ibm.com/redbooks

Building Cognitive Applications with IBM Watson Services: Volume 2 Conversation

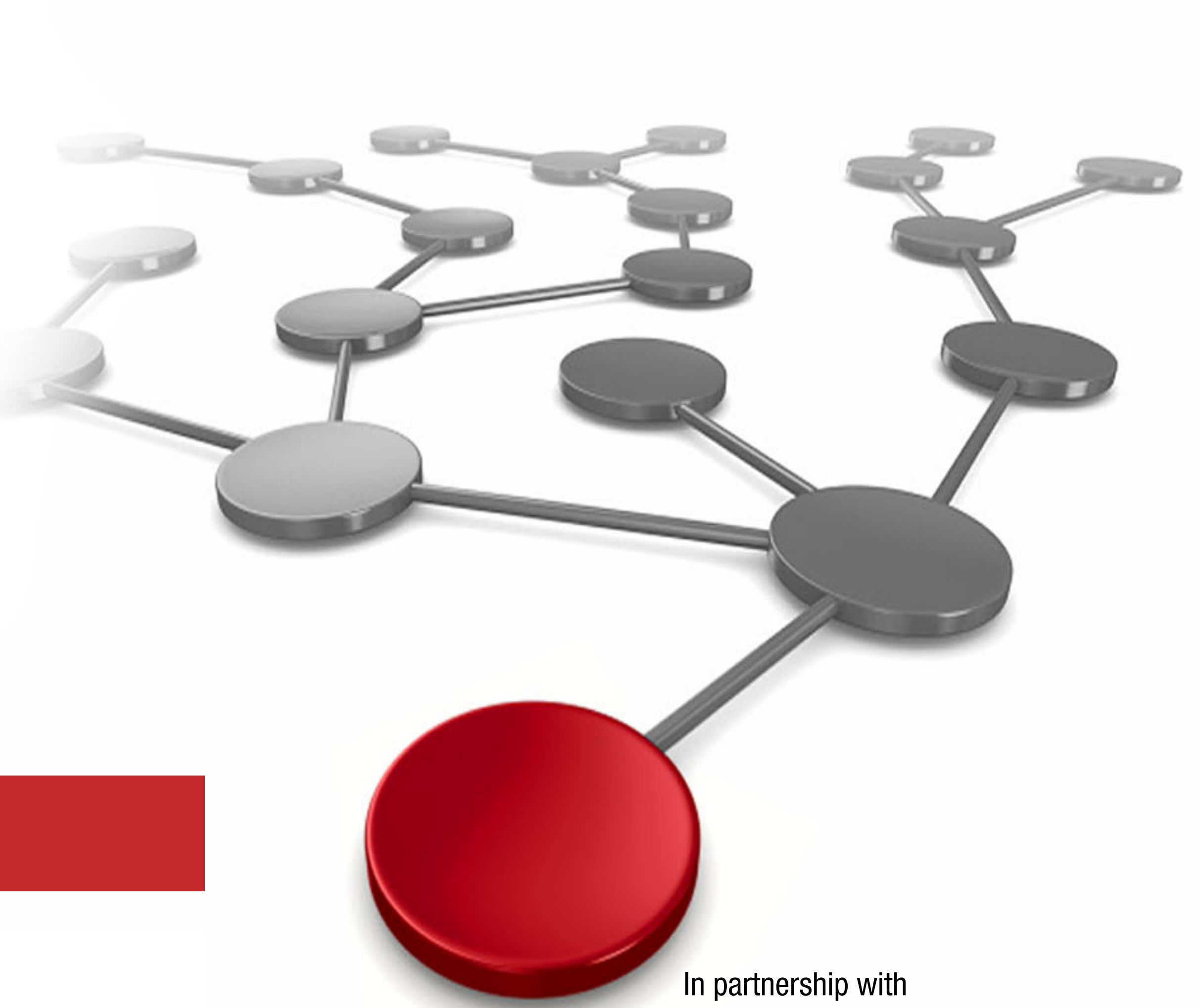
Ahmed Azraq

Hala Aziz

Nicolas Nappe

Cesar Rodriguez Bravo

Lak Sri



Cloud

In partnership with
IBM **Skills Academy Program**

IBM Redbooks

International Technical Support Organization

Building Cognitive Applications with IBM Watson Services: Volume 2 Conversation

May 2017

SG24-8394-00

Note: Before using this information and the product it supports, read the information in "Notices" on page vii.

First Edition (May 2017)

This edition applies to IBM Watson services in IBM Bluemix.

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Ask Watson™
Bluemix®
developerWorks®
Global Business Services®
Global Technology Services®
IBM®
IBM MobileFirst™
IBM Watson®
IBM Watson IoT™
Redbooks®
Redbooks (logo) ®
Redpapers™
Tivoli®
Watson™
Watson IoT™
WebSphere®

The following terms are trademarks of other companies:

The Weather Company, and Wundersearch are trademarks or registered trademarks of TWC Product and Technology LLC, an IBM Company.

ITIL is a Registered Trade Mark of AXELOS Limited.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The *Building Cognitive Applications with IBM Watson Services* series is a seven-volume collection that introduces IBM® Watson™ cognitive computing services. The series includes an overview of specific IBM Watson® services with their associated architectures and simple code examples. Each volume describes how you can use and implement these services in your applications through practical use cases.

The series includes the following volumes:

- *Volume 1 Getting Started*, SG24-8387
- *Volume 2 Conversation*, SG24-8394
- *Volume 3 Visual Recognition*, SG24-8393
- *Volume 4 Natural Language Classifier*, SG24-8391
- *Volume 5 Language Translator*, SG24-8392
- *Volume 6 Speech to Text and Text to Speech*, SG24-8388
- *Volume 7 Natural Language Understanding*, SG24-8398

Whether you are a beginner or an experienced developer, this collection provides the information you need to start your research on Watson services. If your goal is to become more familiar with Watson in relation to your current environment, or if you are evaluating cognitive computing, this collection can serve as a powerful learning tool.

This IBM Redbooks® publication, Volume 2, describes how the Watson Conversation service can be used to create chatbots and user agents that understand natural-language input and communicate with your users simulating a real human conversation. It introduces the concepts that you need to understand in order to use the Watson Conversation service. It provides examples of applications that integrate the Watson Conversation service with other IBM Bluemix® services, such as the IBM IoT Platform, Text to Speech, Speech to Text, and Weather Company Data, to implement practical use cases. You can develop and deploy the sample applications by following along in a step-by-step approach and using provided code snippets. Alternatively, you can download an existing Git project to more quickly deploy the application.

Authors

This book was produced by a team of specialists from around the world working in collaboration with the IBM International Technical Support Organization.

Ahmed Azraq is a Certified IT Specialist in IBM Egypt. Since joining IBM in 2012, Ahmed worked as a Senior Cloud Developer, Technical Team Leader, and Architect in the IBM Middle East and Africa (MEA) Client Innovation Center, which is part of IBM Global Business Services® (GBS). His areas of expertise include cloud, IBM Business Process Manager, middleware integration, Java, and IBM Watson. Ahmed has acquired several professional certifications, including Open Group IT Specialist, IBM Bluemix, Java EE, IBM Business Process Manager, Agile development process, and IBM Design Thinking. Ahmed has delivered training on IBM Bluemix, DevOps, hybrid cloud Integration, Node.js, Watson APIs, and IBM WebSphere® Liberty Profile to IBM clients, IBM Business Partners, and university students and professors around the world. He is the recipient of several awards, including Eminence and Excellence Award in the IBM Watson worldwide competition *Cognitive Build*, the IBM Service Excellence Award for showing excellent client value behaviors, and knowledge-sharing award. Ahmed is also a published author for IBM Redbooks *Essentials of Cloud Application Development on IBM Bluemix*, SG24-8374.

Hala Aziz is an Experienced Certified IT Specialist in the Cairo Technology Development Center (CTDC) in IBM Egypt. She has more than 10 years of experience in IBM Application and Integration Middleware software and IBM Cloud such as IBM WebSphere Application Server, IBM WebSphere Portal, IBM MobileFirst™, IBM Endpoint Manager, IBM Bluemix, and IBM Watson services. She worked as a consultant on eGovernment, telecom, and banking solutions for clients in Egypt, Saudi Arabia, Dubai, Oman, and Switzerland. Hala has several technical professional certifications, such as Certified Application Developer for IBM Web Content Manager, IBM MobileFirst and Cloud Platform Application Developer v1, and she has published several articles and IBM Redbooks publications. Hala has delivered IBM internal education and client enablement training workshops around the world.

Nicolas Nappe is an Open Group Master Certified IT Specialist and IBM Certified Cloud Advisor working in IBM Global Technology Services®, IBM Argentina. Nicolas works as a DevOps Specialist, with a focus in infrastructure automation and cloud computing. Nicolas has more than 15 years of experience in UNIX technologies, Information Technology Infrastructure Library (ITIL), and IT service management (ITSM). Nicolas developed the *Cognimation* solution that uses Watson cognitive service to summarize documents and deliver them in a presentation format. Cognimation uses Watson Alchemy Language and Natural Language Processing to extract the most relevant concepts and deliver a presentation explaining the concepts customized for the user.

Cesar Rodriguez Bravo is a Program Manager in the IBM North America Cyber Security Project Office. Cesar holds a Master of Science degree in Cyber Security and many certifications in Project Management including PMP, Scrum Master, Scrum Developer, Scrum Product Owner, Agile Expert, and Scrum Trainer. Cesar is also certified as an IBM Expert Project Manager and is currently the Project Manager competence leader for IBM Costa Rica. Cesar is a university professor; he enjoys teaching students about new technologies such as Internet of Things (IoT) and cognitive computing. Cesar is currently working with IBM Master Inventors developing patents in the cognitive and cyber security domains. Cesar won the Internet of Things contest in the regional IBM Technical Exchange with a project based on IBM Watson technologies. Cesar won an IBM worldwide contest (with votes from 41 countries) with the idea of an IoT robot that helps children learn by using IBM Watson capabilities.

Lak Sri currently serves as a Program Director in IBM developerWorks® part of the IBM Digital Business Group organization. Lak leads innovation in the developer activation space. He was the Technical Leader for the *Building Cognitive Applications with IBM Watson Services* Redbooks series. Lak led the development of the IBM Cloud Application Developer Certification program and the associated course. Earlier he worked as a Solution Architect for Enterprise Solutions in Fortune 500 companies using IBM Tivoli® products. He also built strategic partnerships in education and IBM Watson IoT™. Lak is an advocate and a mentor in several technology areas, and he volunteers to plan and support local community programs.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks Project Leader, ITSO.

Thanks to the following people for their contributions to this project:

Swin Voon Cheok
Ecosystem Development (EcoD) Strategic Initiative, IBM Systems

Iain McIntosh
IBM Watson and Cloud Platform

Juan Pablo Napoli
Skills Academy Worldwide Leader, IBM Global University Programs

Teja Tummalapalli
IBM Digital Business Group

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Basics of Conversation service

With the IBM Watson Conversation service, you can create an application and user agents that understand natural-language input and communicate with your users simulating a real human conversation. Conversation service uses machine learning to respond to customers in a way that simulates a conversation between humans.

This chapter introduces the concepts you need to understand to use the Watson Conversation service.

The following topics are covered in this chapter:

- Introduction to Watson Conversation service
- How to use the Conversation service
- Conversation concepts
- Conclusion
- References

1.1 Introduction to Watson Conversation service

Figure 1-1 depicts the overall architecture of a solution that includes an application that integrates the Conversation service.

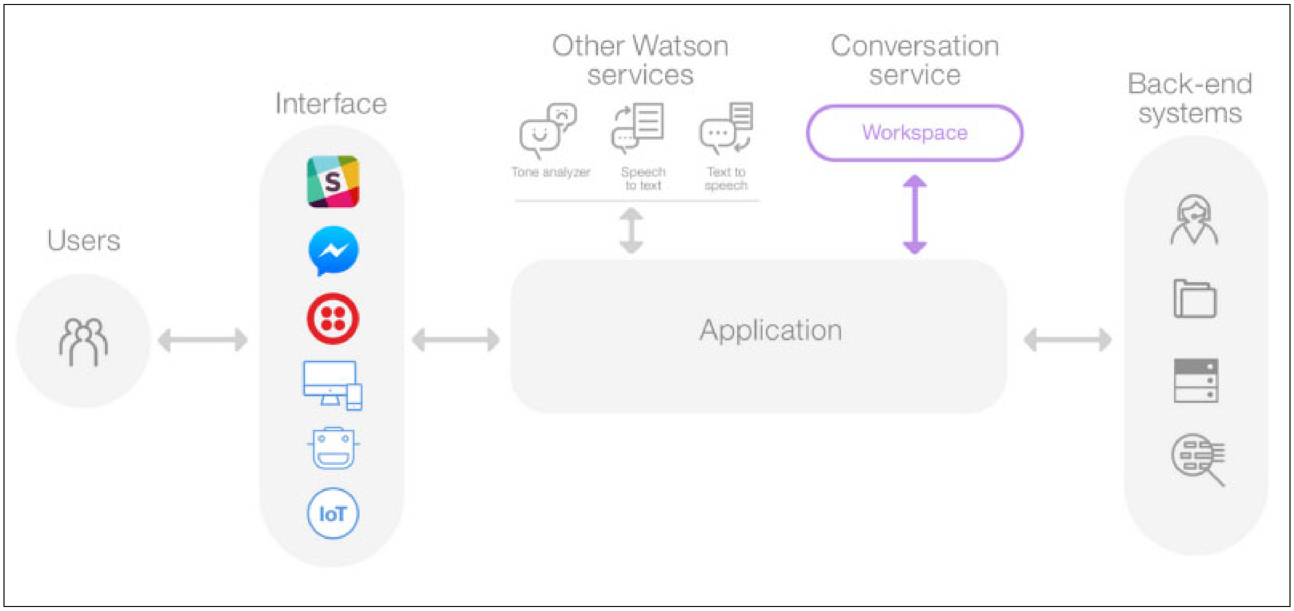


Figure 1-1 Typical architecture of a Conversation application

Consider this information about the architecture in Figure 1-1:

- Users interact with your application through one or more of your chosen interfaces. Common choices might be messaging services, a chat window within a website, or even audio interfaces when combined with Watson Speech to Text services.
- The application sends the user input to the Conversation service:
  - The application connects to a *workspace*. The natural-language processing for the Conversation service happens inside a workspace, which is a container for all of the artifacts that define the conversation flow for an application. You can define multiple workspaces in a Watson Conversation service instance. Each workspace will be trained to recognize certain concepts and to direct the conversation flow that governs user interaction.
  - The Conversation service interprets the user input, directs the flow of the conversation and gathers information that it needs. The Watson Conversation service uses machine learning to identify the concepts it was trained for. Based on what concepts it identifies, it directs the conversation flow, to provide the user with information or to gather additional information from users.
  - You can connect additional Watson services to analyze user input, such as Tone Analyzer or Speech to Text.
- Your application can also interact with existing back-end systems based on the user's intent and additional information. For example, search for information in public or private databases, open tickets, show diagrams and maps, or write the user input into your systems of record.

The steps for setting up a working Conversation service are described in 1.2, "How to use the Conversation service" on page 3.

1.1.1 Supported languages

The natural language classifiers used in the Conversation service support English, Portuguese (Brazilian), French, Italian, Spanish, and Japanese, and has experimental support for German, Traditional Chinese, Simplified Chinese, and Dutch. Arabic is supported through the use of the Conversation API but not through the tooling interface.

1.1.2 Innovative ways to use the Watson Conversation service

After completing this book, you should be able to implement all kinds of innovative and creative interactions with your users in your applications. Here are some examples:

- You can integrate your application with the Watson Conversation, Speech to Text and Text to Speech services and drive your application by speaking to it. You can use Watson Tone Analyzer to identify the emotions, social tendencies, and writing style expressed by your users.
- In Watson Developer Cloud, you can find an example of a Conversation agent helping your users while they drive cars. You can integrate this application with Weather Company data, to retrieve weather-related information while driving your car.
- You can build an agent to chat with young people around the world and engage them in community issues, similar to the UNICEF custom social platform, U-Report.
- You can build a natural language tutor to chat with your users and teach them as they learn to play a game that you built, giving advice or supporting them.
- A chatterbot can be created that is present in a dialog between two other people and identifies when they talk about going out, and offers making a reservation, or calling a taxi.

1.2 How to use the Conversation service

These are the steps for using the Conversation service:

1. Create a workspace in a Watson Conversation service instance.
2. Train the Conversation service instance to recognize concepts in the user input (intents and entities):
   - Train the Conversation service instance with natural language examples of each possible *intent*. At least five examples are required for minimal training. Providing many examples will give more accurate results, especially if they are varied and representative of possible input from users.
   - Train the Conversation service instance with natural language examples of each possible *entity*. Add as many synonyms as you expect your user to possibly use. The *Improve* interface will allow you to refine this process later on, adding more synonyms as you test your dialog.
3. Create a workflow of the stages of the dialog. Use logical conditions evaluating the concepts identified in the user’s reply.
4. Test your dialog in the embedded chat in the Conversation workspace. You can monitor how the Watson Conversation service interprets the flow, what intents and entities it detects, and improve its training data in real time.
5. Call your workflow from your application using the REST API.

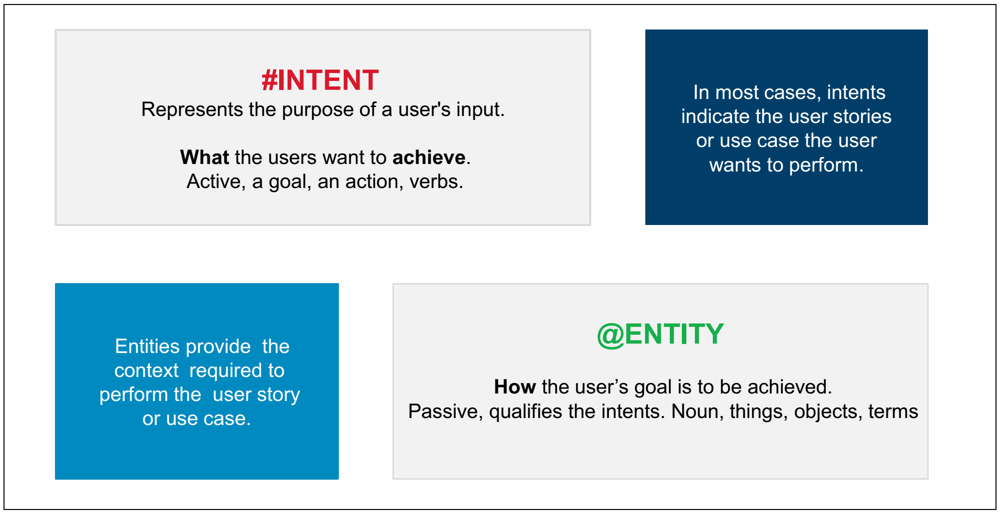
1.3 Conversation concepts

This section describes the main concepts you need to understand about Watson Conversation service.

1.3.1 Intents and entities

Watson Conversation service uses a natural language processing (NLP) to identify key information from user's interactions. The information that the Conversation service extracts falls into two categories, as explained in Figure 1-2:

- Intent: The purpose of a user's input (the user's intent).
- Entity: A term or object that is relevant to the intent (context for the intent).



#INTENT
Represents the purpose of a user's input.

What the users want to **achieve**.
Active, a goal, an action, verbs.

In most cases, intents indicate the user stories or use case the user wants to perform.

Entities provide the context required to perform the user story or use case.

@ENTITY

How the user's goal is to be achieved.
Passive, qualifies the intents. Noun, things, objects, terms

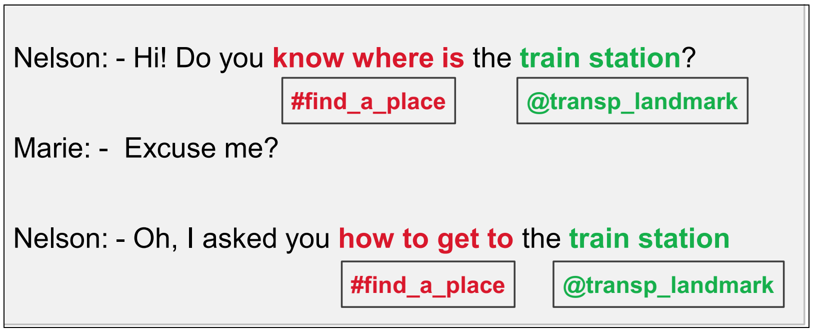
Figure 1-2 Intent and entity definitions

The *dialog* component of the Conversation service uses the intents and entities that are identified in the user's input to gather required information and provide a useful response to each user input. The dialog is the logical flow that determines the responses your bot will give when certain intents and/or entities are detected.

The dialog can be considered a user interface to extract the intents and entities from the users, process them to create a helpful response, and return the results in the form of natural language.

1.3.2 An example of intents and entities in a conversation

You can try to extract intents and entities from a conversation between two people (Figure 1-3).



Nelson: - Hi! Do you know where is the train station?

Marie: - Excuse me?

Nelson: - Oh, I asked you how to get to the train station

Figure 1-3 Example of intents and entities in a conversation

If you want to create a conversational application that is able to help Nelson in the same way that Marie can, you must train it to identify the intent `#find_a_place` and the entity `@transp_landmark`, and its possible values. Then, you can trigger a mapping API to direct Nelson to his destination.

1.3.3 Dialog

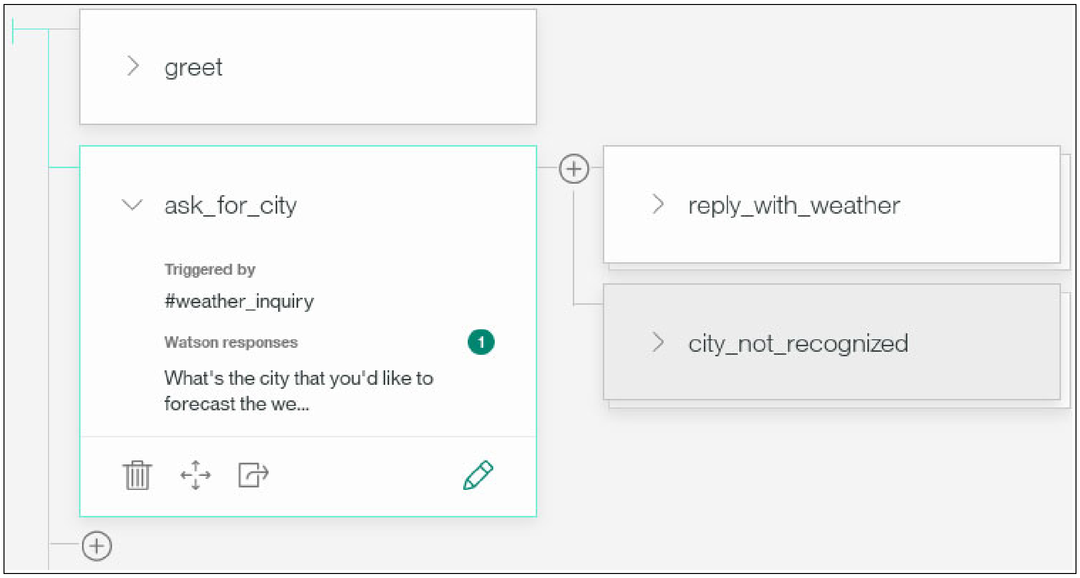
Your users will unlikely provide all of the required information in one pass. Instead, you must organize a conversation flow. The flow will ask users the questions that are useful in order to gather all the necessary input to provide a helpful answer.

A *dialog* is a branching conversation flow that defines how your application responds when it recognizes the defined intents and entities. It is composed of many branching dialog nodes. Create a dialog branch for each intent, to gather any required information and make a helpful response.

Figure 1-4 on page 6 shows the dialog for a weather Conversation flow, which is composed of the following dialog nodes:

- A greeting node
- A node to ask the user the city of interest
- A reply after the city is identified
- A backup reply in case the program cannot identify the city

More details about how to build intents, entities, and the dialog for weather Conversation are in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157.



greet

ask_for_city

Triggered by

#weather_inquiry

Watson responses

What's the city that you'd like to forecast the we...

reply_with_weather

city_not_recognized

Figure 1-4 Example of dialog flow

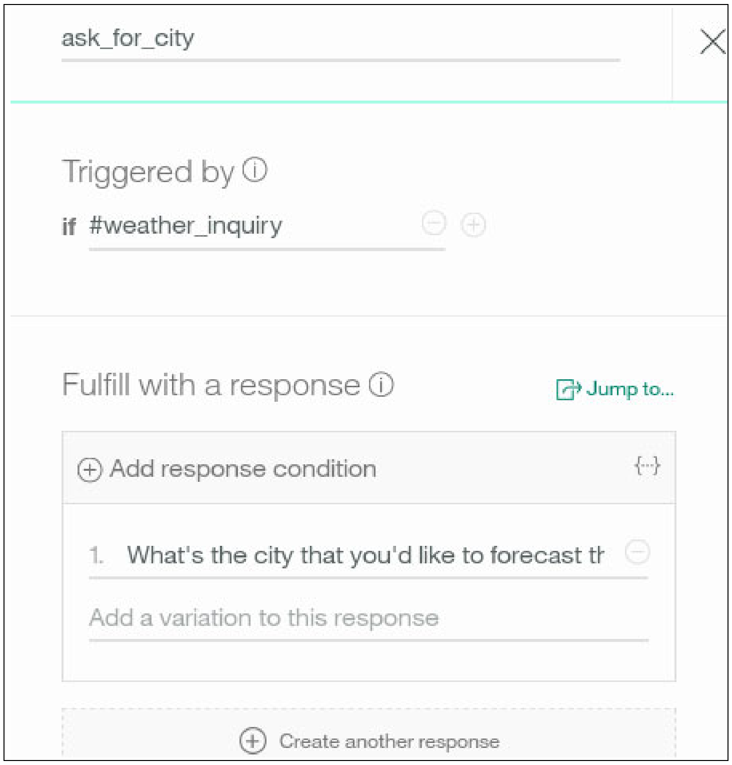
1.3.4 Dialog node

The dialog is made up of nodes that define steps in the conversation. Dialog nodes are chained together in a tree structure to create an interactive conversation with the user.

Each node starts with one or more lines that the bot shows to the user to request a response. Each node includes conditions for the node to be active, and also an output object that defines the response provided. You can think of the node as an if-then construction: *if* this condition is true, *then* return this response. The simplest condition is a single intent, which means that the response is returned if the user's input maps to that intent

Dialog nodes that originate on another node are their *children nodes*. Dialog nodes that do not depend on other nodes are *base nodes*.

Figure 1-5 shows a sample dialog node, with a labeling name, a condition, and an example response.



ask_for_city

Triggered by ⓘ

if #weather_inquiry

Fulfill with a response ⓘ

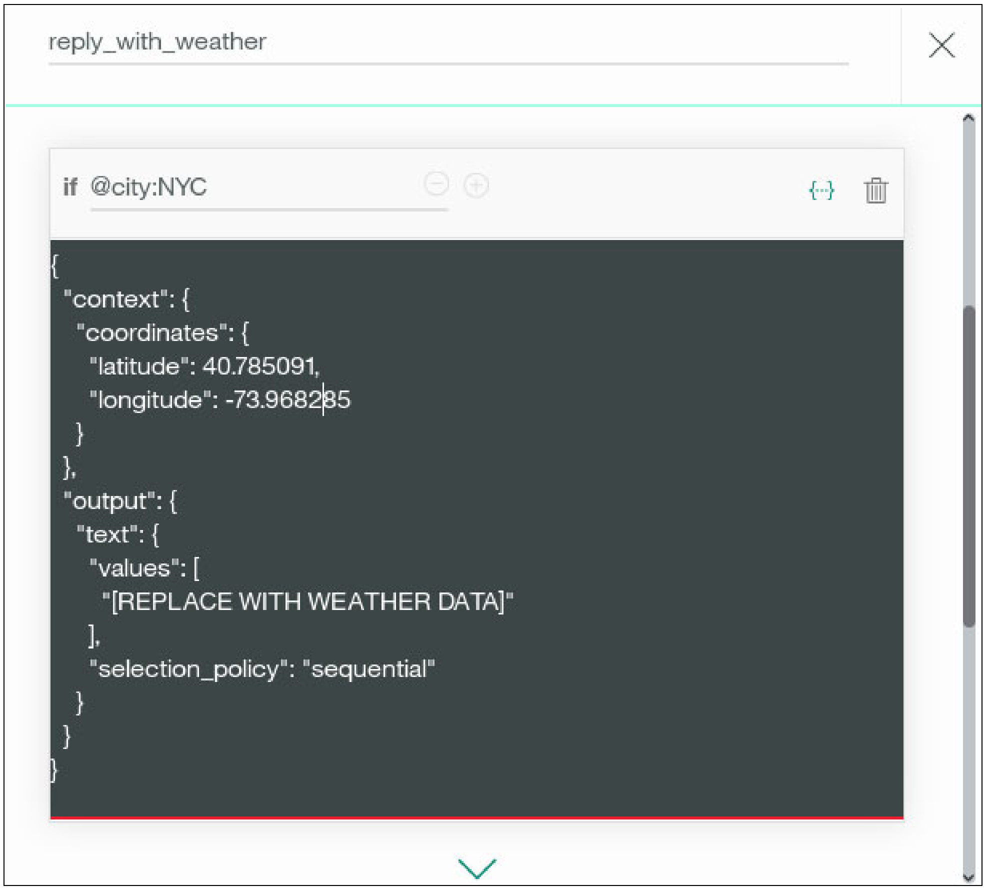
1. What's the city that you'd like to forecast tł

Figure 1-5 Example dialog node

1.3.5 Context

As in a real life conversation, context matters. The dialog context is the mechanism for passing information between the dialog and your application code. Context allows you to store information to continue passing it back and forth across different dialog nodes. For example, if you identify the names of your users in the Conversation flow, you could store the information in the context and retrieve it any time you want to call your user by name. Context is described as a JSON entry within the node, or can be modified in your app before the REST call.

Figure 1-6 shows a sample that sets NYC coordinates in the context, for use later.



```
{
 "context": {
  "coordinates": {
   "latitude": 40.785091,
   "longitude": -73.968285
  }
 },
 "output": {
  "text": {
   "values": [
    "[REPLACE WITH WEATHER DATA]"
   ],
   "selection_policy": "sequential"
  }
 }
}
```

Figure 1-6 Example context, setting the NYC coordinates in the context for future use

The dialog is *stateless*, meaning that it does not retain information from one interchange to the next. Your application is responsible for maintaining any continuing information. However, the application can pass information to the dialog, and the dialog can update the context information and pass it back to the application.

In the context, you can define any supported JSON types, such as simple string variables, numbers, JSON arrays, or JSON objects.

1.3.6 Condition and responses

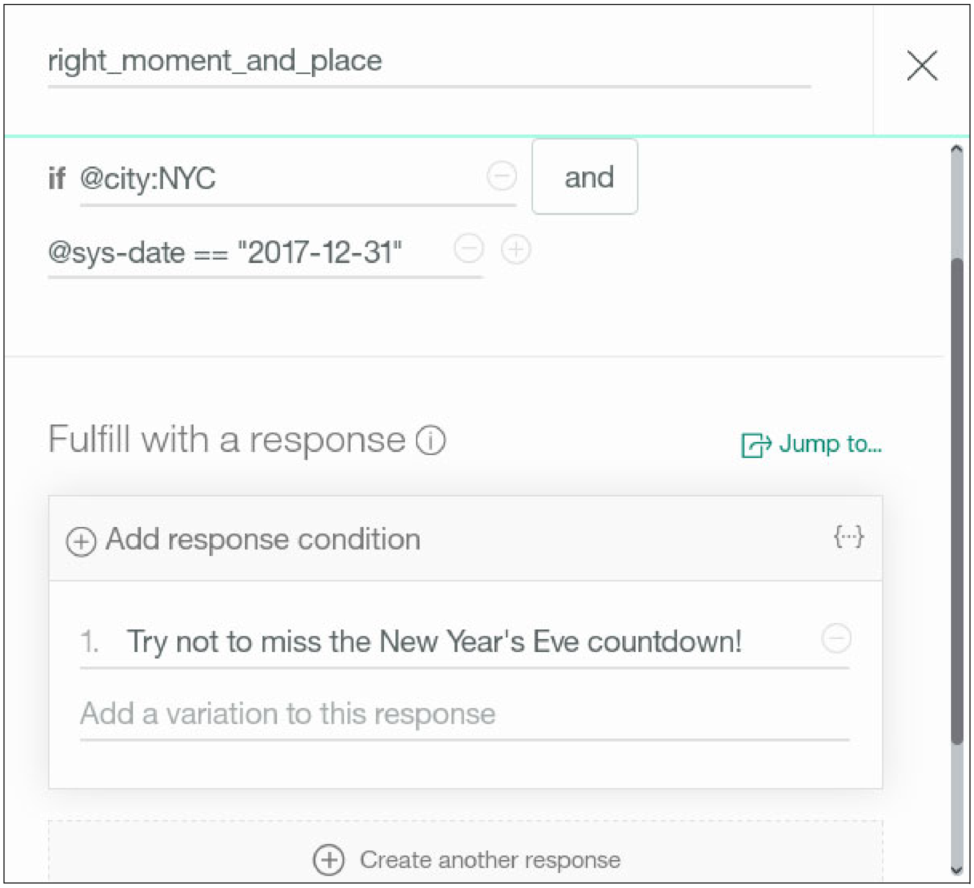
The condition portion of a dialog node determines whether that node is used in the conversation. Conditions are logical expressions that are evaluated to *true* or *false*. Conditions are used to select the next dialog node in the flow, or to choose among the possible responses to the user.

Conditions are expressed in the Spring Expression Language (SpEL).

Conditions usually evaluate the intents and entities identified in the user responses but also can evaluate information stored in the context. This information in the context can be stored in previous dialog nodes or in your application code as part of an API call.

Figure 1-7 shows a sample dialog node conditioned on a specific location (NYC) and time (31-Dec-2017) so you can recommend visiting Times Square for New Year's Eve.

Responses are messages based on the identified intents and entities that are communicated to the user when the dialog node is activated. You can add variations of the response for a more natural experience, or add conditions to pick one response out of many in the same dialog node.



right_moment_and_place

if @city:NYC and

@sys-date == "2017-12-31"

Fulfill with a response

1. Try not to miss the New Year's Eve countdown!

Figure 1-7 Special condition (place and time) to celebrate New Year's Eve in Times Square

1.3.7 Conversation turn

A single cycle of user input and a response is called *conversation turn* (Figure 1-8). Each conversation turn starts in one dialog node, called the *active node*.

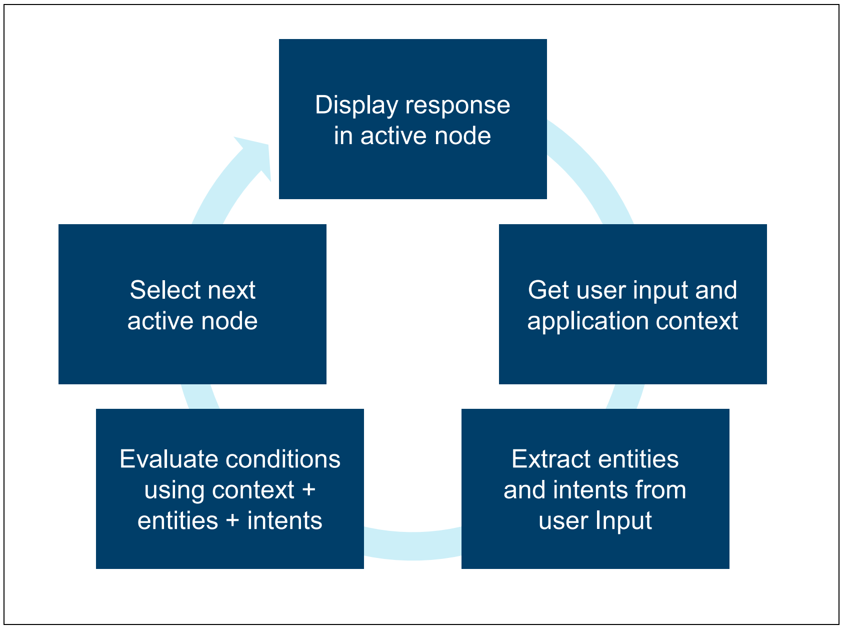

Display response in active node
Get user input and application context
Extract entities and intents from user Input
Evaluate conditions using context + entities + intents
Select next active node

Figure 1-8 Conversation turn

1.3.8 Typical conversation flow

Figure 1-9 on page 11 shows a typical conversation flow and how the nodes are selected:

1. The conversation starts in an initial node set up with the `conversation_start` special condition.
2. After some conversation turns, the dialog progresses to the node marked as *active node*. The response configured in this node is shown to the user. The user input is analyzed for intents and entities and used to select the next dialog node in the flow.
3. The conditions in the child nodes are evaluated in descending order using the extracted intents and entities. The first child node to match a condition is selected as the next active node and a new conversation turn starts (not shown in the figure).
4. If no child node matches the condition, the Conversation service evaluates the conditions of each base node in the dialog and selects the first matching dialog node as the next active node.
5. A useful approach is to have a base node configured with the `anything_else` special condition so that the conversation defaults to this node when no other nodes match the conditions. The special `anything_else` condition always evaluates to *true*. You can use this node in the dialog to tell the user that the input was not understood and suggest valid interaction.

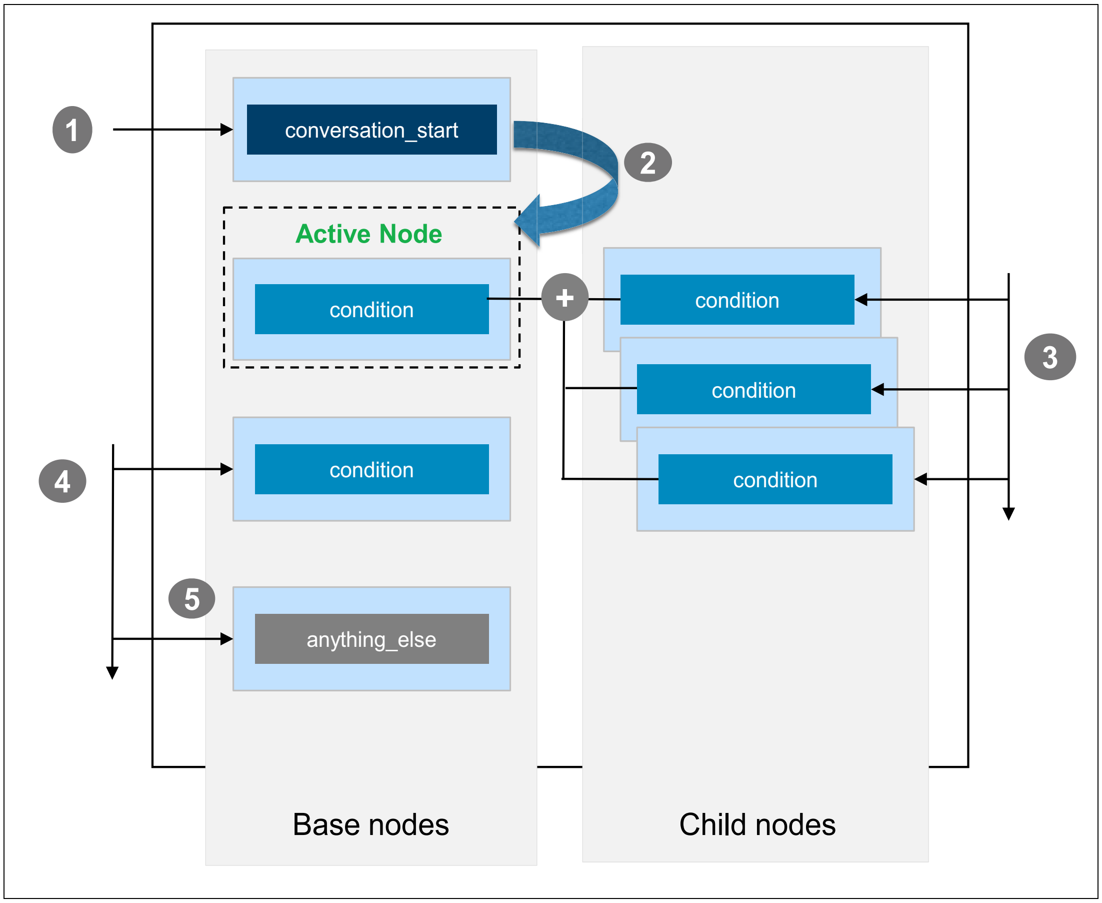


Figure 1-9 Next active node selection criteria

1.4 Conclusion

In this chapter, you learned the basic concepts that apply to the Watson Conversation service. In the next chapters, you learn to combine the concepts introduced in this chapter to create meaningful conversations with your users.

The Conversation service will extract intents and entities from user input. It will use this information and context information to traverse a flow of dialog nodes, called a dialog. Each node will be selected based on its configured conditions, and will have a response to present to the user.

These simple basic concepts allow you to create a complex, powerful, and practical user interaction experience.

1.5 References

For more information, see the following resources:

- Overview of the Watson Conversation service:

  https://www.ibm.com/watson/developercloud/doc/conversation/index.html

- How Watson Conversation Service Works (video):

  https://youtu.be/CV8nNIIQh1c

- Building chatbots with Watson (video):

  https://www.youtube.com/watch?v=ccLKDBg8Ht8

2

Conversation service workspace

The natural language processing for the Watson Conversation service happens in a *workspace*, which is a container for all of the artifacts that define the conversation flow for an application.

This chapter explains how to create and use a Conversation workspace with the Conversation tool. This chapter shows, by example, how to add intents and entities to the workspace and how to build a dialog.

The information in this chapter is a prerequisite for the other chapters in this book.

The following topics are covered in this chapter:

- How to use the Conversation service
- Exporting the workspace
- References

2.1 How to use the Conversation service

Using the Conversation service involves the following steps:

1. Creating a Watson Conversation service instance
2. Launching the Conversation tool
3. Working with a workspace
4. Adding intents
5. Adding entities
6. Building a dialog

In the following sections, you import the Weather Forecast workspace to your Conversation service instance. You add new intents and entities to it to become a complete car chatbot, which gives weather information and can also provide traffic information.

Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Create a Conversation service instances in Bluemix.
- Use the Conversation tool.
- Create and import a workspace.
- Create intents.
- Create entities.
- Build dialogs.

2.1.1 Creating a Watson Conversation service instance

Bluemix provides resources to your applications through a service instance. Before you can use the Watson APIs you must create an instance of the corresponding service. You will need to create a Watson Conversation service instance for use in all the examples in this book.

To create an instance of the Conversation service, follow these steps:

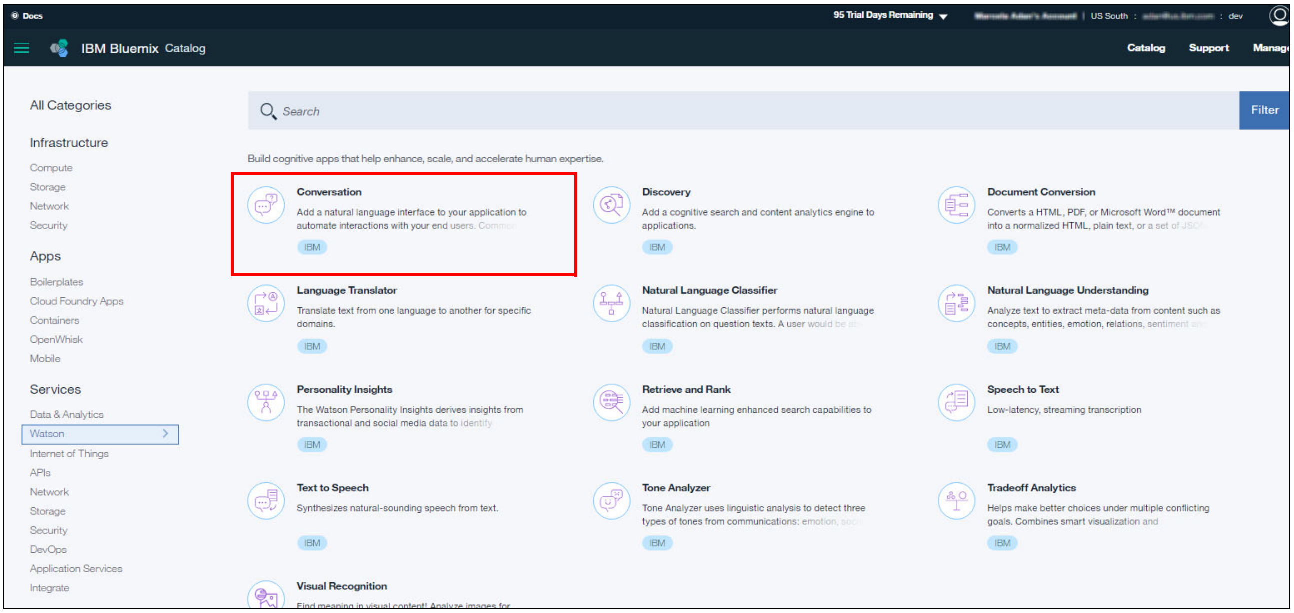
1. Create an IBM Bluemix account if you do not have one.

   You must have a Bluemix account to access the Watson APIs. You can create a free trial Bluemix account.

2. Log in to IBM Bluemix.
3. Click **Watson** (under Services).

   The Watson services that are available in Bluemix are listed.

4. Click **Conversation** (Figure 2-1 on page 15).



Build cognitive apps that help enhance, scale, and accelerate human expertise.

Conversation
Add a natural language interface to your application to automate interactions with your end users. Commo
IBM

Discovery
Add a cognitive search and content analytics engine to applications.
IBM

Document Conversion
Converts a HTML, PDF, or Microsoft Word™ document into a normalized HTML, plain text, or a set of JSO
IBM

Language Translator
Translate text from one language to another for specific domains.
IBM

Natural Language Classifier
Natural Language Classifier performs natural language classification on question texts. A user would be a
IBM

Natural Language Understanding
Analyze text to extract meta-data from content such as concepts, entities, emotion, relations, sentiment an
IBM

Personality Insights
The Watson Personality Insights derives insights from transactional and social media data to identify
IBM

Retrieve and Rank
Add machine learning enhanced search capabilities to your application
IBM

Speech to Text
Low-latency, streaming transcription
IBM

Text to Speech
Synthesizes natural-sounding speech from text.
IBM

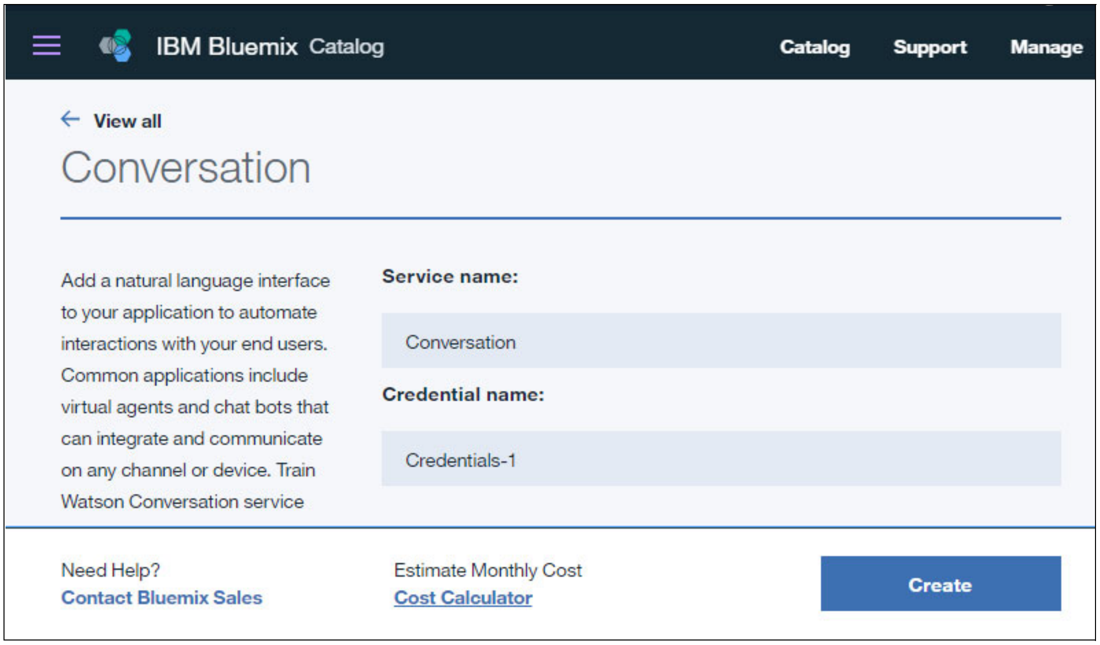
Tone Analyzer
Tone Analyzer uses linguistic analysis to detect three types of tones from communications: emotion, soc
IBM

Tradeoff Analytics
Helps make better choices under multiple conflicting goals. Combines smart visualization and
IBM

Visual Recognition
Find meaning in visual content! Analyze images for

Figure 2-1 Watson services in Bluemix: Select Conversation

5. Do these steps on the next web page (Figure 2-2):
   a. Enter `Conversation` as the service instance name.
   b. Notice the credential name, `Credentials-1`.
   c. Select the pricing plan you want to use.
   d. Click **Create** and wait for Bluemix to create an instance of your Conversation service.



Conversation

Add a natural language interface to your application to automate interactions with your end users. Common applications include virtual agents and chat bots that can integrate and communicate on any channel or device. Train Watson Conversation service

Service name: Conversation

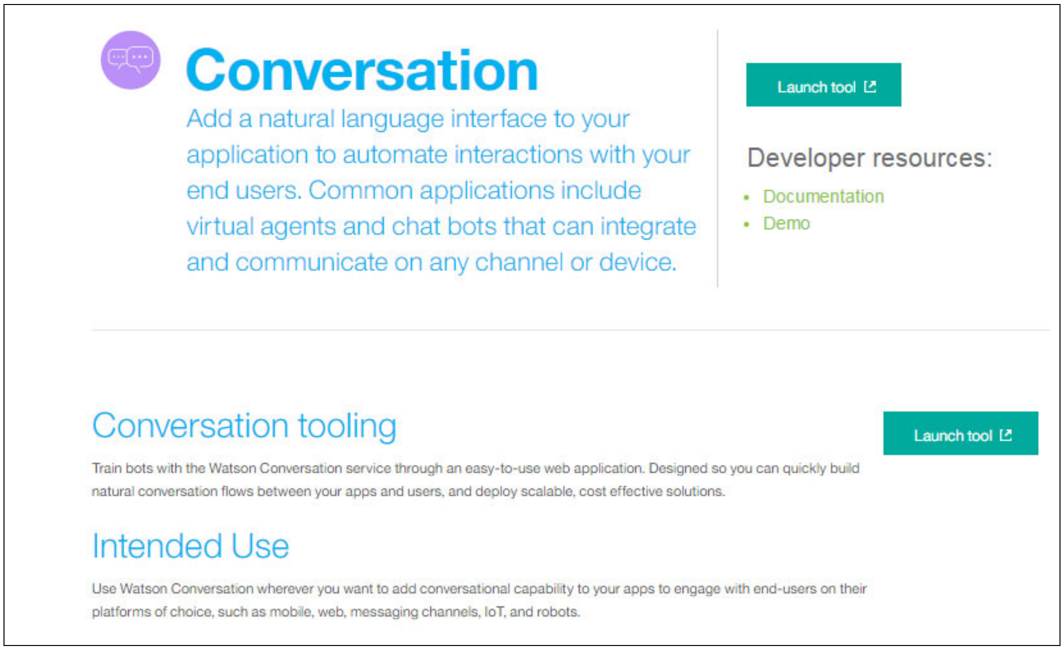
Credential name: Credentials-1

Figure 2-2 Conversation service instance name

2.1.2 Launching the Conversation tool

The Conversation tool is a visual dialog builder to help you create natural conversations between your apps and users, without any coding experience required. Complete these steps to launch the tooling:

1. After creating the Conversation service instance, click **Launch tool** (Figure 2-3).



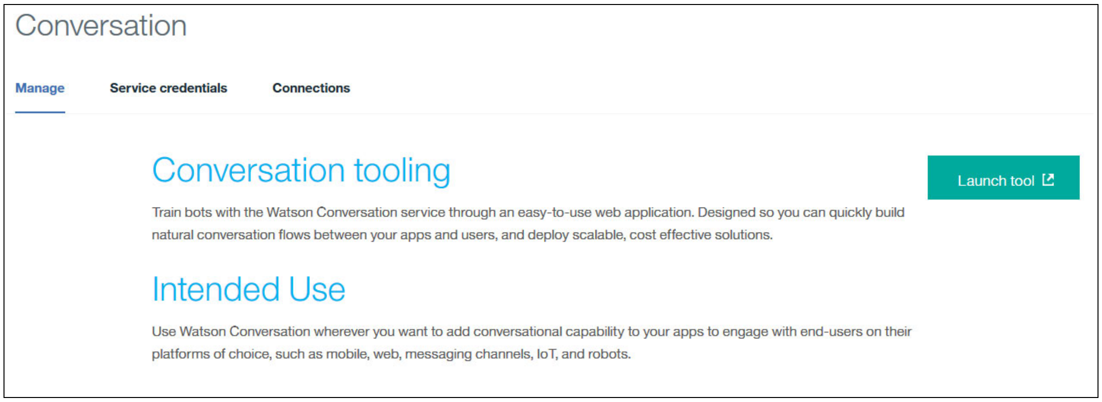
Add a natural language interface to your application to automate interactions with your end users. Common applications include virtual agents and chat bots that can integrate and communicate on any channel or device.

Train bots with the Watson Conversation service through an easy-to-use web application. Designed so you can quickly build natural conversation flows between your apps and users, and deploy scalable, cost effective solutions.

Use Watson Conversation wherever you want to add conversational capability to your apps to engage with end-users on their platforms of choice, such as mobile, web, messaging channels, IoT, and robots.

Figure 2-3 Launching the conversation tool immediately after creating the service instance

2. Alternatively, you can launch the tool at a later time:
   a. Go to the Bluemix dashboard.
   b. Click your Conversation service instance.
   c. On the service details page, click the **Manage** tab (Figure 2-4), scroll to Conversation tooling, and click **Launch tool**.



Train bots with the Watson Conversation service through an easy-to-use web application. Designed so you can quickly build natural conversation flows between your apps and users, and deploy scalable, cost effective solutions.

Use Watson Conversation wherever you want to add conversational capability to your apps to engage with end-users on their platforms of choice, such as mobile, web, messaging channels, IoT, and robots.

Figure 2-4 Launch Conversation tooling

3. If this is the first workspace, the Watson Conversation login page opens (Figure 2-5). If you have an IBMid, click **Log in with IBM ID**; otherwise, click **Sign up for IBM ID**.

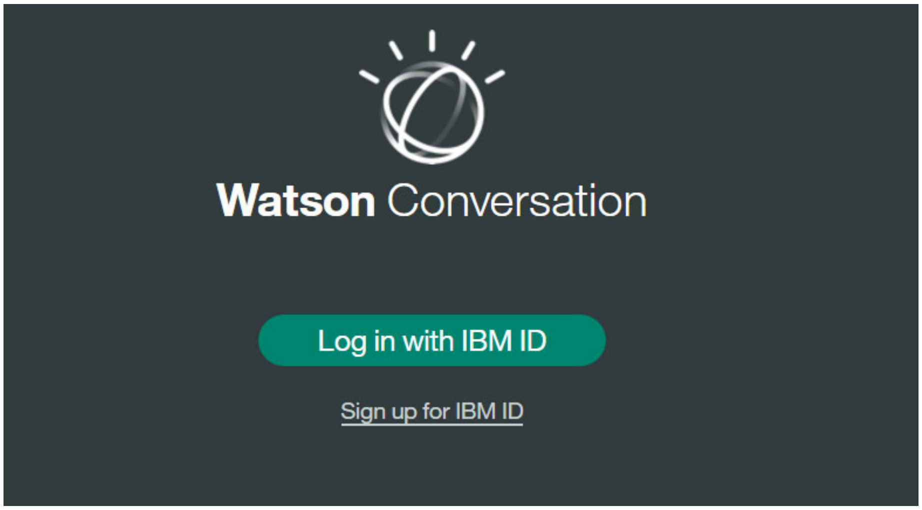


Figure 2-5 Log in Watson Conversation tooling

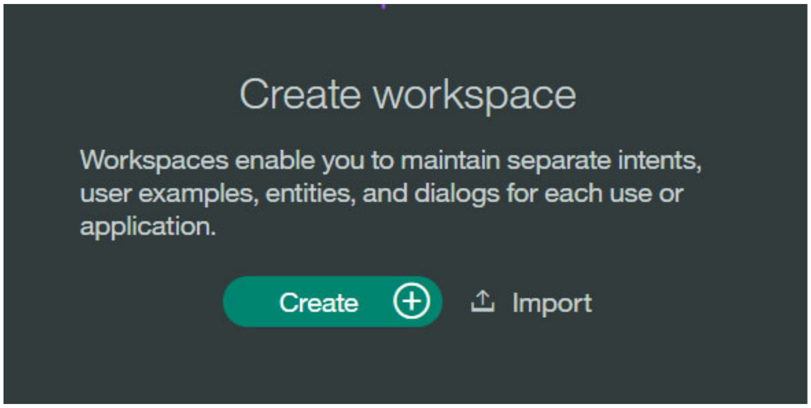
2.1.3 Working with a workspace

This section describes how to create, delete, import, and rename a workspace.

Create a new workspace

Complete the following steps:

1. Launch Conversation tooling.
2. Click **Create** to create a workspace (Figure 2-6).

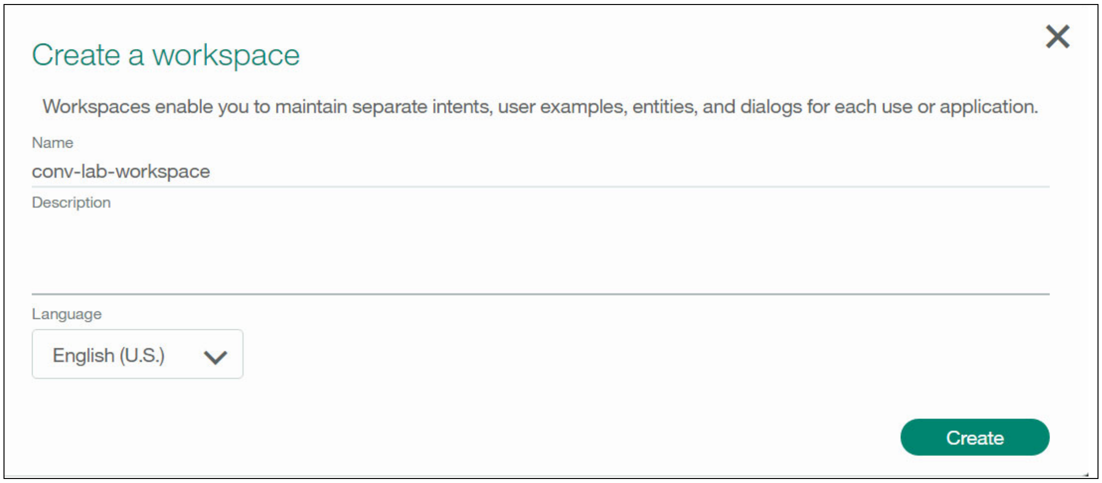


Create workspace

Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

Figure 2-6 Create new workspace

3. As shown in Figure 2-7, specify the details of the new workspace:
   - Name: `conv-lab-workspace`
   - Description: Any description not more than 128 characters.
   - Language: Language of user input that the workspace will be trained to understand; Keep as default: English (U.S.).



Create a workspace

Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

Name: conv-lab-workspace

Description:

Language: English (U.S.)

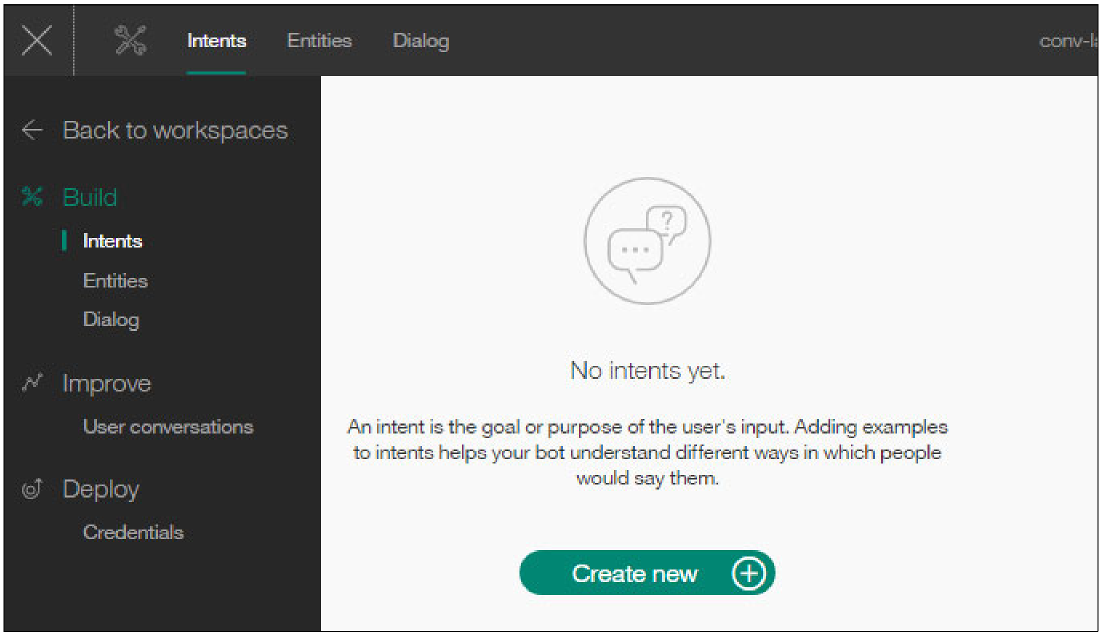
Figure 2-7 New workspace details

4. Click **Create**.

Delete a workspace

Complete the following steps:

1. Click the menu icon ☰ and then click **Back to workspaces** (Figure 2-8).



No intents yet.

An intent is the goal or purpose of the user's input. Adding examples to intents helps your bot understand different ways in which people would say them.

Figure 2-8 Conversation workspace

2. Click the three vertical dots, then click **Delete** (Figure 2-9).

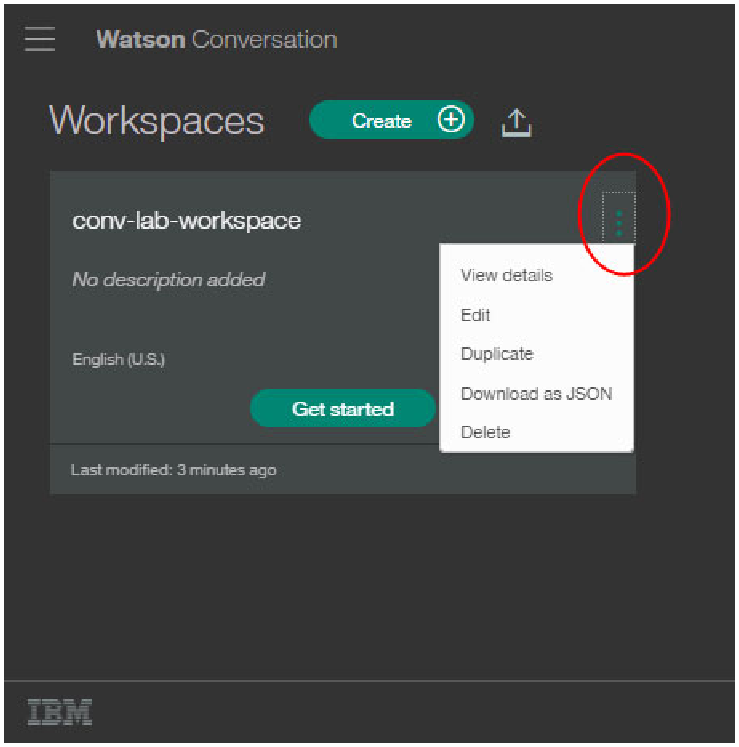
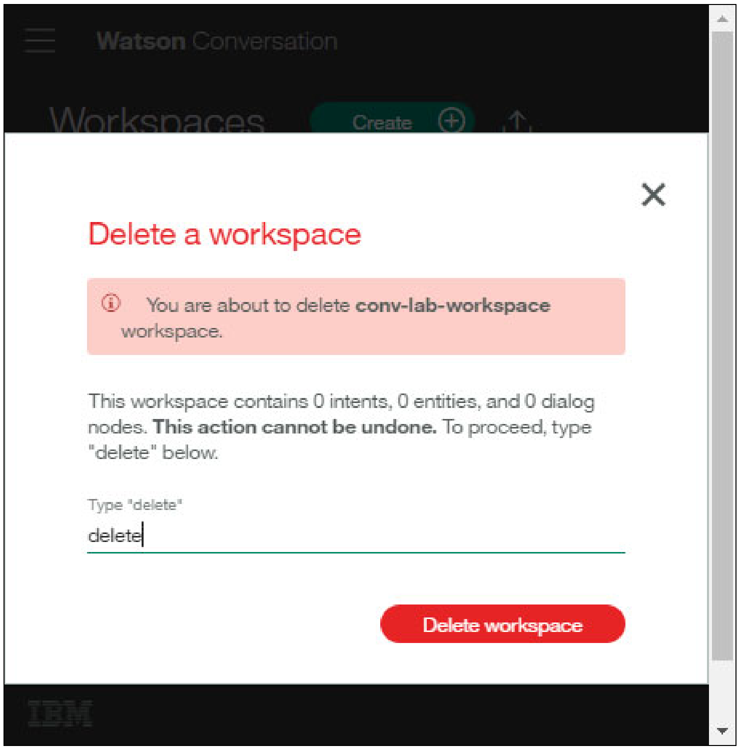


Figure 2-9 Delete workspace

3. Type the word `delete` in the “Delete a workspace” confirmation dialog and then click **Delete workspace** (Figure 2-10).



Delete a workspace

You are about to delete **conv-lab-workspace** workspace.

This workspace contains 0 intents, 0 entities, and 0 dialog nodes. **This action cannot be undone.** To proceed, type "delete" below.

Type "delete"

delete

Figure 2-10 Delete workspace confirmation dialog

Import a workspace

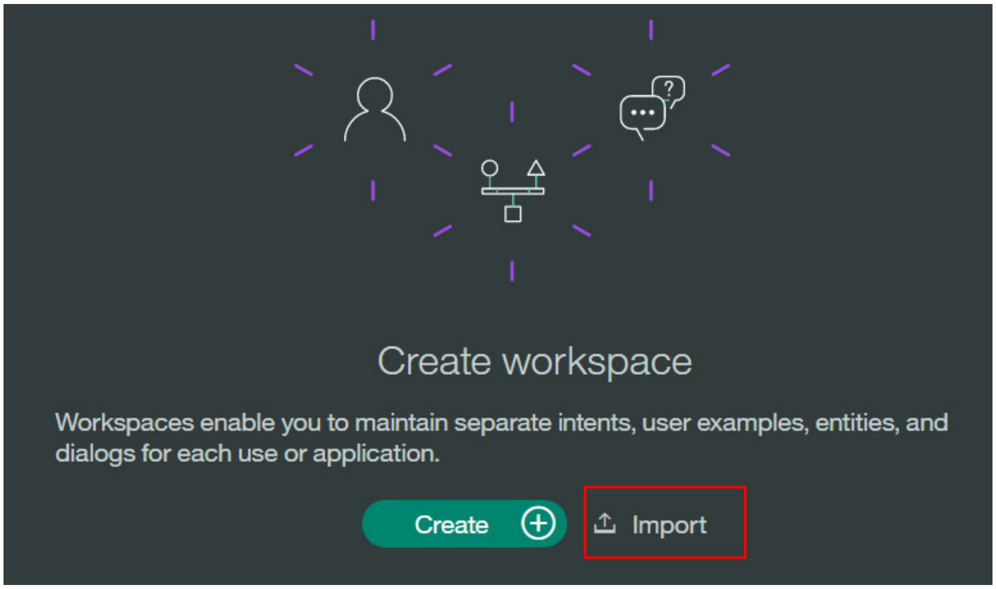
Complete the following steps:

1. Download the Weather Forecast workspace JSON file:

   https://github.com/snippet-java/redbooks-conv-201-weather-nodejs/blob/master/training/1.4-conv-101-createservice-incomplete.json

2. Launch the Conversation tooling by doing one of the following steps:

   – If this is your first workspace, click **Import**. Figure 2-11 shows an empty service with no workspaces created.



Create workspace

Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

Figure 2-11 First time Import workspace

   – If this is *not* your first workspace, and workspaces are already associated with the Conversation instance, click the **Import workspace** button at the top of the page (Figure 2-12).

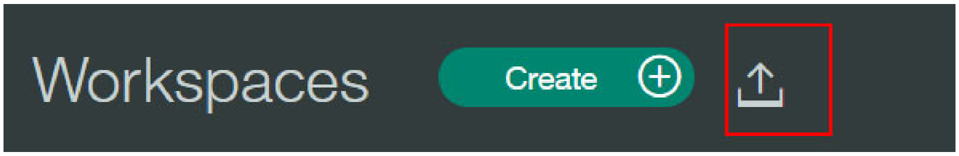
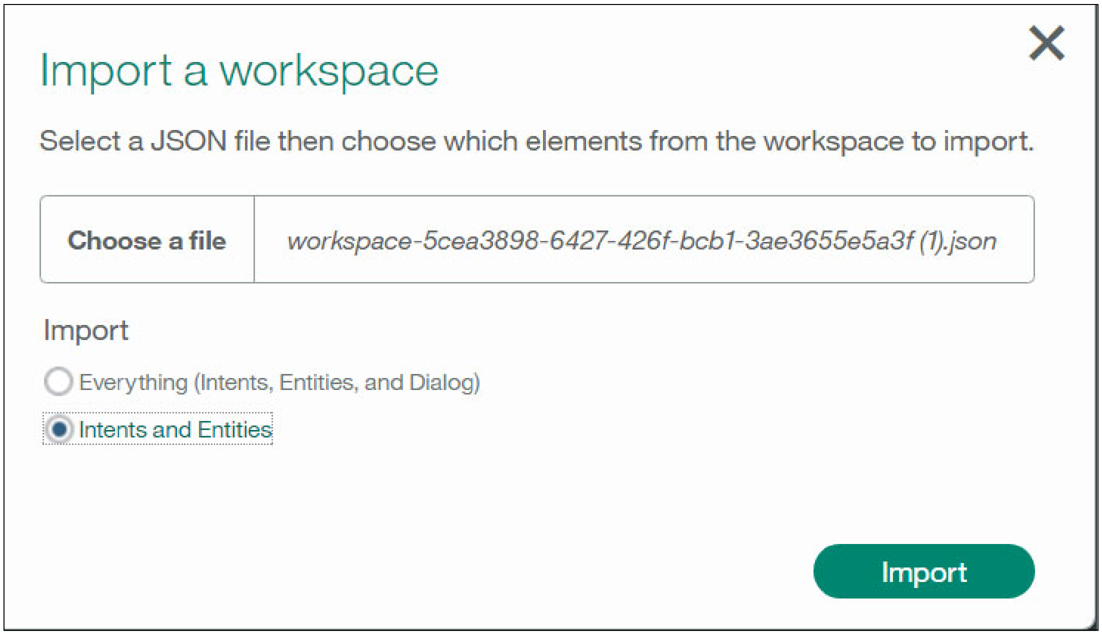


Figure 2-12 Import workspace

3. In the “Import a workspace” dialog (Figure 2-13 on page 21), use these steps:

   a. Click **Choose a file** and select the downloaded JSON file.

   b. Select **Intents and Entities** to use the intents and entities from the exported workspace; you will build a new dialog. Figure 2-13 on page 21 shows how to import intents and entities from the workspace JSON file.



Import a workspace

Select a JSON file then choose which elements from the workspace to import.

workspace-5cea3898-6427-426f-bcb1-3ae3655e5a3f (1).json

Import

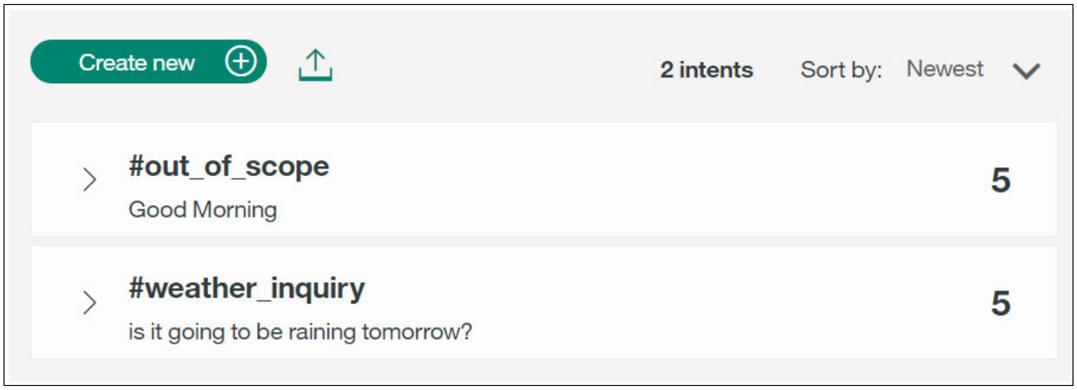
○ Everything (Intents, Entities, and Dialog)

● Intents and Entities

Figure 2-13 Choose JSON file to import

4. Click **Import** to import the intents and entities.

   Figure 2-14 shows the imported intents.



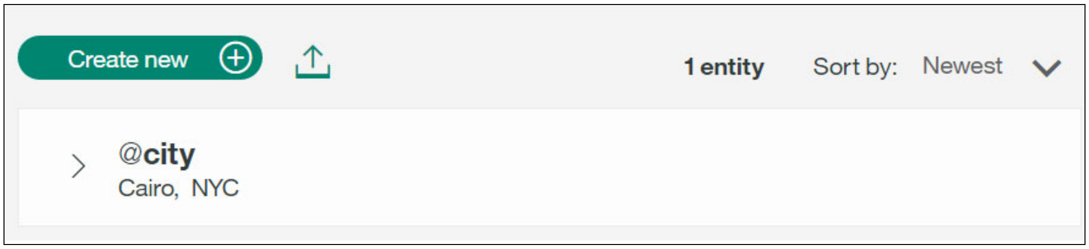
2 intents Sort by: Newest

#out_of_scope 5
Good Morning

#weather_inquiry 5
is it going to be raining tomorrow?

Figure 2-14 Weather Forecast intents imported

Figure 2-15 shows the imported entities.



1 entity Sort by: Newest

@city
Cairo, NYC

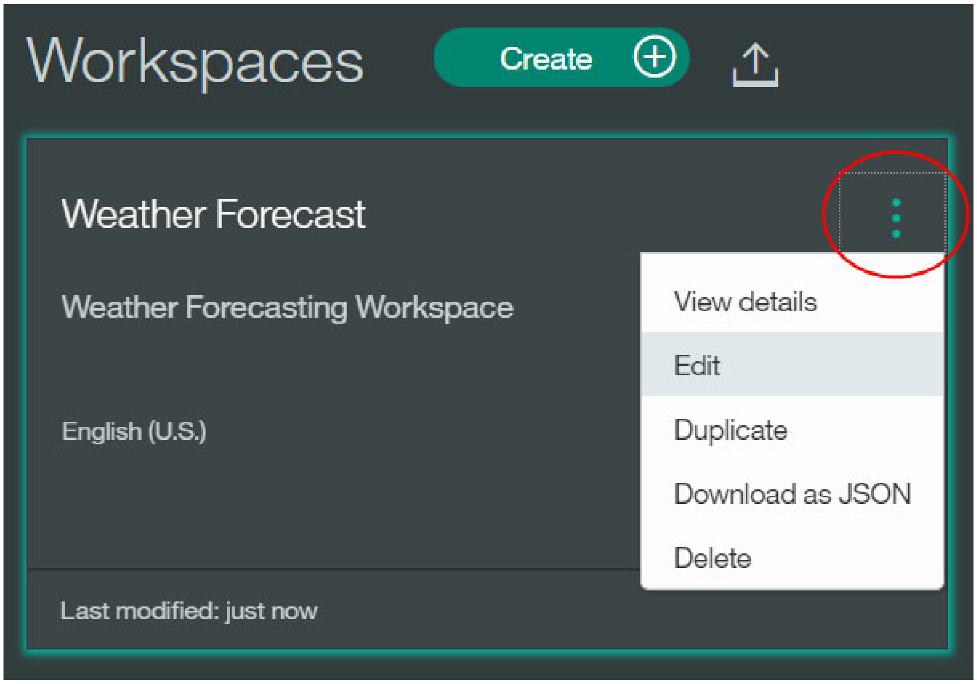
Figure 2-15 Weather Forecast entities imported

Rename the Weather Forecast workspace

After importing the Weather Forecast workspace, rename it to `Car Chat-bot` to add more car-related features to it.

Complete the following steps to rename the workspace:

1. Go back to Workspaces by clicking the menu button in the upper left corner.
2. Click the **Actions** icon (three vertical dots) and select **Edit** (Figure 2-16).



Weather Forecast

Weather Forecasting Workspace

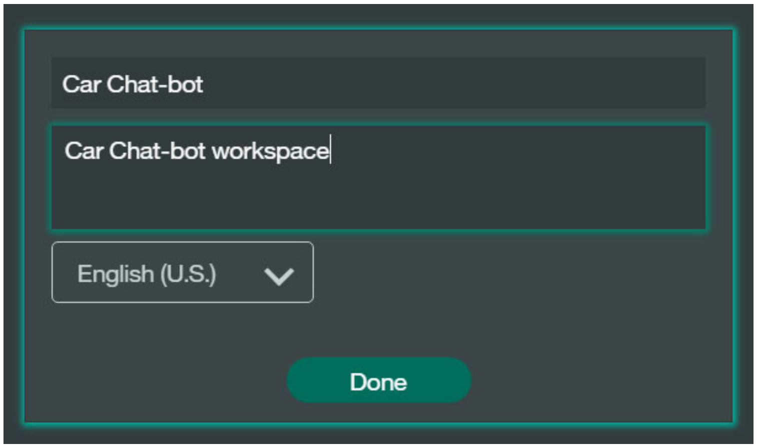
English (U.S.)

Last modified: just now

Figure 2-16 Edit the workspace

3. Change the name and description of the workspace (Figure 2-17):
   - Name: `Car Chat-bot`
   - Description: `Car Chat-bot workspace`

   Click **Done**.



Car Chat-bot

Car Chat-bot workspace

English (U.S.)

Figure 2-17 Rename workspace

2.1.4 Adding intents

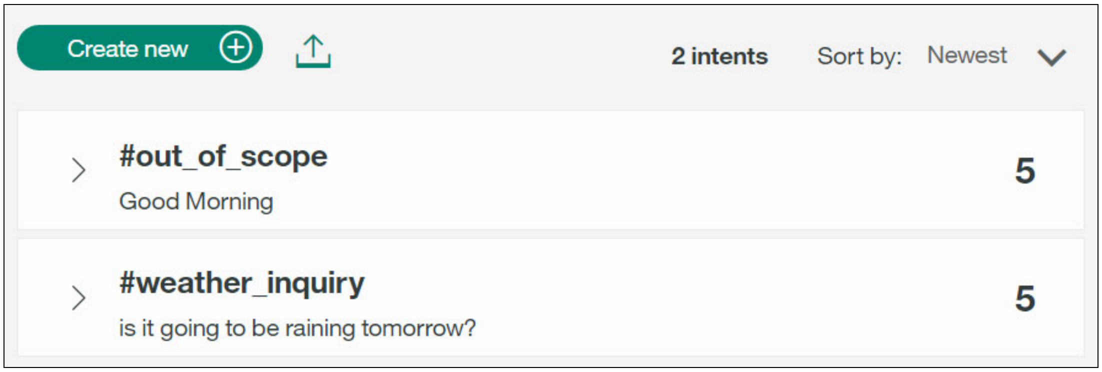
In this section, you add the following intents to the workspace. The workspace currently has the imported intents `weather_inquiry` and `out_of_scope`.

- Greeting
- Traffic
- Goodbye

Create a greeting intent

Use the Conversation tool to create a new intent:

1. Click the **Car Chat-bot workspace**. The Intents tab opens automatically.
2. Click **Create new** (Figure 2-18).



#out_of_scope Good Morning	5
#weather_inquiry is it going to be raining tomorrow?	5

Figure 2-18 Create new Intent

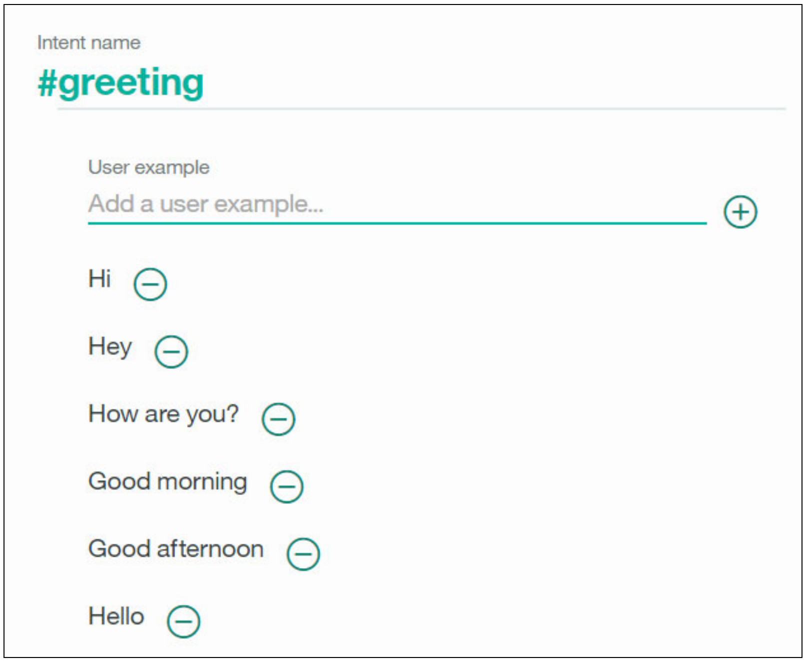
3. Name the intent `#greeting`.

> **Note:** The hashtag symbol (#) is added by default to the name; do not add it yourself.

4. In the User example section (Figure 2-19 on page 24), add these greeting examples to the `#greeting` intent; click the plus sign (**+**) or press Enter to add each user example:

   – `Hi`
   – `How are you?`
   – `Hello`
   – `Hey`
   – `Good morning`
   – `Good afternoon`

   Add as many greeting examples as you can, so that the application can be more accurate (five examples is the minimum).



Intent name

#greeting

User example

Add a user example...

Hi

Hey

How are you?

Good morning

Good afternoon

Hello

Figure 2-19 Add greeting intent and examples

5. When you finish adding user examples, click **Create** to save the intent.

   After you create the intent, the system starts to train itself with the new data.

Create a traffic intent

Use the Conversation tool to create a traffic intent:

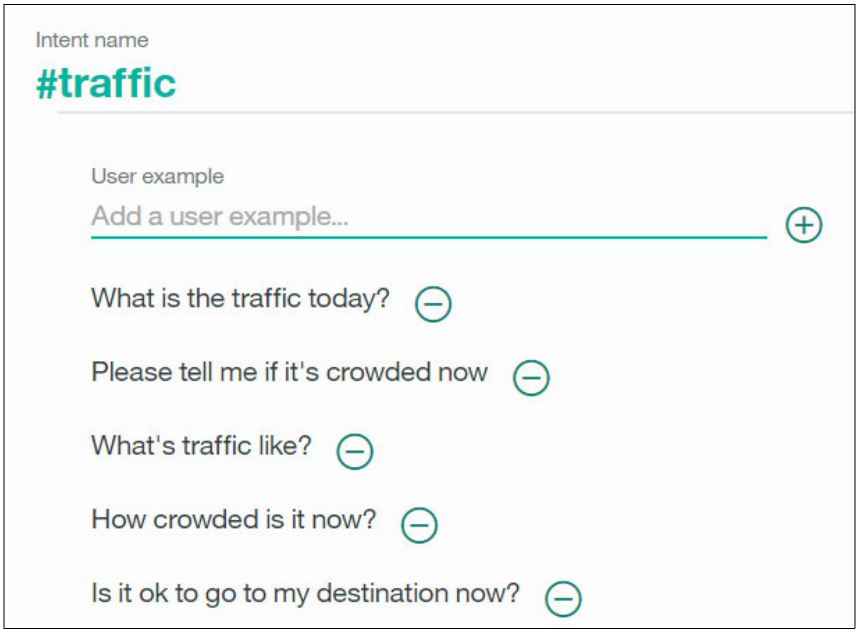
1. Click **Create new.** Name the intent: `#traffic`.

> **Note:** The hashtag symbol (#) is added by default to the name; do not add it yourself.

2. In the User example section (Figure 2-20 on page 25), add these traffic examples to the `#traffic` intent; click the plus sign (**+**) or press Enter to add each user example:

   – `What is the traffic today?`
   – `Please tell me if it's crowded now`
   – `What's the traffic like?`
   – `How crowded is it now?`
   – `Is it ok to go to my destination now?`

   Add as many traffic examples as you can, so that the application can be more accurate (five examples is the minimum).



Intent name

#traffic

User example

Add a user example...

What is the traffic today?

Please tell me if it's crowded now

What's traffic like?

How crowded is it now?

Is it ok to go to my destination now?

Figure 2-20   Add traffic intent and examples

3. When you finish adding user examples, click **Create** to save the intent.

   After you create the intent, the system starts to train itself with the new data.

Create a goodbye intent

Complete these steps:

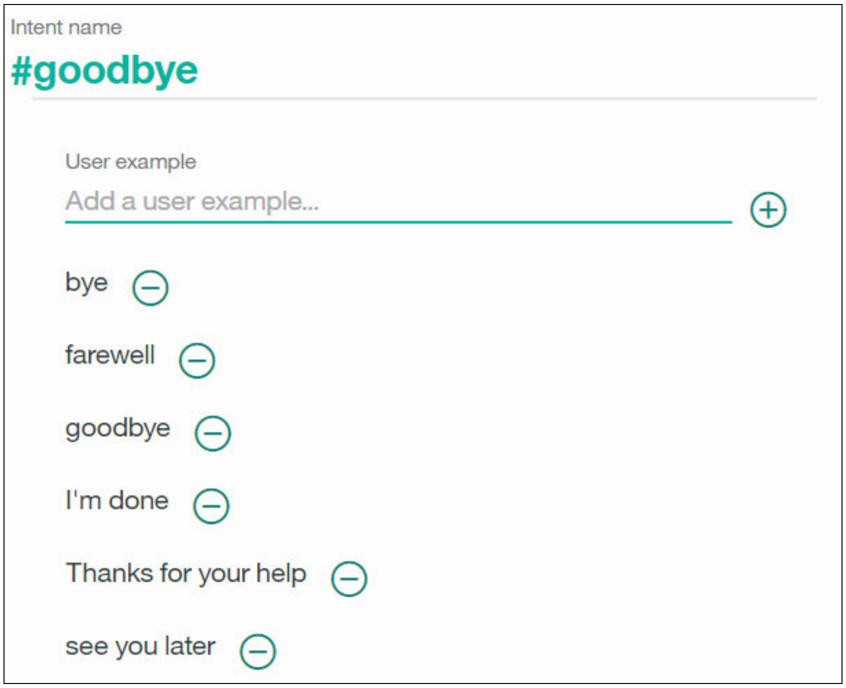
1. Click **Create new**. Name the intent: `#goodbye`.

   > **Note:** The hashtag symbol (#) is added by default to the name; do not add it yourself.

2. In the User example section (Figure 2-21 on page 26), add these goodbye examples to the `#goodbye` intent; click the plus sign (**+**) or press Enter to add each user example:

   – `bye`
   – `farewell`
   – `goodbye`
   – `I'm done`
   – `see you later`
   – `Thanks for your help`

   Add as many goodbye examples as you can, so that the application can be more accurate (five examples is the minimum).



Intent name

#goodbye

User example

Add a user example...

bye

farewell

goodbye

I'm done

Thanks for your help

see you later

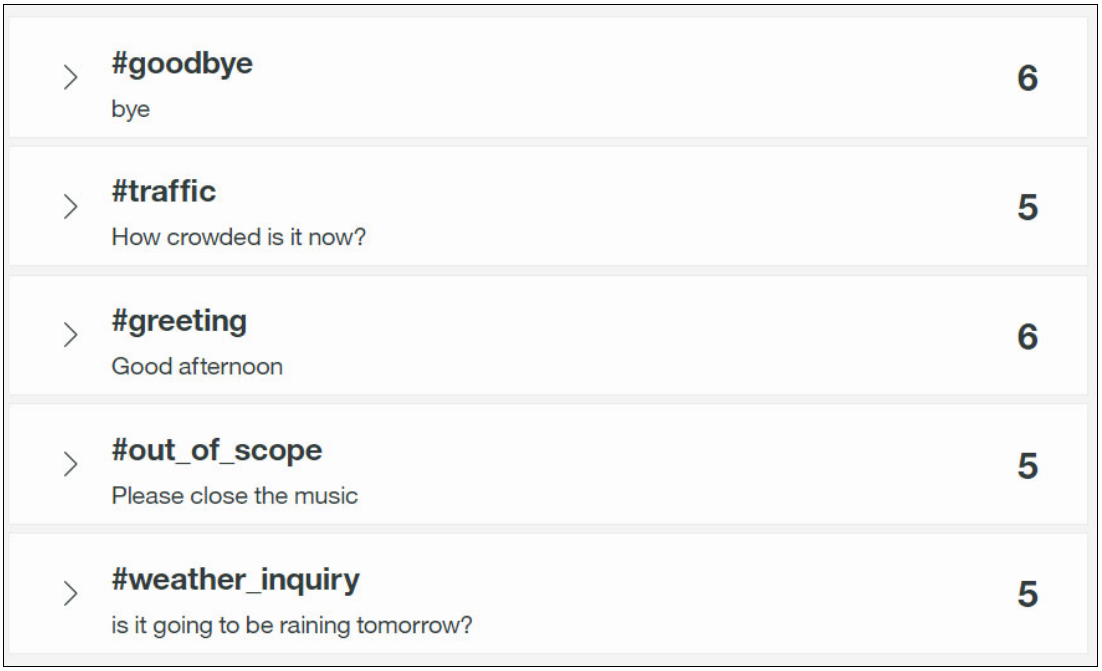
Figure 2-21   Add goodbye intent and examples

3. When you finish adding user examples, click **Create** to save the intent.

   After you create the intent, the system starts to train itself with the new data.

Final intents list in workspace

Figure 2-22 shows the final list of intents in the Car Chat-bot workspace.



#goodbye bye	6
#traffic How crowded is it now?	5
#greeting Good afternoon	6
#out_of_scope Please close the music	5
#weather_inquiry is it going to be raining tomorrow?	5

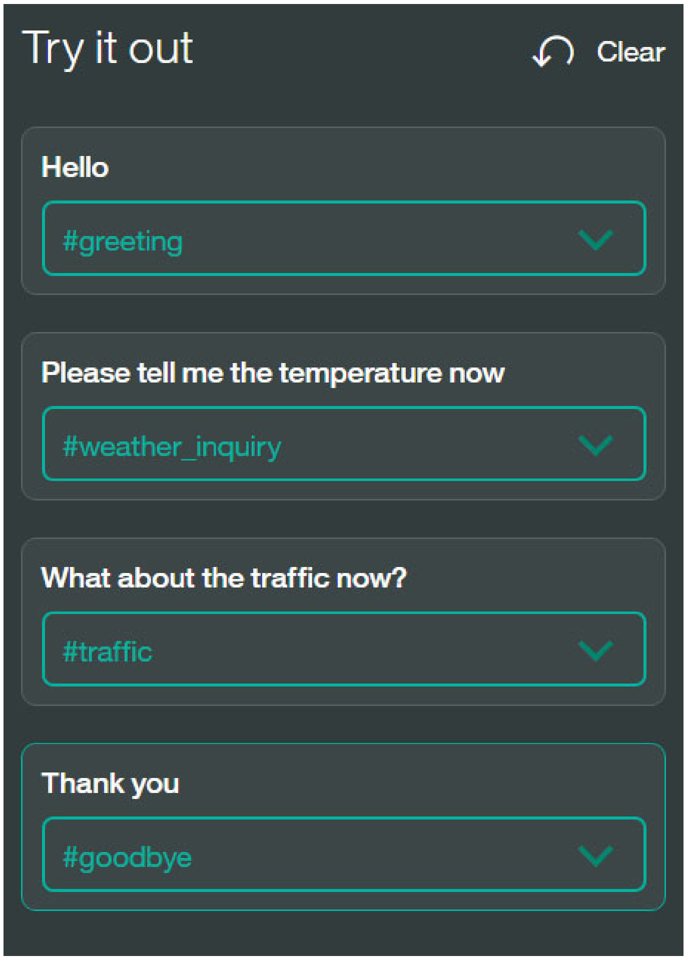
Figure 2-22   Car chatbot intents

Test your intent

After defining the new intents and examples. You can test your system to be sure it accurately recognizes the intents. If not, then the intents must be refined.

Complete these steps to test your system:

1. Click the ellipses button 💬 at the top right corner of the page.
2. Enter a question or a phrase to test whether the system recognizes the correct intent (Figure 2-23).



Try it out

Clear

Hello

#greeting

Please tell me the temperature now

#weather_inquiry

What about the traffic now?

#traffic

Thank you

#goodbye

Figure 2-23 Testing intents

3. If the system does not recognize the correct intent, you can correct it by clicking on the displayed intent and choosing the correct intent from the list. After selecting the intent, the system starts training itself with the new data.

2.1.5 Adding entities

An entity represents a class of object or a data type that is relevant to a user's purpose. By recognizing the entities that are mentioned in the user's input, the Conversation service can choose the specific actions to take to fulfill an intent.

The workspace has an imported `city` entity. In this section, you add a `destination` entity to the workspace.

Create destination entity

Use the Conversation tool to create a new entity:

1. Click the **Entities** tab.
2. Click **Create new** (Figure 2-24).

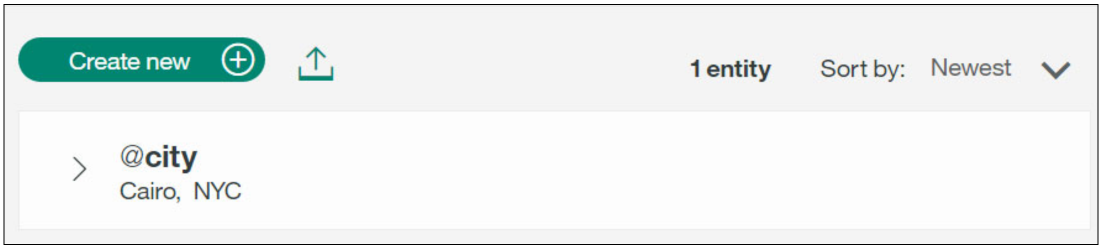


Figure 2-24 Create new entity

3. Name the entity `@destination`.

> **Note:** The at sign (`@`) is added by default to the name; do not add it yourself.

