Microsoft Cloud Workshops

Intelligent analytics hackathon

Lab guide

February 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](#_Toc474143635)

[Intelligent analytics hackathon answers 17](#_Toc474143636)

# 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

Adventure Works has provided a starter solution for you. They have asked you to use this as the starting point for creating the ConciergePlus 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.

#### Tasks to complete

* Provision a Web App to host the website in an App Service Plan in the Resource Group “awchat”. Name the Web App something like “ConciergePlusWeb”.
* Provision another Web App to host a Web Job in the same App Service Plan and Resource Group. Name it something like “ChatProcessorWebJob”.
* Configure the Web Job Web App to have Always On enabled.

#### Exit criteria

* You can navigate to the empty websites for both deployed Web Apps using a browser.

### Task 3: Service Bus

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

#### Tasks to complete

* Provision a Service Bus Topic in the same region/resource group as your App Services.
* Provision an Event Hub in the same region/resource group as your App Services.

#### Exit criteria

* Your Service Bus Topic is listed in the Azure Portal.

### Task 4: Event Hubs

In this section, you will provision the two Event Hub instances.

#### Tasks to complete

* Provision two Event Hub instances in a new namespace and in the same region/resource group as your App Services. Use a partition count of 32 and message retention of 1 day. The second Event Hub will store messages for archival and be processed by Stream Analytics.

#### Exit criteria

* Your Event Hubs are listed in the Azure Portal.

### 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.

#### Tasks to complete

* Provision a new DocumentDB account in the same resource group and region as your other services.
* Add DocumentDB Database.
* Add a Collection. The pricing tier should be left at Standard with the Throughput at 1000.

#### Exit criteria

* You can view your Collection in the Azure Portal.

### Task 6: Azure Search

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

#### Tasks to complete

* Provision a new instance of Azure Search in the same resource group and region as your other services, at the Basic Pricing tier.

#### Exit criteria

* You can view your Azure Search instance in the Azure Portal.

### 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.

#### Tasks to complete

* Provision a new Stream Analytics Job in the same region as your other resources.
* Add an Input to it that reads from your second Event Hub (the one used for archival) and be sure to choose the Consumer Group ASA01. The serialization should be JSON/UTF8.
* Add an Output to the Job that targets your Collection in DocumentDB. The Partition Key should be sessionid (all lowercase) and the Document ID messageid (all lower case).
* Add a Query that selects all data from the Event Hub and sends it to DocumentDB.

#### Exit criteria

* You can view your Stream Analytics instance in the Azure Portal.

### Task 8: Start the Stream Analytics Job

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

#### Tasks to complete

* Start the Stream Analytics Job.

#### Exit criteria

* Your Job starts without error.

### 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.

#### Tasks to complete

* Provision a resource model based Storage Account of type Standard LRS in the same Location and Resource Group as your other services.

#### Exit criteria

* You can view your Storage Account in the Portal.

### 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.

#### Tasks to complete

* Provision a Cognitive Services account of API Type Text Analytics API in the same Location and Resource Group as your other services. Take note of the value of KEY 1.
* Provision a Cognitive Services account of API Type Bing Speech API in the same Location and Resource Group as your other services. Take note of the value of KEY 1.
* Provision a Cognitive Services account of API Type Language Understanding Intelligent Service (LUIS) in the same Location and Resource Group as your other services. Take note of the value of KEY 1.

#### Exit criteria

* You can view your Cognitive Services accounts in the Portal, you should have one for Text Analytics API, another for Bing Speech API, and a third 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

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

#### Tasks to complete

* Open SentimentEventProcessor.cs and navigate to the IEventProcessor.ProcessEventsAsync method.
* Complete the TODOs numbered 1 through 6.

#### Exit criteria

* There are no errors in the IEventProcessor.ProcessEventsAsync method in Visual Studio. Note that at this point the solution will not yet run.

### 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.

#### Tasks to complete

* Create a Shared Access Policy for Event Hub with Manage, Send, and Listen permissions.
* Create a Shared Access Policy for Service Bus with Manage, Send, and Listen permissions.
* Copy the connection string from the policy into the eventHubConnectionString setting in the app.config.
* Copy the connection string from the policy into the serviceBusConnectionString setting in the app.config.
* Set sourceEventHubName to the name of your first Event Hub.
* Set destinationEventHubName to the name of your second Event Hub.
* Set storageAccountName to the name of the storage account you created.
* Set storageAccountKey to the Key for the storage account.
* Set chatTopicPath the name of the Service Bus Topic you created.
* Set textAnalyticsBaseUrl to the Endpoint of the Text Analytics Cognitive Services account. Be sure to include a trailing slash in the URL.
* Set textAnalyticsAccountName to the Account Name of the Text Analytics Cognitive Services account.
* Set textAnalyticsAccountKey to the value of KEY 1 from this same Cognitive Services account.

#### Exit criteria

* You should have values for all the app settings except LuisAppId and LuisKey.

## Exercise 3: Configure the Chat Web App

Duration: 10 minutes

Within Visual Studio Solution Explorer, expand the ChatWebApp project and open Web.Config. You will update the appSettings in this file.

### Task 1: Configure the Chat Web App settings

#### Tasks to complete

* Copy the connection string from the Event Hub policy you created into the eventHubConnectionString setting.
* Copy the connection string from the Service Bus policy you created into the serviceBusConnectionString setting.
* Set eventHubName to the name of your first Event Hub.
* Set chatTopicPath to the name of the Service Bus Topic you created.
* Set chatRequestTopicPath to the name of the Service Bus Topic you created.

#### Exit criteria

* You should have values for all the app settings except chatSearchApiBase.

## 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

#### Tasks to complete

* Publish the ChatMessageSentimentProcessor Web Job with a run mode of Run Continuously to the Web App you had provisioned for it.

#### Exit criteria

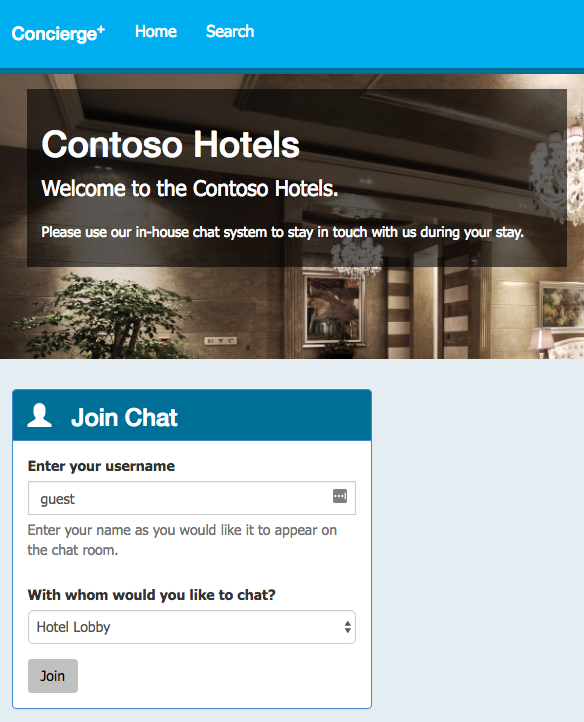
* The Output dialog indicates your publish was successful.

### Task 2: Publish the ChatWebApp

#### Tasks to complete

* Publish the ChatWebApp to the Web App you had provisioned for it.

#### Exit criteria

* A browser window should appear with content similar to the following:  
  

### Task 3: Testing hotel lobby chat

#### Tasks to complete

* Open a browser instance and navigate to the deployed Web App.
* Join the chat.
* Repeat with one or more additional browser tabs or from different a device and browser.
* Start a conversation between these users in the Hotel Lobby.

#### Exit criteria

* You should see your messages appear in all other browser instances joined to the chat.

## 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.

#### Tasks to complete

* Complete the TODOs numbered 7 through 11 in the GetSentimentScore method.
* Complete TODO 12 in the ProcessEventsAsync method.

#### Exit criteria

* You should see no errors in either the GetSentimentScore or ProcessEventsAsync methods.

### 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.

#### Tasks to complete

* Use <http://www.luis.ai> to create a new App.
* Set the App Settings in the LUIS website to use your subscription key (KEY 1) from your LUIS Account setup in the Azure Portal and activate it.
* Add an intent named OrderIn with an example utterance of “order a pizza.”
* Add an entity named RoomService with hierarchical children FoodItem and RoomItem.
* Review the label for the utterance “order a pizza” and set the intent to OrderIn and entity (pizza) to FoodItem.
* Add new utterances for “Bring me toothpaste”, “Bring me blankets”, “Order a soda”, and “Order me a pizza.”
* Train the model.
* Publish the model.
* Test the model with the query “order me a pizza” and verify you get the intent of OrderIn (with a score close to 1.0) and entity pizza with a type of “RoomService::FoodItem”.
* Copy the App ID and Susbcription Key from your LUIS app into the App.config of the ChatMessageSentimentProcessor project, to the luisAppID and luisKey settings respectively.
* Complete TODO 13 in SentimentEventProcessor.cs.

#### Exit criteria

* All TODO items in SentimentEventProcessor.cs should be completed.

### Task 3: Implement speech to text

#### Tasks to complete

* Add your Speech API Key under the TODO in scripts\chatClient.js of the ChatWebApp project.

#### Exit criteria

* You should have all TODO items complete in chatClient.js.

### Task 4: Re-deploy and test

#### Tasks to complete

* Publish the ChatMessageSentimentProcessor.
* Publish ChatWebApp.

#### Exit criteria

* Open a browser and navigate to your deployed website using HTTPS. Use multiple browser instances or tabs to simulate multiple users.
* Send chat messages between them and verify that you see the sentiment indicator (a thumbs up or thumbs down icon next to each chat message).
* You can order something from room service, like “bring me towels” and you get a response from the ConciergeBot.
* You can select the microphone to the left of the text box and speak for two to three seconds. Your spoken message should appear.



## Exercise 6: Building the Power BI dashboard

Duration: 30 minutes

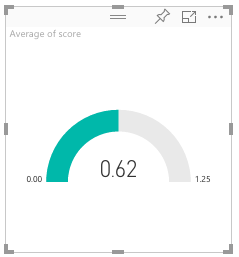
Now that you have the solution deployed and exchanging messages, you can build a Power BI dashboard that monitors the sentiments of the messages being exchanged in real time. The following steps walk through the creation of the dashboard.

### Task 1: Create the static dashboard

#### Tasks to complete

* Using PowerBI.com, create a new report from your streamed dataset that contains a semi-circular gauge. It should chart the average score, with a range of 0.0 to 1.0.

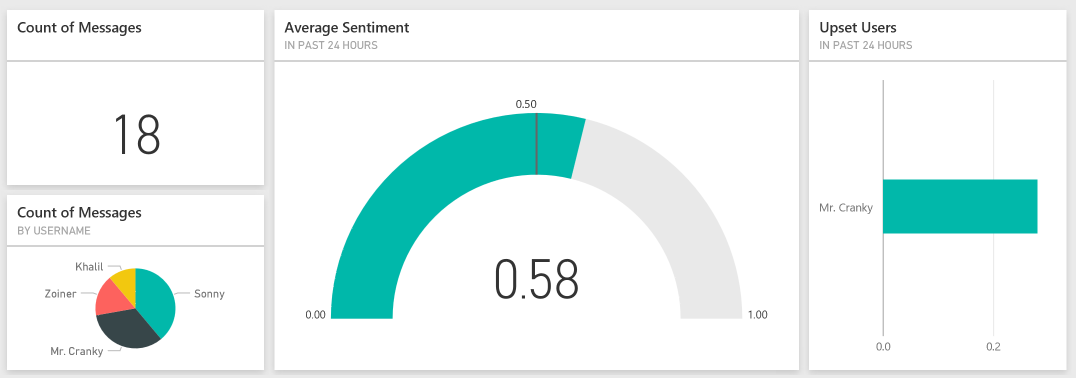
#### Exit criteria

* Your report should look like the following:  
  

### 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.

#### Tasks to complete

* Create a new dashboard by pinning the gauge you created previously.
* Open the new dashboard.
* Enter the QA query “average score created between yesterday and today” and visualize it with the Gauge chart.
* Format the chart so it ranges from 0.0 to 1.0 and has an indicator at 0.5.
* Pin this visual to the dashboard you created.
* Delete the old static chart from the dashboard.
* Navigate to the ConciergePlus website and send some messages, observing that your Gauge updates in real time.
* Add three additional visualizations so your dashboard looks as follows:  
  

#### Exit criteria

* Your chart should update in real-time and appear as above.

## Exercise 7: Enabling search indexing

Duration: 30 minutes

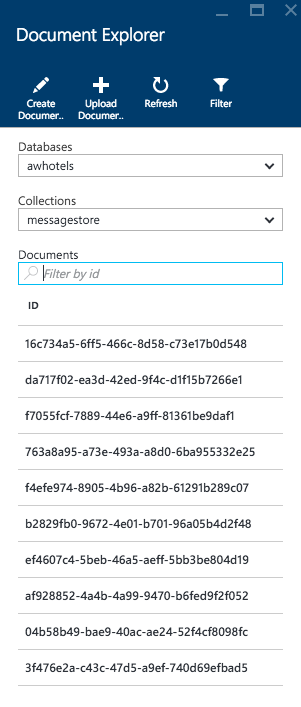
Now that you have primed the system with some messages, you will create a Search Index and an Indexer in Azure Search upon the messages that are collected in DocumentDB.

### Task 1: Verifying message archival

#### Tasks to complete

* Using the Azure Portal, navigate to Document Explorer for your DocumentDB Collection.

#### Exit criteria

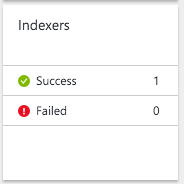
* You should see a list of document IDs, similar to the following:  
  

### Task 2: Creating the index and the indexer

#### Tasks to complete

* In your Azure Search Index, use the Connect to your data feature to index the collection.
* Provide any name for the index, but leave the Key set to id.
* Ensure that the id, message, createDate, and username are Retrievable.
* Ensure that createDate, username, and sessionId are Filterable.
* Ensure that create date, username, and sessionId are Sortable.
* Ensure that the message field is Searchable.
* Create an Indexer with a 5-minute interval that starts today.

#### Exit criteria

* After a few moments, your indexer status should appear similar to the following:  
  

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

#### Tasks to complete

* Modify the web.config of the ChatWebApp project.
* Set the chatSearchApiBase to the URI of the deployed Search API app.

#### Exit criteria

* You should have all app settings completed with values in the web.config.

### Task 4: Configure the Search API App

#### Tasks to complete

* Modify the web.config of the ChatApi project.
* Set the SearchServiceName to the name of your Search service.
* Set the SearchServiceQueryApiKey to the Key of your Search service.
* Set the SearchIndexName to the name of the Index you created in Search.

#### Exit criteria

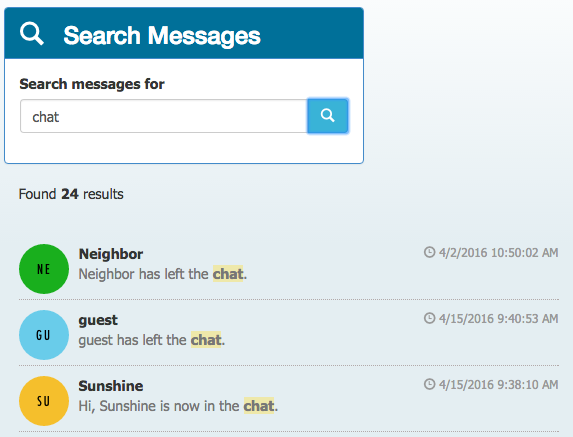
* You should have all app settings completed with values in the web.config.

### Task 5: Re-publish apps

#### Tasks to complete

* Publish the ChatWebApp.
* Publish the ChatApiApp to your API App.

#### Exit criteria

* 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

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

### Task 1: Clean up Azure Resources

#### Tasks to complete

* Delete the Resource Group you created.

#### Exit criteria

* You have no services deployed for this hackathon remaining in your subscription.

# Intelligent analytics hackathon answers

## Overview

In this hackathon, attendees will construct an end-to-end solution for an IoT scenario that includes device management; telemetry ingest; hot and cold path processing; and reporting.

## 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. Note that you will need to use both the Azure Portal (<https://portal.azure.com)> and the Manage Portal (<https://manage.windowsazure.com)> to completely provision all resources. 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.

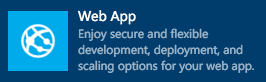
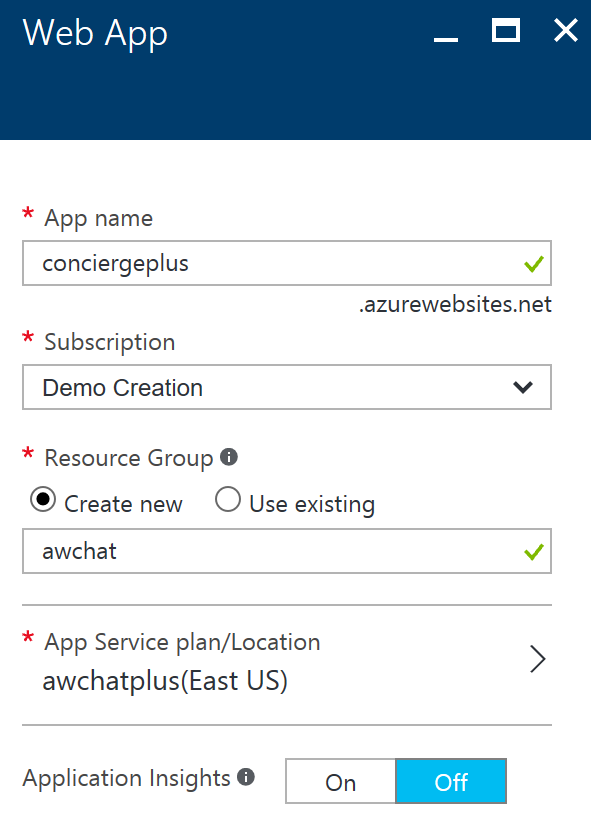
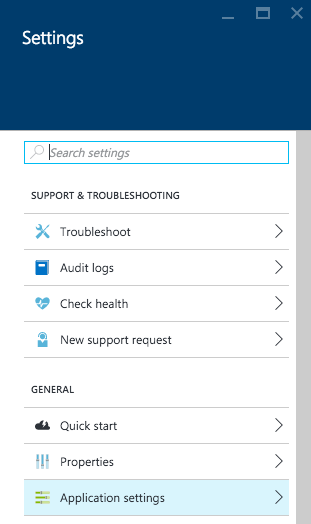
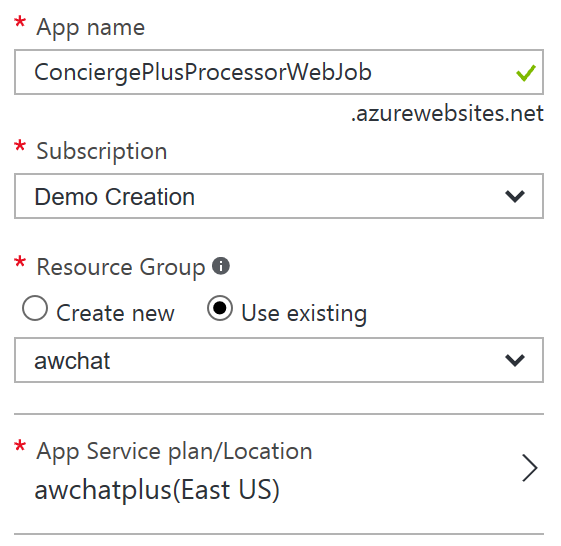
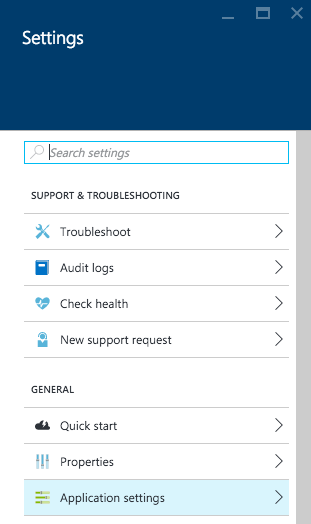
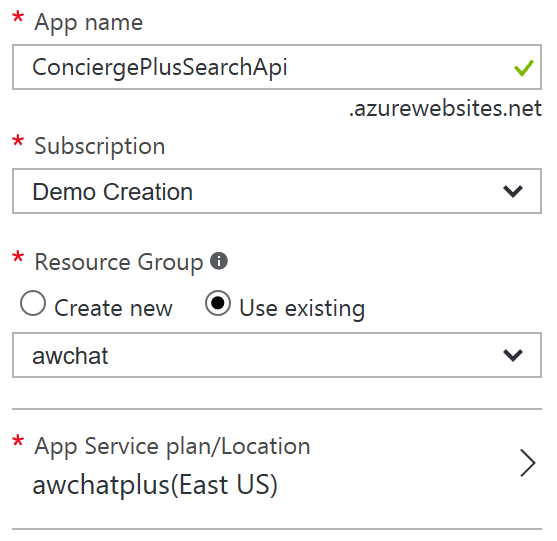
### Task 1: Download and open the ConciergePlus starter

