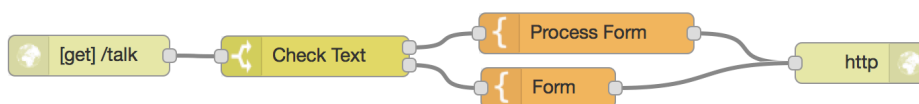


Text to Speech in Node-RED

Hands-On Lab



Add a web endpoint to convert text to audio
(see *Add Text to Speech in Node-RED*)



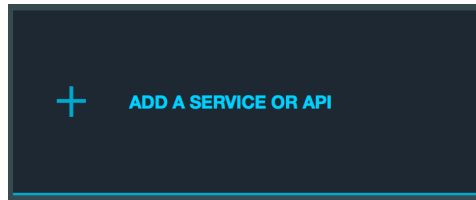
Create a webpage to input text and play audio
(see *Creating an Interactive Web UI*)



Add Text to Speech Service in IBM Bluemix

The Text to Speech node in Node-RED requires API credentials. In this section, we will create and bind the Text to Speech IBM Watson service to the Node-RED application.

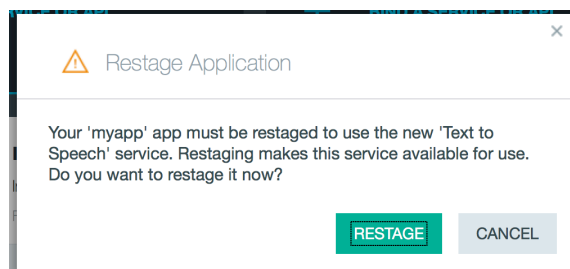
1. Go to the application overview for the Node-RED application in the IBM Bluemix dashboard and click **Add a service or API**.



2. Click the **Text to Speech** node under the Watson section. You can use the default values, or customize the name. Click on **Create**.




3. IBM Bluemix will prompt to restage the application. Click on **Restage**. The application will restart and include the new service credentials in the environment.

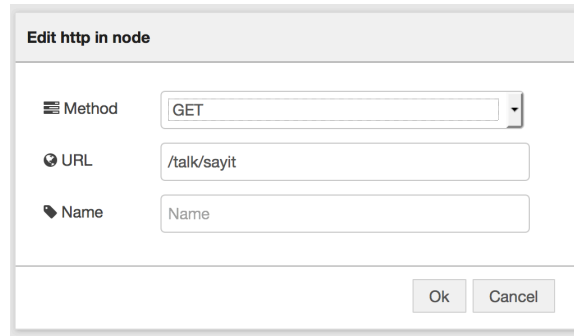


4. When the application has finished restaging, open the Node-RED Flow Editor. If Node-RED is already open, refresh the page.

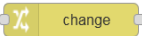
Add Text to Speech in Node-RED

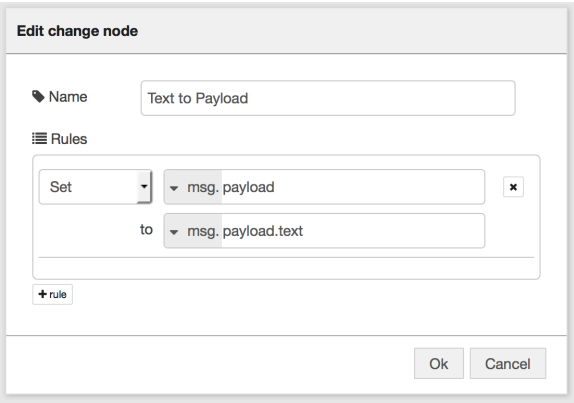
In this section, we will use the IBM Watson Text to Speech service to produce a .wav audio file from input text through a simple web endpoint generated using a Node-RED flow.

1. Add a  node as shown below to collect the incoming speech request:




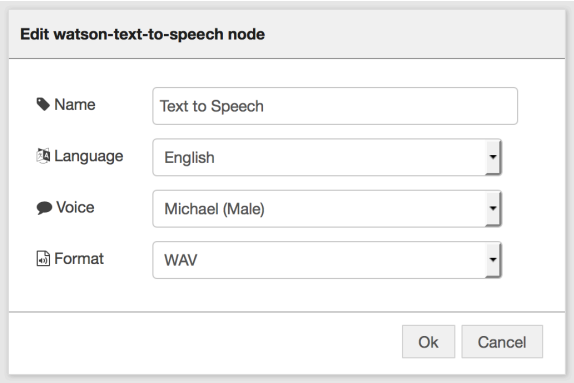
The 'Edit http in node' dialog box shows the configuration for an HTTP node. The Method is set to GET, the URL is /talk/sayit, and the Name is Name. There are Ok and Cancel buttons at the bottom right.

2. Add a  node as shown below to extract the query parameter msg.payload.text and set it as the msg.payload. When invoked with query parameters such as ?text=Hello, the text will be placed into the msg.payload object.




