



900 XP

Discover sentiment in text with the Text Analytics API

48 min • Module • 8 Units

★★★★☆ 4.6 (1,125) [Rate it](#)

Intermediate

Developer

AI Engineer

Student

Azure

Cosmos DB

Cloud Shell

Functions

Azure Portal

Storage

Cognitive Services

Learn what your customers are really saying about your product or brand when they send feedback. We'll create a solution that uses Azure Functions and the intelligence of the Text Analytics API to discover sentiment in text messages.

In this module, you will:

- Learn about Text Analytics API
- Sign up for a Text Analytics API key
- Design and build a service with Azure Functions that uses the Text Analytics API to sort text feedback
- Read and write Azure Queue storage messages in a function app with the help of bindings

[Start >](#)[Bookmark](#)[Add to collection](#)

Prerequisites

None

This module is part of these learning paths

[Evaluate text with Azure Cognitive Language Services](#)

Introduction

2 min

Sentiment analysis and the Text Analytics API

4 min

Exercise - Call the Text Analytics API from the online testing console

10 min

Design a feedback sorter

6 min

Exercise - Create a function app from the portal to host our business logic

7 min

Exercise - Call the Text Analytics API from an app

9 min

Exercise - Sort messages into different queues based on sentiment score

8 min

Summary

2 min

Summary

Introduction	4
Learning objectives	4
Sentiment analysis and the Text Analytics API.....	5
Azure Cognitive Services	5
Text Analytics API	6
Exercise - Call the Text Analytics API from the online testing console.....	7
Create an access key	7
Get the access key	9
Call the API from the testing console	11
Make some API calls	13
Detect Language	17
Entities.....	19
Design a feedback sorter	22
Manage customer feedback more efficiently	22
A solution based on Azure Functions, Azure Queue Storage, and Text Analytics API	23
Steps to implement our solution.....	24
Exercise - Create a function app from the portal to host our business logic	25
Create a Function App to host our function.....	25
Create a function to execute our logic	30
Try it out.....	33
Exercise - Call the Text Analytics API from an app	36
Try it out.....	39
Add a message to the queue.....	42
Exercise - Sort messages into different queues based on sentiment score.....	47
Add output bindings to function.json	47
Update the function implementation to sort feedback into queues based on sentiment score	49
Try it out.....	52
Summary.....	57
Suggestions for further enhancement of our solution	57

Further reading.....	58
Clean up	58
Check your knowledge	59

Introduction

If I showed you text written in ALL CAPITAL LETTERS, what emotions would it convey? If a book review I wrote claims that "the ending was unpredictable," is that statement a good or bad thing? How can I find out programmatically what language an email was written in? Text Analytics is about understanding and analyzing unstructured text to answer these kinds of questions. It covers sentiment analysis, key phrase extraction, language detection, and more.

In this module, we're going to focus our attention on sentiment analysis. We'll learn about the Text Analytics API. This cloud-based service from Microsoft provides advanced natural language processing (NLP) over raw text. When you've completed the module, you'll have built a solution with Azure Functions that sorts text feedback based on sentiment.

Learning objectives

In this module, you will:

- Learn about Text Analytics API
- Sign up for a Text Analytics API key
- Design and build a service with Azure Functions that uses the Text Analytics API to sort text feedback
- Read and write Azure Queue storage messages in a function app with the help of bindings

Sentiment analysis and the Text Analytics API

We all want to know what our customers think of our brand, our product, and our message. How does their opinion change over time? Looking for sentiment in what they write can unlock some clues. Sentiment analysis helps answer the question, *What do our customers really want?* It's used to analyze tweets and other social media content, customer reviews, and emails.



A popular approach to sentiment analysis is to train machine learning models that detect sentiment. However, that process is complex. It involves having good-quality training data that is labeled, creating features from that data, training a classifier, and then using the classifier to predict sentiment of new pieces of text. Not every company has the money and expertise to invest in building AI solutions from scratch. Thankfully, Microsoft and other companies can, and do, invest in state-of-the-art research in these areas. As developers, we get to benefit from their findings through the APIs, SDKs, and platforms they ship. Microsoft Cognitive Services is one such offering.

Azure Cognitive Services

Microsoft Cognitive Services consists of a set of APIs, SDKs and services. The goal is to help developers make their apps more intelligent, engaging, and discoverable.

Azure Cognitive Services offers intelligent algorithms in vision, speech, language, knowledge, and search. To see what's on offer, check out the [Cognitive Services Directory](#). You can try each service for free. When you decide to integrate one or more of these services into your applications, you sign up for a paid subscription. The service we'll use throughout this module is the Text Analytics API, so let's hear more about it.

Text Analytics API

Text Analytics API is a Cognitive Service designed to help you extract information from text. Through the service you can identify language, discover sentiment, extract key phrases, and detect well-known entities from text.

In this lesson, we'll get to know the sentiment analysis part of this API. Under the covers, the service uses a machine learning classification algorithm to generate a sentiment score between 0 and 1. Scores closer to 1 indicate positive sentiment, while scores closer to 0 indicate negative sentiment. A score close to 0.5 indicates no sentiment or a neutral statement. You don't have to worry about the implementation details of the algorithm. You focus on using the service by making calls to it from your app. As we'll see shortly, you structure a **POST** request, send it to the `/sentiment` endpoint, and receive a JSON response that tells you a *sentiment score*.

We'll first experiment with the Text Analytics API using an online API testing console. Once we're comfortable with the API, we'll use it in a scenario to detect sentiment in messages so that we can sort them for further processing.

Exercise - Call the Text Analytics API from the online testing console

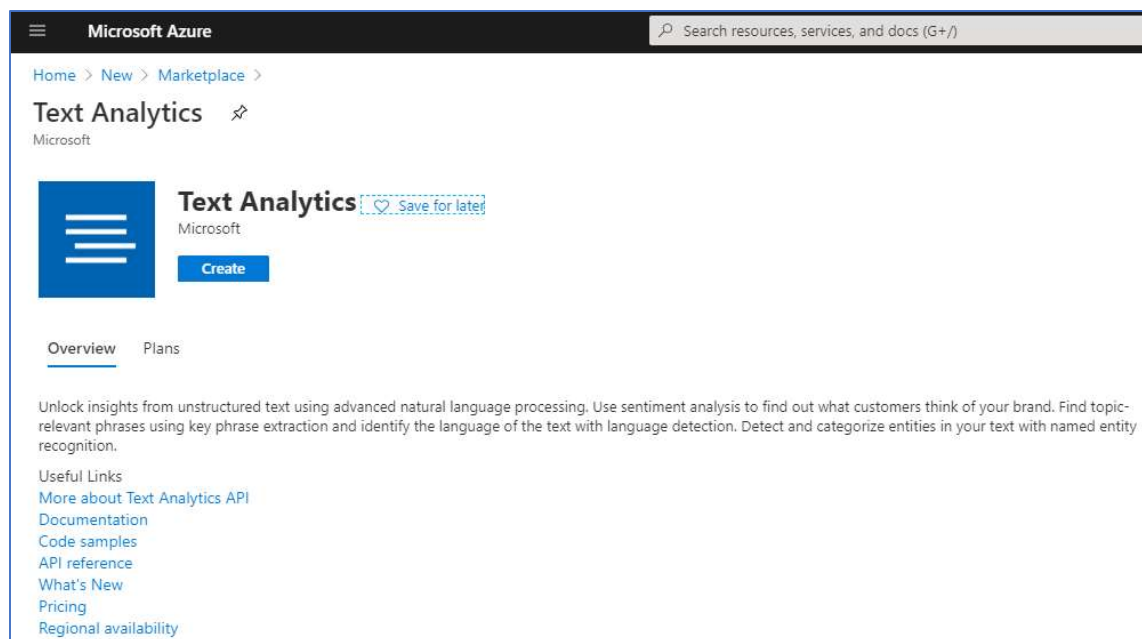
39% Creating resource group...

To see the Text Analytics API in action, let's make some calls using the built-in API testing console tool located in the online reference documentation. Before we do that, we'll need an access key to make those calls.

Create an access key

Every call to Text Analytics API requires a subscription key. Often called an access key, it is used to validate that you have access to make the call. We'll use the Azure portal to grab a key.

1. Sign into the [Azure portal](#) using the same account you used to activate the sandbox.
2. On the Azure portal menu or from the **Home** page, select **Create a resource**.
3. In the **Search the Marketplace** search box, type in *text analytics* and hit the Enter Or Return key.
4. Select **Text Analytics** in the search results and then select the **Create** button in the bottom right of the screen.



5. In the **Create** page that opens, enter the following values into each field.

Property	Value	Description
Name	<i>Choose a unique name</i>	The name of the Cognitive Services account. We recommend using a descriptive name. Valid characters are a-z, 0-9, and -.
Subscription	Concierge Subscription	The subscription under which this new Cognitive Services API account with Text Analytics API is created.
Location	<i>Choose a region from the dropdown</i>	Select a location from the dropdown.
Pricing tier	F0 Free	The cost of your Cognitive Services account depends on the actual usage and the options you choose. We recommend selecting the free tier for our purposes here.
Resource group	Select Use existing and choose [sandbox resource group name]	Name for the new resource group in which to create your Cognitive Services Text Analytics API account.

Tip

Remember the location you selected when creating the Text Analytics cognitive services account. You'll use it to make API calls shortly.=

Microsoft Azure

Home > New > Marketplace > Text Analytics >

Create

Text Analytics

Name *

PrepAI100-TextAnalytics

Subscription *

Azure Pass – Sponsorship

Location *

((US) South Central US

Pricing tier ([View full pricing details](#)) *

Free F0 (5K Transactions per 30 days)

Resource group *

Prep-AI100-DougLuna

[Create new](#)

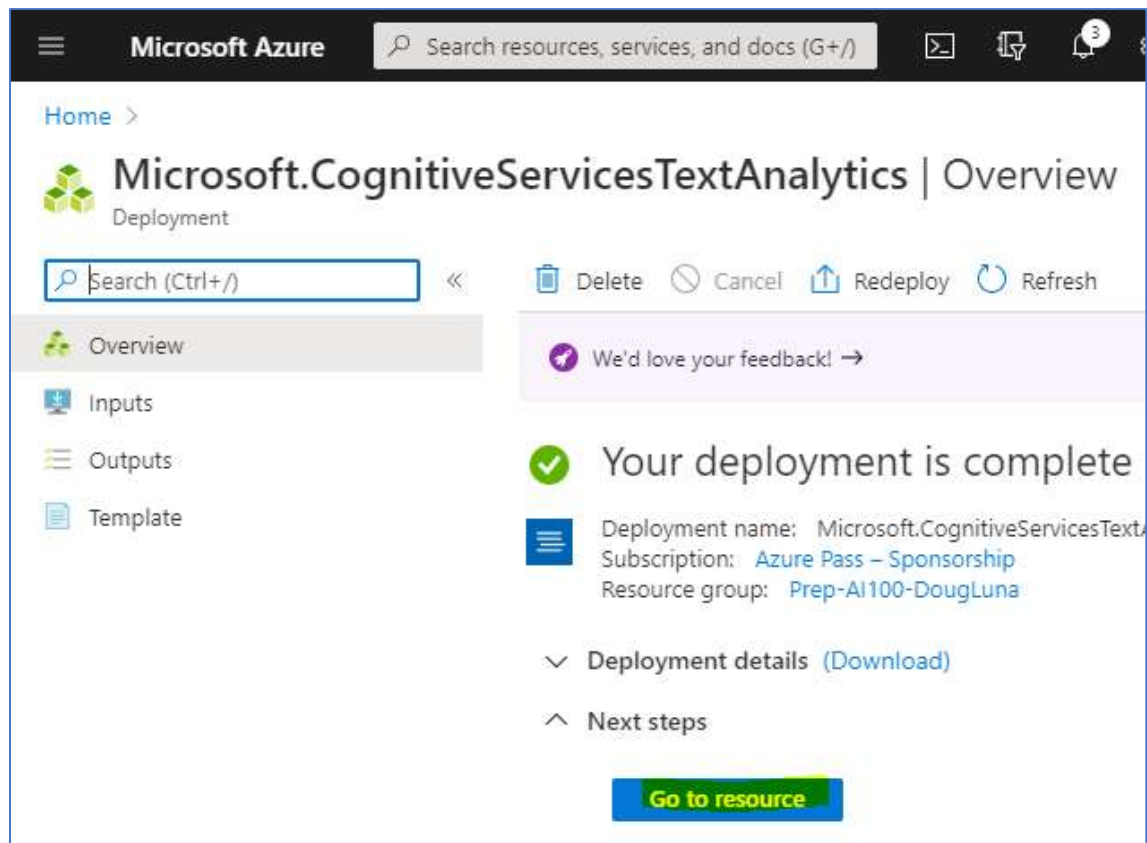
6. Select **Create** at the bottom of the page to start the account creation process. Watch for a notification that the deployment is in progress. You'll then get a notification that the account has been deployed successfully to your resource group.



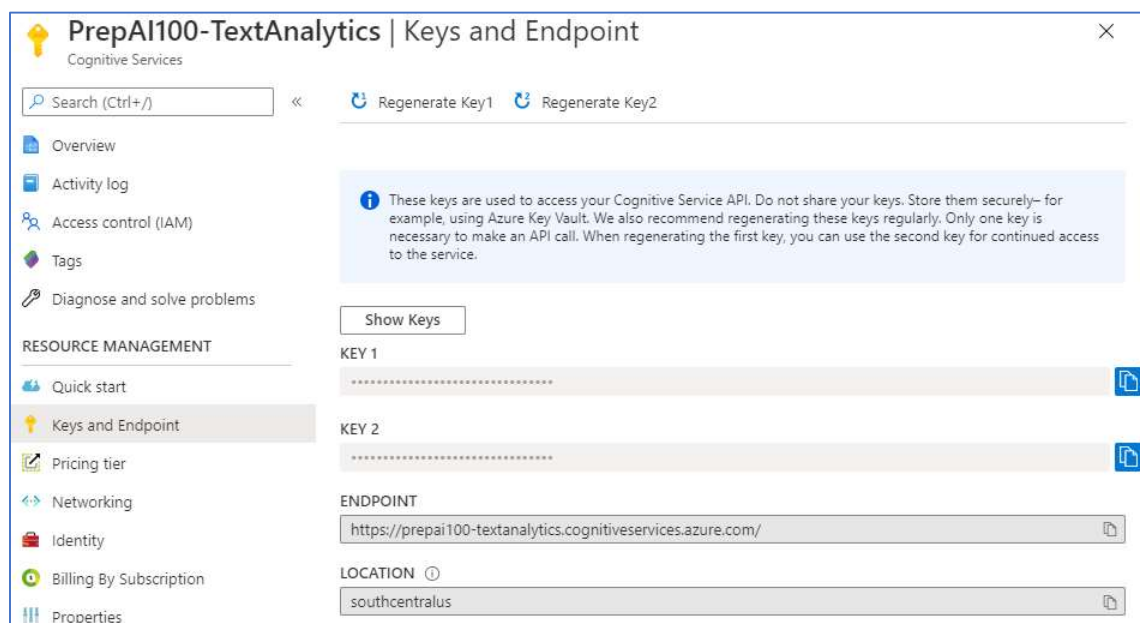
Get the access key

Now that we have our Cognitive Services account, let's find the access key so we can start calling the API.

1. Click on the **Go to resource** button on the *Deployment succeeded* notification. This action opens the account Quickstart.



