

Hackathon Ibérico

A quick introduction to Bluemix and Watson

Contents

Introduction	3
What you'll need	4
Getting an IBM Bluemix Account	4
Getting the sample source code	4
Let's make a bot	7
Provisioning the Conversation service	7
Creating your first Conversation Workspace	10
Intents	12
Entities.....	12
Dialog.....	13
Testing our bot	14
Hello, Watson!.....	15
Getting Watson to know our name.....	19
Recap.....	22
Watson, what's the weather?	23
Importing an already existing workspace	23
Looking at the Weather workspace	23
A small problem.....	25
The web application	25
Setting up the Watson Conversation SDK	25
Running the application locally	27
Running the app on Bluemix	27
Creating the app.....	27
Making Watson's services available to our app.....	28
Putting our code in Bluemix	29
Enable DevOps	29
Configuring Git to push to our new repository	31
Collaboration.....	32

Introduction

This document is a guide to building a simple web application using Node.JS and Watson, and then publishing it on IBM Bluemix.

What you'll need

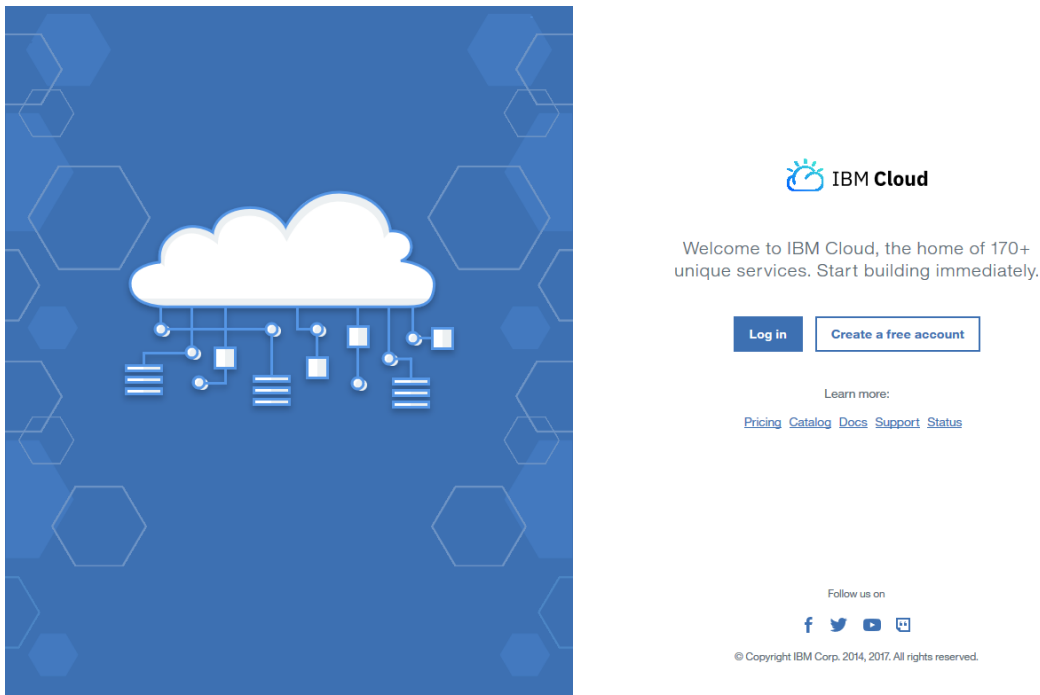
You'll need the following to participate in this session:

- An IBM Bluemix account
- Node.JS version 8 or above
 - This document was done with Node.JS 8.9.1
- Your favourite text editor/IDE
 - This document was done with Visual Studio Code
- Git
- Cloud Foundry CLI
 - This is optional, but it may come in handy.
- Your favourite web browser, provided it's up-to-date
 - Except Internet Explorer, please 😊

Getting an IBM Bluemix Account

If you already have a Bluemix Account, you can skip this step.

Go to <https://console.bluemix.net/> and choose to create a free account.



Afterwards, follow the directions on the screen.

Getting the sample source code

If you haven't got Git installed, please install it now. Then, go to the following page:

<https://github.com/RedRoserade/watson-demo-hackathon-salamanca>

This page contains the starter code we'll use for this demo.

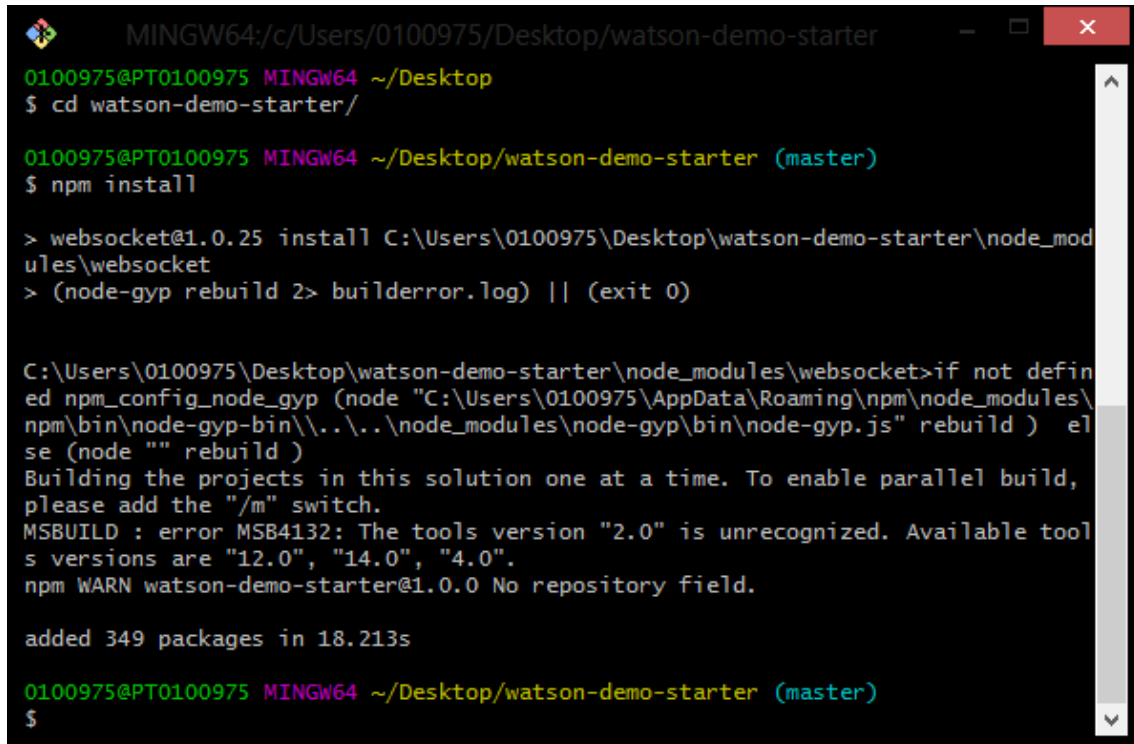
To clone it to your PC, open Git Bash (or Git CMD), cd into a folder of your choice (for example, the Desktop), and type in the following command:

```
git clone https://github.com/RedRoserade/watson-demo-hackathon-salamanca.git
```

If your output is like the above, it worked!

When git clones a repository, it creates a folder for it. cd into that folder (cd watson-demo-starter), and run the following command:

```
npm install
```



```
MINGW64:/c/Users/0100975/Desktop/watson-demo-starter
0100975@PT0100975 MINGW64 ~/Desktop
$ cd watson-demo-starter/

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-starter (master)
$ npm install

> websocket@1.0.25 install C:\Users\0100975\Desktop\watson-demo-starter\node_modules\websocket
> (node-gyp rebuild 2> builderror.log) || (exit 0)

C:\Users\0100975\Desktop\watson-demo-starter\node_modules\websocket>if not defined npm_config_node_gyp (node "C:\Users\0100975\AppData\Roaming\npm\node_modules\npm\bin\node-gyp-bin\..\..\node_modules\node-gyp\bin\node-gyp.js" rebuild ) else (node "" rebuild )
Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch.
MSBUILD : error MSB4132: The tools version "2.0" is unrecognized. Available tools versions are "12.0", "14.0", "4.0".
npm WARN watson-demo-starter@1.0.0 No repository field.

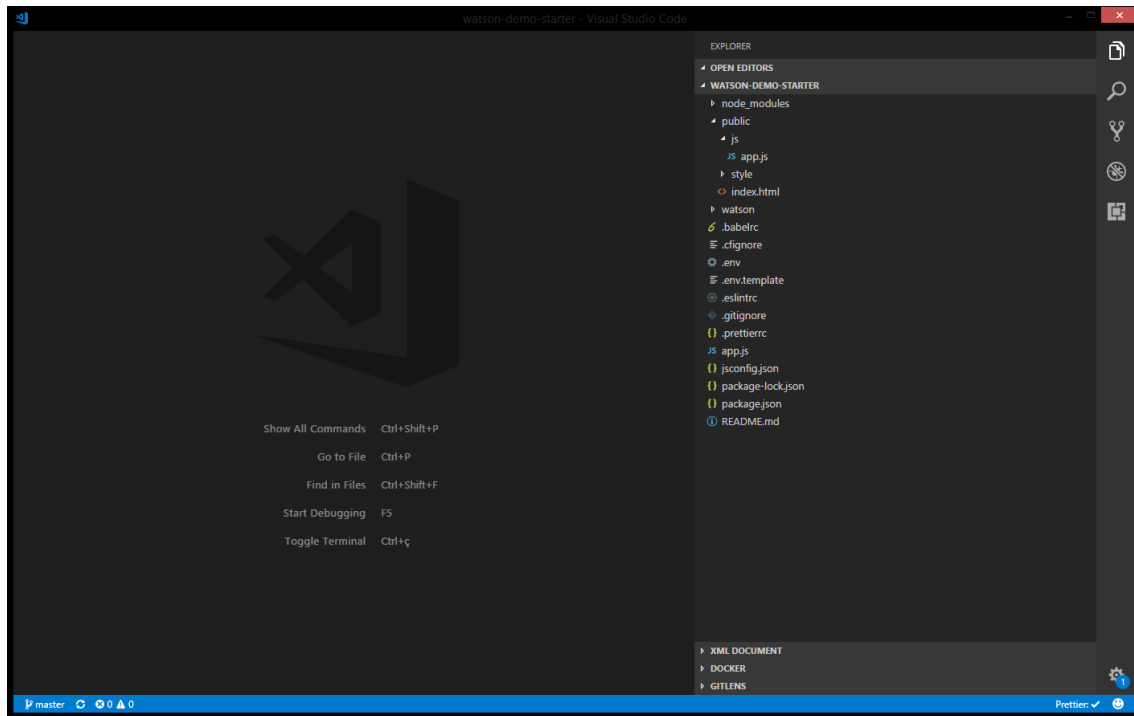
added 349 packages in 18.213s

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-starter (master)
$
```

If an MSBUILD error occurs, don't worry, it should work anyway.

Keep the terminal open, you'll need it for later.

