

Backoffice HealthCheck Bot Workshop using Microsoft's Bot Framework

DYI Workshop Documentation | Cognitive Bots

Jared Perry

Sr. Researcher – Cloud and Cognitive
Miracle Software Systems, Inc.

April 6th, 2017

Microsoft Bot Framework Workshop

This document contains a step-by-step guide to take you through the various options with the Microsoft Bot Framework and will teach you how to create a NLU Bot that can handle System Health Check tasks for you.

This guide was prepared by **Miracle's Innovation Labs** 😊

Pre-Reqs and Installations

All attendees must have their own workstation(with Internet) to participate in the lab(Both PC and Mac are compatible). The following installation pre-requisites will help to make the hands-on lab experience easier.

- Create a **Microsoft Account**(Will be needed for using the Bot Framework and LUIS)
- Install the following items and ensure that you are able to access them from your command line,
 - **Node JS** - <https://nodejs.org/en/download/>
 - **NPM** – Should come along with Node JS
 - **Mongo DB** - <https://www.mongodb.com/download-center?jmp=nav#community>
 - **Git(Optional)** - <https://git-scm.com/downloads>
 - **ngrok** - <https://ngrok.com/download>
- Create your own Slack Organizations to be able to add bots - <https://slack.com/get-started>

Let's Get Started!

The following steps will outline how you can create a HealthCheck Bot and integrate it into Slack. Users will be able to directly message your bot and perform operations such as running a new health check.

To start it off please download the following repository from GitHub.

<https://github.com/MiracleLabs/ms-bot-framework-workshop>

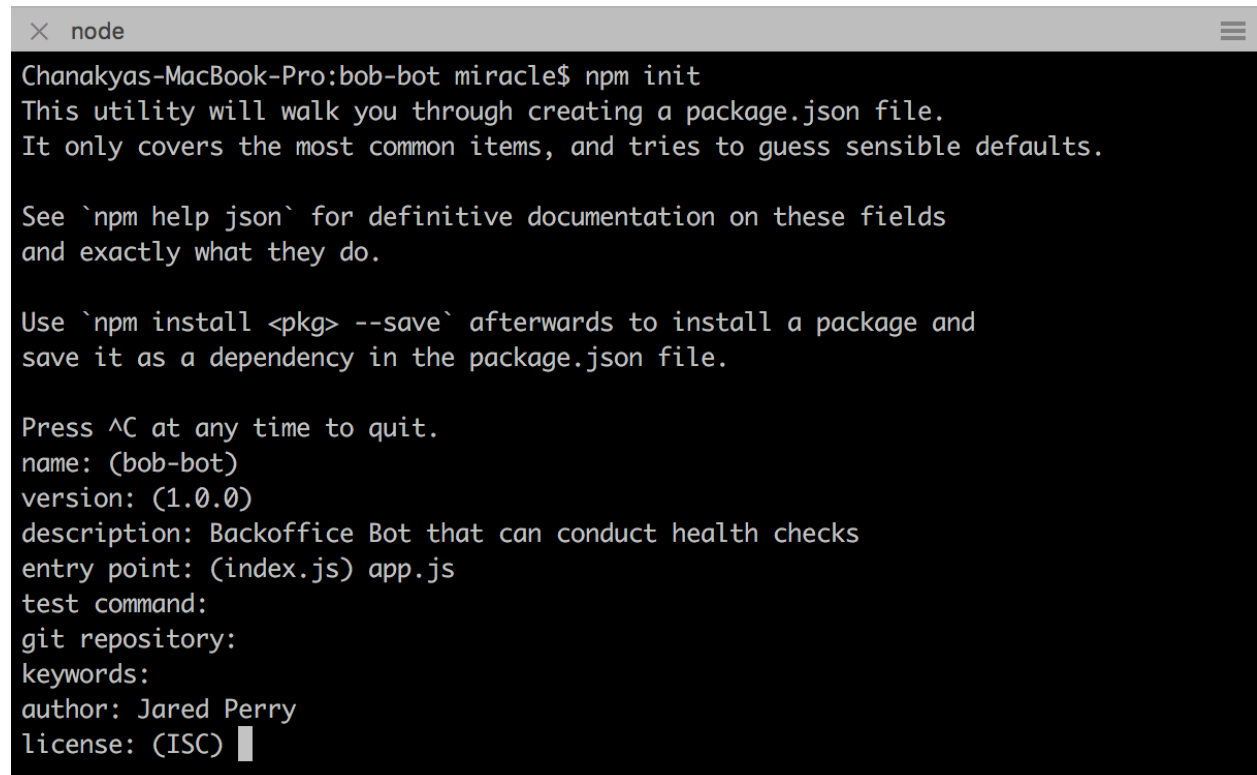
Lab #1 – Creating a Static Bot with MS Bot Framework

The first step will be to use the Bot Builder SDK for Node JS and build a static backend that can respond to user messages. We will then first simulate the bot with the Bot Emulator and will then register your bot with the bot framework.

Download the **Bot Emulator** here : <https://emulator.botframework.com/>

Step #1 – Create Bot Backend and Test with Emulator

Create a folder for your bot backend and then navigate into that folder. Initialize your node project using the `npm init` command.

A terminal window titled 'node' showing the output of the 'npm init' command. The text is as follows:

```
Chanakyas-MacBook-Pro:bob-bot miracle$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (bob-bot)
version: (1.0.0)
description: Backoffice Bot that can conduct health checks
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Jared Perry
license: (ISC)
```

Add the required node modules using the following command,

```
npm install -save botbuilder restify moment
```

Create an **app.js** file and paste the following code to handle the user's messages and send the responses.

```
//Add the modules that are required
var restify = require('restify');
var builder = require('botbuilder');
var moment = require('moment');

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978,
function () {
    console.log("-----
-----");
    console.log(moment().format('MMMM Do YYYY, hh:mm:ss a') + " |
Backoffice Bot is running with the address : "+server.url);
    console.log("-----
-----");
});

// Create chat bot
var connector = new builder.ChatConnector({
    appId: "",
    appPassword: ""
});

var bot = new builder.UniversalBot(connector);
server.post('/api/messages', connector.listen());

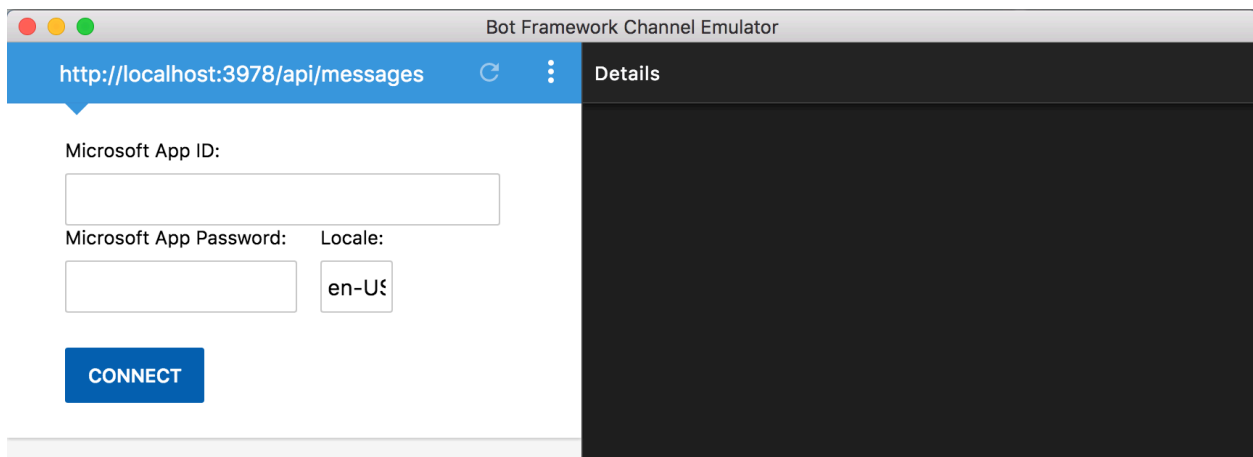
//=====
// Bots Dialogs
//=====

bot.dialog('/', function (session) {
    session.send("Hello World");
});
```