2. Select the **Keys and Endpoint** menu item from the menu on the left, or in the *Grab your keys* section of the quickstart. This action opens the **Manage keys** page.
3. Copy one of the keys using the copy button.



Important

Always keep your access keys safe and never share them.

4. Store this key for the rest of this module. We'll use it shortly to make API calls from the testing console and throughout the rest of the module.

Call the API from the testing console

Now that we have our key, we can head over to the testing console and take the API for a spin.

1. Navigate to the following URL in your favorite browser.
Replace [location] with the location you selected when creating the Text Analytics cognitive services account earlier in this unit. For example, if you created the account in *eastus*, you'd replace [location] with *eastus* in the URL.

Bash
<code>https://[location].dev.cognitive.microsoft.com/docs/services/TextAnalytics.V2.0</code>

In my case:
<code>https://southcentralus.dev.cognitive.microsoft.com/docs/services/TextAnalytics.V2.0/</code>

The landing page displays a menu on the left and content to the right. The menu lists the POST methods you can call on the Text Analytics API. These endpoints are **Detect Language**, **Entities**, **Key Phrases**, and **Sentiment**. To call one of these operations, we need to do a few things.

- Select the method we want to call.
- Add the access key that we saved earlier in the lesson to each call.

The screenshot shows the Microsoft Cognitive Services Text Analytics API (v2.0) documentation page. The left sidebar contains a menu with the following items: 'POST Detect Language', 'POST Entities', 'POST Key Phrases', and 'POST Sentiment'. The 'Sentiment' item is selected. The main content area is titled 'Text Analytics API (v2.0)' and contains the following text: 'The Text Analytics API is a suite of text analytics web services built with best-in-class Microsoft machine learning models to analyze unstructured text for tasks such as sentiment analysis, key phrase extraction and language detection. This API uses advanced natural language processing techniques to deliver best in class predictions. Further documentation can be found in <https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/>. This API is currently available in:

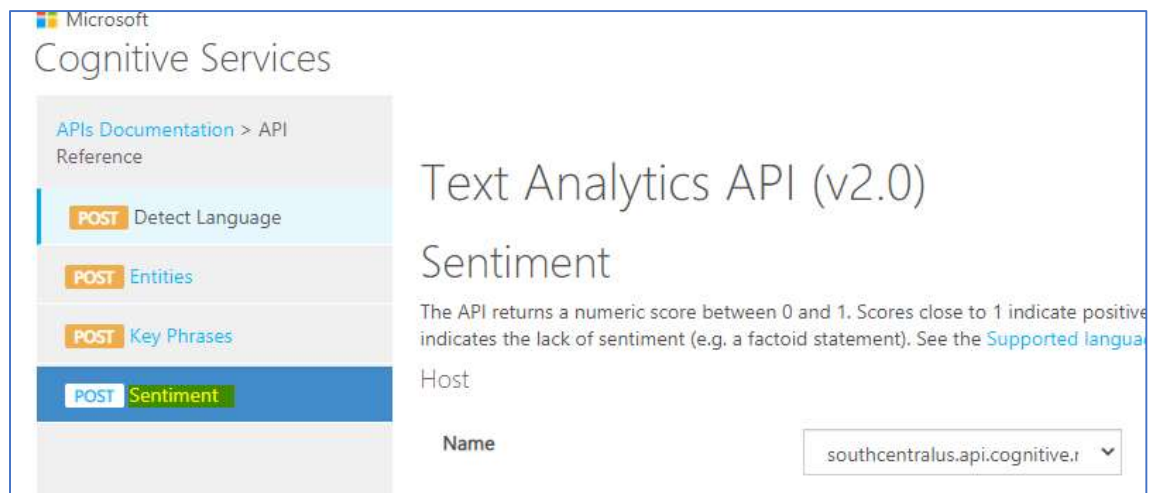
- Australia East - australiaeast.api.cognitive.microsoft.com
- Brazil South - brazilsouth.api.cognitive.microsoft.com
- Canada Central - canadacentral.api.cognitive.microsoft.com
- Central India - centralindia.api.cognitive.microsoft.com
- Central US - centralus.api.cognitive.microsoft.com
- East Asia - eastasia.api.cognitive.microsoft.com
- East US - eastus.api.cognitive.microsoft.com
- East US 2 - eastus2.api.cognitive.microsoft.com
- France Central - francecentral.api.cognitive.microsoft.com
- Japan East - japaneast.api.cognitive.microsoft.com
- Japan West - japanwest.api.cognitive.microsoft.com
- Korea Central - koreacentral.api.cognitive.microsoft.com
- North Central US - northcentralus.api.cognitive.microsoft.com
- North Europe - northeurope.api.cognitive.microsoft.com
- South Africa North - southafricanorth.api.cognitive.microsoft.com
- South Central US - southcentralus.api.cognitive.microsoft.com
- Southeast Asia - southeastasia.api.cognitive.microsoft.com
- UK South - uksouth.api.cognitive.microsoft.com
- West Central US - westcentralus.api.cognitive.microsoft.com
- West Europe - westeurope.api.cognitive.microsoft.com
- West US - westus.api.cognitive.microsoft.com
- West US 2 - westus2.api.cognitive.microsoft.com

2. From the left menu, select **Sentiment**. This selection opens the Sentiment documentation to the right. As the documentation shows, we'll be making a REST call in the following format.

```
https://[location].api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
```

[location] is replaced with the location that you selected when you created the Text Analytics account.

We'll pass in our subscription key, or access key, in the **ocp-Apim-Subscription-Key** header.



Make some API calls

1. In the **Open API testing console** section of this page, select the button corresponding to the location or region in which you created the Cognitive Services account. Selecting this button opens the live, interactive, API console.
2. Paste the access key you saved earlier into the field labeled **Ocp-Apim-Subscription-Key**. Notice that the key is written automatically into the HTTP request window as a header value.
3. Scroll to the bottom of the page and click **Send**.

Let's examine the sections of this screen in more detail.

In the Headers section of the user interface, we set the access, or subscription, key in the header of our request.



Next, we have the request body section, which holds a **documents** array. Each document in the array has three properties. The properties are **"language"**, **"id"**, and **"text"**. The **"id"** is a number in this example, but it can be anything you want as long as it's unique in the documents array. In this example, we're also passing in documents written in three different languages. Over 15 languages are supported in the Sentiment feature of the Text Analytics API. For more information, check out [Supported languages in the Text Analytics API](#). The maximum size of a single

document is 5,000 characters, and one request can have up to 1,000 documents.

Request body

Collection of documents to analyze

```
1 {
2   "documents": [
3     {
4       "language": "en",
5       "id": "1",
6       "text": "Hello world. This is some input text that I love."
7     },
```

The complete request, including the headers and the request URL are displayed in the next section. In this example, you can see that the requests are routed to a URL that begins with **southcentralus**. The URL differs depending on the location you selected when you created your Text Analytics account.

Request URL

```
https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
```

HTTP request

```
POST https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment HTTP/1.1
Host: southcentralus.api.cognitive.microsoft.com
Content-Type: application/json
Ocp-Apim-Subscription-Key: .....
```

```
{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "Hello world. This is some input text that I love."
    },
    {
      "language": "fr",
      "id": "2",
      "text": "Bonjour tout le monde"
    },
    {
      "language": "es",
      "id": "3",
      "text": "La carretera estaba atascada. Había mucho tráfico el día de ayer."
    }
  ]
}
```

Send

Then we have information about the response. In the example, we see that the request was a success and code 200 was returned.

Response status

200 OK

Response latency

190 ms

Finally, we see the response to our request. The response holds the insight the Text Analytics API had about our documents. An array of documents is returned to us, without the original text. We get back an *"id"* and *"score"* for each document. The API returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment, while scores close to 0 indicate negative sentiment. A score of 0.5 indicates the lack of sentiment, a neutral statement. In this example, we have two pretty positive documents and one negative document.

Response content

```
Transfer-Encoding: chunked
csp-billing-usage: CognitiveServices.TextAnalytics.BatchScoring=3
x-envoy-upstream-service-time: 105
apim-request-id: 8ce9b2b1-25d2-466d-892a-61dd04a2747a
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
x-content-type-options: nosniff
Date: Mon, 13 Jul 2020 01:20:04 GMT
Content-Type: application/json; charset=utf-8

{
  "documents": [{
    "id": "1",
    "score": 0.98690706491470337
  }, {
    "id": "2",
    "score": 0.84012651443481445
  }, {
    "id": "3",
    "score": 0.334433376789093
  }],
  "errors": []
}
```

Congratulations! You've made your first call to the Text Analytics API without writing a line of code. Feel free to stay in the console and try out more calls. Here are some suggestions:

- Change the documents in section number 2 and see what the API returns.
- Try the other methods, **Detect Language**, **Entities**, and **Key Phrases**, using the same subscription key.
- Try to make a call from a different region with your subscription and observe what happens.

HTTP request

```
POST https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment HTTP/1.1
Host: southcentralus.api.cognitive.microsoft.com
Content-Type: application/json
Ocp-Apim-Subscription-Key: .....

{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "Have a nice day."
    },
    {
      "language": "fr",
      "id": "2",
      "text": "Ça craint"
    },
    {
      "language": "pt-PT",
      "id": "3",
      "text": "Estou a ficar muito feliz!"
    }
  ]
}
```

Response content

```
Transfer-Encoding: chunked
csp-billing-usage: CognitiveServices.TextAnalytics.BatchScoring=3
x-envoy-upstream-service-time: 53
apim-request-id: 99d1b984-7ece-47db-8999-d0719de54861
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
x-content-type-options: nosniff
Date: Mon, 13 Jul 2020 01:24:09 GMT
Content-Type: application/json; charset=utf-8

{
  "documents": [{
    "id": "1",
    "score": 0.99487078189849854
  }, {
    "id": "2",
    "score": 0.28429701924324036
  }, {
    "id": "3",
    "score": 1.0
  }],
  "errors": []
}
```


The API testing console is a great way to explore the capabilities of this API. Now that you've explored for yourself, let's move on to putting this intelligence into a real-world scenario.

Detect Language

Microsoft

Cognitive Services

APIs Documentation > API Reference

POST Detect Language

POST Entities

POST Key Phrases

POST Sentiment

Text Analytics API (v2.1)

Detect Language

The API returns the detected language and a numeric score between 0 and 1. Scores close to 1 indicate 100% confidence. 120 languages are supported.

Host

Name

southcentralus.api.cognitive.microsoft.com

Query parameters

showStats

Value

Remove parameter

Add parameter

Headers

Content-Type

application/json

Remove header

Ocp-Apim-Subscription-Key

.....

Request URL

https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.1/languages

HTTP request

POST https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.1/languages HTTP/1.1
Host: southcentralus.api.cognitive.microsoft.com
Content-Type: application/json
Ocp-Apim-Subscription-Key:

```
{
  "documents": [
    {
      "id": "1",
      "text": "Hello world! Beautiful day"
    },
    {
      "id": "2",
      "text": "Qual é a sua mano."
    },
    {
      "id": "3",
      "text": "La carretera estaba atascada. Había mucho tráfico el día de ayer."
    },
    {
      "id": "4",
      "text": ":) :( :D"
    }
  ]
}
```

Response status

200 OK

Response latency

44 ms

Response content

```
Transfer-Encoding: chunked
csp-billing-usage: CognitiveServices.TextAnalytics.BatchScoring=4
x-envoy-upstream-service-time: 4
apim-request-id: 5ef8018c-7957-4394-8c95-0adc53868262
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
x-content-type-options: nosniff
Date: Mon, 13 Jul 2020 01:32:12 GMT
Content-Type: application/json; charset=utf-8
```

```
{
  "documents": [{
    "id": "1",
    "detectedLanguages": [{
      "name": "English",
      "iso6391Name": "en",
      "score": 1.0
    }]
  }, {
    "id": "2",
    "detectedLanguages": [{
      "name": "Portuguese",
      "iso6391Name": "pt",
      "score": 1.0
    }]
  }, {
    "id": "3",
    "detectedLanguages": [{
      "name": "Spanish",
      "iso6391Name": "es",
      "score": 1.0
    }]
  }, {
    "id": "4",
    "detectedLanguages": [{
      "name": "(Unknown)",
      "iso6391Name": "(Unknown)",
      "score": 0.0
    }]
  }],
  "errors": []
}
```


Request URL

`https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.1/entities`

HTTP request

POST `https://southcentralus.api.cognitive.microsoft.com/text/analytics/v2.1/entities` HTTP/1.1

Host: `southcentralus.api.cognitive.microsoft.com`

Content-Type: `application/json`

Ocp-Apim-Subscription-Key: `*****`

```
{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "Azure Portal"
    },
    {
      "language": "fr",
      "id": "2",
      "text": "Bonjour"
    },
    {
      "language": "es",
      "id": "3",
      "text": "Mexico"
    }
  ]
}
```

Response content

Transfer-Encoding: chunked
x-ms-transaction-count: 3
CSP-Billing-Usage: CognitiveServices.TextAnalytics.BatchScoring|3
x-aml-ta-request-id: 9428ab4c-16e8-4aaf-9808-5b2d53bad543
X-Content-Type-Options: nosniff
apim-request-id: 1af3c29f-6fae-4308-87a0-652b808c4965
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Date: Mon, 13 Jul 2020 01:41:15 GMT
Content-Type: application/json; charset=utf-8

```
{
  "documents": [{
    "id": "1",
    "entities": [{
      "name": "Azure Portal",
      "matches": [{
        "entityTypeScore": 0.9980621337890625,
        "text": "Azure Portal",
        "offset": 0,
        "length": 12
      }],
      "type": "Organization"
    }]
  }, {
    "id": "2",
    "entities": []
  }, {
    "id": "3",
    "entities": [{
      "name": "Mexico",
      "matches": [{
        "entityTypeScore": 0.60327303409576416,
        "text": "Mexico",
        "offset": 0,
        "length": 6
      }],
      "type": "Person"
    }]
  }],
  "errors": []
}
```

Design a feedback sorter

Let's put our knowledge of Text Analytics to work in a practical solution. Our solution will focus on Sentiment Analysis of text documents. Let's set the context by describing the problem we want to tackle.

Manage customer feedback more efficiently

Social media is active with talk of your company's product. Your feedback email alias is also active with customers eager to share their opinion of your product.

As is the case with any new startup, you live by the mantra of listening to your customers. However, the success of your product has made keeping this promise easier said than done. It's a good problem but a problem all the same.

The team can't keep up with the volume of feedback anymore. They need help sorting the feedback so that issues can be managed as efficiently as possible. As the lead developer in the organization, you have been asked to build a solution.

