Microsoft Cloud Workshop

Intelligent Analytics

Leader Hackathon Guide

May 2017

© 2017 Microsoft Corporation. All rights reserved. This document is confidential and proprietary to Microsoft. Internal use only. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

Contents

[Intelligent analytics hackathon 1](#_Toc481586441)

[Overview 1](#_Toc481586442)

[Requirements 1](#_Toc481586443)

[Exercise 0: Before the hackathon 2](#_Toc481586444)

[Task 1: Provision Power BI 2](#_Toc481586445)

[Exercise 1: Environment setup 3](#_Toc481586446)

[Task 1: Download and open the ConciergePlus starter 3](#_Toc481586447)

[Task 2: App Services 3](#_Toc481586448)

[Task 3: Service Bus 8](#_Toc481586449)

[Task 4: Event Hubs 9](#_Toc481586450)

[Task 5: DocumentDB 12](#_Toc481586451)

[Task 6: Azure Search 14](#_Toc481586452)

[Task 7: Stream Analytics 15](#_Toc481586453)

[Task 8: Start the Stream Analytics Job 20](#_Toc481586454)

[Task 9: Storage Account 21](#_Toc481586455)

[Task 10: Cognitive Services 22](#_Toc481586456)

[Exercise 2: Implement message forwarding 24](#_Toc481586457)

[Task 1: Implement the event processor 24](#_Toc481586458)

[Task 2: Configure the Chat Message Processor Web Job 25](#_Toc481586459)

[Exercise 3: Configure the Chat Web App settings 30](#_Toc481586460)

[Exercise 4: Deploying the App Services 31](#_Toc481586461)

[Task 1: Publish the ChatMessageSentimentProcessor Web Job 31](#_Toc481586462)

[Task 2: Publish the ChatWebApp 35](#_Toc481586463)

[Task 3: Testing hotel lobby chat 37](#_Toc481586464)

[Exercise 5: Add intelligence 39](#_Toc481586465)

[Task 1: Implement sentiment analysis 39](#_Toc481586466)

[Task 2: Implement linguistic understanding 40](#_Toc481586467)

[Task 3: Implement speech to text 48](#_Toc481586468)

[Task 4: Re-deploy and test 48](#_Toc481586469)

[Exercise 6: Building the Power BI dashboard 50](#_Toc481586470)

[Task 1: Create the static dashboard 50](#_Toc481586471)

[Task 2: Create the real-time dashboard 53](#_Toc481586472)

[Exercise 7: Enabling search indexing 57](#_Toc481586473)

[Task 1: Verifying message archival 57](#_Toc481586474)

[Task 2: Creating the index and indexer 59](#_Toc481586475)

[Task 3: Update the Web App web.config 62](#_Toc481586476)

[Task 4: Configure the Search API App 62](#_Toc481586477)

[Task 5: Re-publish apps 63](#_Toc481586478)

[Exercise 8: Cleanup 65](#_Toc481586479)

[Task 1: Clean up Azure Resources 65](#_Toc481586480)

# Intelligent analytics hackathon

## Overview

Adventure Works Travel specializes in building software solutions for the hospitality industry. Their latest product is an enterprise mobile/social chat product called Concierge+ (aka ConciergePlus). The mobile web app enables guests to easily stay in touch with the concierge and other guests, enabling greater personalization and improving their experience during their stay. Sentiment analysis is performed on top of chat messages as they occur, enabling hotel operators to keep tabs on guest sentiments in real-time.

## Requirements

* Microsoft Azure subscription must be pay-as-you-go or MSDN.
  + Trial subscriptions will *not* work.
* Local machine or a virtual machine configured with:
  + Visual Studio 2015 Community Edition with Update 1 or later
  + Azure SDK 2.8.2 for Visual Studio
  + Azure PowerShell 1.0.0 or later

## Exercise 0: Before the hackathon

Duration: 10 minutes

Synopsis: You should follow all of the steps provided in Exercise 0 of the Proctor Guide (included in this document) *before* attending the hackathon.

### Task 1: Provision Power BI

1. If you do not already have a Power BI account, go to <https://www.powerbi.com>.
2. On the page, enter your work email address (which should be the same account as the one you use for your Azure subscription) and select **Use it free**.



1. Follow the on-screen prompts and your Power BI environment should be ready within minutes. You can always return to it via <https://app.powerbi.com>.

## Exercise 1: Environment setup

Duration: 45 minutes

Synopsis: The following section walks you through the manual steps to provision the services required using the Azure portals. Adventure Works has provided a starter solution for you. They have asked you to use this as the starting point for creating the Concierge Plus intelligent chat solution in Azure.

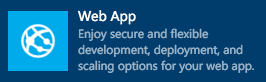
### Task 1: Download and open the ConciergePlus starter

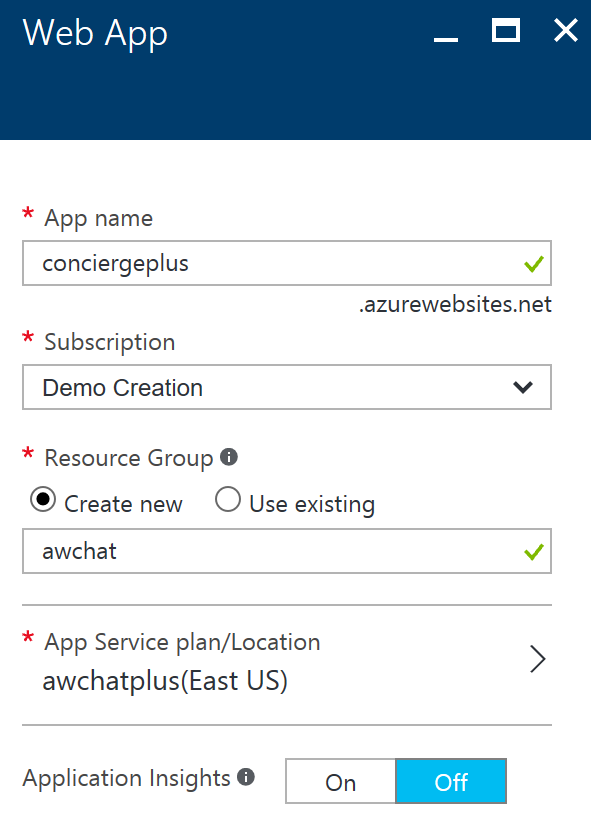
1. Download the starter project from the following URL:  
   <http://bit.ly/2jktWV3>
2. Unzip the contents.
3. Open ConciergePlusSentiment.sln with Visual Studio.

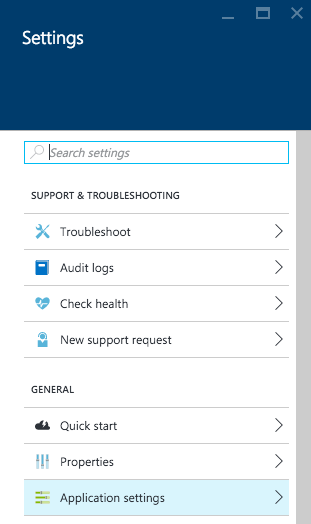
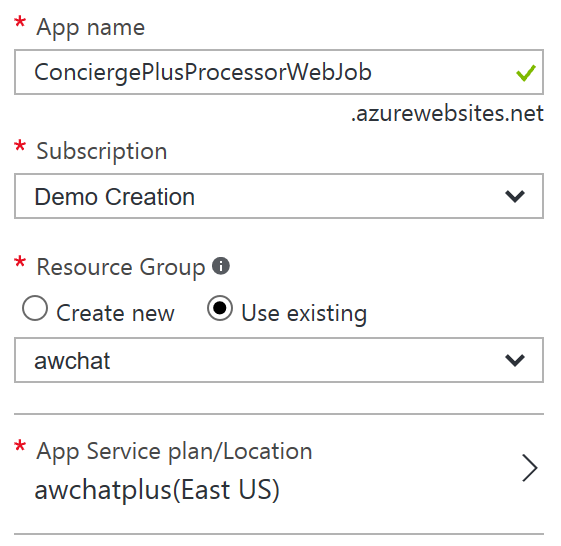
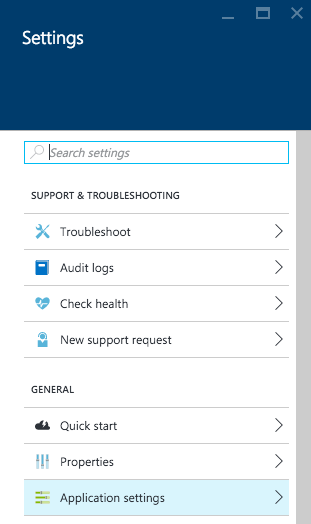
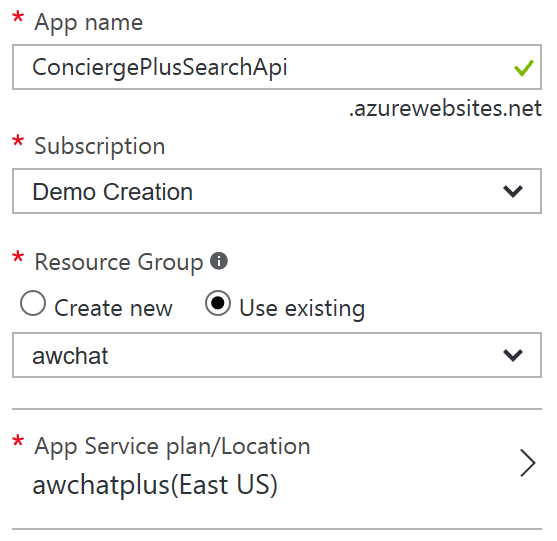
**Note:** If you attempt to build the solution at this point, you will see many build errors. This is intentional. You will correct these in the exercises that follow.

### Task 2: App Services

In these steps, you will provision two Web Apps and an API App within a single App Service Plan.

1. Sign in to the Azure Portal.
2. Select +NEW.  
   ../../../../Captures/Screen%20Shot%202016-03-25%20at%207.24.52%20PM.png
3. Select Web + Mobile.  
   ../../../../Captures/Screen%20Shot%202016-03-25%20at%207.25.44%20PM.png
4. Select Web App.  
   
5. On the Web App blade, provide an App Name that is indicative of this resource being used to host the Concierge+ chat website.
6. Choose a Subscription.
7. Create new Resource Group that you will use for all resources (e.g., named “awchat”).

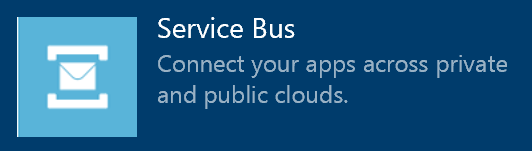
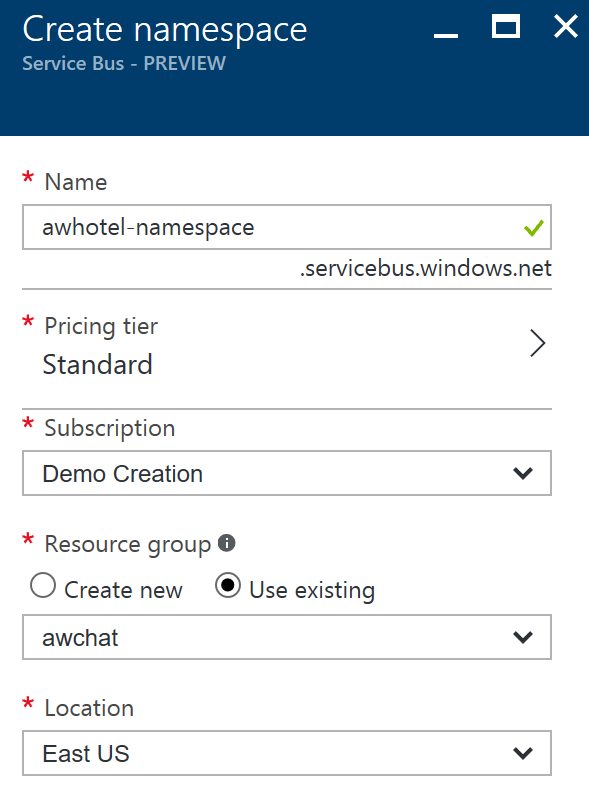
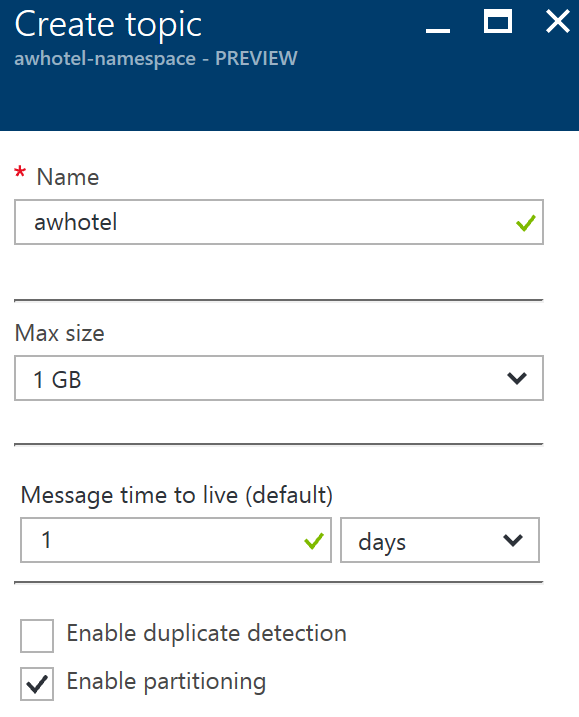


1. Create a new App Service plan in a region near you, naming it as you desire. It should be set with a Pricing tier of S1 Standard.
2. Select Create. This will provision both the App Service Plan and the Web App.
3. When the provisioning completes, navigate to your new Web App in the portal.
4. On the Settings blade, select Application settings.  
   
5. Select the toggle for Web Sockets to On.  
   ../../../../Captures/Screen%20Shot%202016-03-26%20at%2010.48.34%20AM.png
6. Select Save.  
   ../../../../Captures/Screen%20Shot%202016-03-26%20at%2010.48.48%20AM.png
7. You are now ready to provision the other Web App using this same service plan.
8. Select +New, Web + Mobile, Web App.
9. Provide a name for this new Web App that indicates its use as the host for the Event Processor Web Job (e.g., ChatProcessorWebJob).
10. Select the same subscription as used previously.
11. Select the Resource Group that you created previously.
12. Select the App Service plan that you created previously.  
    
13. Select Create.
14. When the provisioning completes, navigate to your new Web App in the portal.
15. On the Settings blade, select Application settings.  
    
16. Set the Always On toggle to the On position. You want to make sure Always On is enabled for this so that the Web App hosting the Web Job never goes to sleep and is always processing chat messages.  
    ../../../../Captures/Screen%20Shot%202016-04-15%20at%209.44.17%20AM.png
17. Select Save.  
    ../../../../Captures/Screen%20Shot%202016-03-26%20at%2010.48.48%20AM.png
18. Select + New, Web + Mobile, API App.  
    *(Sometimes API App does not appear on the list. If that happens, simply click* ***+ New*** *and search for* ***API App****.)*  
    
19. Provide an App name for this API app that reflects it will host the Chat Search API (e.g., ChatSearchApi).
20. Select the same subscription as used previously.
21. Select the Resource Group that you created previously.
22. Select the App Service plan that you created previously.
23. Select Create.  
    

You now have all the App Services pre-created that you will need for this hackathon.

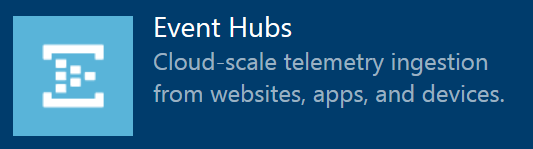
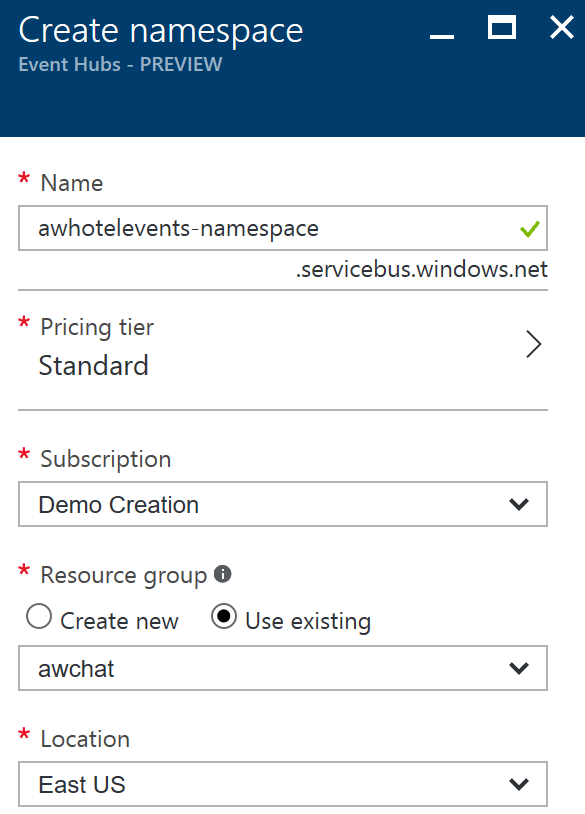
### Task 3: Service Bus

In this section, you will provision the Service Bus Namespace and Service Bus Topic, and Event Hubs instance.

1. Continuing within the Azure Portal, select + New.
2. Select Enterprise Integration, then Service Bus.
3. Provide a name for the namespace (e.g., awhotel-namespace).
4. Select Standard Pricing Tier.
5. Select the same subscription as used previously.
6. Select the Resource Group that you created previously.
7. Select the same Location you have been using.  
    
8. Select Create.
9. When the provisioning completes, navigate to your new Service Bus in the portal.
10. In the Overview blade, click + Topic.  
    
11. In the Create topic dialog, provide a name for the topic (e.g., awhotel) that represents that this topic will handle the messages for a particular hotel.
12. Leave the Max Size at 1 GB.
13. Set the Default Message Time to Live to 1 days.  
    
14. Select Create.  
    

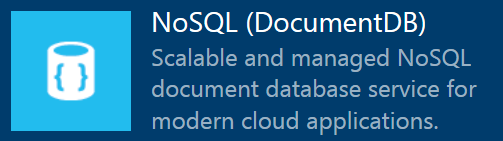
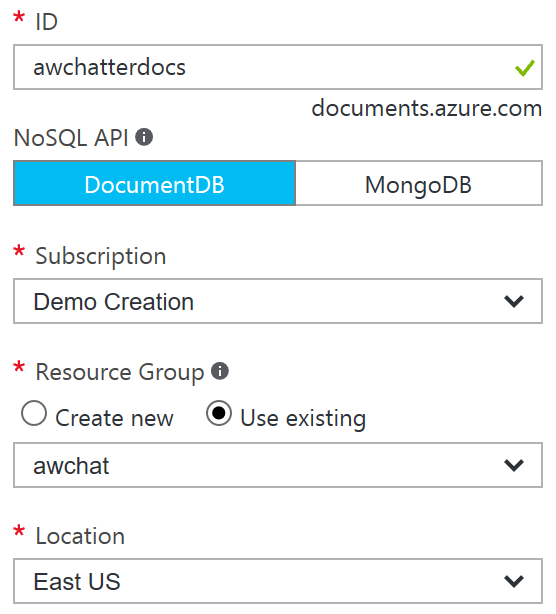
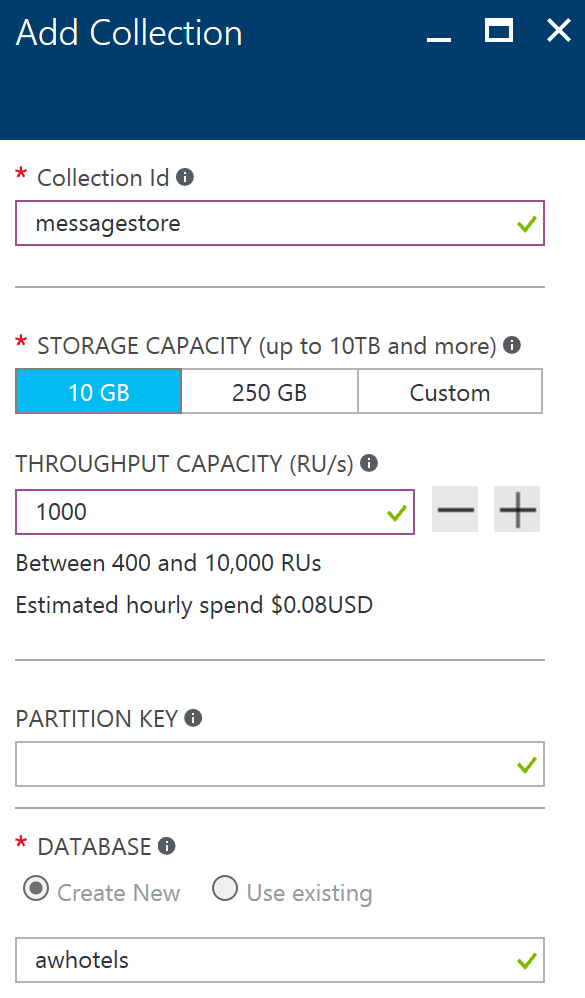
### Task 4: Event Hubs

You will create a new Event Hubs namespace and instance.

