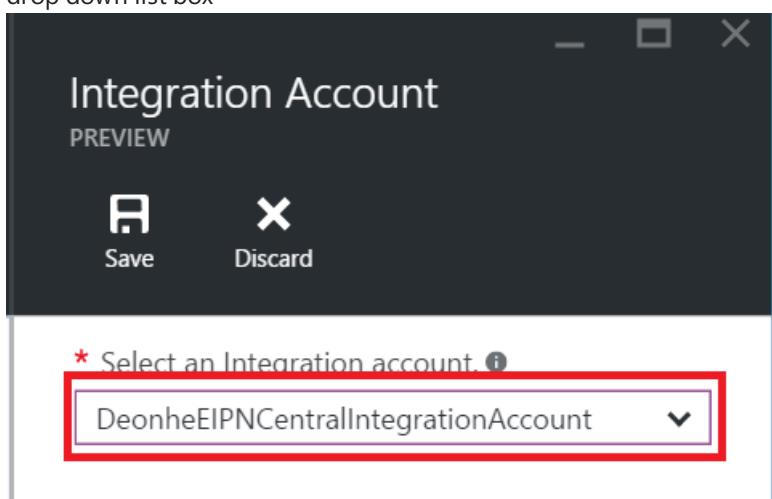
1. Select **Settings** link from the menu of your Logic app



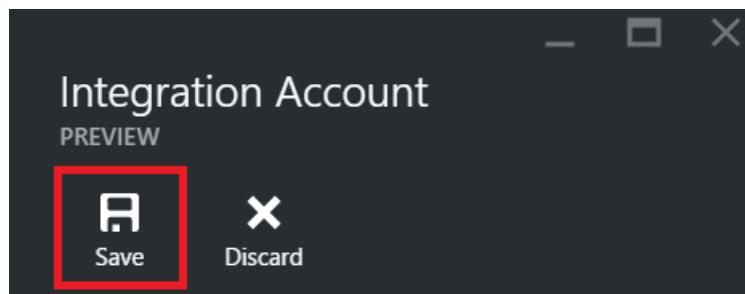
2. Select the **Integration Account** item from the Settings blade



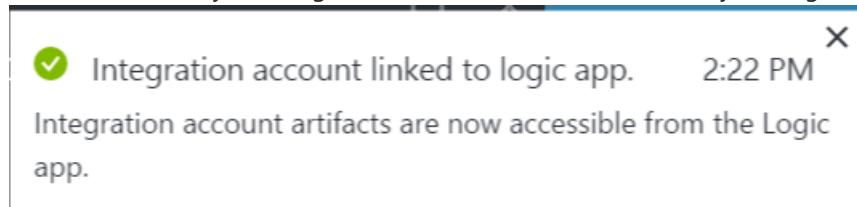
3. Select the integration account you wish to link to your Logic app from the **Select an Integration account** drop down list box



4. Save your work



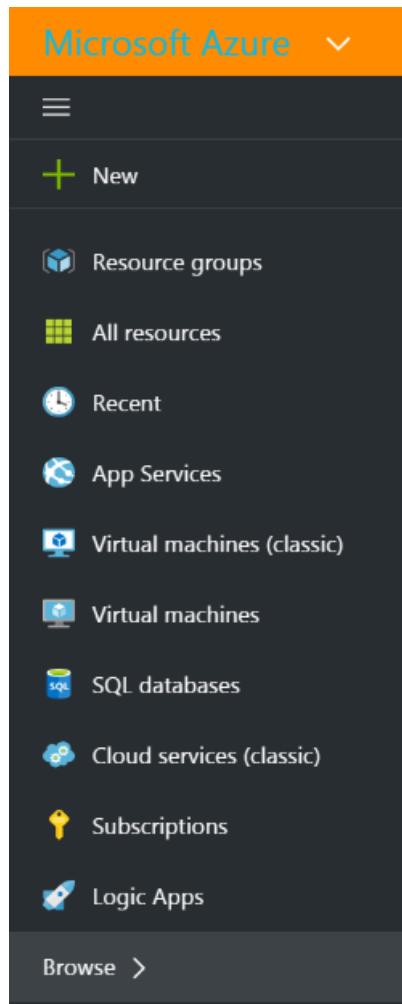
5. You will see a notification that indicates that your integration account has been linked to your Logic app and that all artifacts in your integration account are now available to your Logic app.



Now that your integration account is linked to your Logic app, you can go to your Logic app and use B2B connectors such as the XML Validation, Flat file encode/decode or Transform to create apps with B2B features.

How to delete an integration account?

1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list

A screenshot of the Microsoft Azure portal. At the top, there is an orange header bar. Below it, a search bar contains the text "integration". A tooltip above the search bar says "Shift+Space to toggle favorites". Below the search bar, a navigation bar includes "Integration Accounts" (with a plus icon), "Preview", and a star icon.

3. Select the **integration account** that you wish to delete

A screenshot of the "Integration Accounts" blade in the Microsoft Azure portal. The title is "Integration Accounts" and it says "Microsoft - PREVIEW". The interface includes buttons for "Add", "Columns", and "Refresh". It shows a list of three integration accounts:

NAME	RESOURCE GROUP	LOCATION
DeonheIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South

4. Select the **Delete** link that's located on the menu

MyNewIntegrationAccount
Integration Account - PREVIEW

Move Delete Step 4

Essentials ▾

Resource group: MyNewRG
Name: MyNewIntegrationAccount
Location: brazilsouth
Subscription: ICBCS9
Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

All settings →

Components

Add tiles (+)

Schemas	Maps	Certificates
0	0	0

Partners	Agreements
1	0

Add a section (+)

5. Confirm your choice

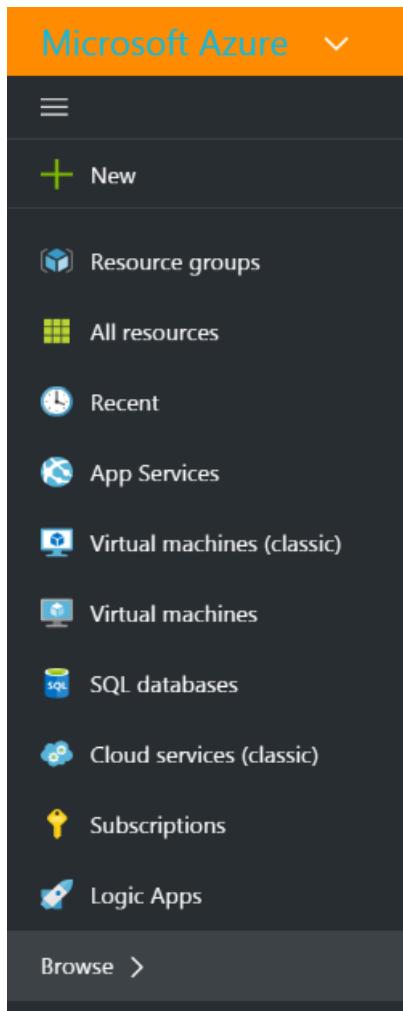
How to move an integration account?

You can easily move an integration account to a new subscription and a new resource group. Follow these steps if you need to move your integration account:

IMPORTANT

You will need to update all scripts to use the new resource IDs after you move an integration account.

1. Select **Browse**



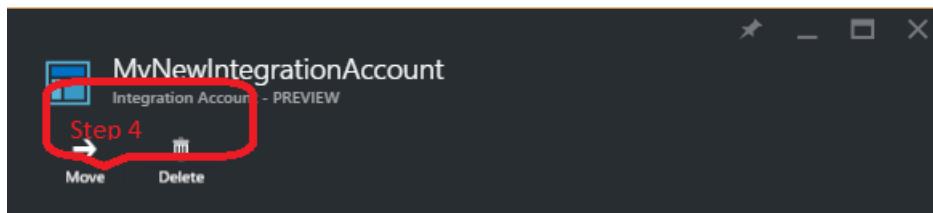
2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list

A screenshot of a search results page. At the top is a red search bar with the word "integration". Below it is a white header bar with the text "Shift+Space to toggle favorites" and a close button "X". A search input field contains "integration". Below the input field is a navigation bar with icons for "Integration Accounts", "Preview", and a star icon. The main content area is currently empty, showing a message "No results found".

3. Select the **integration account** that you wish to delete

A screenshot of the "Integration Accounts" blade in the Azure portal. The title is "Integration Accounts" and it says "Microsoft - PREVIEW". Below the title are three buttons: "Add", "Columns", and "Refresh". The main area has a header "Subscriptions: All 3 selected" with a "Filter items..." input field and a "All subscriptions" link. Below this is a table with columns "NAME", "RESOURCE GROUP", and "LOCATION". Two rows are visible: one for "DeonheIntegrationAccount" in "DeonheResGroup" located in "Brazil South", and another for "MyNewIntegrationAccount" in "MyNewRG" located in "Brazil South".

4. Select the **Move** link that's located on the menu



A screenshot of the 'Components' section in the Azure portal for the 'MyNewIntegrationAccount' integration account. The section is divided into four categories: Schemas, Maps, Certificates, Partners, Agreements, and a blank area. The 'Schemas' category is highlighted with a blue border. It shows 0 schemas. The 'Maps' category shows 0 maps. The 'Certificates' category shows 0 certificates. The 'Partners' category shows 1 partner. The 'Agreements' category shows 0 agreements. There is also a 'Add a section +' button at the bottom.

5. Confirm your choice

Next Steps

- Learn more about agreements

Learn about agreements and Enterprise Integration Pack

1/25/2017 • 1 min to read • [Edit on GitHub](#)

Overview

Agreements are the cornerstone of business-to-business (B2B) communications, allowing business entities to communicate seamlessly using industry standard protocols.

What is an agreement?

An agreement, as far as the Enterprise Integration Pack is concerned, is a communications arrangement between B2B trading partners. An agreement is based on the communications the partners wish to achieve and is protocol or transport specific.

Enterprise integration supports three protocol/transport standards:

- [AS2](#)
- [X12](#)
- [EDIFACT](#)

Why use agreements

Some of the common benefits of using agreements are:

- Enables different organizations and businesses to be able to exchange information in a well known format.
- Improves efficiency when conducting B2B transactions
- Easy to create, manage and use them when creating enterprise integration apps

How to create agreements

- [Create an AS2 agreement](#)
- [Create an X12 agreement](#)
- [Create an EDIFACT agreement](#)

How to use an agreement

After creating an agreement, you can use it via the Azure portal to create [Logic apps](#) with B2B features.

How to edit an agreement

You can edit any agreement by following these steps:

1. Select the Integration account that contains the agreement you wish to modify.
2. Select the **Agreements** tile
3. Select the agreement you wish to modify on the **Agreements** blade
4. Select **Edit** from the menu above
5. On the Edit menu that opens, make your changes then select the **OK** button to save the changes

How to delete an agreement

You can delete any agreement by following these steps from within the integration account that contains the agreement you wish to delete:

1. Select the **Agreements** tile
2. Select the agreement you wish to delete on the **Agreements** blade
3. Select **Delete** from the menu above
4. Confirm that you really want to delete the agreement
5. Notice that the agreement is no longer listed on the Agreements blade

Next steps

- [Create an AS2 agreement](#)

Learn about receiving data using the B2B features of the Enterprise Integration Pack

1/20/2017 • 3 min to read • [Edit on GitHub](#)

Overview

This document is part of the Logic Apps Enterprise Integration Pack. Check out the overview to learn more about the [capabilities of the Enterprise Integration Pack](#).

Prerequisites

To use the AS2 and X12 actions you will need an Enterprise Integration Account

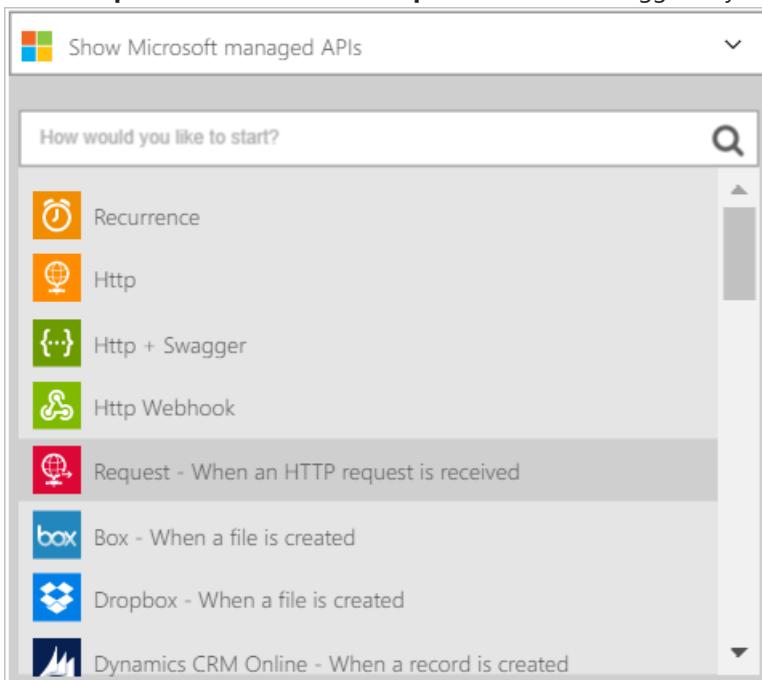
[How to create an Enterprise Integration Account](#)

How to use the Logic Apps B2B connectors

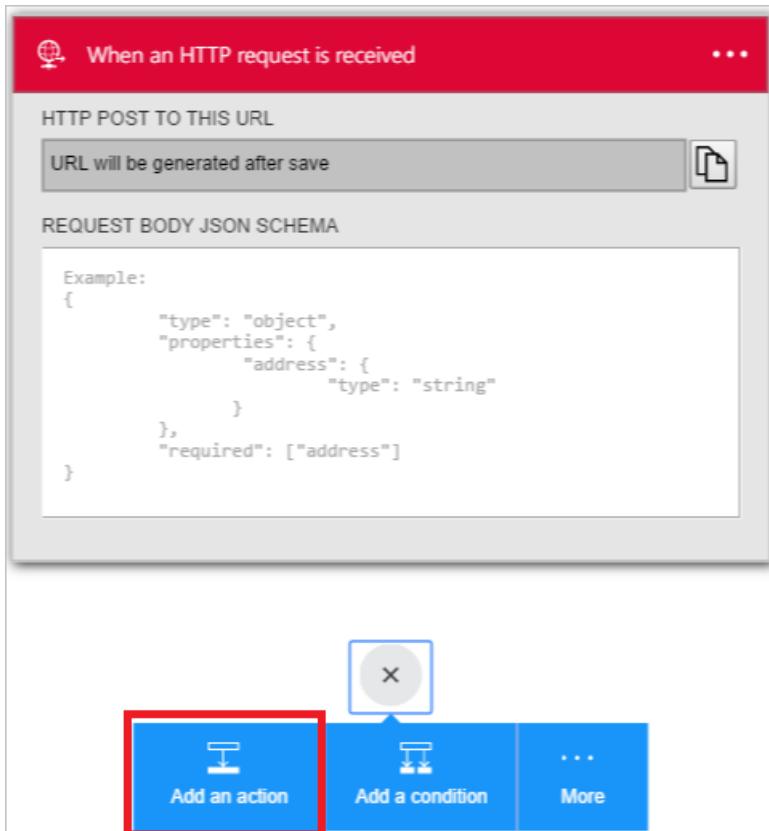
Once you have created an integration account and added partners and agreements to it you are ready to create a Logic App that implements a business to business (B2B) workflow.

In this walkthrough you'll see how to use the AS2 and X12 actions to create a business to business Logic App that receives data from a trading partner.

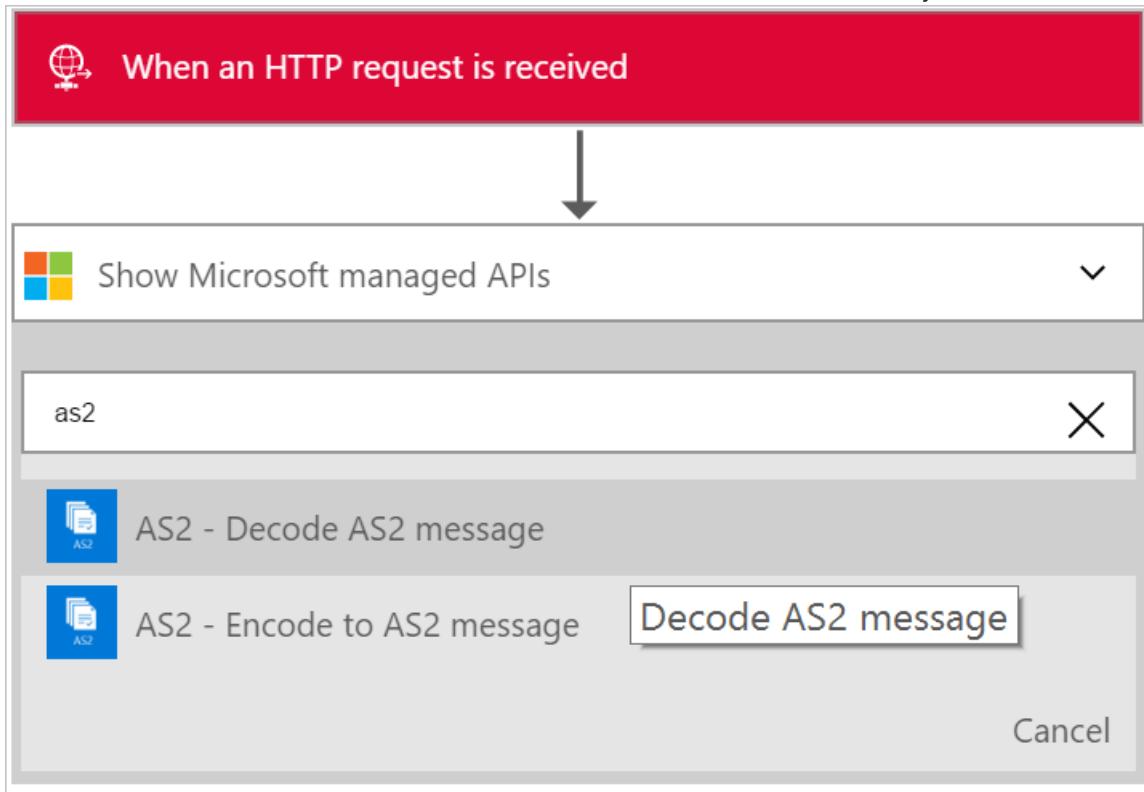
1. Create a new Logic app and [link it to your integration account](#).
2. Add a **Request - When an HTTP request is received** trigger to your Logic app



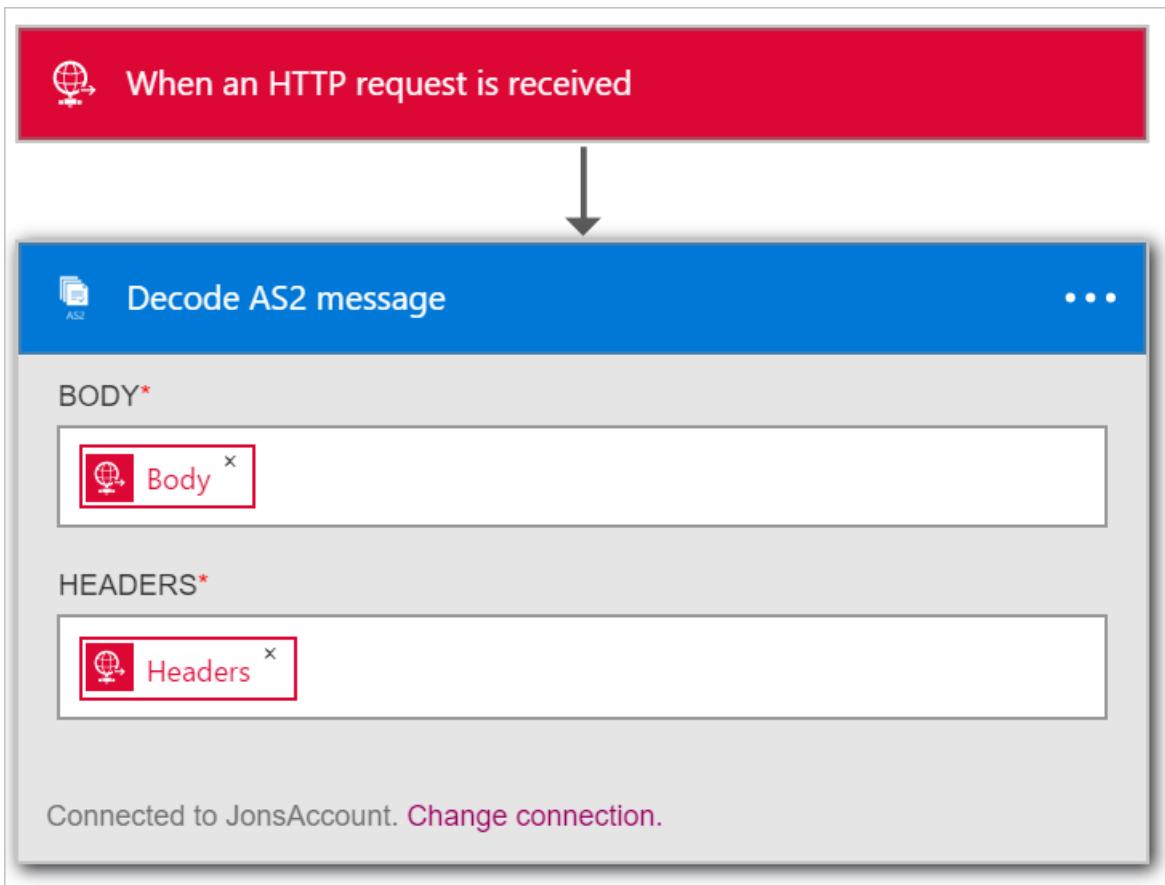
3. Add the **Decode AS2** action by first selecting **Add an action**



4. Enter the word **as2** in the search box in order to filter all the actions to the one that you want to use



5. Select the **AS2 - Decode AS2 message** action



6. As shown, add the **Body** that you will take as input. In this example, select the body of the HTTP request that triggered the Logic app. You can alternatively enter an expression to input the headers in the **HEADERS** field:

```
@triggerOutputs()['headers']
```

7. Add the **Headers** that are required for AS2. These will be in the HTTP request headers. In this example, select the headers of the HTTP request that triggered the Logic app.
8. Now add the Decode X12 message action by again selecting **Add an action**



When an HTTP request is received



Decode AS2 message

...

BODY*



HEADERS*



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



Add an action

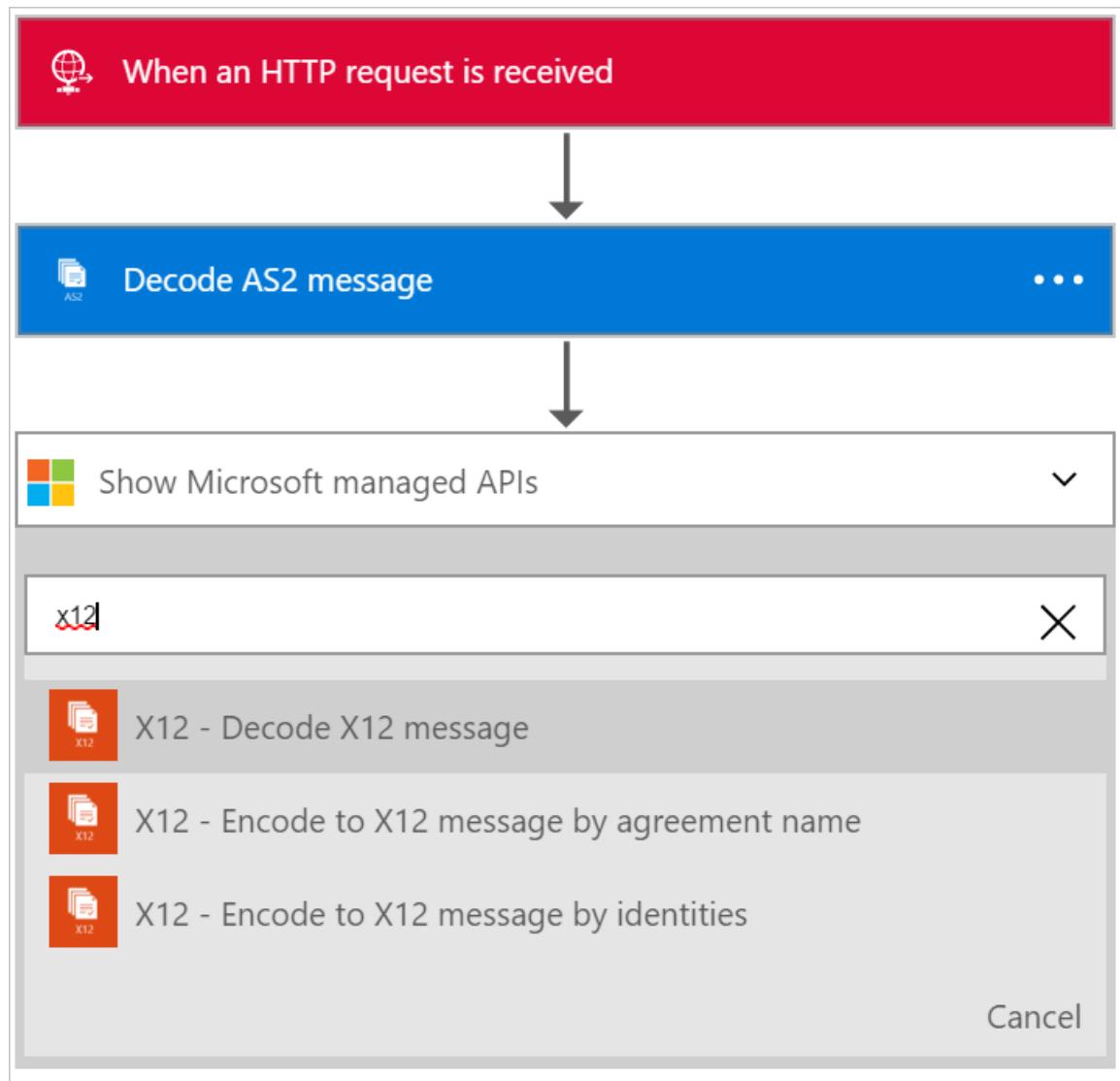


Add a condition

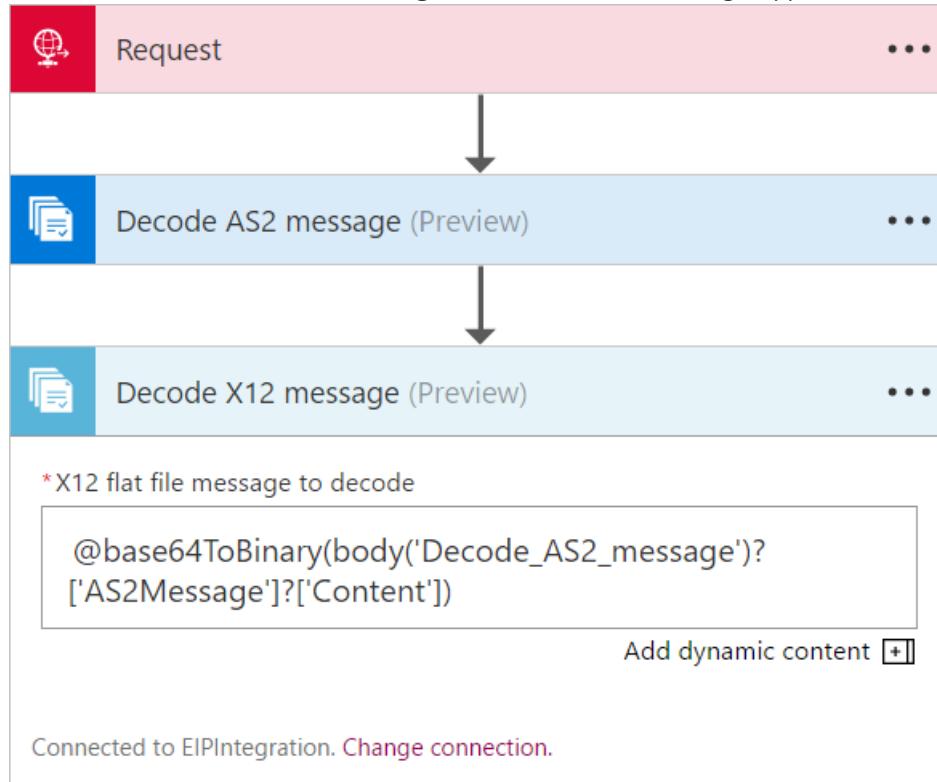
...

More

9. Enter the word **x12** in the search box in order to filter all the actions to the one that you want to use



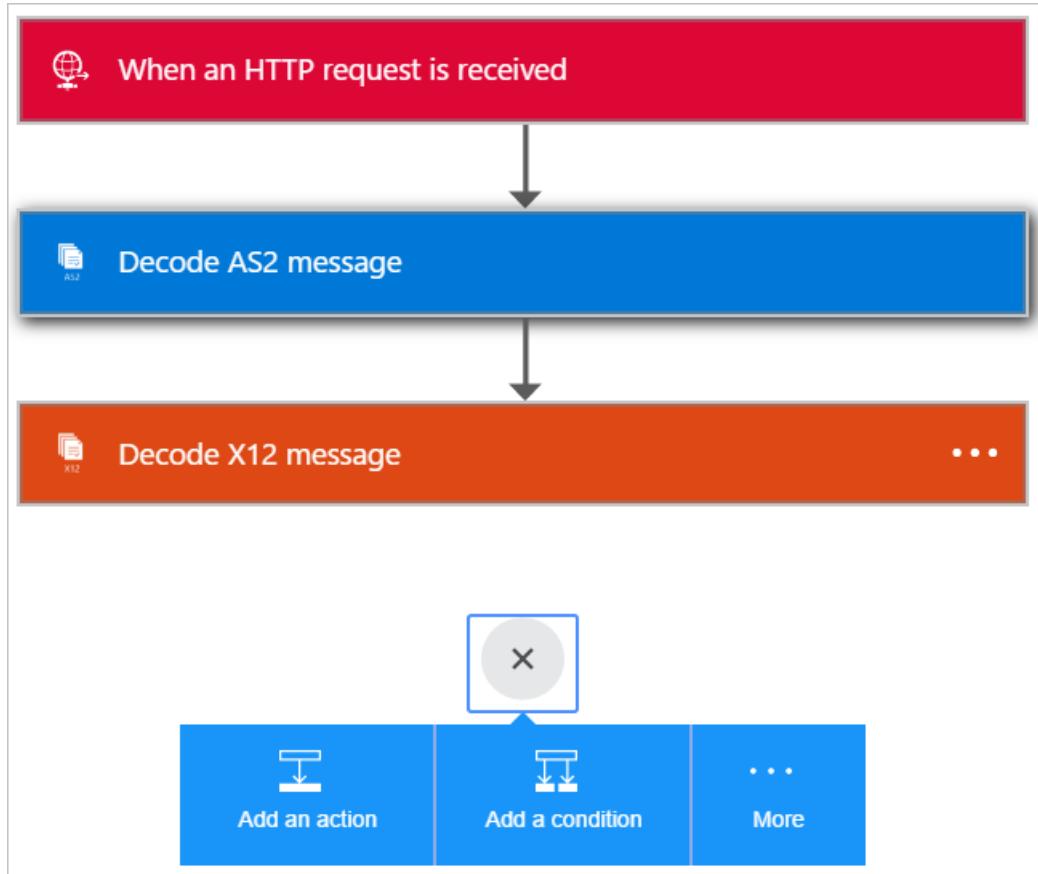
10. Select the **X12 - Decode X12 message** action to add it to the Logic app



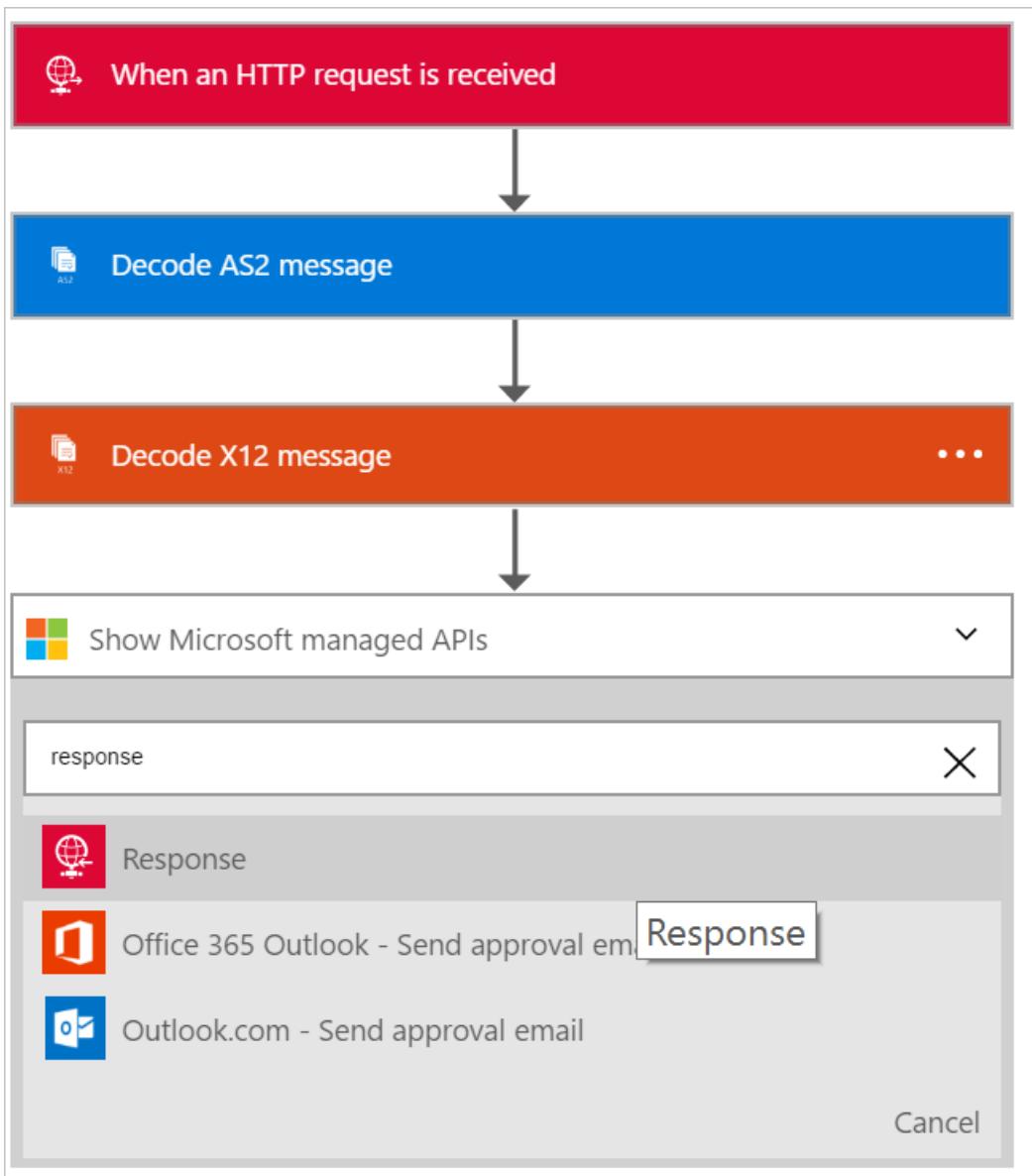
11. You now need to specify the input to this action which will be the output of the AS2 action above. The actual message content is in a JSON object and is base64 encoded. You therefore need to specify an expression as the input so enter the following expression in the **X12 FLAT FILE MESSAGE TO DECODE** input field

```
@base64ToString(body('Decode_AS2_message')?['AS2Message']?['Content'])
```

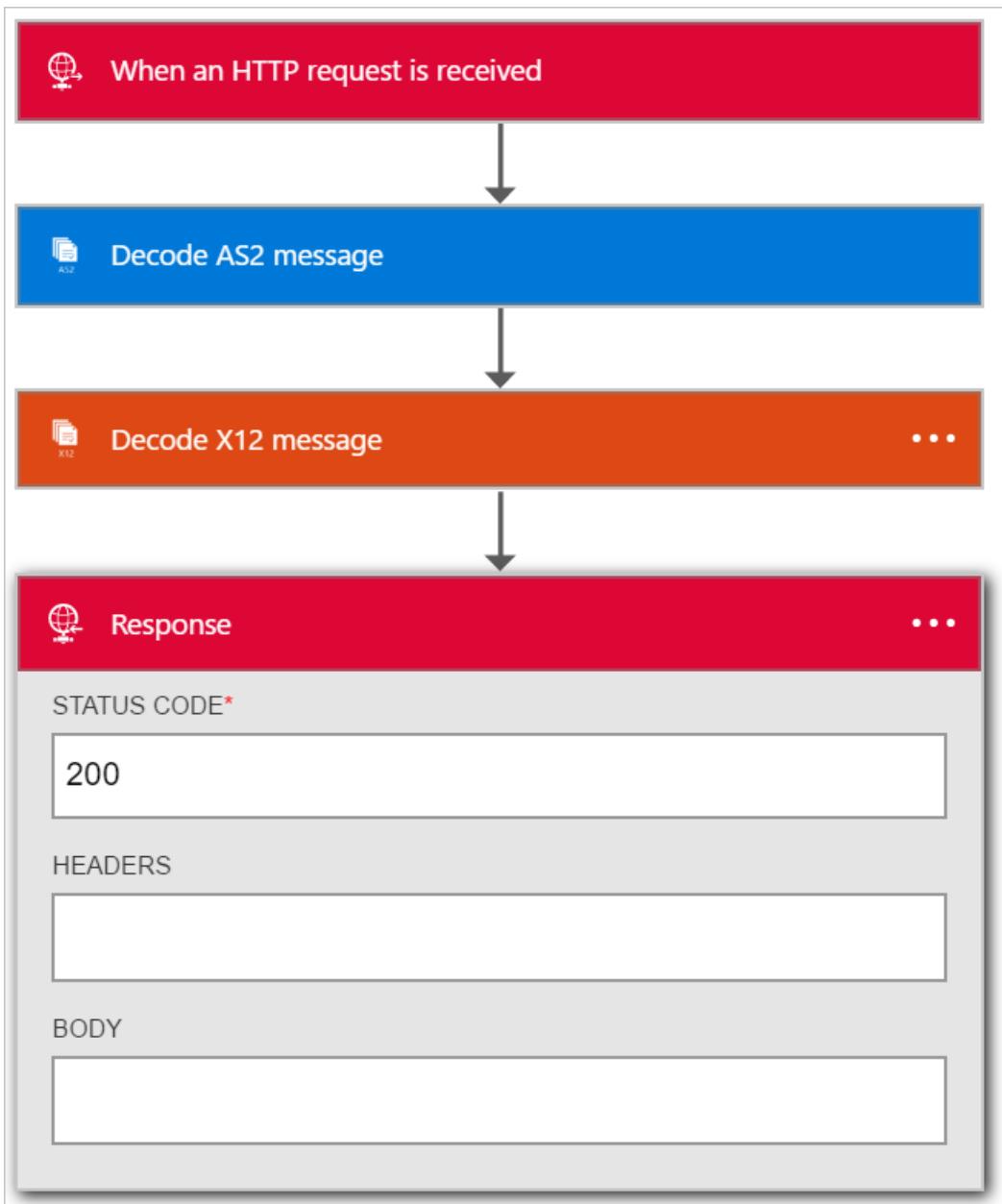
12. This step will decode the X12 data received from the trading partner and will output a number of items in a JSON object. In order to let the partner know of the receipt of the data you can send back a response containing the AS2 Message Disposition Notification (MDN) in an HTTP Response Action
13. Add the **Response** action by selecting **Add an action**



14. Enter the word **response** in the search box in order to filter all the actions to the one that you want to use

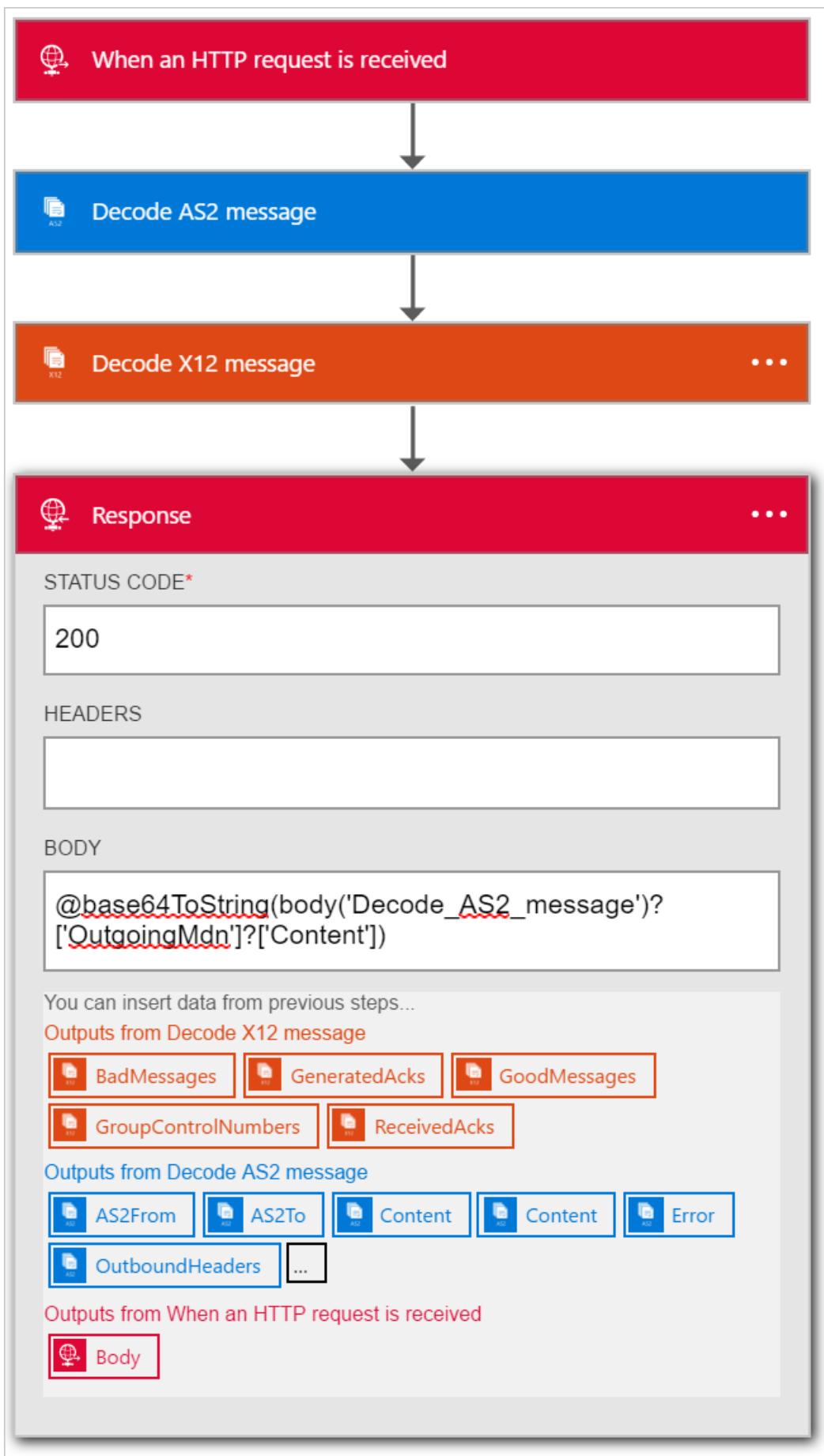


15. Select the **Response** action to add it

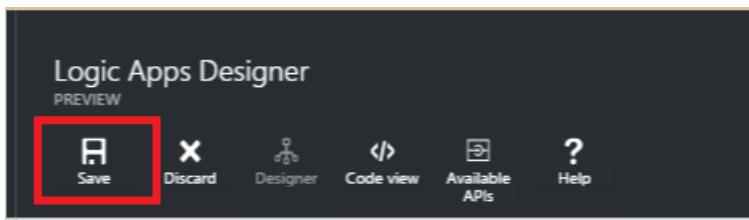


16. Set the response **BODY** field by using the following expression to access the MDN from the output of the **Decode X12 message** action

```
@base64ToString(body('Decode_X12_message')?['OutgoingMdn']?['Content'])
```



1. Save your work



At this point, you are finished setting up your B2B Logic app. In a real world application, you may want to store the decoded X12 data in an LOB application or data store. You can easily add further actions to do this or write custom APIs to connect to your own LOB applications and use these APIs in your Logic app.

Features and use cases

- The AS2 and X12 decode and encode actions allow you to receive data from and send data to trading partners using industry standard protocols using Logic apps
- You can use AS2 and X12 with or without each other to exchange data with trading partners as required
- The B2B actions make it easy to create partners and agreements in the Integration Account and consume them in a Logic app
- By extending your Logic app with other actions you can send and receive data to and from other applications and services such as SalesForce

Learn more

[Learn more about the Enterprise Integration Pack](#)

XML processing

1/20/2017 • 1 min to read • [Edit on GitHub](#)

The Enterprise Integration Pack makes it easy to validate and process XML documents that you exchange with business partners. Here are the ways you can process these XML messages using Logic apps:

- [XML validation](#) - XML validation provides the ability to validate a message that originates from a source endpoint against a specific schema.
- [XML transform](#) - XML transform provides the ability to convert an XML message based on the requirements of a destination endpoint.
- [Flat file encoding and flat file decoding](#) - Flat file encoding/decoding provides the ability to encode or decode a flat file.
- [XPath](#) - Provides the ability to enrich a message and extract specific properties from the message. The extracted properties can then be used to route the message to a destination or an intermediary endpoint.

Try it for yourself

Why not give it a try. Click [here](#) to deploy a fully operational logic app of your own using the XML features of Logic Apps

Learn more

[Learn more about the Enterprise Integration Pack](#)

Enterprise integration with flat files

1/20/2017 • 3 min to read • [Edit on GitHub](#)

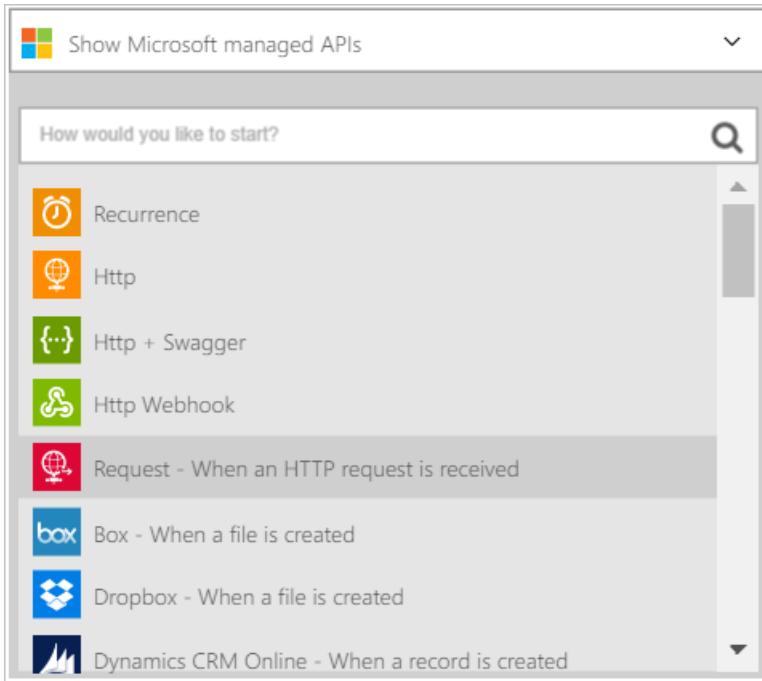
Overview

You may want to encode XML content before you send it to a business partner in a business-to-business (B2B) scenario. In a logic app, you can use the flat file encoding connector to do this. The logic app that you create can get its XML content from a variety of sources, including from an HTTP request trigger, from another application, or even from one of the many [connectors](#). For more information about logic apps, see the [logic apps documentation](#).

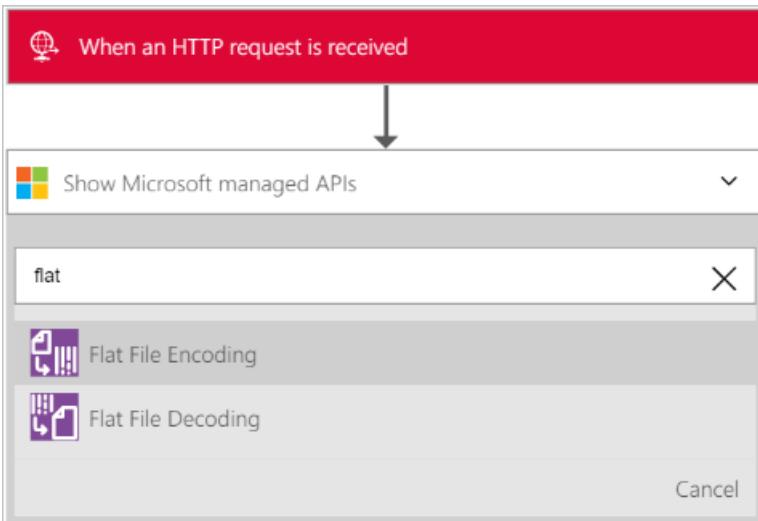
How to create the flat file encoding connector

Follow these steps to add a flat file encoding connector to your logic app.

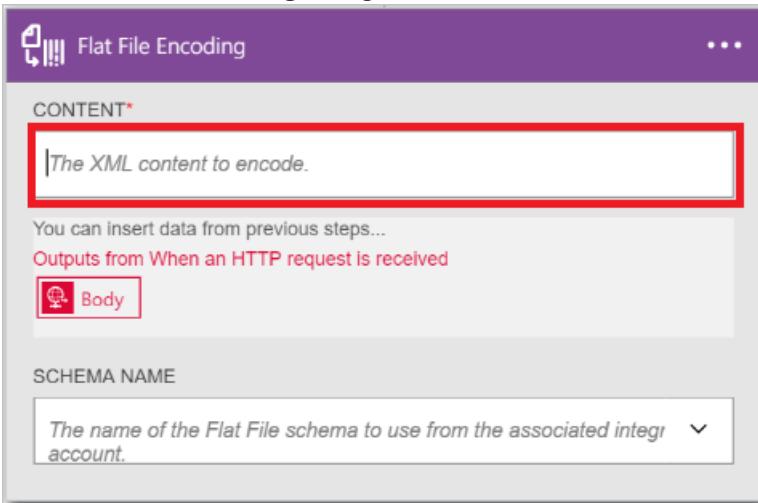
1. Create a logic app and [link it to your integration account](#). This account contains the schema you will use to encode the XML data.
2. Add a **Request - When an HTTP request is received** trigger to your logic app.