4. Add the following values and synonyms (Figure 2-25).
   - Value: `Home`
   - Synonyms: `My Address`
   - Value: `Work`
   - Synonyms: `IBM`, `Office`

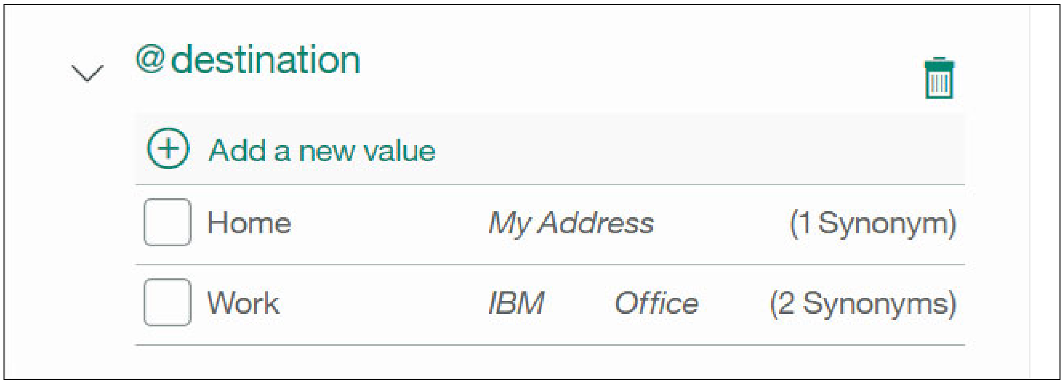


Figure 2-25 Add location entity

5. Click **Create**.

   The entity you created is added to the Entities tab, and the system begins to train itself with the new data.

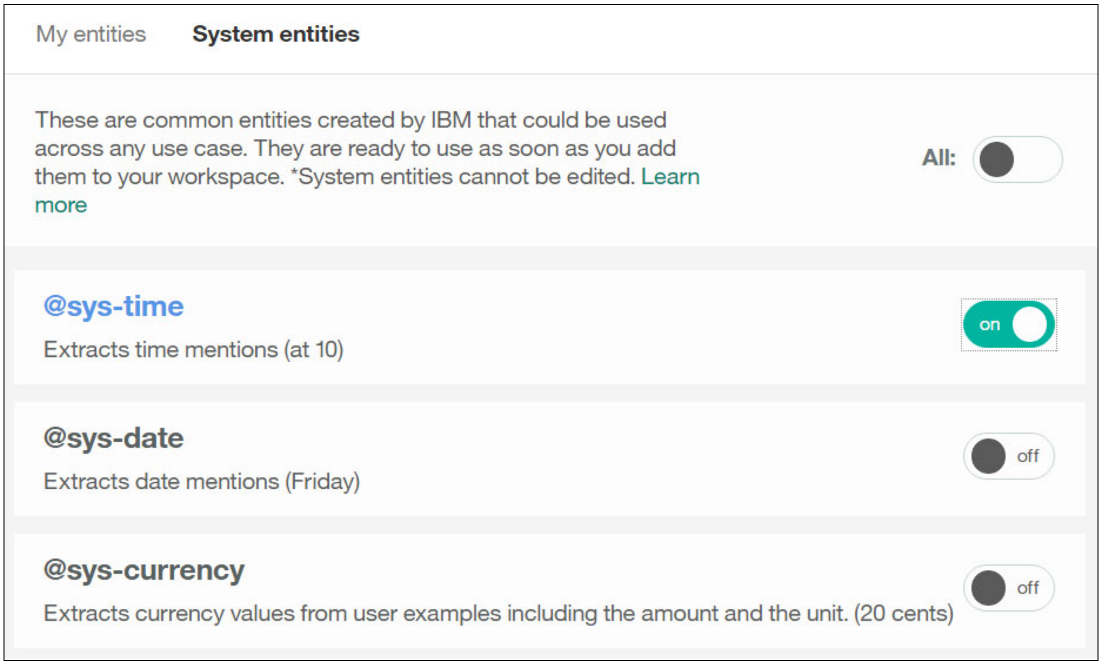
Add sys-time system entity

The Conversation service provides a number of system entities, which are common entities that you can use for any application.

The `@sys-time system` entity extracts *mentions* such as 2pm, at 4, or 15:30. The value of this entity stores the time as a string in the `HH:mm:ss` format, for example, `13:00:00`.

Complete the following steps to add a system entity from the Conversation tool:

1. Select the **System entities** tab. You can then choose from a list of system entities.
2. Click the on/off toggle switch next to the **@sys-time** entity to enable it (Figure 2-26).



My entities System entities

These are common entities created by IBM that could be used across any use case. They are ready to use as soon as you add them to your workspace. *System entities cannot be edited. Learn more

All: off

@sys-time
Extracts time mentions (at 10) on

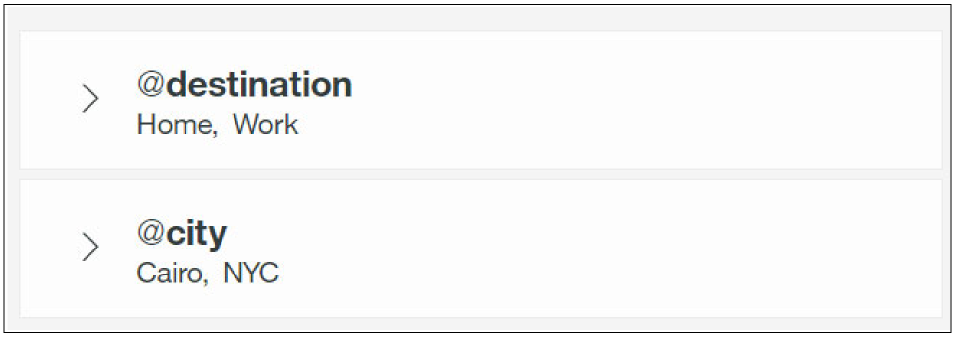
@sys-date
Extracts date mentions (Friday) off

@sys-currency
Extracts currency values from user examples including the amount and the unit. (20 cents) off

Figure 2-26 Add @sys-time system entity

Final entities list in workspace

Figure 2-27 shows the final My Entities list in the Car Chat-bot workspace.

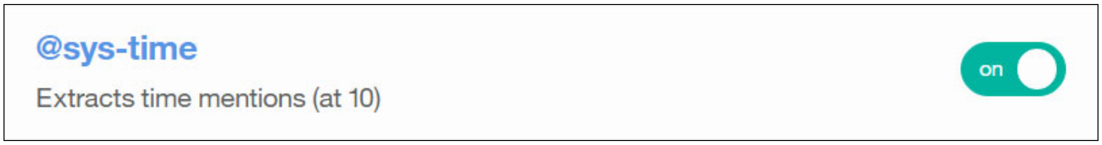


@destination
Home, Work

@city
Cairo, NYC

Figure 2-27 My Entities final list

Figure 2-28 shows the final system entities list in the Car Chat-bot workspace.



@sys-time
Extracts time mentions (at 10) on

Figure 2-28 System entities final list

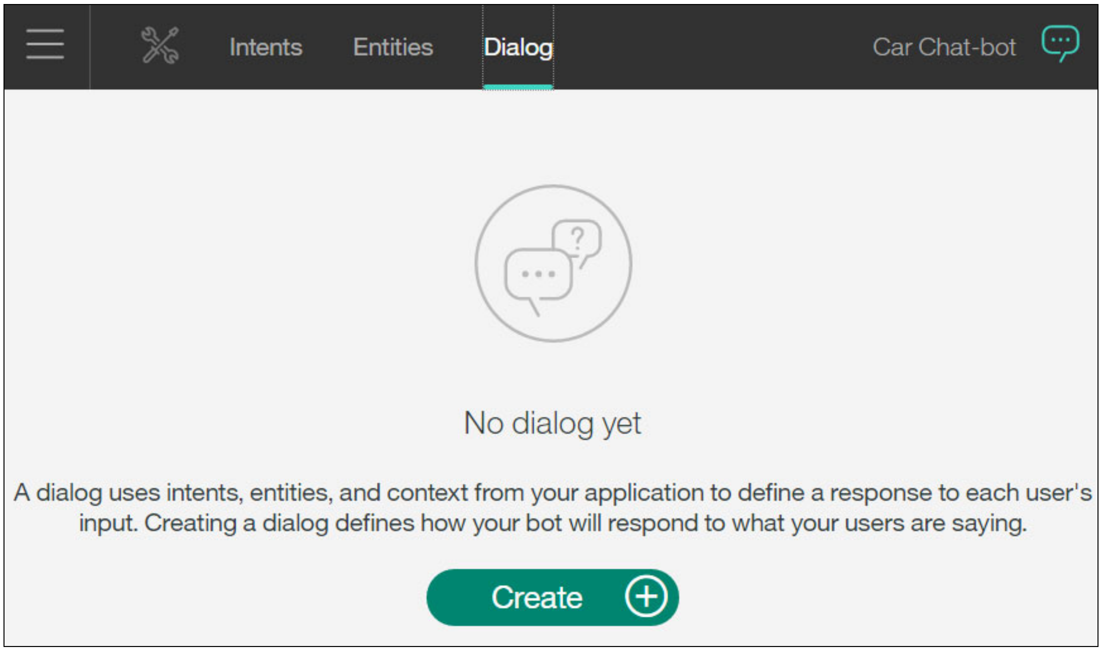
2.1.6 Building a dialog

In this section, you build the Conversation dialog for the car chatbot by using the created and imported intents and entities.

Start the dialog

Complete the following steps:

1. Click the **Dialog** tab and click **Create** (Figure 2-29).

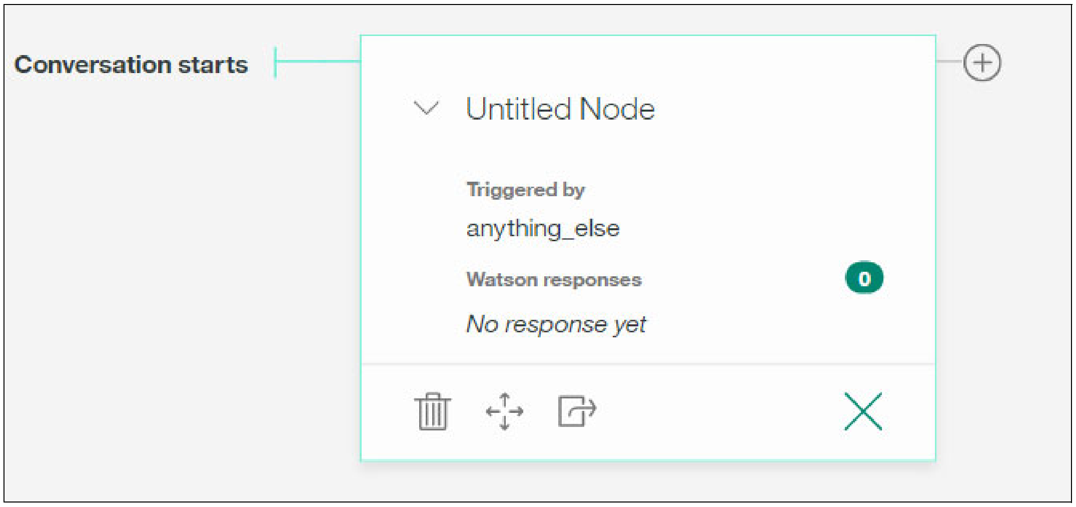


No dialog yet

A dialog uses intents, entities, and context from your application to define a response to each user's input. Creating a dialog defines how your bot will respond to what your users are saying.

Figure 2-29 Create new dialog

An untitled node is displayed in the dialog, when it is first created (Figure 2-30).



Untitled Node

Triggered by
anything_else

Watson responses 0
No response yet

Figure 2-30 Dialog created with a default node

2. In the edit view (Figure 2-31), enter the following details:

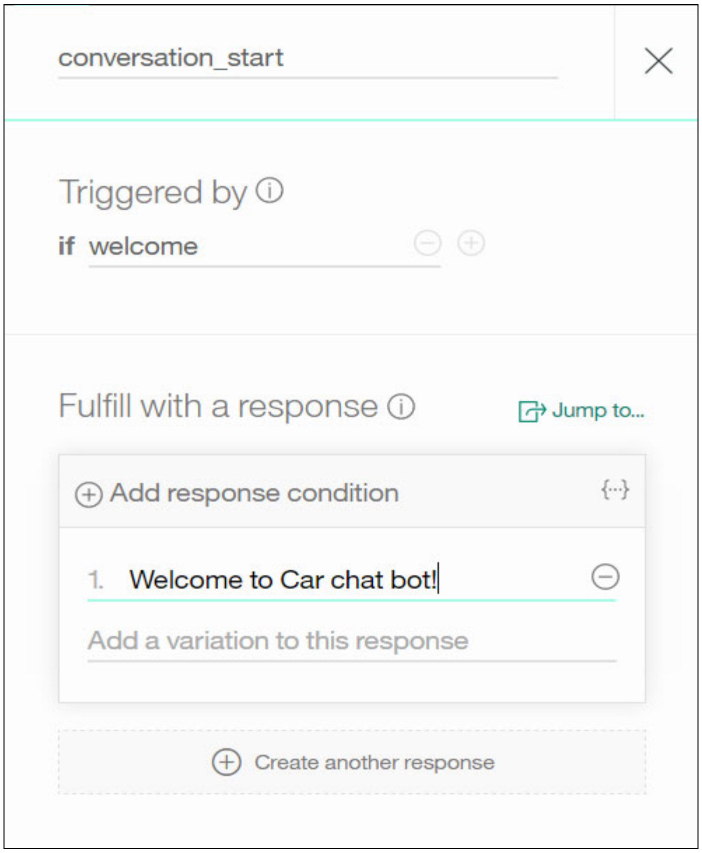
   – Node name: `conversation_node`
   – In the “Triggered by” (if) section:

     i. Start typing the word `welcome`.
     ii. From the list, select **Welcome (create new condition)**.

     > **Note:** When you create the condition in your first dialog node, a node with the `anything_else` condition is created in the dialog tree.

   – In the “Fulfill with a response” section, add the following text:

   ```
   Welcome to Car chat bot!
   ```



conversation_start

if welcome

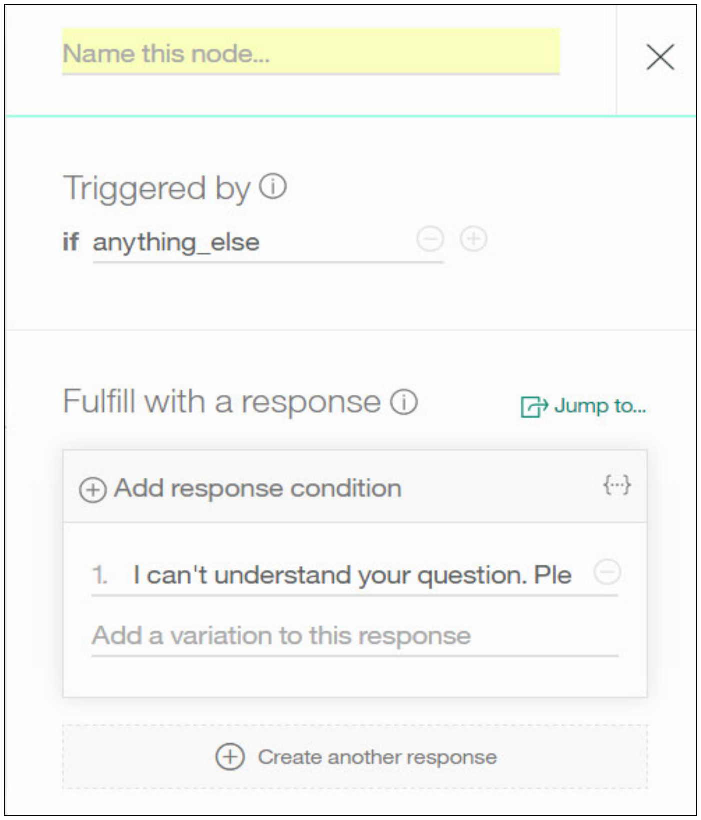
1. Welcome to Car chat bot!

Figure 2-31 First node details

3. In the dialog, click the **anything_else** node, to edit its details.

4. In the edit view (Figure 2-32), add a response in the “Fulfill with a response” section:

```
I can't understand your question. Please try again.
```

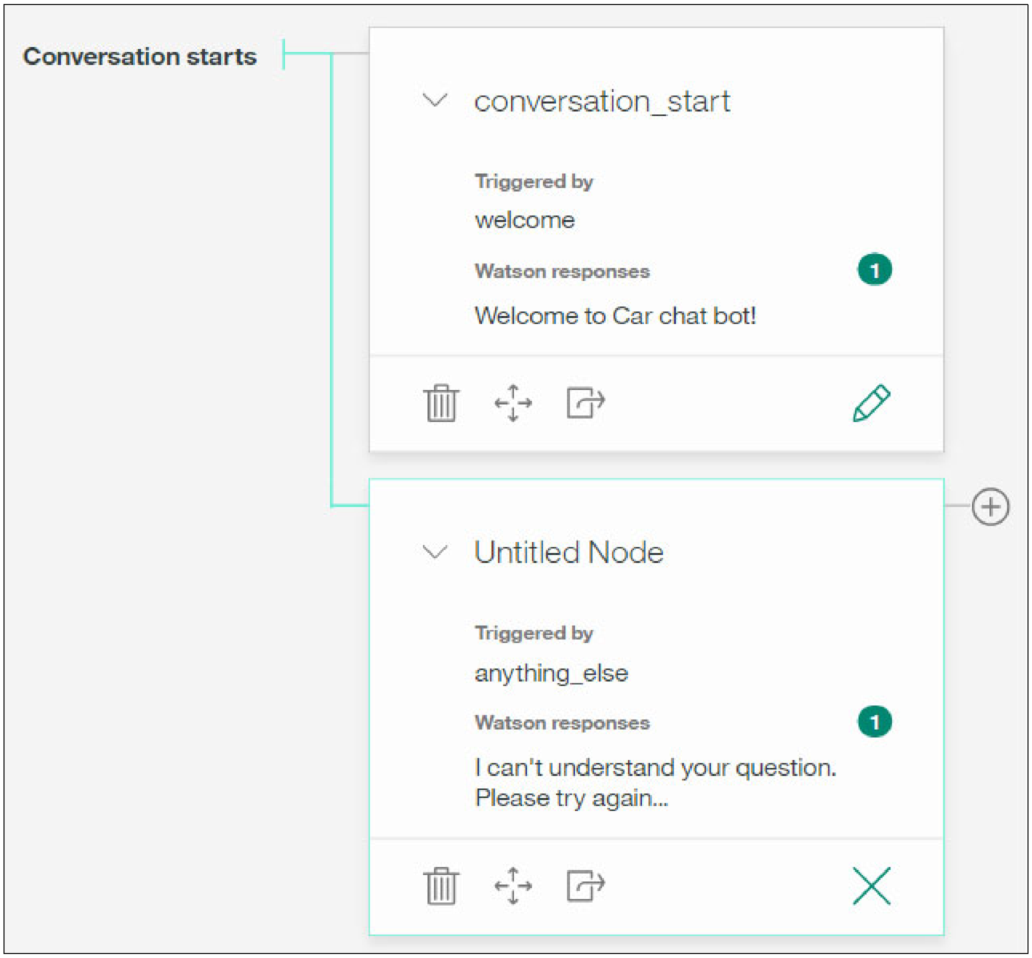


if anything_else

1. I can't understand your question. Ple

Figure 2-32 Details of the anything_else node

Figure 2-33 shows the dialog with the two nodes created so far.



Conversation starts

conversation_start

Triggered by

welcome

Watson responses

Welcome to Car chat bot!

Untitled Node

Triggered by

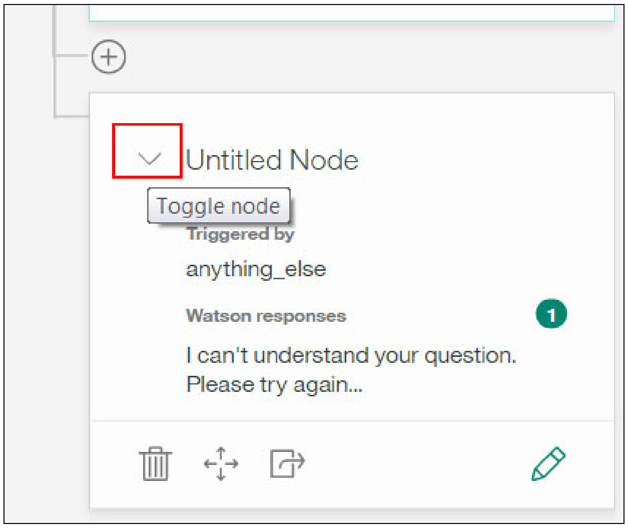
anything_else

Watson responses

I can't understand your question. Please try again...

Figure 2-33 Dialog with two initial nodes

5. You can collapse the `anything_else` node by clicking its Toggle node button (Figure 2-34).



Triggered by

anything_else

Watson responses

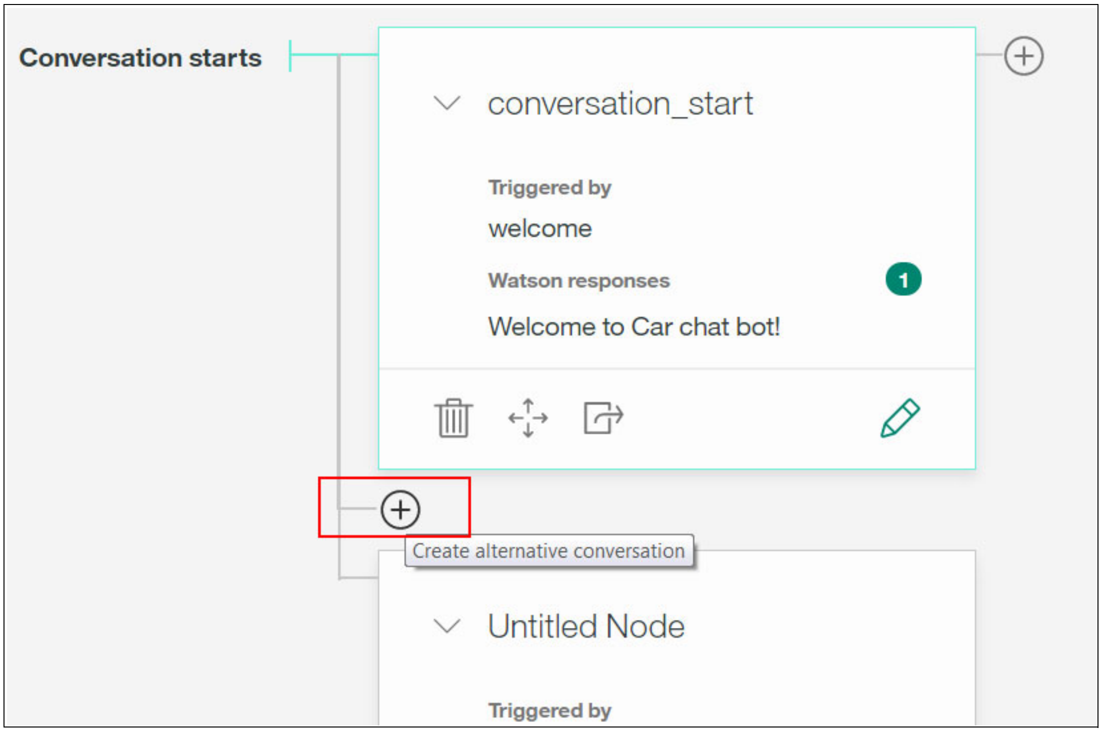
I can't understand your question. Please try again...

Figure 2-34 Collapsing the anything_else node

Create a branch to respond to the greeting intent

Complete the following steps

1. In the dialog, click the **conversation_start** node.
2. Click the plus sign (**+**) below the `conversation_start` node (Figure 2-35), to create a base node peer of the `conversation_start` node.



conversation_start

Triggered by

welcome

Watson responses

Welcome to Car chat bot!

Untitled Node

Triggered by

Figure 2-35 Create greeting node

3. In the edit view (Figure 2-36), add these details:

   – Node name: `greeting`
   – In the “Triggered by if” section:

     i. Start typing the word `greeting`.
     ii. From the list, select **#greeting**, which is the greeting intent you created previously.

   – In “Fulfill with a response” section, add the following text:

```
Hi! What can I do for you?
```

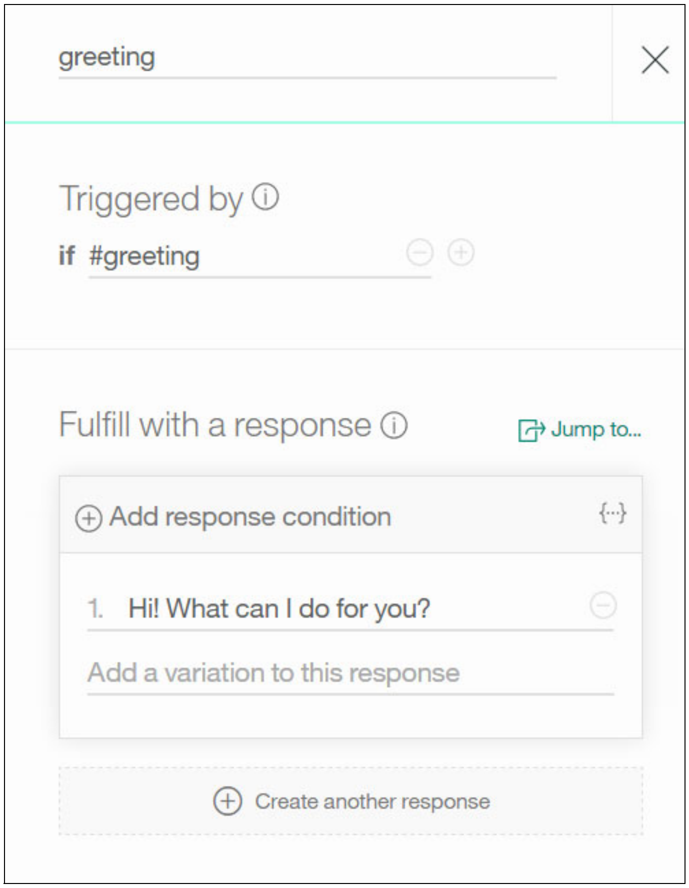


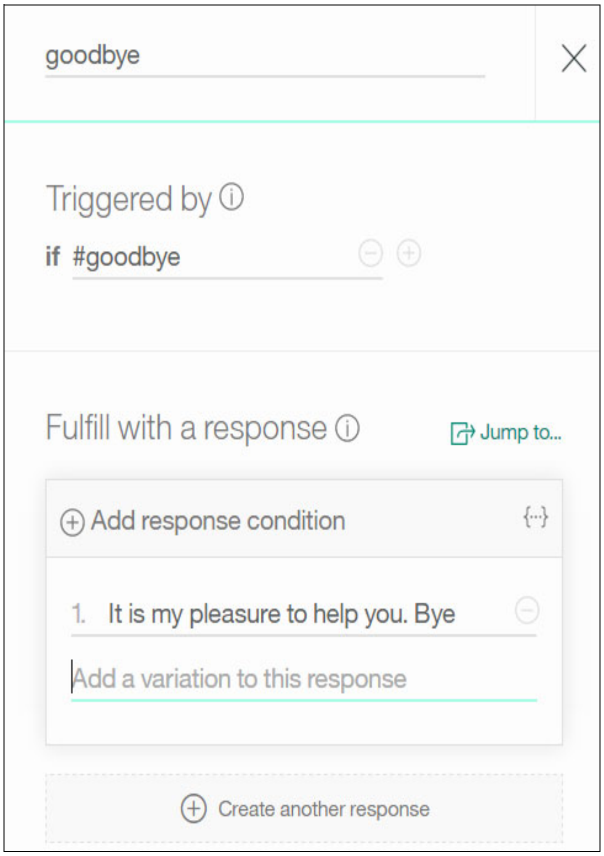
Figure 2-36 The greeting node details

Create a branch to respond to the goodbye intent

Complete the following steps:

1. In the dialog, click the **greeting** node.
2. Click the plus sign (**+**) below the `greeting` node (Figure 2-35 on page 34), to create a base node peer of the `greeting` node.
3. In the edit view (Figure 2-37), add these details:
   - Node name: `goodbye`
   - In the "Triggered by if" section:
     i. Start typing the word `goodbye`.
     ii. From the list, select **#goodbye**, which is the goodbye intent you created previously.
   - In "Fulfill with a response" section, add the following text:

     `It is my pleasure to help you. Bye`



goodbye

Triggered by ⓘ

if #goodbye

Fulfill with a response ⓘ

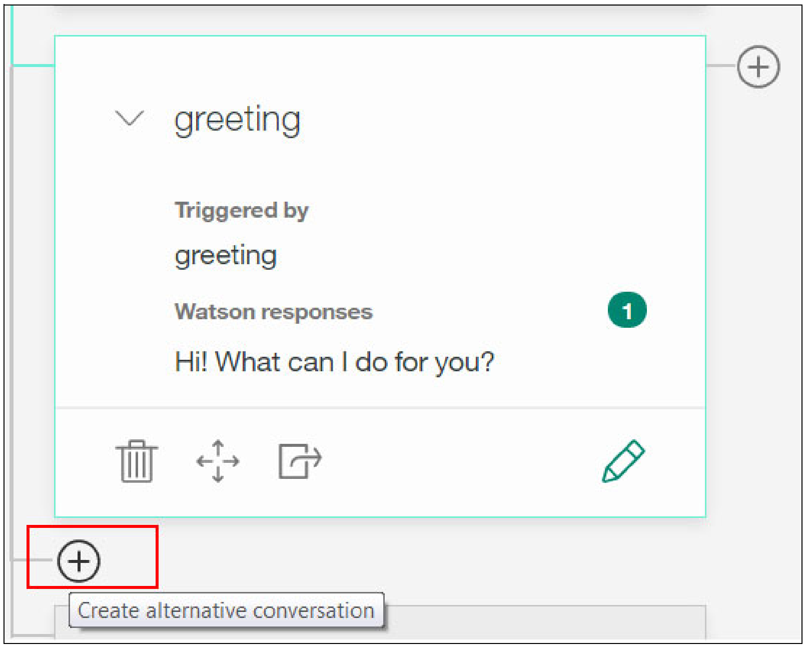
1. It is my pleasure to help you. Bye

Figure 2-37 The goodbye node details

Create a branch to respond to the traffic intent

Complete the following steps:

1. In the dialog, click the **greeting** node.
2. Click the plus sign (**+**) below the `greeting` node (Figure 2-38), to create a base node peer of the `greeting` node (that is, create an alternative conversation).



greeting

Triggered by

greeting

Watson responses

Hi! What can I do for you?

Figure 2-38 Create traffic node

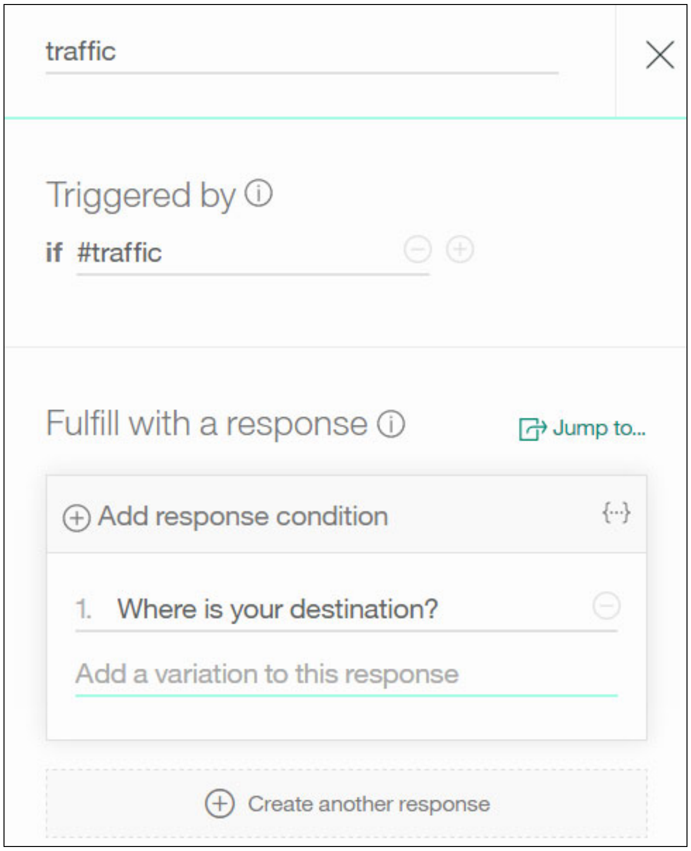
3. In the edit view (Figure 2-39). add these details:

   – Node name: `traffic`
   – In the “Triggered by if” section:

     i. Start typing the word `traffic`.
     ii. From the list, select **#traffic**, which is the traffic intent you created previously.

   – In “Fulfill with a response” section, add the following text:

     `Where is your destination?`


traffic

Triggered by ⓘ

if #traffic

Fulfill with a response ⓘ

1. Where is your destination?

Figure 2-39 The traffic node details

Create a child node for the traffic node

The #traffic intent requires additional processing, because the dialog needs to determine the location to get the traffic information for. To handle this, create a location child node for the traffic node:

1. In the dialog, click the **traffic** node.
2. Click the plus sign (**+**) next to the traffic node (Figure 2-40), to create a child node of the traffic node.

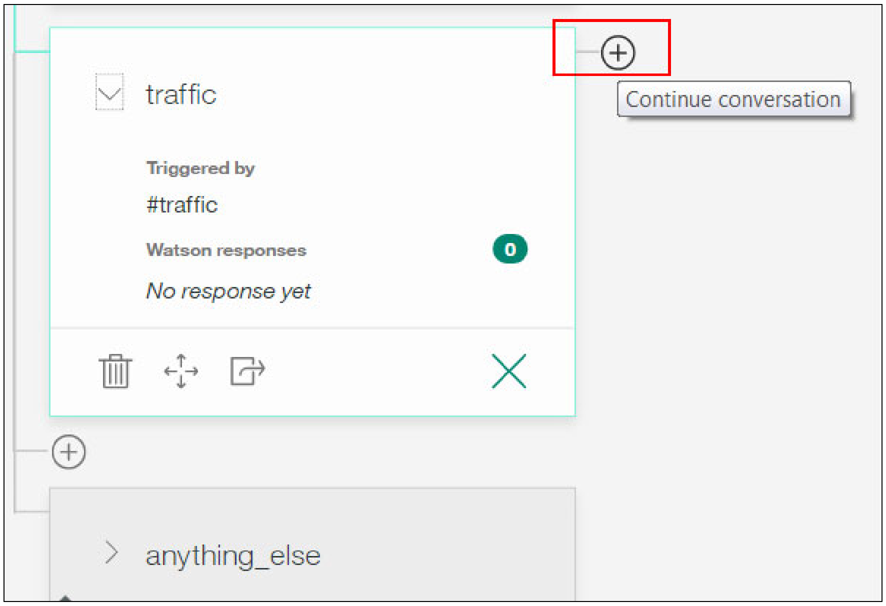
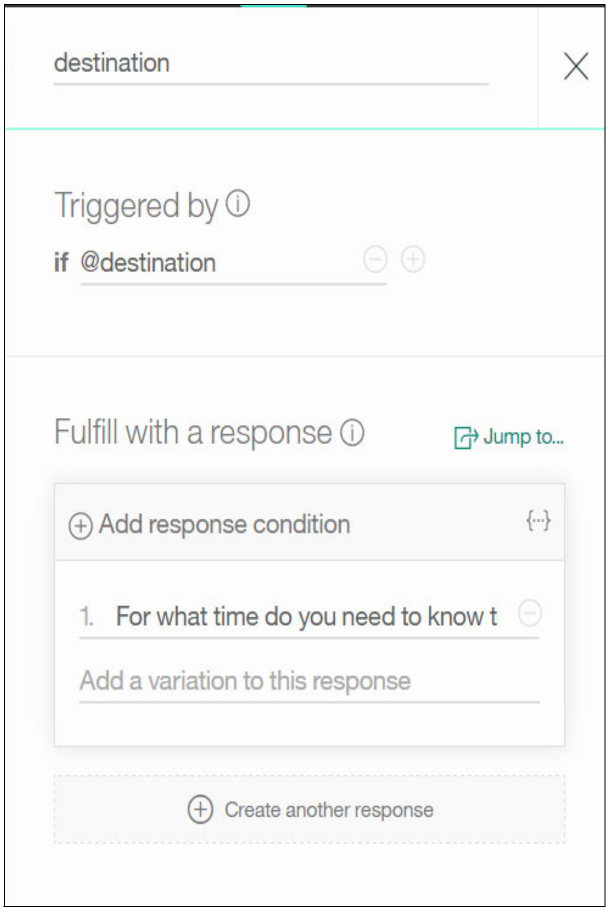


Figure 2-40 Create a destination node

3. In the edit view (Figure 2-41 on page 40), add these details:
   - Node name: `destination`
   - In the “Triggered by if” section:
     1. Start typing the word `destination`.
     2. From the list, select **@destination**, which is the destination entity you created previously.
   - In “Fulfill with a response” section, add the following text:

     ```
     For what time do you need to know the traffic information
     ```



destination

Triggered by

if @destination

Fulfill with a response

1. For what time do you need to know t

Figure 2-41 The destination node details

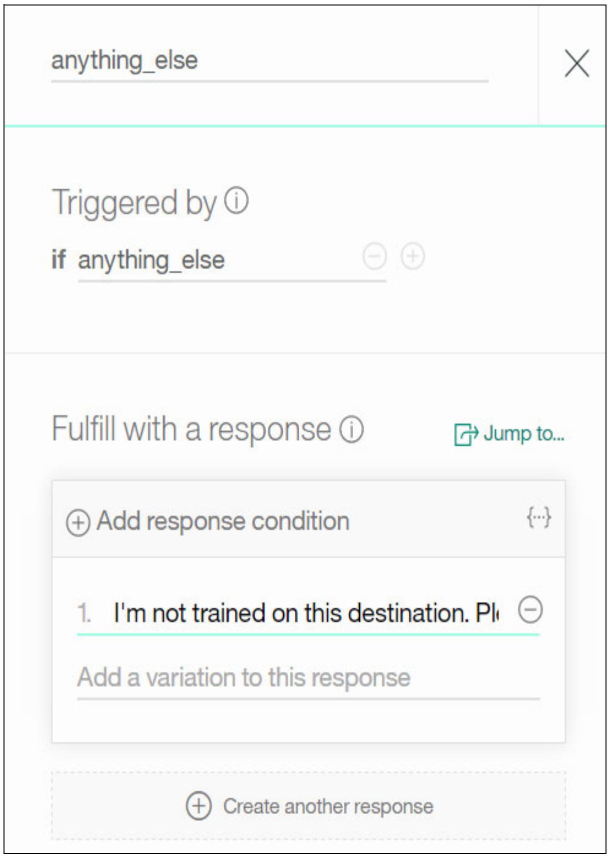
Create a fallback node for the destination node

Create a fallback node, in case the user did not enter valid input for the destination, which is either the synonym of `@destination.Home` or `@destination.Work`.

Complete these steps:

1. Click the plus sign (**+**) next to the `destination` node to create a child node of the `destination` node.
2. In the edit view (Figure 2-42 on page 41), add these details:
   - Node name: `anything_else`
   - In the “Triggered by if” section:
     i. Start typing the word `anything_else`.
     ii. From the list, select **anything_else (create new condition)**.
   - In “Fulfill with a response” section, add the following text:

```
I'm not trained for this destination. Please enter Home or Work only as a
destination.
```


anything_else

Triggered by ⓘ

if anything_else

Fulfill with a response ⓘ

⊕ Add response condition

1. I'm not trained on this destination. Pl

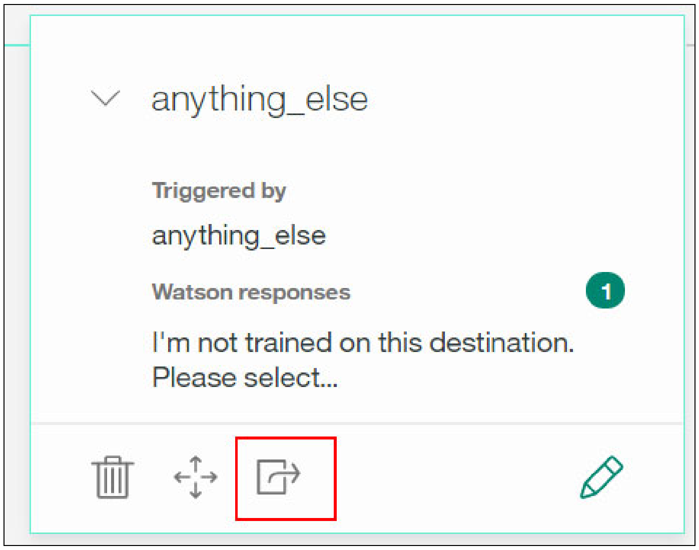
Add a variation to this response

⊕ Create another response

Figure 2-42 The destination fallback node details

After the response is fulfilled, you need to repeat the destination question again to let the user re-enter the destination. This can be done by using the `Jump to` function. You create a `Jump to` response as follows:

1. Click the **Jump to** button at the bottom of the `anything_else` node you just created (Figure 2-43).


anything_else

Triggered by

anything_else

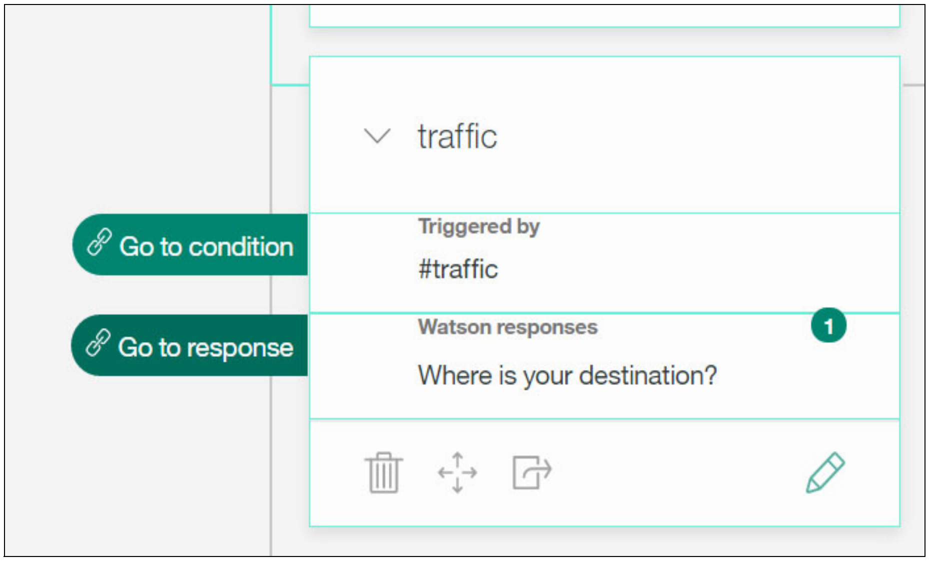
Watson responses

I'm not trained on this destination. Please select...

Figure 2-43 The Jump to button

2. Click the node that you want the response to go to. In this case, it is the traffic node to ask for the location again.

3. Select **Go to response** (Figure 2-44).



#traffic

Watson responses

Where is your destination?

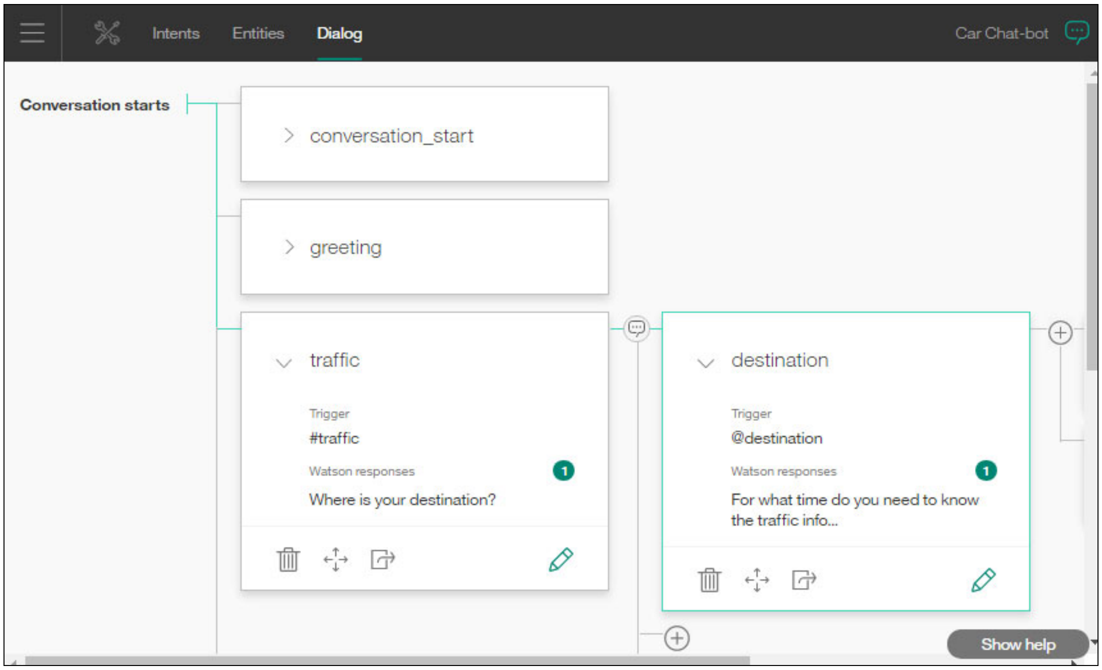
Figure 2-44 Go to response of traffic node

Create a child node for the destination node

After choosing the destination in the `@destination` entity, the dialog needs to know the time for which to get traffic information. Therefore, you create a *time* child node for the destination so the user can enter the time:

1. In the dialog, click the **destination** node.

2. Click the plus sign (**+**) next to the `destination` node (Figure 2-45) to create a child node of the `destination` node.



#traffic

Where is your destination?

@destination

For what time do you need to know the traffic info...

Figure 2-45 Creating child of destination node

3. In the edit view (Figure 2-46), add these details:

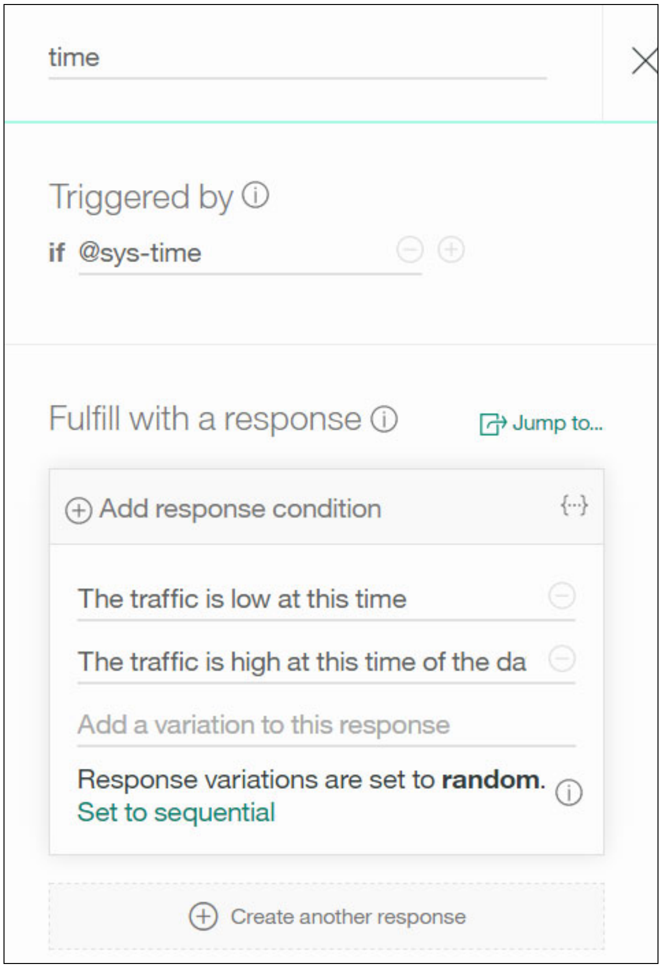
   – Node name: `time`
   – In the "Triggered by if" section:

     i. Start typing the word `sys-time`.
     ii. From the list, select **@sys-time**, which is the system entity @sys-time that you selected previously.

   – In "Fulfill with a response" section, add two random responses (press Enter after you add the first response):

     - `The traffic is low at this time`
     - `The traffic is high at this time of the day`

   – Click the **Set to random** link, to make sure the dialog randomly selects a response.


time

Triggered by

if @sys-time

Fulfill with a response

The traffic is low at this time

The traffic is high at this time of the da

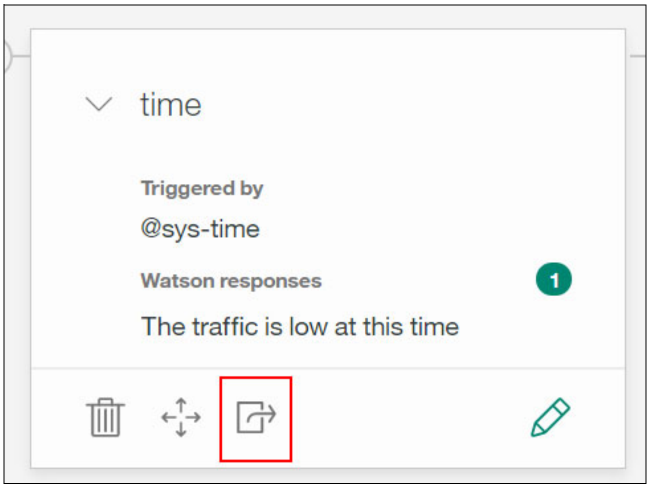
Response variations are set to **random**.

Figure 2-46 The time node details

After the chatbot responds with the traffic information, the dialog goes to the goodbye node to end the conversation.

To ensure that the dialog flows to the goodbye node, complete these steps:

1. On the time node, click the **Jump to** button (Figure 2-47).



time

Triggered by

@sys-time

Watson responses 1

The traffic is low at this time

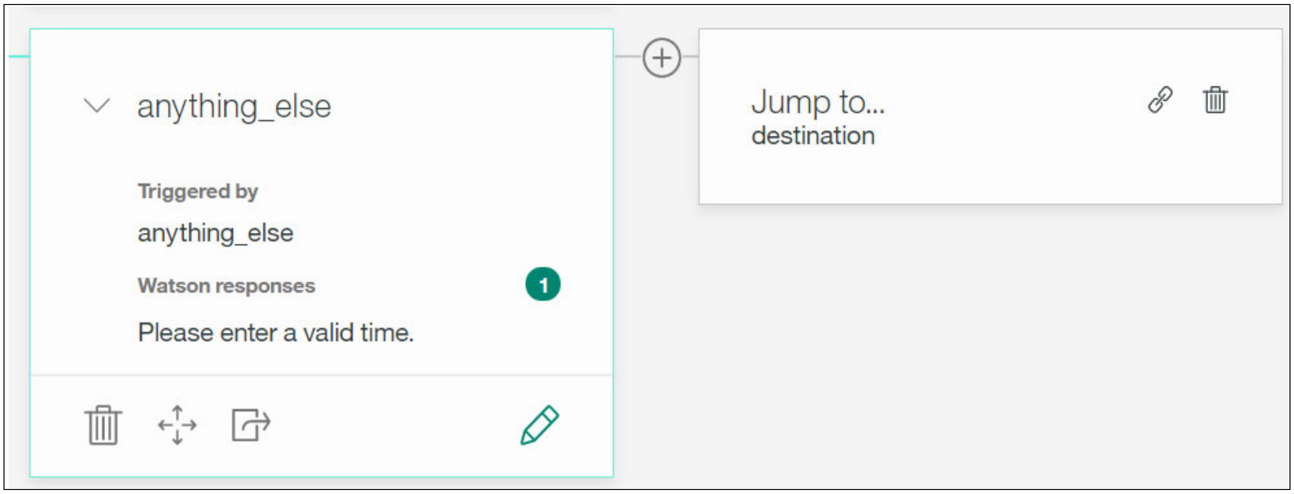
Figure 2-47 Jump to the goodbye node

2. Select the **goodbye** node, then select **Go to response**.

Create a fallback node for the time node

As for the location node, create a fallback node for the time node so that the dialog can go to it if the user did not enter a valid time.

Make the fallback node jump to a destination node response (Figure 2-48).



anything_else

Triggered by

anything_else

Watson responses 1

Please enter a valid time.

Jump to...

destination

Figure 2-48 The time node fallback

Create a branch to respond to the weather_inquiry intent

Complete the following steps:

1. In the dialog, click the **traffic** node.
2. Click the plus sign (**+**) at the bottom of the `traffic` node, to create a base node peer of the `traffic` node.
3. In the edit view, add these details:

   – Node name: `weather`
   – In the "Triggered by if" section:

     i. Start typing the word `weather`.
     ii. From the list, select **#weather_inquiry**, which is the weather_inquiry intent you created previously.

   – In "Fulfill with a response" section, add the following text:

   ```
   What's the city that you'd like to forecast the weather?
   ```

Figure 2-49 shows the weather node after creation.

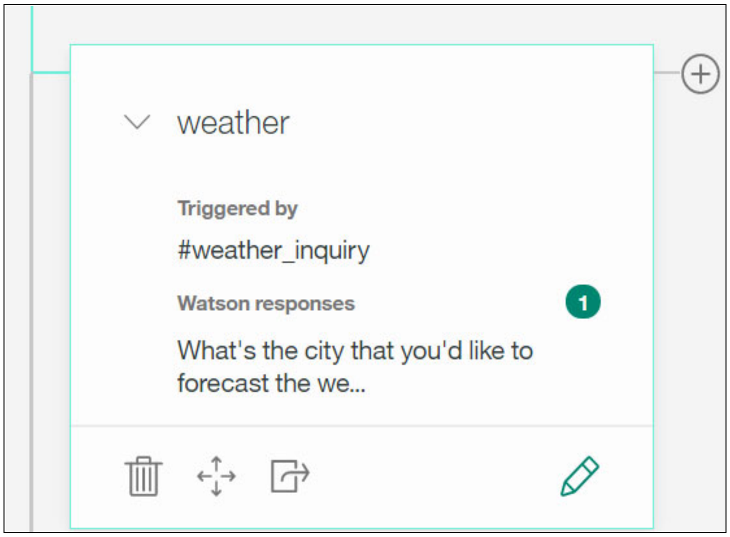


Figure 2-49 weather dialog node

Create a child node for the weather node

The `#weather_inquiry` intent requires additional processing because the dialog needs to determine the city in order to get the weather data for it. To handle this, create a city child node for the weather node:

1. In the dialog, click the **weather** node.
2. Click the plus sign (**+**) next to the `weather` node, to create a child node of the `weather` node.
3. In the edit view of the created node, add these details:

   – Node name: `city`
   – In the "Triggered by if" section:

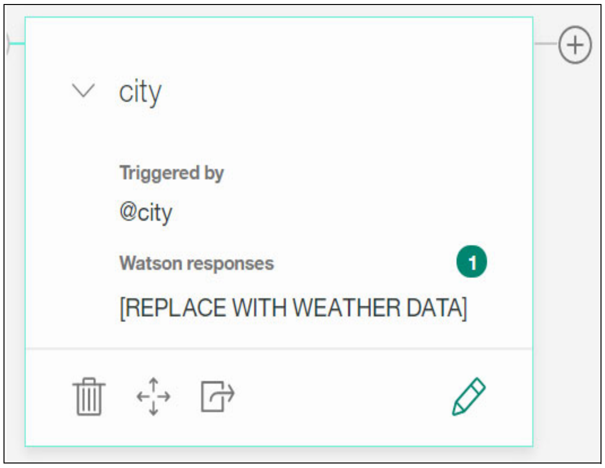
     i. Start typing the word `city`.
     ii. From the list, select **@city**, which is the city entity you created previously.

   – In "Fulfill with a response" section, add the following text:

   ```
   [REPLACE WITH WEATHER DATA]
   ```

Important: Do not provide a response here. In Chapter 6, "Chatting about the weather: Integrating Weather Company Data with the Conversation service" on page 157, this part will be integrated with the Weather Data Company service to get the weather information.

Figure 2-50 shows the city node after creation.



city

Triggered by

@city

Watson responses

[REPLACE WITH WEATHER DATA]

Figure 2-50 The city dialog node

After the chatbot responds with the weather data, the dialog goes to the `goodbye` node to end the conversation

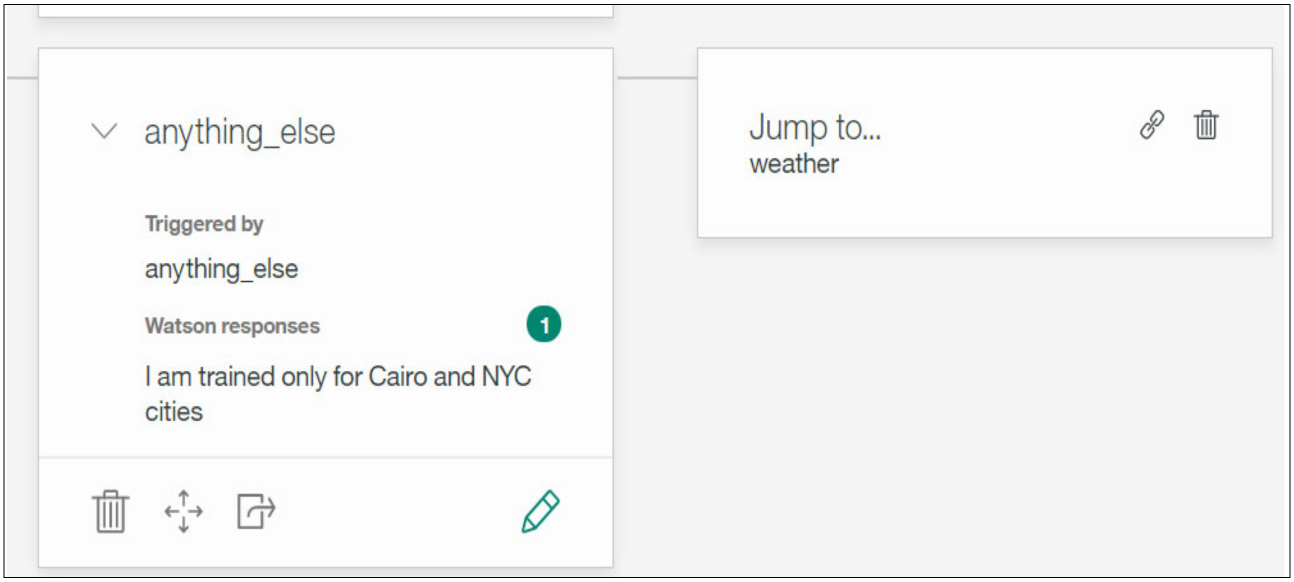
To ensure the dialog flows to the goodbye node, complete these steps:

1. On the city node, click the **Jump to** button.
2. Select the **goodbye** node, and then select **Go to response**.

Create a fallback node for the city node

Create a fallback node for the city node, for the dialog to go to if the user did not specify the NYC or Cairo cities.

Make the fallback node jump to the weather node response (Figure 2-51).



anything_else

Triggered by

anything_else

Watson responses 1

I am trained only for Cairo and NYC cities

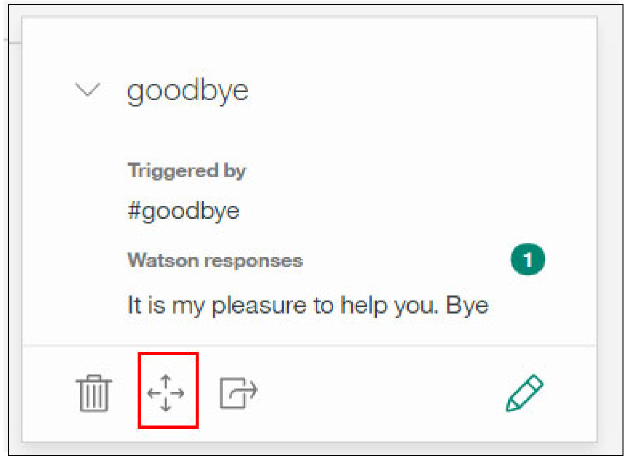
Jump to...
weather

Figure 2-51 the city node fallback

Move the goodbye node to the bottom

Complete the following steps to move the goodbye node to the bottom of the weather node:

1. On the goodbye node, click the **Move** button (Figure 2-52).



goodbye

Triggered by

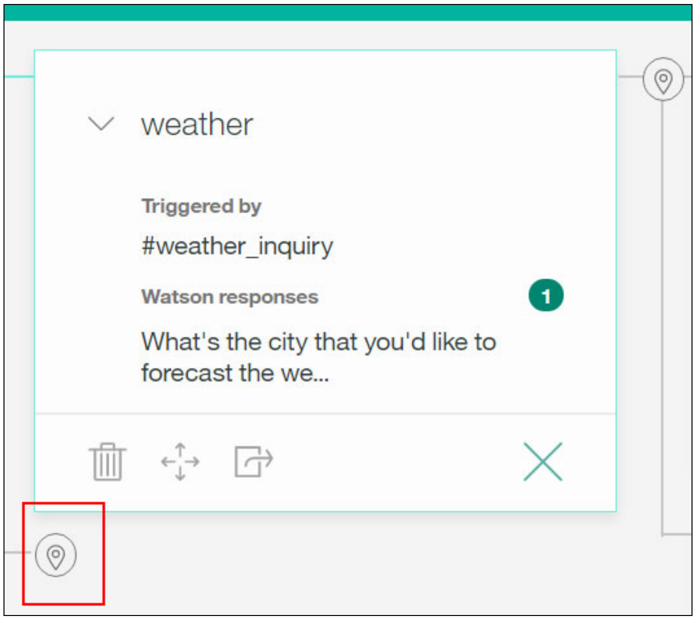
#goodbye

Watson responses 1

It is my pleasure to help you. Bye

Figure 2-52 Move dialog node

2. Select the **weather** node, then click the **Move** icon below it (Figure 2-53).



weather

Triggered by

#weather_inquiry

Watson responses

What's the city that you'd like to forecast the we...

Figure 2-53 Moving goodbye node to the bottom of weather node

The complete car chatbot dialog

This section acts as a checkpoint to make sure the dialog is created as it should be. The following sections show the first level dialog nodes and the traffic and weather child nodes.

Base nodes

Figure 2-54 shows the base nodes created with the child nodes collapsed.

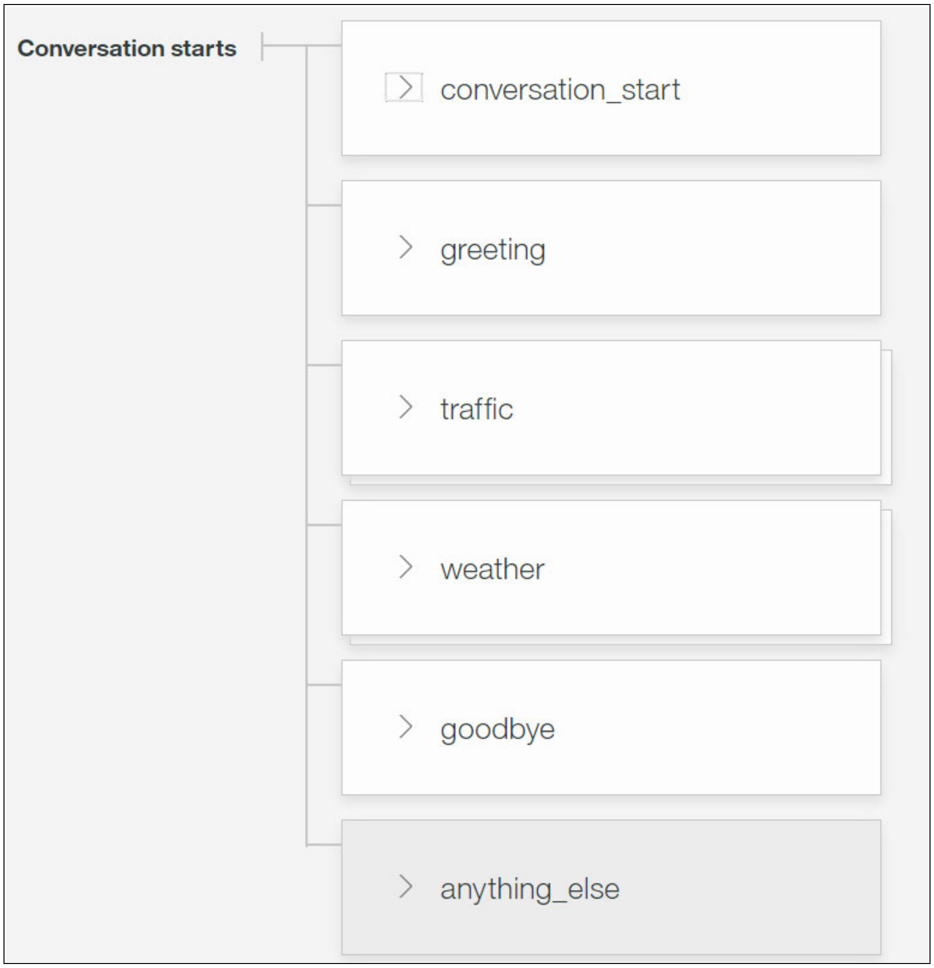
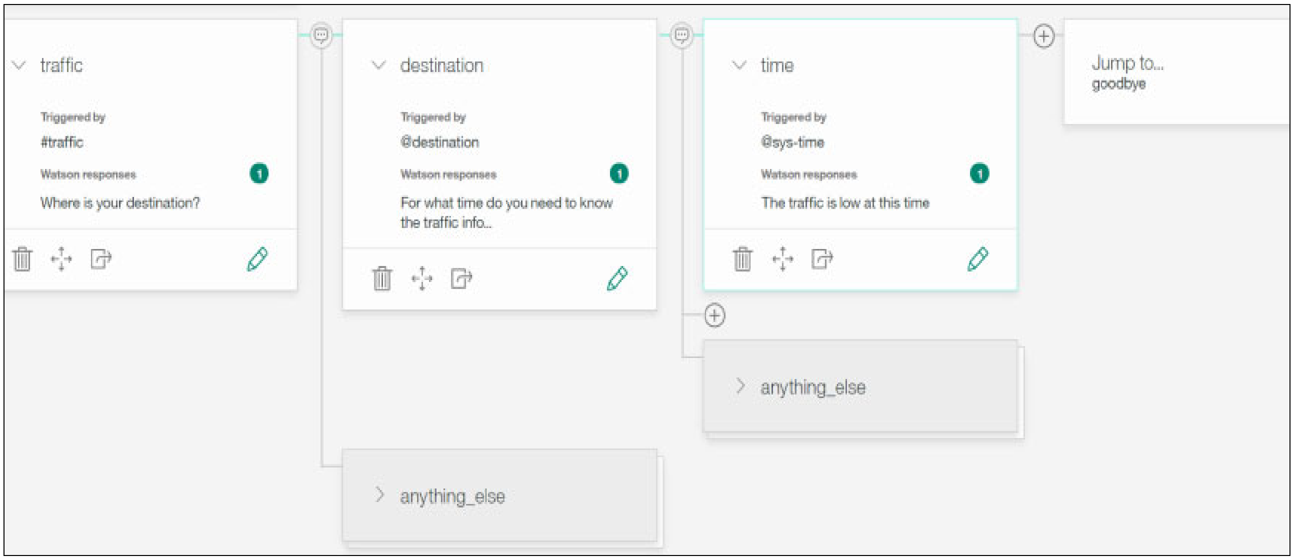


Figure 2-54 Base nodes in the dialog

The traffic child nodes

Figure 2-55 shows the traffic child nodes created with the fallback nodes collapsed.



traffic
Triggered by
#traffic
Watson responses
Where is your destination?

destination
Triggered by
@destination
Watson responses
For what time do you need to know the traffic info...

time
Triggered by
@sys-time
Watson responses
The traffic is low at this time

Jump to...
goodbye

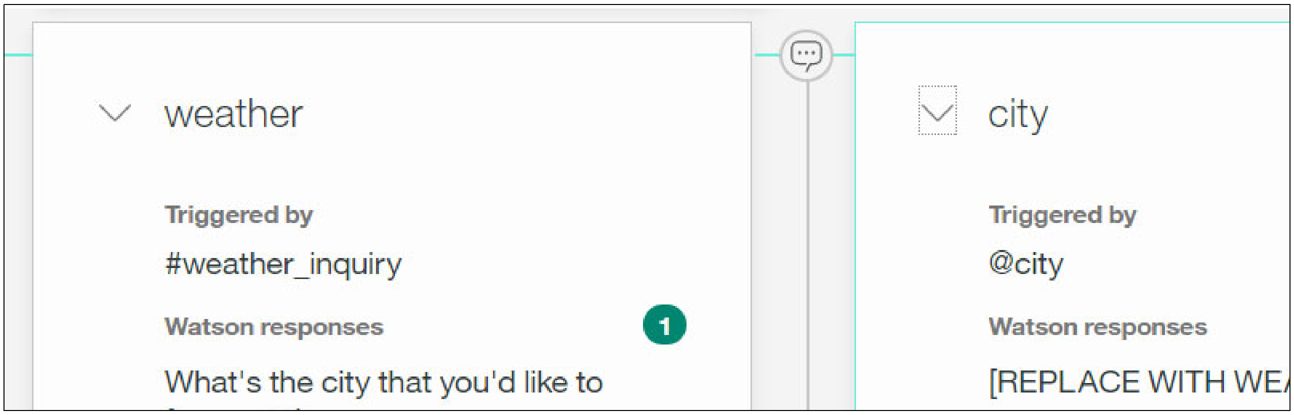
anything_else

anything_else

Figure 2-55 The traffic child nodes

The weather child nodes

Figure 2-56 shows the weather child nodes created with the fallback nodes collapsed.



weather
Triggered by
#weather_inquiry
Watson responses
What's the city that you'd like to

city
Triggered by
@city
Watson responses
[REPLACE WITH WEA

Figure 2-56 The weather child nodes

Test the dialog

After creating the nodes of the dialog, test it to determine how it responds to user inputs:

1. From the Dialog tab, click the 💬 icon at the upper right corner of the page (Figure 2-57).

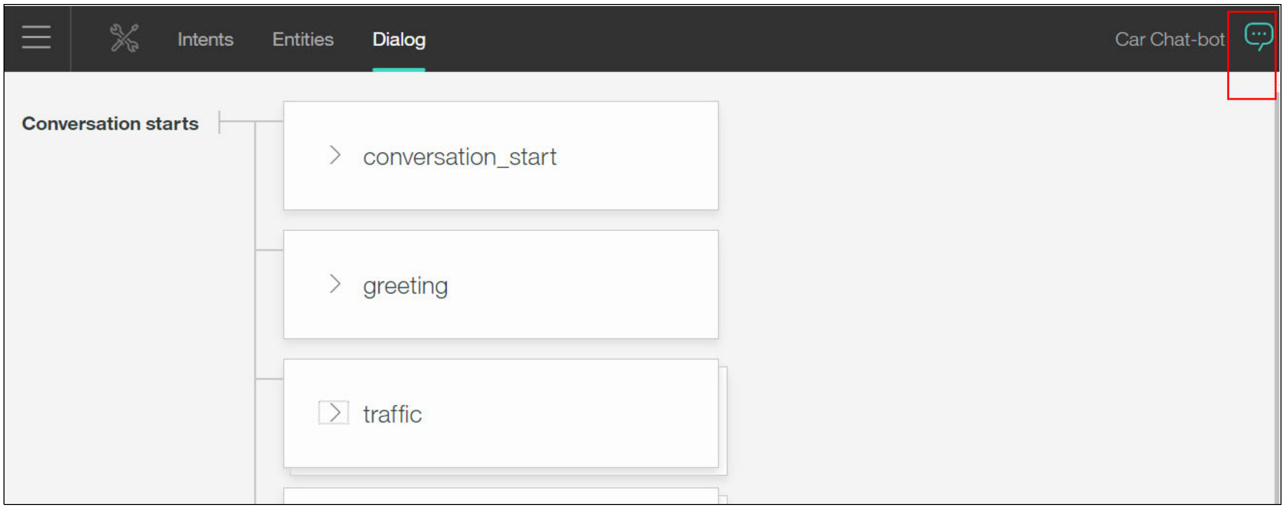
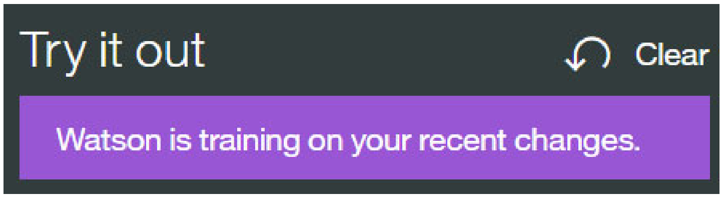


Figure 2-57 Test Dialog icon

2. Wait until the system finishes training your most recent changes before you start testing the dialog. If the system is still training, a message appears at the top of the chat pane (Figure 2-58).



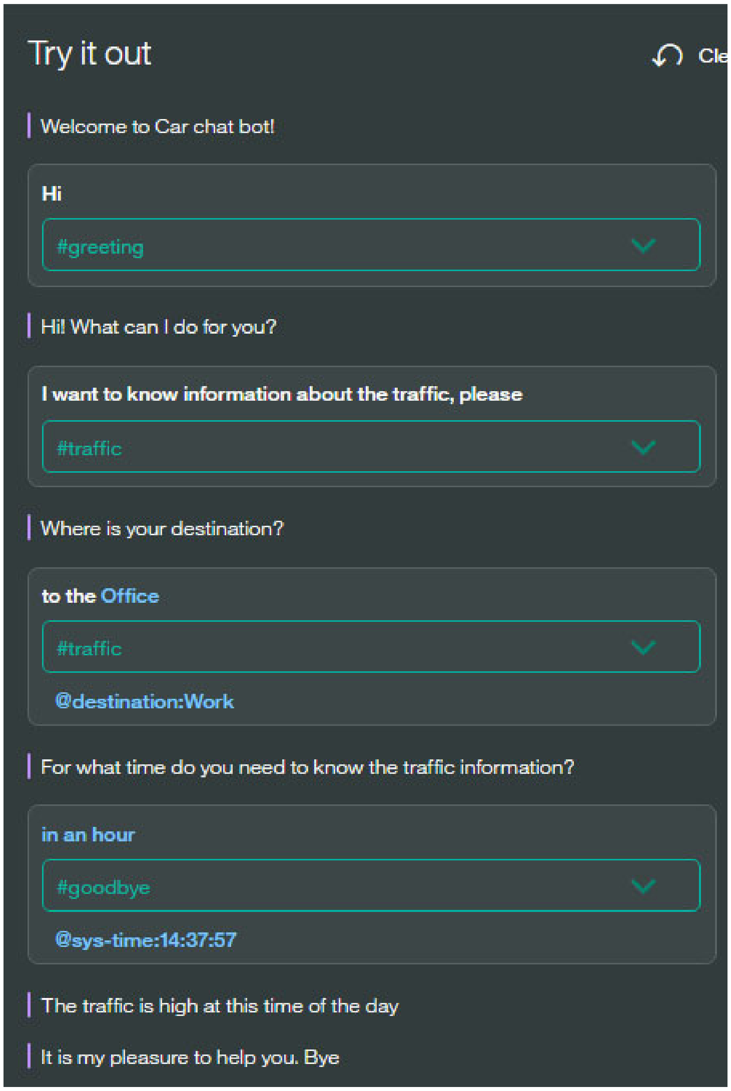
Watson is training on your recent changes.

Figure 2-58 Watson is training message

3. Start testing the dialog after the system finishes the training. Check the response to see if the dialog correctly interpreted your input and chose the correct response.

   The chat window indicates what intents and entities were recognized in the input.

Figure 2-59 shows the Car chatbot conversation dialog to get the traffic information.



Welcome to Car chat bot!

Hi

#greeting

Hi! What can I do for you?

I want to know information about the traffic, please

#traffic

Where is your destination?

to the Office

#traffic

@destination:Work

For what time do you need to know the traffic information?

in an hour

#goodbye

@sys-time:14:37:57

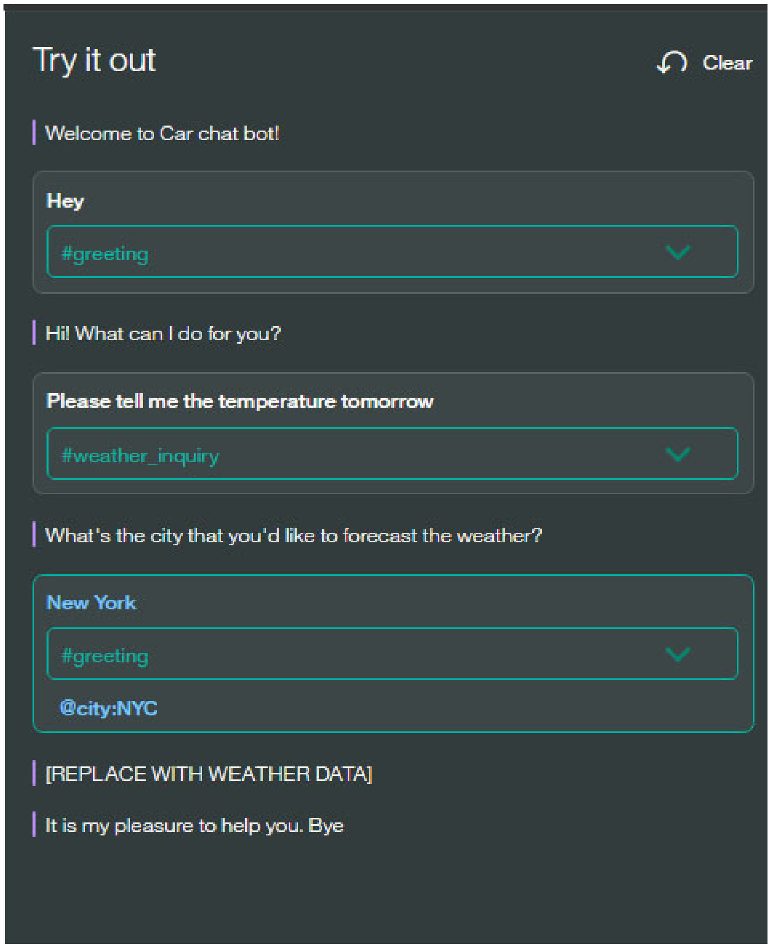
The traffic is high at this time of the day

It is my pleasure to help you. Bye

Figure 2-59 Testing traffic conversation dialog

Figure 2-60 on page 53 shows the Car chatbot conversation dialog to get the weather data.

> **Note:** The weather response is now `[REPLACE WITH WEATHER DATA]`. In Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157, the response will be replaced with the real-time weather after integrating the Conversation service with the Weather Data Company service to provide real-time weather data for the selected city.



Welcome to Car chat bot!

Hey

#greeting

Hi! What can I do for you?

Please tell me the temperature tomorrow

#weather_inquiry

What's the city that you'd like to forecast the weather?

New York

#greeting

@city:NYC

[REPLACE WITH WEATHER DATA]

It is my pleasure to help you. Bye

Figure 2-60 Testing weather conversation dialog

As you continue to interact with the dialog, you can see how the conversation flows through the dialog.

If you determine that the wrong intents or entities are being recognized, you might need to modify your intent or entity definitions. If the correct intents and entities are being recognized, but the wrong nodes are being triggered in your dialog, make sure your conditions are written correctly.

2.2 Exporting the workspace

You created intents, entities, and the dialog in the previous sections of this chapter.

Now you can export the workspace to a JSON file with all intents, entities, and dialog. To do this, click the **Actions** button (vertical dots) at the top right of the Workspaces box, and then select **Download as JSON** (Figure 2-61).

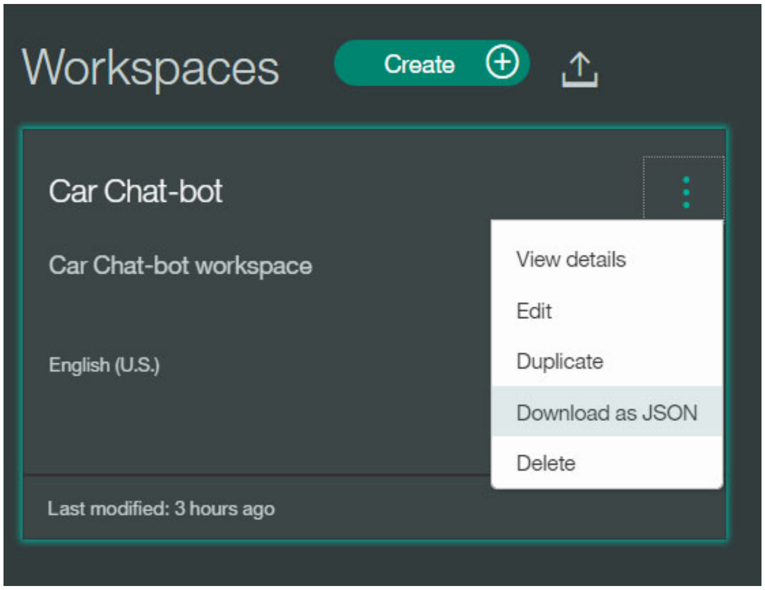


Figure 2-61 Export workspace

A JSON file will be downloaded automatically.

2.3 References

Watch the following videos about the Watson Conversation service:

- Watson Conversation Service Overview:

  https://www.youtube.com/watch?v=1rTl1WEbg5U
- IBM Watson Conversation: Working with intents:

  https://www.youtube.com/watch?v=DmvN6ZJrZE4
- IBM Watson Conversation: Working with entities:

  https://www.youtube.com/watch?v=oSNF-QCbuDc
- IBM Watson Conversation: Working with dialog:

  https://www.youtube.com/watch?v=3HSaVfr3ty0
- IBM Watson Conversation: Working with Conditional Responses:

  https://www.youtube.com/watch?v=KcvVQAsnhLM

3

Cognitive Calculator chatbot

This chapter guides you through building the Cognitive Calculator chatbot sample application. The app demonstrates the use of Watson Conversation service in creating a calculator chatbot. The chatbot chats with the user in natural language, the Conversation service determines the user request and the application performs simple calculations to respond to the user.

The sample application demonstrates the integration of the Conversation service with a Node.js application.

The following topics are covered in this chapter:

- Getting started
- Architecture
- Two ways to deploy the application: Step-by-step and quick deploy
- Step-by-step implementation
- Quick deployment of application
- References

3.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

3.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Create a Conversation service instance in IBM Bluemix.
- Create a Conversation workspace, add intents, entities, system entities, and a dialog for the Cognitive Calculator chatbot application.
- Integrate the Watson Conversation service in a Node.js application to perform the calculation functionality.

3.1.2 Prerequisites

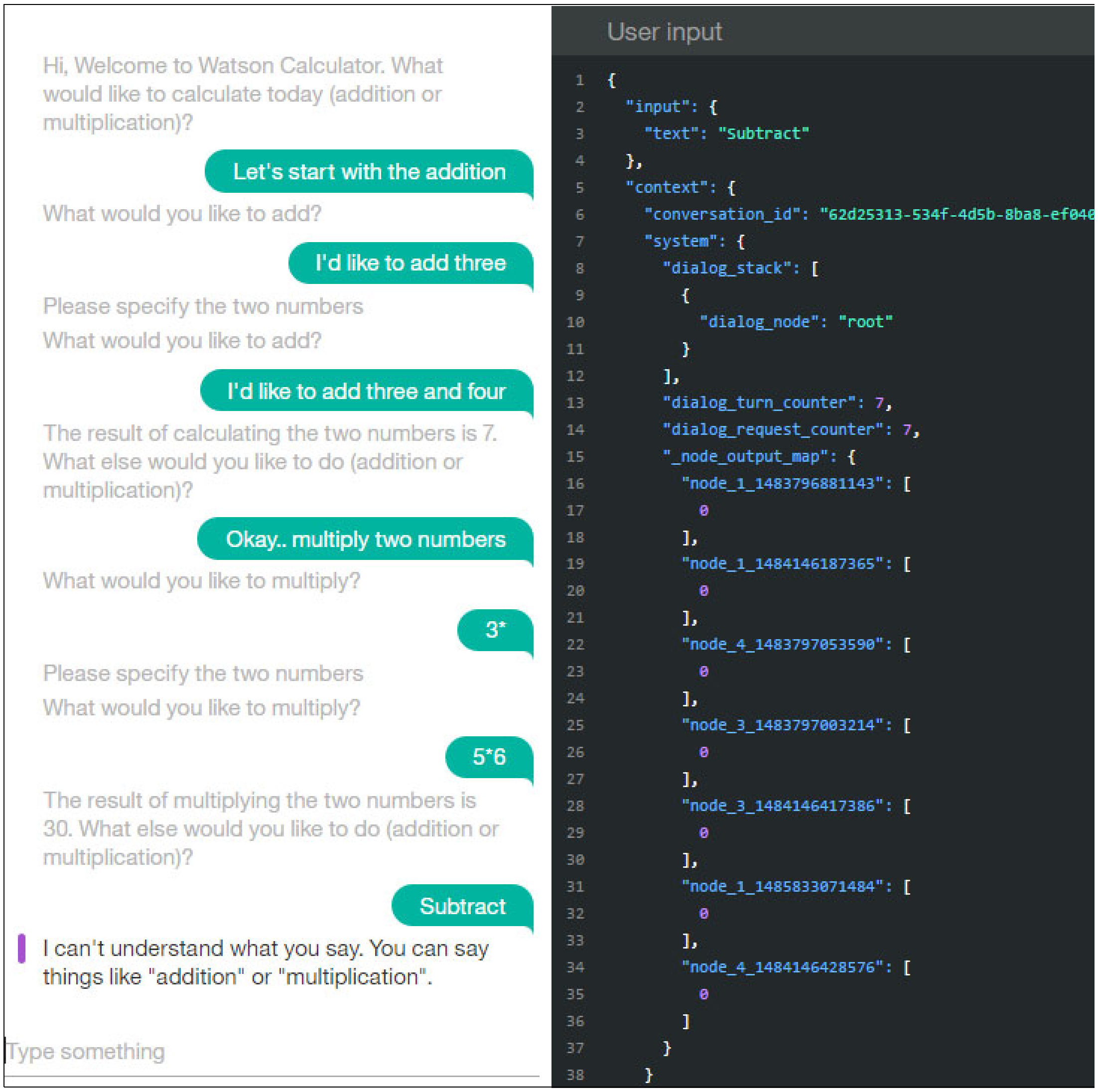
To complete the steps in this chapter, be sure these prerequisites are met:

- Review Chapter 1, “Basics of Conversation service” on page 1, and Chapter 2, “Conversation service workspace” on page 13
- Access to a web browser (Chrome, Firefox, or Internet Explorer)
- Basic JavaScript skills
- Understand Bluemix DevOps basics
- Understand Git basics
- Have a Bluemix account
- Have an account on GitHub

3.1.3 Expected results

Figure 3-1 on page 57 shows the simple Cognitive Calculator chatbot application:

1. The user starts the conversation with the addition operation.
2. The user tries to add two numbers but specifies only one number without specifying the other number.
3. The chatbot application prompts the user to specify two numbers to be able to perform the addition operation.
4. The user specifies the two numbers to add.
5. The application adds the two numbers and returns the result to the user.
6. The user then wants to multiply two numbers.
7. The chatbot prompts the user to enter the numbers to multiply.
8. The user requests a subtraction operation which the chatbot application does not understand.



Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

Let's start with the addition

What would you like to add?

I'd like to add three

Please specify the two numbers

What would you like to add?

I'd like to add three and four

The result of calculating the two numbers is 7. What else would you like to do (addition or multiplication)?

Okay.. multiply two numbers

What would you like to multiply?

3*

Please specify the two numbers

What would you like to multiply?

5*6

The result of multiplying the two numbers is 30. What else would you like to do (addition or multiplication)?

Subtract

I can't understand what you say. You can say things like "addition" or "multiplication".

```
{
  "input": {
    "text": "Subtract"
  },
  "context": {
    "conversation_id": "62d25313-534f-4d5b-8ba8-ef040
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 7,
      "dialog_request_counter": 7,
      "_node_output_map": {
        "node_1_1483796881143": [
          0
        ],
        "node_1_1484146187365": [
          0
        ],
        "node_4_1483797053590": [
          0
        ],
        "node_3_1483797003214": [
          0
        ],
        "node_3_1484146417386": [
          0
        ],
        "node_1_1485833071484": [
          0
        ],
        "node_4_1484146428576": [
          0
        ]
      }
    }
```

Figure 3-1 Cognitive Calculator chatbot

3.2 Architecture

Figure 3-2 shows the components and runtime flow of the application.

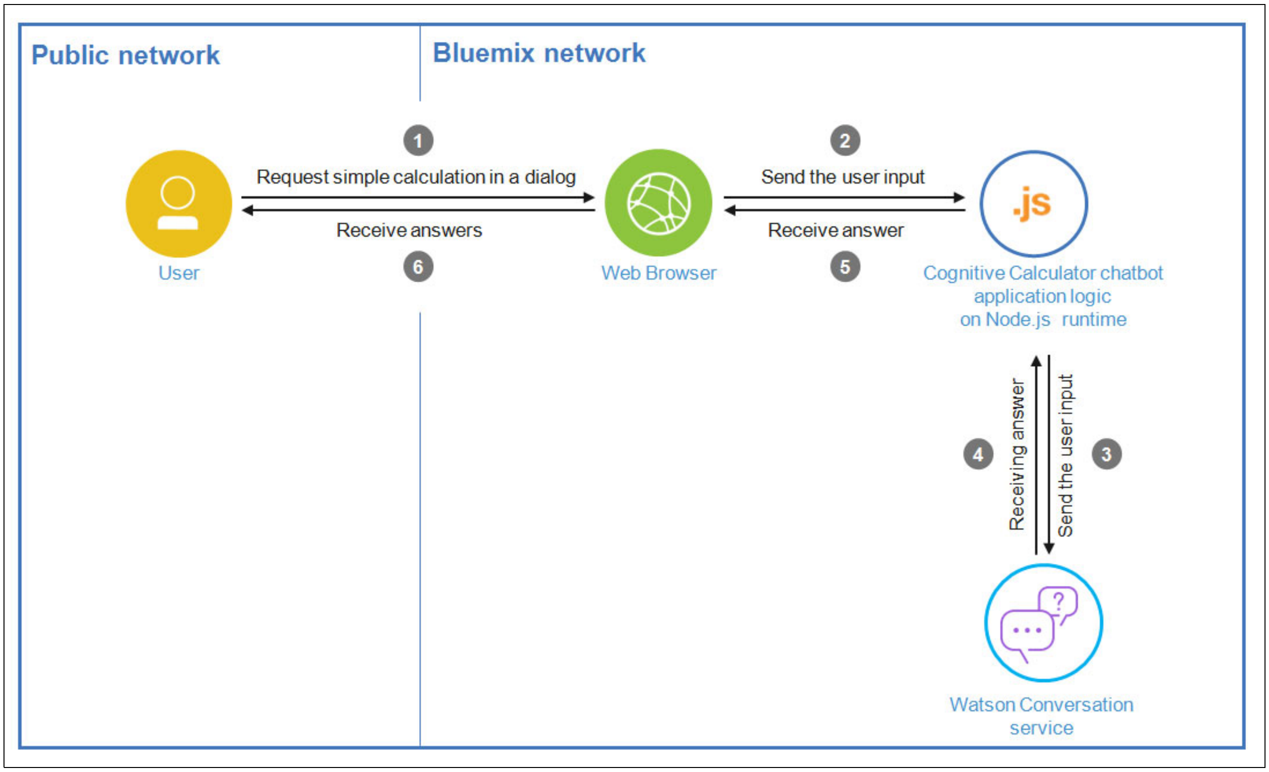


Figure 3-2 Architecture

The figure describes these steps:

1. In a web browser, the user engages in a conversation with the Cognitive Calculator chatbot application, requesting a simple calculation operation, such as `I'd like to calculate the addition of 3 and 5`.
2. The request is passed from the web browser to the chatbot application that runs on Node.js.
3. The application passes the request to the Conversation service.
4. The Conversation service understands the intent and entities passed by the application. For the user request `I'd like to calculate the addition of 3 and 5`, the *intent* is addition and the *entities* are `3` and `5`. Then, it returns a response to the application based on the dialog built in the workspace associated with Conversation service instance. It returns a response (`The result of calculating the two numbers is _result_. What else would you like to do (addition or multiplication)?`) and the entities to the calling chatbot application.
5. The chatbot Node.js application adds the two entities returned from the Conversation service, replaces the `_result_` with the calculation results and sends the response to the web browser.
6. The user sees the response on the web browser: `The result of calculating the two numbers is _result_. What else would you like to do (addition or multiplication)?`

3.3 Two ways to deploy the application: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- Step-by-step deployment (incomplete) version of the application

  This repository contains an incomplete version of the application and is used in all sections of 3.4, “Step-by-step implementation” on page 59. This version takes you through the key steps to integrate the IBM Watson APIs with the application logic.

- Quick deployment (complete) version of the application

  This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 3.5, “Quick deployment of application” on page 107.

3.4 Step-by-step implementation

Implementing this use case involves the following steps:

1. Downloading the project from the Git repository.
2. Configuring the Conversation workspace for the Cognitive Calculator chatbot.
3. Developing the Cognitive Calculator chatbot application.
4. Testing the application.

3.4.1 Downloading the project from the Git repository

The version of the repository that you use in these steps includes the *incomplete* version of the application code. You will follow the steps to complete the code.

Download the code from GitHub:

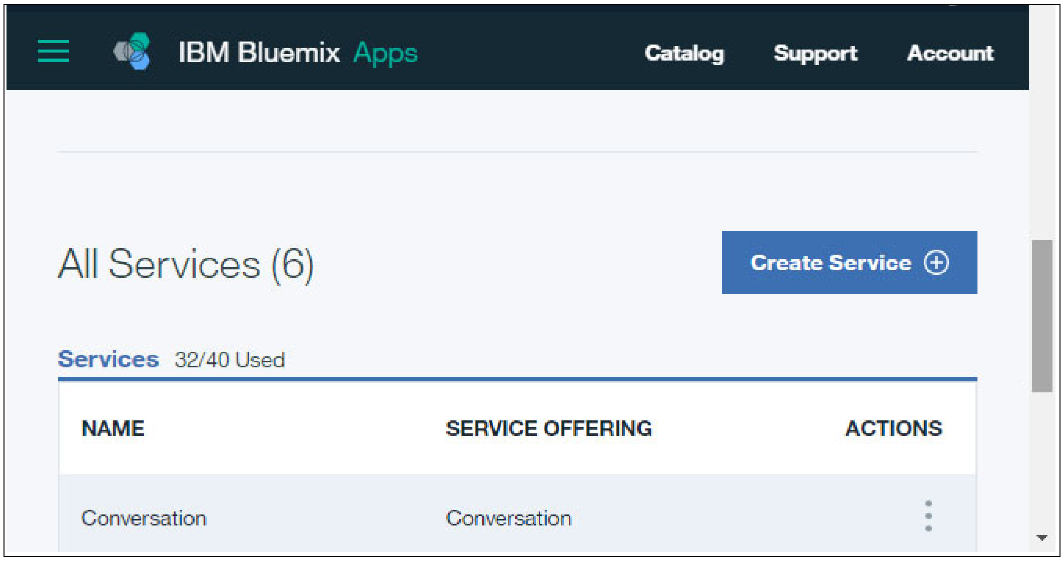
`https://github.com/watson-developer-cloud/conversation-simple`

3.4.2 Configuring the Conversation workspace for the Cognitive Calculator chatbot

This section guides you through creating the `Calculator` Conversation workspace for the Cognitive Calculator chatbot, and developing the relevant intents, entities, and dialog that are specific to the application. It also shows you how to test the conversation flow.

Complete these steps:

1. Log in to Bluemix.
2. On the Bluemix Dashboard, click the **Conversation** service instance that you created in 2.1.1, “Creating a Watson Conversation service instance” on page 14, which is listed under Services (Figure 3-3 on page 60).