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

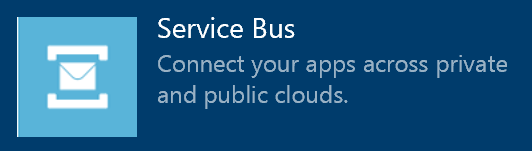
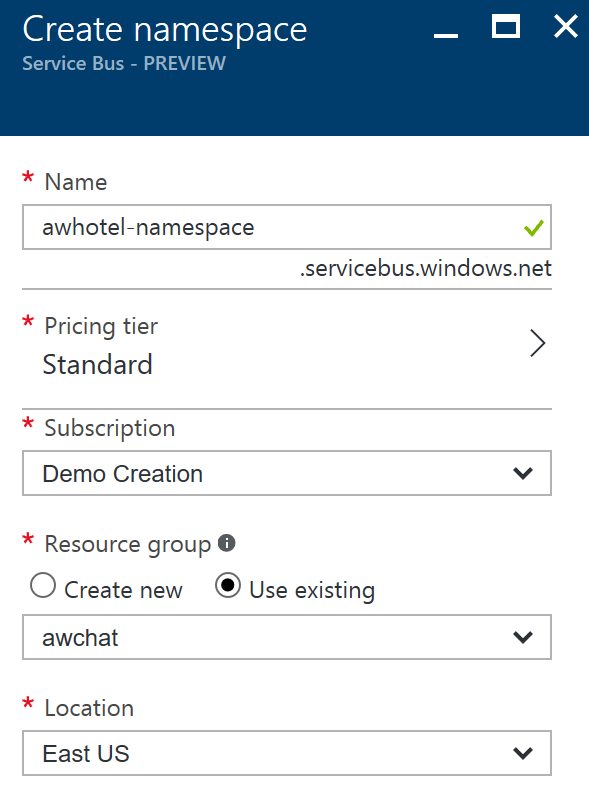
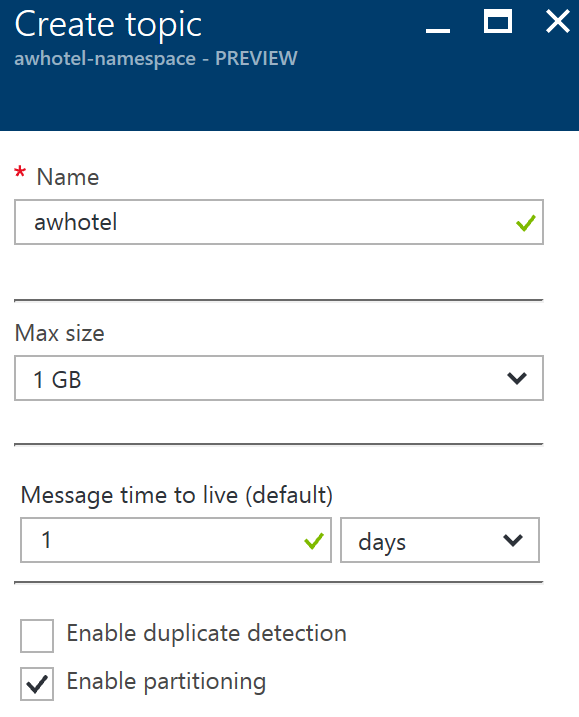
In these steps, you will provision two Web Apps and an API App within a single App Service Plan.

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”).
8. 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.  
   
9. Select Create. This will provision both the App Service Plan and the Web App.
10. When the provisioning completes, navigate to your new Web App in the portal.
11. On the Settings blade, select Application settings.  
    
12. Select the toggle for Web Sockets to On.  
    ../../../../Captures/Screen%20Shot%202016-03-26%20at%2010.48.34%20AM.png
13. Select Save.  
    ../../../../Captures/Screen%20Shot%202016-03-26%20at%2010.48.48%20AM.png
14. You are now ready to provision the other Web App using this same service plan.
15. Select +New, Web + Mobile, Web App.
16. Provide a name for this new Web App that indicates its use as the host for the Event Processor Web Job (e.g., ChatProcessorWebJob).
17. Select the same subscription as used previously.
18. Select the Resource Group that you created previously.
19. Select the App Service plan that you created previously.  
    
20. Select Create.
21. When the provisioning completes, navigate to your new Web App in the portal.
22. On the Settings blade, select Application settings.  
    
23. 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
24. Select Save.  
    ../../../../Captures/Screen%20Shot%202016-03-26%20at%2010.48.48%20AM.png
25. 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****.)*  
    
26. Provide an App name for this API app that reflects it will host the Chat Search API (e.g., ChatSearchApi).
27. Select the same subscription as used previously.
28. Select the Resource Group that you created previously.
29. Select the App Service plan that you created previously.  
    

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

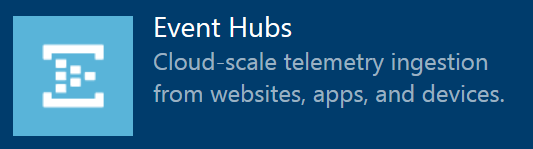
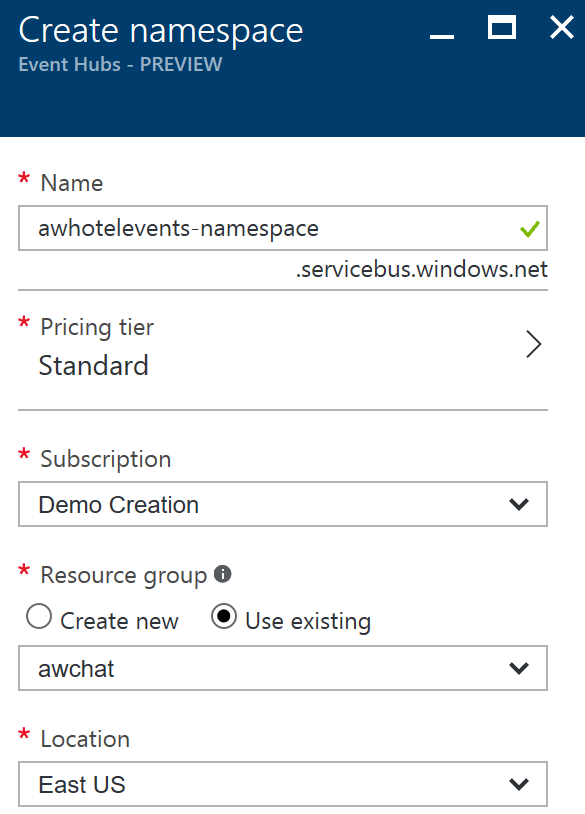
### Task 3: Service Bus

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. When the provisioning completes, navigate to your new Service Bus in the portal.
9. In the Overview blade, click + Topic.  
   
10. 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.
11. Leave the Max Size at 1 GB.
12. Set the Default Message Time to Live to 1 days.  
    
13. Click Create.  
    

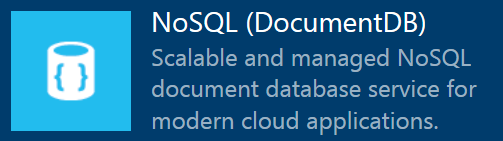
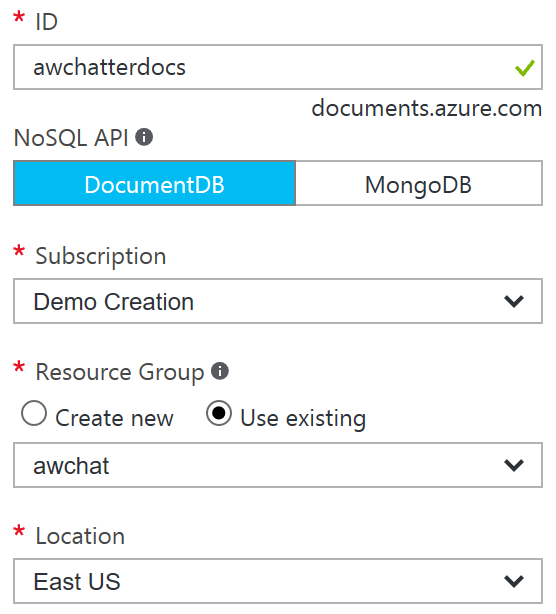
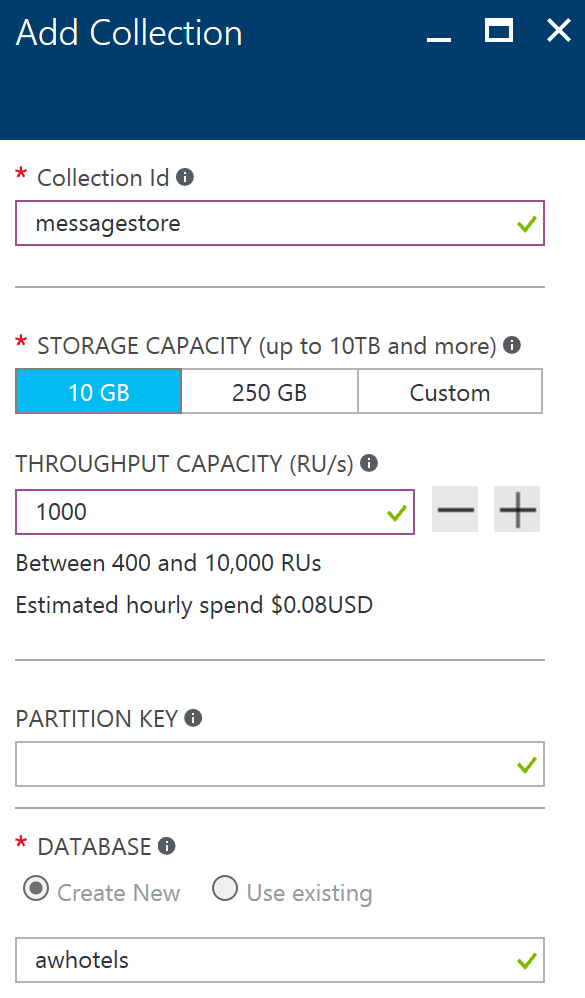
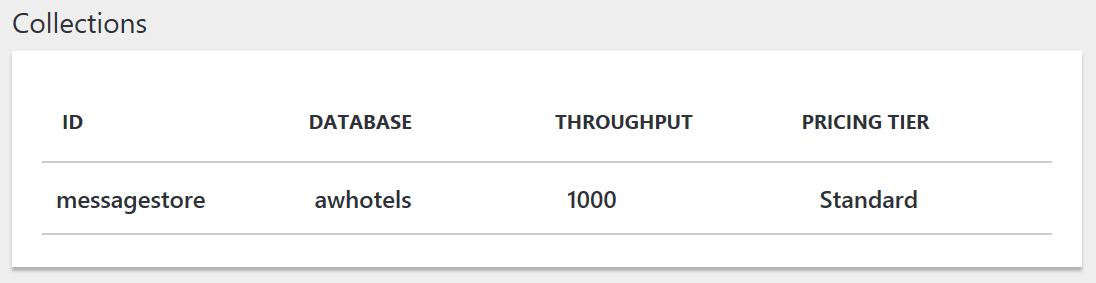
### Task 4: Event Hubs

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. Click 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. Select the check mark to create the Event Hub.
17. Repeat steps 10 through 15 to create another Event Hub. (This one will store messages for archival and be processed by Stream Analytics.)

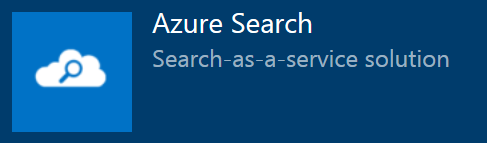
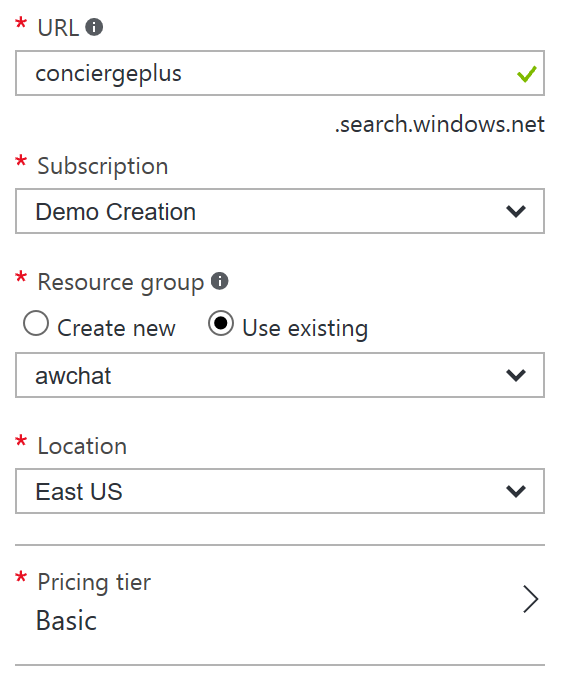
### Task 5: DocumentDB

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, Data, 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 your newly added database in the Databases tile.  
     
16. Click OK.  
    

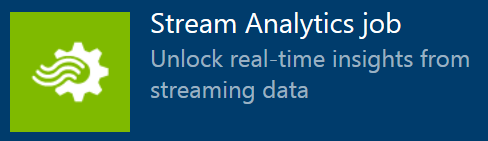
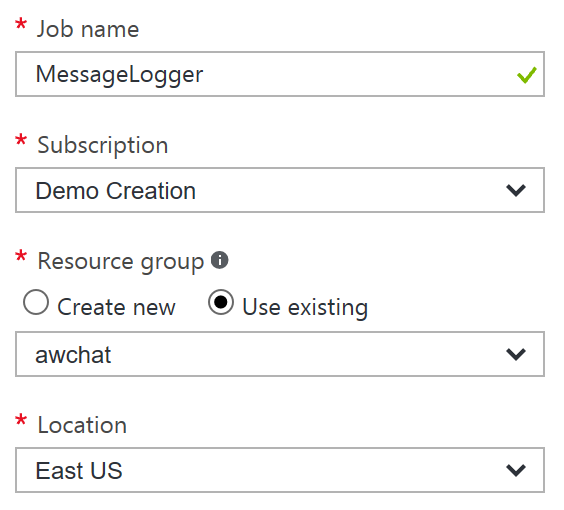
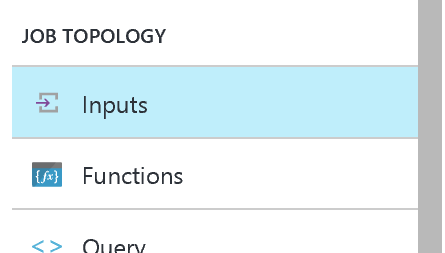
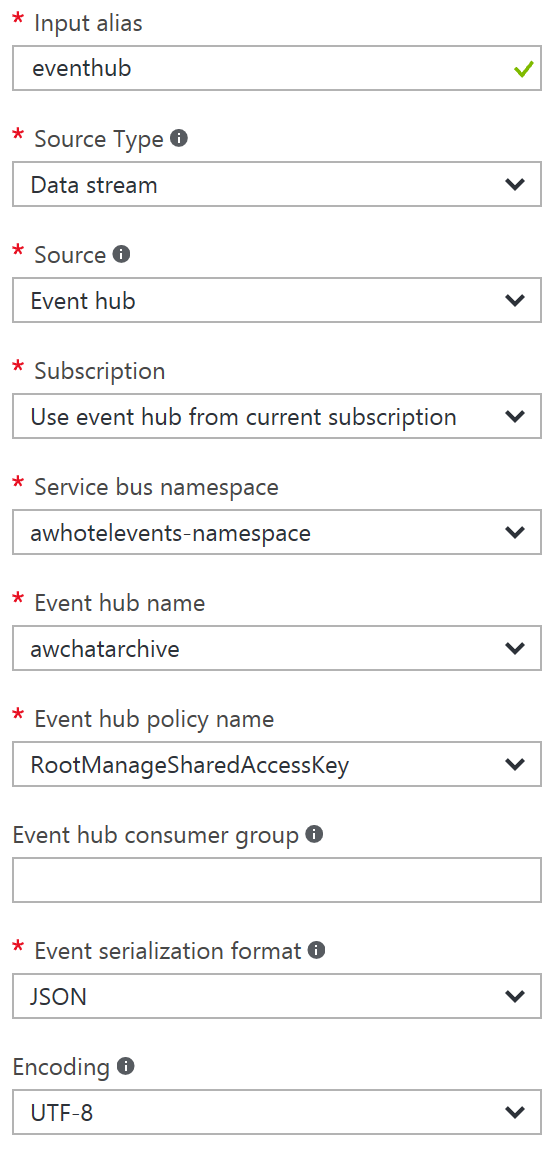
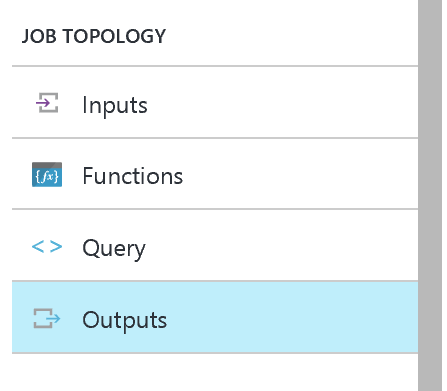
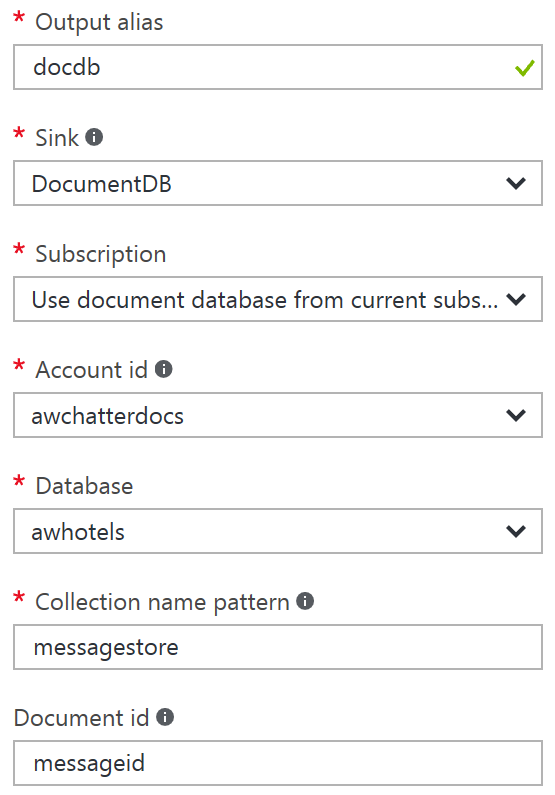
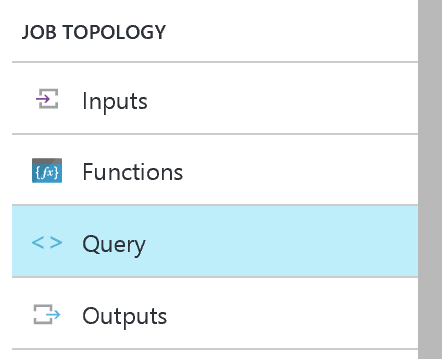
### Task 6: Azure Search

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. Click 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. Click 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. Click 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. Click Create.  
    