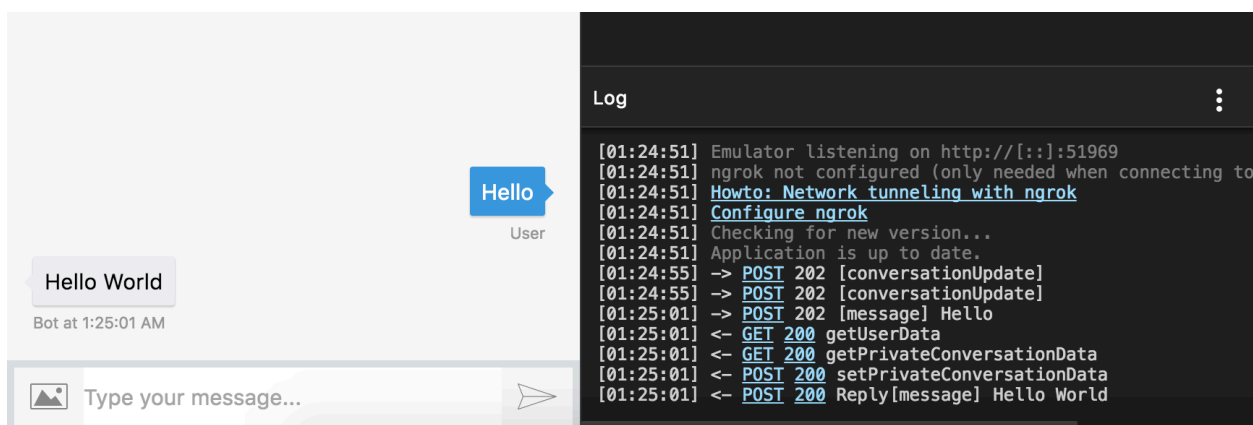
Run your backend by using the following command, **node app.js**

```
node
Chanakyas-MacBook-Pro:~$ node app.js
-----
April 7th 2017, 01:17:50 am | Backoffice Bot is running with the address : http://[::]:3978
-----
```

You can then open the Bot Emulator and configure your endpoint as follows.

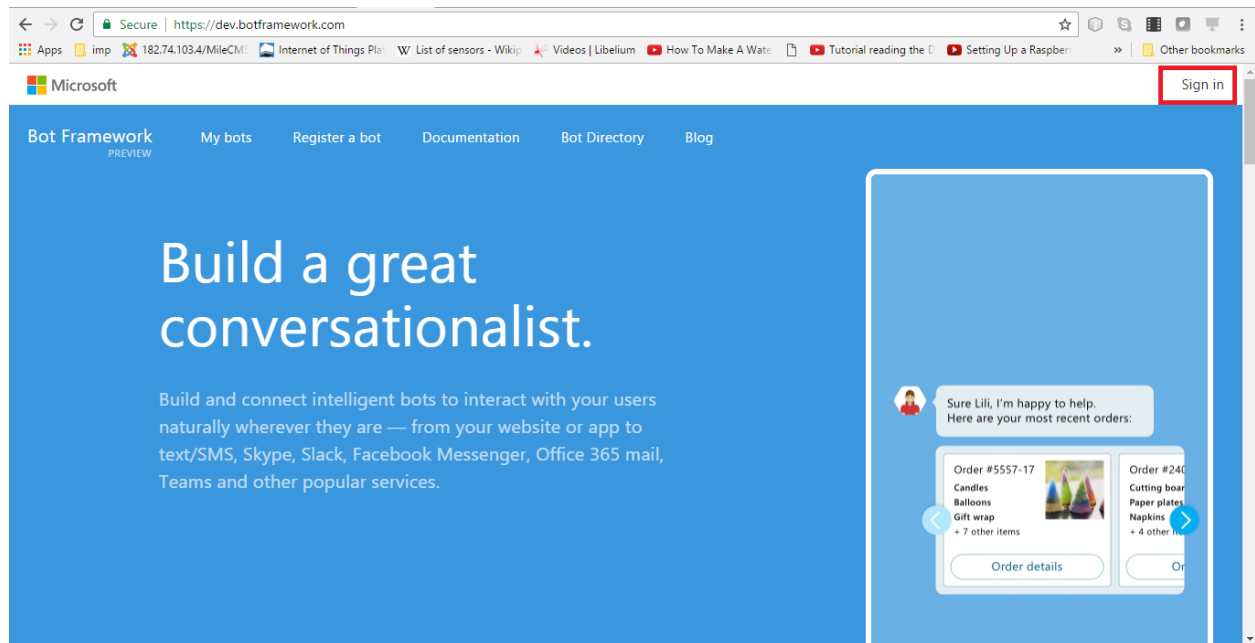


You can then send a message and receive the “Hello World” response.

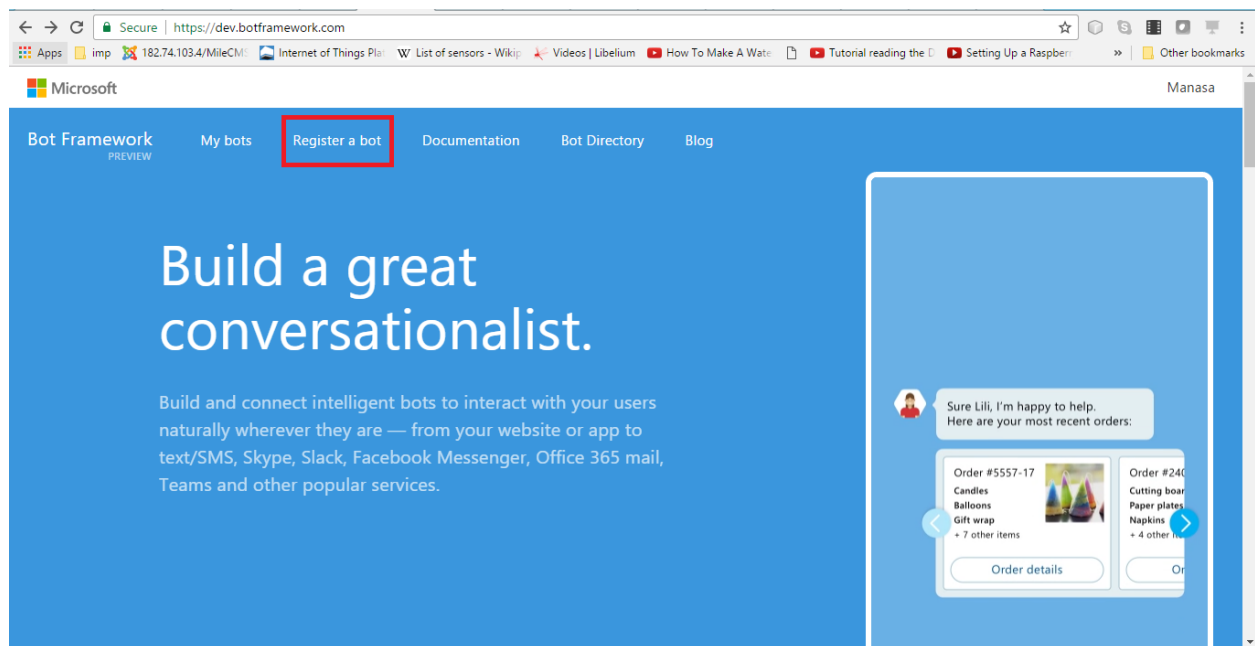


Step #2 – Register and Test with the Bot Framework

Login at <https://dev.botframework.com/> using your Microsoft ID.