NAME	SERVICE OFFERING	ACTIONS
Conversation	Conversation	⋮

Figure 3-3 Conversation service instance

3. Click **Launch tool** (Figure 3-4) to open the Conversation tool.

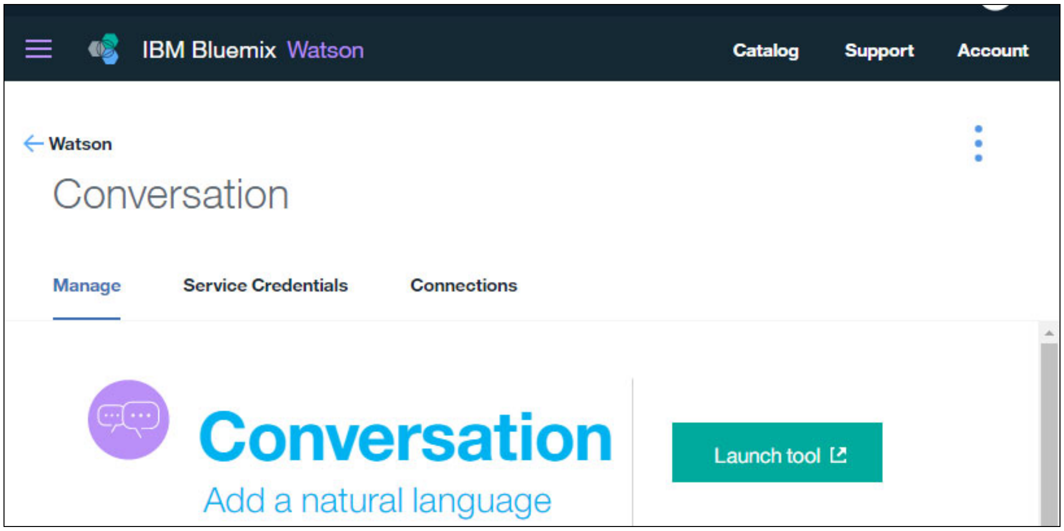
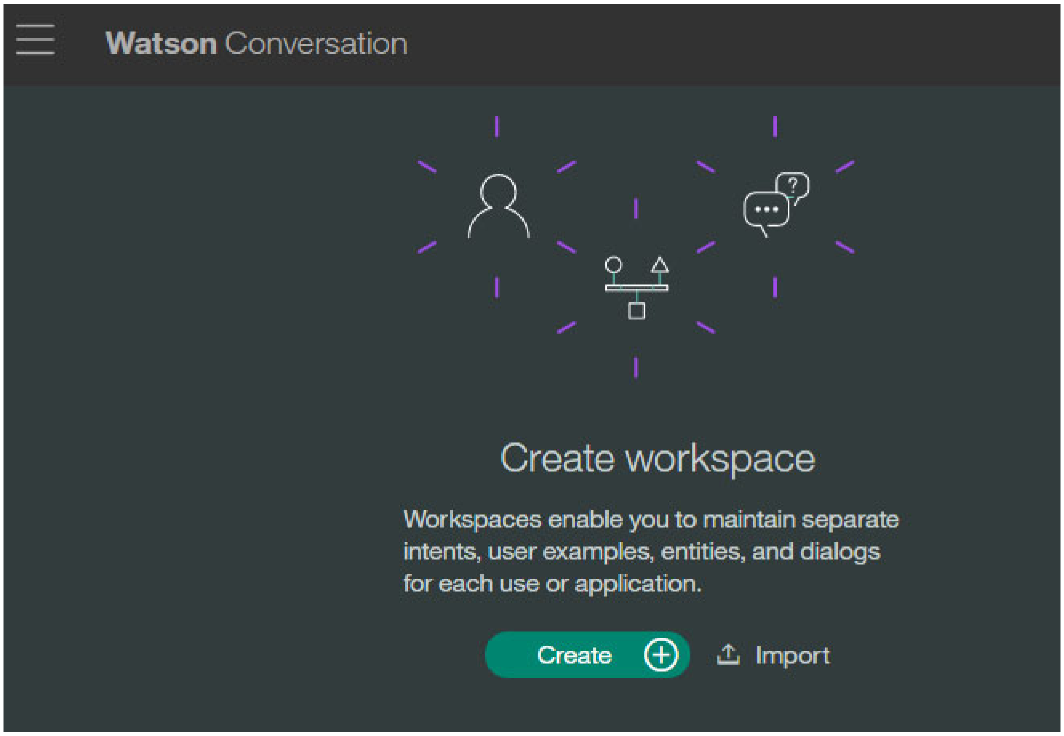


Figure 3-4 Launch Conversation tool

4. On the Watson Conversation dashboard, click **Create** to create a workspace (Figure 3-5).



Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

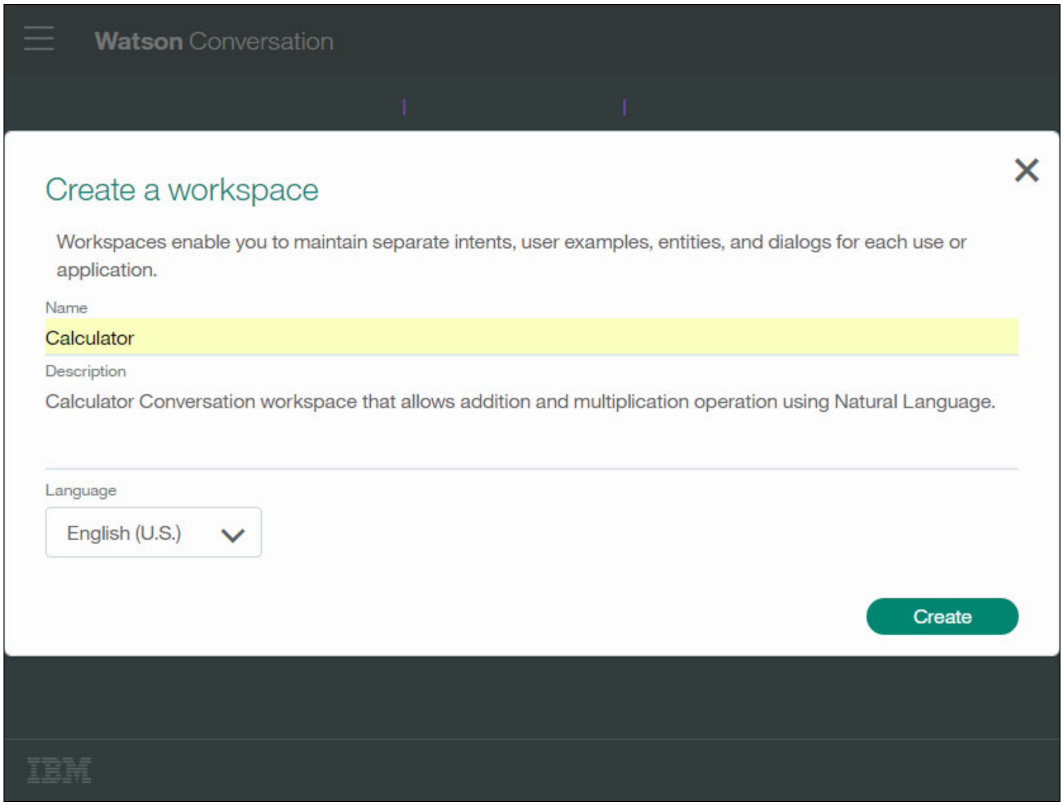
Figure 3-5 Watson Conversation Dashboard

5. In the Create a workspace window (Figure 3-6 on page 62), enter the following information and then click **Create**:

   – Name: `Calculator`

   – Description: `Calculator Conversation workspace that allows addition and multiplication operations using Natural Language.`

   – Language: `English (U.S.)`



Watson Conversation

Create a workspace

Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

Name
Calculator

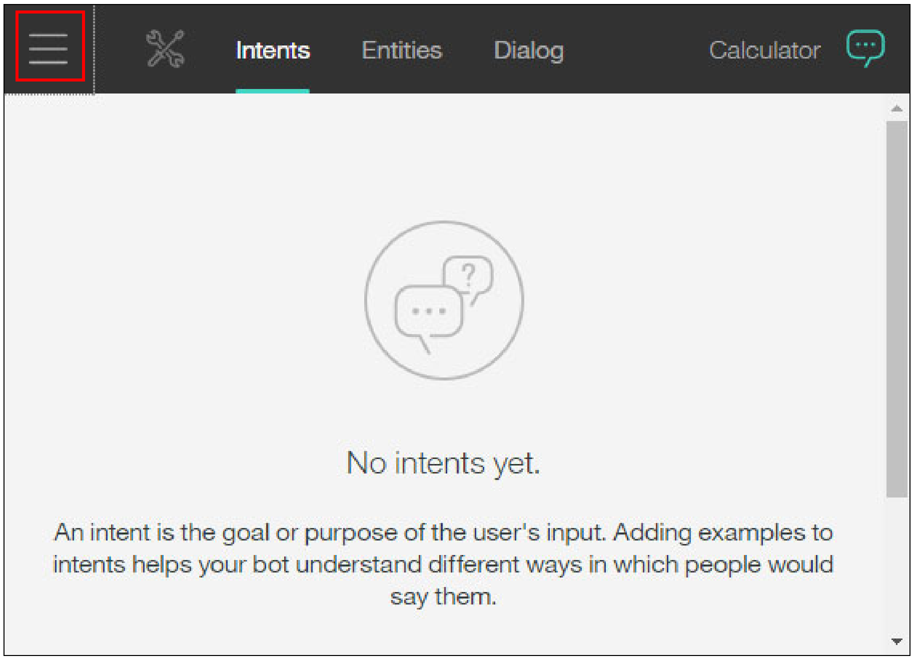
Description
Calculator Conversation workspace that allows addition and multiplication operation using Natural Language.

Language
English (U.S.)

Figure 3-6 Create the Calculator workspace

6. Get the Workspace ID so that you can configure your application to point to this workspace in step 1 on page 91:

   a. Click the **three horizontal bars** at the top-left corner (Figure 3-7).



No intents yet.

An intent is the goal or purpose of the user's input. Adding examples to intents helps your bot understand different ways in which people would say them.

Figure 3-7 Calculator workspace

b. Click **Back to workspaces** (Figure 3-8).

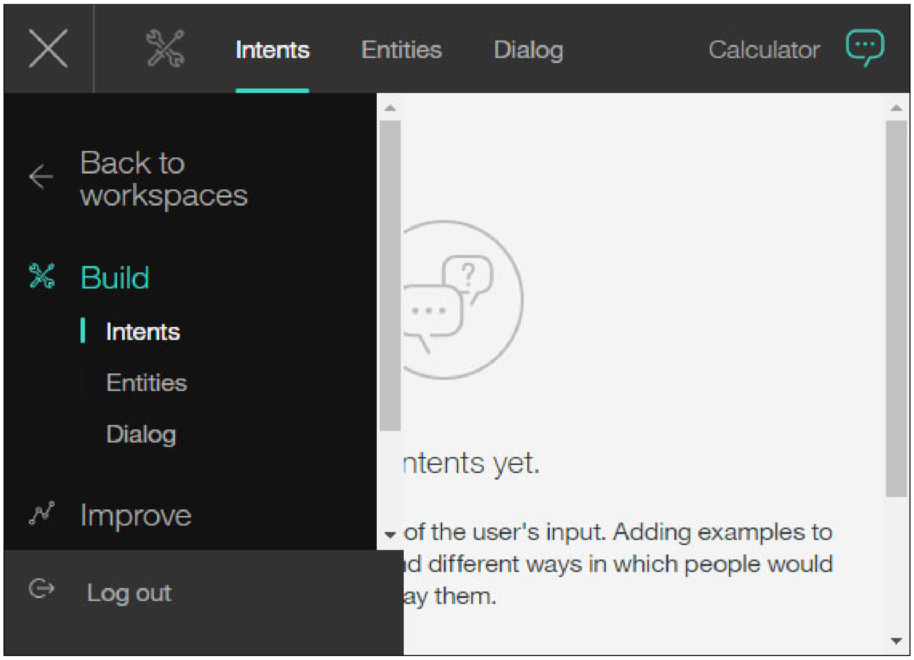


Figure 3-8 Calculator Conversation workspace

c. Click the **three vertical dots** at the top right of the Calculator box and then select **View details** (Figure 3-9).

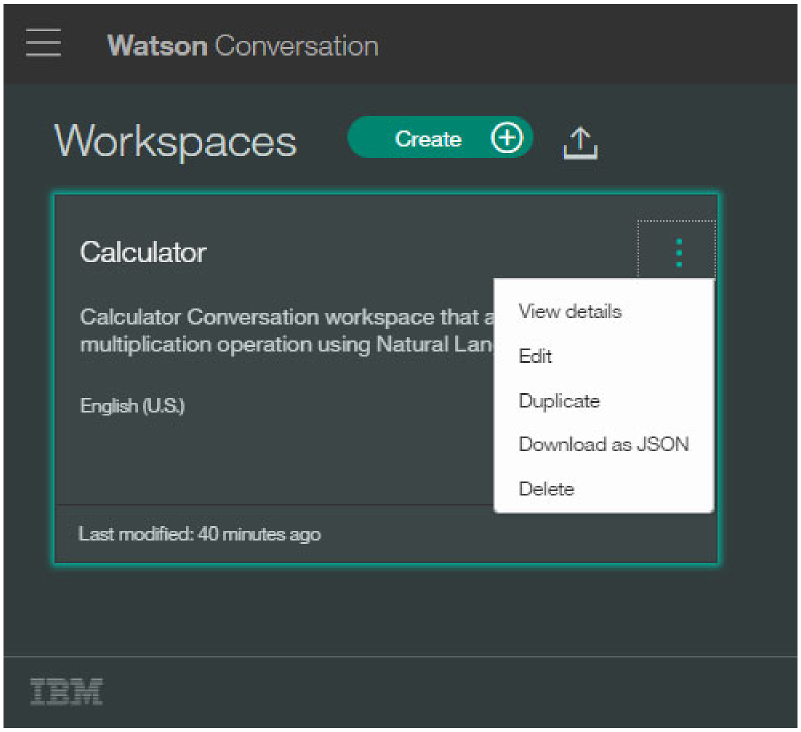
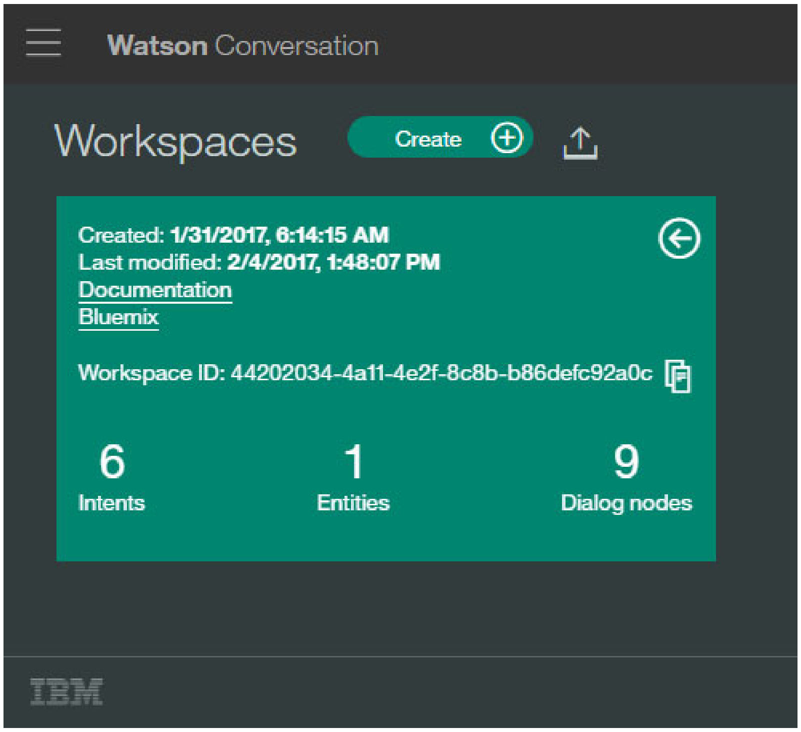

Watson Conversation
Workspaces
Create
Calculator
Calculator Conversation workspace that a
multiplication operation using Natural Lan
English (U.S.)
Last modified: 40 minutes ago
View details
Edit
Duplicate
Download as JSON
Delete
IBM

Figure 3-9 Calculator workspace menu

d. Copy the Workspace ID value and save it in a local text file (Figure 3-10). You will use the value of the Workspace ID in step 1 on page 91.



Created: 1/31/2017, 6:14:15 AM
Last modified: 2/4/2017, 1:48:07 PM
Documentation
Bluemix

Workspace ID: 44202034-4a11-4e2f-8c8b-b86defc92a0c

6 Intents
1 Entities
9 Dialog nodes

Figure 3-10 Workspace ID

Add intents

For the Conversation service to be able to understand the goal or purpose of the user's input in natural language, you must train the workspace with some examples for each intent. You will create an intent for the *addition* operation functionality and another intent for the *multiplication* operation functionality. Although you are required to train the workspace by providing a minimum of five examples of user input for each intent, to improve the accuracy, you should provide more than five examples.

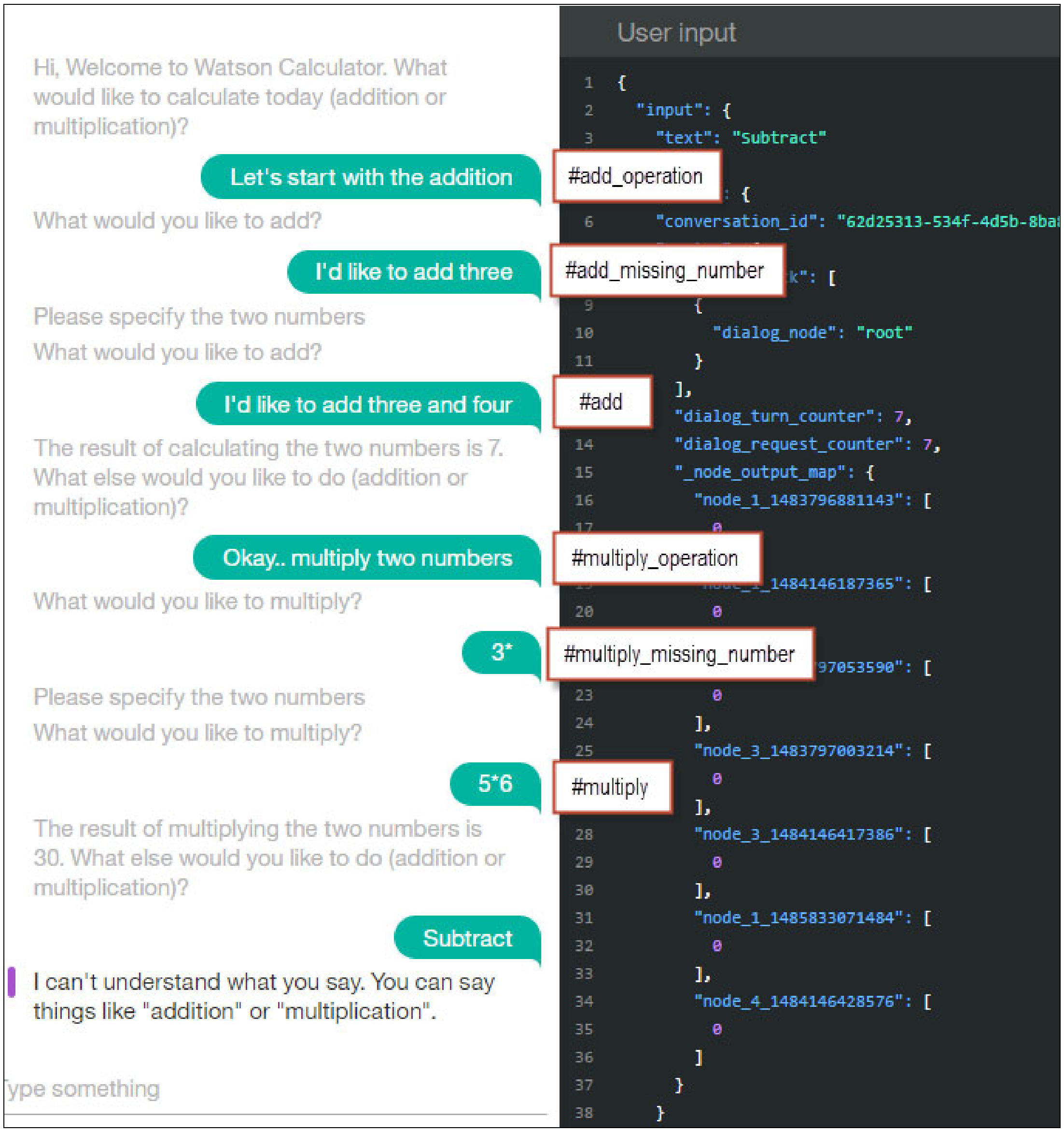
The steps in this section describe how you create the intents that are listed in Table 3-1.

Table 3-1 Intents to be created for the Calculator chatbot use case

Intent	Description
#add_operation	Identifies that the user wants to perform an addition operation. User examples: ► Add ► Addition ► Add Operation ► Sum ► Summation
#multiply_operation	Identifies that the user wants to perform a multiplication operation. User examples: ► Multiply ► Multiplication ► Multiply Operation ► I have two numbers and I'd like to multiply them ► Please help me multiply two numbers.

Intent	Description
#add	Identifies that the user provided two operands and wants to calculate the result of adding them. User examples: ► 3+2 ► 42534+52 ► calculate 4+6 ► five plus six equals? ► I’d like to add 3 and 4 ► tell me the results of adding eight and two ► three plus eleven ► what's the result of adding ten to fifteen? ► what's the sum of 1 and 5?
#add_missing_number	Identifies that the user provided only one operand for the addition. User examples: ► 3+ ► calculate 4+ ► calculate adding 76 ► I'd like to add 8 ► what's the sum of 2? ► would you please calculate adding six to the result?
#multiply	Identifies that the user provided two operands and wants to calculate the result of multiplying them. User examples: ► 2 * 6 ► 2 X 5 ► 3*5 ► 3x1 ► 5*53 ► 5 multiply 7 equals? ► 6*8 ► 9X2 ► 9 x 5 ► calculate 69*54 ► tell me the results of multiplying four and seven ► twenty multiply thirty
#multiply_missing_number	Identifies that the user provided only one operand for the multiplication. User examples: ► 2X ► 3* ► 5x ► 9 x ► multiply 6

Figure 3-11 shows a conversation between the user and the Calculator chatbot application, and shows how the Conversation service maps the user input in natural language to the corresponding intent configured in the Conversation workspace.



Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

Let's start with the addition

What would you like to add?

I'd like to add three

Please specify the two numbers

What would you like to add?

I'd like to add three and four

The result of calculating the two numbers is 7. What else would you like to do (addition or multiplication)?

Okay.. multiply two numbers

What would you like to multiply?

3*

Please specify the two numbers

What would you like to multiply?

5*6

The result of multiplying the two numbers is 30. What else would you like to do (addition or multiplication)?

Subtract

I can't understand what you say. You can say things like "addition" or "multiplication".

ype something

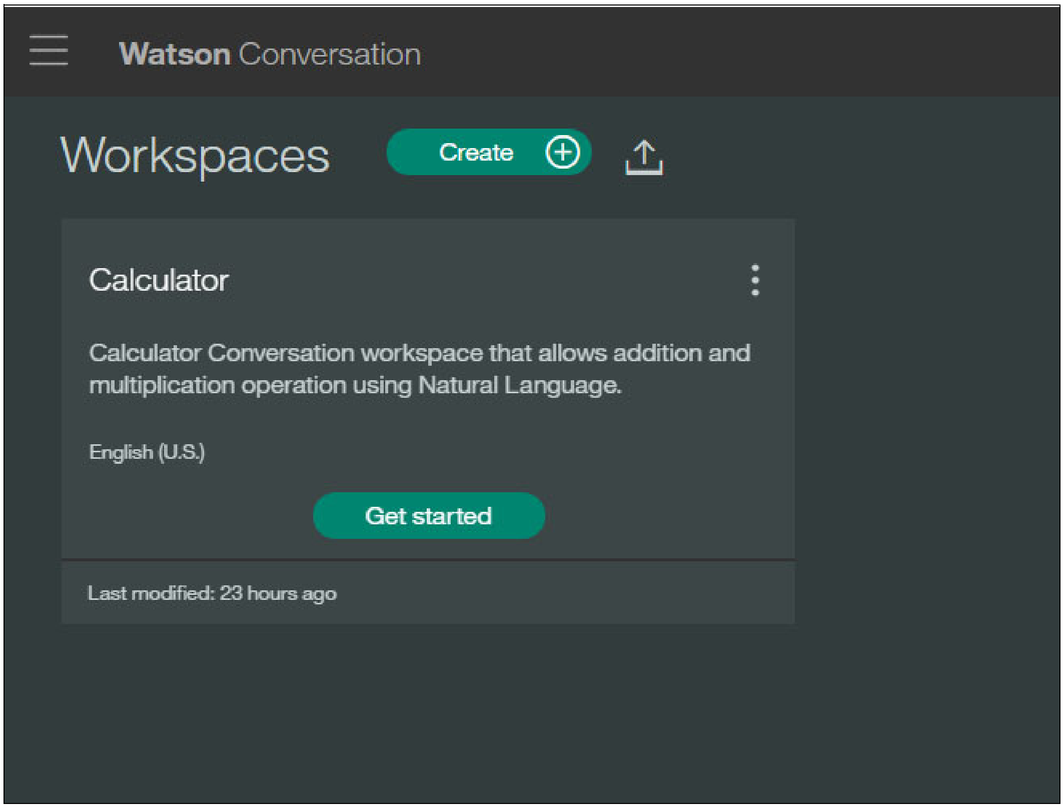
User input

```
{
  "input": {
    "text": "Subtract"
             : {
    "conversation_id": "62d25313-534f-4d5b-8ba
              k": [
      {
        "dialog_node": "root"
      }
    ],
    "dialog_turn_counter": 7,
    "dialog_request_counter": 7,
    "_node_output_map": {
      "node_1_1483796881143": [
        0
                _1484146187365": [
        0
                97053590": [
        0
      ],
      "node_3_1483797003214": [
        0
      ],
      "node_3_1484146417386": [
        0
      ],
      "node_1_1485833071484": [
        0
      ],
      "node_4_1484146428576": [
        0
      ]
    }
  }
```

Figure 3-11 Cognitive Calculator chatbot showing intents extracted from the user input

To add the intents that are listed in Table 3-1 on page 64 to the Calculator workspace, follow these steps:

1. Open the Calculator workspace (Figure 3-12).

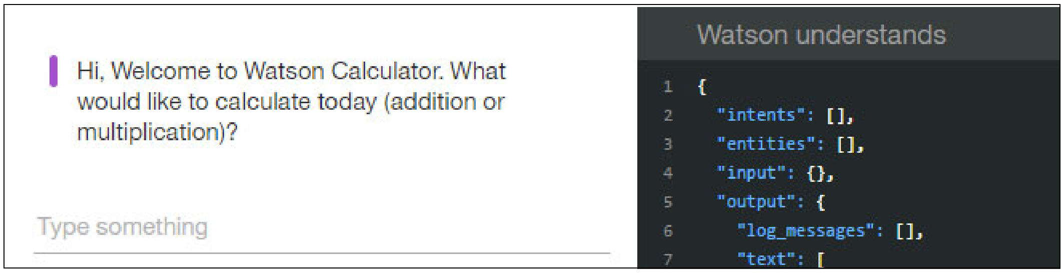


Calculator Conversation workspace that allows addition and multiplication operation using Natural Language.

Last modified: 23 hours ago

Figure 3-12 Conversation Workspaces

2. At the start of the conversation, the user specifies the mathematical operation to be performed, addition or multiplication, (Figure 3-13).



Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

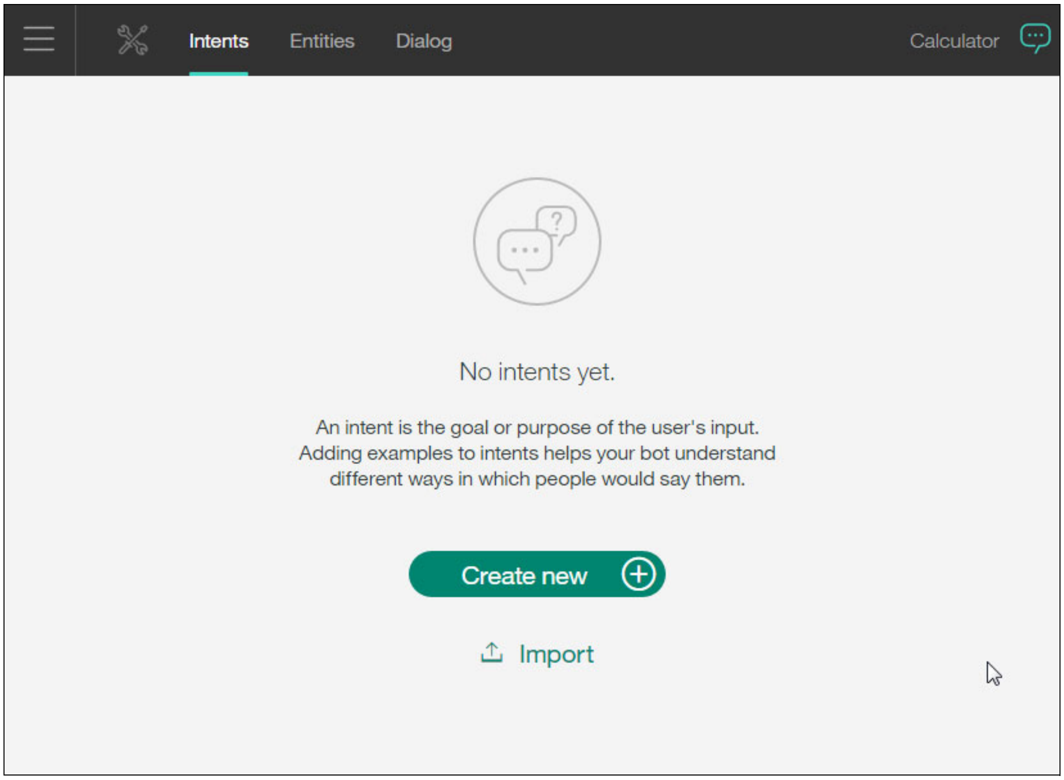
```
{
  "intents": [],
  "entities": [],
  "input": {},
  "output": {
    "log_messages": [],
    "text": [
```

Figure 3-13 Calculator chatbot

Create the intents that will enable the Conversation service to interpret the user input:

a. Create an intent for the addition operation capability:

   i. Click **Create new** to create new intent (Figure 3-14 on page 68).

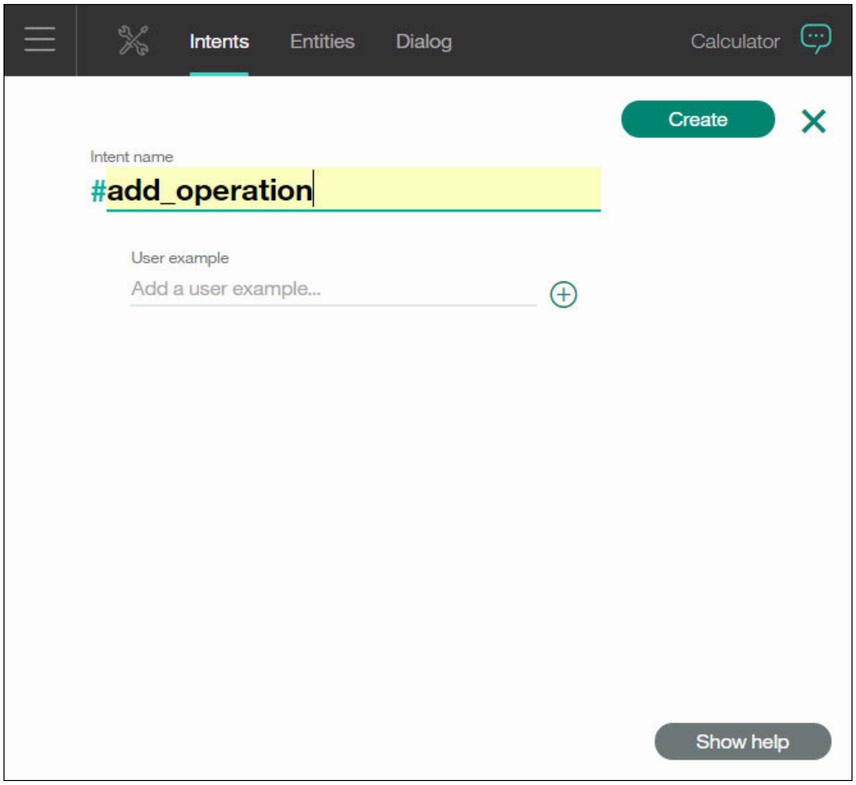


No intents yet.

An intent is the goal or purpose of the user's input. Adding examples to intents helps your bot understand different ways in which people would say them.

Figure 3-14 Create #add_operation intent (1 of 3)

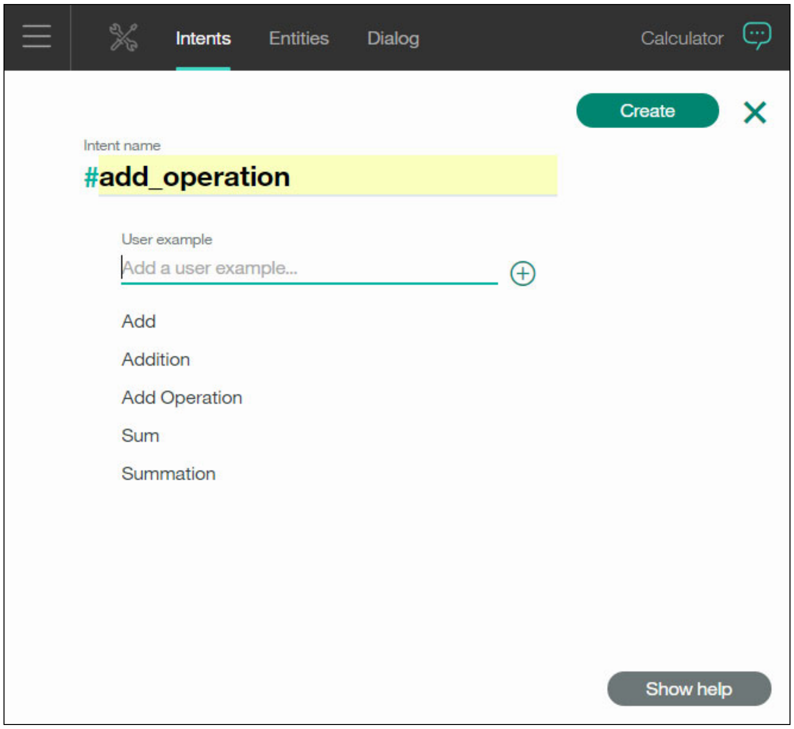
i. Type `add_operation` in the Intent name field (Figure 3-15).



Intent name
#add_operation

Figure 3-15 Create #add_operation intent (2 of 3)

ii. Add a minimum of five user examples for this intent (Figure 3-16), then click **Create**.



#add_operation

Add

Addition

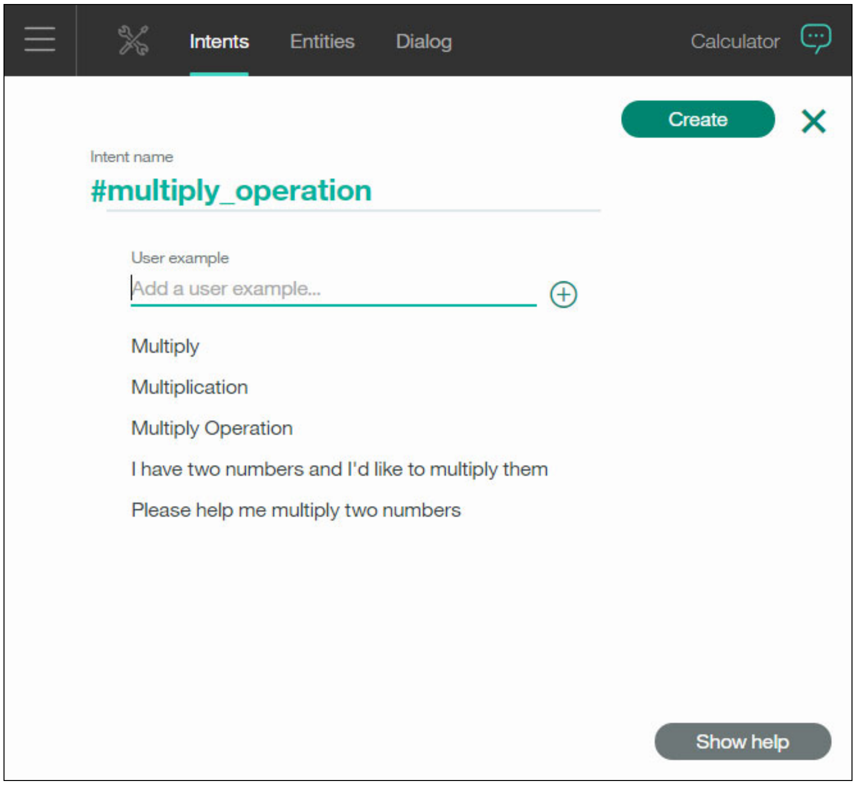
Add Operation

Sum

Summation

Figure 3-16 Create #add_operation intent: user examples (3 of 3)

b. Create a `multiply_operation` intent for the multiplication operation capability and provide user examples (Figure 3-17).



Intent name
#multiply_operation

User example
Add a user example...

Multiply

Multiplication

Multiply Operation

I have two numbers and I'd like to multiply them

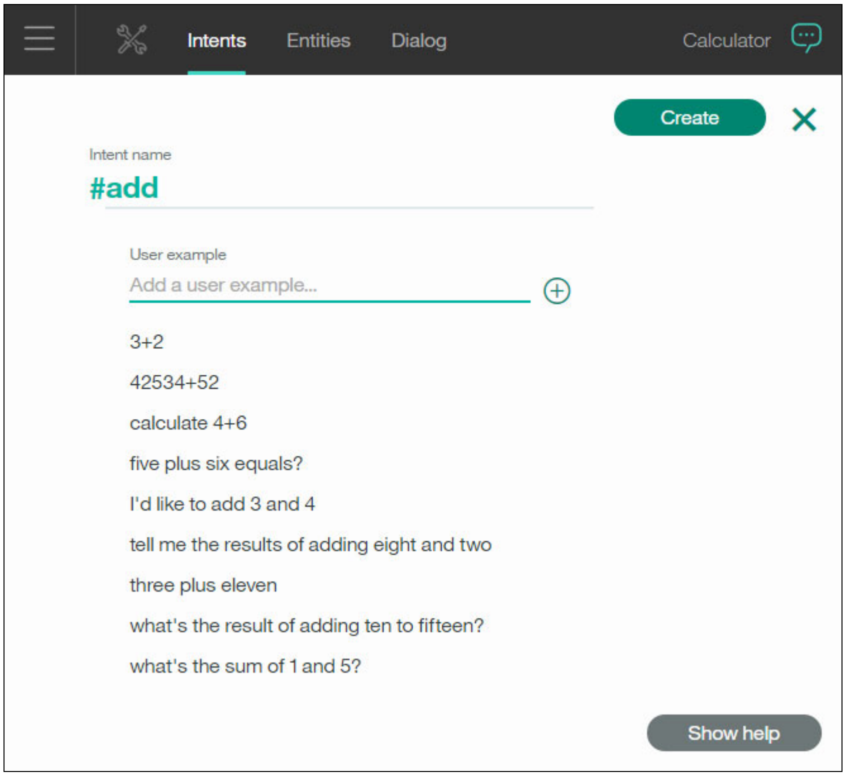
Please help me multiply two numbers

Figure 3-17 Create #multiply_operation intent with user examples

3. After the user requests the operation to be performed, the user specifies the actual addition or multiplication calculation. The Conversation service must be able to identify the intent of the user for addition or multiplication. The service must also be able to identify whether the user provides only one operand and respond accordingly.

So that the Conversation service can understand user inputs, create `add`, `multiply`, `add_missing_number`, and `multiply_missing_number` intents:

a. Create the `add` intent with the user examples (Figure 3-18).



#add

3+2

42534+52

calculate 4+6

five plus six equals?

I'd like to add 3 and 4

tell me the results of adding eight and two

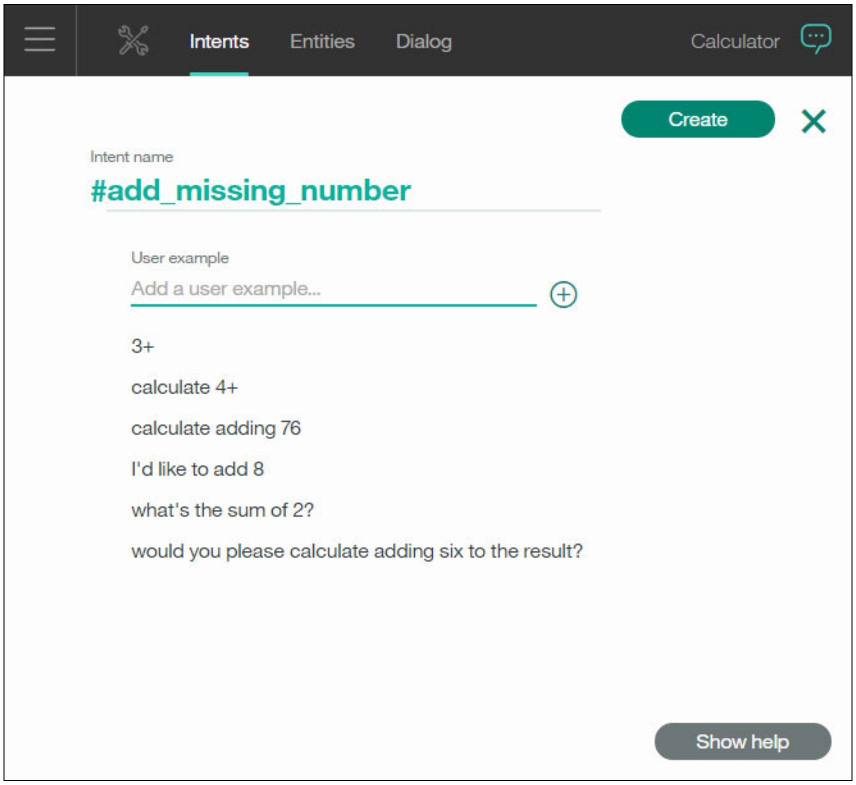
three plus eleven

what's the result of adding ten to fifteen?

what's the sum of 1 and 5?

Figure 3-18 Create #add intent and user examples

b. Create `add_missing_number` intent with the user examples (Figure 3-19).



#add_missing_number

3+

calculate 4+

calculate adding 76

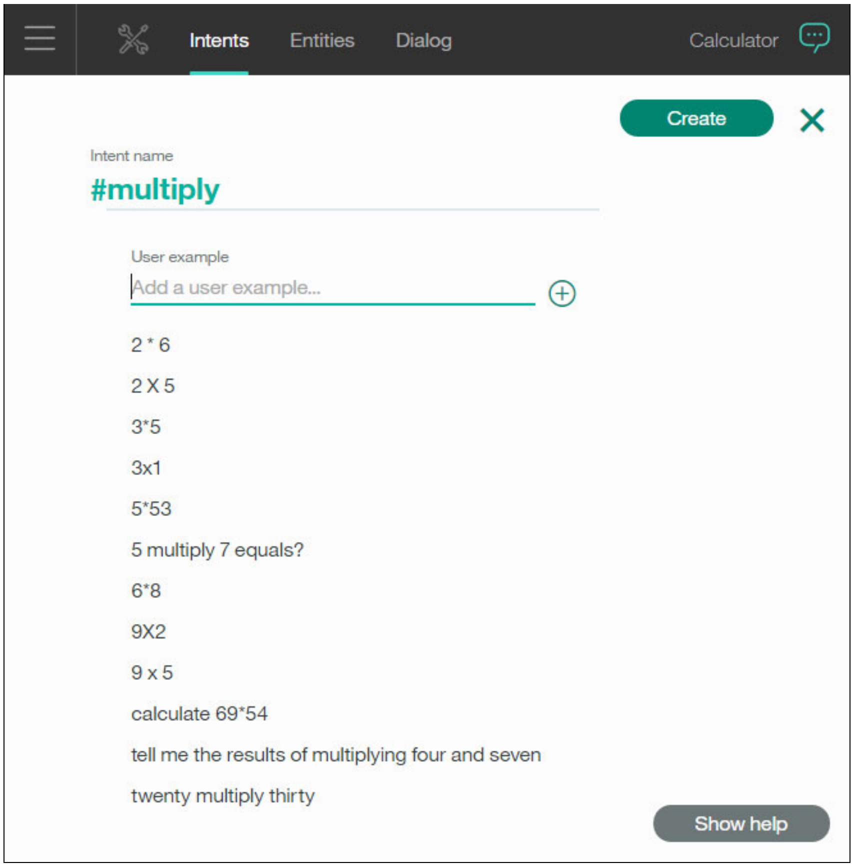
I'd like to add 8

what's the sum of 2?

would you please calculate adding six to the result?

Figure 3-19 Create #add_missing_number intent with user examples

c. Create the `multiply` intent with the user examples (Figure 3-20).



Intent name

#multiply

User example

Add a user example...

2 * 6

2 X 5

3*5

3x1

5*53

5 multiply 7 equals?

6*8

9X2

9 x 5

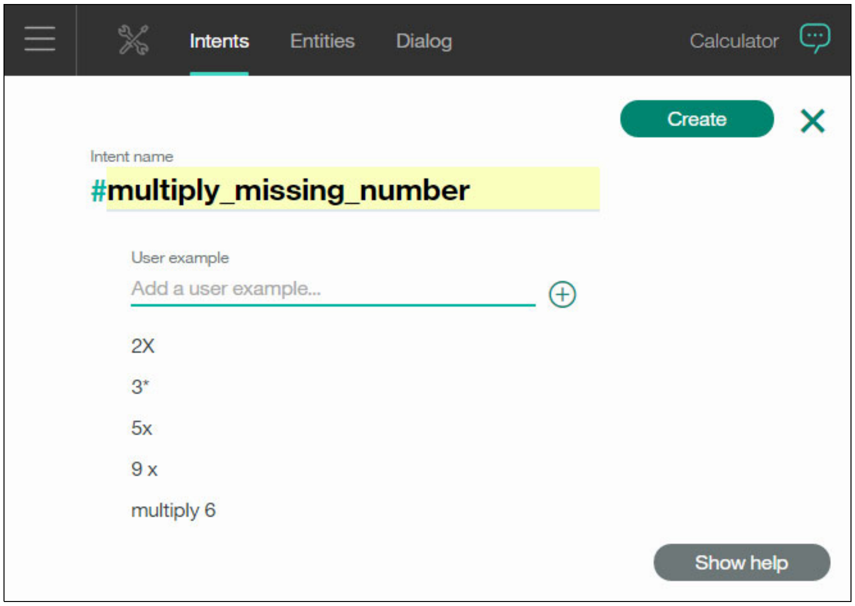
calculate 69*54

tell me the results of multiplying four and seven

twenty multiply thirty

Figure 3-20 Create #multiply intent with user examples

d. Create the `multiply_missing_number` intent with the user examples (Figure 3-21).



Intent name

#multiply_missing_number

User example

Add a user example...

2X

3*

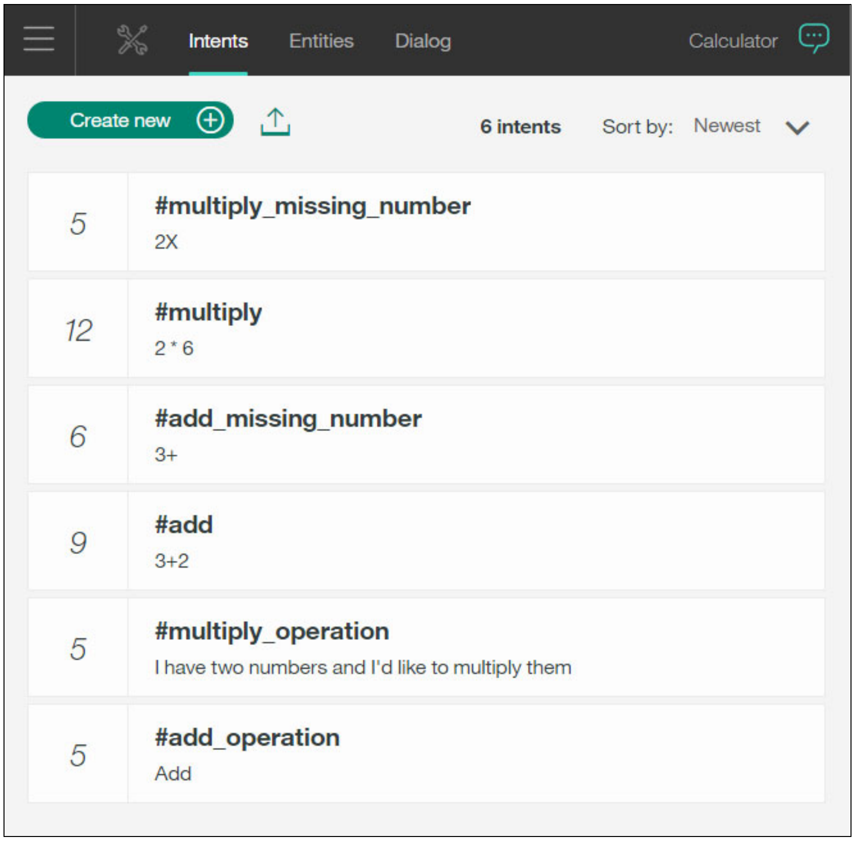
5x

9 x

multiply 6

Figure 3-21 Create #multiply_missing_number intent with user examples

Now, you have all the intents needed for the Cognitive Calculator chatbot (Figure 3-22).



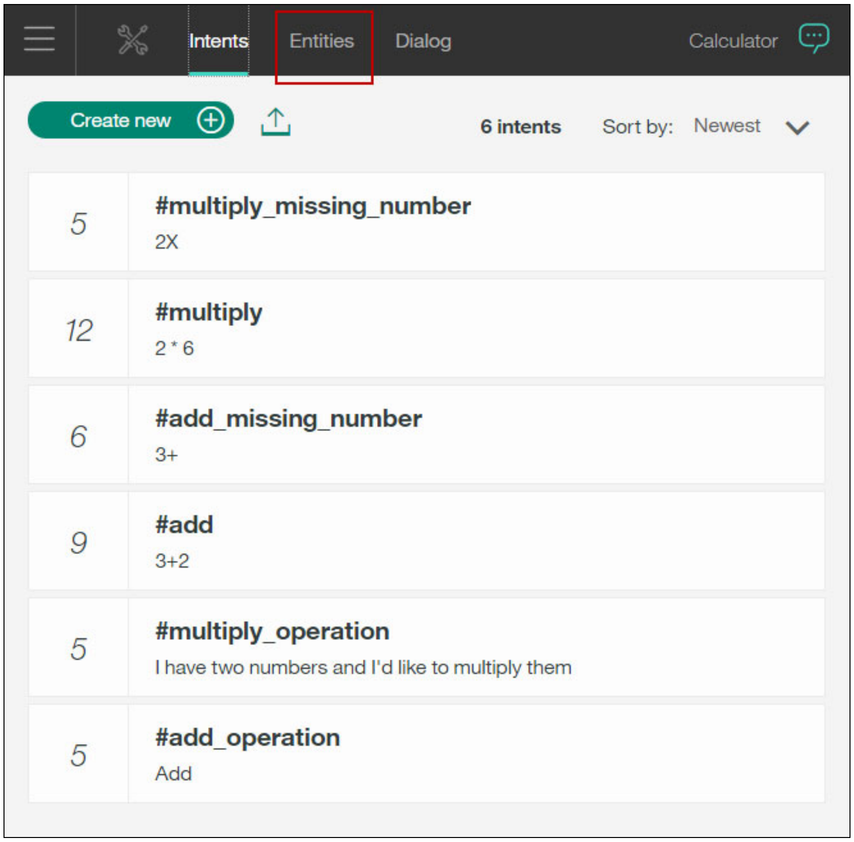
5	**#multiply_missing_number** 2X
12	**#multiply** 2 * 6
6	**#add_missing_number** 3+
9	**#add** 3+2
5	**#multiply_operation** I have two numbers and I'd like to multiply them
5	**#add_operation** Add

Figure 3-22 Calculator workspace intents

Add entities

You want the service to identify the operands of the addition and multiplication operations. The operands are numbers written as either digits (3, 64, 873, and so on) or text (one, two, eighty-seven, and so on). Use an available system entity that identifies the numbers:

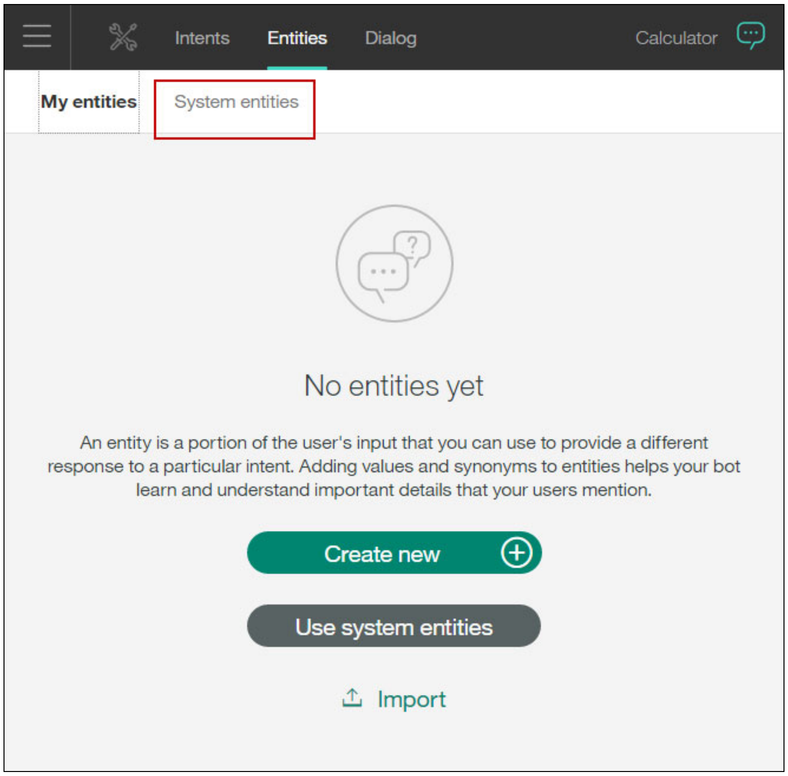
1. Click **Entities** on the top toolbar (Figure 3-23).



5	**#multiply_missing_number** 2X
12	**#multiply** 2 * 6
6	**#add_missing_number** 3+
9	**#add** 3+2
5	**#multiply_operation** I have two numbers and I'd like to multiply them
5	**#add_operation** Add

Figure 3-23 Calculator workspace: Adding entities

2. Click **System entities** (Figure 3-24).

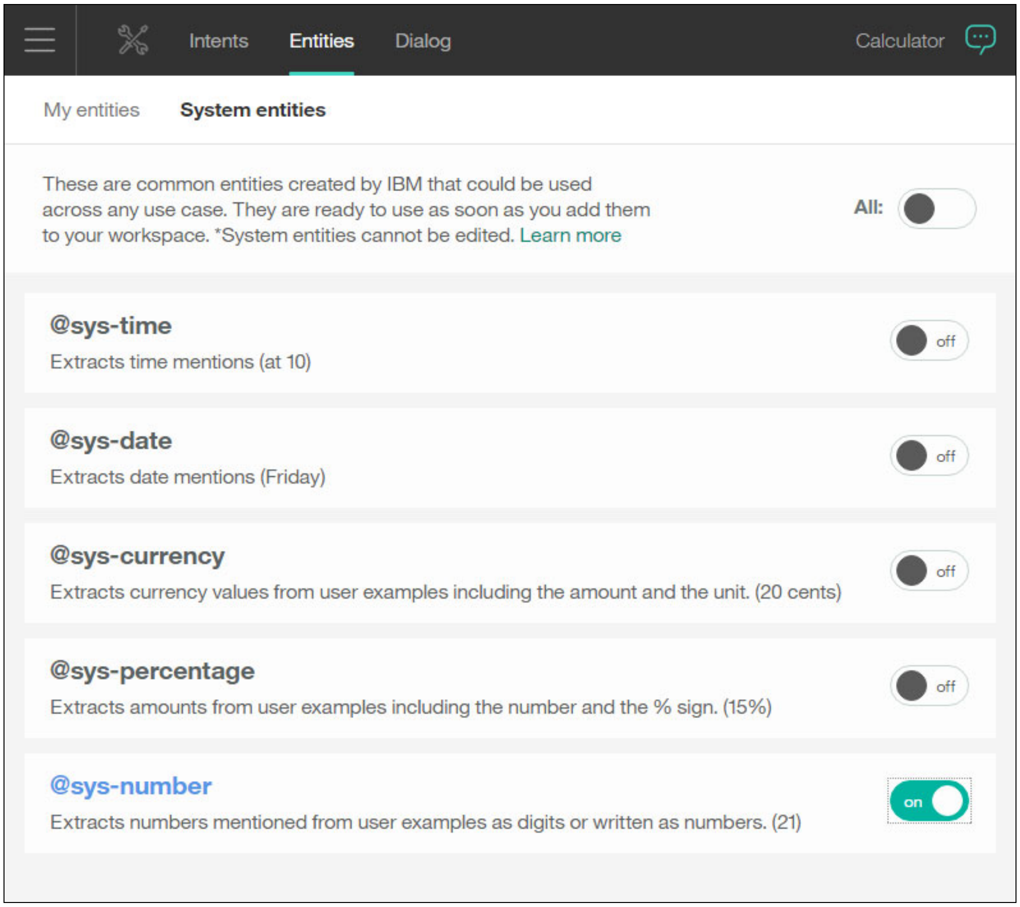


No entities yet

An entity is a portion of the user's input that you can use to provide a different response to a particular intent. Adding values and synonyms to entities helps your bot learn and understand important details that your users mention.

Figure 3-24 System entities

3. Switch the **off** toggle to the **on** position beside `@sys-number` to enable this system entity (Figure 3-25).



These are common entities created by IBM that could be used across any use case. They are ready to use as soon as you add them to your workspace. *System entities cannot be edited. Learn more

All: off

@sys-time
Extracts time mentions (at 10) — off

@sys-date
Extracts date mentions (Friday) — off

@sys-currency
Extracts currency values from user examples including the amount and the unit. (20 cents) — off

@sys-percentage
Extracts amounts from user examples including the number and the % sign. (15%) — off

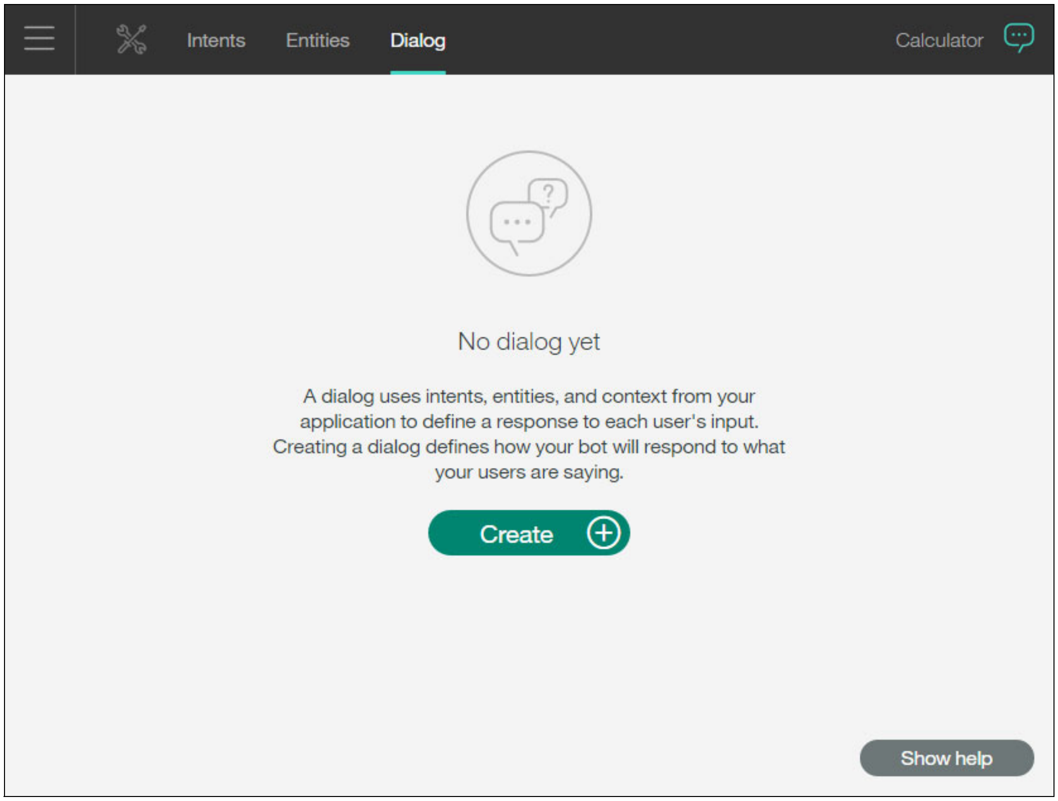
@sys-number
Extracts numbers mentioned from user examples as digits or written as numbers. (21) — on

Figure 3-25 System entities: Enable @sys-number

Create the dialog

Follow these steps:

1. Click **Dialog** in the top toolbar and click **Create** to create the dialog (Figure 3-26).

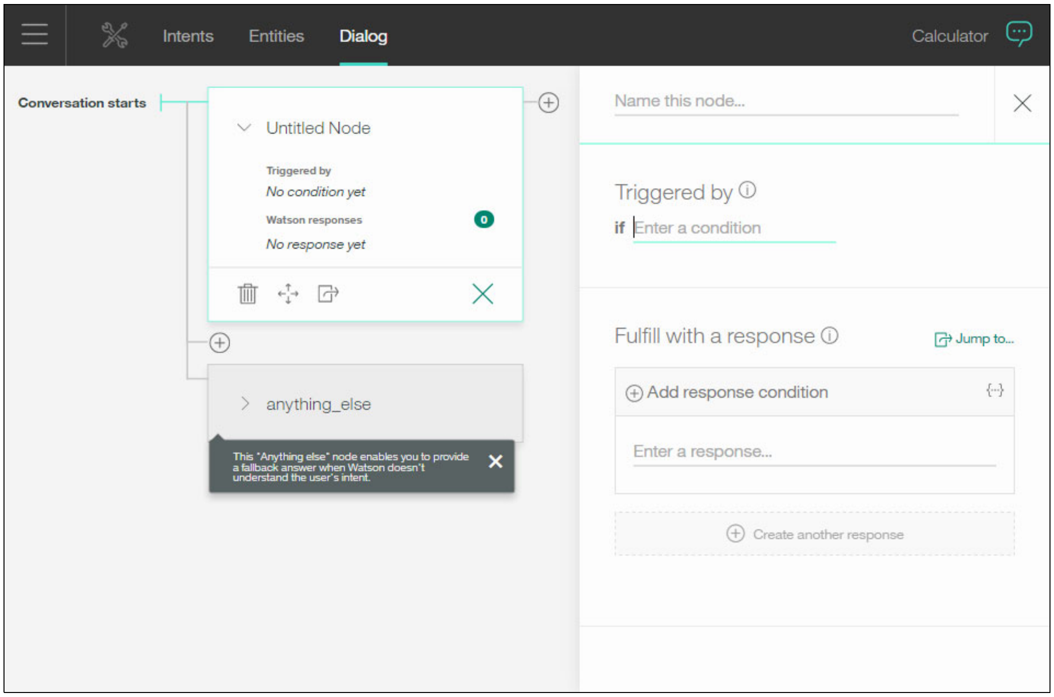


No dialog yet

A dialog uses intents, entities, and context from your application to define a response to each user's input. Creating a dialog defines how your bot will respond to what your users are saying.

Figure 3-26   Dialog

A default node is created (Figure 3-27).



This "Anything else" node enables you to provide a fallback answer when Watson doesn't understand the user's intent.

Figure 3-27 Dialog default base node

2. Under `Triggered by`, begin typing `conversation_start` and then select **conversation_start (create_new condition)**, as shown in Figure 3-28.

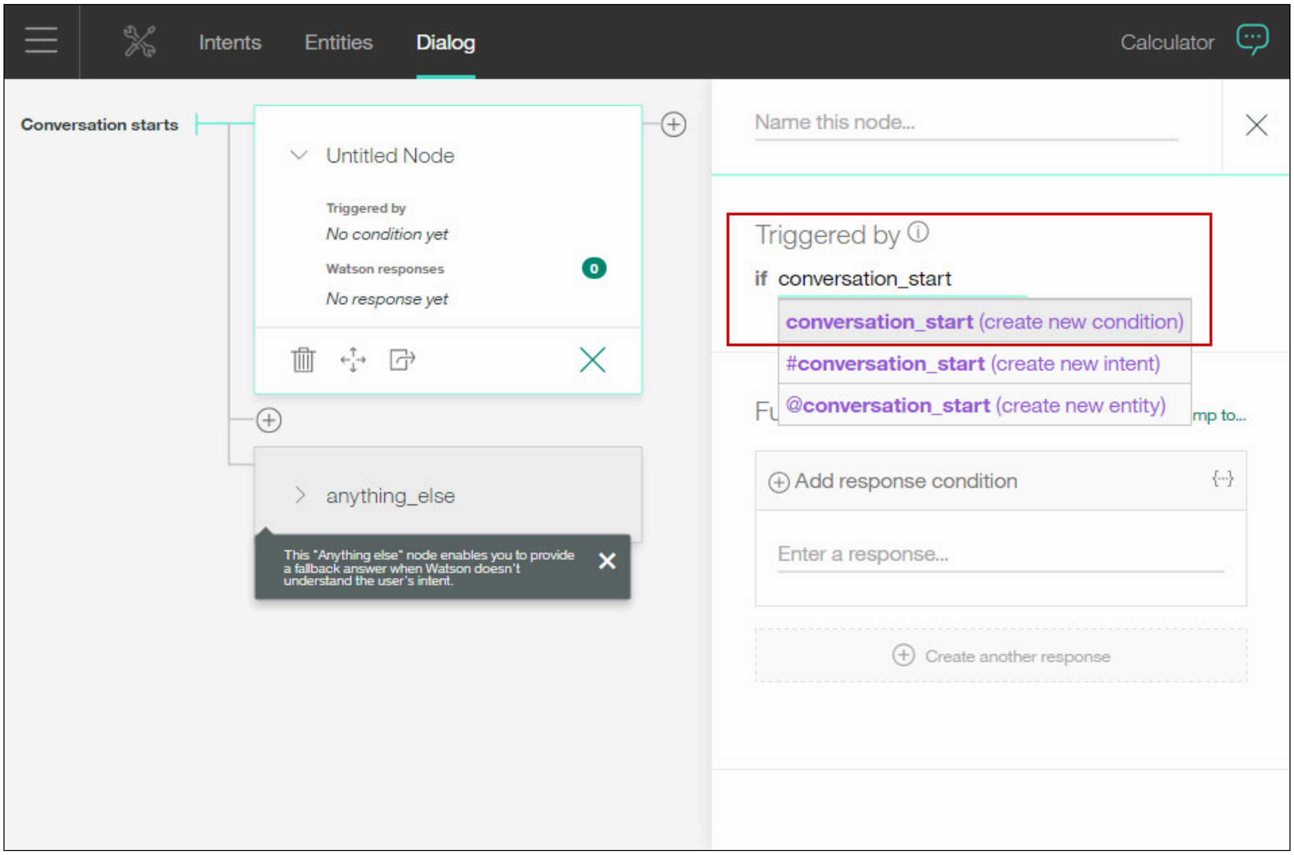


Figure 3-28 The conversation_start condition

3. Write the response that you want the chatbot to provide and then press **Enter** (Figure 3-29). In this case, you might want the chatbot to respond with this greeting: `Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?.`

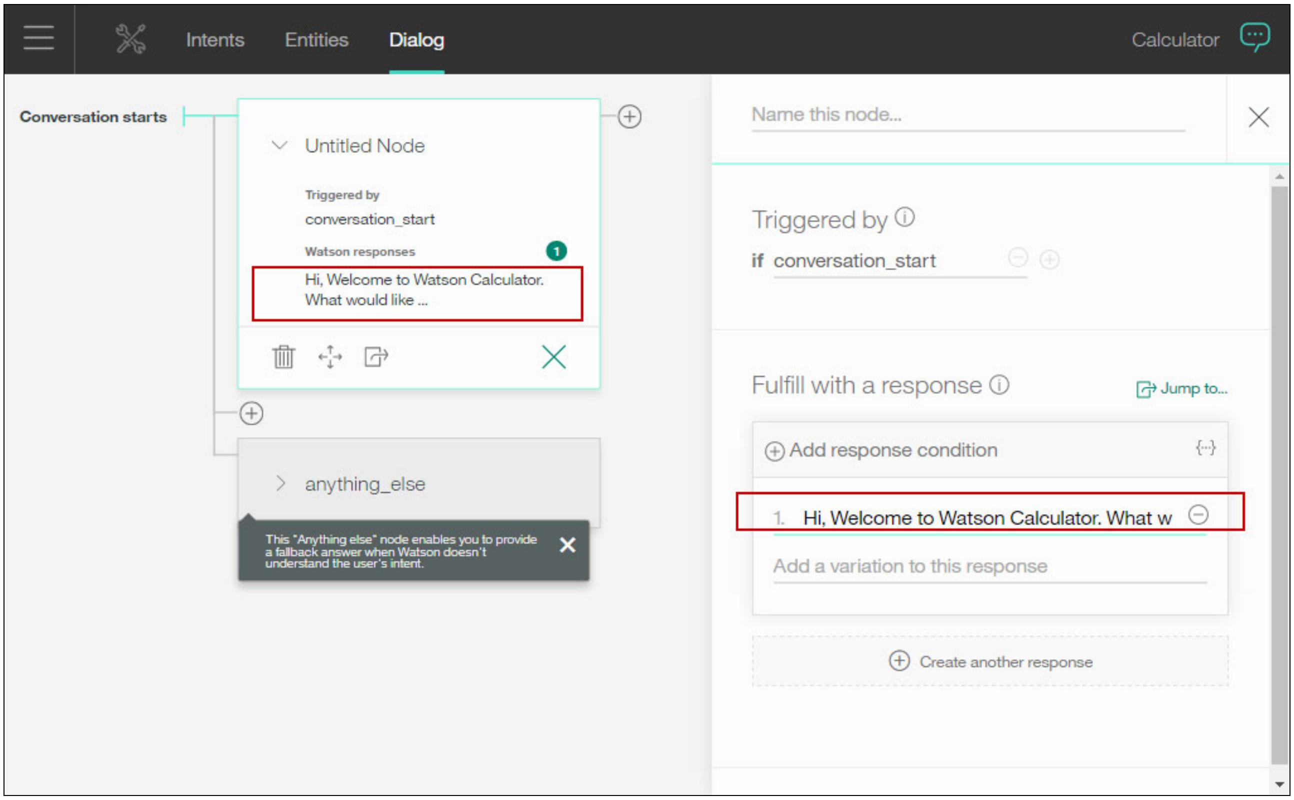
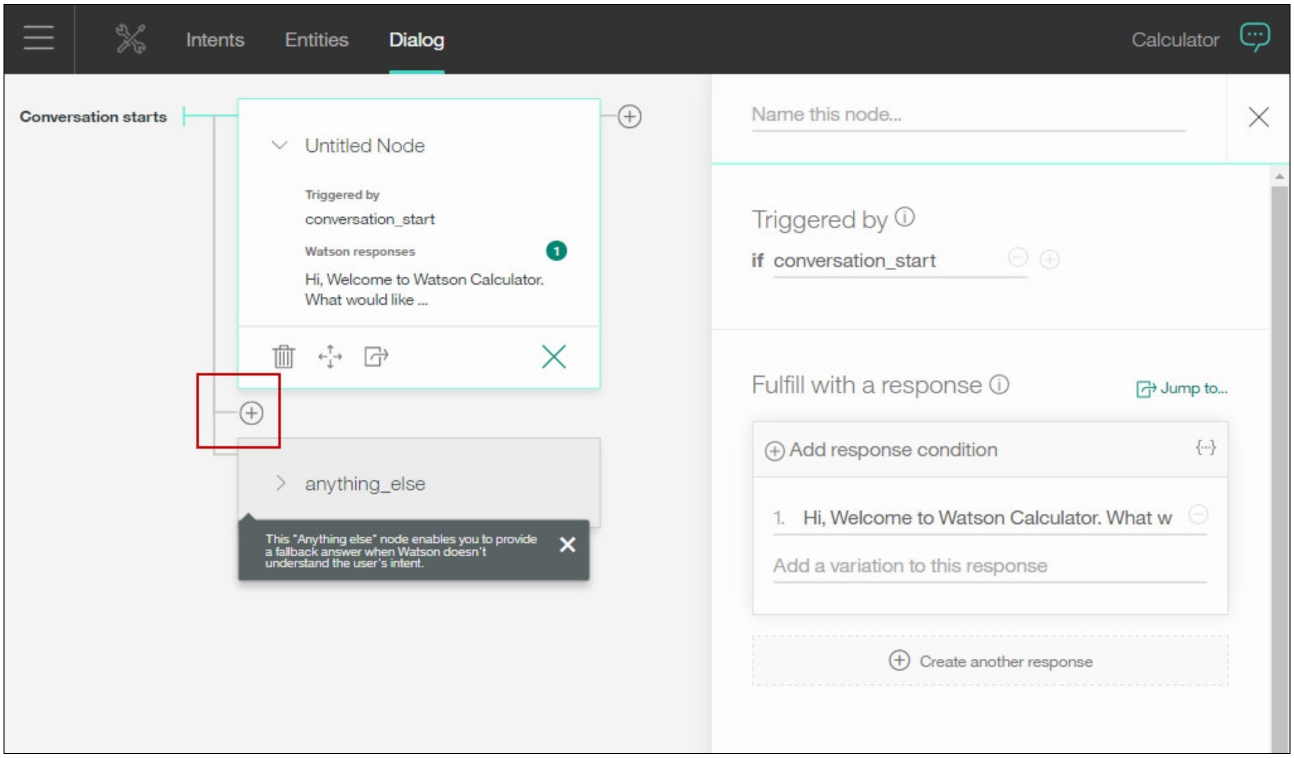


Figure 3-29 The conversation_start response

4. Click the plus sign (**+**) to create a new node (Figure 3-30).



conversation_start

Hi, Welcome to Watson Calculator. What would like ...

This "Anything else" node enables you to provide a fallback answer when Watson doesn't understand the user's intent.

if conversation_start

1. Hi, Welcome to Watson Calculator. What w

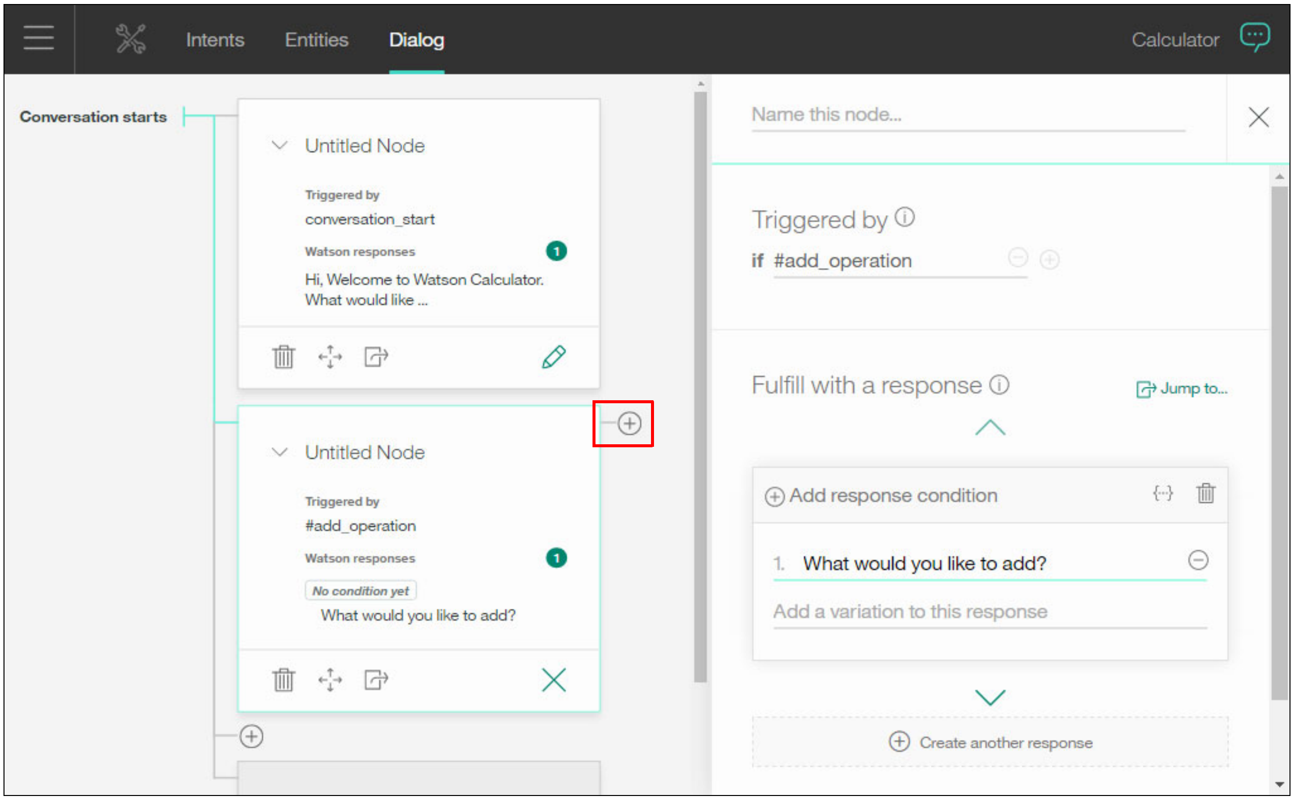
Figure 3-30 Create a new node

5. This node will be triggered when the user input is recognized as the `#add_operation` intent (Figure 3-31).

   Specify this information and then press Enter:

   a. Under `Triggered by`, start typing `#add_operation` and then select it from the autocomplete text box.

   b. Under `Add response condition`, type `What would you like to add?`



conversation_start

Hi, Welcome to Watson Calculator. What would like ...

#add_operation

What would you like to add?

if #add_operation

1. What would you like to add?

Figure 3-31 The add_operation node

6. Click the plus sign (**+**) on the right side of the node that you just created (see #add_operation Figure 3-31) to continue building the flow of the conversation.

7. This node will be triggered when the user input is recognized as an `#add` intent (Figure 3-32 on page 84).

   Specify this information and then press **Enter**:

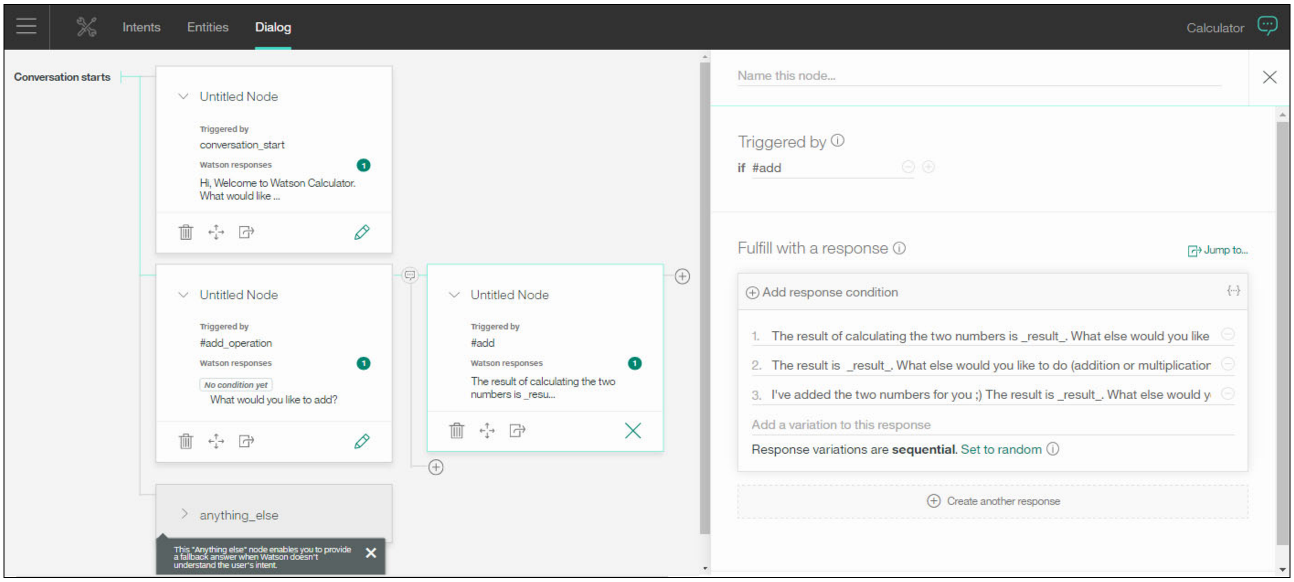
   a. Under `Triggered by`, type `#add`.

   b. You can let the Conversation service return various responses. Under `Add response condition`, provide the following responses. The `_result_` text is a placeholder that you will replace with the actual result after developing the application logic in the Node.js application.

      i. `The result of calculating the two numbers is _result_. What else would you like to do (addition or multiplication)?`

      ii. `The result is  _result_. What else would you like to do (addition or multiplication)?`

      iii. `I've added the two numbers for you;) The result is _result_. What else would you like to do (addition or multiplication)?`



if #add

1. The result of calculating the two numbers is _result_. What else would you like
2. The result is _result_. What else would you like to do (addition or multiplicatior
3. I've added the two numbers for you ;) The result is _result_. What else would y

Response variations are **sequential**. Set to random

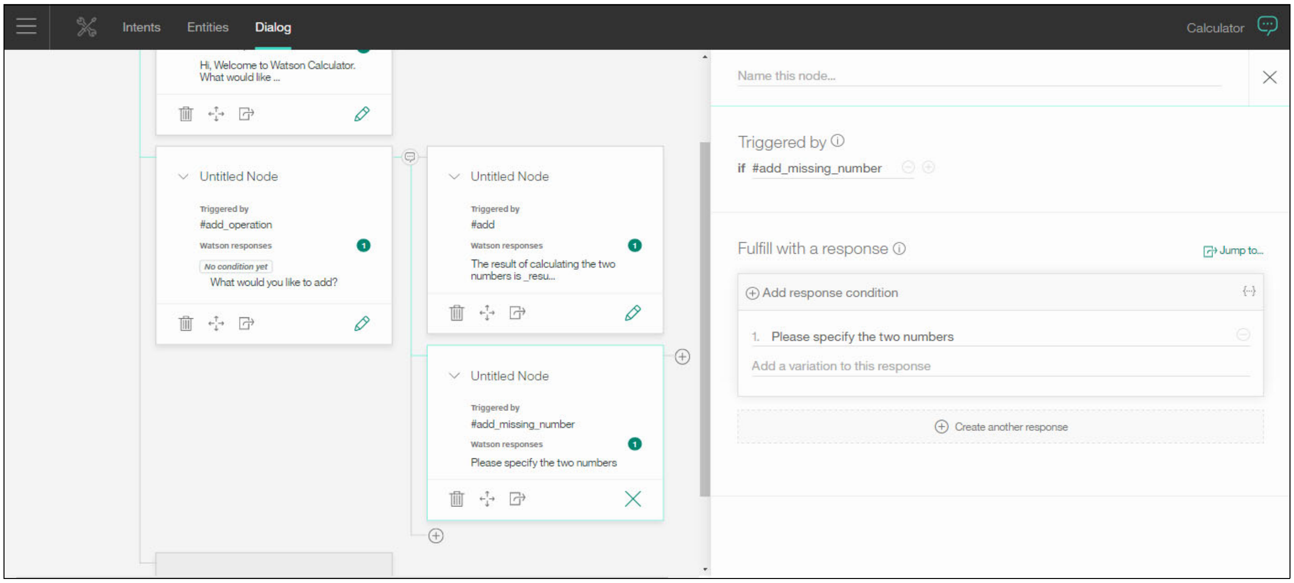
Figure 3-32 The add node

8. Click the plus sign (**+**) at the bottom of the node that you just created to create an alternative conversation.
9. This node will be triggered when the user input is recognized as an `#add_missing_number` intent (Figure 3-33).

   Specify this information and then press **Enter**:

   a. Under `Triggered by`, type `#add_missing_number`.

   b. Under `Add response condition`, enter `Please specify the two numbers`.



if #add_missing_number

1. Please specify the two numbers

Figure 3-33 The add_missing_number node

10. In case of a missing number, the chatbot should return to the user the response `What would you like to add?` Then, allow the user to try again. To accomplish this, click the **Jump to** icon in the node (Figure 3-34).

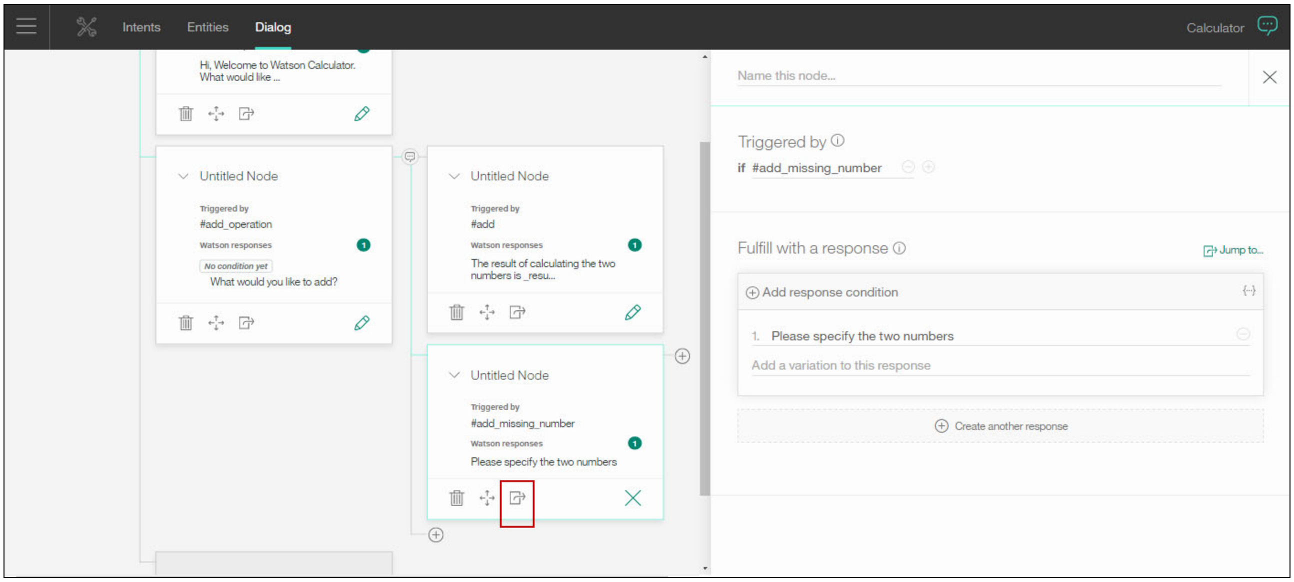


Figure 3-34 Click Jump to icon

11. Click the **#add_operation** node and then click **Go to response** (Figure 3-35).

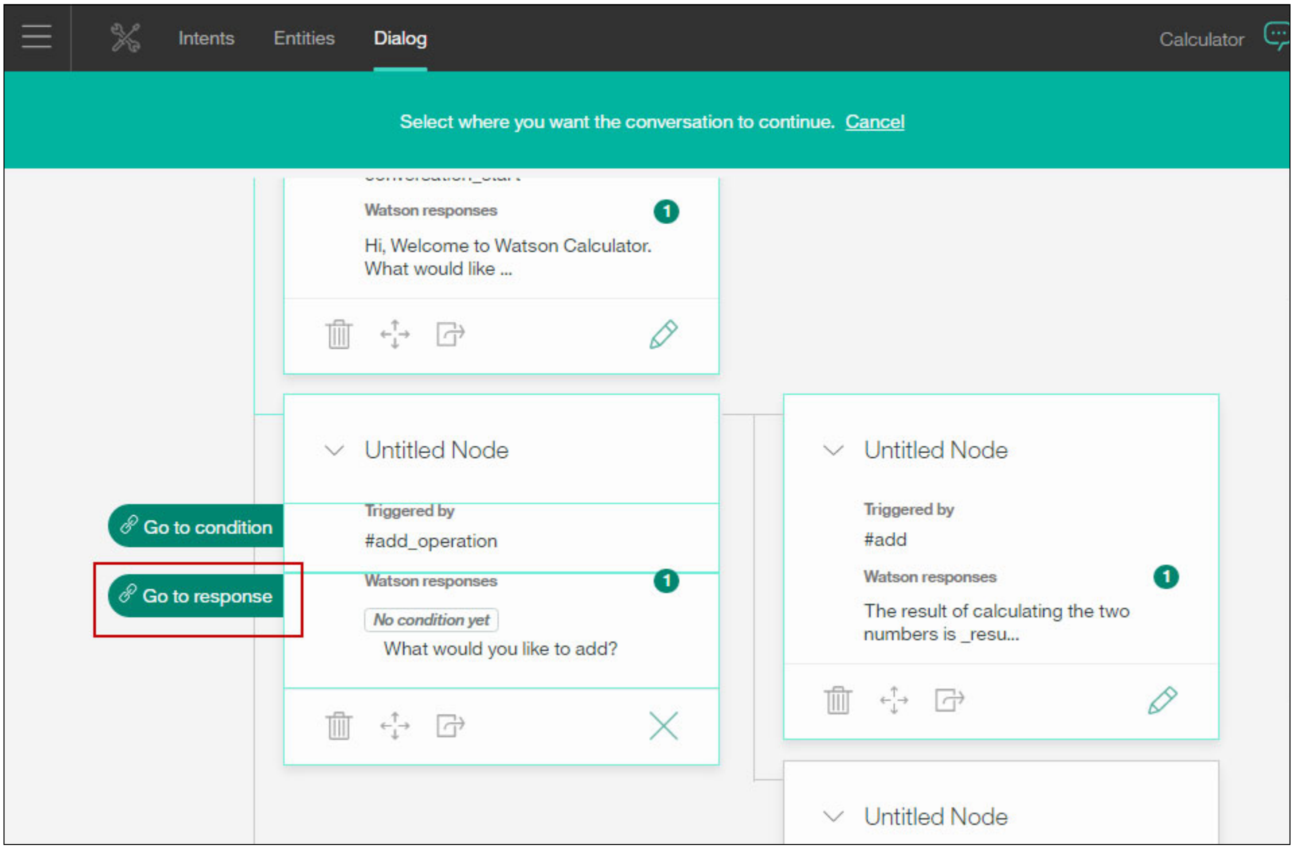
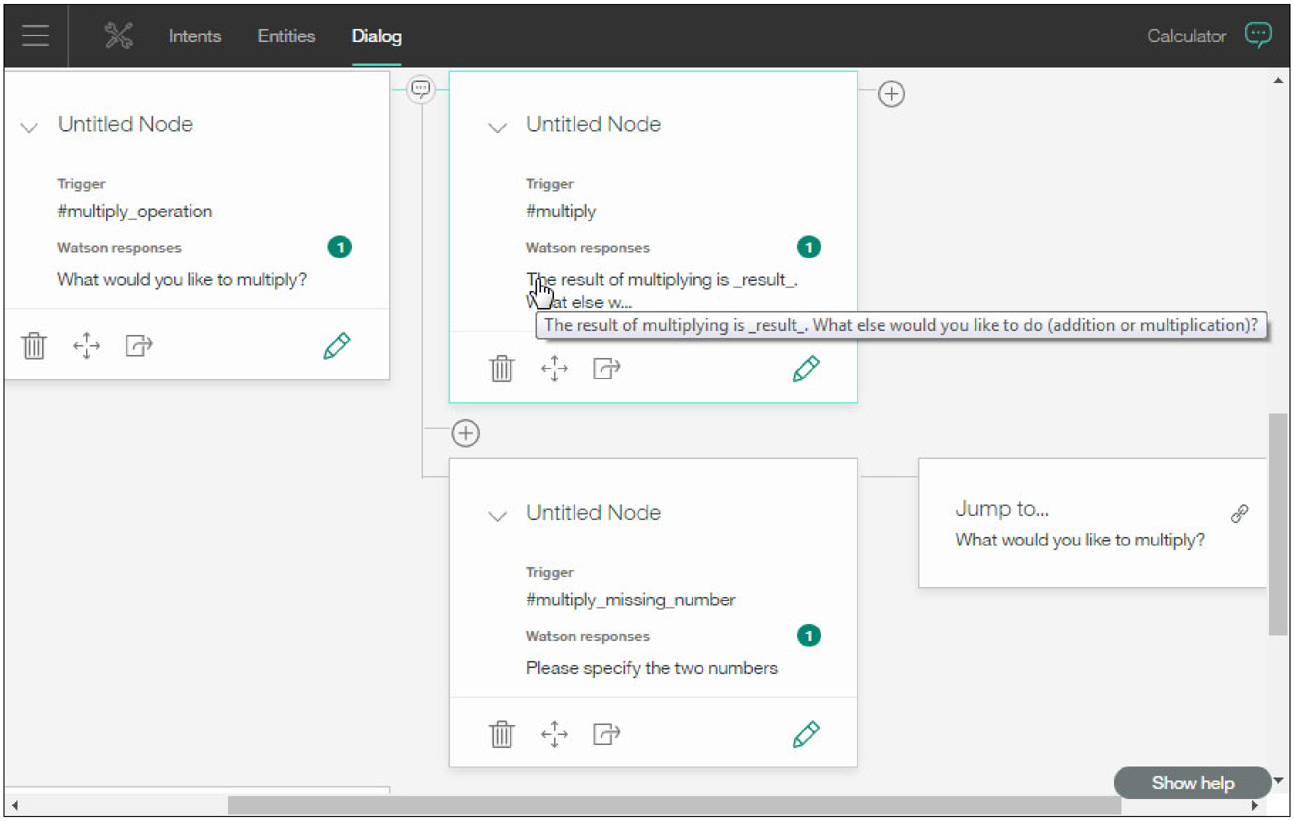


Figure 3-35 Go to another node response

12. Similarly create the nodes to handle the multiplication conversation flow (Figure 3-36).



Untitled Node
Trigger
#multiply_operation
Watson responses 1
What would you like to multiply?

Untitled Node
Trigger
#multiply
Watson responses 1
The result of multiplying is _result_. What else w...
The result of multiplying is _result_. What else would you like to do (addition or multiplication)?

Untitled Node
Trigger
#multiply_missing_number
Watson responses 1
Please specify the two numbers

Jump to...
What would you like to multiply?