29. Select Query from the left-hand menu.  
    
30. In the query text box, enter the following query:

SELECT

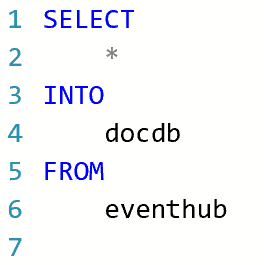
\*

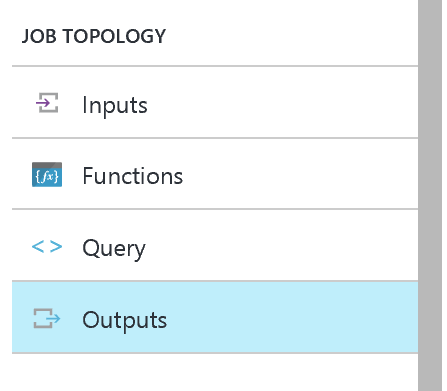
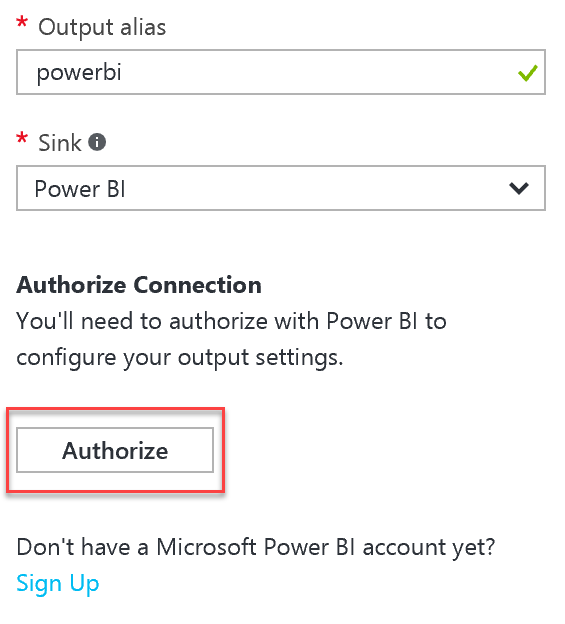
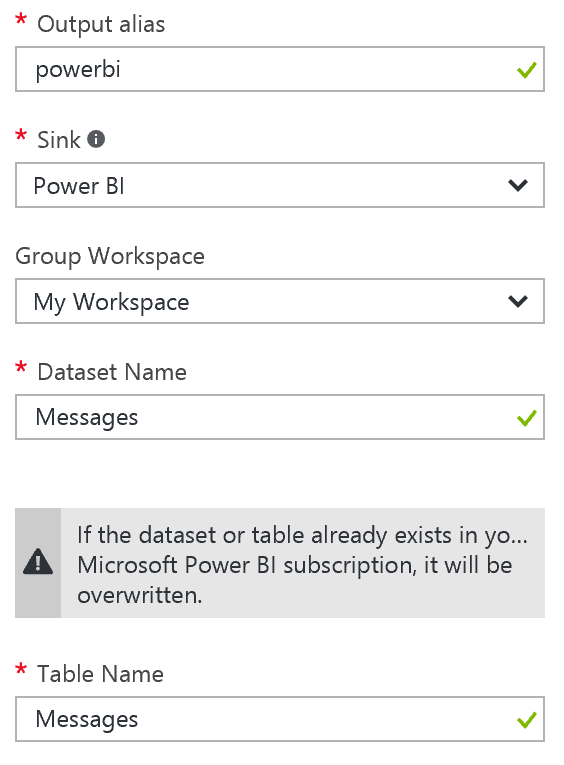
INTO

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 click Create to create the output.   
    
9. Modify your query so that below the first query, you append the following query text:

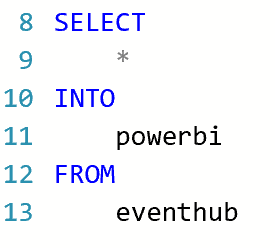
SELECT   
   
\*

INTO

powerbi

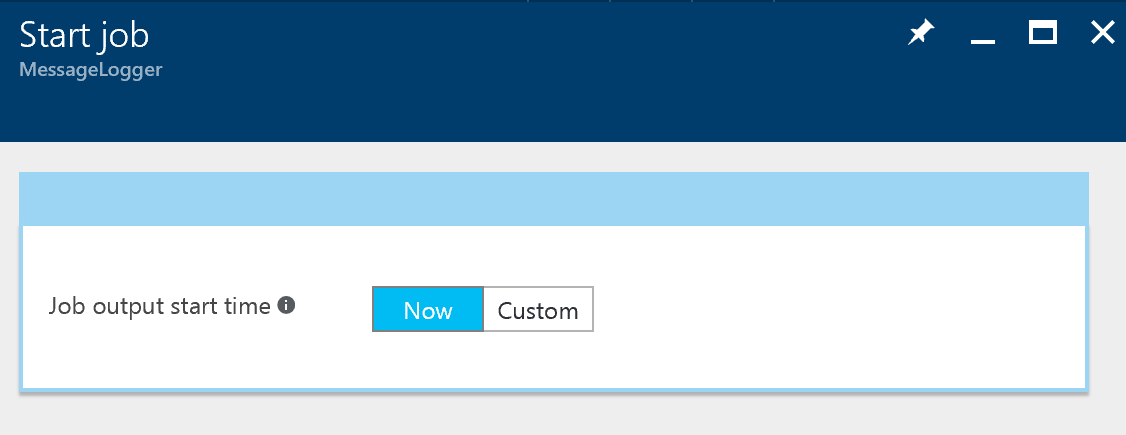
FROM

eventhub



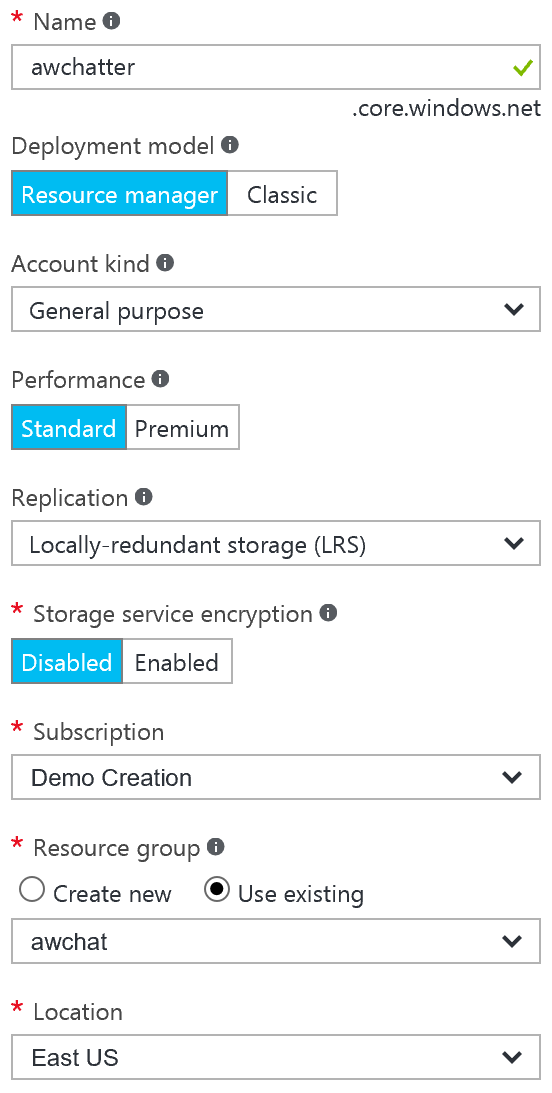
1. Select Save and Yes when prompted with the confirmation.  
    

### 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. Click Start.
5. Allow your Stream Analytics Job a few minutes to start.

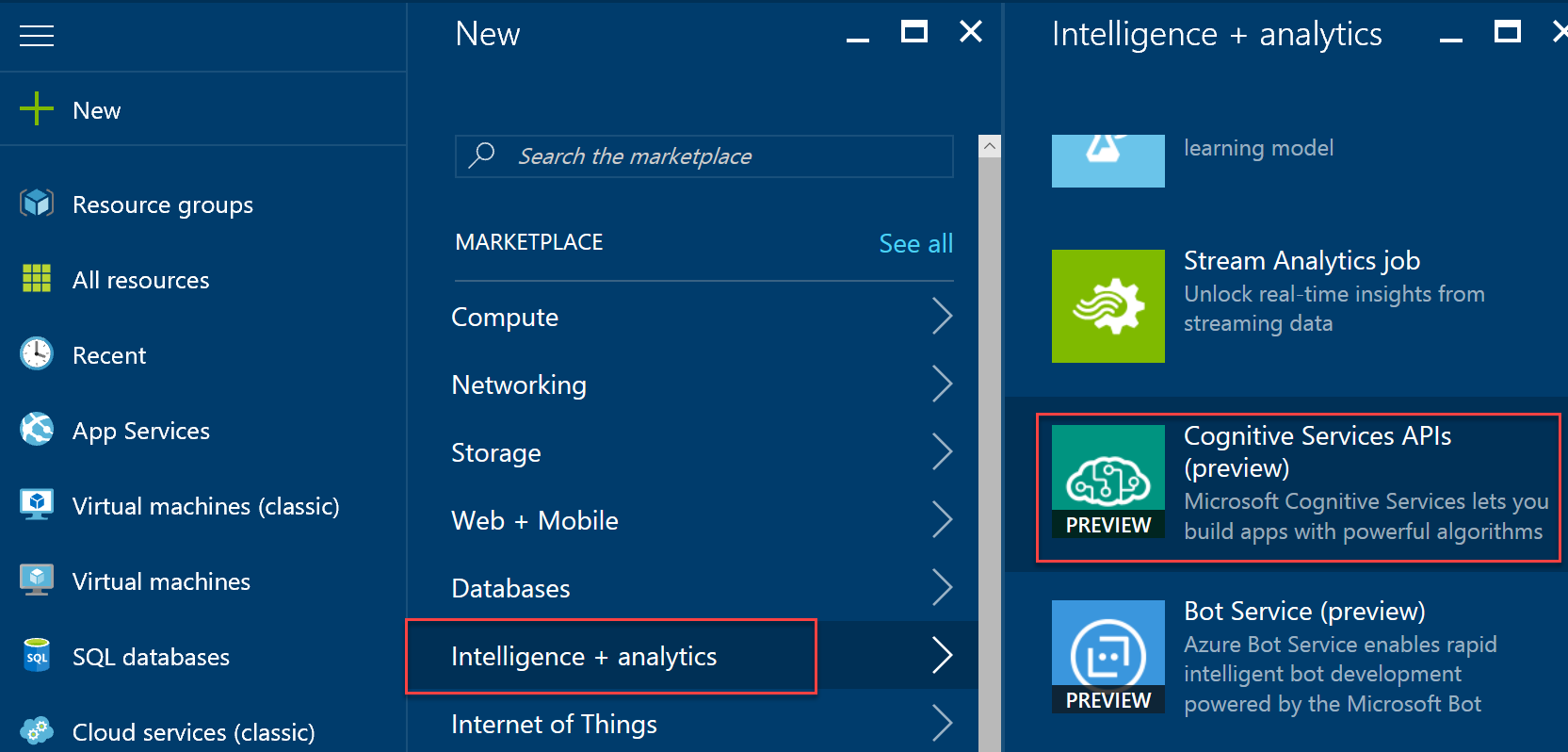
### Task 9: Storage Account

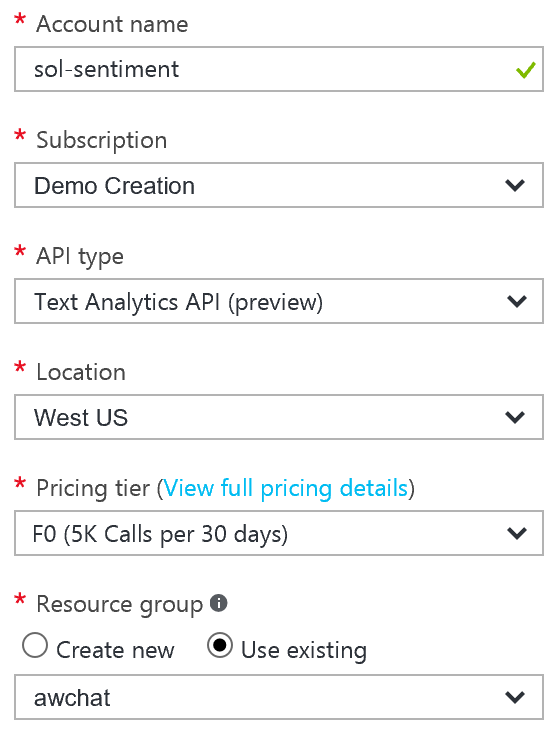
The EventProcessorHost requires an Azure Storage Account that it will use to manage its state among multiple instances. In this section, you create that Storage Account.

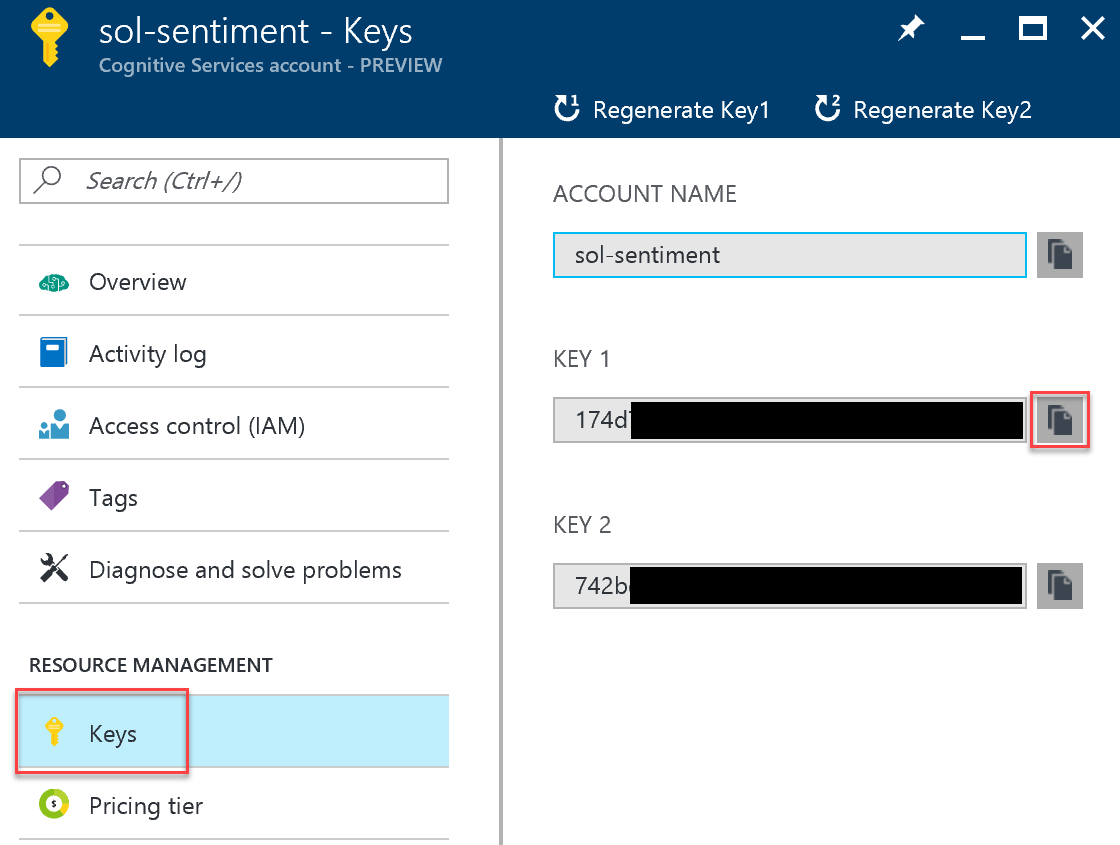
1. Using the Azure Portal, select + New, Storage, Storage account.
2. In the Create storage account blade, provide a unique name for the account.
3. Leave Resource Manager selected for the deployment model.
4. Leave General purpose selected for the Account kind.
5. Set the Performance to Standard, and 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. Click 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 the previous steps, this time selecting Bing Speech API for the API Type. Take note of Key 1 for Speech.
4. Repeat the previous steps, 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 three 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="" />

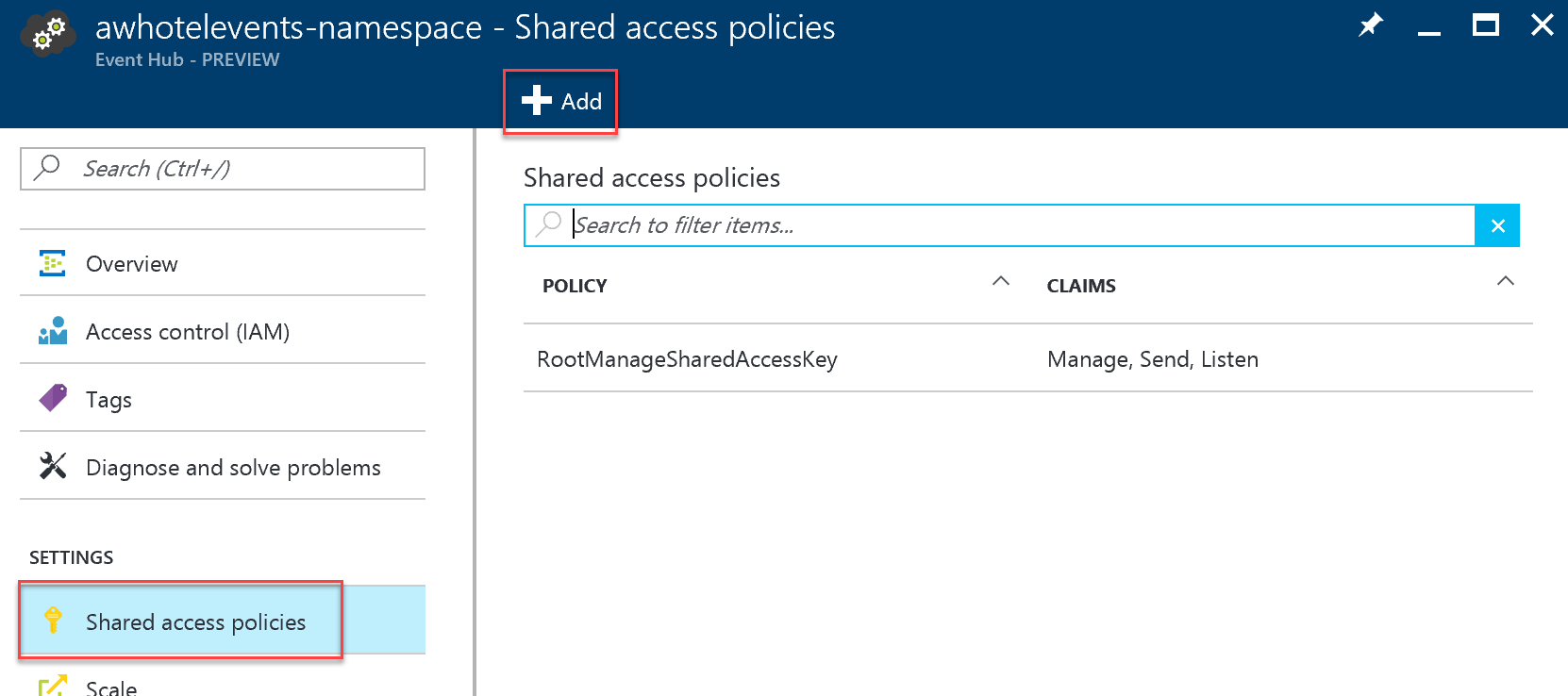
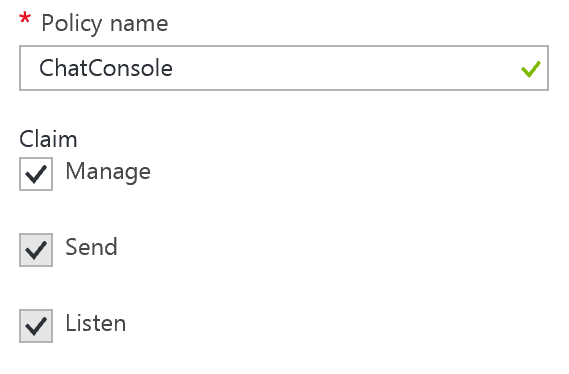
<add key="chatTopicPath" value="" />

<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. Click 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

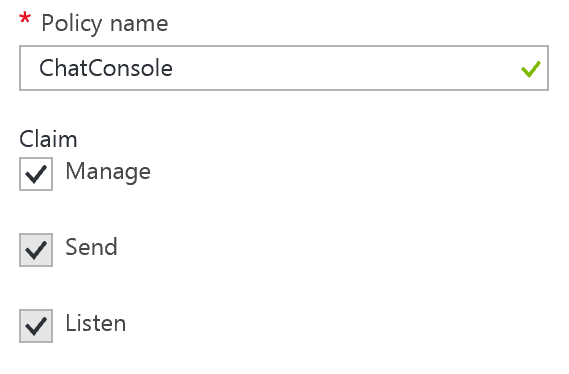
1. For the sourceEventHubName setting in app.config, enter the name of your first Event Hub.
2. For the destinationEventHubName, enter the name of your second Event Hub.