Once you are signed in, click on “Register a Bot” to register a new bot.



Fill in the required details as shown below.

Tell us about your bot

Bot profile



Icon

[Upload custom icon](#)
30K max, png only

Name: * ?

Bot handle: * ?

Description: * ?

1

Create your new APP_ID and APP_PWD.

Configuration

Messaging endpoint:

Register your bot with Microsoft to generate a new App ID and password

[Create Microsoft App ID and password](#)

Paste your app ID below to continue

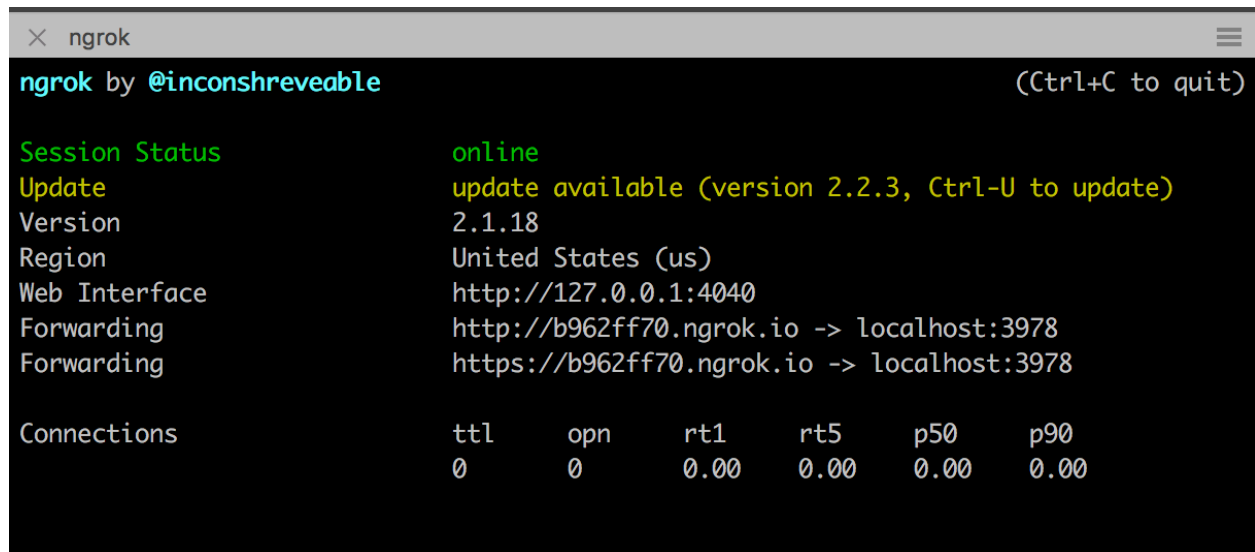
Configure your Bot credentials in your app.js file as shown below.

```
var connector = new builder.ChatConnector({
  appId: "<app_id>",
  appPassword: "<app_pwd>"
});
```

Now go ahead and run your backend once again with the below command and then open another terminal and start a ngrok tunnel for port 3978.

```
node app.js
```

```
ngrok http 3978
```

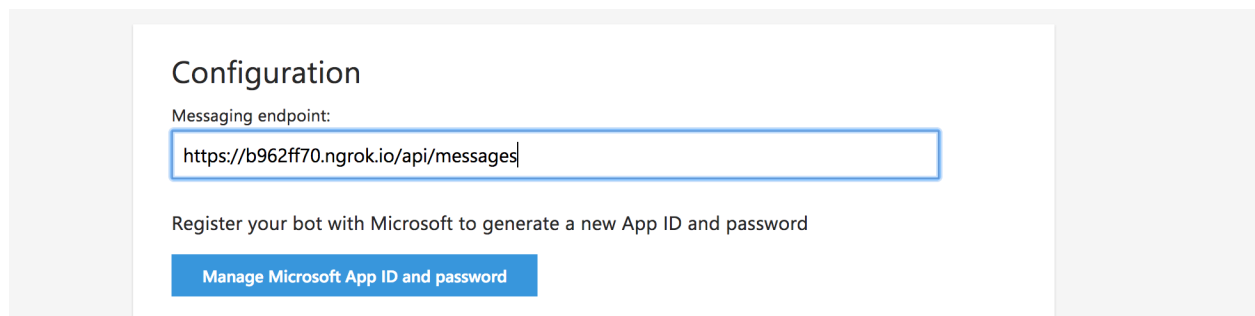


```
ngrok by @inconshreveable (Ctrl+C to quit)

Session Status      online
Update              update available (version 2.2.3, Ctrl-U to update)
Version             2.1.18
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://b962ff70.ngrok.io -> localhost:3978
Forwarding           https://b962ff70.ngrok.io -> localhost:3978

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
```

Copy the above HTTPs Forwarding URL and add **/api/messages** to the end. Configure the messaging endpoint in the Bot Framework registration.



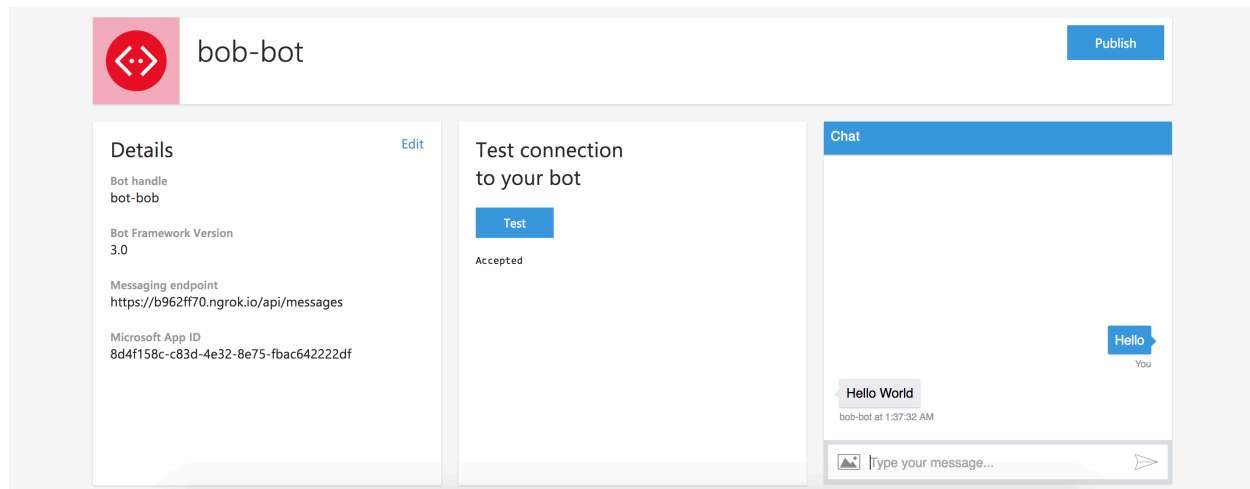
Configuration

Messaging endpoint:

Register your bot with Microsoft to generate a new App ID and password

[Manage Microsoft App ID and password](#)

Go ahead and save the registration, click on Test Connection and then test it out.