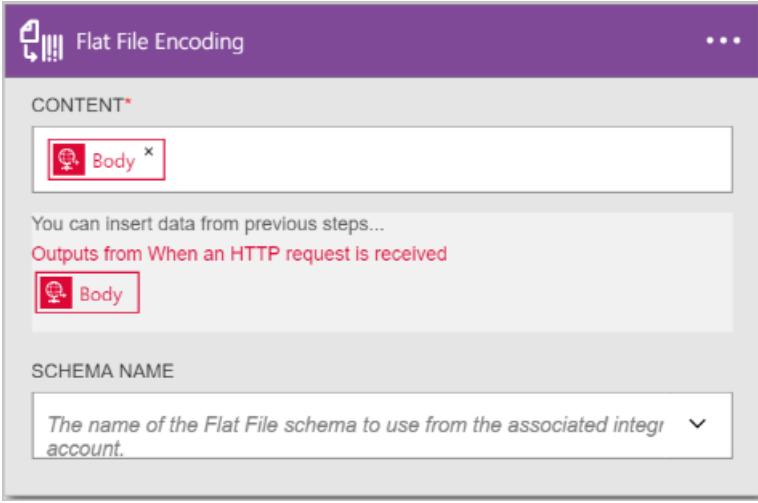
3. Add the flat file encoding action, as follows:
 - a. Select the **plus** sign.
 - b. Select the **Add an action** link (appears after you have selected the plus sign).
 - c. In the search box, enter *Flat* to filter all the actions to the one that you want to use.
 - d. Select the **Flat File Encoding** option from the list.



4. On the **Flat File Encoding** dialog box, select the **Content** text box.



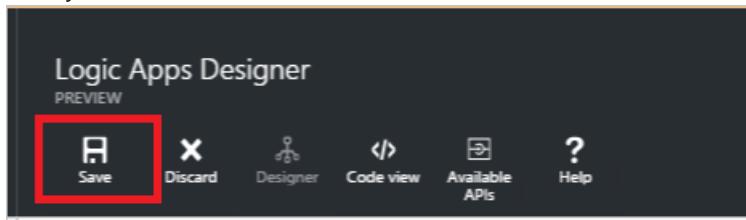
5. Select the body tag as the content that you want to encode. The body tag will populate the content field.



6. Select the **Schema Name** list box, and choose the schema you want to use to encode the input content.

The screenshot shows the 'Flat File Encoding' configuration page. At the top, there's a 'CONTENT*' section containing a red 'Body' placeholder box. Below it is a note: 'You can insert data from previous steps... Outputs from When an HTTP request is received'. Under 'SCHEMA NAME', there's a dropdown menu with the placeholder text: 'The name of the Flat File schema to use from the associated integration account.' A red box highlights this dropdown.

7. Save your work.



At this point, you are finished setting up your flat file encoding connector. In a real world application, you may want to store the encoded data in a line-of-business application, such as Salesforce. Or you can send that encoded data to a trading partner. You can easily add an action to send the output of the encoding action to Salesforce, or to your trading partner, by using any one of the other connectors provided.

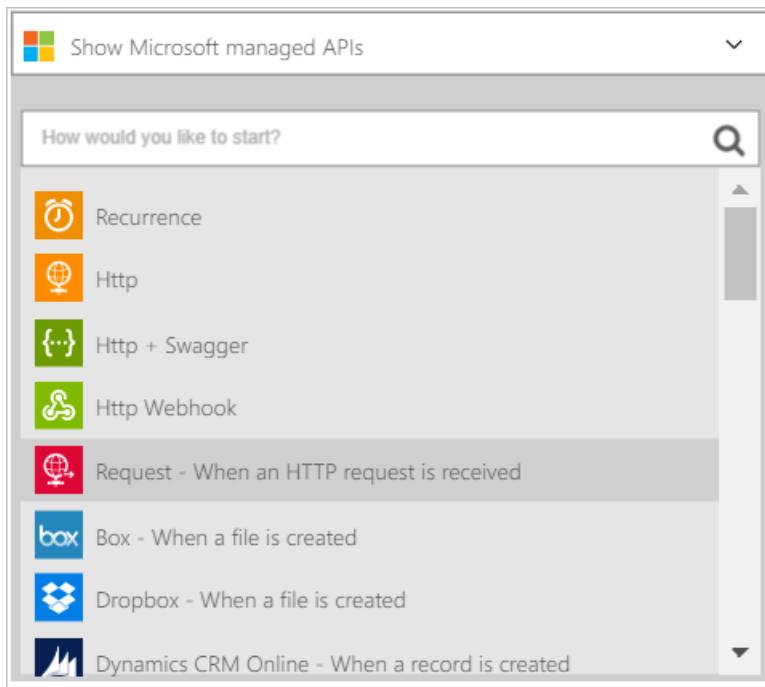
You can now test your connector by making a request to the HTTP endpoint, and including the XML content in the body of the request.

How to create the flat file decoding connector

NOTE

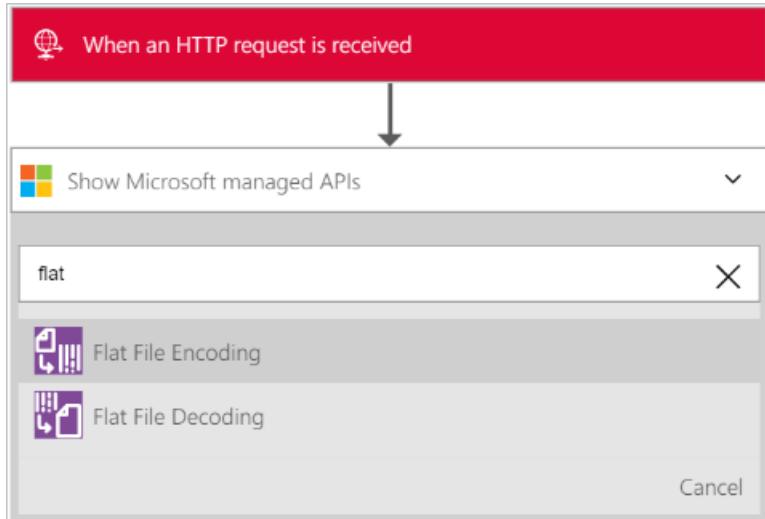
To complete these steps, you need to have a schema file already uploaded into your integration account.

1. Add a **Request - When an HTTP request is received** trigger to your logic app.

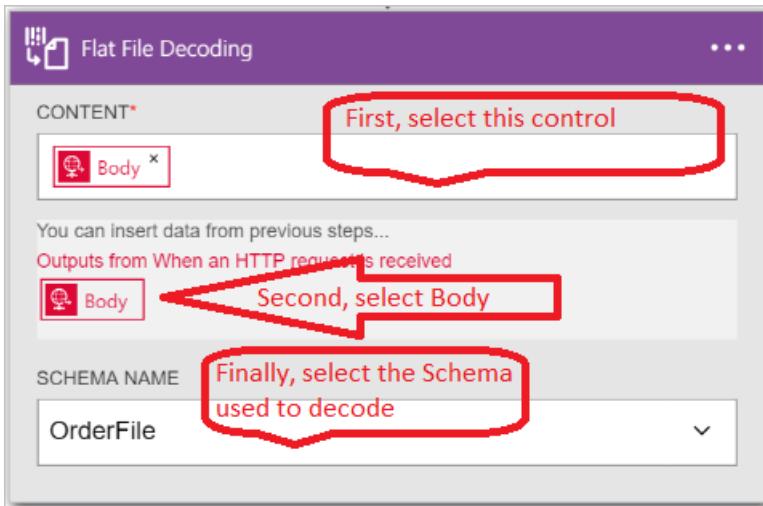


2. Add the flat file decoding action, as follows:

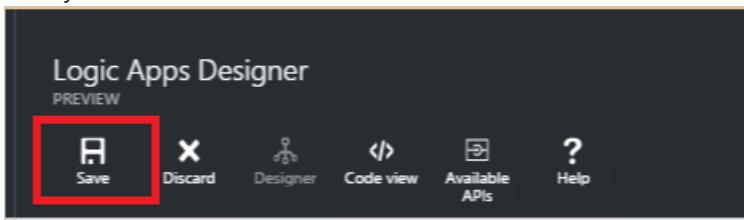
- Select the **plus** sign.
- Select the **Add an action** link (appears after you have selected the plus sign).
- In the search box, enter *Flat* to filter all the actions to the one that you want to use.
- Select the **Flat File Decoding** option from the list.



- Select the **Content** control. This produces a list of the content from earlier steps that you can use as the content to decode. Notice that the *Body* from the incoming HTTP request is available to be used as the content to decode. You can also enter the content to decode directly into the **Content** control.
- Select the *Body* tag. Notice the body tag is now in the **Content** control.
- Select the name of the schema that you want to use to decode the content. The following screenshot shows that *OrderFile* is the selected schema name. This schema name had been uploaded into the integration account previously.



6. Save your work.



At this point, you are finished setting up your flat file decoding connector. In a real world application, you may want to store the decoded data in a line-of-business application such as Salesforce. You can easily add an action to send the output of the decoding action to Salesforce.

You can now test your connector by making a request to the HTTP endpoint and including the XML content you want to decode in the body of the request.

Next steps

- [Learn more about the Enterprise Integration Pack.](#)

Learn about maps and the Enterprise Integration Pack

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Overview

Enterprise integration uses maps to transform XML data from one format to another format.

What is a map?

A map is an XML document that defines which data in a document should be transformed into another format.

Why use maps?

Let's assume you regularly receive B2B orders or invoices from a customers who uses the YYMMDD format for dates. However, in your organization, you store dates in the MMDDYY format. You can use a map to *transform* the YYMMDD date format into the MMDDYY before storing the order or invoice details in your customer activity database.

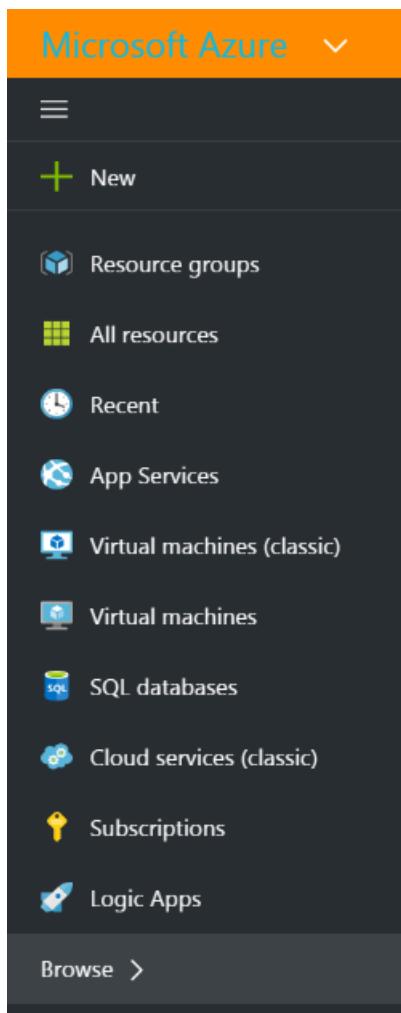
How do i create a map?

The [Enterprise Integration Pack](#) for Visual Studio 2015 allows Biztalk Integration projects to be created. Creating an Integration Map file will allow you to visually map items between two XML schema files. After building this project, an XSLT document is output.

How to upload a map?

From the Azure portal:

1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list

A screenshot showing the search results for 'integration'. An orange bar at the top has an 'X' icon. Below it is a search bar containing the text 'integration'. A message 'Shift+Space to toggle favorites' is displayed above the search bar. Below the search bar is a list of results. The first result is 'Integration Accounts' with an icon of a blue square with a white gear. To its right are 'Preview' and a star icon. The rest of the results are cut off by a horizontal scroll bar.

3. Select the **integration account** into which you will add the map

A screenshot of the 'Integration Accounts' preview page. The title is 'Integration Accounts' with 'Microsoft - PREVIEW' below it. There are three buttons at the top: 'Add' (with a plus sign), 'Columns' (with a grid icon), and 'Refresh' (with a circular arrow). Below this is a section titled 'Subscriptions: All 3 selected'. It contains two input fields: 'Filter items...' and 'All subscriptions'. A table then lists three integration accounts: 'DeonheIntegrationAccount' (Resource Group: DeonheResGroup, Location: Brazil South) and 'MyNewIntegrationAccount' (Resource Group: MyNewRG, Location: Brazil South).

NAME	RESOURCE GROUP	LOCATION
DeonheIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South

4. Select the **Maps** tile

MyNewIntegrationAccount
Integration Account - PREVIEW

Move Delete

Essentials ^

Name: MyNewIntegrationAccount
Resource group: MyNewRG
Location: brazilsouth
Subscription: ICBCS9
Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

All settings →

Components Add tiles +

Schemas	Maps	Certificates
1	0	0
Partners	Agreements	
1	0	

Add a section +

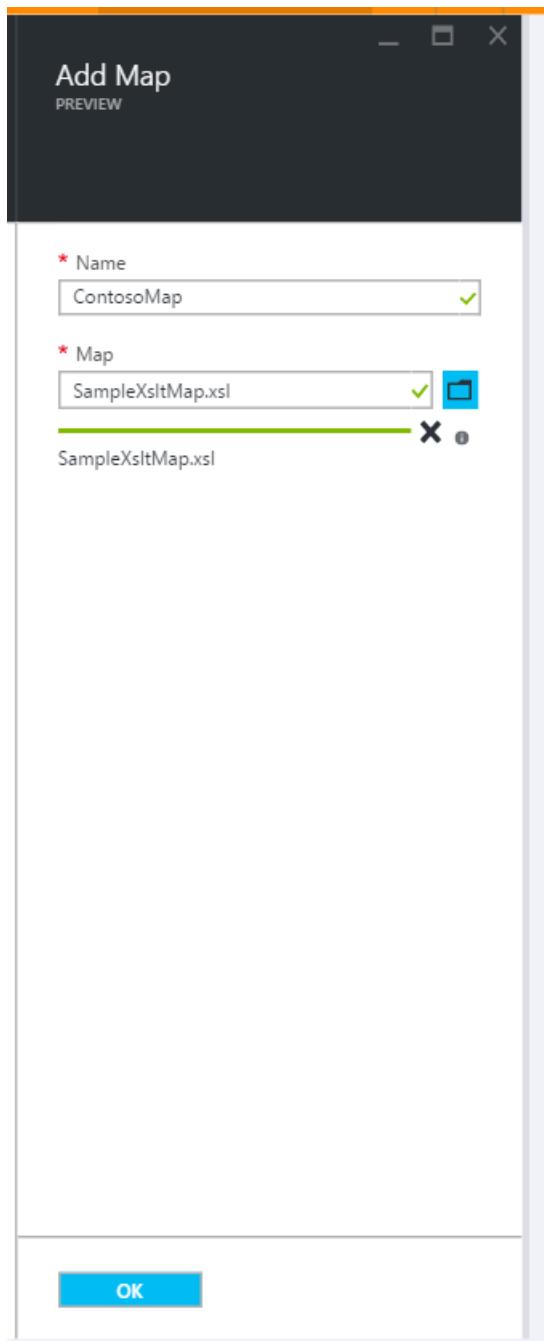
5. Select the **Add** button in the Maps blade that opens

Maps
MyNewIntegrationAccount - PREVIEW

Add Delete Update Download

NAME	TYPE	CONTENT SIZE
There are no maps to display.		

6. Enter a **Name** for your map then, to upload the map file, select the folder icon on the right side of the **Map** text box. After the upload process is completed, select the **OK** button.



7. The map is now being added into your integration account. You will receive an onscreen notification that indicates the success or failure of adding the map file. After you receive the notification, select the **Maps** tile, you will then see your newly added map in the Maps blade:

Essentials

Name: MyNewIntegrationAccount

Resource group: MyNewRG

Location: brazilsouth

Subscription: ICBCS9

Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

Components

Schemas	Maps	Certificates
1	1	2
2	2	

Maps

MyNewIntegrationAccount - PREVIEW

Add Delete Update Download

NAME	TYPE	CONTENT SIZE
ContosoMap	Xslt	3056

Properties

PREVIEW

NAME	ContosoMap
TYPE	Xslt
CONTENT SIZE	3056
CONTENT LINK	https://flowprodcu02cgsn01.blob.core.windows.net/
CREATED TIME	6/25/2016, 1:13 AM
CHANGED TIME	6/25/2016, 1:13 AM

How to edit a map?

To edit a map, you must upload a new map file with the changes you desire. You can first download the map and edit it.

Follow these steps to upload a new map that replaces an existing map:

1. Select the **Maps** tile
2. Select the map you wish to edit when the Maps blade opens up
3. On the **Maps** blade, select the **Update** link

Essentials

Name: MyNewIntegrationAccount

Resource group: MyNewRG

Location: brazilsouth

Subscription: ICBCS9

Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

Components

Schemas	Maps	Certificates
1	1	3
2	2	

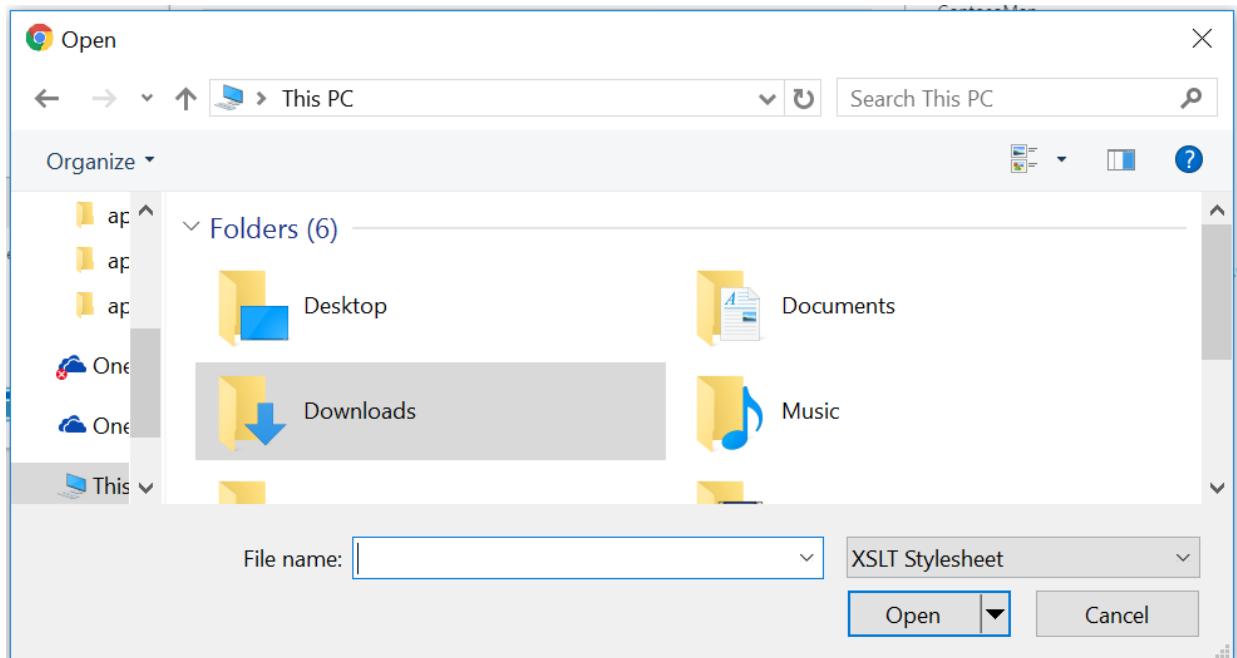
Maps

MyNewIntegrationAccount - PREVIEW

Add Delete **Update** Download

NAME	TYPE	CONTENT SIZE
ContosoMap	Xslt	3056

4. Select the map file you wish to upload by using the file picker dialog that opens up then select **Open** in the file picker



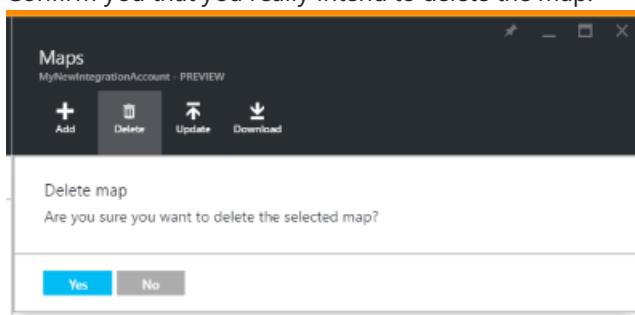
5. You will receive a notification popup after the map is uploaded.

How to delete a map?

1. Select the **Maps** tile
2. Select the map you wish to delete when the Maps blade opens up
3. Select the **Delete** link

NAME	TYPE	CONTENT SIZE
ContosoMap	Xslt	3056

4. Confirm you that you really intend to delete the map.



Next Steps

- [Learn more about the Enterprise Integration Pack](#)
- [Learn more about agreements](#)
- [Learn more about transforms](#)

Enterprise integration with XML transforms

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Overview

The Enterprise integration Transform connector converts data from one format to another format. For example, you may have an incoming message that contains the current date in the YearMonthDay format. You can use a transform to reformat the date to be in the MonthDayYear format.

What does a transform do?

A Transform, which is also known as a map, consists of a Source XML schema (the input) and a Target XML schema (the output). You can use different built-in functions to help manipulate or control the data, including string manipulations, conditional assignments, arithmetic expressions, date time formatters, and even looping constructs.

How to create a transform?

You can create a transform/map by using the Visual Studio [Enterprise Integration SDK](#). When you are finished creating and testing the transform, you upload the transform into your integration account.

How to use a transform

After you upload the transform/map into your integration account, you can use it to create a Logic app. The Logic app runs your transformations whenever the Logic app is triggered (and there is input content that needs to be transformed).

Here are the steps to use a transform:

Prerequisites

- Create an integration account and add a map to it

Now that you've taken care of the prerequisites, it's time to create your Logic app:

1. Create a Logic app and [link it to your integration account](#) that contains the map.
2. Add a **Request** trigger to your Logic app

The screenshot shows the 'Show Microsoft managed APIs' dropdown open, revealing a list of triggers. The 'Request' trigger is highlighted with a red box. Other triggers listed include Recurrence, HTTP, HTTP + Swagger, HTTP Webhook, appFigures - When an app is rated (Preview), appFigures - When an event occurs (Preview), and appFigures - When there is a new review (Preview).

3. Add the **Transform XML** action by first selecting **Add an action**

The screenshot shows the Logic App builder interface. A red box highlights the 'Add an action' button in the central toolbar. Above the toolbar, a pink bar displays the trigger name 'Request'. Below the toolbar, a white box labeled '+ New step' is visible.

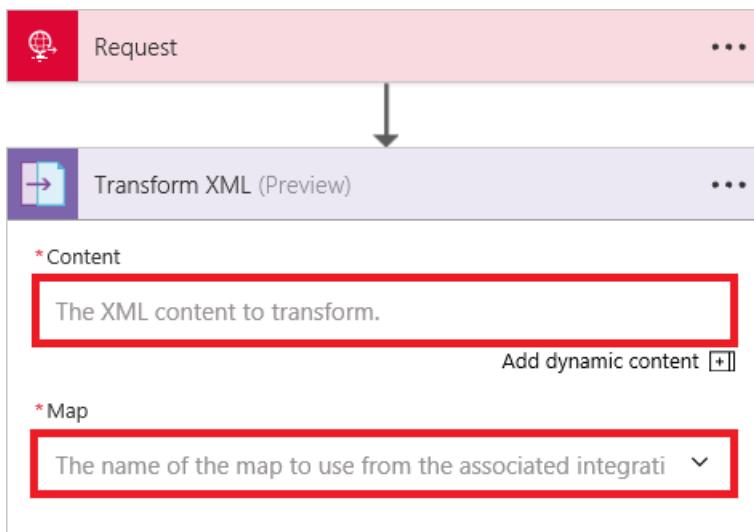
4. Enter the word *transform* in the search box to filter all the actions to the one that you want to use

The screenshot shows the Logic App builder interface after searching for 'trans'. A red box highlights the 'Transform XML (Preview)' action in the search results. Other actions listed are 'Mandrill - Send mail' and 'Show Microsoft managed APIs'.

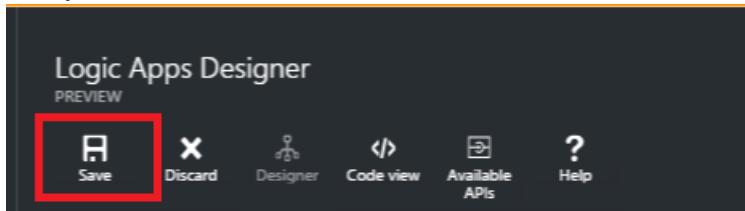
5. Select the **Transform XML** action

6. Add the XML **CONTENT** that you transform. You can use any XML data you receive in the HTTP request as the **CONTENT**. In this example, select the body of the HTTP request that triggered the Logic app.

7. Select the name of the **MAP** that you want to use to perform the transformation. The map must already be in your integration account. In an earlier step, you already gave your Logic app access to your integration account that contains your map.



8. Save your work



At this point, you are finished setting up your map. In a real world application, you may want to store the transformed data in an LOB application such as SalesForce. You can easily as an action to send the output of the transform to Salesforce.

You can now test your transform by making a request to the HTTP endpoint.

Features and use cases

- The transformation created in a map can be simple, such as copying a name and address from one document to another. Or, you can create more complex transformations using the out-of-the-box map operations.
- Multiple map operations or functions are readily available, including strings, date time functions, and so on.
- You can do a direct data copy between the schemas. In the Mapper included in the SDK, this is as simple as drawing a line that connects the elements in the source schema with their counterparts in the destination schema.
- When creating a map, you view a graphical representation of the map, which shows all the relationships and links you create.
- Use the Test Map feature to add a sample XML message. With a simple click, you can test the map you created, and see the generated output.
- Upload existing maps
- Includes support for the XML format.

Learn more

- [Learn more about the Enterprise Integration Pack](#)
- [Learn more about maps](#)

Enterprise integration with XML validation

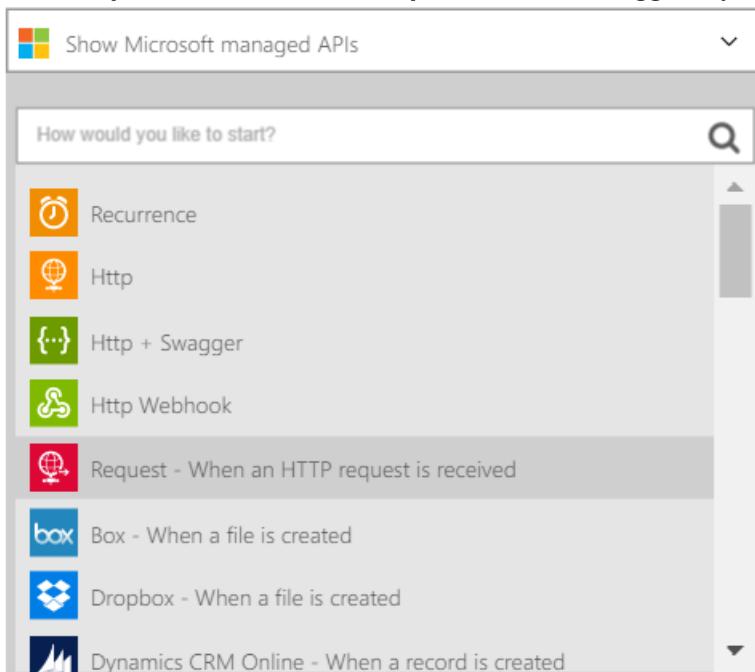
1/20/2017 • 1 min to read • [Edit on GitHub](#)

Overview

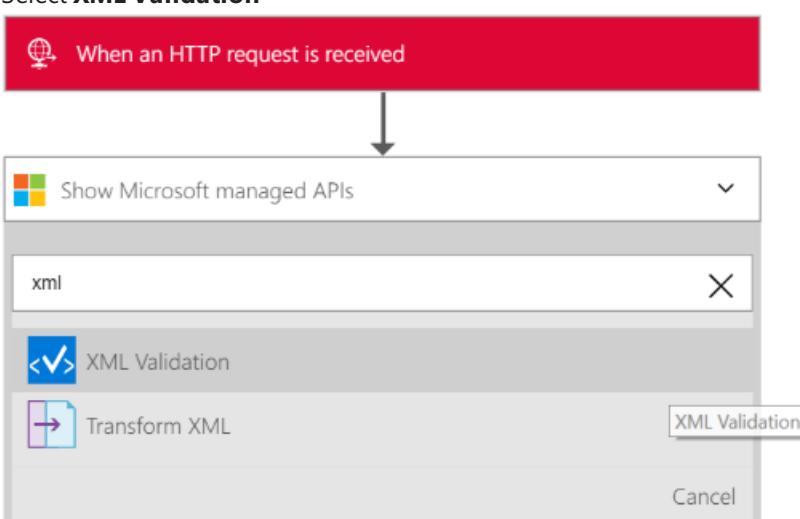
Often, in B2B scenarios, the partners to an agreement need to validate that messages they exchange among each other are valid before processing of the data can begin. In the Enterprise Integration Pack, you can use the XML Validation connector to validate documents against a predefined schema.

How to validate a document with the XML Validation connector

1. Create a Logic app and [link it to your integration account](#) that contains the schema you will use to validate the XML data.
2. Add a **Request - When an HTTP request is received** trigger to your Logic app



3. Add the **XML Validation** action by first selecting **Add an action**
4. Enter *xml* in the search box in order to filter all the actions to the one that you want to use
5. Select **XML Validation**



6. Select the **CONTENT** text box

XML Validation

CONTENT*

The XML content to validate.

You can insert data from previous steps...
Outputs from When an HTTP request is received

Body

SCHEMA NAME

The name of the XML schema to use from the associated integrat... ▾

7. Select the body tag as the content that will be validated.

When an HTTP request is received

↓

XML Validation

CONTENT*

Body

You can insert data from previous steps...
Outputs from When an HTTP request is received

Body

SCHEMA NAME

The name of the XML schema to use from the associated integrat... ▾

8. Select the **SCHEMA NAME** list box and chose the schema you want to use to validate the input content above

When an HTTP request is received

↓

XML Validation

CONTENT*

Body

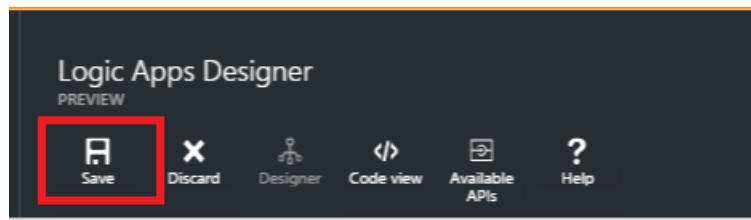
You can insert data from previous steps...
Outputs from When an HTTP request is received

Body

SCHEMA NAME

The name of the XML schema to use from the associated integrat... ▾

9. Save your work



At this point, you are finished setting up your validation connector. In a real world application, you may want to store the validated data in an LOB application such as SalesForce. You can easily add an action to send the output of the validation to Salesforce.

You can now test your validation action by making a request to the HTTP endpoint.

Next steps

[Learn more about the Enterprise Integration Pack](#)

Learn about certificates and Enterprise Integration Pack

1/25/2017 • 2 min to read • [Edit on GitHub](#)

Overview

Enterprise integration uses certificates to secure B2B communications. You can use two types of certificates in your enterprise integration apps:

- Public certificates, which must be purchased from a certification authority (CA).
- Private certificates, which you can issue yourself. These certificates are sometimes referred to as self-signed certificates.

What are certificates?

Certificates are digital documents that verify the identity of the participants in electronic communications and that also secure electronic communications.

Why use certificates?

Sometimes B2B communications must be kept confidential. Enterprise integration uses certificates to secure these communications in two ways:

- By encrypting the contents of messages
- By digitally signing messages

How do you upload certificates?

Public certificates

To use a *public certificate* in your logic apps with B2B capabilities, you first need to upload the certificate into your integration account. To use a *self-signed certificate*, on the other hand, you must first upload it to [Azure Key Vault](#).

After you upload a certificate, it's available to help you secure your B2B messages when you define their properties in the [agreements](#) that you create.

Here are the detailed steps for uploading your public certificates into your integration account after you sign in to the Azure portal:

1. Select **More services** and enter **integration** in the filter search box. Select **Integration Accounts** from the results list

Preview Microsoft Azure Integration Accounts

Shift+Space to toggle favorites

integration

All resources

Resource groups

App Services

SQL databases

SQL data warehouses

NoSQL (DocumentDB)

Virtual machines

Load balancers

Storage accounts

Virtual networks

Azure Active Directory

Monitor

Azure Advisor

Security Center

Billing

Help + support

More services >

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various service icons and names. A red box highlights the 'More services >' button at the bottom of this sidebar. At the top right, there's a search bar with the word 'integration' typed into it, also enclosed in a red box. Below the search bar, the 'Integration Accounts' service is listed with a star icon next to it, indicating it's a favorite.

2. Select the integration account to which you want to add the certificate.

Integration Accounts

Microsoft

Add Columns Refresh

Subscriptions: 1 of 5 selected

Filter by name...

146 items

NAME	TYPE	RESOURCE GROUP	LOCATION
EIPIntegration	Integration Account	EIPTemplates	West US
FabrikamIntegrationAccount	Integration Account	as2tryit	West US
FabrikamIntegrationAccount	Integration Account	JonsIARG	Brazil South

The screenshot shows the 'Integration Accounts' blade in the Microsoft Azure portal. It lists three integration accounts: 'EIPIntegration', 'FabrikamIntegrationAccount', and 'FabrikamIntegrationAccount'. The first account is in the 'West US' region under the 'EIPTemplates' resource group. The second account is in the 'West US' region under the 'as2tryit' resource group. The third account is in 'Brazil South' under the 'JonsIARG' resource group. A red box highlights the 'EIPIntegration' account.

3. Select the **Certificates** tile.

The screenshot shows the Azure portal interface for the 'EIPIntegration' Integration Account. On the left, there's a navigation menu with options like 'Overview', 'Access control (IAM)', 'Tags', 'SETTINGS', 'Callback URL', 'Schemas', 'Maps', 'Certificates', 'Partners', and 'Agreements'. The 'Certificates' option is highlighted with a red box. The main content area has a 'Essentials' section with details: Resource group (EIPTemplates), Location (West US), Subscription ID (redacted). Below that is a 'Components' section with five categories: Schemas (11), Maps (6), Certificates (0, highlighted with a red box), Partners (16), and Agreements (3).

4. In the **Certificates** blade that opens, select the **Add** button.

The screenshot shows the 'Certificates' blade for the 'EIPIntegration' account. At the top, it says 'Certificates' and 'EIPIntegration'. Below that is a toolbar with 'Add' (highlighted with a red box), 'Edit', and 'Delete'. The main area has columns for 'NAME', 'TYPE', and 'CREATED TIME'. A message at the bottom says 'There are no certificate to display.'

5. Enter a **Name** for your certificate, and then select the certificate type as **public** from the dropdown.
6. Select the folder icon on the right side of the **Certificate** text box. When the file picker opens, find and select the certificate file that you want to upload to your integration account.
7. Select the certificate, and then select **OK** in the file picker. This validates and uploads the certificate to your integration account.
8. Finally, back on the **Add certificate** blade, select the **OK** button.

Add Certificate

* Name
publiccert

* Certificate Type
Public

* Certificate
aiissit.cer

Resource Group

* Key Vault

* Key name
Filter

OK

9. Select the **Certificates** tile. You should see the newly added certificate.

The screenshot shows the EIP Integration interface with two main panes. On the left, the 'Essentials' pane displays resource group information: Name (EIIntegration), Location (West US), and Subscription ID (redacted). Below it, the 'Components' section has tiles for Schemas (11), Maps (6), Certificates (1, highlighted with a red box), Partners (16), and Agreements (3). On the right, a 'Certificates' table lists the newly added certificate:

NAME	TYPE	CREATED TIME
publiccert	Public	1/24/2017 2:49 PM

Private certificates

You can upload private certificates into your integration account by taking the following steps:

1. [Upload your private key to Key Vault](#) and provide a **Key Name**

TIP

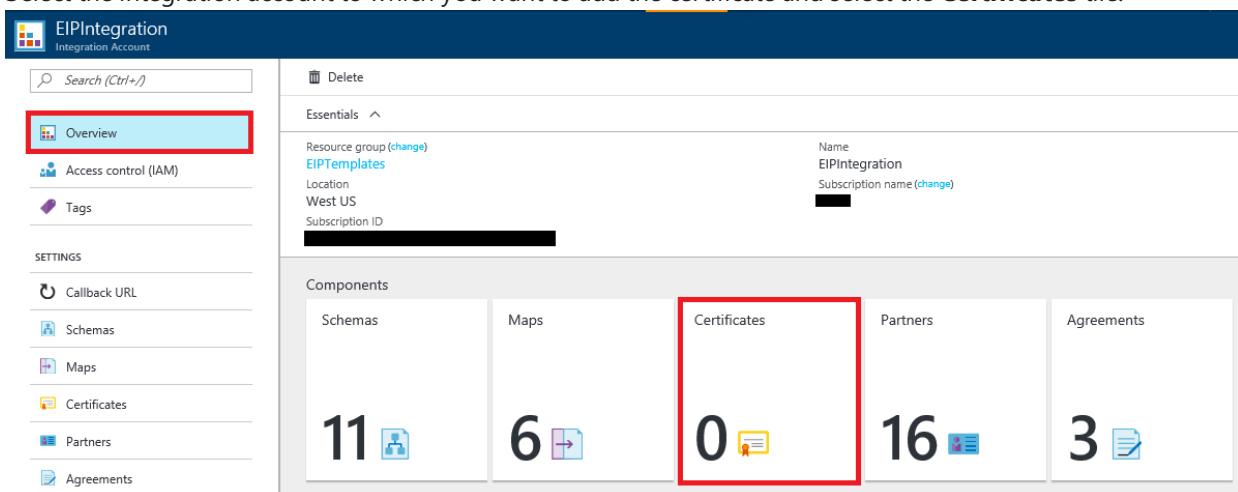
You must authorize Logic Apps to perform operations on Key Vault. You can grant access to the Logic Apps service principal by using the following PowerShell command:

```
Set-AzureRmKeyVaultAccessPolicy -VaultName 'TestcertKeyVault' -ServicePrincipalName '7cd684f4-8a78-49b0-91ec-6a35d38739ba' -PermissionsToKeys decrypt, sign, get, list
```

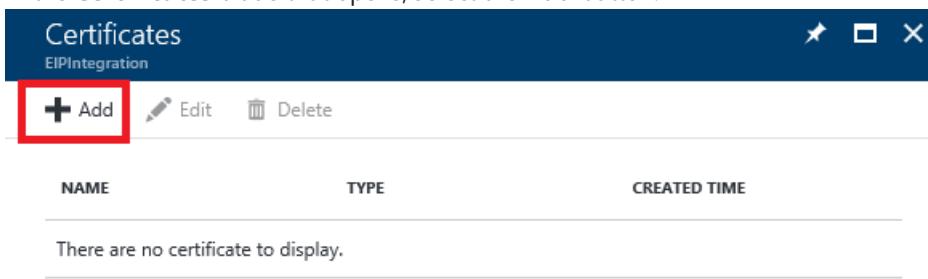
After you've taken the previous step, add a private certificate to integration account.

Following are the detailed steps for uploading your private certificates into your integration account after you sign in to the Azure portal:

1. Select the integration account to which you want to add the certificate and select the **Certificates** tile.



2. In the **Certificates** blade that opens, select the **Add** button.



3. Enter a **Name** for your certificate, and select the certificate type as **private** from the dropdown.

4. select the folder icon on the right side of the **Certificate** text box. When the file picker opens, find the corresponding public certificate that you want to upload to your integration account.

NOTE

While adding a private certificate it is important to add corresponding public certificate to show in [AS2 agreement receive](#) and [send](#) settings for signing and encrypting the messages.

5. Select the **Resource Group, Key Vault, Key Name** and select the **OK** button.

Add Certificate

* Name
privatecert ✓

* Certificate Type
Private

Certificate
aissit.cer

aissit.cer

Resource Group
EIPTemplates

* Key Vault
padmakv

* Key name
Filter

OK

6. Select the **Certificates** tile. You should see the newly added certificate.

The screenshot shows the Azure portal interface. On the left, there's a dashboard with various tiles: Schemas (11), Maps (6), Certificates (2, highlighted with a red box), Partners (16), and Agreements (3). To the right, a blade titled "Certificates" is open under the "EIIntegration" resource group. It shows a table with two rows:

NAME	TYPE	CREATED TIME
privatecert	Private	1/24/2017 3:16 PM
publiccert	Public	1/24/2017 2:49 PM

- [Create a B2B agreement](#)
- [Learn more about Key Vault](#)

Learn about partners and Enterprise Integration Pack

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Overview

Before you can create a partner, you and the organization you intend to do business with must share information that will help you both identify and validate messages that are sent by each other. After you have these discussion, and you are ready to begin your business relationship, you can to create a *partner* in your integration account.

What is a partner?

Partners are the entities that participate in Business-To-Business (B2B) messaging and transactions.

How are partners used?

Partners are used to create agreements. An agreement defines the details about the messages that will be exchanged between partners.

Before you can create an agreement, you need to have added at least two partners to your integration account. One of the partners to an agreement must be your organization. The partner that represents your organization is referred to as the **host partner**. The second partner would represent the other organization with which your organization exchanges messages. The second partner is known as the **guest partner**. The guest partner can be another company, or even a department within your own organization.

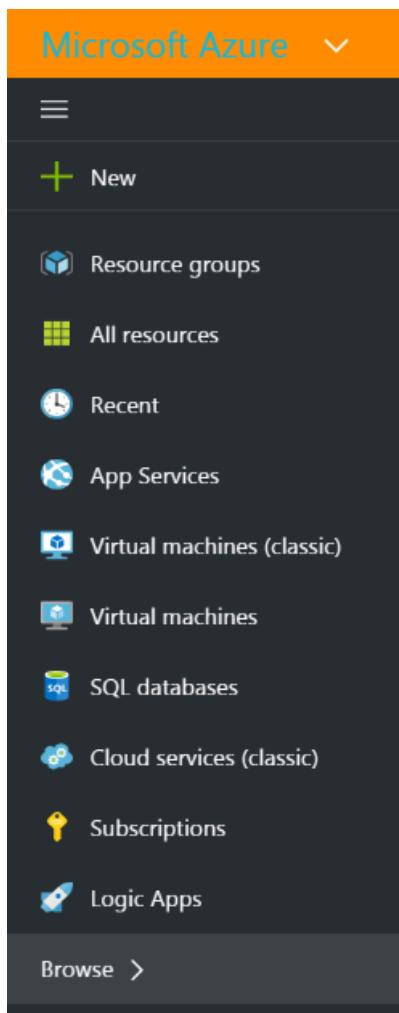
After you have added the partners, you would use those partners to create an agreement.

Receive and Send settings are oriented from the point of view of the Hosted Partner. For example, the receive settings in an agreement determine how the hosted partner receives messages sent from a guest partner. Likewise, the send settings on the agreement indicate how the hosted partner sends messages to the guest partner.

How to create a partner?

From the Azure portal:

1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list

A screenshot of a search interface. At the top is a red search bar with the word "integration". Below it is a white search results area with a grey header bar containing the text "Shift+Space to toggle favorites" and a search input field with "integration" in it. Underneath is a list item with a blue square icon, the text "Integration Accounts", and a "Preview" button. To the right of the preview button is a grey star icon.

3. Select the **integration account** to which you will add the partners

A screenshot of the "Integration Accounts" list page. The title is "Integration Accounts" and it says "Microsoft - PREVIEW". There are three buttons at the top: "Add" (with a plus sign), "Columns" (with a grid icon), and "Refresh" (with a circular arrow). Below this is a section titled "Subscriptions: All 3 selected" with a "Filter items..." input field and a "All subscriptions" button. The main table has columns "NAME", "RESOURCE GROUP", and "LOCATION". It contains two rows: one for "DeonheIntegrationAccount" in "DeonheResGroup" located in "Brazil South", and another for "MyNewIntegrationAccount" in "MyNewRG" located in "Brazil South".

4. Select the **Partners** tile

The screenshot shows the 'MyNewIntegrationAccount' integration account in preview mode. At the top, there are 'Move' and 'Delete' buttons. Below the header, there's a 'Essentials' section with a dropdown arrow, followed by user and settings icons. The 'Components' section contains four tiles: 'Schemas' (0), 'Maps' (0), 'Certificates' (0), and 'Partners' (0). The 'Partners' tile is highlighted with a blue border. Below the tiles is a button labeled 'Add a section +'. At the bottom right of the main area is a link 'All settings →'.

Resource group
Name
MyNewRG
Location
Subscription
brazilsouth
ICBCS9
Subscription ID
1217a102-55fc-461a-9e2d-56a0aacc2972

All settings →

Components

Add tiles +

Schemas 0

Maps 0

Certificates 0

Partners 0

Agreements 0

Add a section +

5. Select the **Add** button in the Partners blade that opens

The screenshot shows the 'Partners' blade for the 'MyNewIntegrationAccount'. It features a header with the title 'Partners' and the account name. Below the header are 'Add' and 'Delete' buttons. A search bar at the top right has the placeholder 'TYPE'. A large 'Add' button is highlighted with a blue border. Below the buttons, a message states 'There are no partners to display.'

Partners

MyNewIntegrationAccount - PREVIEW

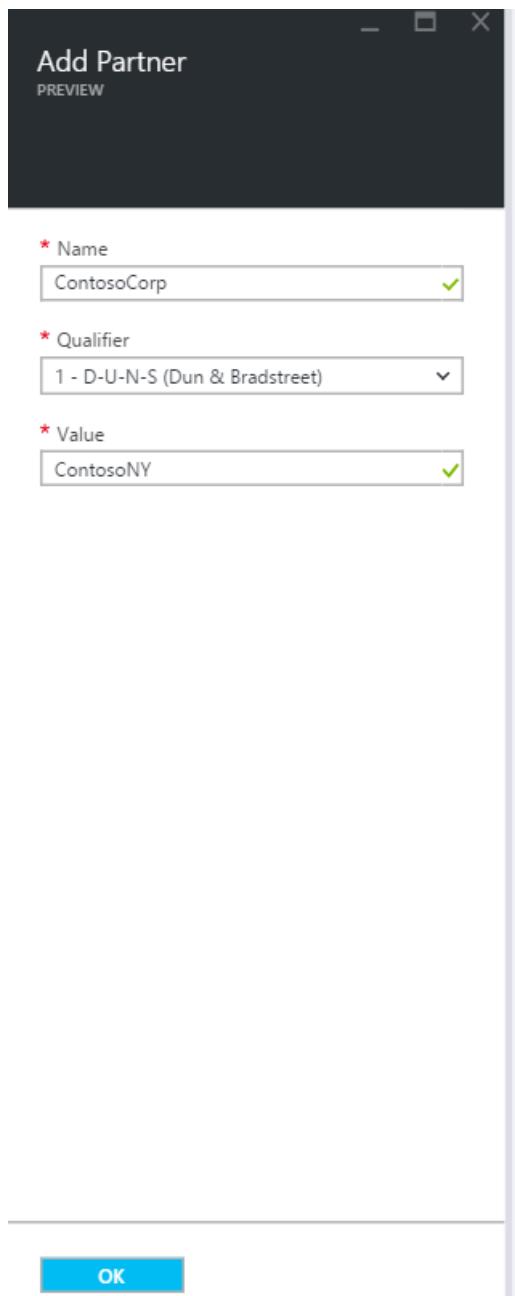
Add Delete

TYPE

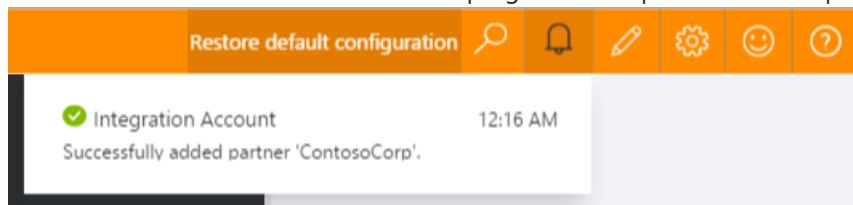
Add

There are no partners to display.

6. Enter a **Name** for your partner, then select the **Qualifier ****, **finally, enter a **Value**. The value is used to help identify documents that come into your apps.



7. Select the *bell* notification icon to see the progress of the partner creation process.



8. Select the **Partners** tile. This refreshes the tile and you should see the number of partners increase, reflecting the new partner has been added successfully.

The screenshot shows the 'MyNewIntegrationAccount' integration account preview. At the top, there are 'Move' and 'Delete' buttons. Below the title, there's a 'Essentials' section with resource group details: MyNewRG, Location: brazilsouth, and Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972. To the right are 'Name' (MyNewIntegrationAccount), 'Subscription' (ICBCS9), and a 'All settings' link. The main area is titled 'Components' with an 'Add tiles +' button. It contains six tiles: 'Schemas' (0), 'Maps' (0), 'Certificates' (0), 'Partners' (1, highlighted with a blue border), 'Agreements' (0), and an 'Add a section +' button.