Requirement	Details
Categorize feedback so we can react to it.	<p>Not all feedback is equal. Some is glowing testimony. Other feedback is scathing criticism from a frustrated customer. Perhaps you can't tell what the customer wants in other cases.</p> <p>At a minimum, having an indication of the sentiment, or tone, of feedback would help us categorize it.</p>
The solution should scale up or down to meet demand.	<p>We're a startup. Fixed costs are difficult to justify, and we haven't figured out the exact pattern of feedback traffic. We'll need a solution that can tackle bursts of activity, but cost as little as possible during quiet times.</p> <p>A serverless architecture billed on a consumption plan is a good candidate in this case.</p>
Produce a Minimal Viable Product (MVP), but make the solution adaptable.	<p>Today, we want to categorize feedback so we can apply our limited resources to the feedback that matters. If a customer is frustrated, we want to know immediately and start chatting with them. In the future, we'll enhance this solution to do more. One idea for a new feature is to examine key phrases in feedback to detect pain points before they reach critical mass with our customers. Another idea is to automate</p>

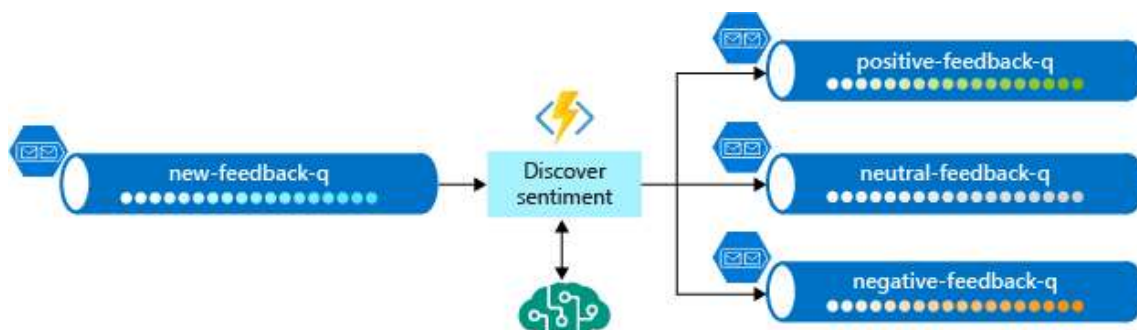
	<p>responses back to customers who are either positive or neutral. Even though they love us, we want them to know we are still listening to their feedback.</p> <p>A solution that offers a plug-and-play architecture is a good fit here. We could, for example, use queues as a form of factory line. You perform one task, then place the result into a queue for the next part of the system to pick it up and process.</p>
Deliver quickly.	<p>We've all heard this one before! Remember, this solution is an MVP, and we want to test it with our scenario quickly. To deliver at speed and with quality will mean writing less code.</p> <p>Taking advantage of the Text Analytics API means we don't have to train a model to detect sentiment. Using Azure Functions and binding to queues declaratively reduces the amount of code we have to write. A serverless solution also means we don't have to worry about server management.</p>

Let's look at some high-level requirements:

Our proposed solution for each requirement in the preceding table offers a glimpse into how to map requirements to solutions. Let's now see what a solution might look like based on Azure.

A solution based on Azure Functions, Azure Queue Storage, and Text Analytics API

The following diagram is a design proposal for a solution. It uses three core components of Azure: Azure Queue Storage, Azure Functions, and Azure Cognitive Services.

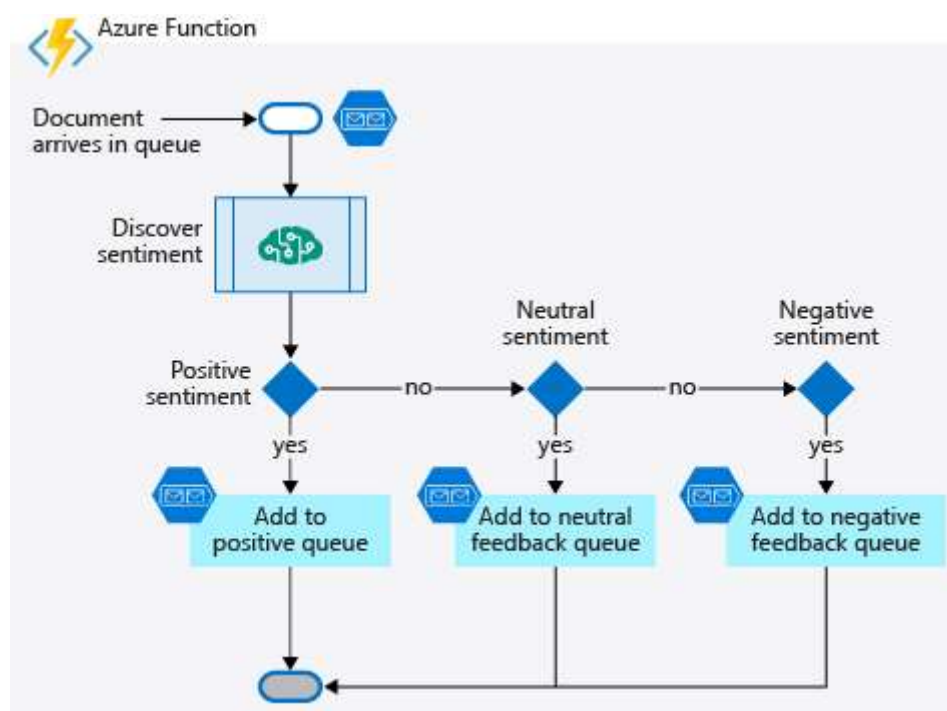


The idea is that text documents containing user feedback are placed into a queue that we've named *new-feedback-q* in the preceding diagram. The arrival of a message containing the text document into the queue will trigger, or start,

function execution. The function reads messages containing new documents from the input queue and sends them for analysis to the Text Analytics API. Based on the results that the API returns, a new message containing the document is placed into an output queue for further processing.

The result we get back for each document is a sentiment score. The output queues are used to store feedback sorted into positive, neutral, and negative. Hopefully, the negative queue will always be empty! :-) Once we've bucketed each incoming piece of feedback into an output queue based on sentiment, you can imagine adding logic to take action on the messages in each queue.

Let's look at a flowchart next to see what the function logic needs to do.



Our logic is like a router. It takes text input and routes it to an output queue based on the sentiment score of the text. We have a dependency on Text Analytics API. While the logic seems trivial, this function will remove the need for people on the team to analyze feedback manually.

Steps to implement our solution

To implement the solution described in this unit, we'll need to complete the following steps.

1. Create a function app to host our solution.

2. Look for sentiment in incoming feedback messages using the Text Analytics API. We'll use our access key from the preceding exercise and write some code to send the requests.
3. Post feedback to processing queues based on sentiment.

Let's move on to creating our function app.

Exercise - Create a function app from the portal to host our business logic

Sandbox activated! Time remaining:

3 hr 10 min

You have used 1 of 10 sandboxes for today. More sandboxes will be available tomorrow.

A function app provides a context for managing and executing your functions. Let's create a function app and then add a function to it.

Create a Function App to host our function

1. Sign into the [Azure portal](#) using the same account you activated the sandbox with.
2. On the Azure portal menu or from the **Home** page, select **Create a resource**, and then select **Compute > Function App**.

Microsoft Azure

Search resources, services, and docs (G+/)

Home >

New

Search the Marketplace

Azure Marketplace

See all

Get started

Recently created

AI + Machine Learning

Analytics

Blockchain

Compute

Containers

Databases

Developer Tools

DevOps

Identity

Integration


Internet of Things

IT & Management Tools


Media

Featured


See all

Virtual machine


[Learn more](#)

SQL Server 2017 Enterprise Windows Server 2016


[Learn more](#)

Reserved VM Instances


[Quickstarts + tutorials](#)

Kubernetes Service


[Quickstarts + tutorials](#)

Service Fabric Cluster

[Quickstarts + tutorials](#)

Web App for Containers

[Quickstarts + tutorials](#)

Function App

[Quickstarts + tutorials](#)

Setting	Value	Description
App name	Globally unique name	Name that identifies your new function app. Valid characters are a-z, 0-9, and -.
Subscription	Concierge Subscription	The subscription under which this new function app is created.
Resource group	learn-43e82f9f-9590-45a4-96e5-f7bbe7cc88bf	<p>Name for the resource group in which to create your function app.</p> <p>Make sure to select Use existing and use the resource group from the last exercise. That way, all the resources we make in this module are kept together.</p>
OS	Windows	The operating system that hosts the function app.
Hosting Plan	Consumption plan	Hosting plan that defines how resources are allocated to your function app. In the default Consumption Plan , resources are added dynamically as required by your functions. In this serverless hosting, you only pay for the time your functions run.
Location	Select the same location you used earlier.	<p>Choose a region near you or near other services your functions access.</p> <p>Select the same region that you used when creating the Text Analytics API account in the last exercise.</p>
Runtime Stack	Node.js	The sample code in this module is written in JavaScript.
Storage	Globally unique name	Name of the new storage account used by your function app. Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only. This dialog populates the field with a unique name that is derived from the name you gave the app. However, feel free to use a different name or even an existing account.

[Home](#) > [New](#) >

Create Function App

Basics

[Hosting](#)

[Monitoring](#)

[Tags](#)

[Review + create](#)

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Concierge Subscription

Resource Group * ⓘ

learn-43e82f9f-9590-45a4-96e5-f7bbe7cc88bf

[Create new](#)

Instance Details

Function App name *

douglasluna

.azurewebsites.net

Publish *



Code



Docker Container

Runtime stack *

Node.js

Version *

12

Region *

Brazil South

[Home](#) > [New](#) >

Create Function App

Basics

[Hosting](#)

[Monitoring](#)

[Tags](#)

[Review + create](#)

Storage

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage.

Storage account *

(New) storageaccountlearn94d2

[Create new](#)

Operating system

The Operating System has been recommended for you based on your selection of runtime stack.

Operating System *



Linux



Windows

Plan

The plan you choose dictates how your app scales, what features are enabled, and how it is priced. [Learn more](#) [↗](#)

Plan type * ⓘ

Consumption (Serverless)


3. Enter the function app settings as specified in the following table.
4. Select **Create** to provision and deploy the function app.

[Home](#) > [New](#) >

Create Function App

[Basics](#) [Hosting](#) [Monitoring](#) [Tags](#) [Review + create](#)

Summary

 **Function App**
by Microsoft

Details

Subscription	c42268a6-fac4-4638-b014-76ff9880e070
Resource Group	learn-43e82f9f-9590-45a4-96e5-f7bbe7cc88bf
Name	douglasluna
Runtime stack	Node.js - 12

Hosting

Storage (New)

Storage account	storageaccountlearnb7df
-----------------	-------------------------

Plan (New)

Plan type	Consumption (Serverless)
Name	ASP-learn43e82f9f959045a496e5f7bbe7-93a7
Operating System	Windows
Region	South Central US
SKU	Dynamic

Monitoring (New)

Application Insights	Enabled
Name	douglasluna
Region	South Central US

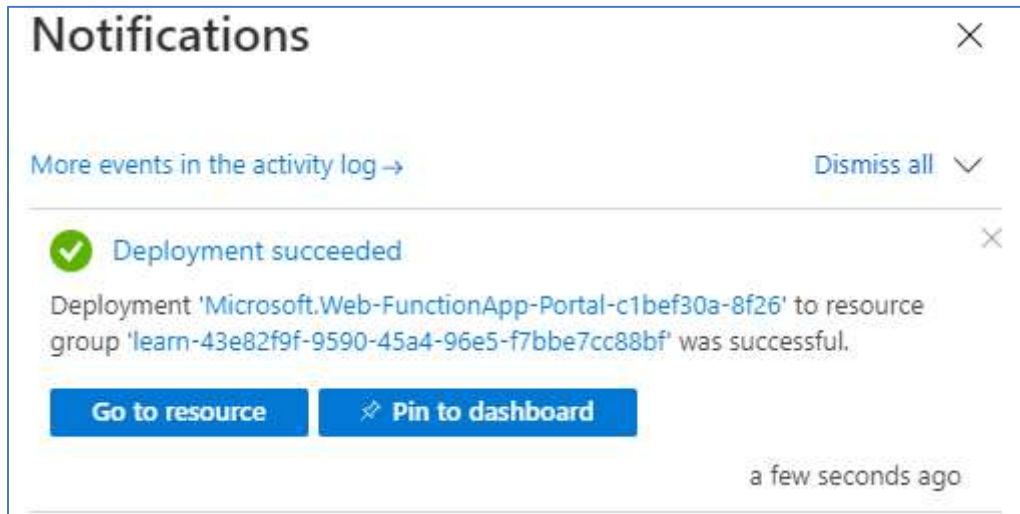
Create

< Previous

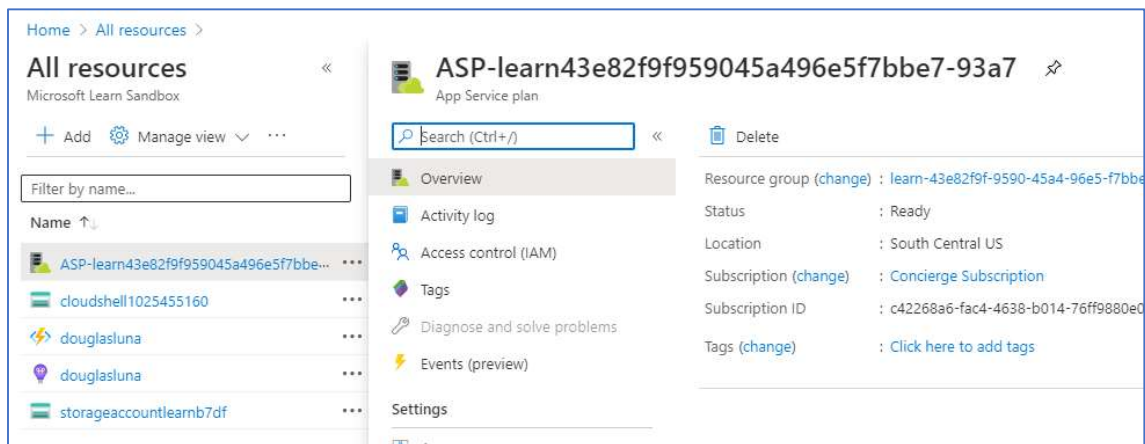
Next >

[Download a template for automation](#)

5. Select the Notification icon in the upper-right corner of the portal, and watch for a **Deployment in progress** message.
6. Deployment can take some time. So, stay in the notification hub and watch for a **Deployment succeeded** message.



7. Once the function app is deployed, go to **All resources** in the portal. The function app will be listed with type **App Service** and has the name you gave it. Select the function app in the list, to open it.



Congratulations! You've created and deployed your function app.