Figure 3-36 Multiplication conversation flow

13.Edit the response in the `anything_else` node (Figure 3-37) to be:

```
I can't understand what you say. You can say things like "addition" or
"multiplication".
```

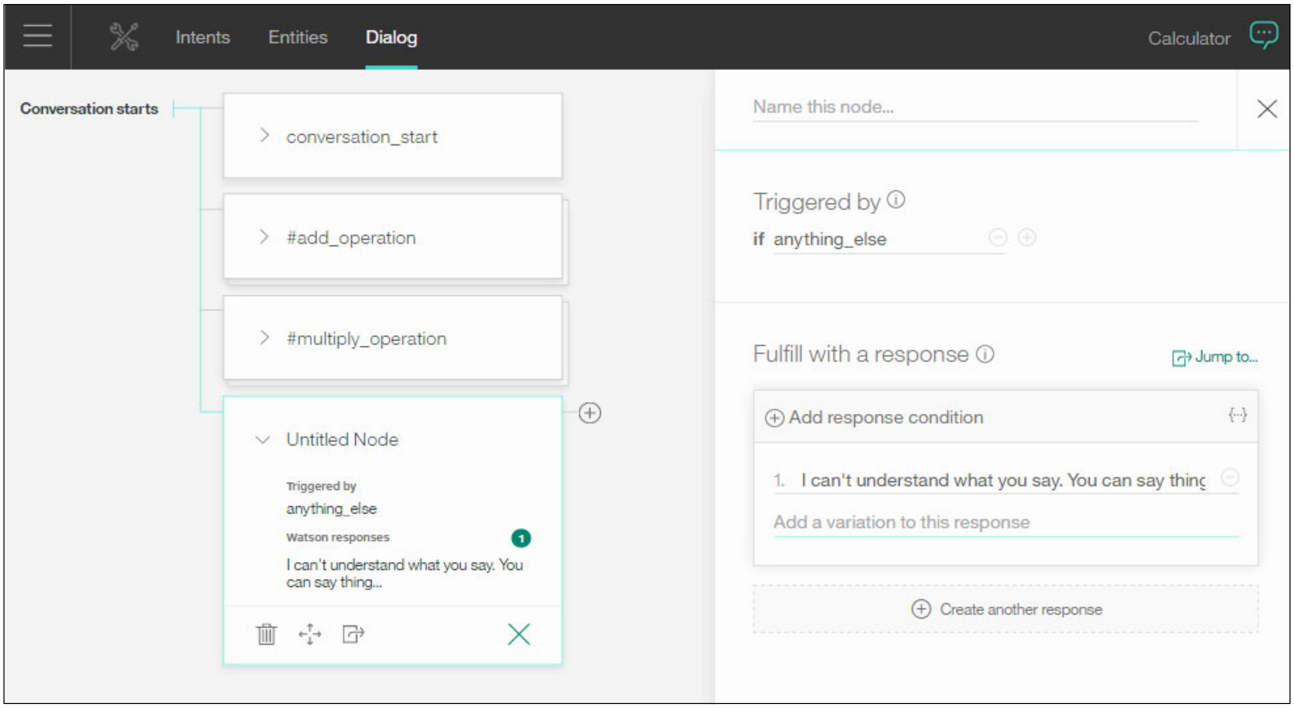


Figure 3-37 The anything_else node

Test the conversation flow

Follow these steps:

1. Click the **Ask Watson** icon at the top right (Figure 3-38).

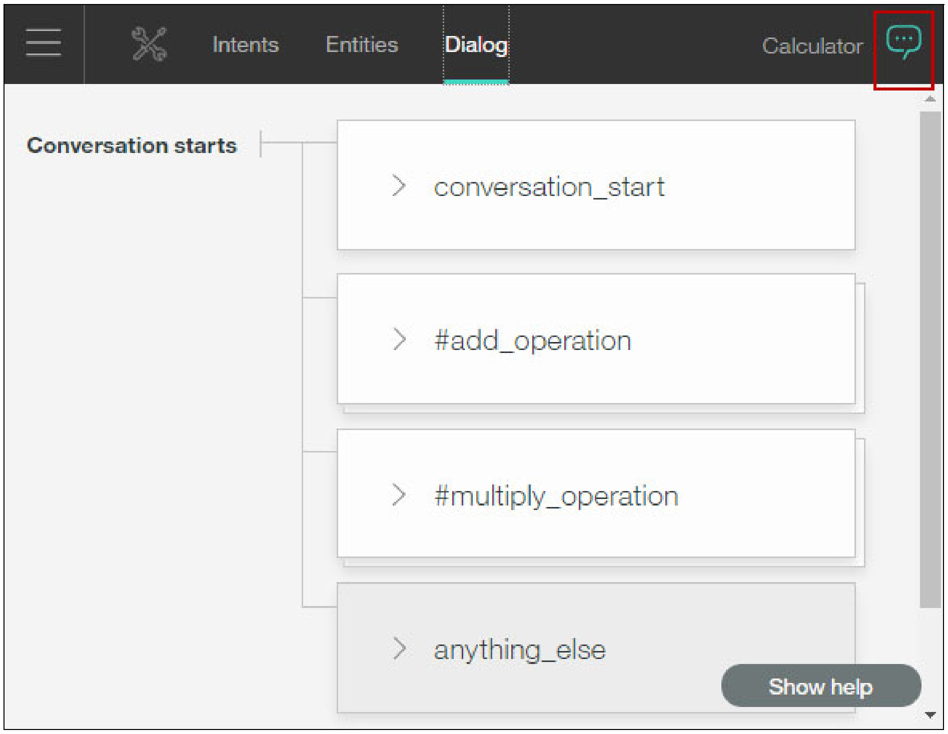
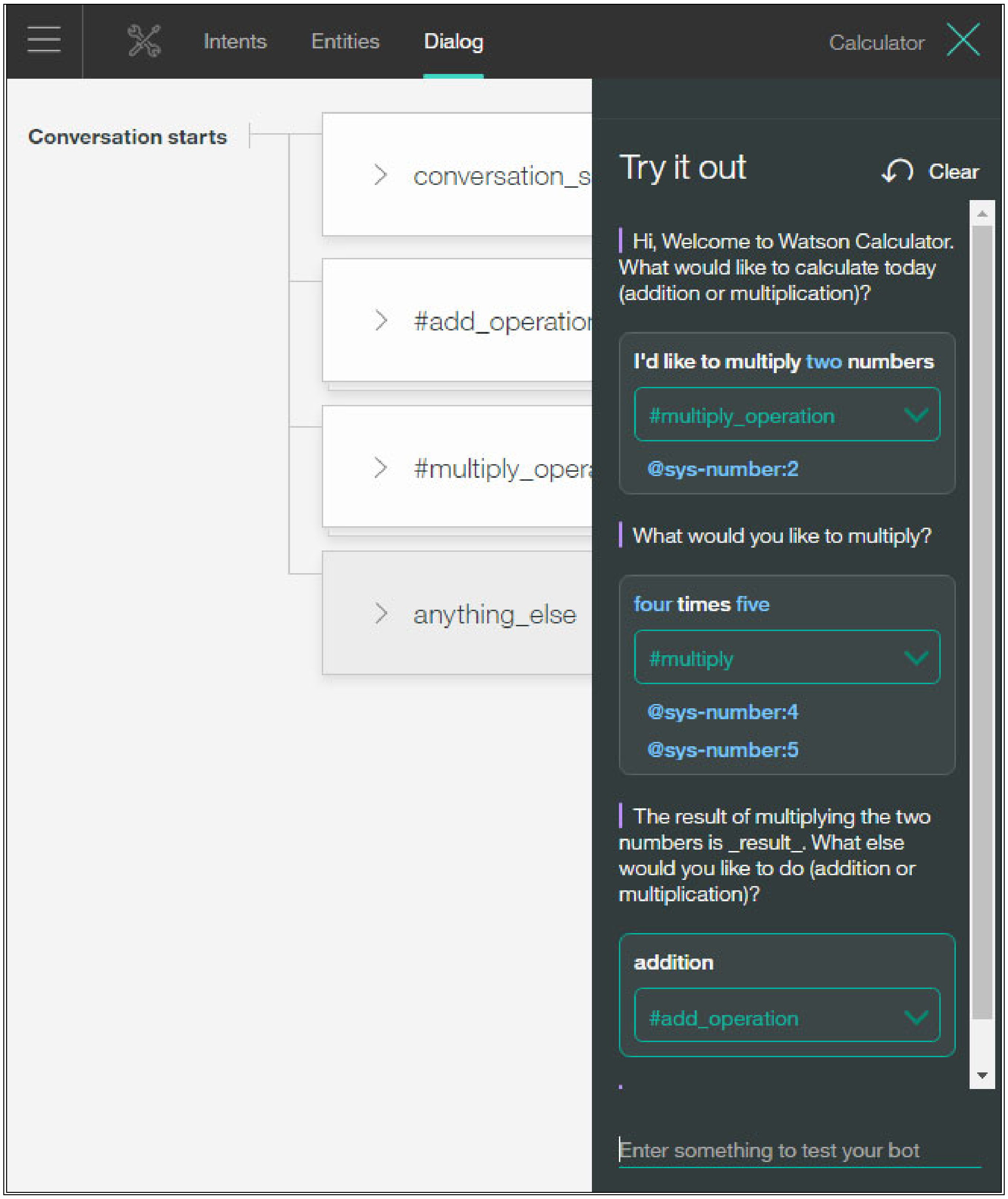


Figure 3-38 Calculator Conversation workspace

2. Test the dialog. For each user input, the Conversation service analyzes intents and entities and responds according to the conversation flow in the dialog (Figure 3-39).



Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

I'd like to multiply two numbers

#multiply_operation

@sys-number:2

What would you like to multiply?

four times five

#multiply

@sys-number:4

@sys-number:5

The result of multiplying the two numbers is _result_. What else would you like to do (addition or multiplication)?

addition

#add_operation

Enter something to test your bot

Figure 3-39 Testing the dialog

3.4.3 Developing the Cognitive Calculator chatbot application

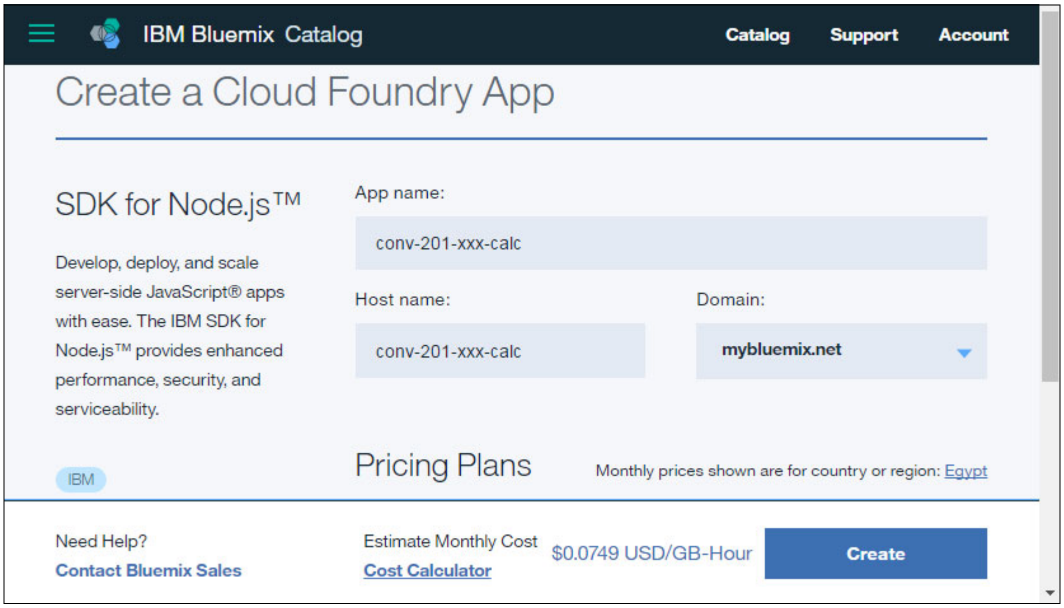
This section shows how to develop the Cognitive Calculator chatbot application that integrates with the Conversation service in Node.js.

Create a Node.js application on Bluemix

Follow these steps:

1. From the Bluemix dashboard, click **Create App**.
2. From the Cloud Foundry Apps section, click **SDK for Node.js**.
3. In the Create a Cloud Foundry App window (Figure 3-40) enter the following information, and then click **Create**:
   - App name: `conv-201-xxx-calc`
   - Host name: `conv-201-xxx-calc`

   Replace `xxx` with a random value; the host name of the application must be unique.



Create a Cloud Foundry App

SDK for Node.js™

Develop, deploy, and scale server-side JavaScript® apps with ease. The IBM SDK for Node.js™ provides enhanced performance, security, and serviceability.

App name: conv-201-xxx-calc

Host name: conv-201-xxx-calc

Domain: mybluemix.net

Monthly prices shown are for country or region: Egypt

Need Help?
Contact Bluemix Sales

Estimate Monthly Cost
Cost Calculator
$0.0749 USD/GB-Hour

Figure 3-40 Create Node.js application

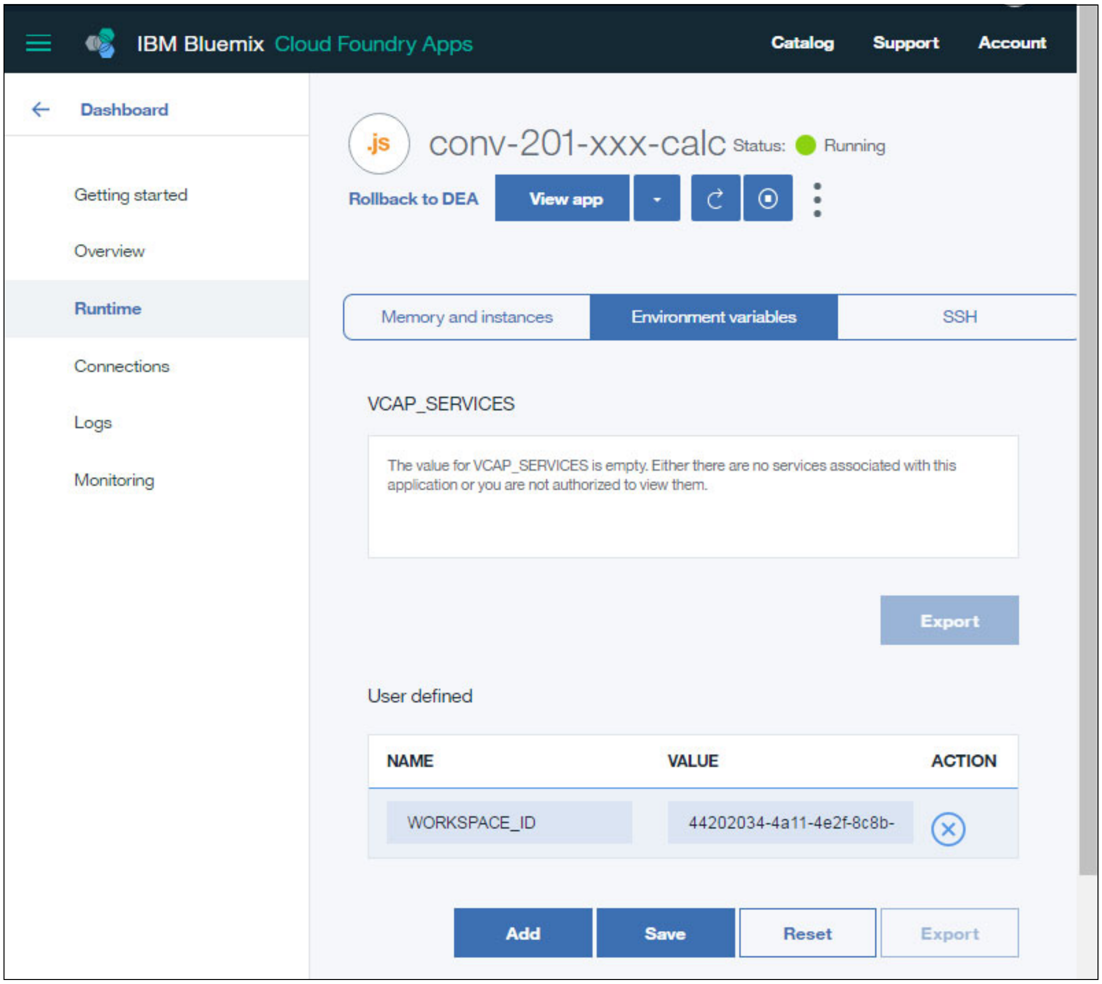
Stop: Wait until the application is started to proceed. The application status should indicate `Running`, as shown in Figure 3-41 on page 91.

Configure the application

Follow these steps:

1. Configure the application environment variables. Add the WORKSPACE_ID environment variable with the Workspace ID of your Calculator Conversation workspace (Figure 3-41):

   a. Click **Runtime** on the left navigation bar.
   b. Click the **Environment variables** tab.
   c. Click **Add**.
   d. For the name, specify `WORKSPACE_ID`.
   e. For the value, specify the Workspace ID value that you copied in step 6 on page 62.
   f. Click **Save**.



The value for VCAP_SERVICES is empty. Either there are no services associated with this application or you are not authorized to view them.

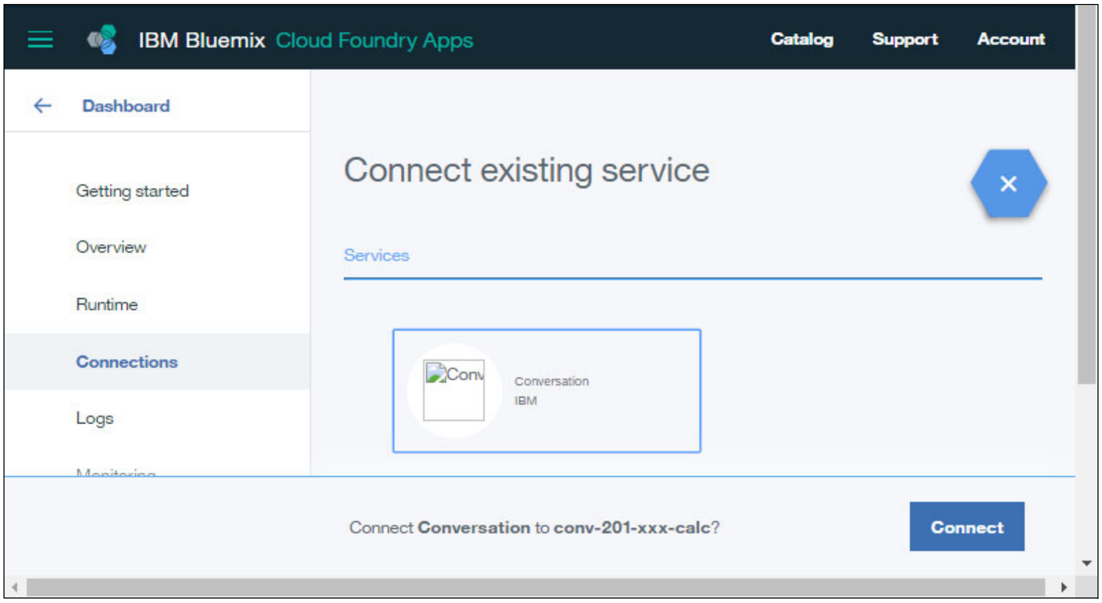
NAME	VALUE	ACTION
WORKSPACE_ID	44202034-4a11-4e2f-8c8b-	⊗

Figure 3-41 Adding WORKSPACE_ID as environment variable

> **Stop:** Wait until the application is restaged before you continue.

2. Bind the Conversation service to your application (Figure 3-42 on page 92):

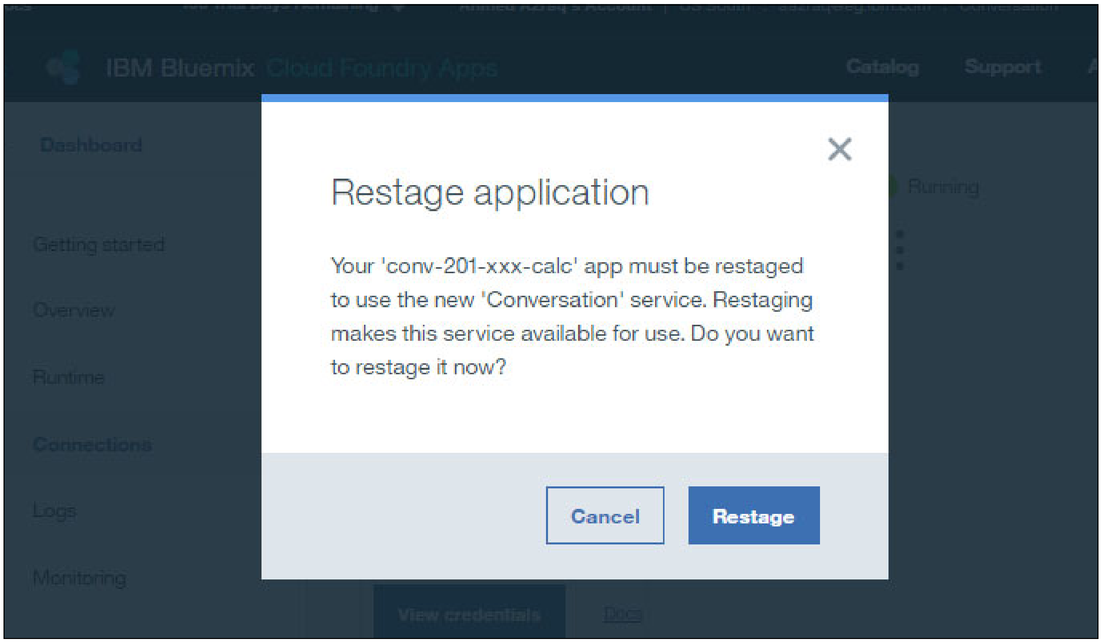
   a. Click **Connections** from the left toolbar.
   b. Click **Connect existing**.
   c. Click **Conversation**.
   d. Click **Connect**.



Connect **Conversation** to **conv-201-xxx-calc**?

Figure 3-42 Connect existing service

e. Click **Restage** to make the service available for use by the application (Figure 3-43).



Restage application

Your 'conv-201-xxx-calc' app must be restaged to use the new 'Conversation' service. Restaging makes this service available for use. Do you want to restage it now?

Figure 3-43 Restage application

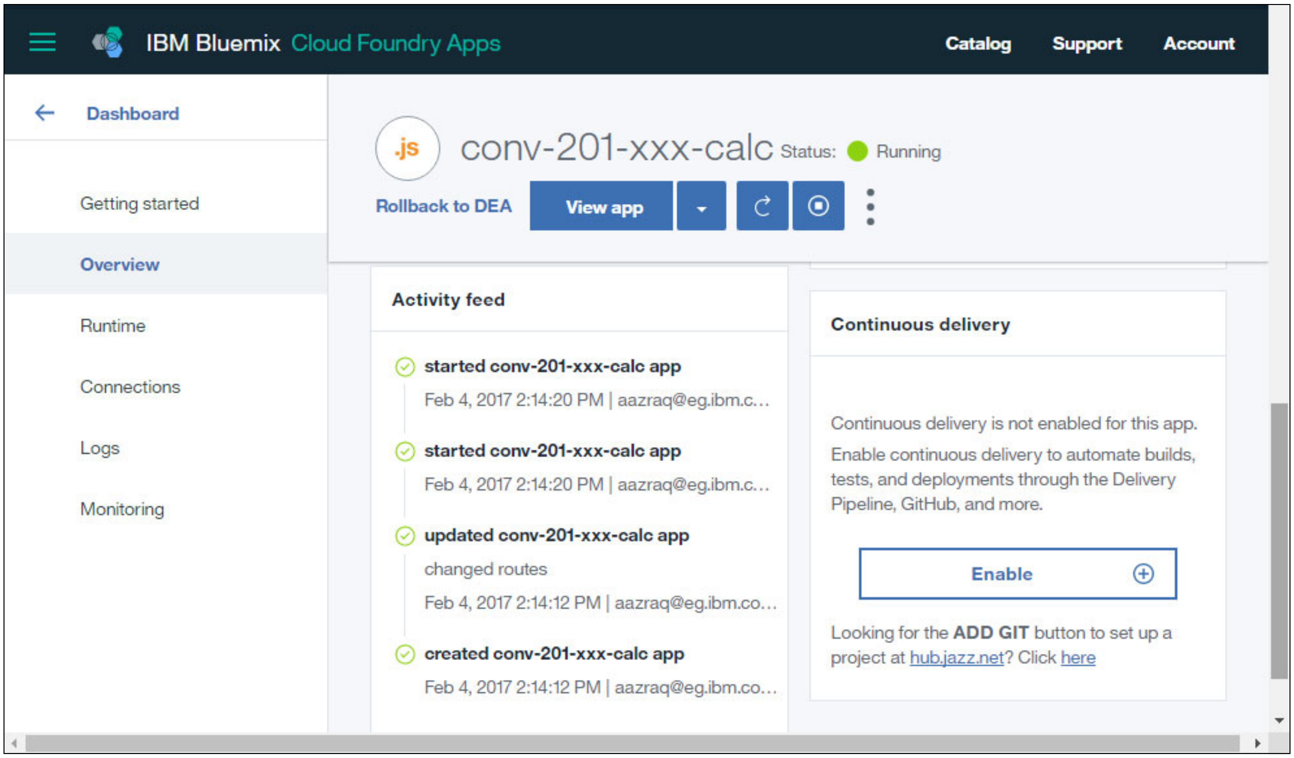
Stop: Wait until the restaging is completed and the application is in a running state before you continue.

Clone the Conversation sample application

In the next steps, you clone a sample Node.js application, which is a simple chatbot, to your Bluemix workspace.

1. Click **Overview** in the left navigation toolbar (Figure 3-44).
2. Scroll to the Continuous delivery panel, on the right, and click **Enable**.

   This enables the continuous delivery toolchain. With it, you can automate builds, tests, and deployments through the Delivery Pipeline, GitHub, and more.



started conv-201-xxx-calc app
Feb 4, 2017 2:14:20 PM | aazraq@eg.ibm.c...

started conv-201-xxx-calc app
Feb 4, 2017 2:14:20 PM | aazraq@eg.ibm.c...

updated conv-201-xxx-calc app
changed routes
Feb 4, 2017 2:14:12 PM | aazraq@eg.ibm.co...

created conv-201-xxx-calc app
Feb 4, 2017 2:14:12 PM | aazraq@eg.ibm.co...

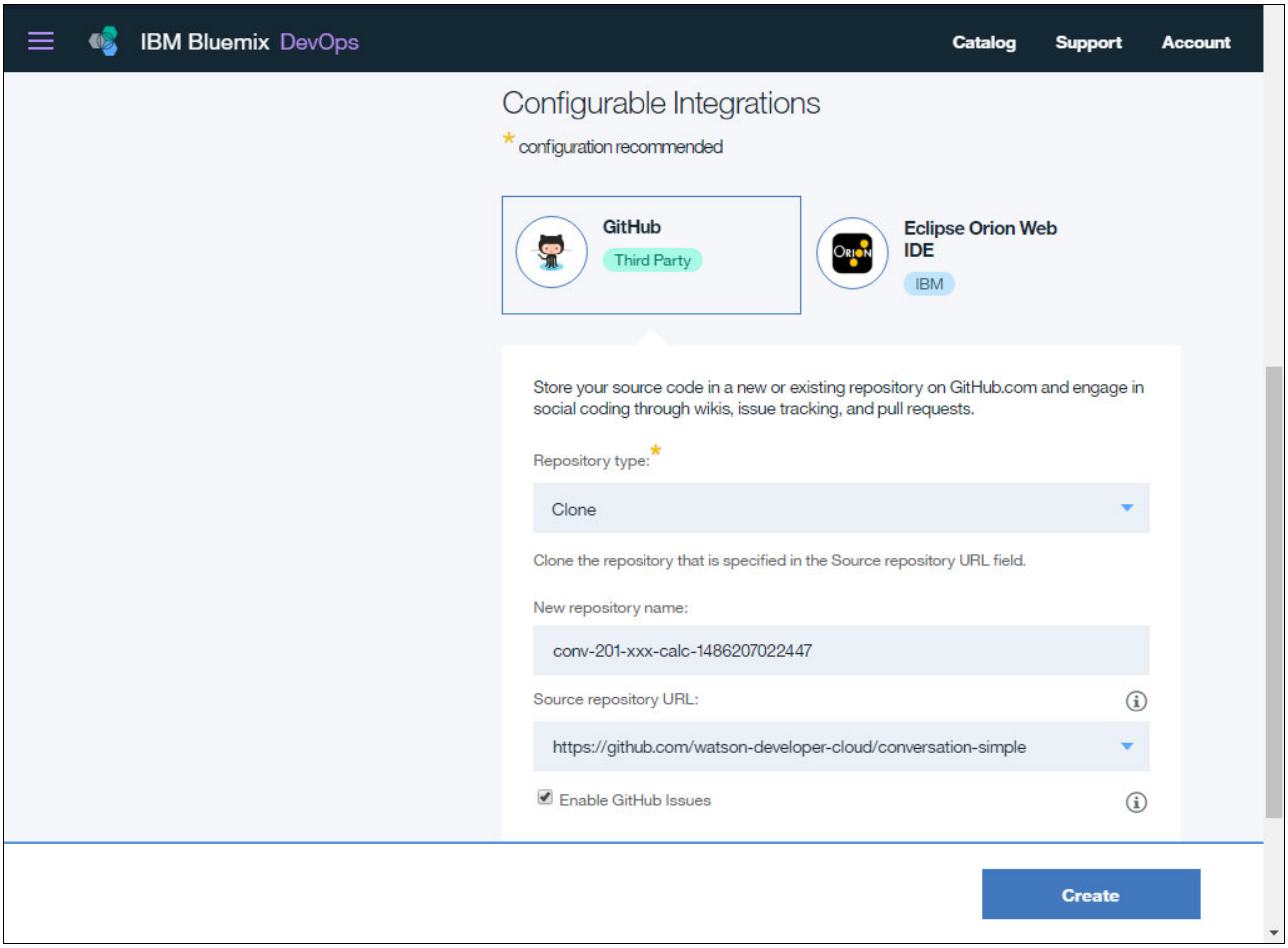
Continuous delivery is not enabled for this app.

Enable continuous delivery to automate builds, tests, and deployments through the Delivery Pipeline, GitHub, and more.

Looking for the **ADD GIT** button to set up a project at hub.jazz.net? Click here

Figure 3-44 Application overview

3. A new tab opens (Figure 3-45):
   a. Scroll to **Configurable Integrations** and click **GitHub**.
   b. Keep the repository type as `Clone`.
   c. Keep the default new repository name.
   d. For the Source repository URL, specify this GitHub repository URL:
      `https://github.com/watson-developer-cloud/conversation-simple`
   e. Click **Create**.



Store your source code in a new or existing repository on GitHub.com and engage in social coding through wikis, issue tracking, and pull requests.

Repository type:*

Clone

Clone the repository that is specified in the Source repository URL field.

New repository name:

conv-201-xxx-calc-1486207022447

Source repository URL:

https://github.com/watson-developer-cloud/conversation-simple

☑ Enable GitHub Issues

Figure 3-45 GitHub configurations

Edit the application code

In this section, you edit the code to implement the calculation functionality:

1. In the Toolchains window, click **Eclipse Orion Web IDE** (Figure 3-46).

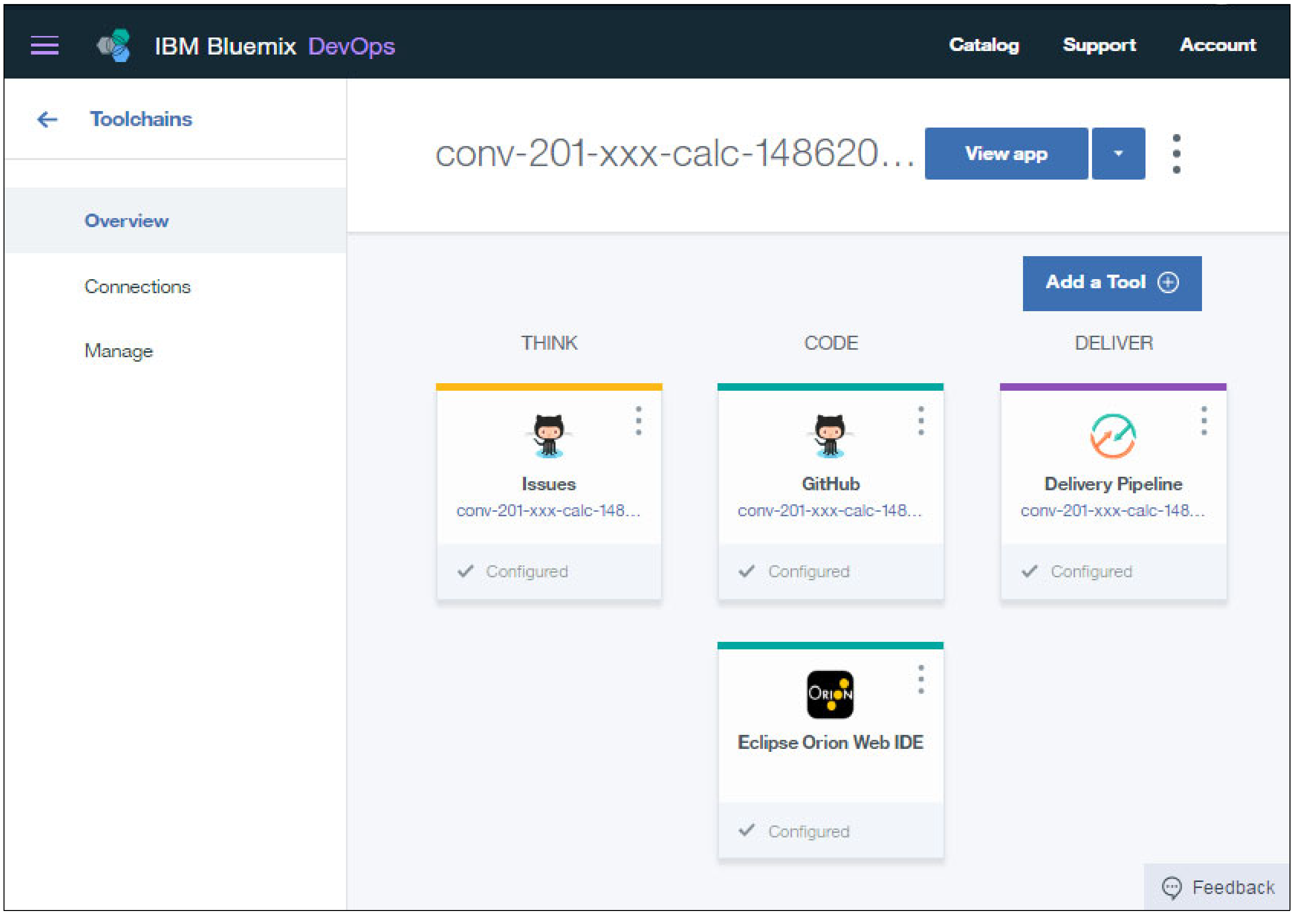
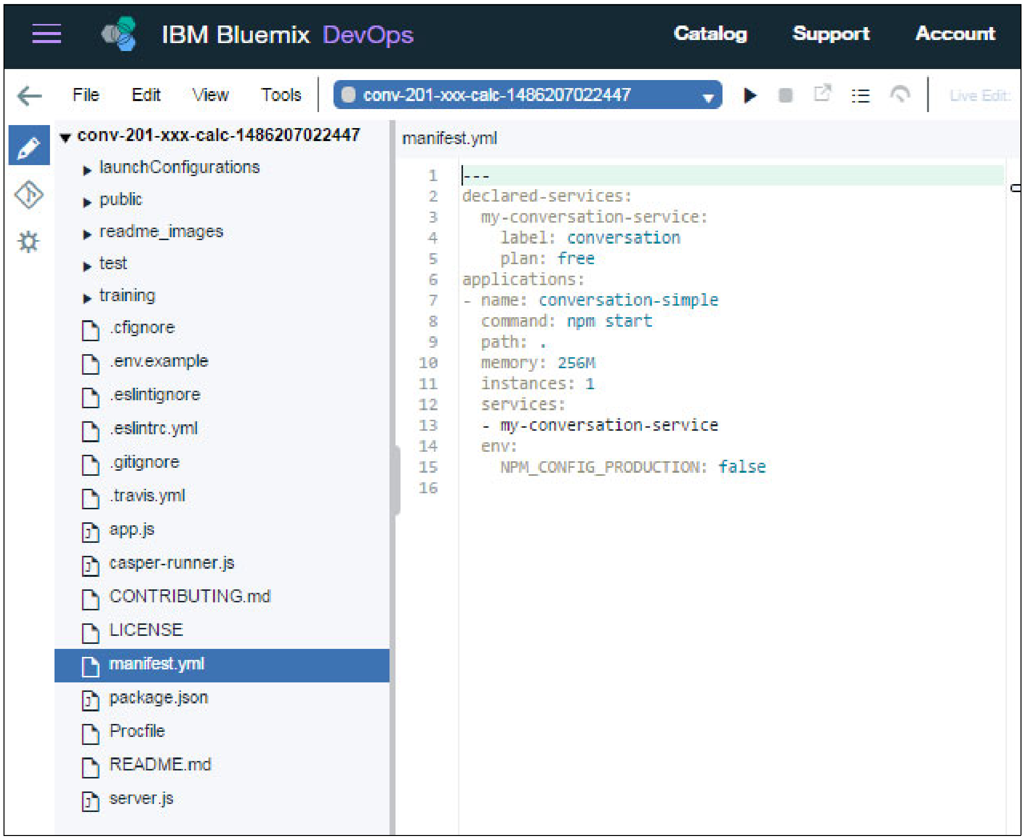


Figure 3-46 Toolchains window: Click Eclipse Orion Web IDE

2. Update the `manifest.yml` file with the host name and service name (Figure 3-47):

   a. In the list of files on the left, click the `manifest.yml` file.

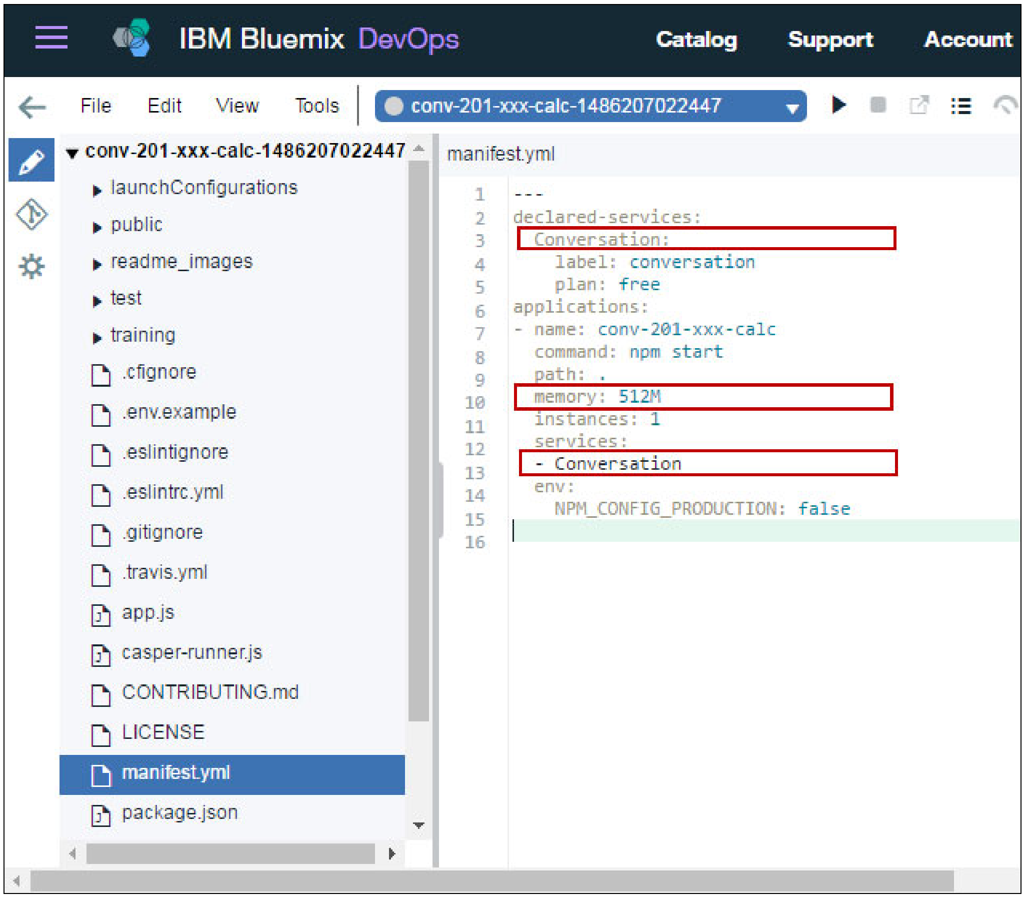


```
---
declared-services:
  my-conversation-service:
    label: conversation
    plan: free
applications:
- name: conversation-simple
  command: npm start
  path: .
  memory: 256M
  instances: 1
  services:
  - my-conversation-service
  env:
    NPM_CONFIG_PRODUCTION: false

```

Figure 3-47 The manifest.yml file before update

b. In the `manifest.yml` file shown in Figure 3-48, update this information:

- Update the Conversation service to match the name of the Conversation service instance created in 2.1.1, “Creating a Watson Conversation service instance” on page 14. To do this, replace `my-conversation-service` in line 3 and line 13 by `Conversation`.
- Update the application name to match the name of your application. To do this, update line 7 to `conv-201-xxx-calc` (where `xxx` is the value that you used to make your application and host names unique in step 3 on page 90).
- Increase the memory to `512M`, by updating line 10.



```
1   ---
2   declared-services:
3     Conversation:
4       label: conversation
5       plan: free
6   applications:
7   - name: conv-201-xxx-calc
8     command: npm start
9     path: .
10    memory: 512M
11    instances: 1
12    services:
13    - Conversation
14    env:
15      NPM_CONFIG_PRODUCTION: false
16
```

Figure 3-48   The manifest.yml file after the update

3. Edit app.js to perform the calculation and update the response received from the Conversation service with the calculation results based on the intents and entities:

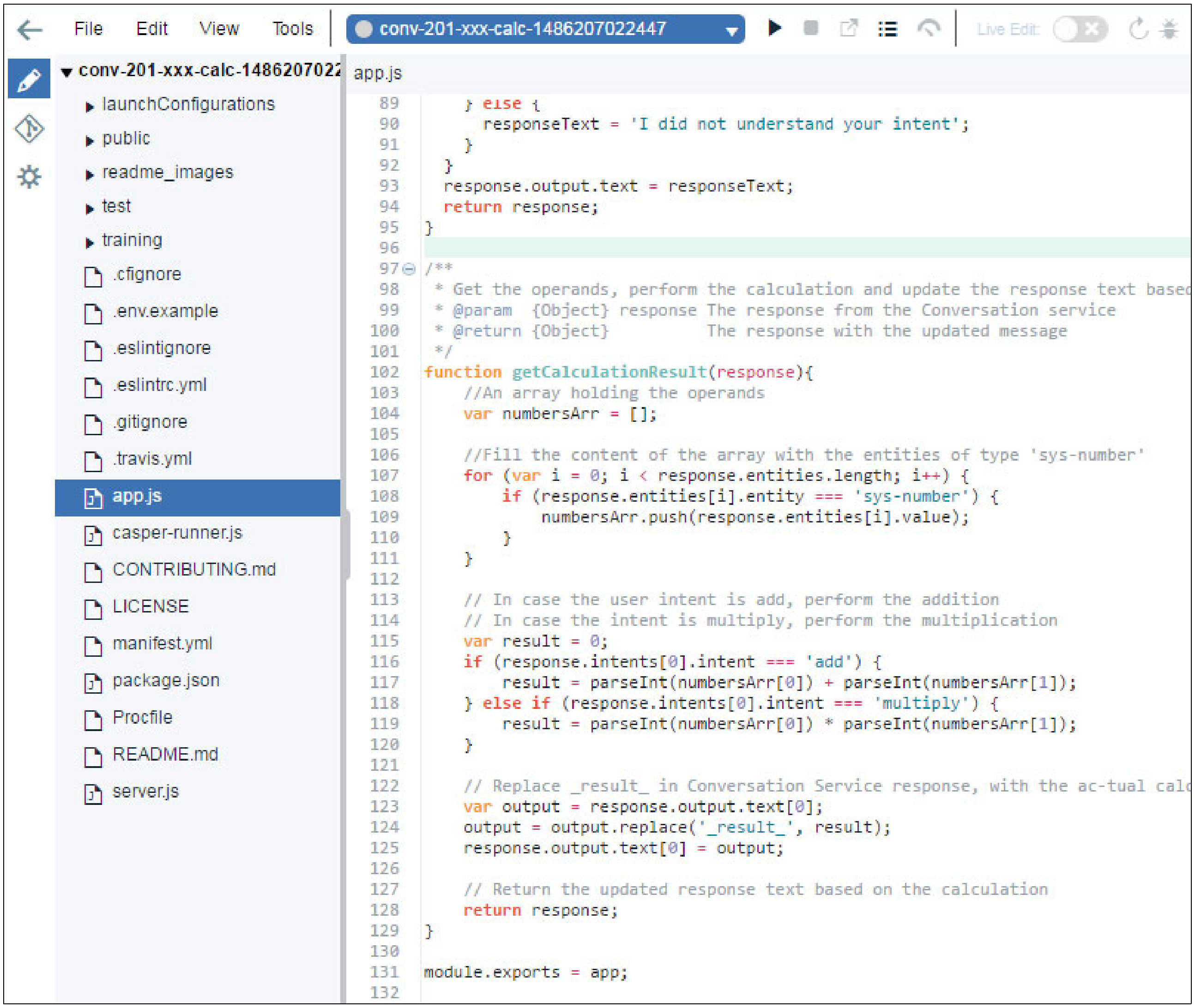
   a. From the list of files on the left, click the app.js file.

   b. Add function getCalculationResult (Example 3-1) before the last line in the code, which is (module.exports = app) as shown in Figure 3-49 on page 99. This function performs the calculation and updates the response text.

Example 3-1 Get calculation result function

```
/**
 * Get the operands, perform the calculation and update the response text based on the
 * calculation.
 * @param  {Object} response The response from the Conversation service
 * @return {Object}          The response with the updated message
 */
function getCalculationResult(response){
   //An array holding the operands
   var numbersArr = [];

   //Fill the content of the array with the entities of type 'sys-number'
   for (var i = 0; i < response.entities.length; i++) {
      if (response.entities[i].entity === 'sys-number') {
         numbersArr.push(response.entities[i].value);
      }
   }

   // In case the user intent is add, perform the addition
  // In case the intent is multiply, perform the multiplication
   var result = 0;
   if (response.intents[0].intent === 'add') {
      result = parseInt(numbersArr[0]) + parseInt(numbersArr[1]);
   } else if (response.intents[0].intent === 'multiply') {
      result = parseInt(numbersArr[0]) * parseInt(numbersArr[1]);
   }

   // Replace _result_ in Conversation Service response, with the actual calculated result
   var output = response.output.text[0];
   output = output.replace('_result_', result);
   response.output.text[0] = output;

   // Return the updated response text based on the calculation
   return response;
}
```

Figure 3-49 shows the result of adding the `getCalculationResult` function to the `app.js` file.



```
      } else {
        responseText = 'I did not understand your intent';
      }
    }
    response.output.text = responseText;
    return response;
}

/**
 * Get the operands, perform the calculation and update the response text based
 * @param  {Object} response The response from the Conversation service
 * @return {Object}          The response with the updated message
 */
function getCalculationResult(response){
    //An array holding the operands
    var numbersArr = [];

    //Fill the content of the array with the entities of type 'sys-number'
    for (var i = 0; i < response.entities.length; i++) {
        if (response.entities[i].entity === 'sys-number') {
            numbersArr.push(response.entities[i].value);
        }
    }

    // In case the user intent is add, perform the addition
    // In case the intent is multiply, perform the multiplication
    var result = 0;
    if (response.intents[0].intent === 'add') {
        result = parseInt(numbersArr[0]) + parseInt(numbersArr[1]);
    } else if (response.intents[0].intent === 'multiply') {
        result = parseInt(numbersArr[0]) * parseInt(numbersArr[1]);
    }

    // Replace _result_ in Conversation Service response, with the ac-tual calc
    var output = response.output.text[0];
    output = output.replace('_result_', result);
    response.output.text[0] = output;

    // Return the updated response text based on the calculation
    return response;
}

module.exports = app;

```

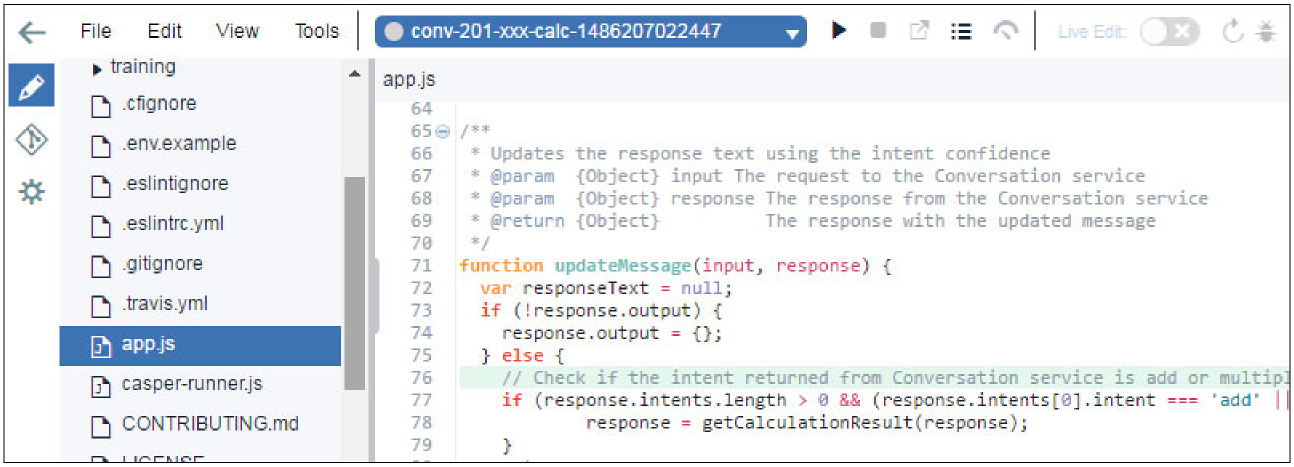
Figure 3-49 The app.js file after adding the getCalculationResult function

c. Call the `getCalculationResult` function (Example 3-2) on line 76 (Figure 3-50 on page 100).

Example 3-2 Check intent

```
// Check if the intent returned from Conversation service is add or multiply,
// perform the calculation and update the response
if (response.intents.length > 0 && (response.intents[0].intent === 'add' ||
response.intents[0].intent === 'multiply')) {
   response = getCalculationResult(response);
}
```

Figure 3-50 shows calling `getCalculationResult` on line 76.



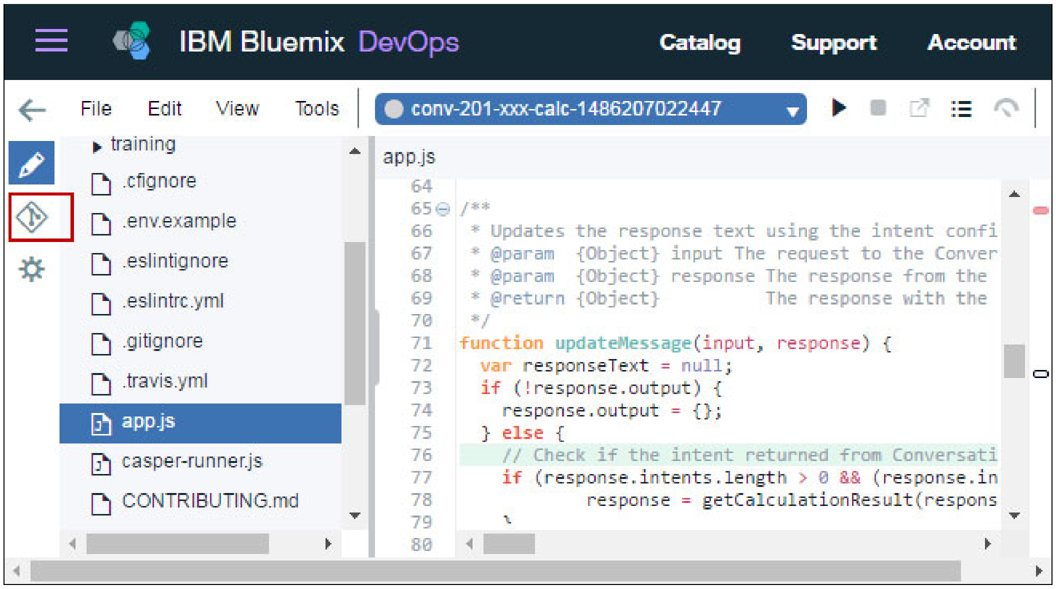
```
64
65 /**
66  * Updates the response text using the intent confidence
67  * @param  {Object} input The request to the Conversation service
68  * @param  {Object} response The response from the Conversation service
69  * @return {Object}          The response with the updated message
70  */
71 function updateMessage(input, response) {
72   var responseText = null;
73   if (!response.output) {
74     response.output = {};
75   } else {
76     // Check if the intent returned from Conversation service is add or multipl
77     if (response.intents.length > 0 && (response.intents[0].intent === 'add' ||
78             response = getCalculationResult(response);
79     }
```

Figure 3-50 Calling getCalculationResult

Push the changes to Git

Follow these steps:

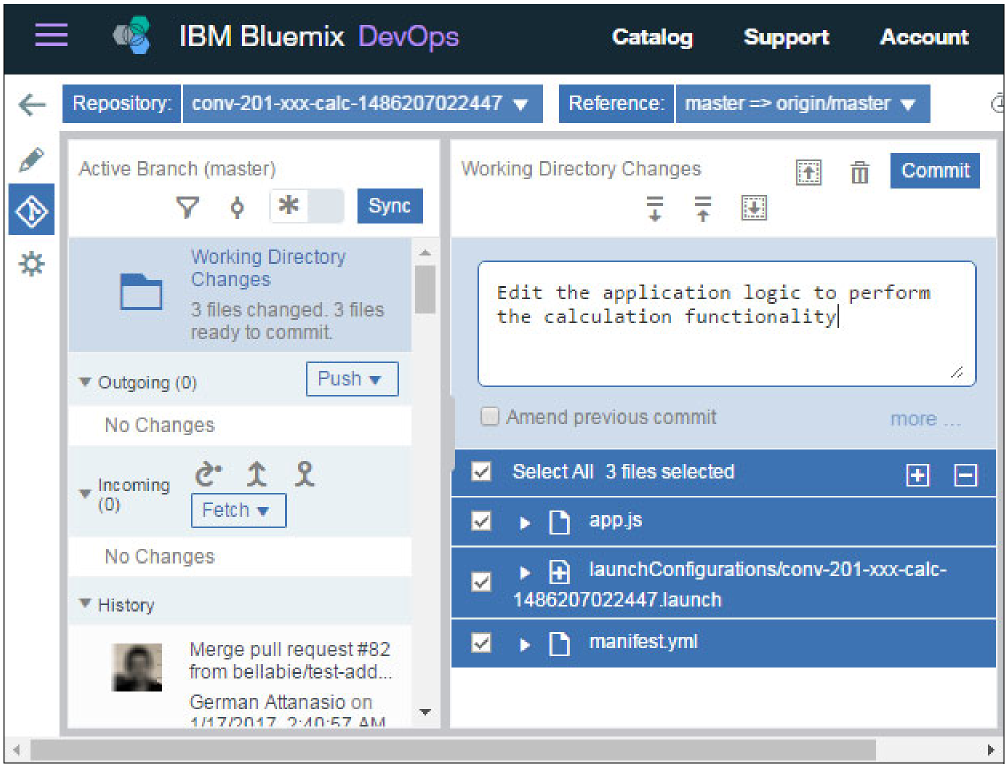
1. Click the **Git** icon on the left toolbar (Figure 3-51).



```
64
65 /**
66  * Updates the response text using the intent confi
67  * @param  {Object} input The request to the Conver
68  * @param  {Object} response The response from the
69  * @return {Object}          The response with the
70  */
71 function updateMessage(input, response) {
72   var responseText = null;
73   if (!response.output) {
74     response.output = {};
75   } else {
76     // Check if the intent returned from Conversati
77     if (response.intents.length > 0 && (response.in
78             response = getCalculationResult(respons
79     }
80
```

Figure 3-51 Click the Git icon in the IBM Bluemix DevOps page

2. Enter any descriptive commit message (such as `Edit the application logic to perform the calculation functionality`), and click **Commit** (Figure 3-52 on page 101).

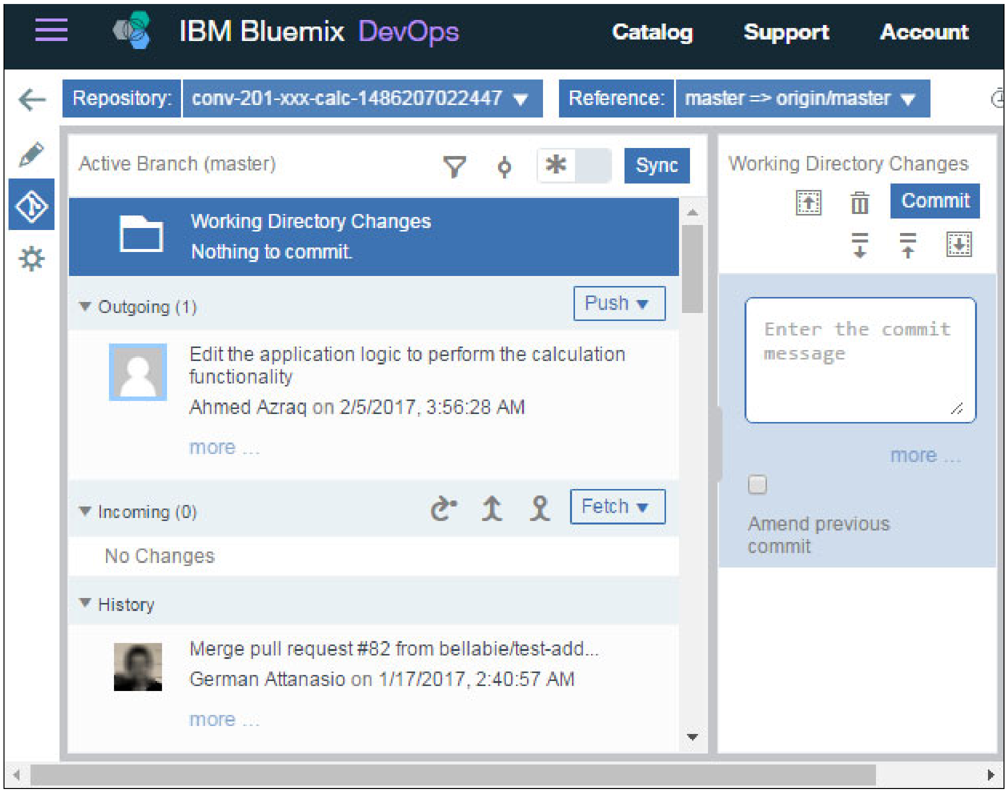

IBM Bluemix DevOps
Catalog
Support
Account
Repository: conv-201-xxx-calc-1486207022447
Reference: master => origin/master
Active Branch (master)
Sync
Working Directory Changes
3 files changed. 3 files ready to commit.
Outgoing (0)
Push
No Changes
Incoming (0)
Fetch
No Changes
History
Working Directory Changes
Commit
Amend previous commit
more ...
Select All 3 files selected
app.js
launchConfigurations/conv-201-xxx-calc-1486207022447.launch
manifest.yml

Edit the application logic to perform the calculation functionality

Merge pull request #82 from bellabie/test-add...
German Attanasio on 1/17/2017, 2:40:57 AM

Figure 3-52 DevOps Git: Commit changes

3. Click **Push** to push your committed changes to the remote branch (Figure 3-53).



Edit the application logic to perform the calculation functionality
Ahmed Azraq on 2/5/2017, 3:56:28 AM

Merge pull request #82 from bellabie/test-add...
German Attanasio on 1/17/2017, 2:40:57 AM

Figure 3-53 DevOps Git: Push changes to remote branch

4. Return to the **Toolchains** tab, and click on **Delivery Pipeline** (Figure 3-54).

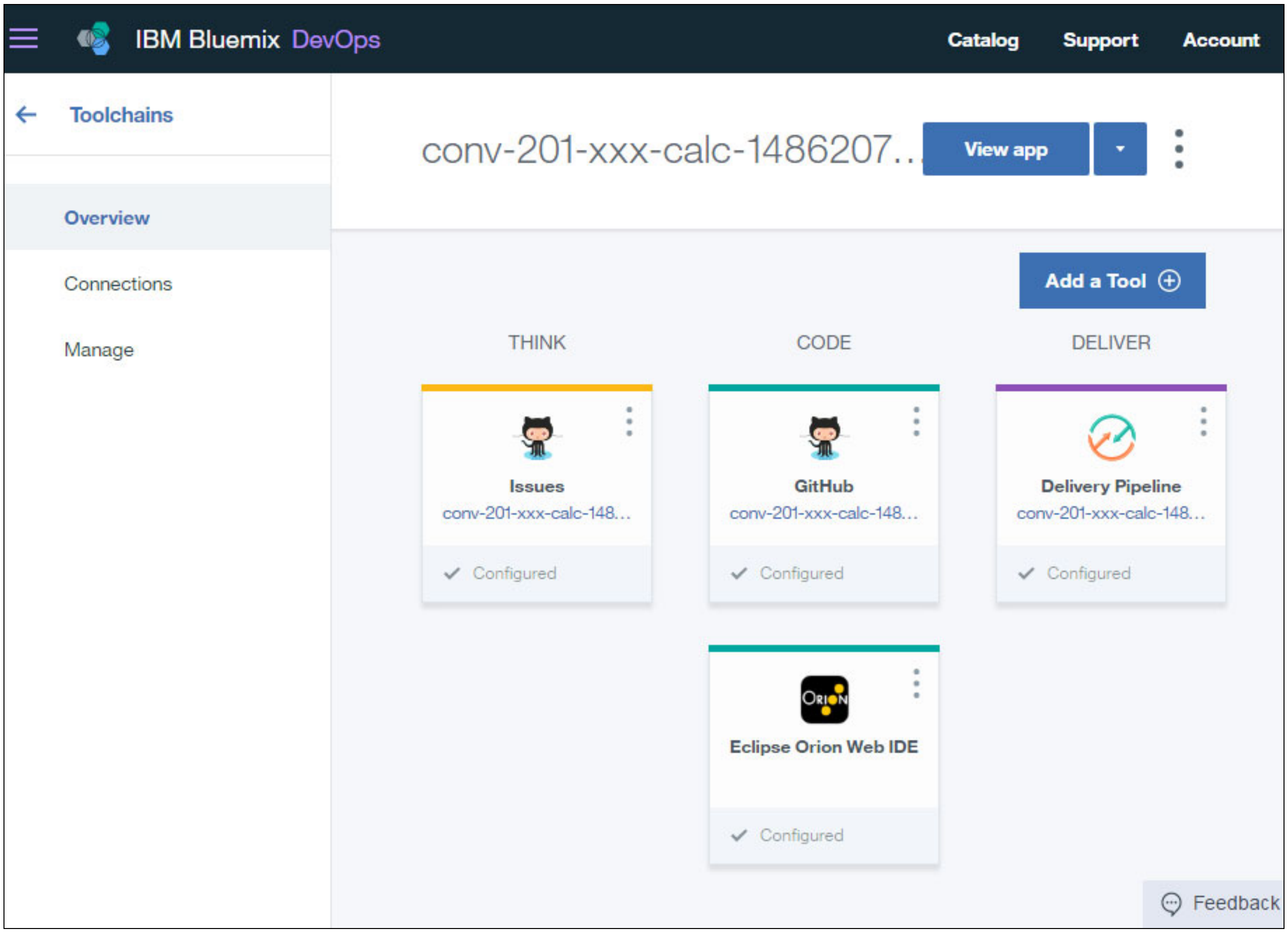
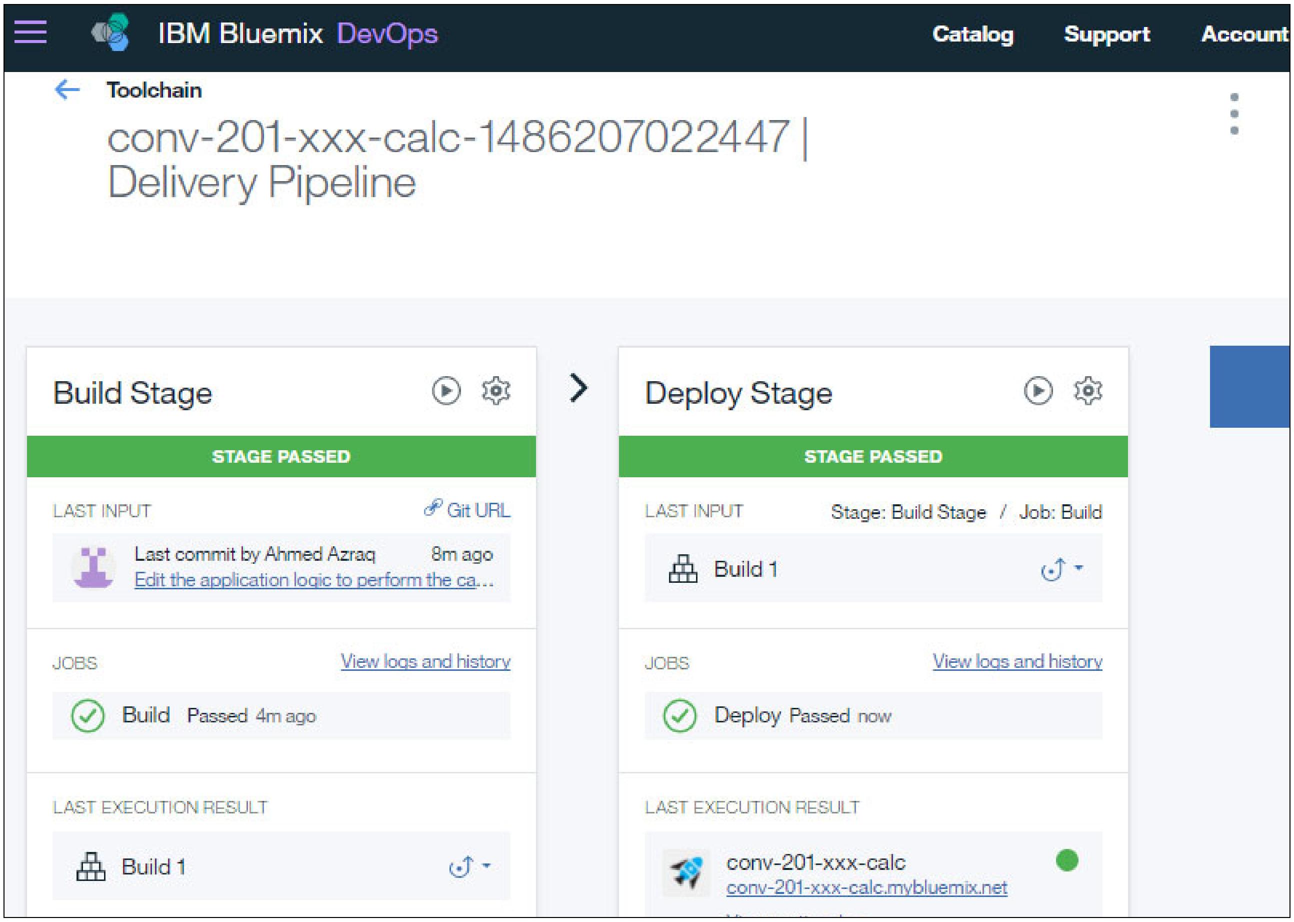


Figure 3-54 Toolchains: Select Delivery Pipeline

5. Wait until the build and deploy stages are completed (Figure 3-55). When they are completed, and with no errors, your application is ready to be tested.



conv-201-xxx-calc-1486207022447 |
Delivery Pipeline

Build Stage

STAGE PASSED

LAST INPUT — Git URL

Last commit by Ahmed Azraq 8m ago
Edit the application logic to perform the ca…

JOBS — View logs and history

Build Passed 4m ago

LAST EXECUTION RESULT

Build 1

Deploy Stage

STAGE PASSED

LAST INPUT — Stage: Build Stage / Job: Build

Build 1

JOBS — View logs and history

Deploy Passed now

LAST EXECUTION RESULT

conv-201-xxx-calc
conv-201-xxx-calc.mybluemix.net

Figure 3-55 Delivery Pipeline: Build Stage and Deploy Stage

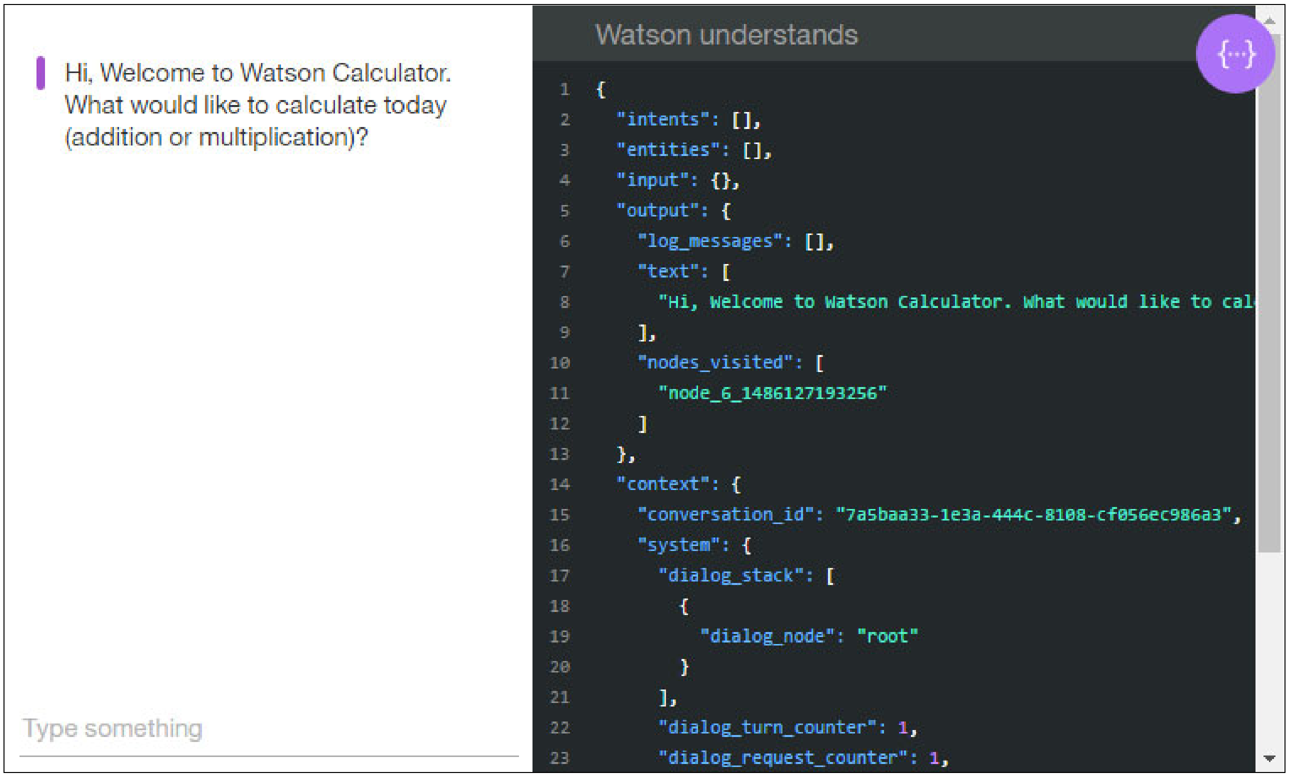
Stop: Wait until the build and deploy stages are completed before testing.

3.4.4 Testing the application

Follow these steps:

1. Open your application route (the URL to access your application) in a web browser with the following address, where xxx is the value that you added in step 3 on page 90 to make your application name unique (Figure 3-56):

   `http://conv-201-xxx-calc.mybluemix.net/`

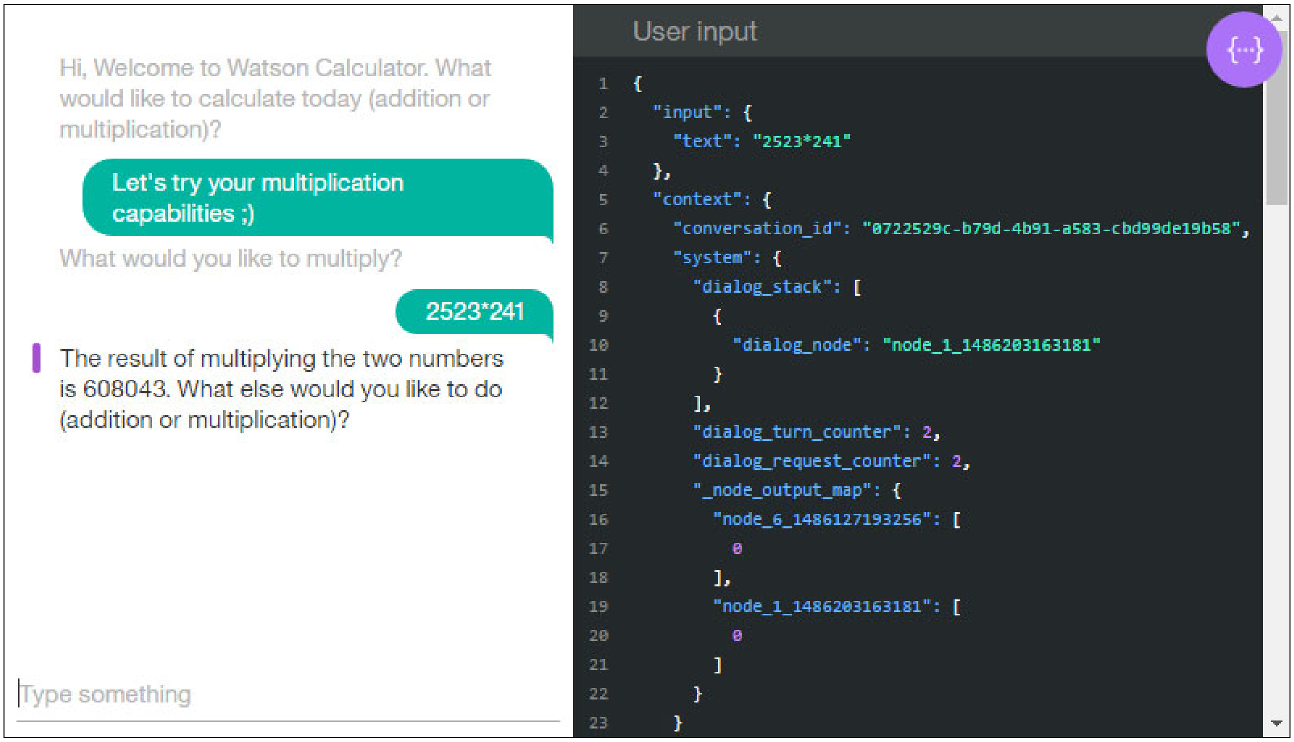


Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

```
{
  "intents": [],
  "entities": [],
  "input": {},
  "output": {
    "log_messages": [],
    "text": [
      "Hi, Welcome to Watson Calculator. What would like to cal
    ],
    "nodes_visited": [
      "node_6_1486127193256"
    ]
  },
  "context": {
    "conversation_id": "7a5baa33-1e3a-444c-8108-cf056ec986a3",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1,
```

Figure 3-56 Calculator chatbot application

2. Test the addition and multiplication functionalities by chatting with the application (Figure 3-57).



Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

Let's try your multiplication capabilities ;)

What would you like to multiply?

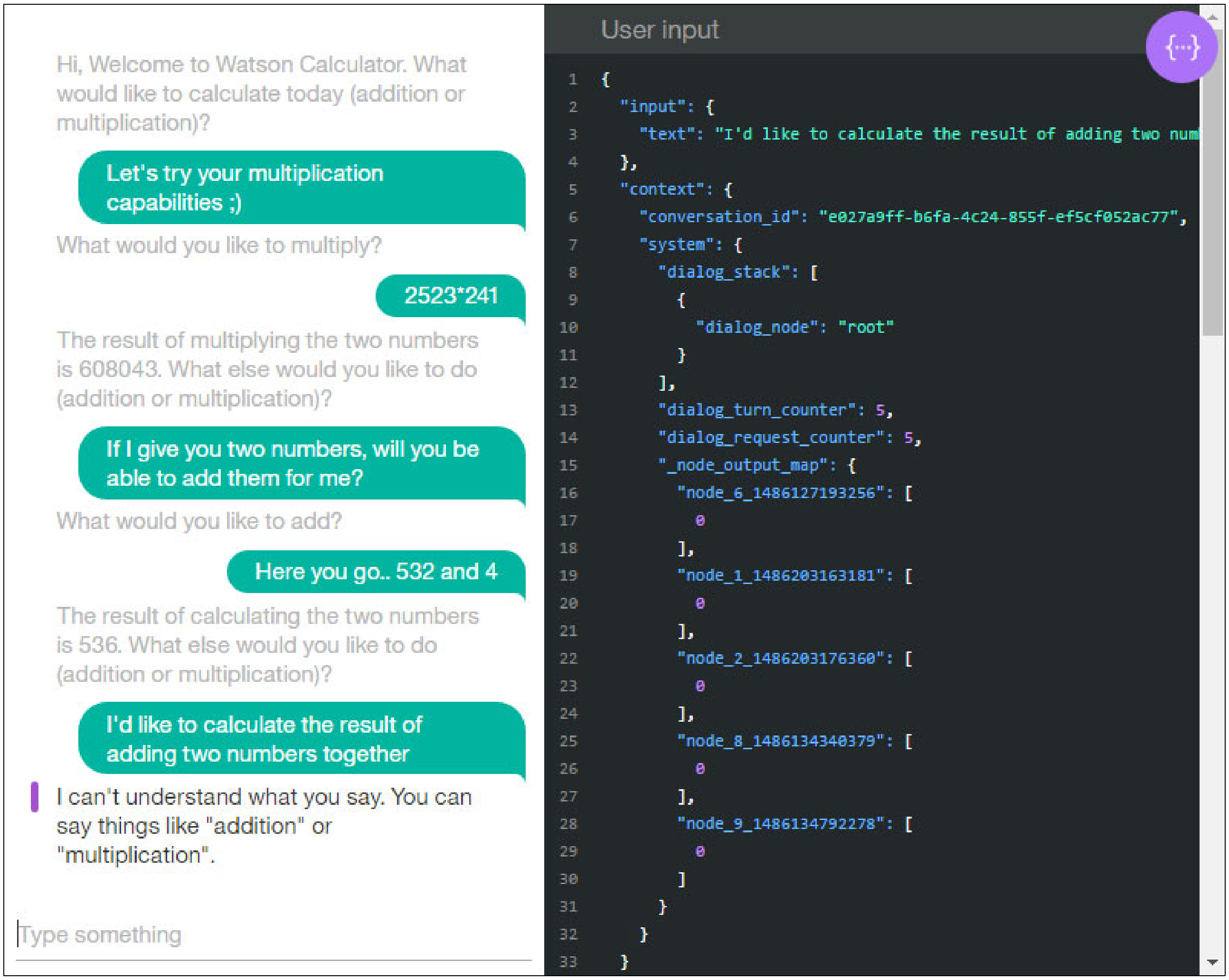
2523*241

The result of multiplying the two numbers is 608043. What else would you like to do (addition or multiplication)?

```
{
  "input": {
    "text": "2523*241"
  },
  "context": {
    "conversation_id": "0722529c-b79d-4b91-a583-cbd99de19b58",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "node_1_1486203163181"
        }
      ],
      "dialog_turn_counter": 2,
      "dialog_request_counter": 2,
      "_node_output_map": {
        "node_6_1486127193256": [
          0
        ],
        "node_1_1486203163181": [
          0
        ]
      }
    }
```

Figure 3-57 Multiplication test on the Cognitive Calculator chatbot

3. Try various scenarios and identify those for which the application fails to respond appropriately. Failing to respond correctly means that more training is needed. Training is performed by adding more user examples to the intents in the Calculator Conversation workspace (Figure 3-58).



Hi, Welcome to Watson Calculator. What would like to calculate today (addition or multiplication)?

Let's try your multiplication capabilities ;)

What would you like to multiply?

2523*241

The result of multiplying the two numbers is 608043. What else would you like to do (addition or multiplication)?

If I give you two numbers, will you be able to add them for me?

What would you like to add?

Here you go.. 532 and 4

The result of calculating the two numbers is 536. What else would you like to do (addition or multiplication)?

I'd like to calculate the result of adding two numbers together

I can't understand what you say. You can say things like "addition" or "multiplication".

```
{
  "input": {
    "text": "I'd like to calculate the result of adding two num
  },
  "context": {
    "conversation_id": "e027a9ff-b6fa-4c24-855f-ef5cf052ac77",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 5,
      "dialog_request_counter": 5,
      "_node_output_map": {
        "node_6_1486127193256": [
          0
        ],
        "node_1_1486203163181": [
          0
        ],
        "node_2_1486203176360": [
          0
        ],
        "node_8_1486134340379": [
          0
        ],
        "node_9_1486134792278": [
          0
        ]
      }
    }
  }
```

Figure 3-58 Various scenarios in the Cognitive Calculator chatbot shows that the intents need more training

3.5 Quick deployment of application

A second GIT repository is provided so that you can build and deploy the full Cognitive Calculator chatbot even if you did not perform the steps described in 3.4, "Step-by-step implementation" on page 59. This section is independent from the rest of the chapter and it contains instructions to run the application more quickly.

You can find the full version of the application in the following Git repository:

https://github.com/snippet-java/redbooks-conv-201-calc-nodejs

The file `calculator_workspace.json` includes the Calculator workspace created in this chapter and is at this GitHub location:

https://github.com/snippet-java/redbooks-conv-201-calc-nodejs/blob/master/training/calculator_workspace.json

Use the following steps to quickly deploy the full application:

1. Click **Deploy this application to Bluemix** at the following web page:

   https://bluemix.net/deploy?repository=https://github.com/snippet-java/redbooks-conv-201-calc-nodejs

2. Import the Calculator workspace into your Conversation service. For information on importing a Conversation workspace see "Import a workspace" on page 20.
3. Follow the steps in "Configure the application" on page 91 to configure your application to point to the Calculator workspace.
4. Test the application as described in 3.4.4, "Testing the application" on page 104.

3.6 References

For helpful information, see the following resources:

- Explore other sample applications to understand the types of apps you can develop with the Conversation service:

  https://www.ibm.com/watson/developercloud/doc/conversation/sample-applications.html

- See the `README.md` file in the *incomplete* GitHub repository of the application:

  https://github.com/watson-developer-cloud/conversation-simple

4

Help Desk Assistant chatbot

This chapter describes how to create a chatbot application quickly without coding and integrate it with the Watson Conversation service. For this use case example, you create a *Help Desk Assistant* chatbot, however you can customize the chatbot to take any other role such as delivery service, Q&A, student assistant, and more.

To create the chatbot application, you use the Node-RED programming tool. With this powerful tool you can create, edit, and deploy applications quickly. Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

Node-RED, created by IBM but now part of JS Foundation, provides full integration with Watson APIs, allowing you to make great applications quickly and easy.

The following topics are covered in this chapter:

- Getting started
- Architecture
- Two ways to deploy the application: Step-by-step and quick deploy
- Step-by-step implementation
- Quick deployment of application
- Next steps
- References

4.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

4.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Understand the basics of Node-RED.
- Configure a conversation workspace with intents, entities and dialog.
- Create a Node-RED application and integrate the Watson Conversation service in the application.
- Configure a Slack chatbot to call your Node-RED application.

4.1.2 Prerequisites

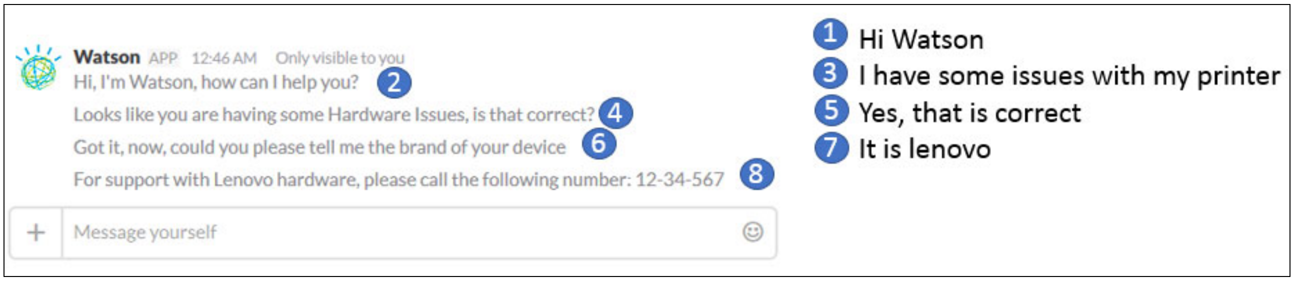
To complete the steps in this chapter, be sure you have these prerequisites:

- Access to a Bluemix account
- Basic knowledge of Bluemix
- Basic knowledge of the IBM Watson Conversation service
- Access to a Slack account (you can create a free account at www.slack.com)

Also be sure you completed the previous chapters in this book.

4.1.3 Expected results

Figure 4-1 shows the Help Desk Assistant chatbot application interface during a conversation in Slack. Although this chatbot uses Slack, consider that the chatbot can be also integrated with other chat services such as Facebook Messenger.



Watson APP 12:46 AM Only visible to you

Hi, I'm Watson, how can I help you? 2

Looks like you are having some Hardware Issues, is that correct? 4

Got it, now, could you please tell me the brand of your device 6

For support with Lenovo hardware, please call the following number: 12-34-567 8

1 Hi Watson

3 I have some issues with my printer

5 Yes, that is correct

7 It is lenovo

Figure 4-1 Help Desk Assistant chatbot interface

4.2 Architecture

Figure 4-2 shows the components of the application.

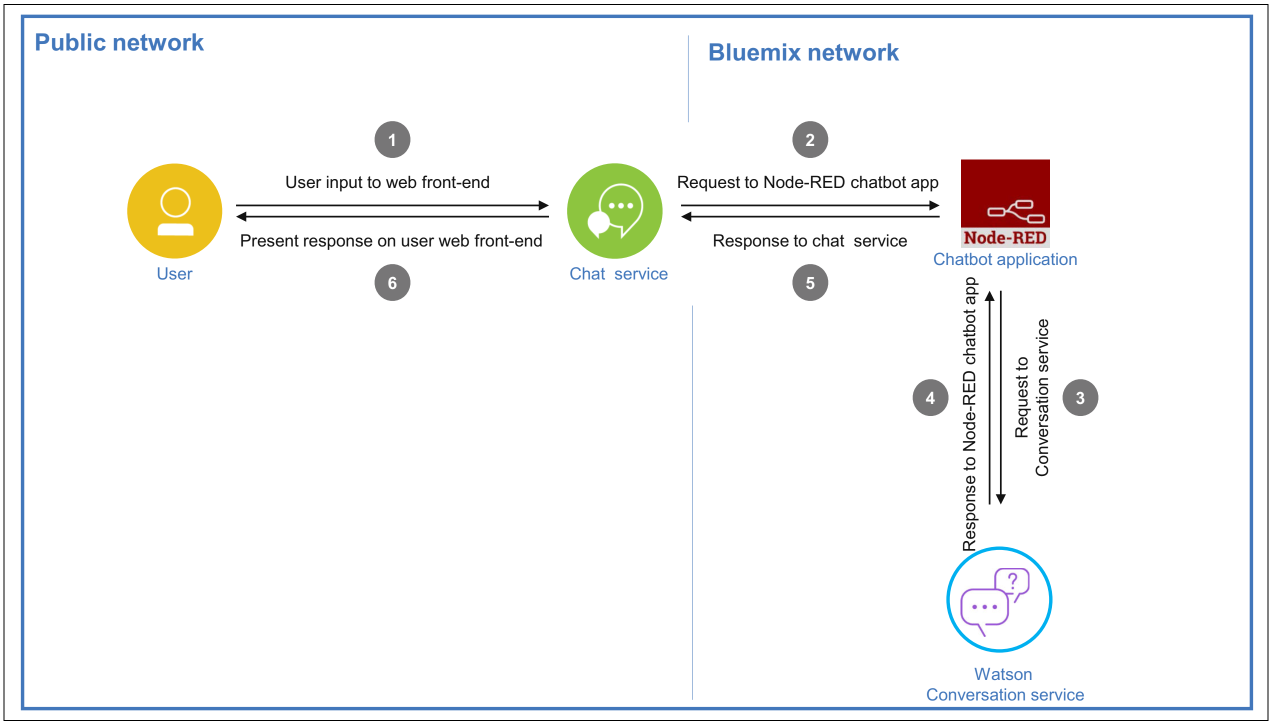

Public network
Bluemix network
1
User input to web front-end
Present response on user web front-end
6
User
Chat service
2
Request to Node-RED chatbot app
Response to chat service
5
Node-RED
Chatbot application
4
Response to Node-RED chatbot app
Request to Conversation service
3
Watson Conversation service

Figure 4-2 Architecture

Notice that the flow shown in the figure represents one loop of a conversation, therefore this cycle repeats several times during a conversation:

1. The user sends a message to the web front-end (chat service).
2. The chat service (for example, Slack, Facebook Messenger, web app) determines whether the message is for the Help Desk Assistant chatbot application. If the message is for the chatbot, then the chat service sends the message to your chatbot application (Node-RED).
3. Your application parses the message and sends the filtered message to the Watson Conversation service for processing.
4. The Watson Conversation service processes the message and provides a response.
5. The response is received and filtered by your application, which then sends the response to the chat service.
6. The chat service identifies that the inputs are from the Help Desk Assistant chatbot and presents the message as a response from the chatbot to the user.

4.2.1 Project structure

These are the components you use in this use case:

- A Node-RED instance that is created in Bluemix, which is cloud-based, so installing software is not necessary
- A Watson Conversation service instance
- A team space in Slack, which is the cloud collaboration tool that provides the chat service in this use case

4.3 Two ways to deploy the application: Step-by-step and quick deploy

These are the two ways to experience this use case:

- Step-by-step implementation

  This approach takes you through the key steps to integrate the IBM Watson Conversation service with the application logic. All sections of 4.4, “Step-by-step implementation” on page 112 take you through step-by-step deployment.
- Quick deployment

  A Git repository is provided with a version of the Node-RED application. You only need to perform the required steps to customize the application for your specific Conversation service instance and Slack team. This approach is explained in 4.5, “Quick deployment of application” on page 136.

4.4 Step-by-step implementation

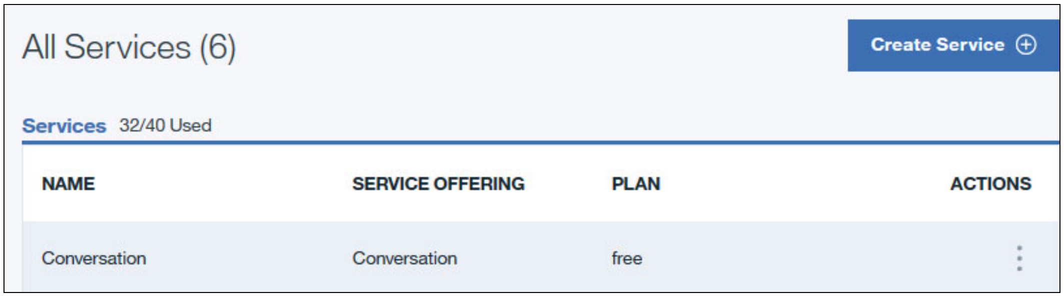
Implementing this use case involves the following steps:

1. Creating a new Conversation workspace
2. Adding intents
3. Adding entities
4. Creating the dialog
5. Testing the dialog
6. Creating the Help Desk Assistant chatbot application in Node-RED
7. Setting up the chat service (Slack)

4.4.1 Creating a new Conversation workspace

Complete the following steps:

1. Log in to Bluemix and open the Dashboard.
2. Find the Watson service instance created in 2.1.1, “Creating a Watson Conversation service instance” on page 14 and click to open it (Figure 4-3).



NAME	SERVICE OFFERING	PLAN	ACTIONS
Conversation	Conversation	free	⋮

Figure 4-3 Access the Conversation service instance

3. Click **Launch tool** to access your Conversation workspaces (Figure 4-4).

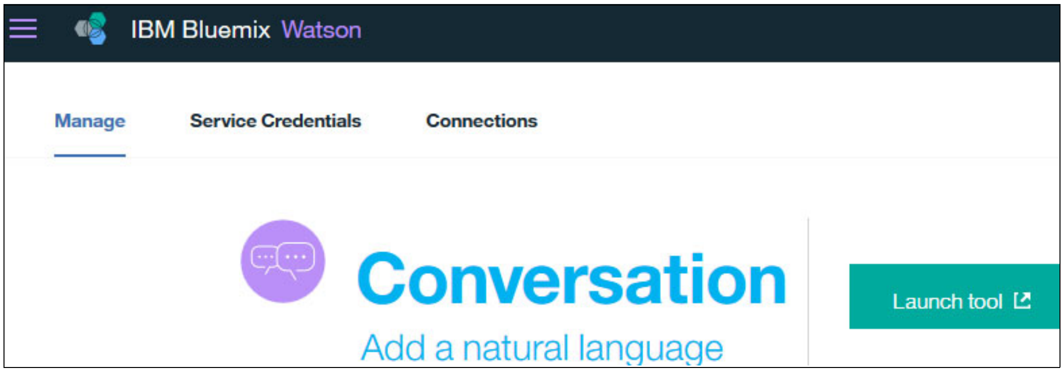
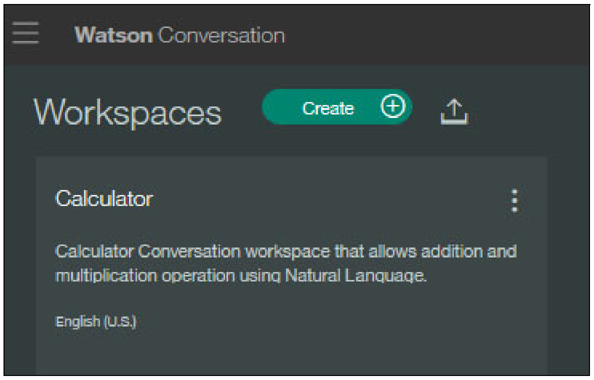


Figure 4-4 Launch Conversation service tool

4. Previously created workspaces are listed (Figure 4-5). However, for this app you need a new workspace, so click **Create**.



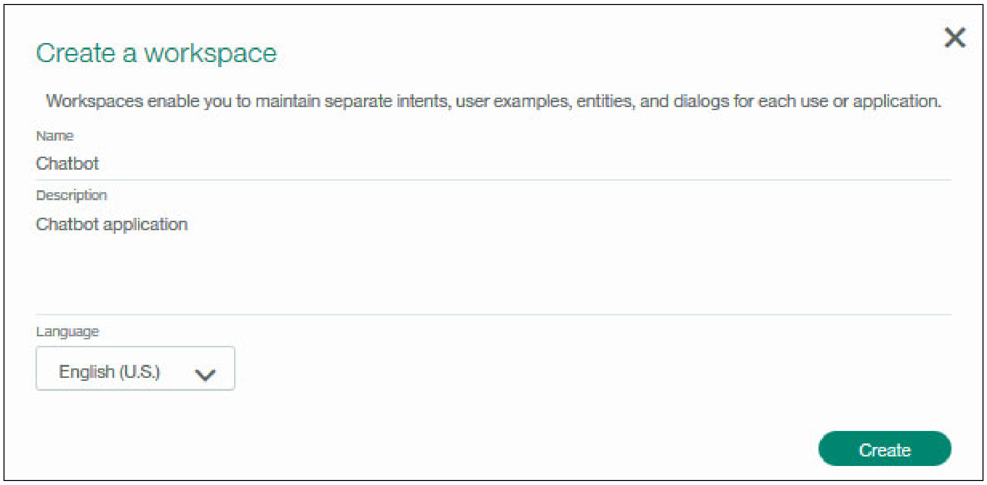
Calculator

Calculator Conversation workspace that allows addition and multiplication operation using Natural Language.

English (U.S.)