Finally, open the folder in Visual Studio Code, and it should look something like this:



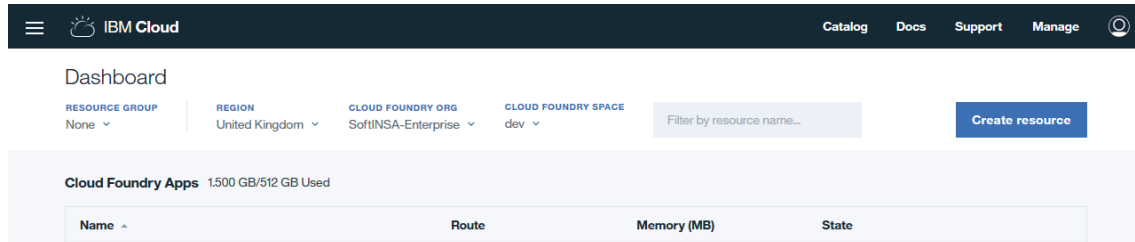
Let's make a bot

We'll start with the bot. There's no point in having an application when there's no bot to talk to (as it would be a *monologue* rather than a *dialogue*).

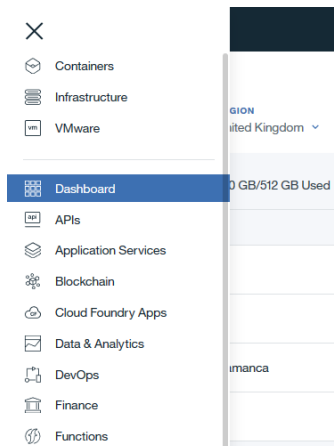
Provisioning the Conversation service

For us to use Watson's services on Bluemix, first we need to ask Bluemix for Watson's services.

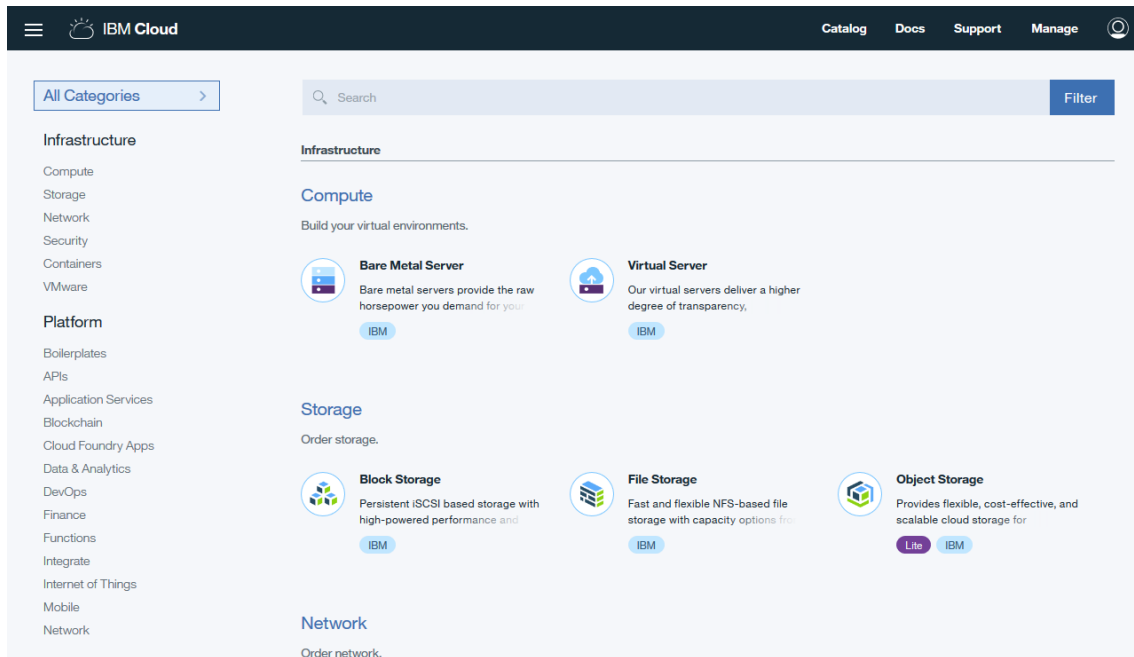
When you log into Bluemix, at the top right corner of the page, click "Create resource":



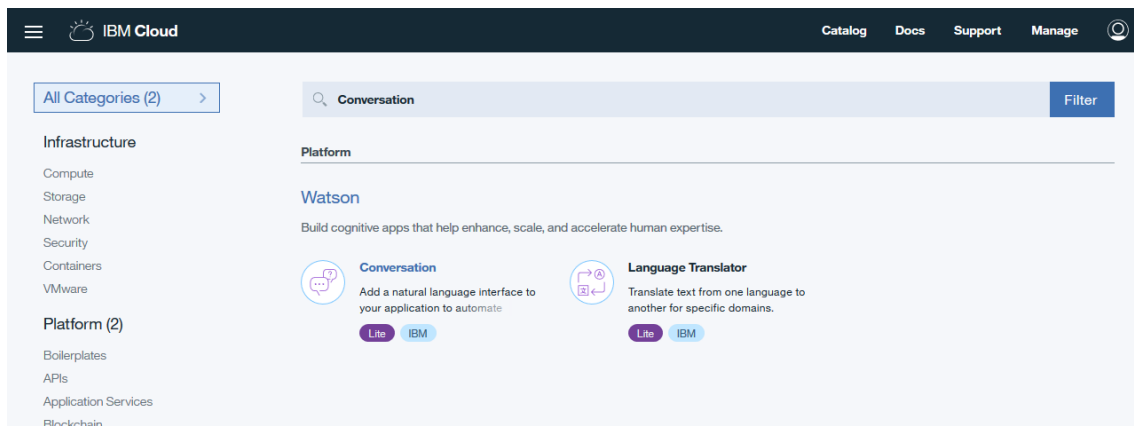
If you don't see it, ensure that you're on the dashboard by using the left-hand menu:



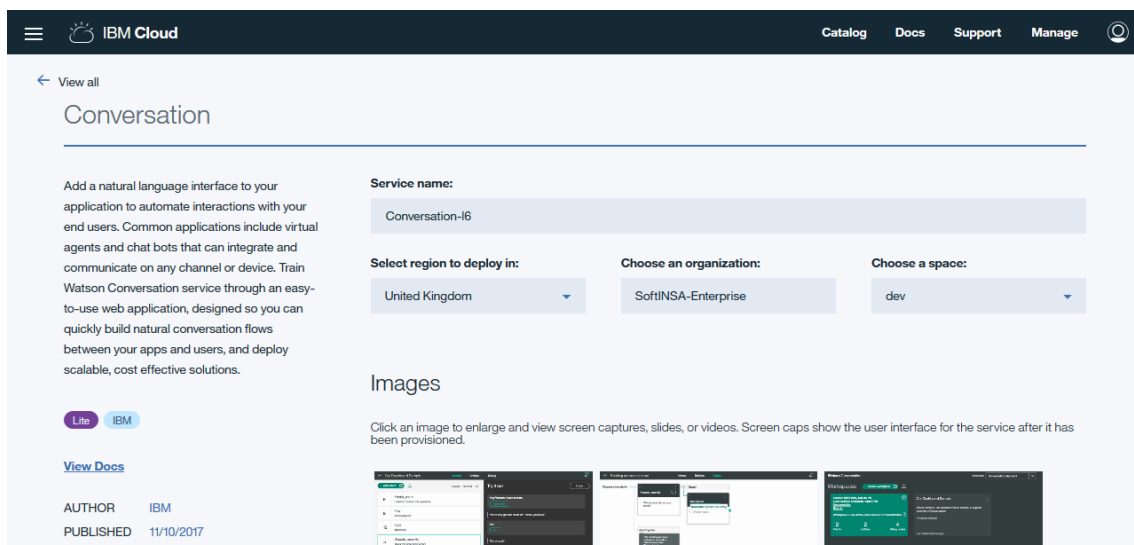
After clicking on "Create resource", the Bluemix catalogue appears. Here you can browse and search for all the services Bluemix has to offer, from databases, to servers and network, to security, to Watson.



Our bot is going to make use of Watson's "Conversation" API, so let's search for it using the search bar at the top:



And select the "Conversation" item. It will open the following window:



You can consult the details of this service on this page, including pricing, terms and conditions, and so on. We'll choose the "Lite" pricing plan, which will be more than enough for what we need:

Pricing Plans Monthly prices shown are for country or region: Portugal

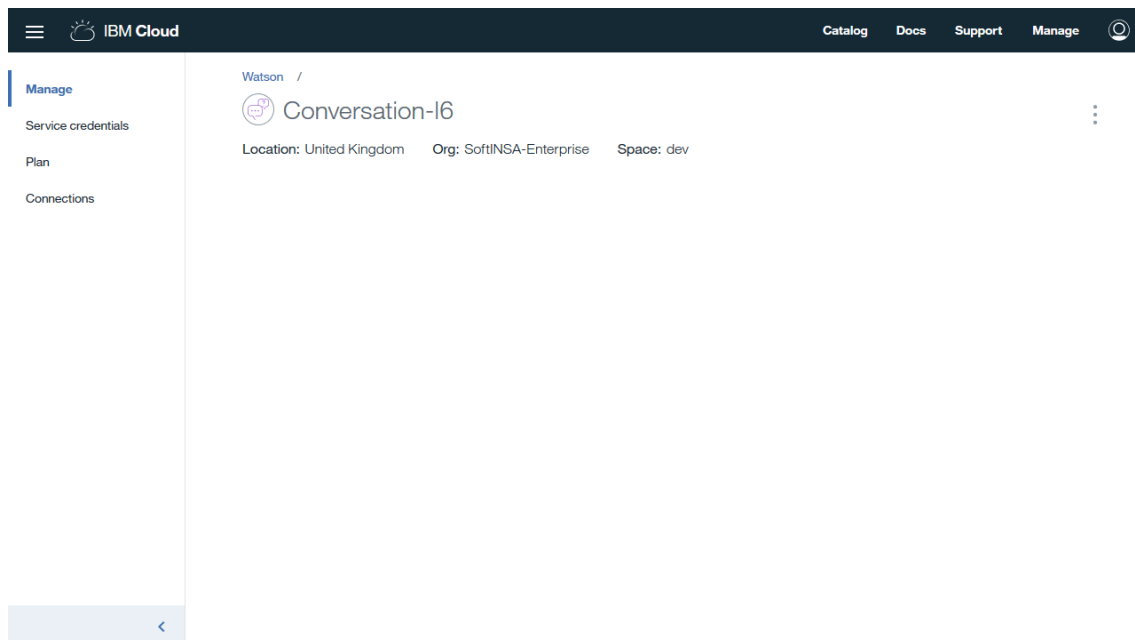
PLAN	FEATURES	PRICING
✓ Lite	10,000 API Calls per Month* Up to 5 Workspaces Up to 25 Intents Up to 25 Entities *POST /message method calls only	Free

The Lite plan gets you started with 10,000 API calls per month at no cost. And when you upgrade to a paid plan, you'll keep all your intents, entities, dialog flows, and chat logs.

Lite plan services are deleted after 30 days of inactivity.