#### 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. Click 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.

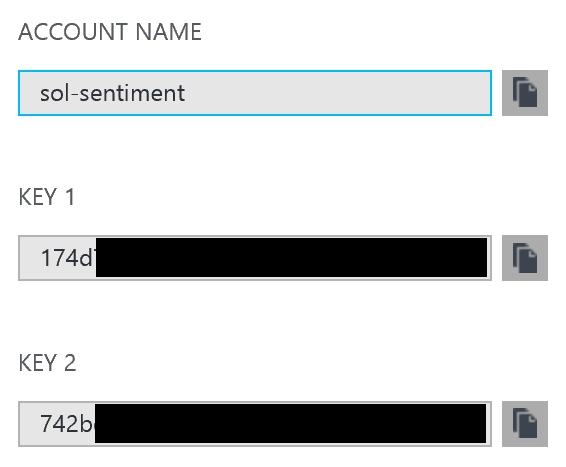


Return to the app.config and paste this as the value for serviceBusConnectionString.

#### 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

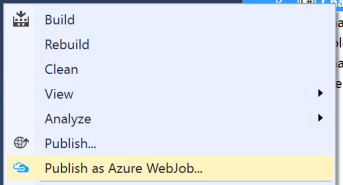
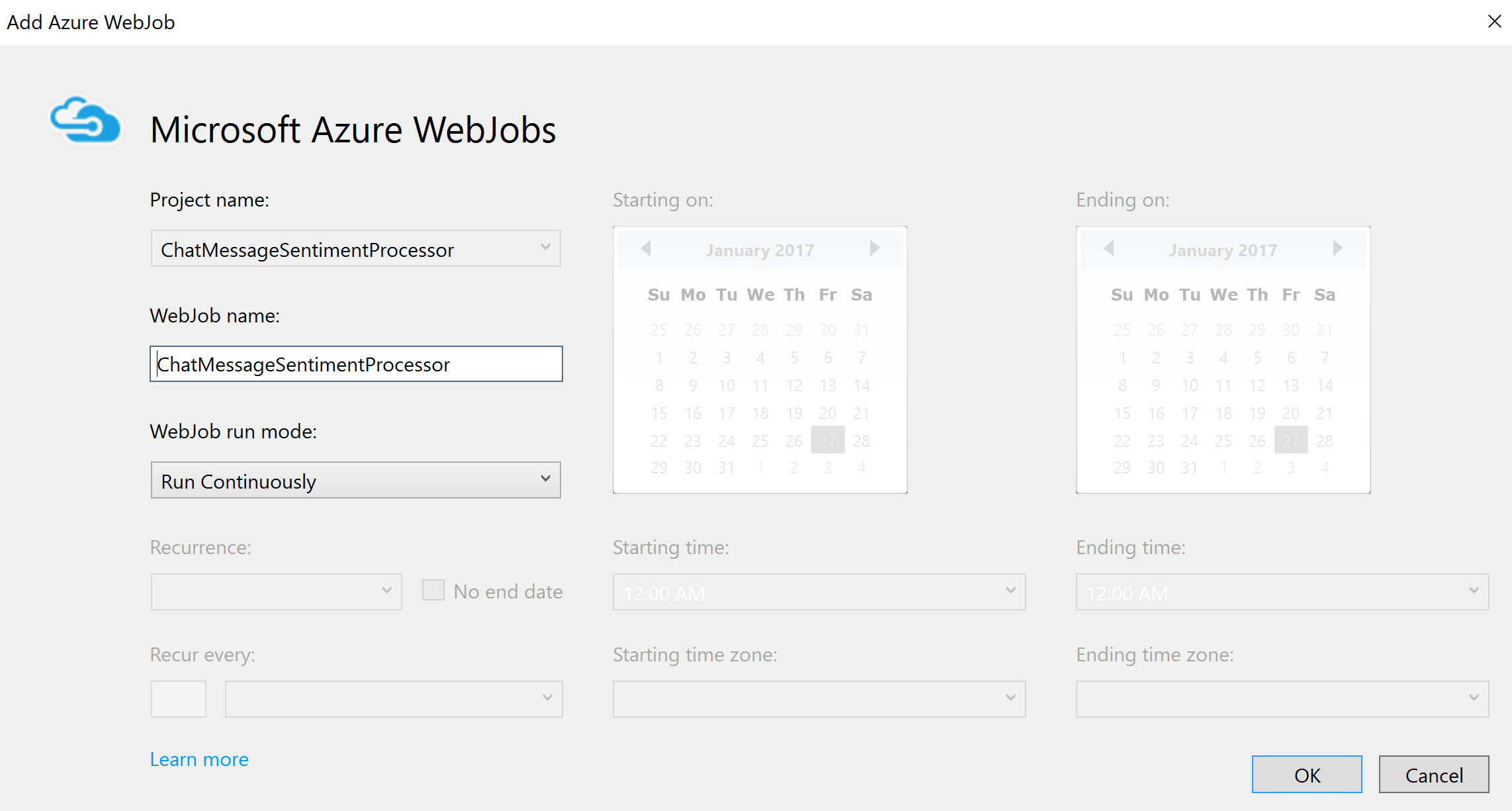
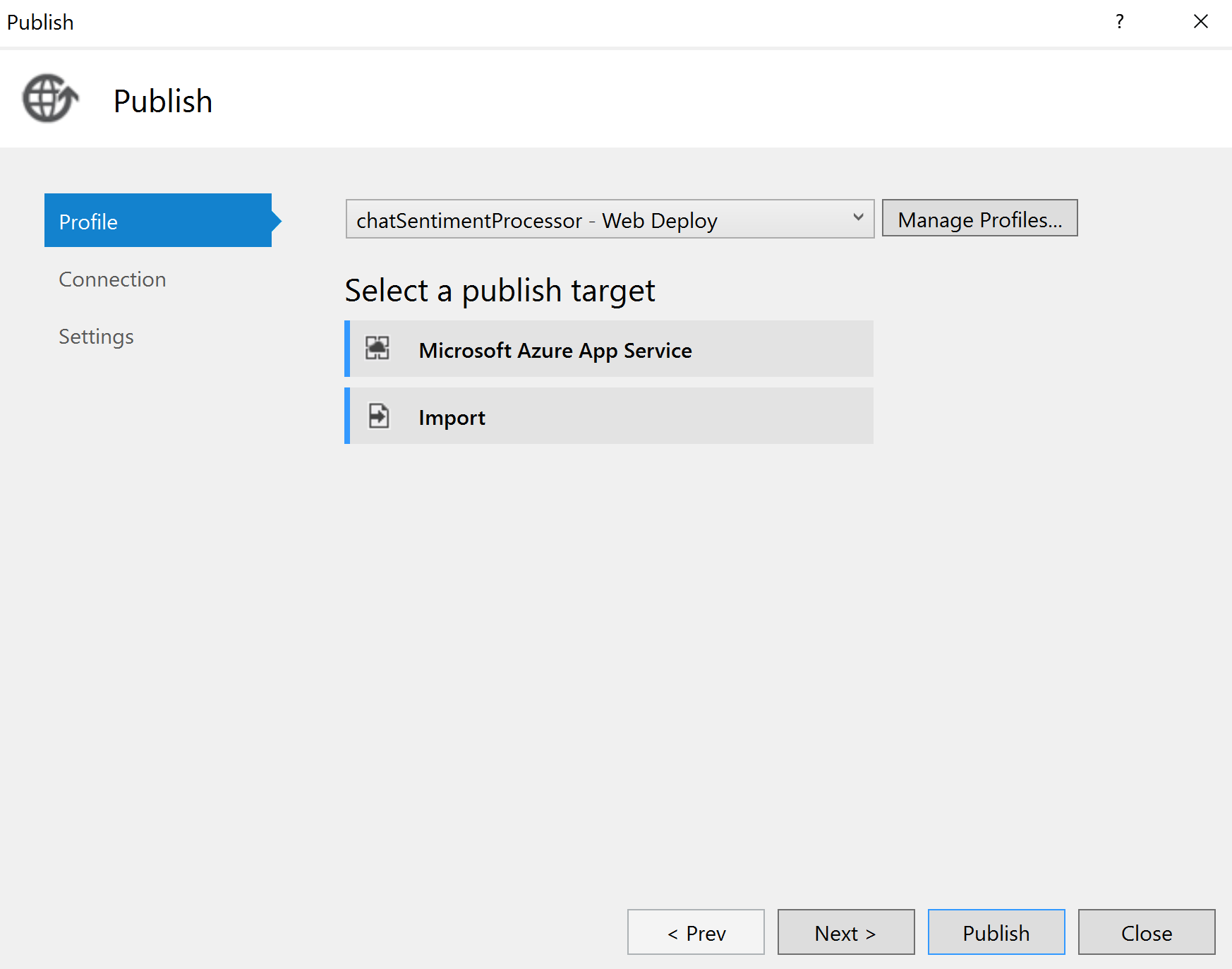
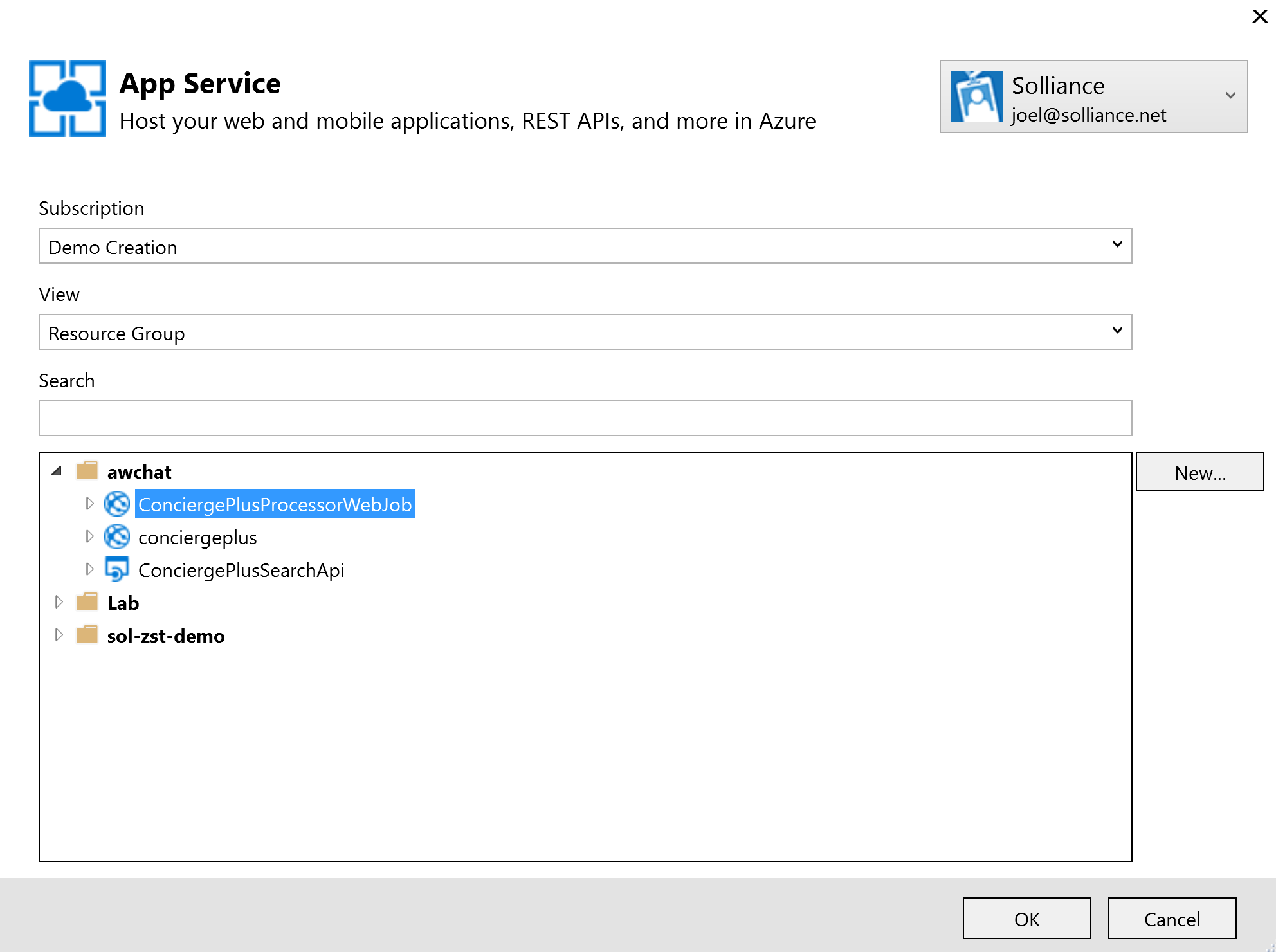
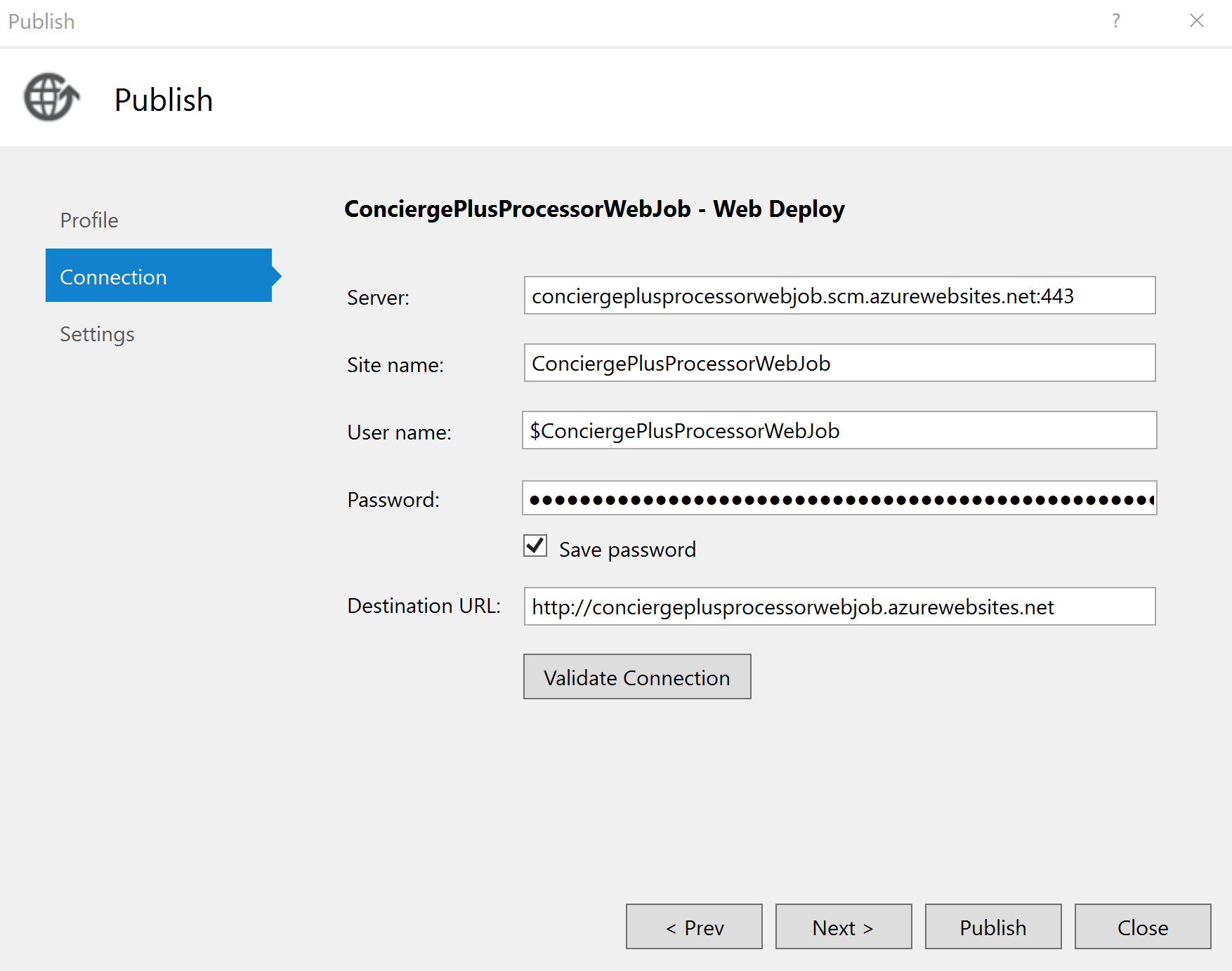
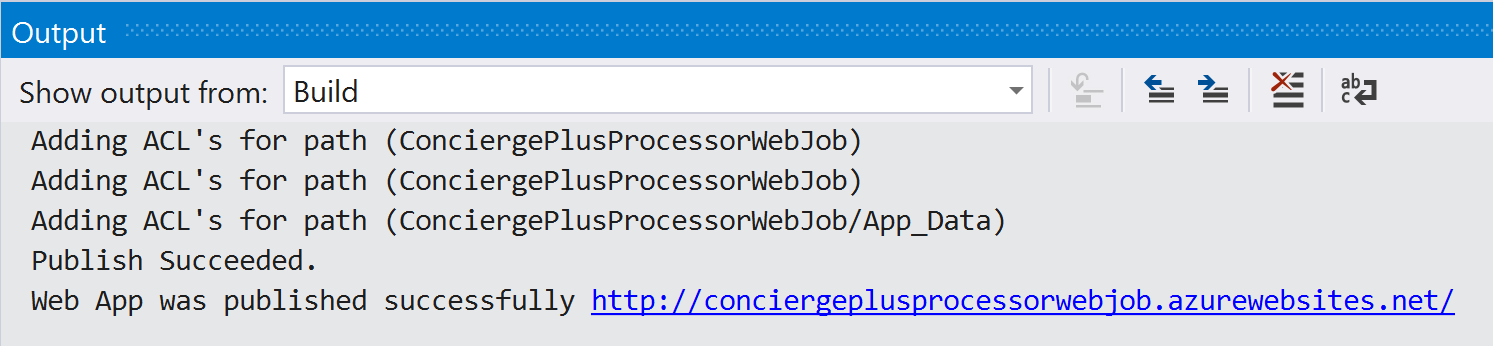
1. For the chatTopicPath and chatRequestTopicPath, enter the name of the Service Bus Topic you had created. The value is the same for both settings in this case.