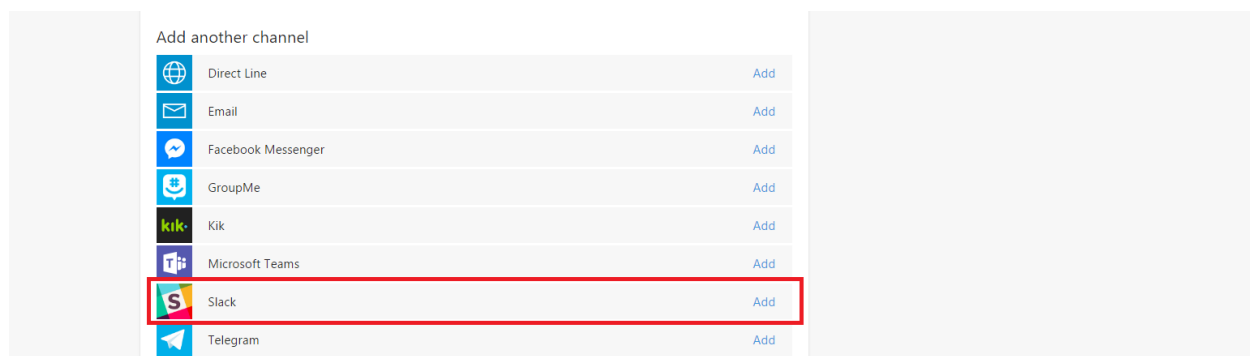
Lab #2 – Integrating your bot with Slack

Once we have a sample bot up and running, we can now integrate with the channel of our choice – in this lab we will see how you can integrate your bot with Slack.

Step #1 – Configure Slack in Bot Framework Channels

Note : Please create your own Slack Team before you start this lab.

In new window, go to your My Bots page on dev.botframework.com, select the bot you created in Lab #1, and scroll down to “Add another channel”.

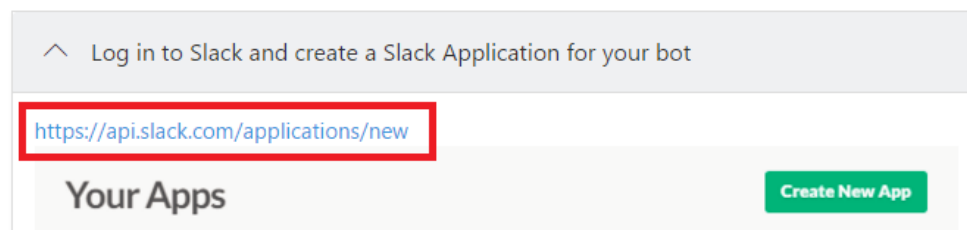


Click on Add for Slack, and follow the first link for creating a Slack Application for your bot.

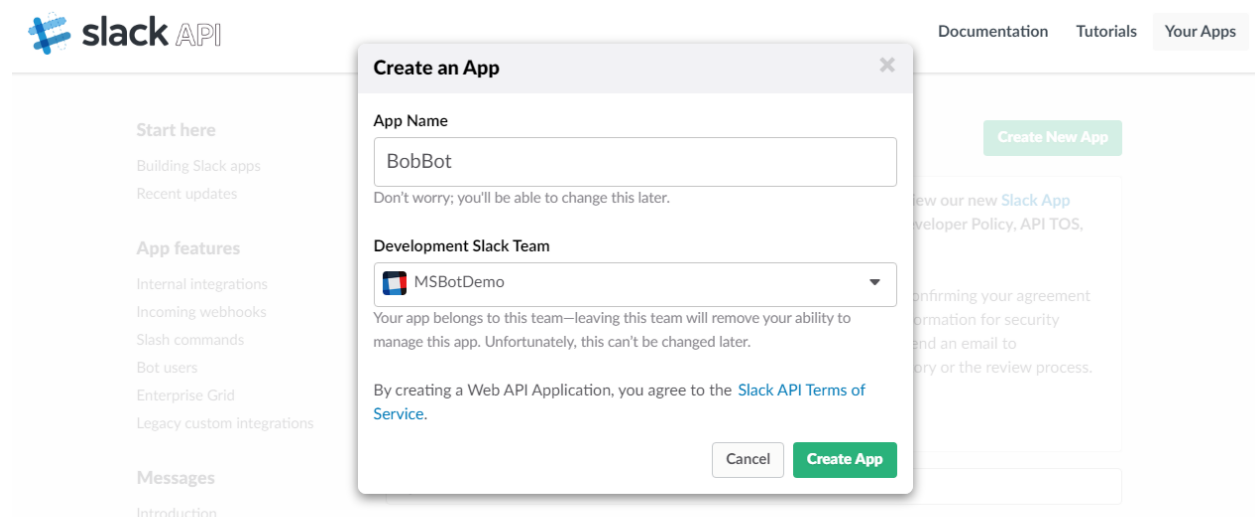
Configure Slack



How to



Click **Create New App**, name your application, select the Slack team you created, and press **Create App**.



You will then be forwarded to your newly created Slack app. From here select OAuth and Permissions under Features, and add the following URL link to your Redirect URLs : <https://slack.botframework.com>.

The screenshot shows the 'OAuth & Permissions' settings page for a bot named 'BobBot'. On the left sidebar, under the 'Features' section, 'OAuth & Permissions' is highlighted with a red box. The main content area is titled 'OAuth & Permissions' and contains a section 'OAuth Tokens & Redirect URLs'. Below this, there is an 'Install App to Team' button. Further down, the 'Redirect URLs' section explains that redirect URLs are needed for automatic Slack integration and lists a URL: 'https://slack.botframework.com'. Below the URL list, there is a red box around the 'Add a new Redirect URL' button, and a green 'Save URLs' button at the bottom.

Then select **Bot Users** under Features, and click **Add Bot User**.

The screenshot shows the 'Bot User' settings page for 'BobBot'. In the left sidebar, under the 'Features' section, 'Bot Users' is highlighted with a red box. The main content area is titled 'Bot User' and contains a description of bot users. Below the description, there is a 'Default username' field with the value '@bobbot'. A note below the field states that if the username is unavailable, it will be changed. There is also a toggle switch for 'Always Show My Bot as Online', which is currently set to 'Off'. At the bottom, there is a green 'Add Bot User' button.

You will then need your credentials of your bot, to find these click Basic Information under Settings and scroll down to your App Credentials, copy both your Client ID and Client Secret and save them in a text document.

BobBot

Settings

Basic Information

Collaborators

Install App

Manage Distribution

Features

Incoming Webhooks

Interactive Messages

Slash Commands

OAuth & Permissions

Event Subscriptions

Bot Users

Slack

Help

Contact

Policies

Our Blog

Basic Information

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

Install your app to your team

Manage Distribution

App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

Client ID

165054398673.165860842133

Client Secret

ddf5bd774d7d8166026dd964a691ef91

Regenerate

You'll need to send this secret along with your client ID when making your [oauth.access](#) request.

Now, redirect to the page where you added Slack channel, and click on **Submit Your credentials**. Paste Client ID, Client Secret then click **Submit Slack credentials**.

Gather your Credentials

Submit your Credentials

Client Id

jds2ddasd323asdfvd3234325af

Client Secret

Bcas389asdc89asdc89s

Verification Token

HMK9cgaPSrt0hOntqInNaRR (optional)

Landing Page URL

https://yourbot.example.com/slack_help (optional)

Users will be redirected to this URL after adding your bot to Slack.

Submit Slack Credentials

☐ Enable this bot on Slack

Click on **Authorize** button, then it will redirect previous page and click on **I'm done Configuring Slack**.



