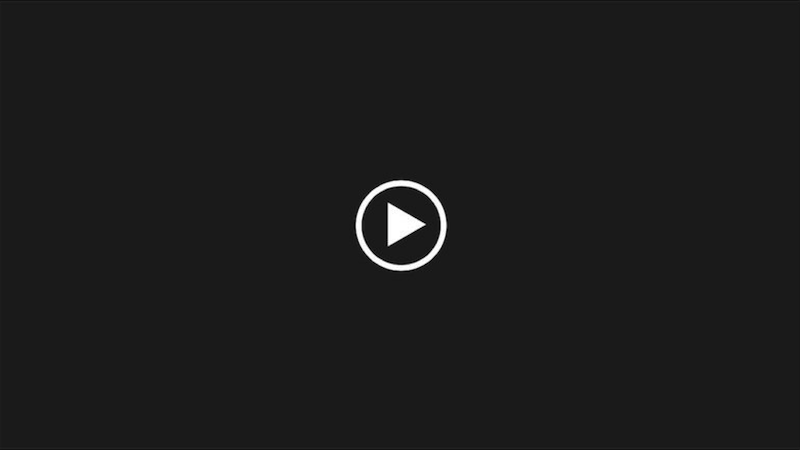
Route and process data automatically using Logic Apps

Last modified on 5/10/2019 by robmcm@microsoft.com

# Introduction

The following video shows what you'll be building in this module.

[](https://channel9.msdn.com/Shows/Docs-Learn/Intro-to-Logic-Apps/player?format=ny)

In this module, you'll create a business workflow using Azure Logic Apps to automate the processing of tweets for the basketball shoes produced by an athletic shoe company. Your workflow will trigger when a new tweet is available. It will use cloud services to determine if the content is positive or negative and take different steps based on the results. After processing, it will save the data in a cloud-hosted database. By the end of this module, you'll be able to create workflows which route and process data using Azure Logic Apps and its built-in connectors.

## Learning objectives

In this module, you will:

* Construct a workflow model that will solve your business problem
* Launch your logic app in response to an external event
* Apply conditional logic to filter data and select a processing path
* Integrate Azure services into your workflow

## Prerequisites

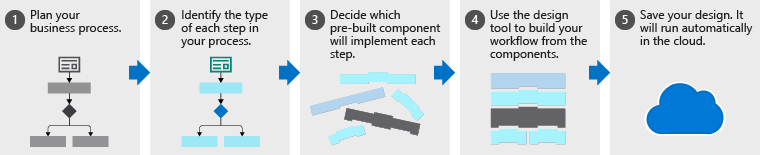
* Basic knowledge of the Azure Logic Apps concepts of connector, trigger, and action
* Experience creating and managing resources using the Azure portal at the beginner level
* Familiarity with programming concepts like conditional statements and loops

# Design Logic Apps for your workflows

Implementing any business process is difficult because you need to make diverse services work together. Think about everything your company uses to store and process data: Salesforce, Office 365, Oracle, Twitter, YouTube, Dropbox, Google services, Azure Cognitive Services, and so on. How do you integrate all these products?

Azure Logic Apps gives you pre-built components to connect to hundreds of services. You put the pieces together in any combination you need. For example, in this athletic shoe company scenario, we want to monitor social media reaction to our new basketball shoe product. We'll build a logic app to integrate Twitter, Azure Cognitive Services, SQL Server, and Outlook email.

In this unit, we'll plan the sequence of steps needed to implement a business process. Then we'll map those steps to the pre-built components in Azure Logic Apps. Finally, we'll assemble the parts into an app using the Logic Apps Designer. The following illustration shows an overview of the app-development process.



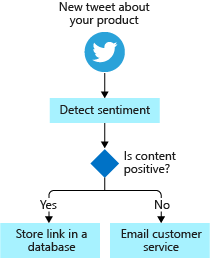
Each of these five steps in the app-development process is described in the following sections.

## Plan your business process

The first step to creating a logic app is planning the steps of your business process. In the social media monitoring scenario, the steps would be:

* Detect tweets about the product
* Analyze the sentiment
* Store a link to positive tweets, and
* Email customer service for negative tweets

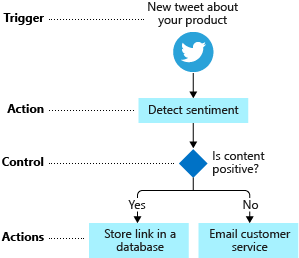
It's common to use a flowchart to capture the steps of the process. The following illustration shows the flowchart for the social media monitor app.



## Identify the type of each step in your process

The steps of a business process do different types of operations. Some respond to external events, some process or store data, and others make decisions based on the data. Logic Apps uses the terms *trigger*, *action*, and *control action* for these three categories.

For example, in our social media monitor scenario, we *trigger* the process when a new tweet is posted, perform *actions* like detect the sentiment, and make a *control* decision based on the sentiment score. The following illustration shows the type of each step in the social media monitor app.

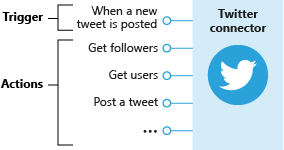


## Map your steps to Logic Apps components

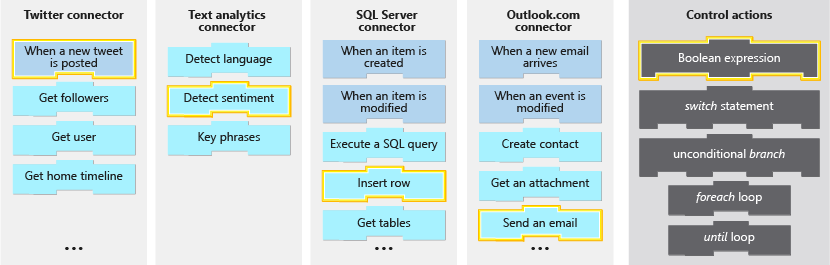
When deciding which pre-built component to implement, we need to be clear about component types. Let's be more formal about the definitions of the component types:

* A *trigger* is an event that occurs when a specific set of conditions is satisfied. Triggers activate automatically when the conditions are right (for example, when a timer expires or data becomes available). Every logic app must start with a trigger. In our example, we'll trigger the app when a new tweet mentions our product.
* An *action* is an operation that executes one of the tasks in your business process. Actions run when a trigger activates or another action completes. Our social media monitor app has three actions: detect sentiment, insert database row, and send email.
* *Control actions* are special built-in actions that let you add decisions and loops to your app. Our example will use a control action to branch based on the sentiment score.

A group of related triggers and actions are packaged inside a *connector*. Conceptually, you can think of a connector as a component that lets you access an external service. For example, the Twitter connector lets you send and receive tweets while the Microsoft 365 Outlook connector allows you to manage your email, calendar, and contacts. The following illustration shows the Twitter connector with its operations divided into the two categories.