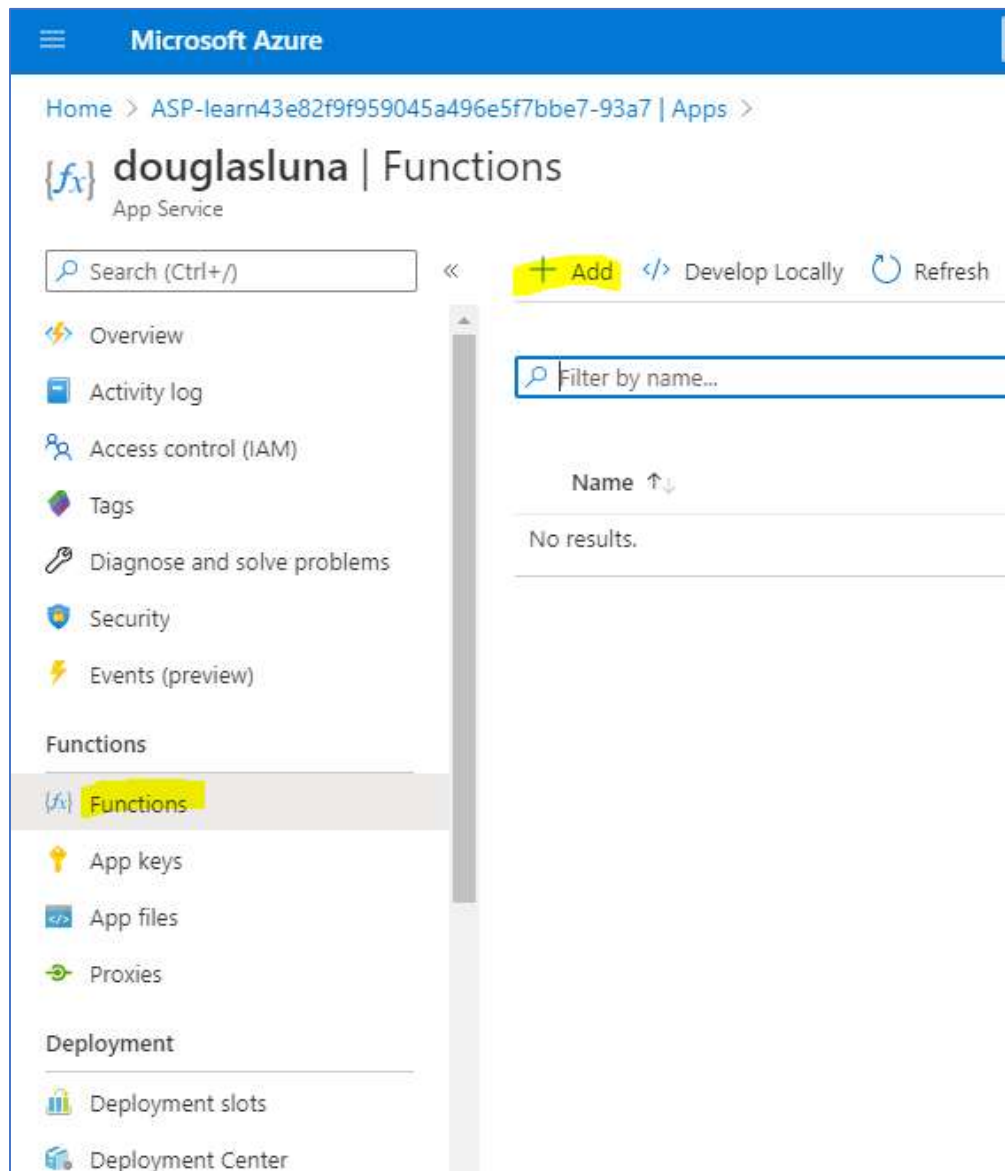
Tip

Having trouble finding your function apps in the portal? Try [adding Function Apps to your favorites in the Azure portal](#).

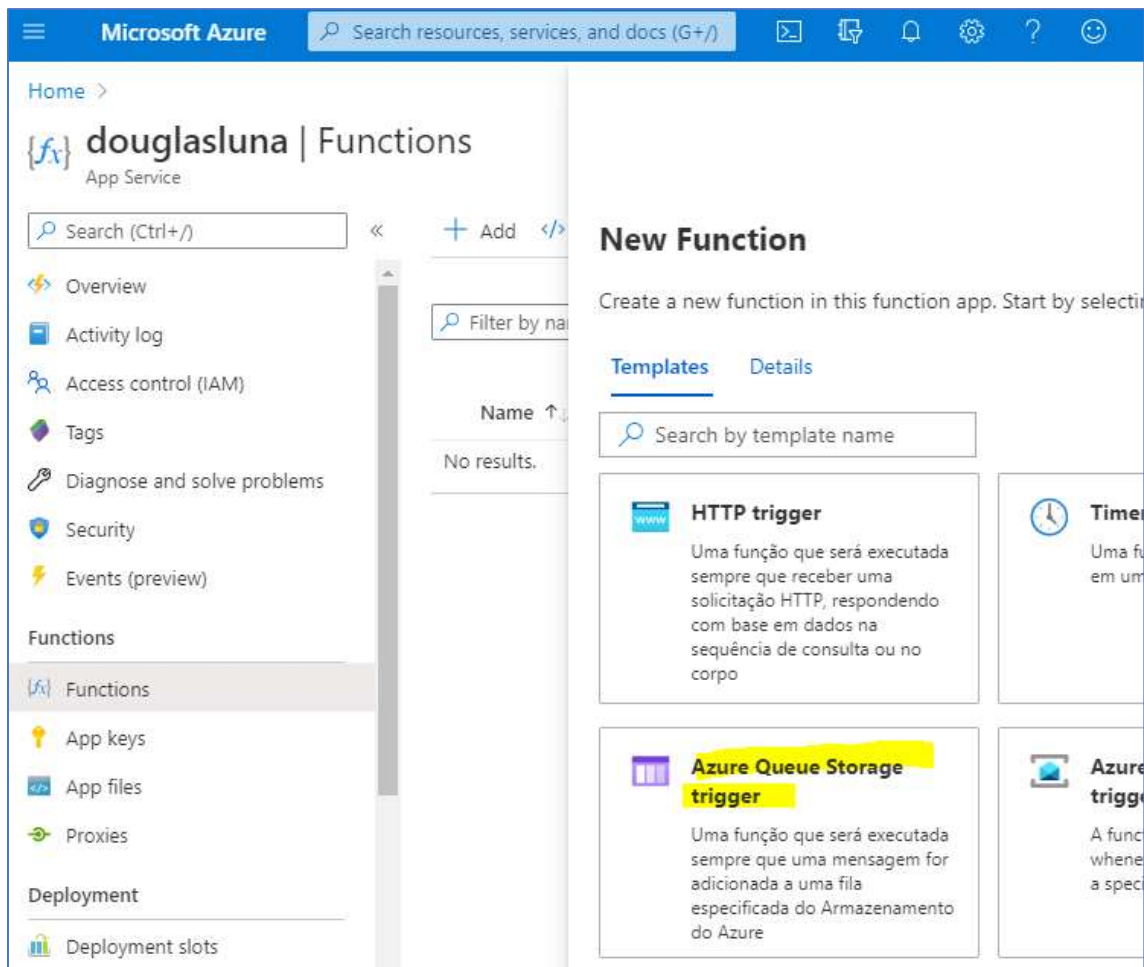
Create a function to execute our logic

Now that we have a function app, it's time to create a function. A function is activated through a trigger. In this module, we'll use a Queue trigger. The runtime will poll a queue and start this function to process a new message.

1. Select the Add (+) button next to **Functions**. This action starts the function creation process.



2. On the **Azure Functions for JavaScript - getting started** page, select **Import** and then select **continue**.
3. In the **Create a function** step, select **More templates...** and then select **Finish and view templates**.
4. In the list of all templates available to this function app, select **Azure Queue Storage trigger**.



5. If you see a message saying **Extensions not installed**, select **Install**.
Dependency installation can take a couple of minutes. Please wait until the installation completes before continuing.
6. In the **New Function** dialog that appears, enter the following values.

Property	Value
Name	discover-sentiment-function
Queue name	new-feedback-q
Storage account connection	AzureWebJobsStorage

7. Select **Create** to begin the function creation process.

New Function

Create a new function in this function app. Start by

[Templates](#)[Details](#)

New Function *

discover-sentiment-function

Nome da fila * ⓘ

new-feedback-q

Conexão da conta de armazenamento * ⓘ

AzureWebJobsStorage

New

Create Function

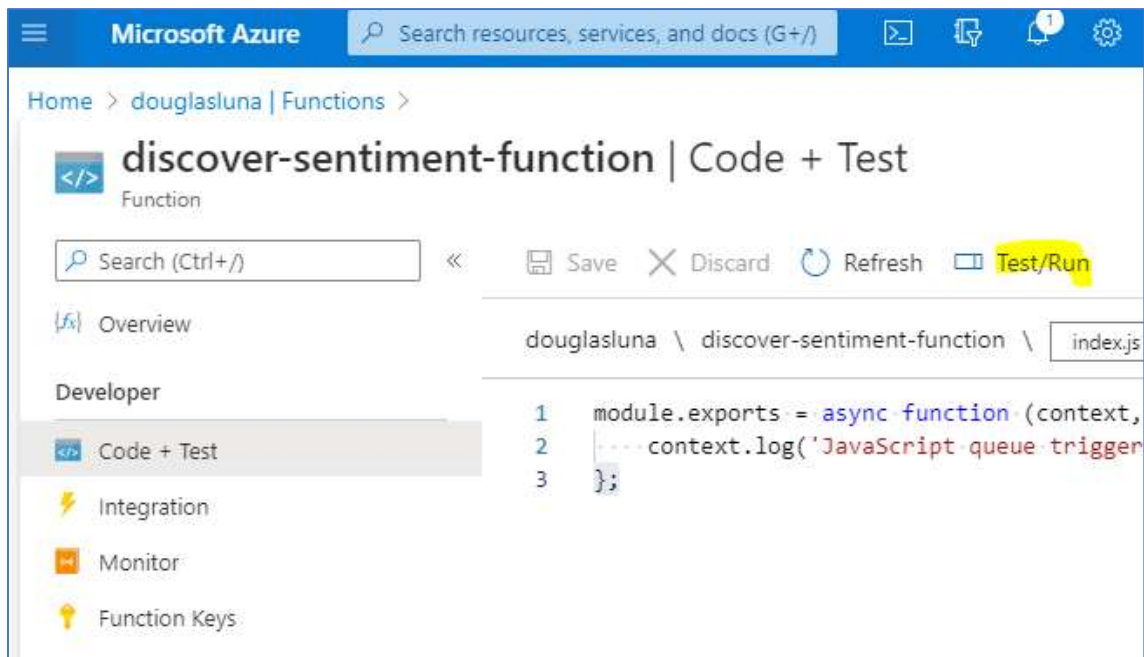
8. A function is created in your chosen language using the Queue Trigger function template. While we'll implement the function in JavaScript in this module, you can create a function in any [supported language](#).

When the create process is complete, the code editor opens in the portal and loads the *index.js* page. This file is the code file where we write our function logic.

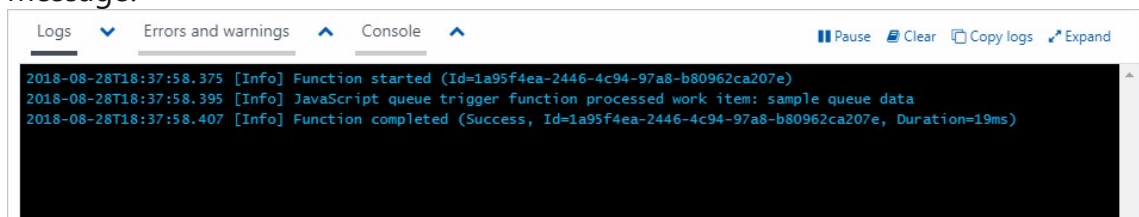
Try it out

Let's test what we have so far. We haven't written any code yet, so this test is to make sure what we've configured so far, runs.

1. Click **Test/Run** at the top of the code editor.



2. Observe the **Logs** tab that opens at the bottom of the screen. If everything works as planned, you'll see a message similar to the following message.



The **Run** button started our function and passed *sample queue data*, the default text from the **Test** request window to our function.

Tip

If the function times out or does not return successfully, try restarting the functions app. Select your functions app in the menu on the left and then select **Restart** from the *Overview* panel. Wait for the functions app to restart and then try running your function again.

Nice work! You've successfully added a Queue-triggered function to your function app and tested to make sure it's working as expected! We'll add more functionality to the function in the next exercise.

Let's look briefly at the function's other file, the *function.json* config file. The configuration data from this file is shown in the following JSON listing.

JSON

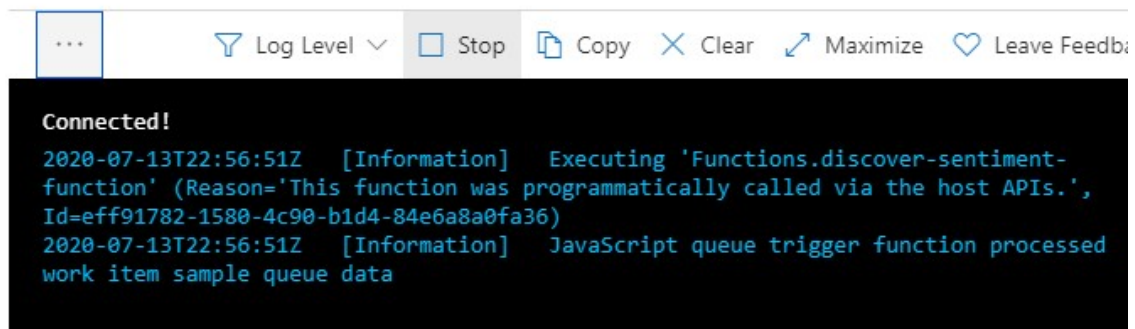
```

{
  "bindings": [
    {
      "name": "myQueueItem",
      "type": "queueTrigger",
      "direction": "in",
      "queueName": "new-feedback-q",
      "connection": "AzureWebJobsDashboard"
    }
  ],
  "disabled": false
}

```



As you can see, this function has a trigger binding named **myQueueItem** of type `queueTrigger`. When a new message arrives in the queue we've named **new-feedback-q**, our function is called. We reference the new message through the `myQueueItem` binding parameter. Bindings really do take care of some of the heavy lifting for us!



In the next step, we'll add code to call the Text Analytics API service.

Tip

You can see `index.js` and `function.json` by expanding the **View Files** menu on the right of the function panel in the Azure portal.

This exercise was all about getting our Azure Functions infrastructure in place. We have a working function hosted in a function app that runs when a new message arrives in our queue that we've named **new-feedback-q**. The real fun begins in the next exercise when we add code to call an Azure Cognitive Service to do sentiment analysis.

Exercise - Call the Text Analytics API from an app

Sandbox activated! Time remaining:
3 hr 15 min

You have used 2 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Let's update our function implementation to call the Text Analytics API service and get back a sentiment score.

1. Select our function, **discover-sentiment-function**, in our function app in the portal.
2. Expand the **View files** menu on the right of the screen.
3. Under the **View files** tab, select **index.js** to open the code file in the editor.
4. Replace the entire content of **index.js** with the following JavaScript and **Save**.

JavaScript

```
module.exports = function (context, myQueueItem) {  
    context.log('Processing queue message', myQueueItem);  
  
    let https = require('https');  
  
    // Replace the accessKey string value with your valid access key.  
    let accessKey = '<YOUR ACCESS KEY HERE>';
```

```

// Replace [region], including square brackets, in the uri variable below.
// You must use the same region in your REST API call as you used to obtain your
access keys.
// For example, if you obtained your access keys from the northeurope region, replace
// "westus" in the URI below with "northeurope".
let uri = '[region].api.cognitive.microsoft.com';
let path = '/text/analytics/v2.0/sentiment';

let response_handler = function (response) {
    let body = "";

    response.on('data', function (chunk) {
        body += chunk;
    });

    response.on('end', function () {
        let body_ = JSON.parse (body);
        let body__ = JSON.stringify (body_, null, ' ');
        context.log (body__);
        context.done();
        return;
    });

    response.on('error', function (e) {
        context.log ('Error: ' + e.message);
        context.done();
        return;
    });
};

let get_sentiments = function (documents) {
    let body = JSON.stringify (documents);

    let request_params = {
        method : 'POST',
        hostname : uri,
        path : path,
        headers : {
            'Ocp-Apim-Subscription-Key' : accessKey,
        }
    };

    let req = https.request (request_params, response_handler);
    req.write (body);
    req.end ();
}

// Create a documents array with one entry.