BobBot would like access to MSBotDemo

This will allow BobBot to:

Confirm your identity on MSBotDemo

⚠ Add a bot user with the username @bobbot Show more

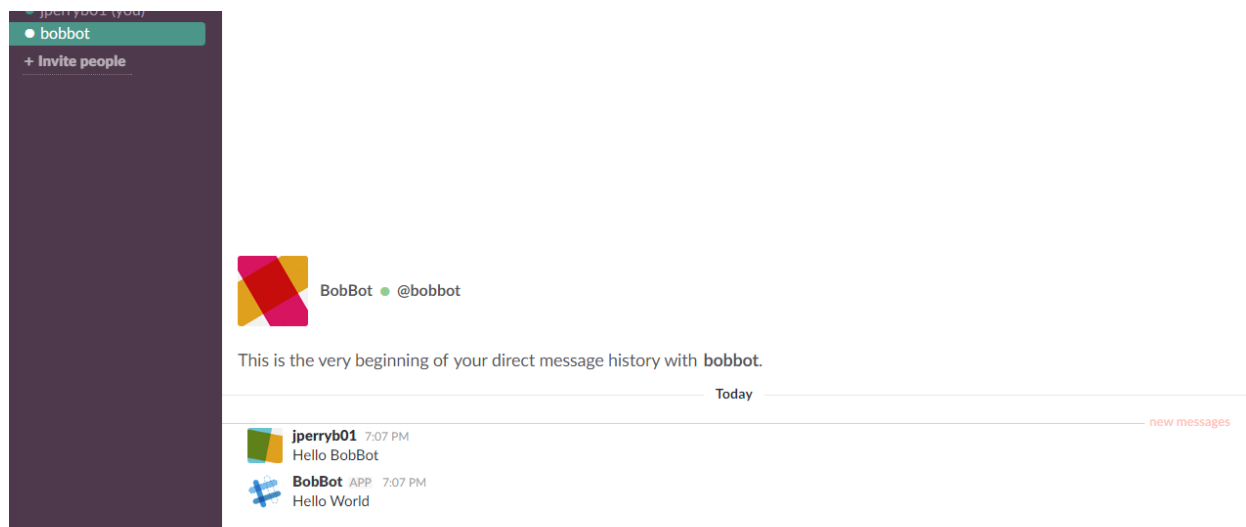
Please only share your team's private information with apps that you have reviewed and trust.

Authorize

Cancel

Step #3 – Test the bot with Slack

To verify your bot was created successfully, go back to your slack channel and under Direct Messages you should see the bot you created is now there. Open Slack, and navigate to bobbot and enter Hello bobbot, it will give response as **Hello World**.



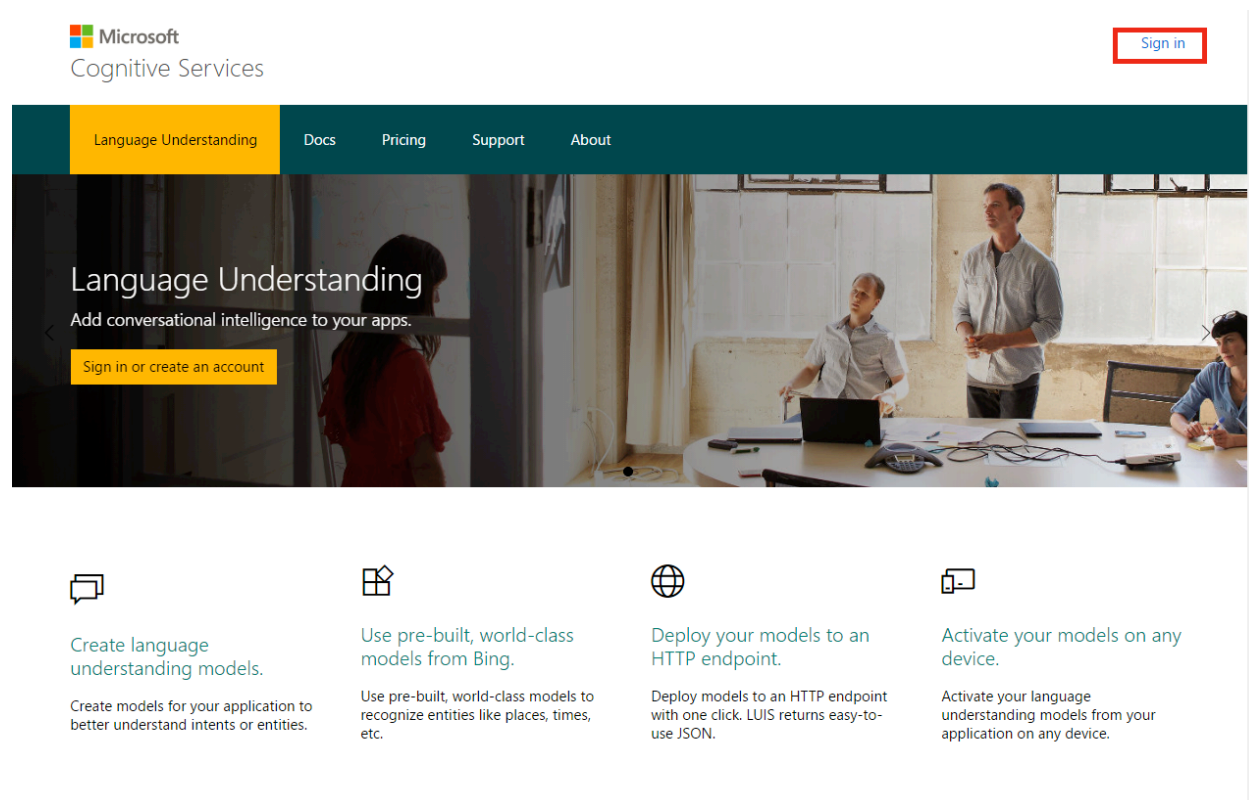
Lab #3 – Creating your LUIS Model

Our bot is pretty cool now, but it would be cooler if we can add Natural Language Understanding capabilities. In this lab you will add intents and entities to your LUIS Model and test them out.

Step #1 – Create Your Entities and Intents

First, head over to the LUIS homepage at **www.luis.ai** where you will be making your dialog model.

Log in using your Microsoft account, and after a few moments your workspace will be configured. You will be prompted to enter information about yourself, and agree to the Terms of Service.



Once you have accepted, you will be forwarded to your My Apps page where you will create your Natural Language dialog model.

My Apps

Create and manage your LUIS applications ... [Learn more](#)

New App

Import App

Cortana prebuilt apps ▾

Name ↓

Culture

Created date

Endpoint hits

Controls

No applications yet in your account. Start by building a new application or import a JSON file of an existing application.

First click on New App, input a name, and then press Create.

You will then find yourself at the Dashboard for your newly created app, and can start adding Intents and Entities that form the framework of your bots' conversation.

Your first step will be to create and train an Intent. You can think of an intent as an action your end user wishes to carry out that is processed and executed by your bot, either by replying to the user with relevant dialog and information, or triggering your backend application to perform certain task.

To begin creating an intent, click on the Intent tab under your apps dashboard, and select Add Intent

BobBot

Dashboard

Intents

[Entities](#)

[Features](#)

[Train & Test](#)

[Publish App](#)

[← Back to App list](#)

Dashboard (App Id: 1102f4c8-9e05-408f-97a7-7f2e235a13a7)

Facts & statistics about the app's data and the received endpoint hits at any period of time ... [Learn more](#)

You have no intents yet. Intents are the building blocks of your app; they link user requests with the actions that should be taken by your app. Get started by creating your first intent.

[Create an intent](#) [Next tip](#)

App status Last train: Not trained yet Last published: Not published yet

Intent Count	Entity Count	Prebuilt Entity Count	Labeled Utterances Count
1 / 80	0 / 30	0 / 13	0

Endpoint Hits Per Period <small>PER DAY (LAST WEEK)</small>	Total Endpoint Hits <small>SINCE APP CREATION</small>
No endpoint hits or utterances to show.	0
	Key Usage

We will begin by adding a simple intent that responds with the capabilities of the bot when the end user is facing an issue or has an off topic question. Give the name **intent.Capabilities** to describe what the intent does, and click save.