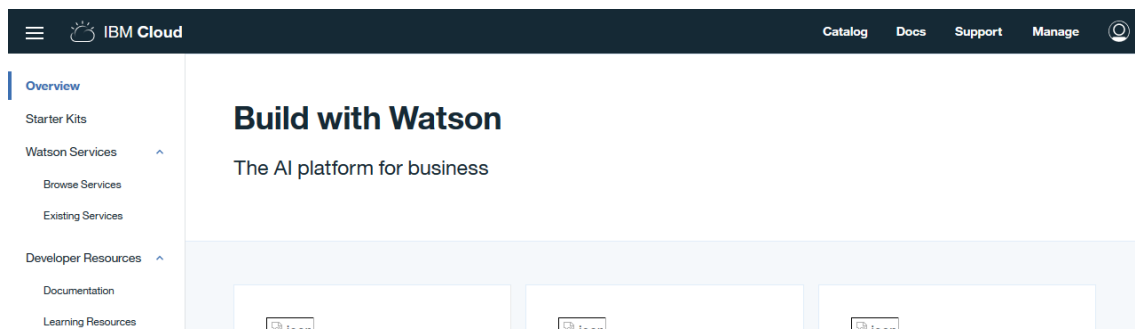
Don't forget to give this service a descriptive name. Something you'll easily recognize. What you choose is up to you.

Choose a name and click "Create" at the bottom of the screen. After a while, Watson will be ready for you:



You don't land in the most exciting (or the most intuitive) of pages, but if you see this page (the "Conversation-l6" name will likely be different for you), everything went well.

At the top of the page, click "Watson". You'll be taken to the Watson developer page:



On the left menu, click on “Existing Services” under “Watson Services”. Here you can see the services you have (even non-Watson ones):

Existing Services

REGION: **X** United Kingdom | CLOUD FOUNDRY ORG: SoftINSA-Enterprise | CLOUD FOUNDRY SPACE: **X** dev | [Browse services](#)

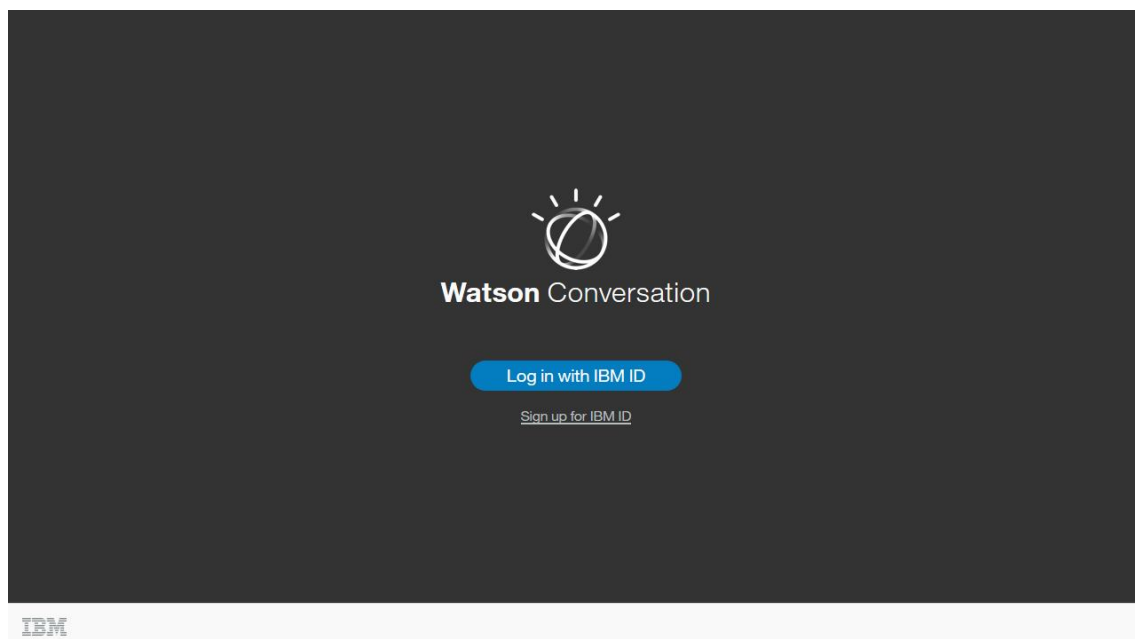
Watson Services (4)

NAME	SERVICE OFFERING	PROJECT	TRAINING	PLAN
availability-monitoring-auto	Availability Monitoring	—	—	
conversation-demo-hackathon	Conversation	—	Launch tool	Lite
Conversation-16	Conversation	—	Launch tool	Lite
Monitoring and Analytics-83	Monitoring and Analytics	—	—	

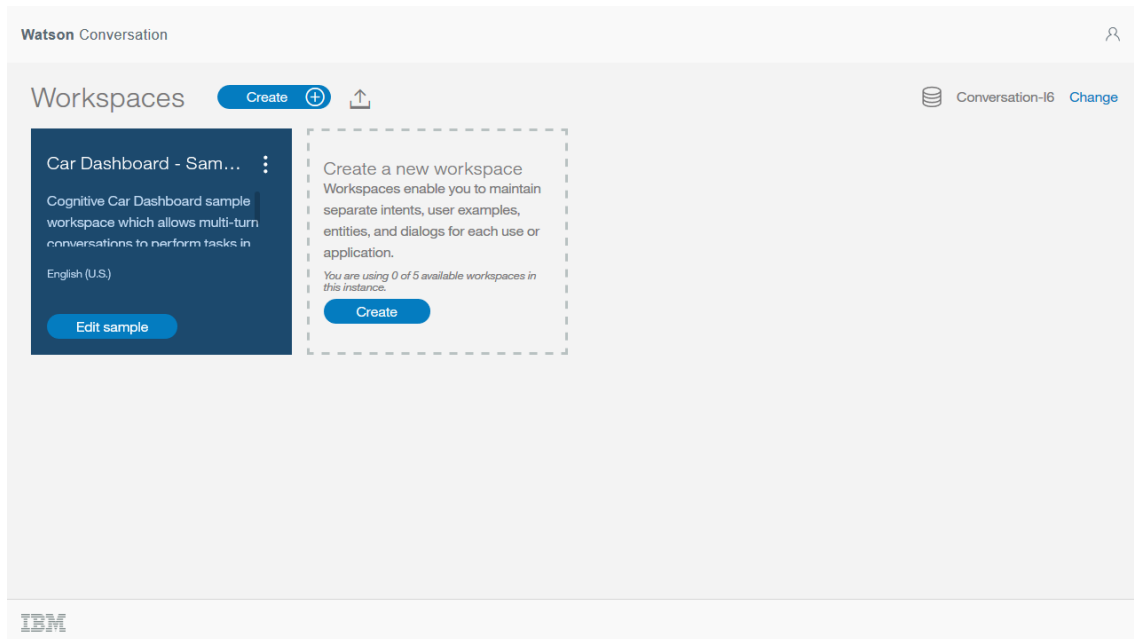
Find the conversation service you created earlier, and click on “Launch tool”. This will open the Watson Conversation developer tools application.

Creating your first Conversation Workspace

If this page appears first:



Just log in with your Bluemix credentials. You will then be taken here:

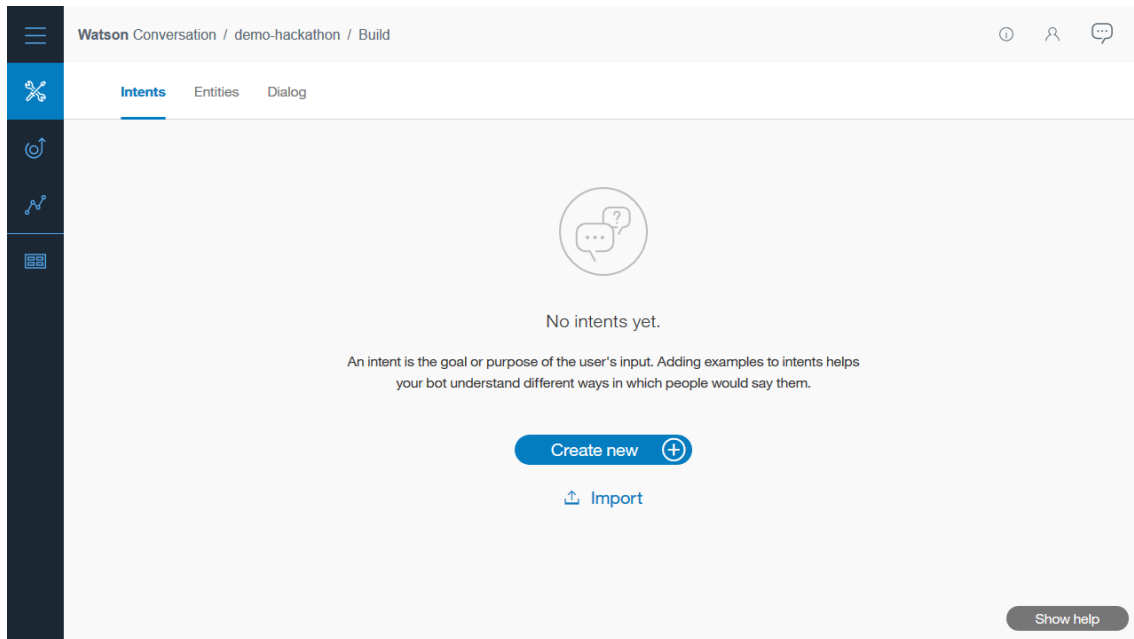


A “Car Dashboard” sample workspace has already been created for you. It shows how you can use Watson Conversation to make a bot that helps drivers.

We’ll create our own, though. So, under “Create a new workspace”, click “Create”. You can create 5 workspaces for free.

Give your workspace a name, a description, and “English (U.S.)” as the language. I’ll leave it up to you to create bots in other languages 😊.

Finally, click “Create”, and you’ll be greeted with this:



Our workspace has very little right now, but we'll add things to it in a moment.

At the top of the page, you'll see 3 menu options:

- Intents
- Entities
- Dialog

Intents

Intents are patterns that Watson will use to determine what the user wants. The more intents you have, the better will your bot understand what your user wants.

An example intent would be a greeting, such as "Hello", "Hi", or "Good morning".