9. After you select the **Partners** tile, you will also see the newly added partner displayed in the Partners blade.

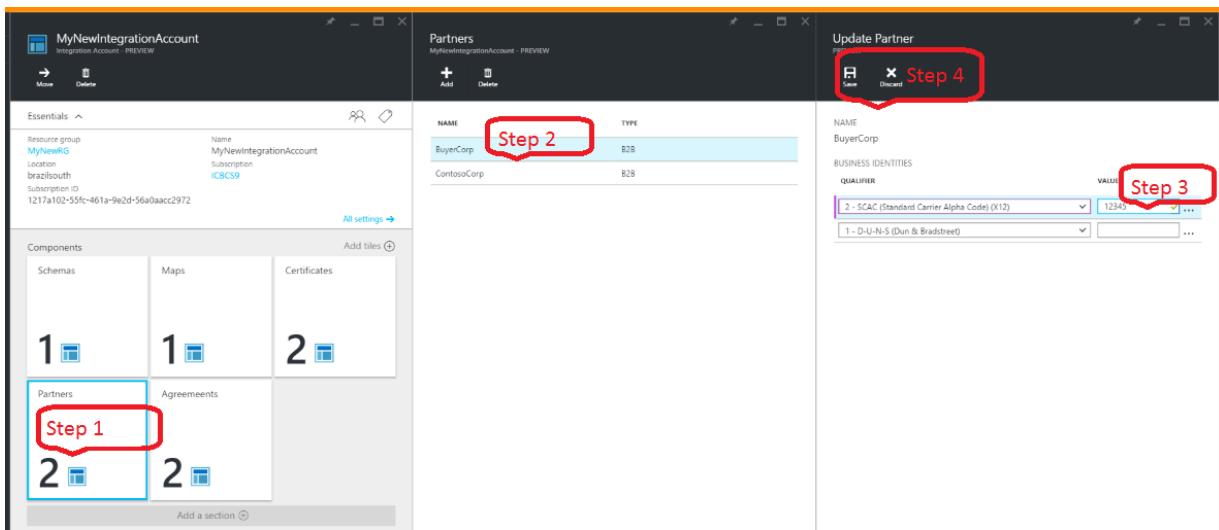
The screenshot shows the 'Partners' blade for the 'MyNewIntegrationAccount' integration account. It has 'Add' and 'Delete' buttons at the top. The table below lists one partner:

NAME	TYPE
ContosoCorp	B2B

How to edit a partner

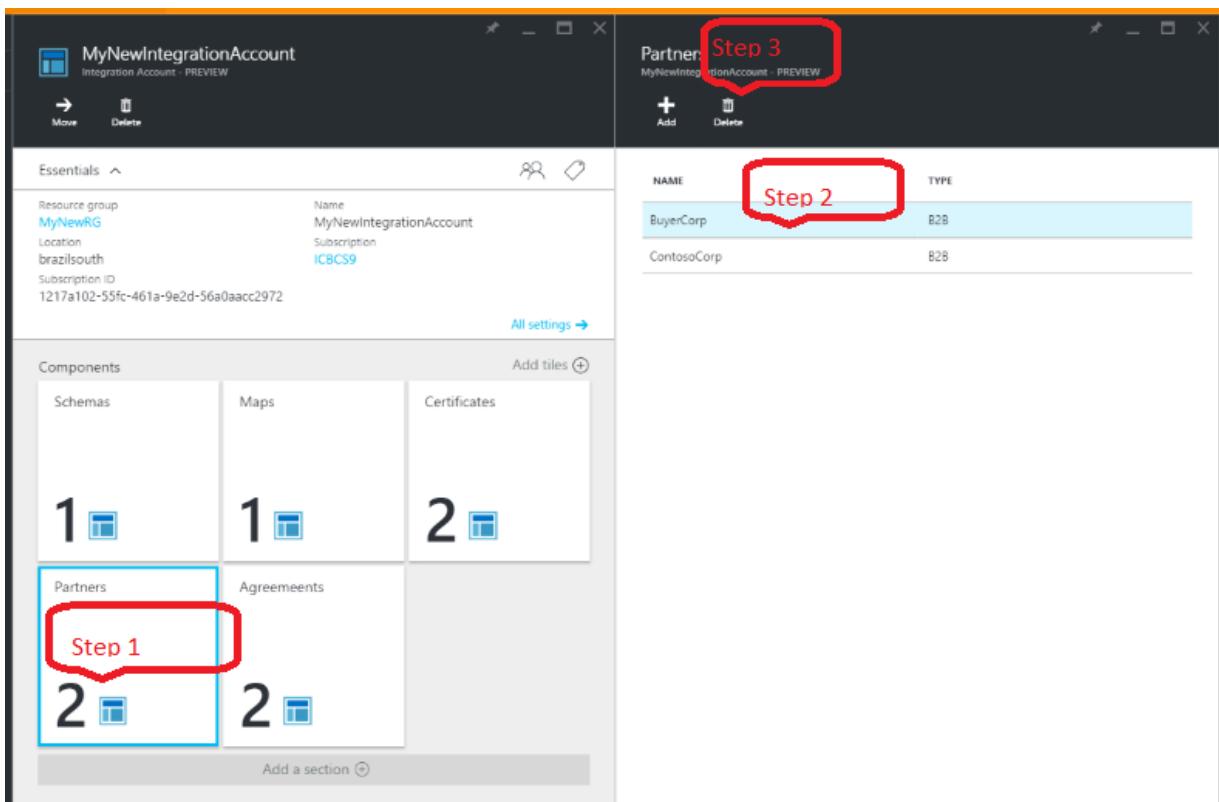
Follow these steps to edit a partner that already exists in your integration account:

1. Select the **Partners** tile
2. Select the partner you wish to edit when the Partners blade opens up
3. On the **Update Partner** tile, make the changes you need
4. If you are satisfied with your changes, select the **Save** link, else, select the **Discard** link to throw away your changes.



How to delete a partner

1. Select the **Partners** tile
2. Select the partner you wish to edit when the Partners blade opens up
3. Select the **Delete** link



Next steps

- Learn more about agreements

Learn about schemas and the Enterprise Integration Pack

1/20/2017 • 2 min to read • [Edit on GitHub](#)

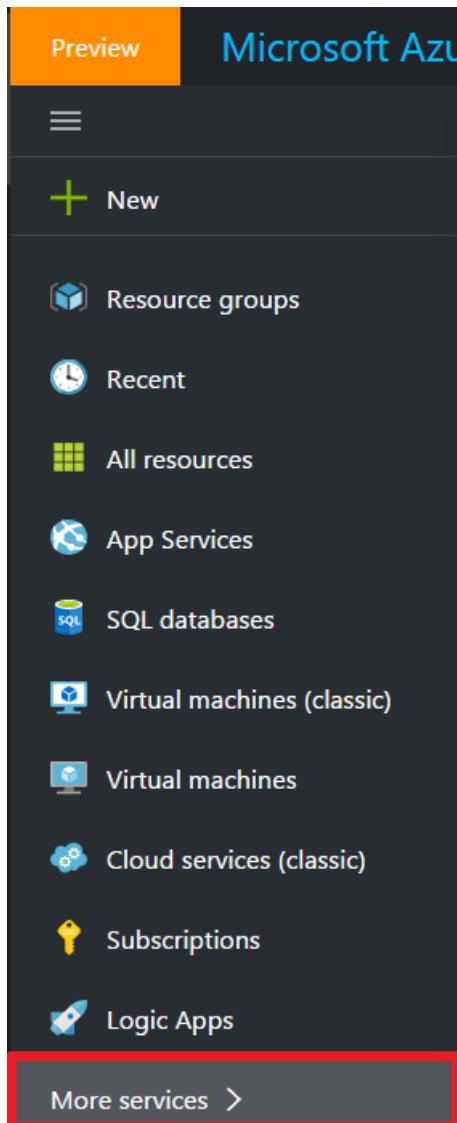
Why use a schema?

Use schemas to confirm that XML documents you receive are valid, with the expected data in a predefined format. Schemas are used to validate messages that are exchanged in a B2B scenario.

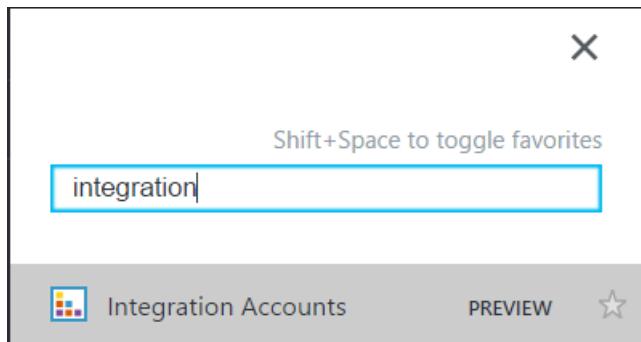
Add a schema

From the Azure portal:

1. Select **More services**.



2. In the filter search box, enter **integration**, and select **Integration Accounts** from the results list.



3. Select the **integration account** to which you add the schema.

The screenshot shows the "Integration Accounts" blade in the Azure portal. The title bar says "Integration Accounts" and "Microsoft - PREVIEW". It features a "Subscriptions: 1 of 5 selected" message, a "Filter items..." input field, and a search bar with the value "BTS4". A table lists five integration accounts:

NAME	TYPE	RESOURCE GROUP
EIPIntegration	Integration Account	EIPTemplates
FabrikamIntegrationAccount	Integration Account	as2tryit
FabrikamIntegrationAccount	Integration Account	JonsIARG
FabrikamIntegrationAccount	Integration Account	padas2try

4. Select the **Schemas** tile.

The screenshot shows the 'Essentials' section of the EIPIntegration Integration Account - PREVIEW. It displays basic account information: Resource group (EIPTemplates), Location (West US), Subscription ID (redacted), Name (EIPIntegration), and Subscription name (BTS4). Below this is a 'Components' section with six items:

Component	Count	Icon
Schemas	0	File icon
Maps	2	Map icon
Certificates	0	Certificate icon
Partners	5	Person icon
Agreements	3	Document icon

Add a schema file less than 2 MB

1. In the **Schemas** blade that opens (from the preceding steps), select **Add**.

The screenshot shows the 'Schemas' blade for the EIPIntegration integration account. At the top, there are buttons for '+ Add' (highlighted with a red border), 'Edit', 'Download', and 'Delete'. Below is a table listing existing schemas:

NAME	TYPE	CONTENT SIZE	CHANGED TIME
EFACT_d10b_CUSDEC	Xml	1.33 MiB	8/22/2016, 11:15 AM
EFACT_D97A_ORDERS	Xml	675.01 KiB	8/22/2016, 11:06 AM
X1200401850	Xml	1.34 MiB	7/27/2016, 10:53 AM

2. Enter a name for your schema. Then, to upload the schema file, select the folder icon next to the **Schema** text box. After the upload process is completed, select **OK**.

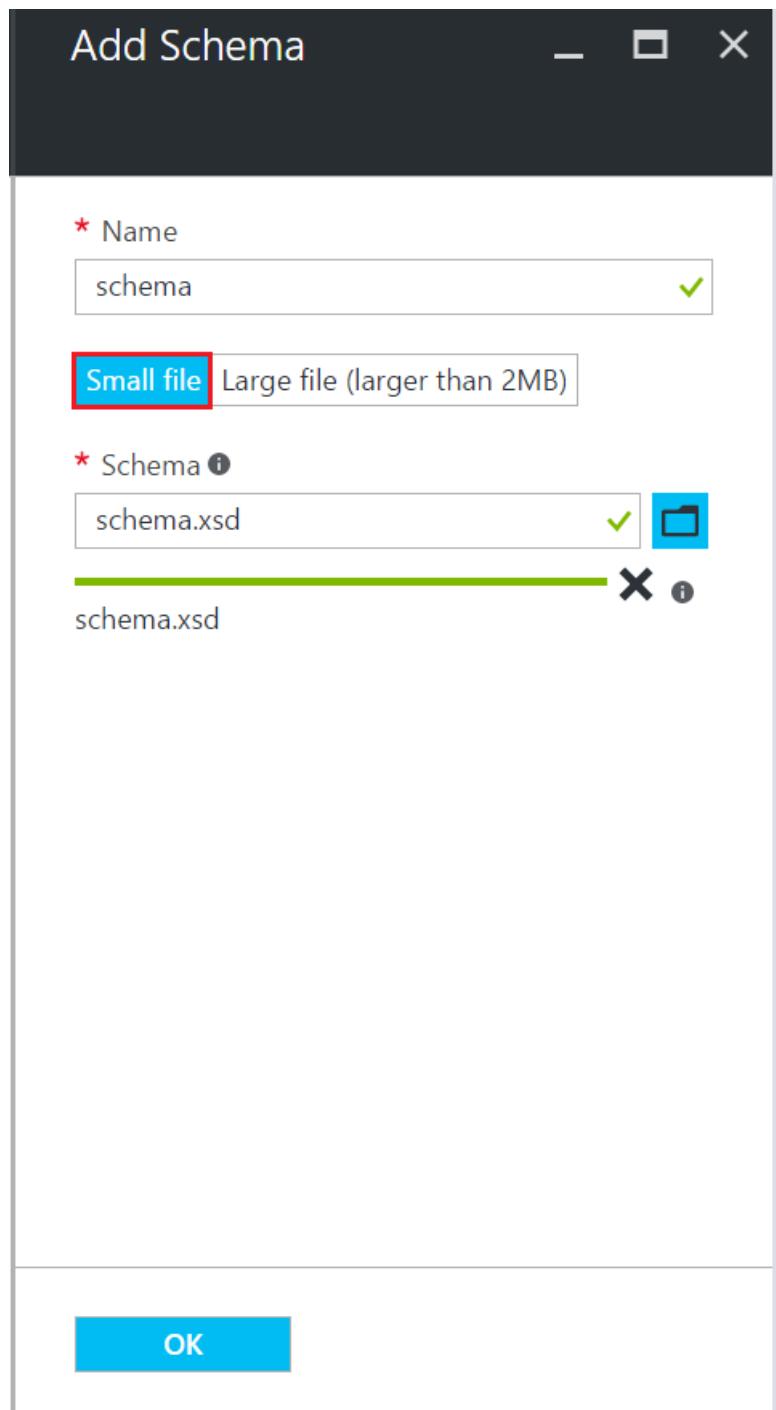
Add Schema

* Name
schema ✓

Small file Large file (larger than 2MB)

* Schema ⓘ
schema.xsd ✓ 
 X ⓘ schema.xsd

OK



Add a schema file larger than 2 MB (up to a maximum of 8 MB)

The process for this depends on the blob container access level: **Public** or **No anonymous access**. To determine this access level, in **Azure Storage Explorer**, under **Blob Containers**, select the blob container you want. Select **Security**, and select the **Access Level** tab.

1. If the blob security access level is **Public**, follow these steps.

Azure Storage Explorer interface showing the blob container 'aaa' in the 'padstorageoms' account. The 'Security' button is highlighted. The 'Access Level' dropdown shows 'Public Container: Anonymous clients can read blob and container content/metadata' selected.

a. Upload the schema to storage, and copy the URI.

Storage Account interface showing the blob 'X1200401855.xsd' in the 'aaa' container. The 'View' button is highlighted. The 'Content' tab shows the blob type as 'Block' and the 'Snapshot Qualified Storage Uri' as 'https://padstorageoms.blob.core.windows.net/aaa/X1200401855.'

b. In **Add Schema**, select **Large file**, and provide the URI in the **Content URI** text box.

Schemas interface showing the 'Add Schema' dialog. The 'Large file (larger than 2MB)' option is selected in the 'Content URI' dropdown. The 'Content URI' field contains the copied blob URI.

2. If the blob security access level is **No anonymous access**, follow these steps.

a. Upload the schema to storage.

Name	Type	Last Modified	Length	Content Type
X1200401810.xsd	Block	9/16/2016 2:28:58 PM +00:00	2.25M	application/octet-stream
X1200401855.xsd	Block	9/12/2016 9:51:01 PM +00:00	3.35M	application/octet-stream
papiNetCommonDefsV2	Block	9/15/2016 2:59:34 PM +00:00	790.40K	application/octet-stream

b. Generate a shared access signature for the schema.

c. In **Add Schema**, select **Large file**, and provide the shared access signature URI in the **Content URI** text box.

The screenshot shows the 'Schemas' blade of the EIP Integration Account. On the left, there's a table with columns: NAME, TYPE, CONTENT SIZE, and CHANGED TIME. It lists three schemas: 'EFACT_d10b_CUSDEC' (Xml, 1.33 MiB, 8/22/2016, 11:15 AM), 'EFACT_D97A_ORDERS' (Xml, 675.01 KiB, 8/22/2016, 11:06 AM), and 'X1200401850' (Xml, 1.34 MiB, 7/27/2016, 10:53 AM). On the right, an 'Add Schema' dialog box is open, prompting for a 'Name' (X1200401855) and a 'Content URI' (https://padstorageoms.blob.core.windows...).

- In the **Schemas** blade of the EIP Integration Account, you should now see the newly added schema.

The screenshot shows the 'Schemas' blade of the EIP Integration Account. The table lists four schemas: 'sapBinding_Common' (Xml, 17.51 KiB, 9/16/2016, 7:31 AM), 'schema' (Xml, 1.34 MiB, 9/13/2016, 1:31 PM), and the newly added 'X1200401855.xsd' (Xml, 1.68 MiB, 9/16/2016, 8:26 AM). The 'Components' section on the left has a 'Schemas' tile highlighted with a red box, showing the count '3'.

Edit schemas

- Select the **Schemas** tile.
- Select the schema you want to edit from the **Schemas** blade that opens.
- On the **Schemas** blade, select **Edit**.

The screenshot shows the 'Schemas' blade of the EIP Integration Account. The 'Edit' button in the top navigation bar is highlighted with a red box. The table lists two schemas: 'schema' (Xml, 1.34 MiB, 9/13/2016, 1:21 PM) and 'schema2' (Xml, 1.68 MiB, 9/13/2016, 1:23 PM). The 'Components' section on the left has a 'Schemas' tile highlighted with a red box, showing the count '2'.

- Select the schema file you want to edit by using the file picker dialog box that opens.

- Select **Open** in the file picker.

NAME	TYPE	CONTENT SIZE	CHANGED TIME
schema	Xml	1.34 MiB	9/13/2016, 11:02 AM
X1200401850	Xml	1.34 MiB	7/27/2016, 10:53 AM

* Name: schema

Small file Large file (larger than 2MB)

* Schema: schema.xsd

- You receive a notification that indicates the upload was successful.

Delete schemas

- Select the **Schemas** tile.
- Select the schema you want to delete from the **Schemas** blade that opens.
- On the **Schemas** blade, select **Delete**.

NAME	TYPE	CONTENT SIZE	CHANGED TIME
schema	Xml	1.34 MiB	9/13/2016, 1:21 PM
schema2	Xml	1.68 MiB	9/13/2016, 1:23 PM

- To confirm your choice, select **Yes**.

Delete schema

Are you sure you want to delete the selected schema?

Yes **No**

- Finally, notice that the list of schemas in the **Schemas** blade refreshes, and the schema you deleted is no longer listed.

EIPIntegration
Integration Account - PREVIEW

Essentials

Resource group: EIPTemplates
Location: West US
Subscription ID: [REDACTED]

Name: EIPIntegration
Subscription name: BTS4

All settings →

Components

Schemas	Maps	Certificates
1	2	0
Partners	Agreements	
5	3	

Schemas

NAME	TYPE	CONTENT SIZE	CHANGED TIME
X1200401850	Xml	1.34 MiB	7/27/2016, 10:53 AM

Next steps

- Learn more about the Enterprise Integration Pack.

Azure Logic Apps Integration Account Metadata

1/20/2017 • 1 min to read • [Edit on GitHub](#)

Overview

Partners, agreements, schemas, maps added to an integration account, store metadata with key-value pairs. You can define custom metadata, which can be retrieved during runtime. Right now the artifacts do not have the capability to create metadata in UI; you can use rest APIs to create them. Partner, agreements, and schema have **EDIT as JSON** and allows for keying metadata information. In a logic app, **Integration Account Artifact LookUp** helps in retrieving the metadata information.

How to Store metadata

1. Create an [Integration Account](#)
2. Add a [partner](#) or an [agreement](#) or a [schema](#) in integration account
3. Select a partner or an agreement, or a schema. select **Edit as JSON** and enter metadata details

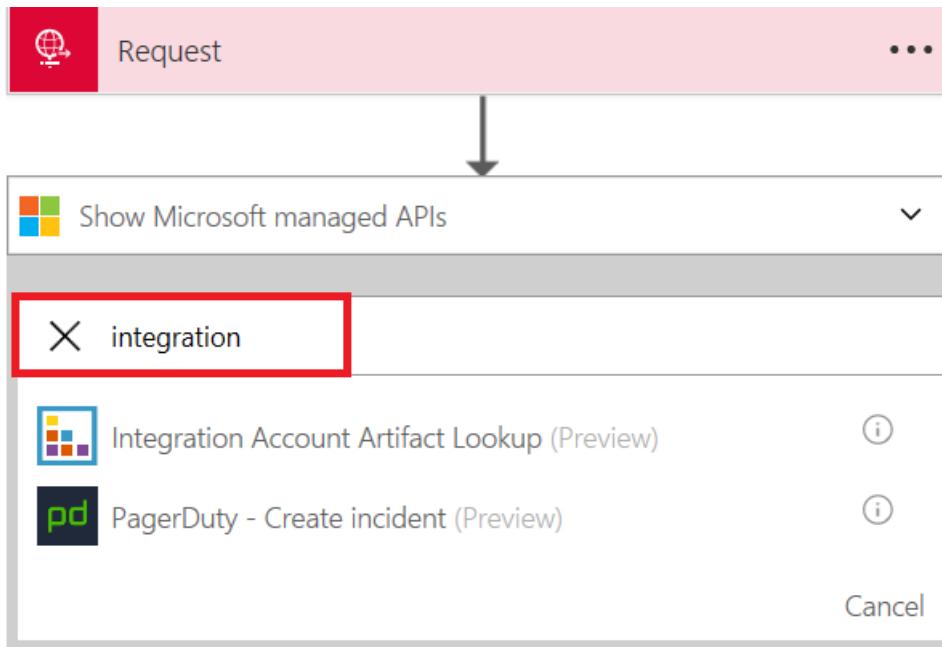
The screenshot shows the Azure portal interface for managing partners in an integration account. On the left, there is a list of partners with columns for NAME and TYPE. One partner, 'fabrikamtest1', is selected and highlighted with a red box. On the right, there is a modal window titled 'Edit as JSON' containing the JSON representation of the selected partner. The JSON code is as follows:

```
1 {
2     "properties": {
3         "partnerType": "B2B",
4         "content": {
5             "b2b": {
6                 "businessIdentities": [
7                     {
8                         "qualifier": "AS2Identity",
9                         "value": "Fabrikamtest"
10                    }
11                ]
12            }
13        },
14        "createdTime": "2016-11-08T17:39:50.5588065Z",
15        "changedTime": "2016-11-18T18:19:27.1080365Z",
16        "metadata": {
17            "name": "Padma",
18            "SAPID": "1234567"
19        }
20    },
21    "id": "/subscriptions/[REDACTED]",
22    "name": "fabrikamtest1",
23    "type": "Microsoft.Logic/integrationAccounts/partners"
24 }
```

The 'Edit as JSON' button in the top-left of the modal and the 'OK' button at the bottom-right are also highlighted with red boxes.

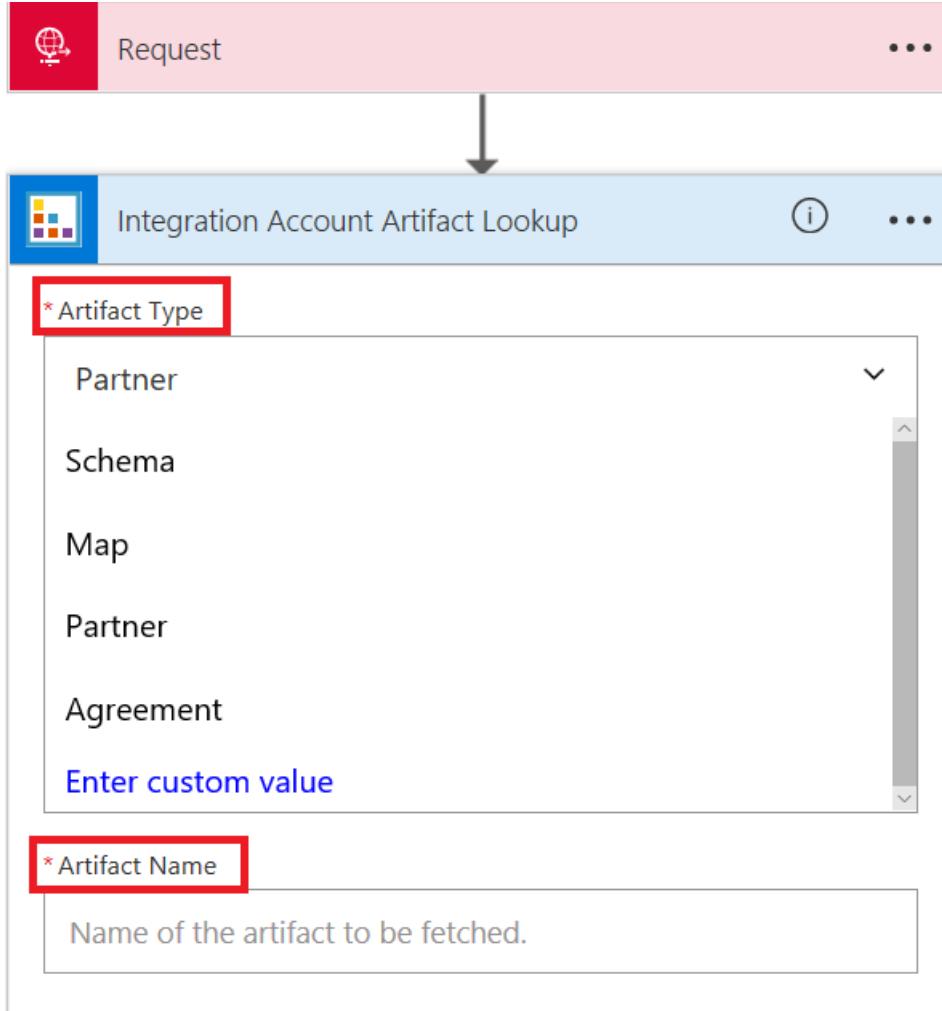
Call **Integration Account Artifact LookUp** from a logic app

1. Create a [Logic App](#)
2. [Link](#) Logic App with an Integration Account
3. Create a trigger, for example using *Request* or *HTTP* before searching for **Integration Account Artifact LookUp**. Search **integration** to look for **Integration Account Artifact LookUp**



4. Select **Integration Account Artifact LookUp**

5. Select **Artifact Type** and provide **Artifact Name**



An example to retrieve partner metadata

1. Partner metadata has routing url details

Edit as JSON

```
1 [
2     "properties": {
3         "partnerType": "B2B",
4         "content": {
5             "b2b": {
6                 "businessIdentities": [
7                     {
8                         "qualifier": "AS2Identity",
9                         "value": "Fabrikamtest"
10                    }
11                ]
12            }
13        },
14        "createdTime": "2016-11-08T17:39:50.5588065Z",
15        "changedTime": "2016-11-21T17:49:48.7499957Z",
16        "metadata": {
17            "name": "Padma",
18            "SAPID": "1234567",
19            "routingUrl": "http://[REDACTED]"
20        }
21    },
22    "id": "/subscriptions/[REDACTED]/resource",
23    "name": "fabrikamtest1",
24    "type": "Microsoft.Logic/integrationAccounts/partners"
25 }
```

2. In a logic app configure **Integration Account Artifact LookUp** and **HTTP**

```
graph TD; A[Request] --> B[Integration Account Artifact Lookup]; B --> C[HTTP]
```

The logic app flow consists of three main steps:

- Request**: The first step in the flow.
- Integration Account Artifact Lookup**: The second step, triggered by the previous step. It has a dynamic content panel on the right side:

 - Add dynamic content from the apps and services used in this flow.**
 - Search dynamic content** input field.
 - Integration Account Artifact Lookup** section:
 - Body**: Contains the expression `metadata.routingUrl`.
 - Name**: Contains the expression `Properties`.
 - Properties**: Contains the expression `Properties`.

- HTTP**: The third step, triggered by the previous step. It has configuration fields:
 - *Method**: Set to `POST`.
 - *Uri**: Contains the dynamic content `metadata.routingUrl`.
 - Headers**: Contains the dynamic content `Headers`.
 - Body**: Contains the dynamic content `Properties`.

3. To retrieve URL, the code view should look like

```
"actions": {
    "HTTP": {
        "inputs": {
            "body": "@outputs('Integration_Account_Artifact_Lookup')['properties']",
            "headers": "@triggerOutputs()['headers']",
            "method": "POST",
            "uri": "@{outputs('Integration_Account_Artifact_Lookup')['properties']['metadata']['routingUrl']}"
        },
        "runAfter": {
            "Integration_Account_Artifact_Lookup": [
                "Succeeded"
            ]
        }
    }
},
```

Next steps

- [Learn more about agreements](#)

Enterprise integration with AS2

1/25/2017 • 5 min to read • [Edit on GitHub](#)

Create an AS2 agreement

To use the enterprise features in Logic apps, you must first create agreements.

Here's what you need before you get started

- An [integration account](#) defined in your Azure subscription
- At least two [partners](#) already defined in your integration account

NOTE

When creating an agreement, the content in the agreement file must match the agreement type.

After you've [created an integration account](#) and [added partners](#), you can create an agreement by following these steps:

From the Azure portal home page

After you log in to the [Azure portal](#):

1. Select **More services** and enter **integration** in the filter search box. Select **Integration Accounts** from the results list

Preview Microsoft Azure Integration Accounts

New All resources Resource groups App Services SQL databases SQL data warehouses NoSQL (DocumentDB) Virtual machines Load balancers Storage accounts Virtual networks Azure Active Directory Monitor Azure Advisor Security Center Billing Help + support More services >

Shift+Space to toggle favorites

integration

Integration Accounts

2. Select the integration account to which you want to add the certificate.

Integration Accounts Microsoft - PREVIEW

Add Columns Refresh

Subscriptions: All 3 selected

Filter items... All subscriptions

NAME	RESOURCE GROUP	LOCATION
DeonheIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South

3. Select the **Agreements** tile. If you don't see the agreements tile, add it first.

The screenshot shows the Azure portal's Integration Accounts blade for the 'EIPIntegration' account. On the left, there's a navigation menu with 'Overview' selected. The main area displays 'Essentials' information like Resource group (EIPTemplates), Location (West US), and Subscription ID. Below that is a 'Components' section with five categories: Schemas (11), Maps (6), Certificates (1), Partners (16), and Agreements (0). The 'Agreements' section is highlighted with a red box.

4. Select the **Add** button in the Agreements blade that opens.

The screenshot shows the 'Add' blade for creating a new agreement. It has tabs for 'Edit' and 'Edit as JSON'. The main form fields are: Name (highlighted with a red box), Agreement type (set to AS2), Host Partner (dropdown), Host Identity (dropdown), Guest Partner (dropdown), and Guest Identity (dropdown). Below the form are sections for 'Receive Settings' and 'Send Settings' with expand arrows.

5. Enter a **Name** for your agreement, select **AS2** from the **Agreement Type, Host Partner, Host Identity, Guest Partner, Guest Identity** in the Agreements blade.

Add

*** Name**
AS2agreement ✓

*** Agreement type**
AS2

*** Host Partner**
Contoso

*** Host Identity**
AS2Identity : Contoso

*** Guest Partner**
Fabrikam

*** Guest Identity**
AS2Identity : Fabrikam

Receive Settings >

Send Settings >

OK

Here are a few details you may find useful when configuring the settings for your agreement:

PROPERTY	DESCRIPTION
Host Partner	An agreement needs both a host and guest partner. The host partner represents the organization that is configuring the agreement.
Host Identity	An identifier for the host partner.
Guest Partner	An agreement needs both a host and guest partner. The guest partner represents the organization that's doing business with the host partner.
Guest Identity	An identifier for the guest partner.
Receive Settings	These properties apply to all messages received by an agreement

PROPERTY	DESCRIPTION
Send Settings	These properties apply to all messages sent by an agreement

Let's continue:

1. Select **Receive Settings** to configure how messages received via this agreement are to be handled.
 - Optionally, you can override the properties in the incoming message. To do this, select the **Override message properties**.
 - Select the **Message should be signed** if you'd like to require all incoming messages to be signed. If you select this option, you need to select the *guest partner public certificate* to validate the signature on the messages.
 - Select the **Message should be encrypted** if you'd like to require all incoming messages to be encrypted. If you select this option, you need to select the *host partner private certificate* to decrypt the incoming messages.
 - You can also require messages to be compressed. To do this, select the **Message should be compressed**.
 - Select the **Send MDN** to send sync MDN for the messages received
 - Select the **Send signed MDN** to send signed MDN for the messages received
 - Select the **Send asynchronous MDN** to send async MDN for the messages received

Add

* Name
AS2agreement

* Agreement type
AS2

* Host Partner
Contoso

* Host Identity
AS2Identity : Contoso

* Guest Partner
Fabrikam

* Guest Identity
AS2Identity : Fabrikam

Receive Settings >

Send Settings >

Receive Settings

Override message properties ⓘ

Message

Message should be signed

* Certificate
publiccert

Message should be encrypted

* Certificate
privatecert

Message should be compressed

Acknowledgement

MDN Text
Contoso got your message

Send MDN

Send signed MDN

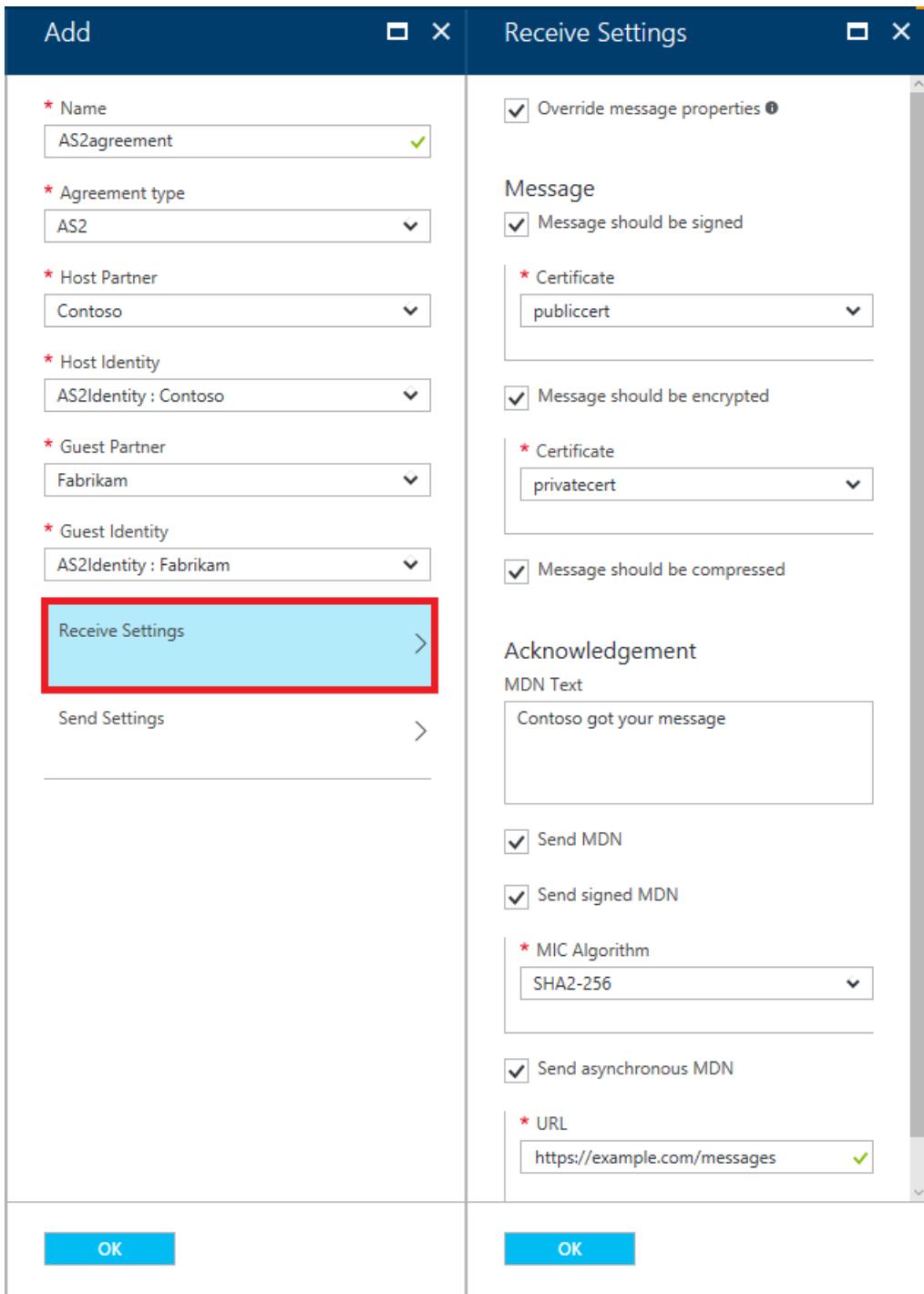
* MIC Algorithm
SHA2-256

Send asynchronous MDN

* URL
<https://example.com/messages>

OK

OK



See the table below if you would like to learn more about what the receive settings enable.

PROPERTY	DESCRIPTION
Override message properties	Select this to indicate that properties in received messages can be overridden
Message should be signed	Enable this to require messages to be digitally signed. Configure the guest partner public certificate for signature verification
Message should be encrypted	Enable this to require messages to be encrypted. Non-encrypted messages will be rejected. Configure host partner private certificate to decrypt the messages
Message should be compressed	Enable this to require messages to be compressed. Non-compressed messages will be rejected.

PROPERTY	DESCRIPTION
MDN Text	This is a default MDN to be sent to the message sender
Send MDN	Enable this to allow MDNs to be sent.
Send signed MDN	Enable this to require MDNs to be signed.
MIC Algorithm	
Send asynchronous MDN	Enable this to require messages to be sent asynchronously.
URL	This is the URL to which the MDNs will be sent.

Now, let's continue:

1. Select **Send Settings** to configure how messages sent via this agreement are to be handled.
 - Select the **Enable message signing** to send signed messages to the partner. If you select this option, you need to select the *host partner private certificate MIC Algorithm* and *host partner private certificate* to sign the messages.
 - Select the **Enable message encryption** to send encrypted messages to the partner. If you select this option, you need to select the *guest partner public certificate algorithm* and *guest partner public certificate* to encrypt the messages.
 - Select the **Message should be compressed** to compress the message
 - Select the **Unfold HTTP headers** to unfold the HTTP content-type header into a single line
 - Select the **Request MDN** to receive sync MDN for the messages sent
 - Select the **Request signed MDN** to receive signed MDN for the messages sent
 - Select the **Request asynchronous MDN** to receive async MDN for the messages sent. If you select this option, you need to provide a URL to which MDNs are sent
 - Select the **Enable NRR** to enable the non-repudiation of receipt

Add

* Name
AS2agreement ✓

* Agreement type
AS2

* Host Partner
Contoso

* Host Identity
AS2Identity : Contoso

* Guest Partner
Fabrikam

* Guest Identity
AS2Identity : Fabrikam

Receive Settings >

Send Settings >

Send Settings

Messages

Enable message signing

* MIC Algorithm
SHA1

* Certificate
privatecert

Enable message encryption

* Encryption Algorithm
DES3

* Certificate

Enable message compression

Unfold HTTP headers

Acknowledgement

Request MDN

Request signed MDN

Request asynchronous MDN

* URL
https://example.com/messages ✓

NRR Status

Enable NRR

OK
OK

See the table below if you would like to learn more about what the send settings enable.

PROPERTY	DESCRIPTION
Enable message signing	Select this to enable all messages sent from the agreement to be signed.
MIC Algorithm	Select the algorithm to use for message signing. Configure the host partner private certificate MIC Algorithm to sign the messages
Certificate	Select the certificate to use for message signing. Configure the host partner private certificate to sign the messages
Enable message encryption	Select this to encrypt all messages sent from this agreement. Configure the guest partner public certificate algorithm to encrypt the messages

PROPERTY	DESCRIPTION
Encryption Algorithm	Select the encryption algorithm to use for message encryption. Configure the guest partner public certificate to encrypt the messages
Unfold HTTP headers	Select this to unfold the HTTP content-type header into a single line
Request MDN	Enable this to request an MDN for all messages sent from this agreement
Request signed MDN	Enable to request that all MDNs sent to this agreement are signed
Request asynchronous MDN	Enable to request asynchronous MDN to be sent to this agreement
URL	The URL to which MDNs will be sent
Enable NRR	Select this to enable Non-Repudiation of Receipt

Select the **Agreements** tile on the Integration Account blade and you will see the newly added agreement listed.

The screenshot shows the Azure portal interface for managing an Integration Account named 'EIPIntegration'. On the left, there's a summary card for the resource group 'EIPTemplates' located in 'West US' with a subscription ID starting 'f34b22a3-2202-4fb1-b040-1332bd928c84'. Below this are sections for 'Components' (Schemas: 11, Maps: 6, Certificates: 2) and 'Partners' (16). The 'Agreements' tile is highlighted with a red box and contains the number '1', indicating one agreement has been added. To the right, the 'Agreements' blade is open, showing a table with one row:

NAME	TYPE	HOST PARTNER	GUEST PARTNER
AS2agreement	AS2	Contoso	Fabrikam

Next Steps

- Learn more about the Enterprise Integration Pack

Get started with Encode AS2 Message

1/20/2017 • 1 min to read • [Edit on GitHub](#)

Connect to Encode AS2 Message to establish security and reliability while transmitting messages. It provides digital signing, encryption, and acknowledgements via Message Disposition Notifications (MDN), which also leads to support for Non-Repudiation.

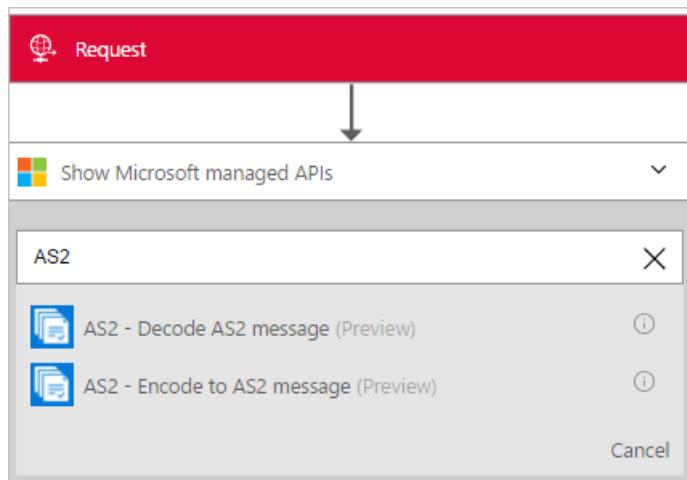
Create the connection

Prerequisites

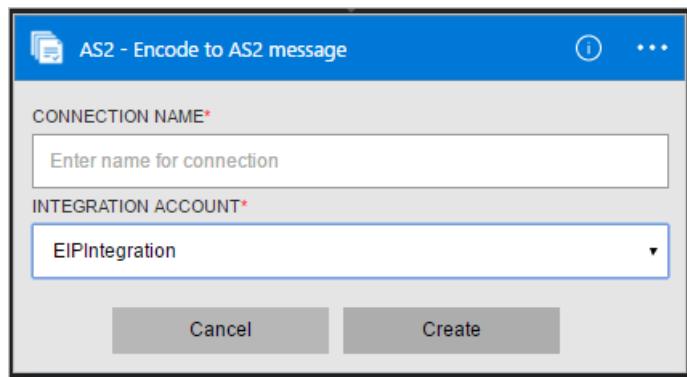
- An Azure account; you can create a [free account](#)
- An Integration Account is required to use Encode AS2 message connector. See details on how to create an [Integration Account](#), [partners](#) and an [AS2 agreement](#)

Connect to Encode AS2 Message using the following steps:

1. [Create a Logic App](#) provides an example
2. This connector does not have any triggers. Use other triggers to start the Logic App, such as a Request trigger. In the Logic App designer, add a trigger and add an action. Select Show Microsoft managed APIs in the drop-down list and then enter "AS2" in the search box. Select AS2 – Encode AS2 Message



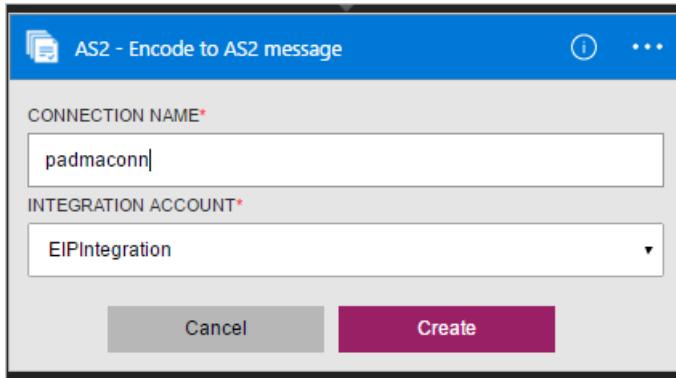
3. If you haven't previously created any connections to Integration Account, you are prompted for the connection details



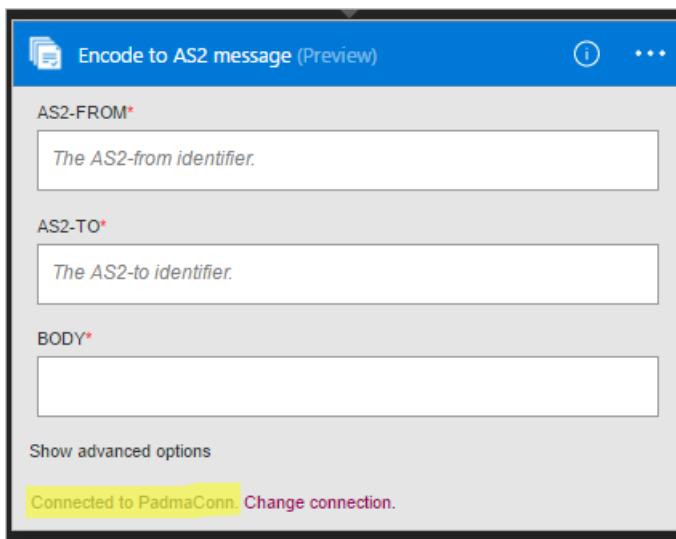
4. Enter the Integration Account details. Properties with an asterisk are required

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection
Integration Account *	Enter the Integration Account name. Be sure your Integration Account and Logic app are in the same Azure location

Once complete, your connection details look similar to the following



5. Select **Create**
6. Notice the connection has been created. Provide AS2-From, AS2-To identifiers (as configured in agreement) and Body (the message payload) details.



The AS2 Encode does the following

- Applies AS2/HTTP headers
- Signs outgoing messages (if configured)
- Encrypts outgoing messages (if configured)
- Compresses the message (if configured)

Try it for yourself

Why not give it a try. Click [here](#) to deploy a fully operational logic app of your own using the Logic Apps AS2 features

Next steps

[Learn more about the Enterprise Integration Pack](#)

Get started with Decode AS2 Message

1/20/2017 • 1 min to read • [Edit on GitHub](#)

Connect to Decode AS2 Message to establish security and reliability while transmitting messages. It provides digital signing, decryption, and acknowledgements via Message Disposition Notifications (MDN).

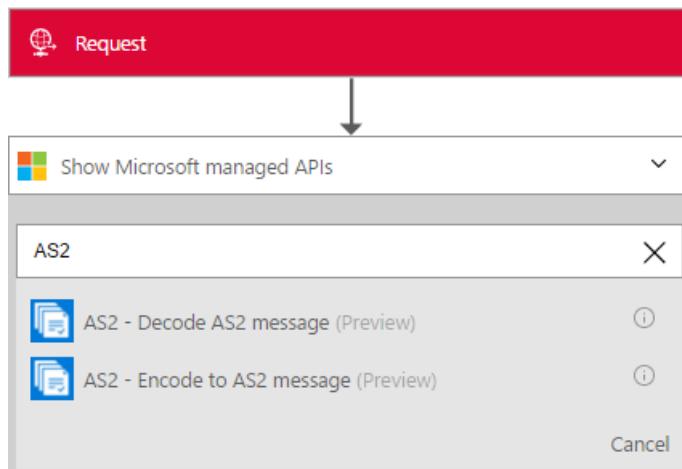
Create the connection

Prerequisites

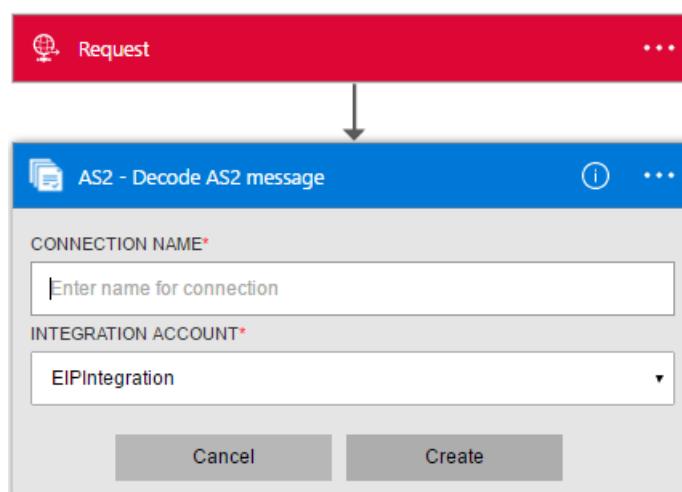
- An Azure account; you can create a [free account](#)
- An Integration Account is required to use Decode AS2 message connector. See details on how to create an [Integration Account](#), [partners](#) and an [AS2 agreement](#)

Connect to Decode AS2 Message using the following steps:

1. [Create a Logic App](#) provides an example.
2. This connector does not have any triggers. Use other triggers to start the Logic App, such as a Request trigger. In the Logic App designer, add a trigger and add an action. Select Show Microsoft managed APIs in the drop-down list and then enter "AS2" in the search box. Select AS2 – Decode AS2 Message



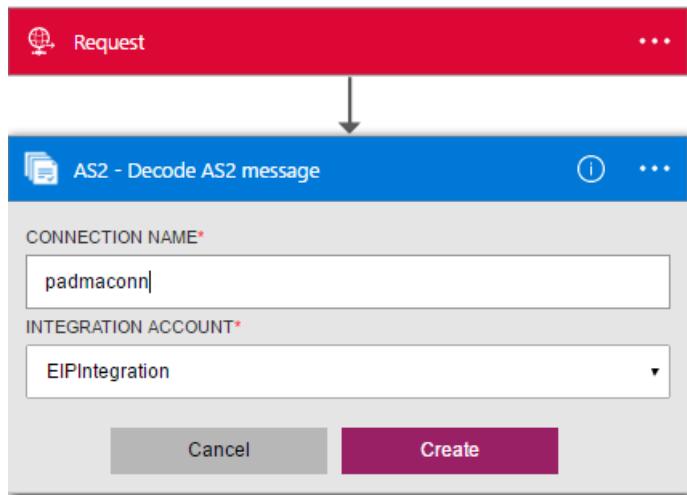
3. If you haven't previously created any connections to Integration Account, you are prompted for the connection details



4. Enter the Integration Account details. Properties with an asterisk are required

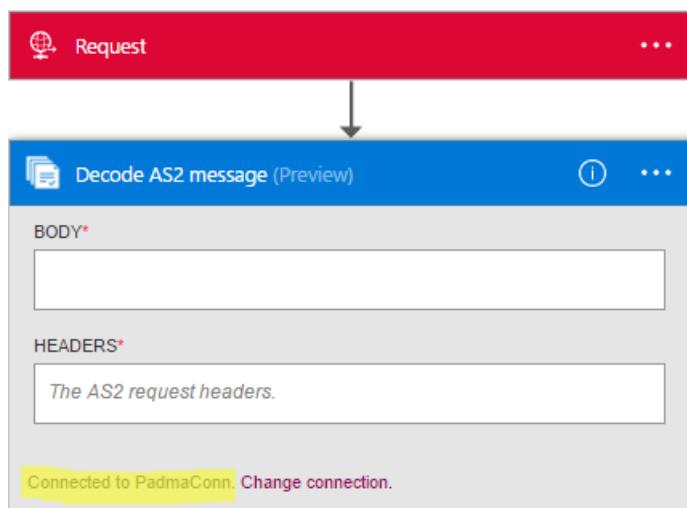
PROPERTY	DETAILS
Connection Name *	Enter any name for your connection
Integration Account *	Enter the Integration Account name. Be sure your Integration Account and Logic app are in the same Azure location

Once complete, your connection details look similar to the following

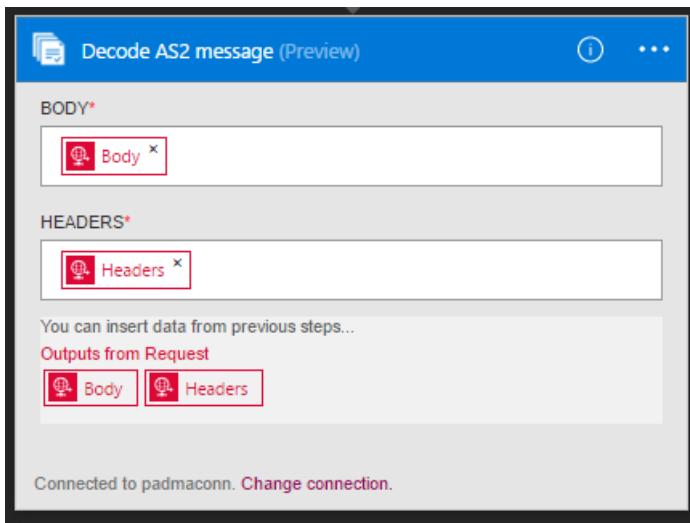


5. Select **Create**

6. Notice the connection has been created. Now, proceed with the other steps in your Logic App



7. Select Body and Headers from Request outputs



The AS2 Decode does the following

- Processes AS2/HTTP headers
- Verifies the signature (if configured)
- Decrypts the messages (if configured)
- Decompresses the message (if configured)
- Reconciles a received MDN with the original outbound message
- Updates and correlates records in the non-repudiation database
- Writes records for AS2 status reporting
- The output payload contents are base64 encoded
- Determines whether an MDN is required, and whether the MDN should be synchronous or asynchronous based on configuration in AS2 agreement
- Generates a synchronous or asynchronous MDN (based on agreement configurations)
- Sets the correlation tokens and properties on the MDN

Try it for yourself

Why not give it a try. Click [here](#) to deploy a fully operational logic app of your own using the Logic Apps AS2 features

Next steps

[Learn more about the Enterprise Integration Pack](#)

Enterprise integration with EDIFACT

1/20/2017 • 12 min to read • [Edit on GitHub](#)

NOTE

This page covers the EDIFACT features of Logic Apps. See [X12](#) for more information.