1. Continuing within the Azure Portal, select + New.
2. Select Internet of Things, then Event Hubs.  
   
3. Provide a name for the namespace (e.g., awhotelevents-namespace).
4. Select Standard Pricing Tier.
5. Select the same subscription as used previously.
6. Select the Resource Group that you created previously.
7. Select the same Location you have been using.  
   
8. Select Create.  
   
9. When the provisioning completes, navigate to your new Event Hub namespace in the portal.
10. In the Overview blade, click + Event Hub.  
    
11. Provide a name for your Event Hub (e.g., awchathub).
12. Set the partition count to the max value of 32. This will enable you to significantly scale up the number of downstream processors on the Event Hub, where each partition consumer (as handled by the EventProcessorHost) can reach up to 1 Throughput Unit per partition should the need arise. You cannot change this value later.
13. Set the Message Retention to 1 day.
14. Leave Archive set to Off.  
    
15. Click Create.  
    
16. Repeat steps 10 through 15 to create another Event Hub. This one will store messages for archival and be processed by Stream Analytics. Name it awchathub2.

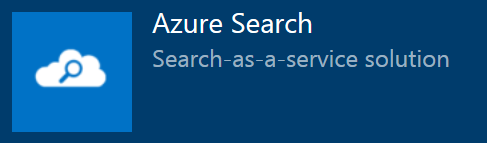
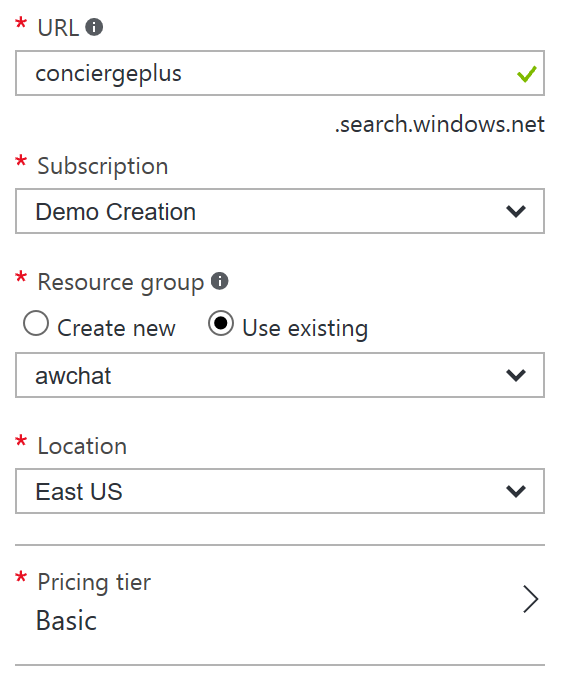
### Task 5: DocumentDB

In this section, you will provision a DocumentDB account, a DocumentDB Database, and a DocumentDB collection that will be used to collect all of the chat messages.

1. Continuing within the Azure Portal, select + New, Databases, NoSQL (DocumentDB).  
   
2. Provide a unique name for the DocumentDB account.
3. Select the DocumentDB NoSQL API.
4. Choose the same subscription you have been using thus far.
5. Choose the same Resource Group as you have used previously.
6. Choose the same Location as you have for your other resources.  
    
7. Click Create.  
   
8. When the provisioning completes, navigate to your new NoSQL (DocumentDB) account in the portal.
9. In the Overview blade, click + Add Collection.  
   
10. Provide a Collection Id (e.g., messagestore).
11. Set the Storage Capacity to 10 GB.
12. Set the Throughput Capacity to 1000.
13. Leave the Partition Key blank. This will create the collection as a single-partition collection instead of a partitioned collection. We recommend using partitioned collections over single-partitioned collections in general. Because we expect only a small amount of data storage and requests for this hackathon, we are continuing with a single-partitioned collection.
14. Select the Create New radio underneath Database, and enter a Database Id (e.g., awhotels).  
    
15. Select Create.

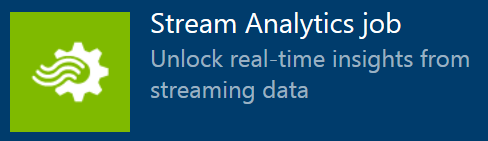
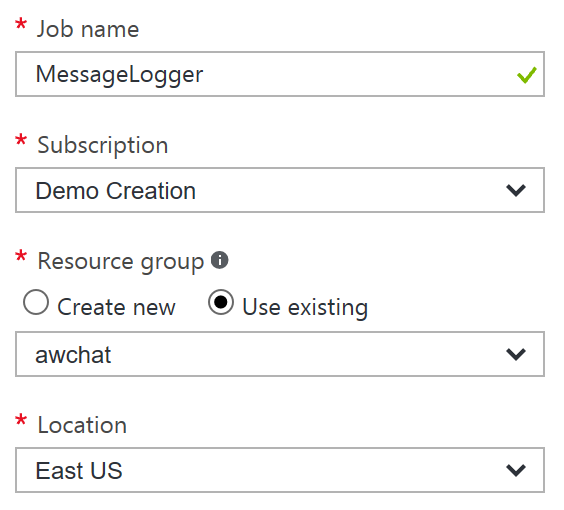
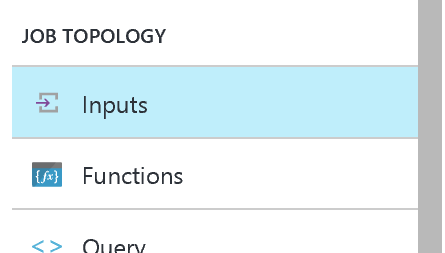
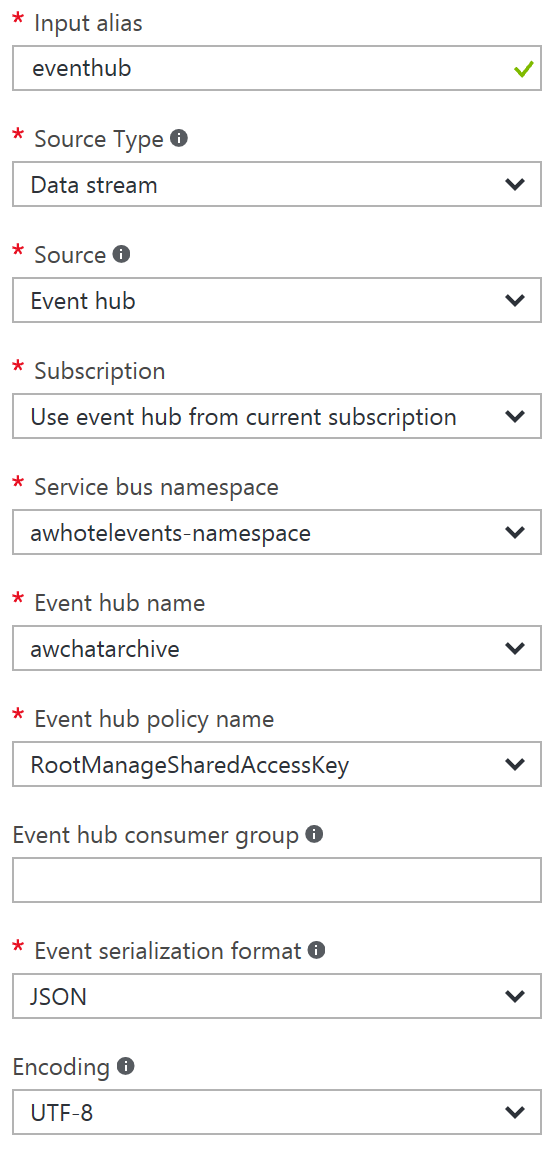
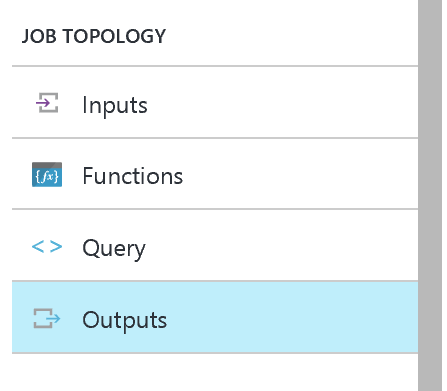
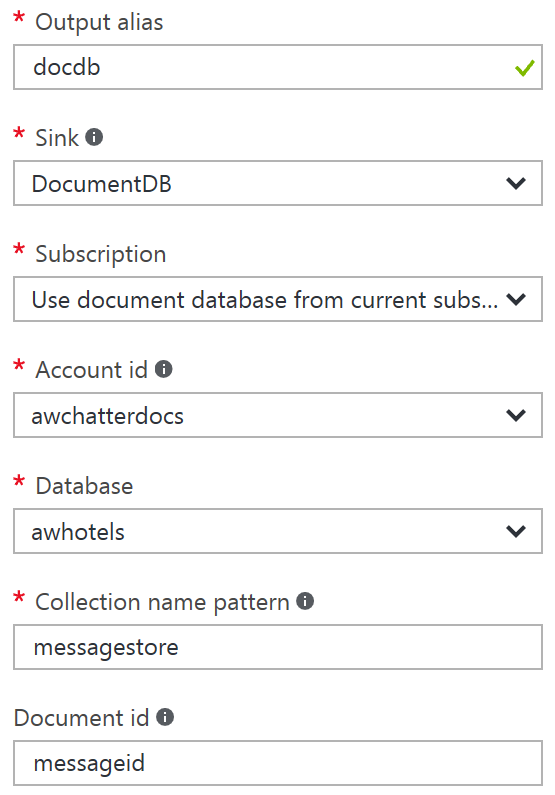
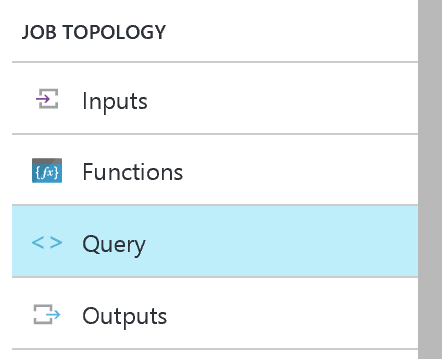
### Task 6: Azure Search

In this section, you will create an Azure Search instance.

1. Select + New, Web + Mobile, Azure Search.  
   
2. Provide a URL for the search service.
3. Choose the Subscription, Resource Group, and Location as you have done for the previous services.
4. Set the Pricing tier to Basic.  
    
5. Select Create.  
   

### Task 7: Stream Analytics

In this section, you will create the Stream Analytics Job that will be used to read chat messages from the Event Hub and write them to DocumentDB.

1. Select + New, Intelligence + analytics, Stream Analytics job.  
   
2. Provide a Job Name (e.g., MessageLogger).
3. Choose the same subscription you have been using thus far.
4. Choose the same Resource Group as you have used previously.
5. Choose the same Location as you have for your other resources.  
   
6. Select Create.  
   
7. When the provisioning completes, navigate to your new Stream Analytics job in the portal.
8. Select Inputs on the left-hand menu.  
    
9. In the Inputs blade, click + Add.  
   
10. For the input alias, set the value to eventhub.
11. For the Source Type, choose Data stream.
12. Choose Event hub for your source.
13. Choose the Subscription which contains your Event Hubs instance.
14. Choose the Namespace which contains your Event Hubs instance.
15. Choose the second Event Hub instance name you created.
16. Leave the Event hub consumer group empty ($Default consumer group will be used).
17. Leave the Serialization settings at JSON format and UTF 8 encoding.  
     
18. Select Create.  
    
19. Select Outputs from the left-hand menu.  
    
20. In the Outputs blade, click + Add.  
    
21. For the Output alias, enter docdb.
22. For the Sink, choose DocumentDB.
23. Leave Subscription set to Use document database from current subscription.
24. Select your DocumentDB Account id.
25. Select your DocumentDB database.
26. Set the Collection Name Pattern to the name of your single collection (e.g., messagestore).
27. Set the Document ID to messageid (again all lowercase).  
    
28. Select Create.  
    
29. Select Query from the left-hand menu.  
    
30. In the query text box, enter the following query:

SELECT

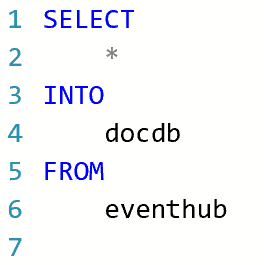
\*

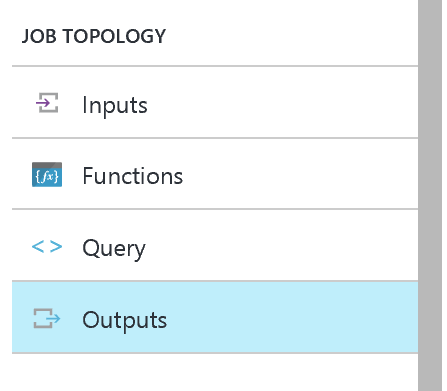
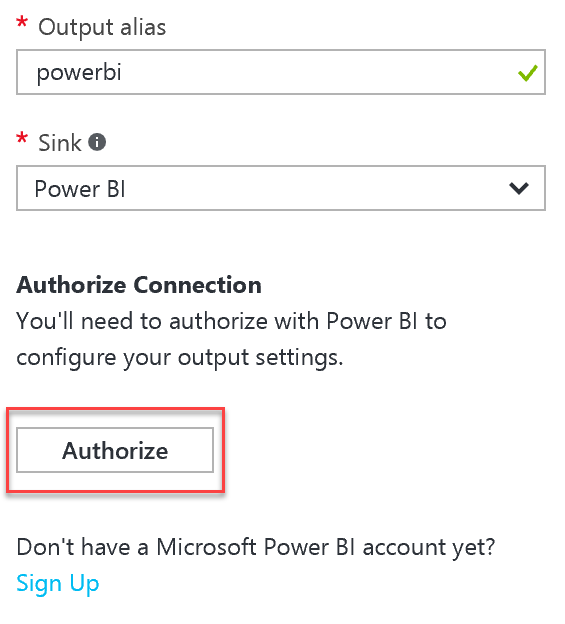
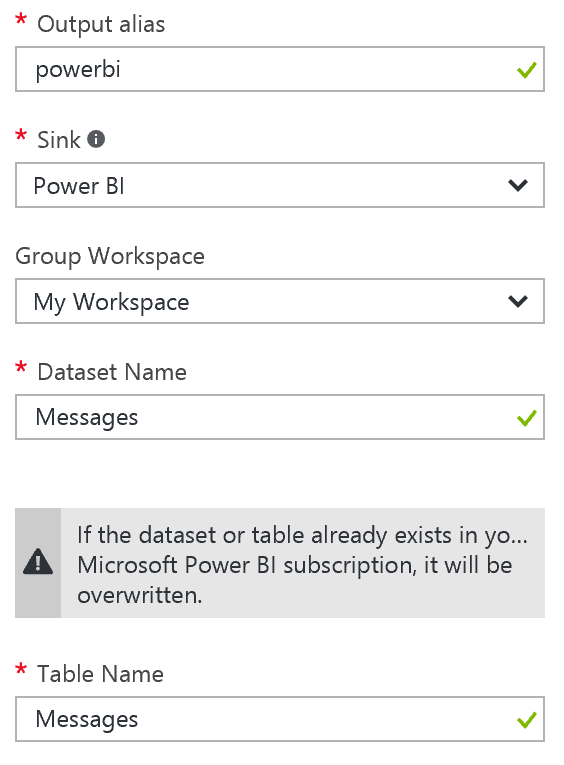
INTO

docdb

FROM

eventhub



1. Select Save and Yes when prompted with the confirmation.  
    
2. Add another Output for Power BI.
3. Select Outputs from the left-hand menu.  
   
4. In the Outputs blade, click + Add.  
   
5. For the Output alias, enter powerbi.
6. For the Sink, choose Power BI.
7. Click Authorize to authorize the connection to your Power BI account. When prompted in the popup window, enter your account credentials.  
    
8. For the remaining Power BI Settings, select the Group Workspace (default is My Workspace), set the Dataset Name to Messages, and Table Name to Messages, then select Create to create the output.   
    
9. Modify your query so that below the first query, you append the following query text:

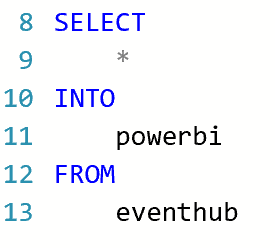
SELECT   
   
\*

INTO

powerbi

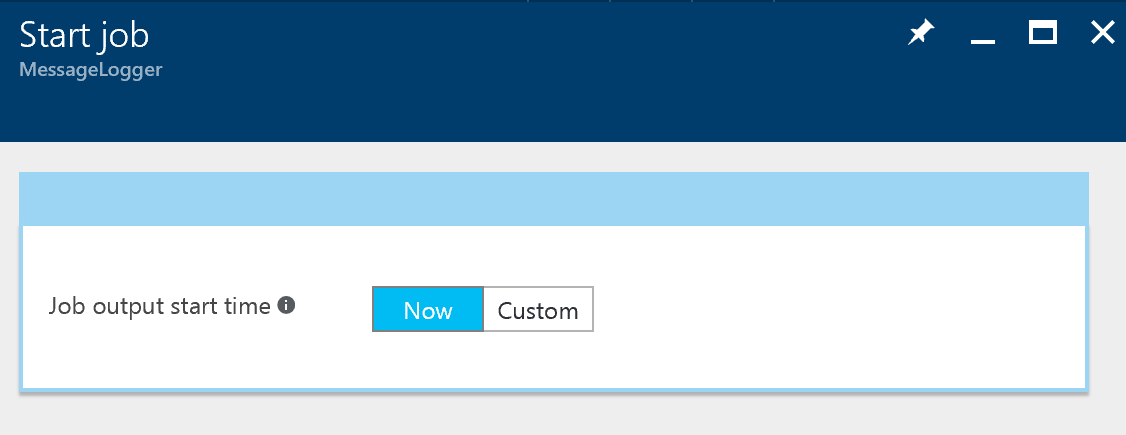
FROM

eventhub



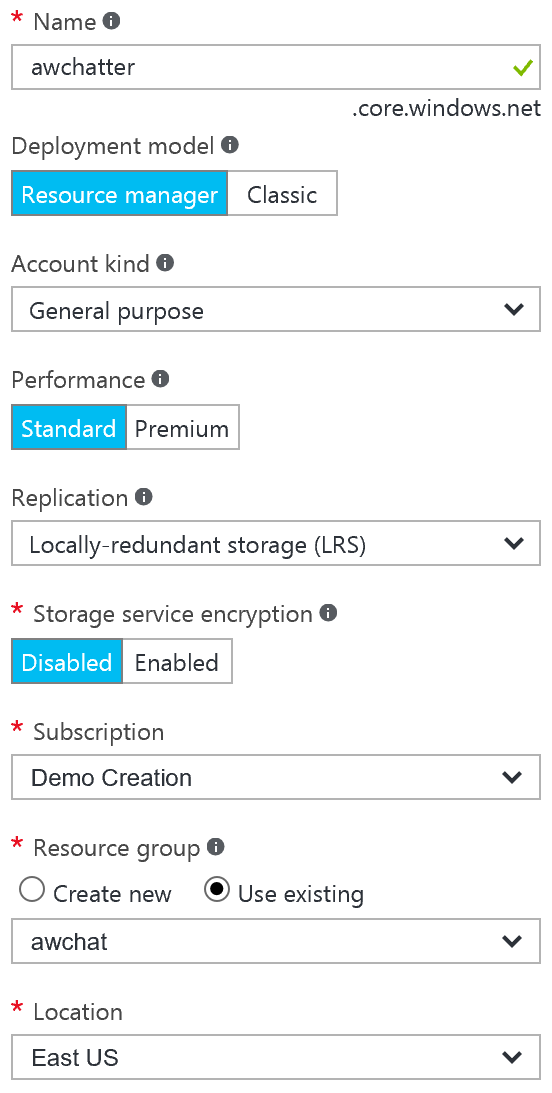
1. Select Save and Yes when prompted with the confirmation.  
    