Add Intent

Intent name (REQUIRED)

[Save](#) [Cancel](#)

You will then need to train your Capabilities intent on example inputs or utterances the user might enter when wanting to know what your bot can do. Try adding a few example like

- What are your capabilities?
- What can you do
- Help
- Tell me about yourself


Once you have a few examples, click save to add your inputs to the training data used to make the model for this intent.


intent.Capabilities


Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (1) Entities in use Suggested utterances



Type a new utterance & press Enter ... ×

 Save

 Discard

 Delete

Labels view (Ctrl+E): Entities ▼

Search in utterances ...  

Reassign Intent ▼

<input type="checkbox"/>	Utterance text	Predicted Intent
<input checked="" type="checkbox"/>	tell me about yourself	Not trained
<input checked="" type="checkbox"/>	help	Not trained
<input checked="" type="checkbox"/>	what are your capabilities	Not trained
<input checked="" type="checkbox"/>	what you can do ?	Not trained

1

Now it is time to create your first entity. An entity can be thought of as a label for an object, topic, or key identifier for something specific an end user wants to perform an action on.

We will be creating a Hierarchical entity that contains multiple items (called Childs) that are related to the single entity type.

First navigate to the Entities tab under the Dashboard for your application

BobBot

Dashboard

Intents

Entities

Features

Train & Test

Publish App

← Back to App list

Entities

Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)

Entities list Labeled utterances Suggested utterances

Add custom entity

Add prebuilt entity

Entity Name ↓

Entity Type

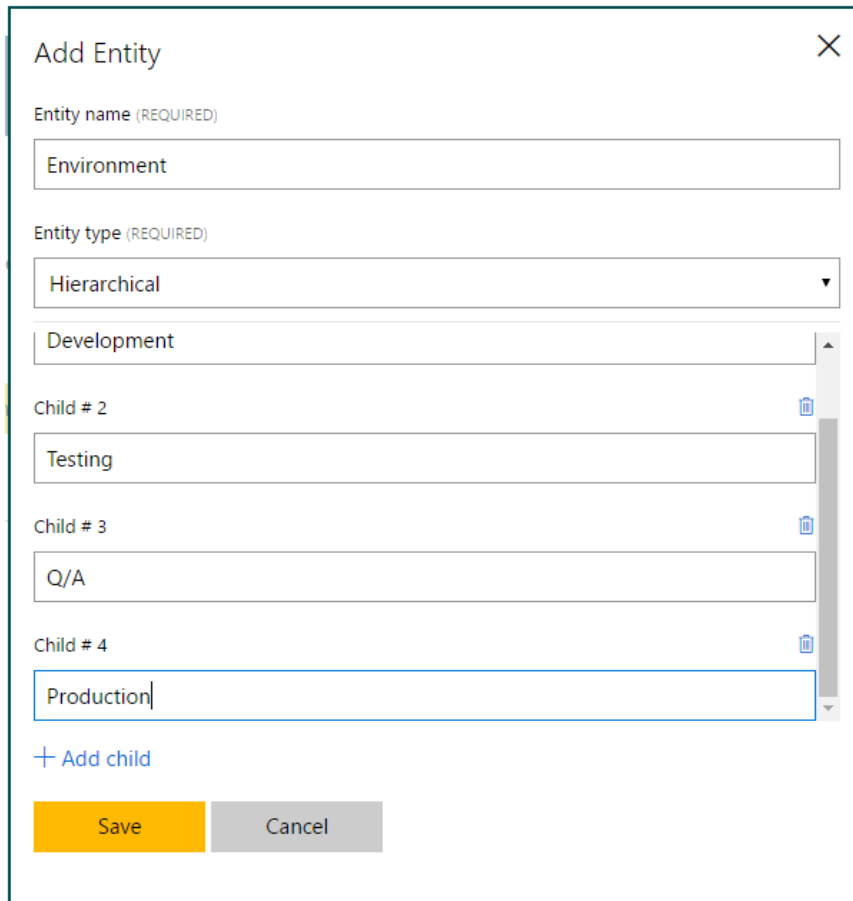
No entities yet. Add your first entity now.

Next, click Add custom entity, and give it the name “Environment”.

Then from the drop down menu under Entity type, select Hierarchical and add four Childs. Name them as follows,

- Development
- Testing
- Q/A
- Production

When finished your screen should look similar to this,



Add Entity

Entity name (REQUIRED)

Environment

Entity type (REQUIRED)

Hierarchical

Development

Child # 2

Testing

Child # 3

Q/A

Child # 4

Production

+ Add child

Save Cancel

To finish your custom entity, click Save and LUIS will add it to your application.

Now that you are familiar with creating an Intent and Entity, import our finished dialog model to LUIS where you will train and publish the finished app and integrate it into your backend.

<https://github.com/MiracleLabs/ms-bot-framework-workshop/bob/dialog/bob-dialog.json>

To add the finished model, first save the **bob/dialog/bob-dialog.json** file from our Github repository to your desktop. Then click on My Apps on the top pane, click Import App, and upload the file you saved to your desktop.

My Apps

Create and manage your LUIS applications ... [Learn more](#)

New App Import App Cortana prebuilt apps

Name	Culture	Created date	Endpoint hits	Controls
BobBot	en-us	Apr 7, 2017, 11:48:48 AM	0	Edit Refresh Delete

Step #2 – Train, Test and Publish your LUIS App

Now that you have a completed LUIS model, you will need to train your application on your example utterances to create the model used for identifying your Intents. To do this, select Train and Test under your apps dashboard, and click Train Application.

Language Understanding My apps My keys Docs Pricing Support About

backoffice-bot

- Dashboard
- Intents
- Entities
- Features
- Train & Test**
- Publish App

< Back to App list

Test your application

Use this tool to test the current and published versions of your application, to check if you are progressing on the right track ... [Learn more](#)

Train Application Please train your application before testing.

Interactive Testing Batch Testing

☐ Enable published model Labels view (Ctrl+E) Entities Reset console

Please train your application before testing.

Train

LUIS will then create the model used for handling Natural Language. Make note that you will need to re-train your model when entering new intents, entities, or example utterances to have your bots dialog reflect your new changes.

Once the training is completed, try entering the below sentences in the Interactive Testing window to see how LUIS classifies intents and labels your entities,

- Hi, how are you?
- What are your capabilities?
- Run a health check on all IIB systems on the dev environment.

You should see the top scoring intent for each utterance and your entity keywords labeled with the entity name they fall under.

The last step in creating your LUIS model is to publish your App and create the endpoint URL for integrating into your backend.

To do this, click on Publish App under the Dashboard, select the pre-made BootstrapKey in the drop down box under Endpoint Key, and click Publish.

backofficebot

Dashboard
Intents
Entities
Features
Train & Test
Publish App

← Back to App list

Publish App

Publish your app as a web service or as a chat bot. You can publish a new app or an updated version of a published app ... [Learn more](#)

Essentials

Latest publish: You haven't published your application yet

Endpoint Key (REQUIRED)

BootstrapKey

This key is for experimental use and is valid for 30 days and 1000 endpoint hits only.
[Add a new key to your account](#)

Publish settings

Endpoint slot

Production

This slot has no published application.

Train Publish

English (United States) Privacy & Cookies Terms of use Developer Code of Conduct Trademarks © 2016 Microsoft

After a few moments your application should be published, and you will be given an Endpoint URL. Make note of your URL as later labs will need it for integrating your LUIS dialog model.

Lab #4 – Adding Basic Dialog Matching Capabilities

Once we have a LUIS Model we can now configure our backend to be able to handle these user intents. We will add a Recognizer to our code and create dialogs that match and respond to the user.