```

```

let documents = { 'documents': [
  {
    'id': '1',
    'language': 'en',
    'text': myQueueItem
  },
]};

get_sentiments (documents);

};

```

The screenshot shows the Azure Functions 'Code + Test' interface. The breadcrumb navigation at the top reads 'Home > douglasluna | Functions >'. The function name 'discover-sentiment-function' is displayed, along with the file 'index.js'. The left sidebar shows the 'Developer' section with 'Code + Test' selected. The main editor area displays the following JavaScript code:

```

1 module.exports = async function (context, myQueueItem) {
2   ...context.log('JavaScript queue trigger function processed work item', myQ
3 };module.exports = function (context, myQueueItem) {
4   ...context.log('Processing queue message', myQueueItem);
5
6   ...let https = require('https');
7
8   ...// Replace the accessKey string value with your valid access key.
9   ...let accessKey = '<YOUR ACCESS KEY HERE>';
10
11   ...// Replace [region], including square brackets, in the uri variable belo
12   ...// You must use the same region in your REST API call as you used to obt
13   ...// For example, if you obtained your access keys from the northeurope re
14   ...// "westus" in the URI below with "northeurope".
15   ...let uri = '[region].api.cognitive.microsoft.com';
16   ...let path = '/text/analytics/v2.0/sentiment';
17
18   ...let response_handler = function (response) {
19     ...let body = '';
20
21     ...response.on('data', function (chunk) {
22       ...body += chunk;
23     });
24
25     ...response.on('end', function () {
26       ...let body_ = JSON.parse (body);

```

5. Update the value of **accessKey** in the code you pasted with the access key for the Text Analytics API that you saved earlier in this module.
6. Update the **uri** value with the region from which you obtained your access key.

discover-sentiment-function | Code + Test

Function

Search (Ctrl+/)



Overview

douglasluna \ discover-sentiment-function \ index.js

Developer

Code + Test

Integration

Monitor

Function Keys

```

1 module.exports = async function (context, myQueueItem) {
2   ... context.log('JavaScript queue trigger function processed work item', myQueueItem);
3 }; module.exports = function (context, myQueueItem) {
4   ... context.log('Processing queue message', myQueueItem);
5
6   ... let https = require('https');
7
8   ... // Replace the accessKey string value with your valid access key.
9   ... let accessKey = 'REDACTED';
10
11   ... // Replace [region], including square brackets, in the uri variable below.
12   ... // You must use the same region in your REST API call as you used to obtain your access keys.
13   ... // For example, if you obtained your access keys from the northeurope region, replace
14   ... // "westus" in the URI below with "northeurope".
15   ... let uri = 'southcentralus.api.cognitive.microsoft.com';
16   ... let path = '/text/analytics/v2.0/sentiment';
17
18   ... let response_handler = function (response) {
19     ... let body = '';
20
21     ... response.on('data', function (chunk) {
22       ... body += chunk;
23     });
24
25     ... response.on('end', function () {
26       ... let body_ = JSON.parse(body);
27       ... let body__ = JSON.stringify(body_, null, ' ');

```

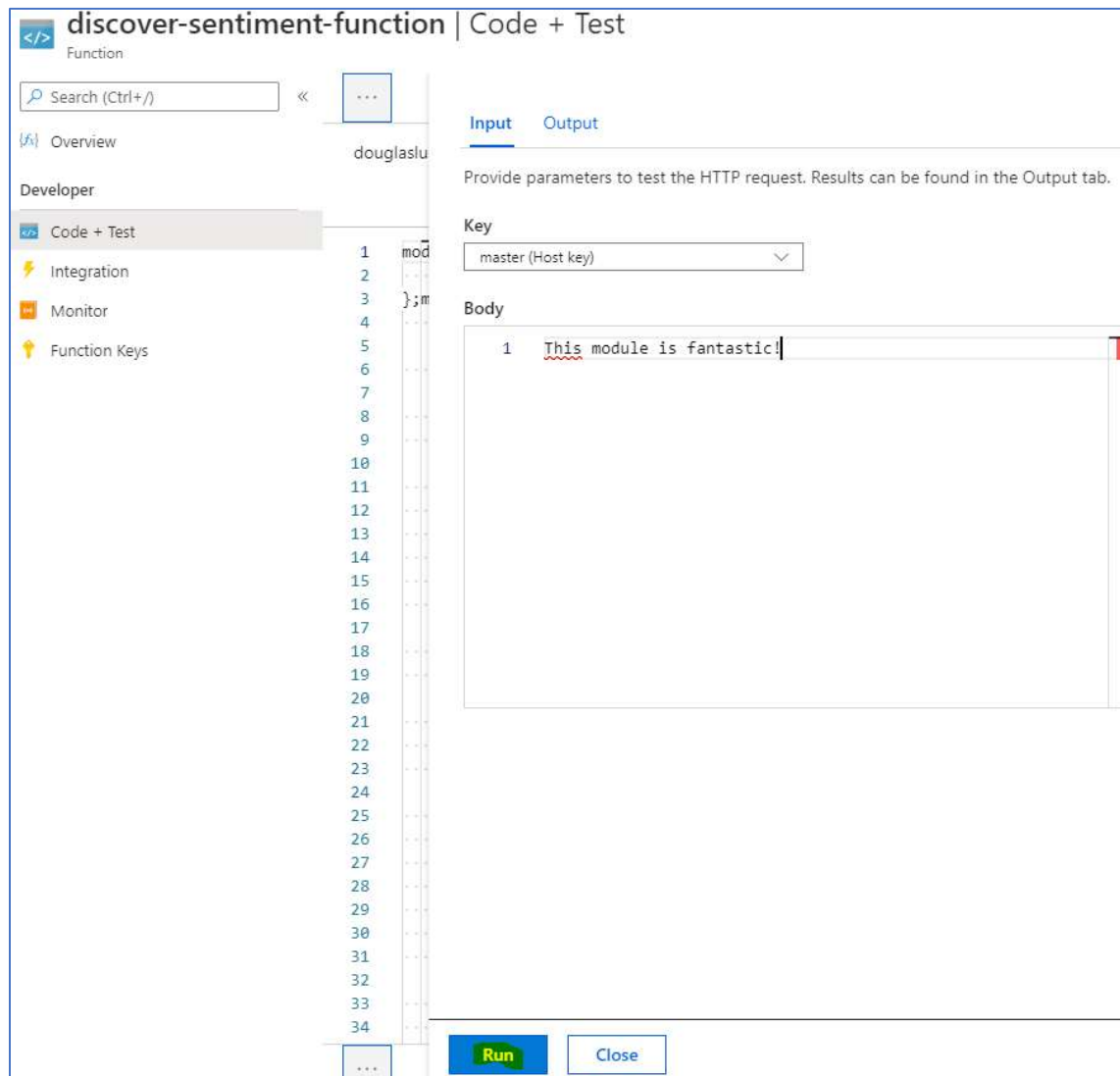
Let's look at what's happening in this code:

- Each call to the Text Analytics service, needs the access key, which we add as the **Ocp-Apim-Subscription-Key** header.
- Each call is made to a region-specific endpoint, defined by **uri** in our code.
- At the bottom of the code file, we've defined a **documents** array. This array is the payload we send to the Text Analytics service.
- The **documents** array has a single entry in this case, which is the queue message that triggered our function. Although we only have one document in our array, it doesn't mean that our solution can only handle one message at a time. The Azure Functions runtime retrieves and processes messages in batches, calling several instances of our function *in parallel*. Currently, the default batch size is 16 and the maximum batch size is 32.
- The **id** must be unique within the array. The **language** property specifies the language of the document text.
- We then call our method **get_sentiments**, which uses the HTTPS module to make the call to Text Analytics API. Notice that we pass our subscription, or access, key in the header of every request.
- When the service returns, our **response_handler** is called, and we log the response to the console using **context.log**

Try it out

Before we look at sorting into queues, let's take what we have for a test run.

1. With our function, **discover-sentiment-function**, selected in the Function Apps area of the portal, click on the **Test** menu item on the far right.
2. Select the **Test** menu item, and verify that you have the test panel open.
3. Add a string of text into the request body as shown in the screenshot.



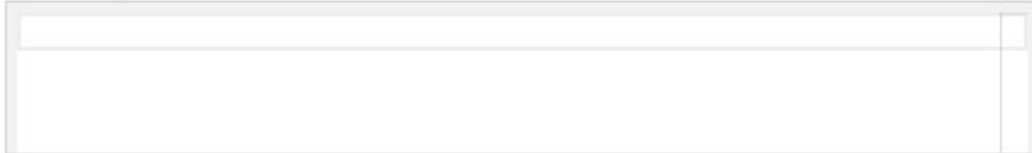
4. Click **Run** at the bottom of the test panel.

Input **Output**

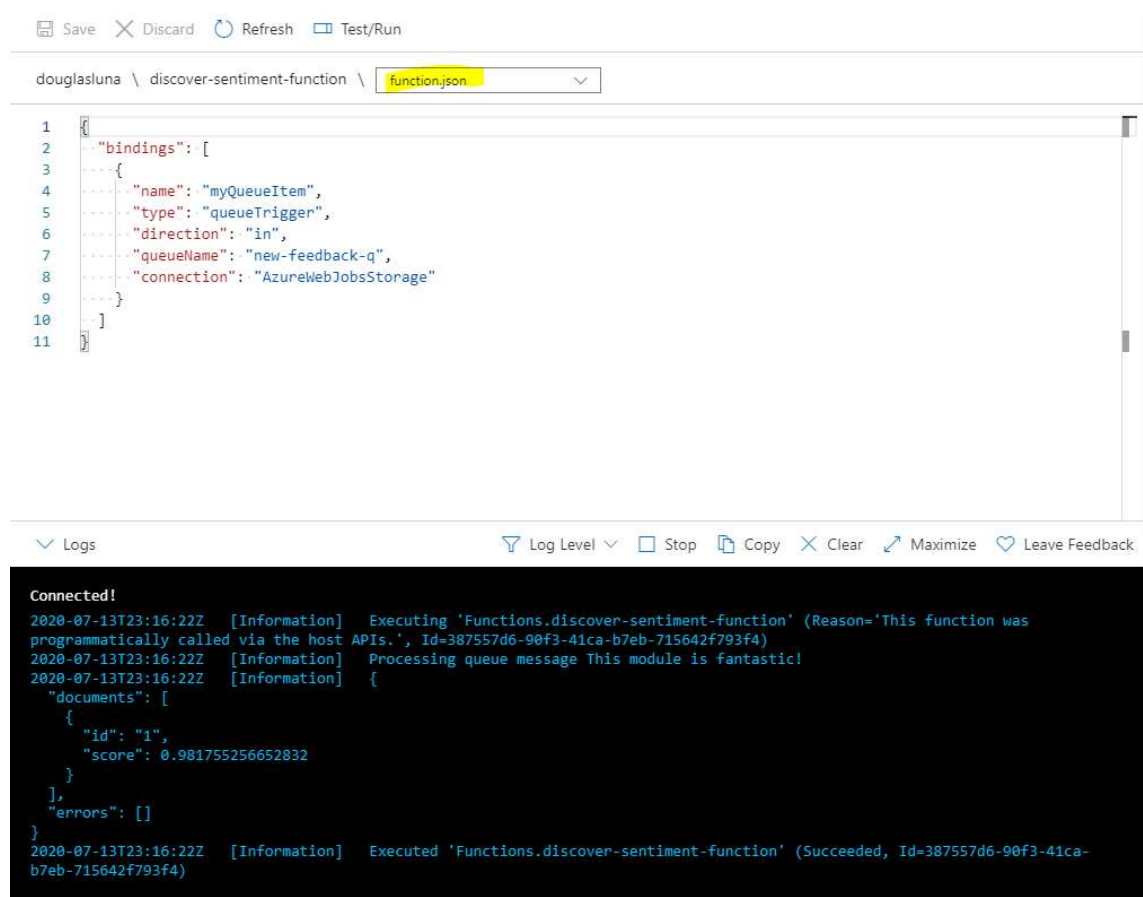
HTTP response code

202 Accepted

HTTP response content



5. Make sure the **Logs** tab is expanded at the bottom left of the main screen, under the code editor.
6. Verify that the **Logs** tab displays log information that the function completed. The window will also display the response from the Text Analytics API call.



The screenshot shows the Azure Functions portal interface. At the top, there are buttons for 'Save', 'Discard', 'Refresh', and 'Test/Run'. Below these is a breadcrumb trail: 'douglasluna \ discover-sentiment-function \ function.json'. The main area displays the 'function.json' file with the following content:

```
1 {
2   "bindings": [
3     {
4       "name": "myQueueItem",
5       "type": "queueTrigger",
6       "direction": "in",
7       "queueName": "new-feedback-q",
8       "connection": "AzureWebJobsStorage"
9     }
10  ]
11 }
```

At the bottom, the 'Logs' tab is expanded, showing a log window with the following content:

```
Connected!
2020-07-13T23:16:22Z [Information] Executing 'Functions.discover-sentiment-function' (Reason='This function was
programmatically called via the host APIs.', Id=387557d6-90f3-41ca-b7eb-715642f793f4)
2020-07-13T23:16:22Z [Information] Processing queue message This module is fantastic!
2020-07-13T23:16:22Z [Information] {
  "documents": [
    {
      "id": "1",
      "score": 0.981755256652832
    }
  ],
  "errors": []
}
2020-07-13T23:16:22Z [Information] Executed 'Functions.discover-sentiment-function' (Succeeded, Id=387557d6-90f3-41ca-
b7eb-715642f793f4)
```

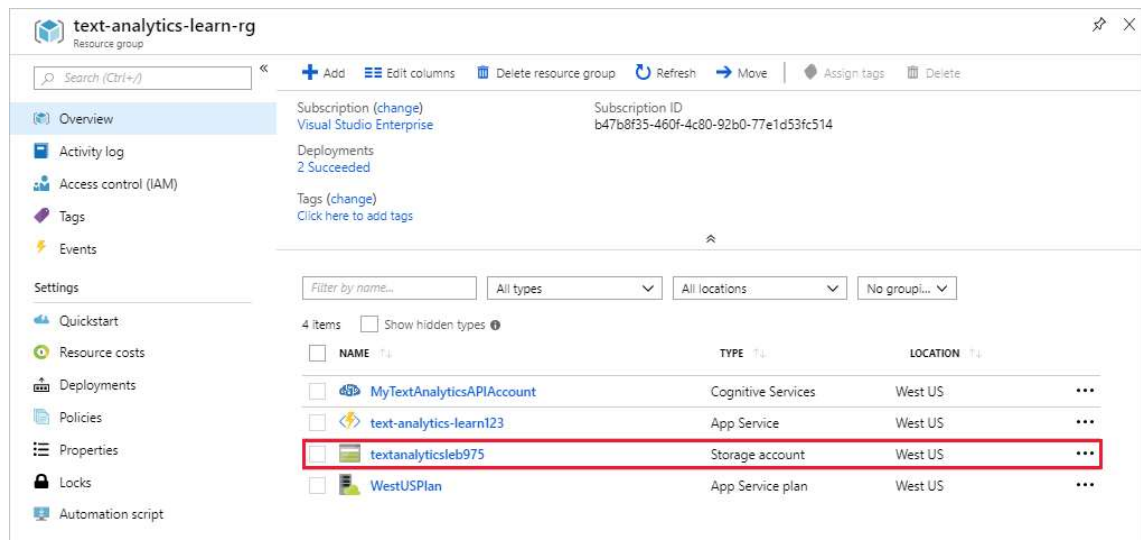
Congratulations! The **discover-sentiment-function** works as designed. In this example, we passed in a very upbeat message and received a score of over 0.98.