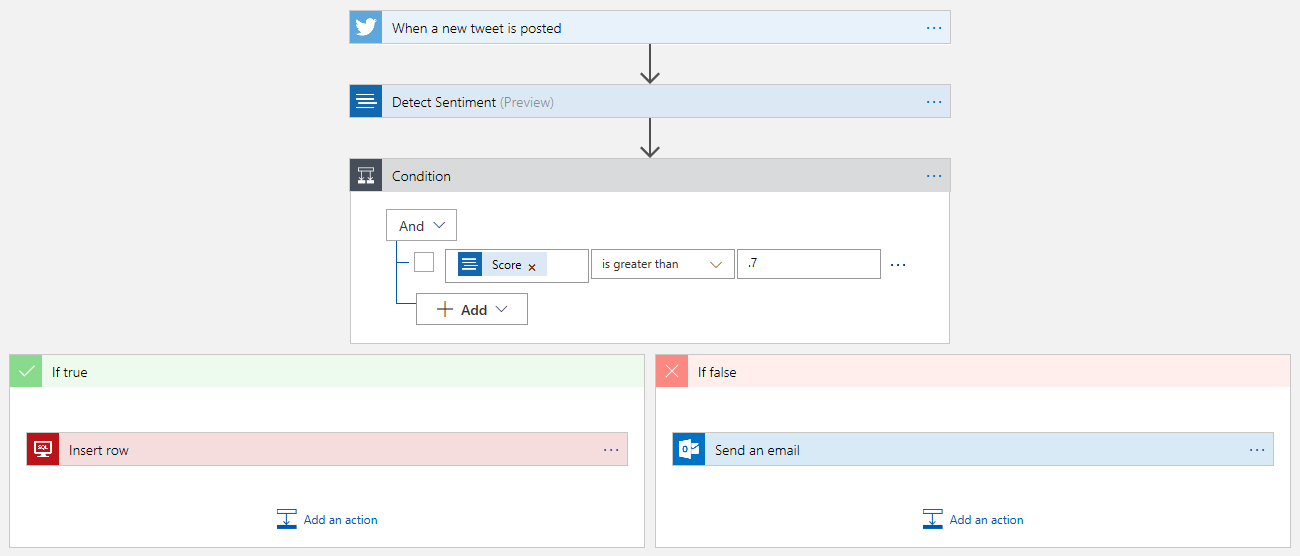


Your job is to map each task in your flowchart to Logic Apps triggers and actions. The following illustration shows the connectors we'd use in the social media app with the relevant triggers and actions highlighted.



## Define your app using the Logic Apps Designer

The Logic Apps Designer is a graphical tool for creating your workflows. It lets you pick from a gallery of connectors that contain the triggers and actions you can use in your app. You'll use the graphical Logic Apps Designer to arrange the trigger, actions, and control actions. The following screenshot shows the designer with the completed application.



When you save your app, it will be live and will run automatically whenever the trigger activates.

## Knowledge check

The questions in this knowledge check relate to the workflows shown in the following illustrations. Each question will ask you to evaluate one of the workflows to find the correct answer.

|  |  |
| --- | --- |
| Social media workflow | Email attachment processing workflow |
| An illustration of a social media monitoring workflow. This workflow triggers when a user posts a new tweet that mentions a specific product. It sends the text of the tweet through Text Analytics to determine sentiment. If the sentiment score is greater than 0.7, then a row containing the tweet is added to a database. If the tweet is rated less than 0.7, an email will be sent to customer support. | An illustration of an email attachment processing workflow. This workflow is triggered when a new email arrives. Next, there is a *if* statement that checks if the email has an attachment. If there are no attachments on the email, the workflow ends. If there are attachments, the workflow creates a blob for the email body. Next, a *foreach* loop creates a blob for every attachment. Finally, an email is sent for review. |
|  |  |

### Examine the illustration of the social-media workflow to answer this question. What would be the outcome of the workflow if a tweet was rated with a sentiment score of exactly 0.7?

* ❎ Send email
* ⬜ Insert row
* ⬜ Both send email and insert row

### Examine the illustration of the email-attachment processing workflow to answer this question. How many blobs would be created from an email with six attachments?

* ⬜ 6
* ❎ 7
* ⬜ 8

# Detect an external event using a trigger

You use a trigger to launch your logic app. You need to find the best one and configure it to run your app correctly without wasting time or money. In this scenario, we will use a Twitter trigger to launch our app when a tweet containing our product name is available.

In this unit, we'll examine the types of triggers and the strengths and weaknesses of the two most common options. We'll see how to create a logic app using the Azure portal and how to add a trigger using the Logic Apps Designer.

## Trigger types

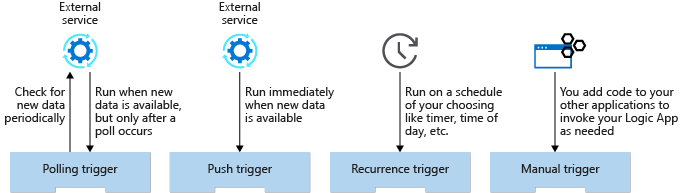
Think about the different conditions that businesses might use to launch their Logic Apps.

Most of the examples we've seen are in the *data becomes available* category. For example, a new tweet is posted, a new row is inserted into a database, a new email arrives, or a new file is uploaded to your cloud storage. This category doesn't cover all cases though.

Suppose you wanted to launch your logic app every Saturday at midnight? This trigger would be great for administrative tasks, like running backups or archiving old data. Logic Apps provides a built-in *recurrence* trigger to help you do exactly this type of thing.

There's one more case to consider: Suppose you wanted total control? Imagine you need to launch your logic app using code in your web or mobile applications? You can use the built-in *manual request* trigger to do this action.

This description shows that we have three broad categories of triggers: data, time, and manual. Data triggers use two different techniques to detect that new data is available: some use *polling* and some rely on the external service to *push* a notification. These two types of data triggers are so different, that we should think of them as separate categories. Altogether, we have four types of triggers, the following illustration shows a summary of the cases.

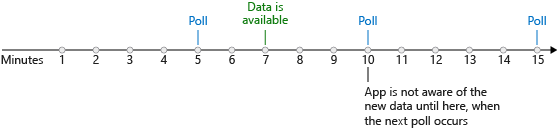


## What is a polling trigger?

A *polling trigger* periodically checks an external service for new data. For example, the trigger that looks for new posts in an RSS feed is implemented using polling.

When you create a polling trigger, you set the **Frequency** and an **Interval** to control how often the trigger will run. The frequency is the unit of measurement and has values like **Second**, **Minute**, and **Hour**. Interval is a number that represents how often to execute. For example, a polling trigger with a frequency of **Minute** and an interval of **5** would run every five minutes.