### Task 8: Start the Stream Analytics Job

1. Navigate in the Azure Portal to your Stream Analytics Job.
2. From the Overview blade, select Start.  
    
3. In the Start job blade, leave the toggle to Now (the job will start processing messages from the current point in time onward).   
    
4. Select Start.
5. Allow your Stream Analytics Job a few minutes to start.

### Task 9: Storage Account

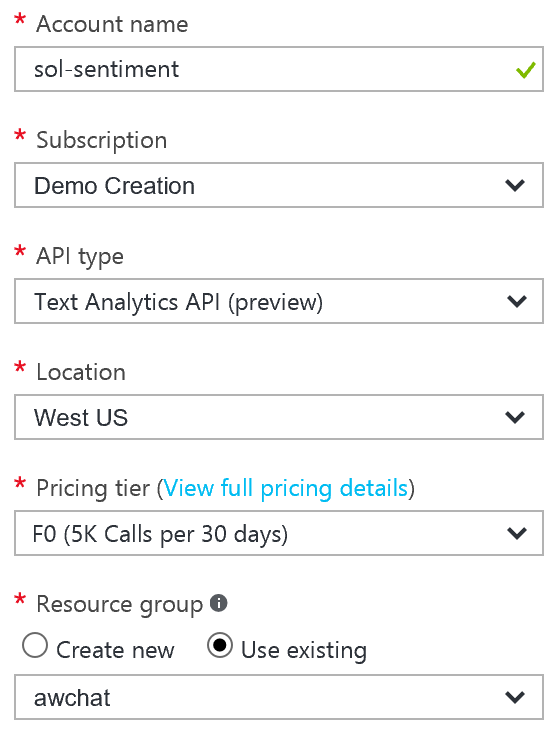
The EventProcessorHost requires an Azure Storage Account that it will use to manage its state among multiple instances. In this section, you create that Storage Account.

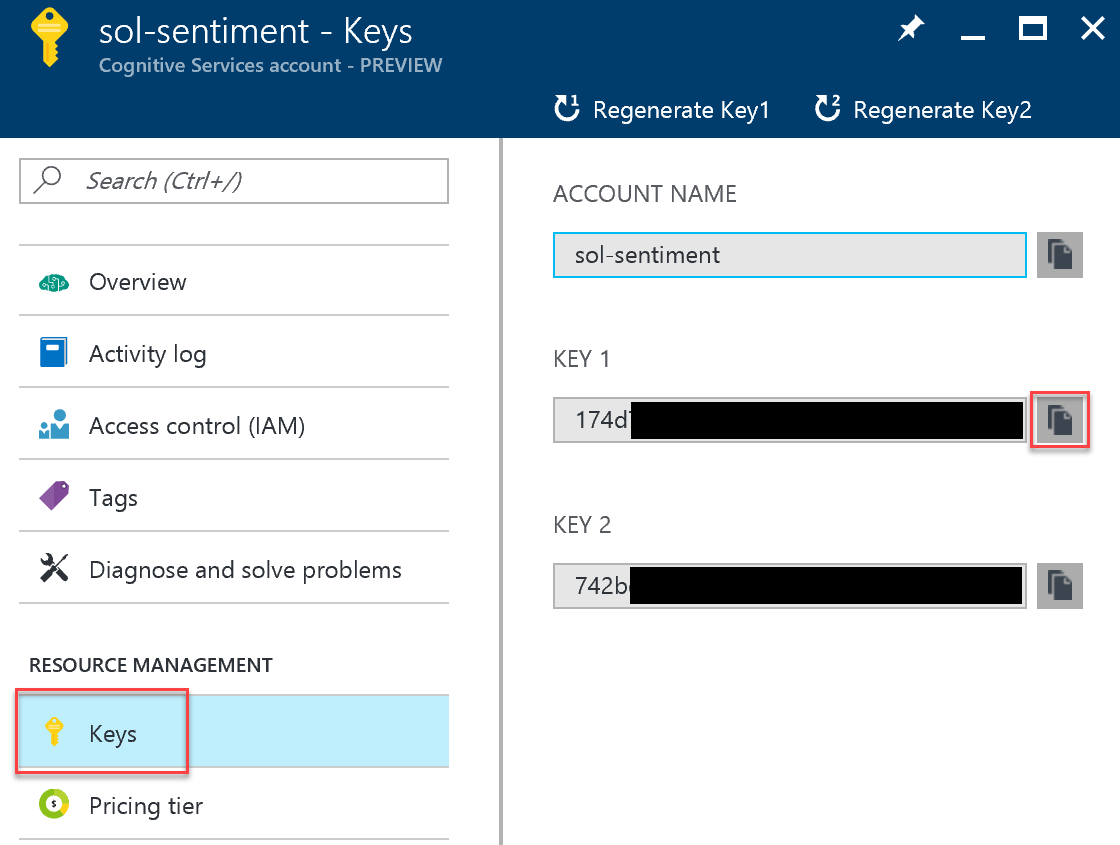
1. Using the Azure Portal, select + New, Storage, Storage account.
2. In the Create storage account blade, provide a unique name for the account.
3. Leave Resource Manager selected for the deployment model.
4. Leave General purpose selected for the Account kind.
5. Set the Performance to Standard, and the Replication to LRS.
6. Select Disabled for the Storage service encryption.
7. Choose your Subscription, Resource Group, and Location to be consistent with the other resources you have created.  
    
8. Select Create.  
   

### Task 10: Cognitive Services

To provision access to the Text Analytics API (which provides sentiment analysis features), you will need to provision a Cognitive Services account.

1. In the Azure Portal, select + NEW.
2. Select Intelligence + analytics.
3. Select Cognitive Services APIs.   
   
4. Provide an Account Name.
5. Select API type and choose Text Analytics API.
6. For the Pricing tier, select F0.
7. Choose an existing Resource Group (or create a new one as desired).
8. Choose a Location near you.



1. Select Create to provision the Cognitive Services account.
2. When it finishes provisioning, acquire the key for the API by selecting Keys on the left-hand menu, and copying the value for Key 1.  
    
3. Repeat steps 1-9, this time selecting Bing Speech API for the API Type. Take note of Key 1 for Speech.
4. Repeat steps 1-9, this time selecting Language Understanding Intelligent Service (LUIS) for the API Type. Take note of Key 1 for LUIS.

## Exercise 2: Implement message forwarding

Duration: 45 minutes

In this section, you will implement the message forwarding from the ingest Event Hub instance to an Event Hub instance and a Service Bus Topic. You will also configure the web-based components, which consist of three parts: the Web App UI, a Web Job that runs the EventProcessorHost, and the API App that provides a wrapper around the Search API.

### Task 1: Implement the event processor

1. With the starter solution open in Visual Studio, open SentimentEventProcessor.cs (found within the ChatMessageSentimentProcessor project).
2. Scroll down to the IEventProcessor.ProcessEventsAsync method. This method represents the heart of the message processing logic utilized by the Event Processor Host running in a Web Job. It is provided a collection of EventData instances, each of which represent a chat message in the solution.
3. Locate TODO: 1 and replace the lines that follow the comment with the following:

//TODO: 1.Extract the JSON payload from the binary message

var eventBytes = eventData.GetBytes();

var jsonMessage = Encoding.UTF8.GetString(eventBytes);  
Console.WriteLine("Message Received. Partition '{0}', SessionID '{1}' Data '{2}'", context.Lease.PartitionId, eventData.Properties["SessionId"], jsonMessage);

1. Locate TODO: 2 and replace the line that follows with:

//TODO: 2.Deserialize the JSON message payload into an instance of MessageType

var msgObj = JsonConvert.DeserializeObject<MessageType>(jsonMessage);

1. Locate TODO: 3 and replace the line that follows with:

//TODO: 3. Create a BrokeredMessage (for Service Bus) and EventData instance (for EventHubs) from source message body

var updatedEventBytes = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(msgObj));

BrokeredMessage chatMessage = new BrokeredMessage(updatedEventBytes);

EventData updatedEventData = new EventData(updatedEventBytes);

1. Locate TODO: 4 and replace the lines that follow with:

//TODO: 4.Copy the message properties from source to the outgoing message instances

foreach (var prop in eventData.Properties)

{

chatMessage.Properties.Add(prop.Key, prop.Value);

updatedEventData.Properties.Add(prop.Key, prop.Value);

}

1. Locate TODO: 5 and replace the line that follows with:

//TODO: 5.Send chat message to Topic

\_topicClient.Send(chatMessage);

Console.WriteLine("Forwarded message to topic.");

1. Locate TODO: 6 and replace the line that follows with:

//TODO: 6.Send chat message to next EventHub (for archival)

\_eventHubClient.Send(updatedEventData);

Console.WriteLine("Forwarded message to event hub.");

1. Save the file.

### Task 2: Configure the Chat Message Processor Web Job

Within Visual Studio Solution Explorer, expand the ChatMessageSentimentProcessor project and open App.Config. You will update the appSettings in this file. The following sections walk you through the process of retrieving the values for the following settings:

<add key="eventHubConnectionString" value="" />

<add key="sourceEventHubName" value="" />

<add key="destinationEventHubName" value="" />

<add key="storageAccountName" value="" />

<add key="storageAccountKey" value="" />

<add key="serviceBusConnectionString" value="" />

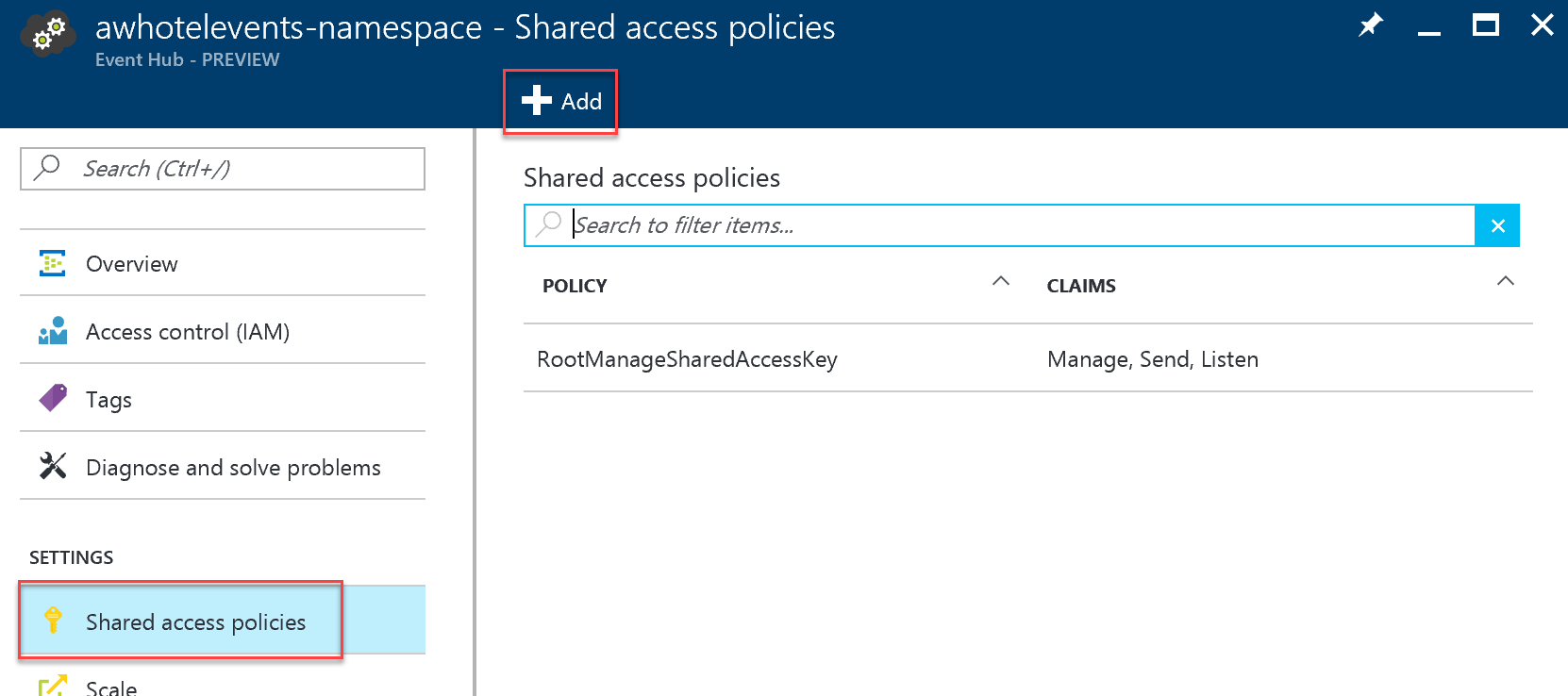
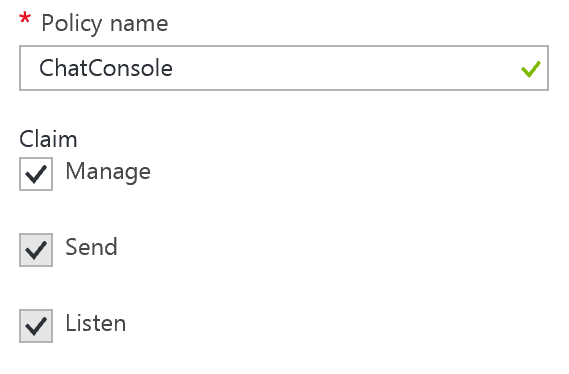
<add key="chatTopicPath" value="" />

<add key="textAnalyticsAccountName" value="" />

<add key="textAnalyticsAccountKey" value="" />

#### Event Hub Connection String

The connection string required by the ChatMessageSentimentProcessor is different from the typical Event Hub consumer, because not only does it need Listen permissions, but it also needs Send and Manage permissions on the Service Bus Namespace (because it receives messages, as well as creates Subscriptions).

1. To get the eventHubConnectionString, navigate to the Event Hub namespace in the Azure Portal.
2. Select Shared access policies within the left-hand menu.
3. In the Shared access policies, you are going to create a new policy that the ChatConsole can use to retrieve messages. Click + Add.  
   
4. For the New Policy Name, enter ChatConsole.
5. In the list of Claims, select Manage, Send, and Listen.  
    
6. Select Create.  
   
7. After the ChatConsole shared access policy is created, select it from the list of policies, and then copy the Connection String – Primary Key value.



1. Return to the app.config and paste this as the value for eventHubConnectionString.

#### Event Hub Name

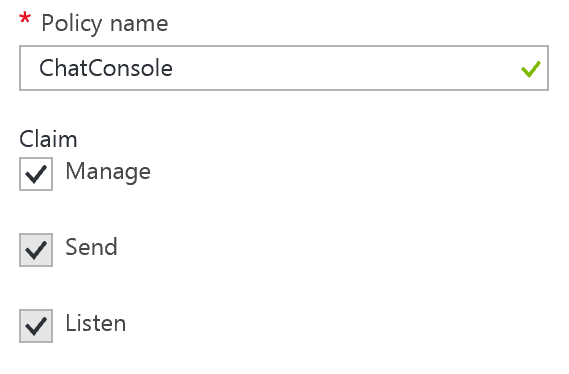
1. For the sourceEventHubName setting in app.config, enter the name of your first Event Hub.
2. For the destinationEventHubName, enter the name of your second Event Hub.

#### Storage Account

1. For the storageAccountName enter the name of the storage account you created.
2. For the storageAccountKey enter the Key for the storage account you created (which you can retrieve from the Portal).

#### Service Bus Connection String

The namespace, and therefore connection string, for the service bus is different from the one for the event hub. As we did for the event hub, we need to create a shared access policy to allow the ChatMessageSentimentProcessor Manage, Send, and Listen permissions.

1. To get the serviceBusConnectionString, navigate to the Service Bus namespace in the Azure Portal.
2. Select Shared access policies within the left menu.
3. In the Shared access policies, you are going to create a new policy that the ChatConsole can use to retrieve messages. Click **+ Add**.  
   
4. For the New Policy Name, enter ChatConsole.
5. In the list of Claims, select Manage, Send, and Listen.  
    
6. Select Create.  
   
7. After the ChatConsole shared access policy is created, select it from the list of policies, and then copy the Connection String – Primary Key value.

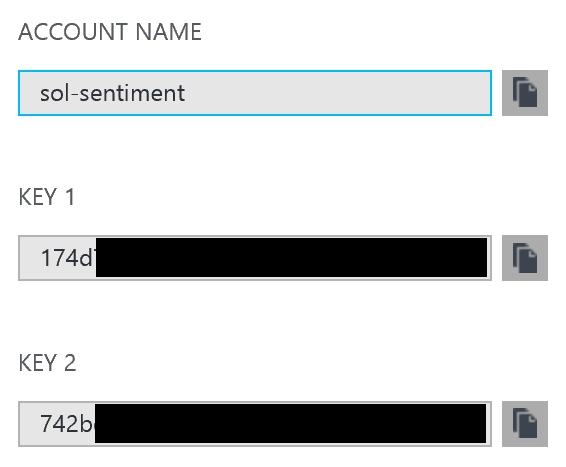


Return to the app.config and paste this as the value for serviceBusConnectionString.

#### Chat Topic

1. For the chatTopicPath, enter the name of the Service Bus Topic you had created (e.g., awhotel).

#### Text Analytics Settings

1. Using the Azure Portal, from the Cognitive Service account blade for the account you created for the Text API, copy the value under Endpoint into the textAnalyticsBaseUrl setting. Be sure to include a trailing slash in the URL (e.g. [https://westus.api.cognitive.microsoft.com/text/analytics/v2.0**/**)](https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/)).  
   ../../../../../../Captures/Screen%20Shot%202016-07-18%20at%203.31.48%20PM.png
2. On the left-hand menu of the Cognitive Service account blade, select Keys.  
    
3. Copy the value of Account Name into the value attribute of textAnalyticsAccountName in the app.config.
4. Copy the value of Key 1 from the blade into the value attribute of the textAnalyticsAccountKey in the app.config.

## Exercise 3: Configure the Chat Web App settings

Duration: 10 minutes

Within Visual Studio Solution Explorer, expand the ChatWebApp project and open Web.Config. You will update the appSettings in this file. The following sections walk you through the process of retrieving the values for the following settings:

<add key="eventHubConnectionString" value=" "/>

<add key="eventHubName" value=" "/>

<add key="serviceBusConnectionString" value=" "/>

<add key="chatRequestTopicPath" value=" "/>

<add key="chatTopicPath" value=" "/>

#### Event Hub Connection String

1. Use the same connection string you used for the eventHubConnectionString in the Web Job project.

#### Event Hub Name

1. For the eventHubName setting in Web.config, enter the name of your first Event Hub. This event Hub will receive messages from the website chat clients.