Create an EDIFACT agreement

Before you can exchange EDIFACT messages, you need to create an EDIFACT agreement and store it in your integration account. The following steps will walk you through the process of creating an EDIFACT agreement.

Here's what you need before you get started

- An [integration account](#) defined in your Azure subscription
- At least two [partners](#) already defined in your integration account

NOTE

When creating an agreement, the content in the messages you will receive/send to and from the partner must match the agreement type.

After you've [created an integration account](#) and [added partners](#), you can create an EDIFACT agreement by following these steps:

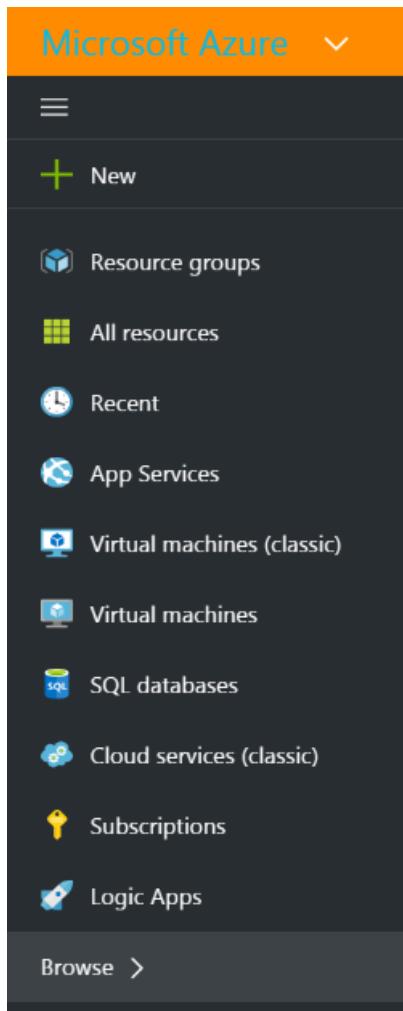
From the Azure portal home page

After you log into the [Azure portal](#):

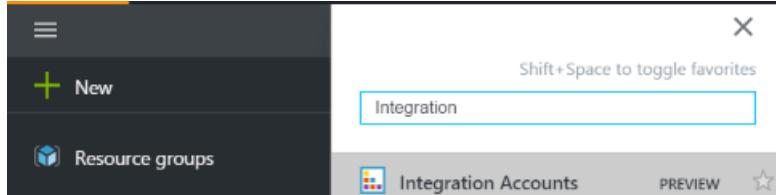
1. Select **Browse** from the menu on the left.

TIP

If you don't see the **Browse** link, you may need to expand the menu first. Do this by selecting the **Show menu** link that's located at the top left of the collapsed menu.



1. Type *integration* into the filter search box then select **Integration Accounts** from the list of results.



2. In the **Integration Accounts** blade that opens up, select the integration account in which you will create the agreement. If you don't see any integration accounts lists, [create one first](#).

The screenshot shows the 'Integration Accounts' blade in the Azure portal. The title bar says 'Integration Accounts' with a 'PREVIEW' link. Below the title are buttons for '+ Add', 'Columns', and 'Refresh'. A status bar at the top left says 'Subscriptions: 1 of 3 selected'. The main area is a table with the following data:

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
DeonheEIPNCentralIntegrationAccount	Integration Account	DeonheNCentralResGroyp	North Central US	ICBCS9
DeonheIntegrationAccount	Integration Account	DeonheResGroup	Brazil South	ICBCS9
MyNewIntegrationAccount	Integration Account	MyNewRG	Brazil South	ICBCS9

3. Select the **Agreements** tile. If you don't see the agreements tile, add it first.

4. Select the **Add** button in the Agreements blade that opens.

NAME	TYPE	HOST PARTNER	GUEST PARTNER
WarrantySalesAgreem...	AS2	ContosoCorp	BuyerCorp

5. Enter a **Name** for your agreement then select the **Agreement type** for EDIFACT, **Host Partner**, **Host Identity**, **Guest Partner**, **Guest Identity**, in the Agreements blade that opens.

Name: WidgetAgreement

Agreement type: EDIFACT

Host Identity: [dropdown]

Guest Partner: [dropdown]

Guest Identity: [dropdown]

Receive Settings

Send Settings

6. After you have set the agreement properties, select **Receive Settings** to configure how messages received via

this agreement are to be handled.

7. The Receive Settings control is divided into the following sections, including Identifiers, Acknowledgment, Schemas, Control Numbers, Validation, Internal Settings and Batch processing. Configure these properties based on your agreement with the partner you will be exchanging messages with. Here is a view of these controls, configure them based on how you want this agreement to identify and handle incoming messages:

Receive Settings

Identifiers

UNB6.1 (Recipient Reference Password)

UNB6.2 (Recipient Reference Qualifier)

Acknowledgement

Receipt of Message (CTRL) Acknowledgement (CTRL)

Schemas

UNH2.1 (TYPE) UNH2.2 (VERSION) UNH2.3 (RELEASE) UNH2.5 (ASSOCIAT...) UNH2.1 (APP SEND...) UNH2.2 (APP SEND...) SCHEMA

No results

No schema... ...

Control Numbers

Disallow Interchange control number duplicates

Check for duplicate UNB5 every (days)

30

Disallow Group control number duplicates

Disallow Transaction set control number duplicates

EDIFACT Acknowledgement Control Number to

Validations

MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAILING SPACES	TRIM LEADING/TRAILING SPACES	TRAILING SEPARATOR...
--------------	----------------	---------------------	-------------------------------	------------------------------	-----------------------

Default	true	false	false	false	NotAllowed	<input type="button"/> ...
<input type="button" value="APERAK"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button" value="Not Allowed"/>	<input type="button"/> ...

Internal Settings

Create empty XML tags if trailing separators are allowed

Inbound batch processing

- Split Interchange as transaction sets - suspend transaction sets on error
- Split Interchange as transaction sets - suspend interchange on error
- Preserve Interchange - suspend transaction sets on error
- Preserve Interchange - suspend interchange on error

8. Select the **OK** button to save your settings.

Identifiers

PROPERTY	DESCRIPTION
UNB6.1 (Recipient Reference Password)	Enter an alphanumeric value ranging between 1 and 14 characters.
UNB6.2 (Recipient Reference Qualifier)	Enter an alphanumeric value with a minimum of one character and a maximum of two characters.

Acknowledgments

PROPERTY	DESCRIPTION
Receipt of Message (CTRL)	Select this checkbox to return a technical (CTRL) acknowledgment to the interchange sender. The acknowledgment is sent to the interchange sender based on the Send Settings for the agreement.
Acknowledgement (CTRL)	Select this checkbox to return a functional (CTRL) acknowledgment to the interchange sender. The acknowledgment is sent to the interchange sender based on the Send Settings for the agreement.

Schemas

PROPERTY	DESCRIPTION
UNH2.1 (TYPE)	Select a transaction set type.
UNH2.2 (VERSION)	Enter the message version number. (Minimum, one character; maximum, three characters).
UNH2.3 (RELEASE)	Enter the message release number. (Minimum, one character; maximum, three characters).
UNH2.5 (ASSOCIATED ASSIGNED CODE)	Enter the assigned code. (Maximum, six characters. Must be alphanumeric).
UNG2.1 (APP SENDER ID)	Enter an alphanumeric value with a minimum of one character and a maximum of 35 characters.
UNG2.2 (APP SENDER CODE QUALIFIER)	Enter an alphanumeric value, with a maximum of four characters.
SCHEMA	Select the previously uploaded schema you want to use from your associated Integration Account.

Control Numbers

PROPERTY	DESCRIPTION
Disallow Interchange Control Number duplicates	Select this checkbox to block duplicate interchanges. If selected, the EDIFACT Decode Action checks that the interchange control number (UNB5) for the received interchange does not match a previously processed interchange control number. If a match is detected, then the interchange is not processed.

PROPERTY	DESCRIPTION
Check for duplicate UNB5 every (days)	If you opted to disallow duplicate interchange control numbers, you can specify the number of days at which the check is performed by giving the appropriate value for Check for duplicate UNB5 every (days) option.
Disallow Group control number duplicates	Select this checkbox to block interchanges with duplicate group control numbers (UNG5).
Disallow Transaction set control number duplicates	Select this checkbox to block interchanges with duplicate transaction set control numbers (UNH1).
EDIFACT Acknowledgement Control Number	To designate the transaction set reference numbers to be used in an acknowledgment, enter a value for the prefix, a range of reference numbers, and a suffix.

Validations

PROPERTY	DESCRIPTION
Message Type	Specify the message type. As each validation row is completed, another will be automatically added. If no rules are specified, then the row marked as default is used for validation.
EDI Validation	Select this check box to perform EDI validation on data types as defined by the EDI properties of the schema, length restrictions, empty data elements, and trailing separators.
Extended Validation	Select this check box to enable extended (XSD) validation of interchanges received from the interchange sender. This includes validation of field length, optionality, and repeat count in addition to XSD data type validation.
Allow Leading/Trailing Zeroes	Select Allow to allow leading/trailing zeros; NotAllowed to not allow leading/trailing zeros, or Trim to trim the leading and trailing zeroes.
Trim Leading/Trailing Zeroes	Select this check box to trim any leading or trailing zeroes
Trailing Separator Policy	Select Not Allowed if you do not want to allow trailing delimiters and separators in an interchange received from the interchange sender. If the interchange contains trailing delimiters and separators, it is declared invalid. Select Optional to accept interchanges with or without trailing delimiters and separators. Select Mandatory if the received interchange must contain trailing delimiters and separators.

Internal Settings

PROPERTY	DESCRIPTION
Create empty XML tags if trailing separators are allowed	Select this check box to have the interchange sender include empty XML tags for trailing separators.

PROPERTY	DESCRIPTION
Inbound batching processing	<p>Options include:</p> <p>Split Interchange as Transaction Sets - suspend Transaction Sets on Error: Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope to the transaction set. With this option, if one or more transaction sets in the interchange fail validation, then only those transaction sets are suspended.</p> <p>Split Interchange as Transaction Sets - suspend Interchange on Error: Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope. With this option, if one or more transaction sets in the interchange fail validation, then the entire interchange will be suspended.</p> <p>Preserve Interchange - suspend Transaction Sets on Error: Preserve Interchange - suspend Interchange on Error: Leaves the interchange intact, creating an XML document for the entire batched interchange. With this option, if one or more transaction sets in the interchange fail validation, then only those transaction sets are suspended, while all other transaction sets are processed.</p>

Your agreement is ready to handle incoming messages that conform to the settings you selected.

To configure the settings that handle messages you send to partners:

1. Select **Send Settings** to configure how messages sent via this agreement are to be handled.

The Send Settings control is divided into the following sections, including Identifiers, Acknowledgment, Schemas, Envelopes, Character Sets and Separators, Control Numbers and Validation.

Here is a view of these controls. Make the selections based on how you want to handle messages you send to partners via this agreement:

Send Settings

Identifiers

UNB1.2 (Syntax Version)	<input type="text" value="3"/>	
-------------------------	--------------------------------	--

UNB2.3 (Sender Reverse Routing Address)	<input type="text" value="UNB2.3 (Sender Reverse Routing Address)"/>
---	--

UNB3.3 (Recipient Reverse Routing Address)	<input type="text" value="UNB3.3 (Recipient Reverse Routing Address)"/>
--	---

UNB6.1 (Recipient Reference Password)	<input type="text" value="UNB6.1 (Recipient Reference Password)"/>
---------------------------------------	--

UNB6.2 (Recipient Reference Qualifier)	<input type="text" value="UNB6.2 (Recipient Reference Qualifier)"/>
--	---

UNB7 (Application Reference ID)	<input type="text" value="UNB7 (Application Reference ID)"/>
---------------------------------	--

Acknowledgement

<input type="checkbox"/> Receipt of Message (CONTRL)	<input type="checkbox"/> Acknowledgement (CONTRL)
--	---

Generate SG1/SG4 loop for accented transaction sets

Schemas

UNH2.1 (TYPE)	UNH2.2 (VERSION)	UNH2.3 (RELEASE)	SCHEMA
No results			
APERAK	▼		No schemas found ▼ ...

Envelopes

UNB8 (Processing Priority Code)	UNB8 (Processing Priority Code)
UNB10 (Communication Agreement)	UNB10 (Communication Agreement)
<input type="checkbox"/> UNB11 (Test Indicator)	
<input type="checkbox"/> Apply UNA Segment (Service String Advice)	
<input type="checkbox"/> Apply UNG Segments (Function Group Header)	

Character Sets and Separators

UNB1.1 (System Identifier)	UNOB	▼
----------------------------	------	---

SCHEMA	INPUT TYPE	COMPONENT...	DATA ELEMENT...	UNA3 (DECIM...	UNA4 (RELEA...	UNA5 (REPETI...	SEGMENT TER...	SUFFIX
Default	:	+	Comma	?	*	'	None	...
No sch... ▾	Char ▾			Decimal ▾			None ▾	...

Control Numbers

UNB5 (Interchange Control Number)	Prefix	1	to	999999999	Suffix
UNG5 (Group Control Number)	Prefix	1	to	999999999	Suffix
UNH1 (Message Header Reference Number)	Prefix	1	to	999999999	Suffix

Validations

MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAILING SPACES	TRIM LEADING/TRAILING SPACES	.TRAILING SEPARATOR
Default	true	false	false	false	NotAllowed
APERAK	▼	□	□	□	Not Allowed ▾

- Select the **OK** button to save your settings.

Identifiers

PROPERTY	DESCRIPTION
UNB1.2 (Syntax version)	Select a value between 1 and 4 .
UNB2.3 (Sender Reverse Routing Address)	Enter an alphanumeric value with a minimum of one character and a maximum of 14 characters.
UNB3.3 (Recipient Reverse Routing Address)	Enter an alphanumeric value with a minimum of one character and a maximum of 14 characters.

PROPERTY	DESCRIPTION
UNB6.1 (Recipient Reference Password)	Enter an alphanumeric value with a minimum of one and a maximum of 14 characters.
UNB6.2 (Recipient Reference Qualifier)	Enter an alphanumeric value with a minimum of one character and a maximum of two characters.
UNB7 (Application Reference ID)	Enter an alphanumeric value with a minimum of one character and a maximum of 14 characters

Acknowledgment

PROPERTY	DESCRIPTION
Receipt of Message (CTRL)	Select this checkbox if the hosted partner expects to receive to receive a technical (CTRL) acknowledgement. This setting specifies that the hosted partner, who is sending the message, requests an acknowledgement from the guest partner.
Acknowledgement (CTRL)	Select this checkbox if the hosted partner expects to receive a functional (CTRL) acknowledgment. This setting specifies that the hosted partner, who is sending the message, requests an acknowledgement from the guest partner.
Generate SG1/SG4 loop for accepted transaction sets	If you chose to request a functional acknowledgement, select this checkbox to force generation of SG1/SG4 loops in functional CTRL acknowledgments for accepted transaction sets.

Schemas

PROPERTY	DESCRIPTION
UNH2.1 (TYPE)	Select a transaction set type.
UNH2.2 (VERSION)	Enter the message version number.
UNH2.3 (RELEASE)	Enter the message release number.
SCHEMA	Select the schema to use. Schemas are located in your integration account. To access your schemas, first link your integration account to your Logic app.

Envelopes

PROPERTY	DESCRIPTION
UNB8 (Processing Priority Code)	Enter an alphabetical value which is not more than one character long.
UNB10 (Communication Agreement)	Enter an alphanumeric value with a minimum of one character and a maximum of 40 characters.

PROPERTY	DESCRIPTION
UNB11 (Test Indicator)	Select this checkbox to indicate that the interchange generated is test data
Apply UNA Segment (Service String Advice)	Select this checkbox to generate a UNA segment for the interchange to be sent.
Apply UNG Segments (Function Group Header)	<p>Select this checkbox to create grouping segments in the functional group header in the messages sent to the guest partner. The following values are used to create the UNG segments:</p> <p>For UNG1, enter an alphanumeric value with a minimum of one character and a maximum of six characters.</p> <p>For UNG2.1, enter an alphanumeric value with a minimum of one character and a maximum of 35 characters.</p> <p>For UNG2.2, enter an alphanumeric value, with a maximum of four characters.</p> <p>For UNG3.1, enter an alphanumeric value with a minimum of one character and a maximum of 35 characters.</p> <p>For UNG3.2, enter an alphanumeric value, with a maximum of four characters.</p> <p>For UNG6, enter an alphanumeric value with a minimum of one and a maximum of three characters.</p> <p>For UNG7.1, enter an alphanumeric value with a minimum of one character and a maximum of three characters.</p> <p>For UNG7.2, enter an alphanumeric value with a minimum of one character and a maximum of three characters.</p> <p>For UNG7.3, enter an alphanumeric value with a minimum of 1 character and a maximum of 6 characters.</p> <p>For UNG8, enter an alphanumeric value with a minimum of one character and a maximum of 14 characters.</p>

Character Sets and Separators

Other than the character set, you can enter a different set of delimiters to be used for each message type. If a character set is not specified for a given message schema, then the default character set is used.

PROPERTY	DESCRIPTION
UNB1.1 (System Identifier)	Select the EDIFACT character set to be applied on the outgoing interchange.

PROPERTY	DESCRIPTION
Schema	<p>Select a schema from the drop-down list. As each row is completed a new row will be added. For the selected schema, select the separators set to be used:</p> <p>Component element separator – Enter a single character to separate composite data elements.</p> <p>Data Element Separator – Enter a single character to separate simple data elements within composite data elements.</p> <p>Replacement Character – Select this check box if the payload data contains characters that are also used as data, segment, or component separators. You can then enter a replacement character. When generating the outbound EDIFACT message, all instances of separator characters in the payload data are replaced with the specified character.</p> <p>Segment Terminator – Enter a single character to indicate the end of an EDI segment.</p> <p>Suffix – Select the character that is used with the segment identifier. If you designate a suffix, then the segment terminator data element can be empty. If the segment terminator is left empty, then you must designate a suffix.</p>

Control Numbers

PROPERTY	DESCRIPTION
UNB5 (Interchange Control Number)	Enter a prefix, a range of values for the interchange control number, and a suffix. These values are used to generate an outgoing interchange. The prefix and suffix are optional; the control number is required. The control number is incremented for each new message; the prefix and suffix remain the same.
UNG5 (Group Control Number)	Enter a prefix, a range of values for the interchange control number, and a suffix. These values are used to generate the group control number. The prefix and suffix are optional; the control number is required. The control number is incremented for each new message until the maximum value is reached; the prefix and suffix remain the same.
UNH1 (Message Header Reference Number)	Enter a prefix, a range of values for the interchange control number, and a suffix. These values are used to generate the message header reference number. The prefix and suffix are optional; the reference number is required. The reference number is incremented for each new message; the prefix and suffix remain the same.

Validations

PROPERTY	DESCRIPTION
Message Type	Selecting this option enables validation on the interchange receiver. This validation performs EDI validation on transaction-set data elements, validating data types, length restrictions, and empty data elements and training separators.

PROPERTY	DESCRIPTION
EDI Validation	Select this check box to perform EDI validation on data types as defined by the EDI properties of the schema, length restrictions, empty data elements, and trailing separators.
Extended Validation	Selecting this option enables extended validation of interchanges received from the interchange sender. This includes validation of field length, optionality, and repeat count in addition to XSD data type validation. You can enable extension validation without enabling EDI validation, or vice versa.
Allow leading/trailing zeroes	Selecting this option specifies that an EDI interchange received from the party does not fail validation if a data element in an EDI interchange does not conform to its length requirement because of or trailing spaces, but does conform to its length requirement when they are removed.
Trim Leading/Trailing Zeroes	Selecting this option will trim the leading and trailing zeroes.
Trailing separator	<p>Selecting this option specifies an EDI interchange received from the party does not fail validation if a data element in an EDI interchange does not conform to its length requirement because of leading (or trailing) zeroes or trailing spaces, but does conform to its length requirement when they are removed.</p> <p>Select Not Allowed if you do not want to allow trailing delimiters and separators in an interchange received from the interchange sender. If the interchange contains trailing delimiters and separators, it is declared invalid.</p> <p>Select Optional to accept interchanges with or without trailing delimiters and separators.</p> <p>Select Mandatory if the received interchange must contain trailing delimiters and separators.</p>

After you select **OK** on the open blade:

1. Select the **Agreements** tile on the Integration Account blade and you will see the newly added agreement listed.

Essentials ^

Resource group
[DeonheEIPNCentralResGroup](#)

Location
North Central US

Subscription ID
1217a102-55fc-461a-9e2d-56a0aacc2972

Name
DeonheEIPNCentralIntegrationAccount

Subscription name
[ICBCS9](#)

All settings →

Components

Add tiles ⓘ

Schemas	Maps	Certificates
2	1	1

Partners	Agreements
2	2

Add a section +

Learn more

- [Learn more about the Enterprise Integration Pack](#)

Get started with Encode EDIFACT Message

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Validates EDI and partner-specific properties

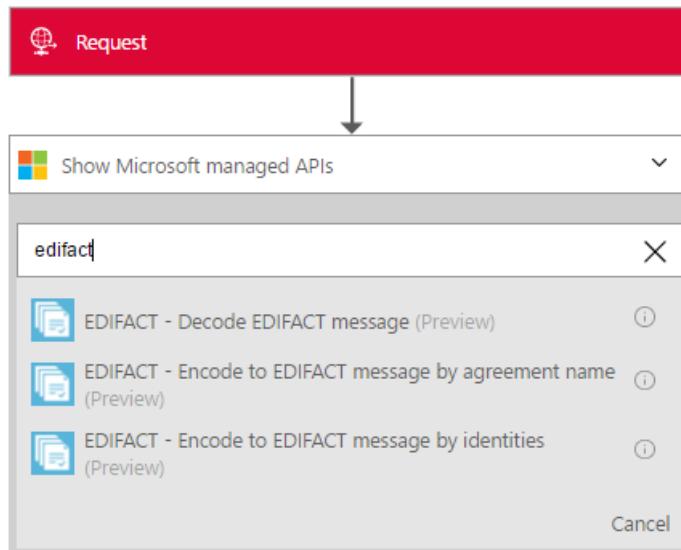
Create the connection

Prerequisites

- An Azure account; you can create a [free account](#)
- An Integration Account is required to use Encode EDIFACT message connector. See details on how to create an [Integration Account](#), [partners](#) and [EDIFACT agreement](#)

Connect to Decode EDIFACT Message using the following steps:

1. [Create a Logic App](#) provides an example.
2. This connector does not have any triggers. Use other triggers to start the Logic App, such as a Request trigger. In the Logic App designer, add a trigger and add an action. Select Show Microsoft managed APIs in the drop-down list and then enter "EDIFACT" in the search box. Select either Encode EDIFACT Message by agreement name or Encode to EDIFACT message by identities.



3. If you haven't previously created any connections to Integration Account, you are prompted for the connection details



4. Enter the Integration account details. Properties with an asterisk are required

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection
Integration Account *	Enter the Integration Account name. Be sure your Integration Account and Logic app are in the same Azure location

Once complete, your connection details look similar to the following

EDIFACT - Encode to EDIFACT message by agreement...

CONNECTION NAME*

Enter name for connection

INTEGRATION ACCOUNT*

EIPIntegration

Cancel Create

5. Select **Create**
6. Notice the connection has been created

Encode to EDIFACT message by agreement name (Pr...)

NAME OF EDIFACT AGREEMENT*

Name of EDIFACT agreement

XML MESSAGE TO ENCODE*

XML message to encode

Connected to padmaconn. Change connection.

Encode EDIFACT Message by agreement name

1. Provide EDIFACT agreement name and xml message to encode.

Encode to EDIFACT message by agreement name (P...)

NAME OF EDIFACT AGREEMENT*

Name of EDIFACT agreement

XML MESSAGE TO ENCODE*

XML message to encode

Connected to padint. Change connection.

Encode EDIFACT Message by identities

1. Provide sender identifier, sender qualifier, receiver identifier, and receiver qualifier as configured in the EDIFACT agreement. Select xml message to encode

Encode to EDIFACT message by identities (Preview) ⓘ ...

SENDER IDENTIFIER*

Sender identifier

SENDER QUALIFIER*

Sender qualifier

RECEIVER IDENTIFIER*

Receiver identifier

RECEIVER QUALIFIER*

Receiver qualifier

XML MESSAGE TO ENCODE*

XML message to encode

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

EDIFACT Encode does following

- Resolve the agreement by matching the sender qualifier & identifier and receiver qualifier and identifier
- Serializes the EDI interchange, converting XML-encoded messages into EDI transaction sets in the interchange.
- Applies transaction set header and trailer segments
- Generates an interchange control number, a group control number, and a transaction set control number for each outgoing interchange
- Replaces separators in the payload data
- Validates EDI and partner-specific properties
 - Schema validation of the transaction-set data elements against the message schema.
 - EDI validation performed on transaction-set data elements.
 - Extended validation performed on transaction-set data elements
- Generates an XML document for each transaction set.
- Requests a Technical and/or Functional acknowledgment (if configured).
 - As a technical acknowledgment, the CONTRL message indicates receipt of an interchange.
 - As a functional acknowledgment, the CONTRL message indicates acceptance or rejection of the received interchange, group, or message, with a list of errors or unsupported functionality

Next steps

[Learn more about the Enterprise Integration Pack](#)

Get started with Decode EDIFACT Message

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Validates EDI and partner-specific properties, generates XML document for each transaction set and generates acknowledgment for processed transaction.

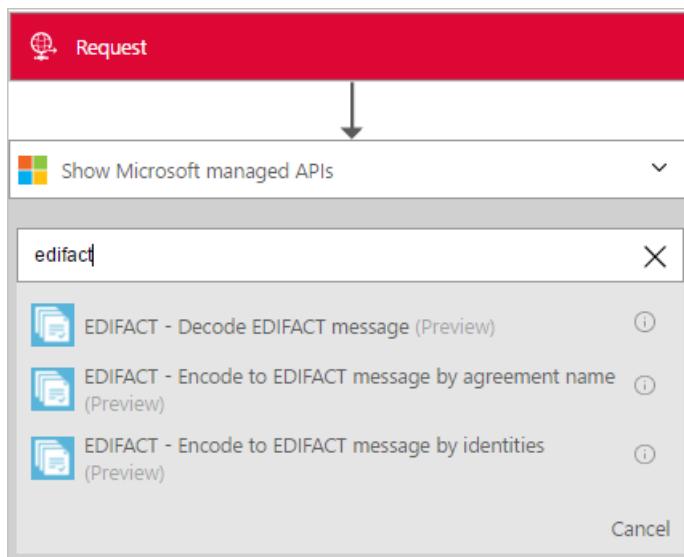
Create the connection

Prerequisites

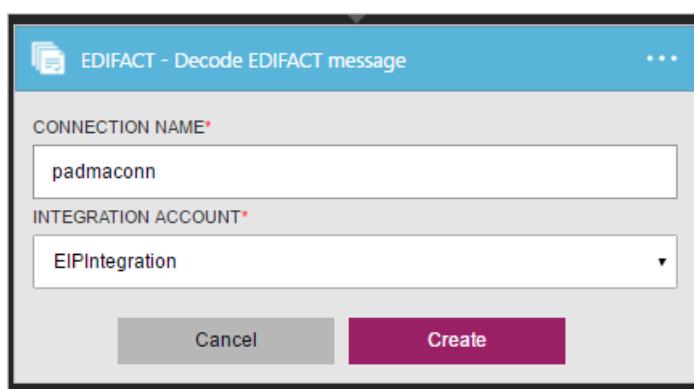
- An Azure account; you can create a [free account](#)
- An Integration Account is required to use Decode EDIFACT message connector. See details on how to create an [Integration Account](#), [partners](#) and [EDIFACT agreement](#)

Connect to Decode EDIFACT Message using the following steps:

1. [Create a Logic App](#) provides an example.
2. This connector does not have any triggers. Use other triggers to start the Logic App, such as a Request trigger. In the Logic App designer, add a trigger and add an action. Select Show Microsoft managed APIs in the drop-down list and then enter "EDIFACT" in the search box. Select Decode EDIFACT Message



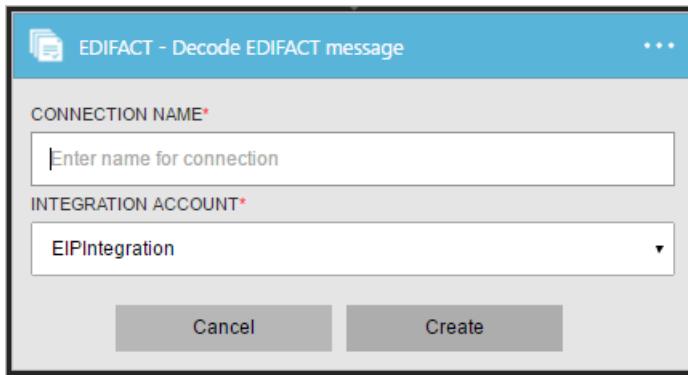
3. If you haven't previously created any connections to Integration Account, you are prompted for the connection details



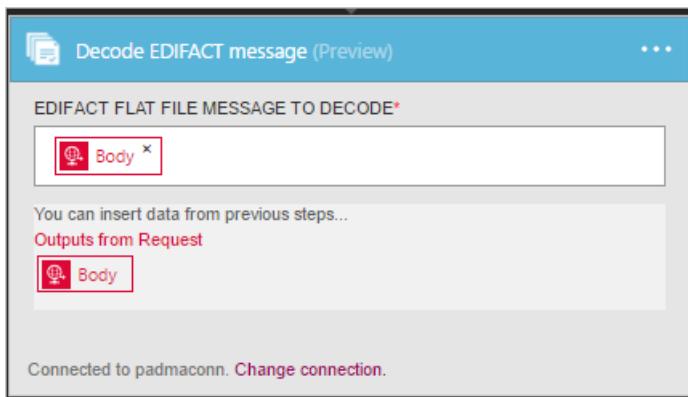
4. Enter the Integration Account details. Properties with an asterisk are required

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection
Integration Account *	Enter the Integration Account name. Be sure your Integration Account and Logic app are in the same Azure location

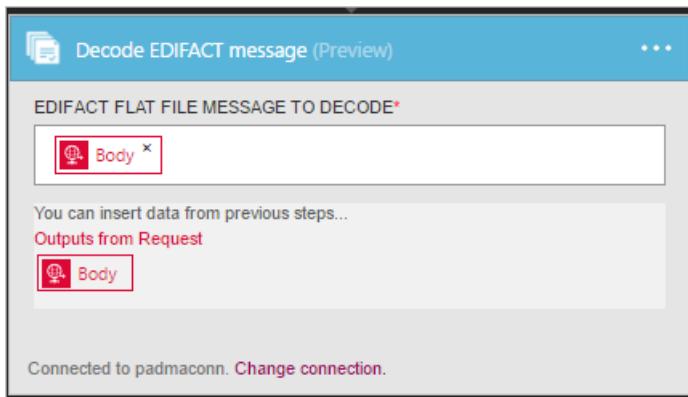
Once complete, your connection details look similar to the following



5. Select **Create**
6. Notice the connection has been created



7. Select EDIFACT flat file message to decode



EDIFACT Decode does following

- Resolve the agreement by matching the sender qualifier & identifier and receiver qualifier & identifier
- Splits multiple interchanges in a single message into separate.
- Validates the envelope against trading partner agreement
- Disassembles the interchange.

- Validates EDI and partner-specific properties includes
 - Validation of the structure of the interchange envelope.
 - Schema validation of the envelope against the control schema.
 - Schema validation of the transaction-set data elements against the message schema.
 - EDI validation performed on transaction-set data elements
- Verifies that the interchange, group, and transaction set control numbers are not duplicates (if configured)
 - Checks the interchange control number against previously received interchanges.
 - Checks the group control number against other group control numbers in the interchange.
 - Checks the transaction set control number against other transaction set control numbers in that group.
- Generates an XML document for each transaction set.
- Converts the entire interchange to XML
 - Split Interchange as transaction sets - suspend transaction sets on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, then EDIFACT Decode suspends only those transaction sets.
 - Split Interchange as transaction sets - suspend interchange on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, then EDIFACT Decode suspends the entire interchange.
 - Preserve Interchange - suspend transaction sets on error: Creates an XML document for the entire batched interchange. EDIFACT Decode suspends only those transaction sets that fail validation, while continuing to process all other transaction sets
 - Preserve Interchange - suspend interchange on error: Creates an XML document for the entire batched interchange. If one or more transaction sets in the interchange fail validation, then EDIFACT Decode suspends the entire interchange,
- Generates a Technical (control) and/or Functional acknowledgment (if configured).
 - A Technical Acknowledgment or the CONTRL ACK reports the results of a syntactical check of the complete received interchange.
 - A functional acknowledgment acknowledges accept or reject a received interchange or a group

Next steps

[Learn more about the Enterprise Integration Pack](#)

Enterprise integration with X12

1/20/2017 • 13 min to read • [Edit on GitHub](#)

NOTE

This page covers the X12 features of Logic Apps. For information on EDIFACT click [here](#).

Create an X12 agreement

Before you can exchange X12 messages, you need to create an X12 agreement and store it in your integration account. The following steps will walk you through the process of creating an X12 agreement.

Here's what you need before you get started

- An [integration account](#) defined in your Azure subscription
- At least two [partners](#) already defined in your integration account

NOTE

When creating an agreement, the content in the agreement file must match the agreement type.

After you've [created an integration account](#) and [added partners](#), you can create an X12 agreement by following these steps:

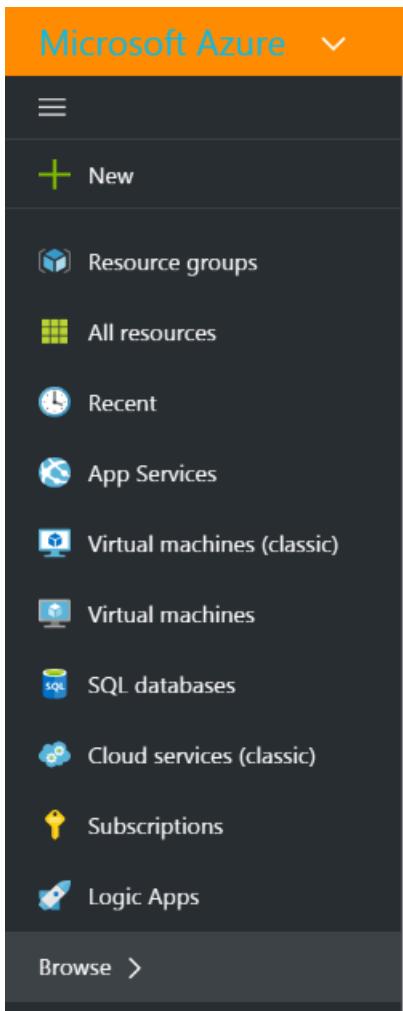
From the Azure portal home page

After you log into the [Azure portal](#):

1. Select **Browse** from the menu on the left.

TIP

If you don't see the **Browse** link, you may need to expand the menu first. Do this by selecting the **Show menu** link that's located at the top left of the collapsed menu.



1. Type *integration* into the filter search box then select **Integration Accounts** from the list of results.

A screenshot of the Azure search interface. A search bar at the top contains the text 'Integration'. Below the search bar, a card titled 'Integration Accounts' is highlighted with a gray background, while other cards like 'Resource groups' and 'Preview' are in the background. A tooltip above the search bar says 'Shift+Space to toggle favorites'.

2. In the **Integration Accounts** blade that opens up, select the integration account in which you will create the agreement. If you don't see any integration accounts lists, [create one first](#).

A screenshot of the 'Integration Accounts' blade. The title bar says 'Integration Accounts' with a 'PREVIEW' button. Below the title are buttons for '+ Add', 'Columns', and 'Refresh'. A status bar at the top left says 'Subscriptions: 1 of 3 selected'. The main area is a table with the following data:

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION	...
DeonheEIPNCentralIntegrationAccount	Integration Account	DeonheNCentralResGrop	North Central US	ICBCS9	...
DeonheIntegrationAccount	Integration Account	DeonheResGroup	Brazil South	ICBCS9	...
MyNewIntegrationAccount	Integration Account	MyNewRG	Brazil South	ICBCS9	...

3. Select the **Agreements** tile. If you don't see the agreements tile, add it first.

Subscriptions: 1 of 3 selected
Filter items...
ICBCS9
NAME
DeonheEIPNCentralIntegrationAccount
DeonheIntegrationAccount
MyNewIntegrationAccount

Essentials ^
Resource group DeonheEIPNCentralResGrp
Location North Central US
Subscription ID 1217a102-55fc-461a-9e2d-56a0acc2972
Name DeonheEIPNCentralIntegrationAccount
Subscription name ICBCS9
All settings →

Components
Schemas Maps Certificates
2 1 1
Partners Agreements
2 1
Add a section ⓘ

NAME	TYPE	HOST PARTNER	GUEST PARTNER
AgreementWithNYTra...	X12	MyCompany	NYTradingPartner

4. Select the **Add** button in the Agreements blade that opens.

Agreements EIPIntegration
+ Add Edit Edit as JSON Delete

NAME	TYPE	HOST PARTNER	GUEST PARTNER
There are no agreements to display.			

Add

* Name
Enter agreement name

* Agreement type
AS2

* Host Partner

* Host Identity

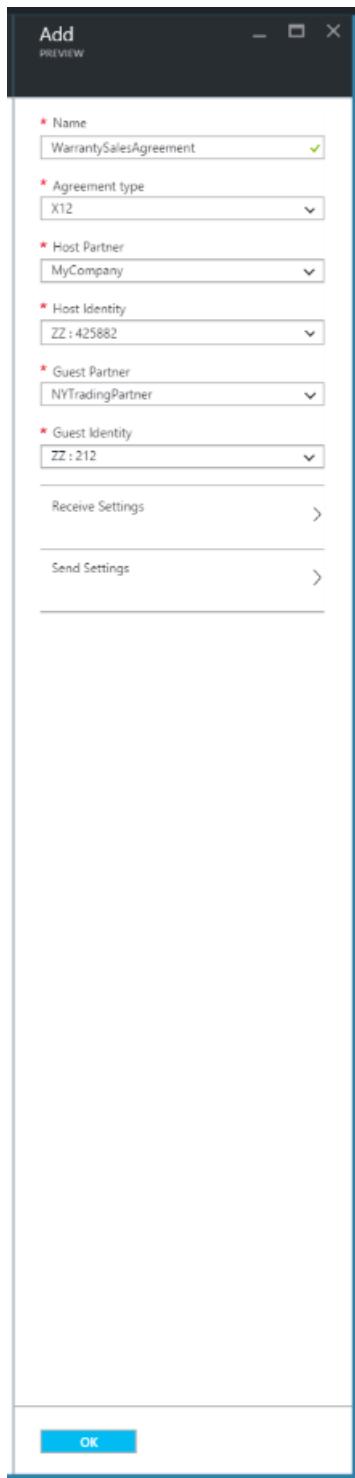
* Guest Partner

* Guest Identity

Receive Settings >

Send Settings >

5. Enter a **Name** for your agreement then select the **Agreement type, Host Partner, Host Identity, Guest Partner, Guest Identity**, in the Agreements blade that opens.



6. After you have set the receive settings properties, select the **OK** button

Let's continue:

7. Select **Receive Settings** to configure how messages received via this agreement are to be handled.
8. The Receive Settings control is divided into the following sections, including Identifiers, Acknowledgment, Schemas, Envelopes, Control Numbers, Validations and Internal Settings. Configure these properties based on your agreement with the partner you will be exchanging messages with. Here is a view of these controls, configure them based on how you want this agreement to identify and handle incoming messages:

Receive Settings

PREVIEW



Identifiers

ISA1 (Authorization Qualifier)	00 - No Authorization Information P... ▾	ISA3 (Security Qualifier)	00 - No Security Information Present ▾
ISA2	ISA2	ISA4	ISA4

Acknowledgement

<input type="checkbox"/> TA1 Expected	<input type="checkbox"/> FA Expected	
FA Version		
<input type="checkbox"/> 997 (Version 4010)	<input type="checkbox"/> 997 (Version 5010)	<input type="checkbox"/> 999 (Version 5010)

Include AK2 / IK2 Loop

Schemas

VERSION	TRANSACTION TYPE (ST01)	SENDER APPLICATION (GS02)	SCHEMA	...
No results				
00200 - ASC X12 Standards I...	100 - Insurance Plan Descrip...		OrderFile	...

Envelopes

ISA11 Usage	<input type="checkbox"/> Standard Identifier	<input type="checkbox"/> Repetition Separator
-------------	--	---

Control Numbers

Disallow Interchange control number duplicates

Check for duplicate ISA13 every (days)

30

Disallow Group control number duplicates

Disallow Transaction set control number duplicates

Validations

MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATIO...	ALLOW LEADING/TRAI...	TRIM LEADING/TRAII...	TRAILING SEPARATOR
Default	true	false	false	false	NotAllowed	...
100 - Insurance...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Mandatory	...

Internal Settings

Convert implied decimal format Nn to base 10 numeric value

Create empty XML tags if trailing separators are allowed

Inbound batch processing

- Split Interchange as transaction sets - suspend transaction sets on error
- Split Interchange as transaction sets - suspend interchange on error
- Preserve Interchange - suspend transaction sets on error
- Preserve Interchange - suspend interchange on error

OK

- Select the **OK** button to save your settings.

Identifiers

PROPERTY	DESCRIPTION
ISA1 (Authorization Qualifier)	Select the Authorization qualifier value from the drop-down list.
ISA2	Optional. Enter Authorization information value. If the value you entered for ISA1 is other than 00, enter a minimum of one alphanumeric character and a maximum of 10.
ISA3 (Security Qualifier)	Select the Security qualifier value from the drop-down list.
ISA4	Optional. Enter the Security information value. If the value you entered for ISA3 is other than 00, enter a minimum of one alphanumeric character and a maximum of 10.

Acknowledgments

PROPERTY	DESCRIPTION
TA1 expected	Select this checkbox to return a technical (TA1) acknowledgment to the interchange sender. These acknowledgments are sent to the interchange sender based on the Send Settings for the agreement.
FA expected	Select this checkbox to return a functional (FA) acknowledgment to the interchange sender. Then select whether you want the 997 or 999 acknowledgements, based on the schema versions you are working with. These acknowledgments are sent to the interchange sender based on the Send Settings for the agreement.
Include AK2/IK2 Loop	Select this checkbox to enable generation of AK2 loops in functional acknowledgments for accepted transaction sets. Note: This checkbox is enabled only if you selected the FA expected checkbox.

Schemas

Choose a schema for each transaction type (ST1) and Sender Application (GS2). The receive pipeline disassembles the incoming message by matching the values for ST1 and GS2 in the incoming message with the values you set here, and the schema of the incoming message with the schema you set here.

PROPERTY	DESCRIPTION
Version	Select the X12 version
Transaction Type (ST01)	Select the transaction type
Sender Application (GS02)	Select the sender application
Schema	Select the schema file you want to use. Schema files are located in your integration account.

Envelopes

PROPERTY	DESCRIPTION
ISA11 Usage	<p>Use this field to specify the separator in a transaction set: Select the Standard identifier to use the decimal notation of “.” instead of the decimal notation of the incoming document in the EDI receive pipeline.</p> <p>Select Repetition separator to specify the separator for repeated occurrences of a simple data element or a repeated data structure. For example, (^) is usually used as repetition separator. For HIPAA schemas, you can only use (^).</p>

Control Numbers

PROPERTY	DESCRIPTION
Disallow Interchange Control Number duplicates	<p>Check this option to block duplicate interchanges. If selected, the BizTalk Services Portal checks that the interchange control number (ISA13) for the received interchange does not match the interchange control number. If a match is detected, the receive pipeline does not process the interchange.</p> <p>If you opted to disallow duplicate interchange control numbers, then you can specify the number of days at which the check is performed by giving the appropriate value for Check for duplicate ISA13 every x days.</p>
Disallow Group control number duplicates	Check this option to block interchanges with duplicate group control numbers.
Disallow Transaction set control number duplicates	Check this option to block interchanges with duplicate transaction set control numbers.

Validations

PROPERTY	DESCRIPTION
Message Type	EDI Message type, like 850-Purchase Order or 999-Implementation Acknowledgement.
EDI Validation	Performs EDI validation on data types as defined by the EDI properties of the schema, length restrictions, empty data elements, and trailing separators.
Extended Validation	If the data type is not EDI, validation is on the data element requirement and allowed repetition, enumerations, and data element length validation (min/max).
Allow Leading/Trailing Zeros	Any additional space and zero characters that are leading or trailing are retained. They are not removed.
Trailing Separator Policy	Generates trailing separators on the interchange received. Options include NotAllowed, Optional, and Mandatory.

Internal Settings

PROPERTY	DESCRIPTION
Convert implied decimal format Nn to base 10 numeric value	Converts an EDI number that is specified with the format Nn into a base-10 numeric value in the intermediate XML in the BizTalk Services Portal.
Create empty XML tags if trailing separators are allowed	Select this check box to have the interchange sender include empty XML tags for trailing separators.
Inbound batching processing	<p>Split Interchange as transaction sets - suspend transaction sets on error: Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope to the transaction set. With this option, if one or more transaction sets in the interchange fail validation, then BizTalk Services suspends only those transaction sets.</p> <p>Split Interchange as transaction sets - suspend interchange on error: Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope. With this option, if one or more transaction sets in the interchange fail validation, then BizTalk Services suspends the entire interchange.</p> <p>Preserve Interchange - suspend transaction sets on error: Leaves the interchange intact, creating an XML document for the entire batched interchange. With this option, if one or more transaction sets in the interchange fail validation, then BizTalk Services suspends only those transaction sets, while continuing to process all other transaction sets.</p> <p>Preserve Interchange - suspend interchange on error: Leaves the interchange intact, creating an XML document for the entire batched interchange. With this option, if one or more transaction sets in the interchange fail validation, then BizTalk Services suspends the entire interchange.</p>

Your agreement is ready to handle incoming messages that conform to the schema you selected.

To configure the settings that handle messages you send to partners:

1. Select **Send Settings** to configure how messages sent via this agreement are to be handled.

The Send Settings control is divided into the following sections, including Identifiers, Acknowledgment, Schemas, Envelopes, Control Numbers, Character Sets and Separators and Validation.

Here is a view of these controls. Make the selections based on how you want to handle messages you send to partners via this agreement:

Send Settings

PREVIEW

Identifiers

ISA1 (Authorization Qualifier)	00 - No Authorization Information P... ▾	ISA3 (Security Qualifier)	00 - No Security Information Present ▾
ISA2	ISA2	ISA4	ISA4

Acknowledgement

TA1 Expected FA Expected

FA Version

997 999

Schemas

VERSION	TRANSACTION TYPE (ST01)	SCHEMA	...
No results			
00200 - ASC X12 Standards Issued by A... ▾	100 - Insurance Plan Description ▾	OrderFile ▾	...

Envelopes

ISA11 Usage

Repetition Separator

U

Control Version Number (ISA12): 00401 - Standards Approved for... ▾ Usage Indicator (ISA15): Test ▾

SCHEMA	GS1	GS2	GS3	GS4	GS5	GS7	GS8	...
Default	BTS-SENDER	RECEIVE-APP	CCYYMMDD	HHMM	Transportation...	00401		...
OrderFile ▾	AA - Acc.. ▾			CCYYMM... ▾	HHMM ▾	Transpor... ▾		...

Control Numbers

Interchange Control Number (ISA13) : <input type="text" value="1"/>	to <input type="text" value="999999999"/>
Group Control Number (GS06) : <input type="text" value="1"/>	to <input type="text" value="999999999"/>
Transaction Set Control Number (ST02) : <input type="text" value="1"/>	to <input type="text" value="999999999"/>
Prefix : <input type="text"/>	
Suffix : <input type="text"/>	

Character Sets and Separators

* Character Set to be used :

UTF8

SCHEMA	INPUT TYPE	COMPONENT SEPARATOR	DATA ELEMENT SEPARATOR	REPLACEMENT CHARACTER	SEGMENT TERMINATOR	SUFFIX	
Default	:	*	\$	~	None	...	
OrderFile	Char					None	...

Validation

MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAILING SPACES	TRIM LEADING/TRAILING SPACES	TRAILING SEPARATOR	
Default	true	false	false	false	NotAllowed	...
100 - Insurance...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Not Allowed	...

OK

- Select the **OK** button to save your settings.

Identifiers

PROPERTY	DESCRIPTION
Authorization qualifier (ISA1)	Select the Authorization qualifier value from the drop-down list.
ISA2	Enter Authorization information value. If this value is other than 00, then enter a minimum of one alphanumeric character and a maximum of 10.
Security qualifier (ISA3)	Select the Security qualifier value from the drop-down list.
ISA4	Enter the Security information value. If this value is other than 00, for the Value (ISA4) text box, then enter a minimum of one alphanumeric value and a maximum of 10.