Polling triggers force you to make a choice between how much they cost and how quickly they respond to new data. There is often a delay between when new data becomes available and when it is detected by the app. The following illustration shows the issue.



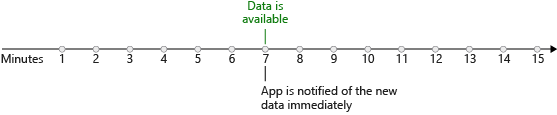
In the worst case, the potential delay for detecting new data is equal to the polling interval. So why not use a smaller interval? To check for new data, the Logic Apps execution engine needs to run your app, which means you incur a cost. In general, the shorter the interval, the higher the cost but the quicker you respond to new data. The best polling interval for your logic app depends on your business process and its tolerance for delay.

## What is a push trigger?

A *push trigger* subscribes to an event offered by the external service to get notified immediately when data is available. For example, the trigger that detects when a message is added to an Azure Service Bus queue is a push trigger.

NOTE  
Push triggers are implemented using webhooks. The Logic Apps infrastructure generates a callback URL for you and registers it with the external service. This registration happens when you first create your app and again when you make changes to your app's configuration. Similarly, Logic Apps de-registers the callback for you as needed (for example, if you disable or delete your app).

The favorable thing about push triggers is that they don't incur any costs polling for data when none is available. They also respond immediately when new data is ready. The following illustration shows this immediate response.



If push triggers respond more quickly and cost less than polling triggers, then why not use them all the time? The reason is that not every connector offers a push trigger. Sometimes the trigger author chose not to implement push and sometimes the external service didn't support push. Generally, you'll find a connector offers either push or polling triggers, but not both. In the rare cases where both options are available, consider using the push trigger because it should be more efficient.

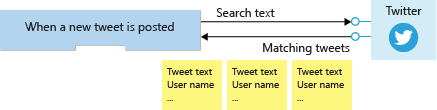
In this module, we're going to focus on polling triggers. These triggers are the most common and are perfect for the "route and process data" scenarios that we've been discussing.

## Trigger parameters and return values

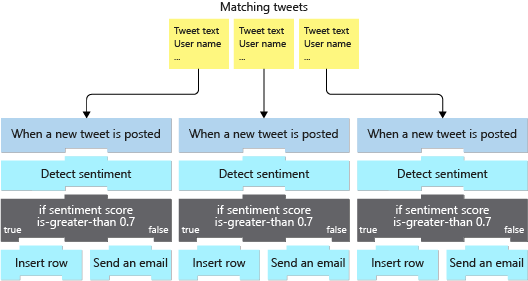
You can think of trigger operations as function calls that have parameters and return values.

Trigger *parameters* let you configure the operation. The Twitter **When-a-new-tweet-is-posted** trigger has a parameter called **Search text** that it uses to select matching tweets for us. Some operations have a mix of required and optional parameters. The SQL Server **When an item is created** trigger has one required parameter named **Table name** and several optional parameters like **Order By** and **Select Query**.

Trigger *return values* are the results of the operation. The bitbucket connector has a **When a pull request is merged** trigger. The trigger returns an object containing things like the identity of the **Repository** and the **Actor** who approved the merge. Most triggers actually return a collection instead of a single object. The Twitter **When a new tweet is posted** trigger returns an array of **TweetModel** objects. Each object contains values like the **Tweet text**, **User name**, and **Followers count**. The following illustration shows a collection being returned from a trigger.



You can use a loop to process each item or you can ask the trigger to split the array up for you. The default behavior for most triggers, including the Twitter trigger, is to automatically split the array. The Logic Apps execution engine will create one instance of your logic app for each data item and the instances will run in parallel. The following illustration shows how each item in the returned array is sent to a different instance of the logic app.



## How to create a logic app in the Azure portal

You can use the Azure portal to create a logic app. You select the **Logic App** resource type and enter the standard resource properties **Name**, **Subscription**, **Resource group**, and **Location**. After deployment completes, you can navigate to the Logic Apps resource that you created.

The Logic Apps team has created several *templates* for common application types. For example, there are templates for apps like **Post to Slack if a new tweet matches with some hashtag** and **Get daily reminders emailed to you**.

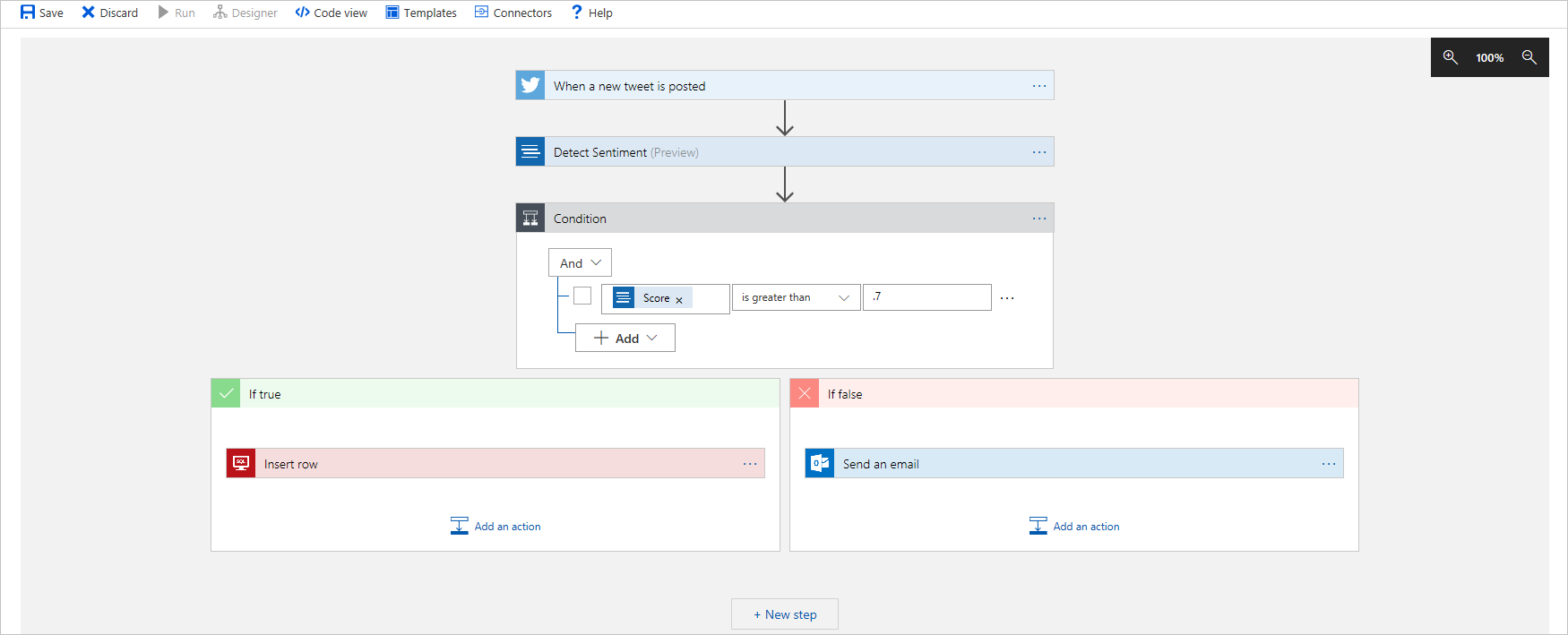
When you first navigate to your newly deployed logic app, you'll find a getting-started page. This page can add a common trigger to your app or generate an entire app for you from one of the templates. If any of these templates matches what you're working on, they can save you some time in getting your app set up. To do all the work yourself, there's also a **Blank Logic App** template.