#### Service Bus Connection String

1. Use the same connection string you used for the serviceBusConnectionString in the Web Job project.

#### Chat Topic Path and Chat Request Topic Path

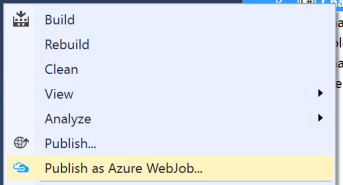
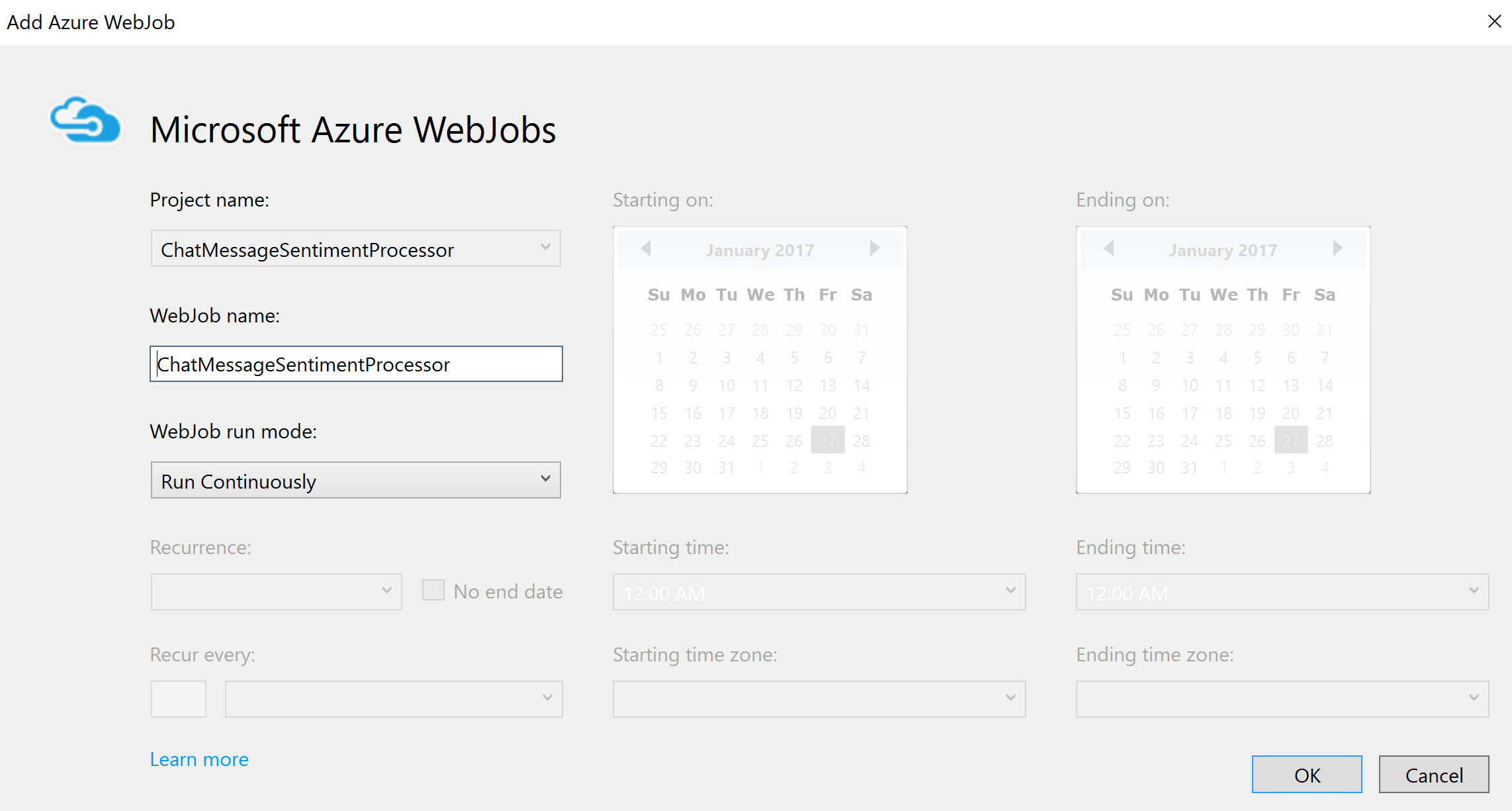
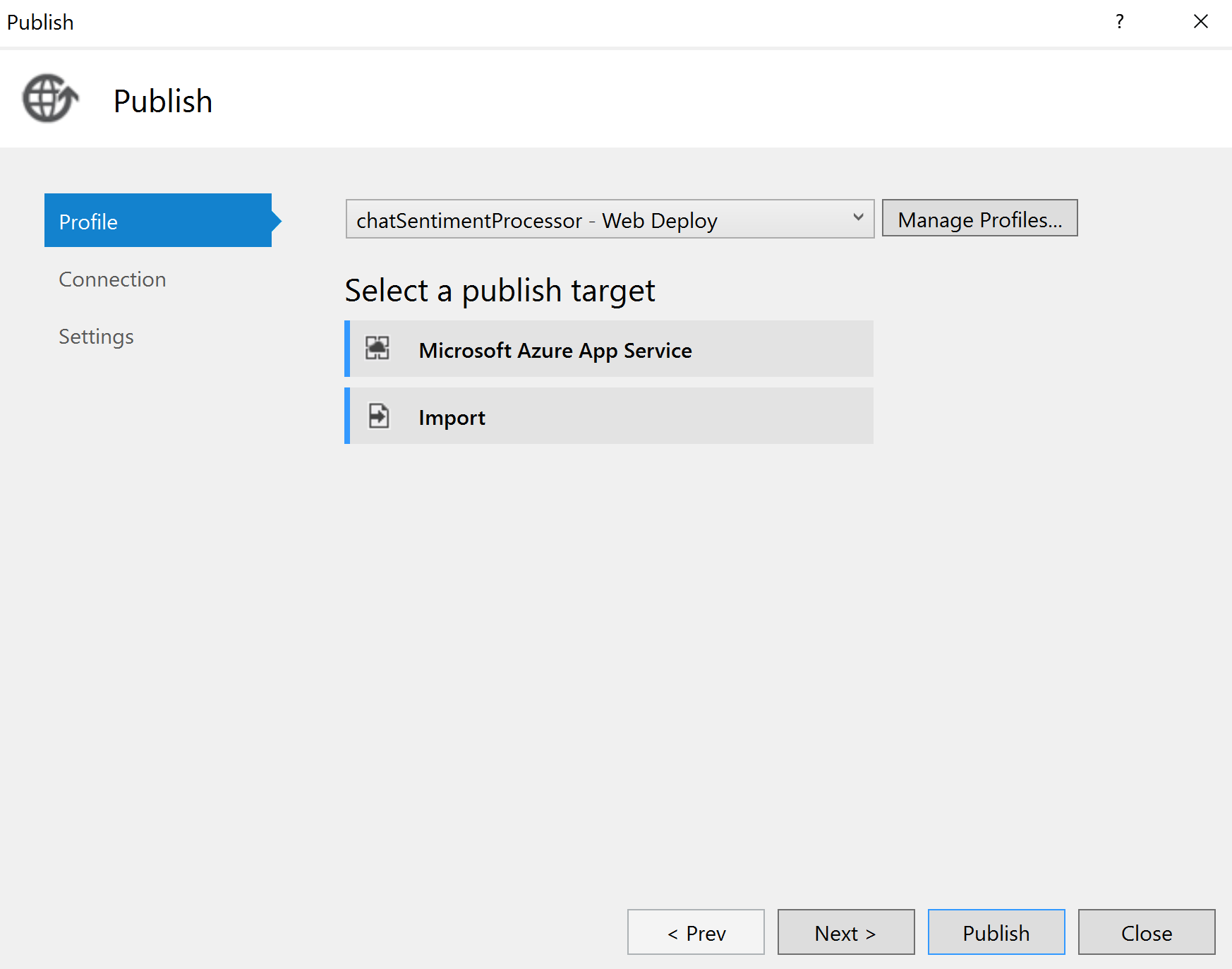
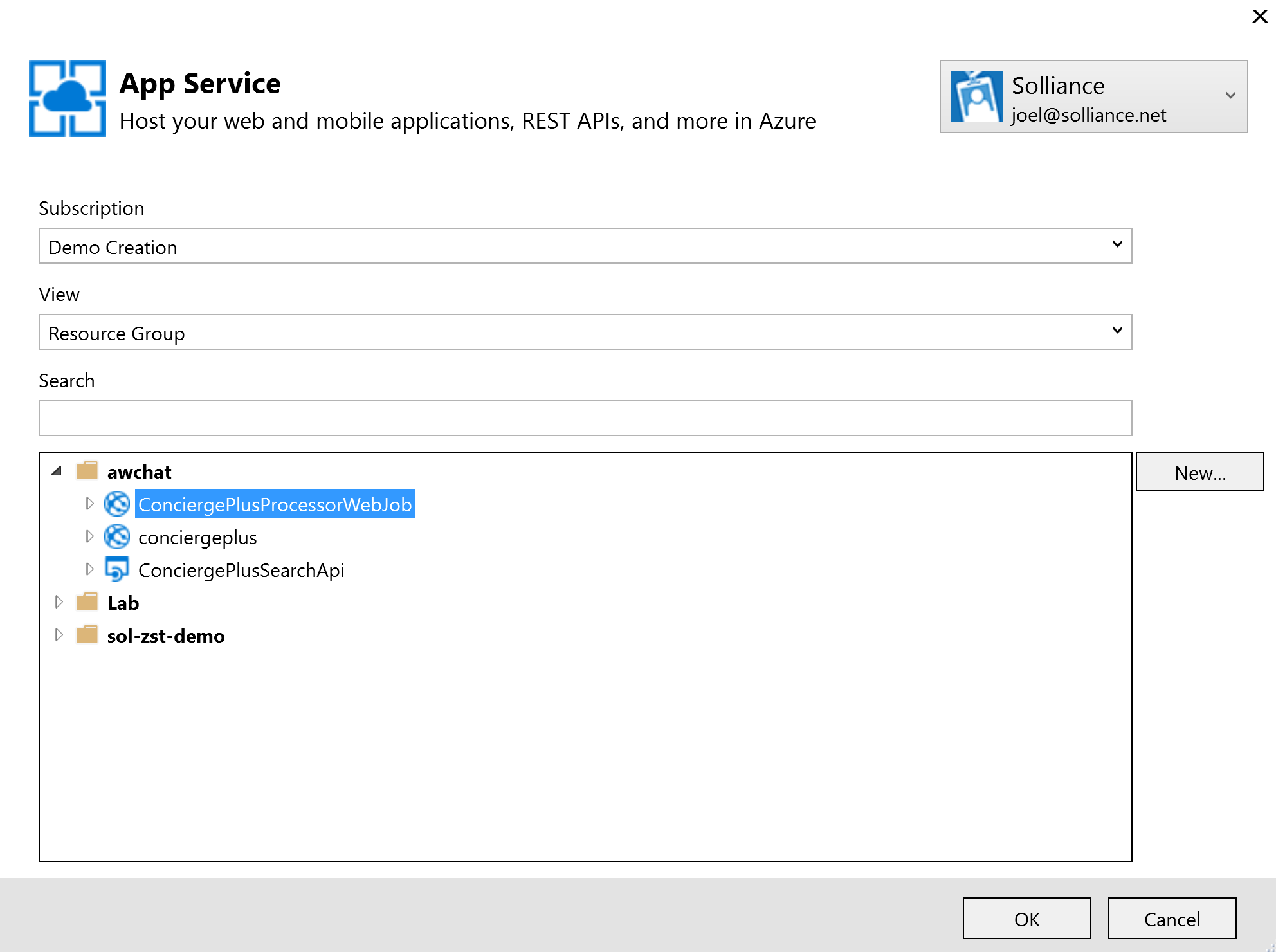
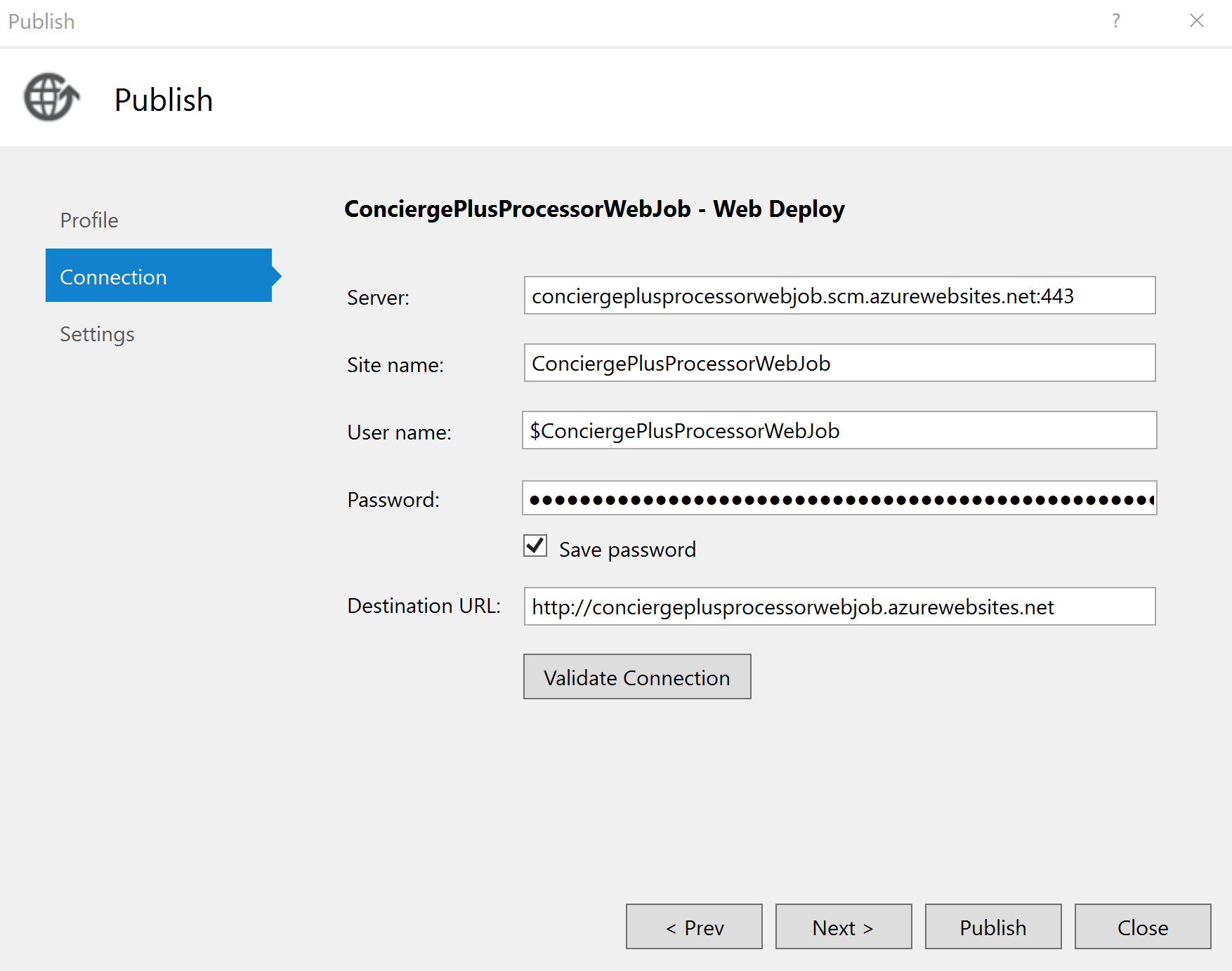
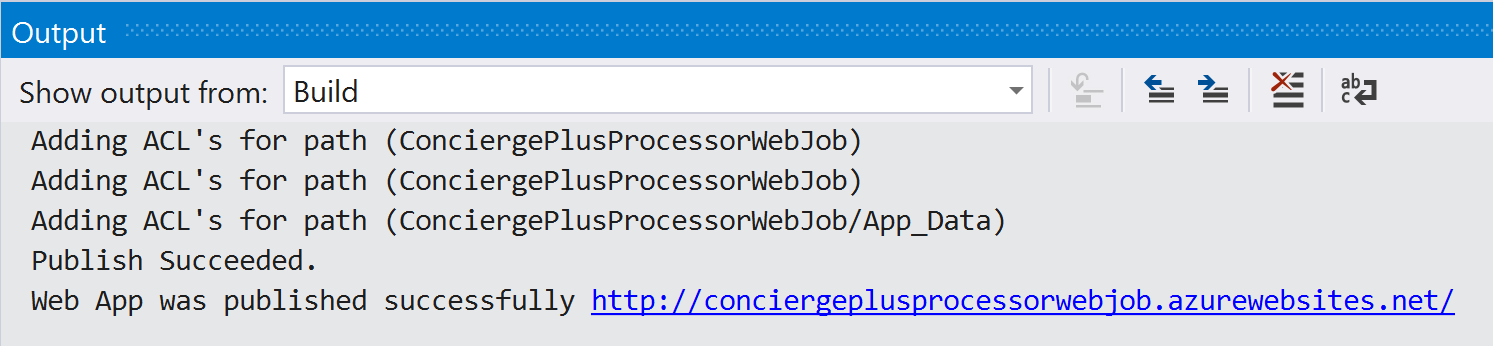
1. For the chatTopicPath and chatRequestTopicPath, enter the name of the Service Bus Topic you had created. The value is the same for both settings in this case.