Acknowledgment

PROPERTY	DESCRIPTION
TA1 expected	Select this checkbox to return a technical (TA1) acknowledgment to the interchange sender. This setting specifies that the host partner who is sending the message requests an acknowledgement from the guest partner in the agreement. These acknowledgments are expected by the host partner based on the Receive Settings of the agreement.
FA expected	Select this checkbox to return a functional (FA) acknowledgment to the interchange sender, and then select whether you want the 997 or 999 acknowledgements, based on the schema versions you are working with. These acknowledgments are expected by the host partner based on the Receive Settings of the agreement.
FA Version	Select the FA version

Schemas

PROPERTY	DESCRIPTION
Version	Select the X12 version
Transaction Type (ST01)	Select the transaction type
SCHEMA	Select the schema to use. Schemas are located in your integration account. To access your schemas, first link your integration account to your Logic app.

Envelopes

PROPERTY	DESCRIPTION
ISA11 Usage	<p>Use this field to specify the separator in a transaction set: Select the Standard identifier to use the decimal notation of “.” instead of the decimal notation of the incoming document in the EDI receive pipeline.</p> <p>Select Repetition separator to specify the separator for repeated occurrences of a simple data element or a repeated data structure. For example, (^) is usually used as repetition separator. For HIPAA schemas, you can only use (^).</p>
Repetition separator	Enter the repetition separator
Control Version Number (ISA12)	Select the version of the X12 standard that is used by the BizTalk Services Portal for generating an outgoing interchange.
Usage Indicator (ISA15)	Enter whether the context of an interchange is information (I), production data (P), or test data (T). The EDI receive pipeline promotes this property to the context.

PROPERTY	DESCRIPTION
Schema	<p>You can enter how the BizTalk Services Portal generates the GS and ST segments for an X12-encoded interchange that it sends to the Send Pipeline.</p> <p>You can associate values of the GS1, GS2, GS3, GS4, GS5, GS7, and GS8 data elements with values of the Transaction Type, and Version/Release data elements. When the BizTalk Services Portal determines that an XML message has the values set for the Transaction Type, and Version/Release elements in a row of the grid, then it populates the GS1, GS2, GS3, GS4, GS5, GS7, and GS8 data elements in the envelope of the outgoing interchange with the values from the same row of the grid. The values of the Transaction Type, and Version/Release elements must be unique.</p> <p>Optional. For GS1, select a value for the functional code from the drop-down list.</p> <p>Required. For GS2, enter an alphanumeric value for the application sender with a minimum of two characters and a maximum of 15 characters.</p> <p>Required. For GS3, enter an alphanumeric value for the application receiver with a minimum of two characters and a maximum of 15 characters.</p> <p>Optional. For GS4, select CCYYMMDD or YYMMDD.</p> <p>Optional. For GS5, select HHMM, HHMMSS, or HHMMSSdd.</p> <p>Optional. For GS7, select a value for the responsible agency from the drop-down list.</p> <p>Optional. For GS8, enter an alphanumeric value for the document identified with a minimum of one character and a maximum of 12 characters.</p> <p>Note:These are the values that the BizTalk Services Portal enters in the GS fields of the interchange it is building if the Transaction Type, and Version/Release elements in the same row are a match for those associated with the interchange.</p>

Control Numbers

PROPERTY	DESCRIPTION
Interchange Control Number (ISA13)	Required. Enter a range of values for the interchange control number used by the BizTalk Services Portal in generating an outgoing interchange. Enter a numeric value with a minimum of 1 and a maximum of 999999999.
Group Control Number (GS06)	Required. Enter the range of numbers that the BizTalk Services Portal should use for the group control number. Enter a numeric value with a minimum of one character and a maximum of nine characters.
Transaction Set Control Number (ST02)	For Transaction Set Control number (ST02), enter a range of numeric values for the required middle fields, and alphanumeric values for the optional prefix and suffix. The maximum length of all four fields is nine characters.

PROPERTY	DESCRIPTION
Prefix	To designate the range of transaction set control numbers used in an acknowledgment, enter values in the ACK Control number (ST02) fields. Enter a numeric value for the middle two fields, and an alphanumeric value (if desired) for the prefix and suffix fields. The middle fields are required and contain the minimum and maximum values for the control number; the prefix and suffix are optional. The maximum length for all three fields is nine characters.
Suffix	To designate the range of transaction set control numbers used in an acknowledgment, enter values in the ACK Control number (ST02) fields. Enter a numeric value for the middle two fields, and an alphanumeric value (if desired) for the prefix and suffix fields. The middle fields are required and contain the minimum and maximum values for the control number; the prefix and suffix are optional. The maximum length for all three fields is nine characters.

Character Sets and Separators

Other than the character set, you can enter a different set of delimiters to be used for each message type. If a character set is not specified for a given message schema, then the default character set is used.

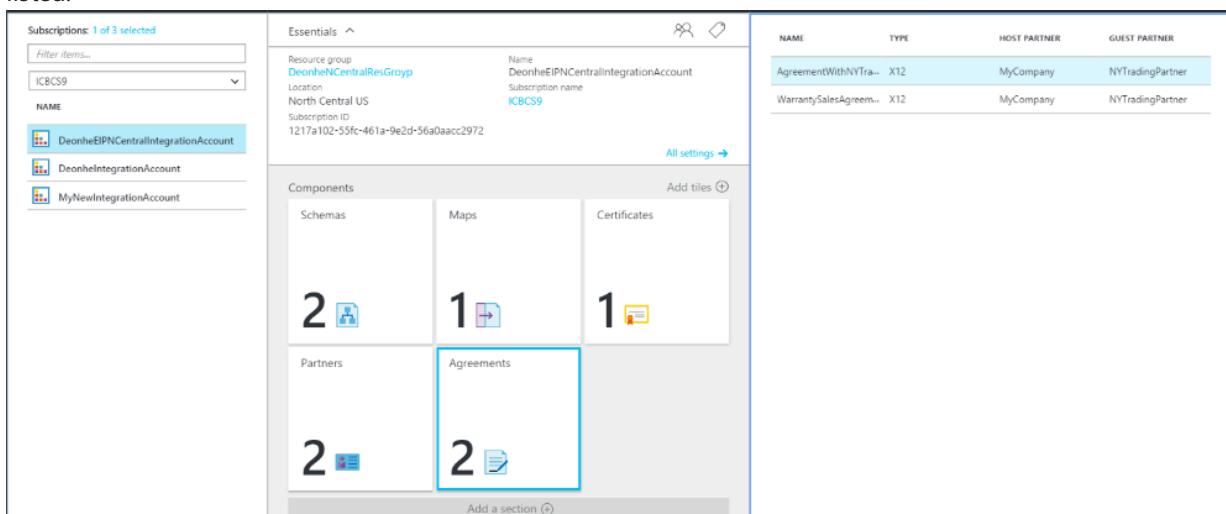
PROPERTY	DESCRIPTION
Character Set to be used	<p>Select the X12 character set to validate the properties that you enter for the agreement.</p> <p>Note: The BizTalk Services Portal only uses this setting to validate the values entered for the related agreement properties. The receive pipeline or send pipeline ignores this character-set property when performing run-time processing.</p>
Schema	<p>Select the (+) symbol and select a schema from the drop-down list. For the selected schema, select the separators set to be used:</p> <p>Component element separator – Enter a single character to separate composite data elements.</p> <p>Data Element Separator – Enter a single character to separate simple data elements within composite data elements.</p> <p>Replacement Character – Select this check box if the payload data contains characters that are also used as data, segment, or component separators. You can then enter a replacement character. When generating the outbound X12 message, all instances of separator characters in the payload data are replaced with the specified character.</p> <p>Segment Terminator – Enter a single character to indicate the end of an EDI segment.</p> <p>Suffix – Select the character that is used with the segment identifier. If you designate a suffix, then the segment terminator data element can be empty. If the segment terminator is left empty, then you must designate a suffix.</p>

Validation

PROPERTY	DESCRIPTION
Message Type	Selecting this option enables validation on the interchange receiver. This validation performs EDI validation on transaction-set data elements, validating data types, length restrictions, and empty data elements and trailing separators.
EDI Validation	
Extended Validation	Selecting this option enables extended validation of interchanges received from the interchange sender. This includes validation of field length, optionality, and repeat count in addition to XSD data type validation. You can enable extension validation without enabling EDI validation, or vice versa.
Allow leading/ trailing zeroes	Selecting this option specifies that an EDI interchange received from the party does not fail validation if a data element in an EDI interchange does not conform to its length requirement because of or trailing spaces, but does conform to its length requirement when they are removed.
Trailing separator	<p>Selecting this option specifies an EDI interchange received from the party does not fail validation if a data element in an EDI interchange does not conform to its length requirement because of leading (or trailing) zeroes or trailing spaces, but does conform to its length requirement when they are removed.</p> <p>Select Not Allowed if you do not want to allow trailing delimiters and separators in an interchange received from the interchange sender. If the interchange contains trailing delimiters and separators, it is declared invalid.</p> <p>Select Optional to accept interchanges with or without trailing delimiters and separators.</p> <p>Select Mandatory if the received interchange must contain trailing delimiters and separators.</p>

After you select **OK** on the open blades:

1. Select the **Agreements** tile on the Integration Account blade and you will see the newly added agreement listed.



The screenshot shows the 'Integration Accounts' blade in Dynamics 365 Business Central. On the left, there's a sidebar with 'Subscriptions: 1 of 3 selected' and a dropdown set to 'ICBCS9'. Below that is a 'NAME' section with three items: 'DeonheEPNCentralIntegrationAccount' (selected), 'DeonheIntegrationAccount', and 'MyNewIntegrationAccount'. The main area has a title 'Essentials' with a gear icon. It displays resource group information: 'Resource group DeonheEPNCentralResGroyp', 'Location North Central US', 'Subscription ID 1217a102-55fc-461a-9e2d-56a0acc2072', and a 'Name DeonheEPNCentralIntegrationAccount' with a 'Subscription name ICBCS9'. There's also a 'All settings' link. Below this is a 'Components' section with tiles for 'Schemas' (2), 'Maps' (1), 'Certificates' (1), 'Partners' (2), and the 'Agreements' tile, which is highlighted with a blue border. To the right, there's a table titled 'NAME', 'TYPE', 'HOST PARTNER', and 'GUEST PARTNER' with two rows: 'AgreementWithNYTra...' (X12, MyCompany, NYTradingPartner) and 'WarrantySalesAgreem...' (X12, MyCompany, NYTradingPartner). At the bottom, there's a 'Add a section' button.

Learn more

- [Learn more about the Enterprise Integration Pack](#)

Get started with Encode X12 Message

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Validates EDI and partner-specific properties, converts XML-encoded messages into EDI transaction sets in the interchange and requests a Technical and/or Functional acknowledgment

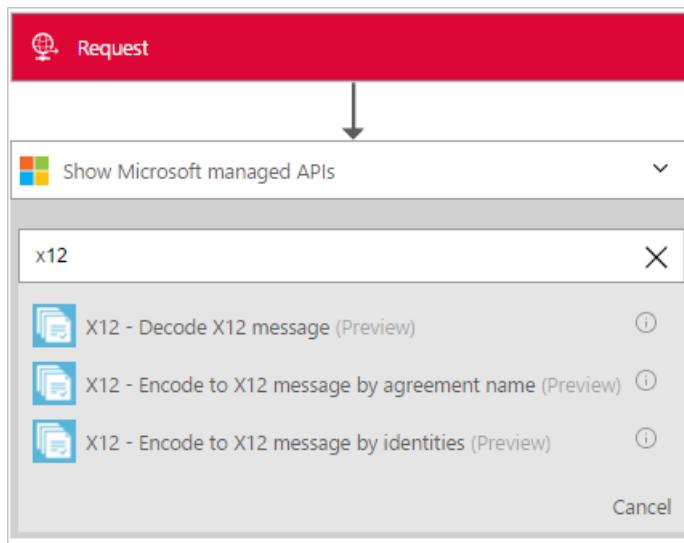
Create the connection

Prerequisites

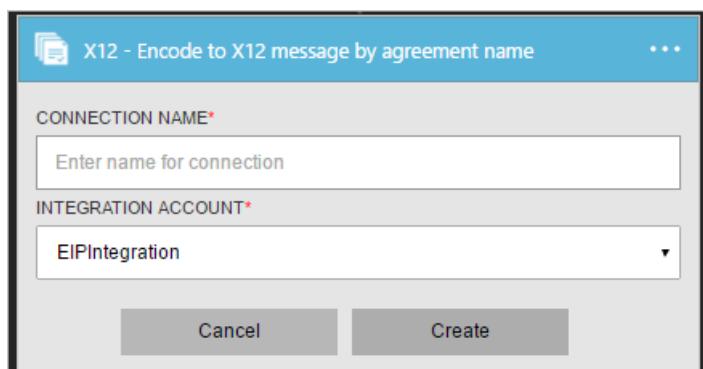
- An Azure account; you can create a [free account](#)
- An Integration Account is required to use Encode x12 message connector. See details on how to create an [Integration Account](#), [partners](#), and [X12 agreement](#)

Connect to Encode X12 Message using the following steps:

1. [Create a Logic App](#) provides an example
2. This connector does not have any triggers. Use other triggers to start the Logic App, such as a Request trigger. In the Logic App designer, add a trigger and add an action. Select Show Microsoft managed APIs in the drop-down list and then enter “x12” in the search box. Select either X12 - Encode X12 Message by agreement name or X12 - Encode to X 12 message by identities.



3. If you haven't previously created any connections to Integration Account, you are prompted for the connection details



4. Enter the Integration Account details. Properties with an asterisk are required

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection
Integration Account *	Enter the Integration Account name. Be sure your Integration Account and Logic app are in the same Azure location

Once complete, your connection details look similar to the following

X12 - Encode to X12 message by agreement name

CONNECTION NAME*

padmaconn

INTEGRATION ACCOUNT*

ElPIIntegration

Cancel Create

5. Select **Create**
6. Notice the connection has been created.

Encode to X12 message by agreement name (Preview)

NAME OF X12 AGREEMENT*

X12Agreement

XML MESSAGE TO ENCODE*

XML message to encode

Connected to padmaconn. Change connection.

X12 - Encode X12 Message by agreement name

1. Select X12 agreement from the drop-down and xml message to encode.

Encode to X12 message by agreement name (Preview)

NAME OF X12 AGREEMENT*

X12Agreement

XML MESSAGE TO ENCODE*

Body

You can insert data from previous steps...

Outputs from Request

Body

Connected to padmaconn. Change connection.

X12 - Encode X12 Message by identities

1. Provide sender identifier, sender qualifier, receiver identifier, and receiver qualifier as configured in the X12 agreement. Select xml message to encode

Encode to X12 message by identities (Preview)

SENDER IDENTIFIER*

Sender identifier

SENDER QUALIFIER*

Sender qualifier

RECEIVER IDENTIFIER*

Receiver identifier

RECEIVER QUALIFIER*

Receiver qualifier

XML MESSAGE TO ENCODE*

XML message to encode

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

X12 Encode does following:

- Agreement resolution by matching sender and receiver context properties.
- Serializes the EDI interchange, converting XML-encoded messages into EDI transaction sets in the interchange.
- Applies transaction set header and trailer segments
- Generates an interchange control number, a group control number, and a transaction set control number for each outgoing interchange
- Replaces separators in the payload data
- Validates EDI and partner-specific properties
 - Schema validation of the transaction-set data elements against the message Schema
 - EDI validation performed on transaction-set data elements.
 - Extended validation performed on transaction-set data elements
- Requests a Technical and/or Functional acknowledgment (if configured).
 - A Technical Acknowledgment generates as a result of header validation. The technical acknowledgment reports the status of the processing of an interchange header and trailer by the address receiver
 - A Functional Acknowledgment generates as a result of body validation. The functional acknowledgment reports each error encountered while processing the received document

Next steps

[Learn more about the Enterprise Integration Pack](#)

Get started with Decode X12 Message

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Validates EDI and partner-specific properties, generates XML document for each transaction set and generates acknowledgment for processed transaction.

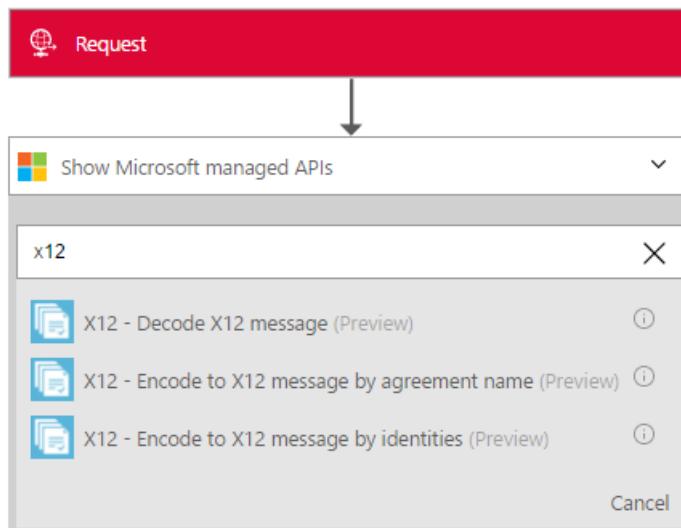
Create the connection

Prerequisites

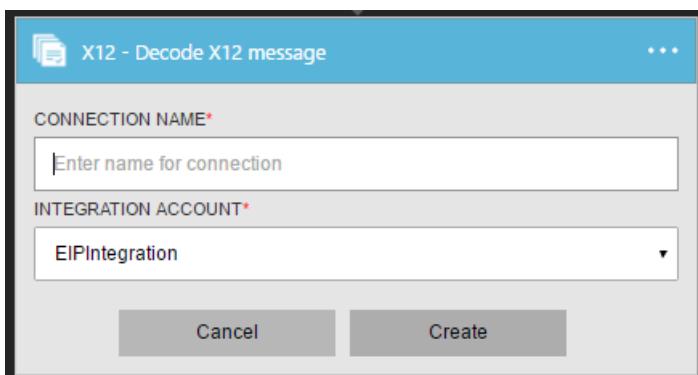
- An Azure account; you can create a [free account](#)
- An Integration Account is required to use Decode X12 message connector. See details on how to create an [Integration Account](#), [partners](#) and [X12 agreement](#)

Connect to Decode X12 Message using the following steps:

1. [Create a Logic App](#) provides an example
2. This connector does not have any triggers. Use other triggers to start the Logic App, such as a Request trigger. In the Logic App designer, add a trigger and add an action. Select Show Microsoft managed APIs in the drop-down list and then enter “x12” in the search box. Select X12 – Decode X12 Message



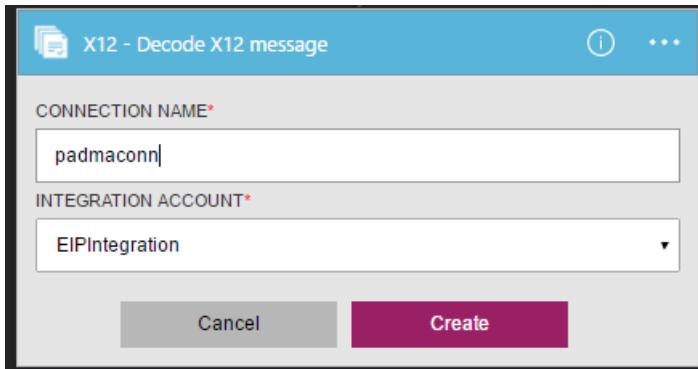
3. If you haven't previously created any connections to Integration Account, you are prompted for the connection details



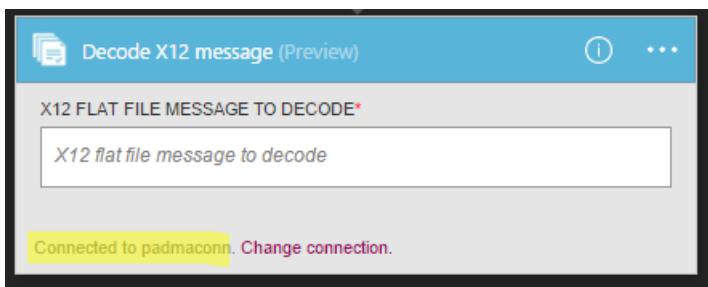
4. Enter the Integration Account details. Properties with an asterisk are required

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection
Integration Account *	Enter the Integration Account name. Be sure your Integration Account and Logic app are in the same Azure location

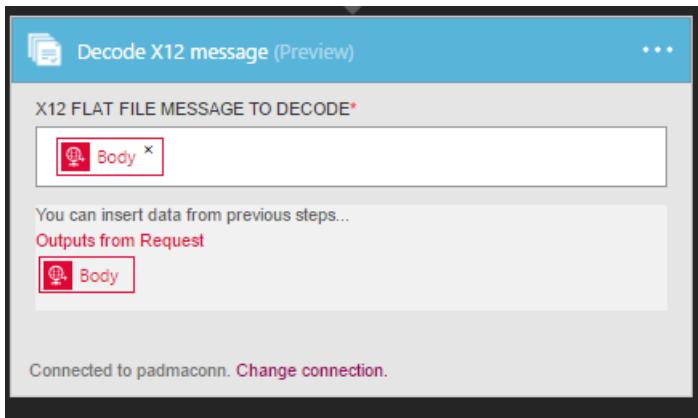
Once complete, your connection details look similar to the following



5. Select **Create**
6. Notice the connection has been created.



7. Select X12 flat file message to decode



X12 Decode does following

- Validates the envelope against trading partner agreement
- Generates an XML document for each transaction set.
- Validates EDI and partner-specific properties
 - EDI structural validation, and extended schema validation
 - Validation of the structure of the interchange envelope.
 - Schema validation of the envelope against the control schema.

- Schema validation of the transaction-set data elements against the message schema.
 - EDI validation performed on transaction-set data elements
- Verifies that the interchange, group, and transaction set control numbers are not duplicates
 - Checks the interchange control number against previously received interchanges.
 - Checks the group control number against other group control numbers in the interchange.
 - Checks the transaction set control number against other transaction set control numbers in that group.
- Converts the entire interchange to XML
 - Split Interchange as transaction sets - suspend transaction sets on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, X12 Decode suspends only those transaction sets.
 - Split Interchange as transaction sets - suspend interchange on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, X12 Decode suspends the entire interchange.
 - Preserve Interchange - suspend transaction sets on error: Creates an XML document for the entire batched interchange. X12 Decode suspends only those transaction sets that fail validation, while continuing to process all other transaction sets
 - Preserve Interchange - suspend interchange on error: Creates an XML document for the entire batched interchange. If one or more transaction sets in the interchange fail validation, X12 Decode suspends the entire interchange,
- Generates a Technical and/or Functional acknowledgment (if configured).
 - A Technical Acknowledgment generates as a result of header validation. The technical acknowledgment reports the status of the processing of an interchange header and trailer by the address receiver.
 - A Functional Acknowledgment generates as a result of body validation. The functional acknowledgment reports each error encountered while processing the received document

Next steps

[Learn more about the Enterprise Integration Pack](#)

Connect to the on-premises data gateway for Logic Apps

1/20/2017 • 1 min to read • [Edit on GitHub](#)

Supported logic apps connectors allow you to configure your connection to access on-premises data via the on-premises data gateway. The following steps will walk you through how to install and configure the on-premises data gateway to work with a logic app.

Prerequisites

- Must be using a work or school email address in Azure to associate the on-premises data gateway with your account (Azure Active Directory based account)
- If you are using a Microsoft Account (e.g. @outlook.com, @live.com) you can use your Azure account to create a work or school email address by [following the steps here](#)
- Must have the on-premises data gateway [installed on a local machine](#).
- Gateway must not have been claimed by another Azure on-premises data gateway ([claim happens with creation of step 2 below](#)) - an installation can only be associated to one gateway resource.

Installing and configuring the connection

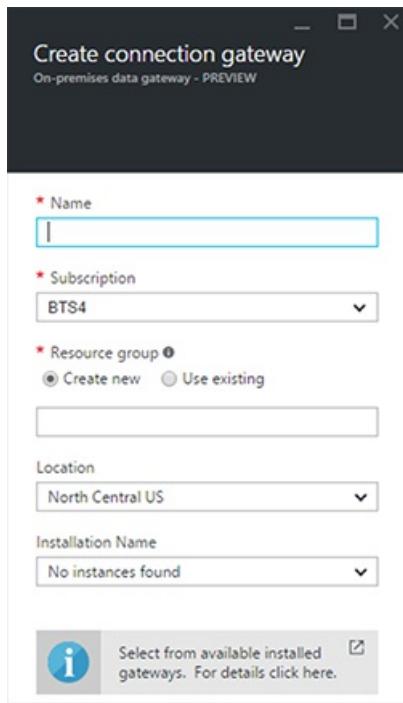
1. Install the on-premises data gateway

Information on installing the on-premises data gateway can be found [in this article](#). The gateway must be installed on an on-premises machine before you can continue with the rest of the steps.

2. Create an Azure on-premises data gateway resource

Once installed, you must associate your Azure subscription with the on-premises data gateway.

1. Login to Azure using the same work or school email address that was used during installation of the gateway
2. Click **New** resource button
3. Search and select the **On-premises data gateway**
4. Complete the information to associate the gateway with your account - including selecting the appropriate **Installation Name**

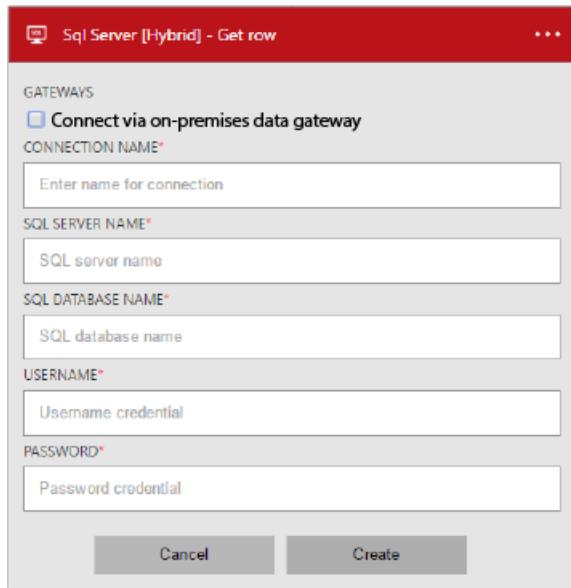


5. Click the **Create** button to create the resource

3. Create a logic app connection in the designer

Now that your Azure subscription is associated with an instance of the on-premises data gateway, you can create a connection to it from within a logic app.

1. Open a logic app and choose a connector that supports on-premises connectivity (as of this writing, SQL Server)
2. Select the checkbox for **Connect via on-premises data gateway**



3. Select the **Gateway** to connect to and complete any other connection information required
4. Click **Create** to create the connection

The connection should now be successfully configured for use in your logic app.

Next Steps

- [Common examples and scenarios for logic apps](#)
- [Enterprise integration features](#)

Install the on-premises data gateway for Logic Apps

1/20/2017 • 8 min to read • [Edit on GitHub](#)

Installation and Configuration

Prerequisites

Minimum:

- .NET 4.5 Framework
- 64-bit version of Windows 7 or Windows Server 2008 R2 (or later)

Recommended:

- 8 Core CPU
- 8 GB Memory
- 64-bit version of Windows 2012 R2 (or later)

Related considerations:

- You can't install a gateway on a domain controller.
- You shouldn't install a gateway on a computer, such as a laptop, that may be turned off, asleep, or not connected to the Internet because the gateway can't run under any of those circumstances. In addition, gateway performance might suffer over a wireless network.

Install a gateway

You can get the [installer for the on-premises data gateway here](#).

Specify **On-premises data gateway** as the mode, sign in with your work or school account, and then either configure a new gateway or migrate, restore, or take over an existing gateway.

- To configure a gateway, type a **name** for it and a **recovery key**, and then click or tap **Configure**.

Specify a recovery key that contains at least eight characters, and keep it in a safe place. You'll need this key if you want to migrate, restore, or take over its gateway.

- To migrate, restore, or take over an existing gateway, provide the recovery key that was specified when the gateway was created.

Restart the gateway

The gateway runs as a Windows service and, as with any other Windows service, you can start and stop it in multiple ways. For example, you can open a command prompt with elevated permissions on the machine where the gateway is running, and then run either of these commands:

- To stop the service, run this command:

```
net stop PBIEgwService
```

- To start the service, run this command:

```
net start PBIEgwService
```

Configure a firewall or proxy

For information about how to provide proxy information for your gateway, see [Configure proxy settings](#).

You can verify whether your firewall, or proxy, may be blocking connections by running the following command from a PowerShell prompt. This will test connectivity to the Azure Service Bus. This only tests network connectivity and doesn't have anything to do with the cloud server service or the gateway. It helps to determine whether your machine can actually get out to the internet.

```
Test-NetConnection -ComputerName watchdog.servicebus.windows.net -Port 9350
```

The results should look similar to this example. If **TcpTestSucceeded** is not true, you may be blocked by a firewall.

```
ComputerName      : watchdog.servicebus.windows.net
RemoteAddress    : 70.37.104.240
RemotePort       : 5672
InterfaceAlias   : vEthernet (Broadcom NetXtreme Gigabit Ethernet - Virtual Switch)
SourceAddress    : 10.120.60.105
PingSucceeded    : False
PingReplyDetails (RTT) : 0 ms
TcpTestSucceeded : True
```

If you want to be exhaustive, substitute the **ComputerName** and **Port** values with those listed under [Configure ports](#) later in this topic.

The firewall may also be blocking the connections that the Azure Service Bus makes to the Azure data centers. If that is the case, you'll want to whitelist (unblock) all of the IP addresses for your region for those data centers. You can get a list of [Azure IP addresses here](#).

Configure ports

The gateway creates an outbound connection to Azure Service Bus. It communicates on outbound ports: TCP 443 (default), 5671, 5672, 9350 thru 9354. The gateway doesn't require inbound ports.

Learn more about [hybrid solutions](#).

DOMAIN NAMES	OUTBOUND PORTS	DESCRIPTION
*.analysis.windows.net	443	HTTPS
*.login.windows.net	443	HTTPS
*.servicebus.windows.net	5671-5672	Advanced Message Queuing Protocol (AMQP)
*.servicebus.windows.net	443, 9350-9354	Listeners on Service Bus Relay over TCP (requires 443 for Access Control token acquisition)
*.frontend.clouddatahub.net	443	HTTPS
*.core.windows.net	443	HTTPS
login.microsoftonline.com	443	HTTPS
*.msftncsi.com	443	Used to test internet connectivity if the gateway is unreachable by the Power BI service.

If you need to white list IP addresses instead of the domains, you can download and use the [Microsoft Azure Datacenter IP ranges list](#). In some cases, the Azure Service Bus connections will be made with IP Address instead of the fully qualified domain names.

Sign-in account

Users will sign in with either a work or school account. This is your organization account. If you signed up for an Office 365 offering and didn't supply your actual work email, it may look like jeff@contoso.onmicrosoft.com. Your account, within a cloud service, is stored within a tenant in Azure Active Directory (AAD). In most cases, your AAD account's UPN will match the email address.

Windows Service account

The on-premises data gateway is configured to use NT SERVICE\PBIEgwService for the Windows service logon credential. By default, it has the right of Log on as a service. This is in the context of the machine on which you're installing the gateway.

This isn't the account used to connect to on-premises data sources or the work or school account with which you sign in to cloud services.

Frequently asked questions

General

Question: What data sources does the gateway support?

Answer: As of this writing, SQL Server.

Question: Do I need a gateway for data sources in the cloud, such as SQL Azure?

Answer: No. A gateway connects to on-premises data sources only.

Question: What is the actual Windows service called?

Answer: In Services, the gateway is called Power BI Enterprise Gateway Service.

Question: Are there any inbound connections to the gateway from the cloud?

Answer: No. The gateway uses outbound connections to Azure Service Bus.

Question: What if I block outbound connections? What do I need to open?

Answer: See the ports and hosts that the gateway uses.

Question: Does the gateway have to be installed on the same machine as the data source?

Answer: No. The gateway will connect to the data source using the connection information that was provided.

Think of the gateway as a client application in this sense. It will just need to be able to connect to the server name that was provided.

Question: What is the latency for running queries to a data source from the gateway? What is the best architecture?

Answer: To reduce network latency, install the gateway as close to the data source as possible. If you can install the gateway on the actual data source, it will minimize the latency introduced. Consider the data centers as well. For example, if your service is using the West US data center and you have SQL Server hosted in an Azure VM, you'll want to have the Azure VM in West US as well. This will minimize latency and avoid egress charges on the Azure VM.

Question: Are there any requirements for network bandwidth?

Answer: It is recommended to have good throughput for your network connection. Every environment is different, and the amount of data being sent will affect the results. Using ExpressRoute could help to guarantee a level of throughput between on-premises and the Azure data centers.

You can use the third-party tool Azure Speed Test app to help gauge what your throughput is.

Question: Can the gateway Windows service run with an Azure Active Directory account?

Answer: No. The Windows service must have a valid Windows account. By default, it will run with the Service SID, NT SERVICE\PBIEgwService.

Question: How are results sent back to the cloud?

Answer: This is done by way of the Azure Service Bus. For more information, see how it works.

Question: Where are my credentials stored?

Answer: The credentials that you enter for a data source are stored encrypted in the gateway cloud service. The credentials are decrypted at the gateway on-premises.

High availability/disaster recovery

Question: Are there any plans for enabling high availability scenarios with the gateway?

Answer: This is on the roadmap, but we don't have a timeline yet.

Question: What options are available for disaster recovery?

Answer: You can use the recovery key to restore or move a gateway. When you install the gateway, specify the recovery key.

Question: What is the benefit of the recovery key?

Answer: It provides a way to migrate or recover your gateway settings after a disaster.

Troubleshooting

Question: Where are the gateway logs?

Answer: See Tools later in this topic.

Question: How can I see what queries are being sent to the on-premises data source?

Answer: You can enable query tracing, which will include the queries being sent. Remember to change it back to the original value when done troubleshooting. Leaving query tracing enabled will cause the logs to be larger.

You can also look at tools that your data source has for tracing queries. For example, you can use Extended Events or SQL Profiler for SQL Server and Analysis Services.

How the gateway works

When a user interacts with an element that's connected to an on-premises data source:

1. The cloud service creates a query, along with the encrypted credentials for the data source, and sends the query to the queue for the gateway to process.
2. The service analyzes the query and pushes the request to the Azure Service Bus.
3. The on-premises data gateway polls the Azure Service Bus for pending requests.
4. The gateway gets the query, decrypts the credentials, and connects to the data source(s) with those credentials.
5. The gateway sends the query to the data source for execution.
6. The results are sent from the data source back to the gateway and then onto the cloud service. The service then uses the results.

Troubleshooting

Update to the latest version

A lot of issues can surface when the gateway version is out of date. It is a good general practice to make sure you are on the latest version. If you haven't updated the gateway for a month, or longer, you may want to consider installing the latest version of the gateway and see if you can reproduce the issue.

Error: Failed to add user to group. (-2147463168 PBIEgwService Performance Log Users)

You may receive this error if you are trying to install the gateway on a domain controller, which isn't supported. You'll need to deploy the gateway on a machine that isn't a domain controller.

Tools

Collecting logs from the gateway configurator

You can collect several logs for the gateway. Always start with the logs!

Installer logs

```
%localappdata%\Temp\Power_BI_Gateway_-Enterprise.log
```

Configuration logs

```
%localappdata%\Microsoft\Power BI Enterprise Gateway\GatewayConfigurator.log
```

Enterprise gateway service logs

```
C:\Users\PBIEgwService\AppData\Local\Microsoft\Power BI Enterprise Gateway\EnterpriseGateway.log
```

Event logs

The Data Management Gateway and PowerBIGateway logs are present under **Application and Services Logs**.

Fiddler Trace

Fiddler is a free tool from Telerik that monitors HTTP traffic. You can see the back and forth with the Power BI service from the client machine. This may show errors and other related information.

Next Steps

- [Create an on-premises connection to Logic Apps](#)
- [Enterprise integration features](#)
- [Logic Apps connectors](#)

Create a logic app deployment template

1/20/2017 • 4 min to read • [Edit on GitHub](#)

After a logic app has been created, you might want to create it as an Azure Resource Manager template. This way, you can easily deploy the logic app to any environment or resource group where you might need it. For an introduction to Resource Manager templates, be sure to check out the articles on [authoring Azure Resource Manager templates](#) and [deploying resources by using Azure Resource Manager templates](#).

Logic app deployment template

A logic app has three basic components:

- **Logic app resource.** This resource contains information about things like pricing plan, location, and the workflow definition.
- **Workflow definition.** This is what is seen in code view. It includes the definition of the steps of the flow and how the engine should execute. This is the `definition` property of the logic app resource.
- **Connections.** These are separate resources that securely store metadata about any connector connections, such as a connection string and an access token. You reference these in a logic app in the `parameters` section of the logic app resource.

You can view all of these pieces for existing logic apps by using a tool like [Azure Resource Explorer](#).

To make a template for a logic app to use with resource group deployments, you need to define the resources and parameterize as needed. For example, if you're deploying to a development, test, and production environment, you'll likely want to use different connection strings to a SQL database in each environment. Or, you might want to deploy within different subscriptions or resource groups.

Create a logic app deployment template

The easiest way to have a valid logic app deployment template is to use the [Visual Studio Tools for Logic Apps](#). The Visual Studio tools generate a valid deployment template that can be used across any subscription or location.

A few other tools can assist you as you create a logic app deployment template. You can author by hand, that is, by using the resources already discussed here to create parameters as needed. Another option is to use a [logic app template creator](#) PowerShell module. This open-source module first evaluates the logic app and any connections that it is using, and then generates template resources with the necessary parameters for deployment. For example, if you have a logic app that receives a message from an Azure Service Bus queue and adds data to an Azure SQL database, the tool will preserve all of the orchestration logic and parameterize the SQL and Service Bus connection strings so that they can be set at deployment.

NOTE

Connections must be within the same resource group as the logic app.

Install the logic app template PowerShell module

The easiest way to install the module is via the [PowerShell Gallery](#), by using the command `Install-Module -Name LogicAppTemplate`.

You also can install the PowerShell module manually:

1. Download the latest release of the [logic app template creator](#).
2. Extract the folder in your PowerShell module folder (usually
%UserProfile%\Documents\WindowsPowerShell\Modules).

For the module to work with any tenant and subscription access token, we recommend that you use it with the [ARMClient](#) command line tool. This [blog post](#) discusses ARMClient in more detail.

Generate a logic app template by using PowerShell

After PowerShell is installed, you can generate a template by using the following command:

```
armclient token $SubscriptionId | Get-LogicAppTemplate -LogicApp MyApp -ResourceGroup MyRG -SubscriptionId $SubscriptionId -Verbose | Out-File C:\template.json
```

\$SubscriptionId is the Azure subscription ID. This line first gets an access token via ARMClient, then pipes it through to the PowerShell script, and then creates the template in a JSON file.

Add parameters to a logic app template

After you create your logic app template, you can continue to add or modify parameters that you might need. For example, if your definition includes a resource ID to an Azure function or nested workflow that you plan to deploy in a single deployment, you can add more resources to your template and parameterize IDs as needed. The same applies to any references to custom APIs or Swagger endpoints you expect to deploy with each resource group.

Deploy a logic app template

You can deploy your template by using any number of tools, including PowerShell, REST API, Visual Studio Release Management, and the Azure Portal Template Deployment. See this article about [deploying resources by using Azure Resource Manager templates](#) for additional information. Also, we recommend that you create a [parameter file](#) to store values for the parameter.

Authorize OAuth connections

After deployment, the logic app works end-to-end with valid parameters. However, OAuth connections still will need to be authorized to generate a valid access token. You can do this by opening the logic app in the designer and then authorizing connections. Or, if you want to automate, you can use a script to consent to each OAuth connection. There's an example script on GitHub under the [LogicAppConnectionAuth](#) project.

Visual Studio Release Management

A common scenario for deploying and managing an environment is to use a tool like Visual Studio Release Management, with a logic app deployment template. Visual Studio Team Services includes a [Deploy Azure Resource Group](#) task that you can add to any build or release pipeline. You need to have a [service principal](#) for authorization to deploy, and then you can generate the release definition.

1. In Release Management, to create a new definition, select **Empty** to start with an empty definition.

Create new release definition

X

Select a template

Deployment



Azure Cloud Service Deployment

Deploy an Azure Cloud Service



Azure Website Deployment

Deploy and test your Azure website

Empty

Start with an empty definition

Next >

Cancel

2. Choose any resources you need for this. This likely will be the logic app template generated manually or as part of the build process.
3. Add an **Azure Resource Group Deployment** task.
4. Configure with a [service principal](#), and reference the Template and Template Parameters files.
5. Continue to build out steps in the release process for any other environment, automated test, or approvers as needed.

Build and Deploy Logic Apps in Visual Studio

1/20/2017 • 4 min to read • [Edit on GitHub](#)

Although the [Azure portal](#) gives you a great way to design and manage your Logic apps, you may also want to design and deploy your logic app from Visual Studio instead. Logic Apps comes with a rich Visual Studio toolset, which allows you to build a logic app using the designer, configure any deployment and automation templates, and deploy into any environment.

Installation steps

Below are the steps to install and configure the Visual Studio tools for Logic Apps.

Prerequisites

- [Visual Studio 2015](#)
- [Latest Azure SDK](#) (2.9.1 or greater)
- [Azure PowerShell](#)
- Access to the web when using the embedded designer

Install Visual Studio tools for Logic Apps

Once you have the prerequisites installed,

1. Open Visual Studio 2015 to the **Tools** menu and select **Extensions and Updates**
2. Select the **Online** category to search online
3. Search for **Logic Apps** to display the **Azure Logic Apps Tools for Visual Studio**
4. Click the **Download** button to download and install the extension
5. Restart Visual Studio after installation

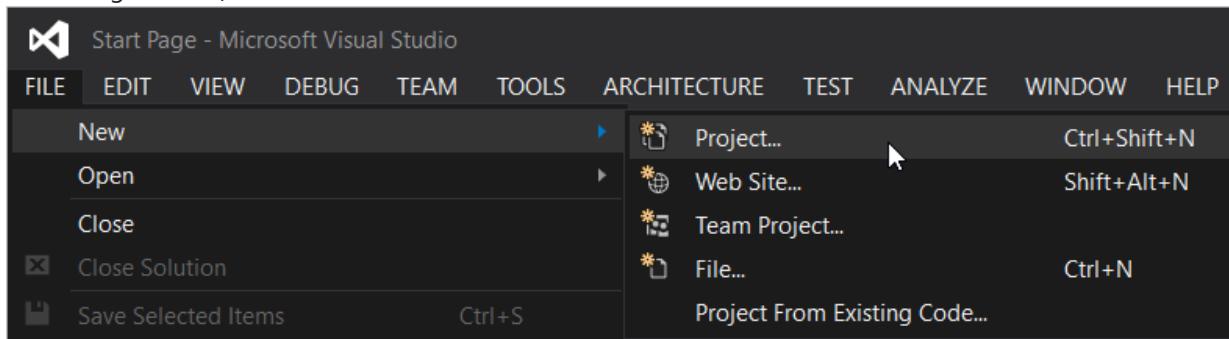
NOTE

You can also download the extension directly from [this link](#)

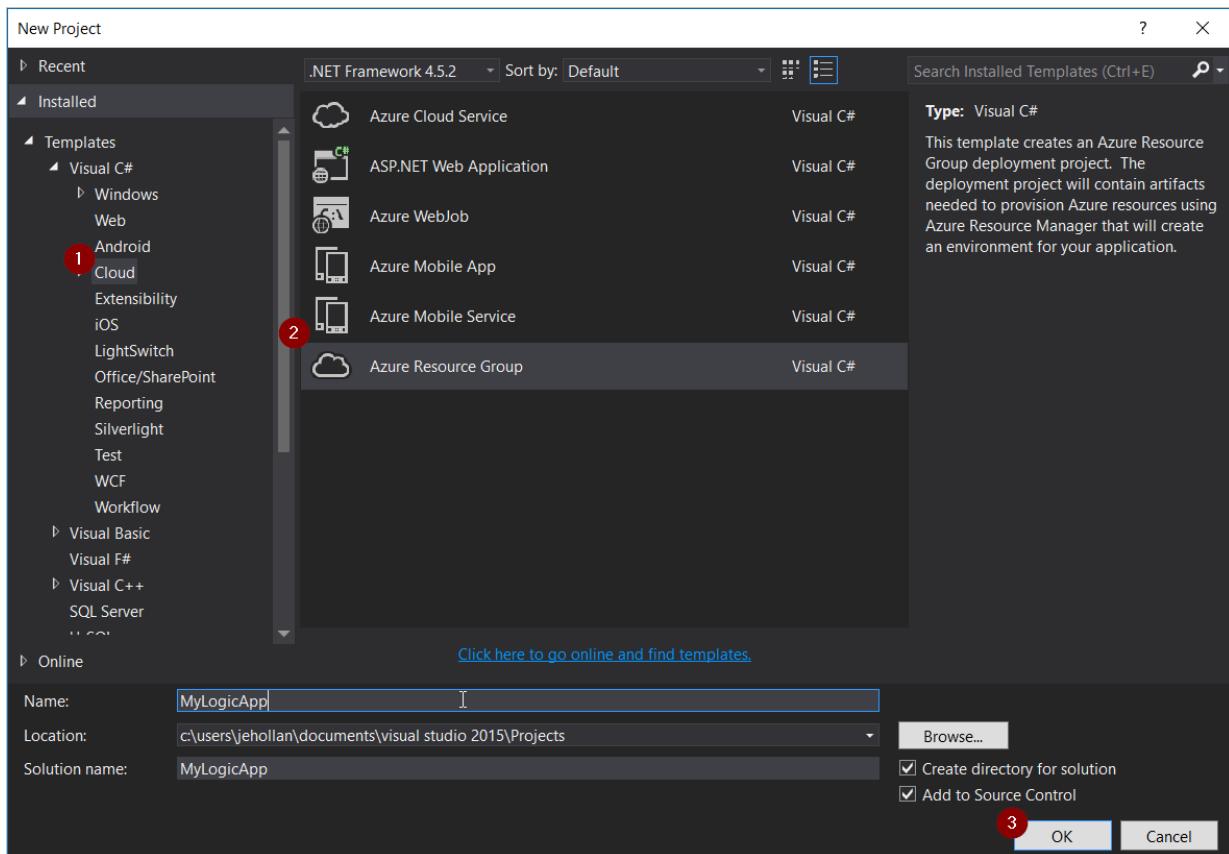
Once installed you are able to use the Azure Resource Group project with the Logic App Designer.