After you select a starting template, you'll automatically navigate to the Logic Apps Designer.

## How to add a trigger using the designer

The Logic Apps Designer lets you pick from a gallery of connectors that contain the triggers and actions you can use in your app. The typical strategy is to use the search feature to locate the connector you are interested in. Then you look through the triggers supplied by the connector to find the one you want. In our case, we will use Twitter's **When-a-new-tweet-is-posted** trigger.

Once you've added the trigger, the Designer displays a GUI to set the trigger properties. We'll set the **Search text**, **Frequency**, and **Interval** parameters. The following screenshot shows the social media monitor logic app displayed in the designer; notice that it begins with the Twitter trigger.



# Exercise - Create the social media tracker Logic App

In this exercise, we'll create our social media logic app using the Azure portal. We'll add a Twitter trigger using the Logic Apps Designer. The following illustration shows a conceptual view of the app with the part that we'll work on highlighted.

## Locate the Azure logic app resource

The first thing we need to do is create an Azure logic app. To do that, we need to locate the Azure Logic Apps resource type in the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/learn.docs.microsoft.com?azure-portal=true) with the same account you used to activate the sandbox.
2. On the Azure portal menu, select **All Services**, and in the **All services** menu, under **Integration**, select **Logic apps**. The **Logic apps** pane appears.
3. In the top menu bar, select **Add**. The **Create Logic App** pane appears.

## Configure your Azure Logic App resource

Let's configure basic settings, like resource group and location.

1. On the **Basics** tab, enter the following values for each setting.

|  |  |
| --- | --- |
| Setting | Value |
| **Project Details** |  |
| Subscription | Concierge Subscription |
| Resource group | From the dropdown list, select {rgn [Sandbox resource group]} |
| **Instance Details** |  |
| Type | Consumption |
| Logic app name | Enter *ShoeTracker* |
| Region | Select a location closest to you from the dropdown list. |

{include "../../../includes/azure-sandbox-regions-first-mention-note-friendly.md" }

1. Select **Review + create** and then select **Create**. The Deployment pane displays the resources that are created. Wait for deployment to succeed.

## Use a template for your Azure logic app

When you create a logic app in the Azure portal, you have the option of selecting a starter template. Let's select a blank template so that we can build our logic app from scratch.

1. After deployment completes, select **Go to resource**. The **Logic Apps Designer** appears for your *ShoeTracker* logic app.
2. Scroll down to the *Templates* section, and select the **Blank Logic App**

## Create a Twitter trigger

Now, let's create the trigger and provide values for all required parameters.

NOTE  
If you do not have a Twitter account and prefer not to create one, use the following substitutions. Substitute the **When a feed item is published** in the search field, and then Select the RSS trigger in the bottom box. Set the **RSS feed URL** to https://blog.feedspot.com/reuters\_rss\_feeds/, the **Chosen property** to UpdatedOn, the **How often to check for items** to 1, and the **Frequency** to Minute. The disadvantage of this approach is that new articles appear infrequently in RSS feeds so you may have to wait a while before this trigger activates.

1. In the *Search connectors and triggers* field, enter *When a new tweet is posted*. In the lower dialog box, select the Twitter **When a new tweet is posted**.
2. A **Twitter** dialog box prompts you to enter:

|  |  |
| --- | --- |
| Setting | Value |
| Connection name | ShoeTrackerTwitterConnection |
| Authentication Type | Accept default (Use default shared application) |

1. Select **Sign in**. Sign in with your existing Twitter account and password, and select **Authorize app**. This action establishes the log in connection to your Twitter account.
2. When the Twitter **When a new tweet is posted** dialog box reappears you have created a valid connection. The dialog box has three required parameters:
   * **Search text:** Text to look for in the tweet text. You can include a hashtag character at the beginning of the search text to search for hashtags.
   * **Frequency:** Unit of time for search frequency. For example, Second, Minute, Hour, or Day.
   * **Interval:** How often to search. For example, an interval of 3 and a frequency of Hour would check for new tweets every three hours.
3. Enter the following values:

|  |  |
| --- | --- |
| Setting | Value |
| Search text | Shoe |
| How often do you want to check for new items? | 1 |
| (Frequency) | Minute |
| Add new parameter | Accept default (blank). |

1. In the command bar, select **Save**.
2. Enter the name of your logic app (ShoeTrackerNNN where NNN represents a unique ID, such as your initials and a number to provide a unique identifier.
3. Select **Run Trigger**.

## Examine the results of your Twitter trigger

At this point, our logic app is scanning Twitter every minute for tweets containing the search text. To verify the app is running and working correctly, we'll look at the **Runs history** table.

1. Scroll to the left, and then in the Logic App menu, select **Overview**. You may need to scroll to the left to see the navigation menu or you can use your browser's search function to find the word "overview" on the page.
2. Select **Refresh** once a minute until you see a row in the **Runs history** table.
3. While you are waiting, locate the section in **Overview** labeled **Trigger history**. Notice the text that looks something like **Evaluated 12 times, fired 3 times in the last 24 hours**. The term *evaluated* means the condition in your trigger was checked; you should see this increase once per minute since you're polling every minute. The term *fired* indicates the number of times the trigger conditions were satisfied; in our case, this number represents how many times the trigger found matching tweets.
4. Go back to **Runs history**. After you see a row appear, select the row. Selecting a row will navigate your view to something that looks like the designer you used to create the trigger. This view lets you see the data that flowed through each step of this run of your app.
5. Select the Twitter trigger.
6. Examine the data in the **OUTPUTS** section. For example, locate the text of the matching tweet.

NOTE  
Select **Show raw outputs** if you would like to see the entire response in JSON.