Figure 4-5 Watson Conversation workspaces

5. Add a name and description and click **Create** (Figure 4-6).



Create a workspace

Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

Name

Chatbot

Description

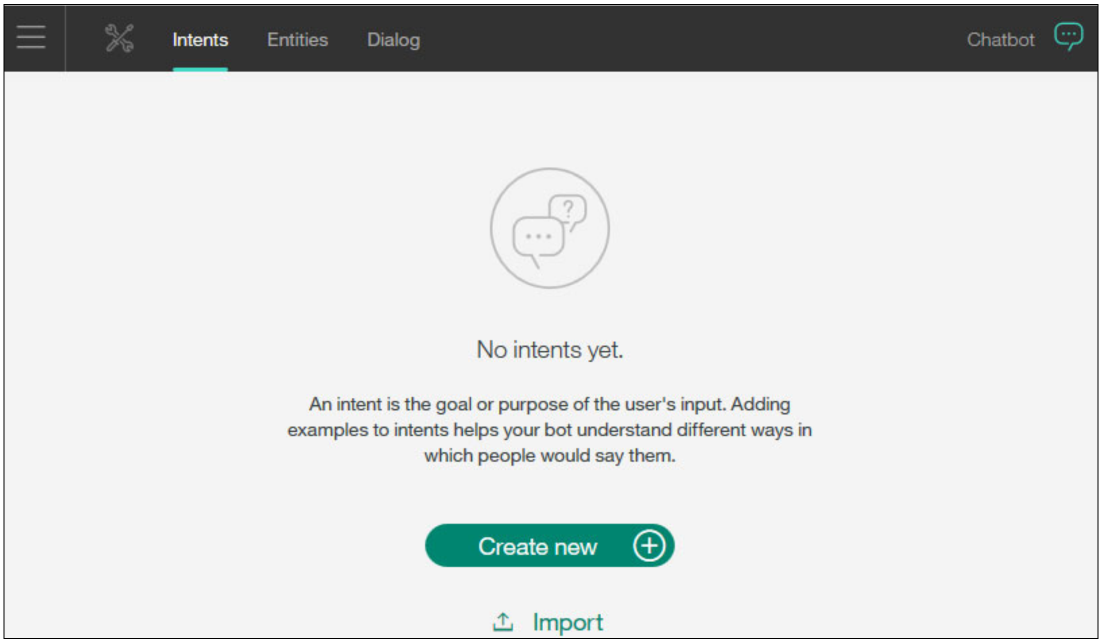
Chatbot application

Language

English (U.S.)

Figure 4-6 Create a workspace

The new Conversation workspace is created (Figure 4-7).



No intents yet.

An intent is the goal or purpose of the user's input. Adding examples to intents helps your bot understand different ways in which people would say them.

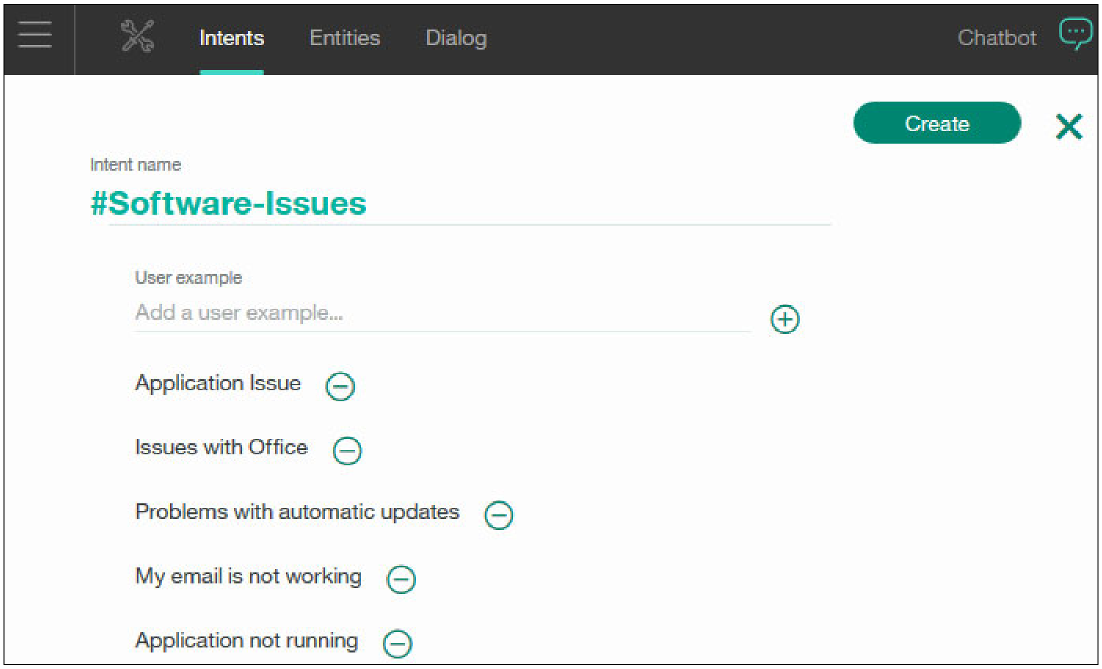
Figure 4-7 Watson Conversation workspace

For more information about creating Conversation workspaces, see 2.1.1, “Creating a Watson Conversation service instance” on page 14.

4.4.2 Adding intents

In this section, you add intents to the Chatbot workspace. The intents should be appropriate for the Help Desk Assistant chatbot. For more information about adding intents to a Conversation workspace, see 2.1.4, “Adding intents” on page 23.

Add the four intents that are shown in Figure 4-8 through Figure 4-11 on page 116.



Intent name
#Software-Issues

User example
Add a user example...

Application Issue

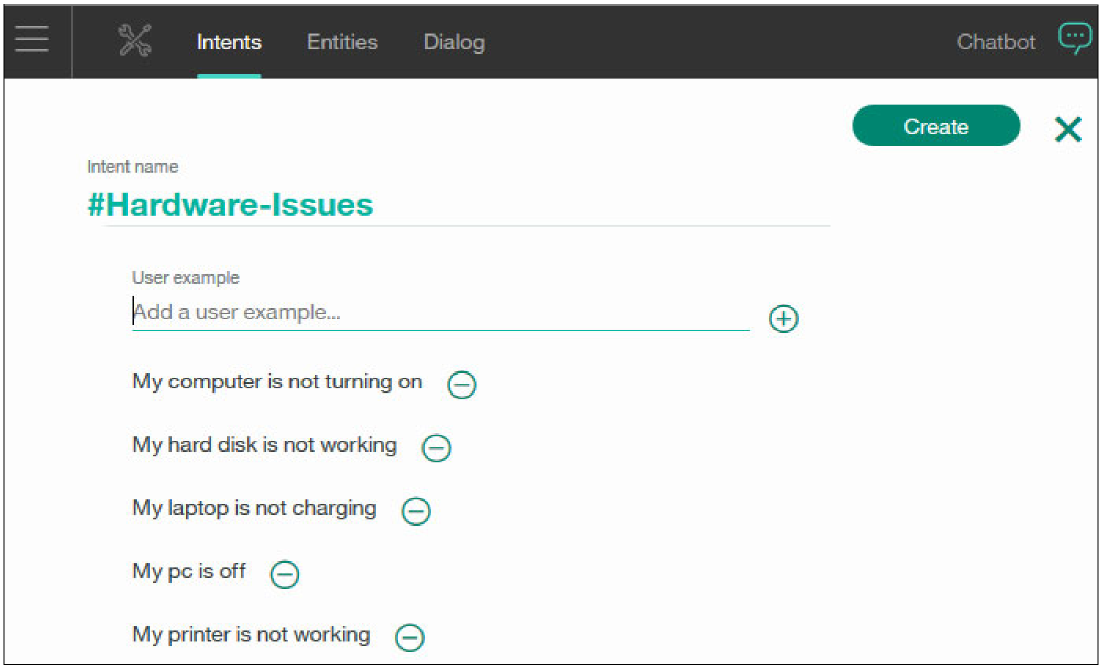
Issues with Office

Problems with automatic updates

My email is not working

Application not running

Figure 4-8 Add #Software-Issues intent (part 1 of 4)



Intent name
#Hardware-Issues

User example
Add a user example...

My computer is not turning on

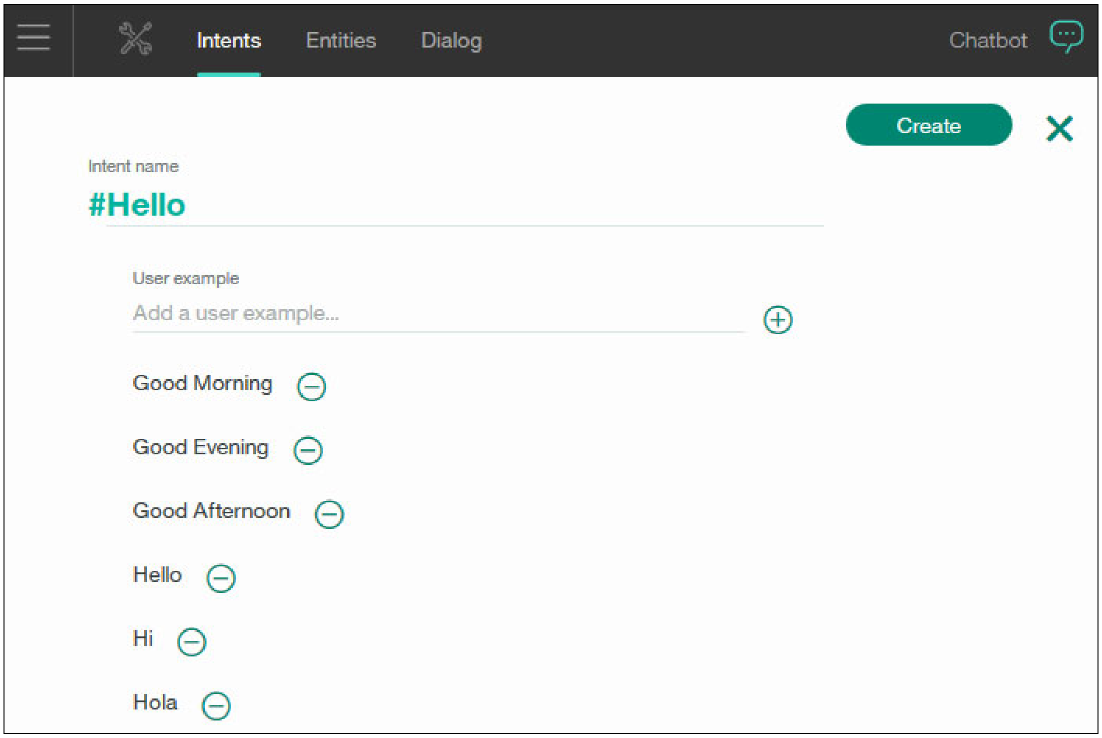
My hard disk is not working

My laptop is not charging

My pc is off

My printer is not working

Figure 4-9 Add #Hardware-Issues intent (part 2 of 4)

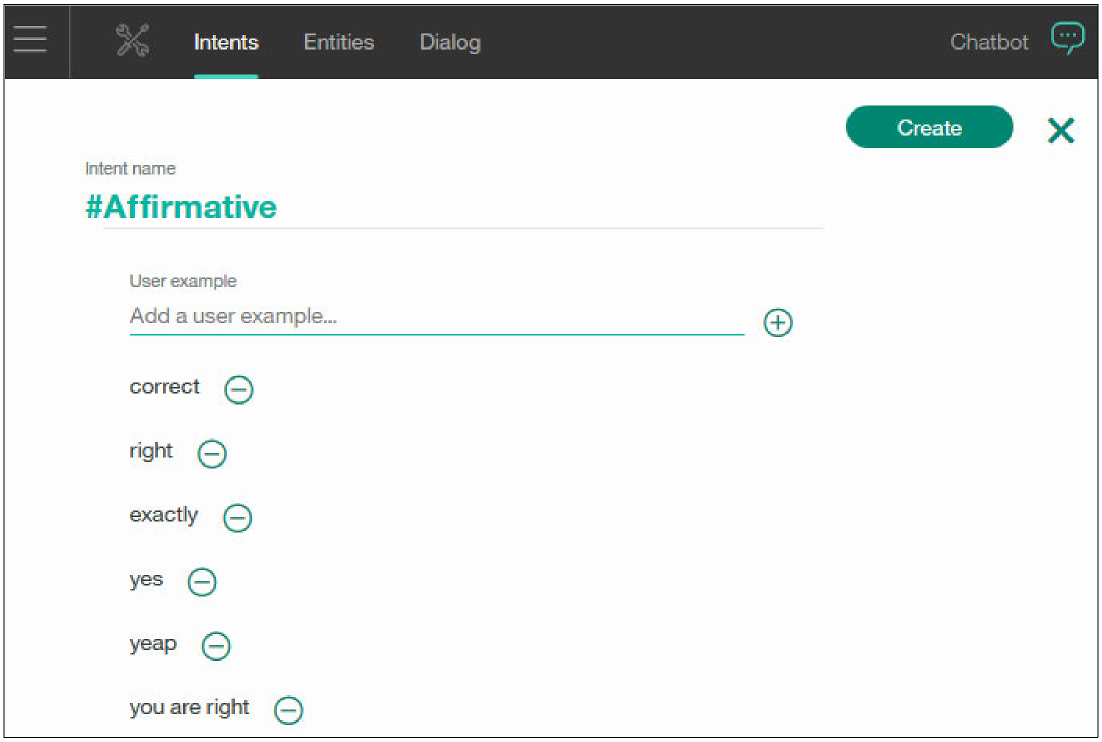


Intent name
#Hello

User example
Add a user example...

Good Morning
Good Evening
Good Afternoon
Hello
Hi
Hola

Figure 4-10 Add #Hello intent (part 3 of 4)



Intent name
#Affirmative

User example
Add a user example...

correct
right
exactly
yes
yeap
you are right

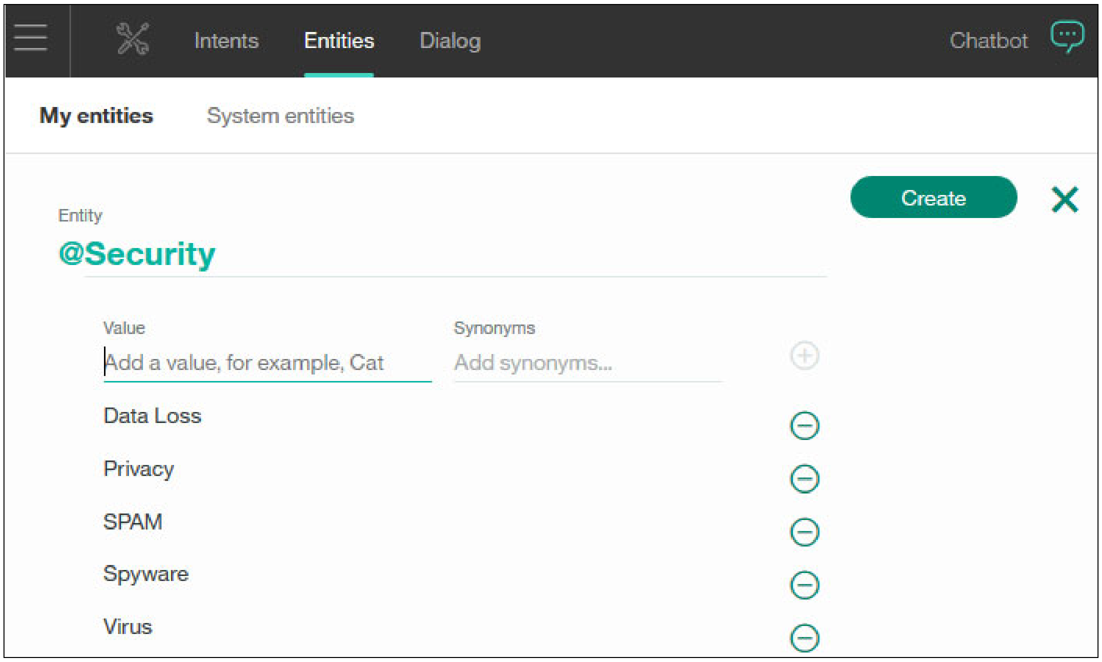
Figure 4-11 Add #Affirmative intents (part 4 of 4)

Those intents are enough for this example; however, you can create as many as you want. Some examples include OutOfScope (for incomprehensible user input), Bye (to close the conversation), and others.

4.4.3 Adding entities

In this section, you add entities to the Chatbot workspace. The entities should be appropriate for the Help Desk Assistant chatbot. For more information about adding entities to a Conversation workspace, see 2.1.5, “Adding entities” on page 27.

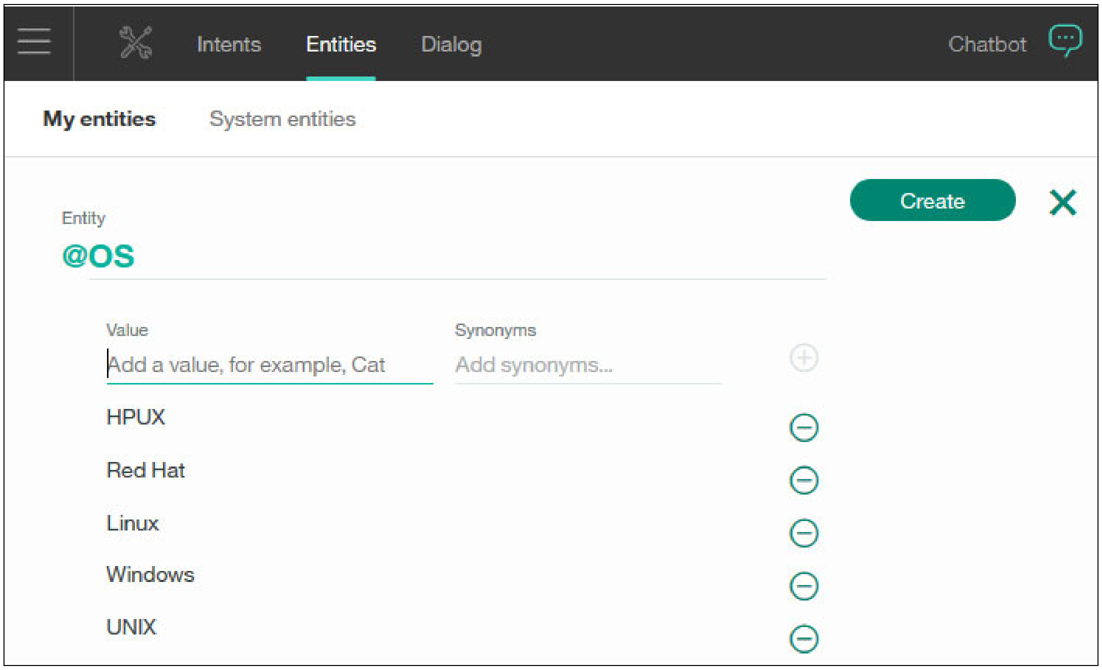
Select **Entities** and create the four entities that are shown in Figure 4-12 through Figure 4-15 on page 118.



Entity
@Security

Data Loss
Privacy
SPAM
Spyware
Virus

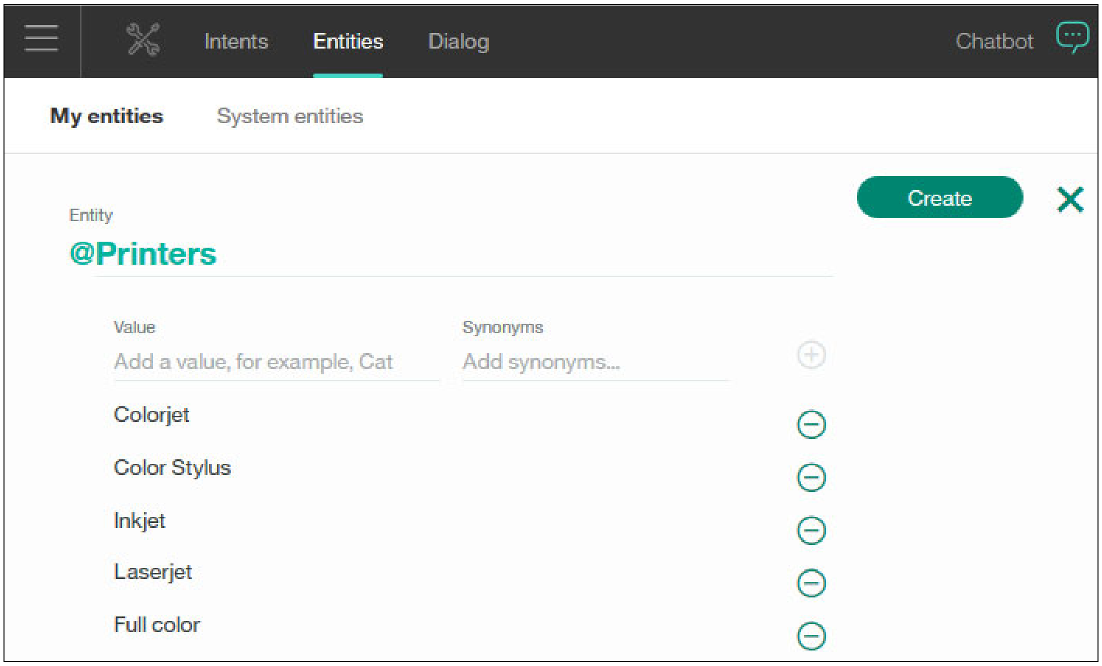
Figure 4-12 Add @Security entity (part 1 of 4)



Entity
@OS

HPUX
Red Hat
Linux
Windows
UNIX

Figure 4-13 Add @OS entity (part 2 of 4)



Entity

@Printers

Colorjet

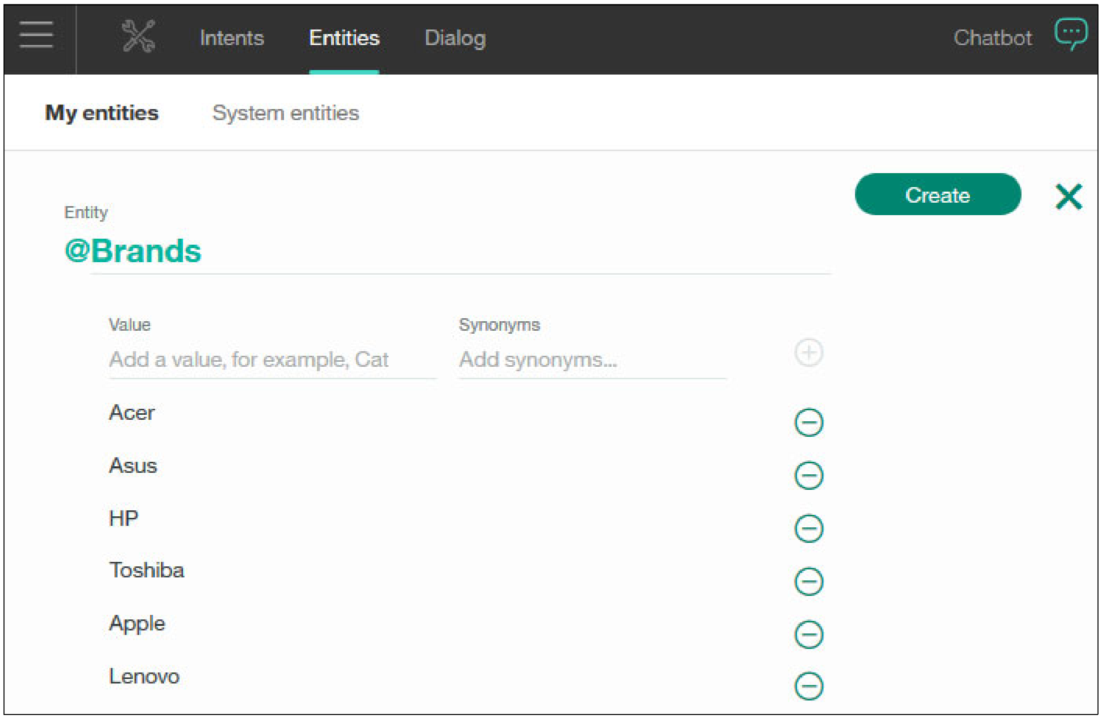
Color Stylus

Inkjet

Laserjet

Full color

Figure 4-14 Add @Printers entity (part 3 of 4)



Entity

@Brands

Acer

Asus

HP

Toshiba

Apple

Lenovo

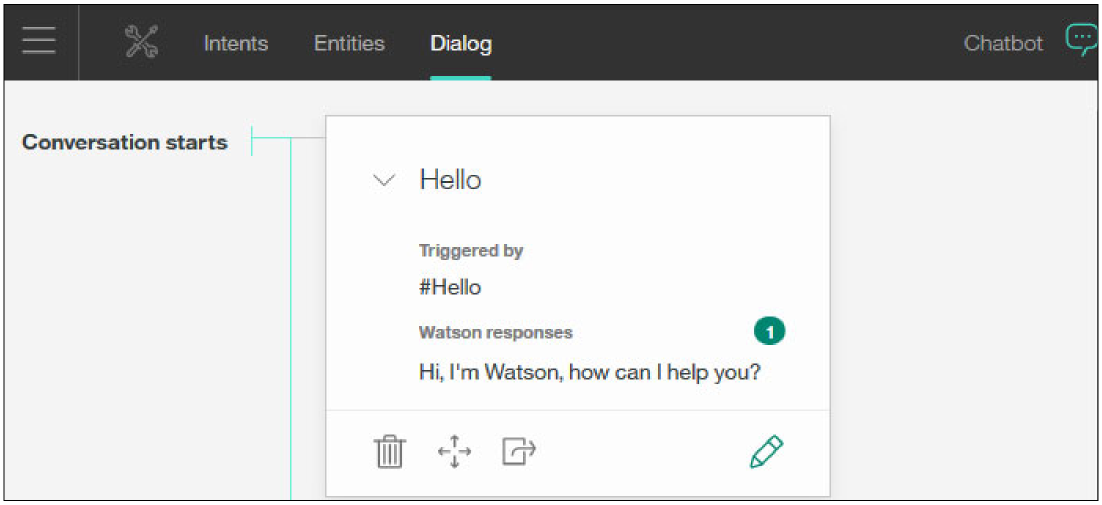
Figure 4-15 Add @Brands entity (part 4 of 4)

4.4.4 Creating the dialog

In this section, you build the Conversation dialog for the Help Desk Assistant chatbot by using the intents and entities created in the previous sections. For more information about building a dialog, see 2.1.6, “Building a dialog” on page 30.

Complete the following steps:

1. Select **Dialog** and create the base node `Hello` as shown in Figure 4-16.



Hello

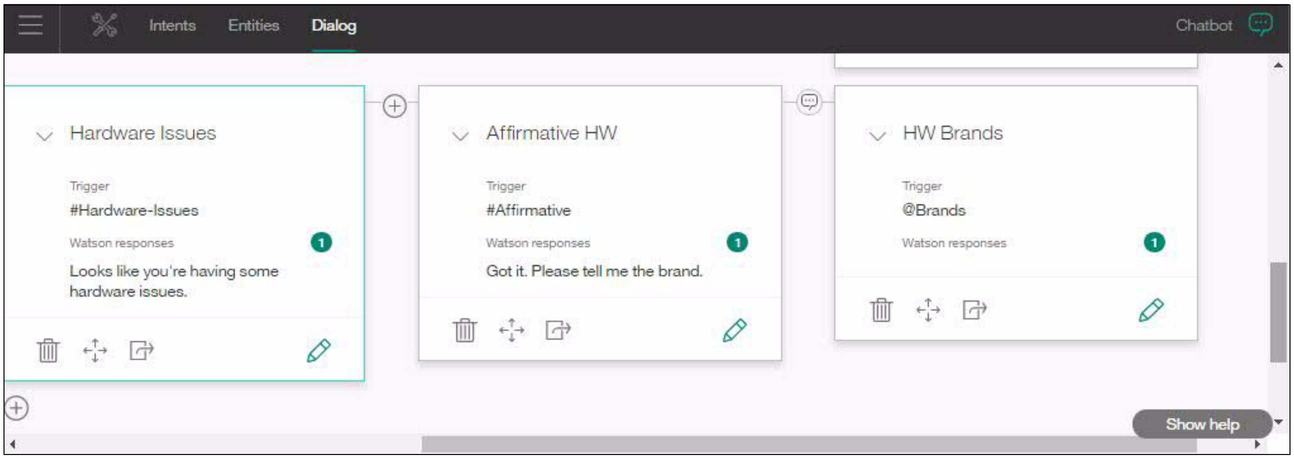
Triggered by
#Hello

Watson responses 1
Hi, I'm Watson, how can I help you?

Figure 4-16 Create the dialog: base node Hello (part 1 of 4)

2. Create the dialog branch shown in Figure 4-17 with the following nodes:

   – Hardware Issues (parent)
   – Affirmative HW (child of Hardware Issues)
   – HW Brands (child of Affirmative HW)



Hardware Issues
Trigger
#Hardware-Issues
Watson responses 1
Looks like you're having some hardware issues.

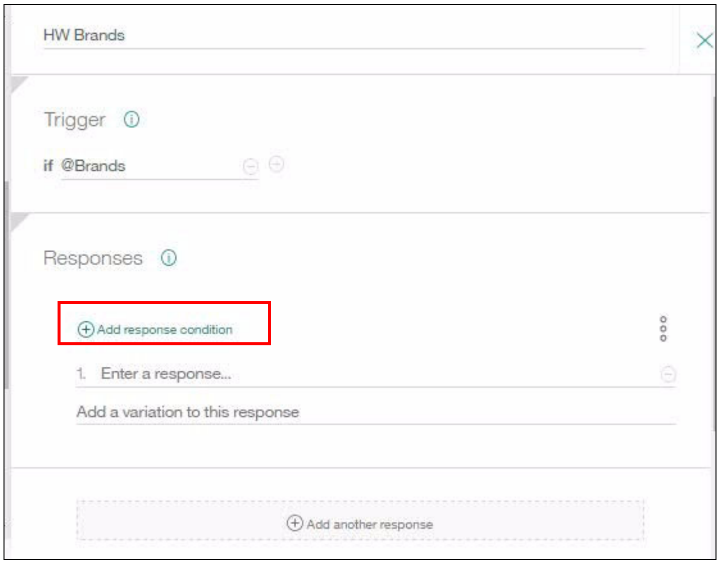
Affirmative HW
Trigger
#Affirmative
Watson responses 1
Got it. Please tell me the brand.

HW Brands
Trigger
@Brands
Watson responses 1

Figure 4-17 Adding Hardware Issues, Affirmative HW, and HW Brands nodes (part 2 of 4)

3. In the HW Brands node, create a response for each example in the @Brands entity (Acer, Asus, HP, Toshiba, Apple, Lenovo, and so on):

   a. Click the **HW Brands** node and then click **Add response condition (**Figure 4-18).

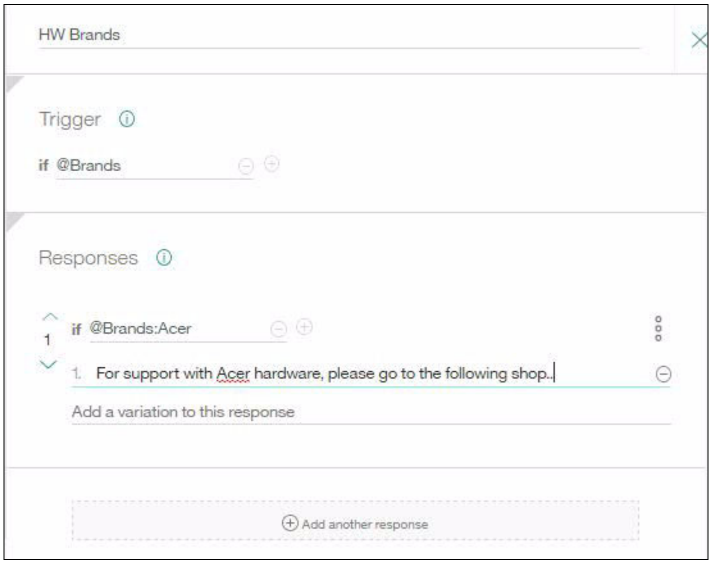


HW Brands

if @Brands

Figure 4-18 Add response condition (part 3 of 4)

   b. Enter the appropriate response for each example in the @Brands entity (Figure 4-19).



HW Brands

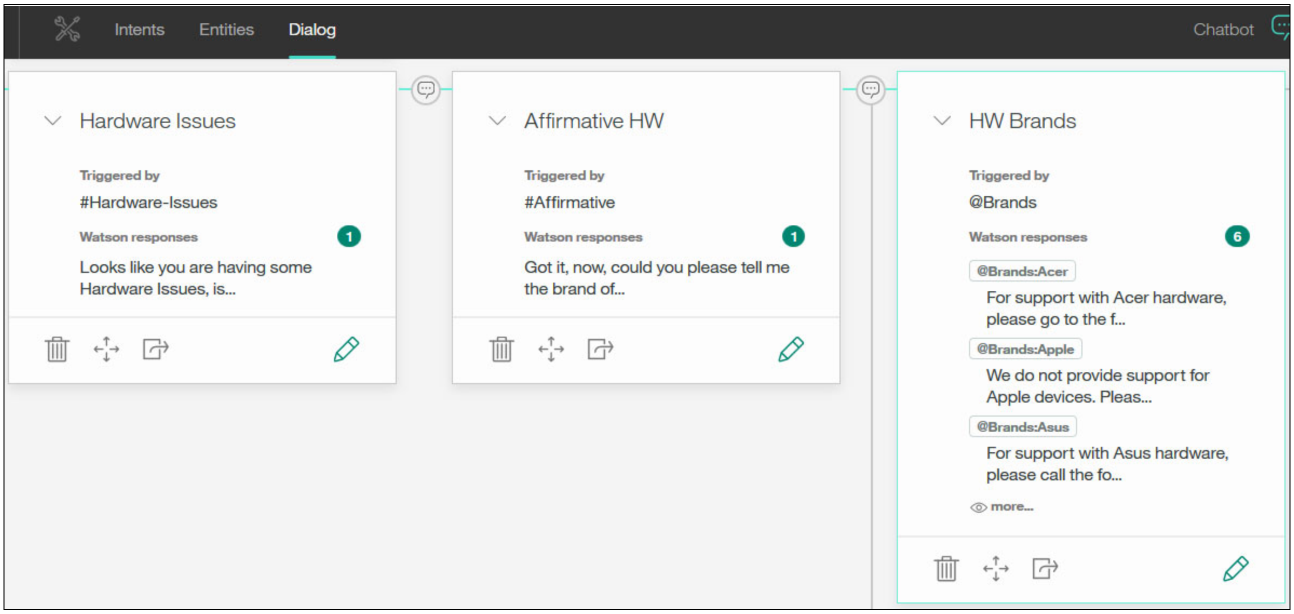
if @Brands

1 if @Brands:Acer

1. For support with Acer hardware, please go to the following shop..

Figure 4-19 Adding a response if brand is Acer part 4 of 4)

Figure 4-20 shows the dialog branch built in this example for hardware issues.



Hardware Issues

Triggered by

#Hardware-Issues

Watson responses 1

Looks like you are having some Hardware Issues, is...

Affirmative HW

Triggered by

#Affirmative

Watson responses 1

Got it, now, could you please tell me the brand of...

HW Brands

Triggered by

@Brands

Watson responses 6

@Brands:Acer

For support with Acer hardware, please go to the f...

@Brands:Apple

We do not provide support for Apple devices. Pleas...

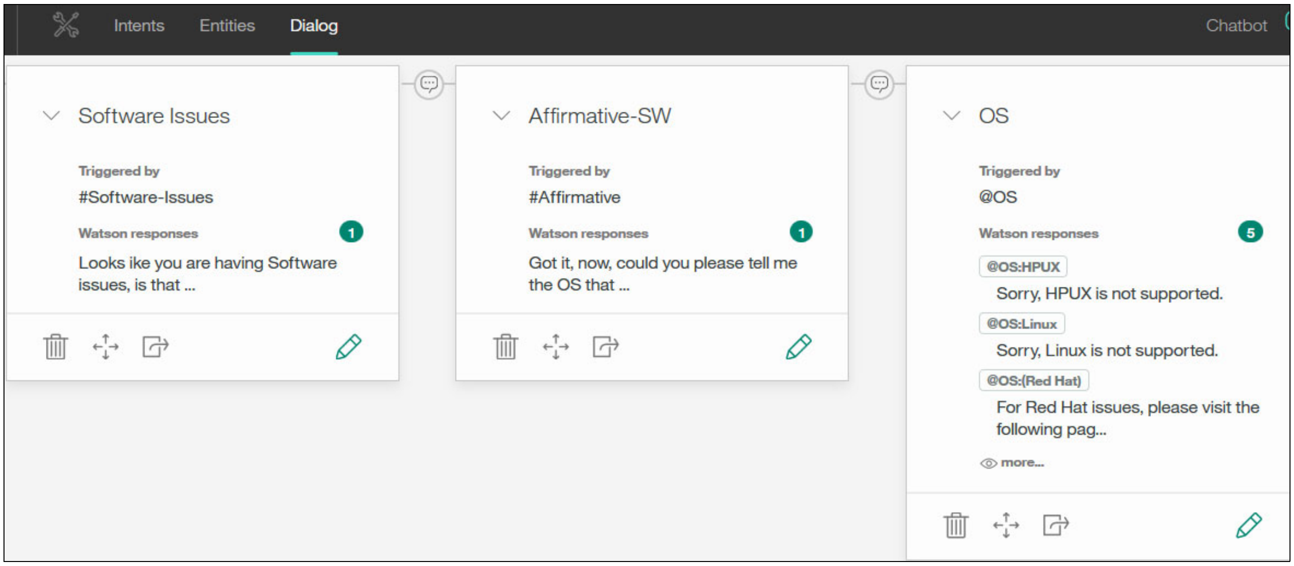
@Brands:Asus

For support with Asus hardware, please call the fo...

more...

Figure 4-20 Dialog branch for hardware issues

4. Repeat the process described in step 2 on page 119 and step 3 on page 120 for software issues. In the OS node, create a response for each example in the @OS entity (HPUX, Red Hat, Linux, Windows, UNIX, and so on).



Software Issues

Triggered by

#Software-Issues

Watson responses 1

Looks ike you are having Software issues, is that ...

Affirmative-SW

Triggered by

#Affirmative

Watson responses 1

Got it, now, could you please tell me the OS that ...

OS

Triggered by

@OS

Watson responses 5

@OS:HPUX

Sorry, HPUX is not supported.

@OS:Linux

Sorry, Linux is not supported.

@OS:(Red Hat)

For Red Hat issues, please visit the following pag...

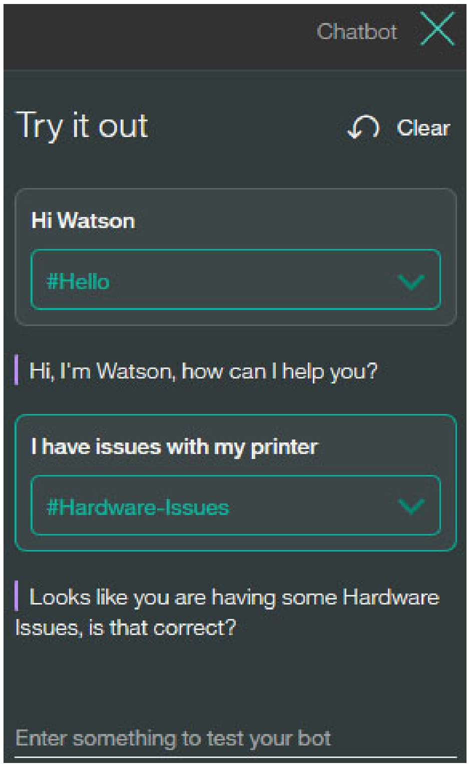
more...

Figure 4-21 Dialog branch for software issues

4.4.5 Testing the dialog

To test the dialog, first click the **Ask Watson** icon 💬 (upper right corner).

The Chatbot panel opens (Figure 4-22). Interact with the chatbot by asking questions to test the responses.



Hi Watson

#Hello

Hi, I'm Watson, how can I help you?

I have issues with my printer

#Hardware-Issues

Looks like you are having some Hardware Issues, is that correct?

Figure 4-22 Test the dialog

4.4.6 Creating the Help Desk Assistant chatbot application in Node-RED

Node-RED is a useful tool to create applications without having to write code. Instead, it uses simple visual components that you configure and connect.

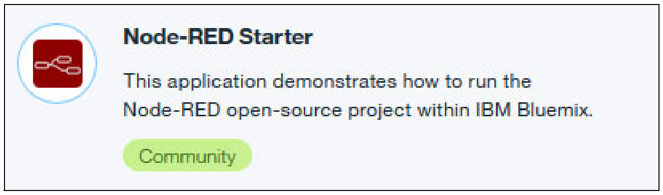
To make this task even easier, you do not need to install Node-RED, because it is available in Bluemix. In this section, you create a Node-RED application and configure the flow:

- Create the Node-RED application in Bluemix
- Create the Help Desk Assistant chatbot application flow with the Node-RED flow editor
- Configure the Help Desk Assistant chatbot application in Node-RED

Create the Node-RED application in Bluemix

Complete the following steps:

1. Go to the Bluemix catalog.
2. In the catalog, go to **Apps** → **Boilerplates** and click **Node-RED Starter** (Figure 4-23).

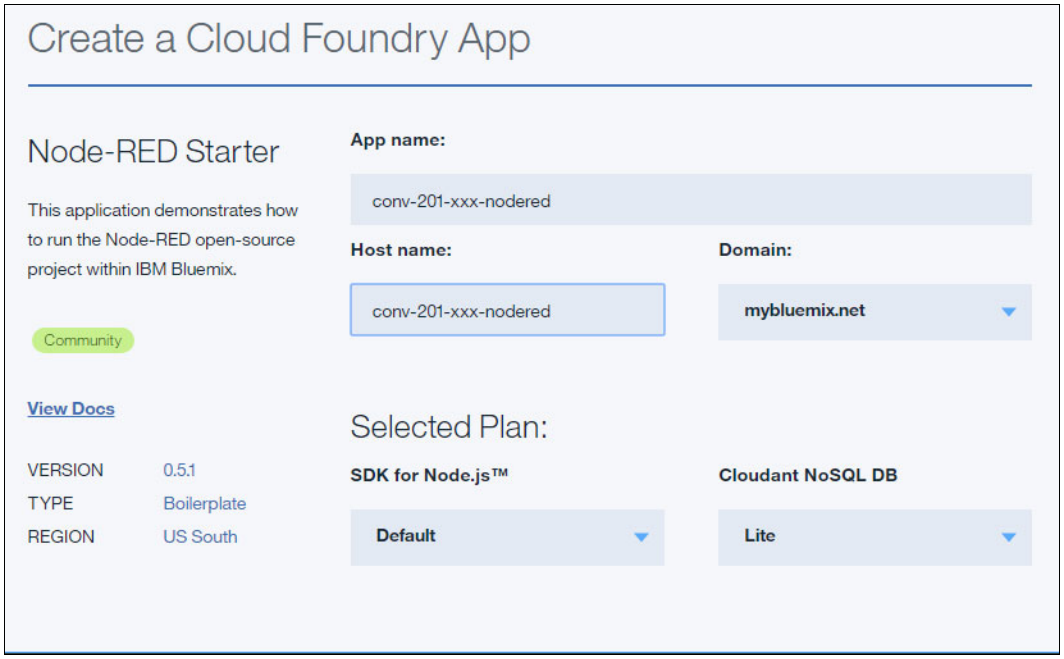


Node-RED Starter

This application demonstrates how to run the Node-RED open-source project within IBM Bluemix.

Figure 4-23 Node-RED Starter app in Bluemix

3. Enter the name of your application and host as `conv-201-xxx-nodered`. Replace `xxx` with any random key because the host name of the application must be unique (Figure 4-24). Accept the default values for the remaining fields and click **Create**.



Create a Cloud Foundry App

Node-RED Starter

This application demonstrates how to run the Node-RED open-source project within IBM Bluemix.

App name: conv-201-xxx-nodered

Host name: conv-201-xxx-nodered

Domain: mybluemix.net

Selected Plan:

SDK for Node.js™: Default

Cloudant NoSQL DB: Lite

VERSION 0.5.1

TYPE Boilerplate

REGION US South

Figure 4-24 Creating a Node-RED application instance

> **Note:** Wait until the application is created and it is started. The application status should be `Running` before you can proceed.

4. While you are waiting for the status to change to `Running` (with a green dot as shown Figure 4-26), read through the `Start coding with Node-RED` information displayed on the page. Also, be sure to record the link to your application (Figure 4-25) because you will need it during the Slack configuration.

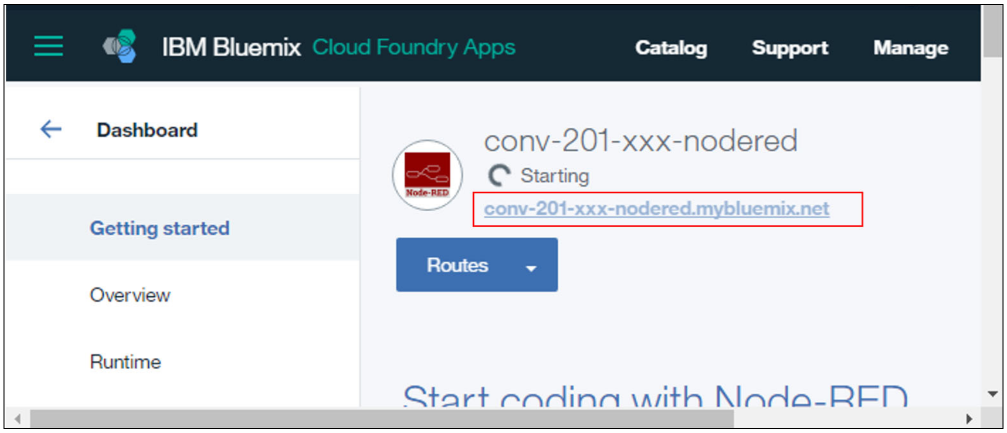


Figure 4-25 The link to your Bluemix application

5. After the application starts, click the route URL (highlighted in Figure 4-26).

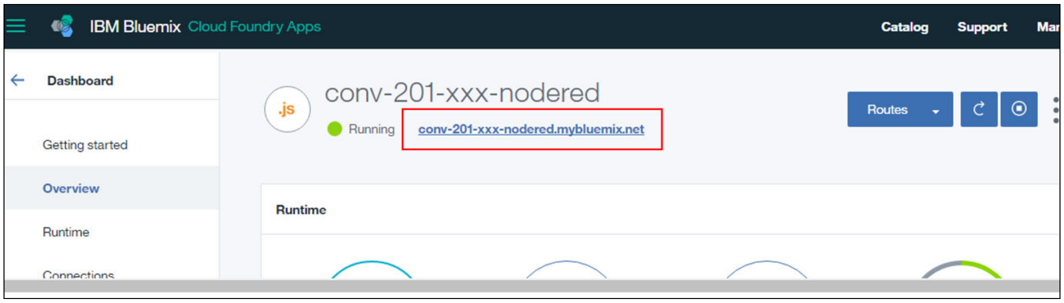


Figure 4-26 Launch your Node-RED instance

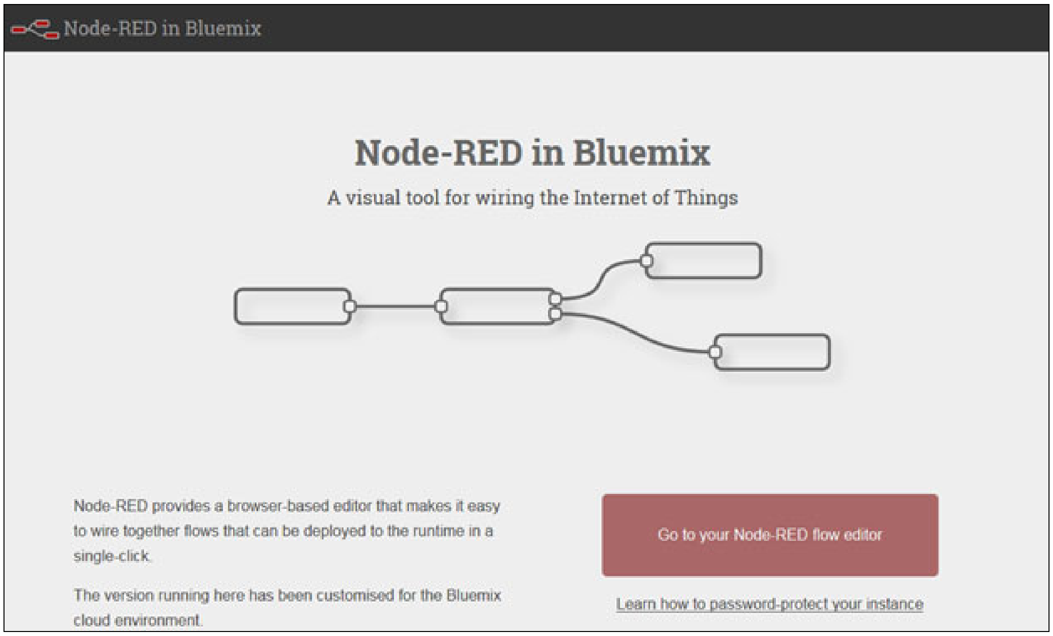
The window shown in Figure 4-27 on page 125 opens. The Node-RED starter application is created.

Create the Help Desk Assistant chatbot application flow with the Node-RED flow editor

Now you can start to create flows. You use the Node-RED flow editor to add nodes and values and create and *wire* (connect) the flows:

1. Click **Go to your Node-RED flow editor** (Figure 4-27).

> **Note:** When you first run this application you are presented with some options to secure the Node-RED flow editor with a username and password. Securing the editor is optional but it is a good practice to do so. Skip through optional windows for this example until you get to the window shown in Figure 4-27.



Node-RED in Bluemix

Node-RED in Bluemix

A visual tool for wiring the Internet of Things

Node-RED provides a browser-based editor that makes it easy to wire together flows that can be deployed to the runtime in a single-click.

The version running here has been customised for the Bluemix cloud environment.

Go to your Node-RED flow editor
Learn how to password-protect your instance

Figure 4-27 Open the Node-RED flow editor

The Node-RED flow editor opens (Figure 4-28). The panel on the left shows a palette of nodes. You can drag nodes to the workspace and connect them together (wire them) to create an application. After dragging a node to a workspace, you can double-click the node to open the *Edit* (configuration) dialog to provide values for the node.

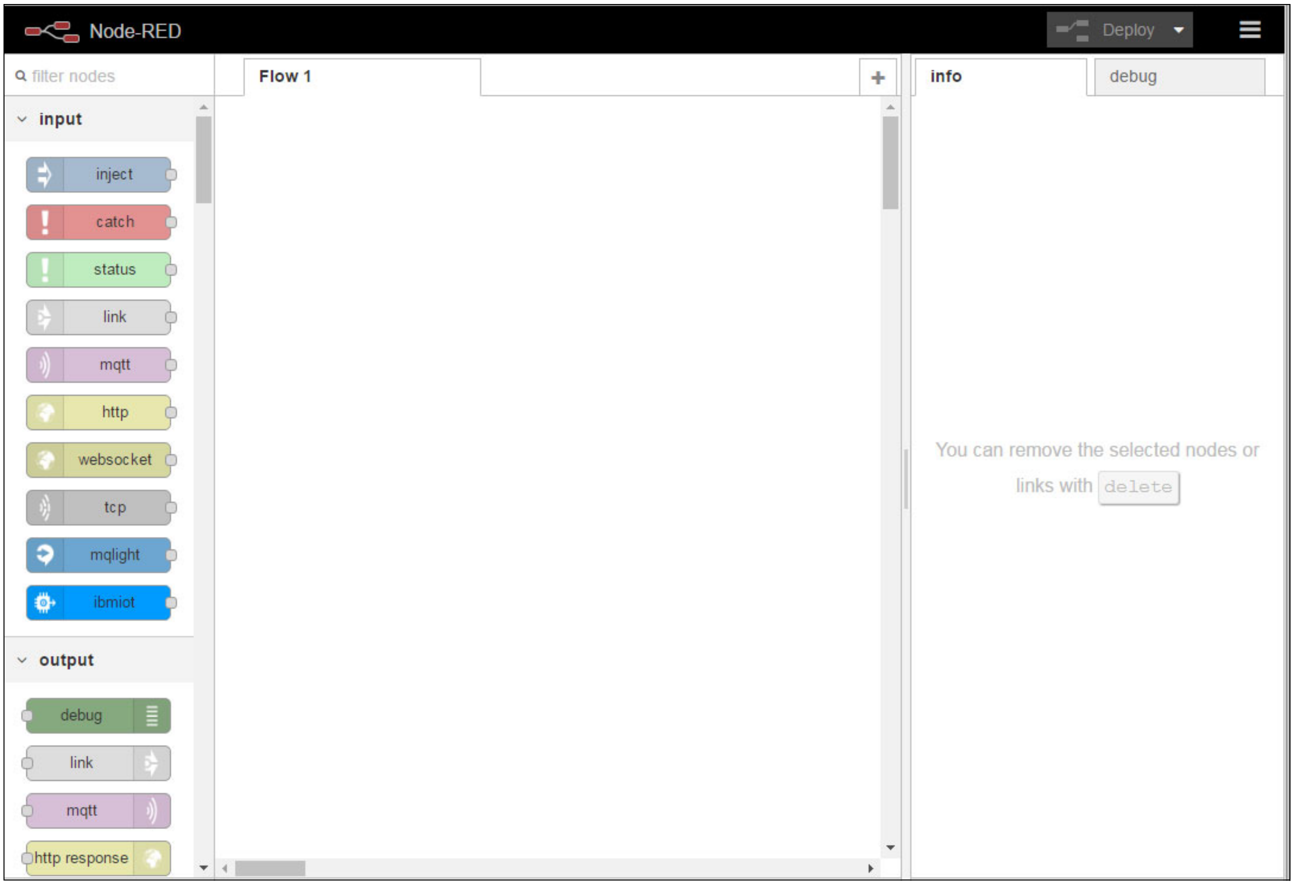


Figure 4-28 Node-RED flow editor workspace

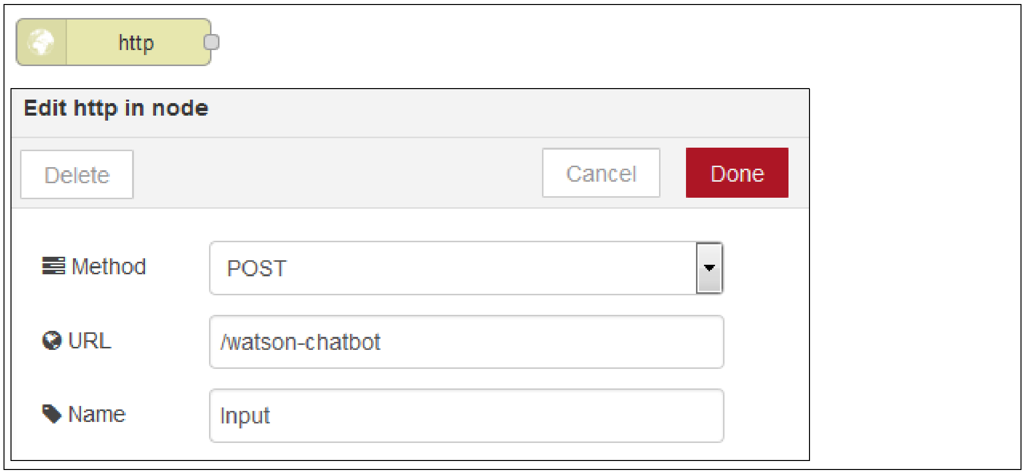
2. In the next steps, drag the following nodes to the workspace, add values as shown in the figures of each step, and then click **Done**:

   a. **http** *input* node (Figure 4-29 on page 127): This node will receive the text that the user submits to the Help Desk Assistant chatbot. Edit the node and add these values:

      - `Method` is the method used to receive the data, POST in this example.
      - `URL` is the last part of the URL (the first part is the route to the Node-RED application as shown in Figure 4-25 on page 124). Enter `/watson-chatbot` for this example. You can customize this value as desired. Just remember that it should always start with a forward slash character (/).

        Remember: You will use this value later in step 8 on page 134, so remember it or keep a record of it.

      - `Name` is the node name (optional)



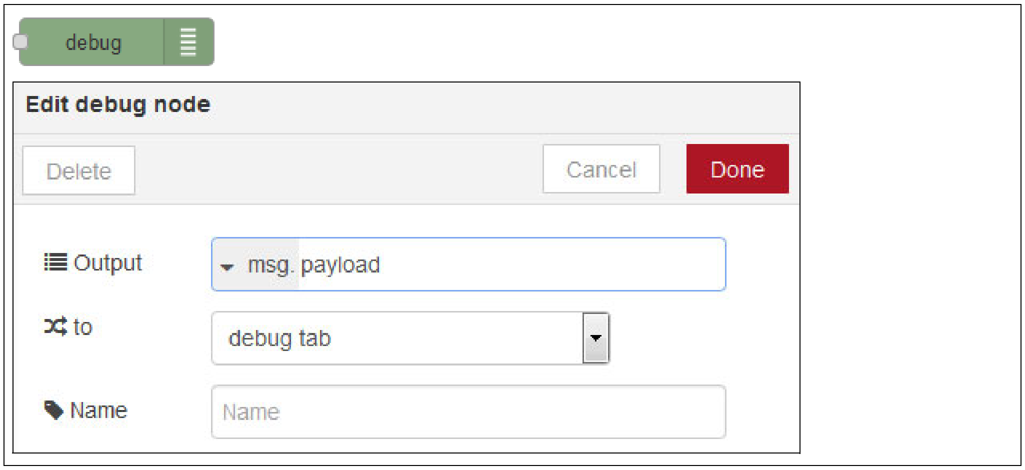
Method: POST

URL: /watson-chatbot

Name: Input

Figure 4-29 Edit http in node

b. **debug** node (Figure 4-30): This node displays the message info (for example, Slack user_id, token, and text) received from Slack. You configure and integrate Slack components later in the chapter. In fact, every time that a user submits text to the Help Desk Assistant chatbot, you can see the information received on the debug panel (at the right of the window). This data is important for troubleshooting and analysis of the flow.



Output: msg. payload

to: debug tab

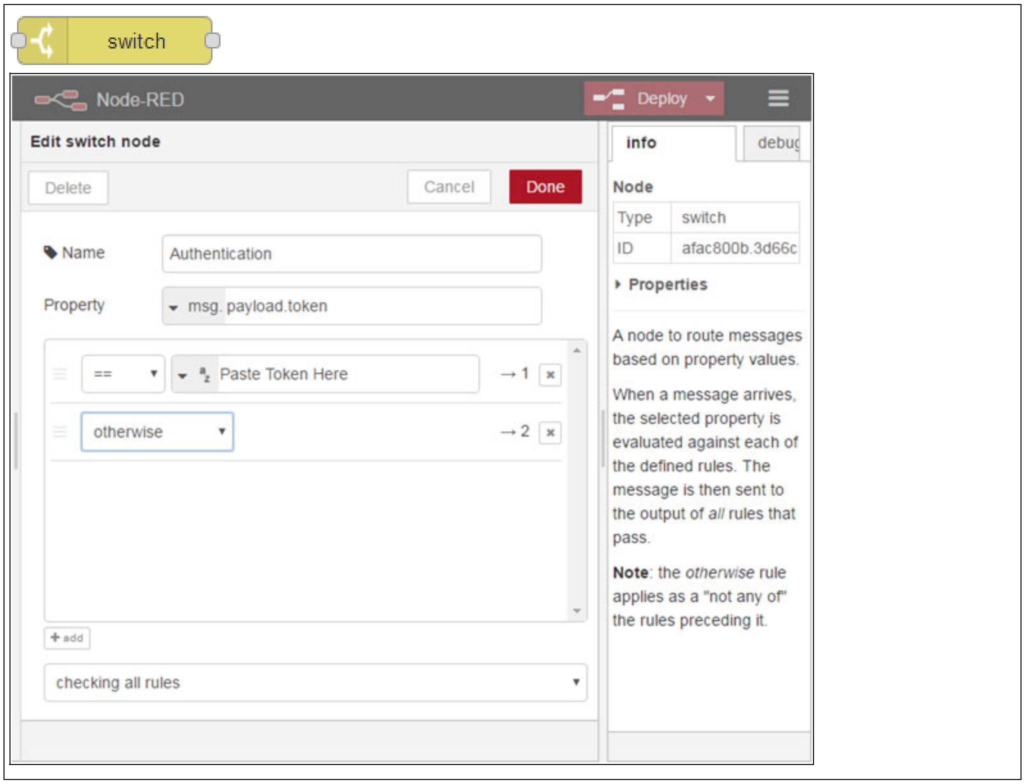
Name: Name

Figure 4-30 Edit debug node

c. **switch** node (Figure 4-31 on page 128): This node is a filter to avoid unauthorized users from using the chatbot.

Add two rules as shown in Figure 4-31 on page 128 which will create two outputs on the node. The token to be pasted in the rule will be created and copied in steps 15 on page 136 and 16 on page 136.

This node routes messages based on the value of the payload. When a message arrives, this node checks the value of the Slack token (contained in `payload.token`) against the values configured in this node. If a match is found then the flow goes to the first output (to continue the flow), otherwise the flow goes to the second output (to exit the flow).



Name: Authentication

Property: msg. payload.token

== Paste Token Here → 1

otherwise → 2

checking all rules

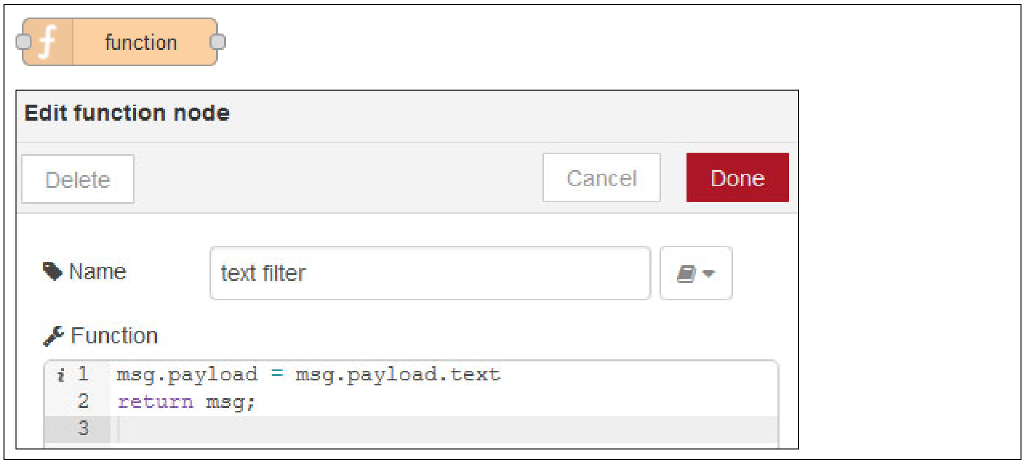
A node to route messages based on property values.

When a message arrives, the selected property is evaluated against each of the defined rules. The message is then sent to the output of *all* rules that pass.

Note: the *otherwise* rule applies as a "not any of" the rules preceding it.

Figure 4-31 Edit switch node

d. **function** node (Figure 4-32): This is the *first* function node you use. Every time a user sends a question to the Help Desk Assistant chatbot, some metadata will be submitted with the text, so this function filters the data to send only the user text to the Conversation service. This example queries just the text from the payload. Be sure you enter the same information as shown in the figure.



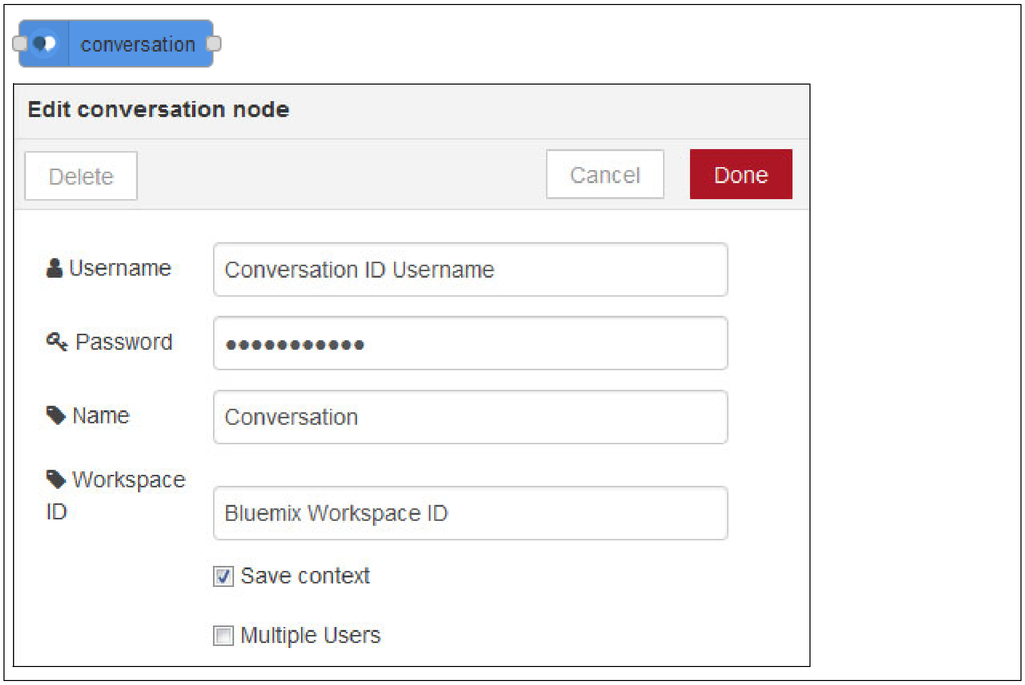
Name: text filter

Function

```
msg.payload = msg.payload.text
return msg;
```

Figure 4-32 Edit function node, 1

e. **conversation** node (Figure 4-33): Here you add the Conversation service and interconnect it to your chatbot application. *Before you can edit the conversation node*, you must gather the credentials and workspace ID as described in the steps after Figure 4-33 (steps i through vi on page 130).



Username	Conversation ID Username
Password	●●●●●●●●●●●
Name	Conversation
Workspace ID	Bluemix Workspace ID

☑ Save context

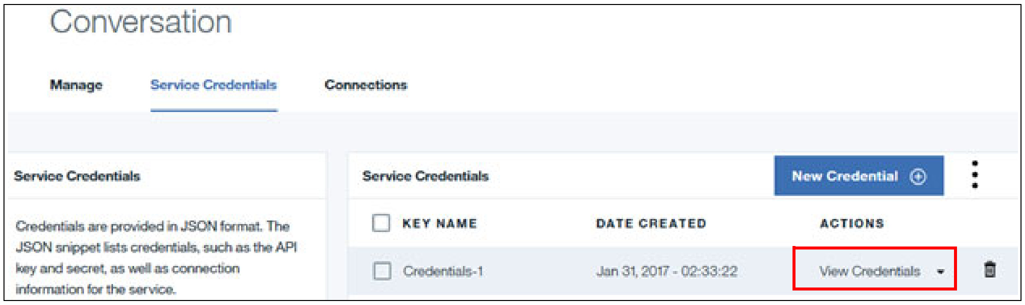
☐ Multiple Users

Figure 4-33 Edit conversation node

Gather the information needed to fill out the values in the conversation node:

i. In another window, open the Bluemix Dashboard, find the Conversation service instance you created in 2.1.1, “Creating a Watson Conversation service instance” on page 14 and click to open it (shown in Figure 4-3 on page 113).

ii. Select **Service Credentials** and click **View Credentials** (Figure 4-34). If you do not yet have any listed credentials, click **New Credential** to create one.



Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

☐ KEY NAME	DATE CREATED	ACTIONS
☐ Credentials-1	Jan 31, 2017 - 02:33:22	View Credentials

Figure 4-34 Watson Conversation credentials

iii. Copy the Username and Password values and paste them in the Node-RED conversation node, as shown in the Edit conversation node window (Figure 4-33).

iv. Click the **Manage** tab and click **Launch tool** to open the Conversation workspace.

v. Find the Chatbot workspace, click the three vertical dots icon (upper right corner as shown in Figure 4-35 on page 130) and select **View details**.

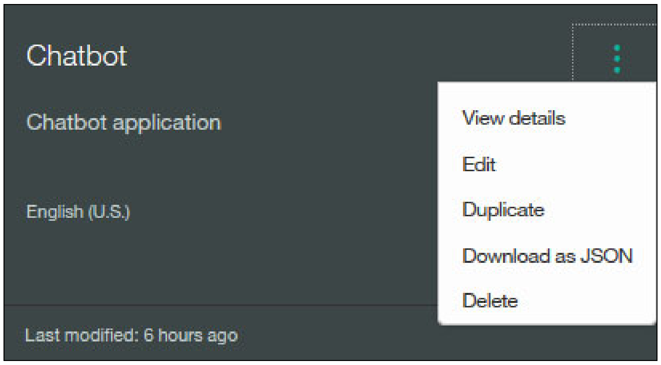
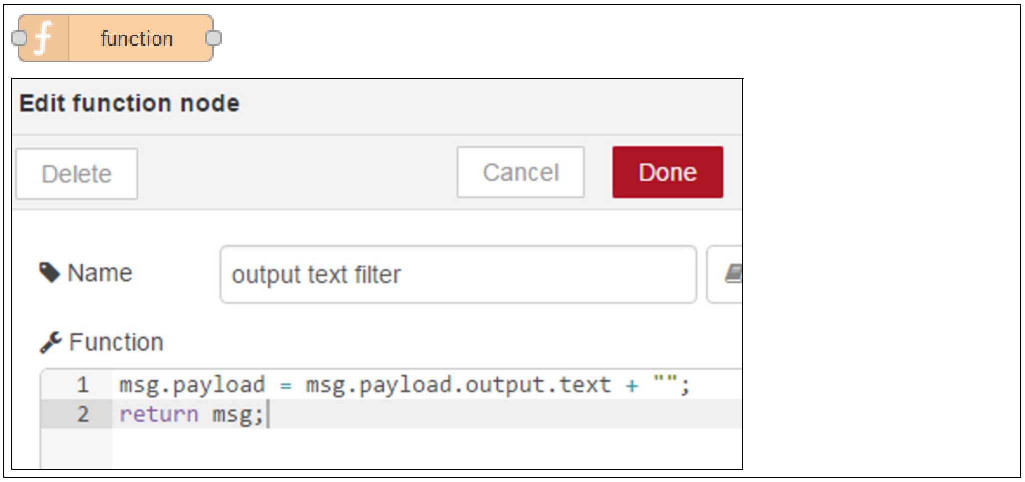


Figure 4-35 Click View details to find the Watson Conversation workspace ID

vi. From the details, copy the Workspace ID and paste it in the Node-RED conversation node, as shown in the Edit conversation node window (Figure 4-33 on page 129).

f. **function** node (Figure 4-36): This is the *second* function. It will filter all the output from the Conversation service and send only the response in the format needed.

Add the values shown in Figure 4-36 (the end of line 1 (+ " ") was added for formatting purposes).



Name: output text filter

Function

```
msg.payload = msg.payload.output.text + "";
return msg;
```

Figure 4-36 Edit function node, 2

g. **http response** node (Figure 4-37): This node takes the response from the Conversion service and sends it back to the chat service (Slack). Add two instances of this node (one for each flow). The configuration for both nodes is the same as shown in Figure 4-37.

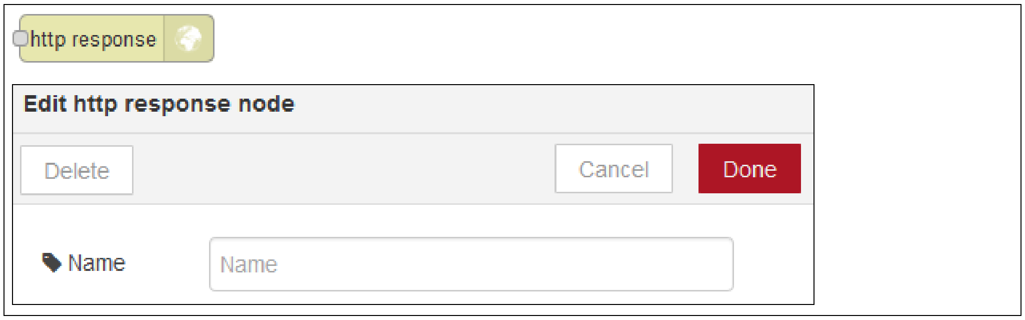


Figure 4-37 Edit http response node

Configure the Help Desk Assistant chatbot application in Node-RED

Now you can connect and configure all the nodes that you dragged to the Node-RED workspace.

Connect the modules (Figure 4-38). To connect each module, click the small grey connector on the edge of the node and drag it to the desired node.

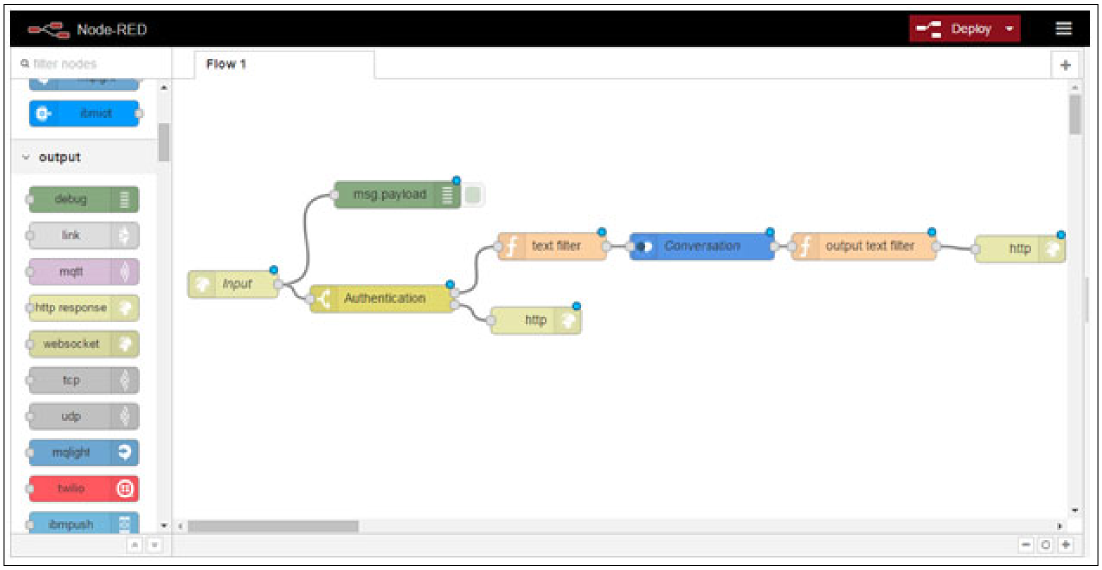


Figure 4-38 Connecting the required nodes for the application

To run your Node-RED application, click **Deploy** at the top right of the window.

4.4.7 Setting up the chat service (Slack)

As described in the architecture of this use case (4.2, “Architecture” on page 111), the chat service (for example, Slack, Facebook Messenger, web app) determines whether the input message from the user is for the Help Desk Assistant chatbot application. If the message is for the chatbot, then the chat service sends the message to your Node-RED application.

This use case uses Slack as an example of a front-end chat service. To configure Slack to work with your Node-RED application, complete the following steps:

1. Sign in to Slack and create a new Slack team if you do not have a team.
2. After you sign in, go to the top of the left panel and click under your room name, and then click **Apps & integrations** (Figure 4-39).

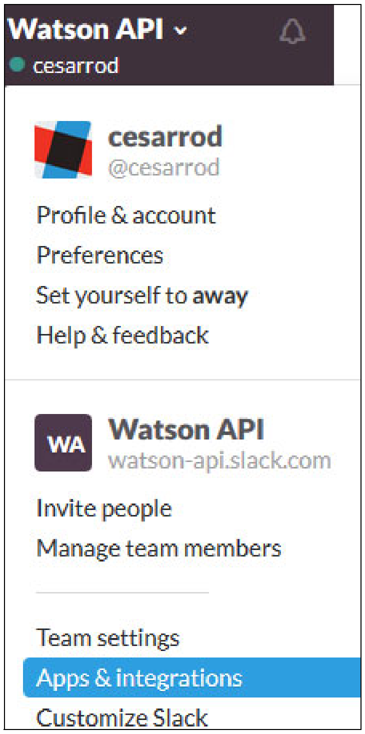


Figure 4-39 The Apps & integrations link

3. At the upper right corner, click **Build** (Figure 4-40).

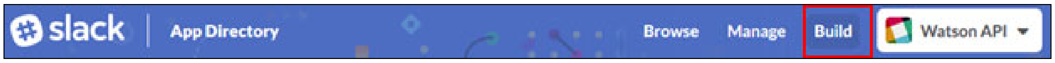


Figure 4-40 Access to build the integration

4. Click **Start Building** to start building the Slack app (Figure 4-41).

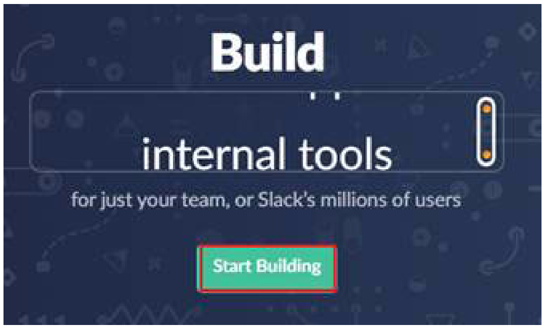
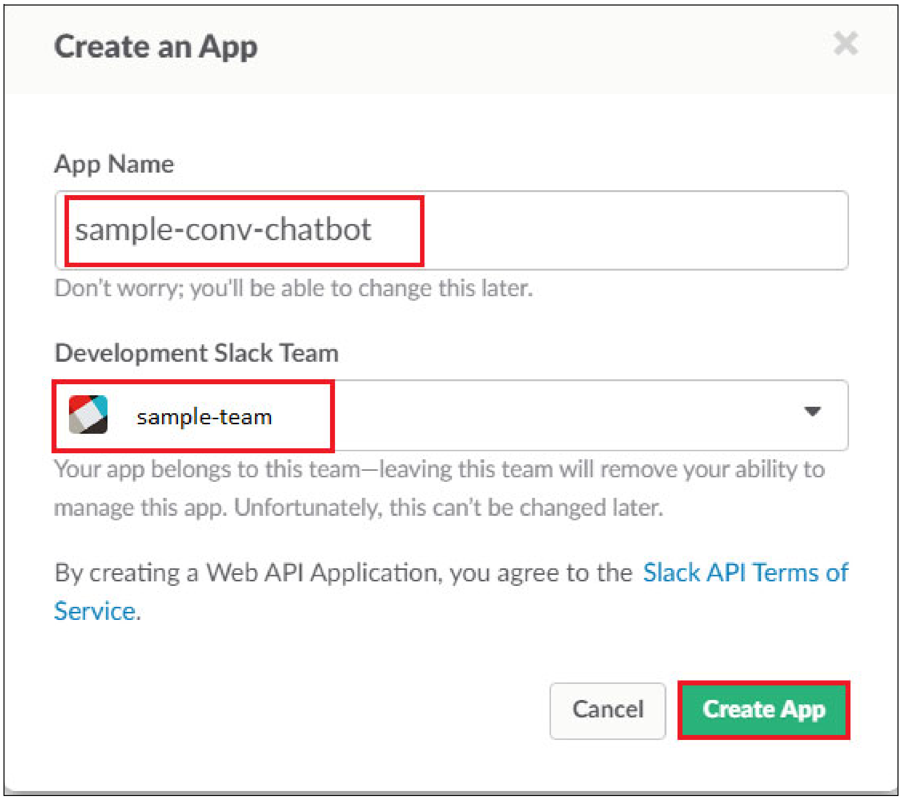


Figure 4-41 Click Start Building

5. The Create an App window opens (Figure 4-42). Enter an app name, select your Slack team, and click **Create App**.



Create an App

App Name

sample-conv-chatbot

Don't worry; you'll be able to change this later.

Development Slack Team

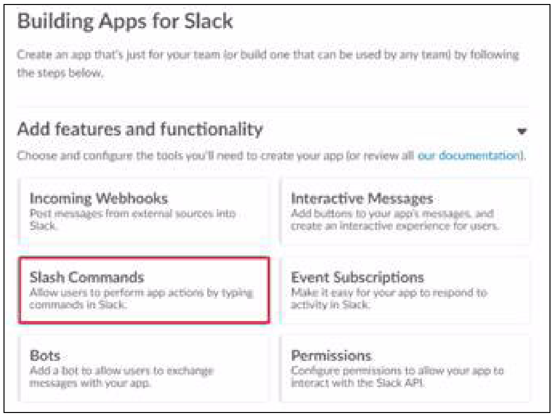
sample-team

Your app belongs to this team—leaving this team will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the Slack API Terms of Service.

Figure 4-42 Create an App in Slack

6. Click **Slash Commands** (Figure 4-43).



Building Apps for Slack

Create an app that's just for your team (or build one that can be used by any team) by following the steps below.

Add features and functionality

Choose and configure the tools you'll need to create your app (or review all our documentation).

Incoming Webhooks
Post messages from external sources into Slack.

Interactive Messages
Add buttons to your app's messages, and create an interactive experience for users.

Slash Commands
Allow users to perform app actions by typing commands in Slack.

Event Subscriptions
Make it easy for your app to respond to activity in Slack.

Bots
Add a bot to allow users to exchange messages with your app.

Permissions
Configure permissions to allow your app to interact with the Slack API.

Figure 4-43 Add features in Slack integration

7. In the next window, click **Create New Command**.

8. In the Create New Command window (Figure 4-44), enter the following information, and then click **Save**:

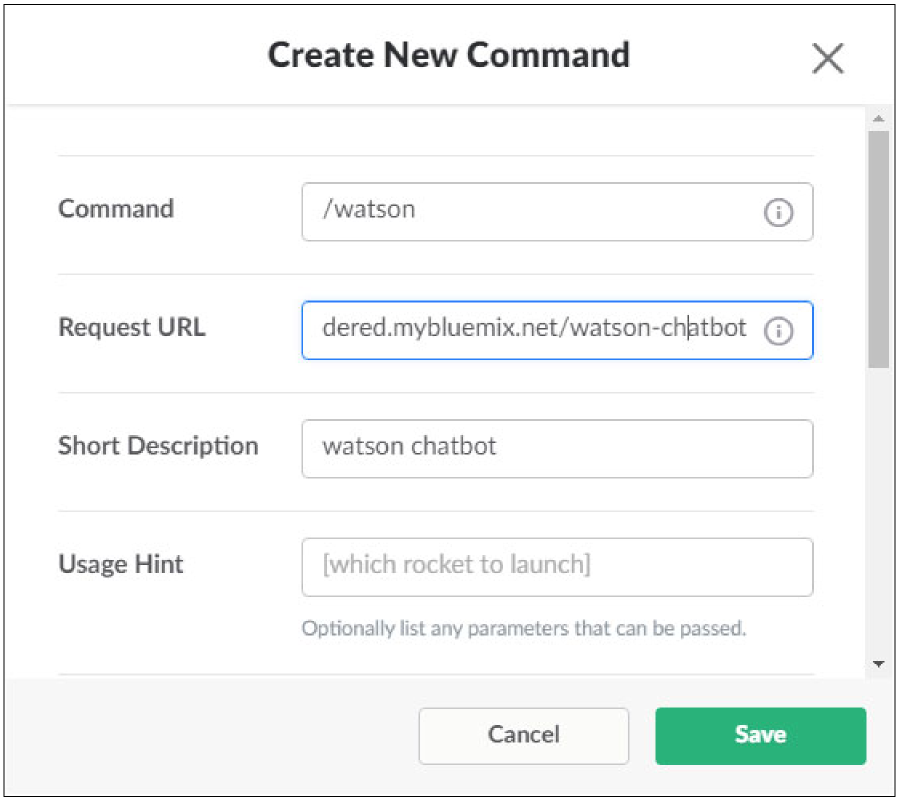
   – Command: `/watson`

     This is the trigger to call `Watson-chatbot` when you type text in Slack.

   – Request URL: `https://conv-201-xxx-nodered.mybluemix.net/watson-chatbot`

     This is the URL of the Node-RED application (`/watson-chatbot`) that you configured in the http input node in step a on page 126 and Figure 4-29 on page 127.

   – Short Description: Any text



Create New Command

Command: /watson

Request URL: dered.mybluemix.net/watson-chatbot

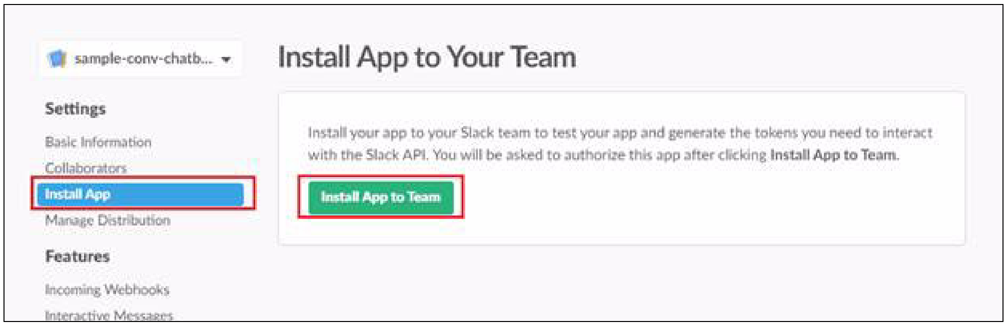
Short Description: watson chatbot

Usage Hint: [which rocket to launch]

Optionally list any parameters that can be passed.

Figure 4-44 Create New Command: Add Slack command

9. Click **Install App**, and then click **Install App to Team** (Figure 4-45).

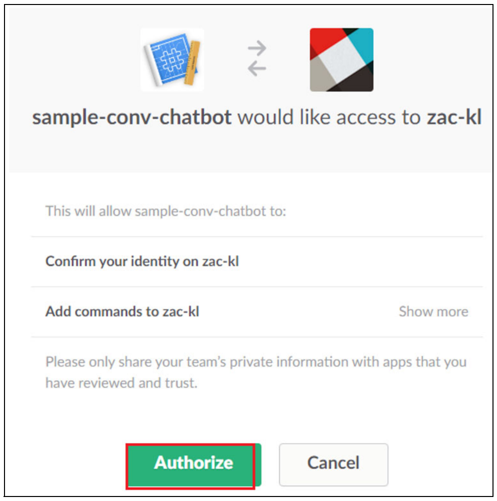


Install App to Your Team

Install your app to your Slack team to test your app and generate the tokens you need to interact with the Slack API. You will be asked to authorize this app after clicking **Install App to Team.**

Figure 4-45 Install the app to the Slack team

10. Click **Authorize** (Figure 4-46).


sample-conv-chatbot would like access to **zac-kl**

This will allow sample-conv-chatbot to:

Confirm your identity on zac-kl

Add commands to zac-kl

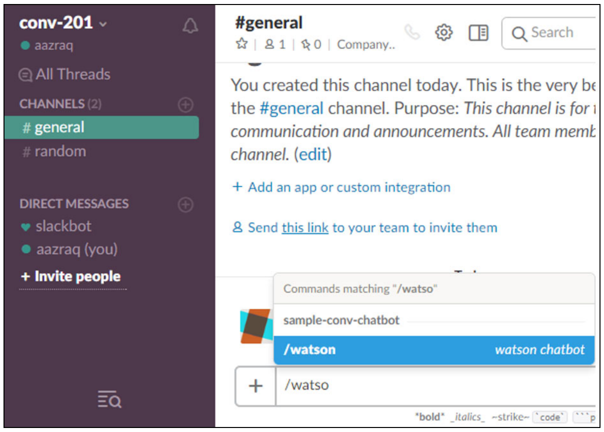
Please only share your team's private information with apps that you have reviewed and trust.

Figure 4-46 Authorize the Slack application

11. Return to the Slack room:

```
http://<room-name>.slack.com
```

12. At any channel (for example the `#general` channel), in the send message text column, type the text `/watson`, and notice the pop-up message (Figure 4-47).


You created this channel today. This is the very be
the #general channel. Purpose: *This channel is for*
communication and announcements. All team memb
channel. (edit)

Commands matching "/watso"

sample-conv-chatbot

/watson *watson chatbot*

/watso

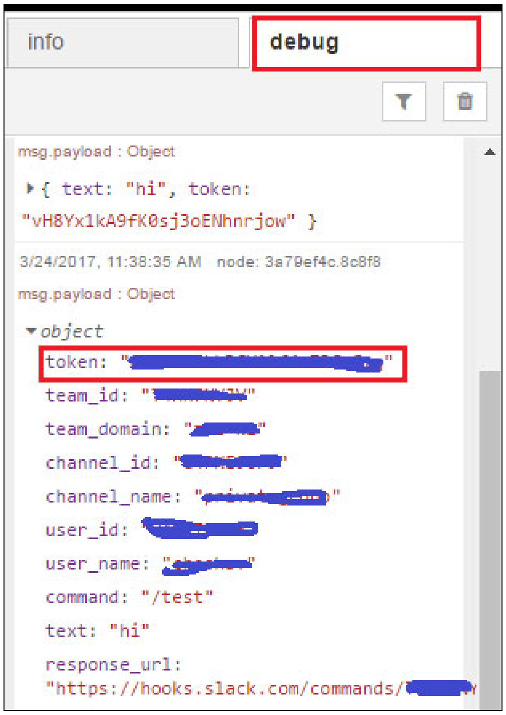
Figure 4-47 Test application

13. Continue typing any message, such as `/watson hi`. For now, the response is only the echo back of the message you send.

14. Go to the Node-RED flow editor:

```
http://<node_red_appname>.mybluemix.net/red
```

15. Click the **debug** tab (Figure 4-48). Notice the `msg.payload` message that contains Slack information including token, command, text, user_name, and others under `object`.



```
msg.payload : Object
▸{ text: "hi", token:
"vH8Yx1kA9fK0sj3oENhnrjow" }
3/24/2017, 11:38:35 AM   node: 3a79ef4c.8c8f8
msg.payload : Object
▾object
   token: "[redacted]"
   team_id: "[redacted]"
   team_domain: "[redacted]"
   channel_id: "[redacted]"
   channel_name: "[redacted]"
   user_id: [redacted]
   user_name: "[redacted]"
   command: "/test"
   text: "hi"
   response_url:
   "https://hooks.slack.com/commands/[redacted]
```

Figure 4-48 Object information in the msg.payload

16. Copy the **token** value (copy only the text inside double quotation marks).

17. Open **switch** node (named Authentication). Paste the token you just copied in the first rule (input box) replacing the text `Paste Token Here` in Figure 4-31 on page 128.

18. Click **Done**, then click **Deploy** (located at the top right).

19. Return to the Slack room:

    `http://<room-name>.slack.com`

20. Now type the text `/watson hi`. Notice that this time, the response is coming from the Conversation service.

4.5 Quick deployment of application

This section provides a quicker way to create the chatbot application in Node-RED if you want to skip many of the steps described in 4.4, “Step-by-step implementation” on page 112:

1. Access the Node-RED Bluemix Starter Application, which is at this GitHub location:

   `https://github.com/snippet-java/Node-RED-bluemix-conversation-starter.git-1487332833126`

2. Scroll to and click **Deploy to Bluemix** (Figure 4-49); then follow the prompts.

Figure 4-49 Click Deploy to Bluemix

3. Open the Node-RED flow editor for your application by entering the following URL in your browser; replace `<HOSTNAME>` with the host name of your application:

   `https://<HOSTNAME>.mybluemix.net/red/`

4. Import the additional nodes developed in this chapter, which are at this GitHub location:

   `https://github.com/snippet-java/redbooks-conv-201-iot-nodered/blob/master/conv-201-iot-nodered-flow.json`

   Copy the content of this file to your clipboard.

5. To import the nodes, click the menu at the top-right and select **Import** → **Clipboard** (Figure 4-50).

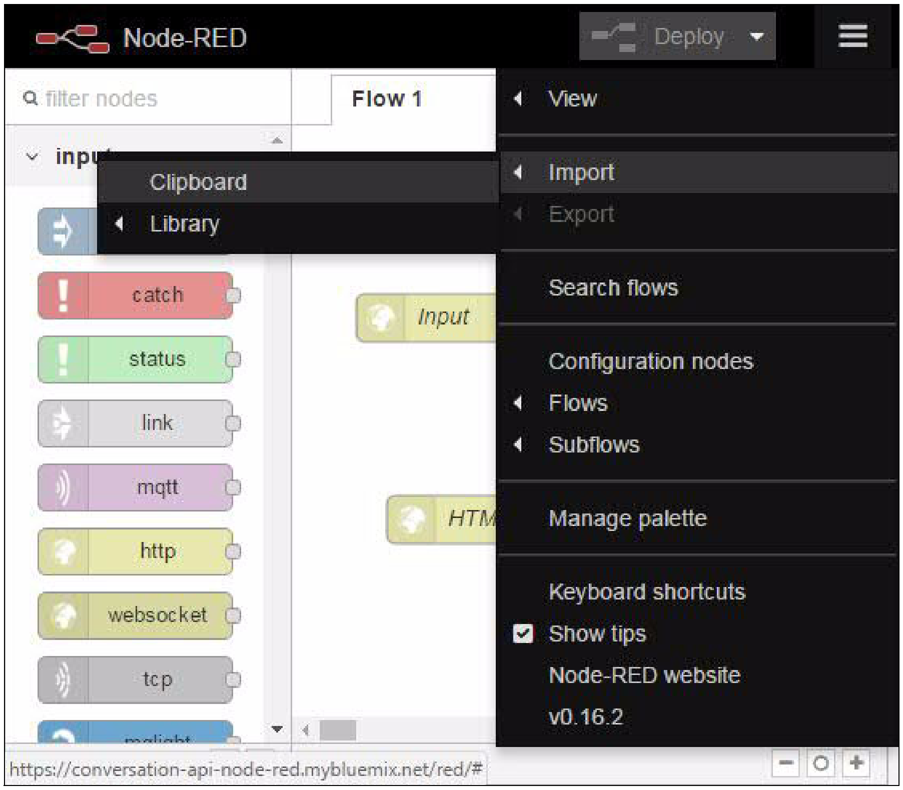


Figure 4-50 Import Node-RED nodes from the clipboard

6. Follow the steps described in these sections:
   - 4.4.1, “Creating a new Conversation workspace” on page 113
   - 4.4.2, “Adding intents” on page 115
   - 4.4.3, “Adding entities” on page 117
   - 4.4.4, “Creating the dialog” on page 119
   - 4.4.7, “Setting up the chat service (Slack)” on page 131

7. Edit the nodes and add the authentication values based on your Conversation service instance credentials, workspace ID (edit conversation node as shown in Figure 4-33 on page 129) and Slack token (edit switch node as shown in Figure 4-31 on page 128).

4.6 Next steps

You can enhance your chatbot. For example, you can add intents, entities, and dialogs.

Also if you identify any unexpected responses, you can make the corrections to improve the answers.

4.7 References

For more information, see the following resources:

- Node-RED:

  `https://nodered.org/`

- Creating apps with Node-RED Starter:

  `https://console.ng.bluemix.net/docs/starters/Node-RED/nodered.html#nodered`

5

Using a cognitive chatbot to manage IoT devices

A cognitive chatbot understands natural language. In Chapter 4, “Help Desk Assistant chatbot” on page 109, you learn how to create a cognitive chatbot to answer questions from users requesting help with software and hardware problems.

In this chapter, you learn to expand the cognitive chatbot capabilities so it can interact with IoT devices and send commands to them in response to user’s requests.

In this use case the Node-RED sample application created in Chapter 4, “Help Desk Assistant chatbot” on page 109 is modified to connect to the Watson Internet of Things Platform service in order to manage a device. The application also integrates the Watson Conversation service to understand the user’s request in natural language.

This example considers a mobile smartphone as an IoT device because getting access to an Android phone for testing purposes is fairly easy. This example can be applied to other IoT devices such as street light sensors, smart meters, sensors to manage household appliances, and so on.

The following topics are covered in this chapter:

- Getting started
- Architecture
- Step-by-step deployment of application
- References

5.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

5.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Create a Watson IoT Platform service instance and connect devices to be managed.
- Integrate the Watson IoT Platform service with the cognitive chatbot application to handle the user's requests and respond to the user.
- Train the Chatbot Conversation workspace with the appropriate intents for understand user's request in natural language to manage IoT devices.
- Add capabilities to the chatbot Node-RED application to send commands to the IoT device.