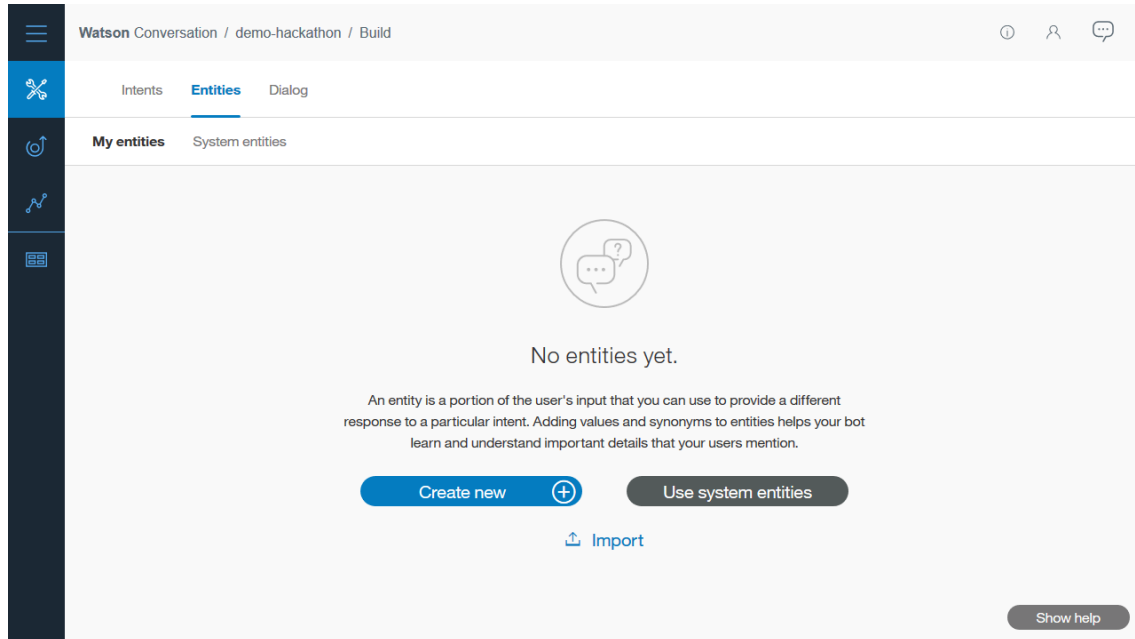
Entities

Entities are information about intents, and provide meaning to them. For example, if you were to ask for the weather, you might ask "What's the weather in Prague?" "What's the weather in" would be the intent, and "Prague" would be an entity for "city". The values captured by entities can be stored as variables. This will be important later.

There are two types of entities:

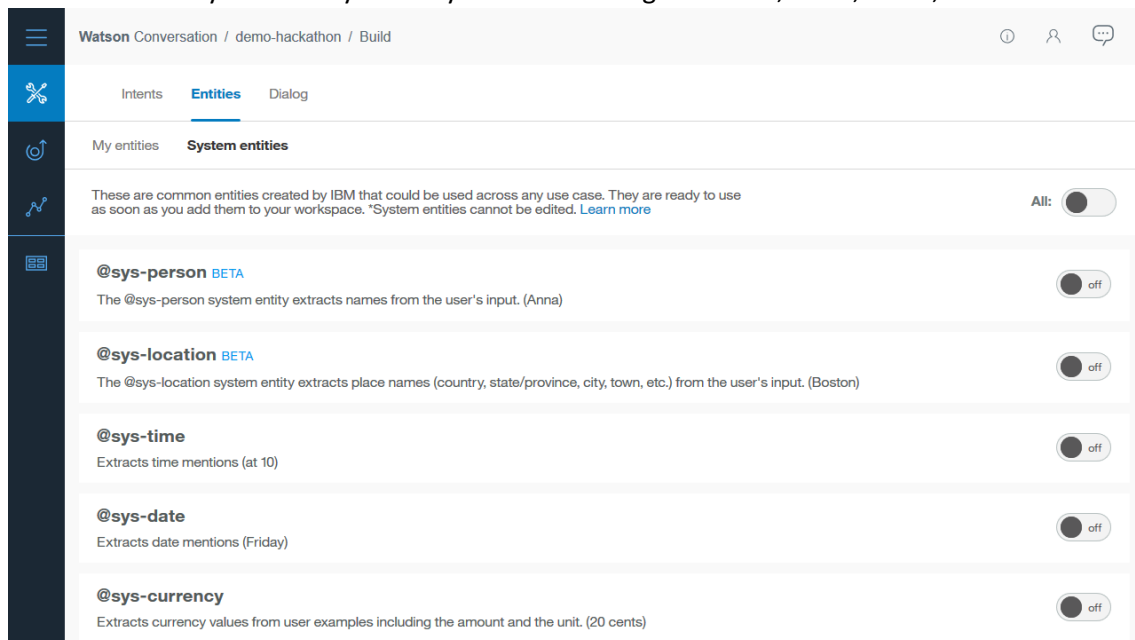
My entities

You can define your own entities here, such as names of people, regions, units, etc. It's whatever you need to "match". You can also match patterns using regular expressions:



System entities

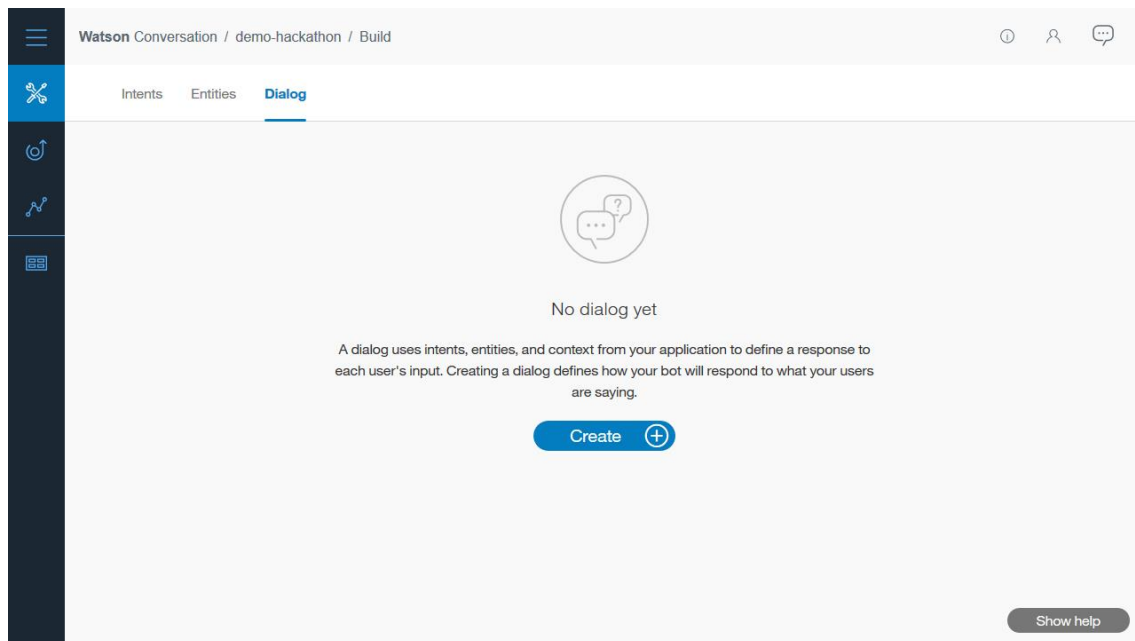
These are already made for you. They can match things like time, cities, dates, and so on:



You turn on whatever system entities you need.

Dialog

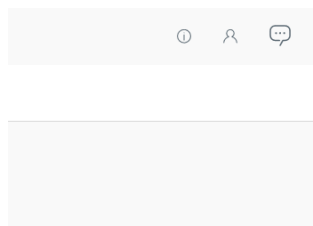
Dialog is where you combine intents and entities to make your bot:



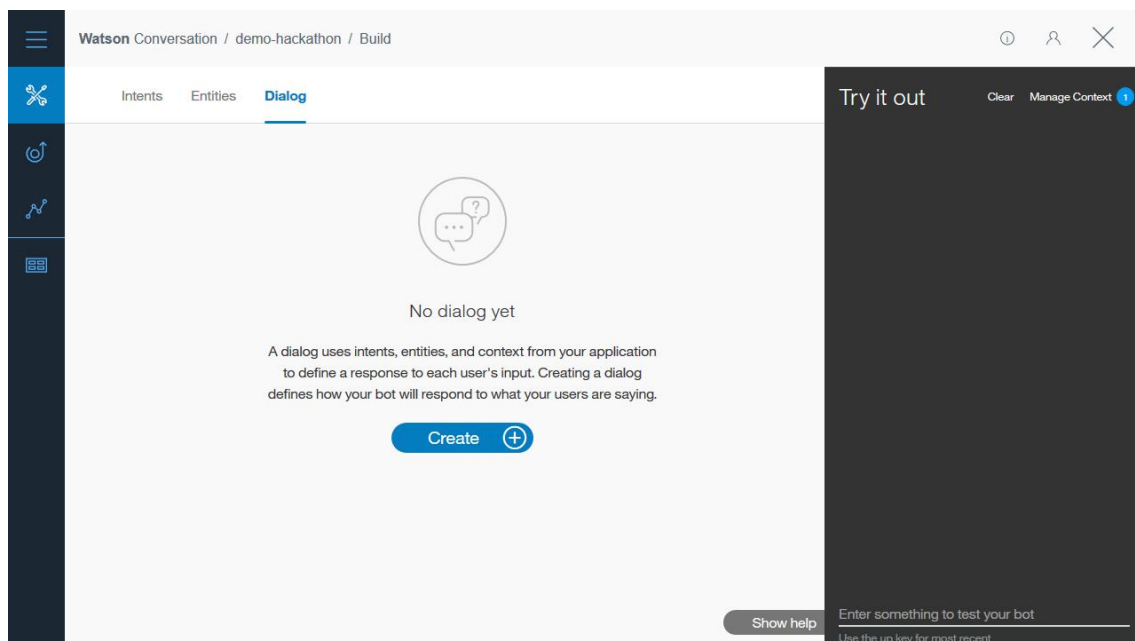
As you can see, our bot doesn't know anything about dialog.

Testing our bot

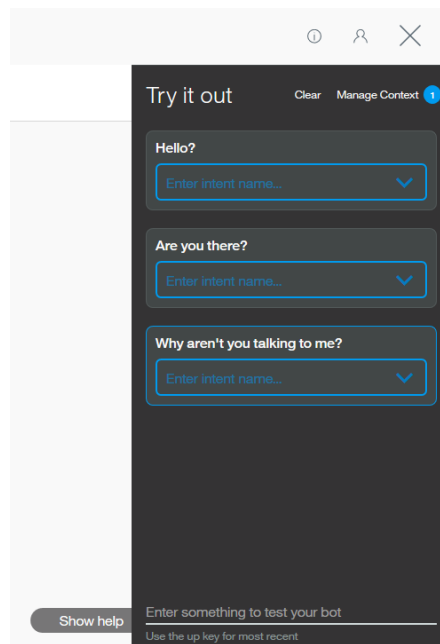
Click the chat bubble icon on the top right corner of the page:



This will open the chat window, where we can test our bot's capabilities:



Try typing something into the chat window!



We've made the *chattiest* bot ever... I'd rather talk to myself!

Close the chat window for now, we'll come back to it later.

Hello, Watson!

Let's make our bot talk about *something*, because right now, he's rather *quiet*.