# Execute an action

Recall that you build Logic Apps from triggers and actions. A trigger starts your app and then actions do the rest of the work. The social media monitor uses actions to check if a tweet is positive or negative and then inserts a row into a database or sends an email.

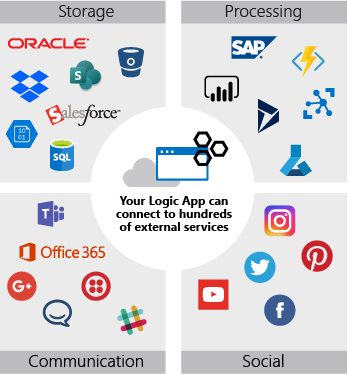
In this unit, we'll do a quick survey of the types of actions. Then we'll discuss how to use actions in your app by passing parameters and processing returned values. Finally, we'll see how to use the Logic Apps Designer to add an action and configure its settings.

## Action types

The core goal of Logic Apps is to help you get services to work together. There are three aspects to this goal: accessing external services, manipulating the data you get back, and altering the control flow through your app. Logic Apps gives you actions to do all of these things. Let's look at each of the categories.

### Access external services

Actions that let you use services outside of your logic app are by far the most common type. In fact, these actions are so common, that they are what most people mean when they use the term *action*. They give you access to hundreds of products like Salesforce, Oracle, YouTube, Dropbox, Gmail, GitHub, Twilio, Facebook, Slack, and Jira. The following illustration shows a few of the available external actions.



When you use an action to connect to an external service, you typically supply connection and authorization values. It might be a username, a password, a connection string, or an account key. The details will be different for each service.

For example, to connect to Cognitive Services, you give the **Site URL** so your logic app knows where to send the request and your **Account Key** to prove you are authorized to access the service. Both of these values are available in your Cognitive Services account. You'll typically copy them into the settings for the **Detect-sentiment** action.

This type of action helps you send out data for processing or pull data into your app. But then what? How do you actually work with that data inside your app? That's what the next category of action is for.

### Manipulate data

*Data operation actions* let you work with the data that you pull into your logic app. There are operations to concatenate multiple values into a single string, parse JSON data, select particular values from an array, and so on.

Let's look at an example. If you've used Twitter for a while, you might be familiar with their URL pattern to link to a specific tweet:

https://twitter.com/[username]/status/[id]

language-plaintext

This type of link could be useful in the social media app. You might want to store it in your database so you can access the tweet again later. Or you might want to include it in an email notification so your coworkers can see the tweet in its original form.

The data you get from the Twitter trigger does not include this complete URL. However, it does contain the username in the **TweetedBy** field and the ID in the **TweetId** field. You can use the **Compose** data operation to put the pieces together into the URL. Conceptually, the operation would be:

Compose: "https://twitter.com/" + [TweetedBy] + "/status/" + [TweetId]

language-plaintext

### Alter control flow

The *control action* feature of Logic Apps lets you add control constructs like conditional statements and loops to your app. The social media monitor app will use a control action to branch based on whether the tweet sentiment is positive or negative. We'll see how this works and discuss all the control actions later in this module.

## Action parameters and return values

You can think of actions as function calls that have parameters and return values.

Action *parameters* let you pass values to the operation. The Cognitive Services **Detect-sentiment** action has a parameter called **Text** that represents the string you want to analyze. It also has an optional parameter named **Language** that lets you provide a hint about the language of the **Text**.

Action *return values* are the results of the operation. The **Detect sentiment** action returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment, while scores close to 0 indicate negative sentiment. The following illustration summarizes the data flow for the **Detect sentiment** action as used by the social media monitor app.

An illustration showing a logic app using the **Detect sentiment** action to invoke the Text Analytics service. The app passes the tweet text to the service and gets back a numeric sentiment score.

## Static vs. dynamic parameters

Some of the parameters you send to an action will be the same every time. For example, you might only be interested in tweets in English so you would always pass "en" as the **Language** parameter to the **Detect sentiment** action. There's nothing tricky about this type of static parameter. You hard-code the value when you configure the action and that value is used every time the action runs.

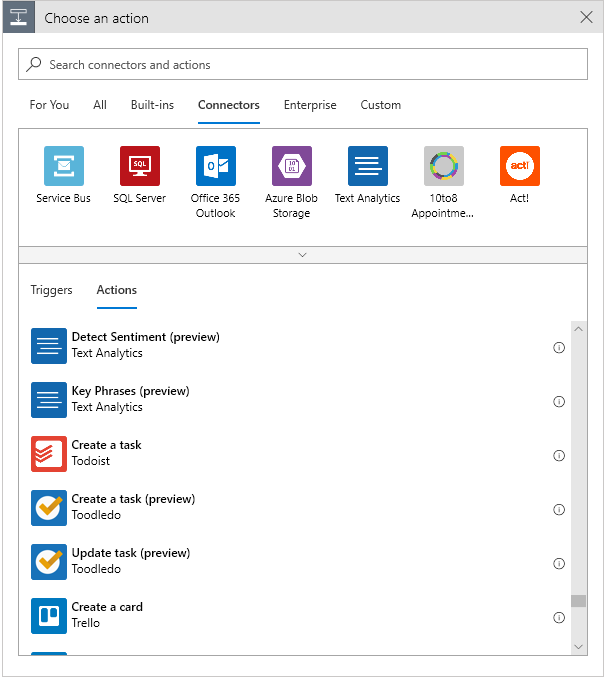
Other parameters will be different every time you execute an action. The **Text** parameter in the **Detect sentiment** action is a good example of this. When you pass in the text of the tweet, it will be different for every run. How do you get access to this dynamic value?

Recall that the tweet we're processing was the return value of the **When a new tweet is posted** trigger. This tweet data is packaged inside an object with several fields containing the details of the tweet. Logic Apps automatically makes this object available throughout the rest of the app. The Logic Apps Designer gives you a simple GUI to select the fields you're interested in.

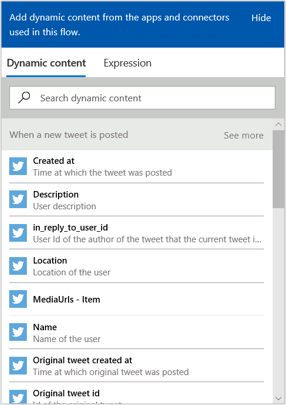
## Add and configure an action using the designer

The Logic Apps Designer gives you a GUI to add and configure actions. You can add new actions to the end of your app or insert an action between existing steps.

The first step is to select the **Add an action** option at the point in your app where you want the new action to appear. This launches the "choose an action" UI. The following screenshot shows how to add a new action using the Logic Apps Designer.