5.1.2 Prerequisites

To complete the steps in this chapter, be sure these prerequisites are met:

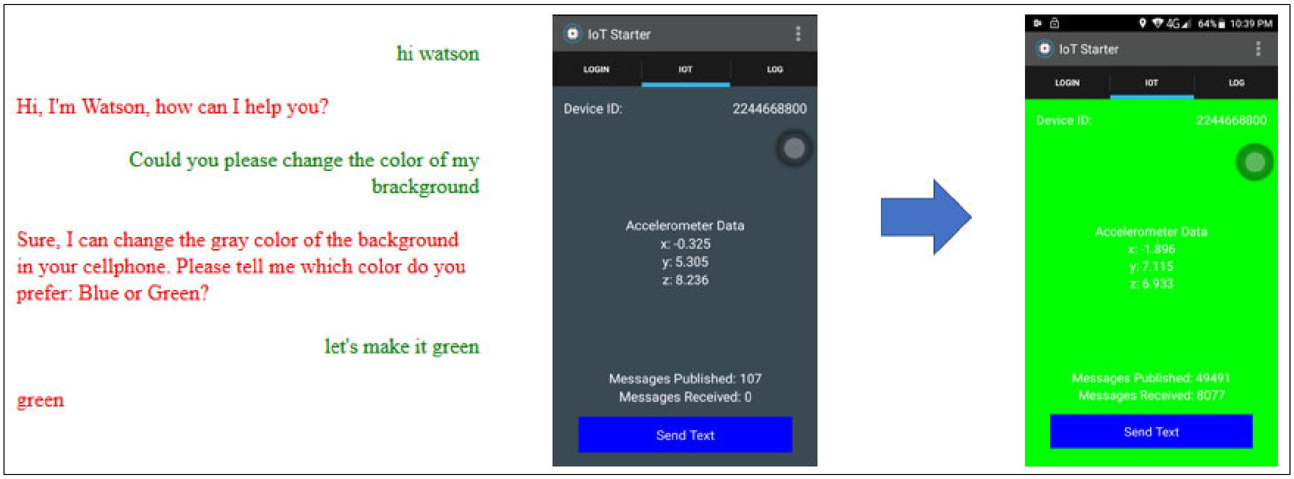
- You implement the use case in Chapter 4, "Help Desk Assistant chatbot" on page 109.
- You have an Android smartphone.

5.1.3 Expected results

In this chapter, the cognitive chatbot that you developed in Chapter 4, "Help Desk Assistant chatbot" on page 109 is enhanced to understand user's request to change the background color of a smart phone by sending commands to the device in response to the user's request.

The approach used in this simple example can be used to send other commands and send and receive information to and from IoT devices.

Figure 5-1 on page 141 shows the final chatbot application. It receives a request from the user to change the background color of the smart phone from gray to green. By integrating with the Watson Conversation service the chatbot is able to understand the user's request in natural language and respond in the user's language. By integrating with the Watson IoT Platform service the chatbot application sends commands to the smart phone to change the background color.



hi watson

Hi, I'm Watson, how can I help you?

Could you please change the color of my background

Sure, I can change the gray color of the background in your cellphone. Please tell me which color do you prefer: Blue or Green?

let's make it green

green

Device ID: 2244668800

Accelerometer Data
x: -0.325
y: 5.305
z: 8.236

Messages Published: 107
Messages Received: 0

Device ID: 2244668800

Accelerometer Data
x: -1.896
y: 7.115
z: 6.933

Messages Published: 49491
Messages Received: 8077

Figure 5-1 Using the chatbot to change the background color of a smart phone

5.2 Architecture

Figure 5-2 shows the components of the application and how the components interact with each other.

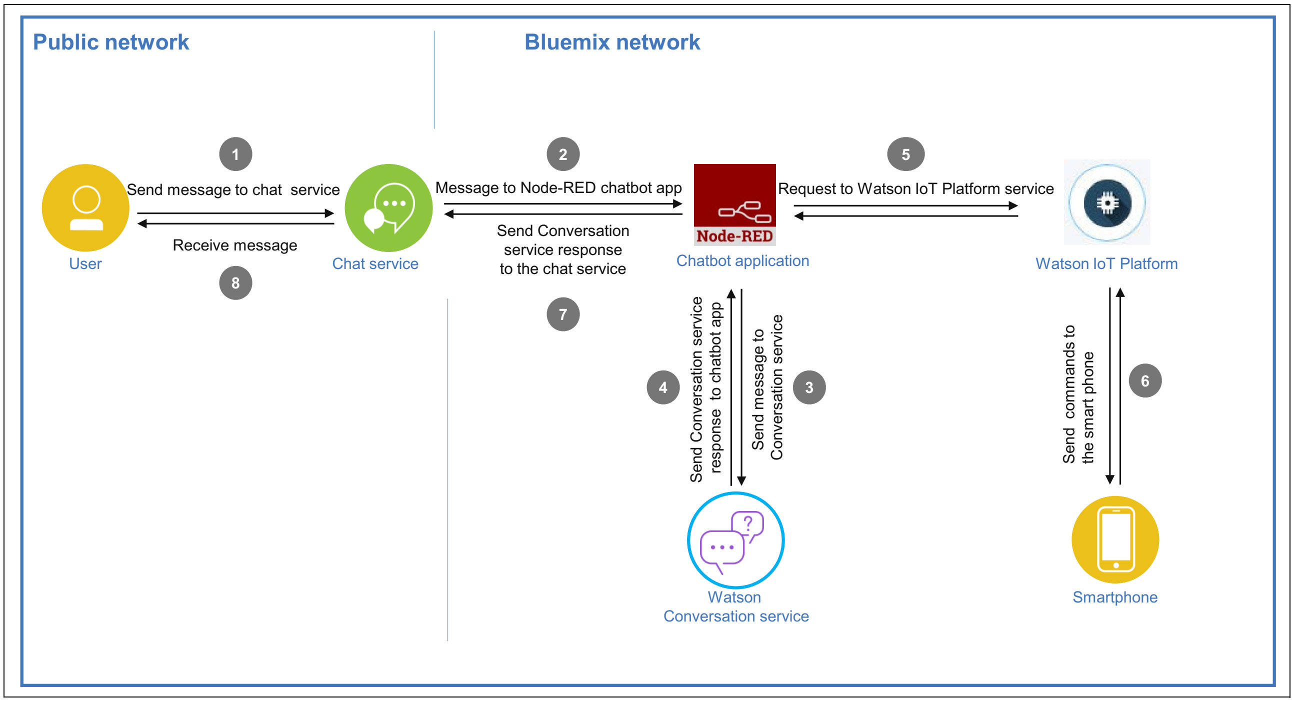

Public network
Bluemix network
1
Send message to chat service
Receive message
8
User
Chat service
2
Message to Node-RED chatbot app
Send Conversation service response to the chat service
7
Node-RED
Chatbot application
5
Request to Watson IoT Platform service
Watson IoT Platform
4
Send Conversation service response to chatbot app
Send message to Conversation service
3
Send commands to the smart phone
6
Watson Conversation service
Smartphone

Figure 5-2 Architecture

The numbers in the diagram represent the following steps:

1. The user sends a message to the chatbot through the chat service (Slack in this example).
2. The chat service checks whether the message is for the chatbot. If it is, the service sends the message to the chatbot application (Node-RED).

3. The application parses the message and sends the filtered message to the Watson Conversation service for processing.
4. The Watson Conversation service processes the message and provides a response.
5. The Node-RED application determines whether an action is required. If an action is required, the application sends a command to the Watson IoT platform to perform the requested action.
6. The Watson IoT service sends a request to the smartphone to perform the action requested.
7. The Node-RED application sends the response from the Conversation service to the chatbot service (Slack).
8. The chatbot service receives the message and displays the message to the user.

5.3 Step-by-step deployment of application

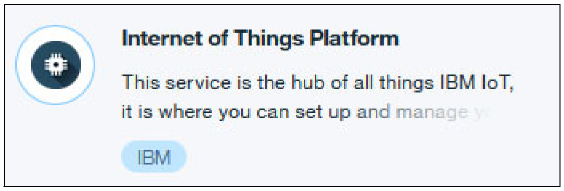
Implementing this use case involves the following steps:

1. Creating the Watson IoT Platform service.
2. Configuring the Android mobile device as an IoT device.
3. Modifying the Chatbot Conversation workspace.
4. Connecting the chatbot application to the IoT platform.
5. Testing the application.

5.3.1 Creating the Watson IoT Platform service

To create the Watson IoT Platform service instance, follow these steps:

1. Go to your Bluemix Dashboard and click **Create Service**.
2. Select the **Internet of Things Platform** service (Figure 5-3).

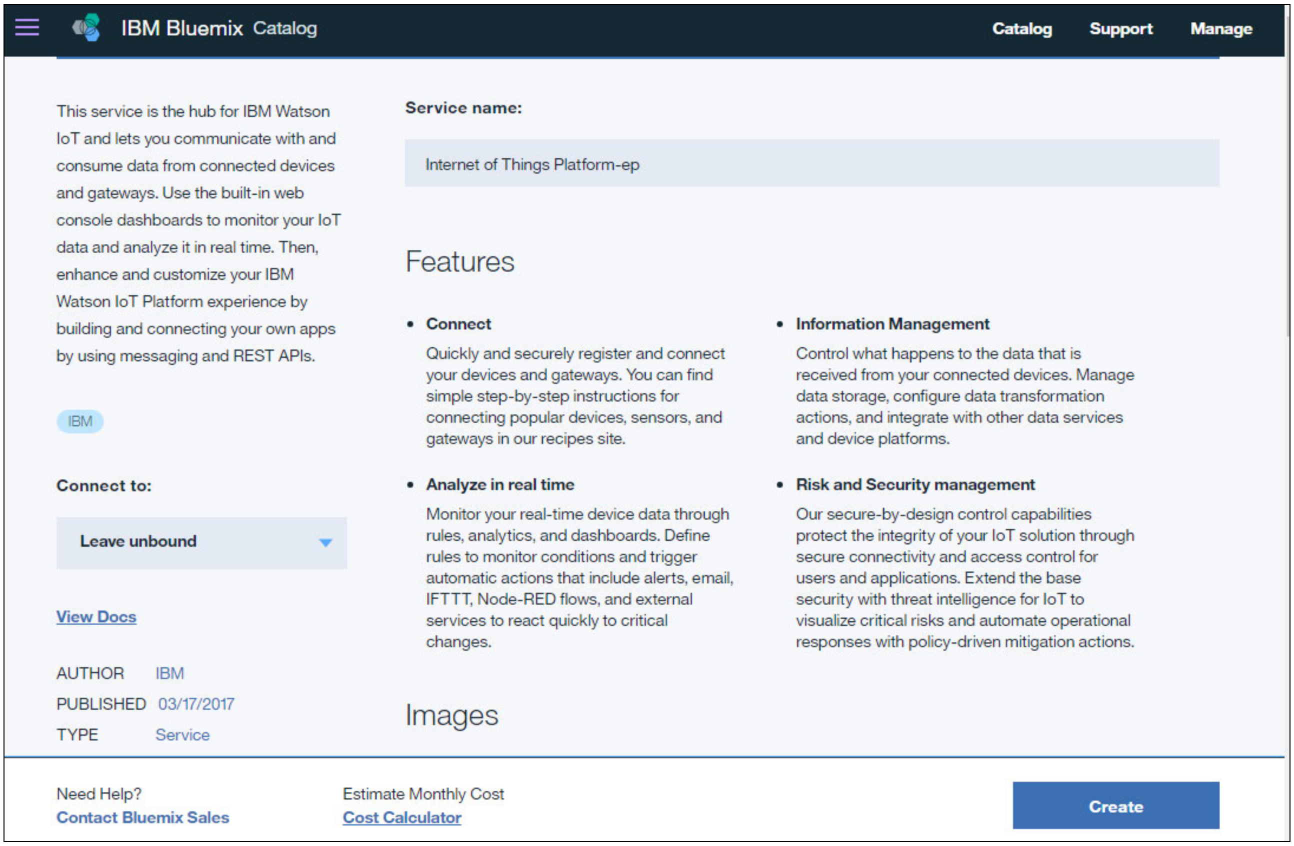


Internet of Things Platform

This service is the hub of all things IBM IoT, it is where you can set up and manage y

Figure 5-3 Internet of Things Platform service

3. Enter a unique name in the Service name field, and click **Create (**Figure 5-4).



This service is the hub for IBM Watson IoT and lets you communicate with and consume data from connected devices and gateways. Use the built-in web console dashboards to monitor your IoT data and analyze it in real time. Then, enhance and customize your IBM Watson IoT Platform experience by building and connecting your own apps by using messaging and REST APIs.

IBM

Connect to: Leave unbound

View Docs

AUTHOR IBM
PUBLISHED 03/17/2017
TYPE Service

Service name: Internet of Things Platform-ep

Features

- **Connect**
  Quickly and securely register and connect your devices and gateways. You can find simple step-by-step instructions for connecting popular devices, sensors, and gateways in our recipes site.
- **Analyze in real time**
  Monitor your real-time device data through rules, analytics, and dashboards. Define rules to monitor conditions and trigger automatic actions that include alerts, email, IFTTT, Node-RED flows, and external services to react quickly to critical changes.
- **Information Management**
  Control what happens to the data that is received from your connected devices. Manage data storage, configure data transformation actions, and integrate with other data services and device platforms.
- **Risk and Security management**
  Our secure-by-design control capabilities protect the integrity of your IoT solution through secure connectivity and access control for users and applications. Extend the base security with threat intelligence for IoT to visualize critical risks and automate operational responses with policy-driven mitigation actions.

Images

Need Help? Contact Bluemix Sales

Estimate Monthly Cost Cost Calculator

Figure 5-4 create IoT Platform service

4. On the Welcome page, click **Launch** to access the service dashboard.

   The IoT dashboard includes much useful information. For example, you can launch the Watson IoT Platform documentation and Quickstart from the dashboard (Figure 5-5).

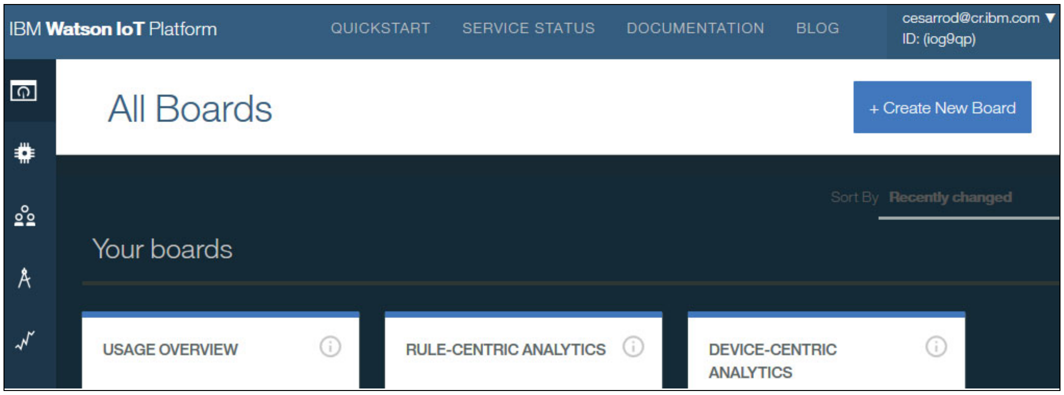
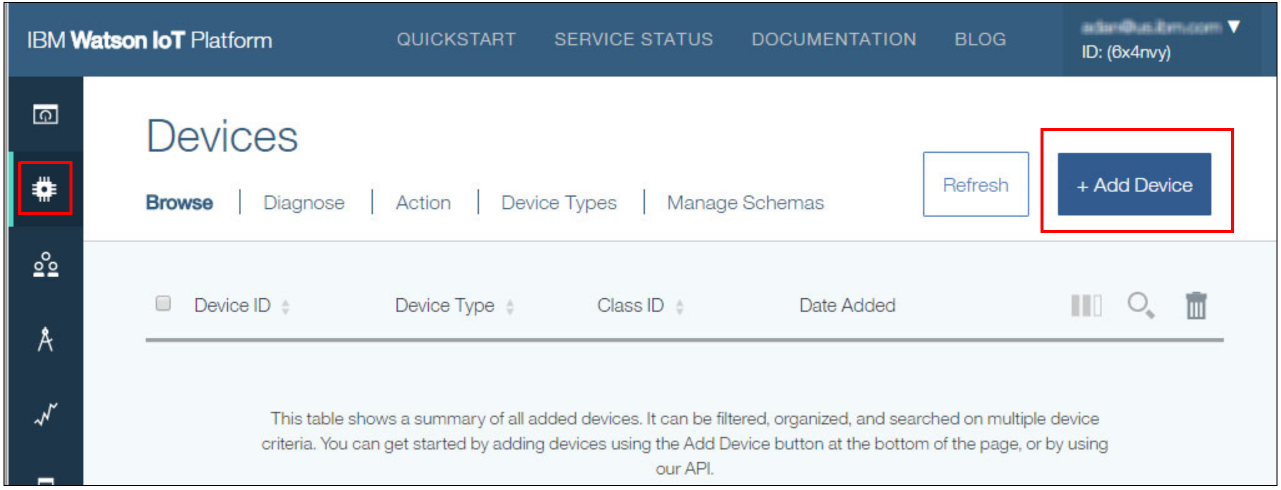


Figure 5-5 IoT dashboard

5. From the menu on the right, click the devices icon. Then, in the Devices window, click **Add Device** (Figure 5-6).

).



This table shows a summary of all added devices. It can be filtered, organized, and searched on multiple device criteria. You can get started by adding devices using the Add Device button at the bottom of the page, or by using our API.

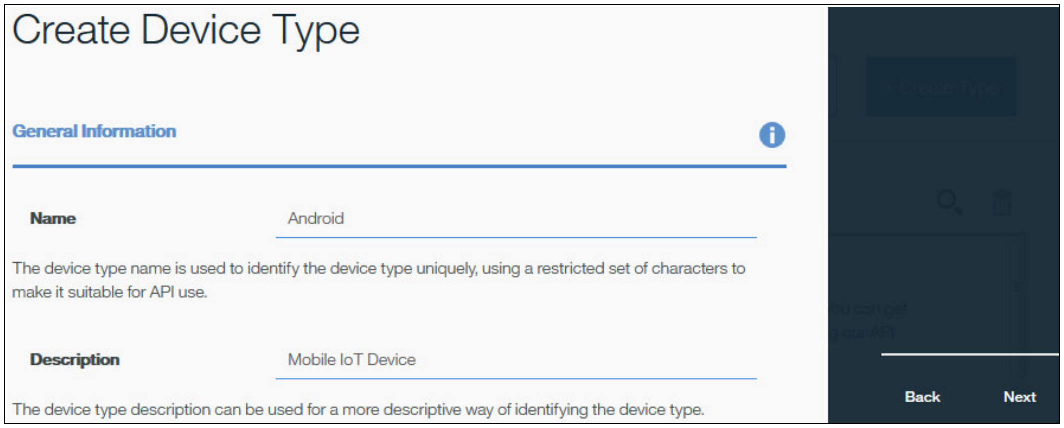
Figure 5-6   Watson IoT Platform dashboard: Add Device

6. Each device must have a device type associated, which is a way to categorize similar devices. So, before creating a device, you must create a *device type*.

   Click **Create device type** (twice), Enter `Android` as the device type name, add a description, and then click **Next**. If you want, you can use the same information as shown in Figure 5-7.

   The remainder of the information is optional, so you can click **Next** until you see the option to click **Create**.

> **Important:** The device type name must be *Android* because this is the value that is hardcoded in the mobile app example that is used in 5.3.2, “Configuring the Android mobile device as an IoT device” on page 147.



Name: Android

The device type name is used to identify the device type uniquely, using a restricted set of characters to make it suitable for API use.

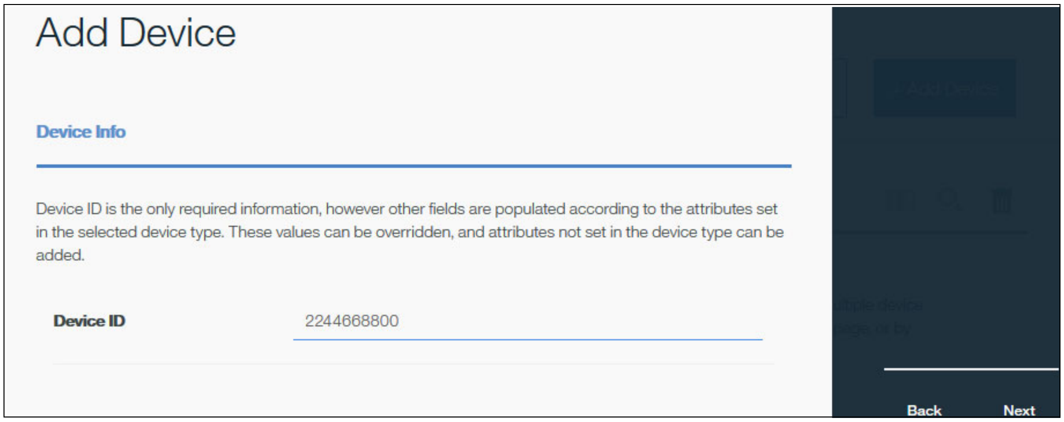
Description: Mobile IoT Device

The device type description can be used for a more descriptive way of identifying the device type.

Figure 5-7   Create Device Type

7. The Add Device window is displayed again, but this time a device type is available to choose (Android). Make sure the device type is selected, and then click **Next** at the lower right corner.

8. Add an ID for the device. The device ID should be unique within your organization. The suggestion is to use something that will identify the device (such as the MAC Address, a phone number, and so on). Enter a device ID value (Figure 5-8), and then click **Next**.



Add Device

Device Info

Device ID is the only required information, however other fields are populated according to the attributes set in the selected device type. These values can be overridden, and attributes not set in the device type can be added.

Device ID 2244668800

Figure 5-8 Adding the device ID

9. The metadata is optional; click **Next**.

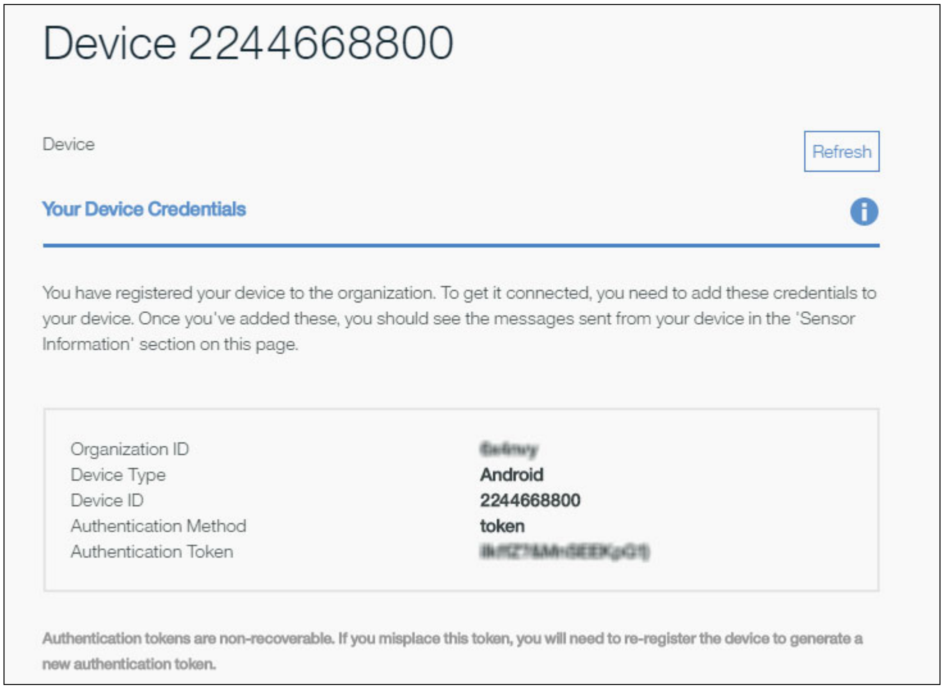
10. Next, you add security. You can generate your own token or allow the system to generate one for you. For this example, click **Next** so that the system automatically generates the token.

11. A summary of all submitted information is displayed. Click **Add** to complete the process.

12. Note all the information on the page (Figure 5-9 on page 146), including the following items, because you will use this information later:

    – Organization ID
    – Device type
    – Device ID
    – Authentication method
    – Authentication token

> **Remember:** The authentication token is non-recoverable; therefore, if you miss it, you must register the device again.



Device 2244668800

Device

Your Device Credentials

You have registered your device to the organization. To get it connected, you need to add these credentials to your device. Once you've added these, you should see the messages sent from your device in the 'Sensor Information' section on this page.

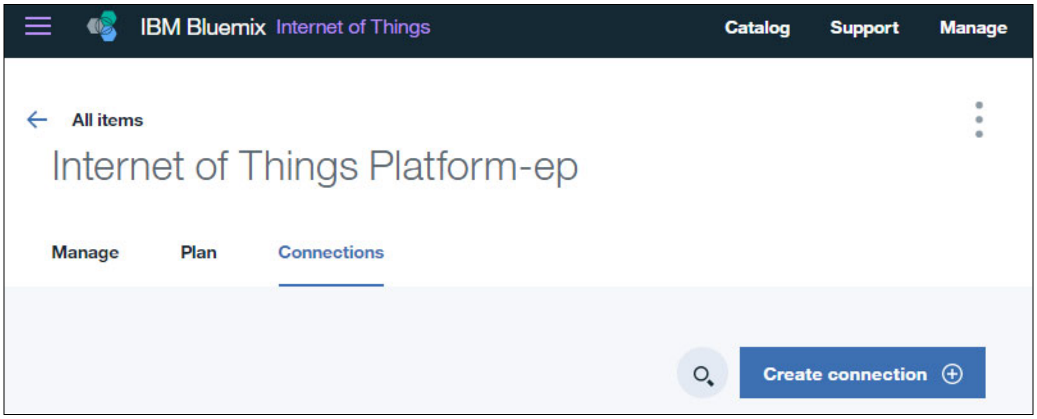
Organization ID	[redacted]
Device Type	Android
Device ID	2244668800
Authentication Method	token
Authentication Token	[redacted]

Authentication tokens are non-recoverable. If you misplace this token, you will need to re-register the device to generate a new authentication token.

Figure 5-9 IoT device credentials

The device is now added to the Watson IoT Platform service instance.

13. Go to the Bluemix Dashboard and find the IoT Platform service instance that you just created. Select it by clicking it. This action opens the Bluemix IoT Service landing window. In this window, go to the Connections tab and click **Create Connection (**Figure 5-10).



Internet of Things Platform-ep

Figure 5-10 Create connection

14. Find the Node-RED application (conv-201-xxx-nodered) that you created in 4.4.6, “Creating the Help Desk Assistant chatbot application in Node-RED” on page 122. Click **Connect**.

15. To apply the changes, the application must be restaged. So, click **Restage**. Keep in mind that if you make any mistake while staging, you can stop the application and restage.

16. After these steps are complete, you will be able to see the Node-RED application under the Connections tab of the IoT Platform service instance, which means that both are successfully connected.

5.3.2 Configuring the Android mobile device as an IoT device

To establish the communication between the Watson IoT platform and a smartphone, you need to install an application.

If you are an Android developer, the code is on the IoT starter for Android page in GitHub:

`https://github.com/ibm-watson-iot/iot-starter-for-android`

This section describes the options to configure and install the application for an Android device. However, if you want to run the application on iOS, see the IoT starter application for IBM Watson IoT on iOS in GitHub:

`https://github.com/ibm-watson-iot/iot-starter-for-ios`

Use these steps to complete the installation:

1. Set up the phone to enable the installation of applications (`.apk`) outside of the Google Play Store. Go to **Settings/Security**, and under **Device Administration**, enable **Unknown Sources**.

   Important: Remember to revert this setting after you install the application.

   The instructions to enable this setting vary in different Android versions. Refer to your device documentation as needed.

2. On your phone, open a browser and go to the following address:

   `http://ibm.biz/mobile-app`

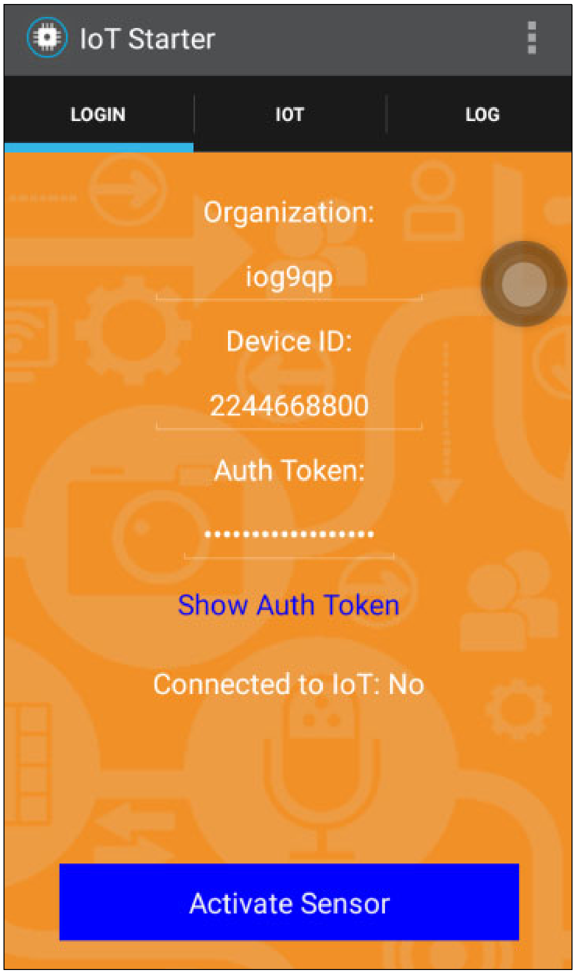
   Case-sensitive: This URL is case-sensitive.

3. Click **Open binary file**. Accept any warning notifications and click **Download**.

   Note: Depending on your Android phone model and operating system level, warning messages can differ.

   After the download is complete, click over the file to install it. If you missed this option, find the downloads folder using any file manager for Android, and then click to install it.

4. After the app is installed, open it. Add the values from step 12 on page 145 (Organization ID, device ID, and authentication token) and click **Activate Sensor** at the bottom of the screen (Figure 5-11).



Organization:

iog9qp

Device ID:

2244668800

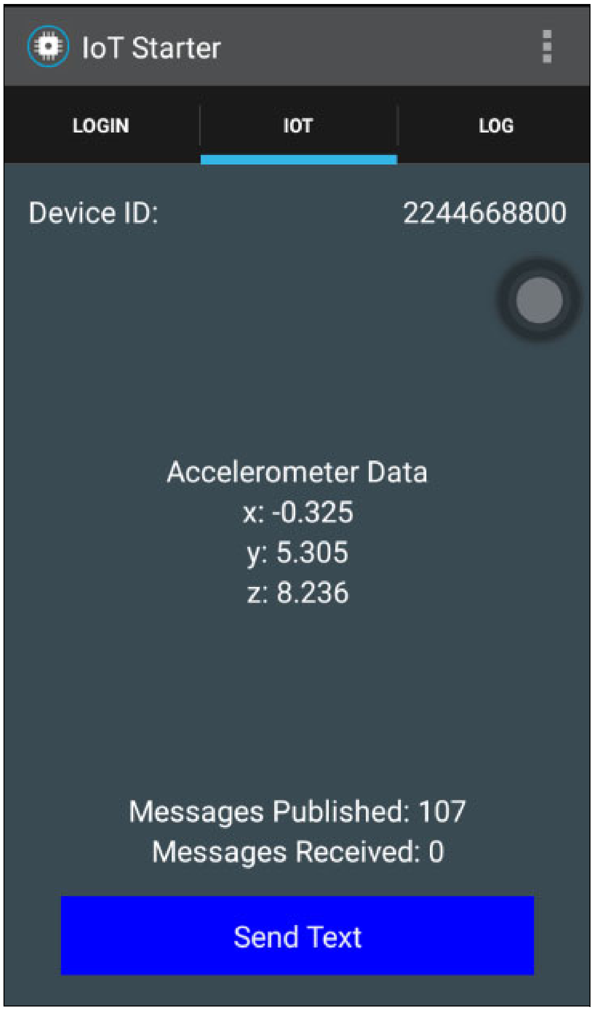
Auth Token:

••••••••••••••••••

Connected to IoT: No

Figure 5-11 Adding the Watson IoT Platform service values to the smartphone app

5. If nothing happens, you probably miss-typed a value. Otherwise, it displays the accelerometer data that is being read by the device (Figure 5-12).



Device ID: 2244668800

Accelerometer Data
x: -0.325
y: 5.305
z: 8.236

Messages Published: 107
Messages Received: 0

Figure 5-12 Reading from the accelerometer sensor

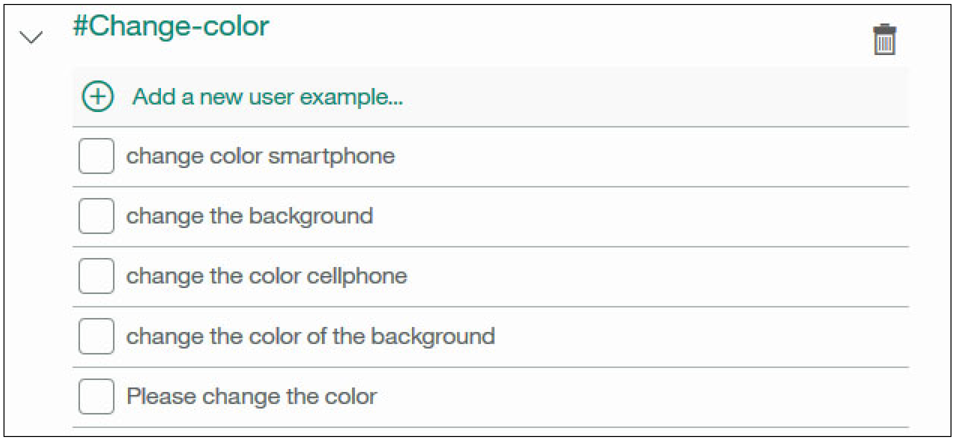
The smartphone is now connected to the Watson IoT Platform service instance that you created in 5.3.1, “Creating the Watson IoT Platform service” on page 142.

5.3.3 Modifying the Chatbot Conversation workspace

In this section, you modify the `Chatbot` Conversation workspace created in 4.4.1, “Creating a new Conversation workspace” on page 113. You will add the intents, entities, and dialog to handle a chat with a user submitting requests through the chatbot to change the color of the phone background. The steps in this section assume you start with the previously created `Chatbot` workspace. Alternatively, you can create a new Conversation workspace for this use case. For information about creating Conversation workspaces, see Chapter 2, “Conversation service workspace” on page 13.

Complete the following steps:

1. Find the Conversation service instance created in 2.1.1, “Creating a Watson Conversation service instance” on page 14 and click to open it.
2. Click **Launch tool** to open Conversation tooling. Previously created workspaces are listed.
3. Find the Chatbot workspace created in 4.4.1, “Creating a new Conversation workspace” on page 113.
4. Add the intent `#Change-color` and the examples shown in Figure 5-13.



#Change-color

⊕ Add a new user example...

□ change color smartphone

□ change the background

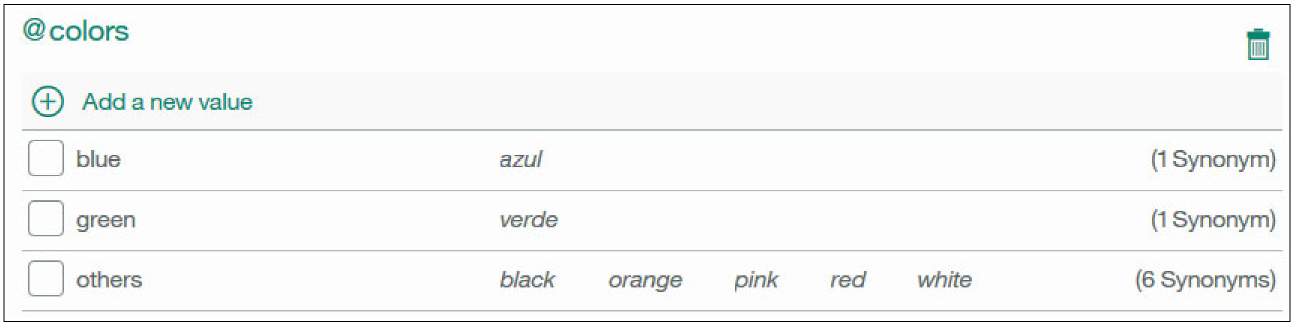
□ change the color cellphone

□ change the color of the background

□ Please change the color

Figure 5-13 Adding the new intent

5. Add the `@colors` entity shown in Figure 5-14. Notice that you can add synonyms to describe colors that are not available.



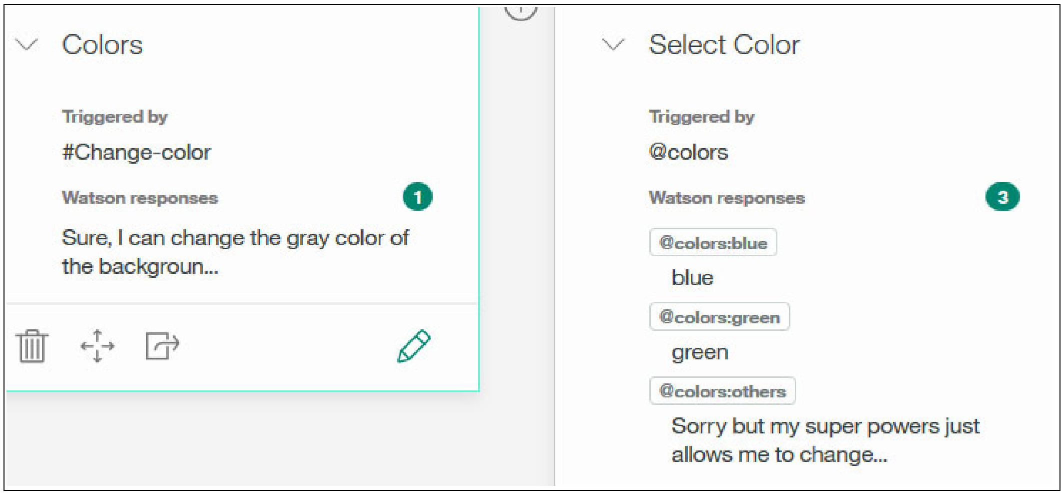
@colors

⊕ Add a new value

□ blue	*azul*					(1 Synonym)
□ green	*verde*					(1 Synonym)
□ others	*black*	*orange*	*pink*	*red*	*white*	(6 Synonyms)

Figure 5-14 Adding the new entity

6. Add the dialog as shown in Figure 5-15.



Colors

Triggered by

#Change-color

Watson responses 1

Sure, I can change the gray color of the backgroun...

Select Color

Triggered by

@colors

Watson responses 3

@colors:blue

blue

@colors:green

green

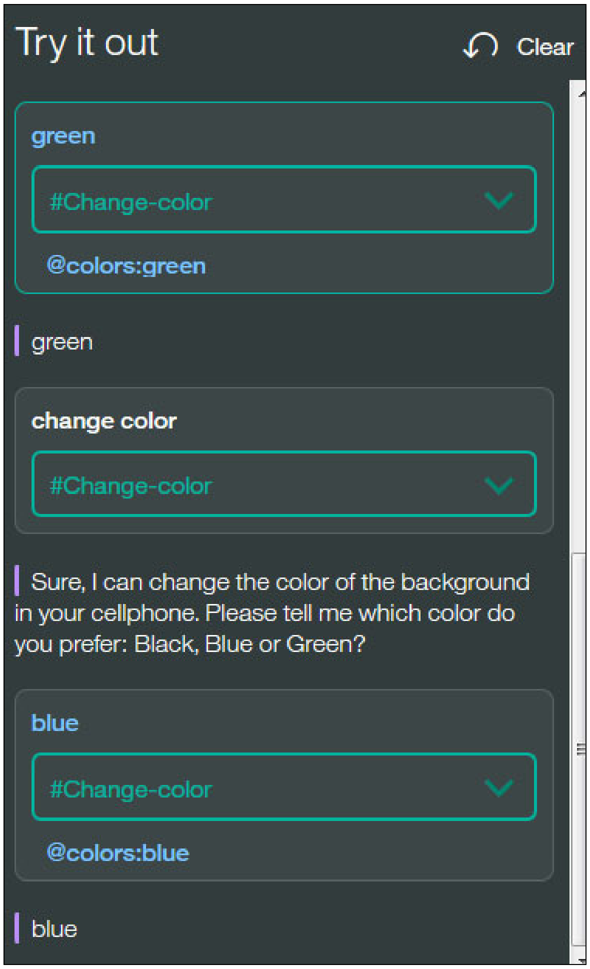
@colors:others

Sorry but my super powers just allows me to change...

Figure 5-15 Adding the new dialog

7. Test the dialog. Click the Ask Watson icon (green bubble on the upper right corner) to test the dialog. Type `change color` and then `green` to get the results shown in Figure 5-16 on page 152. If you have different results, make the corrections by selecting the correct intent or entity.

   Notice that in this step you are only testing the conversation with the user, *not* sending commands to the device to change the color of the background of the cellphone.


Try it out

Clear

green

#Change-color

@colors:green

green

change color

#Change-color

Sure, I can change the color of the background in your cellphone. Please tell me which color do you prefer: Black, Blue or Green?

blue

#Change-color

@colors:blue

blue

Figure 5-16 Testing the dialog

5.3.4 Connecting the chatbot application to the IoT platform

Next, open the Node-RED application created in 4.4.6, “Creating the Help Desk Assistant chatbot application in Node-RED” on page 122. Modify the application by making the following changes:

1. Add a `function` node (named Color change) and one `IBM IoT` output node and then connect them to the conversation node (Figure 5-17).

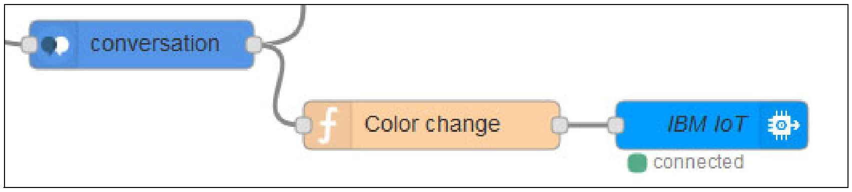
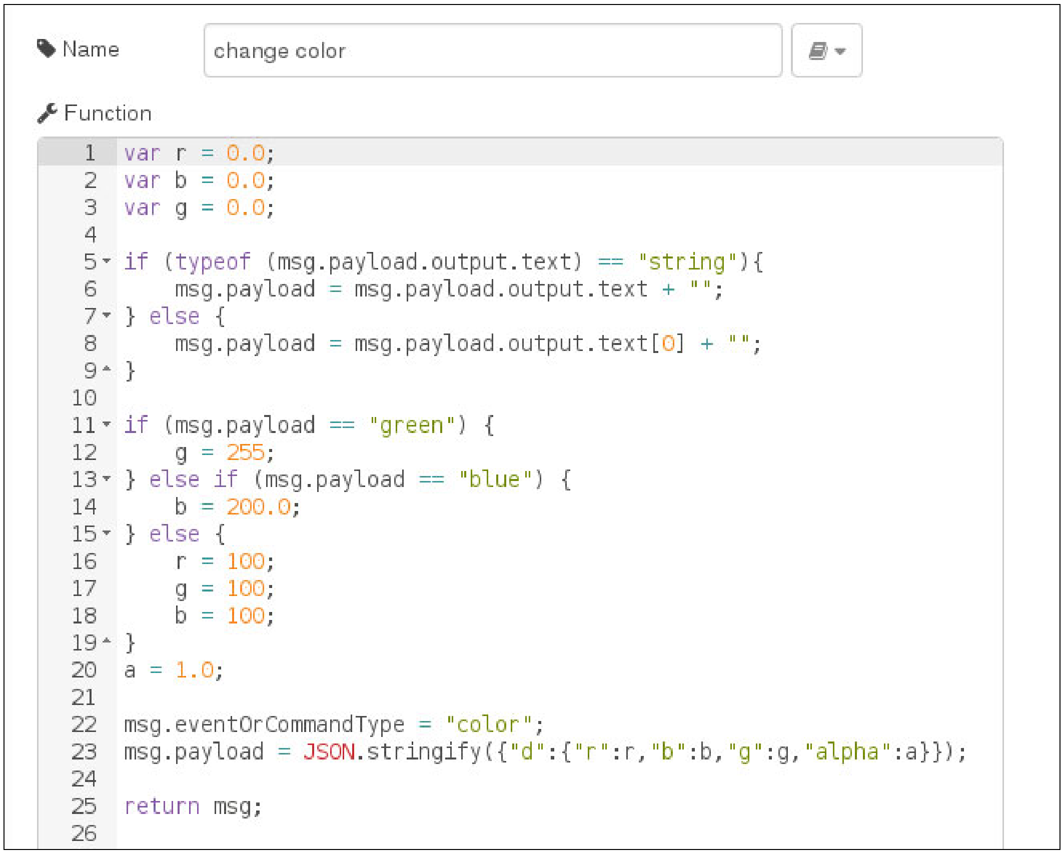


Figure 5-17 New nodes

2. Edit the `Color change` function node. Add lines of code to specify the codes of the colors to use. To do that, add the lines of code shown in Example 5-1 to the function node (Figure 5-18 on page 154). The code in the example creates three variables (one for each color), and then depending on the message received by the chatbot, it will pass the color data to the IBM IoT node to send it to the smartphone. You can change the code of the colors (Example 5-1) to display different backgrounds on your smartphone.

Example 5-1 Code for the function node

```
var r = 0.0;
var b = 0.0;
var g = 0.0;

if (typeof (msg.payload.output.text) == "string"){
   msg.payload = msg.payload.output.text + "";
} else {
   msg.payload = msg.payload.output.text[0] + "";
}

if (msg.payload == "green") {
   g = 255;
} else if (msg.payload == "blue") {
   b = 200.0;
} else {
   r = 100;
   g = 100;
   b = 100;
}
a = 1.0;

msg.eventOrCommandType = "color";
msg.payload = JSON.stringify({"d":{"r":r,"b":b,"g":g,"alpha":a}});

return msg;
```



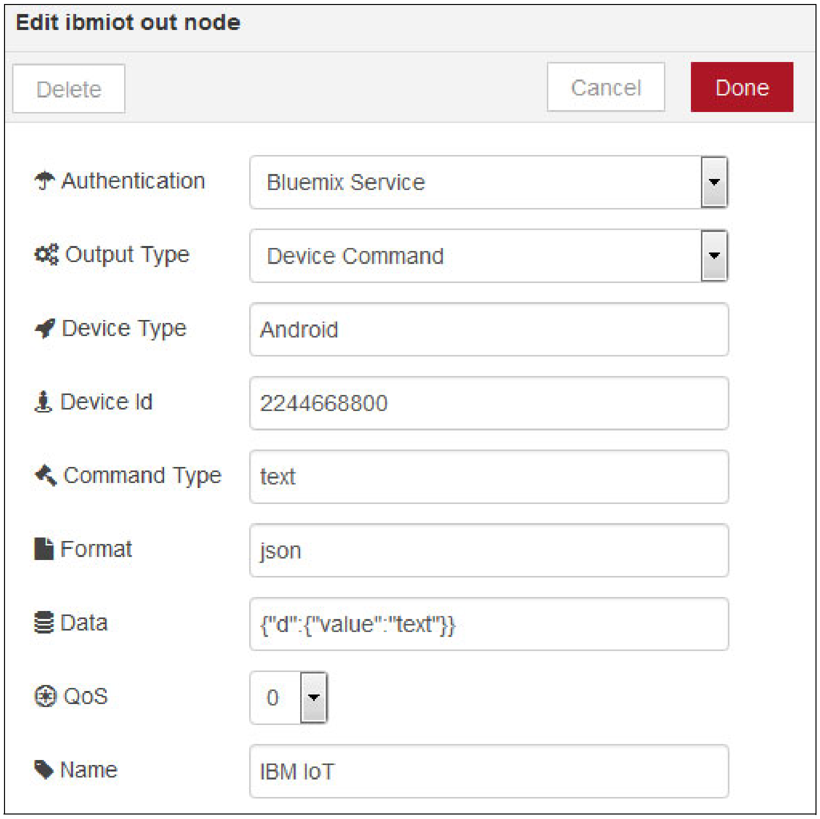
Name change color

Function

```
var r = 0.0;
var b = 0.0;
var g = 0.0;

if (typeof (msg.payload.output.text) == "string"){
    msg.payload = msg.payload.output.text + "";
} else {
    msg.payload = msg.payload.output.text[0] + "";
}

if (msg.payload == "green") {
    g = 255;
} else if (msg.payload == "blue") {
    b = 200.0;
} else {
    r = 100;
    g = 100;
    b = 100;
}
a = 1.0;

msg.eventOrCommandType = "color";
msg.payload = JSON.stringify({"d":{"r":r,"b":b,"g":g,"alpha":a}});

return msg;

```

Figure 5-18 Testing the responses; lines of code added

3. Edit the IBM IoT out node: enter the values of your IBM IoT Platform service to finish the setup (Figure 5-19).



Edit ibmiot out node

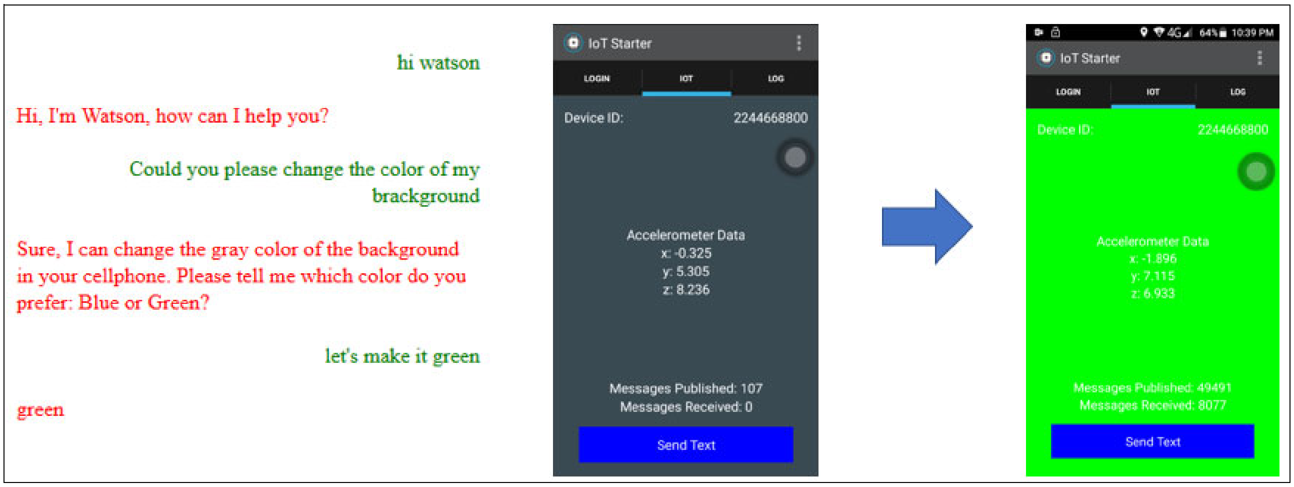
Authentication	Bluemix Service
Output Type	Device Command
Device Type	Android
Device Id	2244668800
Command Type	text
Format	json
Data	{"d":{"value":"text"}}
QoS	0
Name	IBM IoT

Figure 5-19 Configuring the IBM IoT node

4. Click **DEPLOY** (upper right corner) and then close the Node-RED workspace.

5.3.5 Testing the application

Return to the chat service (Slack) that you set up in 4.4.7, “Setting up the chat service (Slack)” on page 131. Enter a request for the chatbot to change the background color of the smartphone. Remember that for this example to work, the application that you installed in 5.3.2, “Configuring the Android mobile device as an IoT device” on page 147 must be open in the smartphone. Figure 5-20 shows the result.



hi watson

Hi, I'm Watson, how can I help you?

Could you please change the color of my brackground

Sure, I can change the gray color of the background in your cellphone. Please tell me which color do you prefer: Blue or Green?

let's make it green

green

Device ID: 2244668800

Accelerometer Data
x: -0.325
y: 5.305
z: 8.236

Messages Published: 107
Messages Received: 0

Device ID: 2244668800

Accelerometer Data
x: -1.896
y: 7.115
z: 6.933

Messages Published: 49491
Messages Received: 8077

Figure 5-20 Testing the application

5.4 References

For more information, see the following resources:

- Watson IoT Platform documentation:

  `https://console.ng.bluemix.net/docs/services/IoT/index.html`

- Watson IoT Platform Quickstart:

  `https://quickstart.internetofthings.ibmcloud.com`

6

Chatting about the weather: Integrating Weather Company Data with the Conversation service

The Weather Company® Data for Bluemix service lets you integrate weather data from The Weather Company into your IBM Bluemix application. You can retrieve weather data for an area specified by a geolocation.

This chapter guides you through the creation of a sample chatbot application, the Cognitive Weather Forecast chatbot, that integrates with the Watson Conversation and Weather Company Data services. The application demonstrates the use of both services to get the forecasted weather for a city through chatting with the user.

The following topics are covered in this chapter:

- Getting started
- Architecture
- Two ways to deploy the application: Step-by-step and quick deploy
- Step-by-step implementation
- Quick deployment of application
- References

6.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

6.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Integrate the Watson Conversation service and Weather Company Data service with your application.
- Develop a cognitive conversation application to retrieve the weather forecast for a specific city.

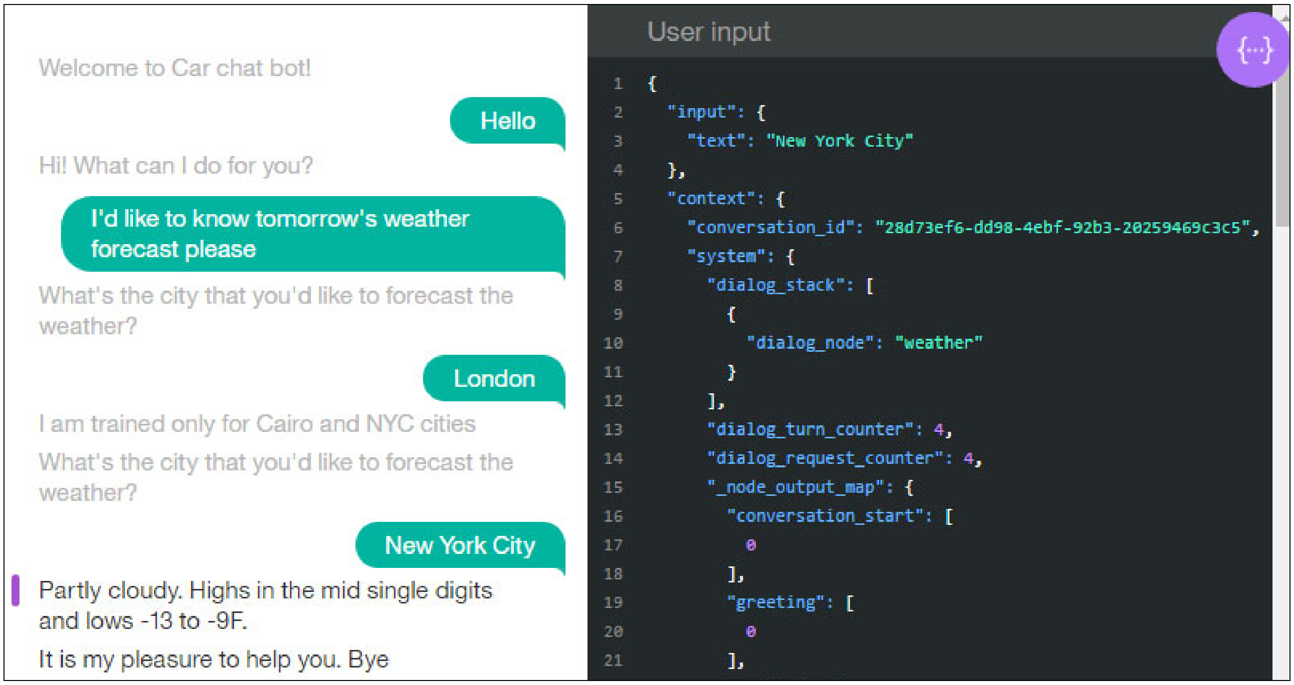
6.1.2 Prerequisites

To complete the steps in this chapter, be sure these prerequisites are met:

- Review Chapter 1, “Basics of Conversation service” on page 1.
- Review Chapter 2, “Conversation service workspace” on page 13 and create a conversation service instance and a Conversation workspace as described in this chapter.
- Use any web browser (Chrome, Firefox, or Internet Explorer).
- Have basic JavaScript skills.
- Have basic knowledge of Git.
- Install Cloud Foundry tool on your workstation.
- Install Git tool on your workstation.

6.1.3 Expected results

Figure 6-1 on page 159 shows the application. The user requests tomorrow’s weather forecast, and the application asks for the name of a city. The user responds with a name, in this case London, and the application responds that only Cairo and NYC are supported. The user chooses a supported city and the application responds with the weather forecast that it receives from Weather Company Data.



Welcome to Car chat bot!

Hello

Hi! What can I do for you?

I'd like to know tomorrow's weather forecast please

What's the city that you'd like to forecast the weather?

London

I am trained only for Cairo and NYC cities

What's the city that you'd like to forecast the weather?

New York City

Partly cloudy. Highs in the mid single digits and lows -13 to -9F.

It is my pleasure to help you. Bye

User input

```
{
  "input": {
    "text": "New York City"
  },
  "context": {
    "conversation_id": "28d73ef6-dd98-4ebf-92b3-20259469c3c5",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "weather"
        }
      ],
      "dialog_turn_counter": 4,
      "dialog_request_counter": 4,
      "_node_output_map": {
        "conversation_start": [
          0
        ],
        "greeting": [
          0
        ],
```

Figure 6-1 Cognitive Weather Forecast chatbot

6.2 Architecture

Figure 6-2 shows the components involved in this use case and the runtime flow.

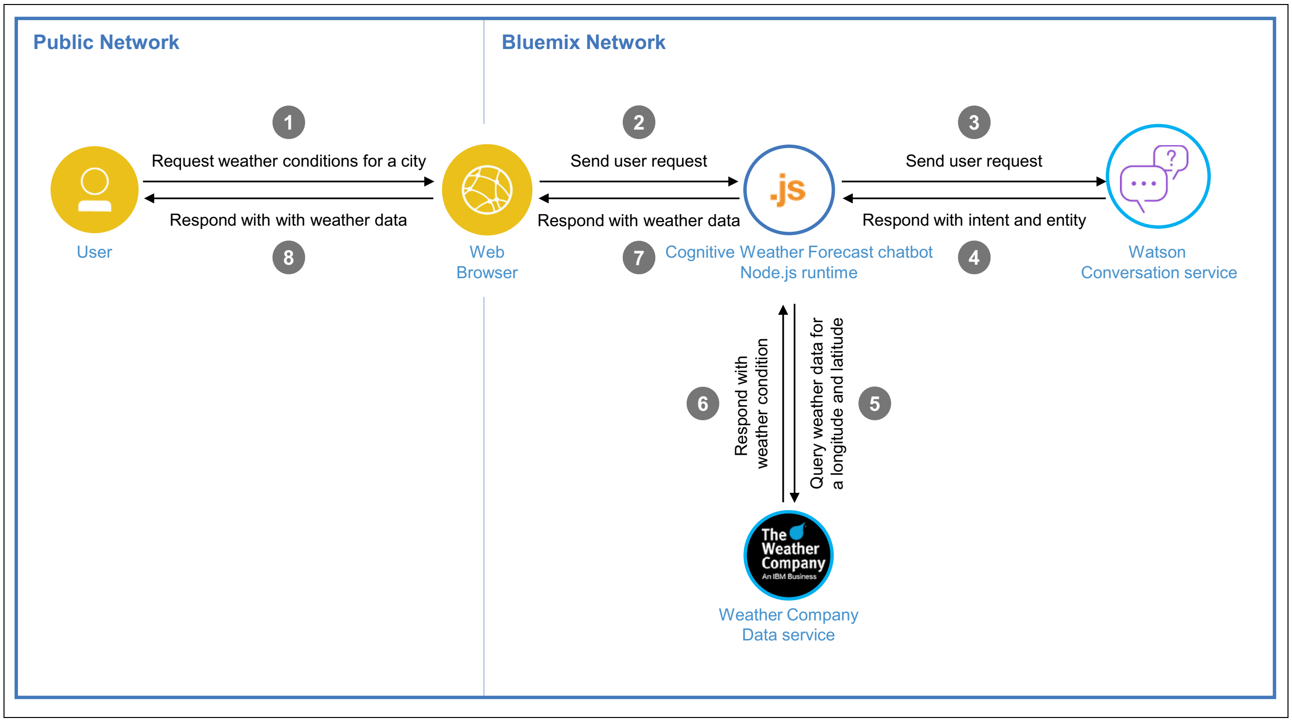


Figure 6-2 Architecture

The flow for this use case is as follows:

1. The user engages in a conversation with the application, requesting the weather forecast for a city; for example, Cairo.
2. The request is passed from the web browser to the Cognitive Weather Forecast application that runs on Node.js.
3. The application passes the user's request in natural language to the Conversation service.
4. The Conversation service understands the intent and entities in the user's message passed by the application. Then it returns a response to the application based on the dialog configured in the Conversation workspace. It returns `'[REPLACE WITH WEATHER DATA]'` and the entities to the calling application (for example: Cairo).
5. The Node.js application queries the Weather Company Data service for the weather forecast for the requested city, passing to it the latitude and longitude of the entity.
6. The Weather Company Data service responds with the weather forecast.
7. The Node.js application replaces `'[REPLACE WITH WEATHER DATA]'` with the result received from the Weather Company Data service and sends it to the web browser.
8. The user sees the response on the web browser. For example `Sunny. Highs in the low 70s and lows in the low 50s`.

6.3 Two ways to deploy the application: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- Step-by-step deployment (incomplete) version of the application

  This repository contains an incomplete version of the application and is used in all sections of 6.4, "Step-by-step implementation" on page 160. This version takes you through the key steps to integrate the IBM Watson service with the application logic.

- Quick deployment (complete) version of the application

  This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 6.5, "Quick deployment of application" on page 182.

6.4 Step-by-step implementation

Implementing this use case involves the following steps:

1. Configuring Conversation workspace for Cognitive Weather Forecast chatbot.
2. Creating the Weather Company Data service instance.
3. Developing the Cognitive Weather Forecast chatbot application.
4. Testing the application.

6.4.1 Configuring Conversation workspace for Cognitive Weather Forecast chatbot

> **Note:** If you created a Conversation workspace by following the instructions in Chapter 2, “Conversation service workspace” on page 13, skip to “Get the Workspace ID” on page 163.

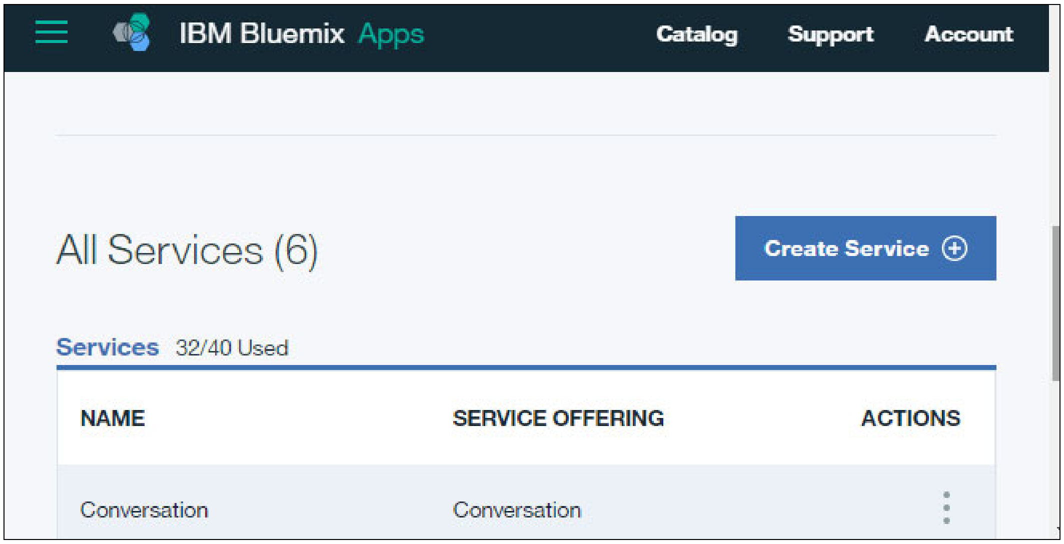
In this section, you create the Conversation workspace that will be used by the Cognitive Weather Forecast chatbot to understand the user’s request regarding to weather conditions in a city. This workspace includes entities, intents, and dialog specific to the application.

To simplify the creation of the Conversation workspace for this use case, import the workspace from the GitHub location:

`https://github.com/snippet-java/redbooks-conv-201-weather-nodejs/blob/master/training/1.4-conv-101-createservice.json`

To import the workspace, follow these steps:

1. Log in to Bluemix.
2. In the Services section of the dashboard, click **Conversation** which is the Conversation service instance that you created in Chapter 2, “Conversation service workspace” on page 13 (Figure 6-3).



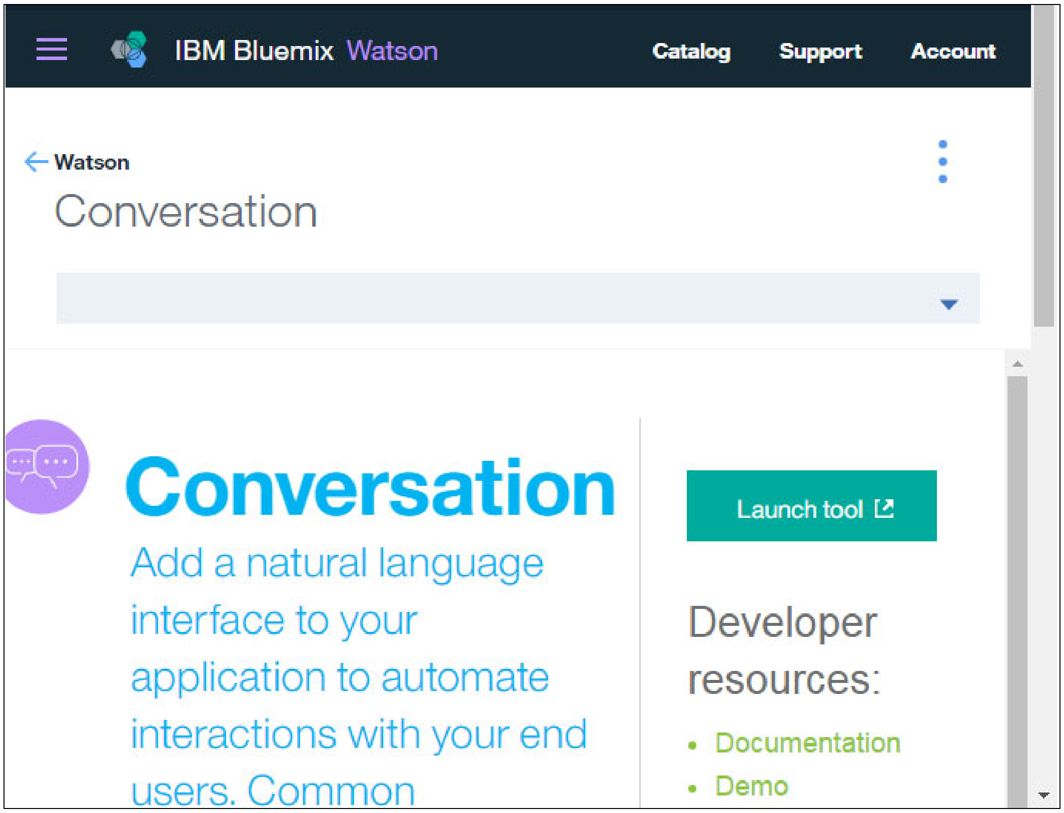
All Services (6)

Services 32/40 Used

NAME	SERVICE OFFERING	ACTIONS
Conversation	Conversation	⋮

Figure 6-3 Conversation service instance for this use case in the Bluemix dashboard

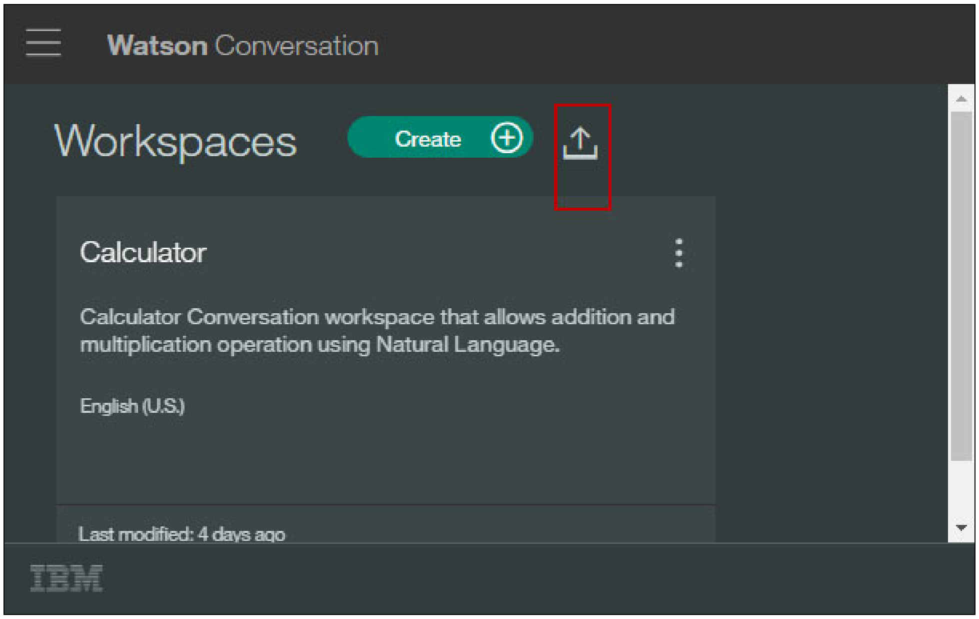
3. Click **Launch tool** to open the Conversation tool (Figure 6-4).



Add a natural language interface to your application to automate interactions with your end users. Common

Figure 6-4 Launching the Conversation tool

4. Click the Import workspace icon to import the workspace (Figure 6-5).



Calculator Conversation workspace that allows addition and multiplication operation using Natural Language.

Figure 6-5 Importing a workspace

5. Click **Choose a file** and select the `1.4-conv-101-createservice.json` file that you downloaded at the start of this section. You should choose to import everything (intents, entities and dialog).
6. Click **Import**.

The Car Chat-bot workspace is imported. It will be used for this use case.

Get the Workspace ID

Get the Workspace ID that you will need in order to configure the application to point to the workspace:

1. Click the Menu icon, which is the three horizontal bars at the upper left corner (Figure 6-6).

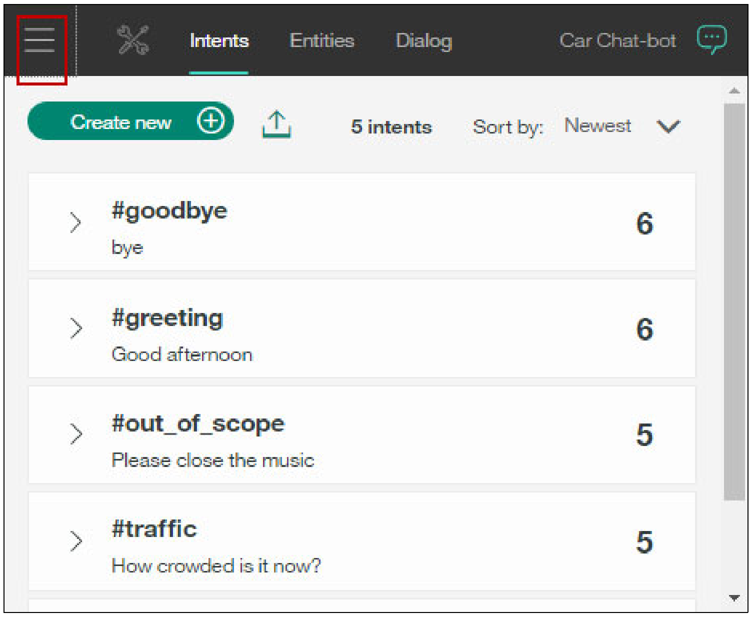


Figure 6-6 Car Chat-bot workspace: Menu

2. Click **Back to workspaces** (Figure 6-7).

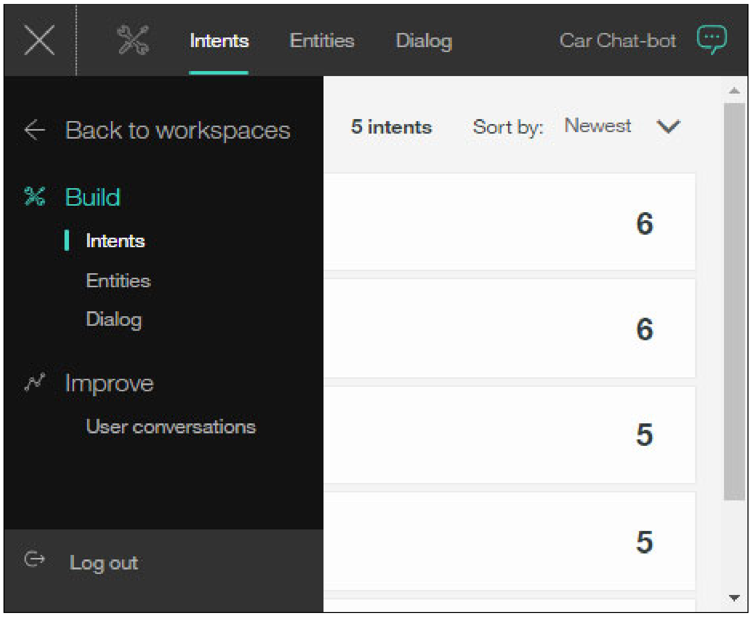
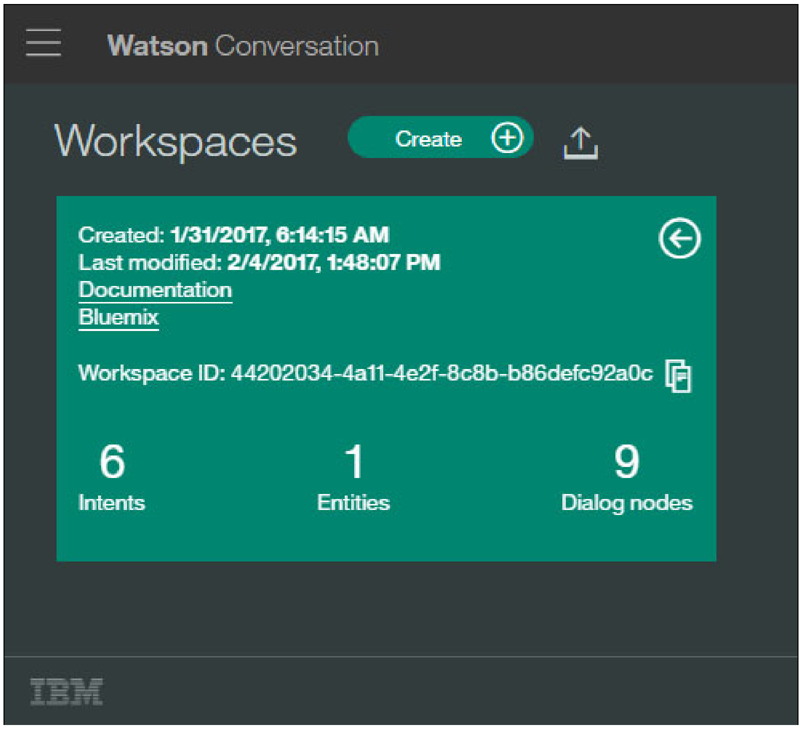


Figure 6-7 Car Chat-bot Conversation workspace: Back to workspaces

3. Click the **Actions** icon (three vertical dots on the top-right corner of the Car Chat-bot workspace) then choose **View Details**.
4. Copy the Workspace ID value and save it in any local text file (Figure 6-8). You will need this value in step 5 on page 179.



Created: **1/31/2017, 6:14:15 AM**
Last modified: **2/4/2017, 1:48:07 PM**
Documentation
Bluemix

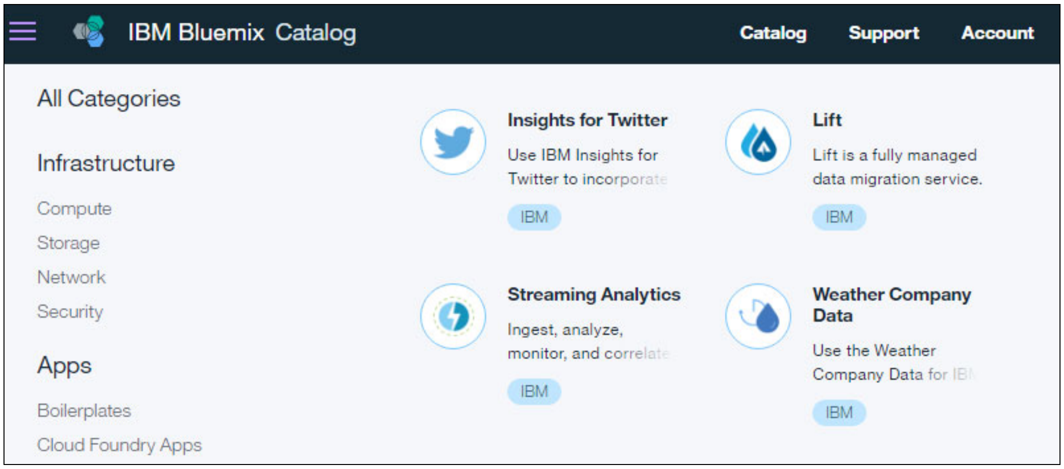
Workspace ID: 44202034-4a11-4e2f-8c8b-b86defc92a0c

Figure 6-8 Workspace ID

6.4.2 Creating the Weather Company Data service instance

To create a Weather Company Data service instance, follow these steps:

1. Open the Bluemix Catalog by clicking **Catalog** at the top bar.
2. Scroll to **Services** and select **Data & Analytics** → **Weather Company Data** (Figure 6-9).



IBM Bluemix Catalog

Catalog Support Account

All Categories

Infrastructure

Compute

Storage

Network

Security

Apps

Boilerplates

Cloud Foundry Apps

Insights for Twitter
Use IBM Insights for Twitter to incorporate
IBM

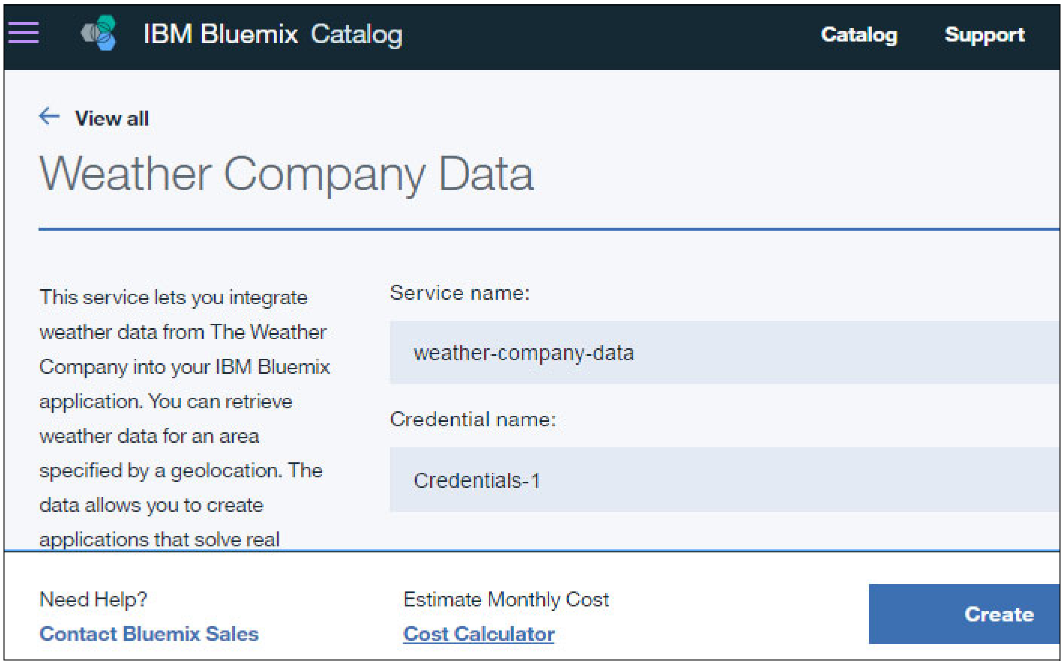
Lift
Lift is a fully managed data migration service.
IBM

Streaming Analytics
Ingest, analyze, monitor, and correlate
IBM

Weather Company Data
Use the Weather Company Data for IBM
IBM

Figure 6-9 Bluemix Catalog: Weather Company Data

3. For the Service name, use `weather-company-data`, and then click **Create** (Figure 6-10).



IBM Bluemix Catalog

Catalog Support

← View all

Weather Company Data

This service lets you integrate weather data from The Weather Company into your IBM Bluemix application. You can retrieve weather data for an area specified by a geolocation. The data allows you to create applications that solve real

Service name:

weather-company-data

Credential name:

Credentials-1

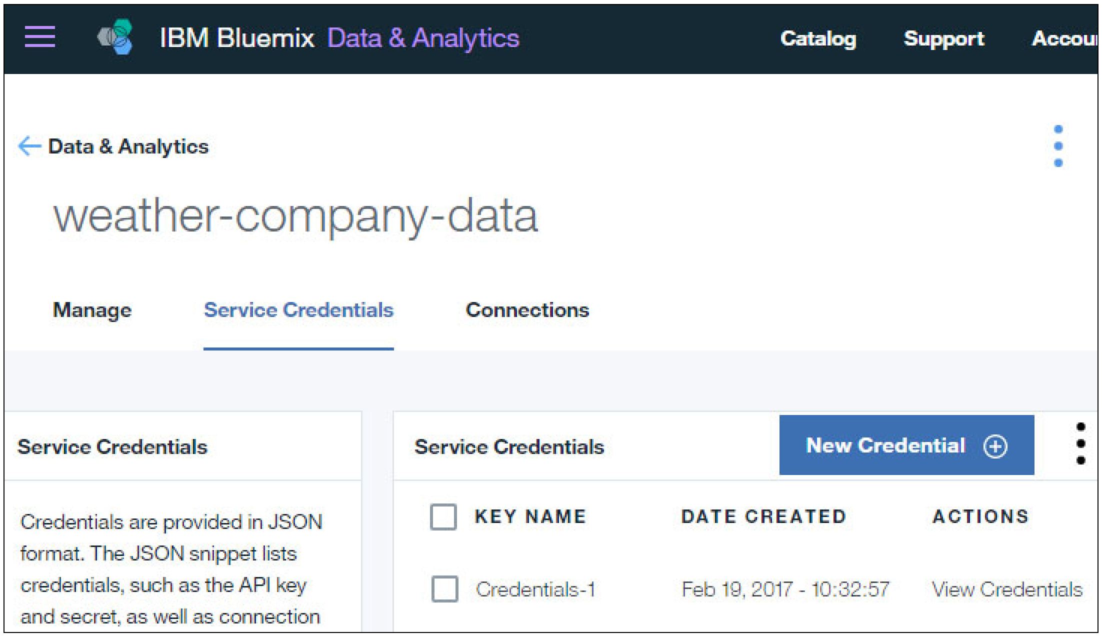
Need Help?
Contact Bluemix Sales

Estimate Monthly Cost
Cost Calculator

Create

Figure 6-10 Create Weather Company Data service instance

4. Click the **Service Credentials** tab (Figure 6-11).

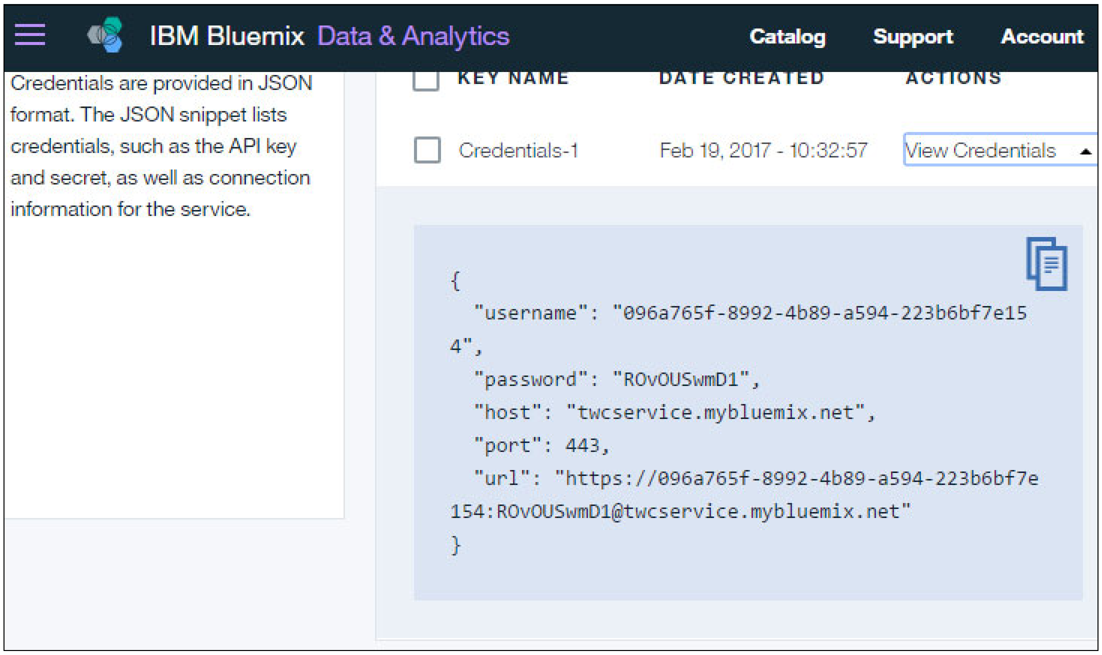


Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection

□	KEY NAME	DATE CREATED	ACTIONS
□	Credentials-1	Feb 19, 2017 - 10:32:57	View Credentials

Figure 6-11 Weather Company Data: Service Credentials tab

5. Under ACTIONS column and in the Credentials-1 row, click **View Credentials** (Figure 6-12) to display the username and password for the service instance. You use this information to test Weather Company Data API in step e on page 170.



Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

□	KEY NAME	DATE CREATED	ACTIONS
□	Credentials-1	Feb 19, 2017 - 10:32:57	View Credentials ▲

```
{
  "username": "096a765f-8992-4b89-a594-223b6bf7e15
4",
  "password": "ROvOUSwmD1",
  "host": "twcservice.mybluemix.net",
  "port": 443,
  "url": "https://096a765f-8992-4b89-a594-223b6bf7e
154:ROvOUSwmD1@twcservice.mybluemix.net"
}
```

Figure 6-12 Weather Company Data: Service Credentials details

Try the Weather Company Data APIs before you use them

Browse through the API documentation and try the APIs before you use them. Complete these steps:

1. Click the **Manage** tab, scroll to Get Started, and click **APIs** (Figure 6-13).

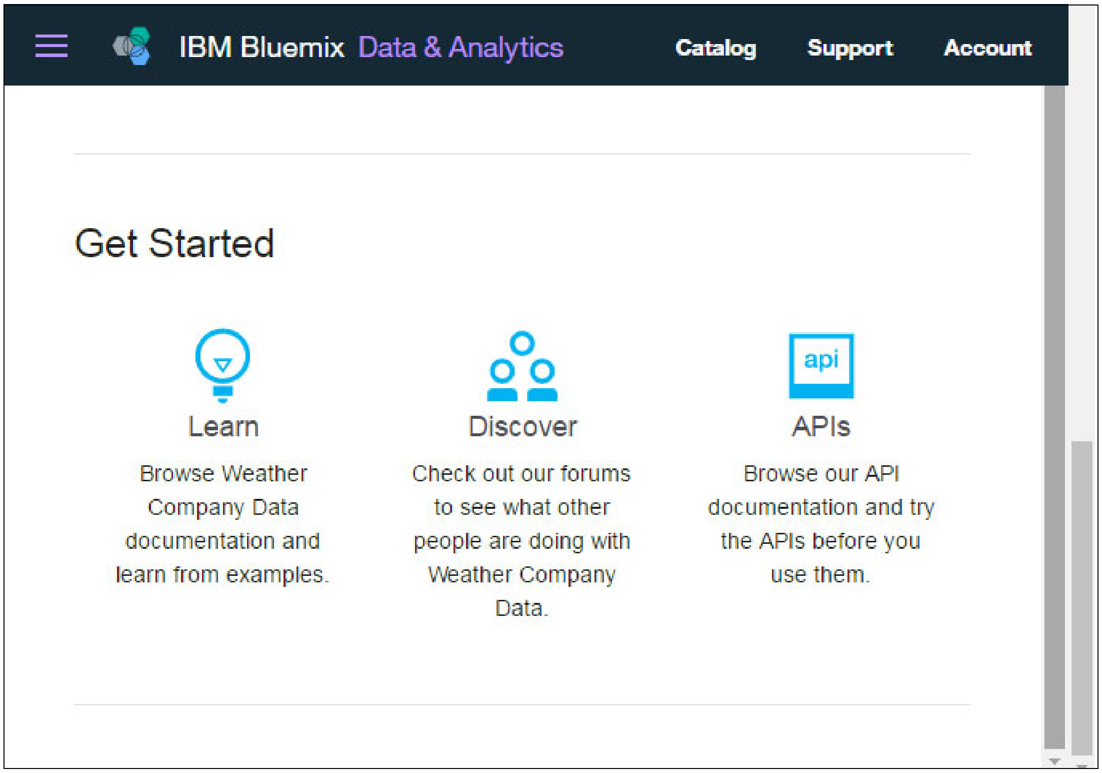
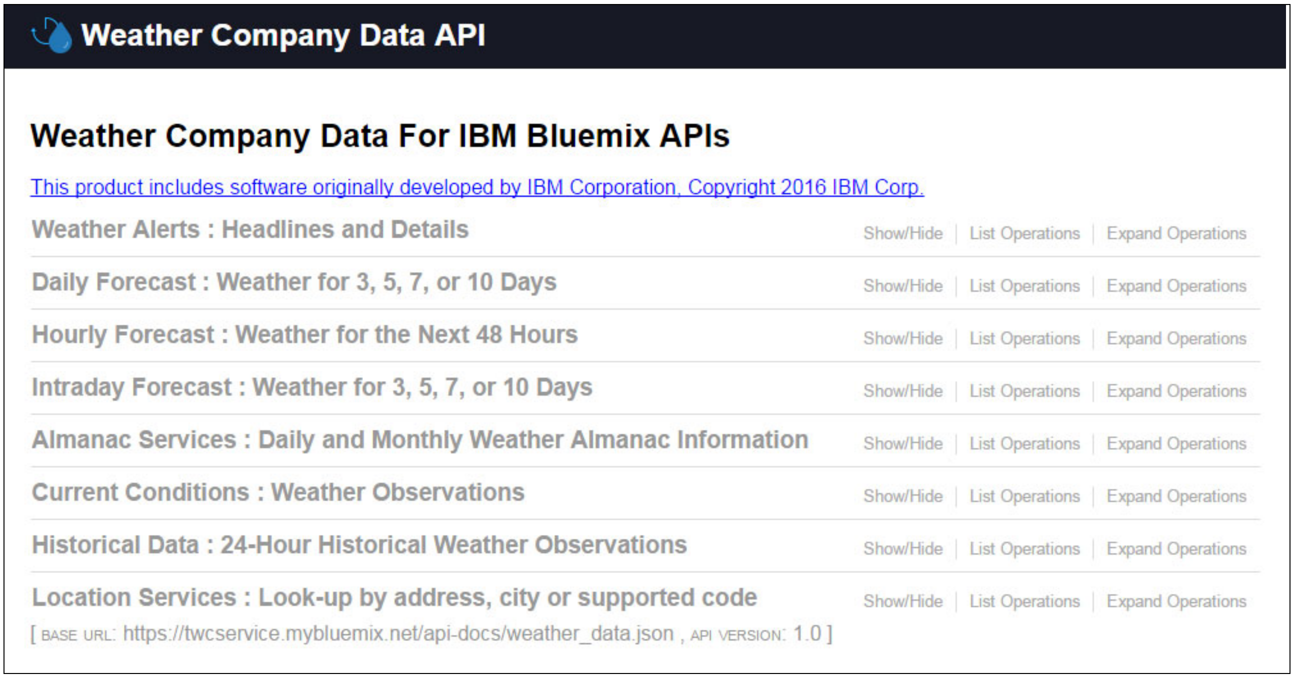

IBM Bluemix Data & Analytics
Catalog
Support
Account
Get Started
Learn
Browse Weather Company Data documentation and learn from examples.
Discover
Check out our forums to see what other people are doing with Weather Company Data.
api
APIs
Browse our API documentation and try the APIs before you use them.

Figure 6-13 Weather Company Data service: Get started

A new tab opens. The Weather Company Data For IBM Bluemix APIs for Bluemix APIs is listed (Figure 6-14).



Weather Company Data API

Weather Company Data For IBM Bluemix APIs

This product includes software originally developed by IBM Corporation, Copyright 2016 IBM Corp.

Weather Alerts : Headlines and Details — Show/Hide | List Operations | Expand Operations

Daily Forecast : Weather for 3, 5, 7, or 10 Days — Show/Hide | List Operations | Expand Operations

Hourly Forecast : Weather for the Next 48 Hours — Show/Hide | List Operations | Expand Operations

Intraday Forecast : Weather for 3, 5, 7, or 10 Days — Show/Hide | List Operations | Expand Operations

Almanac Services : Daily and Monthly Weather Almanac Information — Show/Hide | List Operations | Expand Operations

Current Conditions : Weather Observations — Show/Hide | List Operations | Expand Operations

Historical Data : 24-Hour Historical Weather Observations — Show/Hide | List Operations | Expand Operations

Location Services : Look-up by address, city or supported code — Show/Hide | List Operations | Expand Operations

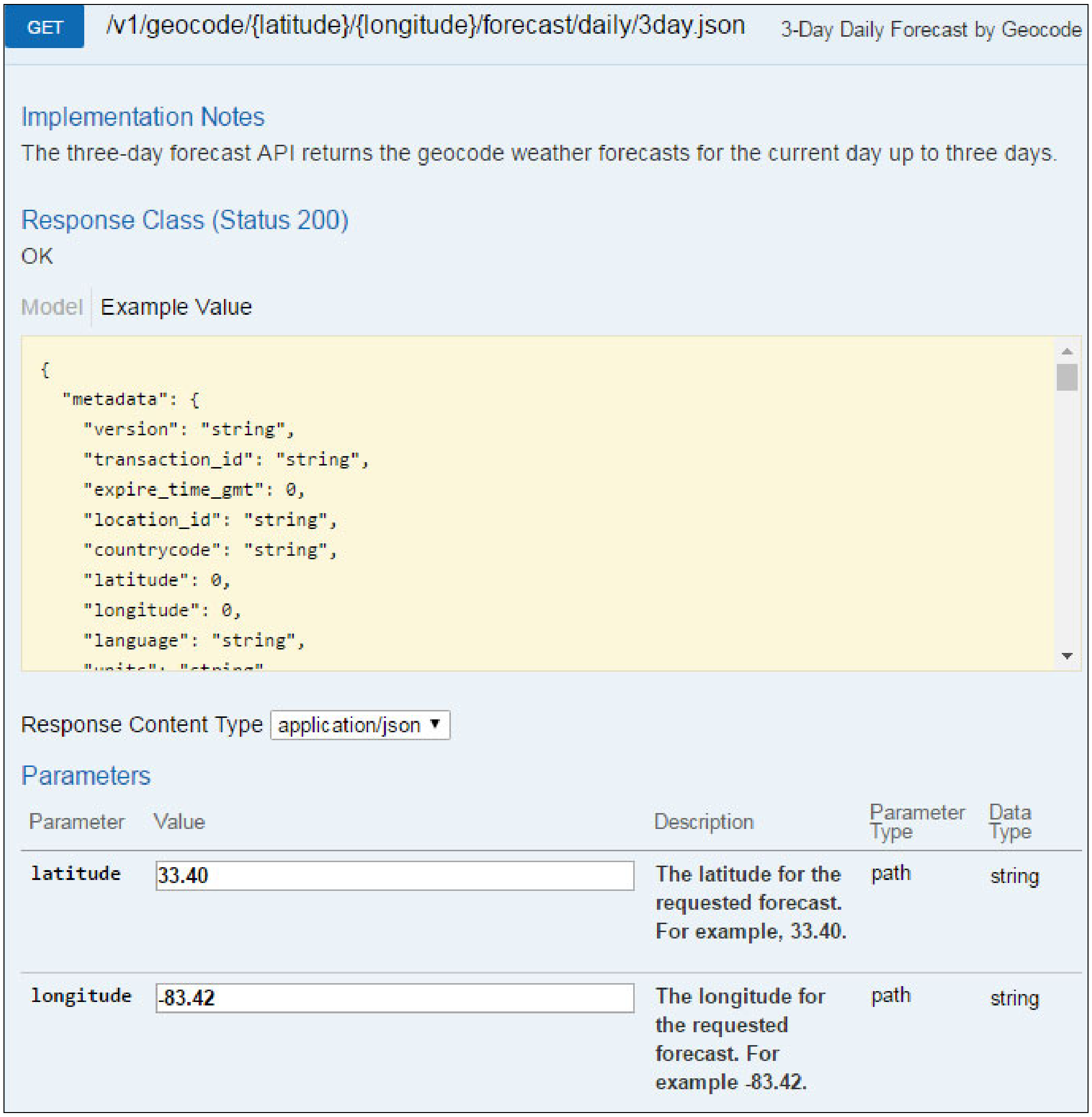
[BASE URL: https://twcservice.mybluemix.net/api-docs/weather_data.json , API VERSION: 1.0]

Figure 6-14 Weather Company Data APIs

2. In these steps, use `Daily Forecast for 3 days` to get the forecast of the weather for tomorrow:

   a. Click **Daily Forecast**.

   b. Click **GET /v1/geocode/{latitude}/{longitude}/forecast/daily/3day.json** (Figure 6-15).