## Exercise 4: Deploying the App Services

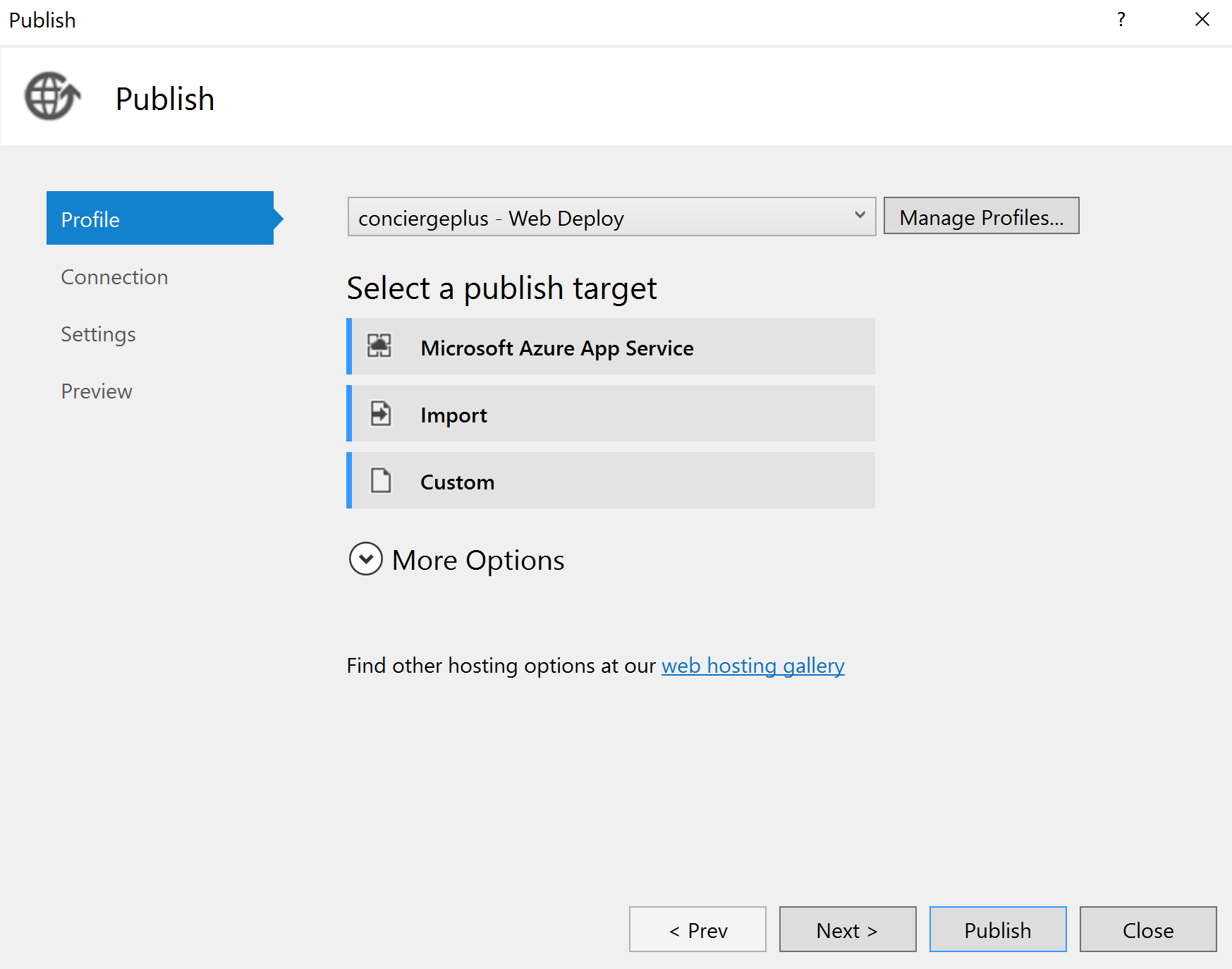
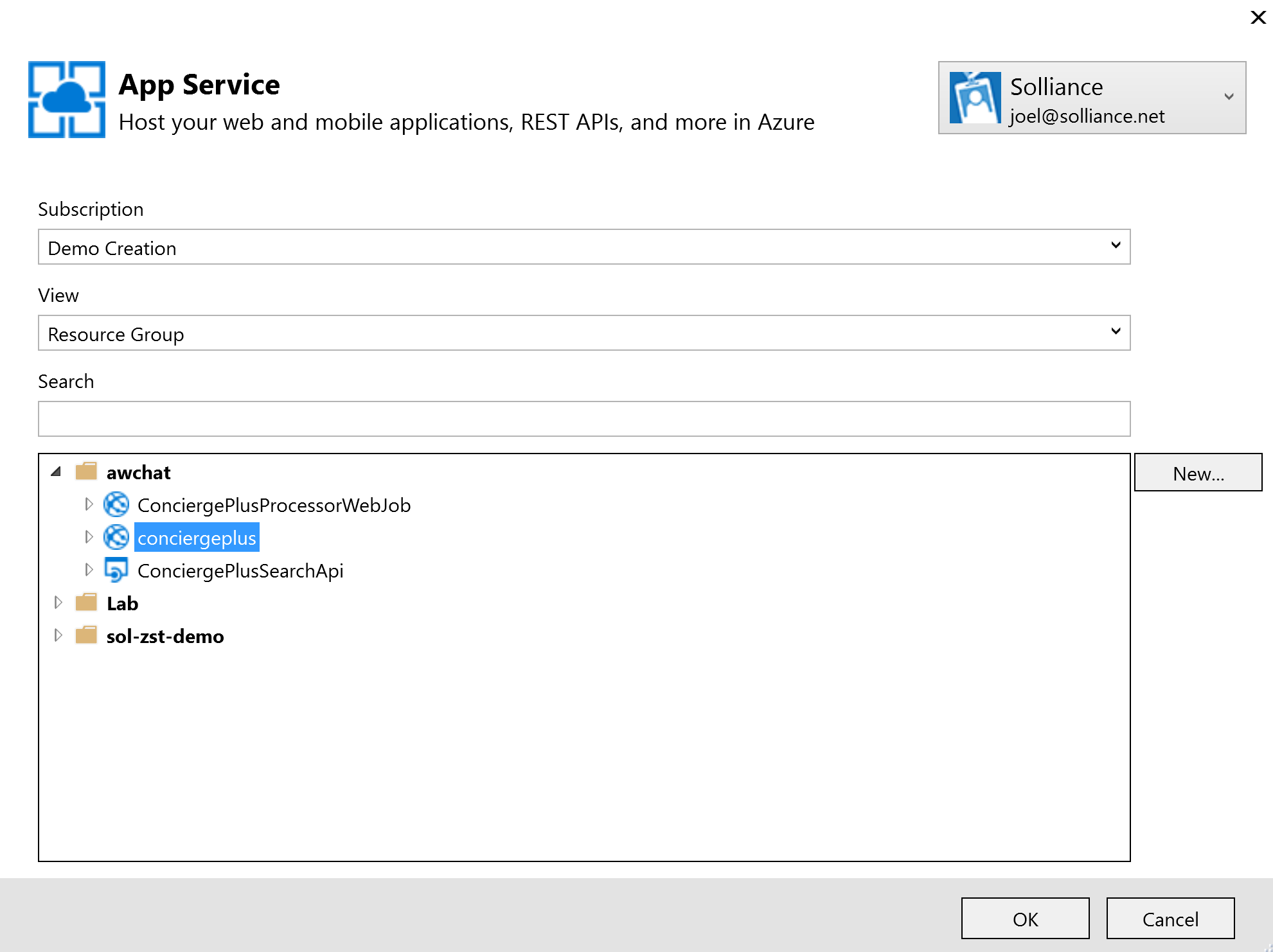
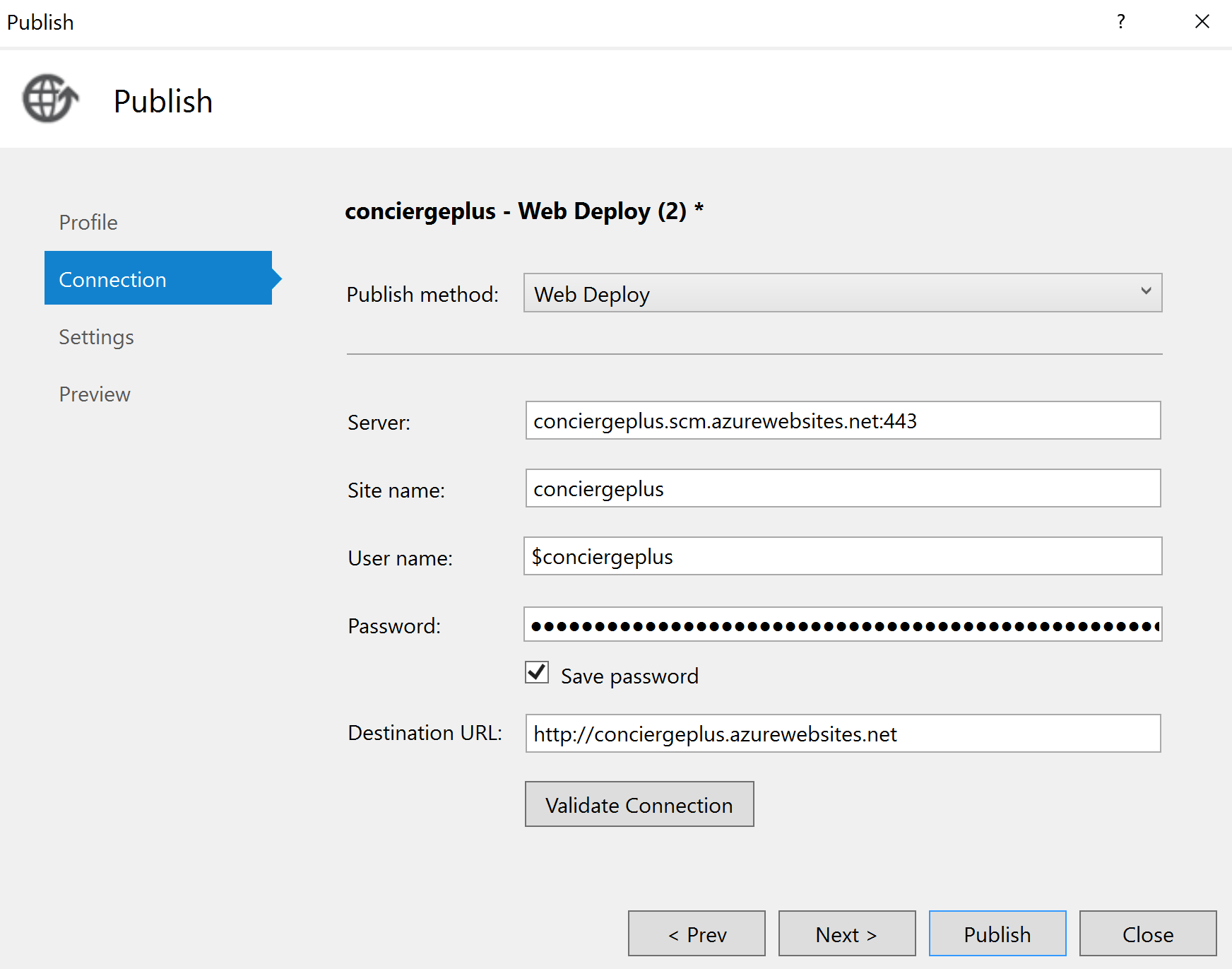
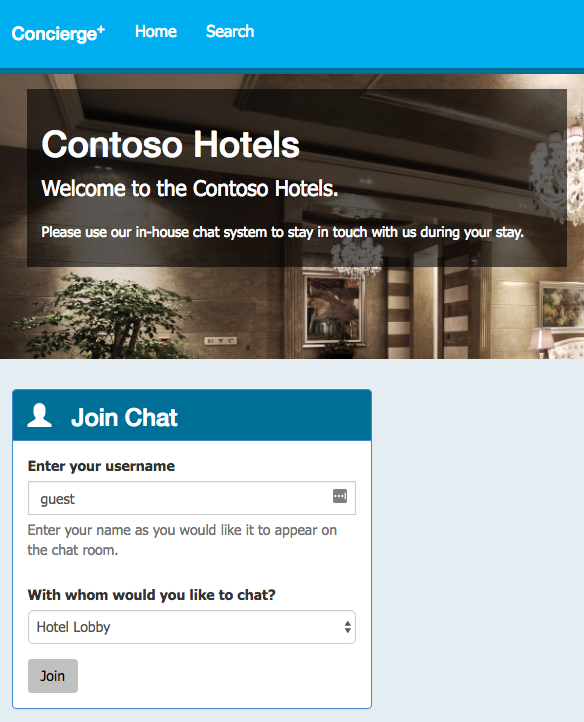
Duration: 15 minutes

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

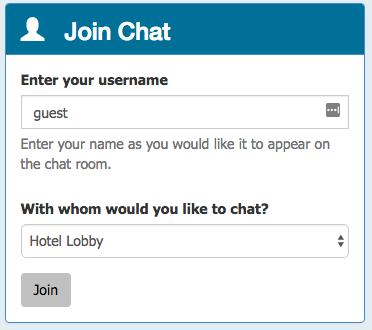
### Task 1: Publish the ChatMessageSentimentProcessor Web Job

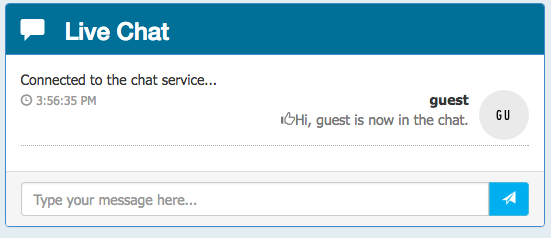
1. Within Visual Studio Solution Explorer, right-click the ChatMessageSentimentProcessor project and select Publish as Azure Web Job.  
   
2. The Add Azure WebJob dialog should appear. Leave the WebJob run mode setting to Run Continuously and select OK. 
3. In the Publish Web dialog, select Profile, 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

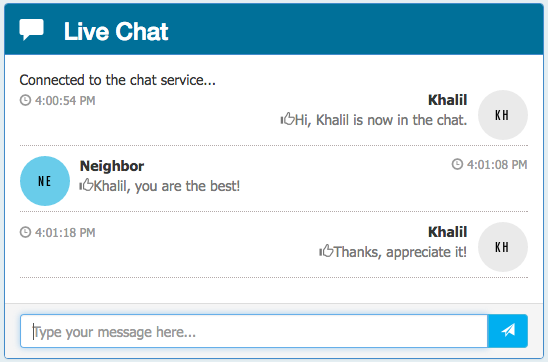
1. Within Visual Studio Solution Explorer, right-click the ChatWebApp project and select Publish.  
   
2. In the Publish Web dialog, select Profile, then select Microsoft Azure App Service publish target. 
3. If prompted, sign in with your credentials to your Azure Subscription.
4. In the App Service 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.  
    
5. Select OK.
6. Select Publish.  
    
7. 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, 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;