Try changing the message to something less optimistic, rerun the test, and note the response.

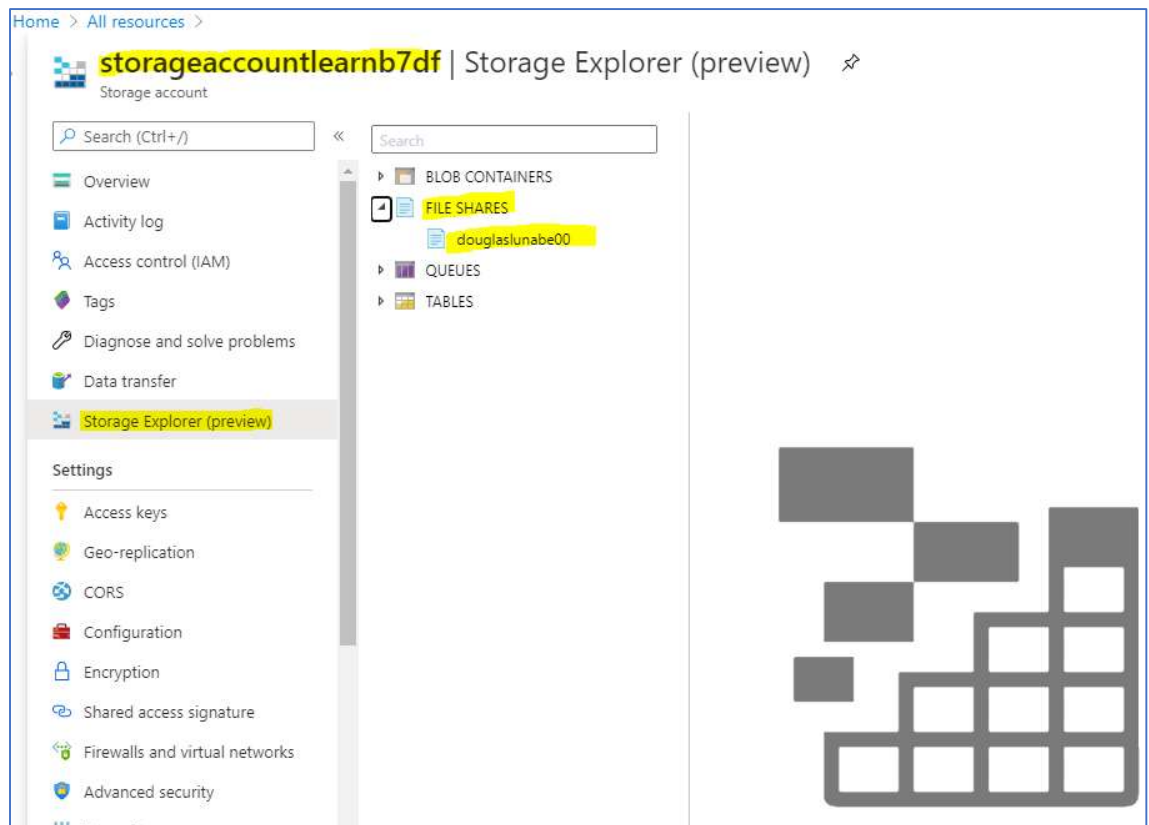
Add a message to the queue

Let's repeat the test. This time, instead of using the Test window of the portal, we'll actually place a message into the input queue and watch what happens.

1. Navigate to your resource group in the **Resource Groups** section of the portal.
2. Select learn-43e82f9f-9590-45a4-96e5-f7bbe7cc88bf, the resource group used in this lesson.
3. In the **Resource group** panel that appears, locate the Storage Account entry, and select it.

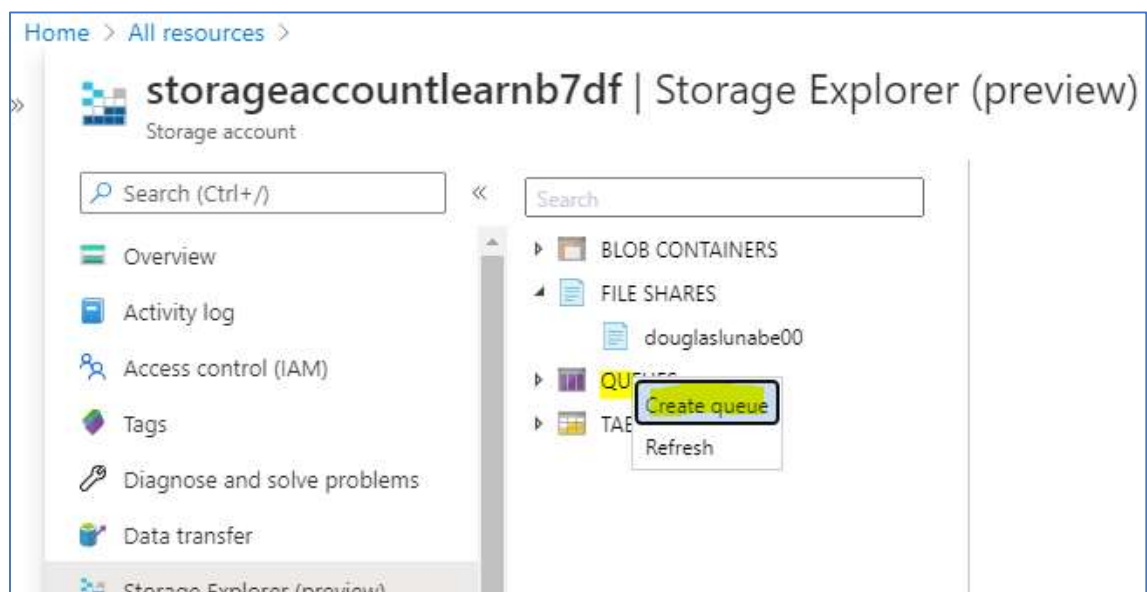


4. Select **Storage Explorer (preview)** from the left menu of the Storage Account main window. This action opens the Azure Storage Explorer inside the portal.

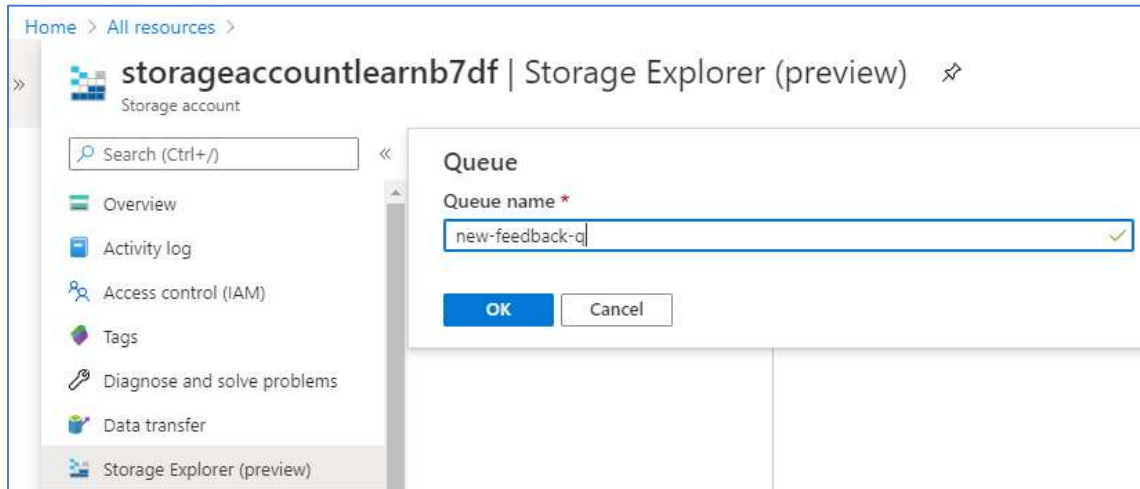


As you can see, we don't have any queues in this storage account yet, so let's add one.

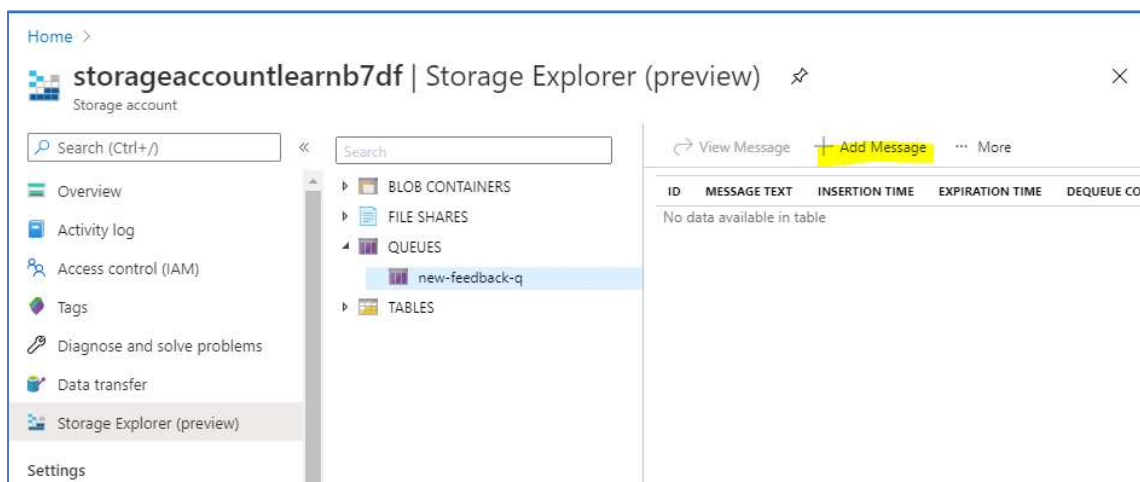
5. If you remember from earlier in this lesson, we named the queue associated with our trigger **new-feedback-q**. Right-click on the **Queues** item in the Storage Explorer, and select *Create Queue*.



6. In the dialog that opens, enter **new-feedback-q** and click **OK**. We now have our input queue.



7. Select the new queue in the left-hand menu to see the data explorer for this queue. As expected, the queue is empty. Let's add a message to the queue using the **Add Message** command at the top of the window.



8. In the **Add Message** dialog, enter "This message came from our input queue, new-feedback-q" into the **Message text** field, and click **OK** at the bottom of the dialog.

```
(preview)
```

View Message

ID	MESSAGE TEXT
No data available in t	

Add Message

Message text

This message came from our input queue, new-feedback-q

Expires in:

7
Days

☒ Encode message body in Base64

- Observe the message, similar to the message in the following screenshot, in the data explorer.

View Message	+ Add Message	— Dequeue Message	Clear Queue	Refresh
ID	MESSAGE TEXT	INSERTION TIME	EXPIRATION TIME	
cc264b19-cde9-41e3-8f04-461d49fcd349	This message came from our input queue, new-feedback-q	8/29/2018, 9:04:58 PM	9/5/2018, 9:04:58 PM	

- After a few seconds, click **Refresh** to refresh the view of the queue. Observe that the queue is **empty** once again. Something must have read the message from the queue.
- Navigate back to our function in the portal, and open the **Monitor** tab. Select the newest message in the list. Observe that our function processed the queue message we had posted to the new-feedback-q. Results may be delayed in this log, so you might have to wait a few minutes and hit *Refresh*.

Home > discover-sentiment-function | Monitor

Function

Search (Ctrl+/) <<

Overview

Developer

- Code + Test
- Integration
- Monitor**
- Function Keys

Invocations Logs

Success Count 11 Last 30 Days

Error Count 0 Last 30 Days

Invocation Traces

The twenty most recent function invocation traces

Run query in Application Insights

Filter invocations

Date (UTC)	Success
2020-07-13 23:32:23.566	✓ Succeeded
2020-07-13 23:26:57.145	✓ Succeeded
2020-07-13 23:24:18.056	✓ Succeeded
2020-07-13 23:23:32.605	✓ Succeeded
2020-07-13 23:22:54.010	✓ Succeeded

Invocation Details

Run query in Application Insights

Timestamp	Message	Type
2020-07-13 23:32:23.609	Executing 'Functions.discover-sentiment-function' (Reason: 'New queue message detected on 'new-feedback-q', Id=34a2b157-d466-4089-ad63-31cca507e2d4)	Information
2020-07-13 23:32:23.616	Trigger Details: MessageId: 57cbe1e1-ba4e-48e5-954f-10e091dd48e7, DequeueCount: 1, InsertionTime: 2020-07-13T23:32:04.000+00:00	Information
2020-07-13 23:32:26.276	Processing queue message This message came from our input queue, new-feedback-q	Information
2020-07-13 23:32:26.707	{ "documents": [{ "id": "1", "score": 0.7731928825378418 }], "errors": [] }	Information
2020-07-13 23:32:26.933	Executed 'Functions.discover-sentiment-function' (Succeeded, Id=34a2b157-d466-4089-ad63-31cca507e2d4)	Information

In this test, we did a complete round trip of posting something into our queue and then seeing the function process it.

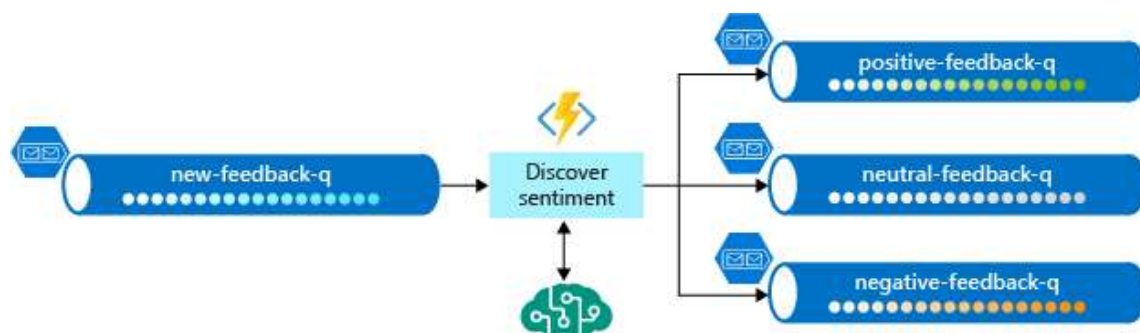
We're making progress with our solution! Our function is now doing something useful. It's receiving text from our input queue and then calling out to the Text Analytics API service to get a sentiment score. We've also learned how to test our function through the Azure portal and the Storage Explorer. In the next exercise, we'll see how easy it is to write to queues using output bindings.

Exercise - Sort messages into different queues based on sentiment score

Sandbox activated! Time remaining:
2 hr 37 min

You have used 2 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Let's look at our solution architecture again.



As you can see on the right side of this diagram, we want to send messages to three queues. So, we'll define those connections as output bindings in our function. We could create those bindings through the **Output binding** UI. However, to save time, we'll edit the config file directly.