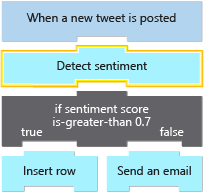
After you add an action, you use the designer to set configuration parameters. Again, the designer provides a GUI for you to enter the values. Static content can be added manually. For dynamic content, the designer can pop up a selection GUI that shows you data that was returned by previous steps. You choose the field you need, and the designer automatically populates that entry in your action. The following image shows the dynamic tweet data from the social media monitor app displayed in the designer.



The designer makes it easy to add dynamic content. You don't need to remember the names of the parameters and return values, and you don't have to worry about the details of the syntax.

# Exercise - Analyze the content of a tweet

In this exercise, we're going to continue work on our social media monitor app. We'll analyze whether the tweets about our product are positive or negative. We'll create a **Detect sentiment** action, which will provide us a numeric *score* that represents the sentiment of the tweet. The following illustration shows a conceptual view of the app with the part that we'll work on highlighted.



## Get Text Analytics key and endpoint

The Azure Text Analytics API provides natural language processing of text. It lets you do sentiment analysis, key phrase extraction, language detection, and entity linking. We'll be using it for sentiment analysis. We'll run a script in this section to programmatically set up a Cognitive Services account in the sandbox, register the Text Analytics service, and return a key and the endpoint URL to us. We'll need those values in this exercise to make calls and get back sentiment scores.

The Cognitive Services account is created in the sandbox environment and is, therefore, free for use in this module.

1. In Azure Cloud Shell to the right, run the following curl command to copy the **setup-textanalytics.sh** script from GitHub.

curl https://raw.githubusercontent.com/MicrosoftDocs/mslearn-route-and-process-data-logic-apps/master/setup-textanalytics.sh > setup-textanalytics.sh

azurecli

1. Run the following command to execute the script. This command will take a couple of minutes.

bash setup-textanalytics.sh

azurecli

1. Wait for the script to complete. When it finishes, Cloud Shell displays values for the following properties.
   * **Cognitive Services account key**
   * **Cognitive Services account endpoint**

Save the values that are displayed in Cloud Shell somewhere safe. We need them in this exercise when we update our app in the portal.

## Locate the **Detect sentiment** action

Here, we'll locate the **Detect sentiment** action using the [Azure portal](https://portal.azure.com/learn.docs.microsoft.com?azure-portal=true).

1. Return to the Logic Apps Designer by selecting **Logic Apps Designer**. This button is located under the **Development Tools** section of the left menu pane.
2. Under the Twitter trigger, select **New step**.
3. In the **Search connectors and actions** search box, enter Text Analytics.
4. Select the **Text Analytics** connector.
5. In the **Actions** section, select **Detect Sentiment**.

## Create a **Detect sentiment** action

Now that we located the **Detect sentiment** action, let's create an instance of it. In this process, we'll also configure it to connect to our Text Analytics subscription.

1. In the **Connection Name** field, enter **CognitiveServicesConnection**.
2. Paste your saved API key in the **Account Key** field, and your saved endpoint address in the **Site URL** field.
3. Select **Create**.

## Configure your **Detect sentiment** action

Next, we'll configure our logic app to pass the tweet text to the **Detect sentiment** action.

1. In the **Detect Sentiment** step, select **documents** from the **Add new parameter** dropdown list.
2. Select the **documents Text** field in the **Detect Sentiment** action.
3. In the **Dynamic** content popup, select **Tweet text**.

NOTE  
If you are using the RSS **When a feed item is published** trigger, send the **FeedSummary** to the Text Analytics service.

1. Select **Save** in the Logic Apps Designer.

## Examine the results of your **Detect sentiment** action

Your app is now live. Tweets containing your product name will be processed by the Text Analytics service and you'll be given a numeric score. Recall that a score close to 1 is positive sentiment while a score near 0 is negative. In this section, you'll see how to monitor the execution of your app and view the data flowing through each step. This tool is great to know about since it helps you verify your app is working correctly.

1. In the left menu pane, select **Overview**.
2. Select **Refresh** once a minute until you see a new row item in the **Runs history** table.

NOTE  
Each item in the **Runs history** table represents a separate tweet that contained the name of the product you entered in the **Search text** of your trigger.

1. After you see a row item appear, select the item.
2. Select the **Detect sentiment** action.
3. Locate the text of the tweet and the score that was given to it by the Cognitive Services engine.

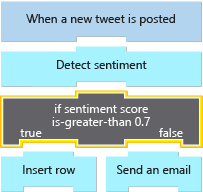
# Alter control flow using a control action

Control actions let your logic app make decisions. The social media monitor will use a control action to test the sentiment score of a tweet and branch based on whether it is positive or negative. In this unit, we'll look at the four control actions that help you manage the execution path through your app. We'll also see how to use the Logic Apps Designer to add a control action and configure its settings.

## *Condition* action

A *condition* control action is an *if* statement that lets your app do different things based on the data you're processing. It consists of a Boolean expression and two actions. At runtime, the execution engine evaluates the expression and chooses an action based on whether the expression is true or false.

For example, you might want to route an expense report to a different manager based on the amount. If you're processing an email, you might need to test whether it is flagged as high-priority. In our social media app, we'll use a *condition* statement to branch based on the sentiment score of the tweet. The following illustration shows the use of the *condition* control action in our app.



### Types and operators

You can test numeric, string, Boolean, and JSON objects in your condition control actions. The following pseudocode shows one example of each of the simple types:

if (score is-greater-than 0.7) ... // Numeric  
if (TweetedBy does-not-contain "MyCompany") ... // String  
if (Favorited is-equal-to true) ... // Boolean

language-plaintext

Each type has a set of operators you can use in your comparisons. The following table lists the operators for each type.

|  |  |  |  |
| --- | --- | --- | --- |
| Numeric | String | Boolean | JSON objects |
| is-equal-to | is-equal-to | is-equal-to | is-equal-to |
| is-not-equal-to | is-not-equal-to | is-not-equal-to | is-not-equal-to |
| is-greater-than | contains |  |  |
| is-greater-than-or-equal-to | does-not-contain |  |  |
| is-less-than | starts-with |  |  |
| is-less-than-or-equal-to | does-not-start-with |  |  |
|  | ends-with |  |  |
|  | does-not-end-with |  |  |

Most of the operations are intuitive, but there are two cases worth mentioning:

* String comparisons are all case sensitive.
* JSON comparisons use what's called *deep* equals. This comparison means the equality operators will compare the entire object, including any descendant tokens inside complex objects.