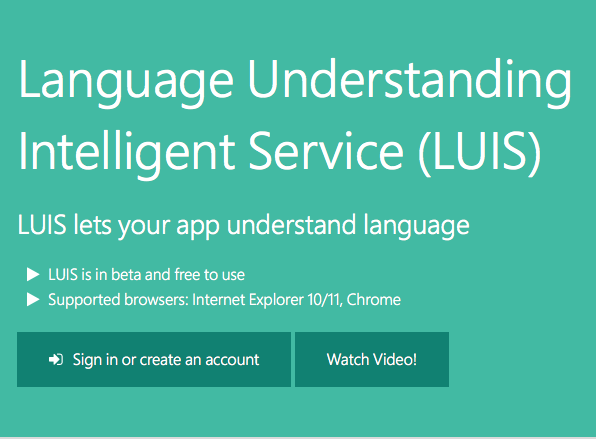
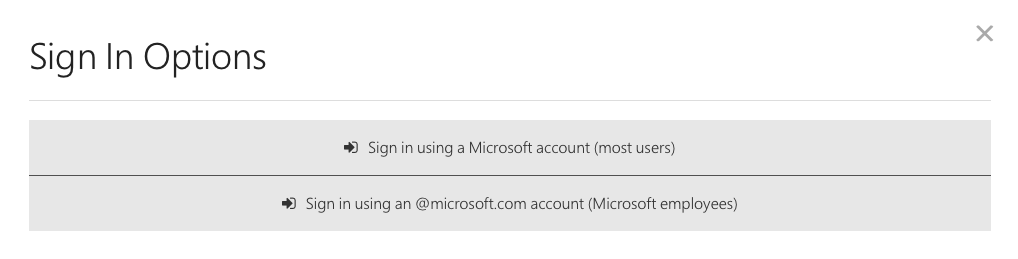
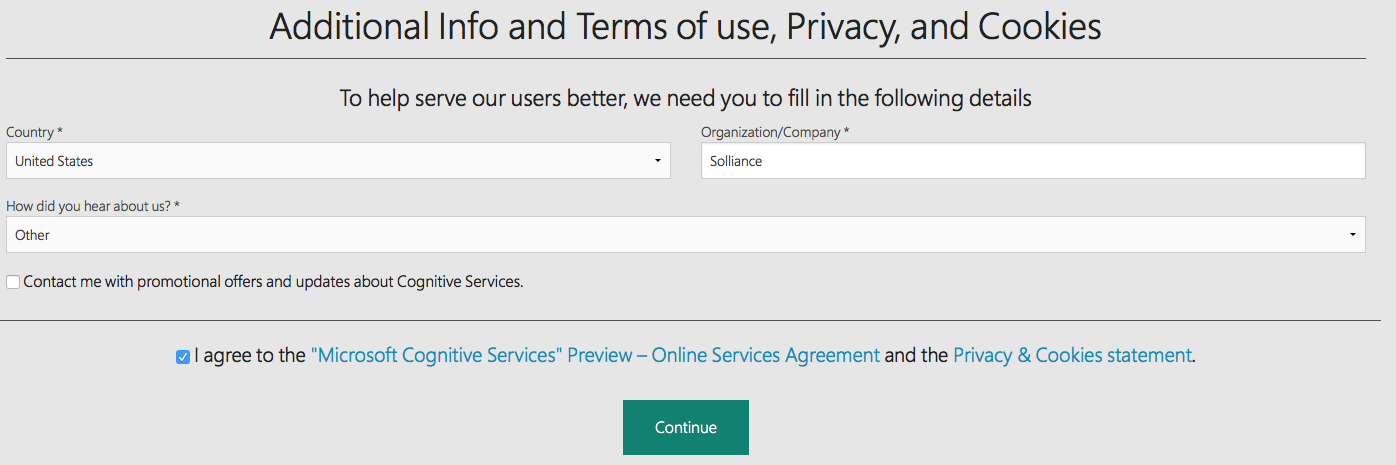
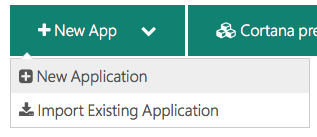
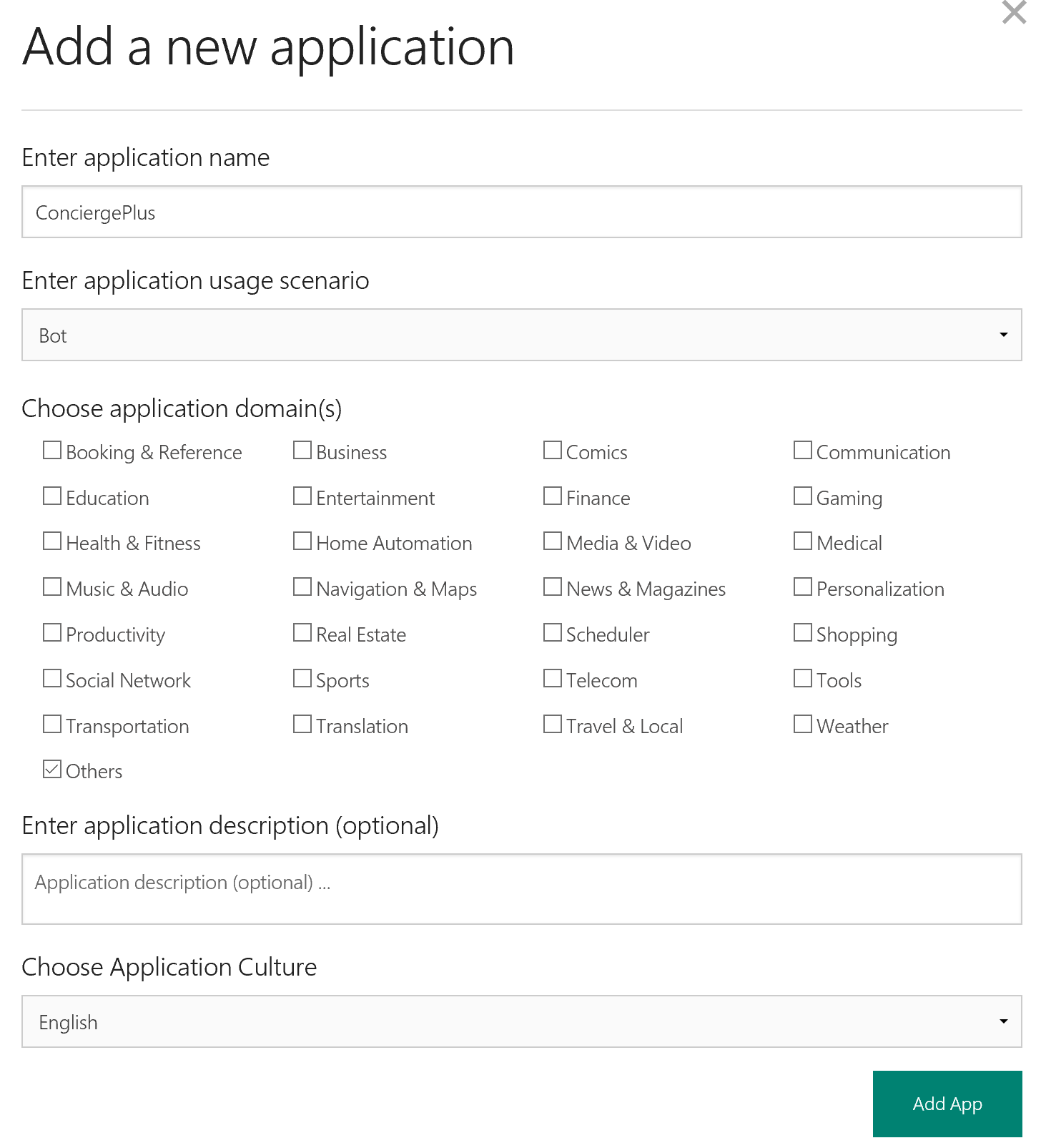
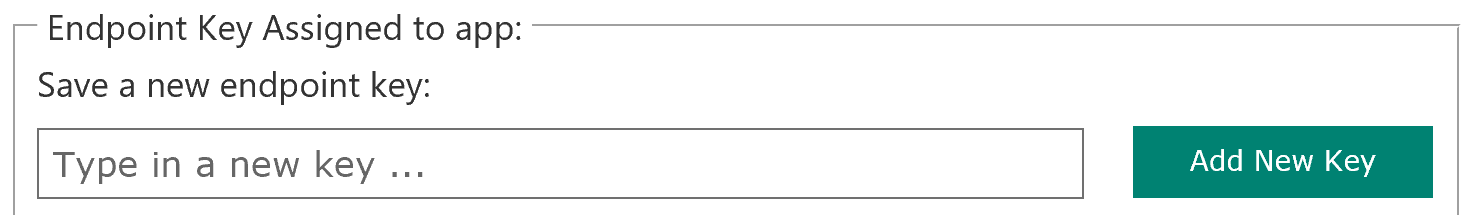
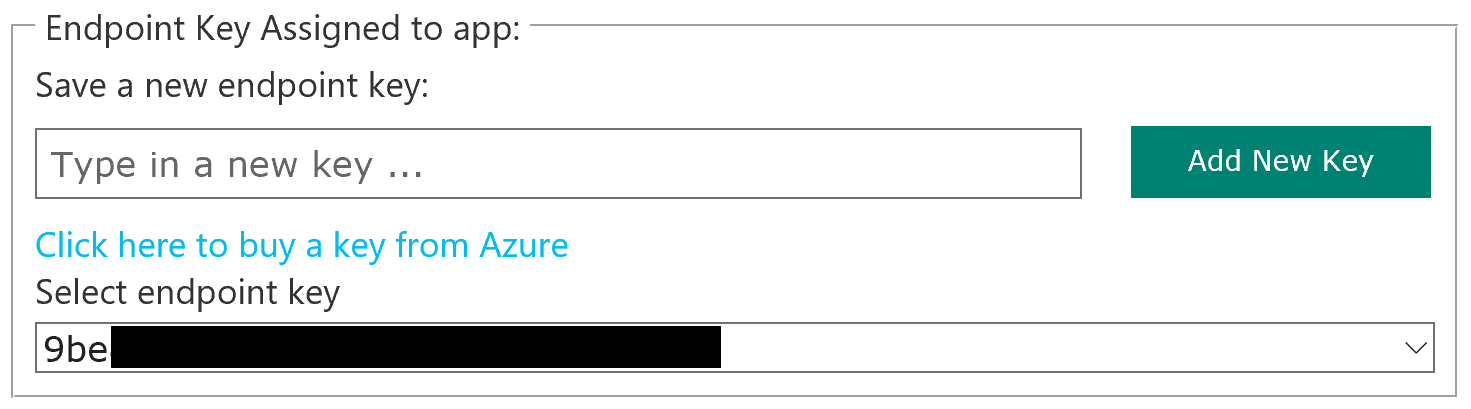
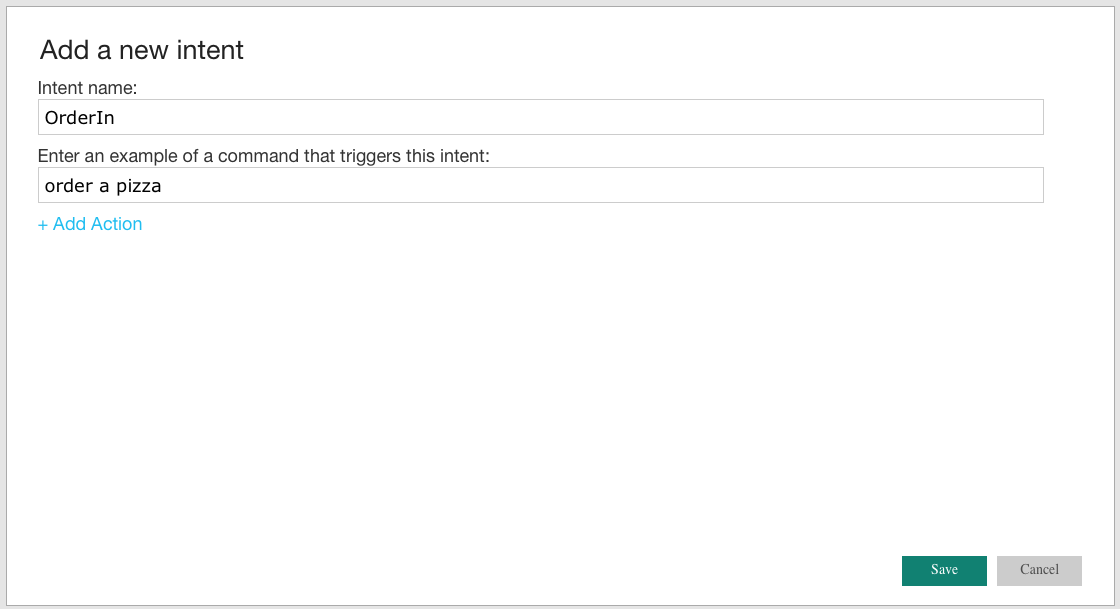
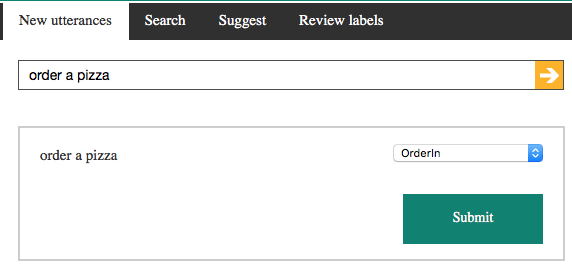
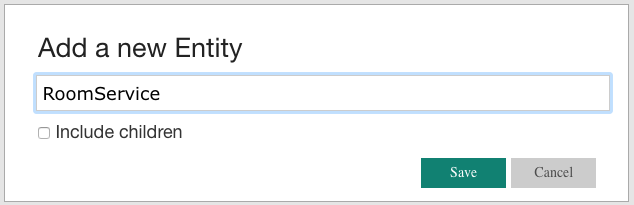
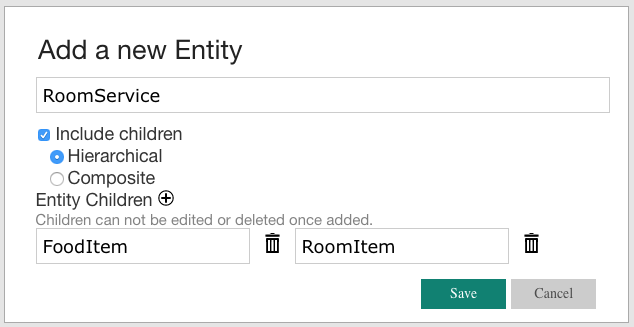
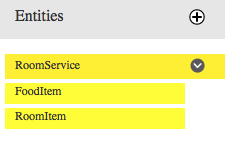
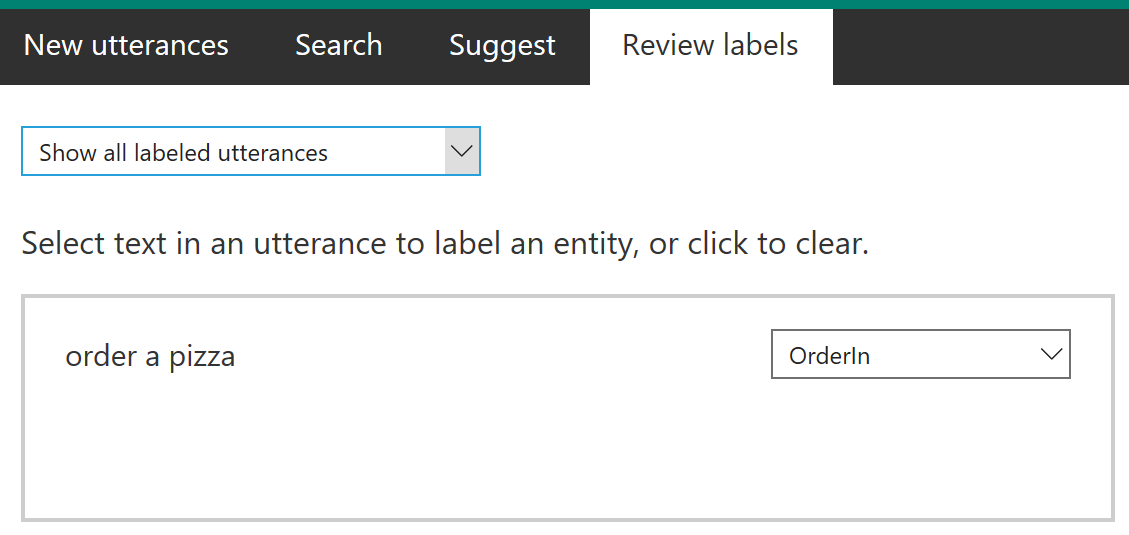
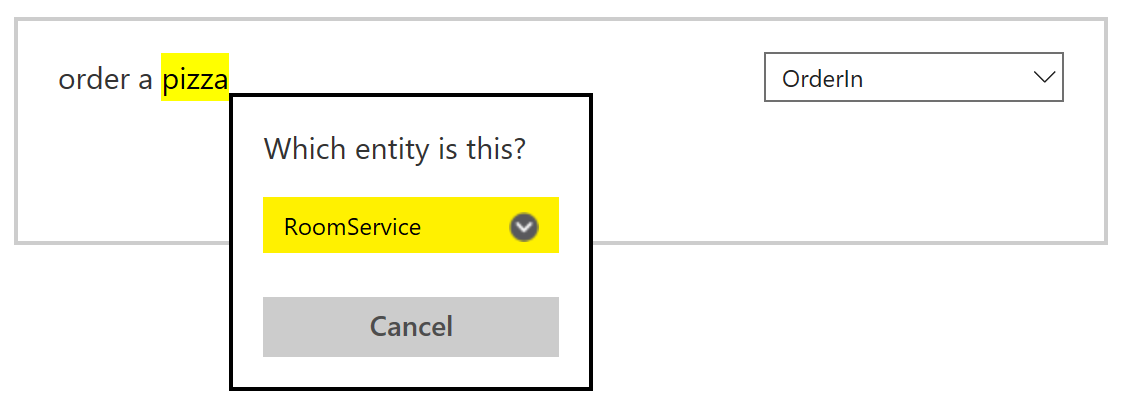
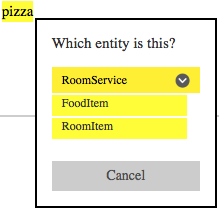
1. Finally, navigate to the IEventProcessor.ProcessEventsAsync and replace the line following TODO: 12 with the following code:

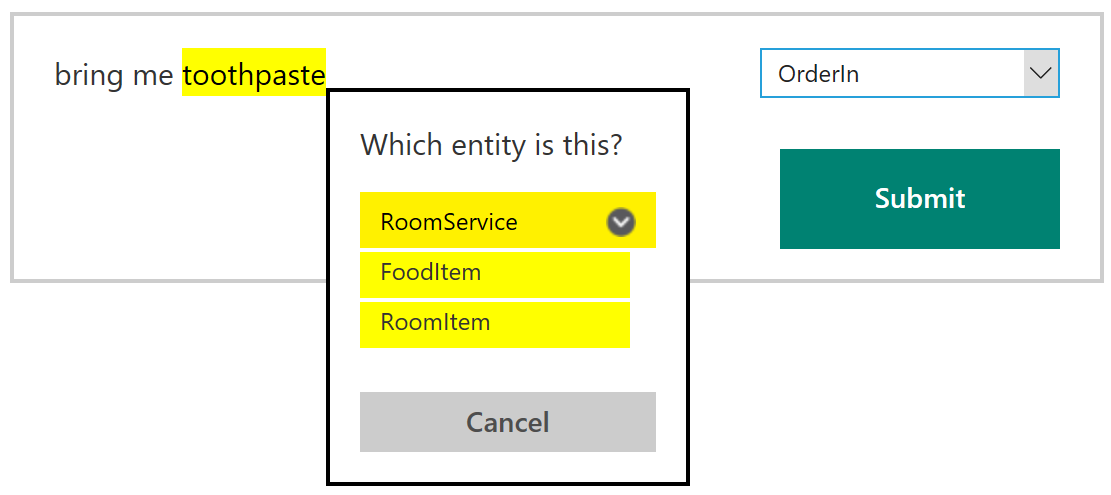
//TODO: 12 Append sentiment score to chat message object

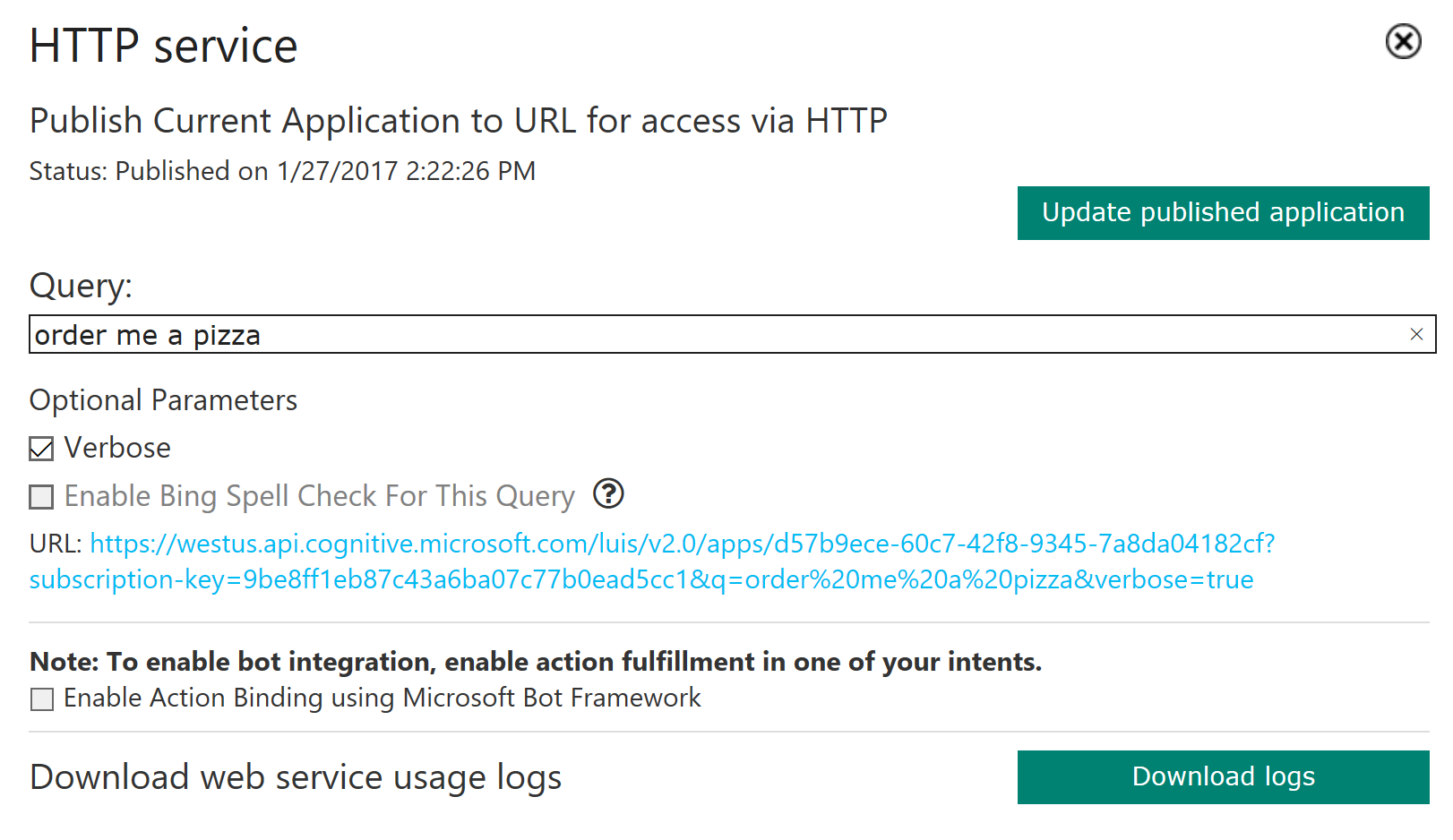
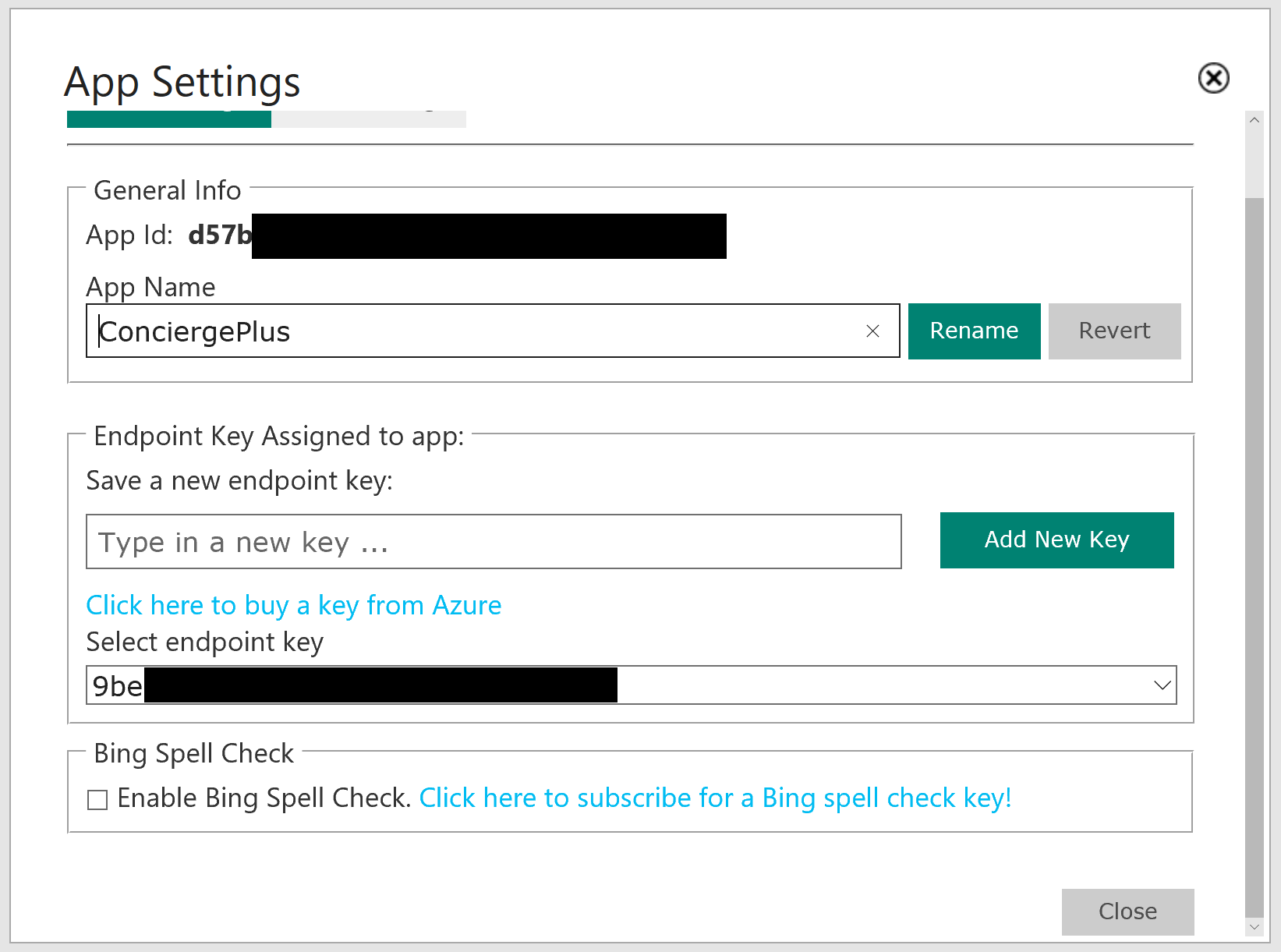
msgObj.score = await GetSentimentScore(msgObj.message);

### 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).  
   
4. Complete the sign-in process.
5. Complete the additional info and terms of use form and select Continue.  
   
6. Step through the walkthrough or select the X in the dialog to close the guided tour.
7. Under My Applications, select New App and in the drop down menu that appears select New Application.  
   