Our first intent

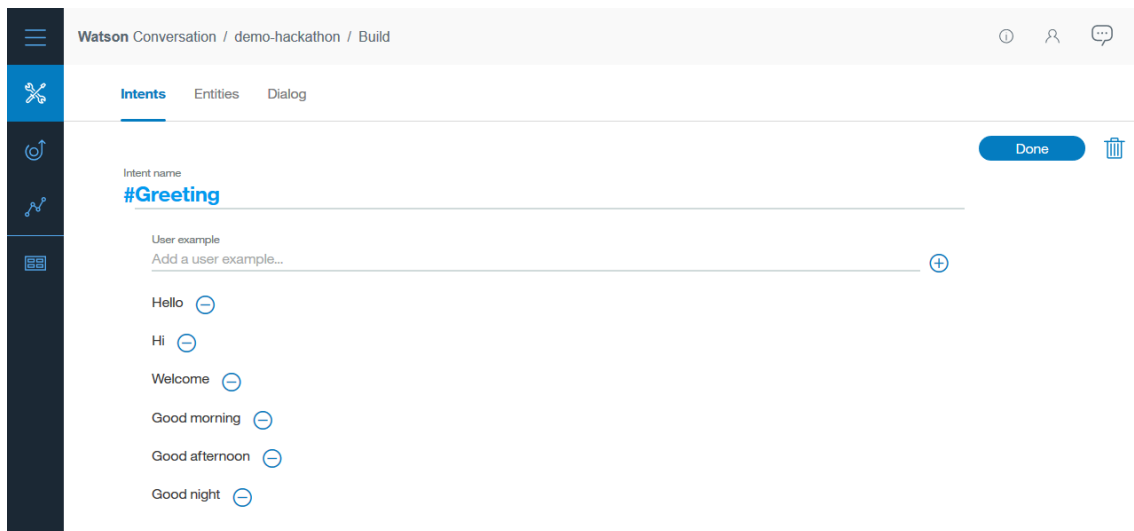
Head over to the "Intents" tab, and add a new intent by clicking on "Create new":



Here, you:

- Give a name to the intent. Let's call it "Greeting".
- Add one or more user examples. These are patterns of text that will be used to match against this intent. You can add as many as you wish.

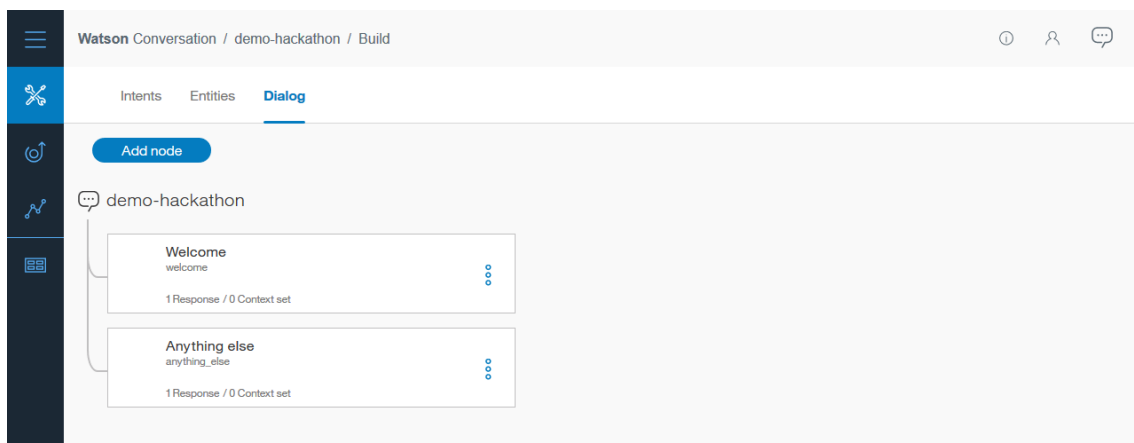
Here's mine:



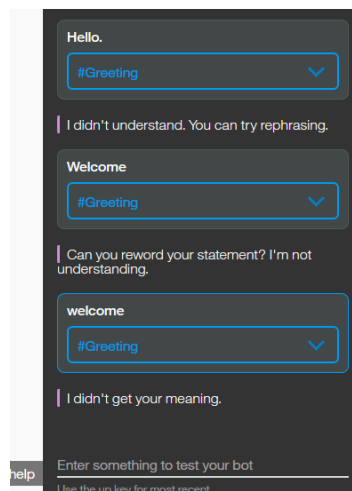
When you're happy, click "Done". You can later edit this intent by clicking on it on the intent tab.

Let's talk

Go back to the "Dialog" tab, and add a new dialog by clicking on "Create". This will add a simple dialog:



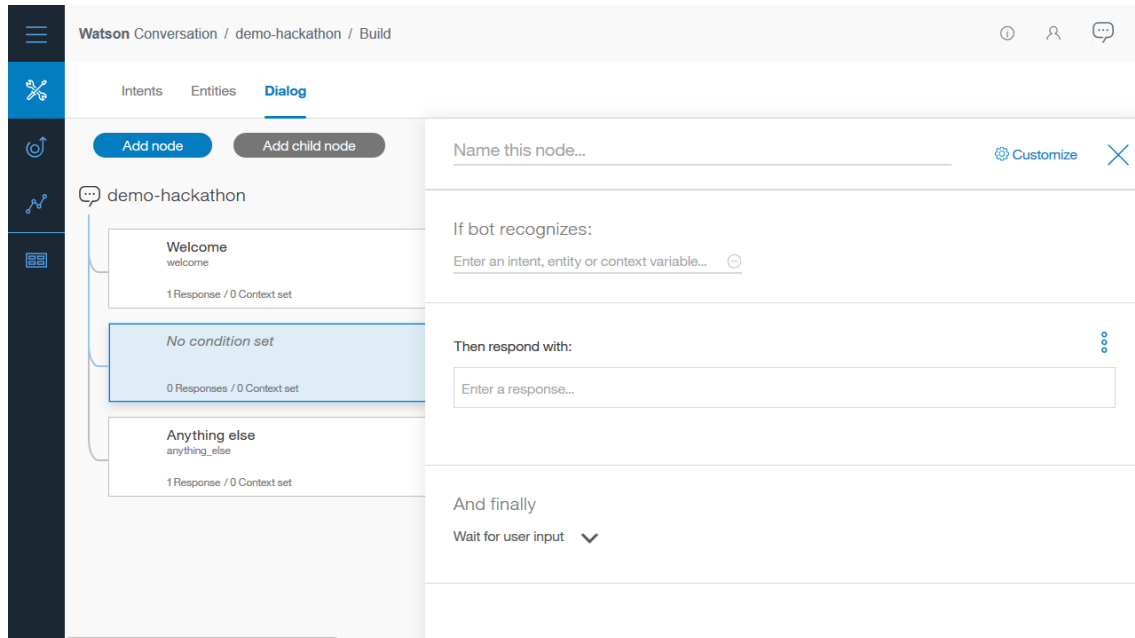
If we were to test our bot now, it would be able to greet you when the chat box opens, and when it doesn't know what your *intent* was, it'll ask for you to rephrase:



Well, at least it *talks* now, even though it's a bit *stupid*. Let's fix that, shall we?

Teaching a bot how to greet

Let's add a new node to the conversation. A *node* is used by Watson to match the user's intent, extract any entities it found, and then reply with a message:



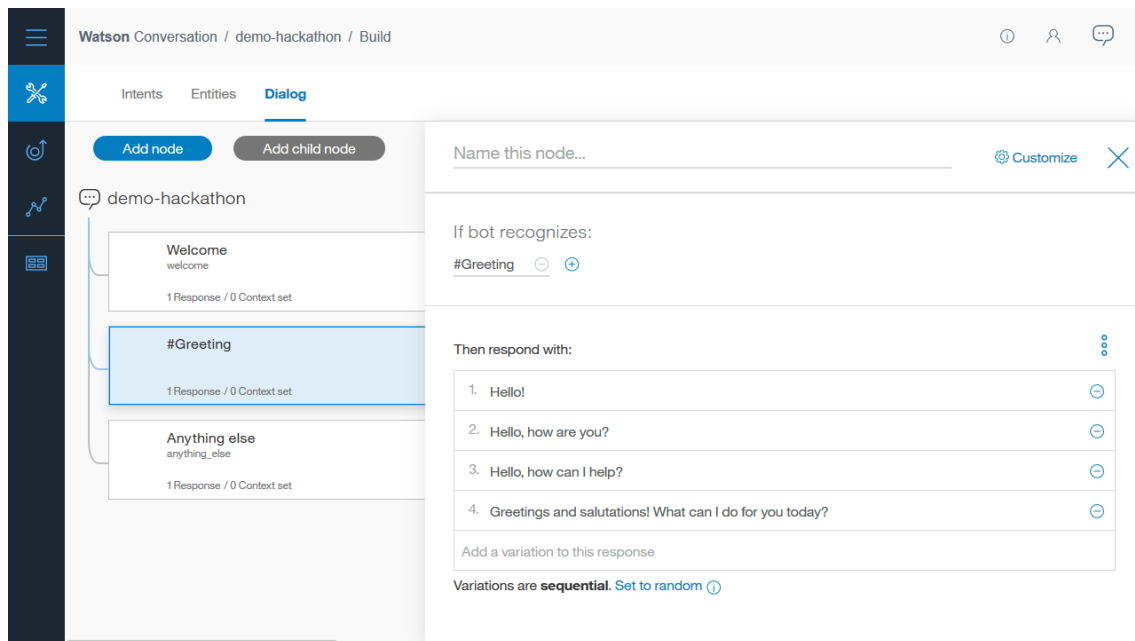
You don't *need* to give a name to the node. If you don't, it'll use whatever you put on the "If the bot recognizes" box as its title. It's up to you.

Add the intent you created earlier by typing "#" followed by the name of your intent. In my case, I named it "Greeting", so I would type "#Greeting".

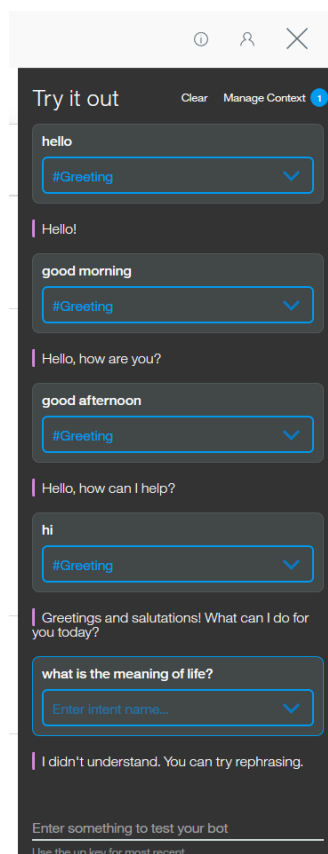
Then, add a few replies. Be polite 😊

The final option we'll leave as is, but you could use it to jump to other nodes, for example.

Here's mine:



Close the pane when you're happy, and test the bot again (you can clear the chat log by clicking on "Clear" once the chat window opens):



It still doesn't know much, but at least it's polite now!

Notice that the order of the replies from Watson are in the same order as the ones defined on the previous image. You can change this by using the "Set to random" option below the responses.

Getting Watson to know our name

Watson doesn't know much about us. So, let's make use of an *Entity* to have Watson reply with our name back.