## Exercise 4: Deploying the App Services

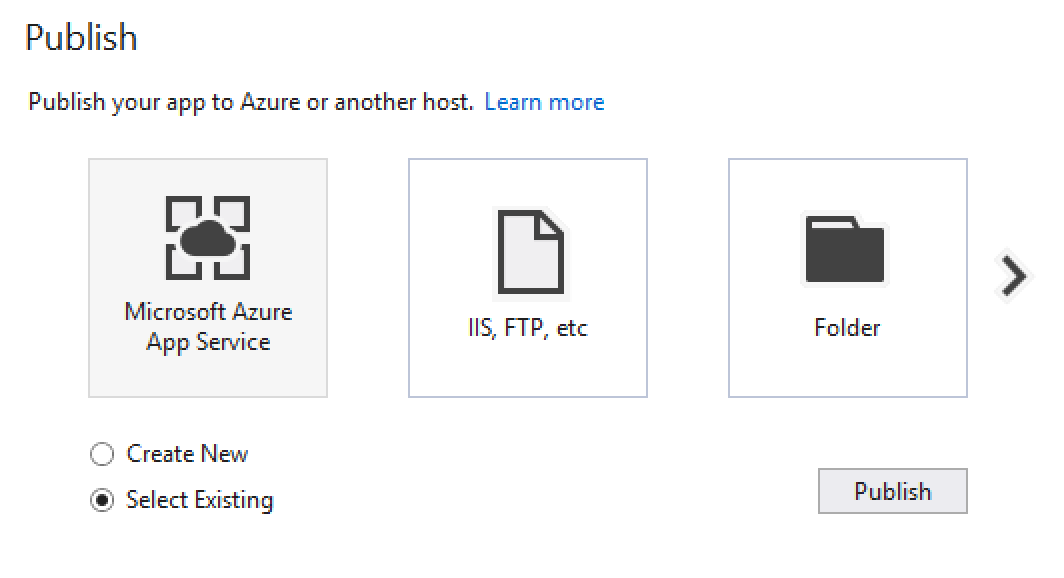
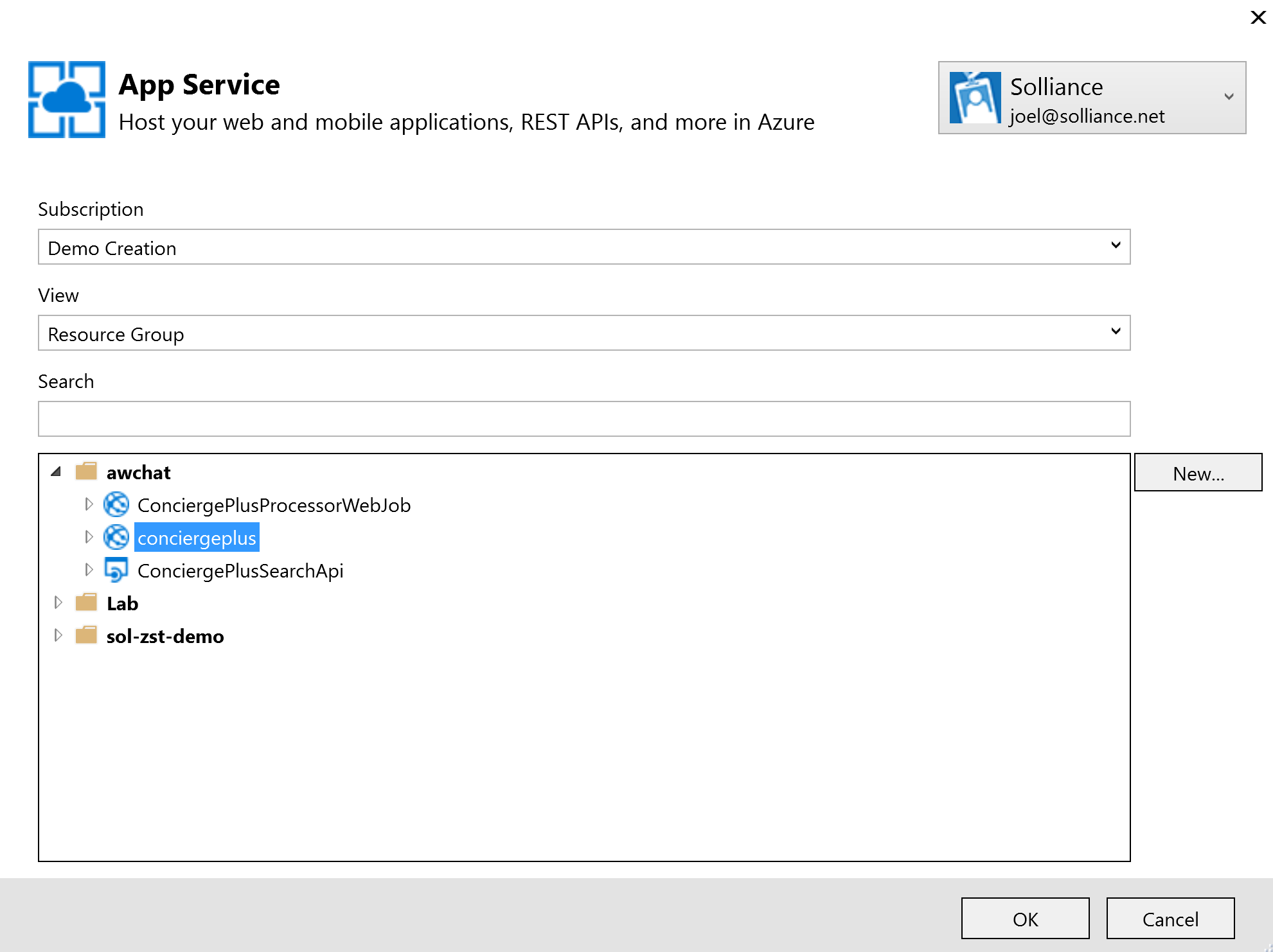
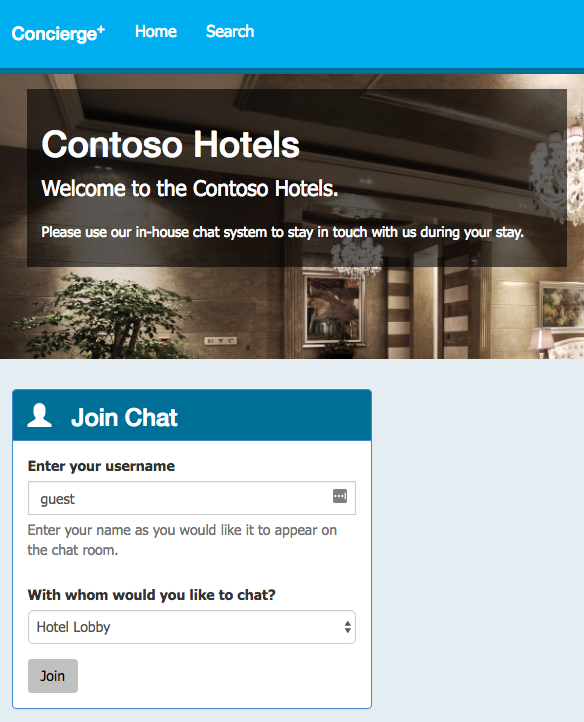
Duration: 15 minutes

With the App Services projects properly configured, you are now ready to deploy them to their pre-created services in Azure.

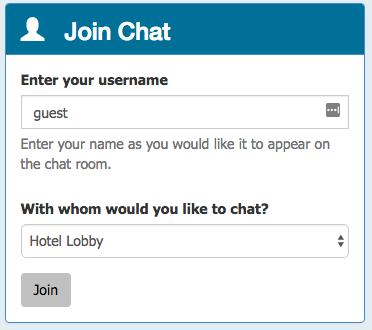
### Task 1: Publish the ChatMessageSentimentProcessor Web Job

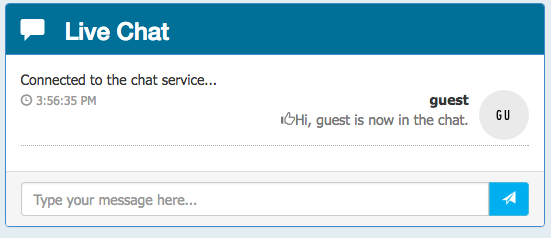
1. Within Visual Studio Solution Explorer, right-click the ChatMessageSentimentProcessor project and select Publish as Azure Web Job.  
   
2. The Add Azure WebJob dialog should appear. Leave the WebJob run mode setting to Run Continuously and select OK. 
3. In the Publish Web dialog, select Profile, and then select Microsoft Azure App Service publish target.  
    
4. In the App Service dialog, choose your Subscription that contains your Web Job Web App you provisioned earlier. Expand your Resource Group (e.g., awchat), then select the node for your Web Job Web App in the tree view to select it.  
    
5. Select OK.
6. Select Publish.  
    
7. When the publish completes, the Output window should indicate success similar to the following:  
   

### Task 2: Publish the ChatWebApp

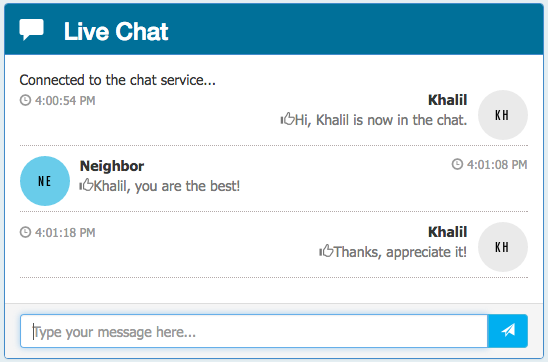
1. Within Visual Studio Solution Explorer, right-click the ChatWebApp project and select Publish.  
   
2. In the Publish document, select Microsoft Azure App Service and choose the Select existing radio button. Select Publish.  
   
3. In the App Service dialog, choose your Subscription that contains your Web App you provisioned earlier. Expand your Resource Group (e.g., awchat), then select the node for your Web App in the tree view to select it.  
    
4. Select OK.
5. When the publishing is complete, a browser window should appear with content like the following.  
   

### Task 3: Testing hotel lobby chat

1. Open a browser instance (Chrome is recommended for this web app), and navigate to the deployment URL for your Web App.
2. Under the Join Chat area, enter your username (anything will do).
3. Leave Hotel Lobby selected.
4. Select Join.  
   
5. The Live Chat should appear. (Notice it auto-announced you joining to the room; this is actually the first message.)



1. Open another browser instance. (You could try this from your mobile device.)
2. Enter another username and join.
3. From either session, fill in the Chat text box and select Send. You can try using @ and # too, just to seed some text for search.



You can join with as many sessions as you want. (The Hotel Lobby is basically a public chat room.)

## Exercise 5: Add intelligence

Duration: 60 minutes

In this exercise, you will implement code to activate multiple cognitive intelligence services that act on the chat messages.

### Task 1: Implement sentiment analysis

In this task, you will add code that enables the Event Processor to invoke the Text Analytics API using the REST API and retrieve a sentiment score (a value between 0.0, negative, and 1.0, positive sentiment) for the text of a chat message.

1. Open SentimentEventProcessor.cs in ChatMessageSentimentProcessor project using Visual Studio.
2. Scroll down to the method GetSentimentScore.
3. Replace the code following TODO: 7 with the following:

//TODO: 7.Configure the HTTPClient base URL and request headers

client.BaseAddress = new Uri(\_textAnalyticsBaseUrl);

client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", \_textAnalyticsAccountKey);

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

1. Replace the code following TODO: 8 with the following:

//TODO: 8.Construct a sentiment request object

var req = new SentimentRequest()

{

documents = new SentimentDocument[]

{

new SentimentDocument() { id = "1", text = messageText }

}

};

1. Replace the code following TODO: 9 with the following:

//TODO: 9.Serialize the request object to a JSON encoded in a byte array

var jsonReq = JsonConvert.SerializeObject(req);

byte[] byteData = Encoding.UTF8.GetBytes(jsonReq);

1. Replace the code following TODO: 10 with the following:

//TODO: 10.Post the rquest to the /sentiment endpoint

string uri = "sentiment";

string jsonResponse = "";

using (var content = new ByteArrayContent(byteData))

{

content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

var sentimentResponse = await client.PostAsync(uri, content);

jsonResponse = await sentimentResponse.Content.ReadAsStringAsync();

}

Console.WriteLine("\nDetect sentiment response:\n" + jsonResponse);

1. Replace the code following TODO: 11 with the following:

//TODO: 11.Deserialize sentiment response and extract the score

var result = JsonConvert.DeserializeObject<SentimentResponse>(jsonResponse);

sentimentScore = result.documents[0].score;

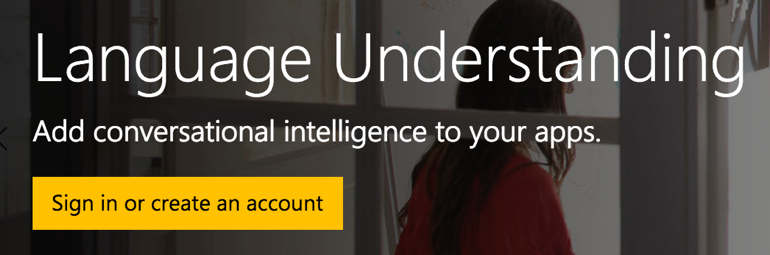
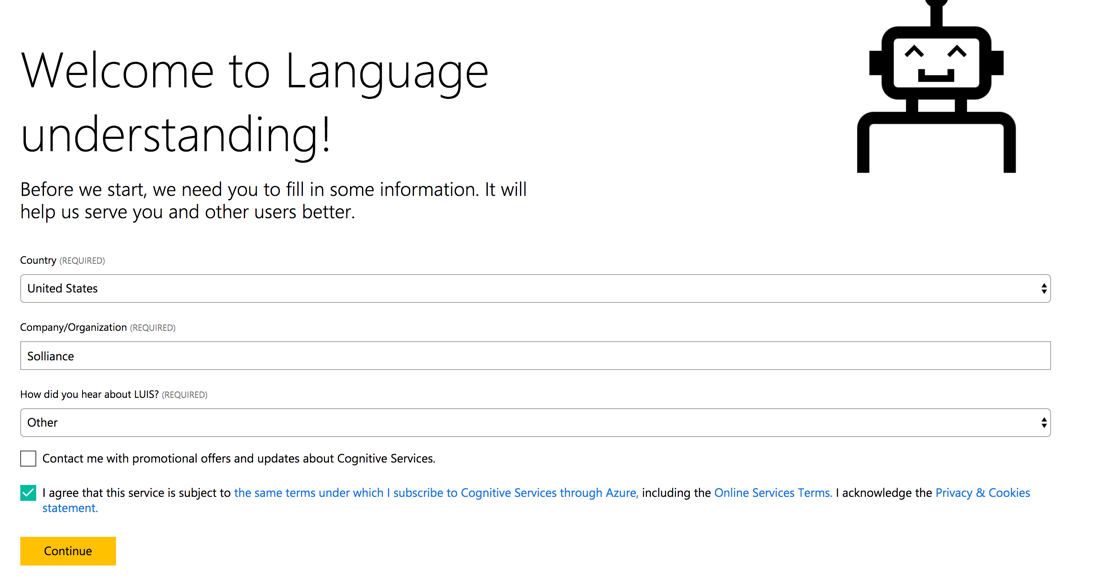
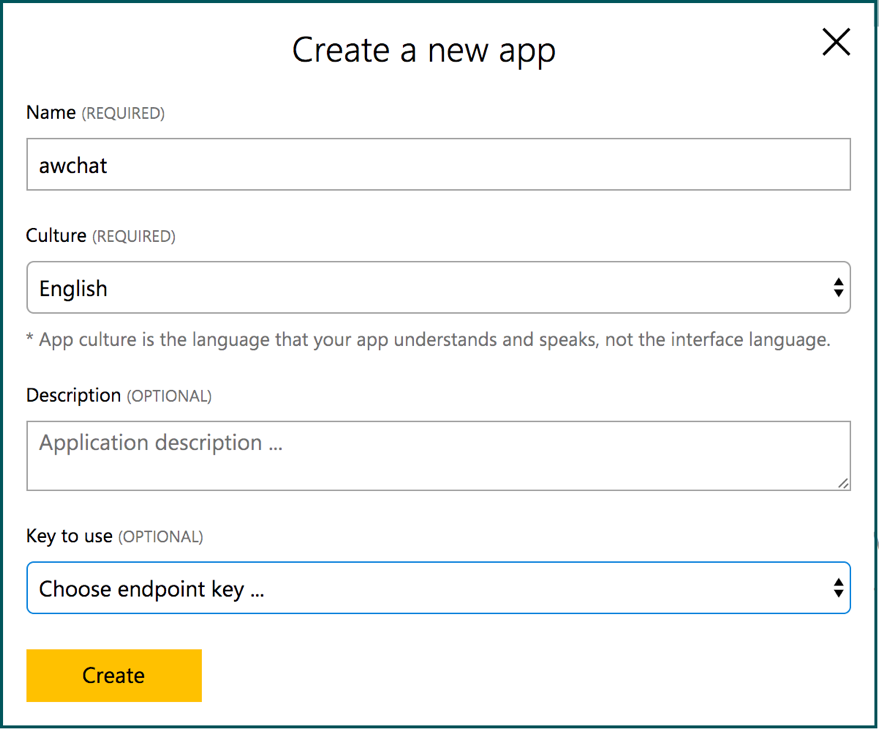
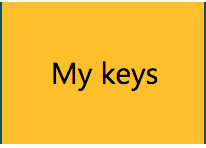
1. Finally, navigate to the IEventProcessor.ProcessEventsAsync and replace the line following TODO: 12 with the following code:

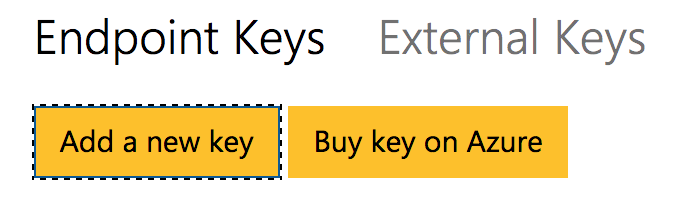
//TODO: 12 Append sentiment score to chat message object

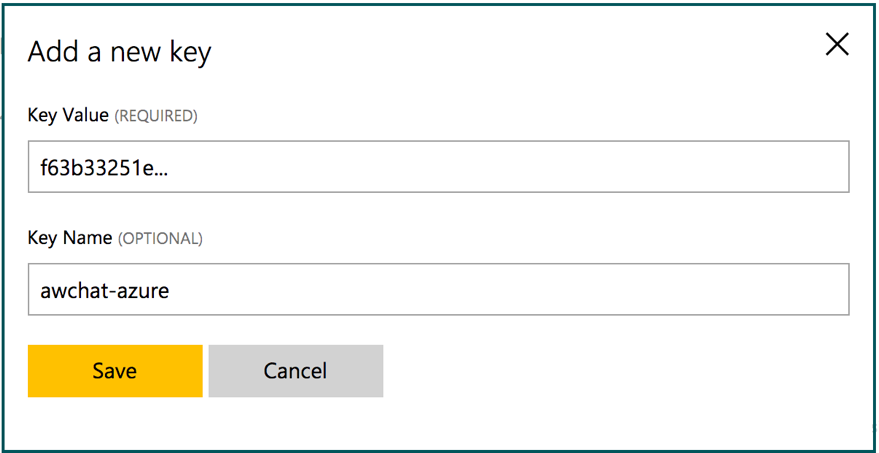
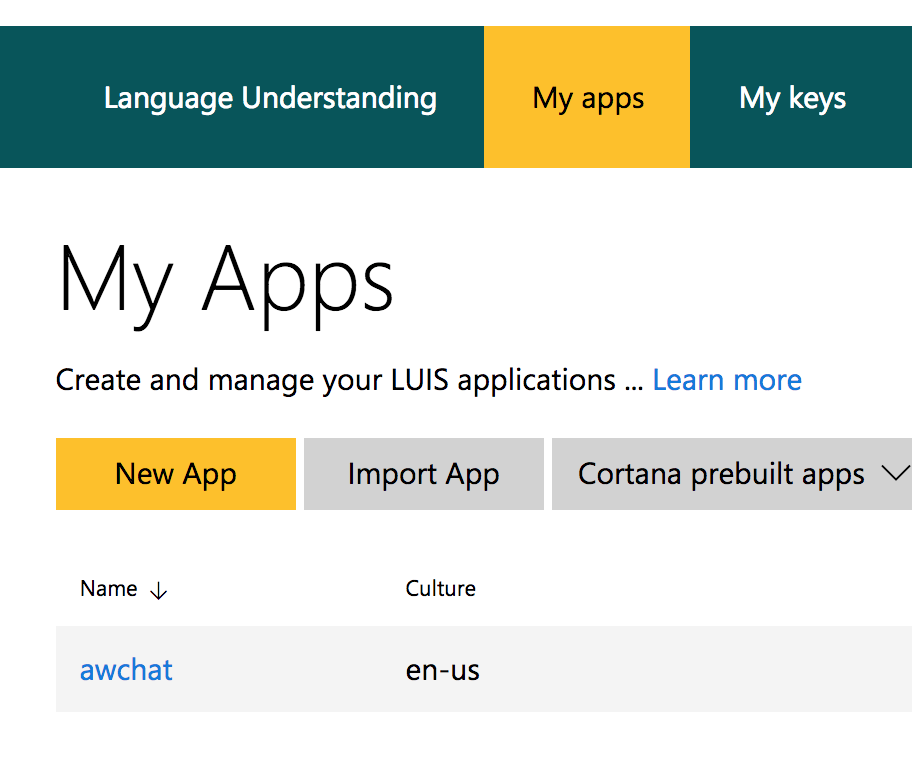
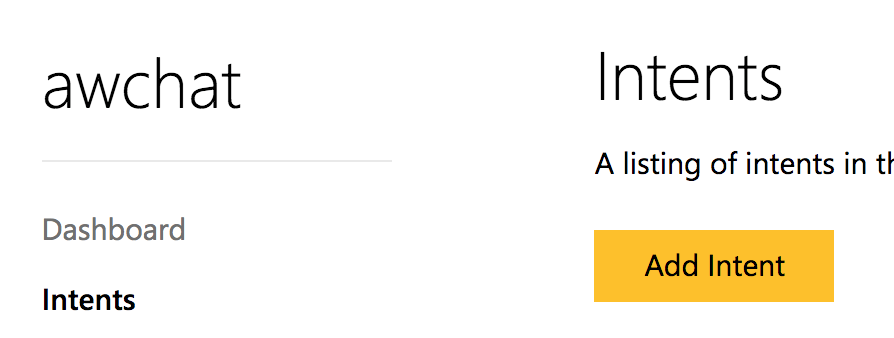
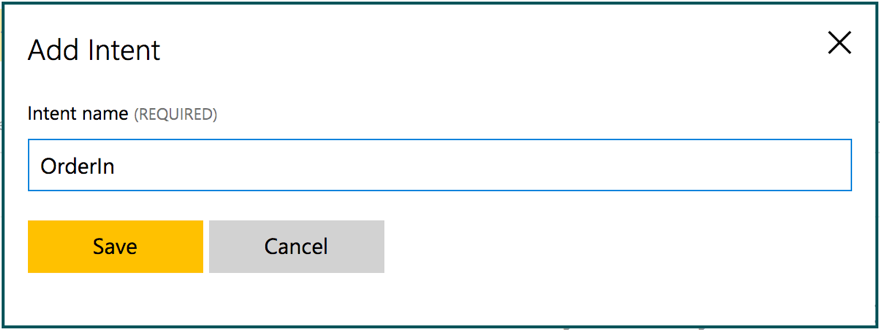
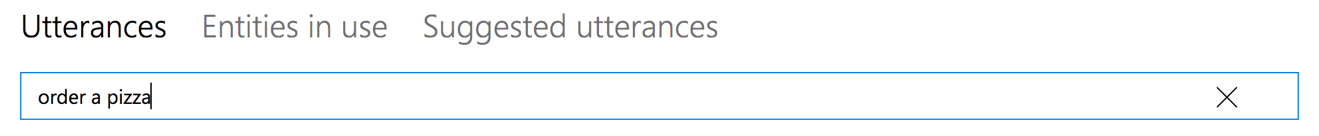
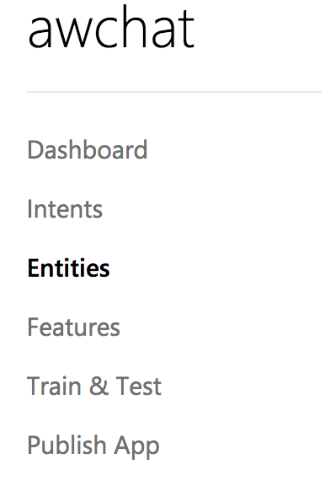
msgObj.score = await GetSentimentScore(msgObj.message);