8. Complete the Add a new application form by providing a name for your LUIS app, the usage scenario (such as Bot), application domain(s), and select Add App.  
   
9. In a moment, your new app will appear.
10. On the left side, select App Settings.  
    ../../../../../../Captures/Screen%20Shot%202016-07-15%20at%203.08.12%20PM.png
11. In the New Subscription Key text box, paste the value of Key 1 as you acquired it from the Azure Portal for your Cognitive Services account and select Add Key.   
    
12. Make sure the key you added is selected within Select endpoint key.  
    
13. Close the dialog.
14. Select the plus icon to the right of Intents to add a new Intent.  
    ../../../../../../Captures/Screen%20Shot%202016-07-15%20at%203.12.27%20PM.png
15. In the Intents dialog, for the Intent Name provide OrderIn and for the example utterance enter “order a pizza”. Select Save.  
    
16. In the New Utterance tab, use the drop down menu to select OrderIn for the utterance “order a pizza” and select Submit.   
    
17. Select the plus icon to the right of Entities to add a new Entity.  
    ../../../../../../Captures/Screen%20Shot%202016-07-15%20at%203.17.28%20PM.png
18. In the Add a new Entity dialog, for the name specify “RoomService” and select the check box Include children.  
    
19. Select Hierarchical and then select the plus button to the right of Entity Children twice. The first child should be named FoodItem and the second RoomItem. Select Save.  
    
20. In the toolbar at left, select the chevron to expand the RoomService Entity so that you can see the two child entities you created.  
    
21. In the tabs at the top, select Review Labels and change the drop down menu to “Show all labeled utterances”. You should see the “order a pizza” utterance.   
    
22. In the utterance, select the word pizza so it becomes highlighted.   
    
23. Expand the chevron next to RoomService and select FoodItem.  
    
24. Select the Submit button that appears in the utterance box.  
    
25. Next, select the New Utterances tab.
26. In the Please, enter an utterance text box enter the following utterances, select the Add new arrow, and train them as described:
    1. Utterance: Bring me toothpaste
    2. Drop-down: OrderIn
    3. Text: toothpaste
    4. Entity: RoomService:RoomItem



1. Select Submit.
2. Repeat this process for the following phrases:
   1. *Bring me towels* | RoomService:RoomItem
   2. *Bring me blankets* | RoomService:RoomItem
   3. *Order a soda* | RoomService:RoomItem
   4. *Order me a pizza* | RoomService:RoomItem
3. The training of the model happens periodically, but to force it, select the Train button in the bottom-left corner of the screen.  
   ../../../../../../Captures/Screen%20Shot%202016-07-15%20at%203.36.00%20PM.png
4. On the left side, select the Publish button.
5. In the HTTP Service dialog, select Publish web service.
6. In the Query field, type the utterance “order me a pizza” and select the link that is generated below the query text box to view the JSON results of invoking the model.  
   
7. You should receive output similar to the following. Observe that it correctly identified the intent as OrderIn (in this case with a confidence of 1 or 100%) and the entity as pizza having an entity type of RoomService::FoodItem (in this case with a confidence score of 95.6%).  
   
8. In the Query, try something new like “order a pasta” that the model has not yet seen.
9. 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 dialog.
10. When you are ready to integrate LUIS into your app, go to the App Settings dialog in LUIS and copy the App ID and Endpoint Key. You will enter these into the configuration of the Event Processor.  
    
11. In Visual Studio, open the App.config for the ChatMessageSentimentProcessor project.
12. 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 and not the name of your LUIS app). For the key luisKey, set the text of the value attribute to the subscription 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
13. Save the app.config. The Event Processor is pre-configured to invoke the LUIS API using the provided App ID and key.
14. Open SentimentEventProcessor.cs and navigate to IEventProcessor.ProcessEventsAsync.
15. 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.

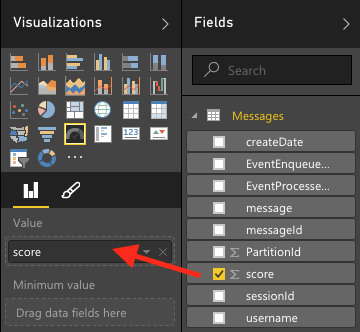
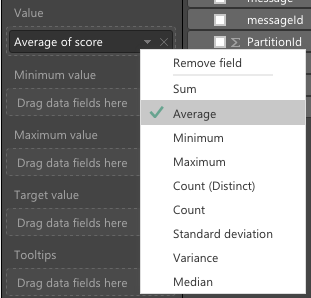
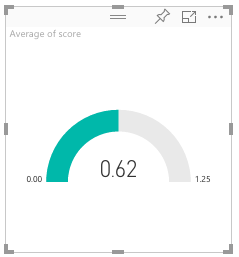
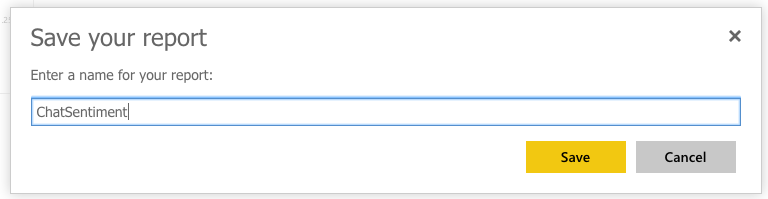


## Exercise 6: Building the Power BI dashboard

Duration: 30 minutes

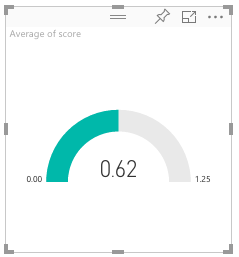
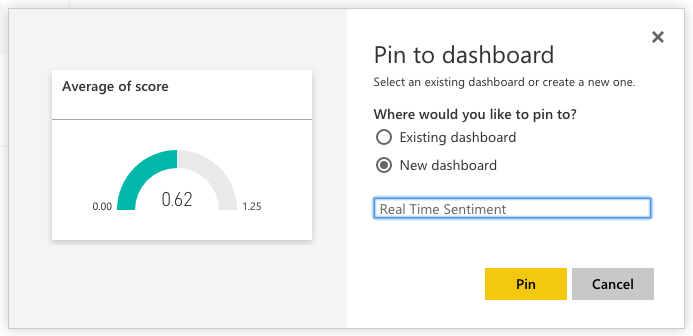
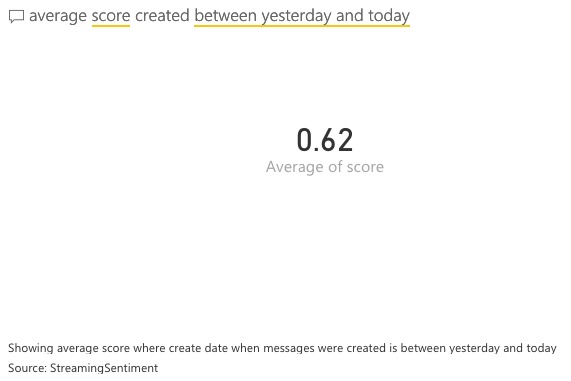
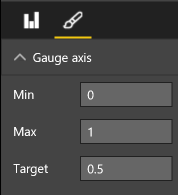
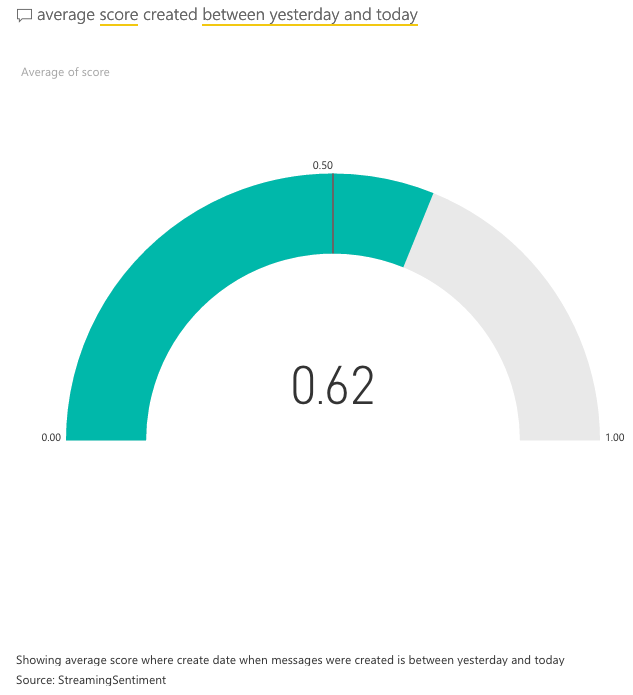
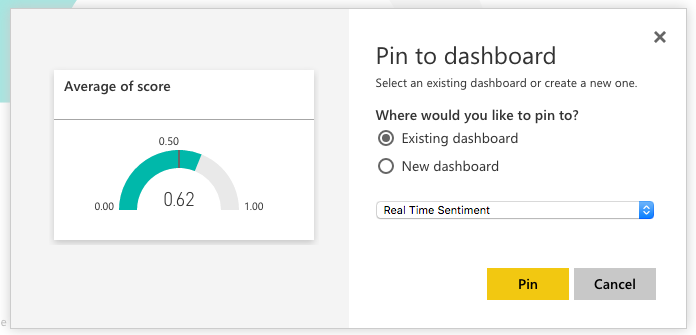
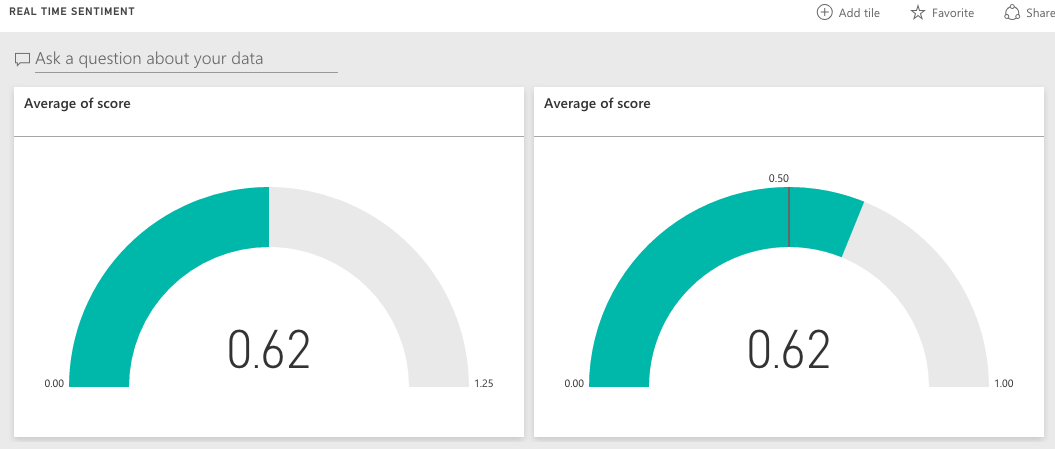
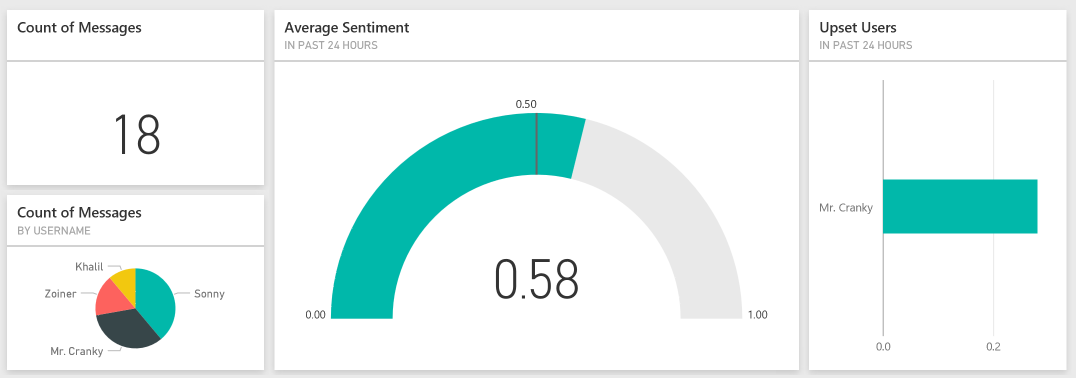
Now that you have the solution deployed and exchanging messages, you can build a Power BI dashboard that monitors the sentiments of the messages being exchanged in real time. The following steps walk through the creation of the dashboard.

### 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 the dataset named as you had configured it in the Output for the Stream Analytics job, and select that dataset.
3. On the Visualizations palette, select Gauge to create a semi-circular gauge.   
   
4. In the Fields listing, select and drag the score field and drop it onto the Value field.  
   
5. Select the drop down menu that appears where you dropped score and select Average.  
   
6. 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:  
   
7. 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.

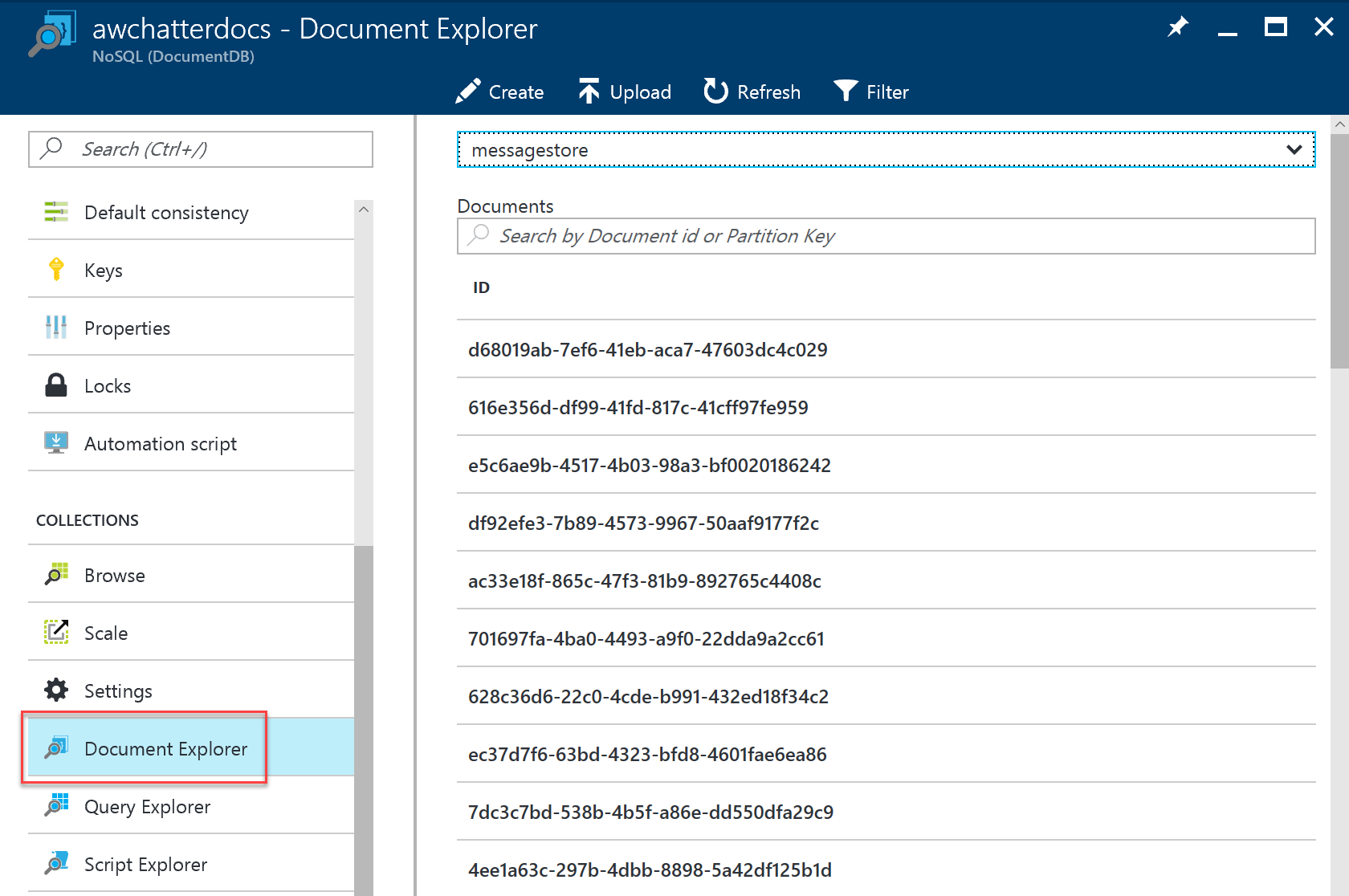
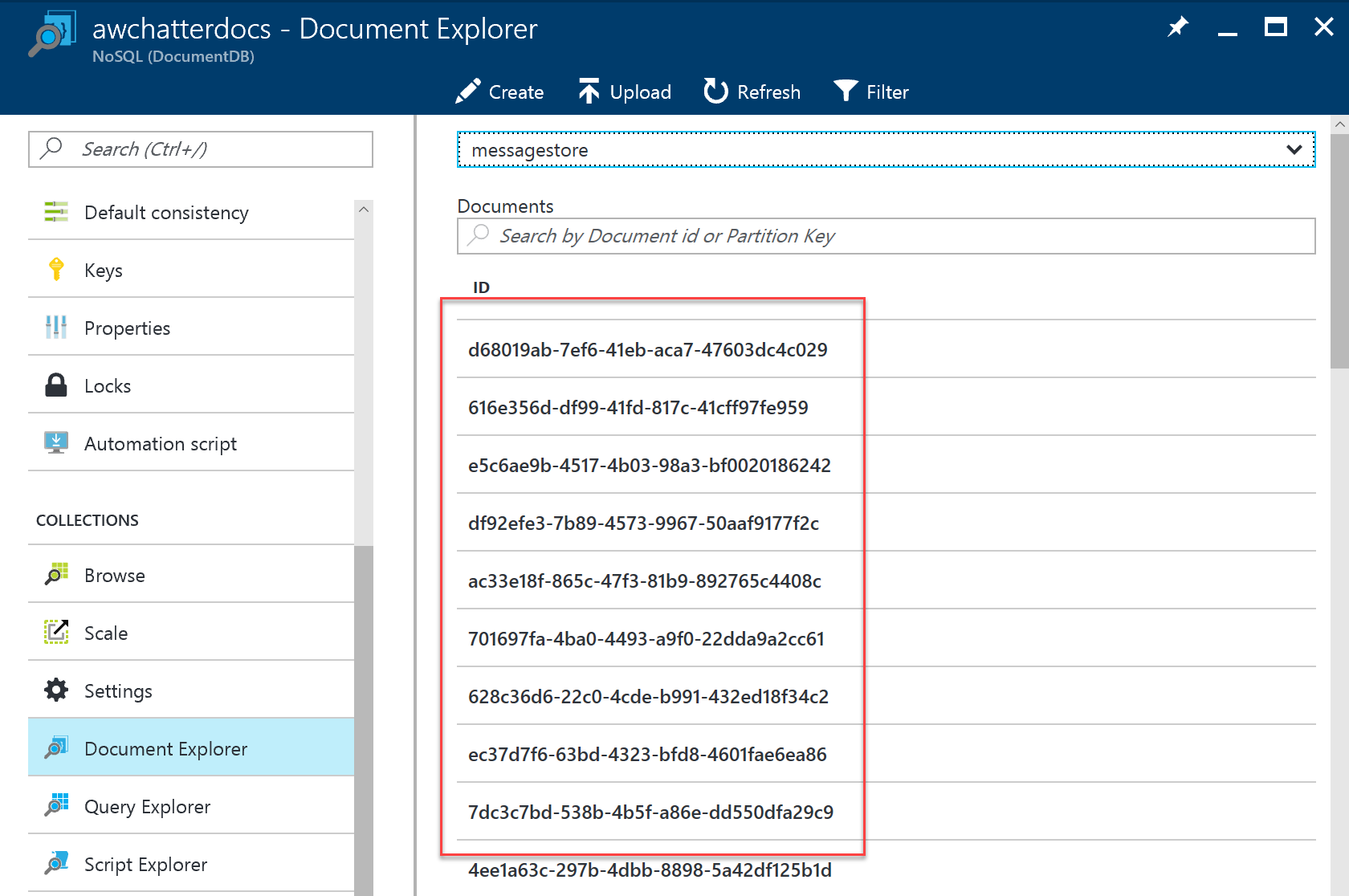
## Exercise 7: Enabling search indexing