### Combine expressions using **AND** and **OR**

Logic Apps lets you create complex expressions by combining conditions using **AND** or **OR**. Suppose you wanted to identify tweets that were neutral in their sentiment (recall that a score close to 0.5 is neutral). You could write an expression similar to the following pseudocode:

if (score is-greater-than 0.4 AND score is-less-than 0.6)

language-plaintext

### Data availability

Logic Apps makes the data from all previous steps available in subsequent steps. This feature means that your expressions can use values generated by any of the previous steps. You can even combine values from different steps in one expression. For example, the following pseudocode looks for tweets with positive sentiment that were *not* sent by your company. Notice that the **score** comes from the **Detect sentiment** action while the **TweetedBy** value is from the **When a new tweet is posted** trigger.

if (score is-greater-than 0.7 AND TweetedBy does-not-contain "MyCompany")

language-plaintext

### Create complex expressions with groups

Suppose you want to build an expression to identify influential tweets. You decide that there are two criteria that would qualify:

* The sentiment score is above 0.9 and the tweet has been marked as a favorite.
* The retweet count is greater than 1000.

You want to build an expression like the following pseudocode to capture this idea:

if (score is-greater-than 0.9 AND Favorited is-equal-to true OR RetweetCount is-greater-than 1000) // Error, cannot mix AND and OR

language-plaintext

Logic Apps does support this expression, but not directly. You can use a feature called "groups" to do it. A *group* is a sequence of expressions combined with either **AND** or **OR**. You can't mix **AND** and **OR** within a group.

Let's look at a few examples expressed as pseudocode. To help make it easier to read, we'll use lowercase letters to represent the expressions that we need to combine. The first example below is a legal group but the second one is not:

if (a AND b AND c) // OK  
if (a AND b OR c) // Error, cannot mix AND and OR

language-plaintext

Instead, you would create a group for the expressions connected by **AND** in the preceding example. Then use **OR** to connect the group to the other expression. We use brackets in the following pseudocode to represent a group:

if ([a AND b] OR c) // OK

language-plaintext

## *Switch* action

A *switch* control action compares a value against several cases and executes only the one that matches. The supported types for a *switch* expression are *string* and *integer*. A *switch* can include a *default* action that runs if there's no match.

For example, suppose you are processing an email message and want to respond differently based on the message **Importance**. The **Importance** in an Outlook email is an integer between 0 and 2. The following pseudocode shows how you might build a *switch* action for email importance.

switch (Importance)  
 case 0: ... // action(s) for low importance  
 case 1: ... // action(s) for medium importance  
 case 2: ... // action(s) for high importance  
 default: ...

language-plaintext

## *Foreach* loop

A *foreach* loop control action processes an array. It performs the same actions on each array item. By default, the actions for each array element run in parallel, although you can control this behavior in the loop's configuration.

For example, part of the tweet data returned by the **When a new tweet is posted** trigger is an array of URLs for the media included in the tweet. Suppose you wanted to insert each URL into its own row in a database. You could use a *foreach* action like the following pseudocode to do the processing:

foreach url in MediaUrls  
 insert-row

language-plaintext

## *Until* loop

The *until* loop control action runs a group of actions multiple times. You can set three different stop criteria and the loop runs until one of them is true:

* Condition: an expression evaluated after every iteration.
* Count: the maximum number of iterations (the default is 60).
* Timeout: the maximum clock time allowed specified using ISO 8601 format (the default is one hour).

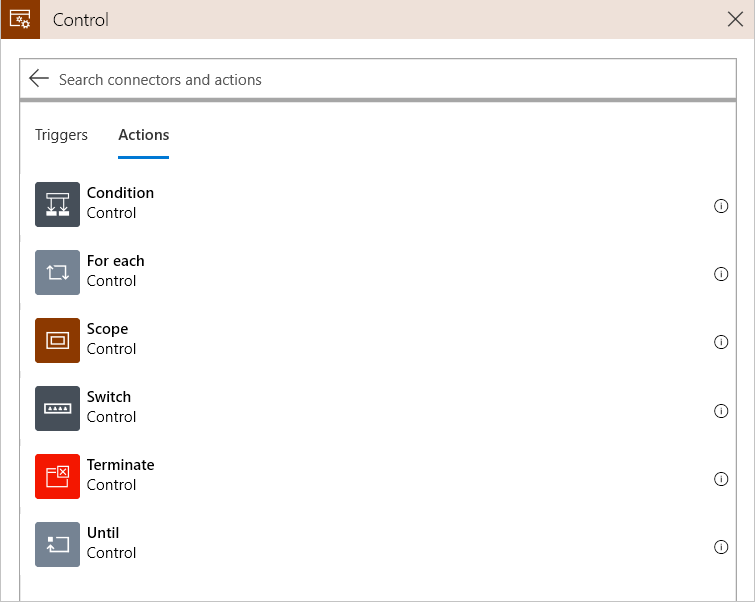
This loop can be used to process data, but it's also a good option when you need to retry a networking operation until it succeeds or times out. For example, suppose your app needed to run an action that made an HTTP request. You could use an *until* loop similar to that shown in the following pseudocode (the time value PT5M is equal to five minutes):

repeat  
 HTTP Get  
until (StatusCode is-equal-to 200 OR Count is-greater-than 3 OR Timeout is-greater-than PT5M)

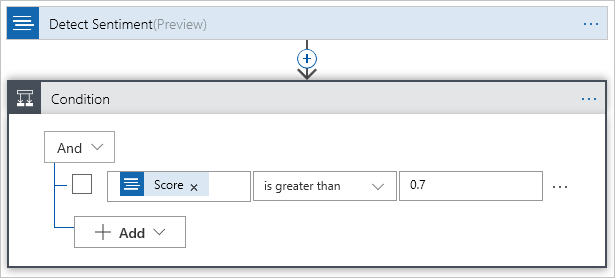
language-plaintext

## Add a control action using the designer

The Logic Apps Designer gives you a GUI to add and configure a control action. They are all packaged inside the **Control** connector, so the first step is finding that connector. Once you've located the connector, you'll see the four control actions we discussed and a few others that are useful, but not directly related to our goal of managing control flow. The following screenshot shows the available control actions displayed in the designer.



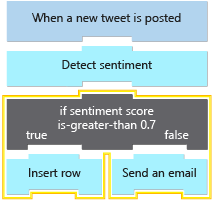
The designer gives you a GUI that lets you build complex expressions, included groups. The following screenshot shows a *condition* action displayed in the designer.



In our social media monitor app, we'll be using the **Condition** control action to add an if-statement to test the sentiment score.

# Exercise - Branch based on tweet sentiment

In this exercise, we're going to continue work on our social media monitor app. We'll add a control action to branch based on the sentiment of the tweet. The following illustration shows a conceptual view of the app with the part that we'll work on highlighted.



## Create SQL Server database to store positive tweets

When we receive a positive tweet, we want to save it to a backend database. In this section, we'll run a script to create a database in the sandbox for us to use. You incur no costs; the database runs in the sandbox and is free for the purposes of this exercise.