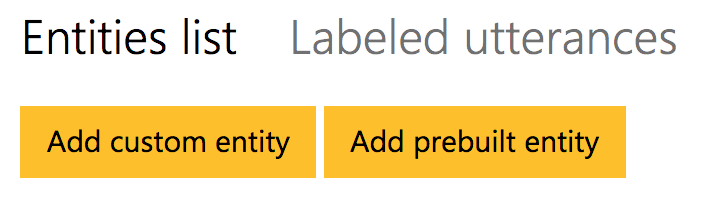
### Task 2: Implement linguistic understanding

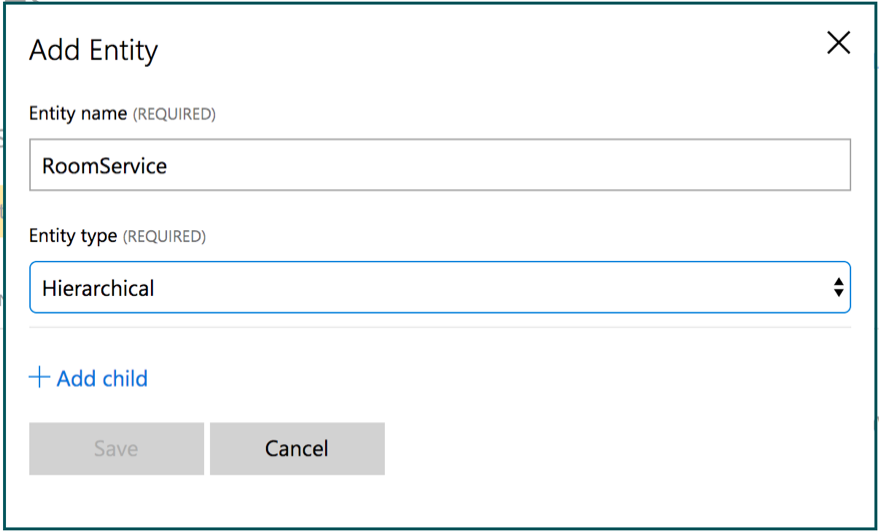
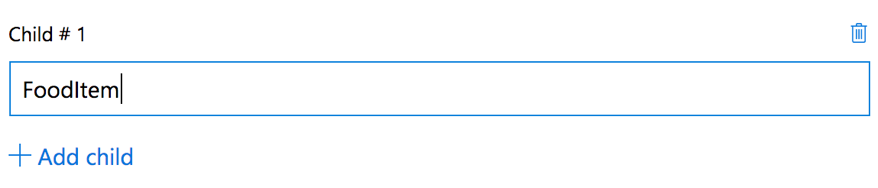
In this task, you will create a LUIS app, publish it, and then enable the Event Processor to invoke LUIS using the REST API.

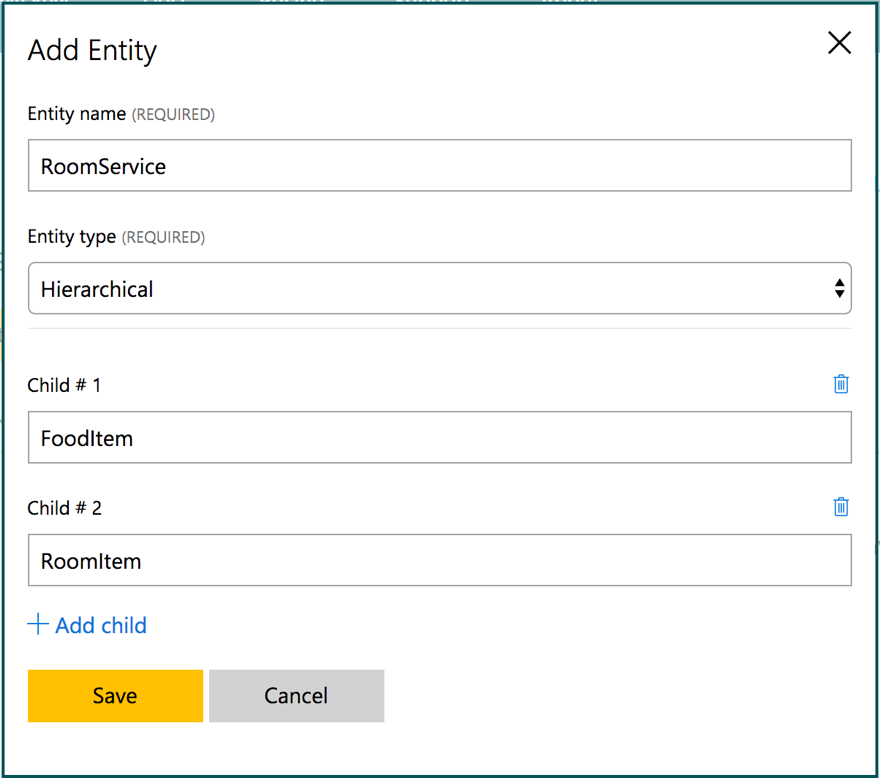
1. Using a browser, navigate to <http://www.luis.ai>.
2. Select Sign in or create an account.  
   
3. Sign in using your Microsoft account (or @Microsoft.com account if that is appropriate to you). The new account startup process may take a few minutes.
4. Complete the additional info and terms of use form and select Continue.  
   
5. Under My Apps, select New App.
6. Complete the Create a new app form by providing a name for your LUIS app, the culture and select Create.  
   
7. In a moment, your new app will appear.
8. In the menu bar, select My Keys.  
   
9. Select Add a new key.

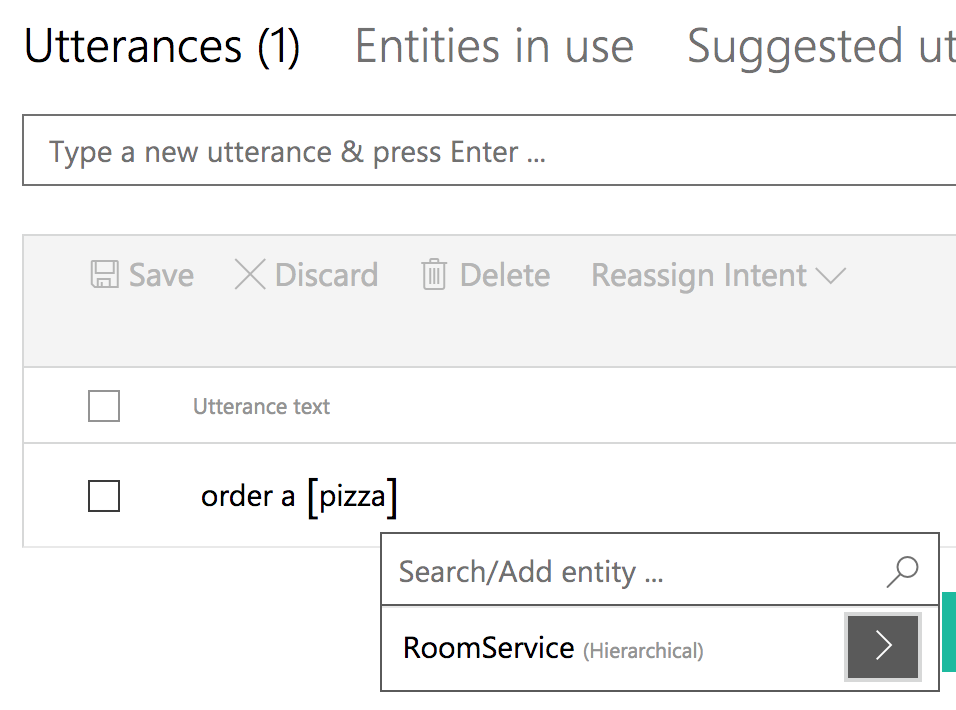
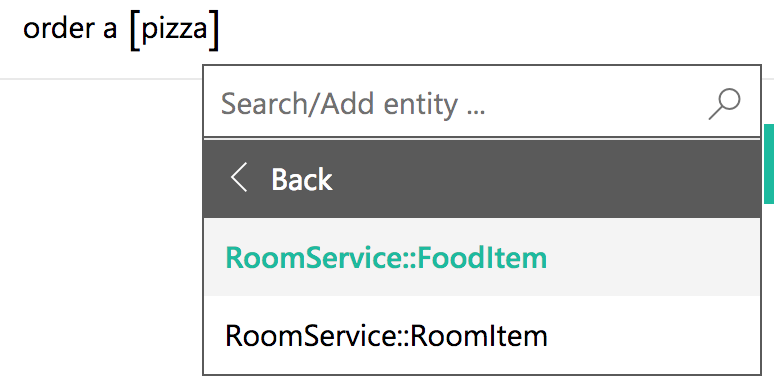
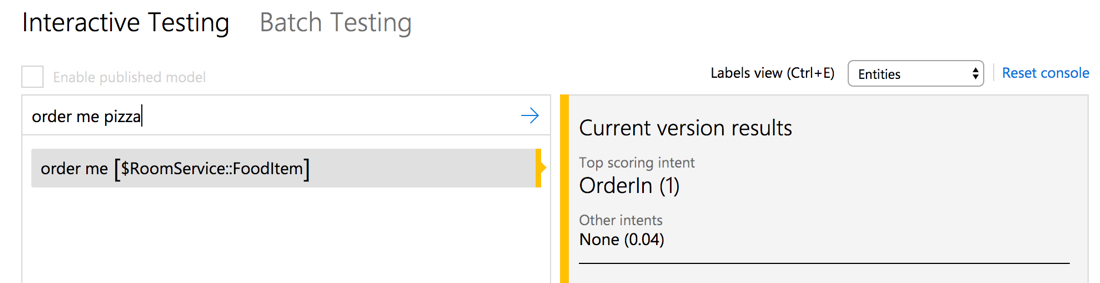
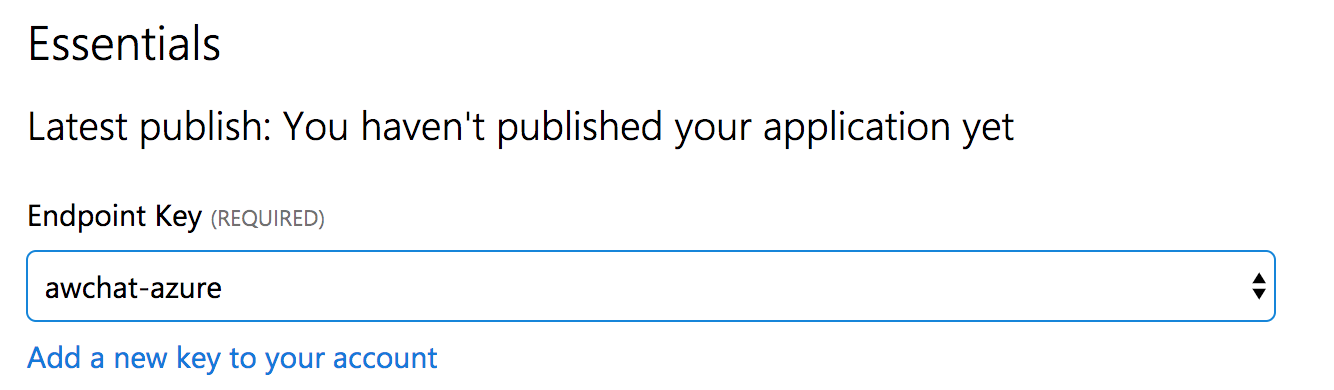
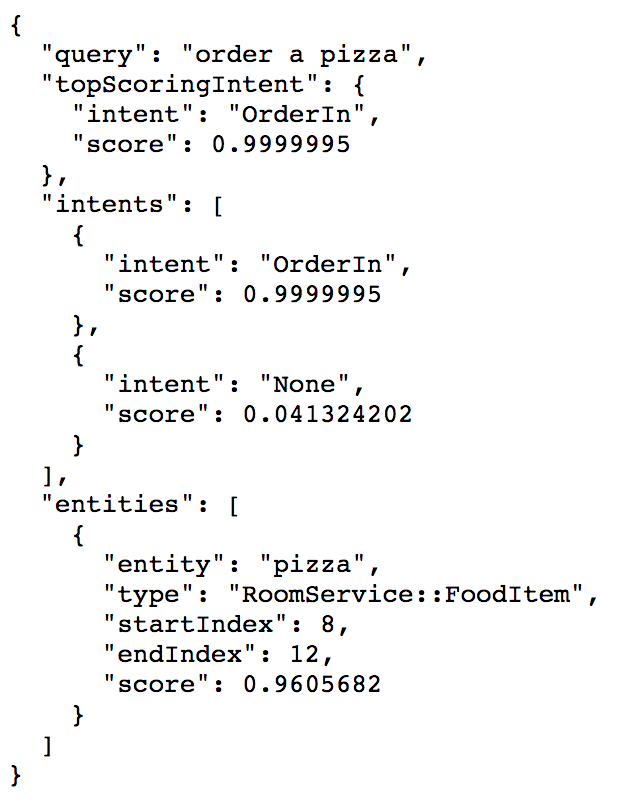
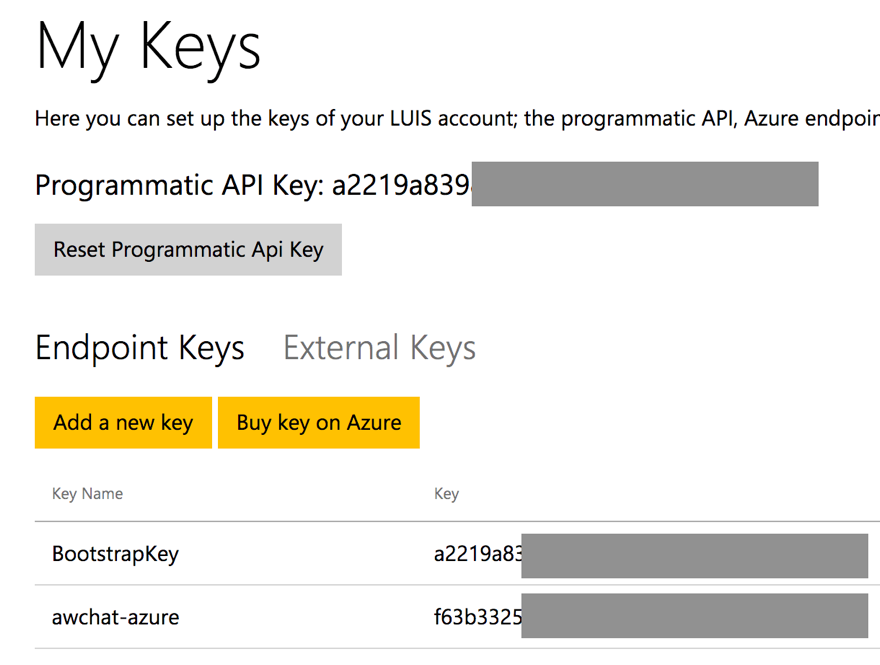


1. In the Add a new key text box, paste the value of Key 1 as you acquired it from the Azure Portal for your Cognitive Services account provisioned with the LUIS API and select Save.   
   
2. Select My Apps from the menu bar and choose your app from the list.  
   
3. Select Intent, then Add Intent.  
   
4. In the Intents dialog, for the Intent Name provide OrderIn and select Save.  
   
5. In the Utterances text box, enter “order a pizza”. Press Enter to add the utterance.  
   
6. Above the list of utterances, select Save.   
   
7. Select Entities from the menu on the left.   
   
8. Select Add custom entity.



1. For the Entity name specify “RoomService” and set the Entity Type to Hierarchical.  
   
2. Select + Add child.
3. For Child #1 provide a name of FoodItem.  
   
4. Select +Add child and provide a name of RoomItem. Select Save.



1. Select Intents from the menu on the left and select the OrderIn intent your created.
2. In the utterance, select the word pizza so it becomes highlighted.   
   
3. Expand the chevron next to RoomService and select FoodItem.  
   
4. Select Save.
5. In the Type a new utterance text box, enter the following utterance:
   1. Utterance: Bring me toothpaste
   2. Text to select: toothpaste
   3. Drop-down: OrderIn
   4. Entity: RoomService:RoomItem
6. Select Save.
7. Repeat this process for the following phrases (text to select is in bold):
   1. *Bring me* ***towels*** | RoomService:RoomItem
   2. *Bring me* ***blankets*** | RoomService:RoomItem
   3. *Order a* ***soda*** | RoomService:FoodItem
   4. *Order me a* ***pizza*** | RoomService:FoodItem
8. Select Save.
9. Select Train & Test from the menu on the left and then select Train.
   1. Experiment with the Interactive Testing by writing some utterances and pressing enter to see the interpretation.  
      
10. Select Publish App from the menu on the left.
11. From the left side menu, select the Publish button.
12. Under Essentials, select the Enpoint Key you created.  
    
13. Select Publish.
14. When the publish completes, click the URL displayed. It will open a new tab in your browser. Modify the end of the URL (the text following q= ) to it contains the phrase “order a pizza” and press ENTER. You should receive output similar to the following. Observe that it correctly identified the intent as OrderIn (in this case with a confidence of 0.9999995 or nearly 100%) and the entity as pizza having an entity type of RoomService::FoodItem (in this case with a confidence score of 96.1%).  
      
    
15. In the URL, take note of the GUID following apps/GUID/, this is your App ID and you will need to use it in configuration later, it similar to the following:  
      
    https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/73a52c62-ea2b-49b1-ba2c-5324edd9525a?subscription-key=f63b33251e4b19d4bd26b3ef35810e13&verbose=true&timezoneOffset=0.0&q=order%20a%20pizza
16. You can add more utterances as desired by repeating the above steps to add new utterances, indicate the entity, train the model, and then update the publish application using the button in the Publish screen.
17. When you are ready to integrate LUIS into your app, go to the My Keys in LUIS and copy the Endpoint Key (from Azure). You will enter this into the configuration of the Event Processor.  
    