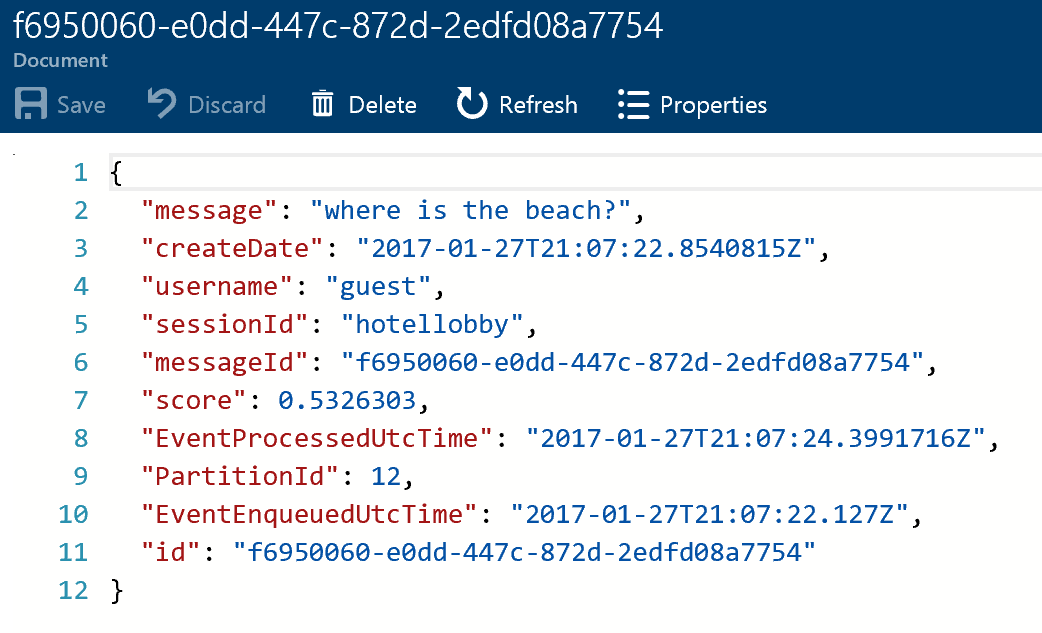
Duration: 30 minutes

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.

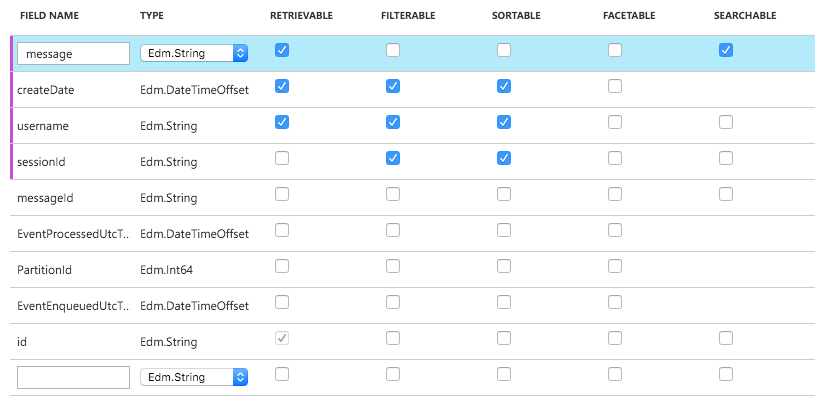
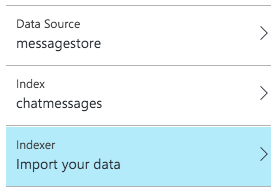
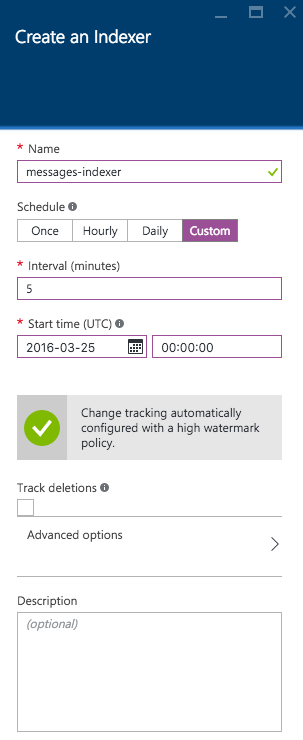
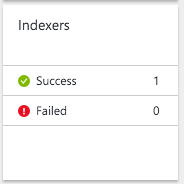
### Task 1: Verifying message archival

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

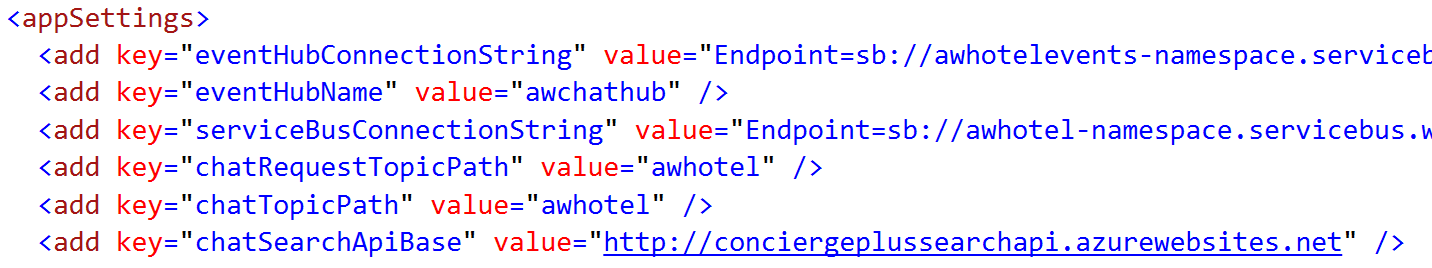
1. In the Azure Portal, navigate to your NoSQL (DocumentDB) account.
2. On the left-hand menu, select Document Explorer.   
    
3. If you see a list of document IDs, you have some data to start with! 
4. If you want to peek at the message contents, select any document in the listing.



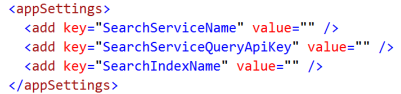
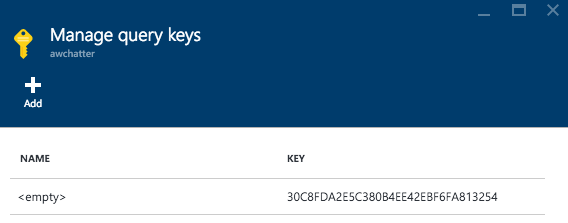
### Task 2: Creating the index and indexer

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.  
    

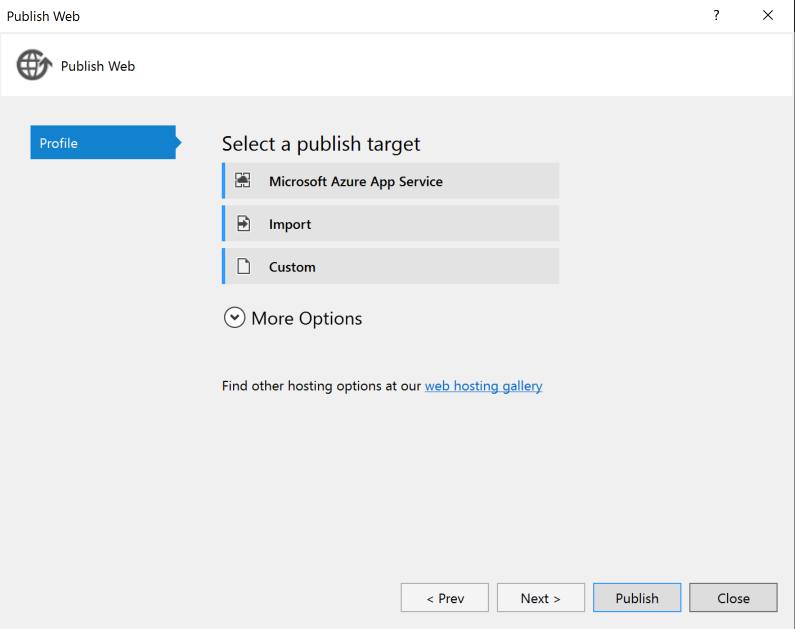
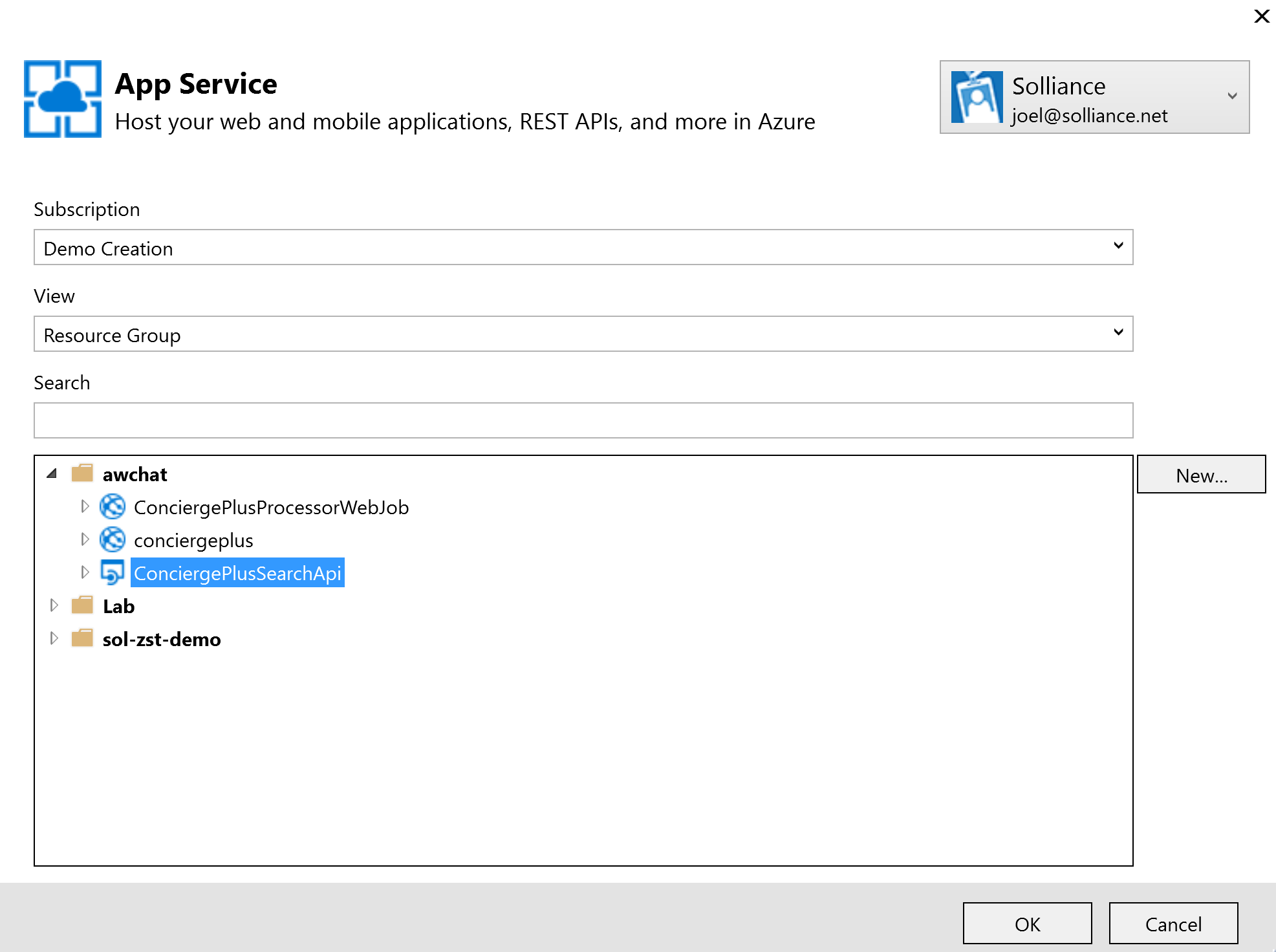
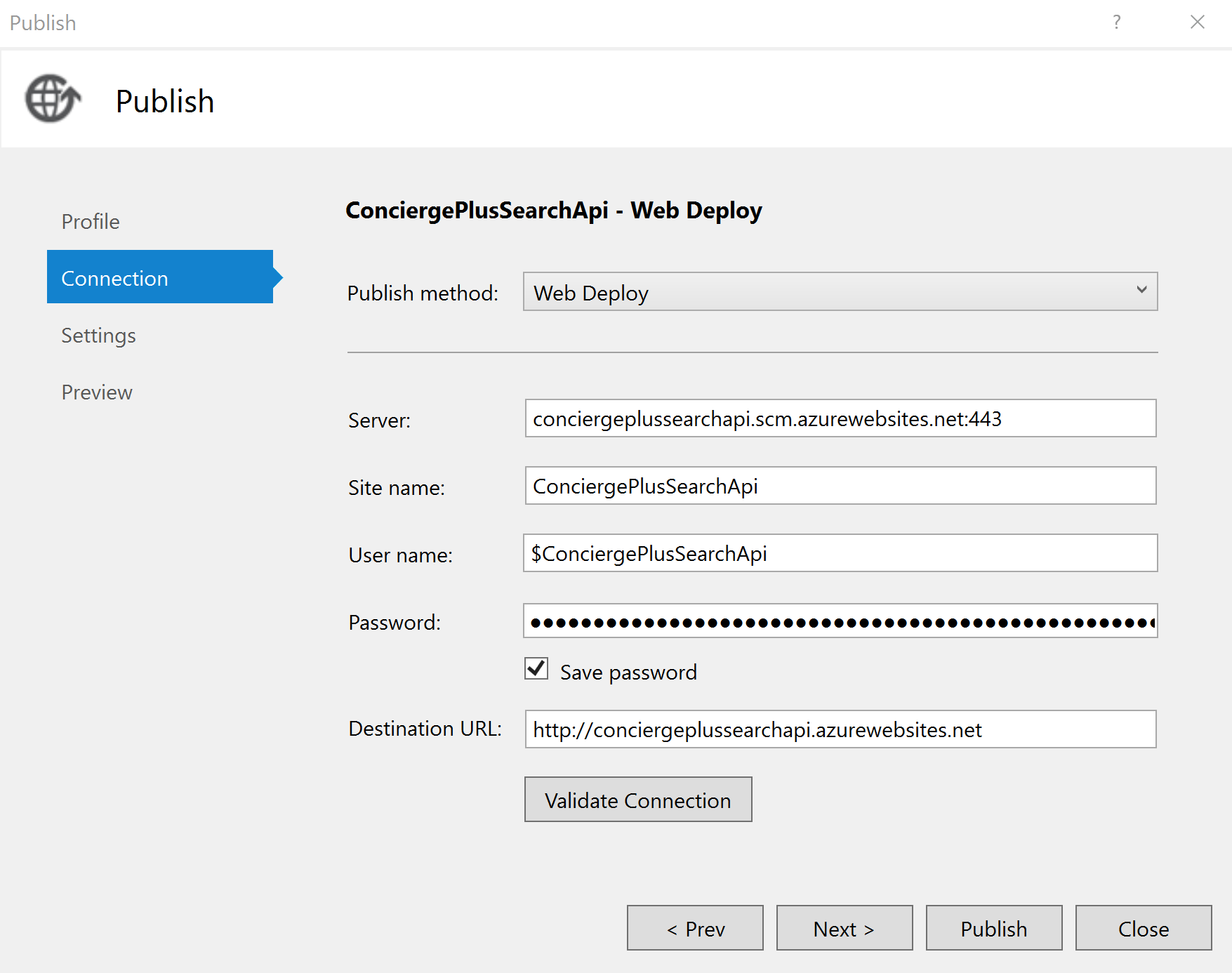
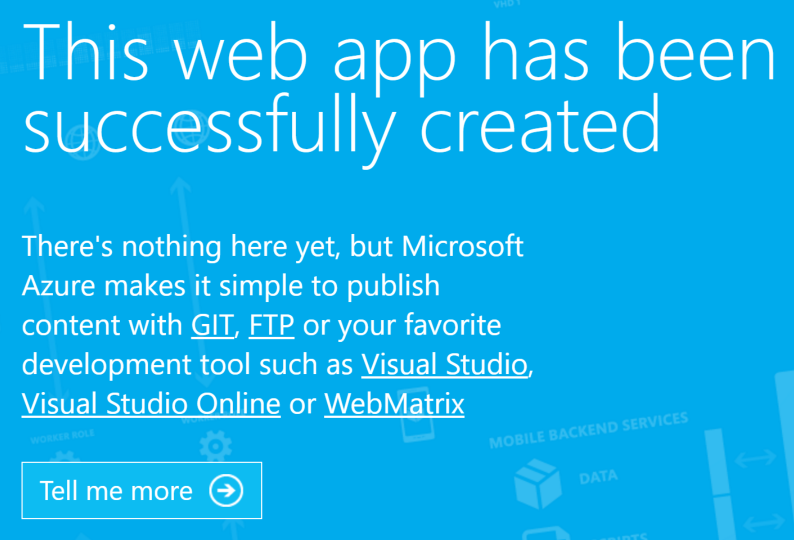
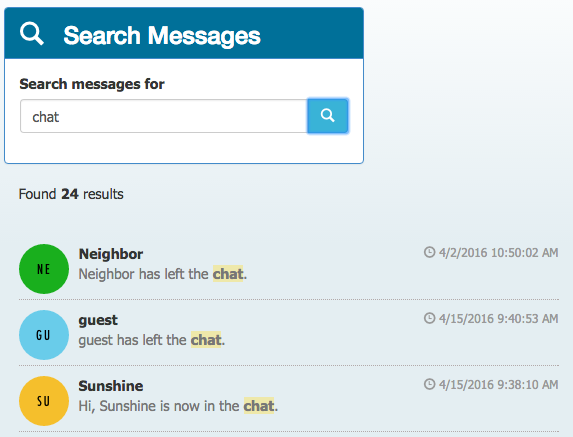
### Task 3: Update the Web App web.config

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 the Key icon.  
   ../../../../Captures/Screen%20Shot%202016-03-26%20at%208.17.55%20AM.png
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.  
   
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. Select Publish. 
8. When the publishing is complete, a browser window should appear with content similar to the following.  
   
9. 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.