Step #1 – Add LUIS Recognizer to your Bot Backend

Navigate back to your backend project folder and run the following command,

```
npm install -save luis-sdk
```

Go back to your **app.js** file and replace the file with the following code,

```
//Add the modules that are required
var restify = require('restify');
var builder = require('botbuilder');
var LUIS = require('luis-sdk');
var moment = require('moment');

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978,
function () {
    console.log("-----
-----");
    console.log(moment().format('MMMM Do YYYY, hh:mm:ss a') + " |
Backoffice Bot is running with the address : "+server.url);
    console.log("-----
-----");
});

// Create chat bot
var connector = new builder.ChatConnector({
    appId: "<bot-app-id>",
    appPassword: "<bot-app-pwd>"
});
```

```
var bot = new builder.UniversalBot(connector);

var model = '<luis-model-publish-url>';
var recognizer = new builder.LuisRecognizer(model);
var dialog = new builder.IntentDialog({ recognizers: [recognizer]
});

server.post('/api/messages', connector.listen());

//=====
// Bots Dialogs
//=====

bot.dialog('/', dialog);
```

Note : Configure the above code with your LUIS and Bot Framework Credentials

Step #2 – Handle Basic Dialog Flows

We can now add a recognizer to match intents and respond back to user. **First add a default handler for any intents that are not recognized.**

```
dialog.onDefault(builder.DialogAction.send("I'm sorry I didn't
understand. I'm the Backoffice Bot, how can I help you? "));
```

Then add a handler for the intent 'intent.Greetings'.

```
dialog.matches('intent.Greetings',[
  function(session,args){
    session.send("Hi, I'm Bob! How can I help you today?");
  }
]);
```

You can add more handlers for your other intents as well. After you have added all of the one-step dialogs your app.js file should look like this.

```
//Add the modules that are required
var restify = require('restify');
```

```
var builder = require('botbuilder');
var LUIS = require('luis-sdk');
var moment = require('moment');

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978,
function () {
    console.log("-----
-----");
    console.log(moment().format('MMM Do YYYY, hh:mm:ss a') + " |
Backoffice Bot is running with the address : "+server.url);
    console.log("-----
-----");
});

// Create chat bot
var connector = new builder.ChatConnector({
    appId: "<bot-app-id>",
    appPassword: "<bot-app-pwd>"
});

var bot = new builder.UniversalBot(connector);

var model = '<luis-model-publish-url>';
var recognizer = new builder.LuisRecognizer(model);
var dialog = new builder.IntentDialog({ recognizers: [recognizer]
});

server.post('/api/messages', connector.listen());

//=====
// Bots Dialogs
//=====

bot.dialog('/', dialog);

dialog.matches('intent.Greetings',[
    function(session,args){
```



```
        session.send("Hi, I'm Bob! How can I help you today?");
    }
});

dialog.matches('intent.Capabilities',[
    function(session,args){
        session.send("I can help with LOE calculations, health checks,
on-boarding and more!");
    }
]);

dialog.matches('intent.ThankYou',[
    function(session,args){
        session.send("Your welcome, have a great day!");
    }
]);

dialog.matches('intent.NoHelp',[
    function(session,args){
        session.send("Have a great day!");
    }
]);

dialog.onDefault(builder.DialogAction.send("I'm sorry I didn't
understand. I'm the Backoffice Bot, how can I help you? "));
```

You can now run your code and ngrok once again and test the messages.

The screenshot displays the Microsoft Bot Framework console interface for a bot named "bob-bot". The interface is divided into three main sections:

- Details:** This section on the left provides information about the bot, including the "Bot handle" (bot-bob), "Bot Framework Version" (3.0), "Messaging endpoint" (<https://b962ff70.ngrok.io/api/messages>), and "Microsoft App ID" (8d4f158c-c83d-4e32-8e75-fbac64222df). There is an "Edit" link next to the details.
- Test connection to your bot:** This section in the middle contains a "Test" button to verify the bot's connection.
- Chat:** This section on the right shows a simulated chat conversation. It includes a "Publish" button at the top right. The chat history shows a user saying "Hello", the bot replying "Hi, I'm Bob! How can I help you today?", the user asking "What can you do?", the bot replying "I can help with LOE calculations, health checks, on-boarding and more!", and the user saying "Thank you".

Lab #5 – Creating the HealthCheck Backend

Now for the final part, we will start a more complex dialog that can handle entity extraction, waterfall dialogs and more to ensure that your user can automate their HealthCheck tasks.

Step #1 – Deploy Database and Backend API

You can find the database scripts and the system information API at the below link.

<https://github.com/MiracleLabs/ms-bot-framework-workshop/bob/data-api>

<https://github.com/MiracleLabs/ms-bot-framework-workshop/bob/scripts>

Run the following command to enable the DB to be populated. Note that you must have Mongo DB running and its location must be in the global path.

sh bob/scripts/db.sh for Mac Users

bob/scripts/db.bat for Windows Users

Optionally you can also run the following command directly in the command line as well,

```
mongo bobdb --eval "db.documents.insertMany([ { system: 'SAP PI',
qa: 'https://www.google.com/' , dev: 'https://www.google.com/'
,test: 'http://192.168.2.13', prod: 'https://www.google.com/' },{
system: 'DataPower', qa: 'https://www.google.com/' , dev:
'https://www.google.com/' ,test: 'http://192.168.2.13', prod:
'https://www.google.com/' } ,{ system: 'IIB', qa:
'https://www.google.com/' , dev: 'https://www.google.com/' , test:
'https://www.google.com', prod: 'https://www.google.com/' } ,{
system: 'SAP BASIS', qa: 'https://www.google.com/' , dev:
'https://www.google.com/' , test: 'http://miraclesoft.com', prod:
'https://www.google.com/' } ]);"
```

Once you have your DB up and running with the required data, you can then deploy your API to retrieve the data.

Move to the directory “/bob/data-api” and then run the following commands,

```
cd rest-api
```

```
npm install
```

```
node app.js
```

You will now see the following screen which shows that your API is running.

```
node
Chanakyas-MacBook-Pro:data-api miracle$ node app.js
-----
April 6th 2017, 02:21:34 pm | System Retrieval API is ready to go
Try retrieving a system with : http://localhost:4000/bob/system?system=iib
-----
```

Head to your browser and paste the sample URL that is given in the command line output, you should receive a JSON response as shown below.

```
localhost:4000/bob/system?system=SAP%20PI
GDrive/Chanakya Mail_Tables/GDrive Hubble/LoginPage GMovies/Chanakya
[{"_id":"58e88d43870632ebc64d5e3d","system":"SAP
PI","qa":"https://www.google.com/","dev":"https://www.google.com/","test":"http://192.1
68.2.13","prod":"https://www.google.com/"}]
```

Your backend data retrieval API is now running successfully.

Step #2 – Create Dialogs for /HealthCheck

We will be creating a waterfall model based dialog flow to handle the health check requests. We will be use the Session and Dialog classes from the Bot Builder SDK to maintain the dialog stack and navigate the user.

The following will be the flow of the dialog,

- Extract entities(System and Environment) from the user's request,
 - If entities are not found then ask user for the inputs
 - The inputs are parsed by LUIS to extract the exact values
 - If the inputs are not recognize reprompt the user
 - Once all entities are found ask user for confirmation
 - If user confirms, process and respond to request
 - Else rerun the entire dialog
 - If all entities are found then ask user for confirmation
 - If user confirms, process and respond to request
 - Else rerun the entire dialog

The first step will be to add the handler that matches to the **"intent.HealthCheck"**. This will in turn call the **"/healthCheck"** dialog which will handle the flow and return back to the next step of the main process where the user's request will be completed.