/v1/geocode/{latitude}/{longitude}/forecast/daily/3day.json 3-Day Daily Forecast by Geocode

Implementation Notes

The three-day forecast API returns the geocode weather forecasts for the current day up to three days.

Response Class (Status 200)

OK

```
{
  "metadata": {
    "version": "string",
    "transaction_id": "string",
    "expire_time_gmt": 0,
    "location_id": "string",
    "countrycode": "string",
    "latitude": 0,
    "longitude": 0,
    "language": "string",
    "units": "string"
```

Response Content Type application/json ▾

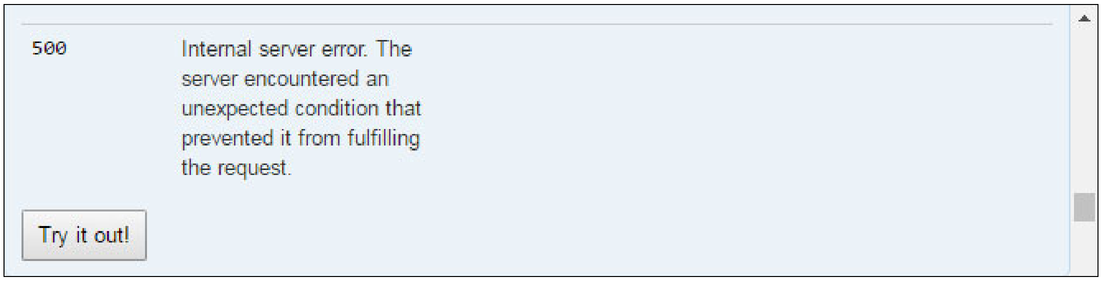
Parameters

Parameter	Value	Description	Parameter Type	Data Type
`latitude`	33.40	The latitude for the requested forecast. For example, 33.40.	path	string
`longitude`	-83.42	The longitude for the requested forecast. For example -83.42.	path	string

Figure 6-15 Three-day forecast Weather Company Data API

   c. In the latitude, and longitude text boxes, type the latitude and longitude of any city. For example, Cairo's latitude is 30.0444, and longitude is 31.2357.

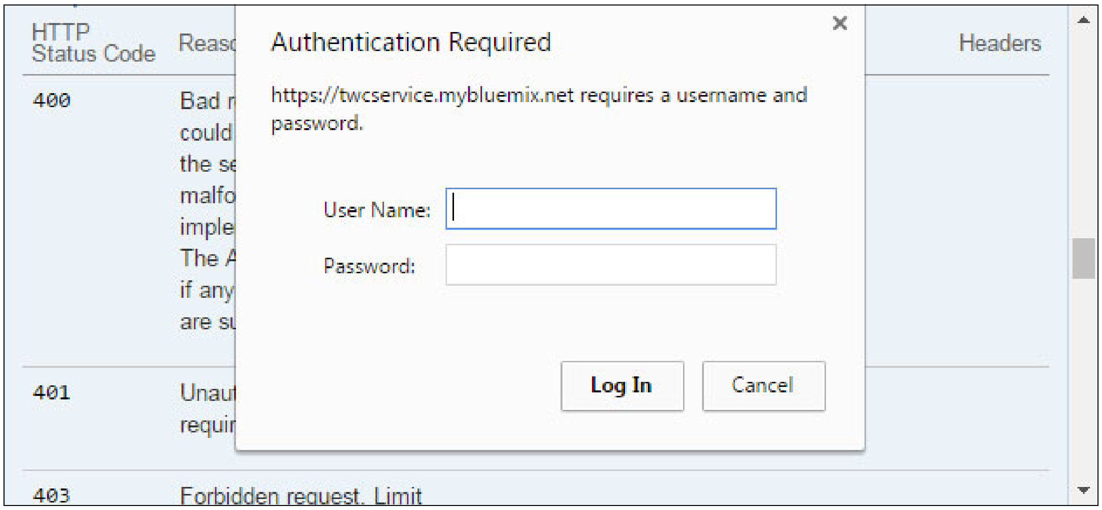
d. Scroll to the bottom and click on **Try it out** (Figure 6-16).



500 Internal server error. The server encountered an unexpected condition that prevented it from fulfilling the request.

Figure 6-16 Testing three-day forecast Weather Company Data API

e. Authentication is required; you are prompted for the user name and password of the Weather Company Data service instance credentials that you obtained in step 5 on page 166. Provide your service credentials to log in (Figure 6-17).

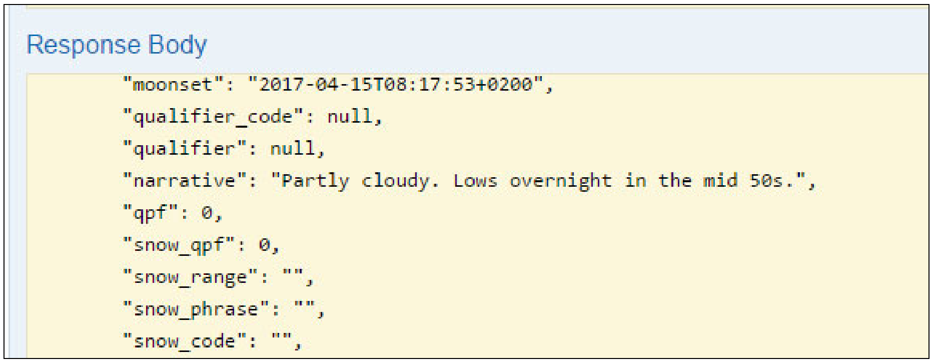


Authentication Required

https://twcservice.mybluemix.net requires a username and password.

Figure 6-17 Testing three-day forecast Weather Company Data API - Authentication

f. The three-day forecast API returns the geocode weather forecasts for the current day and up to three days. The response of the service call is displayed in Response Body section (Figure 6-18).



Response Body

```
        "moonset": "2017-04-15T08:17:53+0200",
        "qualifier_code": null,
        "qualifier": null,
        "narrative": "Partly cloudy. Lows overnight in the mid 50s.",
        "qpf": 0,
        "snow_qpf": 0,
        "snow_range": "",
        "snow_phrase": "",
        "snow_code": "",
```

Figure 6-18 Weather Company Data Response

Example 6-1 shows the weather forecast for tomorrow is under `forecasts[1].narrative`.

Example 6-1 Response body snippet

```
"forecasts": [
    {
      "class": "fod_long_range_daily",
      "expire_time_gmt": 1492289627,
      "fcst_valid": 1492232400,
      "fcst_valid_local": "2017-04-15T07:00:00+0200",
      "num": 1,
      "max_temp": null,
      "min_temp": 54,
      "torcon": null,
      "stormcon": null,
      "blurb": null,
      "blurb_author": null,
      "lunar_phase_day": 18,
      "dow": "Saturday",
      "lunar_phase": "Waning Gibbous",
      "lunar_phase_code": "WNG",
      "sunrise": "2017-04-15T05:32:08+0200",
      "sunset": "2017-04-15T18:28:06+0200",
      "moonrise": "2017-04-15T22:13:12+0200",
      "moonset": "2017-04-15T08:17:53+0200",
      "qualifier_code": null,
      "qualifier": null,
      "narrative": "Partly cloudy. Lows overnight in the mid 50s.",
      "qpf": 0,
      "snow_qpf": 0,
      "snow_range": "",
      "snow_phrase": "",
      "snow_code": "",
      "night": {
```

6.4.3 Developing the Cognitive Weather Forecast chatbot application

This section describes how to develop the application logic by creating a Node.js application that integrates with the Conversation service and the Weather Company Data service. You start by cloning a sample Node.js app, which is a simple chatbot, and deploy it to your Bluemix workspace.

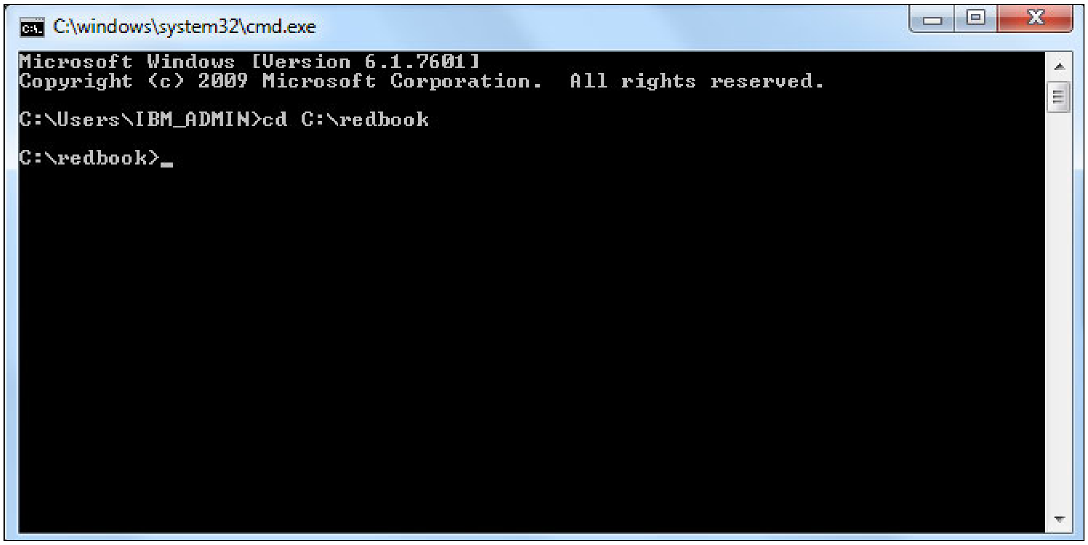
The steps are summarized in the following list:

1. “Clone the Conversation sample app” on page 172
2. “Integrate the application with the Conversation and Weather Company Data services” on page 173
3. “Push the application to Bluemix” on page 177

Clone the Conversation sample app

Clone the *incomplete* repository:

1. Create a new `C:\redbook` directory.
2. Open a command prompt (`cmd.exe`).
3. Open that directory by using the **`cd C:\redbook`** command (Figure 6-19).

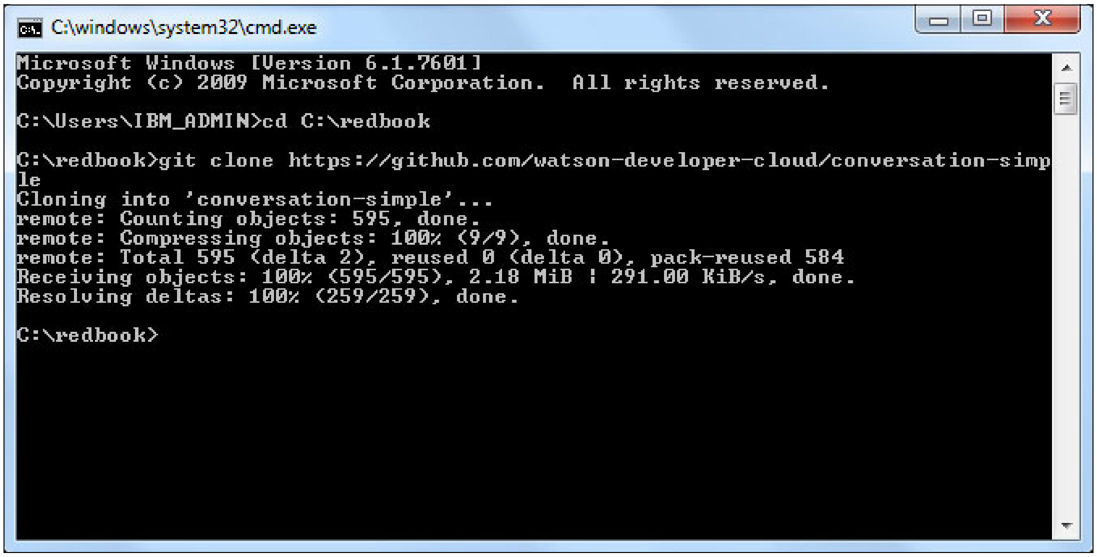


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IBM_ADMIN>cd C:\redbook

C:\redbook>_
```

Figure 6-19 Command to open the directory

4. Clone the incomplete repository (Figure 6-20). Run the following Git command:

   `git clone https://github.com/watson-developer-cloud/conversation-simple`



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IBM_ADMIN>cd C:\redbook

C:\redbook>git clone https://github.com/watson-developer-cloud/conversation-simp
le
Cloning into 'conversation-simple'...
remote: Counting objects: 595, done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 595 (delta 2), reused 0 (delta 0), pack-reused 584
Receiving objects: 100% (595/595), 2.18 MiB | 291.00 KiB/s, done.
Resolving deltas: 100% (259/259), done.

C:\redbook>
```

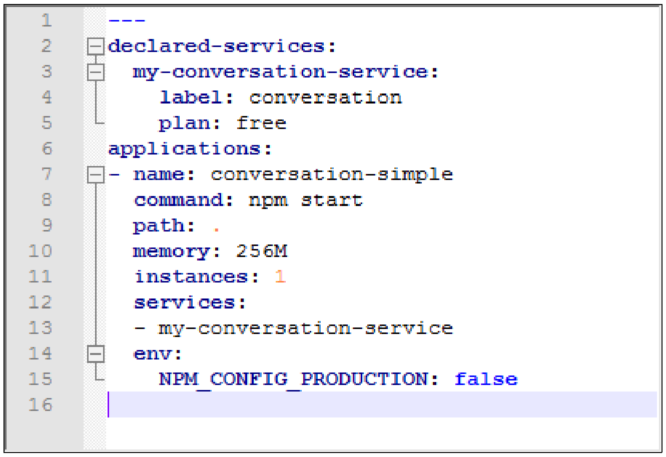
Figure 6-20 Clone the repository with the incomplete code

Integrate the application with the Conversation and Weather Company Data services

Modify the code to integrate the application with the Conversation and Weather Company Data services:

1. Update the `manifest.yml` file with the host name and the details of the Conversation service and the Weather Company Data service:

   a. Open `C:\redbook\conversation-simple\manifest.yml` (Figure 6-21) with your favorite text editor (Figure 6-21).



```
---
declared-services:
  my-conversation-service:
    label: conversation
    plan: free
applications:
- name: conversation-simple
  command: npm start
  path: .
  memory: 256M
  instances: 1
  services:
  - my-conversation-service
  env:
    NPM_CONFIG_PRODUCTION: false
```

Figure 6-21 The manifest.yml file before the update

   b. Update `declared-services` section (Example 6-4 on page 174). In this section, replace lines with the name and details of your Conversation and Weather Company Data service instances (Example 6-2).

Example 6-2 Name and details

```
Conversation:
    label: conversation
    plan: free
  weather-company-data:
    label: weather
    plan: free
```

   c. In the `applications` section, change the application name to `conv-201-xxx-weather`. Replace `xxx` by a random number because this name will also be used as the hostname for your application so it needs to be unique.

   d. In the `services` section, add an application dependency on the declared services (Example 6-3).

Example 6-3 Add application dependency

```
- Conversation
- weather-company-data
```

   e. For `memory`, increase the memory to `512M`.

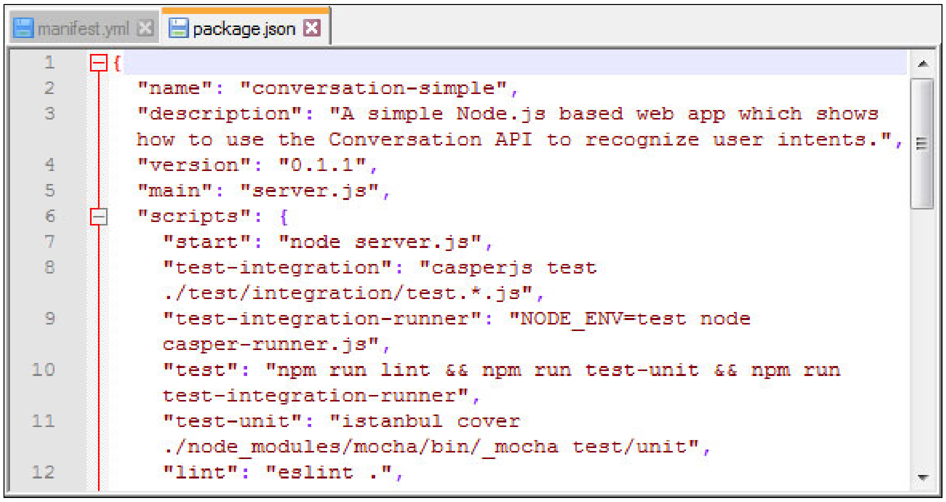
   f. Save the file. It should look like Example 6-4 on page 174.

Example 6-4 The manifest.yml file after the update with the values for this use case

```
---
declared-services:
  Conversation:
    label: conversation
    plan: free
  weather-company-data:
    label: weather
    plan: free
applications:
- name: conv-201-xxx-weather
  command: npm start
  path: .
  memory: 512M
  instances: 1
  services:
  - Conversation
  - weather-company-data
  env:
    NPM_CONFIG_PRODUCTION: false
```

2. Add the `request` module to `package.json`. The `request` module is a third-party module that allows making HTTP calls. Here it is used for interaction with REST APIs exposed by the Weather Company Data service.

   a. Open `C:\redbook\conversation-simple\package.json` (Figure 6-22).



```
{
  "name": "conversation-simple",
  "description": "A simple Node.js based web app which shows how to use the Conversation API to recognize user intents.",
  "version": "0.1.1",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "test-integration": "casperjs test ./test/integration/test.*.js",
    "test-integration-runner": "NODE_ENV=test node casper-runner.js",
    "test": "npm run lint && npm run test-unit && npm run test-integration-runner",
    "test-unit": "istanbul cover ./node_modules/mocha/bin/_mocha test/unit",
    "lint": "eslint .",
```

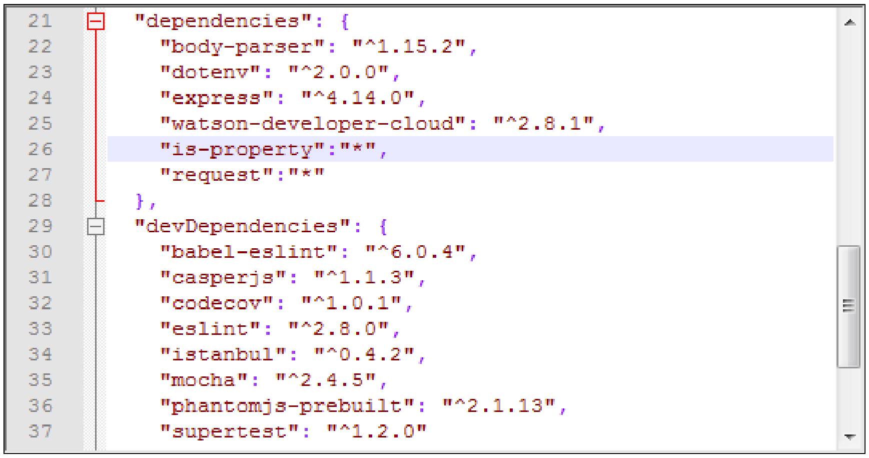
Figure 6-22 The package.json file

   b. Add the latest version of the `"is-property"` and `"request"` modules (Example 6-5) as a dependency on the `dependencies` tag (Figure 6-23 on page 175).

Example 6-5 Add request and is-property

```
"is-property":"*",
"request":"*"
```

Figure 6-23 shows dependencies.



```
"dependencies": {
  "body-parser": "^1.15.2",
  "dotenv": "^2.0.0",
  "express": "^4.14.0",
  "watson-developer-cloud": "^2.8.1",
  "is-property":"*",
  "request":"*"
},
"devDependencies": {
  "babel-eslint": "^6.0.4",
  "casperjs": "^1.1.3",
  "codecov": "^1.0.1",
  "eslint": "^2.8.0",
  "istanbul": "^0.4.2",
  "mocha": "^2.4.5",
  "phantomjs-prebuilt": "^2.1.13",
  "supertest": "^1.2.0"
```

Figure 6-23 The package.json: dependencies

c. Save the file.

3. Edit the application logic to integrate with the Conversation and Weather Company Data services:

a. Open the `C:\redbook\conversation-simple\app.js` file.

b. After the `updateMessage` function, add the function `getLocationCoordinatesForCity` (Example 6-6) to get the latitude and longitude for cities.

Example 6-6 Get latitude and longitude for cities

```
/**
 * Get the latitude and longitude of city
 * @param  {Object} city The target city
 * @return {Object} The latitude and longitude of the city
 */
function getLocationCoordinatesForCity(city) {
    var location = {};
    if (city === 'Cairo') {
        location.latitude = '30.0444';
        location.longitude = '31.2357';
    } else if (city === 'NYC') {
        location.latitude = '40.7128';
        location.longitude = '74.0059';
    }
    return location;
}
```

c. After the last function, add the function`getWeatherForecastForCity` (Example 6-7) that gets tomorrow's weather forecast for a city by calling a Weather Company Data API.

Example 6-7 Get tomorrows weather

```
var request = require('request'); // request module
//Weather Company Endpoint
var vcap = JSON.parse(process.env.VCAP_SERVICES);
var weatherCompanyEndpoint = vcap.weatherinsights[0].credentials.url;
/**
 * Get the weather forecast for a city through calling Weather Company Data
 * @param  {Object} city The target city
 * @return {Object} Weather Forecast for the specified city.
 */
function getWeatherForecastForCity(location, callback) {

    var options = {
        url: weatherCompanyEndpoint + '/api/weather/v1/geocode/' +
location.latitude + '/' + location.longitude + '/forecast/daily/3day.json'
    };
    request(
        options,
        function(error, response, body) {
            try {
                var json = JSON.parse(body);
                var weatherOutput = json.forecasts[1].narrative;
                callback(null, weatherOutput);
            } catch (e) {
                callback(e, null);
            }
        }
    );
};
```

d. Replace the `updateMessage` function with the function in Example 6-8. If the entity is city, then get the location coordinates for the city and call a Weather Company Data API to get the forecast for this city.

Example 6-8 Replacement for updateMessage function

```
/**
 * Updates the response text using the intent confidence
 * @param  {Object} input The request to the Conversation service
 * @param  {Object} response The response from the Conversation service
 * @param  {Object} callback The response from Weather Company Data
 * @return {Object}          The response with the updated message
 */
function updateMessage(input, response, callback) {
    var responseText = null;
    if (!response.output) {
        response.output = {};
        callback(response);
    }
    // In case the entity is city, then get the location coordinates for the city and call
    // Weather Company Data to get the forecast for this city.
    else if (response.entities.length > 0 && response.entities[0].entity === 'city') {
        var location = getLocationCoordinatesForCity(response.entities[0].value);
        getWeatherForecastForCity(location, function(e, weatherOutput) {
            response.output.text[0] = weatherOutput;
            callback(response);

        });
    } else {
        callback(response);
    }
}
```

e. Call the updated `updateMessage` function. In line 61, replace the message call as in Example 6-9.

Example 6-9 Replace message call

```
updateMessage(payload, data, function(response) {
      return res.json(response);
});
```

f. Save the file.

Note: You can find the full listing of the `app.js` code at this GitHub location:

https://github.com/snippet-java/redbooks-conv-201-weather-nodejs/blob/master/app.js

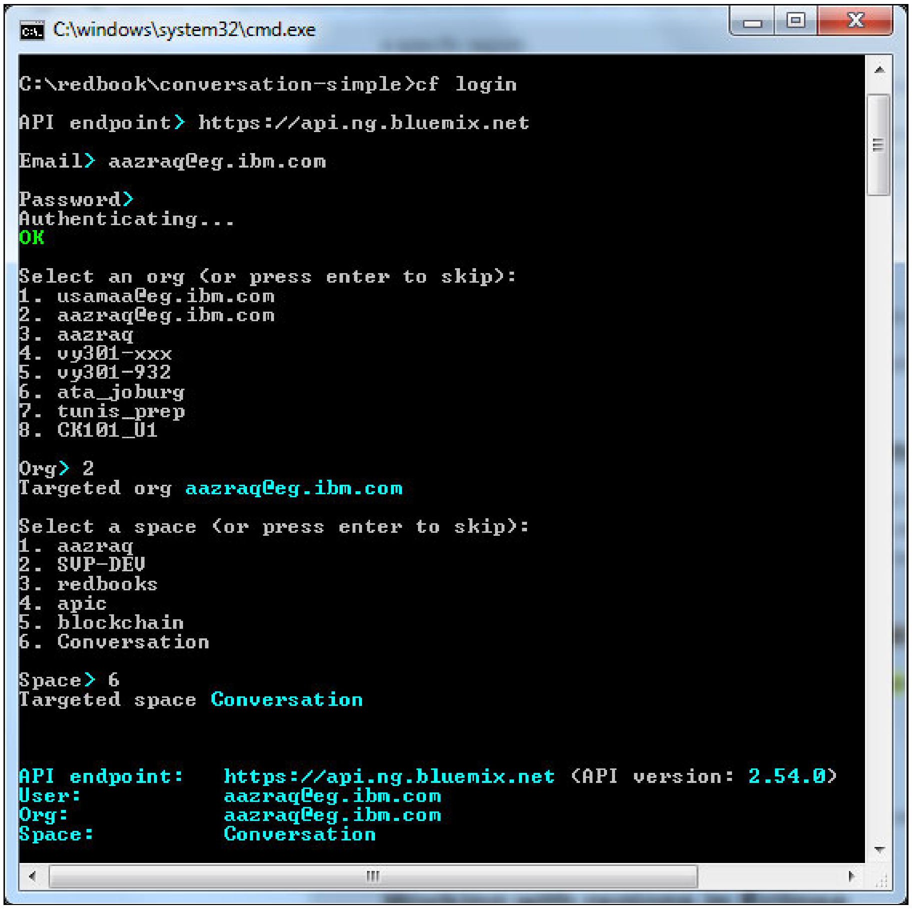
Push the application to Bluemix

Push the modified code to Bluemix:

1. At the command prompt, change to the `C:\redbook\conversation-simple` directory:

   ```
   cd C:\redbook\conversation-simple
   ```

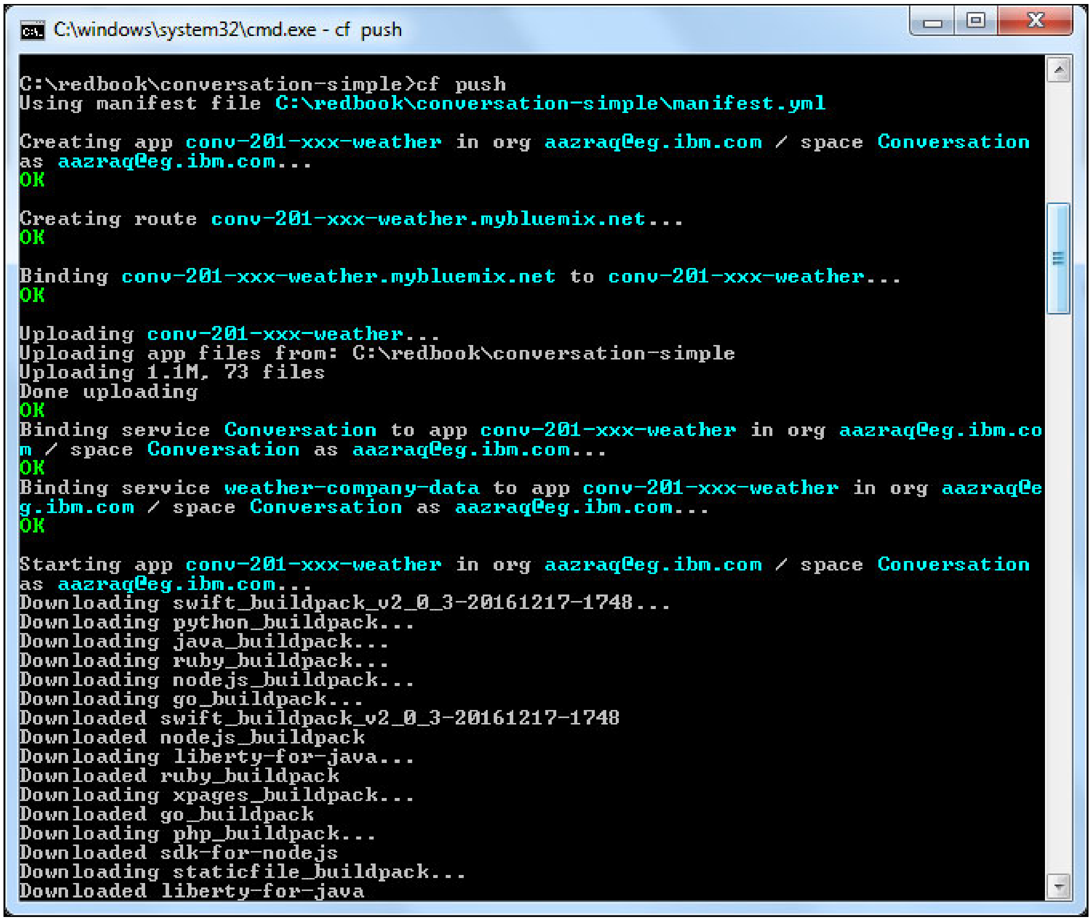
2. Log in to Cloud Foundry by using the **`cf login`** command (Figure 6-24). When prompted enter the email and password that you use to log in to your Bluemix account.



```
C:\redbook\conversation-simple>cf login

API endpoint> https://api.ng.bluemix.net

Email> aazraq@eg.ibm.com

Password>
Authenticating...
OK

Select an org (or press enter to skip):
1. usamaa@eg.ibm.com
2. aazraq@eg.ibm.com
3. aazraq
4. vy301-xxx
5. vy301-932
6. ata_joburg
7. tunis_prep
8. CK101_U1

Org> 2
Targeted org aazraq@eg.ibm.com

Select a space (or press enter to skip):
1. aazraq
2. SVP-DEV
3. redbooks
4. apic
5. blockchain
6. Conversation

Space> 6
Targeted space Conversation


API endpoint:   https://api.ng.bluemix.net (API version: 2.54.0)
User:           aazraq@eg.ibm.com
Org:            aazraq@eg.ibm.com
Space:          Conversation
```

Figure 6-24 Log in to Cloud Foundry (cf login)

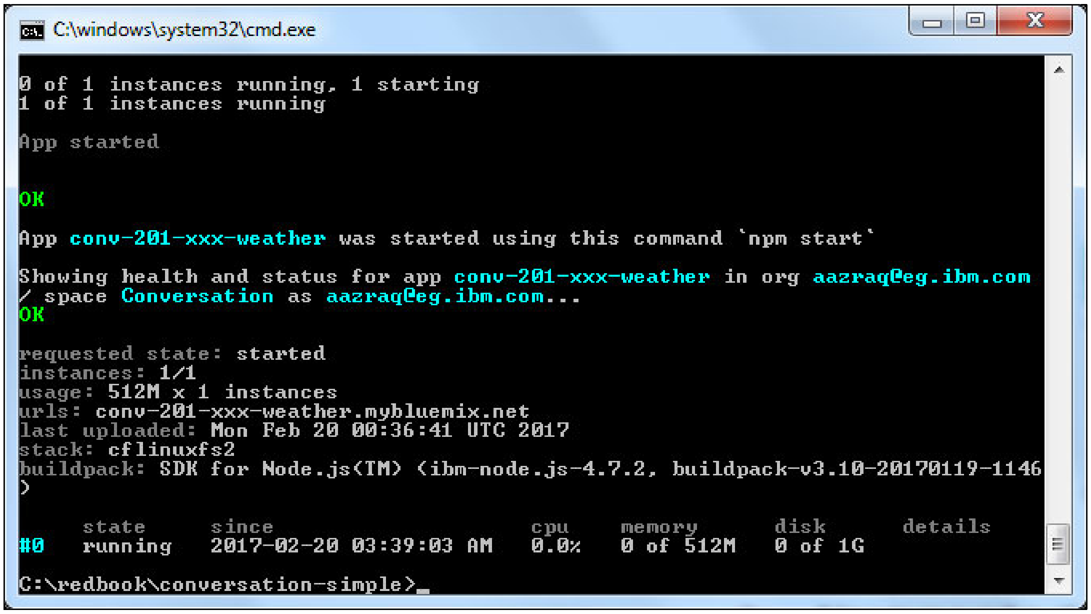
3. Push the application to Bluemix by using the **cf push** command (Figure 6-25).



```
C:\redbook\conversation-simple>cf push
Using manifest file C:\redbook\conversation-simple\manifest.yml

Creating app conv-201-xxx-weather in org aazraq@eg.ibm.com / space Conversation
as aazraq@eg.ibm.com...
OK

Creating route conv-201-xxx-weather.mybluemix.net...
OK

Binding conv-201-xxx-weather.mybluemix.net to conv-201-xxx-weather...
OK

Uploading conv-201-xxx-weather...
Uploading app files from: C:\redbook\conversation-simple
Uploading 1.1M, 73 files
Done uploading
OK
Binding service Conversation to app conv-201-xxx-weather in org aazraq@eg.ibm.co
m / space Conversation as aazraq@eg.ibm.com...
OK
Binding service weather-company-data to app conv-201-xxx-weather in org aazraq@e
g.ibm.com / space Conversation as aazraq@eg.ibm.com...
OK

Starting app conv-201-xxx-weather in org aazraq@eg.ibm.com / space Conversation
as aazraq@eg.ibm.com...
Downloading swift_buildpack_v2_0_3-20161217-1748...
Downloading python_buildpack...
Downloading java_buildpack...
Downloading ruby_buildpack...
Downloading nodejs_buildpack...
Downloading go_buildpack...
Downloaded swift_buildpack_v2_0_3-20161217-1748
Downloaded nodejs_buildpack
Downloading liberty-for-java...
Downloaded ruby_buildpack
Downloading xpages_buildpack...
Downloaded go_buildpack
Downloading php_buildpack...
Downloaded sdk-for-nodejs
Downloading staticfile_buildpack...
Downloaded liberty-for-java
```

Figure 6-25 Pushing the application to Bluemix

4. Wait until the build and deployment are completed (Figure 6-26).

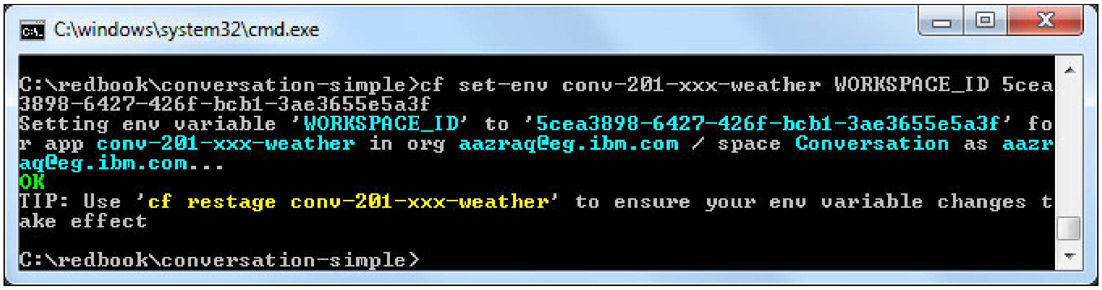


```
0 of 1 instances running, 1 starting
1 of 1 instances running

App started


OK

App conv-201-xxx-weather was started using this command `npm start`

Showing health and status for app conv-201-xxx-weather in org aazraq@eg.ibm.com
/ space Conversation as aazraq@eg.ibm.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: conv-201-xxx-weather.mybluemix.net
last uploaded: Mon Feb 20 00:36:41 UTC 2017
stack: cflinuxfs2
buildpack: SDK for Node.js(TM) (ibm-node.js-4.7.2, buildpack-v3.10-20170119-1146
)

     state     since                    cpu    memory      disk      details
#0   running   2017-02-20 03:39:03 AM   0.0%   0 of 512M   0 of 1G

C:\redbook\conversation-simple>_
```

Figure 6-26 Pushing application completed

5. Set the `WORKSPACE_ID` environment variable to point to the Weather Forecast Conversation Workspace ID that you obtained in “Get the Workspace ID” on page 163 (Figure 6-27):

```
cf set-env conv-201-<xxx>-weather WORKSPACE_ID <WORKSPACE_ID>
```

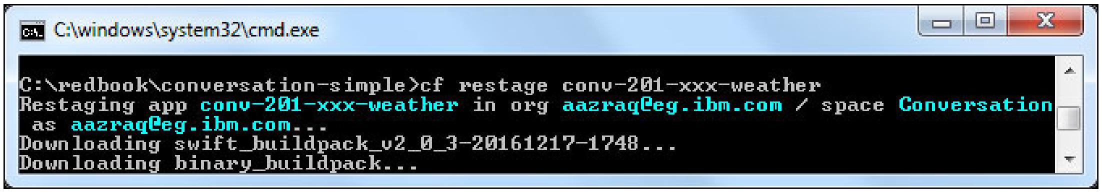


```
C:\redbook\conversation-simple>cf set-env conv-201-xxx-weather WORKSPACE_ID 5cea
3898-6427-426f-bcb1-3ae3655e5a3f
Setting env variable 'WORKSPACE_ID' to '5cea3898-6427-426f-bcb1-3ae3655e5a3f' fo
r app conv-201-xxx-weather in org aazraq@eg.ibm.com / space Conversation as aazr
aq@eg.ibm.com...
OK
TIP: Use 'cf restage conv-201-xxx-weather' to ensure your env variable changes t
ake effect

C:\redbook\conversation-simple>
```

Figure 6-27 Set the environment variable

6. Restage the application so that your environment variable changes take effect (Figure 6-28):

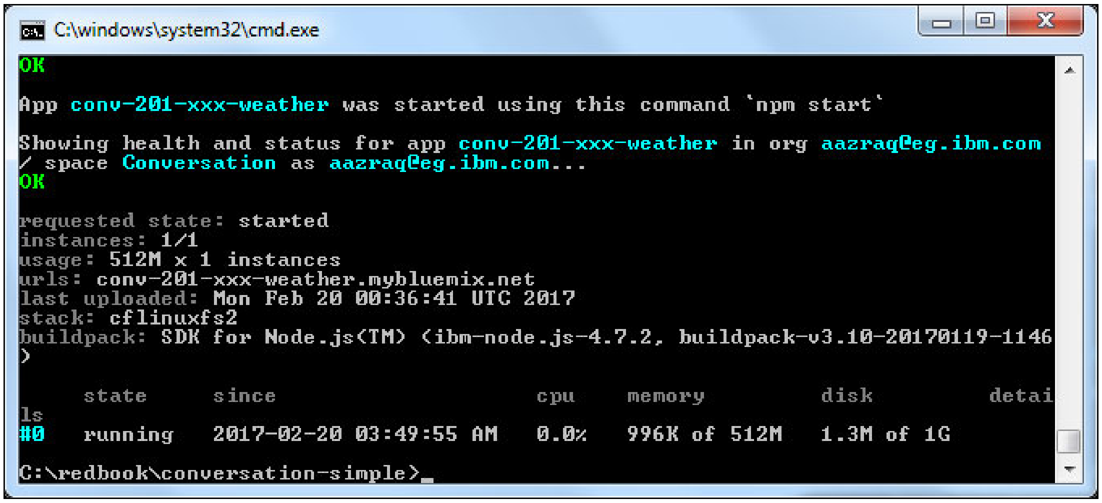
```
cf restage conv-201-<xxx>-weather
```



```
C:\redbook\conversation-simple>cf restage conv-201-xxx-weather
Restaging app conv-201-xxx-weather in org aazraq@eg.ibm.com / space Conversation
 as aazraq@eg.ibm.com...
Downloading swift_buildpack_v2_0_3-20161217-1748...
Downloading binary_buildpack...
```

Figure 6-28 Restage the application

7. Wait until the application is running (Figure 6-29).



```
OK

App conv-201-xxx-weather was started using this command `npm start`

Showing health and status for app conv-201-xxx-weather in org aazraq@eg.ibm.com
/ space Conversation as aazraq@eg.ibm.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: conv-201-xxx-weather.mybluemix.net
last uploaded: Mon Feb 20 00:36:41 UTC 2017
stack: cflinuxfs2
buildpack: SDK for Node.js(TM) (ibm-node.js-4.7.2, buildpack-v3.10-20170119-1146
)

     state     since                    cpu    memory         disk          detai
ls
#0   running   2017-02-20 03:49:55 AM   0.0%   996K of 512M   1.3M of 1G

C:\redbook\conversation-simple>_
```

Figure 6-29 Restaging completed

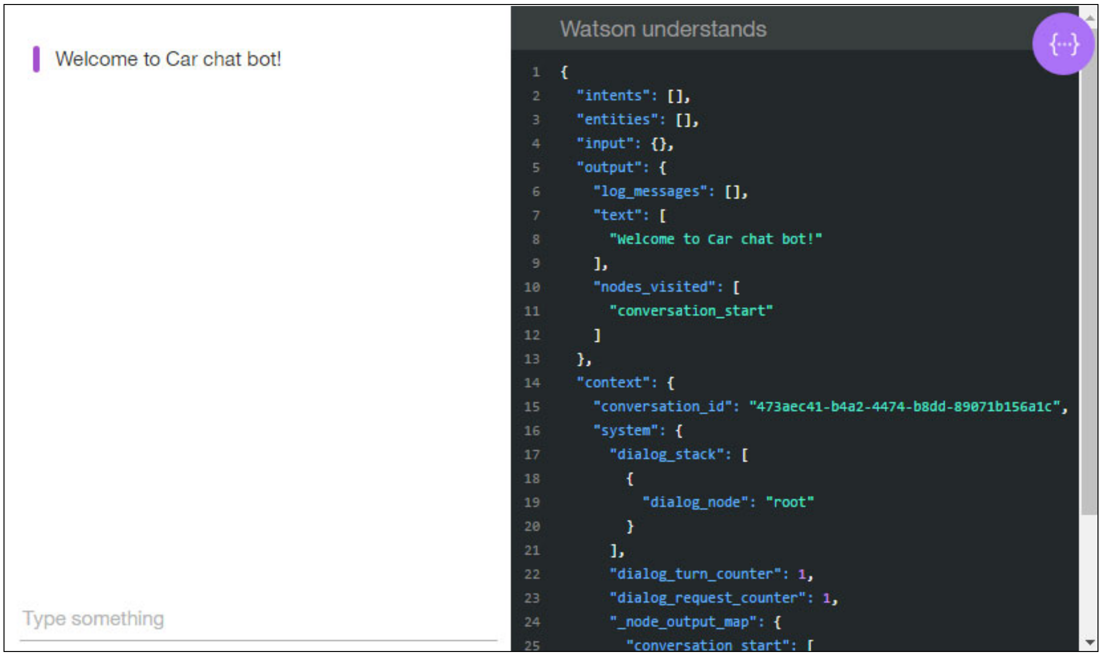
6.4.4 Testing the application

To test the application, follow these steps:

1. Open your application route (URL to access your application) in a web browser; `xxx` is the number you use to make your application name unique:

   `http://conv-201-xxx-weather.mybluemix.net/`

   Your application opens in the browser (Figure 6-30).

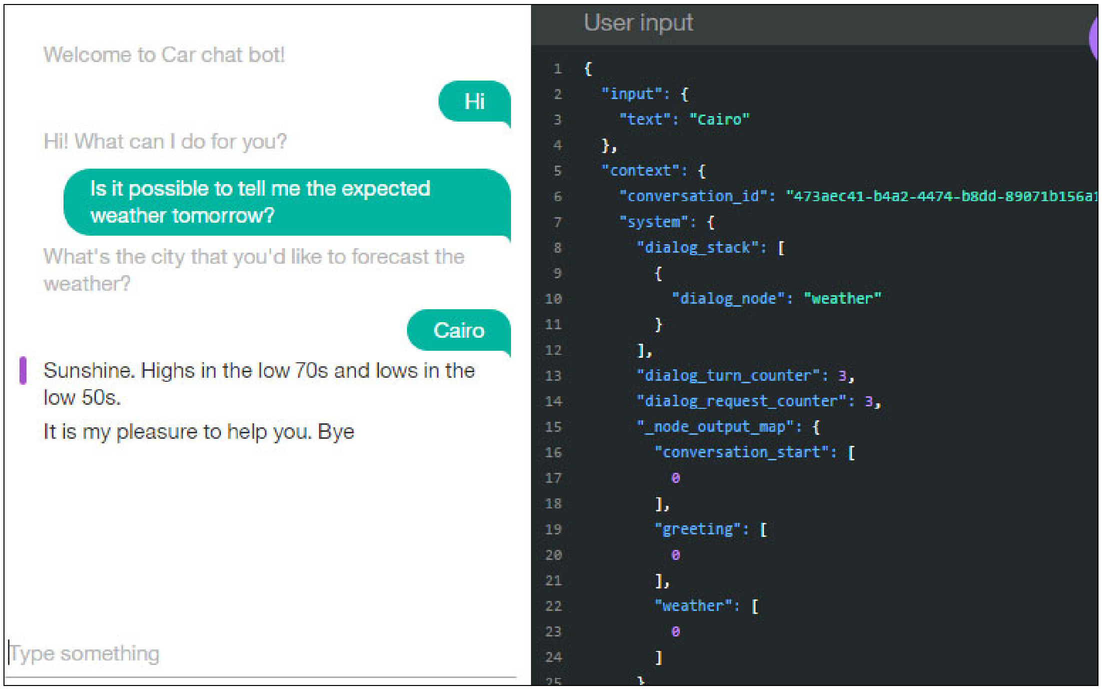


Welcome to Car chat bot!

```
{
  "intents": [],
  "entities": [],
  "input": {},
  "output": {
    "log_messages": [],
    "text": [
      "Welcome to Car chat bot!"
    ],
    "nodes_visited": [
      "conversation_start"
    ]
  },
  "context": {
    "conversation_id": "473aec41-b4a2-4474-b8dd-89071b156a1c",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1,
      "_node_output_map": {
        "conversation_start": [
```

Figure 6-30 Cognitive Weather Forecast chatbot

2. Get the weather for one of the two supported cities (Figure 6-31 on page 181).



Welcome to Car chat bot!

Hi

Hi! What can I do for you?

Is it possible to tell me the expected weather tomorrow?

What's the city that you'd like to forecast the weather?

Cairo

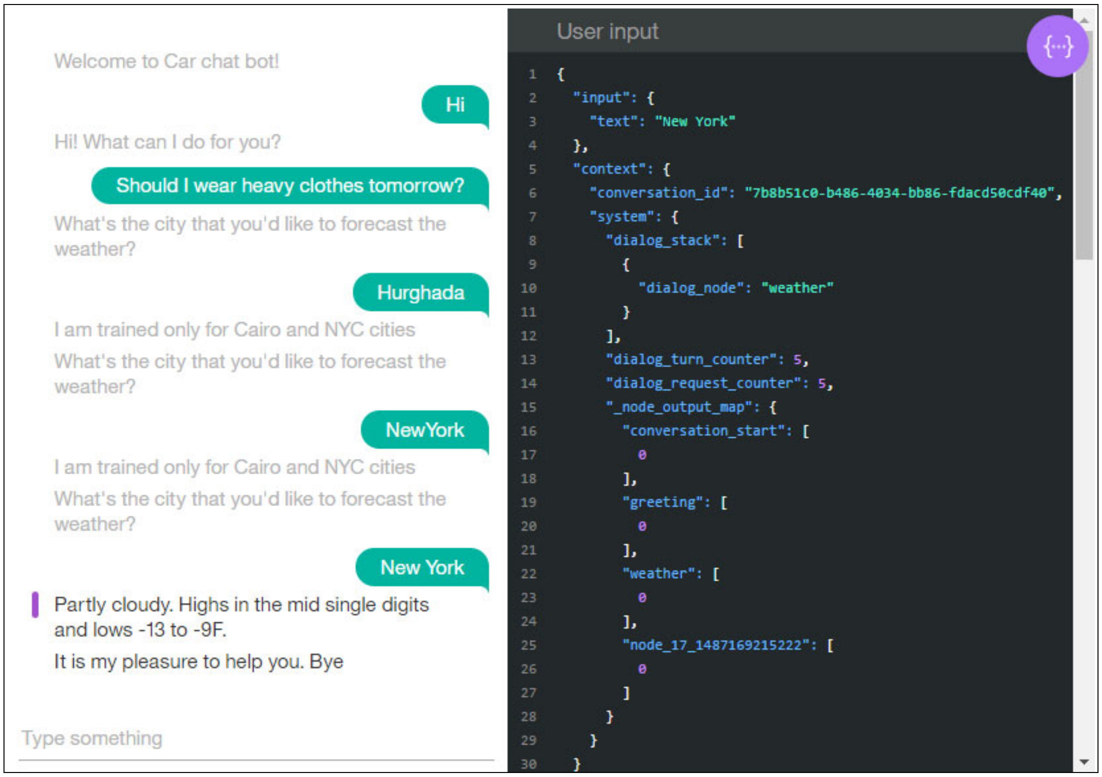
Sunshine. Highs in the low 70s and lows in the low 50s.

It is my pleasure to help you. Bye

```
{
  "input": {
    "text": "Cairo"
  },
  "context": {
    "conversation_id": "473aec41-b4a2-4474-b8dd-89071b156a1
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "weather"
        }
      ],
      "dialog_turn_counter": 3,
      "dialog_request_counter": 3,
      "_node_output_map": {
        "conversation_start": [
          0
        ],
        "greeting": [
          0
        ],
        "weather": [
          0
        ]
      }
```

Figure 6-31 Getting the weather for Cairo on the Cognitive Weather Forecast chatbot

3. Try different scenarios (Figure 6-32). If the chatbot fails, more training is necessary. To provide more training, add more user examples to the intents in the Car Chat-bot Workspace, or edit the entities. Also you can add support for more cities.



Welcome to Car chat bot!

Hi

Hi! What can I do for you?

Should I wear heavy clothes tomorrow?

What's the city that you'd like to forecast the weather?

Hurghada

I am trained only for Cairo and NYC cities

What's the city that you'd like to forecast the weather?

NewYork

I am trained only for Cairo and NYC cities

What's the city that you'd like to forecast the weather?

New York

Partly cloudy. Highs in the mid single digits and lows -13 to -9F.

It is my pleasure to help you. Bye

```
{
  "input": {
    "text": "New York"
  },
  "context": {
    "conversation_id": "7b8b51c0-b486-4034-bb86-fdacd50cdf40",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "weather"
        }
      ],
      "dialog_turn_counter": 5,
      "dialog_request_counter": 5,
      "_node_output_map": {
        "conversation_start": [
          0
        ],
        "greeting": [
          0
        ],
        "weather": [
          0
        ],
        "node_17_1487169215222": [
          0
        ]
      }
    }
  }
```

Figure 6-32 Scenarios for Cognitive Weather Forecast chatbot; more training is needed

6.5 Quick deployment of application

A second Git repository is provided so that you can build and deploy the full Cognitive Weather Forecast chatbot even if you did not perform the steps described in 6.4, “Step-by-step implementation” on page 160. This section is independent from the rest of the chapter and it contains instructions to run the app more quickly.

The full version of the code is in the Git repository:

https://github.com/snippet-java/redbooks-conv-201-weather-nodejs

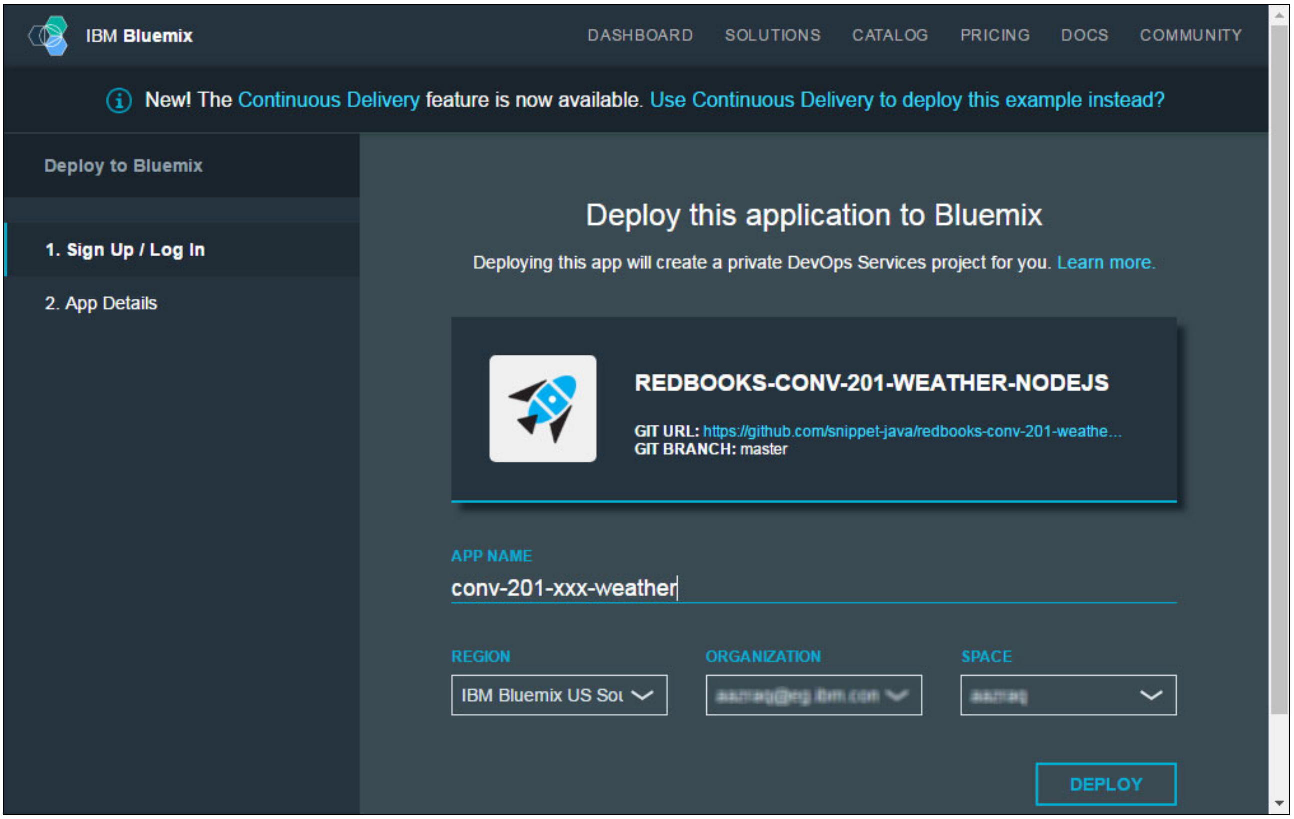
The workspace that was created for this chapter is in the following GitHub location:

https://github.com/snippet-java/redbooks-conv-201-weather-nodejs/blob/master/training/1.4-conv-101-createservice.json

To deploy the full application directly and more quickly, use these steps:

1. Open this location:

   https://bluemix.net/deploy?repository=https://github.com/snippet-java/redbooks-conv-201-weather-nodejs
2. Log in with your Bluemix ID and password.
3. Enter the application name `conv-201-xxx-weather` where xxx is any random number to make your application and host name unique.
4. Click **Deploy** (Figure 6-33).



New! The Continuous Delivery feature is now available. Use Continuous Delivery to deploy this example instead?

Deploy this application to Bluemix

Deploying this app will create a private DevOps Services project for you. Learn more.

REDBOOKS-CONV-201-WEATHER-NODEJS

GIT URL: https://github.com/snippet-java/redbooks-conv-201-weathe...
GIT BRANCH: master

APP NAME: conv-201-xxx-weather

REGION: IBM Bluemix US Sou

ORGANIZATION: [redacted]

SPACE: [redacted]

Figure 6-33 Quick deployment of the application

5. Follow the steps in 6.4.1, “Configuring Conversation workspace for Cognitive Weather Forecast chatbot” on page 161, to import the Car Chat-bot Workspace into your Conversation service. Record the workspace ID.
6. Configure your application to point to the Calculator Workspace by following these three steps:
   - 5 on page 179
   - 6 on page 179
   - 7 on page 179
7. Test the application as described in 6.4.4, “Testing the application” on page 180.

6.6 References

For helpful information, see the following resources:

- Explore Weather Company Data documentation and learn from examples:

  `https://console.ng.bluemix.net/docs/services/Weather/index.html`

- Explore the REST API documentation for Weather Company Data:

  `https://twcservice.mybluemix.net/rest-api/`

7

Improving chatbot understanding

One of the major challenges in developing a conversational interface is anticipating every possible way in which your users will try to communicate with your chatbot.

The *Improve* component of the Conversation service provides a history of conversations with users. You can use this history to improve your chatbot's understanding of user input.

This chapter has an example of how to use the Improve interface to access user conversation logs and identify intents and entities that are not recognized by the sample workspace. The example in this chapter shows how you can improve the workspace understanding.

The following topics are covered in this chapter:

- Getting started
- Use case implementation
- References

7.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

7.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Review past interactions and train the Conversation service with intent examples.
- Review past interactions and train the Conversation service with new entity synonyms.

7.1.2 Prerequisites

To complete the steps in this chapter, be sure these prerequisites are met:

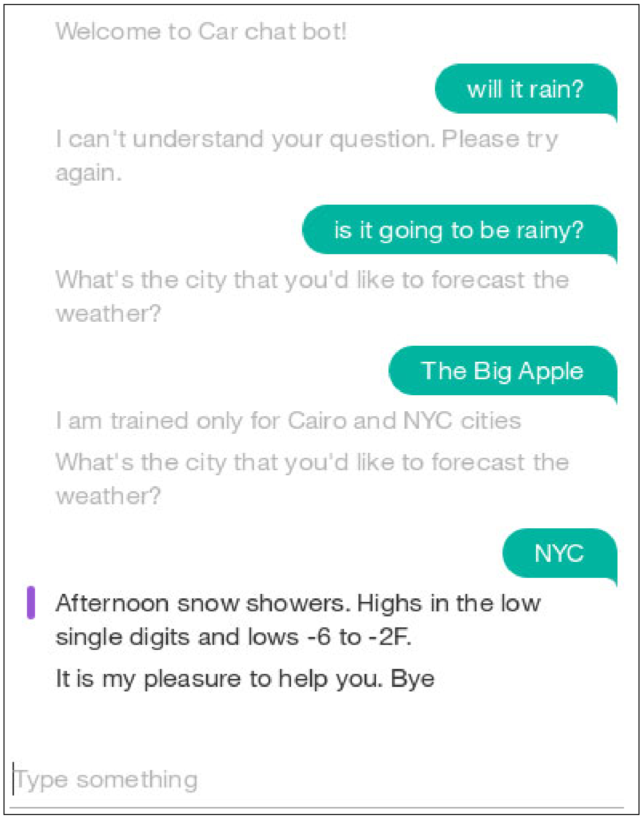
- Have basic knowledge of Watson Conversation service concepts: intents, entities and dialog. Review Chapter 1, “Basics of Conversation service” on page 1.
- Complete the use case by following the example in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157. In this chapter you will use the Conversation workspace and application created in Chapter 6.

7.1.3 Expected results

In this chapter, you modify the Car Chat-bot workspace to recognize these items:

- The `Big Apple` entity synonym for Manhattan, NYC
- The `Will it rain?` intent as a weather-related question

Before this modification, your workspace does not recognize this intent and the entity synonym (Figure 7-1 on page 187).



Welcome to Car chat bot!

will it rain?

I can't understand your question. Please try again.

is it going to be rainy?

What's the city that you'd like to forecast the weather?

The Big Apple

I am trained only for Cairo and NYC cities

What's the city that you'd like to forecast the weather?

NYC

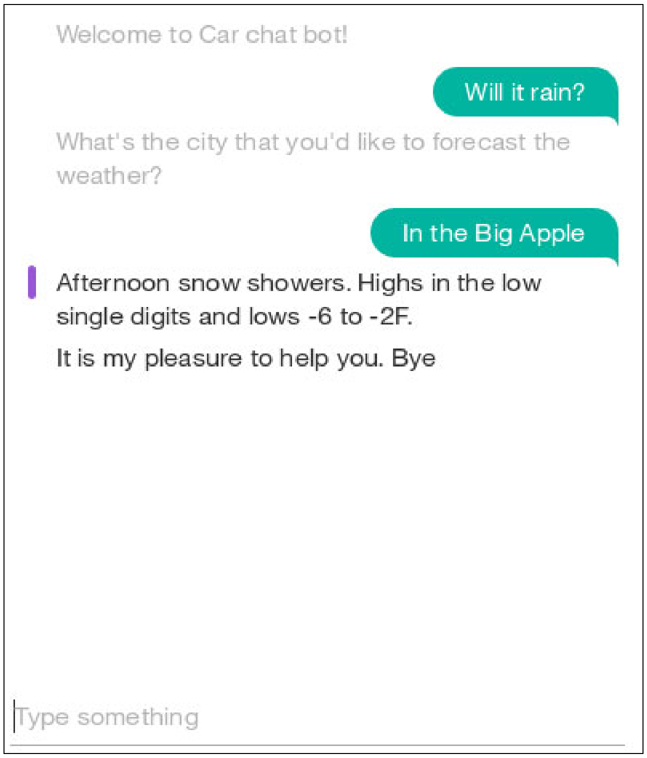
Afternoon snow showers. Highs in the low single digits and lows -6 to -2F.

It is my pleasure to help you. Bye

Type something

Figure 7-1 Before modification: The workspace does not understand some user's terms

After modification, the workspace can recognize both user inputs (Figure 7-2).



Welcome to Car chat bot!

Will it rain?

What's the city that you'd like to forecast the weather?

In the Big Apple

Afternoon snow showers. Highs in the low single digits and lows -6 to -2F.

It is my pleasure to help you. Bye

Type something

Figure 7-2 After modification: The workspace recognizes the user's terms

7.2 Use case implementation

Implementing this use case involves the following steps:

- Identifying the additional training that the Conversation workspace requires.
- Using the Improve component to train the Conversation workspace.
- Testing the improved Conversation workspace.

7.2.1 Identifying the additional training that the Conversation workspace requires

When the user tries to get the weather information by asking `Will it rain?` (as shown in Figure 7-3 on page 189), the workspace does not understand this question. Next, try again by changing your question to `Is it going to be rainy?` When the chatbot asks for the city, the user replies `The Big Apple` (another name for Manhattan). The workspace is not trained to recognize this entity.

Complete these steps:

1. In a web browser, open the application URL. If you followed the naming convention in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157, the URL is as follows, where `xxx` is a random number you selected to make the hostname unique:

   ```
   http://conv-201-xxx-weather.mybluemix.net/
   ```

2. Invoke the service by chatting with the application. In this example, you will input the following intents and entities to the application (Figure 7-3 on page 189):

   – `Will it rain?`

     The Conversation service does not understand this intent.

   – `Is it going to be rainy?`

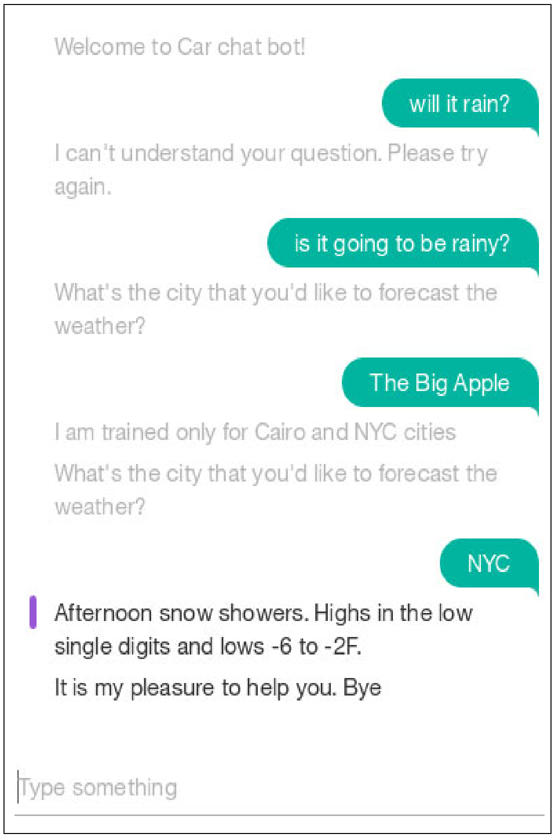
     The Conversation service understands this intent and asks which city you are interested in, to get your entity.

   – `The Big Apple`

     The Conversation service doe not understand this entity.

   – `NYC`

     After training, the Conversation service understands this entity and completes the flow with the `#weather_inquiry` intent and the `@NYC` entity.



Welcome to Car chat bot!

will it rain?

I can't understand your question. Please try again.

is it going to be rainy?

What's the city that you'd like to forecast the weather?

The Big Apple

I am trained only for Cairo and NYC cities

What's the city that you'd like to forecast the weather?

NYC

Afternoon snow showers. Highs in the low single digits and lows -6 to -2F.

It is my pleasure to help you. Bye

Figure 7-3 Trying out user interactions

7.2.2 Using the Improve component to train the Conversation workspace

The *Improve* component of the Conversation service provides a history of conversations with users. You can use this history to improve your chatbot's understanding of user inputs.

While you develop your workspace, you use the *Try it out* panel to verify that it recognizes the correct intents and entities in test inputs, and make corrections as needed. In the Improve panel, you can view actual conversations with your users and make similar corrections to improve the accuracy with which intents and entities are recognized.

In this example, you use the sample Car Chat-bot workspace to conduct a simple dialog with the user, and try to get information by communicating your intents and entities in unexpected ways.

Access the Improve component and open the chat logs

To access the Improve component and open the chat logs for the Car Chat-bot workspace:

1. Open the Car Chat-bot workspace.
2. Click the Menu icon ☰ (three horizontal lines). Then, select **Improve** → **User conversations** (Figure 7-4).

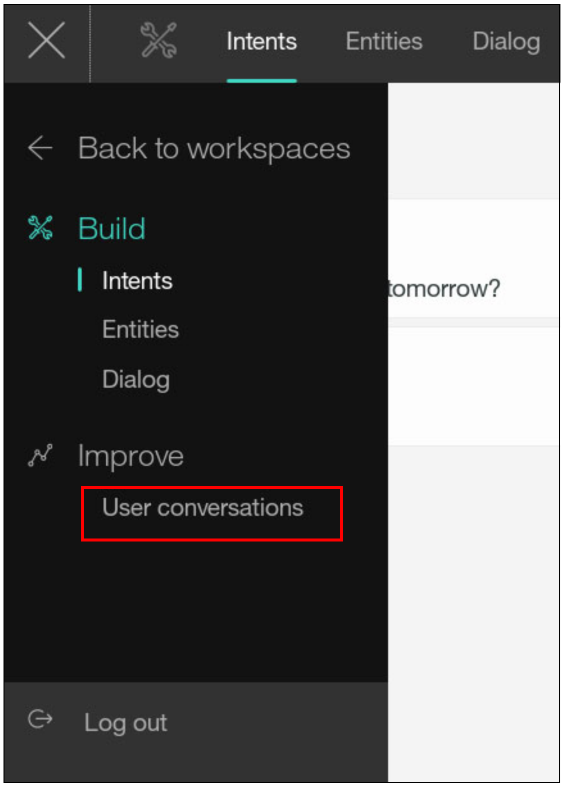
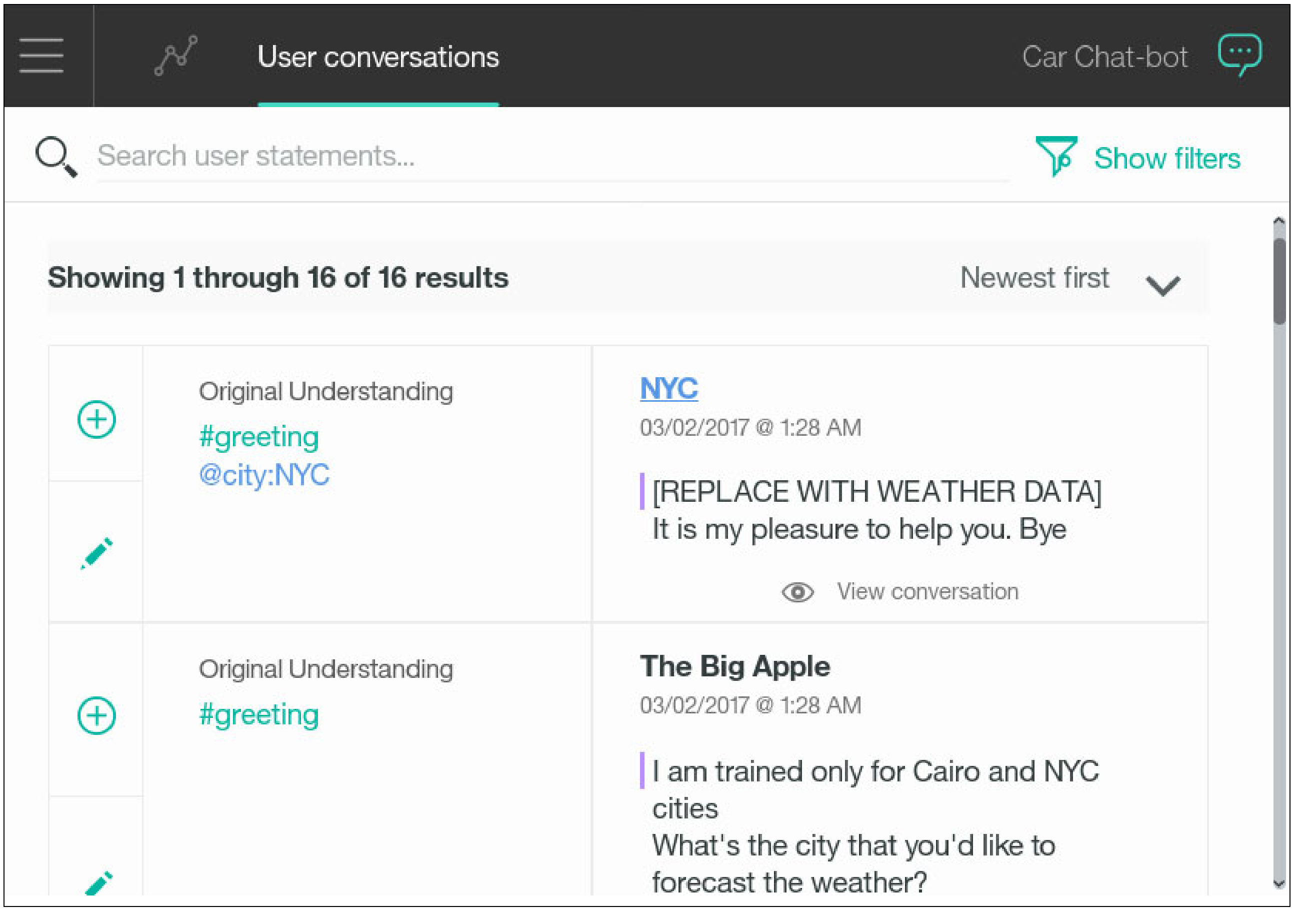


Figure 7-4 Improve component

The chat logs saved represent the user interactions through the API (*not* the interactions through the *Try it out* panel in the workspace). The Improve feature shows you the most recent user interactions. The top intent and any entities used in the message, the message text, and the chatbot's reply are available.

You see each user interaction, starting with the most recent (Figure 7-5).



Showing 1 through 16 of 16 results

Original Understanding
#greeting
@city:NYC

NYC
03/02/2017 @ 1:28 AM

[REPLACE WITH WEATHER DATA]
It is my pleasure to help you. Bye

Original Understanding
#greeting

The Big Apple
03/02/2017 @ 1:28 AM

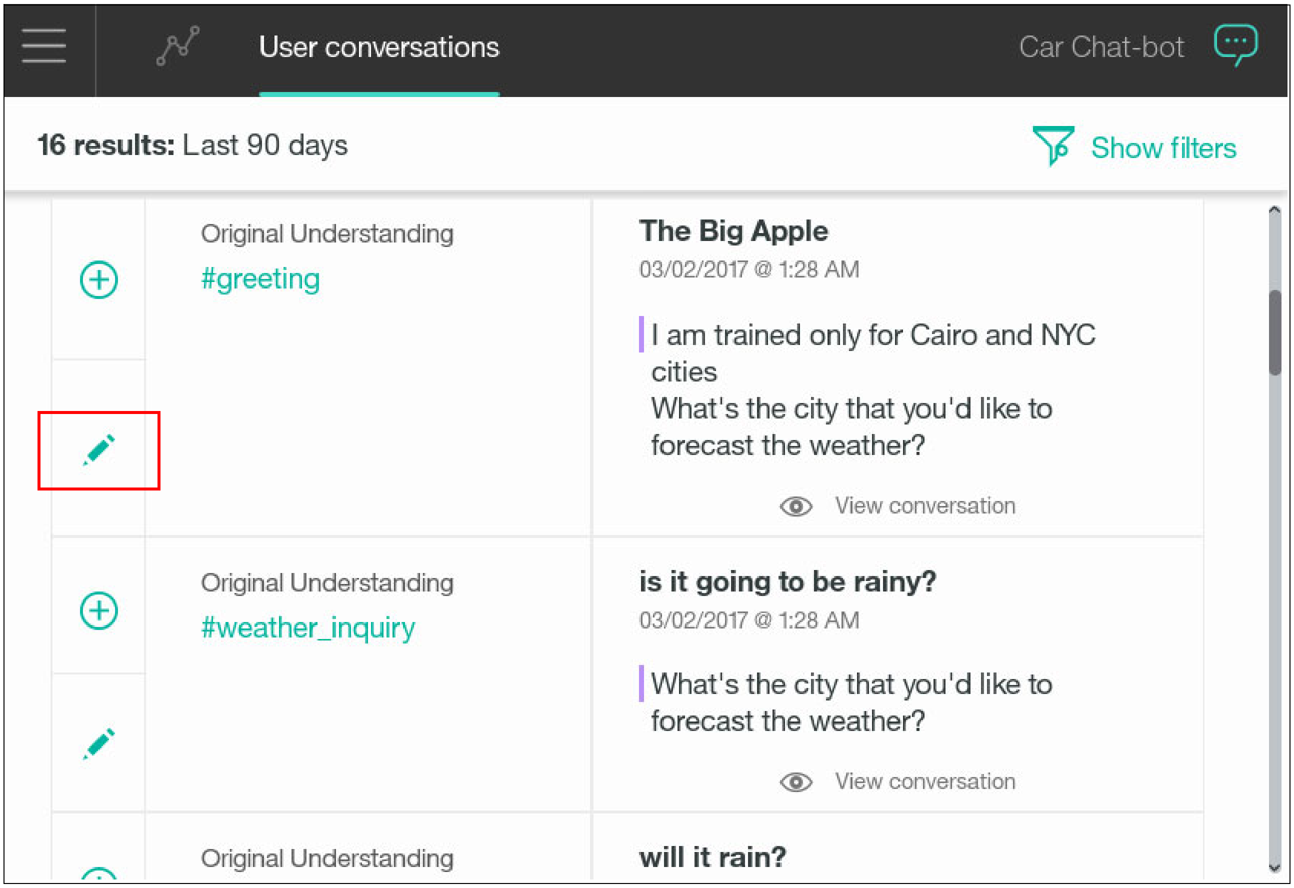
I am trained only for Cairo and NYC cities
What's the city that you'd like to forecast the weather?

Figure 7-5 User conversations history

Find the unrecognized entity synonym and train the workspace to recognize it

You will edit the input where you referred to Manhattan as `The Big Apple` (Figure 7-6). You will see that no entities are found, and the `#greeting` intent is identified. You correct both of these issues by first disassociating the phrase with the `#greeting` intent. Then, you train the workspace to recognize that NYC and Big Apple are synonyms. Complete these steps:

1. Click the **Edit** icon (pencil).



The Big Apple
03/02/2017 @ 1:28 AM

I am trained only for Cairo and NYC cities
What's the city that you'd like to forecast the weather?

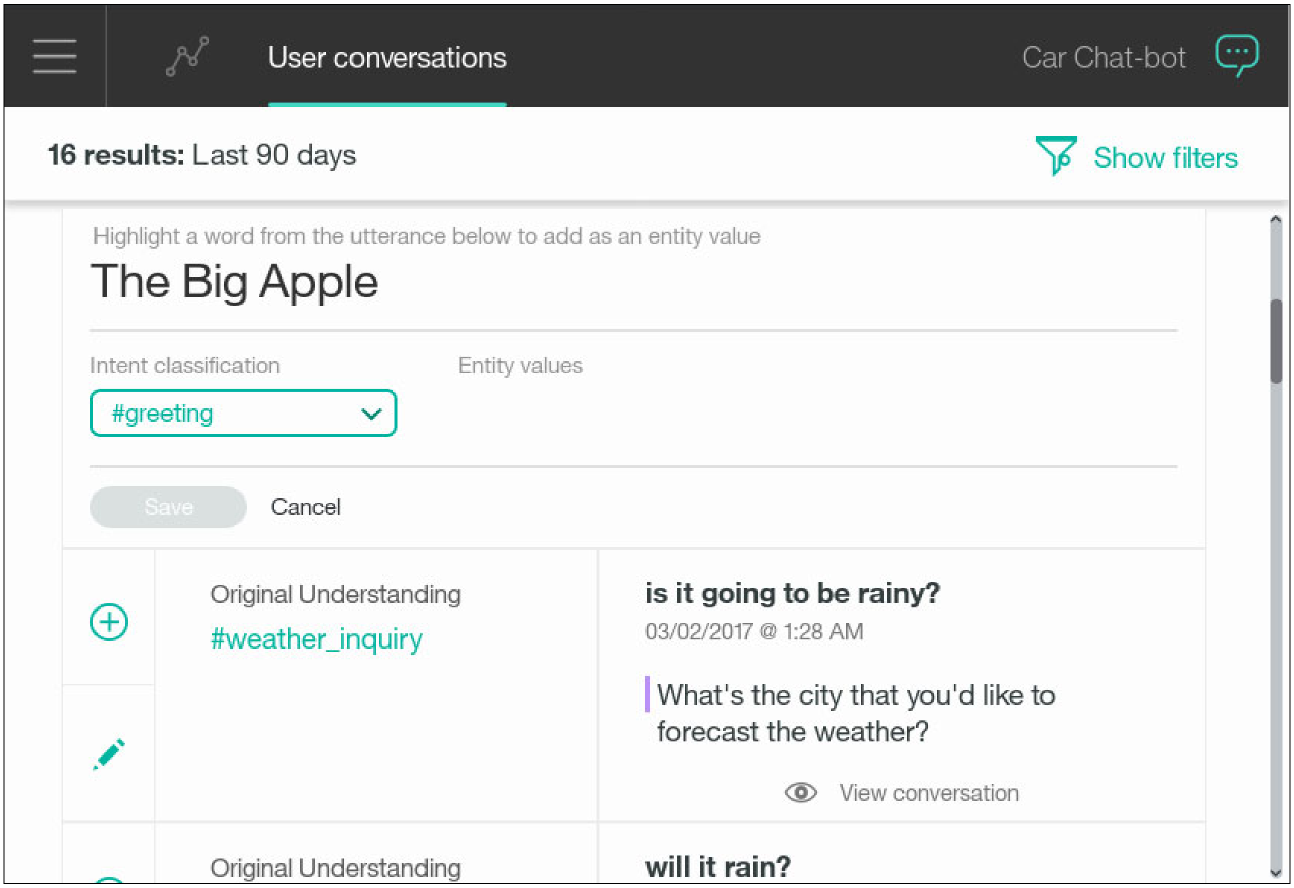
is it going to be rainy?
03/02/2017 @ 1:28 AM

What's the city that you'd like to forecast the weather?

will it rain?

Figure 7-6 Editing an interaction

The window now looks like the one in Figure 7-7.



16 results: Last 90 days

The Big Apple

Intent classification: #greeting

Original Understanding: #weather_inquiry

is it going to be rainy?
03/02/2017 @ 1:28 AM

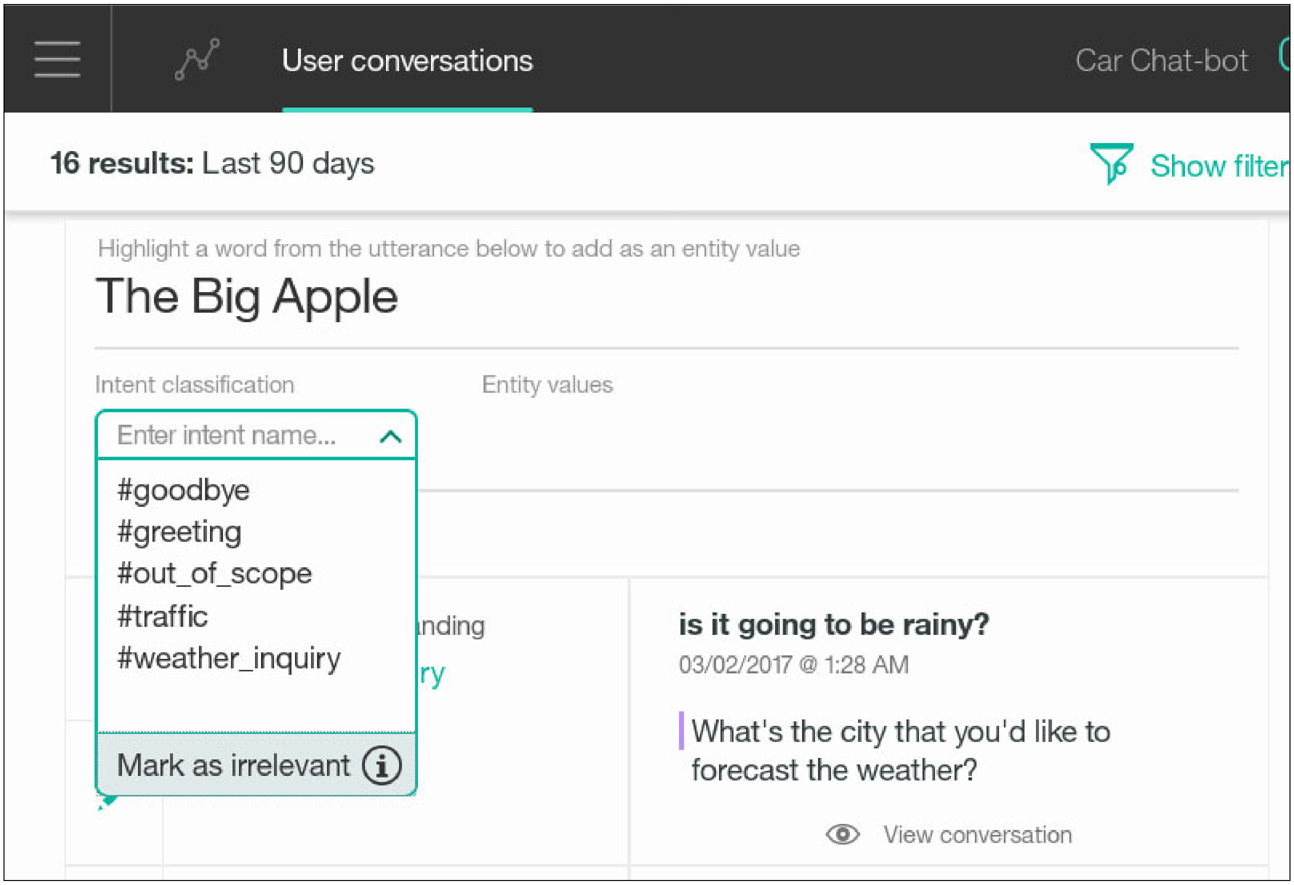
What's the city that you'd like to forecast the weather?

Original Understanding

will it rain?

Figure 7-7 Editing the user interaction

2. Select the intent from the drop-down menu, and replace `#greeting` with **Mark as irrelevant** (Figure 7-8). This ensures that next time `The Big Apple` will not be recognized as a greeting.



Highlight a word from the utterance below to add as an entity value

The Big Apple

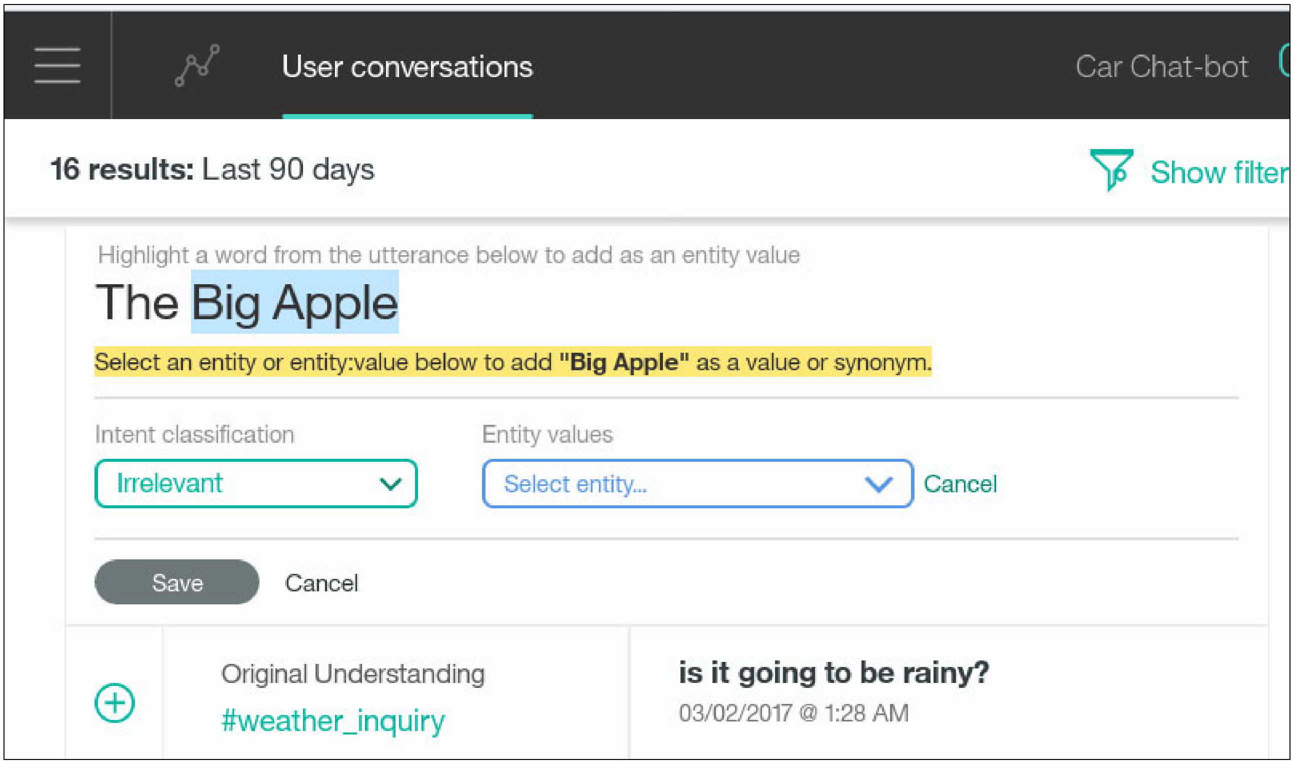
is it going to be rainy?

03/02/2017 @ 1:28 AM

What's the city that you'd like to forecast the weather?

Figure 7-8 Marking the phrase as not matching any intent.

3. Select the part of the phrase that is a synonym of your entity. In this case, use the mouse to highlight **Big Apple** (Figure 7-9). A pull-down menu opens under Entity values (where you will select the matching entity value).



16 results: Last 90 days

Highlight a word from the utterance below to add as an entity value

The Big Apple

Select an entity or entity:value below to add **"Big Apple"** as a value or synonym.

Intent classification: Irrelevant

Entity values: Select entity...

Original Understanding

#weather_inquiry

is it going to be rainy?

03/02/2017 @ 1:28 AM

Figure 7-9 Menu opens so you can select a matching entity value

4. Select the entity value that corresponds to `Big Apple`: **@city:NYC** (Figure 7-10). Then, click **Save**.

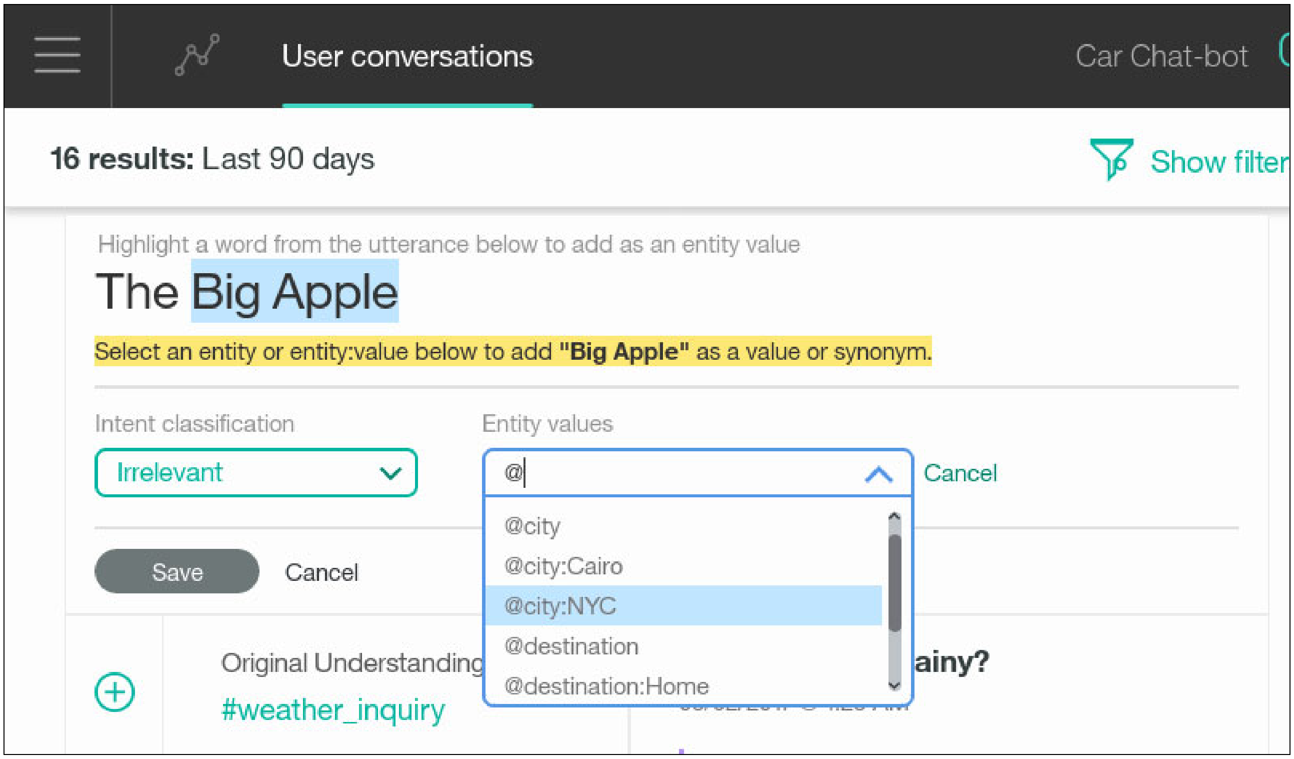
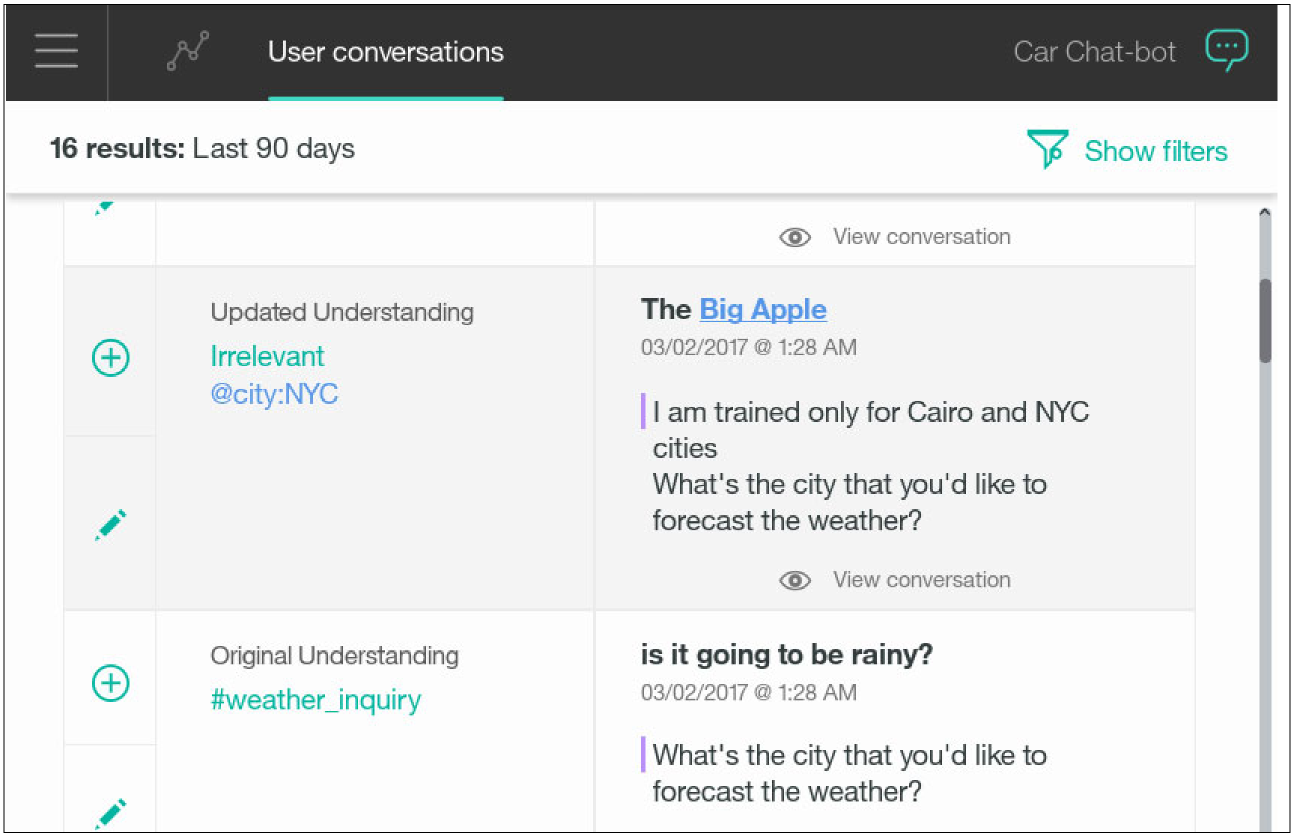


Figure 7-10 Selecting the corresponding entity and value

The result is shown in Figure 7-11.