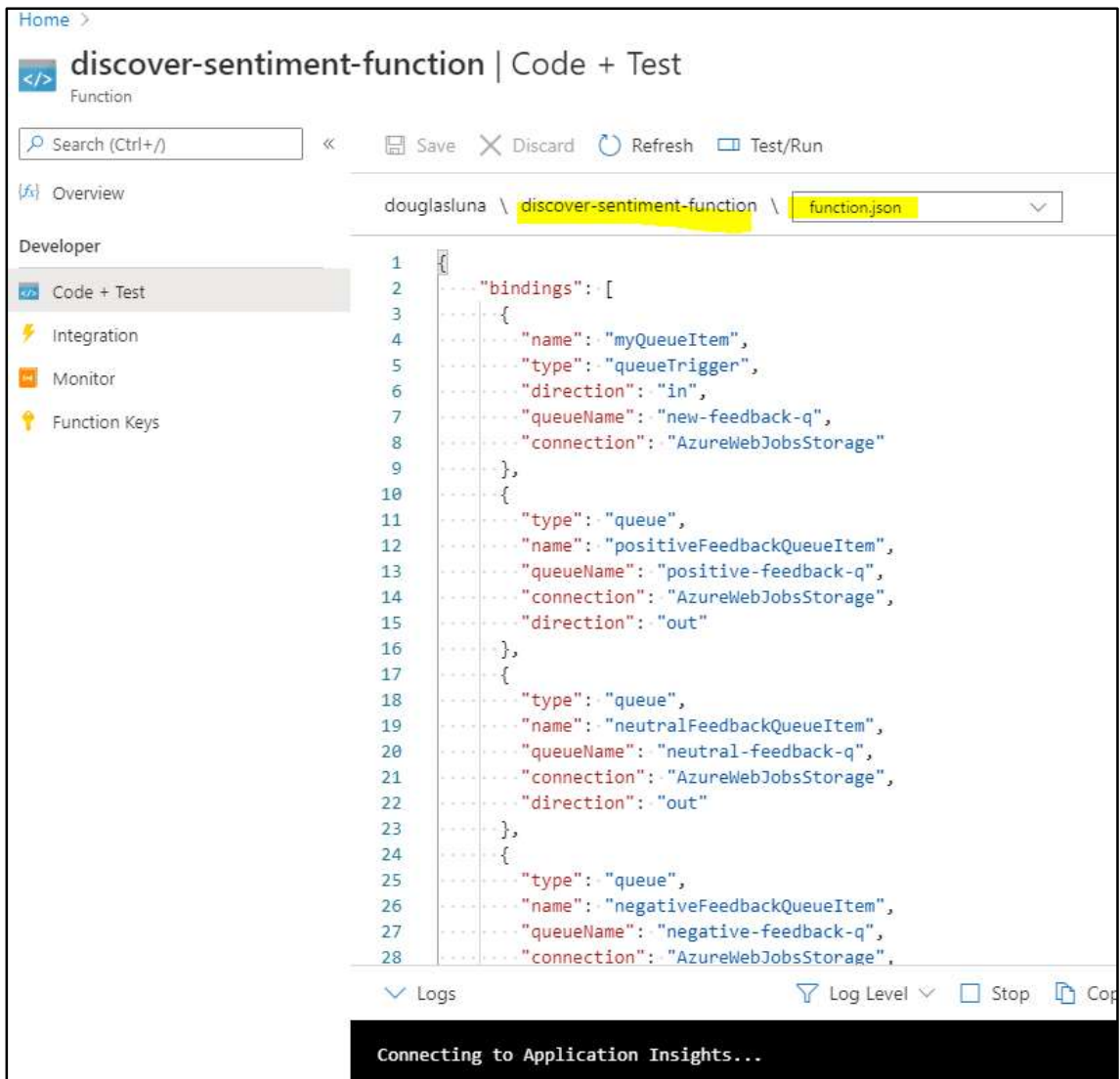
Add output bindings to function.json

1. Select our function, **discover-sentiment-function**, in the Function Apps portal.
2. Expand the **View files** menu on the right of the screen.
3. Under the **View files** tab, select **function.json** to open the config file in the editor.
4. Replace the entire contents of **function.json** with the following JSON and select **Save**.

JSON

```
{
  "bindings": [
    {
      "name": "myQueueItem",
      "type": "queueTrigger",
```

```
"direction": "in",
"queueName": "new-feedback-q",
"connection": "AzureWebJobsStorage"
},
{
  "type": "queue",
  "name": "positiveFeedbackQueueItem",
  "queueName": "positive-feedback-q",
  "connection": "AzureWebJobsStorage",
  "direction": "out"
},
{
  "type": "queue",
  "name": "neutralFeedbackQueueItem",
  "queueName": "neutral-feedback-q",
  "connection": "AzureWebJobsStorage",
  "direction": "out"
},
{
  "type": "queue",
  "name": "negativeFeedbackQueueItem",
  "queueName": "negative-feedback-q",
  "connection": "AzureWebJobsStorage",
  "direction": "out"
}
],
"disabled": false
}
```

The screenshot shows the Azure Functions 'Code + Test' editor. The file 'function.json' is open, displaying the following JSON configuration:

```
1 {
2   "bindings": [
3     {
4       "name": "myQueueItem",
5       "type": "queueTrigger",
6       "direction": "in",
7       "queueName": "new-feedback-q",
8       "connection": "AzureWebJobsStorage"
9     },
10    {
11      "type": "queue",
12      "name": "positiveFeedbackQueueItem",
13      "queueName": "positive-feedback-q",
14      "connection": "AzureWebJobsStorage",
15      "direction": "out"
16    },
17    {
18      "type": "queue",
19      "name": "neutralFeedbackQueueItem",
20      "queueName": "neutral-feedback-q",
21      "connection": "AzureWebJobsStorage",
22      "direction": "out"
23    },
24    {
25      "type": "queue",
26      "name": "negativeFeedbackQueueItem",
27      "queueName": "negative-feedback-q",
28      "connection": "AzureWebJobsStorage",
29      "direction": "out"
30    }
31  ]
32 }
```

At the bottom of the editor, there is a status bar with the text 'Connecting to Application Insights...'.

We've added three new bindings to the config.

- Each new binding is of type queue. These bindings are for the three queues that we'll populate with our feedback messages once we know the sentiment of the feedback.
- Each binding has a direction defined as out, since we'll post messages to these queues.
- Each binding uses the same connection to our storage account.
- Each binding has a unique queueName and name.

Posting a message to a queue is as easy as saying, for example, `context.bindings.negativeFeedbackQueueItem = "<message>"`.

Update the function implementation to sort feedback into queues based on sentiment score

The goal of our feedback sorter is to sort feedback into three buckets: positive, neutral, and negative. So far, we have our input queue, our code to call the Text Analytics API, and we've defined our output queues. In this section, we'll add the logic to move messages into those queues based on sentiment.

1. Navigate to our function, **discover-sentiment-function**, and open `index.js` in the code editor again.
2. Replace the implementation with the following code.

JavaScript

```
module.exports = function (context, myQueueItem) {
  context.log('Processing queue message', myQueueItem);

  let https = require('https');

  // Replace the accessKey string value with your valid access key.
  let accessKey = '<YOUR ACCESS CODE HERE>';

  // Replace [region], including square brackets, in the uri variable below.
  // You must use the same region in your REST API call as you used to obtain your access keys.
  // For example, if you obtained your access keys from the northeurope region, replace
  // "westus" in the URI below with "northeurope".
  let uri = '[region].api.cognitive.microsoft.com';
  let path = '/text/analytics/v2.0/sentiment';

  let response_handler = function (response) {
    let body = '';

    response.on('data', function (chunk) {
      body += chunk;
    });

    response.on('end', function () {
      let body_ = JSON.parse(body);

      // Even though we send and receive a documents array from the Text Analytics API,
      // we only ever pass one document in the array.
      if (body_.documents && body_.documents.length == 1) {
        let score = body_.documents[0].score;

        // Create a message that contains the original message we received and
        // the sentiment score returned by Text Analytics API.
        let messageWithScore = JSON.stringify({
          originalMessage: myQueueItem,
          score: score
        });

        // Place message into appropriate output queue based on sentiment score.
```

```

        if (score > 0.8) {
            context.log ("Positive message arrived");
            context.bindings.positiveFeedbackQueueItem = messageWithScore;
        } else if (score < 0.3) {
            context.log ("Negative message arrived");
            context.bindings.negativeFeedbackQueueItem = messageWithScore;
        } else {
            context.log ("Neutral message arrived");
            context.bindings.neutralFeedbackQueueItem = messageWithScore;
        }
    }
    let body__ = JSON.stringify (body_, null, ' ');
    context.log (body__);
    context.done();
    return;
});

response.on ('error', function (e) {
    context.log ('Error: ' + e.message);
    context.done();
    return;
});
};

let get_sentiments = function (documents) {
    let body = JSON.stringify (documents);

    let request_params = {
        method : 'POST',
        hostname : uri,
        path : path,
        headers : {
            'Ocp-Apim-Subscription-Key' : accessKey,
        }
    };

    let req = https.request (request_params, response_handler);
    req.write (body);
    req.end ();
}

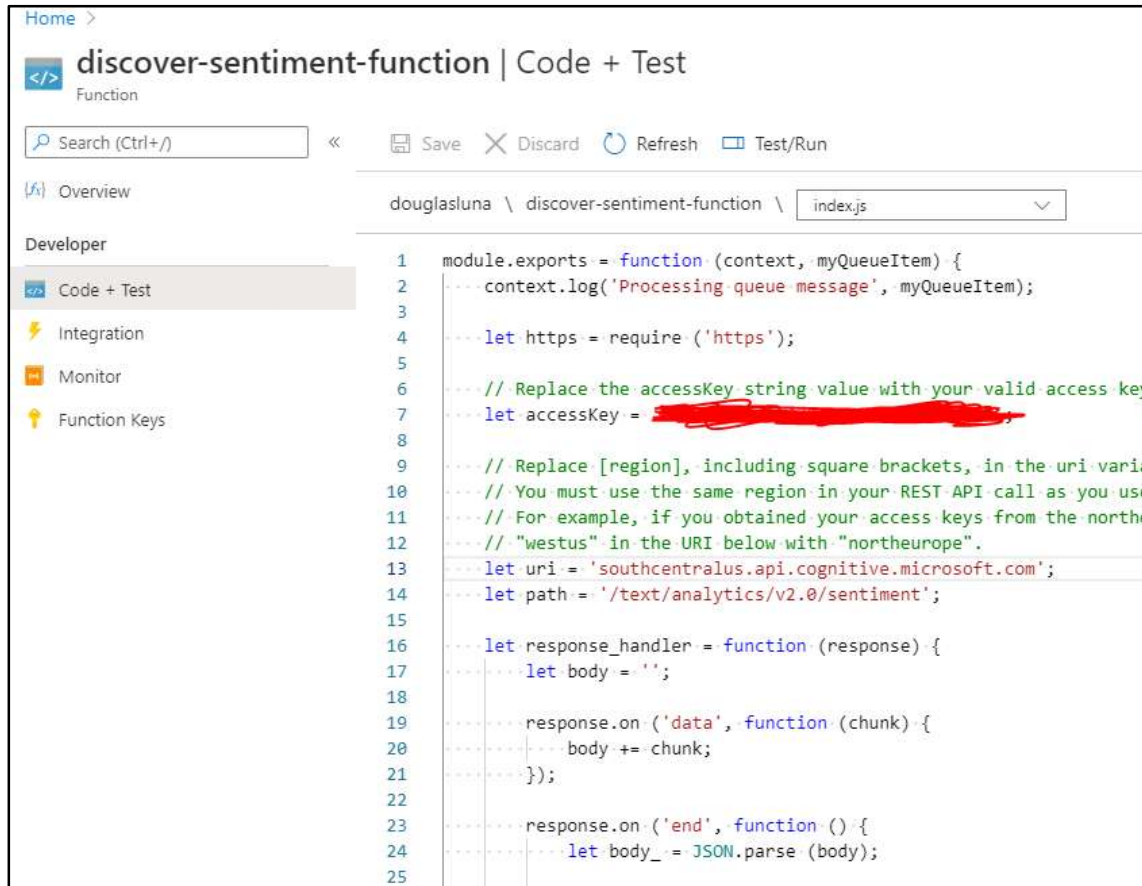
// Create a documents array with one entry.
let documents = { 'documents': [
    {
        'id': '1',
        'language': 'en',
        'text': myQueueItem
    },

```

```
});
```

```
get_sentiments (documents);
```

```
};
```



The screenshot shows the 'Code + Test' view of an Azure Function named 'discover-sentiment-function'. The file 'index.js' is open, displaying the following code:

```
1 module.exports = function (context, myQueueItem) {
2   ... context.log('Processing queue message', myQueueItem);
3
4   ... let https = require('https');
5
6   ... // Replace the accessKey string value with your valid access key
7   ... let accessKey = 'REDACTED';
8
9   ... // Replace [region], including square brackets, in the uri variable
10  ... // You must use the same region in your REST API call as you use
11  ... // For example, if you obtained your access keys from the north
12  ... // "westus" in the URI below with "northeurope".
13  ... let uri = 'southcentralus.api.cognitive.microsoft.com';
14  ... let path = '/text/analytics/v2.0/sentiment';
15
16  ... let response_handler = function (response) {
17    ... let body = '';
18
19    ... response.on('data', function (chunk) {
20      ... body += chunk;
21    });
22
23    ... response.on('end', function () {
24      ... let body_ = JSON.parse (body);
25  ... }
```

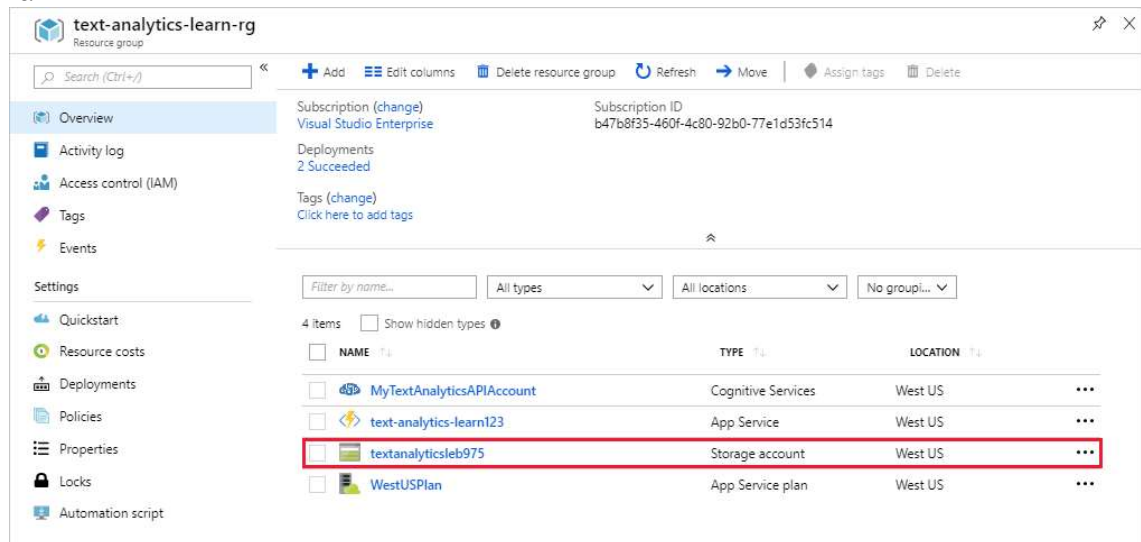
We've added the highlighted code to our implementation. The code parses the response from the Text Analytics API cognitive service. Based on the sentiment score, the message is forwarded to one of our three output queues. The code to post the message is just setting the correct binding parameter.