Go to the *Entities* tab, and add a new entity:

The screenshot shows the 'Entities' tab in the Watson Conversation interface. At the top, there's a breadcrumb 'Watson Conversation / demo-hackathon / Build'. Below it, tabs for 'Intents', 'Entities', and 'Dialog' are visible, with 'Entities' being the active tab. The main area contains a form to add a new entity. The 'Entity' field has a placeholder text '@Add the entity name, for example, Animals'. To the right of this field is a 'Done' button and a close 'X' icon. Below the entity name field, there's a 'Fuzzy Matching BETA' section with a description 'Turning this on will increase the ability for Watson to recognize misspelled entity values.' and a toggle switch currently set to 'off'. At the bottom, there's a table with two columns: 'Value' and 'Type'. The 'Value' column has a placeholder 'Add a value, for example, Cat'. The 'Type' column has a dropdown menu currently showing 'Synonyms' and a plus icon to add more synonyms.

Let's create a "Names" entity. For that, give the entity a name, and then add multiple values to it. Add as many as you'd like:

This screenshot shows the 'Entities' tab after creating a new entity named 'Names'. The 'Entity' field now contains the text 'Names'. The 'Fuzzy Matching BETA' toggle is still 'off'. The table below has been populated with several values. The 'Value' column lists names: 'André', 'Andreia', 'Gonçalo', 'Nuno', 'Marta', 'Teresa', 'Fábio', and 'João'. The 'Type' column for all these entries is 'Synonyms'. To the right of each row in the table is a minus icon to remove the entry. At the bottom right, there is a 'Show help' button.

Value	Type
Add a value, for example, Cat	Synonyms
André	Synonyms
Andreia	Synonyms
Gonçalo	Synonyms
Nuno	Synonyms
Marta	Synonyms
Teresa	Synonyms
Fábio	Synonyms
João	Synonyms

You can also add *synonyms* to some values, so that they're matched a little more easily.

Click "Done" when you're happy with what you have.

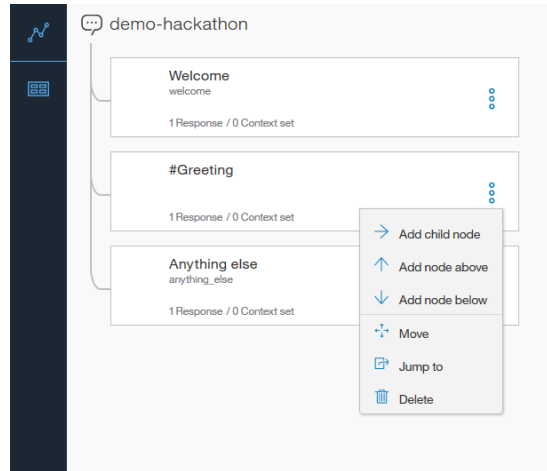
Adding an intent for a user to introduce themselves

Go back to *Intents*, and create a new intent. You can name it whatever you want. In there, add a few expressions that someone might use to introduce themselves, such as:

- I'm
- My name is
- I'm called

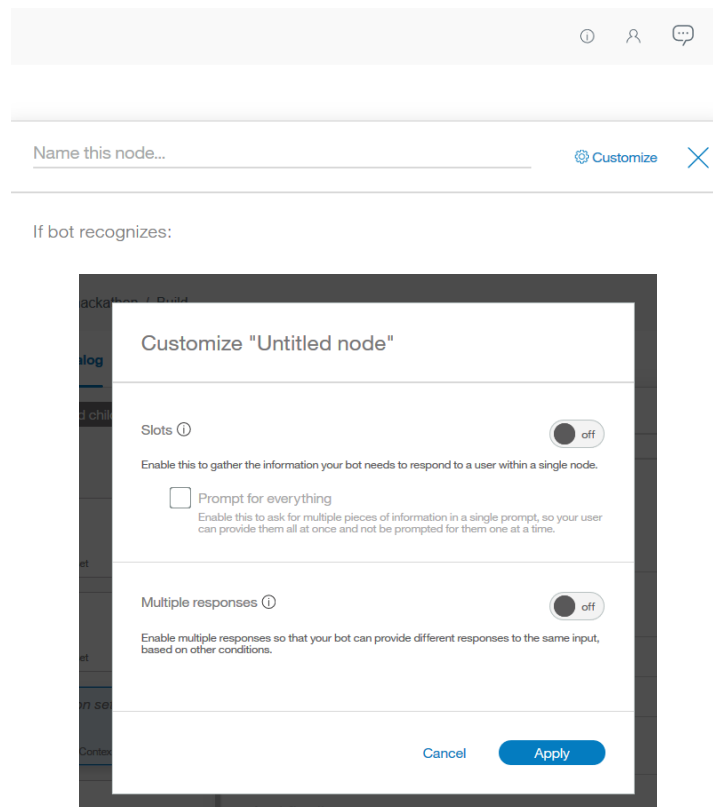
Continuing the conversation

Let's go back to *Dialog*, and add a new “child node” to the one we added earlier, and choose “Add child node”:



Child nodes allow Watson to follow a conversation chain, with back and forth messaging, to reach a goal.

In the new node, click “Customize” on the pane that opened:



Enable the “Slots” feature. Slots allow us to save information in *context*, which can be used for later. The information can be entities, or intents. Click “Apply” when you’re done:

On “If the bot recognizes”, add the intent created earlier.

Watson Conversation / demo-hackathon / Build

Intents Entities **Dialog**

Add node Add child node

demo-hackathon

Welcome
welcome
1 Response / 0 Context set

#Greeting
1 Response / 0 Context set

#Naming
0 Responses / 0 Context set / 1 Slot

Anything else
anything_else
1 Response / 0 Context set

Name this node...

If bot recognizes:
#Naming

Then check for:

Check for	Save it as	If not present, ask	Type
1 Enter entity or intent	Enter context variable	Enter a prompt	Optional

Add slot

Then respond with:
Enter a response...

In “Then check for”, add the entity you created earlier. If the name of the entity is “Names”, you write “@Names”. “#” is for intents, “@” is for entities.

In “Save it as”, write “\$”, followed by a variable name, such as “\$name”. You can use the third box to make your bot ask the user to try again if the entity wasn’t found.

Finally, in the response box, type “Nice to meet you, \$name!”

Here’s mine:

emo-hackathon / Build

Dialog

Add child node

Name this node...

If bot recognizes:
#Naming

Then check for:

Check for	Save it as	If not present, ask	Type
1 @Names	\$name	Sorry, I didn't get that.	Required

Add slot

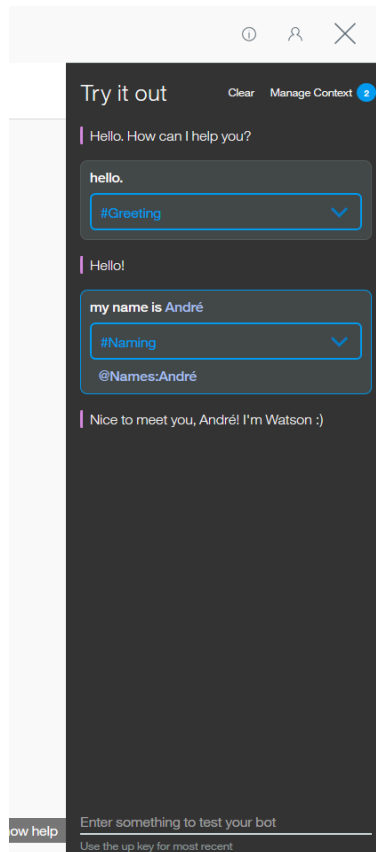
Then respond with:

1. Nice to meet you, \$name! I'm Watson :)

Add a variation to this response

Try it out

Go back to the chat window and clear it. Say “hello”, then tell Watson your name:



It works! 😊

As you can see, on the second message, “André” is highlighted because it matched the “@Names” entity.

Question: What would happen if you were to type “My name is Peter”, and “Peter” wasn’t in your list of names? Also, what could be done to fix this?

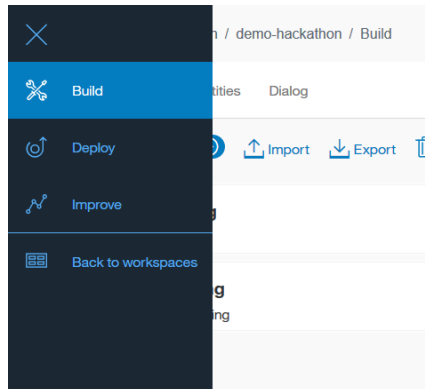
Recap

In this section, we used *Intents* and *Dialog* to create a bot that knows how to greet us. Then, we added an entity with names, so that Watson knows what *names* are. Finally, we combined all 3 things to make a bot that knows how to greet us, and our name.

Watson, what's the weather?

Time to make something a little more useful. We'll make a bot that can tell us the weather.

Let's go back to the workspaces. On the left menu, click the "Back to workspaces" button.



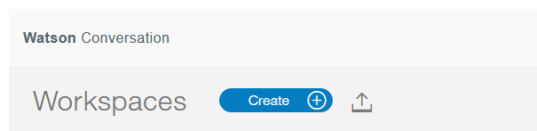
You'll be taken back to the workspaces page.

The workspace we're going to do will be very similar to what we did before, so I'll allow myself to say, "Here's one I prepared earlier", and cut some corners.

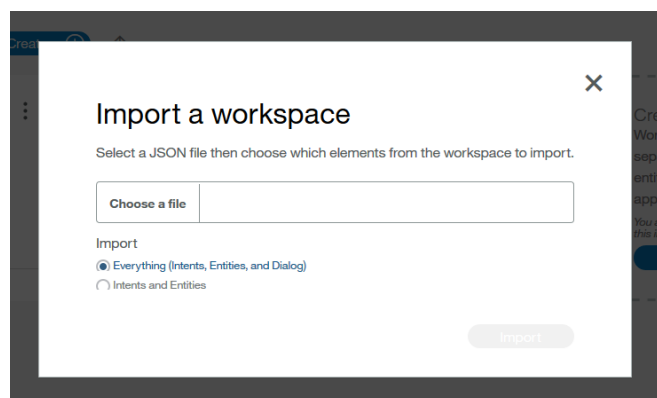
Importing an already existing workspace

There is a JSON file in the folder you cloned earlier, called "weather-workspace.json", in the "watson" folder. This file is for a workspace I created previously, and functions much like the same.

Next to the "Create" button, there is an upload button:



Click it, and on this window, choose the file to upload:



And choose to import everything.

This will add the workspace to the project. It's called "Weather". Open it.