16 results: Last 90 days

Updated Understanding
Irrelevant
@city:NYC

The Big Apple
03/02/2017 @ 1:28 AM

I am trained only for Cairo and NYC cities
What's the city that you'd like to forecast the weather?

Original Understanding
#weather_inquiry

is it going to be rainy?
03/02/2017 @ 1:28 AM

What's the city that you'd like to forecast the weather?

Figure 7-11 After saving your changes

A phrase that includes `Big Apple` can now be recognized as a synonym of the NYC value for the entity `@city`.

Find the unrecognized intent and train the workspace to recognize it

Complete the following steps:

1. Edit this interaction: `Will it rain?` (Figure 7-12).

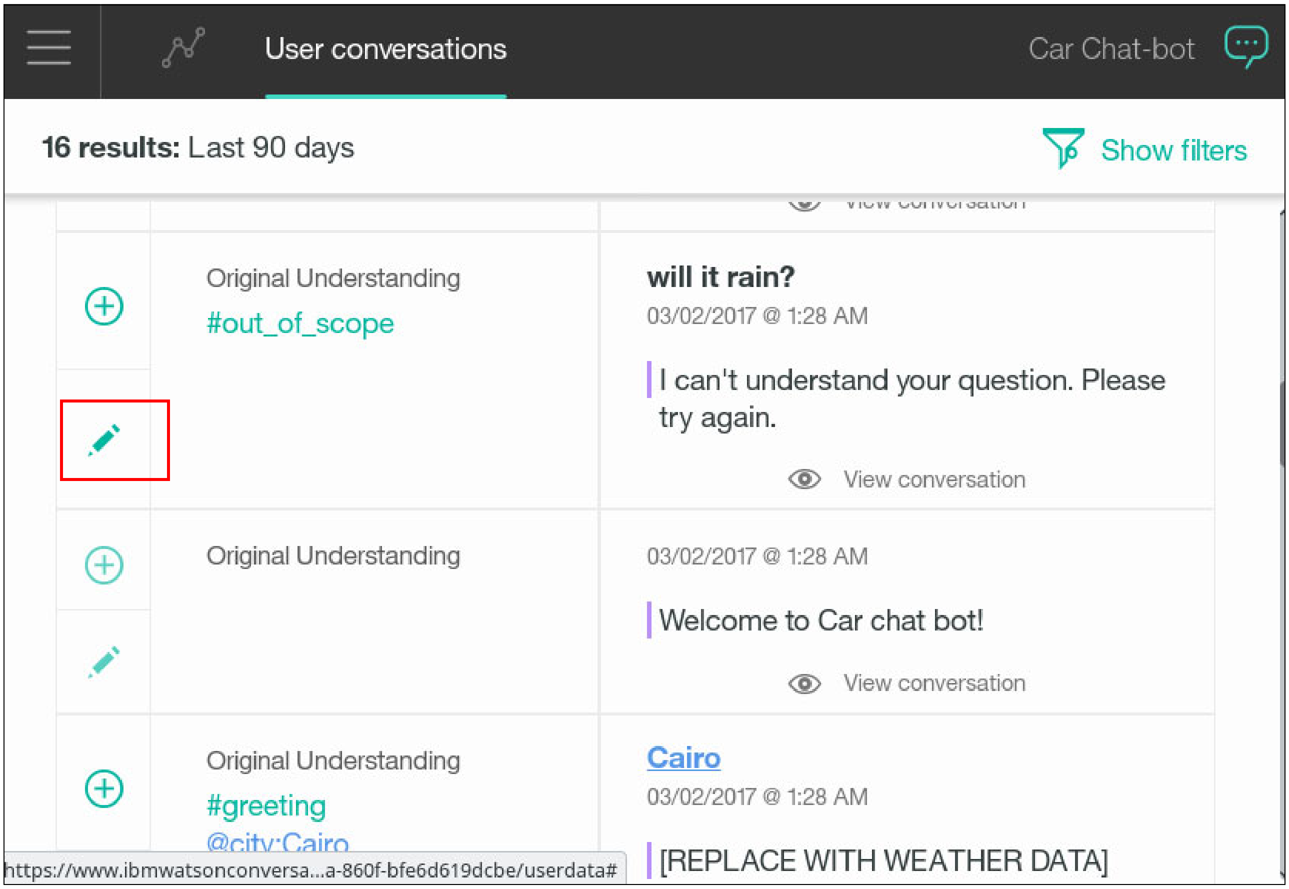
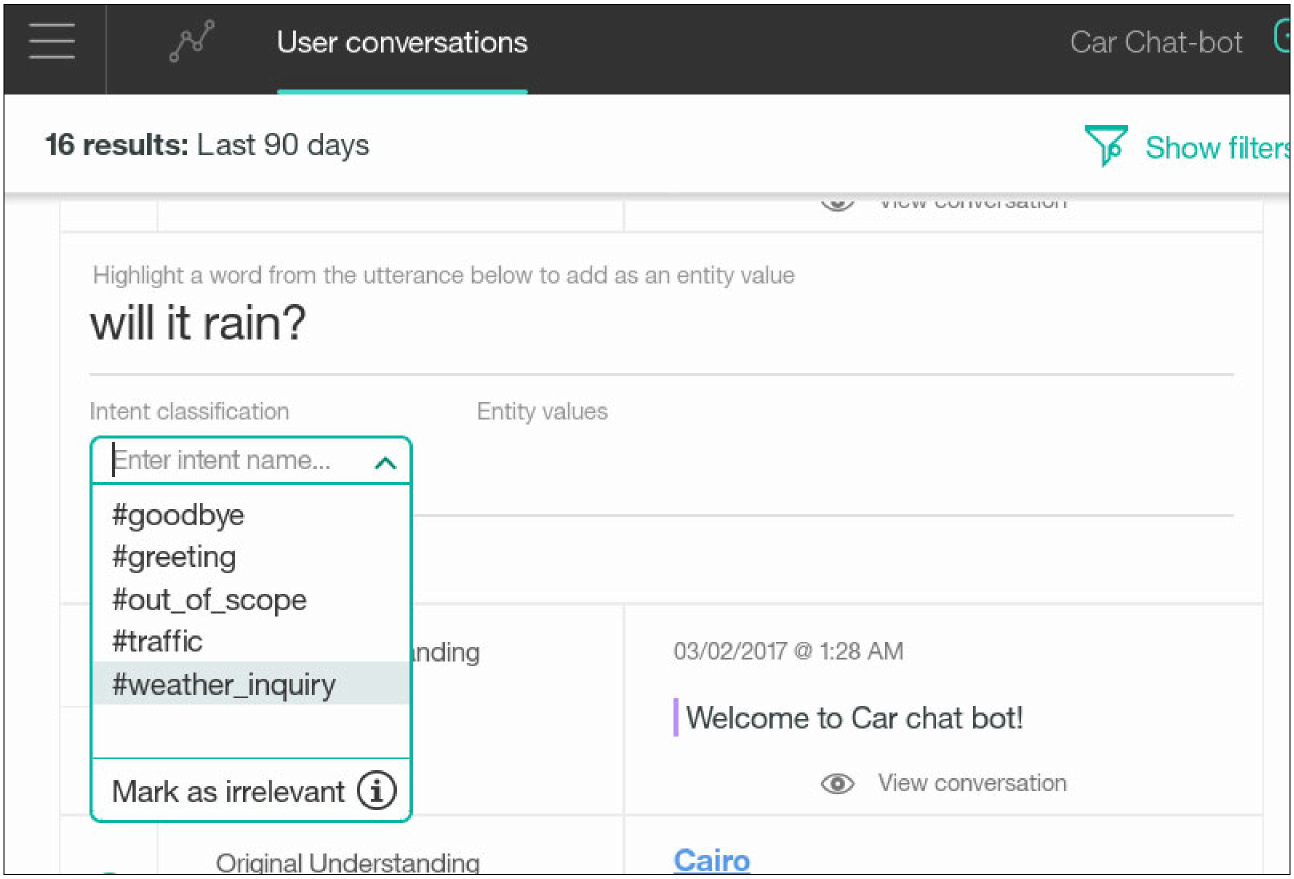


Figure 7-12 Editing intent interaction: will it rain?

2. Add an intent for this interaction. Select the correct `#weather_inquiry` intent (Figure 7-13).



16 results: Last 90 days

Highlight a word from the utterance below to add as an entity value

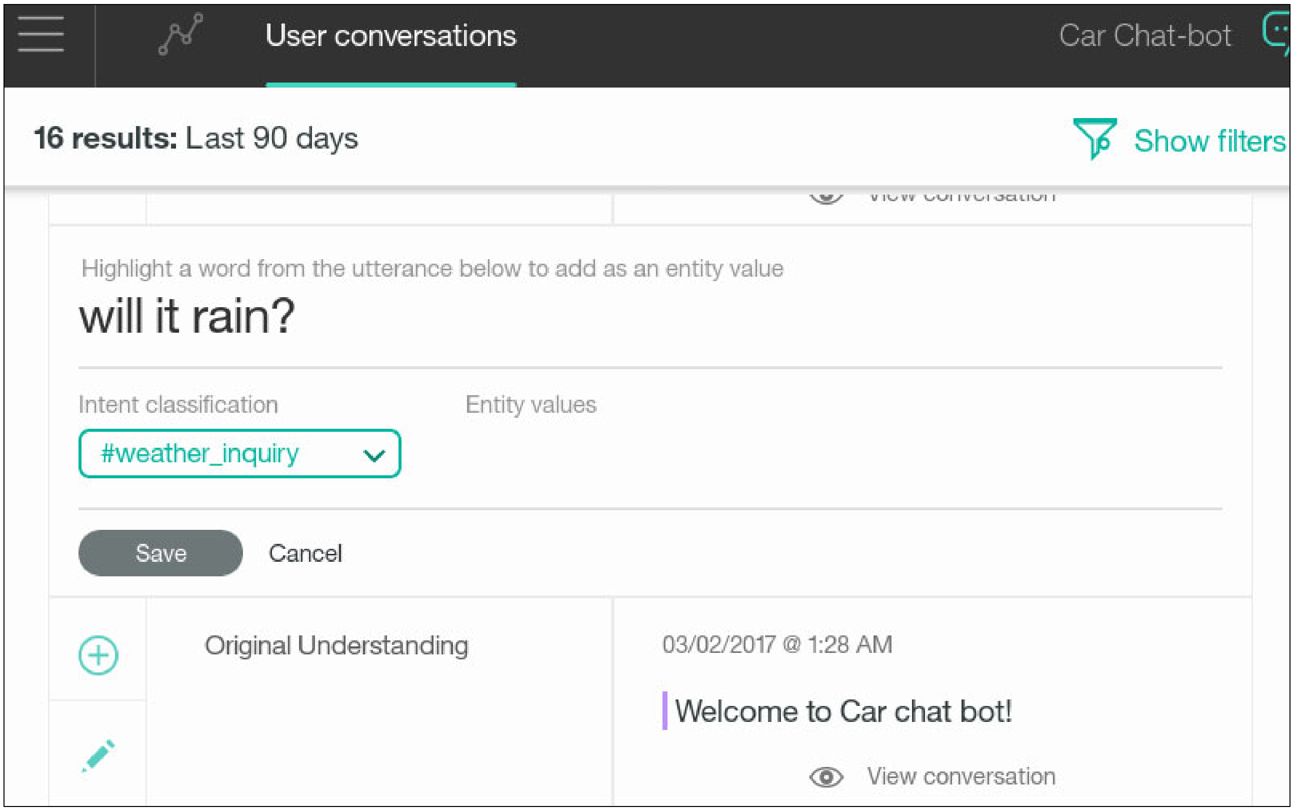
will it rain?

03/02/2017 @ 1:28 AM

Welcome to Car chat bot!

Figure 7-13 Selecting the correct intent

3. Click **Save** to save your intent changes (Figure 7-14).



16 results: Last 90 days

Highlight a word from the utterance below to add as an entity value

will it rain?

#weather_inquiry

Original Understanding

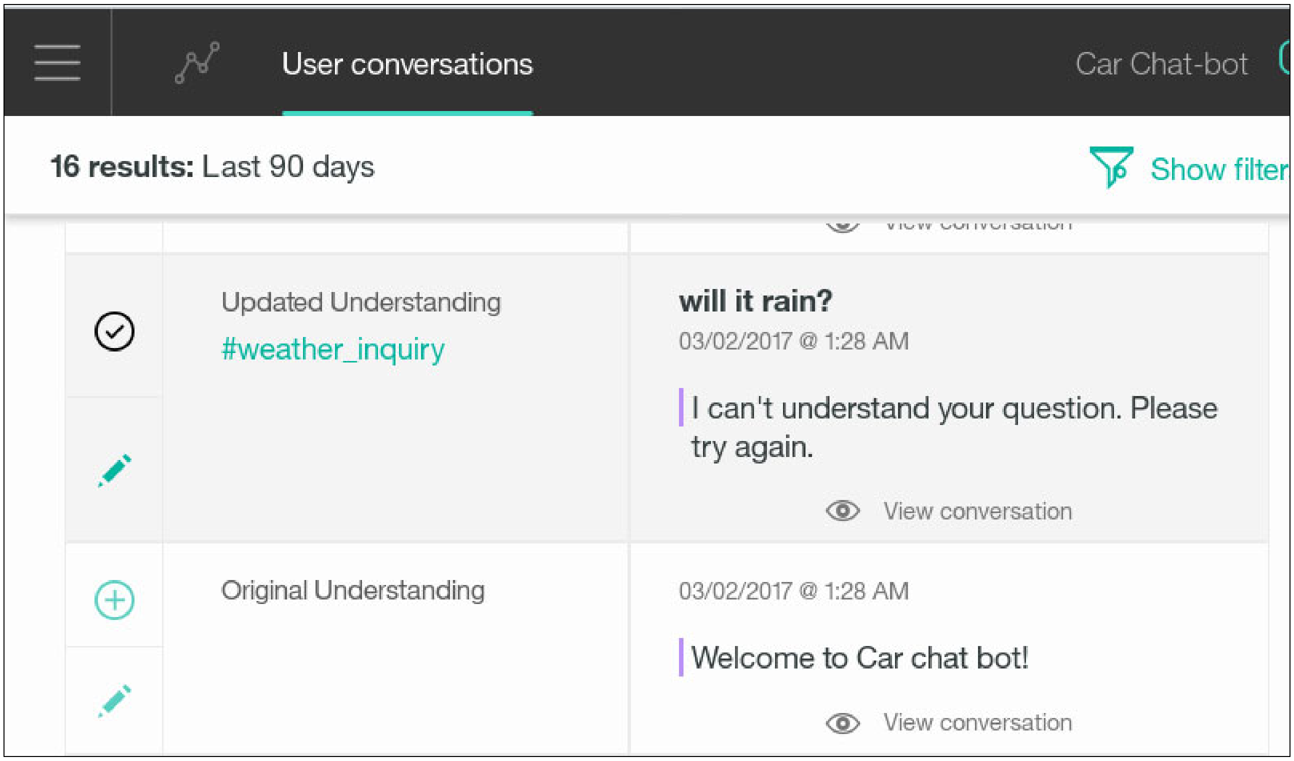
03/02/2017 @ 1:28 AM

Welcome to Car chat bot!

Figure 7-14 Saving intent changes

The interaction (`will it rain?`) is added as another example for the `#weather_inquiry` intent.

The result is shown in Figure 7-15 on page 201.



16 results: Last 90 days

Updated Understanding #weather_inquiry

will it rain?
03/02/2017 @ 1:28 AM
I can't understand your question. Please try again.

Original Understanding

03/02/2017 @ 1:28 AM
Welcome to Car chat bot!

Figure 7-15 After saving intent changes

7.2.3 Testing the improved Conversation workspace

To test the improved Car Chat-bot workspace, complete these steps:

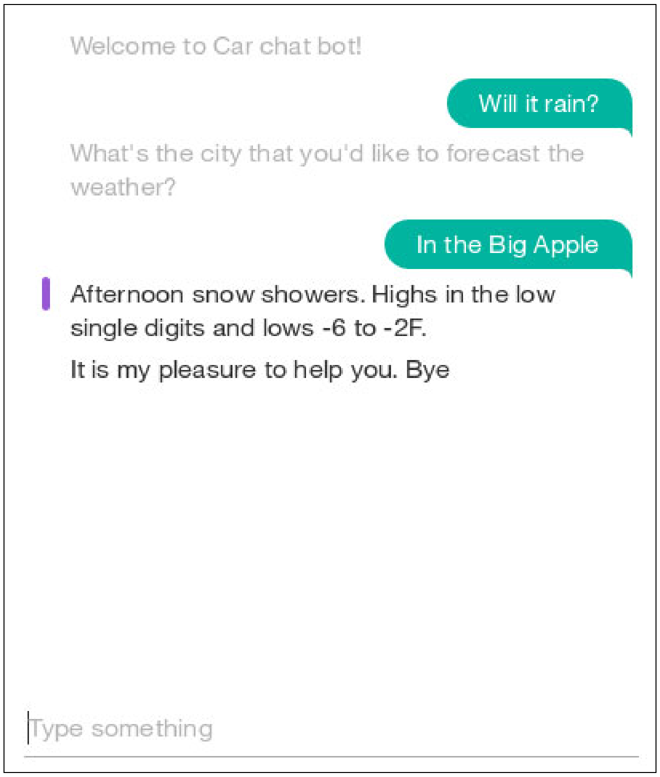
1. Open the application URL again in order to test the newly trained intents and entities. If you followed the naming convention in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157, the URL is as follows, where `xxx` is a random number you selected to make the hostname unique:

   `http://conv-201-xxx-weather.mybluemix.net/`

2. Inquire about the weather forecast by using the following lines:

   – Will it rain?
   – The Big Apple

You can see that it works correctly now (Figure 7-16).



Welcome to Car chat bot!

Will it rain?

What's the city that you'd like to forecast the weather?

In the Big Apple

Afternoon snow showers. Highs in the low single digits and lows -6 to -2F.

It is my pleasure to help you. Bye

Figure 7-16 The application now recognizes intent and entity

7.3 References

For more information, see the following resource:

- Improving understanding:

  https://www.ibm.com/watson/developercloud/doc/conversation/logs.html

Talking about the weather: Integrating Speech to Text and Text to Speech with the Conversation service

This chapter guides you through the process of updating the Cognitive Weather chatbot application created Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157 to integrate it with the Watson Speech to Text (STT) and Text to Speech (TTS) services.

The scenario in this chapter enables the user to send speech queries about weather forecast to the application by integrating with the Speech-to-Text service. The application responds to the user by integrating with the Text to Speech service.

The application demonstrates the use of Text to Speech, Speech to Text, Conversation and Weather Company Data services to get the forecasted weather for a city through *talking* with the user.

The following topics are covered in this chapter:

- Getting started
- Architecture
- Two ways to deploy the application: Step-by-step and quick deploy
- Step-by-step implementation
- Quick deployment of application
- References

8.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

8.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- Create Speech to Text (STT) and Text to Speech (TTS) services in Bluemix.
- Integrate a Conversation service with STT and TTS services in a Node.js application to provide weather information responding to spoken requests from the user.

8.1.2 Prerequisites

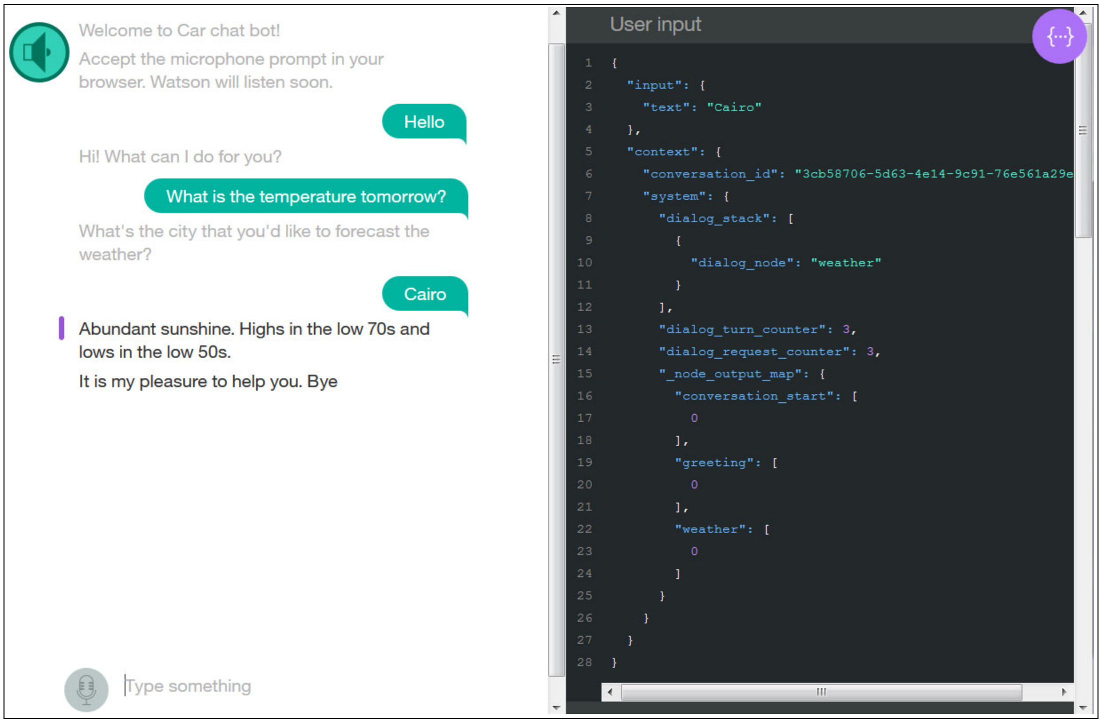
To complete the steps in this chapter, be sure these prerequisites are met:

- Finish the Cognitive Weather Forecast chatbot application implementation as described in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157.
- Use only Chrome or Firefox web browser; these browsers are required for Speech to Text and Text to Speech to work correctly.
- Understand basic JavaScript concepts.
- Have the Git command line installed on local workstation.
- Have the Cloud Foundry (CF) command line installed on the local workstation.
- Ensure that the microphone and speaker are working correctly on the local workstation.

In addition, if you see the word s*nippet* before example code, then use the example to complete the code.

8.1.3 Expected results

Figure 8-1 on page 205 shows the expected results of the running application. It illustrates how the user can talk to the application to request information about tomorrow’s temperature. In addition, it illustrates how the application responds in speech to specify the city to get the weather information about. Then, the user specifies the city as `Cairo`, and the application replies with the specific weather information for that city.



Welcome to Car chat bot!

Accept the microphone prompt in your browser. Watson will listen soon.

Hello

Hi! What can I do for you?

What is the temperature tomorrow?

What's the city that you'd like to forecast the weather?

Cairo

Abundant sunshine. Highs in the low 70s and lows in the low 50s.

It is my pleasure to help you. Bye

Type something

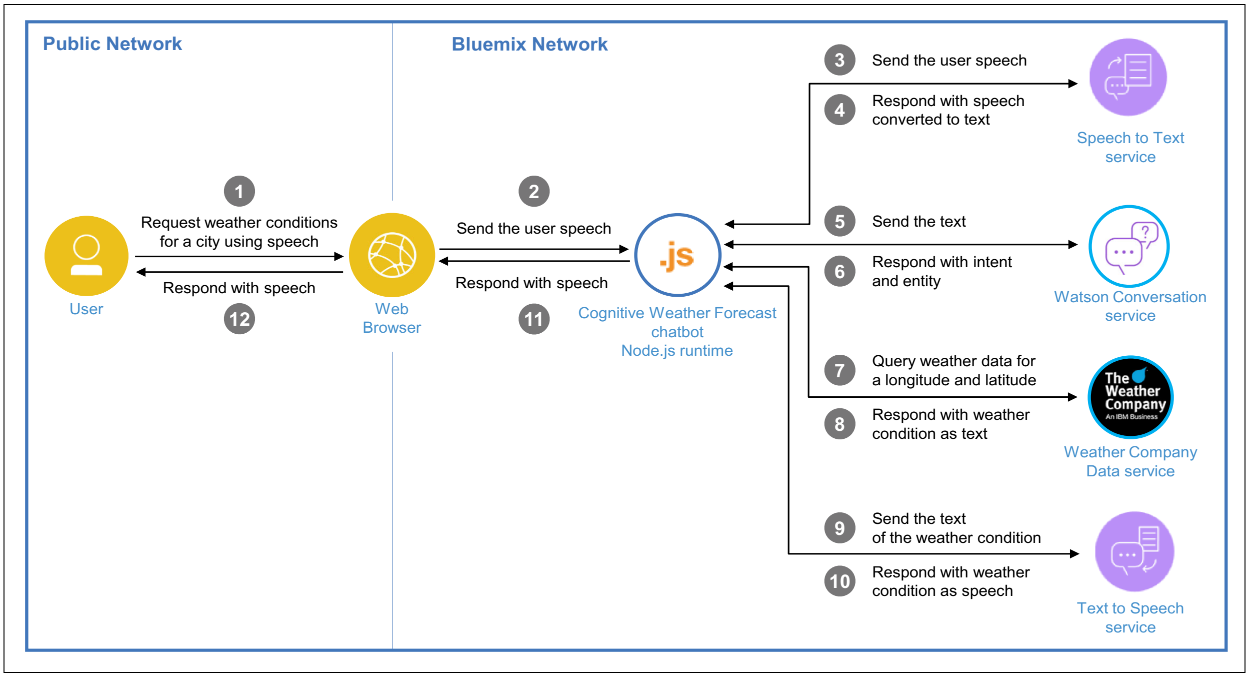
User input

```
{
  "input": {
    "text": "Cairo"
  },
  "context": {
    "conversation_id": "3cb58706-5d63-4e14-9c91-76e561a29e
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "weather"
        }
      ],
      "dialog_turn_counter": 3,
      "dialog_request_counter": 3,
      "_node_output_map": {
        "conversation_start": [
          0
        ],
        "greeting": [
          0
        ],
        "weather": [
          0
        ]
      }
    }
  }
}
```

Figure 8-1 Cognitive Weather Forecast Application

8.2 Architecture

Figure 8-2 shows the components involved in this use case and the runtime flow.



Public Network
Bluemix Network
User
1
Request weather conditions for a city using speech
Respond with speech
12
Web Browser
2
Send the user speech
Respond with speech
11
.js
Cognitive Weather Forecast chatbot Node.js runtime
3
Send the user speech
4
Respond with speech converted to text
Speech to Text service
5
Send the text
6
Respond with intent and entity
Watson Conversation service
7
Query weather data for a longitude and latitude
8
Respond with weather condition as text
The Weather Company An IBM Business
Weather Company Data service
9
Send the text of the weather condition
10
Respond with weather condition as speech
Text to Speech service

Figure 8-2 Architecture

The flow for this use case is as follows:

1. The user speaks to the application to ask for weather information for a city.
2. The request is passed from the web browser to the Node.js application on Bluemix.
3. The Node.js application passes the speech request to the Speech to Text service.
4. The Speech to Text service converts the speech request to text and sends it back to the Node.js application.
5. The Node.js application passes the text to the Conversation service.
6. The Conversation service understands the intent and entities passed by the application. Then it returns a response to the application based on the dialog configuration in the workspace of the Conversation service.
7. The Node.js application receives the response from the Conversation service and passes it to the Weather Company Data service to query the city weather.
8. The Weather Company Data service responds to the Node.js application with the weather information in text.
9. The Node.js application passes the response text to the Text to Speech service.
10. The Text to Speech service converts the text into audio and returns the audio to the Node.js application.
11. The Node.js application passes the audio to the web browser to play it to the user.
12. The user listens to the weather information for the city requested.

8.3 Two ways to deploy the application: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- Step-by-step deployment (incomplete) version of the application

  This repository contains an incomplete version of the application and is used in all sections of 8.4, “Step-by-step implementation” on page 207. This version takes you through the key steps to integrate the IBM Watson APIs with the application logic.

- Quick deployment (complete) version of the application

  This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 8.5, “Quick deployment of application” on page 219.

8.4 Step-by-step implementation

This section shows how to integrate the Cognitive Weather Forecast chatbot application (created in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157) with the Speech-to-Text and Text-to-Speech services.

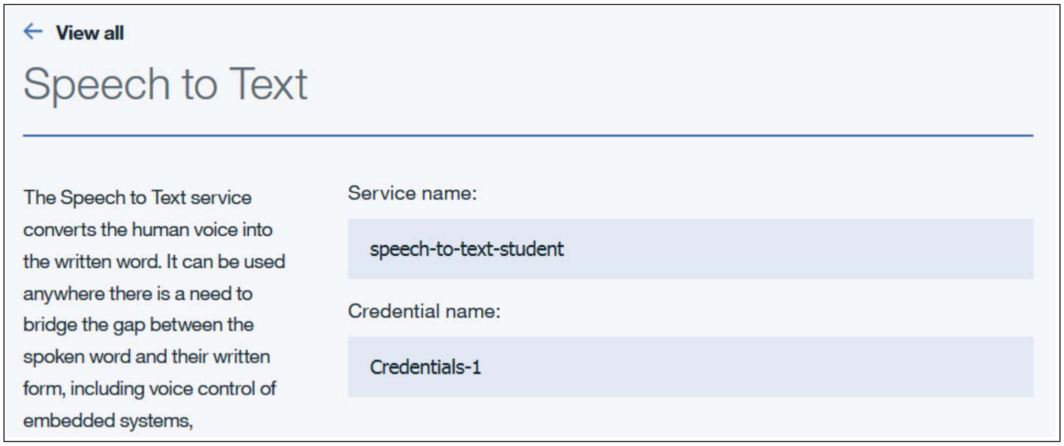
Implementing this use case involves the following steps:

1. Creating the Speech to Text service
2. Creating the Text to Speech service
3. Developing the Cognitive Weather Forecast chatbot application
4. Testing the application

8.4.1 Creating the Speech to Text service

To create the Speech to Text service, complete these steps:

1. In IBM Bluemix Catalog, scroll to Services select Watson, and then click **Speech to Text**.
2. In the Service name field, enter `speech-to-text-student` (Figure 8-3), then click **Create**.



Speech to Text

The Speech to Text service converts the human voice into the written word. It can be used anywhere there is a need to bridge the gap between the spoken word and their written form, including voice control of embedded systems,

Service name:

speech-to-text-student

Credential name:

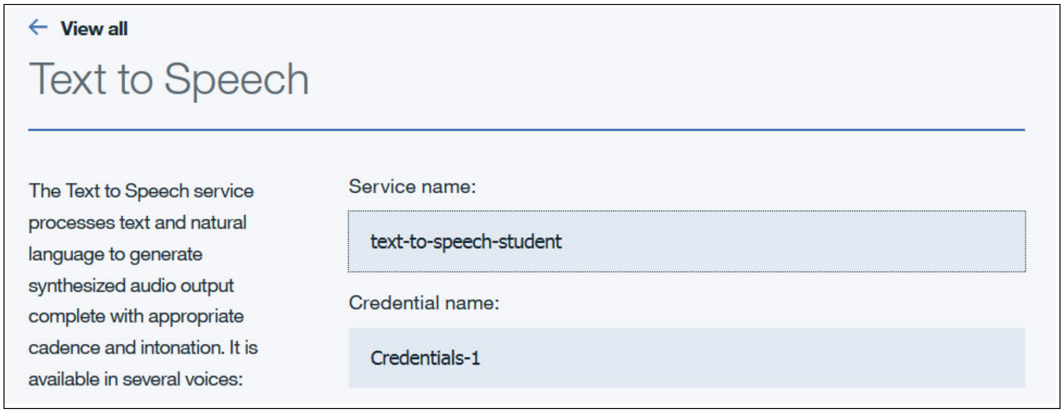
Credentials-1

Figure 8-3 Create STT service

8.4.2 Creating the Text to Speech service

To create the TTS service, follow these steps:

1. In IBM Bluemix Catalog, scroll to Services select Watson, and then click **Text to Speech**.
2. In the Service name field enter `text-to-speech-student` (Figure 8-4), then click **Create**.



View all

Text to Speech

The Text to Speech service processes text and natural language to generate synthesized audio output complete with appropriate cadence and intonation. It is available in several voices:

Service name: text-to-speech-student

Credential name: Credentials-1

Figure 8-4 Create TTS service

8.4.3 Developing the Cognitive Weather Forecast chatbot application

In this section, you modify the application to add integration with the Speech to Text and Text to Speech services.

Clone the application code from the Git repository to your local workstation

Clone the *incomplete* code for the Cognitive Weather Forecast application to your local workstation by using the Git command line. You will then add the integration code to STT and TTS services to it.

Use the following steps:

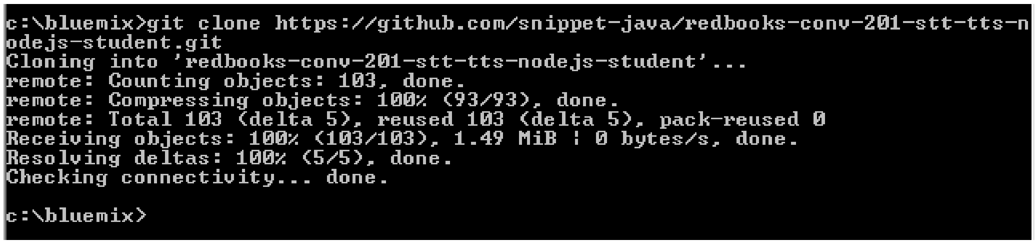
1. Create a new folder under the `C:\` directory and name it `Bluemix`.
2. Open a command prompt (`cmd.exe`), and change the working directory to the new folder that you created:

   ```
   cd C:\Bluemix
   ```

3. Type the following command to clone the incomplete repository to your local workstation:

   ```
   git clone https://github.com/snippet-java/redbooks-conv-201-stt-tts-nodejs-student.git
   ```

Figure 8-5 shows the command prompt result messages when cloning the code.



```
c:\bluemix>git clone https://github.com/snippet-java/redbooks-conv-201-stt-tts-n
odejs-student.git
Cloning into 'redbooks-conv-201-stt-tts-nodejs-student'...
remote: Counting objects: 103, done.
remote: Compressing objects: 100% (93/93), done.
remote: Total 103 (delta 5), reused 103 (delta 5), pack-reused 0
Receiving objects: 100% (103/103), 1.49 MiB | 0 bytes/s, done.
Resolving deltas: 100% (5/5), done.
Checking connectivity... done.

c:\bluemix>
```

Figure 8-5 Git clone result in command prompt

Complete the code

To modify the code so it is ready to be deployed, you update these files as follows:

1. Complete the `manifest.yml` file.
2. Complete the `app.js` file.
3. Complete the `index.html` file.

The sections that follow explain these steps in detail.

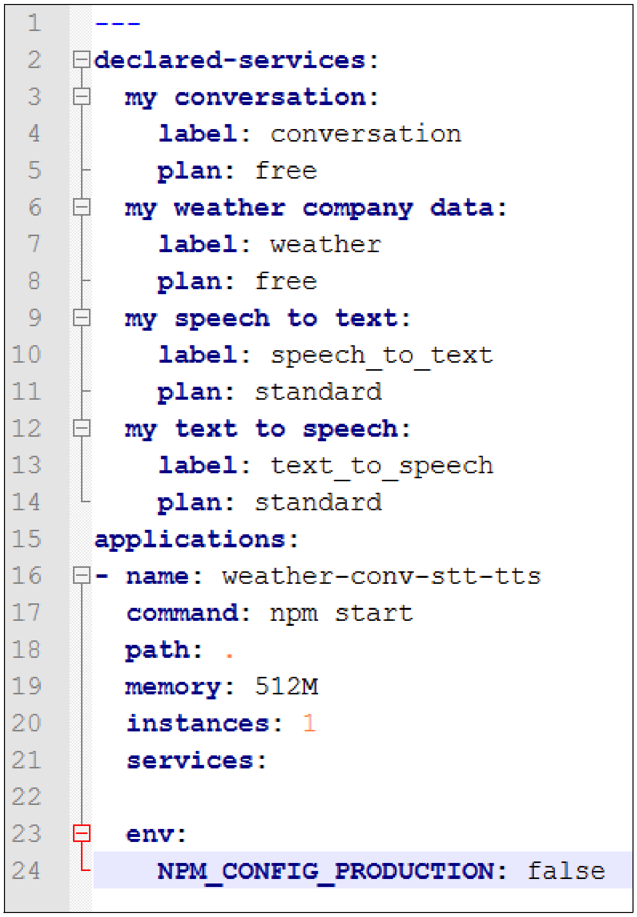
Complete the manifest.yml file

Completing the `manifest.yml` file involves renaming the application and renaming the services to match your Conversation, Weather Company Data, Speech to Text, and Text to Speech services instances in Bluemix:

1. Open the `manifest.yml` file in a text editor. The file is in the following path:

   `C:\BlueMix\conv-201-stt-tts-nodejs-student\manifest.yml`

   The file opens as shown in Figure 8-6.



```
1  ---
2  declared-services:
3    my conversation:
4      label: conversation
5      plan: free
6    my weather company data:
7      label: weather
8      plan: free
9    my speech to text:
10     label: speech_to_text
11     plan: standard
12   my text to speech:
13     label: text_to_speech
14     plan: standard
15 applications:
16 - name: weather-conv-stt-tts
17   command: npm start
18   path: .
19   memory: 512M
20   instances: 1
21   services:
22 
23   env:
24     NPM_CONFIG_PRODUCTION: false
```

Figure 8-6 The manifest.yml file before the update

2. Change the application name and names of the services in the file to match those on Bluemix:

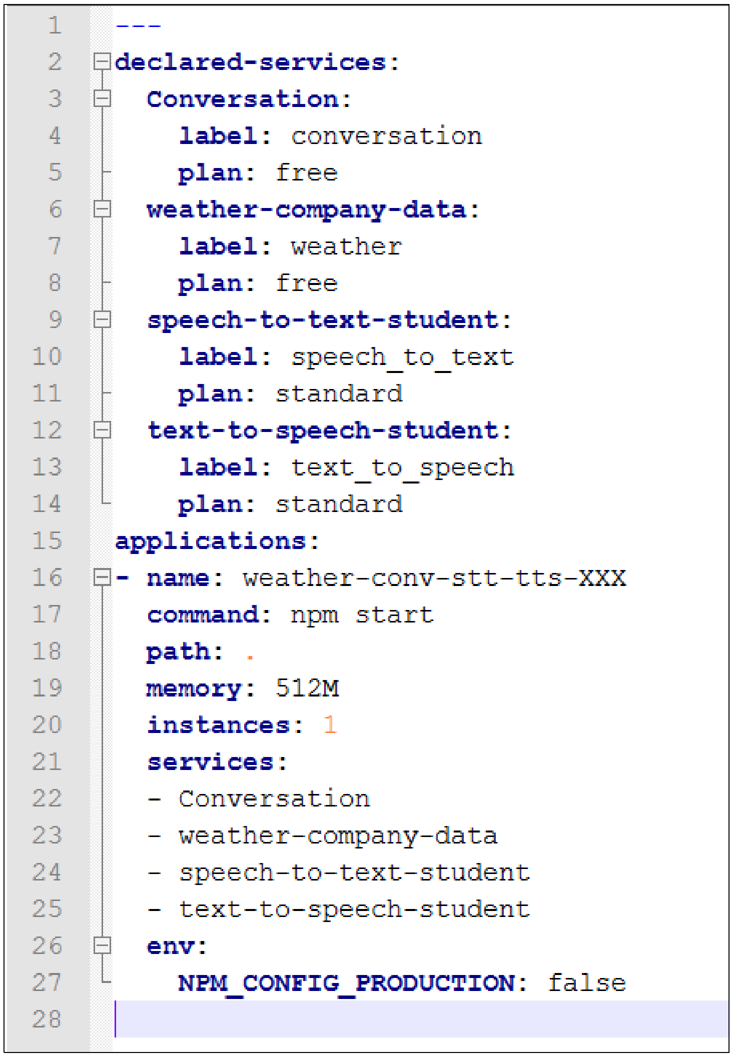
   – Line 3: Change `my conversation` to **`Conversation`**.
   – Line 6: Change `my weather company data` to **`weather-company-data`**.
   – Line 9: Change `my speech to text` to **`speech-to-text-student`**.
   – Line 12: Change `my text to speech` to **`text-to-speech-student`**.

– Line 22: Add the following lines:

  - `Conversation`
  - `weather-company-data`
  - `speech-to-text-student`
  - `text-to-speech-student`

– Line 16: Add a suffix to the application name to ensure uniqueness (for example, `weather-conv-stt-tts-`*`XXX`*, where *`XXX`* is your favorite word).

The completed `manifest.yml` file is shown in Figure 8-7.



```
 1  ---
 2  declared-services:
 3    Conversation:
 4      label: conversation
 5      plan: free
 6    weather-company-data:
 7      label: weather
 8      plan: free
 9    speech-to-text-student:
10      label: speech_to_text
11      plan: standard
12    text-to-speech-student:
13      label: text_to_speech
14      plan: standard
15  applications:
16  - name: weather-conv-stt-tts-XXX
17    command: npm start
18    path: .
19    memory: 512M
20    instances: 1
21    services:
22    - Conversation
23    - weather-company-data
24    - speech-to-text-student
25    - text-to-speech-student
26    env:
27      NPM_CONFIG_PRODUCTION: false
28
```

Figure 8-7 The manifest.yml file after the update

Complete the app.js file

Completing the `app.js` file involves adding the integration code to the Speech to Text and Text to Speech services:

1. Open the `app.js` file in a text editor. The file is in the following path:

   `C:\BlueMix\conv-201-stt-tts-nodejs-student\app.js`

   The `app.js` file contains the application logic and integrations.

2. Add the STT and TTS integration code to the file:

   a. Replace the "`// ADD SPEECH TO TEXT INTEGRATION CODE HERE`" comment with the code snippet (Example 8-1) to integrate the STT service with the application logic.

Example 8-1 Code snippet - STT integration code

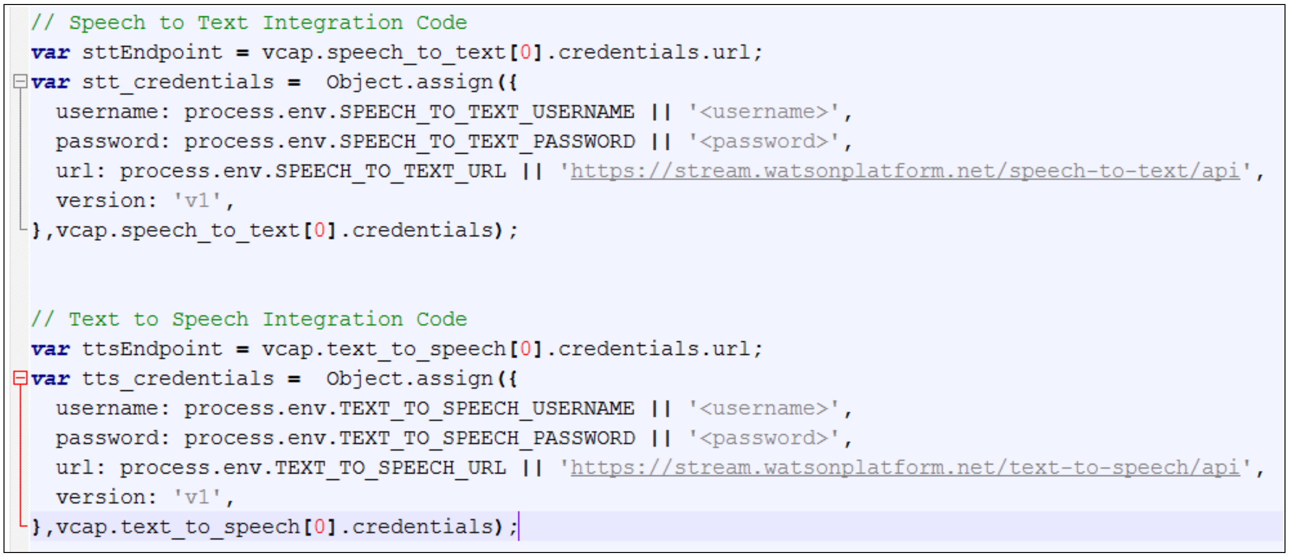
```
// Speech to Text Integration Code
var sttEndpoint = vcap.speech_to_text[0].credentials.url;
var stt_credentials =  Object.assign({
  username: process.env.SPEECH_TO_TEXT_USERNAME || '<username>',
  password: process.env.SPEECH_TO_TEXT_PASSWORD || '<password>',
  url: process.env.SPEECH_TO_TEXT_URL ||
'https://stream.watsonplatform.net/speech-to-text/api',
  version: 'v1',},vcap.speech_to_text[0].credentials);
```

   b. Replace the "`// ADD TEXT TO SPEECH INTEGRATION CODE HERE`" comment with the code snippet (Example 8-2) to integrate the TTS service with the application logic.

Example 8-2 Code snippet - TTS integration code

```
// Text to Speech Integration Code
var ttsEndpoint = vcap.text_to_speech[0].credentials.url;
var tts_credentials =  Object.assign({
  username: process.env.TEXT_TO_SPEECH_USERNAME || '<username>',
  password: process.env.TEXT_TO_SPEECH_PASSWORD || '<password>',
  url: process.env.TEXT_TO_SPEECH_URL ||
'https://stream.watsonplatform.net/text-to-speech/api',
  version: 'v1',
},vcap.text_to_speech[0].credentials);
```

Figure 8-8 shows the `app.js` file after adding the previous integration code.



```
// Speech to Text Integration Code
var sttEndpoint = vcap.speech_to_text[0].credentials.url;
var stt_credentials =  Object.assign({
  username: process.env.SPEECH_TO_TEXT_USERNAME || '<username>',
  password: process.env.SPEECH_TO_TEXT_PASSWORD || '<password>',
  url: process.env.SPEECH_TO_TEXT_URL || 'https://stream.watsonplatform.net/speech-to-text/api',
  version: 'v1',
},vcap.speech_to_text[0].credentials);


// Text to Speech Integration Code
var ttsEndpoint = vcap.text_to_speech[0].credentials.url;
var tts_credentials =  Object.assign({
  username: process.env.TEXT_TO_SPEECH_USERNAME || '<username>',
  password: process.env.TEXT_TO_SPEECH_PASSWORD || '<password>',
  url: process.env.TEXT_TO_SPEECH_URL || 'https://stream.watsonplatform.net/text-to-speech/api',
  version: 'v1',
},vcap.text_to_speech[0].credentials);
```

Figure 8-8 The app.js file after adding integration code

c. Replace the "//ADD TEXT TO SPEECH GET TOKEN ENDPOINT HERE" comment with the code snippet (Example 8-3) to add the TTS get token endpoint. This endpoint is used to get the authorization token of the service that is needed to access the service's APIs.

Example 8-3   Code snippet - TTS get token endpoint

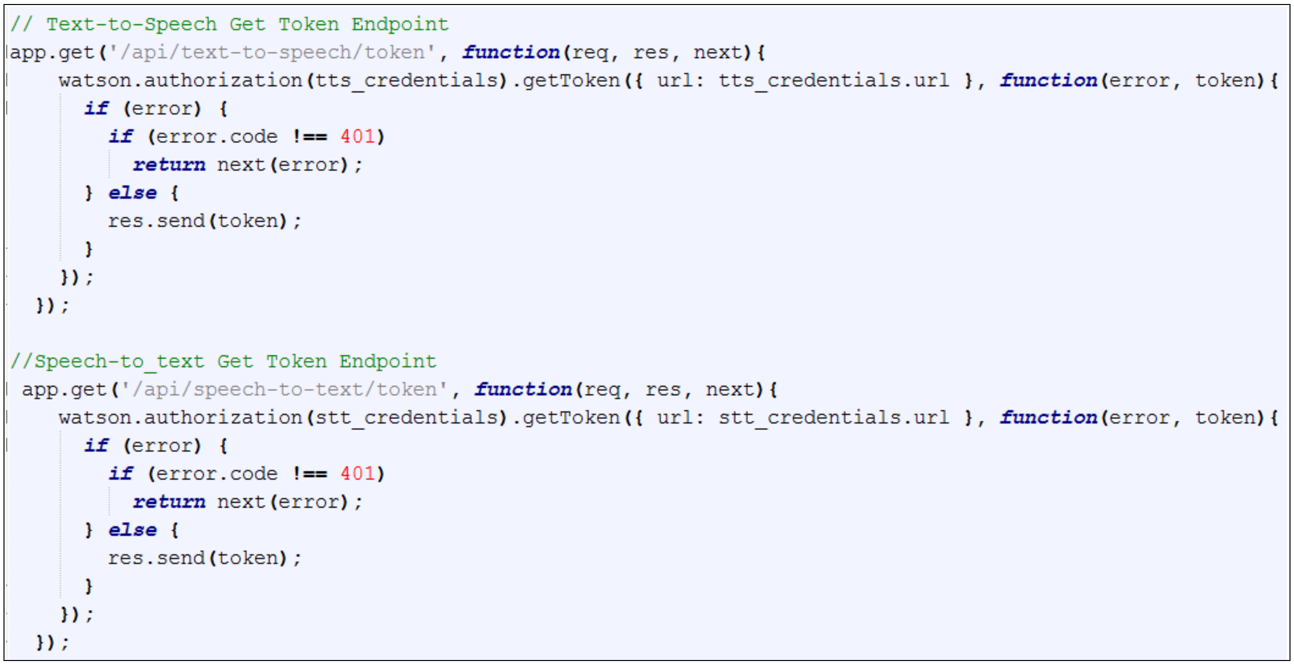
```
// Text-to-Speech Get Token Endpoint
app.get('/api/text-to-speech/token', function(req, res, next){
    watson.authorization(tts_credentials).getToken({ url:
tts_credentials.url }, function(error, token){
      if (error) {
        if (error.code !== 401)
          return next(error);
      } else {
        res.send(token);
      }
    });
});
```

d. Replace the "//ADD SPEECH TO TEXT GET TOKEN ENDPOINT HERE" comment with the code snippet (Example 8-4) to add the STT get token endpoint. This endpoint is used to get the authorization token of the service that is needed in order to access the service's APIs.

Example 8-4   Code snippet - STT get token endpoint

```
//Speech-to_text Get Token Endpoint
app.get('/api/speech-to-text/token', function(req, res, next){
    watson.authorization(stt_credentials).getToken({ url:
stt_credentials.url }, function(error, token){
      if (error) {
        if (error.code !== 401)
          return next(error);
      } else {
        res.send(token);
      }
    });
});
```

Figure 8-9 shows the `app.js` file after adding the endpoint code for the STT and TTS.



```
// Text-to-Speech Get Token Endpoint
app.get('/api/text-to-speech/token', function(req, res, next){
    watson.authorization(tts_credentials).getToken({ url: tts_credentials.url }, function(error, token){
      if (error) {
        if (error.code !== 401)
          return next(error);
      } else {
        res.send(token);
      }
    });
  });

//Speech-to_text Get Token Endpoint
 app.get('/api/speech-to-text/token', function(req, res, next){
    watson.authorization(stt_credentials).getToken({ url: stt_credentials.url }, function(error, token){
      if (error) {
        if (error.code !== 401)
          return next(error);
      } else {
        res.send(token);
      }
    });
  });
```

Figure 8-9 The app.js file after adding the endpoints

Complete the index.html file

Completing the `index.html` file involves adding the user interface changes needed in order to integrate the STT and TTS features:

1. Open the `index.html` file in a text editor. The file is in the following path:

   `C:\BlueMix\conv-201-stt-tts-nodejs-student\public\index.html`

   The `index.html` file contains the user interface of the application.

2. Add the STT and TTS features to the user interface:

   a. Replace the "`<!-- ADD AUDIO ELEMENT HERE -->`" comment with the code snippet (Example 8-5) to integrate the Audio Element to show the user the TTS feature.

Example 8-5 Code snippet - Integrate the Audio Element

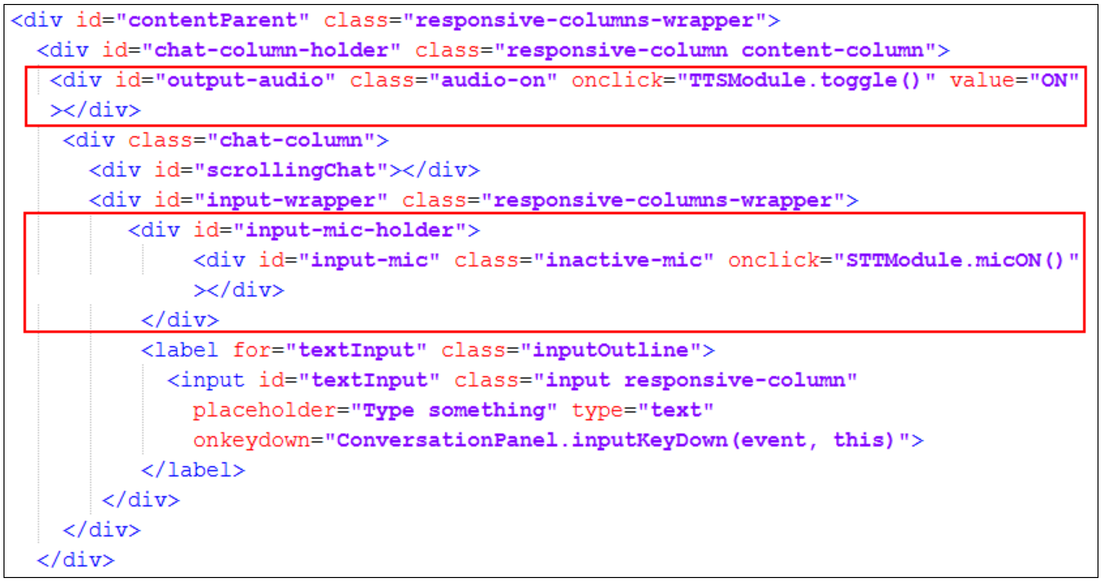
```
<div id="output-audio" class="audio-on" onclick="TTSModule.toggle()" value="ON"></div>
```

   b. Replace the "`<!-- ADD MIC ELEMENT HERE -->`" comment with the code snippet (Example 8-6) to integrate the Microphone Element to show the STT feature.

Example 8-6 Code snippet - Integrate the Microphone Element

```
<div id="input-mic-holder">
<div id="input-mic" class="inactive-mic" onclick="STTModule.micON()">
</div>
</div>
```

Figure 8-10 shows the `index.html` file after adding user interface HTML elements for integrating the STT and TTS features.



```
<div id="contentParent" class="responsive-columns-wrapper">
  <div id="chat-column-holder" class="responsive-column content-column">
   <div id="output-audio" class="audio-on" onclick="TTSModule.toggle()" value="ON"
   ></div>
    <div class="chat-column">
      <div id="scrollingChat"></div>
      <div id="input-wrapper" class="responsive-columns-wrapper">
         <div id="input-mic-holder">
              <div id="input-mic" class="inactive-mic" onclick="STTModule.micON()"
              ></div>
          </div>
          <label for="textInput" class="inputOutline">
            <input id="textInput" class="input responsive-column"
              placeholder="Type something" type="text"
              onkeydown="ConversationPanel.inputKeyDown(event, this)">
          </label>
       </div>
    </div>
  </div>
```

Figure 8-10 The completed index.html file

Deploy the application to Bluemix

After completing the code as described in the previous section, deploy the application to Bluemix, using the CF command line, by completing the following steps:

1. Log in to the Bluemix region, organization and space.
2. Push the application.
3. Set the WORKSPACE_ID environment variable.
4. Restage the application.

The sections that follow explain these steps in detail.

Log in to the Bluemix region, organization and space

To log in to the Bluemix organization and space:

1. At the command prompt (`cmd.exe`), change from the working directory to the directory that contains the application code:

   ```
   cd C:\Bluemix\conv-201-stt-tts-nodejs-student
   ```

2. Type the following command to log in to the Bluemix region:

   ```
   cf api https://api.ng.bluemix.net
   ```

   In this example, you log in to the US South Region.

3. Connect to your organization and space by using the following command:

   ```
   cf login -u <USERNAME> -o <ORG_NAME> -s <SPACE_NAME>
   ```

   The command has the following values:

   – `<USERNAME>` is your Bluemix user name.
   – `<ORG_NAME>` is the organization name that you want to push the application to.
   – `<SPACE_NAME>` is the space name that you want to push the application to.

4. When prompted, enter your password.

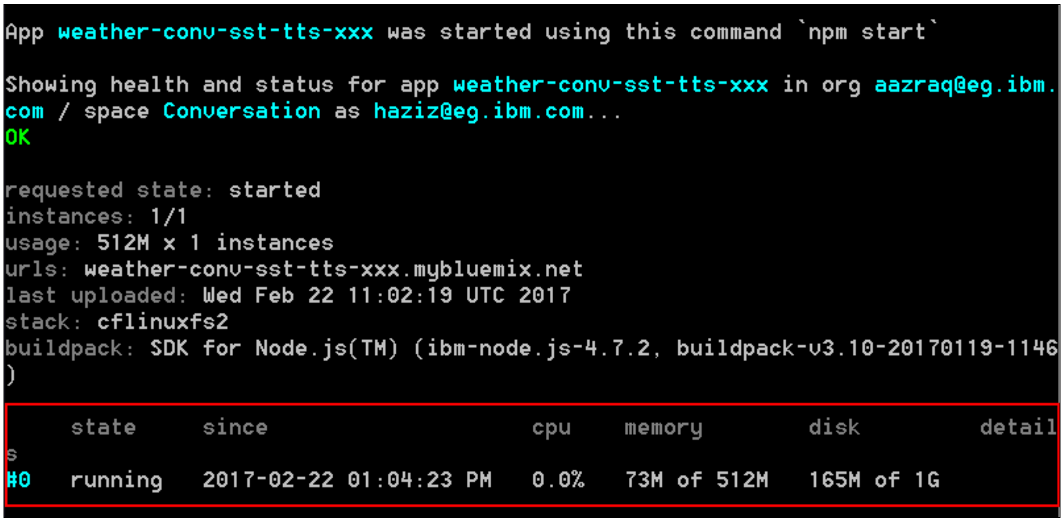
Push the application

To push the application:

1. Type the following command:

   ```
   cf push
   ```

2. Wait until the application deploys and a message indicating that the application is running is logged on the command line, as shown in Figure 8-11.



```
App weather-conv-sst-tts-xxx was started using this command `npm start`

Showing health and status for app weather-conv-sst-tts-xxx in org aazraq@eg.ibm.
com / space Conversation as haziz@eg.ibm.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: weather-conv-sst-tts-xxx.mybluemix.net
last uploaded: Wed Feb 22 11:02:19 UTC 2017
stack: cflinuxfs2
buildpack: SDK for Node.js(TM) (ibm-node.js-4.7.2, buildpack-v3.10-20170119-1146
)

     state     since                    cpu    memory        disk          detail
s
#0   running   2017-02-22 01:04:23 PM   0.0%   73M of 512M   165M of 1G
```

Figure 8-11 Successful application deployment message

Note: Deploying the application to Bluemix and starting it might take some time.

Set the WORKSPACE_ID environment variable

To set the WORKSPACE_ID environment variable:

1. Copy the Workspace ID of the Car Chat-bot workspace, as described in “Copy the Car Chat-bot workspace ID” on page 223.

2. To set the WORKSPACE_ID environment variable to the application to use the Car Chat-bot workspace in the Conversation service, use the following command:

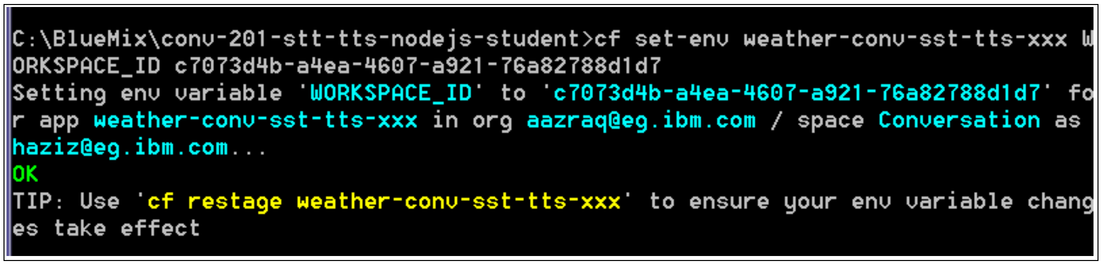
   ```
   cf set-env weather-conv-stt-tts-XXX WORKSPACE_ID $WORKSPACE_ID
   ```

   The command has the following values:

   – `XXX` is a suffix that you added to the application name to make the name unique.

   – `$WORKSPACE_ID` is the Car Chat-bot Workspace ID that you copy as describe in “Copy the Car Chat-bot workspace ID” on page 223.

Figure 8-12 illustrates how to set the environment variable using the command line.



```
C:\BlueMix\conv-201-stt-tts-nodejs-student>cf set-env weather-conv-sst-tts-xxx W
ORKSPACE_ID c7073d4b-a4ea-4607-a921-76a82788d1d7
Setting env variable 'WORKSPACE_ID' to 'c7073d4b-a4ea-4607-a921-76a82788d1d7' fo
r app weather-conv-sst-tts-xxx in org aazraq@eg.ibm.com / space Conversation as
haziz@eg.ibm.com...
OK
TIP: Use 'cf restage weather-conv-sst-tts-xxx' to ensure your env variable chang
es take effect
```

Figure 8-12 Set the environment variable using CF command line

Restage the application

For the setting of the environment to take effect, restage the application by using this command:

```
cf restage weather-conv-stt-tts-XXX
```

Wait for the application to restage and for the message indicating that the application is running in the log. After you deploy the application, proceed to the next section for information about how to use the application and test it.

8.4.4 Testing the application

After deploying the application, using either the full version (from 8.5, “Quick deployment of application” on page 219) or the incomplete code (which you just completed in 8.4.1, “Creating the Speech to Text service” on page 207), you must run the application and test it.

> **Speaker and microphone:** Make sure that the speaker and microphone are turned on for the workstation.
>
> **Support:** Only Chrome and Firefox are supported for testing the application.

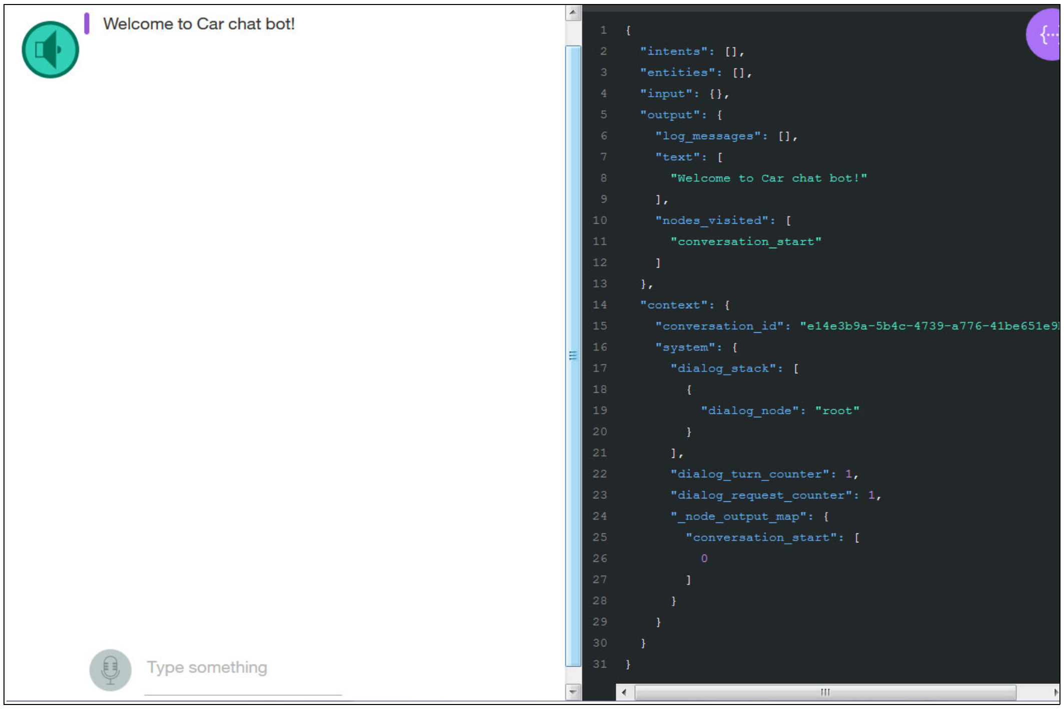
The following steps describe how the application works:

1. Open the application’s URL in your web browser:

   ```
   https://weather-conv-sst-tts-XXX.mybluemix.net/
   ```

   The application opens (Figure 8-13); the audio greeting starts by saying:

   ```
   Welcome to Car chat bot!
   ```

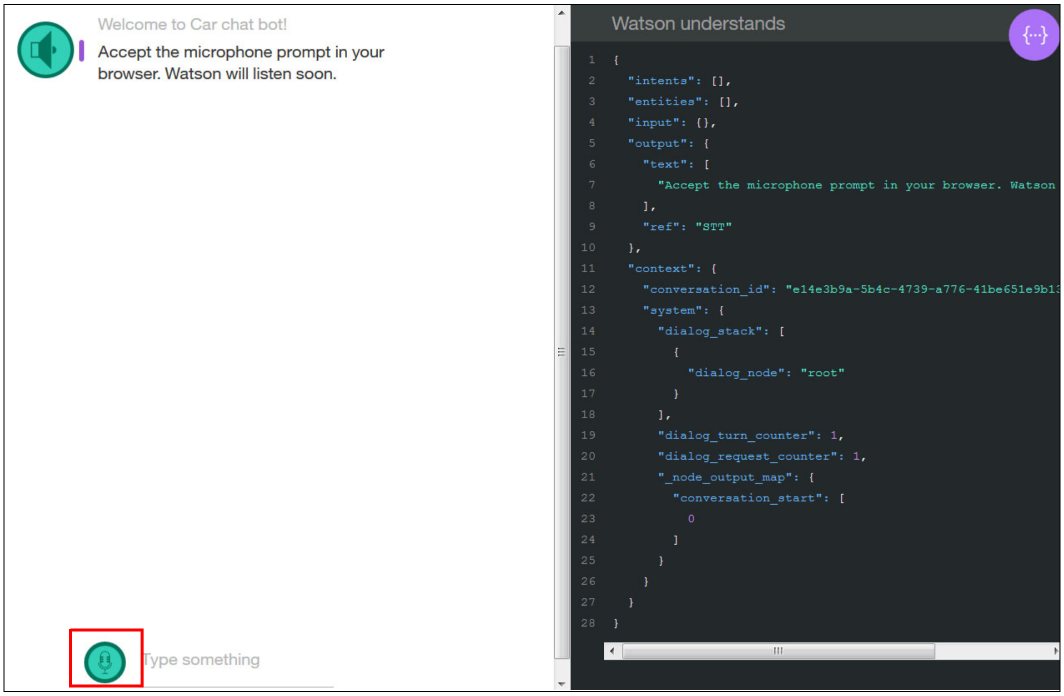


Welcome to Car chat bot!

```
{
  "intents": [],
  "entities": [],
  "input": {},
  "output": {
    "log_messages": [],
    "text": [
      "Welcome to Car chat bot!"
    ],
    "nodes_visited": [
      "conversation_start"
    ]
  },
  "context": {
    "conversation_id": "e14e3b9a-5b4c-4739-a776-41be651e9
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1,
      "_node_output_map": {
        "conversation_start": [
          0
        ]
      }
    }
  }
}
```

Figure 8-13 Cognitive Weather Forecast application opens

2. Click the microphone at the bottom of the page to enable the browser microphone so that you can talk to the application. As shown in Figure 8-14, a message displays to `Accept the microphone prompt in your browser. Watson will listen soon`. The audio greeting says, `Welcome to Car Chat bot!`



Welcome to Car chat bot!

Accept the microphone prompt in your browser. Watson will listen soon.

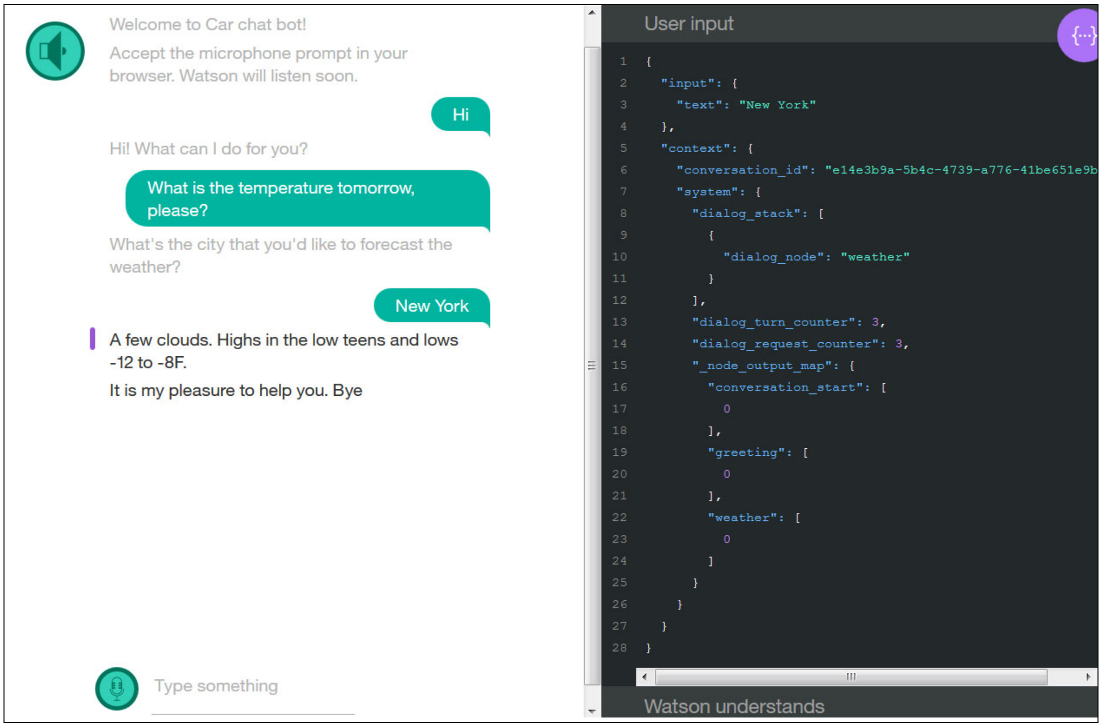
Watson understands

```
{
  "intents": [],
  "entities": [],
  "input": {},
  "output": {
    "text": [
      "Accept the microphone prompt in your browser. Watson
    ],
    "ref": "STT"
  },
  "context": {
    "conversation_id": "e14e3b9a-5b4c-4739-a776-41be651e9b13
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1,
      "_node_output_map": {
        "conversation_start": [
          0
        ]
      }
    }
  }
}
```

Figure 8-14 Enable the microphone on the application

3. Speak into the microphone. Try saying, “Hi.” The application responds in voice and text by saying, `Hi! What can I do for you?`
4. You can speak to the application by asking for the temperature. For example, ask `What is the temperature tomorrow, please?` The application prompts you with both voice and text: `What’s the city that you’d like to forecast the weather?`
5. Choose a city. For example, you can choose New York.
6. The application responds with the expected weather for tomorrow for that city. For example, the application responds with both voice and text: `A few clouds. Highs in the low teens and lows -12 to -8F.`

Figure 8-15 shows the complete exchange between the application and the user.



Welcome to Car chat bot!

Accept the microphone prompt in your browser. Watson will listen soon.

Hi

Hi! What can I do for you?

What is the temperature tomorrow, please?

What's the city that you'd like to forecast the weather?

New York

A few clouds. Highs in the low teens and lows -12 to -8F.

It is my pleasure to help you. Bye

```
{
  "input": {
    "text": "New York"
  },
  "context": {
    "conversation_id": "e14e3b9a-5b4c-4739-a776-41be651e9b
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "weather"
        }
      ],
      "dialog_turn_counter": 3,
      "dialog_request_counter": 3,
      "_node_output_map": {
        "conversation_start": [
          0
        ],
        "greeting": [
          0
        ],
        "weather": [
          0
        ]
      }
    }
  }
}
```

Figure 8-15 Complete exchange asking for the temperature of New York

> **Note:** Try different scenarios to test the application. If the application fails to respond to some scenarios, it needs more training by adding more user examples to the intents in the Car Chat-bot Workspace or by editing the entities.

8.5 Quick deployment of application

A second GitHub repository is provided so that you can run the application in this use case even if you did not perform the steps described in 8.4, “Step-by-step implementation” on page 207. This section is independent from those steps, and it includes instructions to run the application more quickly.

Use the GitHub repository that contains the *complete* code:

https://github.com/snippet-java/redbooks-conv-201-stt-tts-nodejs

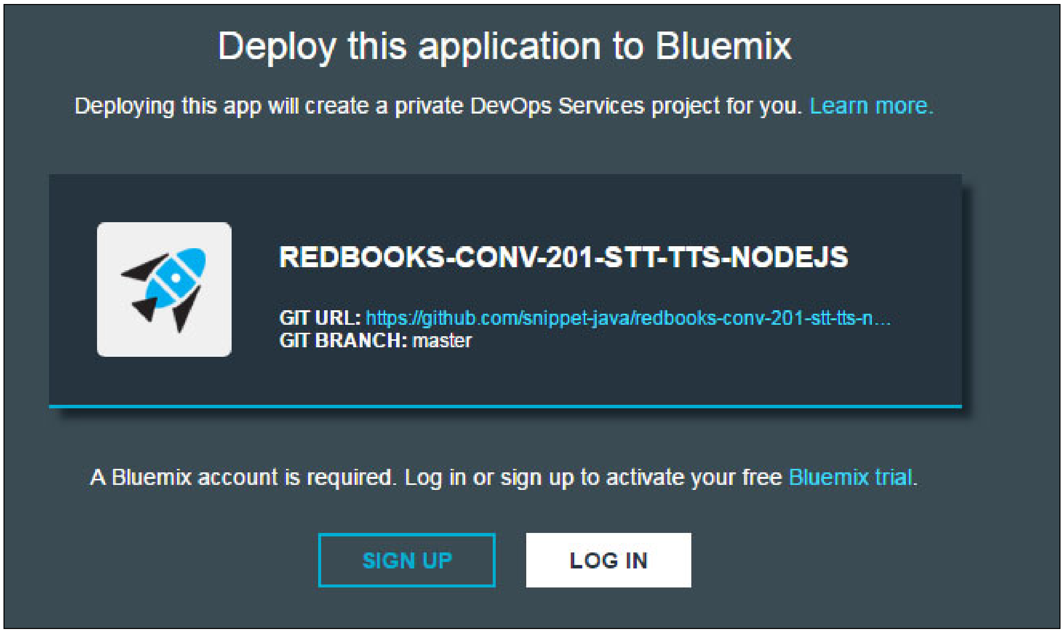
8.5.1 Deploy the application to Bluemix

To deploy the completed code, follow these steps:

1. Click the following link to begin deployment of the application to Bluemix:

   https://bluemix.net/deploy?repository=https://github.com/snippet-java/redbooks-conv-201-stt-tts-nodejs

2. Log in with your account on Bluemix (Figure 8-16 on page 220).



Deploy this application to Bluemix

Deploying this app will create a private DevOps Services project for you. Learn more.

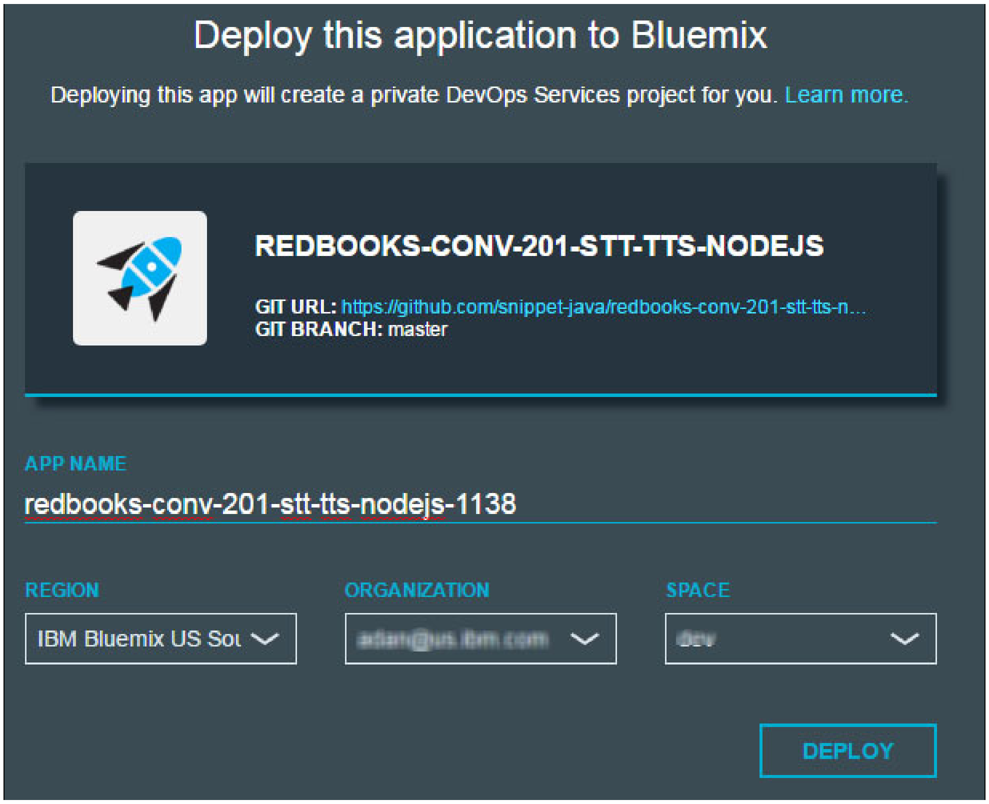
REDBOOKS-CONV-201-STT-TTS-NODEJS

GIT URL: https://github.com/snippet-java/redbooks-conv-201-stt-tts-n...
GIT BRANCH: master

A Bluemix account is required. Log in or sign up to activate your free Bluemix trial.

Figure 8-16 Log in for click to deploy

3. You can leave the default APP NAME, or change it. Change the REGION, ORGANIZATION, and SPACE to match the one used in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157 to use the same Conversation Service and Weather Company Data service, as shown in Figure 8-17.



Deploy this application to Bluemix

Deploying this app will create a private DevOps Services project for you. Learn more.

REDBOOKS-CONV-201-STT-TTS-NODEJS

GIT URL: https://github.com/snippet-java/redbooks-conv-201-stt-tts-n...
GIT BRANCH: master

APP NAME: redbooks-conv-201-stt-tts-nodejs-1138

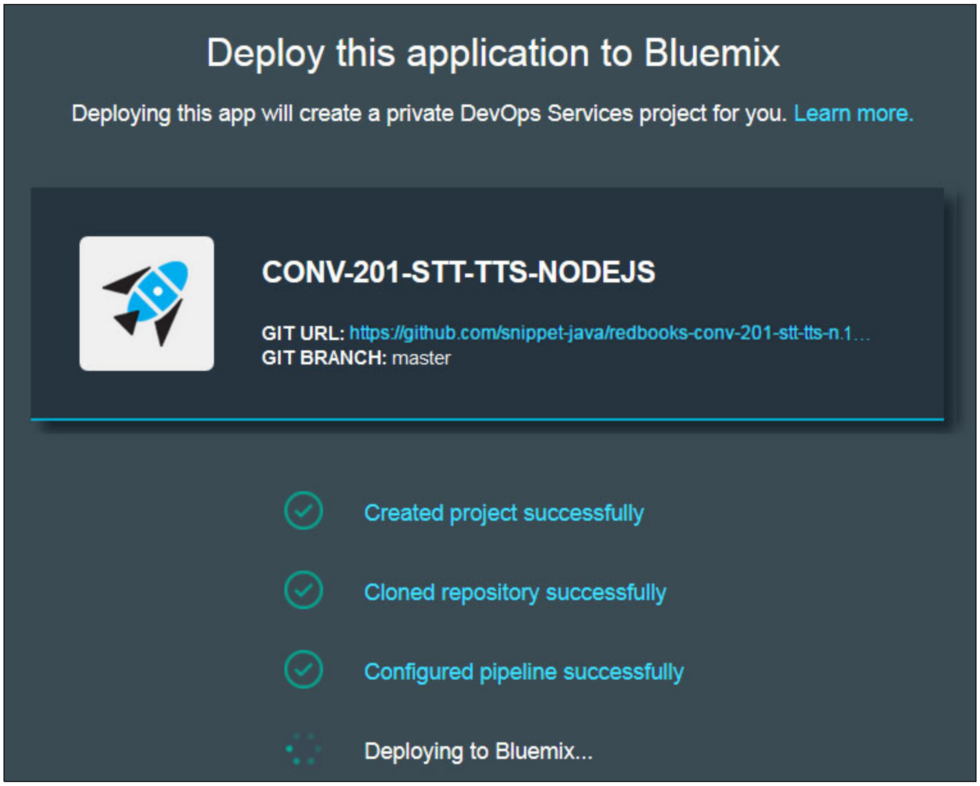
REGION: IBM Bluemix US Sou

ORGANIZATION: [redacted]

SPACE: [redacted]

Figure 8-17 Click to deploy application details

4. Click **DEPLOY**.
5. The application begins to deploy as it goes through the following actions:
   - Creates a private DevOps Service project for the app.
   - Clones the code from the GitHub URL to the new project created.
   - Configures the pipeline to build and deploy automatically.
   - Creates the Node.js application.
   - Binds the Conversation and Weather Company Data service instances created in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157 to the new application.
   - Creates new Speech to Text and Text to Speech instances and binds them to the new application.
6. The status of the deployment is shown in Figure 8-18.



Deploy this application to Bluemix

Deploying this app will create a private DevOps Services project for you. Learn more.

CONV-201-STT-TTS-NODEJS

GIT URL: https://github.com/snippet-java/redbooks-conv-201-stt-tts-n.1…
GIT BRANCH: master

Created project successfully

Cloned repository successfully

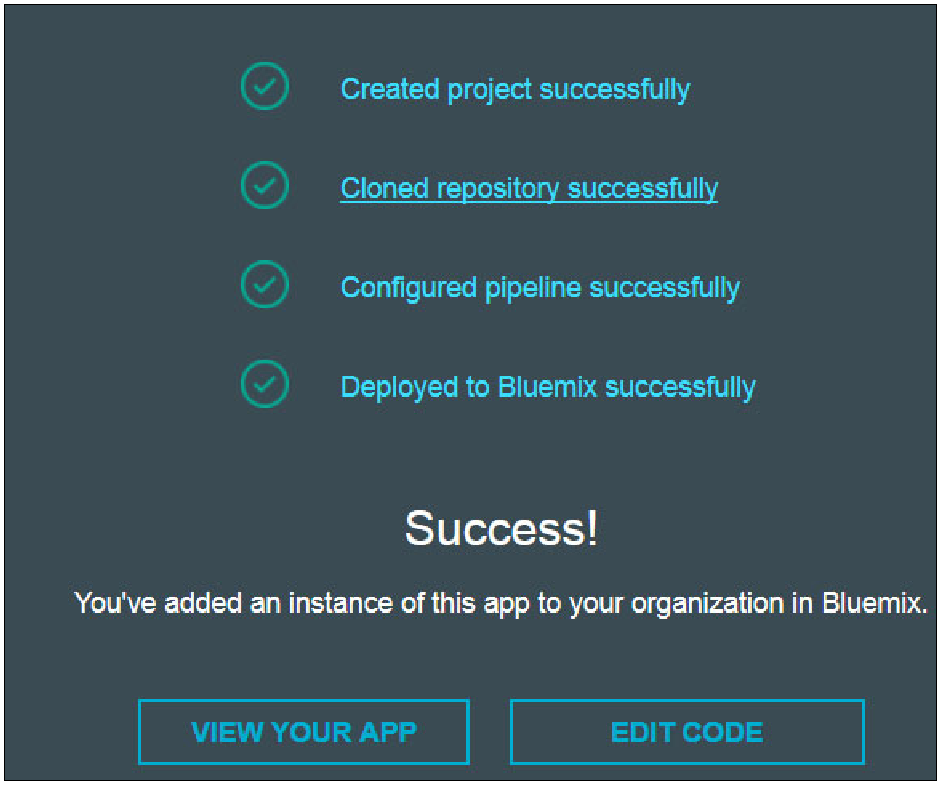
Configured pipeline successfully

Deploying to Bluemix...

Figure 8-18 Click to deploy status

Note: The deployment can take some time.

When deployment is finished, a deployment success message displays (Figure 8-19).


✓ Created project successfully

✓ Cloned repository successfully

✓ Configured pipeline successfully

✓ Deployed to Bluemix successfully

Success!

You've added an instance of this app to your organization in Bluemix.

Figure 8-19 Click to deploy success message

Important: Do not view the application now.

Copy the Car Chat-bot workspace ID

To copy the Car Chat-bot workspace ID, follow these steps:

1. Open your Bluemix Dashboard.
2. Click the Conversation service created in Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157.
3. Launch Conversation Tooling by clicking **Launch tool**.
4. The Workspaces page opens. On the Car Chat-bot workspace, click the **Actions** icon (top left of the Workspaces box) and select **View details** (Figure 8-20).

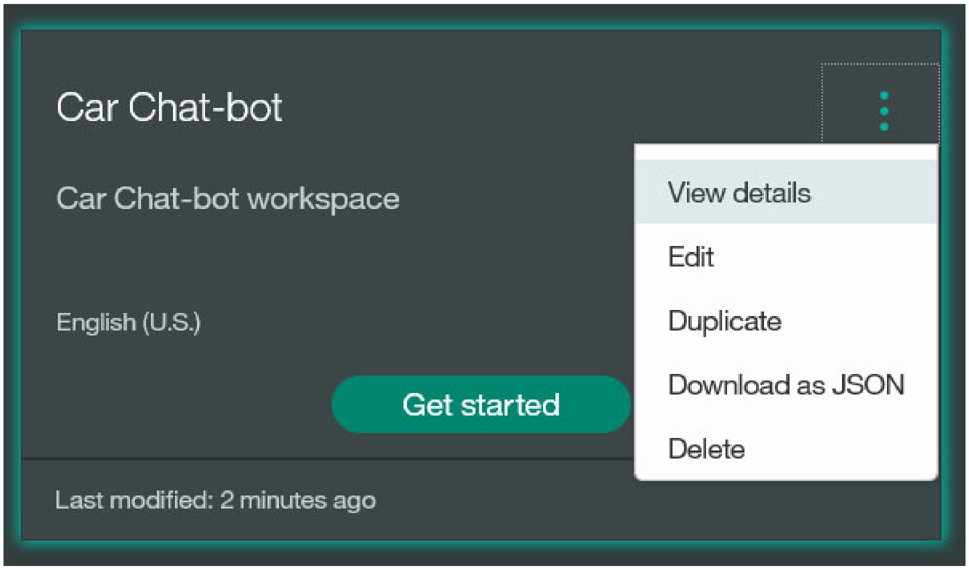
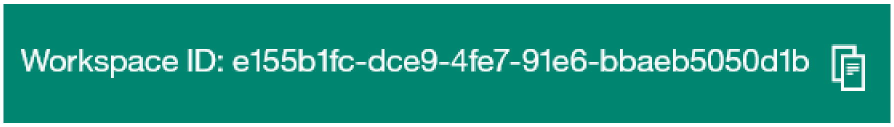


Figure 8-20 Car Chat-bot workspace view details

5. Copy the **Workspace ID**, as shown in Figure 8-21.



Workspace ID: e155b1fc-dce9-4fe7-91e6-bbaeb5050d1b

Figure 8-21 Workspace ID example

Add the WORKSPACE_ID environment variable

To add the WORKSPACE_ID environment variable, follow these steps:

1. Return to Bluemix Dashboard.
2. Click the application deployed previously. (in this example it is named `conv-201-stt-tts-nodejs-1138`). The application details are displayed.
3. Click **Runtime** from the navigation bar (Figure 8-22).

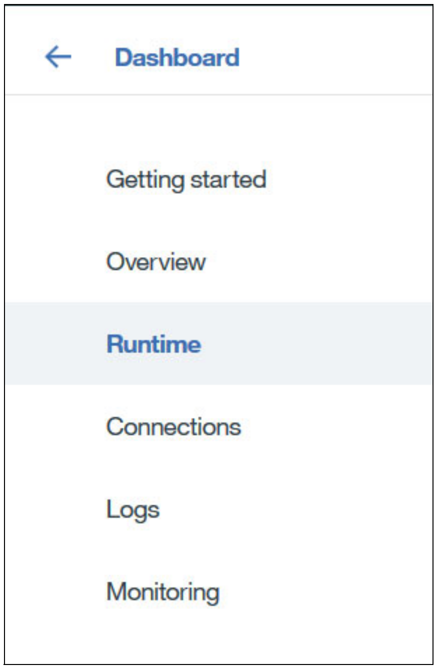


Figure 8-22 Application Runtime details

4. Select the **Environment variables** tab (Figure 8-23).

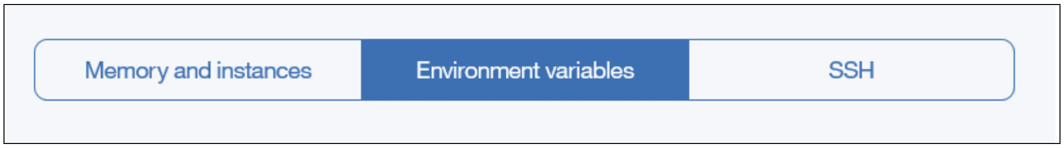
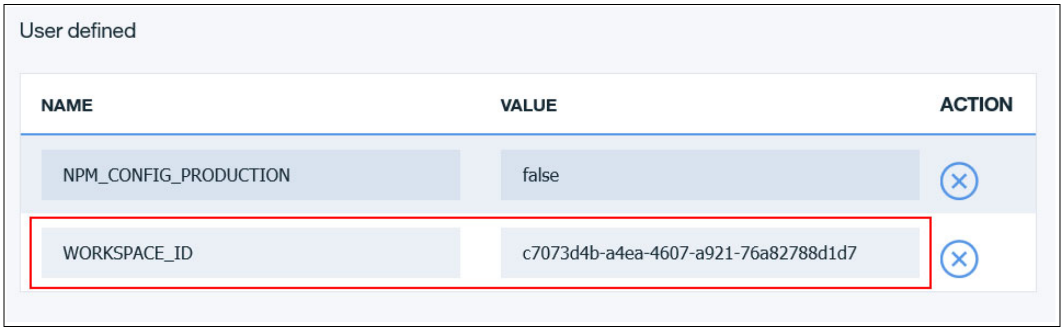


Figure 8-23 Environment variables tab

5. Scroll to the user-defined section, and click **Add**.

6. Enter `WORKSPACE_ID` as the NAME, and paste the Workspace ID copied from “Copy the Car Chat-bot workspace ID” on page 223) as the VALUE (Figure 8-24).

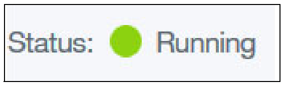


User defined

NAME	VALUE	ACTION
NPM_CONFIG_PRODUCTION	false	ⓧ
WORKSPACE_ID	c7073d4b-a4ea-4607-a921-76a82788d1d7	ⓧ

Figure 8-24 WORKSPACE_ID environment variable

7. Click **Save**. Wait for the application to restart and the status to show as `Running` (Figure 8-25).



Status: ● Running

Figure 8-25 Application running status

8. Click the **View app** button to run the application.

For more information about the expected behavior of the application, see 8.4.4, “Testing the application” on page 217.

8.6 References

For more information about this topic, see the following resources:

- IBM Watson Conversation service documentation and tutorial:

  `https://www.ibm.com/watson/developercloud/doc/conversation/index.html`

- Speech to Text service documentation and tutorial:

  `https://www.ibm.com/watson/developercloud/doc/speech-to-text/index.html`

- Text to Speech service documentation and tutorial:

  `https://www.ibm.com/watson/developercloud/doc/text-to-speech/index.html`

A

Additional material

This book refers to additional material that can be downloaded from the Internet.

Locating the web material

The following Git repositories and files are available to help you with examples in this book:

- Chapter 2, “Conversation service workspace” on page 13
  - https://github.com/snippet-java/redbooks-conv-201-weather-nodejs/blob/master/training/1.4-conv-101-createservice-incomplete.json
- Chapter 3, “Cognitive Calculator chatbot” on page 55
  - https://github.com/snippet-java/redbooks-conv-201-calc-nodejs
  - https://github.com/snippet-java/redbooks-conv-201-calc-nodejs/blob/master/training/calculator_workspace.json
  - https://github.com/watson-developer-cloud/conversation-simple
- Chapter 4, “Help Desk Assistant chatbot” on page 109
  - https://github.com/snippet-java/Node-RED-bluemix-conversation-starter.git-1487332833126
  - https://github.com/snippet-java/redbooks-conv-201-iot-nodered/blob/master/conv-201-iot-nodered-flow.json
- Chapter 5, “Using a cognitive chatbot to manage IoT devices” on page 139
  - https://github.com/ibm-watson-iot/iot-starter-for-android
  - https://github.com/snippet-java/Node-RED-bluemix-conversation-starter.git-1487332833126
  - https://github.com/snippet-java/redbooks-conv-201-iot-nodered/blob/master/conv-201-iot-nodered-flow.json

- Chapter 6, “Chatting about the weather: Integrating Weather Company Data with the Conversation service” on page 157
  - https://github.com/watson-developer-cloud/conversation-simple
  - https://github.com/snippet-java/redbooks-conv-201-weather-nodejs/blob/master/training/1.4-conv-101-createservice.json
  - https://github.com/snippet-java/redbooks-conv-201-weather-nodejs
- Chapter 8, “Talking about the weather: Integrating Speech to Text and Text to Speech with the Conversation service” on page 203
  - https://github.com/snippet-java/redbooks-conv-201-stt-tts-nodejs-student.git
  - https://github.com/snippet-java/redbooks-conv-201-stt-tts-nodejs

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

The volumes in the *Building Cognitive Applications with IBM Watson Services* series:

- *Volume 1 Getting Started*, SG24-8387
- *Volume 2 Conversation*, SG24-8394
- *Volume 3 Visual Recognition*, SG24-8393
- *Volume 4 Natural Language Classifier*, SG24-8391
- *Volume 5 Language Translator*, SG24-8392
- *Volume 6 Speech to Text and Text to Speech*, SG24-8388
- *Volume 7 Natural Language Understanding*, SG24-8398

You can search for, view, download or order these documents and other Redbooks, IBM Redpapers™, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Spring Expression Language (SpEL):

  http://docs.spring.io/spring/docs/current/spring-framework-reference/html/expressions.html

- IBM Bluemix, log in or create an account:

  https://console.ng.bluemix.net

- Node-RED programming tool:

  https://nodered.org/

- JS Foundation:

  https://js.foundation/

- Slack:

  http://slack.com

- Create a new Slack team:

  https://get.slack.help/hc/en-us/articles/206845317-Create-a-Slack-team

- Node-RED Bluemix Starter Application:

  https://github.com/snippet-java/Node-RED-bluemix-conversation-starter.git-1487332833126

- IoT starter app for Android phone:

  https://ibm.ent.box.com/v/iotstarterapp

Also see the list of online resources for the following chapters in this book:

- Basics of Conversation service: 1.5, “References” on page 12
- Conversation service workspace: 2.3, “References” on page 54
- Cognitive Calculator chatbot: 3.6, “References” on page 107
- Help Desk Assistant chatbot: 4.7, “References” on page 138
- Using a cognitive chatbot to manage IoT devices: 5.4, “References” on page 156
- Chatting about the weather: Integrating Weather Company Data with the Conversation service: 6.6, “References” on page 183
- Improving chatbot understanding: 7.3, “References” on page 202
- Talking about the weather: Integrating Speech to Text and Text to Speech with the Conversation service: 8.6, “References” on page 225

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

Building Cognitive Applications with IBM Watson Services: Volume 2 Conversation

(0.2”spine)
0.17”<->0.473”
90<->249 pages

SG24-8394-00

ISBN 0738442569

Printed in U.S.A.