Add the following code to your **app.js** file.

```
dialog.matches('intent.HealthCheck', [  
  function(session, args) {  
    console.log("-----  
-----");  
    console.log(moment().format('MMMM Do YYYY, hh:mm:ss a') +  
" | Health Check Intent Matched");  
    console.log("-----  
-----");  
    var sys = "";  
    var env = "";  
    //Start HealthCheck Dialog and Pass Entities Array to it  
    session.send("Sure, I can help you run a health check");  
    if (args.entities.length == 0) {  
      //Start Dialog Flow for asking individual items  
      entities = {  
        "sys": "",
```

```
        "env": ""
    };
} else {
    //Go through Entities array and check for available
entities
    for (i = 0; i < args.entities.length; i++) {
        if (args.entities[i].type.includes("Systems")) {
            sys = args.entities[i].type.substr(9);
        } else if
        (args.entities[i].type.includes("Environment")) {
            env = args.entities[i].type.substr(13);
        }
    }

    entities = {
        "sys": sys,
        "env": env
    };
}

session.beginDialog('/healthCheck', entities);
},
function(session, args, next) {
    //Retrieve System Details from the DB Instance
    unirest.get("http://localhost:4000/bob/system?system=" +
args.sys).end(function(response, error) {
        if (error) {
            session.endDialog("I was unable to retrieve the
system details");
        } else {
            var url = "";

            //Check environment
            if (args.env == "Development") {
                url = response.body[0].dev;
            } else if (args.env == "Test") {
                url = response.body[0].test;
            } else if (args.env == "QA") {
                url = response.body[0].qa;
            }
        }
    });
}
```

```

    } else if (args.env == "Production") {
        url = response.body[0].prod;
    }

    if (url == "") {
        session.endDialog("I was not able to retrieve
the endpoint for " + args.sys + " - " + args.env);
    } else {
        session.sendTyping();
        //Ping system and check health
        ping.promise.probe(url, {
            timeout: 5,
        }).then(function(isAlive) {
            console.log("-----
-----");
            console.log(moment().format('MMMM Do YYYY,
hh:mm:ss a') + " | Ping Result for "+url+" is
"+JSON.stringify(isAlive));
            console.log("-----
-----");
            if (isAlive.alive) {
                session.endDialog(args.sys + " - " +
args.env + " is currently healthy with response time of
"+isAlive.avg+" ms");
            } else {
                session.endDialog(args.sys + " - " +
args.env + " is currently not reachable");
            }
        });
    }
}
});
}
]);

```

The next step will be to add the `/healthCheck`, `/askSys` and `/askEnv` dialogs. The `/healthCheck` dialog will in turn call the `/askSys` and `/askEnv` dialogs when needed to fill in the required entities for the request.

Each dialog uses the **session.replaceDialog()** functionality to be able to manipulate the dialog stack and re-prompt the user for the System and Environment details. The same replace concept will be used when the user responds back with a “no” to the confirmation prompt.

```
bot.dialog('/healthCheck', [
    function(session, args, next) {
        if (args.sys == "") {
            session.beginDialog('/askSys', {
                "sys": args.sys,
                "env": args.env
            });
        } else {
            next(args);
        }
    },
    function(session, args, next) {
        if (args.env == "") {
            session.beginDialog('/askEnv', {
                "sys": args.sys,
                "env": args.env
            });
        } else {
            next(args);
        }
    },
    function(session, args, next) {
        //Store SYS and ENV in Session Data
        session.dialogData.sys = args.sys;
        session.dialogData.env = args.env;

        //Handle Confirmation from User
        builder.Prompts.confirm(session, "Please confirm(yes/no)
that you would like to run a health check for " + args.sys + " -
" + args.env);
    },
    function(session, args, next) {
```

```
    if (args.response) {
        //Send back to the caller process
        args.sys = session.dialogData.sys;
        args.env = session.dialogData.env;
        next(args);
    } else {
        session.replaceDialog('/healthCheck', {
            "sys": "",
            "env": ""
        });
    }
}
]);

bot.dialog('/askSys', [
    function(session, args) {
        //Set args into Session Dialog Data
        session.dialogData.env = args.env;

        //Check for redo variable
        if (args && args.reprompt) {
            builder.Prompts.text(session, "I was not able to
identify that system, please try again.");
        } else {
            builder.Prompts.text(session, "What system would you
like to run a health check for?");
        }
    },
    function(session, args) {
        //Extract the System Entity
        var sysText = args.response;
        extractEntity(sysText, function(result) {
            //Set args data with session data
            args.env = session.dialogData.env;

            //Check if system was extracted
            if (result.sys != "") {
                args.sys = result.sys;
                session.endDialogWithResult(args);
            }
        });
    }
]);
```



```
    } else {
        //Replace Dialog and run the prompt again
        session.replaceDialog('/askSys', {
            "sys": "",
            "env": args.env,
            "reprompt": true
        });
    }
});

});
}

]);

bot.dialog('/askEnv', [
    function(session, args) {
        //Set args into Session Dialog Data
        session.dialogData.sys = args.sys;

        //Check for redo variable
        if (args && args.reprompt) {
            builder.Prompts.text(session, "I was not able to
identify that environment, please try again.");
        } else {
            builder.Prompts.text(session, "What environment would
you like to run a health check for?");
        }
    },
    function(session, args) {
        //Extract the System Entity
        var envText = args.response;
        extractEntity(envText, function(result) {
            //Set args data with session data
            args.sys = session.dialogData.sys;

            //Check if system was extracted
            if (result.env != "") {
                args.env = result.env;
                session.endDialogWithResult(args);
            } else {
                //Replace Dialog and run the prompt again
```

```
        session.replaceDialog('/askEnv', {
            "sys": args.sys,
            "env": "",
            "reprompt": true
        });
    }
});
}
]);
```

As a final step you will need to configure the LUIS link, APP_ID and APP_PWD to your code.

```
const APP_ID = '<LUIS_APP_ID>';
const APP_KEY = '<LUIS_APP_PWD>';
```

Along with these you will add three functions that enable custom entity extraction for your systems and environments.

```
//Identify and Extract the entity
function extractEntity(message, callback) {
    //Get the prediction from LUIS
    predictLanguage(message, function(prediction) {
        var intent = sortHighestIntent(prediction);
        var sys = "";
        var env = "";

        if (intent == "intent.IdentifyEntity") {
            //Extract the entities
            for (i = 0; i < prediction.EntitiesResults.length;
i++) {
                if
(prediction.EntitiesResults[i].name.includes("Systems")) {
                    sys =
prediction.EntitiesResults[i].name.substr(9);
                } else if
(prediction.EntitiesResults[i].name.includes("Environment")) {
                    env =
```

```
prediction.EntitiesResults[i].name.substr(13);
    }
}

    callback({
        "sys": sys,
        "env": env
    });
} else {
    callback({
        "sys": sys,
        "env": env
    });
}
});
}

//Use LUIS to identify intents and entities
function predictLanguage(message, callback) {
    var url =
    'https://westus.api.cognitive.microsoft.com/luis/v1.0/prog/apps/'
+ APP_ID + '/predict?example=' + message;
    //Retrive the top scoring intents and entities
    unirest.get(url)
        .headers({
            'Ocp-Apim-Subscription-Key': APP_KEY
        })
        .end(function(response) {
            callback(response.body);
        });
}

//Sort and return the highest score prediction
function sortHighestIntent(prediction) {
    var score = 0;
    var intent = "";
    var i;
    //Check for Entity Extraction
    for (i = 0; i < prediction.IntentsResults.length; i++) {
```

```
        if (prediction.IntentResults[i].score > score) {  
            score = prediction.IntentResults[i].score;  
            intent = prediction.IntentResults[i].Name;  
        }  
    }  
  
    return intent;  
}
```

After you add all these steps your final **app.js** file should look similar to the file in the **/bob/bot-backend/app.js** location of the GitHub repository. **Please make sure that you update your bot registration credentials in the below code.**

For any questions regarding the lab please feel free to reach out to innovation@miraclesoft.com. **We hope you enjoyed creating bots with us 😊**