18. In Visual Studio, open the App.config for the ChatMessageSentimentProcessor project.
19. Within the appSettings section, for the key luisAppId set the text of the value attribute to the App ID of your LUIS App (this value should be a GUID you obtained from the URL and not the name of your LUIS app). For the key luisKey, set the text of the value attribute to the Endpoint key used by your LUIS app (as you acquired it from the Azure Portal).  
    ../../../../../../Captures/Screen%20Shot%202016-07-15%20at%203.54.43%20PM.png
20. Save the app.config. The Event Processor is pre-configured to invoke the LUIS API using the provided App ID and key.
21. Open SentimentEventProcessor.cs and navigate to IEventProcessor.ProcessEventsAsync.
22. Locate TODO: 13 and replace it with the following:

//TODO: 13.Respond to chat message intent if appropriate

var intent = await GetIntentAndEntities(msgObj.message);

HandleIntent(intent, msgObj);

1. Take a look at the implementation of both methods if you are curious how the entity and intent information is used to generate an automatic chat message response from a bot.

### Task 3: Implement speech to text

There is one last intelligence service to activate in the application—speech recognition. This is powered by the Bing Speech API and is invoked directly from the web page without going through the web server. In the steps that follow, you insert your Cognitive Services Speech API key into the configuration to enable speech to text.

1. Within Visual Studio Solution Explorer, expand ChatWebApp, scripts, and open chatClient.js.
2. At the top, locate the variable speechApiKey and update its value with the Key 1 you acquired when you provisioned your Speech API in the Azure Portal.   
   ../../../../../../Captures/Screen%20Shot%202016-07-15%20at%204.29.29%20PM.png
3. Save chatClient.js.

**Note**: Embedding the API Key as shown here is done only for convenience. In a production app, you will want to maintain your API Key server-side.

### Task 4: Re-deploy and test

Now that you have added sentiment analysis, language understanding, and speech recognition to the solution, you need to re-deploy the apps so you can test out the new functionality.

1. Publish the ChatMessageSentimentProcessor Web Job using Visual Studio just as you did in Exercise 4, Task 1.
2. Publish the ChatWebApp just as you did in Exercise 4, Task 2.
3. When both have published, navigate to your deployed web app making sure to use HTTPS. (This is required for most browsers to support the microphone needed for speech recognition.)
4. Join a chat with the Hotel Lobby.
5. Type a message with a positive sentiment, like “I love this weather.” Observe the “thumbs-up” icon that appears next to the chat message you sent. This is an indicator of sentiment (as applied by your solution in real-time).
6. Next, try ordering something from room service, like “bring me towels” and see that you get a response from the ConciergeBot.
7. Finally, instead of typing your text, select the microphone to the left of the text box and speak for 2 to 3 seconds. Your spoken message should appear. Select the paper airplane icon to send it.

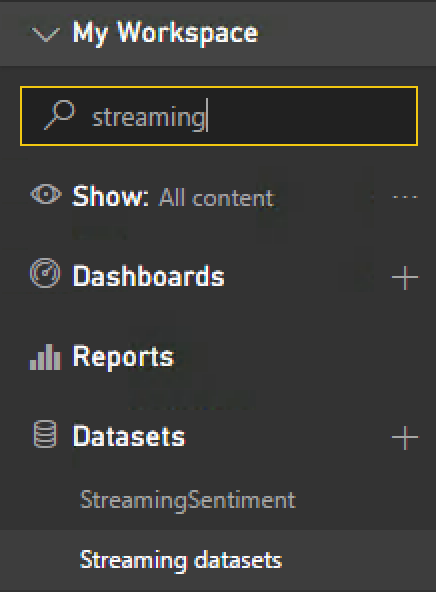
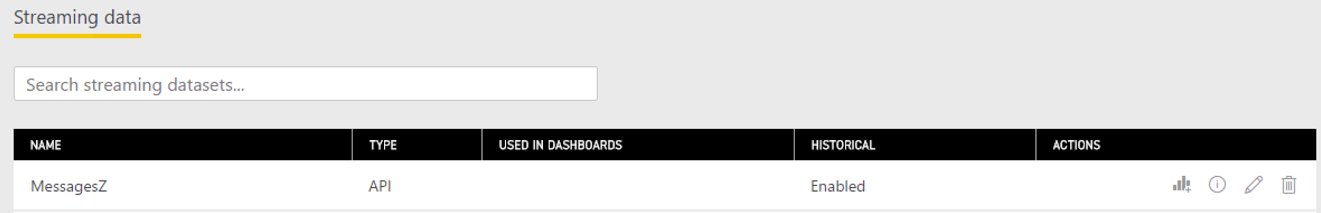
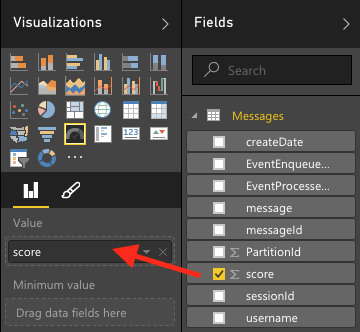
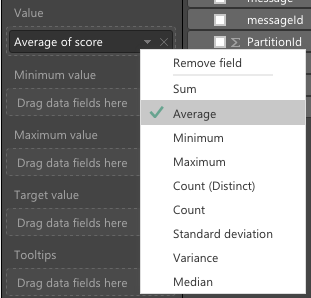
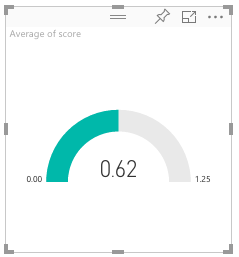
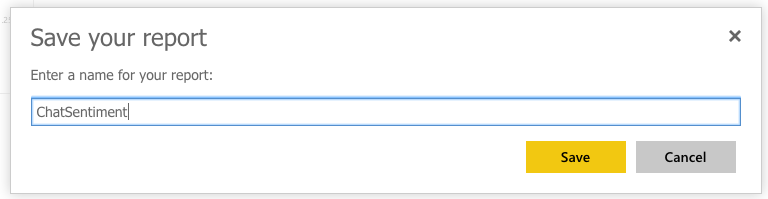


## Exercise 6: Building the Power BI dashboard

Duration: 30 minutes

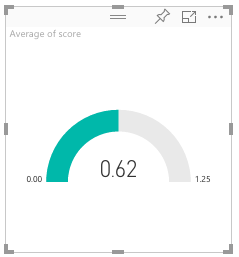
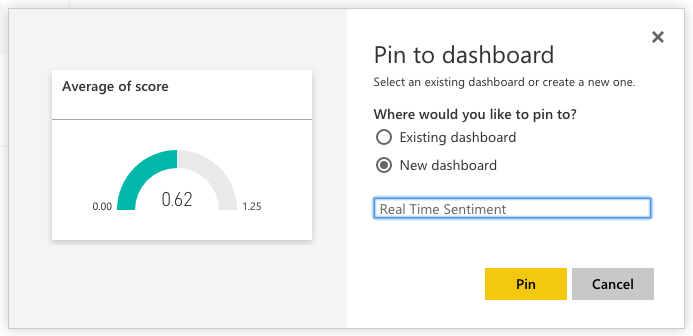
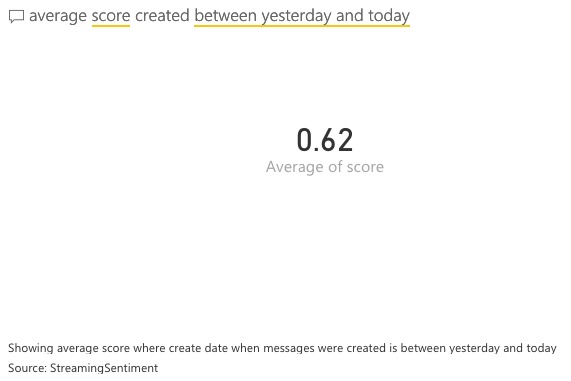
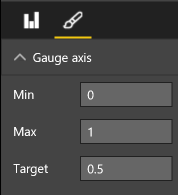
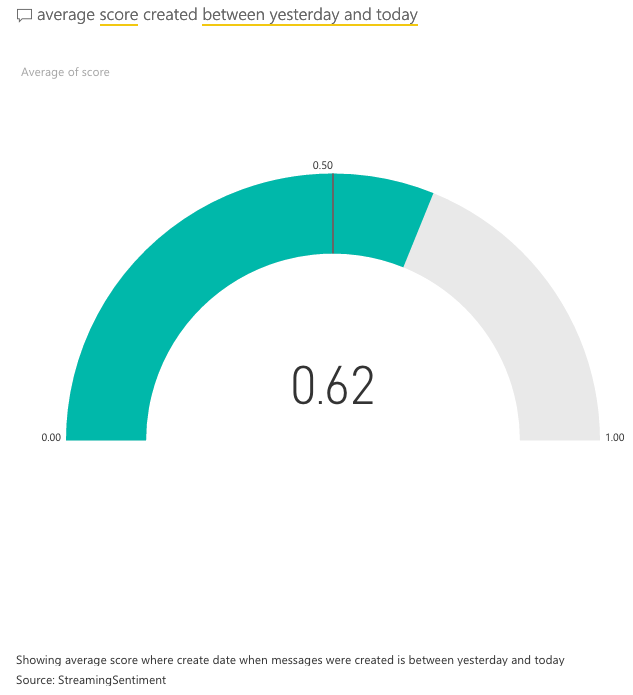
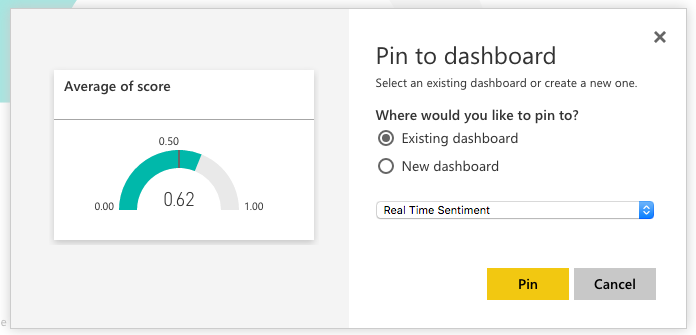
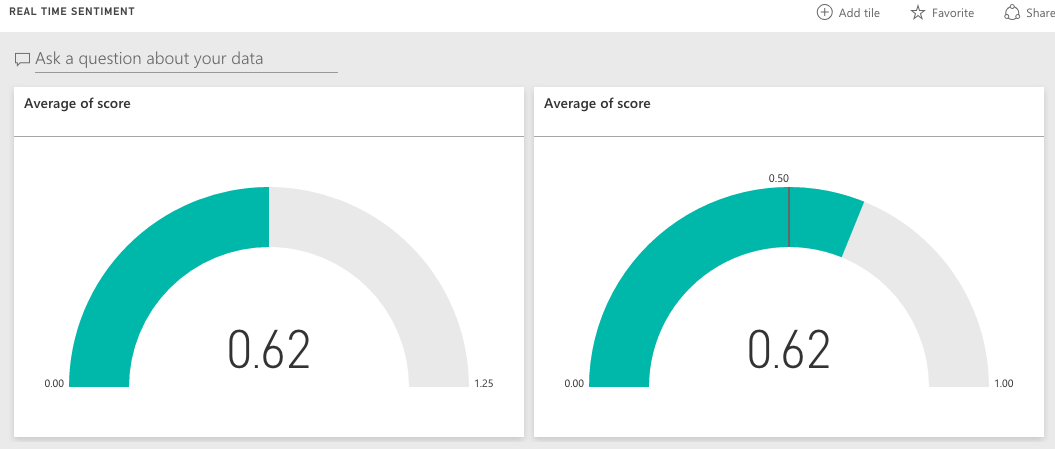
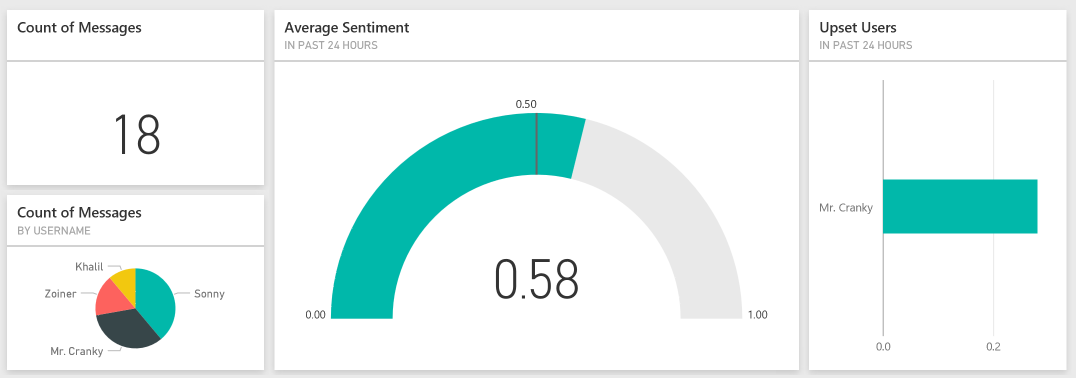
Now that you have the solution deployed and exchanging messages, you can build a Power BI dashboard that monitors the sentiments of the messages being exchanged in real time. The following steps walk through the creation of the dashboard.

### Task 1: Create the static dashboard

1. Sign in to your Power BI subscription (<https://app.powerbi.com)>.
2. Under the Datasets list, look for Streaming datasets.  
   
3. In the Streaming data list, look for the dataset named as you had configured it in the Output for the Stream Analytics job.  
   
4. Select the + Create Report button under the Actions column.  
   
5. On the Visualizations palette, select Gauge to create a semi-circular gauge.   
   
6. In the Fields listing, select and drag the score field and drop it onto the Value field.  
   
7. Select the drop down menu that appears where you dropped score and select Average.  
   
8. You now should have a gauge that shows the average sentiment for all of the data collected so far, which should look similar to the following:  
   
9. Select Save to save your visualization to a new report.  
   

### Task 2: Create the real-time dashboard

This gauge is currently a static visualization. You will use the report just created to seed a dashboard whose visualizations update as new messages arrive.

1. Select the Pin icon located near the top right of the Gauge control.  
   
2. Select New dashboard, provide a name, and select Pin.  
   
3. In the list of dashboards, select your newly created dashboard.  
   ../../../../../../Captures/Screen%20Shot%202016-06-21%20at%204.26.27%20PM.png
4. Real-time dashboards are created in Power BI using the QA feature, by typing in a question to visualize in the space provided. In the “Ask a question about your data” field, enter: “average score created between yesterday and today”.  
   
5. Next, convert this to a Gauge chart by expanding the Visualizations palette at right, and selecting the Gauge control.
6. Format the Gauge control so it ranges between 0.0 and 1.0 and has an indicator at 0.5. To do this, select the brush icon in the Visualization palette, expand the Gauge axis, and for Min enter 0, Max enter 1, and Target enter 0.5.   
   
7. Your gauge should now look similar to the following:  
   
8. In the top-right corner, select Pin visual.  
   ../../../../../../Captures/Screen%20Shot%202016-06-21%20at%204.35.57%20PM.png
9. In the dialog that appears, select the dashboard you recently created and select Pin.  
   
10. In the list of dashboards, select your dashboard. Your new gauge should appear next to your original gauge. You can delete the original gauge if you prefer. (Select the top of the visualization, then ellipses that appear, and the, the trash can icon.)  
    
11. Navigate to the chat website you deployed and send some messages and observe how the sentiment gauge updates with moments of you sending chat messages.
12. Try building out the rest of the real-time dashboard that should look as follows. We provide the following QA questions you can use to get started.  
    
    1. Count of Messages (Card visualization): count of messages between yesterday and today
    2. Count of Messages by Username (Pie chart visualization): count of messages by username between yesterday and today
    3. Upset Users (Bar chart visualization): Average score by username between yesterday and today

Invite some peers to chat and monitor the sentiments using your new, real-time dashboard.

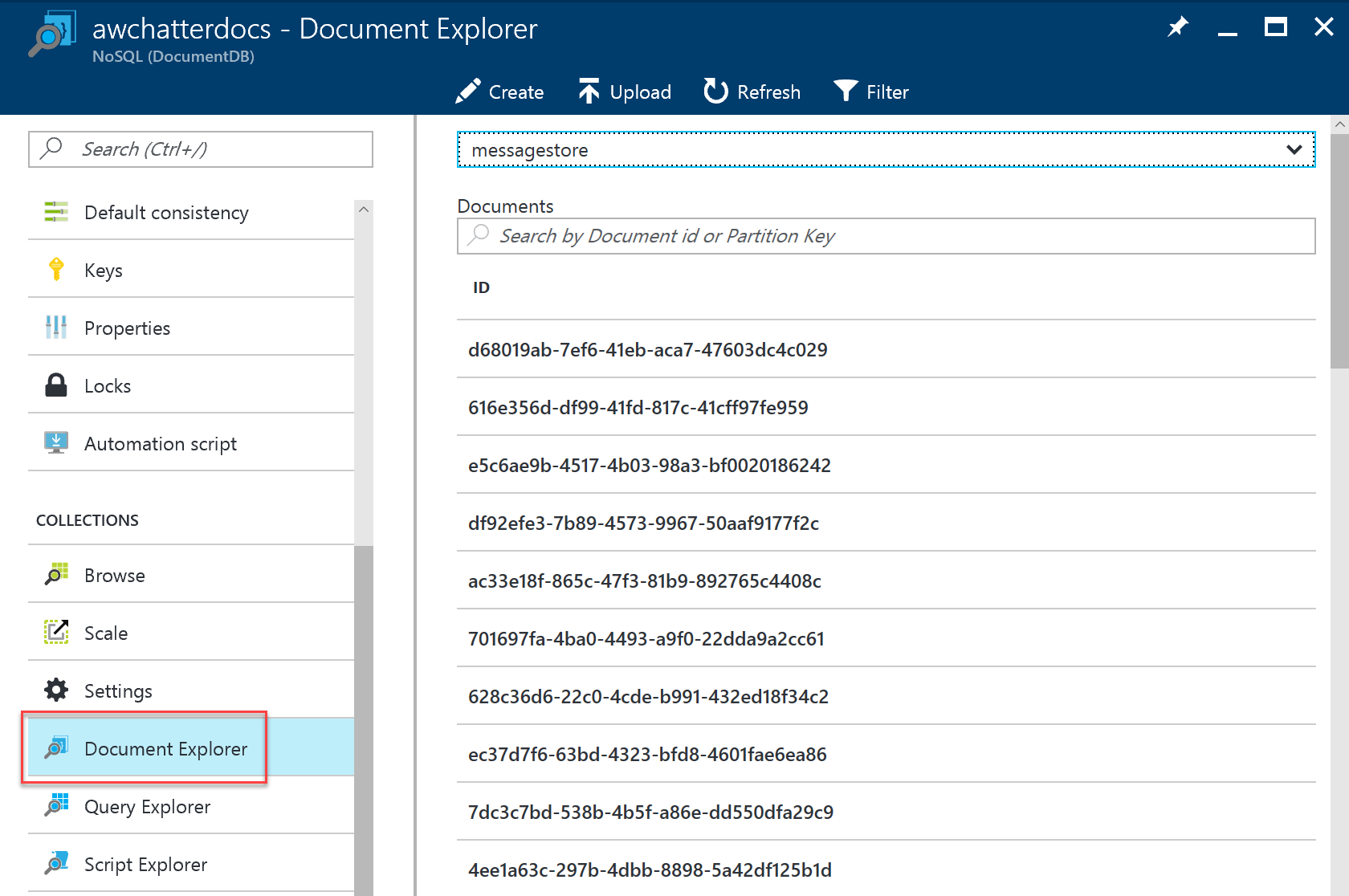
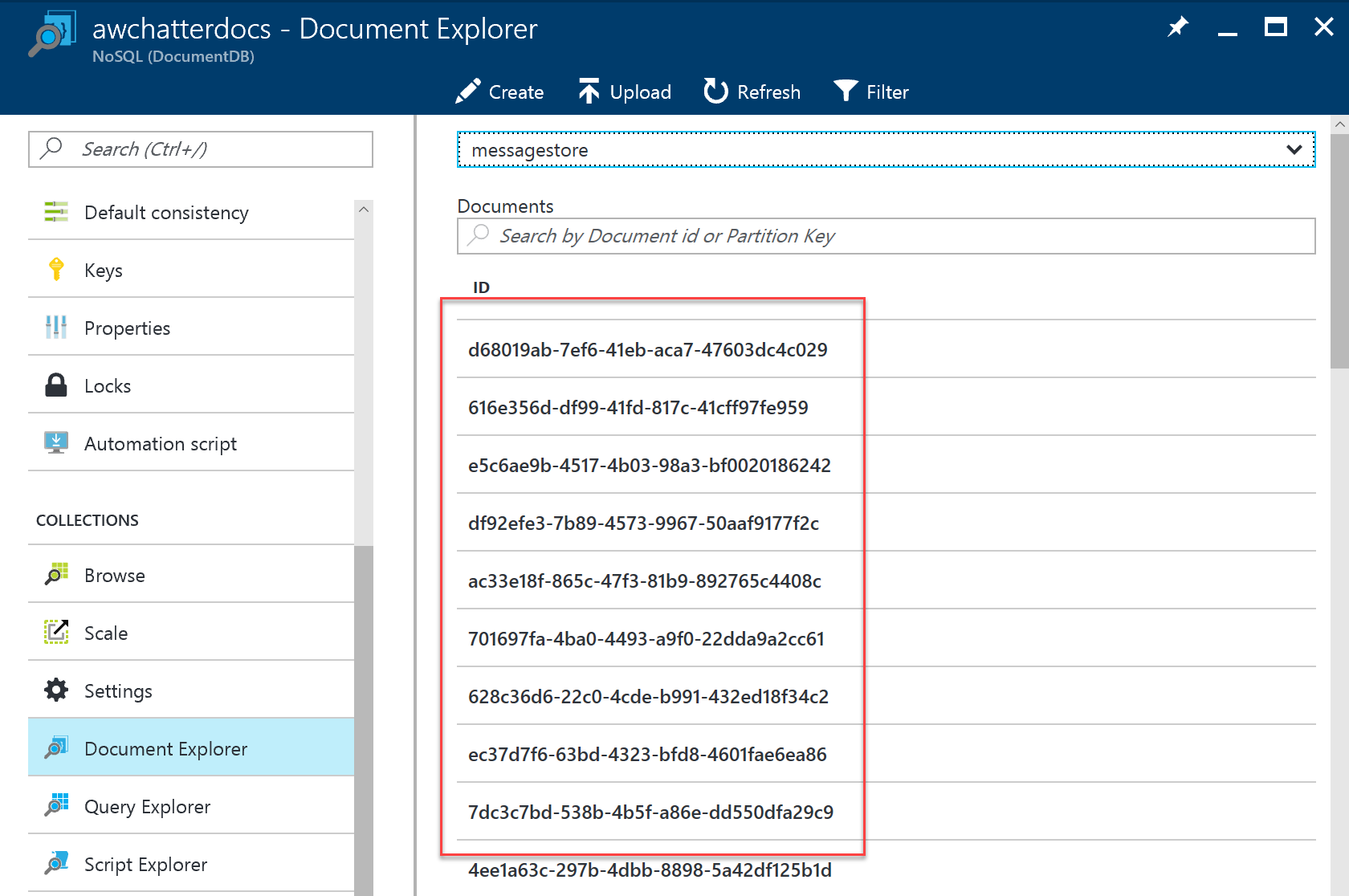
## Exercise 7: Enabling search indexing