Try it out

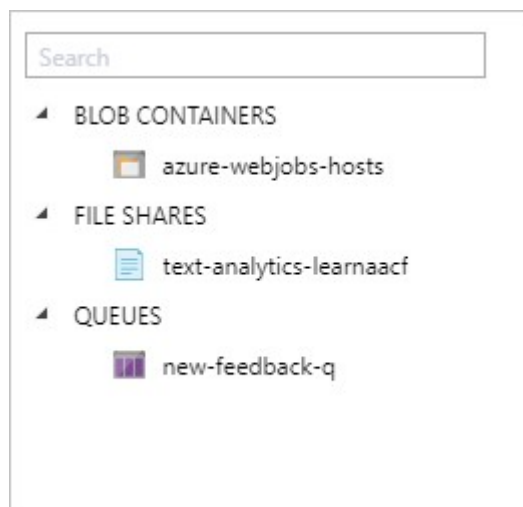
To test the updated implementation, we'll head back to the Storage Explorer.

1. Navigate to your resource group in the **Resource Groups** section of the portal.
2. Select learn-43e82f9f-9590-45a4-96e5-f7bbe7cc88bf, the resource group used in this lesson.

3. In the **Resource group** panel that opens, locate the Storage Account entry and select it.

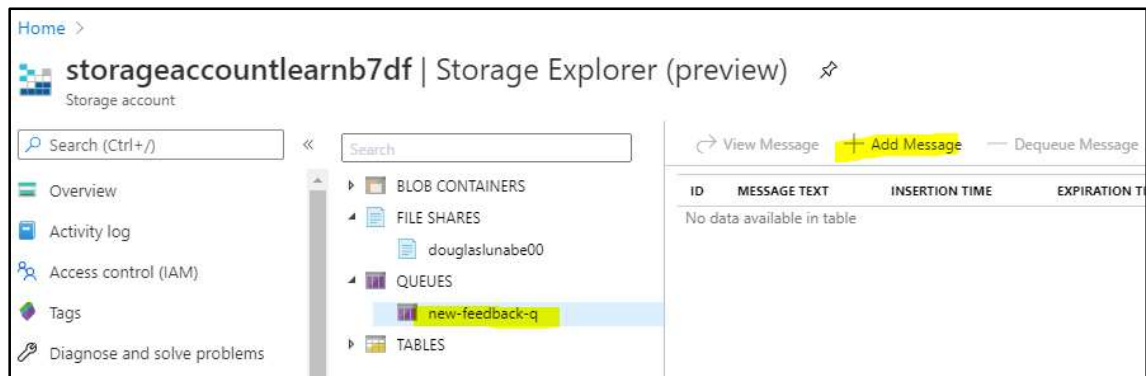


4. Select **Storage Explorer (preview)** from the left menu of the Storage Account main window. This action opens the Azure Storage Explorer inside the portal.

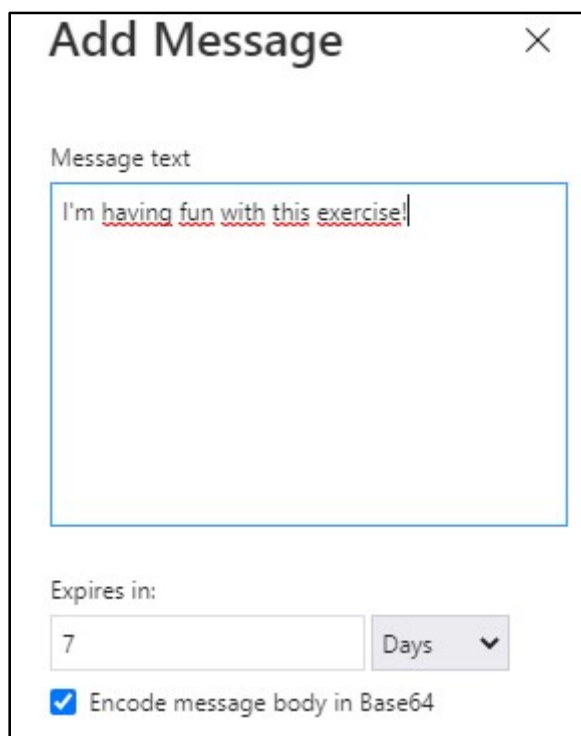


We have one queue listed under the **Queues** collection. This queue is **new-feedback-q**, the input queue we defined in the preceding test section of the module.

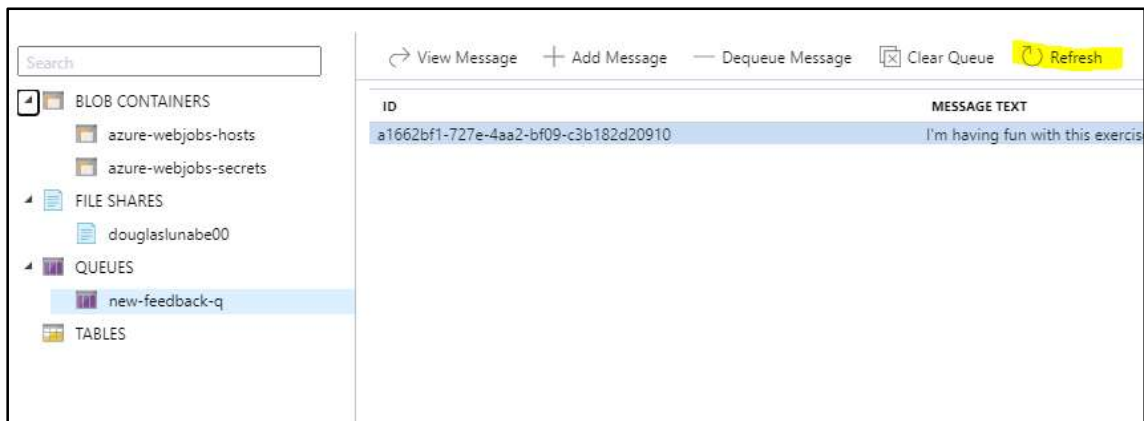
5. Select **new-feedback-q** in the left-hand menu to see the data explorer for this queue. As expected, the queue had no data. Let's add a message to the queue using the **Add Message** command at the top of the window.



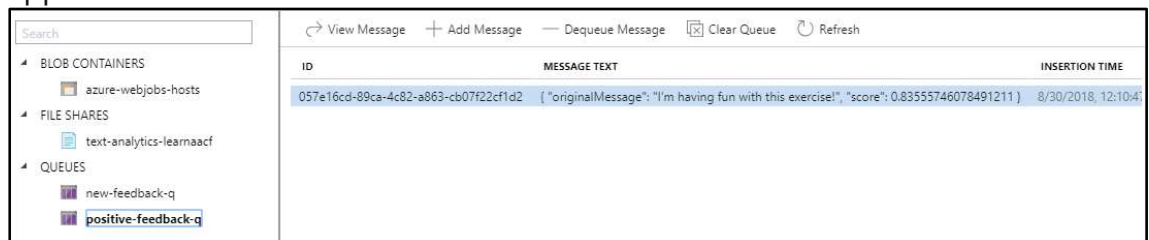
6. In the **Add Message** dialog, enter "I'm having fun with this exercise!" into the **Message text** field, and click **OK** at the bottom of the dialog.



7. The message is displayed in the data window for **new-feedback-q**. After a few seconds, click **Refresh** at the top of the data view to refresh the view of the queue. Observe that the message disappears after a while. So, where did it go?

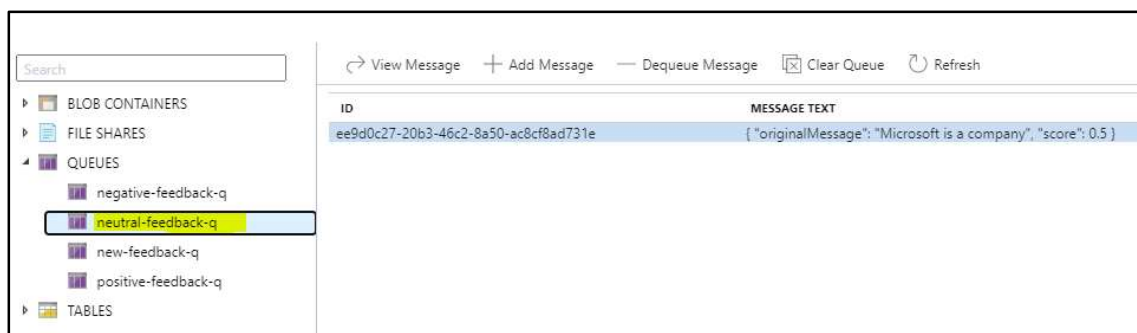
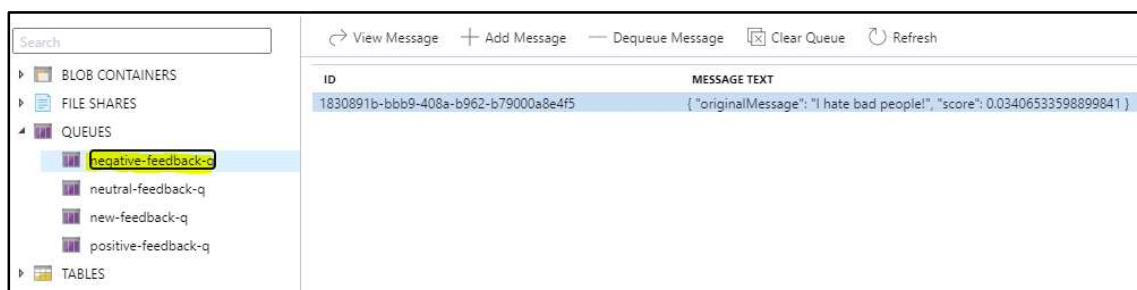
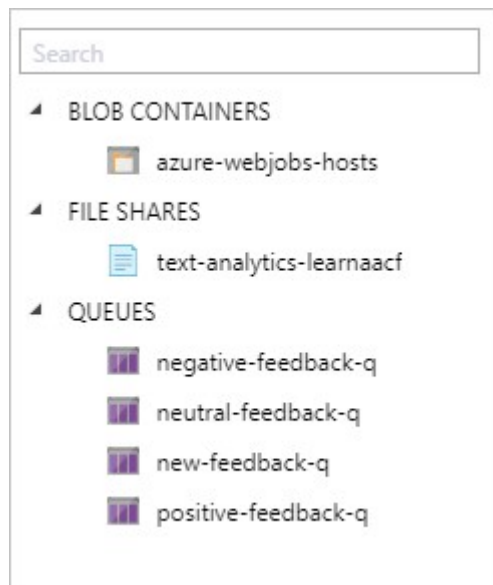


8. Right-click on the **QUEUES** collection in the left-hand menu. Observe that a *new* queue has appeared.



The queue **positive-feedback-queue** was automatically created when a message was posted to it for the first time. With Azure Functions queue output bindings, you don't have to manually create the output queue before posting to it! Now that we see an incoming message has been sorted by our function into **positive-feedback-queue**, let's see where the following messages land.

9. Using the same steps as above, add the following messages to **new-feedback-q**.
 - o "I hate bad people!"
 - o "Microsoft is a company"
10. Click **Refresh** until **new-feedback-q** is empty once again. This process might take a few moments and require several refreshes.
11. Right-click on the **QUEUES** collection and observe two more queues appearing. The queues are named **neutral-feedback-queue** and **negative-feedback-queue**. This might take a few seconds, so continue refreshing the **QUEUES** collection until new queues appear. When complete, your queue list should look like the following.



12. Click on each queue in the list to see whether they have messages. If you added the suggested messages, you should see one in **positive-feedback-queue**, **neutral-feedback-queue**, and **negative-feedback-queue**.

Congratulations! We now have a working feedback sorter! As messages arrive in the input queue, our function uses the Text Analytics API service to get a sentiment score. Based on that score, the function forwards the messages to the appropriate queue. While it seems like the function processes only one queue item at a time, the Azure Functions runtime will actually read batches of queue items and spin up other instances of our function to process them in parallel.

Summary

Azure Cognitive Services is a rich suite of intelligent services that we can use to enrich our apps. The Text Analytics API has several operations to turn text into meaningful insights. We used the service to discover sentiment in text feedback from customers. We created a solution hosted in Azure Functions to sort these text messages into different buckets, or queues, for further processing.

Once you know how to call a REST API, then you can easily integrate these intelligent services into your solutions. They all follow a similar pattern:

- Sign up for an access key.
- Explore in the API testing console.
- Formulate requests using the access key and the correct region.
- POST requests from your solution, and parse the responses for insights.

We added this intelligence to serverless logic created in Azure Functions. You can easily call these services from other types of apps. There are many client libraries, tutorials, and quickstarts to get you started.

Suggestions for further enhancement of our solution

Here are some ideas for you to consider if you want to take what we did further.

- Test the solution with more text samples. Decide whether the thresholds we set to categorize sentiment scores into positive, negative, and neutral are appropriate.
- Consider adding another function into your function app that reads messages from the **negative-feedback-queue** queue and calls the Text Analytics API to find key phrases in the text.
- Our input queue contains raw text feedback. In the real world, we would associate feedback with some form of user ID, such as email address, account number, and so on. Enhance the input queue items to be JSON documents containing an ID field and the text. Then use that ID when working with the text message.
- Currently our solution is "hard coded" to English. Think about the changes you'd implement to make it capable of handling text in all languages supported by the Text Analytics API.
- If you are familiar with Logic Apps, visit the link to the built-in connector for text analytics in the Further Reading section.

Now that you know how to call one of these Cognitive Services APIs, explore some of the other services and think about how you might use them in your solutions.

Further reading

- [Text Analytics overview](#)
- [Text Analytics demo](#)
- [How to detect sentiment in Text Analytics](#)
- [Cognitive Services Documentation](#)
- [Text Analytics Logic Apps Connector](#)

Clean up

The sandbox automatically cleans up your resources when you're finished with this module.

When you're working in your own subscription, it's a good idea at the end of a project to identify whether you still need the resources you created. Resources left running can cost you money. You can delete resources individually or delete the resource group to delete the entire set of resources.

Check your knowledge

1. You're integrating the Text Analytics API into your solution. You went to the portal and signed up for a subscription key in the eastus region. Which of the following is the correct request URL for accessing the sentiment resource?

- ☐ eastus2.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
- ☐ eastus.api.cognitive.microsoft.com/text/analytics/v2.0/entities
- ☐ eastus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment

2. Which of the following is the correct direction setting on a queue binding definition in order to send messages to the queue?

- ☐ out
- ☐ in
- ☐ north

3. If I post a request to the Text Analytics API with some text and the response gives a sentiment score of 0.03, what is the dominant sentiment of the text?

- ☐ neutral
- ☐ negative
- ☐ positive

Check your knowledge

1. You're integrating the Text Analytics API into your solution. You went to the portal and signed up for a subscription key in the eastus region. Which of the following is the correct request URL for accessing the `sentiment` resource?

- ☐ eastus2.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
- ☐ eastus.api.cognitive.microsoft.com/text/analytics/v2.0/entities

☒ eastus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment

The endpoint specifies the region you chose during sign-up, the service URL, and a resource used on the request.

2. Which of the following is the correct `direction` setting on a queue binding definition in order to send messages to the queue?

☒ out

When the message is output from the function to the queue, the direction is out.

- ☐ in
- ☐ north

3. If I post a request to the Text Analytics API with some text and the response gives a sentiment score of 0.03, what is the dominant sentiment of the text?

☐ neutral

☒ negative

The API returns a sentiment score between 0 and 1 for each document, where 1 is the most positive. The score of 0.03 is close to 0 and, therefore, negative.

☐ positive