Looking at the Weather workspace

This workspace has 3 intents defined:

- #Weather: For expressions like "What's the weather like"

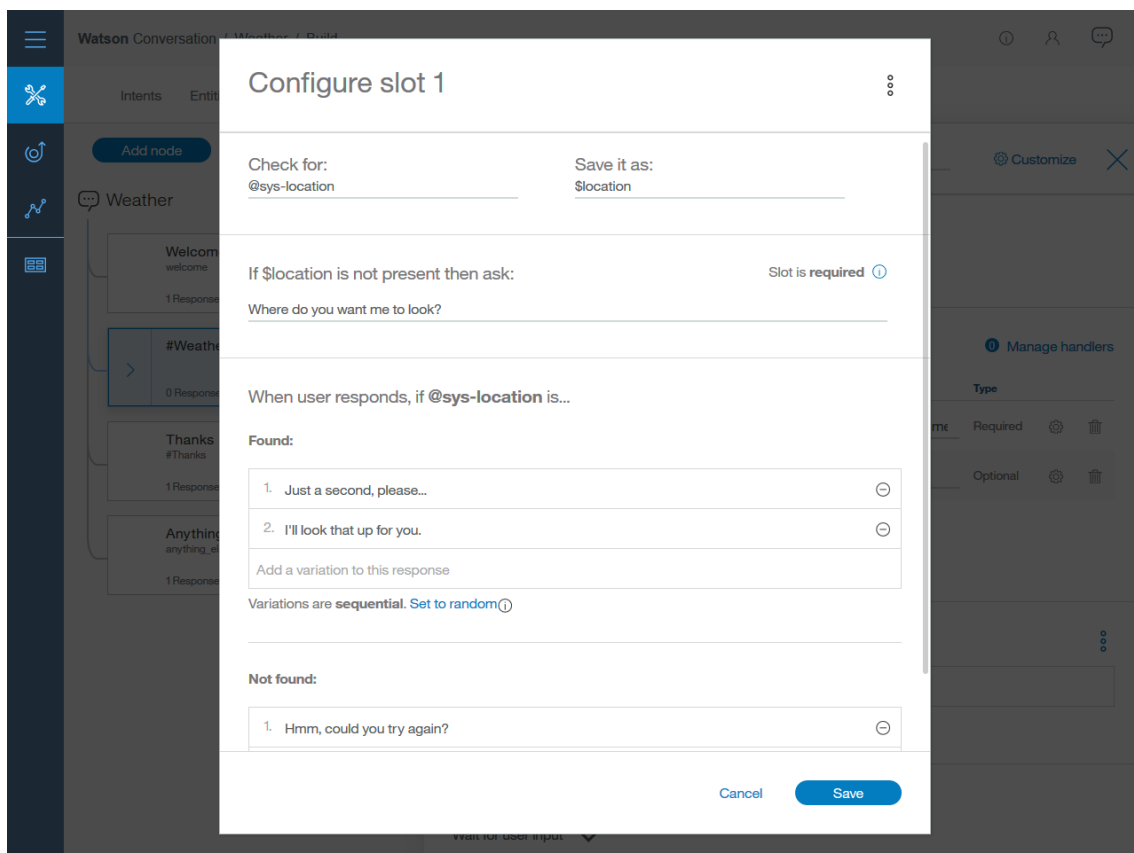
- #HowAboutIn: When the user wants to know about the weather in some other city.
- #Thanks: For the user to thank the bot for their service.

And it has 1 defined entity, which isn't used. It uses, however, 2 system entities:

- @sys-location: For detecting cities. Uses the city's English name (so, "Lisbon", instead of "Lisboa")
- @sys-time: For detecting time.

The dialog has the 2 sample nodes, a #Thanks node that only replies with "Glad I could help!", and a node that allows the user to ask for the weather (#Weather).

If you open the #Weather node, you can see that it uses the @sys-location (required) and @sys-time (optional) entities. If the user doesn't specify the location when asking, Watson will ask the user to input the city:



This means that the following scenarios both work:

Scenario 1: User asks for the weather, and tells the city as well

1. **User:** "What's the weather in Moscow?"
2. **Watson:** "Just a second, please..."
3. **Watson:** (Begin weather search for Moscow)

Scenario 2: User asks for the weather, but doesn't tell the city

1. **User:** "What's the weather?"
2. **Watson:** "Where do you want me to look?"
3. **User:** "Viseu"
4. **Watson:** "I'll look that up for you"

5. **Watson:** (Begin weather search for Viseu)

A small problem...

Watson's Conversation service alone can't be used to search for the weather. It can get us the *information we need* to search for the weather, however, and that's what we've done.

Using the *slots* feature, we create the context required to ask some third-party weather service for the weather (in this example, we have the location, and the time).

If we want to do more, we have to build an app. So, let's get to it!

The web application

Here's what we're going to build. In the interest of time, we're not going to build the app, but you can study the code and see what it does. There's also plenty of documentation and demos on the web on how to do something like this.

Hello. How can I help you?

hello. what's the weather in Lisbon?

Just a second, please...

It is currently 20 °C and Sunny in Lisbon, with a high of 18 and a low of 8.

This weather report courtesy of Yahoo! Weather and IBM Watson.

hmm, what about in Viseu?

Alright, I'll look that up.

It is currently 21 °C and Partly Cloudy in Viseu, with a high of 21 and a low of 6.

This weather report courtesy of Yahoo! Weather and IBM Watson.

quite nice weather, actually! thanks!

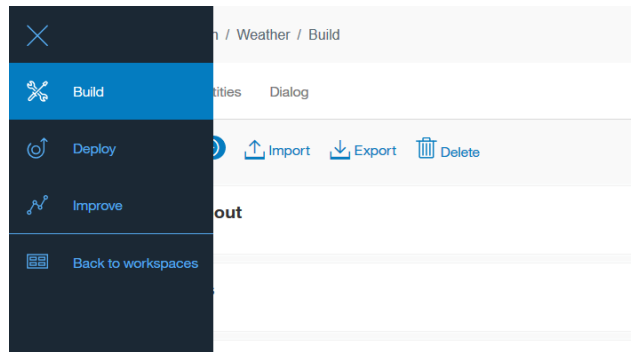
Glad I could help! 🌤️

I know it won't win any design awards, but it works!

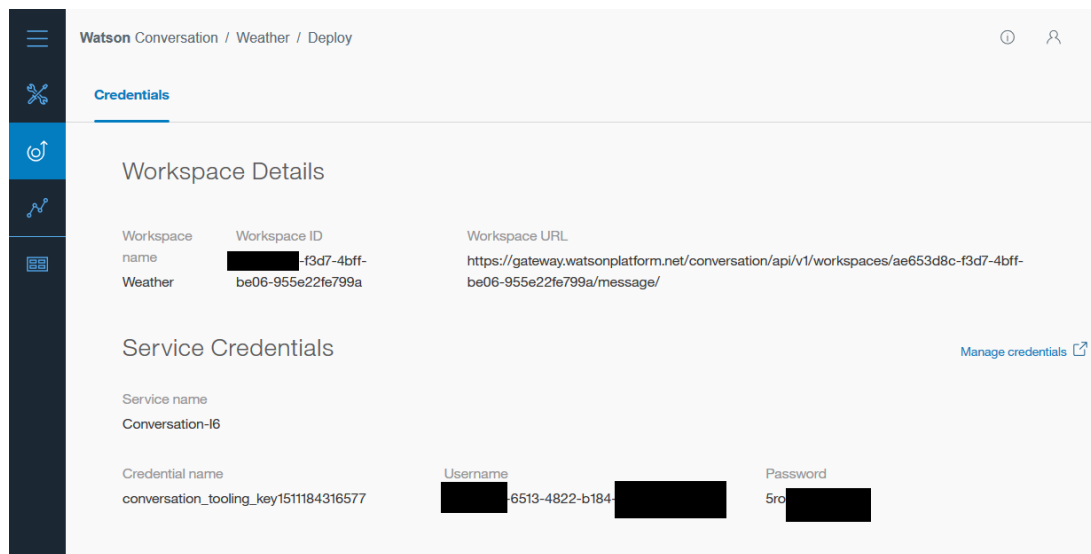
Let's go back to Visual Studio code, and get the app to run.

Setting up the Watson Conversation SDK

The Watson Conversation SDK requires authentication. In order to do this, go back to the Weather workspace we created earlier, and on the left, click on "Deploy".

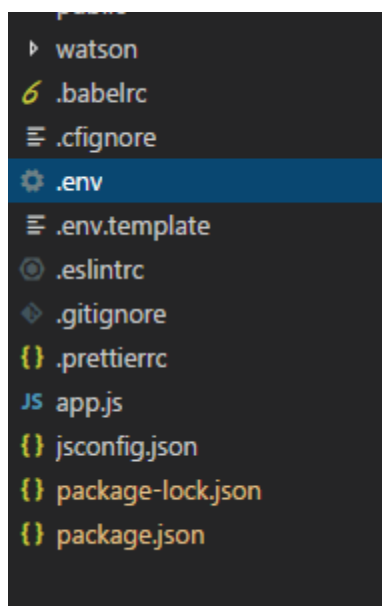


Here, you have the credentials you need to configure the SDK:

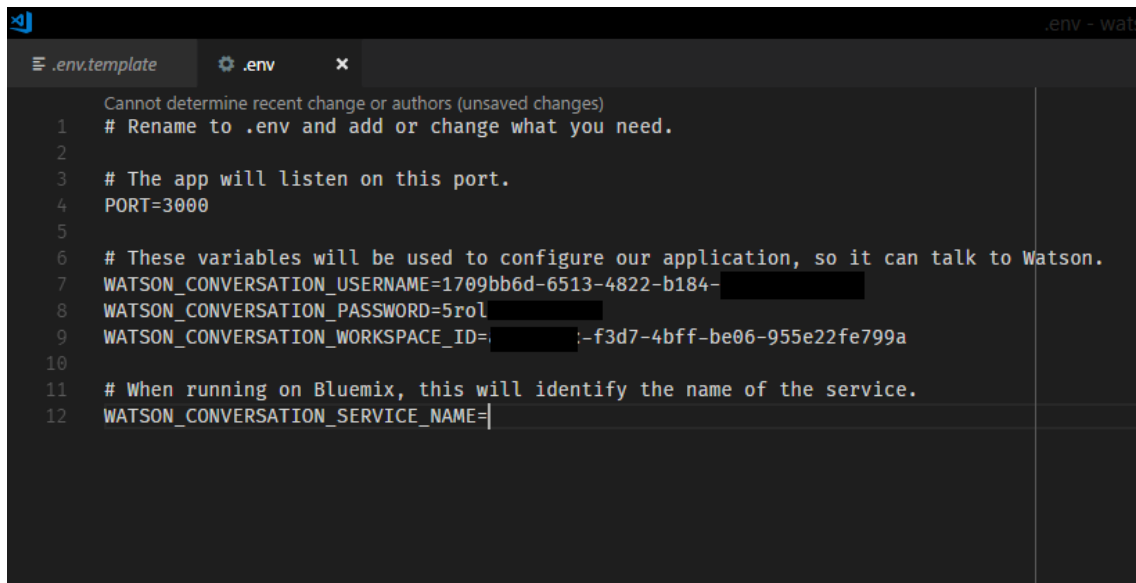


You'll need the Workspace ID, the Username, and the Password. I've hidden mine, so you can use yours.

Now, back on Visual Studio code, right-click the ".env.template" file, and make a copy. Then, rename the copy to be named ".env":



Finally, open the new file and edit it to add the workspace ID, the username, and the password. Should look like this (I've censored mine):



```

Cannot determine recent change or authors (unsaved changes)
1  # Rename to .env and add or change what you need.
2
3  # The app will listen on this port.
4  PORT=3000
5
6  # These variables will be used to configure our application, so it can talk to Watson.
7  WATSON_CONVERSATION_USERNAME=1709bb6d-6513-4822-b184-
8  WATSON_CONVERSATION_PASSWORD=5ro1
9  WATSON_CONVERSATION_WORKSPACE_ID=-f3d7-4bff-be06-955e22fe799a
10
11 # When running on Bluemix, this will identify the name of the service.
12 WATSON_CONVERSATION_SERVICE_NAME=

```

Running the application locally

Open the “app.js” file, and press F5. This will start debugging.