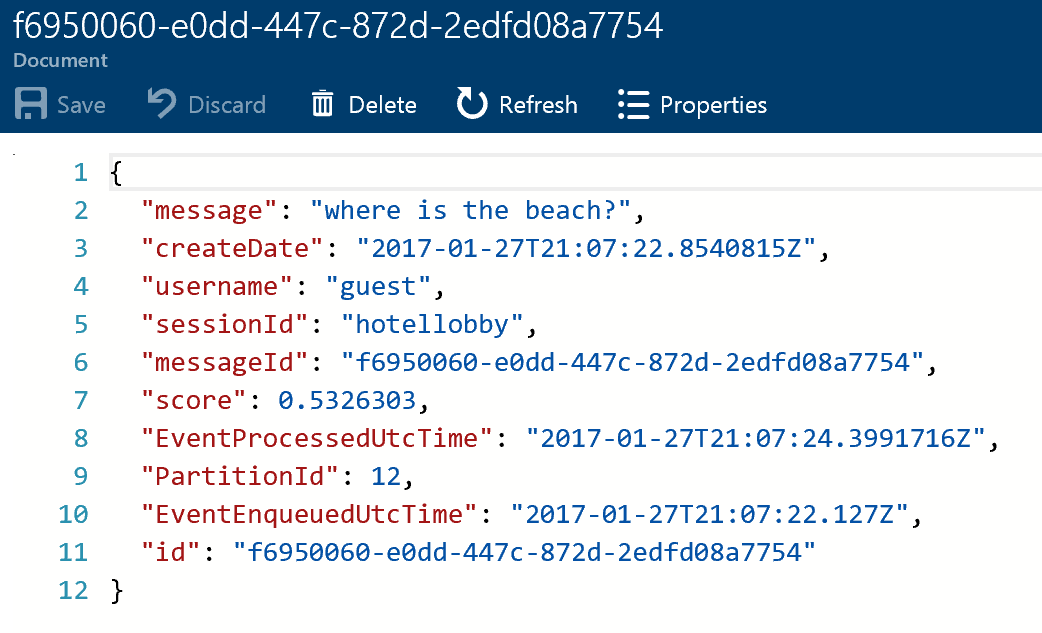
Duration: 30 minutes

Now that you have primed the system with some messages, you will create a Search Index and an Indexer in Azure Search upon the messages that are collected in DocumentDB.

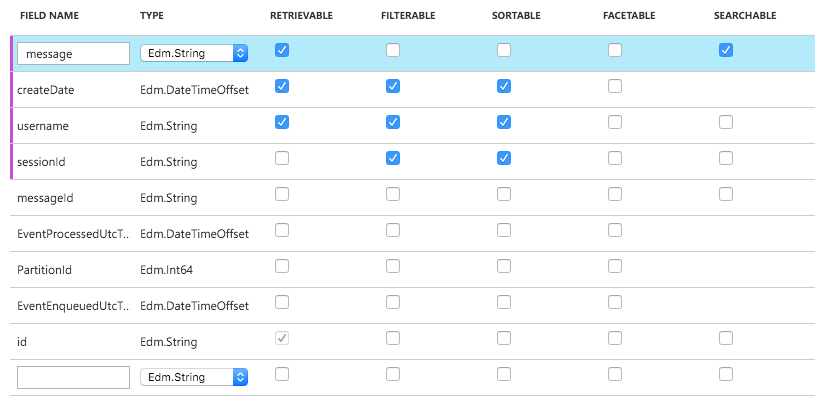
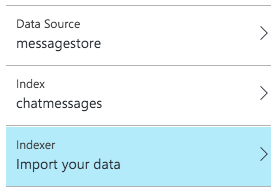
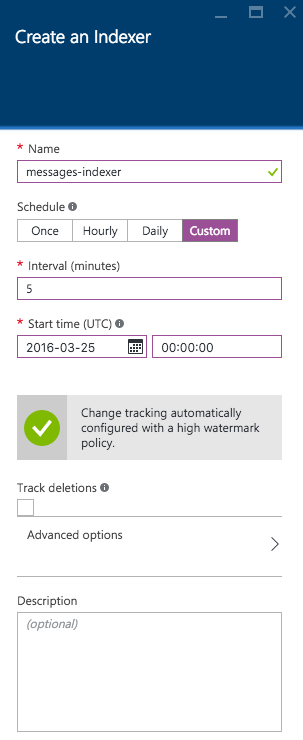
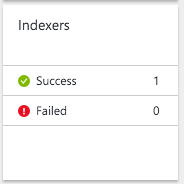
### Task 1: Verifying message archival

Before going further, a good thing to check is whether messages are being written to DocumentDB from the Stream Analytics Job.

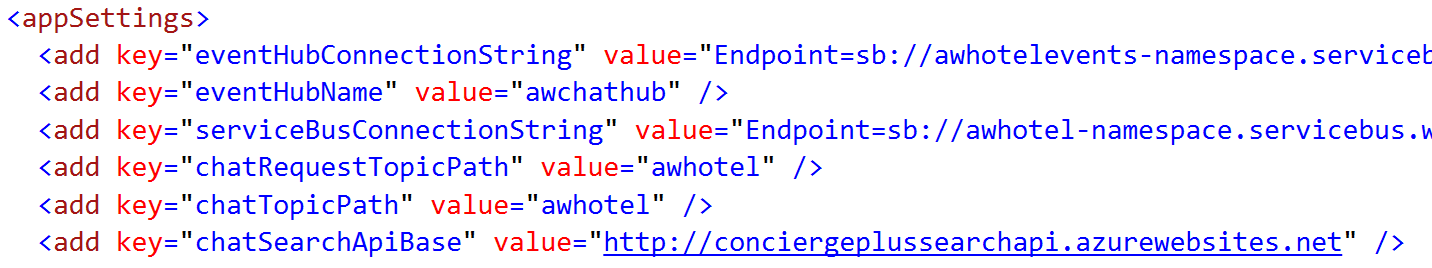
1. In the Azure Portal, navigate to your NoSQL (DocumentDB) account.
2. On the left-hand menu, select Document Explorer.   
    
3. If you see a list of document IDs, you have some data to start with! 
4. If you want to peek at the message contents, select any document in the listing.



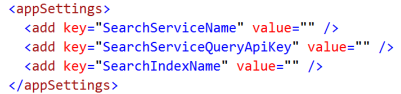
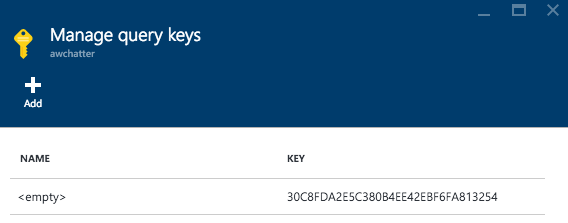
### Task 2: Creating the index and indexer

1. Select Browse, Search services, and choose your new Search instance from the list.
2. Select Import data.  
    
3. On the Import data blade, select Connect to your data.
4. On the Data Source blade, select DocumentDB.
5. Provide a name for the data source (e.g., messagestore).
6. Select your DocumentDB account.
7. Choose your DocumentDB database.
8. Choose your DocumentDB collection.
9. Select OK.
10. Select Customize target index, and observe that the field list has been pre-populated for you based on data in the collection.
11. Provide a name for the index (e.g., chatmessages).  
    ../../../../Captures/Screen%20Shot%202016-03-25%20at%2011.33.15%20PM.png
12. Leave the Key set to id.  
    ../../../../Captures/Screen%20Shot%202016-03-25%20at%2011.33.09%20PM.png
13. Select the Retrievable check box for the following fields: message, createDate, and username (id will be selected automatically). Only these fields will be returned in query results.
14. Select the Filterable check box for createDate, username, and sessionId. These fields can be used with the filter clause only (not used by this Tutorial, but useful to have).
15. Select the Sortable check box for createDate, username, and sessionId. These fields can be used to sort the results of a query.
16. Select the Searchable check box for message. Only these fields are indexed for full text search.
17. Confirm your grid looks similar to the following.  
    
18. Select OK.
19. Select Import your data.  
    
20. On the Create an Indexer blade, provide a name.
21. Set the Schedule toggle to Custom.
22. Enter an interval of 5 minutes (the minimum allowed).
23. Set the Start time to today’s date.  
    
24. Select OK.
25. Select OK once more to begin importing data using your indexer.
26. After a few moments, examine the Indexers tile for the status of the Indexer.  
    

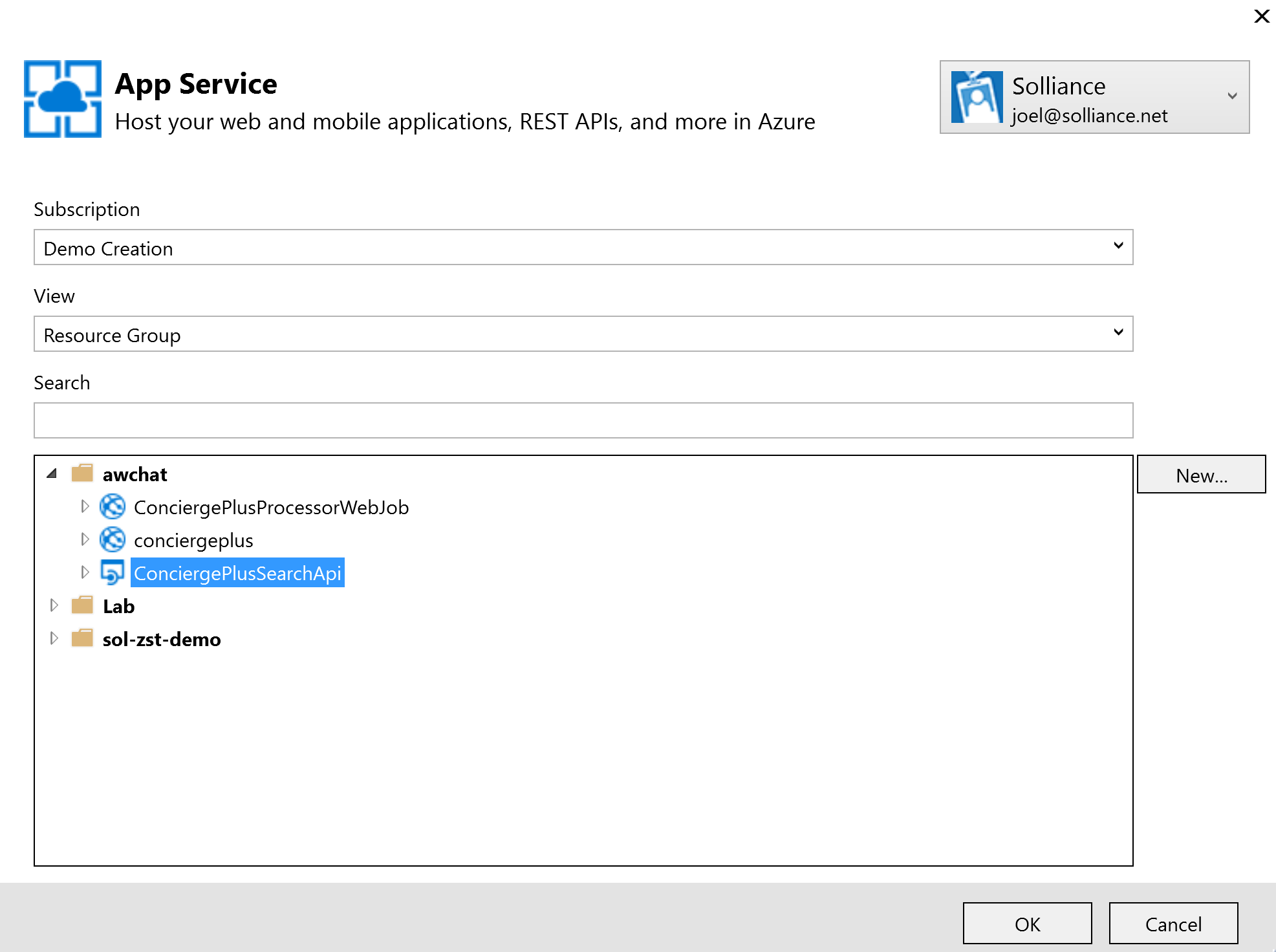
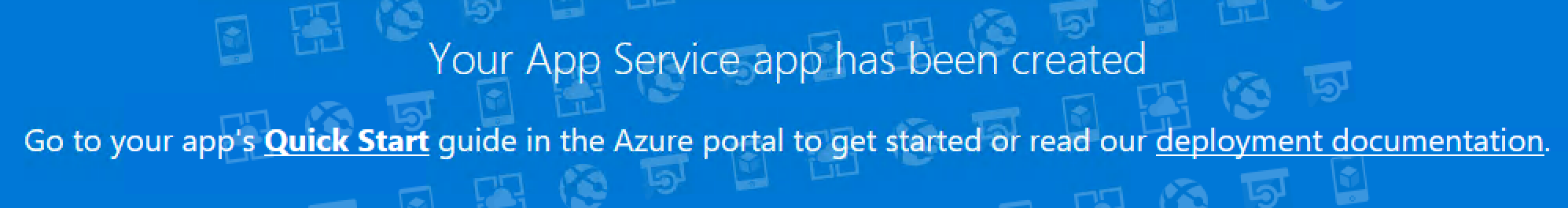
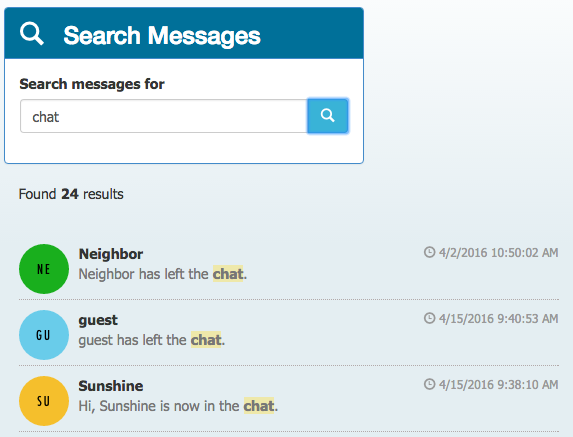
### Task 3: Update the Web App web.config

1. Within Visual Studio Solution Explorer, expand the ChatWebApp project.
2. Open Web.config.
3. For the chatSearchApiBase, enter the URI of the Search API App (e.g., <http://awchatsearch.azurewebsites.net)>. This value should **not** be the URL to your instance of Azure Search. 

### Task 4: Configure the Search API App

1. Within Visual Studio Solution Explorer, expand the ChatAPI project.
2. Open web.config.
3. This project needs the following three settings configured in order to capitalize on Azure Search, all of which you can get from the Azure Portal.  
   
4. Using the Azure Portal, navigate to the blade of your Search service.
5. For the SearchServiceName, enter the name of your Search (e.g., awchatter).
6. For the SearchServiceQueryApiKey, in the Portal, select Keys.
7. Select Manage query keys.  
   
8. Copy this value into the SearchServiceQueryApiKey setting.
9. For the SearchIndexName setting, enter the name of the Index you created in Search (e.g., chatmessages).

### Task 5: Re-publish apps

1. Publish the updated ChatWebApp using Visual Studio, as was shown previously.
2. Within Visual Studio Solution Explorer, right-click the ChatAPI project and select Publish.  
   
3. Select Microsoft Azure App Service, choose the Select existing radio button and select Publish.
4. If prompted, sign in with your credentials to your Azure Subscription.
5. In the App Service dialog, choose your Subscription that contains your API App you provisioned earlier. Expand your Resource Group (e.g., awchat), then select the node for your API App in the tree view to select it. 
6. Select OK.
7. When the publishing is complete, a browser window should appear with content similar to the following.  
   
8. Navigate to the Search tab on the deployed Web App and try searching for chat messages. (Note that there is up to a 5-minute latency before new messages may appear in the search results.)  
   

## Exercise 8: Cleanup

Duration: 5 minutes

Synopsis: In this exercise, attendees will de-provision any Azure resources that were created in support of the hackathon.

### Task 1: Clean up Azure Resources

1. Delete the Resource Group you created.
   1. From the Portal, navigate to the blade of your Resource Group and select **Delete** in the **command bar** at the top.



1. When prompted to confirm the deletion, follow the prompts.