1. In Cloud Shell to the right, run the following curl command to copy the **setup-sql-database.sh** script from GitHub.

curl https://raw.githubusercontent.com/MicrosoftDocs/mslearn-route-and-process-data-logic-apps/master/setup-sql-database.sh > setup-sql-database.sh

azurecli

1. Run the following command to run the script. This command will take a couple of minutes.

bash setup-sql-database.sh

azurecli

1. Wait for the script to complete. When it finishes, Cloud Shell displays values for the following properties.
   * **SQL Server instance name**
   * **SQL Server username**
   * **SQL Server password**
   * **SQL Server database name**

Save the values that appear in Cloud Shell somewhere safe. We need them in this exercise as we update our app in the Azure portal.

## Create a **Condition** control action

This first step is to add the control action to the app in the [Azure portal](https://portal.azure.com/learn.docs.microsoft.com?azure-portal=true). In programming terms, we're adding an *if statement* that will test a condition.

1. Return to the Logic Apps Designer by selecting **Logic Apps Designer**. This button is located under the development tools section of the left menu pane.
2. Below the existing **Detect sentiment** action, select **New step**.
3. In the **Search connectors and actions** search box, enter *control*.
4. Select the **Control** connector.
5. In the actions section, select **Condition**.

## Configure the condition

Now that we have a **Condition** control action created, we need to specify what the condition is. Remember that the **Detect sentiment** action returns a *score*, which is a number between zero and one. If the number is greater than 0.7, we'll consider the tweet positive, otherwise it will be negative.

1. In the **Condition** action, select the leftmost **Choose a value** field.
2. In the **Dynamic** content popup, select **Score**.
3. Expand the dropdown list that currently lists **is equal to**.
4. Select **is greater than**.
5. Enter **0.7** in the rightmost **Choose a value** field.
6. Select **Save** to save your work.

## Locate the SQL Server insert row action

The condition action is configured to detect if the **Score** value is greater than **0.7**; however, we haven't specified what should happen in that case. As a reminder, we want to store parts of a tweet into a SQL Database. To do that, let's locate the SQL Server insert row action.

1. In the **If true** section of the condition action, select **Add an action**.
2. In the search connectors and actions field, enter *SQL*.
3. Select the **SQL Server** connector.
4. Select **Insert row**.

## Create a SQL Server insert row action

Now that we have located the SQL Server action, let's create it. During creation, we'll also provide our SQL Server database name and login credentials.

1. Enter *SQLConnection* in the **Connection Name** field.
2. Select the SQL Server that you previously created.
3. In the **SQL Database Name** field, select PositiveTweetDatabase, which is the name of the database we created with our script.
4. Enter the **SQL username** and **SQL password** that you saved earlier when the setup script finished.
5. Select **Create**.

## Configure your SQL Server **Insert row** action

The SQL Server action is now created; however, we need to specify how the tweet data will be mapped to our database's columns. Let's store the tweet's text in a column called Content, and the username of the person who made the tweet in a column called Source.

1. Select the table Mentions from the **Table name** dropdown list.
2. Select Content from the **Add new parameter** dropdown list.
3. In the **Dynamic** content popup, select **Tweet text**.

NOTE  
If you are using the RSS **When a feed item is published** trigger, use the **FeedSummary** here.

1. Select Source from the **Add new parameter** dropdown list.
2. In the **Dynamic** content popup, select **Tweeted by**.

NOTE  
If you are using the RSS **When a feed item is published** trigger, use the **FeedTitle** here.

1. Select **Save**.

## Locate the Outlook **Send an email** action

When the **Score** value is greater than **0.7**, we add the tweet to a SQL Server database. Let's now take tweets that are **0.7** or less and forward them to customer support through email. To start, we need to locate the Outlook **Send an email** action.

NOTE  
If you do not have an Outlook.com email account and prefer not to create one, you can change the connectors search filter to **Send an email** and select another email provider such as Gmail and Office 365 Outlook.

1. In the **If false** section of the **Condition** action, select **Add an action**.
2. In the **Search connectors and actions** field, enter *Outlook*.
3. Select **Outlook.com**.
4. Select **Send an email**.

## Configure your Outlook **Send an email** action

Now that we located the **Send an email** action, we need to map the tweet data to the respective email fields.

1. Select **Sign in**.
2. Sign in using a Microsoft account. If you don't have an account, you can create one now.
3. Select **Yes** to let your logic app have access to your email information.
4. In the **To** field, enter a valid email address. For testing, you can use your own address.
5. In the **Subject** field, enter *Negative tweet detected from* and then in the dynamic content popup, select **Original tweet tweeted by**.

NOTE  
If you are using the RSS **When a feed item is published** trigger, use the **FeedTitle** here.

1. In the **Body** field, enter *Contents of tweet.* In the dynamic content popup, select **Original tweet text**.

NOTE  
If you are using the RSS **When a feed item is published** trigger, use the **FeedSummary** here.

1. Select **Save**.

## Examine results of positive sentiment tweets

The **Control** action is now properly configured and running. Let's examine the SQL Server database to see our positive tweets.

1. In the Azure portal menu, select **All resources**, and then select **PositiveTweetDatabase** from the list of resources.
2. In the left menu pane, select **Query editor**.
3. Sign in using the **Server admin login** and **Password** that you saved when the script in the preceding unit created.
4. Select **OK**.
5. Select **New Query** on the top menu bar.
6. In the query editor, enter Select \* from dbo.mentions.
7. Select **Run** to run the query, and list all positive tweets that have been written to the database.

## Examine results of negative sentiment tweets

Finally, let's check our email account to see if we received any messages about negative tweets.

Sign into the email account that you provided in the Outlook action, and wait for an email notification to arrive.

# Summary

Our marketing team needed to gauge customer response to our new shoe. We wanted to monitor social media, determine customer reaction, and route the data to a database or to customer service based on sentiment.

Azure Logic Apps let us automate the process. Each step of our workflow was mapped to one of the built-in components. We used the Twitter trigger to detect mentions of our product and launch our app. The Azure Cognitive Services action let us analyze whether the tweets were positive or negative. A control action helped us decide where to route the tweet based on sentiment. Finally, we used actions to insert a row in SQL Server and send an Outlook email.

Imagine how much work it would be to build this app from scratch. We'd have to write code to access each of the services' APIs. We'd need a polling infrastructure to monitor Twitter and trigger our app when new tweets were available. Once all the code was ready, we'd need servers to host the app.

Logic Apps made it easy. The standard connectors did the hard work of integrating systems that were never designed to work together. Building the app took us less than an hour and we didn't have to write any code or set up any servers. We can now analyze every tweet about our product and our business analytics can use the data to shape future products.

{include "../../../includes/azure-sandbox-cleanup.md" }