Create a project

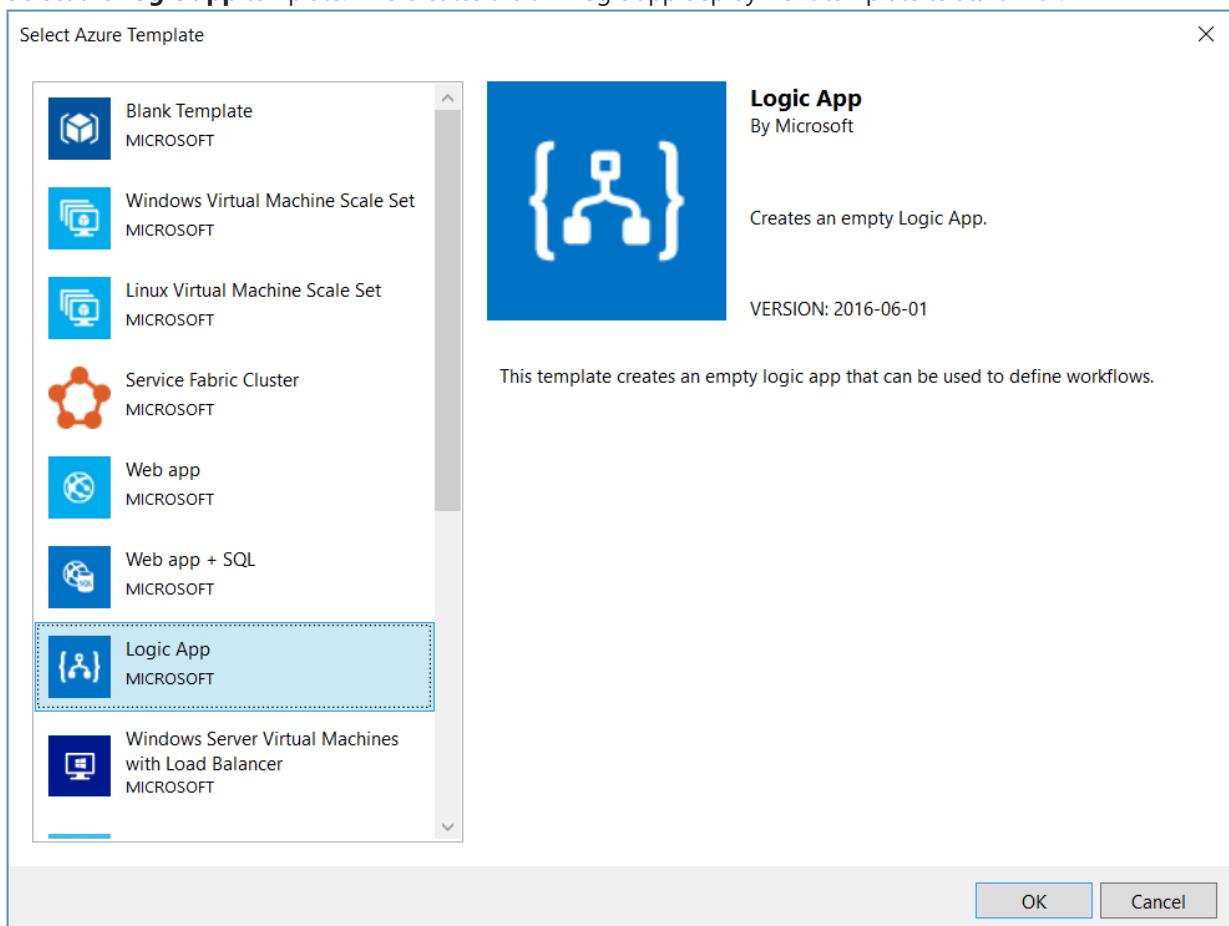
1. Go to the **File** menu and select **New > Project** (or, you can go to **Add** and then select **New project** to add it to an existing solution):



2. In the dialog, find **Cloud**, and then select **Azure Resource Group**. Type a **Name** and then click **OK**.

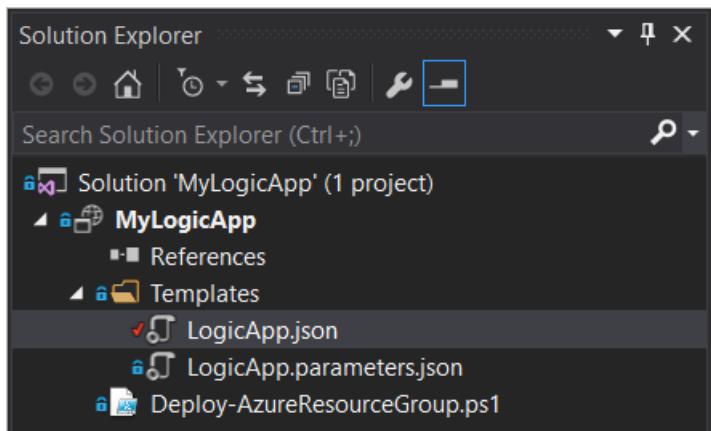


3. Select the **Logic app** template. This creates a blank logic app deployment template to start with.



4. Once you have selected your **Template**, hit **OK**.

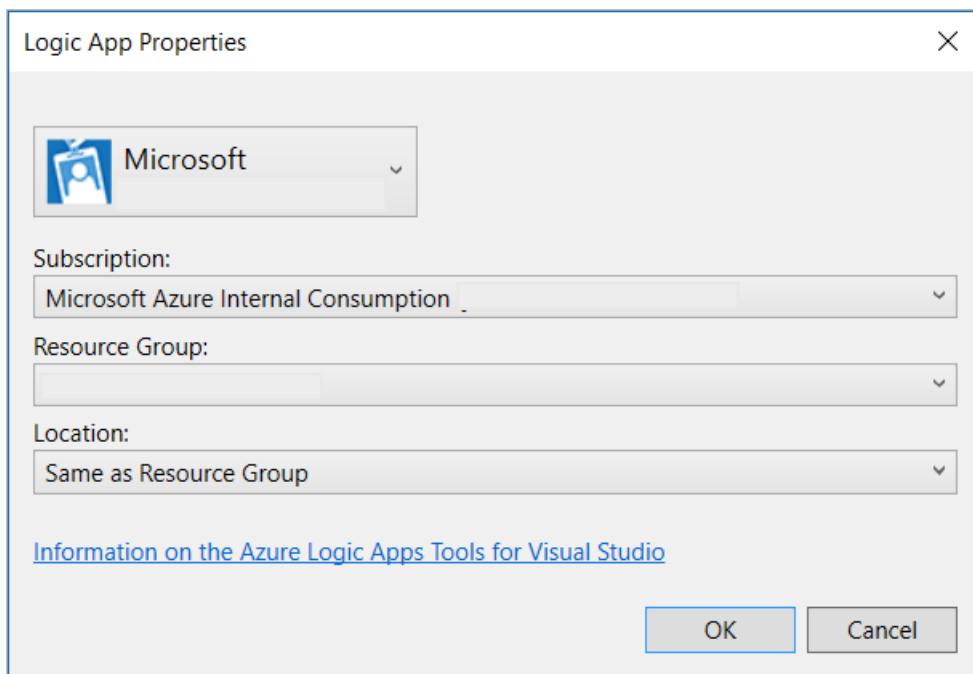
Now your Logic app project is added to your solution. You should see the deployment file in the Solution Explorer:



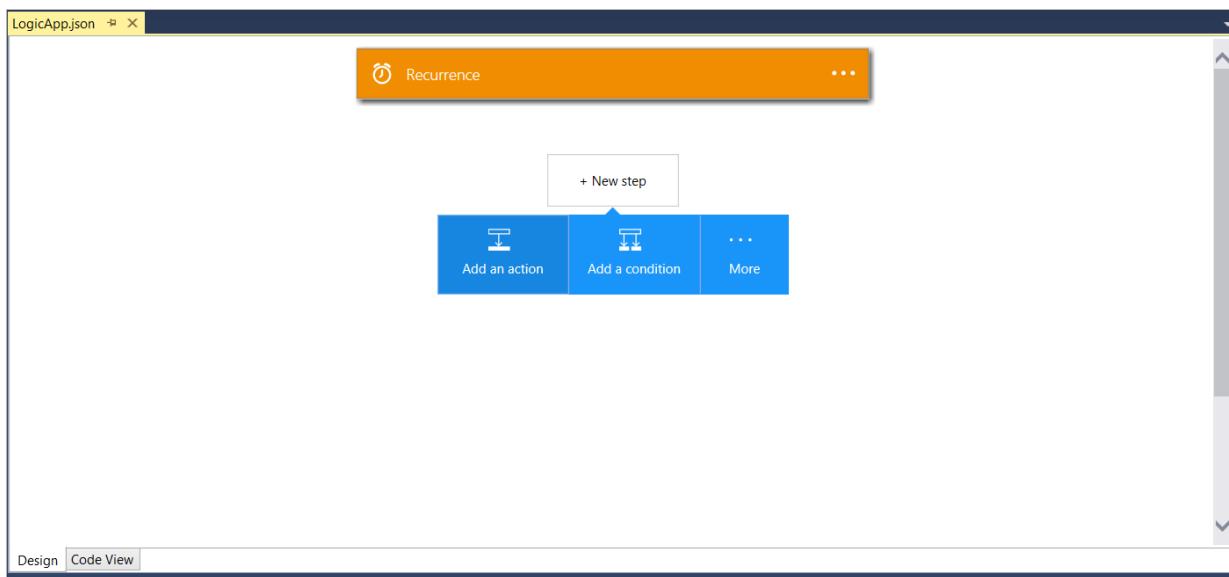
Using the Logic App Designer

Once you have an Azure Resource Group project that contains a logic app, you can open the designer within Visual Studio to assist you in creating the workflow. The designer requires an internet connection to query the connectors for available properties and data (for example, if using the Dynamics CRM Online connector, the designer queries your CRM instance to list available custom and default properties).

1. Right-click on the `<template>.json` file and select **Open with Logic App Designer** (or `Ctrl+L`)
2. Choose the subscription, resource group, and location for the deployment template
 - It's important to note that designing a logic app will create **API Connection** resources to query for properties during design. The resource group selected is used to create those connections during design-time. You can view or modify any API Connections by going to the Azure portal and browsing for **API Connections**.



3. The designer should render based on the definition in the `<template>.json` file.
4. You can now create and design your logic app, and changes are updated in the deployment template.



`Microsoft.Web/connections` resources are added to your resource file for any connections needed for the logic app to function. These connection properties can be set when you deploy, and managed after you deploy in **API Connections** in the Azure portal.

Switching to the JSON code-view

You can select the **Code View** tab on the bottom of the designer to switch to the JSON representation of the logic app. To switch back to the full resource JSON, right-click the `<template>.json` file and select **Open**.

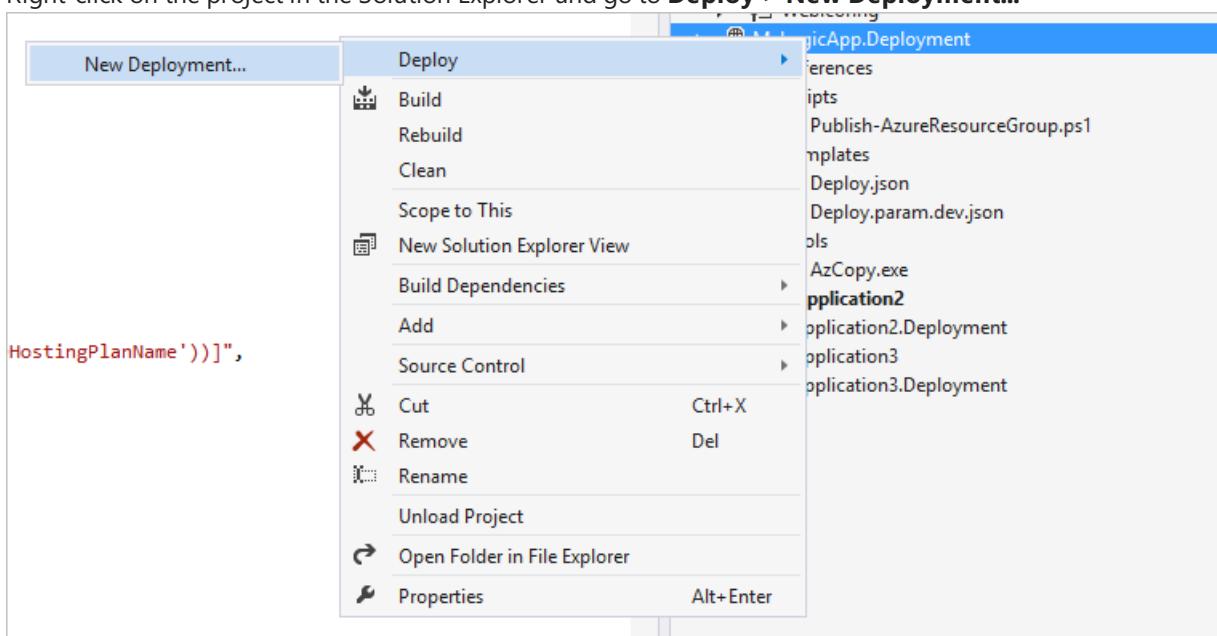
Saving the logic app

You can save the logic app at anytime via the **Save** button or `ctrl+s`. If there are any errors with your logic app at the time you save, they are displayed in the **Outputs** window of Visual Studio.

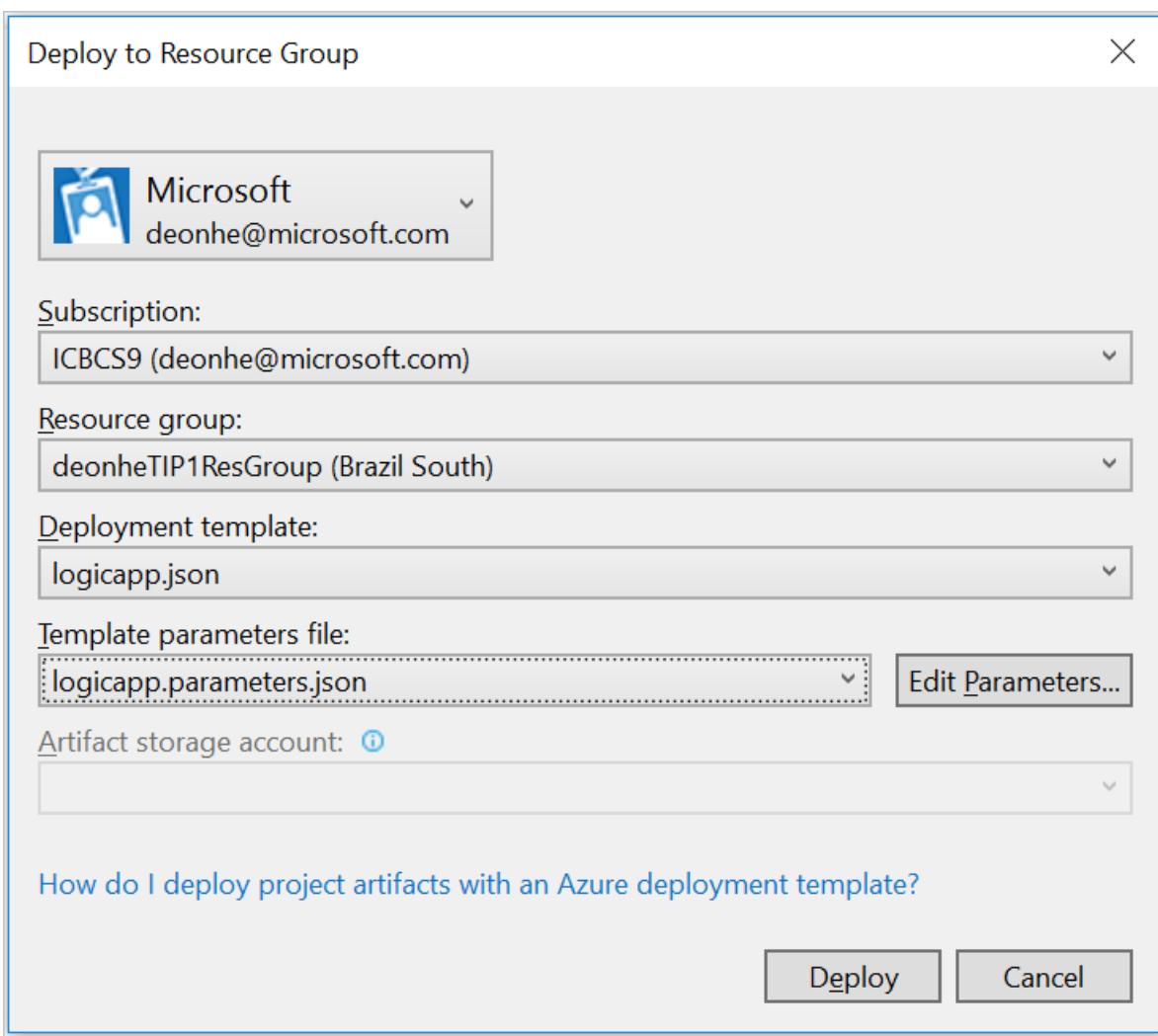
Deploying your Logic app

Finally, after you have configured your app, you can deploy directly from Visual Studio in just a couple steps.

1. Right-click on the project in the Solution Explorer and go to **Deploy > New Deployment...**



2. You are prompted to sign in to your Azure subscription(s).
3. Now you need to choose the details of the resource group that you want to deploy the Logic app to.



NOTE

Be sure to select the right template and parameters files for the resource group (for example if you are deploying to a production environment you'll want to choose the production parameters file).

4. Select the Deploy button
5. The status of the deployment appears in the **Output** window (you may need to choose **Azure Provisioning**.

```
Output: Azure Provisioning
Show output from: Azure Provisioning
05:43:57 - Transfer summary:
05:43:57 - -----
05:43:57 - Total files transferred: 3
05:43:57 - Transfer successfully: 3
05:43:57 - Transfer failed: 0
05:43:57 - Elapsed time: 00.00:00:02
05:43:57 - Done building target "UploadDrop" in project "MyLogicApp.Deployment.deployproj".
05:43:57 - Done building project "MyLogicApp.Deployment.deployproj".
05:43:57 - Build succeeded.
05:43:57 - The following parameter values will be used for this deployment:
05:43:57 -     dropLocation: https://deploymentlogs.blob.core.windows.net/webapplication2
05:43:57 -     dropLocationSasToken: <securestring>
05:43:57 -     webSitePackage: MyLogicApp/package.zip
05:43:57 -     webSiteName: blogordns
05:43:57 -     webSiteHostingPlanName: noentuhoneuth
05:43:57 -     webSiteLocation: westus
05:43:57 -     webSiteHostingPlanSKU: Free
05:43:57 -     webSiteHostingPlanWorkerSize: 0
05:43:57 - Starting deployment. This may take a while...
05:43:57 - Uploading template to Azure storage account 'deploymentlogs'.
05:43:58 - Creating resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10' in location 'westus'.
05:44:12 - Creating deployment 'VS_deploy' in resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10'.
```

In the future, you can revise your Logic app in source control and use Visual Studio to deploy new versions.

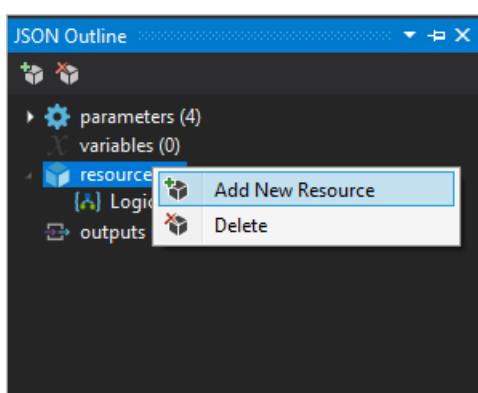
NOTE

If you modify the definition in the Azure portal directly, then the next time you deploy from Visual Studio those changes will be overwritten.

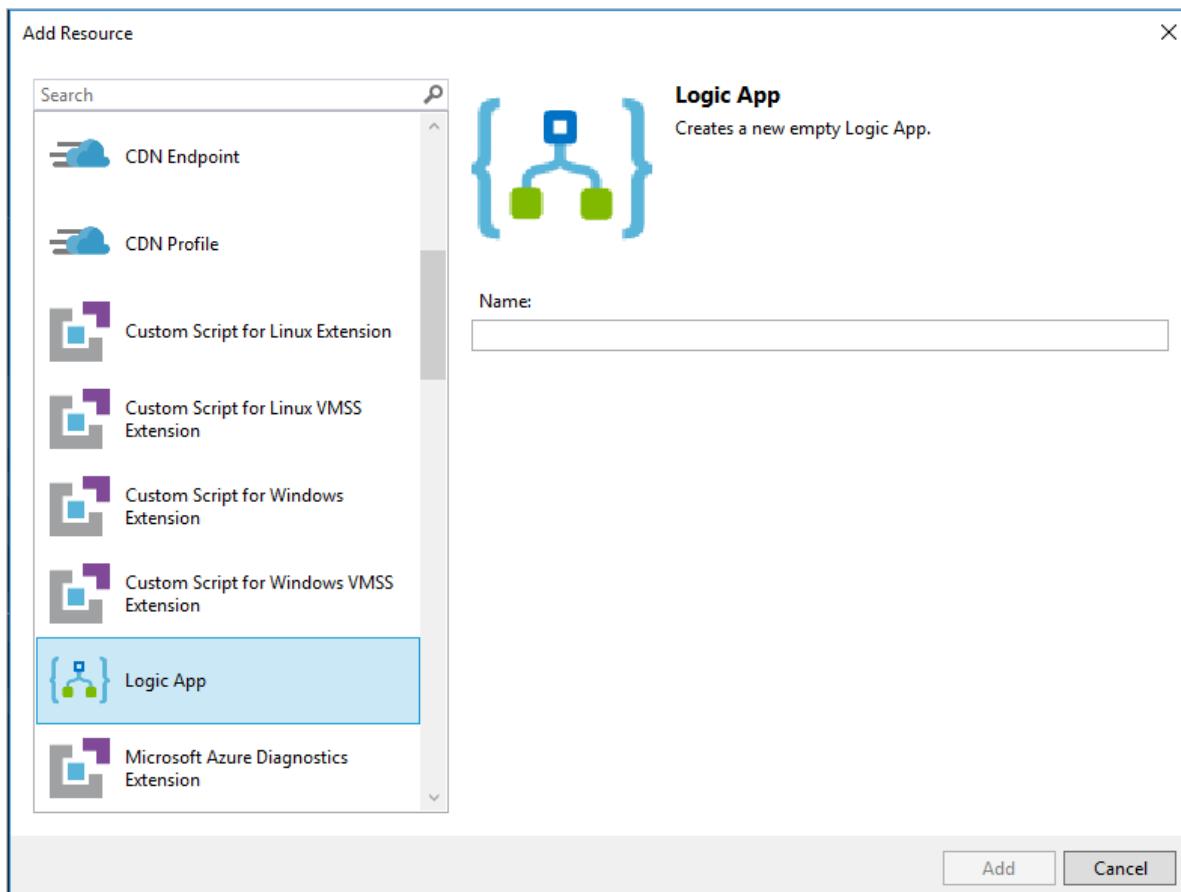
Adding a Logic App to an existing Resource Group project

If you have an existing Resource Group project, then adding a logic app to it, or adding another logic app along side the one you previously created, can be done through the JSON Outline window.

1. Open the `<template>.json` file.
2. Open the JSON Outline window. The JSON Outline window can be found under **View > Other Windows > JSON Outline**.
3. To add a resource to the template file, either click the Add Resource button on the top of the JSON Outline window or right-click on **resources** and select **Add New Resource**.



4. In the **Add Resource** dialog box browse and select **Logic App**, give it a name and select **Add**.



Next Steps

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- You can automate business processes with Logic Apps
- [Learn How to Integrate your systems with Logic Apps](#)

Manage Logic Apps with the Visual Studio Cloud Explorer

1/20/2017 • 2 min to read • [Edit on GitHub](#)

Although the [Azure portal](#) gives you a great way to design and manage your Logic Apps, the Visual Studio Cloud Explorer allows you to manage many of your Azure assets from within Visual Studio including Logic Apps. With the cloud explorer you can browse published Logic Apps, perform actions like enable, disable, edit and view run histories.

Installation steps

Below are the steps to install and configure the Visual Studio tools for Logic Apps.

Prerequisites

- [Visual Studio 2015](#)
- [Latest Azure SDK](#) (2.9.1 or greater)
- [Visual Studio Cloud Explorer](#)
- Access to the web when using the embedded designer

Install Visual Studio tools for Logic Apps

Once you have the prerequisites installed,

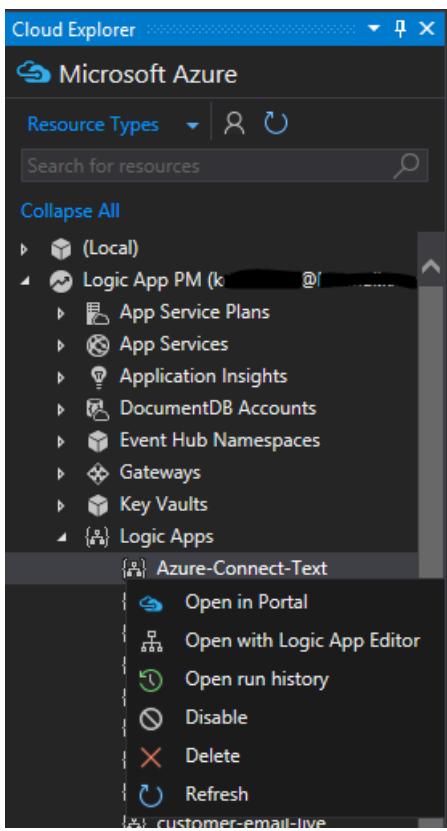
1. Open Visual Studio 2015 to the **Tools** menu and select **Extensions and Updates**
2. Select the **Online** category to search online
3. Search for **Logic Apps** to display the **Azure Logic Apps Tools for Visual Studio**
4. Click the **Download** button to download and install the extension
5. Restart Visual Studio after installation

NOTE

You can also download the extension directly from [this link](#)

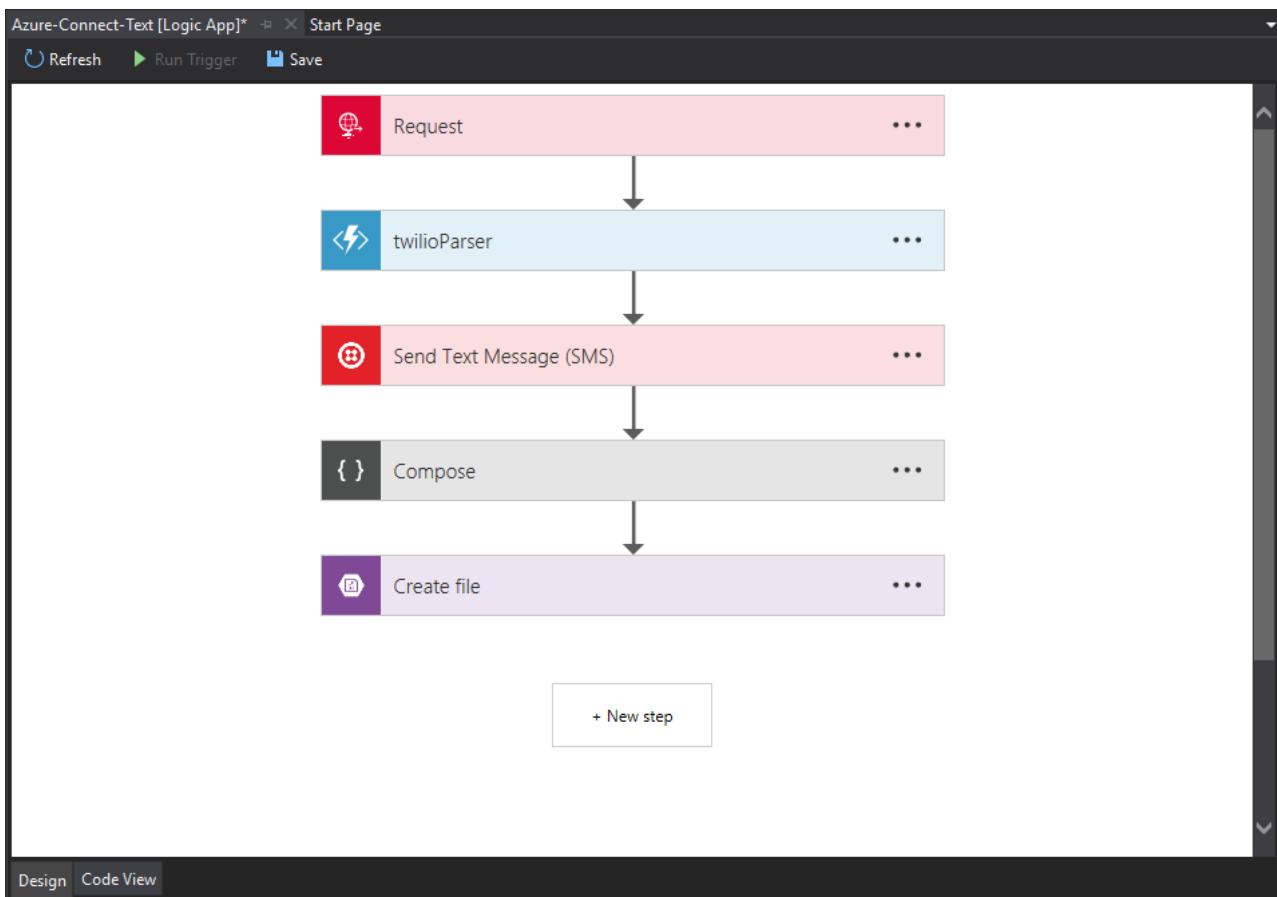
Browsing Logic Apps

To browse Logic Apps in Cloud Explorer open the cloud explorer under View > Cloud Explorer and you can either browse by resource group or by resource type. If browsing by resource type, select a subscription and then expand the Logic Apps section then select a Logic App. You can either right-click one of your Logic Apps or use the Actions menu at the bottom of the cloud explorer to perform the desired action.



Edit Logic App with the designer

To open the currently deployed Logic App in the same designer that is in the Azure portal, right-click on the Logic App and select "Open with Logic App Editor". With the designer, you can edit the Logic App and save it back to the cloud and start a new run with "Run Trigger" button.



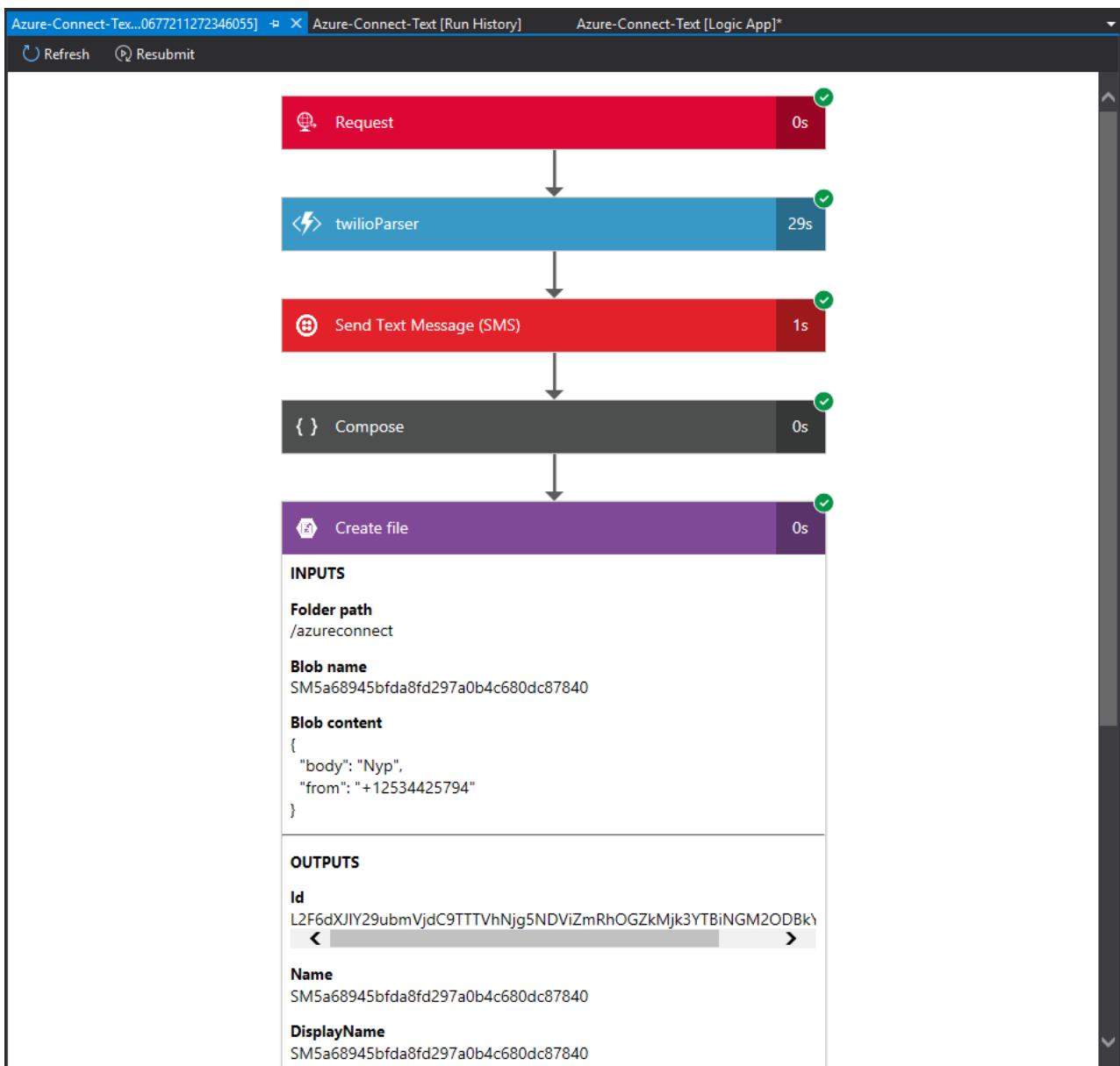
Browse Logic App run history

To list the run history for a Logic App, right-click on the Logic App and select "Open run history". In this view you can order by any of the shown properties by selecting the column header.

Azure-Connect-Text [Run History] ✖ Azure-Connect-Text [Logic App]*				
Status	Identifier	Start Time	End Time	Duration
✔ Succeeded	08587200335520677211272346055	12/12/2016 12:08:53 PM	12/12/2016 12:09:23 PM	30.22 Seconds
✔ Succeeded	08587209094198264575890638948	12/2/2016 8:51:05 AM	12/2/2016 8:51:32 AM	26.67 Seconds
✔ Succeeded	08587236421059776808982596931	10/31/2016 6:46:19 PM	10/31/2016 6:46:51 PM	32.3 Seconds
✔ Succeeded	08587267749235988305407694526	9/25/2016 12:32:41 PM	9/25/2016 12:33:20 PM	38.28 Seconds

4 Results

Double-clicking on one of the run instances shows the run history for that instance where you can see the results of the run including the inputs and outputs of each step.



Next steps

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- [You can automate business processes with Logic Apps](#)
- [Learn How to Integrate your systems with Logic Apps](#)

Securing a Logic App

1/20/2017 • 9 min to read • [Edit on GitHub](#)

There are many tools available to help you secure your logic app.

- Securing access to trigger a logic app (HTTP Request Trigger)
- Securing access to manage, edit, or read a logic app
- Securing access to contents of inputs and outputs for a run
- Securing parameters or inputs within actions in a workflow
- Securing access to services that receive requests from a workflow

Secure access to trigger

When working with a logic app that fires on an HTTP Request ([Request](#) or [Webhook](#)), you can restrict access so only authorized clients can fire the logic app. All requests into a logic app are encrypted and secured via SSL.

Shared Access Signature

Every request endpoint for a logic app includes a [Shared Access Signature](#) (SAS) as part of the URL. Each URL contains a `sp`, `sv`, and `sig` query parameter. Permissions are specified by `sp`, and correspond to HTTP methods allowed, `sv` is the version used to generate, and `sig` is used to authenticate access to trigger. It is generated using the SHA256 algorithm with a secret key on all the URL paths and properties. The secret key is never exposed and published, and is kept encrypted and stored as part of the logic app. Your logic app will only authorize triggers that contain a valid signature created with the secret key.

Regenerate access keys

You can regenerate a new secure key at anytime via the REST API or Azure portal. All current URLs that were generated previously using the old key will be invalidated and no longer authorized to fire the logic app.

1. In the Azure portal, open the logic app you want to regenerate a key
2. Click the **Access Keys** menu item under **Settings**
3. Choose the key to regenerate and complete the process

URLs you retrieve after regeneration are signed with the new access key.

Creating callback URLs with an expiration date

If you are sharing out the URL with other parties, you can generate URLs with specific keys and expiration dates as needed. This allows you to seamlessly roll keys, or ensure access to fire an app is restricted to a certain timespan. You can specify an expiration date for a URL through the [logic apps REST API](#) as follows:

```
POST  
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Logic/workflows/{workflo  
wName}/triggers/{triggerName}/listCallbackUrl?api-version=2016-06-01
```

In the body, include the property `NotAfter` as a JSON date string, which returns a callback URL which will only be valid until the `NotAfter` date and time.

Creating URLs with primary or secondary secret key

When you generate or list callback URLs for request-based triggers, you can also specify which key to use to sign the URL. You can generate a URL signed by a specific key through the [logic apps REST API](#) as follows:

```
POST  
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Logic/workflows/{workflo  
wName}/triggers/{triggerName}/listCallbackUrl?api-version=2016-06-01
```

In the body, include the property `KeyType` as either `Primary` or `Secondary`. This returns a URL signed by the secure key specified.

Restrict incoming IP addresses

In addition to the Shared Access Signature, you may wish to restrict calling a logic app only from specific clients. For example, if you manage your endpoint through Azure API Management, you can restrict the logic app to only accept the request when the request comes from the API Management instance IP address.

This setting can be configured within the logic app settings:

1. In the Azure portal, open the logic app you want to add IP address restrictions
2. Click the **Access control configuration** menu item under **Settings**
3. Specify the list of IP address ranges to be accepted by the trigger

A valid IP range takes the format `192.168.1.1/255`. If you want the logic app to only fire as a nested logic app, select the **Only other logic apps** option. This will write an empty array to the resource, meaning only calls from the service itself (parent logic apps) will successfully fire.

NOTE

A logic app with a request trigger could still be executed through the REST API / Management `/triggers/{triggerName}/run` regardless of IP. This would require authentication against the Azure REST API, and all events would appear in the Azure Audit Log. Set access control policies accordingly.

Setting IP ranges on the resource definition

If you are using a [deployment template](#) to automate your deployments, the IP range settings can be configured on the resource template.

```
{  
    "properties": {  
        "definition": {  
        },  
        "parameters": {},  
        "accessControl": {  
            "triggers": {  
                "allowedCallerIpAddresses": [  
                    {  
                        "addressRange": "192.168.12.0/23"  
                    },  
                    {  
                        "addressRange": "2001:0db8::/64"  
                    }  
                ]  
            }  
        },  
        "type": "Microsoft.Logic/workflows"  
    }  
}
```

Adding Azure Active Directory, OAuth, or other security

If you want to add any additional authorization protocols on top of a logic app, use [Azure API Management](#). This provides rich monitoring, security, policy, and documentation for any endpoint, allowing a logic app to be exposed as an API. Azure API Management can expose a public or private endpoint for the logic app, which could leverage

Azure Active Directory, certificate, OAuth, or other security standards. When a request is received, Azure API Management will forward the request to the logic app (performing any needed transformations or restrictions in-flight). You can use the incoming IP range settings on the logic app to only allow the logic app to be triggered from API Management.

Secure access to manage or edit a logic app

You can restrict access to management operations on a logic app so that only specific users or groups are able to perform operations on the resource. Logic apps use the Azure [Role-Based Access Control \(RBAC\)](#) feature, and can be customized with the same tools. There are a few built-in roles you can assign members of your subscription to as well:

- **Logic App Contributor** - Provides access to view, edit, and update a logic app. Cannot remove the resource or perform admin operations.
- **Logic App Operator** - Can view the logic app and run history, and enable/disable. Cannot edit or update the definition.

You can also leverage the [Azure Resource Lock](#) to prevent modification or deletion of a logic app. This is valuable to prevent production resources from being modified or deleted.

Secure access to contents of the run history

You can restrict access to contents of inputs or outputs from previous runs to specific IP address ranges.

All data within a workflow run is encrypted in transit and at rest. When a call to run history is made, the service authenticates the request and provides links to the request and response inputs and outputs. This link can be protected so only requests to view content from a designated IP address range return the contents. This can be used for additional access control. You could even specify an IP address like `0.0.0.0` so no one could access inputs/outputs. Only someone with admin permissions could remove this restriction, providing the possibility for 'just-in-time' access to workflow contents.

This setting can be configured within the resource settings of the Azure portal:

1. In the Azure portal, open the logic app you want to add IP address restrictions
2. Click the **Access control configuration** menu item under **Settings**
3. Specify the list of IP address ranges for access to content

Setting IP ranges on the resource definition

If you are using a [deployment template](#) to automate your deployments, the IP range settings can be configured on the resource template.

```
{
  "properties": {
    "definition": {
      },
      "parameters": {},
      "accessControl": {
        "contents": [
          "allowedCallerIpAddresses": [
            {
              "addressRange": "192.168.12.0/23"
            },
            {
              "addressRange": "2001:0db8::/64"
            }
          ]
        }
      }
    },
    "type": "Microsoft.Logic/workflows"
  }
}
```

Secure parameters and inputs within a workflow

There are some aspects of a workflow definition you may wish to parametrize for deployment across environments. In addition, some of those parameters may be secure parameters you do not want to appear when editing a workflow - such as a client ID and client secret for [Azure Active Directory authentication](#) of an HTTP action.

Using parameters and secure parameters

The [workflow definition language](#) provides a `@parameters()` operation to access the value of a resource parameter at runtime. In addition, you can [specify parameters in the resource deployment template](#). If you specify the parameter type to be `securestring`, it will not be returned with the rest of the resource definition - meaning it will not be accessible by viewing the resource after deployment.

NOTE

If your parameter is used in the headers or body of a request, it may be visible by accessing the run history and outgoing HTTP request. Be sure to set your content access policies accordingly. Authorization headers are never visible via inputs or outputs - so if the secret is being used there it will not be retrievable.

Resource deployment template with secrets

The following is an example of a deployment which references a secure parameter of `secret` at runtime. In a separate parameters file I could specify the environment value for the `secret`, or leverage [Azure Resource Manager KeyVault](#) to retrieve my secrets at deploy-time.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "secretDeploymentParam": {
            "type": "securestring"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "secret-deploy",
            "type": "Microsoft.Logic/workflows",
            "location": "westus",
            "tags": {
                "displayName": "LogicApp"
            },
            "apiVersion": "2016-06-01",
            "properties": {
                "definition": {
                    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
                    "actions": {
                        "Call_External_API": {
                            "type": "http",
                            "inputs": {
                                "headers": {
                                    "Authorization": "@parameters('secret')"
                                },
                                "body": "This is the request"
                            },
                            "runAfter": {}
                        }
                    },
                    "parameters": {
                        "secret": {
                            "type": "SecureString"
                        }
                    },
                    "triggers": {
                        "manual": {
                            "type": "Request",
                            "kind": "Http",
                            "inputs": {
                                "schema": {}
                            }
                        }
                    },
                    "contentVersion": "1.0.0.0",
                    "outputs": {}
                },
                "parameters": {
                    "secret": {
                        "value": "[parameters('secretDeploymentParam')]"
                    }
                }
            }
        }
    ],
    "outputs": {}
}
```

Secure access to services receiving requests from a workflow

There are many ways to help secure any endpoint the logic app needs to access.

Using authentication on outbound requests

When working with an HTTP, HTTP + Swagger (Open API), or Webhook action, you can add authentication to the request being sent. This could include basic authentication, certificate authentication, or Azure Active Directory authentication. Details on how to configure this authentication can be found [in this article](#).

Restricting access to logic app IP addresses

All calls from logic apps come from a specific set of IP addresses per region. You can add additional filtering to only accept requests from those designated IP addresses. A list of those IP addresses can be found [in this article](#).

On-premises connectivity

Logic apps provides integration with a number of services to provide secure and reliable on-premises communication.

On-premises data gateway

Many of the managed connectors from logic apps provide secure connectivity to on-premises systems, including File System, SQL, SharePoint, DB2, and more. The gateway leverages encrypted channels via Azure Service Bus to relay data on-premises, and all traffic originates from secure outbound traffic from the gateway agent. More details on how the gateway works [in this article](#).

Azure API Management

[Azure API Management](#) has a number of on-premises connectivity options including site-to-site VPN and ExpressRoute integration for secured proxy and communication to on-premises systems. In the logic app designer, you can quickly select an API exposed from Azure API Management within a workflow, providing quick access to on-premises systems.

Hybrid connections from Azure App Services

You can use the on-premises hybrid connection feature for Azure API and Web apps to communicate on-premises. Details on hybrid connections and how to configure can be found [in this article](#).

Next steps

[Create a deployment template](#)

[Exception handling](#)

[Monitor your logic apps](#)

[Diagnosing logic app failures and issues](#)

Monitor your Logic apps

1/20/2017 • 4 min to read • [Edit on GitHub](#)

After you [create a Logic app](#), you can see the full history of its execution in the Azure portal. You can also set up services like Azure Diagnostics and Azure Alerts to monitor events real-time, and alert you for events like "when more than 5 runs fail within an hour."

Monitor in the Azure Portal

To view the history, select **Browse**, and select **Logic Apps**. A list of all logic apps in your subscription is displayed. Select the logic app you want to monitor. You will see a list of all actions and triggers that have occurred for this logic app.

The screenshot shows the Azure Logic App Overview blade. On the left is a navigation menu with sections: QUICK ACCESS (Activity logs, Access control (IAM), Tags), SETTINGS (Locks, Automation script), GENERAL (Logic App Definition, Diagnostics, Properties, Quick Start Guides, Integration Account), and SUPPORT + TROUBLESHOOTING (New support request). The main area is titled 'Summary' and shows a table of 'All runs'. The table has columns: STATUS, START TIME, and DURATION. All runs listed are 'Succeeded' at 7/22/2016, 7:48 PM, with durations ranging from 248 to 331 milliseconds. Below this is a 'Trigger History' section with a similar table structure, showing four trigger executions all labeled 'Fired' at the same time.

STATUS	START TIME	DURATION
Succeeded	7/22/2016, 7:48 PM	248 Milliseconds
Succeeded	7/22/2016, 7:48 PM	248 Milliseconds
Succeeded	7/22/2016, 7:48 PM	294 Milliseconds
Succeeded	7/22/2016, 7:48 PM	275 Milliseconds
Succeeded	7/22/2016, 7:48 PM	249 Milliseconds
Succeeded	7/22/2016, 7:48 PM	275 Milliseconds
Succeeded	7/22/2016, 7:48 PM	260 Milliseconds
Succeeded	7/22/2016, 7:48 PM	331 Milliseconds

STATUS	START TIME	FIRE
Succeeded	7/22/2016, 7:48 PM	Fired
Succeeded	7/22/2016, 7:48 PM	Fired
Succeeded	7/22/2016, 7:48 PM	
Succeeded	7/22/2016, 7:48 PM	Fired

There are a few sections on this blade that are helpful:

- **Summary** lists **All runs** and the **Trigger History**
 - **All runs** list the latest logic app runs. You can click any row for details on the run, or click on the tile to list more runs.
 - **Trigger History** lists all the trigger activity for this logic app. Trigger activity could be a "Skipped" check for new data (e.g. looking to see if a new file was added to FTP), "Succeeded" meaning data was returned to fire a logic app, or "Failed" corresponding an error in configuration.
- **Diagnostics** allows you to view runtime details and events, and subscribe to [Azure Alerts](#)

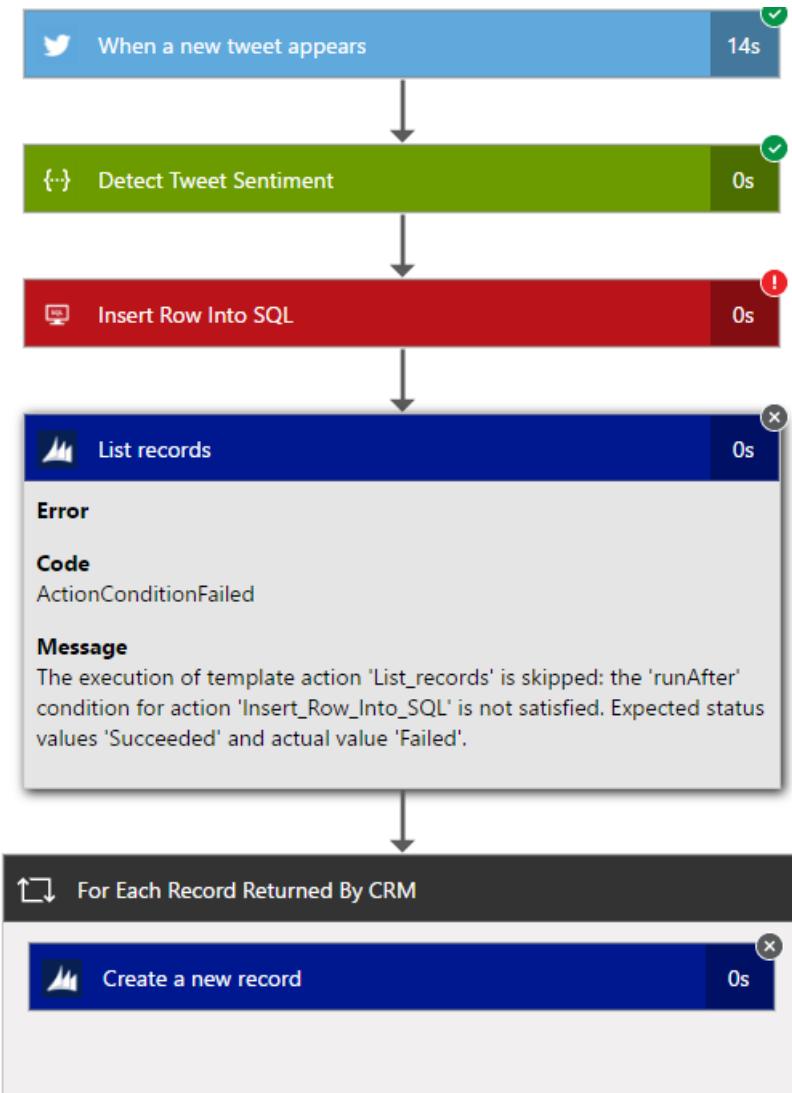
NOTE

All runtime details and events are encrypted at rest within the Logic App service. They are only decrypted upon a view request from a user. Access to these events can also be controlled by Azure Role-Based Access Control (RBAC).

View the run details

This list of runs shows the **Status**, the **Start Time**, and the **Duration** of the particular run. Select any row to see details on that run.

The monitoring view shows you each step of the run, the inputs and outputs, and any error messages that may have occurred.



If you need any additional details like the run **Correlation ID** (that can be used for the REST API), you can click the **Run Details** button. This includes all steps, status, and inputs/outputs for the run.

Azure Diagnostics and alerts

In addition to the details provided by the Azure Portal and REST API above, you can configure your logic app to use Azure Diagnostics for more rich details and debugging.

1. Click the **Diagnostics** section of the logic app blade
2. Click to configure the **Diagnostic Settings**
3. Configure an Event Hub or Storage Account to emit data to

Diagnostics

Save **Discard**

Status

Export to Event Hubs

Service Bus Namespace >

Export to Storage Account

Storage Account >

LOGS

WorkflowRuntime Retention (days)

METRICS

1 minute Retention (days)

 The storage account must be in the same region as the resource.

Adding Azure Alerts

Once diagnostics are configured, you can add Azure Alerts to fire when certain thresholds are crossed. In the **Diagnostics** blade, select the **Alerts** tile and **Add alert**. This will walk you through configuring an alert based on a number of thresholds and metrics.

* Name ⓘ

Description

* Metric ⓘ

Action Latency

- Action Latency
- Action Success Latency
- Action Throttled Events
- Actions Completed
- Actions Failed
- Actions Skipped
- Actions Started
- Actions Succeeded
- Run Latency
- Run Success Latency
- Run Throttled Events
- Runs Cancelled
- Runs Completed
- Runs Failed**
- Runs Started
- Runs Succeeded
- Trigger Fire Latency
- Trigger Latency
- Trigger Success Latency
- Trigger Throttled Events

* Period ⓘ

Over the last 5 minutes

You can configure the **Condition**, **Threshold**, and **Period** as desired. Finally, you can configure an email address to send a notification to, or configure a webhook. You can use the [request trigger](#) in a logic app to run on an alert as well (to do things like [post to Slack](#), [send a text](#), or [add a message to a queue](#)).

Azure Diagnostics Settings

Each of these events contains details about the logic app and event like status. Here is an example of a *ActionCompleted* event:

```
{
    "time": "2016-07-09T17:09:54.4773148Z",
    "workflowId": "/SUBSCRIPTIONS/80D4FE69-ABCD-EFGH-A938-
9250F1C8AB03/RESOURCEGROUPS/MYRESOURCEGROUP/PROVIDERS/MICROSOFT.LOGIC/WORKFLOWS/MYLOGICAPP",
    "resourceId": "/SUBSCRIPTIONS/80D4FE69-ABCD-EFGH-A938-
9250F1C8AB03/RESOURCEGROUPS/MYRESOURCEGROUP/PROVIDERS/MICROSOFT.LOGIC/WORKFLOWS/MYLOGICAPP/RUNS/0858736114692-
2712057/ACTIONS/HTTP",
    "category": "WorkflowRuntime",
    "level": "Information",
    "operationName": "Microsoft.Logic/workflows/workflowActionCompleted",
    "properties": {
        "$schema": "2016-06-01",
        "startTime": "2016-07-09T17:09:53.4336305Z",
        "endTime": "2016-07-09T17:09:53.5430281Z",
        "status": "Succeeded",
        "code": "OK",
        "resource": {
            "subscriptionId": "80d4fe69-ABCD-EFGH-a938-9250f1c8ab03",
            "resourceGroupName": "MyResourceGroup",
            "workflowId": "cff00d5458f944d5a766f2f9ad142553",
            "workflowName": "MyLogicApp",
            "runId": "08587361146922712057",
            "location": "eastus",
            "actionName": "Http"
        },
        "correlation": {
            "actionTrackingId": "e1931543-906d-4d1d-baed-dee72ddf1047",
            "clientTrackingId": "my-custom-tracking-id"
        },
        "trackedProperties": {
            "myProperty": "<value>"
        }
    }
}
```

The two properties that are especially useful for tracking and monitoring are *clientTrackingId* and *trackedProperties*.

Client tracking ID

The client tracking ID is a value that will correlate events across a logic app run, including any nested workflows called as a part of a logic app. This ID will be auto-generated if not provided, but you can manually specify the client tracking ID from a trigger by passing a `x-ms-client-tracking-id` header with the ID value in the trigger request (request trigger, HTTP trigger, or webhook trigger).

Tracked properties

Tracked properties can be added onto actions in the workflow definition to track inputs or outputs in diagnostics data. This can be useful if you wish to track data like an "order ID" in your telemetry. To add a tracked property, include the `trackedProperties` property on an action. Tracked properties can only track a single actions inputs and outputs, but you can use the `correlation` properties of the events to correlate across actions in a run.