Now, go to <http://localhost:3000/>. You should see something like the above!

Running the app on Bluemix

Now that we have both things working, let’s get the app pushed onto Bluemix!

Creating the app

The process for creating an app is like the one for creating the service we created previously, so I’ll skip the images.

Go back to the Bluemix Dashboard, and create a resource. Search for “SDK for Node.js”, and select the one below “Cloud Foundry Apps”. The other option shows how to connect to a database.

Give the app a name, and a host name.

The host name must be unique! If another app already has that host name, Bluemix won’t let you continue. Click “Create” when you’re happy with what you have.

You should see something like this:

Cloud Foundry apps /

demo-hackathon-2 Running [Visit App URL](#)

Org: SoftINSA-Enterprise Location: United Kingdom Space: dev

Getting started tutorial

Last Updated: 2017-10-26 | [Edit in GitHub](#)

- Congratulations, you deployed a Hello World sample application on IBM® Cloud! To get started, follow this step-by-step guide. Or, [download the sample code](#) and explore on your own.

By following the Node.js tutorial, you'll set up a development environment, deploy an app locally and on IBM® Cloud, and integrate a IBM Cloud database service in your app.

Before you begin

You'll need the following accounts and tools:

- [IBM Cloud account](#)
- [Cloud Foundry CLI](#)
- [Git](#)
- [Node](#)

You can click on “Visit App URL” to see that it’s running!

Making Watson’s services available to our app

The process for getting Watson’s services available to our app locally was a bit manual. Thankfully, Bluemix helps us automate things a bit.

On the left, click “Connections”, then click “Create connection”:

Cloud Foundry apps /

demo-hackathon-2 Running [Visit App URL](#)

Org: SoftINSA-Enterprise Location: United Kingdom Space: dev

[Create connection](#)

Find the service you want to connect, hover the mouse over it, and click “Connect”:

Connect Existing Compatible Service

All Resources

SERVICES	RESOURCE GROUP	PLAN	SERVICE OFFERING
availability-monitoring-auto	--	Lite	Availability Monitoring
conversation-demo-hackathon	--	Lite	Conversation
Conversation-l6	--	Lite	Conversation
Monitoring and Analytics-83	--	Free	Monitoring and Analytics

Finally, restage the app. This allows Bluemix to reconfigure your app to get access to the service.

Finally, go to “Runtime” > “Environment Variables”, and add the following two variables on the section “User defined”

NAME	VALUE	ACTION
WATSON_CONVERSATION_SERVICE_NAME	conversation-demo-hackathon	
WATSON_CONVERSATION_WORKSPACE_ID	54d7e249-6e7f-4680-b6ba-81561c5c5c75	

WATSON_CONVERSATION_SERVICE_NAME	Equal to the name you gave to the Conversation service created earlier.
WATSON_CONVERSATION_WORKSPACE_ID	The Workspace ID you created by importing the JSON file.

Finally, click “Save”. Bluemix will restart the app.

Putting our code in Bluemix

Our app still says “Hello World!”. Let’s make it do something more interesting by adding the code we have.

Enable DevOps

Bluemix offers DevOps services that allow teams to collaborate in building Bluemix apps.

Go to the “Overview” tab on Bluemix, and at the bottom, click “Enable”.

IBM Cloud

Cloud Foundry apps /

demo-hackathon-2 Running [Visit App URL](#)

Org: SoftINSA-Enterprise Location: United Kingdom Space: dev

Current charges for billing period: Estimated total for billing period (Nov 1, 2017 - Nov 30, 2017)

Current and estimated cost excludes [connected services](#).

[View full usage details](#)

Activity feed

- started demo-hackathon-2 app**
Nov 20, 2017 5:32:15 PM | Andre.Carvalho@PT.softinsa.com
- stopped demo-hackathon-2 app**
Nov 20, 2017 5:32:14 PM | Andre.Carvalho@PT.softinsa.com
- updated demo-hackathon-2 app**
modified environment
Nov 20, 2017 5:32:13 PM | Andre.Carvalho@PT.softinsa.com
- restaged demo-hackathon-2 app**
Nov 20, 2017 5:27:57 PM | Andre.Carvalho@PT.softinsa.com

Continuous delivery

Continuous delivery is not enabled for this app.
Enable continuous delivery to automate builds, tests, and deployments through the Delivery Pipeline, GitHub, and more.

[Enable](#)

This might take a while, and a few prompts, but eventually you’ll get here:

Continuous Delivery Toolchain

This toolchain includes tools to develop and deploy your app. Depending on your app, when you create the toolchain, the Git repository will either be empty or will contain source code from your app.

This toolchain uses tools that are part of the Continuous Delivery service. If an instance of that service isn't already in the selected organization, when you click **Create**, it is automatically added with the free [Lite](#) plan selected.

To get started, click **Create**.

THINK **CODE** **DELIVER** **RUN**

ISSUE TRACKER → REPOSITORY → PIPELINE → IBM Cloud

WEB IDE

Toolchain Name:
demo-hackathon-2

Select location: US South

Choose an organization: SoftINSA-Enterprise

Tool Integrations

Create

Scroll down, and below where it says “Repository type”, choose “New” (because we already have code). Then, click “create”.

You should end up here:

Toolchains / demo-hackathon-2

demo-hackathon-2

THINK **CODE** **DELIVER**

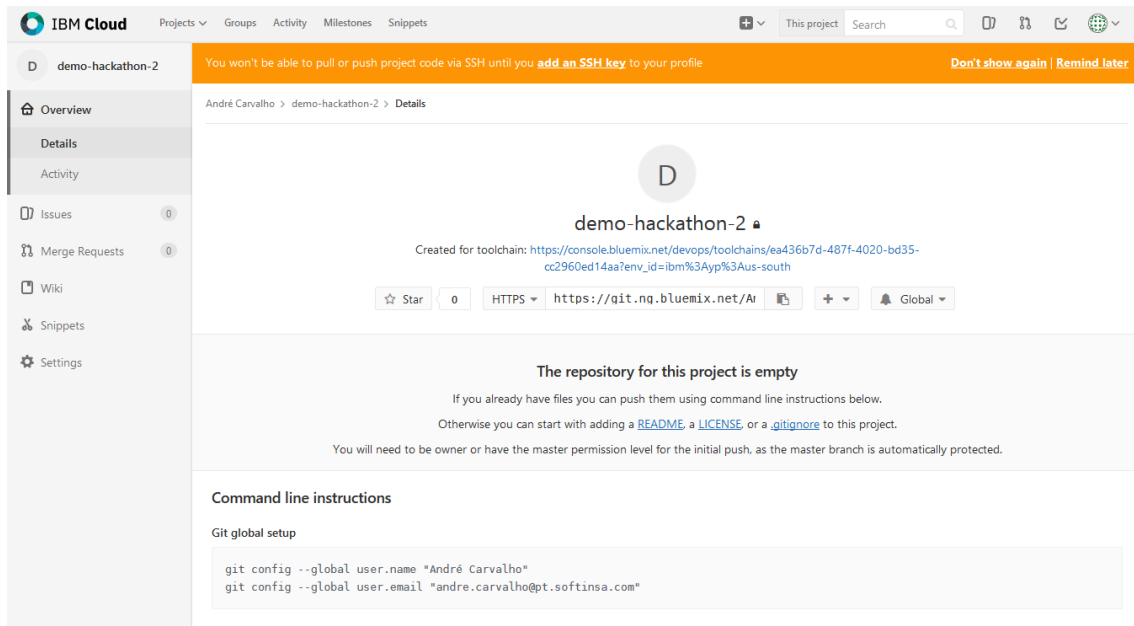
Issues demo-hackathon-2
✓ Configured

Git demo-hackathon-2
✓ Configured

Delivery Pipeline demo-hackathon-2
✓ Configured

Eclipse Orion Web IDE
✓ Configured

Here, click on the “Git” option. This will take you to your repository:



Configuring Git to push to our new repository

At the top of the page above, you have a URL. Copy it, you'll need it.

Then, go back to your terminal, and run the following commands:

```
git remote rm origin
```

```
git remote add origin https://git.ng.bluemix.net/<something>
```

```
git push -u origin master
```

If git asks you for your credentials, they'll be the ones you used to log in to Bluemix.

```
MINGW64:/c:/Users/0100975/Desktop/watson-demo-hackathon-salamanca
added 346 packages in 22.022s

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-hackathon-salamanca (master)
$ code .

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-hackathon-salamanca (master)
$ git remote rm origin

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-hackathon-salamanca (master)
$ git remote add origin https://git.ng.bluemix.net/Andre.Carvalho/demo-hackathon-2.git

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-hackathon-salamanca (master)
$ git push -u origin master
Counting objects: 63, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (55/55), done.
Writing objects: 100% (63/63), 35.55 KiB | 0 bytes/s, done.
Total 63 (delta 24), reused 0 (delta 0)
To https://git.ng.bluemix.net/Andre.Carvalho/demo-hackathon-2.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

0100975@PT0100975 MINGW64 ~/Desktop/watson-demo-hackathon-salamanca (master)
$
```

The app should eventually update. To see if it is, go back to Bluemix, and on the left menu, choose "DevOps", and choose your project. Then, click on the "Delivery Pipeline":

IBM Cloud

Toolchains / demo-hackathon-2 / demo-hackathon-2

demo-hackathon-2 | Delivery Pipeline

Build Stage STAGE PASSED

LAST INPUT: Git URL

JOB: Build Passed 3m ago

LAST EXECUTION RESULT: Build 1

Deploy Stage STAGE RUNNING...

LAST INPUT: Stage: Build Stage / Job: Build

JOB: Deploy Running

LAST EXECUTION RESULT: Loading...

Add Stage

If you're quick enough, you'll see that your app is deploying (like the image above). And when it's done...

Build Stage STAGE PASSED

LAST INPUT: Last commit by André Carvalho 2d ago

JOB: Build Passed 3m ago

LAST EXECUTION RESULT: Build 1

Deploy Stage STAGE PASSED

LAST INPUT: Stage: Build Stage / Job: Build

JOB: Deploy Passed now

LAST EXECUTION RESULT: demo-hackathon-2 demo-hackathon-2.ej-gb.mybluemix...

Add Stage

We're up!

Bluemix DevOps is configured by default for you so that you can push with Git, and it'll take care of building and deploying your app! You can even add stages to do testing, and multiple deployments!

Collaboration

If you go back to your app's project, you can click on "Manage" on the left. Here, you can add users that can push code to this project:

Overview

Connections

Manage

Toolchains / demo-hackathon-2

demo-hackathon-2

Access Control

The following users can view this toolchain. Org managers can always modify or delete the toolchain.
Note: If you grant access to the org, all org members can access the toolchain.

Enter a user ID

Add user

Add org

TYPE	NAME	ADMIN
Org	SoftINSA-Enterprise	<input checked="" type="checkbox"/>

Save Reset

This means that multiple people can collaborate on the same project! 😊

(This page intentionally left blank)