```
{  
    "myAction": {  
        "type": "http",  
        "inputs": {  
            "uri": "http://uri",  
            "headers": {  
                "Content-Type": "application/json"  
            },  
            "body": "@triggerBody()"  
        },  
        "trackedProperties":{  
            "myActionHTTPStatusCode": "@action()['outputs']['statusCode']",  
            "myActionHTTPValue": "@action()['outputs']['body']['foo']",  
            "transactionId": "@action()['inputs']['body']['bar']"  
        }  
    }  
}
```

Extending your solutions

You can leverage this telemetry from the Event Hub or Storage into other services like [Operations Management Suite](#), [Azure Stream Analytics](#), and [Power BI](#) to have real time monitoring of your integration workflows.

Next Steps

- [Common examples and scenarios for logic apps](#)
- [Creating a Logic App Deployment Template](#)
- [Enterprise integration features](#)

Monitor B2B messages

1/20/2017 • 1 min to read • [Edit on GitHub](#)

B2B communication involves message exchanges between two running business processes or applications. The relationship defines an agreement between business processes. Once the communication has been established, there needs to be a way to monitor if the communication is working as expected. Message tracking has been implemented for the B2B protocols: AS2, X12, and EDIFACT. You can configure your Integration Account to use Diagnostics for richer details and debugging

Prerequisites

- An Azure account; you can create a [free account](#)
- An Integration Account; you can create an [Integration Account](#)
- A Logic App; you can create a [Logic App](#) and [enable logging](#)

Enable logging for an integration account

You can enable logging for an integration account either with **Azure portal** or with **Monitor**

Enable logging with Azure portal

1. Select **integration account** and select **diagnostics logs**

The screenshot shows the Azure portal interface. On the left, the 'All resources' blade is open, displaying a list of subscriptions. One subscription, 'integrationaccount', is selected and highlighted with a red box. On the right, the main content area is titled 'IntegrationAccount - Diagnostics logs'. It contains several sections: 'Overview', 'Access control (IAM)', 'Tags', 'SETTINGS' (which includes 'Callback URL', 'Schemas', 'Maps', 'Certificates', 'Partners', 'Agreements', 'Properties', 'Locks', and 'Automation script'), 'MONITORING' (which includes 'Diagnostics logs' and 'Log search'), and a 'Gain insights across Azure' sidebar.

2. Select your **Subscription** and **Resource Group, Integration Account** from Resource Type and select your

Integration Account from Resource drop-down to enable diagnostics. Click **Turn on Diagnostics** to enable diagnostics for the selected Integration Account

* Subscription ⓘ
Resource group ⓘ
Resource type ⓘ
Resource ⓘ
BT54 > EIPTemplates > IntegrationAccount
Turn on diagnostics to collect the following logs.

- IntegrationAccountTrackingEvents

3. Select status **ON**

Diagnostics settings
Save Discard
⚠ Turning off diagnostics will disable monitoring charts and alerts for your resource.
Status

4. Select **Send to Log Analytics** and configure Log Analytics to emit data

Diagnostics settings
Save Discard
Status

 Archive to a storage account
 Stream to an event hub
 Send to Log Analytics
Log Analytics Configure
LOG
 IntegrationAccountTrackingEvents
OMS Workspaces
eastus
[redacted] East US
[redacted] eastus
[redacted] East US
[redacted] southeastasia
[redacted] australiasoutheast
LogicAppTracking East US
[redacted] australiasoutheast

Enable logging with Monitor

1. Select **Monitor** and click **Diagnostics logs**

The screenshot shows the Azure portal's 'Monitor - Diagnostics logs' page. The left sidebar is open, displaying various service categories. The 'Monitor' category is selected and highlighted with a red box. Within the main content area, the 'Diagnostics logs' option under the 'EXPLORE' section is also highlighted with a red box. Other visible sections include 'Activity log', 'Metrics', 'Log search', 'Alerts', 'Application Insights', and 'Management solutions'. A search bar at the top is labeled 'Search (Ctrl+ /)'.

2. Select your **Subscription** and **Resource Group, Integration Account** from Resource Type and select your **Integration Account** from Resource drop-down to enable diagnostics. Click **Turn on Diagnostics** to enable diagnostics for the selected Integration Account

The screenshot shows the 'Turn on diagnostics' configuration page. It includes dropdown menus for 'Subscription' (selected), 'Resource group' (EIPTemplates), 'Resource type' (Integration Accounts), and 'Resource' (IntegrationAccount). A red box highlights the 'Turn on diagnostics' button. Below the button, a note states 'To collect the following logs.' followed by a list: 'IntegrationAccountTrackingEvents'.

3. Select status **ON**

The screenshot shows the 'Diagnostics settings' dialog box. It features a warning message: 'Turning off diagnostics will disable monitoring charts and alerts for your resource.' Below the message is a 'Status' section with two buttons: 'On' (highlighted with a red box) and 'Off'.

4. Select **Send to Log Analytics** and configure Log Analytics to emit data

The screenshot shows two overlapping windows from the Azure portal. The left window, titled 'Diagnostics settings', has a dark header with 'Save' and 'Discard' buttons. It contains sections for 'Status' (radio buttons for 'On' and 'Off'), 'Archive to a storage account', 'Stream to an event hub', and 'Send to Log Analytics' (checkbox checked). A red box highlights the 'Send to Log Analytics' section. Below it is a 'Log Analytics' section with a 'Configure' button, also highlighted by a red box. The right window, titled 'OMS Workspaces', lists various workspace names and regions. A red box highlights the 'LogicAppTracking' workspace in the 'australiasoutheast' region.

Extending your solutions

In addition to the **Log Analytics**, you can configure your Integration Account and **Logic Apps** to an Event Hub or Storage Account

The screenshot shows the 'Diagnostics' configuration window for an Integration Account. The header includes 'Save' and 'Discard' buttons. The 'Status' section has 'On' selected. Under 'LOGS', the 'IntegrationAccountTrackingEvents' checkbox is checked and highlighted with a red box. Other sections include 'Service Bus Namespace' and 'Storage Account'. Under 'LOGS', there are retention settings for 'WorkflowRuntime' (30 days) and '1 minute' (5 days). A note at the bottom states: 'The storage account must be in the same region as the resource.' with an information icon.

You can use this telemetry from the Event Hub or Storage into other services like [Azure Stream Analytics](#), and [Power BI](#) to have real-time monitoring of your integration workflows.

Supported Tracking Schema

We are supporting following tracking schema types. All of them has fixed schemas except Custom type.

- [Custom Tracking Schema](#)
- [AS2 Tracking Schema](#)
- [X12 Tracking Schema](#)

Next steps

[Tracking B2B messages in OMS Portal](#)

[Learn more about the Enterprise Integration Pack](#)

AS2 tracking schemas

1/20/2017 • 2 min to read • [Edit on GitHub](#)

You can use these AS2 tracking schemas in your Azure integration account to help you monitor business-to-business (B2B) transactions:

- AS2 message tracking schema
- AS2 MDN tracking schema

AS2 message tracking schema

```
{  
    "agreementProperties": {  
        "senderPartnerName": "",  
        "receiverPartnerName": "",  
        "as2To": "",  
        "as2From": "",  
        "agreementName": ""  
    },  
    "messageProperties": {  
        "direction": "",  
        "messageId": "",  
        "dispositionType": "",  
        "fileName": "",  
        "isMessageFailed": "",  
        "isMessageSigned": "",  
        "isMessageEncrypted": "",  
        "isMessageCompressed": "",  
        "correlationMessageId": "",  
        "incomingHeaders": {  
        },  
        "outgoingHeaders": {  
        },  
        "isNrrEnabled": "",  
        "isMdnExpected": "",  
        "mdnType": ""  
    }  
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	AS2 message sender's partner name. (Optional)
receiverPartnerName	String	AS2 message receiver's partner name. (Optional)
as2To	String	AS2 message receiver's name, from the headers of the AS2 message. (Mandatory)
as2From	String	AS2 message sender's name, from the headers of the AS2 message. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
agreementName	String	Name of the AS2 agreement to which the messages are resolved. (Optional)
direction	String	Direction of the message flow, receive or send. (Mandatory)
messageId	String	AS2 message ID, from the headers of the AS2 message (Optional)
dispositionType	String	Message Disposition Notification (MDN) disposition type value. (Optional)
fileName	String	File name, from the header of the AS2 message. (Optional)
isMessageFailed	Boolean	Whether the AS2 message failed. (Mandatory)
isMessageSigned	Boolean	Whether the AS2 message was signed. (Mandatory)
isMessageEncrypted	Boolean	Whether the AS2 message was encrypted. (Mandatory)
isMessageCompressed	Boolean	Whether the AS2 message was compressed. (Mandatory)
correlationMessageId	String	AS2 message ID, to correlate messages with MDNs. (Optional)
incomingHeaders	Dictionary of JToken	Incoming AS2 message header details. (Optional)
outgoingHeaders	Dictionary of JToken	Outgoing AS2 message header details. (Optional)
isNrrEnabled	Boolean	Use default value if the value is not known. (Mandatory)
isMdnExpected	Boolean	Use default value if the value is not known. (Mandatory)
mdnType	Enum	Allowed values are NotConfigured , Sync , and Async . (Mandatory)

AS2 MDN tracking schema

```

{
    "agreementProperties": {
        "senderPartnerName": "",
        "receiverPartnerName": "",
        "as2To": "",
        "as2From": "",
        "agreementName": "g"
    },
    "messageProperties": {
        "direction": "",
        "messageId": "",
        "originalMessageId": "",
        "dispositionType": "",
        "isMessageFailed": "",
        "isMessageSigned": "",
        "isNrrEnabled": "",
        "statusCode": "",
        "micVerificationStatus": "",
        "correlationMessageId": "",
        "incomingHeaders": {
        },
        "outgoingHeaders": {
        }
    }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	AS2 message sender's partner name. (Optional)
receiverPartnerName	String	AS2 message receiver's partner name. (Optional)
as2To	String	Partner name who receives the AS2 message. (Mandatory)
as2From	String	Partner name who sends the AS2 message. (Mandatory)
agreementName	String	Name of the AS2 agreement to which the messages are resolved. (Optional)
direction	String	Direction of the message flow, receive or send. (Mandatory)
messageld	String	AS2 message ID. (Optional)
originalMessageId	String	AS2 original message ID. (Optional)
dispositionType	String	MDN disposition type value. (Optional)
isMessageFailed	Boolean	Whether the AS2 message failed. (Mandatory)
isMessageSigned	Boolean	Whether the AS2 message was signed. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
isNrrEnabled	Boolean	Use default value if the value is not known. (Mandatory)
statusCode	Enum	Allowed values are Accepted , Rejected , and AcceptedWithErrors . (Mandatory)
micVerificationStatus	Enum	Allowed values are NotApplicable , Succeeded , and Failed . (Mandatory)
correlationMessageId	String	Correlation ID. The original message ID (the message ID of the message for which MDN is configured). (Optional)
incomingHeaders	Dictionary of JToken	Indicates incoming message header details. (Optional)
outgoingHeaders	Dictionary of JToken	Indicates outgoing message header details. (Optional)

Next steps

- Learn more about the [Enterprise Integration Pack](#).
- Learn more about [monitoring B2B messages](#).
- Learn more about [B2B custom tracking schemas](#).
- Learn more about [X12 tracking schemas](#).
- Learn about [tracking B2B messages in the Operations Management Suite portal](#).

X12 tracking schemas

1/20/2017 • 7 min to read • [Edit on GitHub](#)

You can use these X12 tracking schemas in your Azure integration account to help you monitor business-to-business (B2B) transactions:

- X12 transaction set tracking schema
- X12 transaction set acknowledgement tracking schema
- X12 interchange tracking schema
- X12 interchange acknowledgement tracking schema
- X12 functional group tracking schema
- X12 functional group acknowledgement tracking schema

X12 transaction set tracking schema

```
{  
    "agreementProperties": {  
        "senderPartnerName": "",  
        "receiverPartnerName": "",  
        "senderQualifier": "",  
        "senderIdentifier": "",  
        "receiverQualifier": "",  
        "receiverIdentifier": "",  
        "agreementName": ""  
    },  
    "messageProperties": {  
        "direction": "",  
        "interchangeControlNumber": "",  
        "functionalGroupControlNumber": "",  
        "transactionSetControlNumber": "",  
        "CorrelationMessageId": "",  
        "messageType": "",  
        "isMessageFailed": "",  
        "isTechnicalAcknowledgmentExpected": "",  
        "isFunctionalAcknowledgmentExpected": "",  
        "needAk2LoopForValidMessages": "",  
        "segmentsCount": ""  
    }  
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number. (Optional)
functionalGroupControlNumber	String	Functional control number. (Optional)
transactionSetControlNumber	String	Transaction set control number. (Optional)
CorrelationMessageId	String	Correlation message ID. A combination of {AgreementName} {GroupControlNumber} {TransactionSetControlNumber}. (Optional)
messageType	String	Transaction set or document type. (Optional)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
isTechnicalAcknowledgmentExpected	Boolean	Whether the technical acknowledgement is configured in the X12 agreement. (Mandatory)
isFunctionalAcknowledgmentExpected	Boolean	Whether the functional acknowledgement is configured in the X12 agreement. (Mandatory)
needAk2LoopForValidMessages	Boolean	Whether the AK2 loop is required for a valid message. (Mandatory)
segmentsCount	Integer	Number of segments in the X12 transaction set. (Optional)

X12 transaction set acknowledgement tracking schema

```

{
    "agreementProperties": {
        "senderPartnerName": "",
        "receiverPartnerName": "",
        "senderQualifier": "",
        "senderIdentifier": "",
        "receiverQualifier": "",
        "receiverIdentifier": "",
        "agreementName": ""
    },
    "messageProperties": {
        "direction": "",
        "interchangeControlNumber": "",
        "functionalGroupControlNumber": "",
        "isaSegment": "",
        "gsSegment": "",
        "respondingfunctionalGroupControlNumber": "",
        "respondingFunctionalGroupId": "",
        "respondingtransactionSetControlNumber": "",
        "respondingTransactionSetId": "",
        "statusCode": "",
        "processingStatus": "",
        "CorrelationMessageId": "",
        "isMessageFailed": "",
        "ak2Segment": "",
        "ak3Segment": "",
        "ak5Segment": ""
    }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number of the functional acknowledgement. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)

PROPERTY	TYPE	DESCRIPTION
functionalGroupControlNumber	String	Functional group control number of the functional acknowledgement. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)
isaSegment	String	ISA segment of the message. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)
gsSegment	String	GS segment of the message. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)
respondingfunctionalGroupControlNumber	String	Responding interchange control number. (Optional)
respondingFunctionalGroupId	String	Responding functional group ID, which maps to AK101 in the acknowledgement. (Optional)
respondingtransactionSetControlNumber	String	Responding transaction set control number. (Optional)
respondingTransactionSetId	String	Responding transaction set ID, which maps to AK201 in the acknowledgement. (Optional)
statusCode	Boolean	Transaction set acknowledgement status code. (Mandatory)
segmentsCount	Enum	Acknowledgement status code. Allowed values are Accepted , Rejected , and AcceptedWithErrors . (Mandatory)
processingStatus	Enum	Processing status of the acknowledgement. Allowed values are Received , Generated , and Sent . (Mandatory)
CorrelationMessageId	String	Correlation message ID. A combination of {AgreementName} {GroupControlNumber} {TransactionSetControlNumber}. (Optional)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
ak2Segment	String	Acknowledgement for a transaction set within the received functional group. (Optional)
ak3Segment	String	Reports errors in a data segment. (Optional)
ak5Segment	String	Reports whether the transaction set identified in the AK2 segment is accepted or rejected, and why. (Optional)

X12 interchange tracking schema

```
{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "isaSegment": "",
    "isTechnicalAcknowledgmentExpected": "",
    "isMessageFailed": "",
    "isa09": "",
    "isa10": "",
    "isa11": "",
    "isa12": "",
    "isa14": "",
    "isa15": "",
    "isa16": ""
  }
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number. (Optional)
isaSegment	String	Message ISA segment. (Optional)
isTechnicalAcknowledgmentExpected	Boolean	Whether the technical acknowledgement is configured in the X12 agreement. (Mandatory)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
isa09	String	X12 document interchange date. (Optional)
isa10	String	X12 document interchange time. (Optional)
isa11	String	X12 interchange control standards identifier. (Optional)
isa12	String	X12 interchange control version number. (Optional)
isa14	String	X12 acknowledgement is requested. (Optional)
isa15	String	Indicator for test or production. (Optional)
isa16	String	Element separator. (Optional)

X12 interchange acknowledgement tracking schema

```

{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "isaSegment": "",
    "respondingInterchangeControlNumber": "",
    "isMessageFailed": "",
    "statusCode": "",
    "processingStatus": "",
    "ta102": "",
    "ta103": "",
    "ta105": ""
  }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number of the technical acknowledgement that's received from partners. (Optional)
isaSegment	String	ISA segment for the technical acknowledgement that's received from partners. (Optional)
respondingInterchangeControlNumber	String	Interchange control number for the technical acknowledgement that's received from partners. (Optional)

PROPERTY	TYPE	DESCRIPTION
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
statusCode	Enum	Interchange acknowledgement status code. Allowed values are Accepted , Rejected , and AcceptedWithErrors . (Mandatory)
processingStatus	Enum	Acknowledgement status. Allowed values are Received , Generated , and Sent . (Mandatory)
ta102	String	Interchange date. (Optional)
ta103	String	Interchange time. (Optional)
ta105	String	Interchange note code. (Optional)

X12 functional group tracking schema

```
{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "functionalGroupControlNumber": "",
    "gsSegment": "",
    "isTechnicalAcknowledgmentExpected": "",
    "isFunctionalAcknowledgmentExpected": "",
    "isMessageFailed": "",
    "gs01": "",
    "gs02": "",
    "gs03": "",
    "gs04": "",
    "gs05": "",
    "gs07": "",
    "gs08": ""
  }
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)

PROPERTY	TYPE	DESCRIPTION
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number. (Optional)
functionalGroupControlNumber	String	Functional control number. (Optional)
gsSegment	String	Message GS segment. (Optional)
isTechnicalAcknowledgmentExpected	Boolean	Whether the technical acknowledgement is configured in the X12 agreement. (Mandatory)
isFunctionalAcknowledgmentExpected	Boolean	Whether the functional acknowledgement is configured in the X12 agreement. (Mandatory)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
gs01	String	Functional identifier code. (Optional)
gs02	String	Application sender's code. (Optional)
gs03	String	Application receiver's code. (Optional)
gs04	String	Functional group date. (Optional)
gs05	String	Functional group time. (Optional)
gs07	String	Responsible agency code. (Optional)
gs08	String	Version/release/industry identifier code. (Optional)

X12 functional group acknowledgement tracking schema

```

{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "functionalGroupControlNumber": "",
    "isaSegment": "",
    "gsSegment": "",
    "respondingFunctionalGroupControlNumber": "",
    "respondingFunctionalGroupId": "",
    "isMessageFailed": "",
    "statusCode": "",
    "processingStatus": "",
    "ak903": "",
    "ak904": "",
    "ak9Segment": ""
  }
}
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number, which populates for the send side when a technical acknowledgement is received from partners. (Optional)
functionalGroupControlNumber	String	Functional group control number of the technical acknowledgement, which populates for the send side when a technical acknowledgement is received from partners. (Optional)

PROPERTY	TYPE	DESCRIPTION
isaSegment	String	Same as interchange control number, but populated only in specific cases. (Optional)
gsSegment	String	Same as functional group control number, but populated only in specific cases. (Optional)
respondingfunctionalGroupControlNumber	String	Control number of the original functional group. (Optional)
respondingFunctionalGroupId	String	Maps to AK101 in the acknowledgement functional group ID. (Optional)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
statusCode	Enum	Acknowledgement status code. Allowed values are Accepted , Rejected , and AcceptedWithErrors . (Mandatory)
processingStatus	Enum	Processing status of the acknowledgement. Allowed values are Received , Generated , and Sent . (Mandatory)
ak903	String	Number of transaction sets received. (Optional)
ak904	String	Number of transaction sets accepted in the identified functional group. (Optional)
ak9Segment	String	Whether the functional group identified in the AK1 segment is accepted or rejected, and why. (Optional)

Next steps

- Learn more about [monitoring B2B messages](#).
- Learn more about [AS2 tracking schemas](#).
- Learn more about [B2B custom tracking schemas](#).
- Learn about [tracking B2B messages in the Operations Management Suite portal](#).
- Learn more about the [Enterprise Integration Pack](#).

Custom tracking schemas

1/20/2017 • 1 min to read • [Edit on GitHub](#)

You can use a custom tracking schema in your Azure integration account to help you monitor business-to-business (B2B) transactions.

Custom tracking schema

```
{
  "sourceType": "",
  "source": {

    "workflow": {
      "systemId": ""
    },
    "runInstance": {
      "runId": ""
    },
    "operation": {
      "operationName": "",
      "repeatItemScopeName": "",
      "repeatItemIndex": "",
      "trackingId": "",
      "correlationId": "",
      "clientRequestId": ""
    }
  },
  "events": [
  {
    "eventLevel": "",
    "eventTime": "",
    "recordType": "",
    "record": {
    }
  }
  ]
}
```

PROPERTY	TYPE	DESCRIPTION
sourceType		Type of the run source. Allowed values are Microsoft.Logic/workflows and custom . (Mandatory)
Source		If the source type is Microsoft.Logic/workflows , the source information needs to follow this schema. If the source type is custom , the schema is a JToken. (Mandatory)
systemId	String	Logic app system ID. (Mandatory)
runId	String	Logic app run ID. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
operationName	String	Name of the operation (for example, action or trigger). (Mandatory)
repeatItemScopeName	String	Repeat item name if the action is inside a <code>foreach</code> / <code>until</code> loop. (Mandatory)
repeatItemIndex	Integer	Whether the action is inside a <code>foreach</code> / <code>until</code> loop. Indicates the repeated item index. (Mandatory)
trackingId	String	Tracking ID, to correlate the messages. (Optional)
correlationId	String	Correlation ID, to correlate the messages. (Optional)
clientRequestId	String	Client can populate it to correlate messages. (Optional)
eventLevel		Level of the event. (Mandatory)
eventTime		Time of the event, in UTC format YYYY-MM-DDTHH:MM:SS.00000Z. (Mandatory)
recordType		Type of the track record. Allowed value is custom . (Mandatory)
record		Custom record type. Allowed format is JToken. (Mandatory)

B2B protocol tracking schemas

For information about B2B protocol tracking schemas, see:

- [AS2 tracking schemas](#)
- [X12 tracking schemas](#)

Next steps

- Learn more about [monitoring B2B messages](#).
- Learn about [tracking B2B messages in the Operations Management Suite portal](#).
- Learn more about the [Enterprise Integration Pack](#).

Tracking B2B messages in OMS portal

1/20/2017 • 1 min to read • [Edit on GitHub](#)

B2B communication involves message exchanges between two running business processes or applications.

Tracking B2B messages in OMS portal provides a rich, web-based tracking capabilities that allow to view whether messages processed correctly. You can track

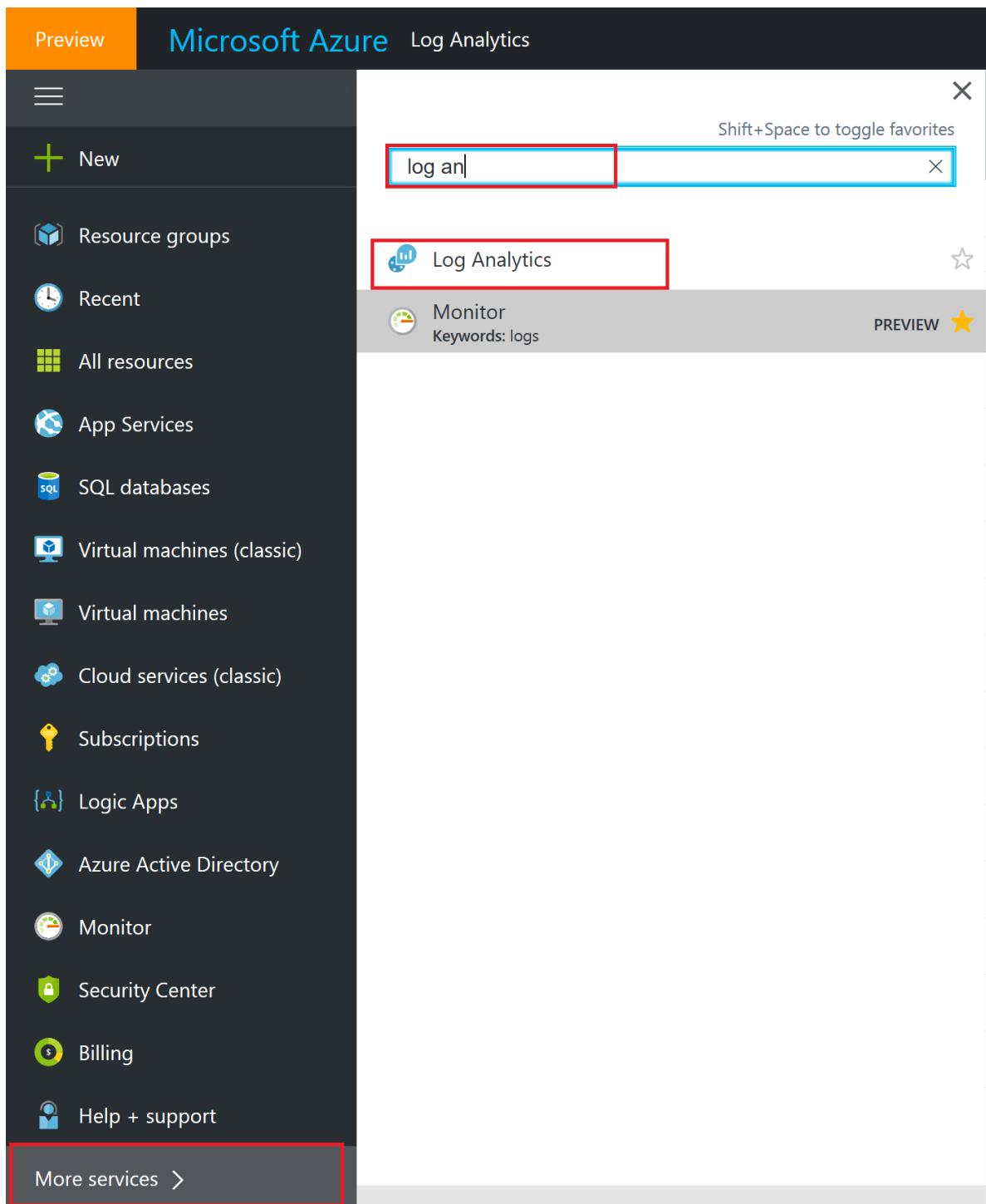
- Count and status of messages
- Acknowledgments status
- Correlating messages with acknowledgments
- Detailed error description for failures
- Search capabilities

Prerequisites

- An Azure account; you can create a [free account](#)
- An Integration Account; you can create an [Integration Account](#) and enable logging; you can find steps [here](#)
- A Logic App; you can create a [Logic App](#) and enable logging; you can find steps [here](#)

Adding Logic Apps B2B solution to OMS portal

1. Select **More Services** in portal, search **log analytics** and select **log analytics**



2. Select your **Log Analytics**

Log Analytics
Microsoft

+ Add Columns Refresh

Subscriptions: 1 of 6 selected

Filter items... BTS4

NAME	RESOURCE GROUP	LOCATION
[REDACTED]	EIPTemplates	East US
[REDACTED]	padmarg	East US
[REDACTED]	eipTestOMS	East US
[REDACTED]	mms-eus	East US
[REDACTED]	jonsigniterg	Southeast Asia
[REDACTED]	DONOTDELETE-TrackingResourceGro...	Australia Southeast
[REDACTED]	mms-eus	East US
[REDACTED]	EIPTemplates	Australia Southeast
padomsportal	EIPTemplates	East US

3. Select **OMS Portal**, opens OMS portal home page

padomsportal

Log Analytics

OMS Portal Delete

Search (Ctrl+ /)

Essentials

Resource group: **eiptemplates**

Status: Active

Location: East US

Subscription name: [REDACTED]

Subscription ID: [REDACTED]

Workspace Name: padomsportal

Workspace Id: [REDACTED]

Pricing tier: Free

Management services: Operations logs

Management

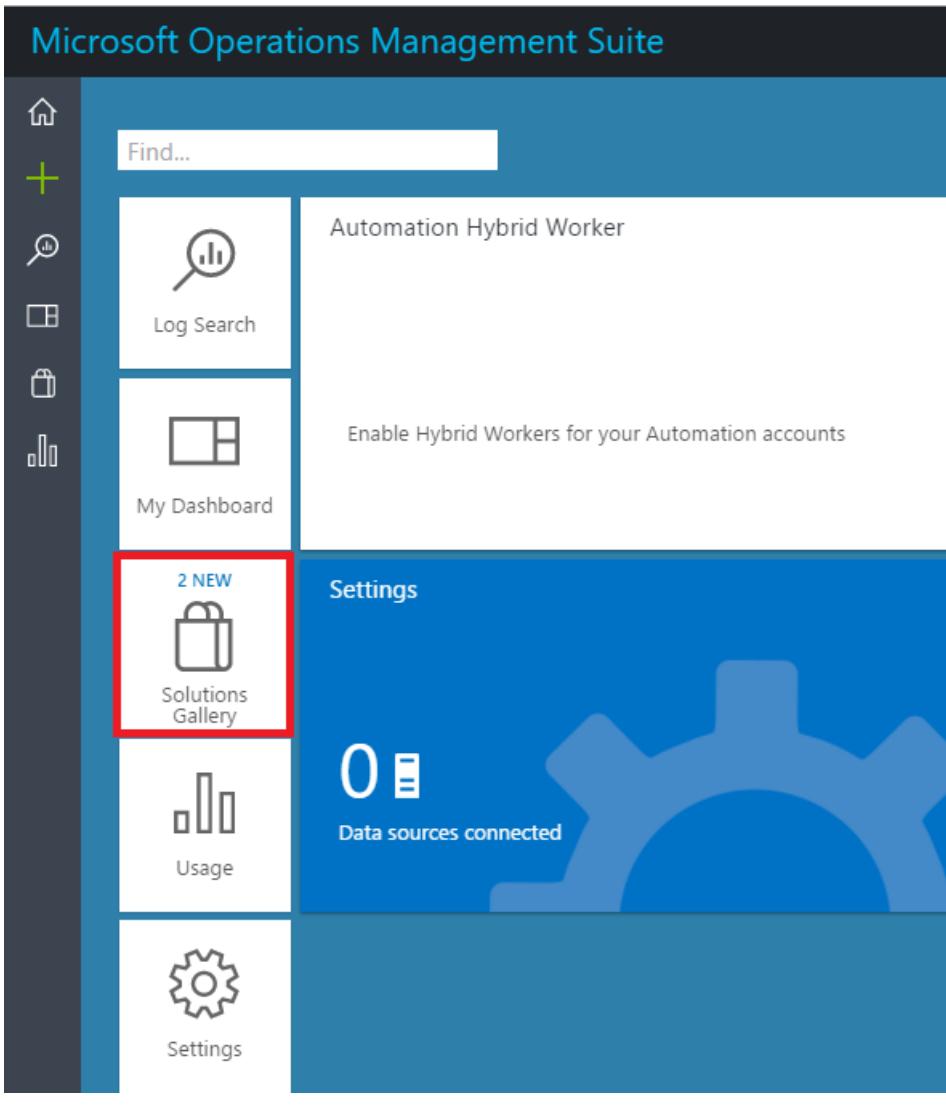
Overview Log Search OMS Portal

Pricing tier

Free PADOMSPORTAL

500 MB daily limit Data retention 7 days

4. Select **Solutions Gallery**



5. Select **Logic Apps B2B**

This screenshot shows the 'Solutions Gallery' page within the OMS. At the top, there's a header with the Microsoft Operations Management Suite logo and various navigation icons. The main content area is titled 'Solutions Gallery' and shows 'Solution offers' and 'All solutions'. Under 'Solution offers', there are two categories: 'Insight & Analytics' (Microsoft) and 'Automation & Control' (Microsoft). Under 'All solutions', there are three cards: 'Logic Apps B2B' (highlighted with a red box), 'AD Assessment', and 'Alert Management'. Each card provides a brief description and status (Available).

6. Click **Add** to add **Logic Apps B2B Messages** to home page

Microsoft Operations Management Suite

Solutions Gallery > Details

Logic Apps B2B NEW

Available

Add

Description

The Logic Apps B2B Solution allow you to monitor and manage your business to business (B2B) Logic Apps across AS2, X12 and EDIFACT protocols. This solutions lets you easily view and react to message exchanges with trading partners and identify root causes.

With rich, out-of-the-box views you can get insights into key processing including:

- At a glance summary of all processing by status
- Partner-specific view to understand which partner message exchanges are having issues
- View by protocol to easily determine where problems are occurring and easily isolate and identify
- Message-centric view to visually understand the processing flows by Logic App including key tracking data items to react quickly to problems if they arise
- Ability to search through all tracking data and find any piece of information including custom tracking data
- Pre-Requisite: this solution works with Azure Logic Apps and the Enterprise Integration Account to collect B2B data and correlate events.

Overview > Logic Apps B2B

AS2 X12

Messages by status

	TOTAL
FAILED	2.3K
PENDING	5.2K
Successful	9.5K

SEND TOTAL

	COUNT
Successful Messages	68K
Failed Messages	50K
Pending Messages	50K

RECEIVE COUNT

	COUNT
Successful Messages	0
Failed Messages	50K
Pending Messages	81K

7. Browse home page to view Logic Apps B2B Messages

Microsoft Operations Management Suite

Find...

Log Search

Automation Hybrid Worker

Enable Hybrid Workers for your Automation accounts

3 NEW

My Dashboard

Solutions Gallery

Usage

Logic Apps B2B Messages

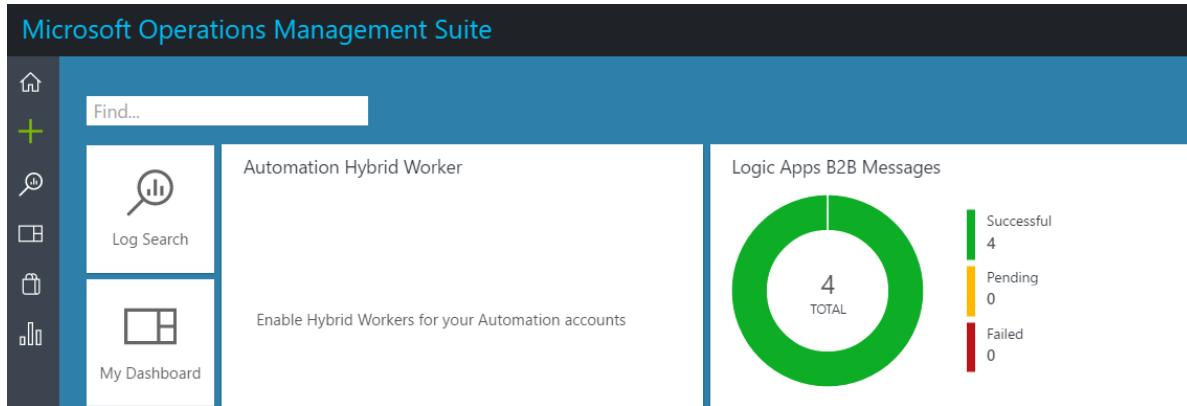
0 TOTAL

Successful: 0
Pending: 0
Failed: 0

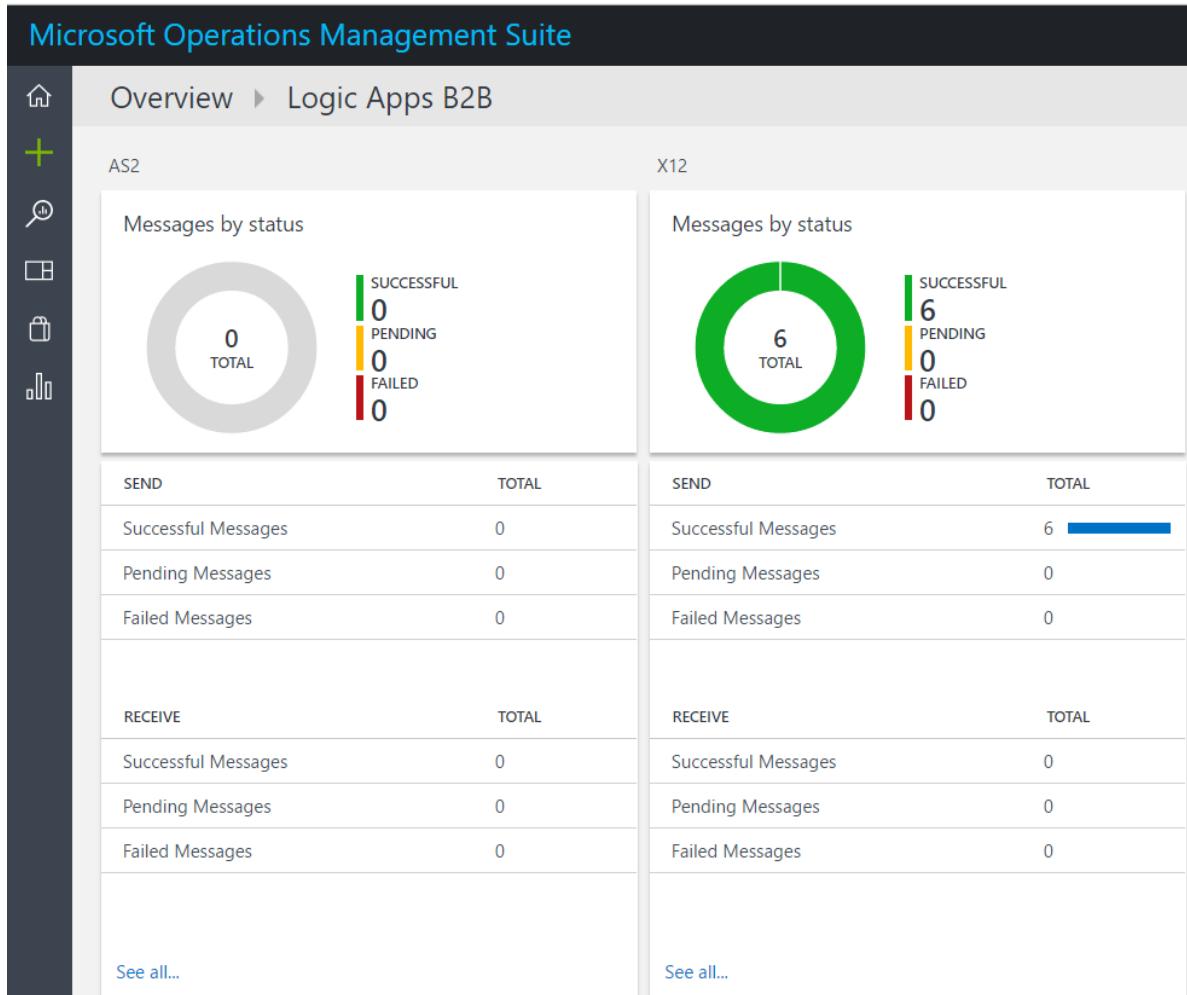
Settings

Tracking data in OMS portal

- Post message process; the home page updates with message count



- Selecting **Logic Apps B2B Messages** on home page leads to AS2 and X12 message status. The data is based on last one day.



- Selecting AS2 or X12 messages by status takes you to the message list

The screenshot shows the "Message List" for AS2 messages. At the top, it says "ALL AS2 MESSAGES (7)". Below is a table with columns: SENDER, RECEIVER, LOGIC APP, STATUS, DIRECTION, CORRELATION ID, MESSAGE ID, and TIMESTAMP. The table lists 7 rows of message details, including sender, receiver, logic app name, status (e.g., "X" for failed, "✓" for successful), direction (Send or Receive), correlation ID, message ID, timestamp, and a small icon.

SENDER	RECEIVER	LOGIC APP	STATUS	DIRECTION	CORRELATION ID	MESSAGE ID	TIMESTAMP	Actions
Fabrikam	Contoso	as2x12transform	✗	Receive	31c464f0-730f-4f62-98cc...	@guid()	11/14/2016, 8:55:45 AM	
Contoso	Fabrikam	AS2Encode	✓	Send	20137b02-9024-4b7c-8e3f...	<RD0003FFA8D11F_B31...	11/14/2016, 8:50:43 AM	
Fabrikam	Contoso	as2x12transform	✗	Receive	0025edee-69b4-4562-850...	@guid()	11/14/2016, 12:50:56 AM	
Fabrikam	Contoso	as2x12transform	✗	Receive	48d5aefa-d7f5-401d-8677...	67C022FA-A0D1-4083-9...	11/14/2016, 12:49:42 AM	
Contoso	Fabrikam	AS2Encode	✓	Send	d88916f4-bec4-4459-93fd...	<RD0003FFA8DC5_5DC...	11/13/2016, 2:08:47 PM	
Contoso	Fabrikam	AS2Encode	✓	Send	7e0eff5c-406e-48bd-9d80...	<RD0003FFA8D11F_3EC...	11/13/2016, 10:47:38 AM	
Contoso	Fabrikam	AS2Encode	✓	Send	c7fd7ed2-a81b-4ef5-84ae...	<RD0003FFA8DC5_175...	11/13/2016, 10:47:38 AM	

SENDER	RECEIVER	LOGIC APP	STATUS	DIRECTION	CORREL...	MSG TYPE	ICN	TSCN	TIMESTAMP
Contoso	Fabrikam	AS2Encode	Running	Send	bb071e...	855	000000026	0026	11/14/2016, 8:50:42 AM
Contoso	Fabrikam	AS2Encode	Running	Send	e909cf...	855	000000025	0025	11/13/2016, 2:08:46 PM
Contoso	Fabrikam	AS2Encode	Running	Send	f719b2f...	855	000000024	0024	11/13/2016, 10:47:38 AM
Contoso	Fabrikam	AS2Encode	Running	Send	166ca0...	855	000000023	0023	11/13/2016, 10:47:38 AM

4. Select a row in AS2 or X12 message list takes you to log search. Log search lists all the actions that have same Run ID

TIMEGENERATED	OPERATIONNAME	CATEGORY	SOURCETYPE_S	SOURC...	SOURC...	SOURC...	SOURC...	SOURCE_R...	SOURCE_RUNINSTAN...	SOURCE_OPERATIONNAME_S
11/13/2016 10:...	Microsoft.Logic/...	IntegrationAcc...	Microsoft.Logic/w...	/loc...	EIP...	AS...	0...	085872...	0858723057421...	Encode_to_A\$2_message
11/13/2016 10:...	Microsoft.Logic/...	IntegrationAcc...	Microsoft.Logic/w...	/loc...	EIP...	AS...	0...	085872...	0858723057421...	Encode_to_X12_message_by_agree...
11/13/2016 10:...	Microsoft.Logic/...	IntegrationAcc...	Microsoft.Logic/w...	/loc...	EIP...	AS...	0...	085872...	0858723057421...	Encode_to_X12_message_by_agree...
11/13/2016 10:...	Microsoft.Logic/...	IntegrationAcc...	Microsoft.Logic/w...	/loc...	EIP...	AS...	0...	085872...	0858723057421...	Encode_to_X12_message_by_agree...

Queries in OMS portal

On the search page, you can create a query, and then when you search, you can filter the results by using facet controls.

How to create a query

1. In the log search, write a query and select **Save**. Here are the steps to write a query

11/16/2016 10:10:40.983 AM AzureDiagnostics
... TimeGenerated : 11/16/2016 10:10:40.983 AM
... event_record_agreementProperties_senderQualifier_s : ZZ
... event_record_agreementProperties_senderIdentifier_s : 87654321
... event_record_agreementProperties_receiverQualifier_s : ZZ
... event_record_agreementProperties_receiverIdentifier_s : 12345678
... event_record_messageProperties_interchangeControlNumber_s : 000000027

2. **Save Search** opens. Give a **name**, **category**, and click **Save**

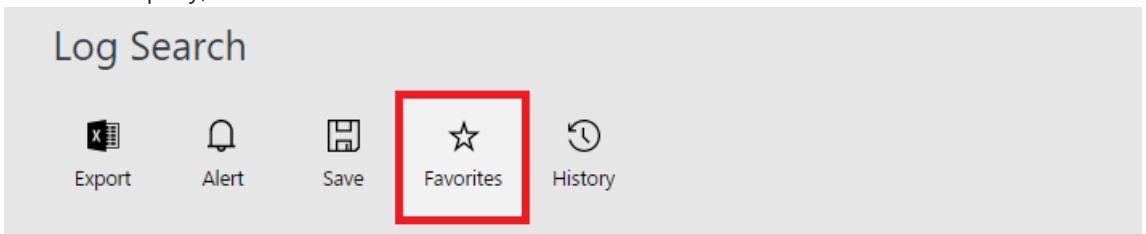
Save Search

Name

Category

Save this query as a computer group:

3. To view the query, select **favorites**



- Saved Searches
- Find...
- SHOW ALL...
- ◀ Logic Apps B2B (Preview)
- Failed AS2 Messages by Receive Partner
 - Failed AS2 Messages by Send Partner
 - Failed AS2 Messages by Workflow
 - Failed X12 Messages by Receive Partner
 - Failed X12 Messages by Send Partner
 - Failed X12 Messages by Workflow
-
- Show Less...

How to use a saved query

1. In the log search, select **favorites** to view saved queries. Selecting one of the gives query results

The screenshot shows the Azure Log Search interface. At the top, there are buttons for Export, Alert, Save, Favorites (which is highlighted with a red box), and History. Below the header is a search bar with the query: Type="AzureDiagnostics" source_runinstance_rundId_s=08587222870686547921761230681 event_r. To the right of the search bar is a magnifying glass icon. The main area displays 3 Results in List view. The results are as follows:

TimeGenerated	Value
... TimeGenerated	: 11/16/2016 10:10:40.983 AM
... event_record_agreementProperties_senderQualifier_s	: ZZ
... event_record_agreementProperties_senderIdentifier_s	: 87654321
... event_record_agreementProperties_receiverQualifier_s	: ZZ
... event_record_agreementProperties_receiverIdentifier_s	: 12345678
... event_record_messageProperties_interchangeControlNumber_s	: 000000027
... event_record_messageProperties_functionalGroupControlNumber_s	: 27
... event_record_messageProperties_gs01_s	: PR
... event_record_messageProperties_gs02_s	: BTS-SENDER
... event_record_messageProperties_gs03_s	: RECEIVE-APP
... event_record_messageProperties_gs04_s	: 20161116

To the right of the results, there is a sidebar titled "Saved Searches" with a "Find..." input field. A list of saved searches is shown, with the first item expanded:

- Logic Apps B2B (Preview)
 - Failed AS2 Messages by Receive Partner
 - Failed AS2 Messages by Send Partner
 - Failed AS2 Messages by Workflow
 - Failed X12 Messages by Receive Partner
 - Failed X12 Messages by Send Partner
 - Failed X12 Messages by Workflow

A red box highlights the "Search by control number" input field at the bottom of the sidebar.

Next steps

[Custom Tracking Schema](#)

[AS2 Tracking Schema](#)

[X12 Tracking Schema](#)

[Learn more about the Enterprise Integration Pack](#)

Track B2B messages in the Operations Management Suite portal by using a query

1/20/2017 • 1 min to read • [Edit on GitHub](#)

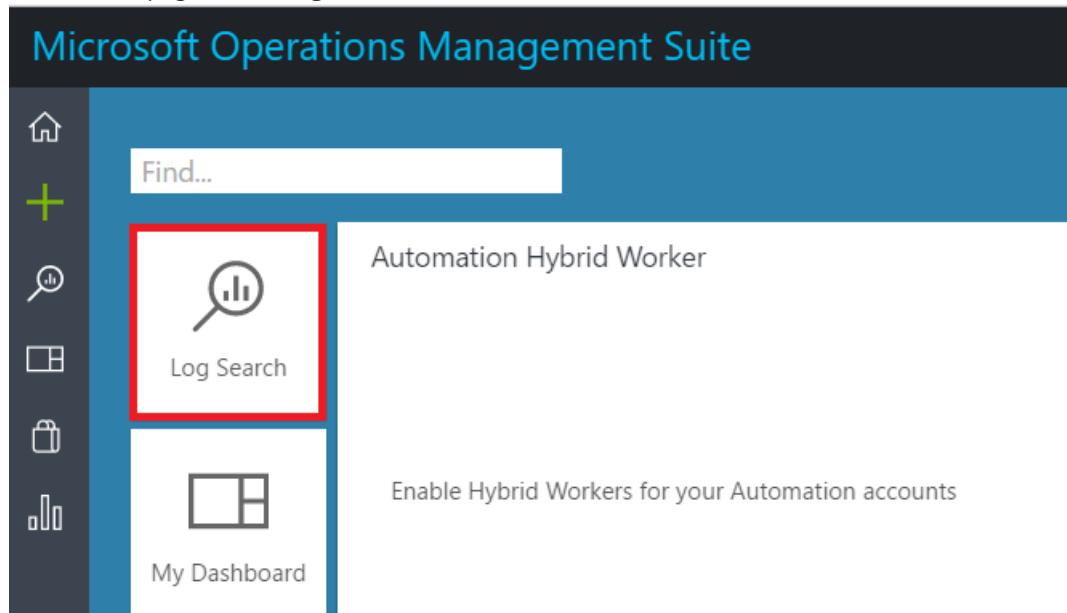
To track business-to-business (B2B) messages in the Operations Management Suite portal, you can create a query that filters data for a specific interchange control number.

Prerequisites

For debugging and for more detailed diagnostics information, turn on diagnostics in your [integration account](#) for your [logic apps](#) that have X12 connectors. Then, do the steps to [publish diagnostic data](#) to the Operations Management Suite portal.

To create a query to search for a specific interchange control number

1. On the start page, select **Log Search**.



2. In the search box, type **Type="AzureDiagnostics"**. To filter data, select **Add**.

Log Search

Export Alert Save Favorites History

Data based on last 7 days

Type="AzureDiagnostics"

1 bar = 6hrs

12:05:35 PM Nov 9, 2016 12:05:35 AM Nov 12, 2016 12:05:35 PM Nov 14, 2016

TYPE (1)

- AzureDiagnostics 213

EVENT_RECORD_MESSAGEPROPERTIES_ORIGINALMESSAGEII (1)

+Add

213 Results List Table

11/16/2016 10:10:56.997 AM | AzureDiagnostics

- ... TimeGenerated : 11/16/2016 10:10:56.997 AM
- ... workflowId_s :
- /SUBSCRIPTIONS/F34B22A3-2202-4FB1-B040-
- 1332BD928C84/RESOURCEGROUPS/EIPTEMPLATES/PROVIDERS/MICROSOFT.
- ... status_s : Succeeded
- ... resource_resourceGroupName_s : EIPTemplates
- ... resource_workflowName_s : AS2Encode
- ... resource_runId_s : 08587222870686547921761230681
- ... resource_location_s : westus
- ... correlation_clientTrackingId_s : 08587222870686547921761230681
- ... resource_subscriptionId_g : f34b22a3-2202-4fb1-b040-1332bd928c84
- ... resource_workflowId_g : f8bef9a0-de60-49df-8c0d-59ee4864f303

3. Type **interchange**, select **event_record_messageProperties_interchangeControlNumber_s**, and then select **Add**.

Log Search

Export Alert Save Favorites History

Type (1)

- AzureDiagnostics 4

EVENT_RECORD_MESSAGEPROPERTIES_INTERCHANGECONT (1)

- 000000026 3

EVENT_RECORD_MESSAGEPROPERTIES_ORIGINALMESSAGEII (0)

EVENT_RECORD_MESSAGEPROPERTIES_DISPOSITIONTYPE_S (0)

+Add

Add Filters

interchange

AzureDiagnostics (1)

event_record_messageProp
event_record_messageProperties_interchangeControlNumber_s

4. Select the control number that you want to filter data for, and then select **Apply**.

Log Search

Export Alert Save Favorites History

TYPE (1)

	AzureDiagnostics	213
EVENT_RECORD_MESSAGEPROPERTIES_INTERCHANGECONT		
(7)	x	
<input type="checkbox"/>	3	
<input type="checkbox"/> 000000022	3	
<input checked="" type="checkbox"/> 000000023	3	
<input type="checkbox"/> 000000024	3	
<input type="checkbox"/> 000000025	3	
<input type="checkbox"/> 000000026	3	
<input type="checkbox"/> 000000027	3	

[–] Less

Apply **Cancel**

5. Find the query that you created to filter data for the selected control number.

Type="AzureDiagnostics" event_record_messageProperties_interchangeControlNumber_s=000000023

3 Results [List](#) [Table](#)

11/13/2016 10:47:38.660 AM | AzureDiagnostics

... TimeGenerated : 11/13/2016 10:47:38.660 AM
... event_record_agreementProperties_senderQualifier_s : ZZ
... event_record_agreementProperties_senderIdentifier_s : 87654321
... event_record_agreementProperties_receiverQualifier_s : ZZ
... event_record_agreementProperties_receiverIdentifier_s : 12345678
... event_record_messageProperties_interchangeControlNumber_s : 000000023

6. Type a name for the query, and then save it.

Type="AzureDiagnostics" event_record_messageProperties_interchangeControlNumber_s=0000000

3 Results List Table

11/13/2016 10:47:38.660 AM | AzureDiagnostics

- ... TimeGenerated : 11/13/2016 10:47:38.660 AM
- ... event_record_agreementProperties_senderQualifier_s : ZZ
- ... event_record_agreementProperties_senderIdentifier_s : 87654321
- ... event_record_agreementProperties_receiverQualifier_s : ZZ
- ... event_record_agreementProperties_receiverIdentifier_s : 12345678
- ... event_record_messageProperties_interchangeControlNumber_s : 000000023
- ... event_record_messageProperties_isa09_s : 161113

Save Search

Name

Category

Save this query as a computer group:

Yes No

Save **Cancel**

7. To view the query, and to use it in future searches, select **Favorites**.

Type="AzureDiagnostics" event_record_messageProperties_interchangeControlNumber_s=000000023

3 Results List Table

11/13/2016 10:47:38.660 AM | AzureDiagnostics

- ... TimeGenerated : 11/13/2016 10:47:38.660 AM
- ... event_record_agreementProperties_senderQualifier_s : ZZ
- ... event_record_agreementProperties_senderIdentifier_s : 87654321
- ... event_record_agreementProperties_receiverQualifier_s : ZZ
- ... event_record_agreementProperties_receiverIdentifier_s : 12345678
- ... event_record_messageProperties_interchangeControlNumber_s : 000000023
- ... event_record_messageProperties_isa09_s : 161113
- ... event_record_messageProperties_isa10_s : 1847
- ... event_record_messageProperties_isa11_s : U
- ... event_record_messageProperties_isa12_s : 00401
- ... event_record_messageProperties_isa14_s : 0
- ... event_record_messageProperties_isa15_s : T
- ... event_record_messageProperties_isa16_s : :
- ... event_record_messageProperties_isTechnicalAcknowledgmentExpected_b : false
- ... OperationName : Microsoft Logic/IntegrationAccounts/trackingEvents

Saved Searches

- Find...
- All Events with level: Warning
- All IIS Log Entries
- All Syslog Records grouped by Facility
- All Syslog Records grouped by ProcessName
- Show All...
- Logic Apps B2B (Preview)
- Failed AS2 Messages by Receive Partner
 - Failed AS2 Messages by Send Partner
 - Failed AS2 Messages by Workflow
 - Failed X12 Messages by Receive Partner
 - Failed X12 Messages by Send Partner
 - Failed X12 Messages by Workflow
- Search by control number**

8. You can update the query to search for a new interchange control number.

Type="AzureDiagnostics" event_record_messageProperties_interchangeControlNumber_s=000000027

3 Results List Table

11/13/2016 10:47:38.660 AM | AzureDiagnostics

- ... TimeGenerated : 11/13/2016 10:47:38.660 AM
- ... event_record_agreementProperties_senderQualifier_s : ZZ
- ... event_record_agreementProperties_senderIdentifier_s : 87654321
- ... event_record_agreementProperties_receiverQualifier_s : ZZ
- ... event_record_agreementProperties_receiverIdentifier_s : 12345678
- ... event_record_messageProperties_interchangeControlNumber_s : 000000023
- ... event_record_messageProperties_isa09_s : 161113
- ... event_record_messageProperties_isa10_s : 1847
- ... event_record_messageProperties_isa11_s : U
- ... event_record_messageProperties_isa12_s : 00401
- ... event_record_messageProperties_isa14_s : 0
- ... event_record_messageProperties_isa15_s : T
- ... event_record_messageProperties_isa16_s : :
- ... event_record_messageProperties_isTechnicalAcknowledgmentExpected_b : false
- ... OperationName : Microsoft Logic/IntegrationAccounts/trackingEvents

Saved Searches

- Find...
- All Events with level: Warning
- All IIS Log Entries
- All Syslog Records grouped by Facility
- All Syslog Records grouped by ProcessName
- Show All...
- Logic Apps B2B (Preview)
- Failed AS2 Messages by Receive Partner
 - Failed AS2 Messages by Send Partner
 - Failed AS2 Messages by Workflow
 - Failed X12 Messages by Receive Partner
 - Failed X12 Messages by Send Partner
 - Failed X12 Messages by Workflow
- Search by control number**

Next steps

- Learn more about [custom tracking schemas](#).
- Learn more about [AS2 tracking schemas](#).

- Learn more about [X12 tracking schemas](#).
- Learn more about the [Enterprise Integration Pack](#).

Overview of integration accounts

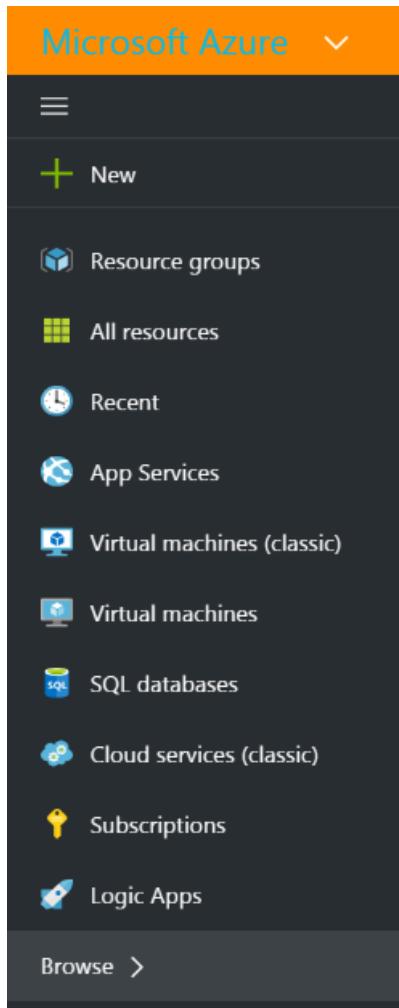
1/20/2017 • 2 min to read • [Edit on GitHub](#)

What is an integration account?

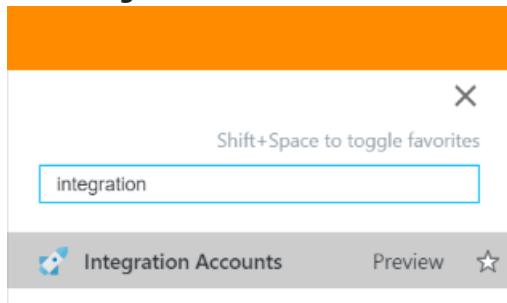
An integration account is an Azure account that allows Enterprise Integration apps to manage artifacts including schemas, maps, certificates, partners and agreements. Any integration app you create will need to use an integration account in order to access a schema, map or certificate, for example.

Create an integration account

1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list



3. Select **Add** button from the menu at the top of the page

The screenshot shows the 'Integration Accounts' blade in the Azure portal. At the top, there's an orange header bar with the title 'Integration Accounts'. Below it is a dark header with the title 'Integration Accounts' and the text 'Microsoft - PREVIEW'. There are three buttons: 'Add' (with a plus sign), 'Columns' (with a grid icon), and 'Refresh' (with a circular arrow). A status message 'Subscriptions: All 3 selected' is displayed above a search bar labeled 'Filter items...'. The main area is titled 'NAME' and contains a single item: 'DeonhelIntegrationAccount' with a small blue icon to its left.

4. Enter the **Name**, select the **Subscription** you want to use, either create a new **Resource group** or select an existing resource group, select a **Location** where your integration account will be hosted, select a **Pricing tier**, then select the **Create** button.

At this point the integration account will be provisioned in the location you selected. This should complete within 1 minute.

Integration Account
PREVIEW

* Name
MyNewIntegrationAccount ✓

* Subscription
ICBCS9

* Resource group ⓘ
Create new Use existing
MyNewRG ✓

* Location
Brazil South

* Pricing tier
Free >

Pin to dashboard

Create

5. Refresh the page. You will see your new integration account listed. Congratulations!

Integration Accounts
Microsoft - PREVIEW

Add Columns Refresh

Subscriptions: All 3 selected

Filter items... All subscriptions

NAME	RESOURCE GROUP	LOCATION
DeonheIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South

How to link an integration account to a Logic app

In order for your Logic apps to access to maps, schemas, agreements and other artifacts that are located in your integration account, you must first link the integration account to your Logic app.

Here are the steps to link an integration account to a Logic app

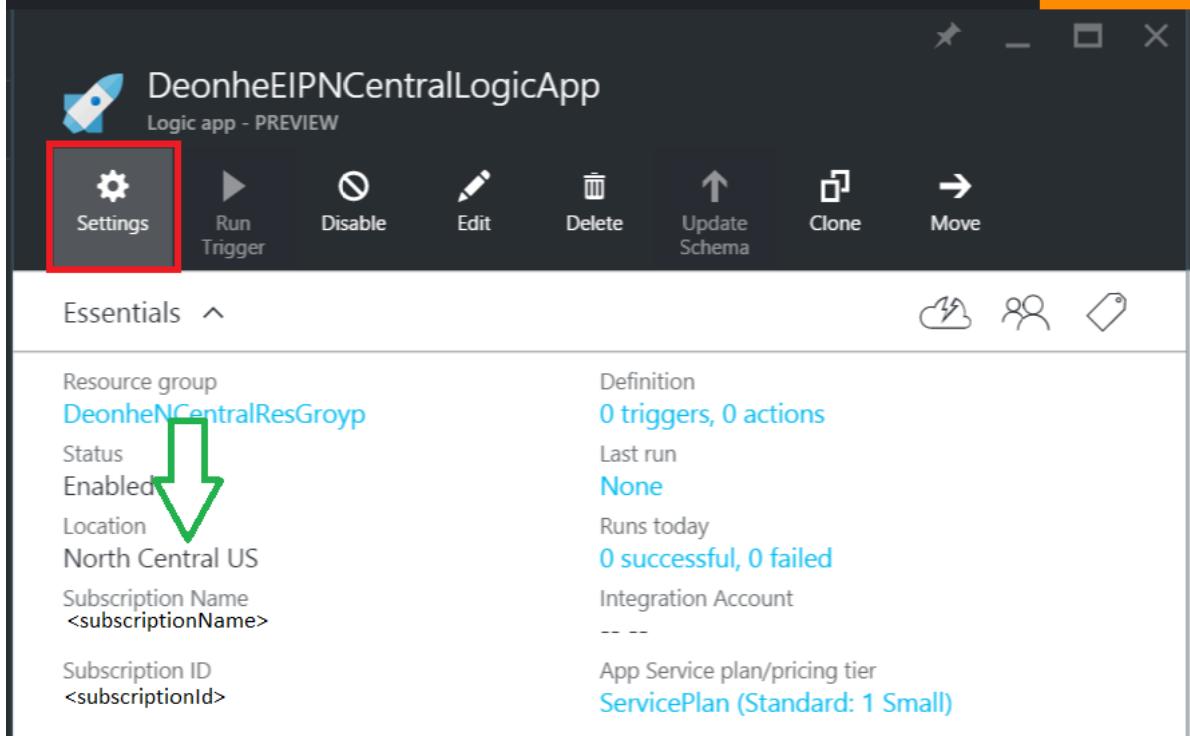
Prerequisites

- An integration account
- A Logic app

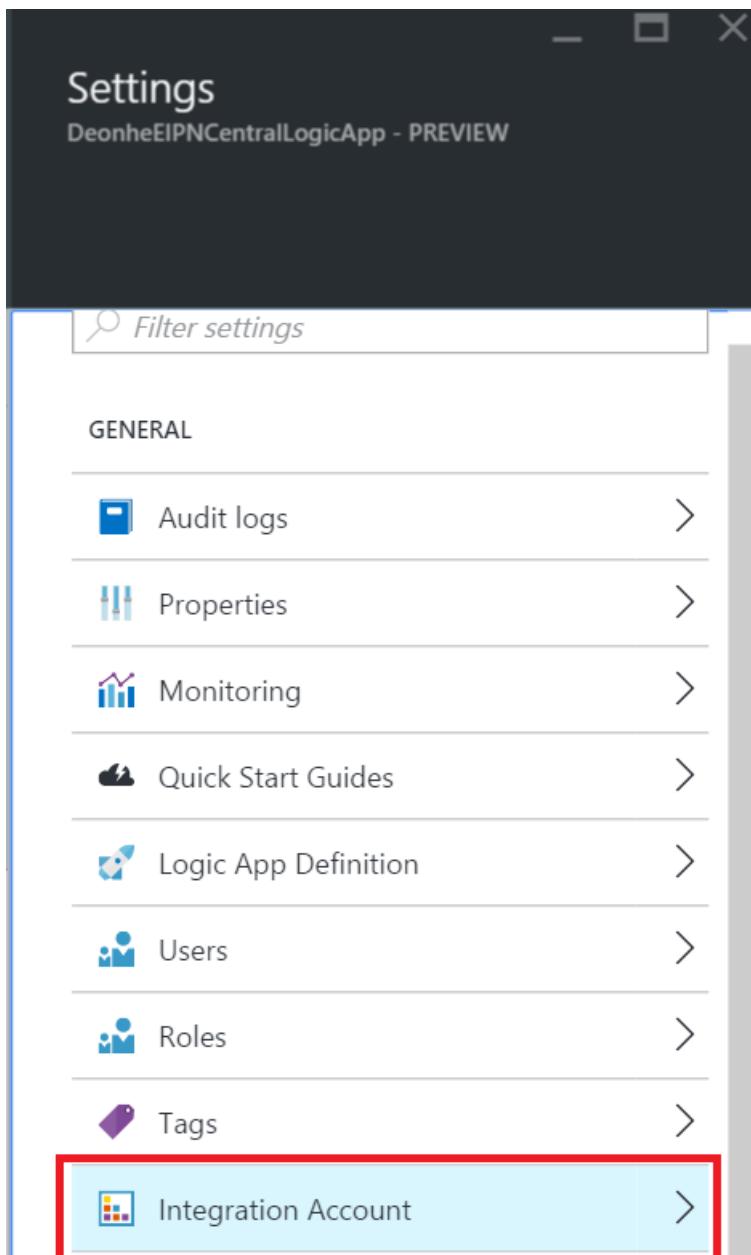
NOTE

Be sure your integration account and Logic app are in the **same Azure location** before you begin

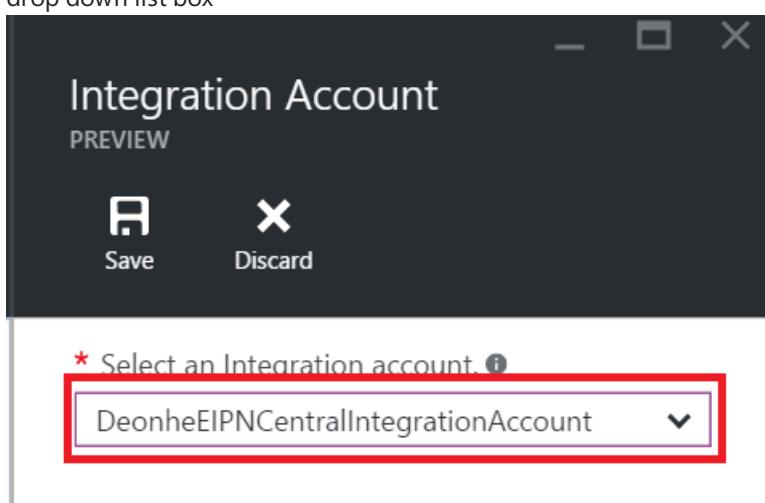
1. Select **Settings** link from the menu of your Logic app



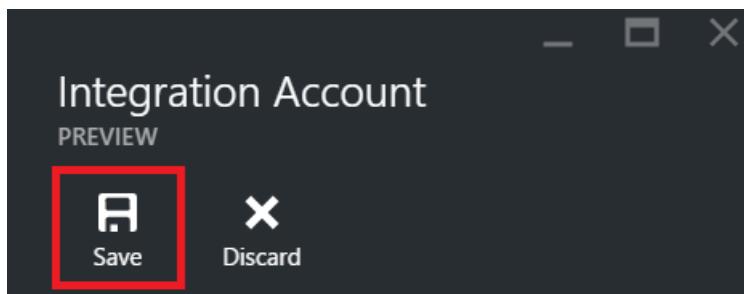
2. Select the **Integration Account** item from the Settings blade



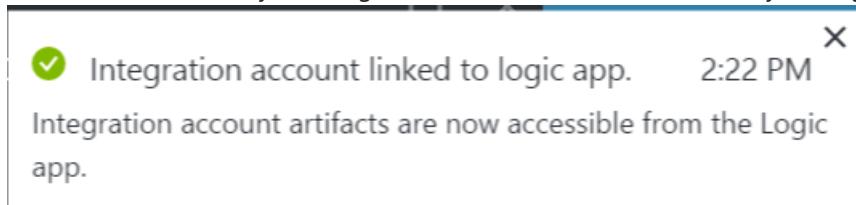
3. Select the integration account you wish to link to your Logic app from the **Select an Integration account** drop down list box



4. Save your work



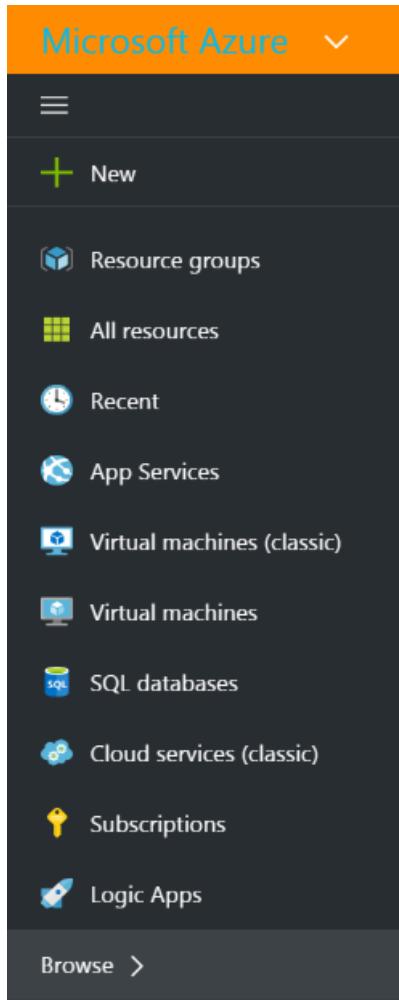
5. You will see a notification that indicates that your integration account has been linked to your Logic app and that all artifacts in your integration account are now available to your Logic app.



Now that your integration account is linked to your Logic app, you can go to your Logic app and use B2B connectors such as the XML Validation, Flat file encode/decode or Transform to create apps with B2B features.

How to delete an integration account?

1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list

The screenshot shows the Azure portal interface for 'Integration Accounts'. At the top, there's a search bar with the text 'integration' and a message 'Shift+Space to toggle favorites'. Below the search bar is a navigation bar with icons for 'Integration Accounts' (selected), 'Preview', and a star icon. The main content area is titled 'Integration Accounts' and includes a Microsoft - PREVIEW badge. It features three buttons: 'Add', 'Columns', and 'Refresh'. A message 'Subscriptions: All 3 selected' is displayed above a table. The table has columns for 'NAME', 'RESOURCE GROUP', and 'LOCATION'. It lists two accounts: 'DeonheIntegrationAccount' (Resource Group: DeonheResGroup, Location: Brazil South) and 'MyNewIntegrationAccount' (Resource Group: MyNewRG, Location: Brazil South). There are also 'Filter items...' and 'All subscriptions' buttons.

NAME	RESOURCE GROUP	LOCATION
DeonheIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South

3. Select the **integration account** that you wish to delete

4. Select the **Delete** link that's located on the menu

MyNewIntegrationAccount
Integration Account - PREVIEW

Move Delete Step 4

Essentials ▾

Resource group: MyNewRG
Location: brazilsouth
Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

Name: MyNewIntegrationAccount
Subscription: ICBCS9

All settings →

Components

Add tiles (+)

Schemas	Maps	Certificates
0	0	0

Partners	Agreements
1	0

Add a section (+)

5. Confirm your choice

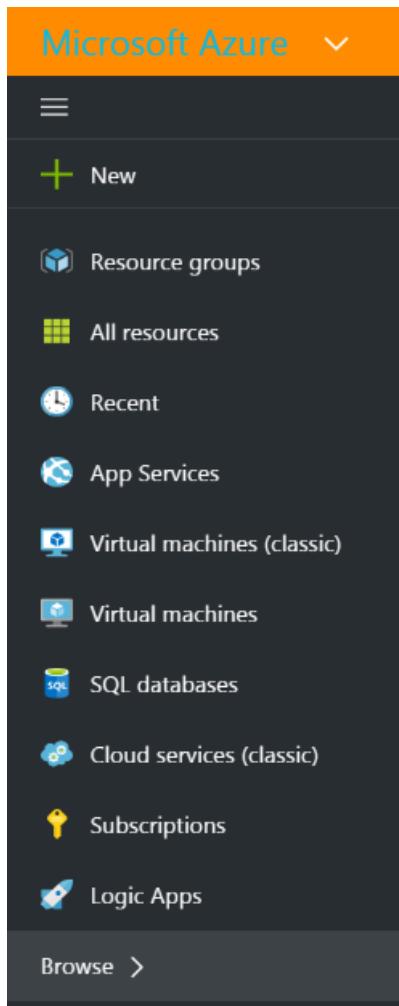
How to move an integration account?

You can easily move an integration account to a new subscription and a new resource group. Follow these steps if you need to move your integration account:

IMPORTANT

You will need to update all scripts to use the new resource IDs after you move an integration account.

1. Select **Browse**



2. Enter **integration** in the filter search box and select **Integration Accounts** from the results list

The screenshot shows a search interface with an orange header bar and a white body. A search bar contains the text "integration". Below it, a list item "Integration Accounts" is highlighted with a blue border. Other items like "Preview" and a star icon are also visible.

3. Select the **integration account** that you wish to delete

The screenshot shows the "Integration Accounts" list view. At the top, there are buttons for "Add", "Columns", and "Refresh". Below that, a message says "Subscriptions: All 3 selected". There are two filter input fields: "Filter items..." and "All subscriptions". The main table has columns: NAME, RESOURCE GROUP, and LOCATION. Two rows are listed:

NAME	RESOURCE GROUP	LOCATION
DeonheIntegrationAccount	DeonheResGroup	Brazil South
MyNewIntegrationAccount	MyNewRG	Brazil South

4. Select the **Move** link that's located on the menu

MyNewIntegrationAccount
Integration Account - PREVIEW

Step 4 →

Move Delete

Resource group: MyNewRG
Name: MyNewIntegrationAccount
Location: brazilsouth
Subscription: ICBCS9
Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

All settings →

Components

Add tiles (+)

Schemas	Maps	Certificates
0	0	0

Partners	Agreements
1	0

Add a section (+)

5. Confirm your choice

Next Steps

- Learn more about the Enterprise Integration Pack
- Learn more about agreements

Logic Apps pricing model

1/25/2017 • 3 min to read • [Edit on GitHub](#)

Azure Logic Apps allows you to scale and execute integration workflows in the cloud. Following are details on the billing and pricing plans for Logic Apps.

Consumption pricing

Newly created Logic Apps use a consumption plan. With the Logic Apps consumption pricing model, you only pay for what you use. Logic Apps are not throttled when using a consumption plan. All actions executed in a run of a logic app instance are metered.

What are action executions?

Every step in a logic app definition is an action, which includes triggers, control flow steps like conditions, scopes, for each loops, do until loops, calls to connectors and calls to native actions. Triggers are special actions that are designed to instantiate a new instance of a logic app when a particular event occurs. There are several different behaviors for triggers, which could affect how the logic app is metered.

- **Polling trigger** – this trigger continually polls an endpoint until it receives a message that satisfies the criteria for creating an instance of a logic app. The polling interval can be configured in the trigger in the Logic Apps designer. Each polling request, even if it doesn't create an instance of a logic app, counts as an action execution.
- **Webhook trigger** – this trigger waits for a client to send it a request on a particular endpoint. Each request sent to the webhook endpoint counts as an action execution. The Request and the HTTP Webhook trigger are both webhook triggers.
- **Recurrence trigger** – this trigger creates an instance of the logic app based on the recurrence interval configured in the trigger. For example, a recurrence trigger can be configured to run every three days or even every minute. Trigger executions can be seen in the Logic Apps resource blade in the Trigger History part. All actions that were executed, whether they were successful or failed are metered as an action execution. Actions that were skipped due to a condition not being met or actions that didn't execute because the logic app terminated before completion are not counted as action executions.

Actions executed within loops are counted per iteration of the loop. For example, a single action in a for each loop iterating through a list of 10 items will be counted as the count of items in the list (10) multiplied by the number of actions in the loop (1) plus one for the initiation of the loop, which, in this example, would be $(10 * 1) + 1 = 11$ action executions. Logic Apps that are disabled cannot have new instances instantiated and therefore while they are disabled are not charged. Be mindful that after disabling a logic app it may take a little time for the instances to quiesce before being completely disabled.

Integration Account Usage

Included in consumption based usage is an [integration account](#) for exploration, development and testing purposes allowing you to use the [B2B/EDI](#) and [XML processing](#) features of Logic Apps at no additional cost. You are able to create a maximum of one account per region and store up to 10 Agreements and 25 maps. Schemas, certificates and partners have no limits and you can upload as many as you need.

In addition to the inclusion of integration accounts with consumption, you can also create standard integration accounts without these limits and with our standard Logic Apps SLA. See [Azure pricing](#) for further details.

App Service plans

Logic apps previously created referencing an App Service Plan will continue to behave as before. Depending on the

plan chosen, are throttled after the prescribed daily executions are exceeded but are billed using the action execution meter. EA customers that have an App Service Plan in their subscription, which does not have to be explicitly associated with the Logic App, get the included quantities benefit. For example, if you have a Standard App Service Plan in your EA subscription and a Logic App in the same subscription then you aren't charged for 10,000 action executions per day (see following table).

App Service Plans and their daily allowed action executions:

	FREE/SHARED/BASIC	STANDARD	PREMIUM
Action executions per day	200	10,000	50,000

Convert from App Service Plan pricing to Consumption

To change a Logic App that has an App Service Plan associated with it to a consumption model, remove the reference to the App Service Plan in the Logic App definition. This change can be done with a call to a PowerShell cmdlet: `Set-AzureRmLogicApp -ResourceGroupName 'rgname' -Name 'wfname' -UseConsumptionModel -Force`

Pricing

For pricing details, see [Logic Apps Pricing](#).

Next steps

- [An overview of Logic Apps](#)
- [Create your first logic app](#)

New schema version 2016-06-01

1/20/2017 • 4 min to read • [Edit on GitHub](#)

The new schema and API version for Logic apps has a number of improvements which improve the reliability and ease-of-use of Logic apps. There are 3 key differences:

1. Addition of scopes, which are actions that contain a collection of actions.
2. Conditions and loops are first-class actions
3. Execution ordering more verbose via `runAfter` property (which replaces `dependsOn`)

For information on upgrading your logic apps from the 2015-08-01-preview schema to the 2016-06-01 schema, [check out the upgrade section below](#).

1. Scopes

One of the biggest changes in this schema is the addition of scopes and the ability to nest actions within each other. This is helpful when grouping a set of actions together, or when needing to nest actions within each other (for example a condition can contain another condition). More details on scope syntax can be found [here](#), but a simple scope example can be found below:

```
{
  "actions": {
    "My_Scope": {
      "type": "scope",
      "actions": {
        "Http": {
          "inputs": {
            "method": "GET",
            "uri": "http://www.bing.com"
          },
          "runAfter": {},
          "type": "Http"
        }
      }
    }
  }
}
```

2. Conditions and loops changes

In the previous versions of the schema, conditions and loops were parameters associated to a single action. This limitation has been lifted in this schema and now conditions and loops show up as a type of action. More information can be found [in this article](#), and a simple example of a condition action is shown below:

```
{
  "If_trigger_is_foo": {
    "type": "If",
    "expression": "@equals(triggerBody(), 'foo')",
    "runAfter": {},
    "actions": {
      "Http_2": {
        "inputs": {
          "method": "GET",
          "uri": "http://www.bing.com"
        },
        "runAfter": {},
        "type": "Http"
      }
    },
    "else": {
      {
        "if_trigger_is_bar": "..."
      }
    }
  }
}
```

3. RunAfter Property

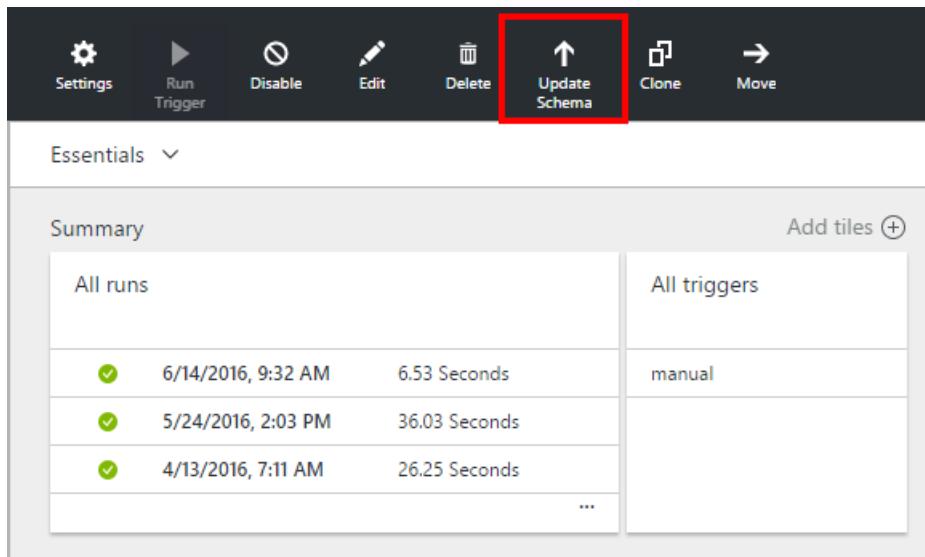
The new `runAfter` property is replacing `dependsOn` to help allow more precision in run ordering. `dependsOn` was synonymous with "the action ran and was successful," however many times you need to execute an action if the previous action is successful, failed, or skipped. `runAfter` allows for that flexibility. It is an object that specifies all of the action names it will run after, and defines an array of status' that are acceptable to trigger from. For example if you wanted to run after step A was succeeded and step B was succeeded or failed, you would construct the following `runAfter` property:

```
{
  "...",
  "runAfter": {
    "A": ["Succeeded"],
    "B": ["Succeeded", "Failed"]
  }
}
```

Upgrading to 2016-06-01 schema

Upgrading to the new 2016-06-01 schema only takes a few steps. Details on the changes from the schema can be found [in this article](#). The upgrade process includes running the upgrade script, saving as a new logic app, and potentially overwriting old logic app if needed.

1. Open your current logic app.
2. Click the **Update Schema** button in the toolbar



The upgraded definition will be returned. You could copy and paste this into a resource definition if you need, but we **strongly recommend** you use the **Save As** button to ensure all connection references are valid in the upgraded logic app.

3. Click the **Save As** button in the toolbar of the upgrade blade.
4. Fill out the name and logic app status and click **Create** to deploy your upgrade logic app.
5. Verify your upgraded logic app is working as expected.

NOTE

If you are using a manual or request trigger, the callback URL will have changed in your new logic app. Use the new URL to verify it works end-to-end, and you can clone over your existing logic app to preserve previous URLs.

6. *Optional* Use the **Clone** button in the toolbar (adjacent to the **Update Schema** icon in the picture above) to overwrite your previous logic app with the new schema version. This is necessary only if you wish to keep the same resource ID or request trigger URL of your logic app.

Upgrade tool notes

Condition mapping

The tool will make a best effort to group the true and false branch actions together in a scope in the upgraded definition. Specifically the designer pattern of `@equals(actions('a').status, 'Skipped')` should show up as an `else` action. However if the tool detects patterns it does not recognize it will potentially create separate conditions for both the true and the false branch. Actions can be re-mapped post upgrade if needed.

ForEach with Condition

The previous pattern of a foreach loop with a condition per item can be replicated in the new schema with the filter action. This should occur automatically on upgrade. The condition becomes a filter action before the foreach loop (to return only an array of items that match the condition), and that array is passed into the foreach action. You can view an example of this [in this article](#)

Resource tags

Resource tags will be removed on upgrade and you will need to set them again for the upgraded workflow.

Other changes

Manual trigger renamed to Request trigger

The type `manual` has been deprecated and renamed to `request` with the kind of `http`. This is more consistent with the type of pattern the trigger is used to build.

New 'filter' action

If you are working with a large array and need to filter it down to a smaller set of items, you can use the new 'filter' type. It accepts an array and a condition and will evaluate the condition for each item and return an array of items that meet the condition.

ForEach and until action restrictions

The foreach and until loop are restricted to a single action.

TrackedProperties on Actions

Actions can now have an additional property (sibling to `runAfter` and `type`) called `trackedProperties`. It is an object that specifies certain action inputs or outputs to be included in the Azure Diagnostic telemetry that is emitted as part of a workflow. For example:

```
{  
    "Http": {  
        "inputs": {  
            "method": "GET",  
            "uri": "http://www.bing.com"  
        },  
        "runAfter": {},  
        "type": "Http",  
        "trackedProperties": {  
            "responseCode": "@action().outputs.statusCode",  
            "uri": "@action().inputs.uri"  
        }  
    }  
}
```

Next Steps

- [Use the logic app workflow definition](#)
- [Create a logic app deployment template](#)

New schema version 2015-08-01-preview

1/20/2017 • 9 min to read • [Edit on GitHub](#)

The new schema and API version for Logic apps has a number of improvements which improve the reliability and ease-of-use of Logic apps. There are 4 key differences:

1. The **APIApp** action type has been updated to a new **APIConnection** action type.
2. **Repeat** has been renamed to **ForEach**.
3. The **HTTP Listener** API app is no longer required.
4. Calling child workflows uses a new schema.

1. Moving to API connections

The biggest change is that you no longer need to deploy API apps into your Azure Subscription to use API's. There are 2 ways you can use APIs:

- Managed API's
- Your custom Web API's

Each of these is handled slightly differently because their management and hosting models are different. One advantage of this model is you're no longer constrained to resources that are deployed in your Resource Group.

Managed APIs

There are a number of API's that are managed by Microsoft on your behalf, such as Office 365, Salesforce, Twitter, FTP etc.... Some of these managed API's can be used as-is, such as Bing Translate, while others require configuration. This configuration is called a *connection*.

For example, when you use Office 365, you need to create a connection that contains your Office 365 sign-in token. This token will be securely stored and refreshed so that your Logic app can always call the Office 365 API. Alternatively, if you want to connect to your SQL or FTP server, you need to create a connection that has the connection string.

Inside of the definition these actions are called `APIConnection`. Here is an example of a connection that calls Office 365 to send an email:

```
{
  "actions": {
    "Send_Email": {
      "type": "ApiConnection",
      "inputs": {
        "host": {
          "api": {
            "runtimeUrl": "https://msmanaged-na.azure-apim.net/apim/office365"
          },
          "connection": {
            "name": "@parameters('$connections')['shared_office365']['connectionId']"
          }
        },
        "method": "post",
        "body": {
          "Subject": "Reminder",
          "Body": "Don't forget!",
          "To": "me@contoso.com"
        },
        "path": "/Mail"
      }
    }
  }
}
```

The portion of the inputs that is unique to API connections is the `host` object. This contains two parts: `api` and `connection`.

The `api` has the runtime URL of where that managed API is hosted. You can see all of the available managed APIs for you by calling

```
GET https://management.azure.com/subscriptions/{subid}/providers/Microsoft.Web/managedApis/?api-version=2015-08-01-preview
```

When you use an API, it may or may not have any **connection parameters** defined. If it doesn't then no **connection** is required. If it does, then you will have to create a connection. When you create that connection it'll have the name you choose, and then you reference that in the `connection` object inside the `host` object. To create a connection in a resource group, call:

```
PUT
https://management.azure.com/subscriptions/{subid}/resourceGroups/{rgname}/providers/Microsoft.Web/connections/{name}?api-version=2015-08-01-preview
```

With the following body:

```
{
  "properties": {
    "api": {
      "id": "/subscriptions/{subid}/providers/Microsoft.Web/managedApis/azureblob"
    },
    "parameterValues" : {
      "accountName" : "{The name of the storage account -- the set of parameters is different for each API}"
    }
  },
  "location" : "{Logic app's location}"
}
```

Deploying managed APIs in an Azure Resource manager template

You can create a full application in an ARM template as long as it doesn't require interactive sign-in. If it requires sign-in, you can set everything up with the ARM template, but will still have to visit the portal to authorize the

connections.

```
"resources": [{"  
    "apiVersion": "2015-08-01-preview",  
    "name": "azureblob",  
    "type": "Microsoft.Web/connections",  
    "location": "[resourceGroup().location]",  
    "properties": {  
        "api": {  
            "id": "  
[concat(subscription().id,'/providers/Microsoft.Web/locations/westus/managedApis/azureblob')]"  
        },  
        "parameterValues": {  
            "accountName": "[parameters('storageAccountName')]",  
            "accessKey": "[parameters('storageAccountKey')]"  
        }  
    }, {  
        "type": "Microsoft.Logic/workflows",  
        "apiVersion": "2015-08-01-preview",  
        "name": "[parameters('logicAppName')]",  
        "location": "[resourceGroup().location]",  
        "dependsOn": [  
            "[resourceId('Microsoft.Web/connections', 'azureblob')]"  
        ],  
        "properties": {  
            "sku": {  
                "name": "[parameters('sku')]",  
                "plan": {  
                    "id": "[concat(resourceGroup().id,  
'/providers/Microsoft.Web/serverfarms/',parameters('svcPlanName'))]"  
                }  
            },  
            "definition": {  
                "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2015-08-01-preview/workflowdefinition.json#",  
                "actions": {  
                    "Create_file": {  
                        "type": "apiconnection",  
                        "inputs": {  
                            "host": {  
                                "api": {  
                                    "runtimeUrl": "https://logic-apis-westus.azure-apim.net/apim/azureblob"  
                                },  
                                "connection": {  
                                    "name": "@parameters('$connections')['azureblob']['connectionId']"  
                                }  
                            },  
                            "method": "post",  
                            "queries": {  
                                "folderPath": "[concat('/',parameters('containerName'))]",  
                                "name": "helloworld.txt"  
                            },  
                            "body": "@decodeDataUri('data:,Hello+world!')",  
                            "path": "/datasets/default/files"  
                        },  
                        "conditions": []  
                    }  
                },  
                "contentVersion": "1.0.0.0",  
                "outputs": {},  
                "parameters": {  
                    "$connections": {  
                        "defaultValue": {},  
                        "type": "Object"  
                    }  
                },  
                "triggers": {}  
            }  
        }  
    }  
}
```

```

        "recurrence": {
            "type": "Recurrence",
            "recurrence": {
                "frequency": "Day",
                "interval": 1
            }
        }
    },
    "parameters": {
        "$connections": {
            "value": {
                "azureblob": {
                    "connectionId": "[concat(resourceGroup().id,'/providers/Microsoft.Web/connections/azureblob')]",
                    "connectionName": "azureblob",
                    "id": "[concat(subscription().id,'/providers/Microsoft.Web/locations/westus/managedApis/azureblob')]"
                }
            }
        }
    }
}
]

```

You can see in this example that the connections are just normal resources that live in your resource group. They reference the managedAPIs available to you in your subscription.

Your custom Web APIs

If you use your own API's (specifically, not Microsoft-managed ones), then you should use the built-in **HTTP** action to call them. In order to have an ideal experience, you should expose a swagger endpoint for your API. This will enable the Logic app designer to render the inputs and outputs for your API. Without a swagger, the designer will only be able to show the inputs and outputs as opaque JSON objects.

Here is an example showing the new `metadata.apiDefinitionUrl` property:

```

{
    "actions": {
        "mycustomAPI": {
            "type": "http",
            "metadata" : {
                "apiDefinitionUrl" : "https://mysite.azurewebsites.net/api/apidef/"
            },
            "inputs": {
                "uri": "https://mysite.azurewebsites.net/api/getsomeodata",
                "method" : "GET"
            }
        }
    }
}

```

If you host your Web API on **App Service** then it will automatically show up in the list of actions available in the designer. If not, you'll have to paste in the URL directly. The swagger endpoint must be unauthenticated in order to be usable inside of the Logic apps designer (although you may secure the API itself with whatever methods are supported in the Swagger).

Using your already deployed API apps with 2015-08-01-preview

If you previously deployed an API app, you can call it via the **HTTP** action.

For example, if you use Dropbox to list files, you may have something like this in your **2014-12-01-preview**

schema version definition:

```
{  
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2014-12-01-preview/workflowdefinition.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "/subscriptions/423db32d-...-  
b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token": {  
            "defaultValue": "eyJ0eX...wCn90",  
            "type": "String",  
            "metadata": {  
                "token": {  
                    "name": "/subscriptions/423db32d-...-  
b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token"  
                }  
            }  
        },  
        "actions": {  
            "dropboxconnector": {  
                "type": "ApiApp",  
                "inputs": {  
                    "apiVersion": "2015-01-14",  
                    "host": {  
                        "id": "/subscriptions/423db32d-...-  
b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector",  
                        "gateway": "https://avdemo.azurewebsites.net"  
                    },  
                    "operation": "ListFiles",  
                    "parameters": {  
                        "FolderPath": "/myfolder"  
                    },  
                    "authentication": {  
                        "type": "Raw",  
                        "scheme": "Zumo",  
                        "parameter": "@parameters('/subscriptions/423db32d-...-  
b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token')"  
                    }  
                }  
            }  
        }  
    }  
}
```

You can construct the equivalent HTTP action like below (the parameters section of the Logic app definition remains unchanged):

```
{
    "actions": {
        "dropboxconnector": {
            "type": "Http",
            "metadata" : {
                "apiDefinitionUrl" : "https://avdemo.azurewebsites.net/api/service/apidef/dropboxconnector/?api-version=2015-01-14&format=swagger-2.0-standard"
            },
            "inputs": {
                "uri": "https://avdemo.azurewebsites.net/api/service/invoke/dropboxconnector/ListFiles?api-version=2015-01-14",
                "method" : "POST",
                "body": {
                    "FolderPath": "/myfolder"
                },
                "authentication": {
                    "type": "Raw",
                    "scheme": "Zumo",
                    "parameter": "@parameters('/subscriptions/423db32d-...-b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token')"
                }
            }
        }
    }
}
```

Walking through these properties one-by-one:

ACTION PROPERTY	DESCRIPTION
<code>type</code>	<code>Http</code> instead of <code>APIapp</code>
<code>metadata.apiDefinitionUrl</code>	If you want to use this action in the Logic apps designer, you'll want to include the metadata endpoint. This is constructed from: <code>{api app host.gateway}/api/service/apidef/{last segment of the api app host.id}/?api-version=2015-01-14&format=swagger-2.0-standard</code>
<code>inputs.uri</code>	This is constructed from: <code>{api app host.gateway}/api/service/invoke/{last segment of the api app host.id}/{api app operation}?api-version=2015-01-14</code>
<code>inputs.method</code>	Always <code>POST</code>
<code>inputs.body</code>	Identical to the api app parameters
<code>inputs.authentication</code>	Identical to the api app authentication

This approach should work for all API app actions. However, please keep in mind that these previous API apps are no longer supported, and you should move to one of the two other options above (either a managed API or hosting your custom Web API).

2. Repeat renamed to Foreach

For the previous schema version we received a lot of customer feedback that **Repeat** was confusing and didn't properly capture that it was really a for each loop. As a result, we have renamed it to **Foreach**. For example:

```
{
    "actions": {
        "pingBing": {
            "type": "Http",
            "repeat": "@range(0,2)",
            "inputs": {
                "method": "GET",
                "uri": "https://www.bing.com/search?q=@{repeatItem()}"
            }
        }
    }
}
```

Would now be written as:

```
{
    "actions": {
        "pingBing": {
            "type": "Http",
            "foreach": "@range(0,2)",
            "inputs": {
                "method": "GET",
                "uri": "https://www.bing.com/search?q=@{item()}"
            }
        }
    }
}
```

Previously the function `@repeatItem()` was used to reference the current item being iterated over. This has been simplified to just `@item()`.

Referencing the outputs of the Foreach

To further simplify, the outputs of **Foreach** actions will not be wrapped in an object called **repeatItems**. This means, whereas the outputs of the above repeat were:

```
{
    "repeatItems": [
        {
            "name": "pingBing",
            "inputs": {
                "uri": "https://www.bing.com/search?q=0",
                "method": "GET"
            },
            "outputs": {
                "headers": { },
                "body": "<!DOCTYPE html><html lang=\"en\" xml:lang=\"en\"  

xmlns=\"http://www.w3.org/1999/xhtml\" xmlns:Web=\"http://schemas.live.com/Web/\">...</html>"
            }
            "status": "Succeeded"
        }
    ]
}
```

Now it will be:

```
[
  {
    "name": "pingBing",
    "inputs": {
      "uri": "https://www.bing.com/search?q=0",
      "method": "GET"
    },
    "outputs": {
      "headers": { },
      "body": "<!DOCTYPE html><html lang=\"en\" xml:lang=\"en\" xmlns=\"http://www.w3.org/1999/xhtml\" xmlns:Web=\"http://schemas.live.com/Web/\">...</html>",
      "status": "Succeeded"
    }
  }
]
```

When referencing these outputs, to get to the body of the action you'd have to do:

```
{
  "actions": {
    "secondAction" : {
      "type" : "Http",
      "repeat" : "@outputs('pingBing').repeatItems",
      "inputs" : {
        "method" : "POST",
        "uri" : "http://www.example.com",
        "body" : "@repeatItem().outputs.body"
      }
    }
  }
}
```

Now you can do instead:

```
{
  "actions": {
    "secondAction" : {
      "type" : "Http",
      "foreach" : "@outputs('pingBing')",
      "inputs" : {
        "method" : "POST",
        "uri" : "http://www.example.com",
        "body" : "@item().outputs.body"
      }
    }
  }
}
```

With these changes, the functions `@repeatItem()`, `@repeatBody()` and `@repeatOutputs()` are removed.

3. Native HTTP listener

The HTTP Listener capabilities are now built-in, so you no longer need to deploy an HTTP Listener API app. Read about [the full details for how to make your Logic app endpoint callable here](#).

With these changes, the function `@accessKeys()` is removed and has been replaced with the `@listCallbackURL()` function for the purposes of getting the endpoint (when needed). In addition, you now must define at least one trigger in your Logic app now. If you want to `/run` the workflow, you'll need to have one of a `manual`, `apiConnectionWebhook` or `httpWebhook` triggers.

4. Calling child workflows

Previously, calling child workflows required going to that workflow, getting the access token, and then pasting that in to the definition of the Logic app that you want to call that child. With the new schema version, the Logic apps engine will automatically generate a SAS at runtime for the child workflow, which means that you don't have to paste any secrets into the definition. Here is an example:

```
"mynestedwf" : {
    "type" : "workflow",
    "inputs" : {
        "host" : {
            "id" : "/subscriptions/xxxxyyyyzzz/resourceGroups/rg001/providers/Microsoft.Logic/mywf001",
            "triggerName" : "myendpointtrigger"
        },
        "queries" : {
            "extrafield" : "specialValue"
        },
        "headers" : {
            "x-ms-date" : "@utcnow()",  
"Content-type" : "application/json"
        },
        "body" : {
            "contentFieldOne" : "value100",
            "anotherField" : 10.001
        }
    },
    "conditions" : []
}
```

A second improvement is we will be giving the child workflows full access to the incoming request. That means that you can pass parameters in the *queries* section and in the *headers* object and that you can fully define the entire body.

Finally, there are required changes to the child workflow. Whereas before you could just call a child workflow directly; now, you'll need to define a trigger endpoint in the workflow for the parent to call. Generally, this means you'll add a trigger of type **manual** and then use that in the parent definition. Note that the `host` property specifically has a `triggerName`, because you must always specify which trigger you are invoking.

Other changes

New `queries` property

All action types now support a new input called **queries**. This can be a structured object rather than you having to assemble the string by hand.

`parse()` function renamed

As we will soon be adding more content types, the `parse()` function has been renamed to `json()`.

Coming soon: Enterprise Integration APIs

At this point in time, we do not yet have managed versions of the Enterprise Integration APIs available (such as AS2). These will be coming soon as covered in the [roadmap](#). In the meanwhile, you can use your existing deployed BizTalk APIs via the HTTP action, as covered above in "Using your already deployed API apps."