The 'Edit change node' dialog box shows the configuration for a change node. The Name is Text to Payload. The Rules section shows a rule where the Set operator is applied to msg.payload, and the value is msg.payload.text. There is a + rule button at the bottom left and Ok/Cancel buttons at the bottom right.

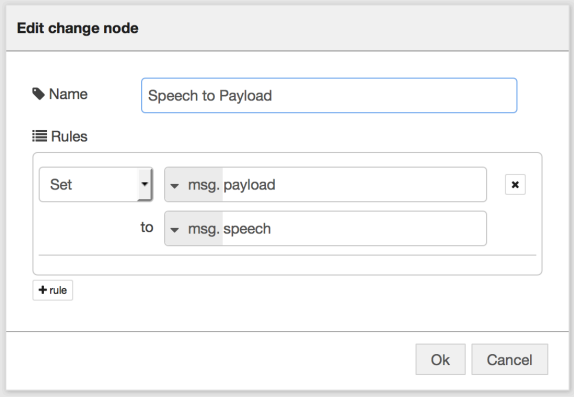
3. Now add a  node as shown below. This node will generate the binary wav stream content and put it in the msg.speech property.




The 'Edit watson-text-to-speech node' dialog box shows the configuration for a Watson Text to Speech node. The Name is Text to Speech. The Language is set to English, the Voice is Michael (Male), and the Format is WAV. There are Ok and Cancel buttons at the bottom right.

You can change the language, the voice, or the format of the audio that is returned.

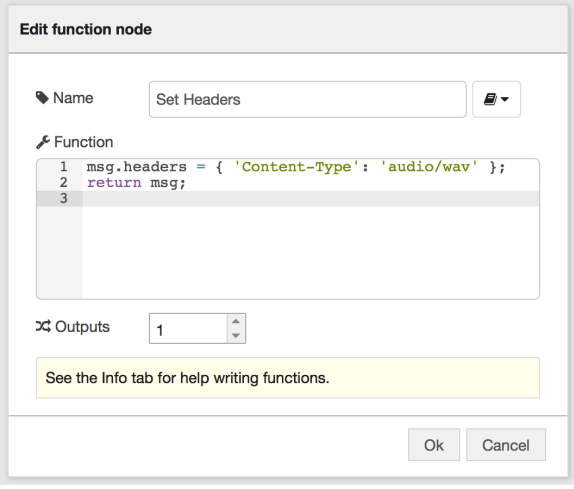
4. Add another  change node as shown below to extract the msg.speech and place it in msg.payload.



The 'Edit change node' dialog shows a name 'Speech to Payload'. Under the 'Rules' section, there is a rule where the 'Set' dropdown is selected, the 'to' field is 'msg.speech', and the 'msg.payload' field is selected with a delete button (x) next to it.


5. Add a  function node as shown below to add HTTP headers to the response:

```
msg.headers = { 'Content-Type': 'audio/wav' };  
return msg;
```

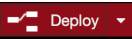


The 'Edit function node' dialog shows a name 'Set Headers'. The 'Function' field contains the code: `1 msg.headers = { 'Content-Type': 'audio/wav' };
2 return msg;
3`. The 'Outputs' field is set to 1.

We set the HTTP response headers by setting the msg.headers to the Content Type of audio/wav. This is required in order to let browsers know that this is an audio file and not HTML.

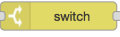
6. Finally, add a  http response node. This node will simply return what's in msg.payload to the HTTP response. The completed flow should look like the flow shown below:

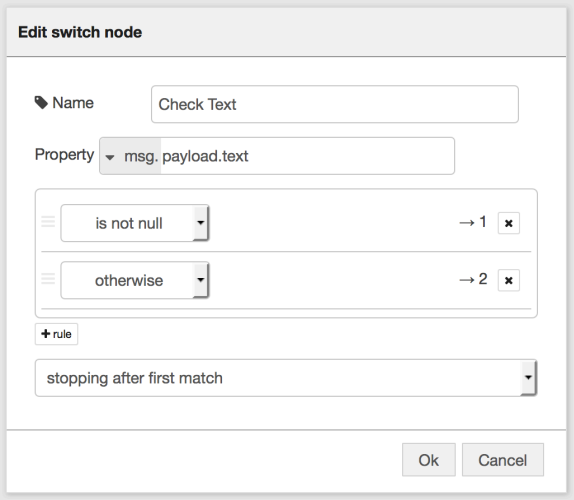


7. Click on  to save the changes. Test the flow by opening the application URL, appended with /talk/sayit?text=Hello as shown below:

`https://_____.mybluemix.net/talk/sayit?text=Hello`

This should prompt you to save a file. Depending on how your browser is configured, it may download the audio file or play it within the browser. Either way, play it and you should hear the text.

8. This flow has a caveat, however. The flow will fail when the text query parameter is not set. Add a  node as shown below to check if the text query parameter is present.



Edit switch node

Name: Check Text

Property: msg.payload.text

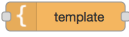
Rule 1: is not null → 1

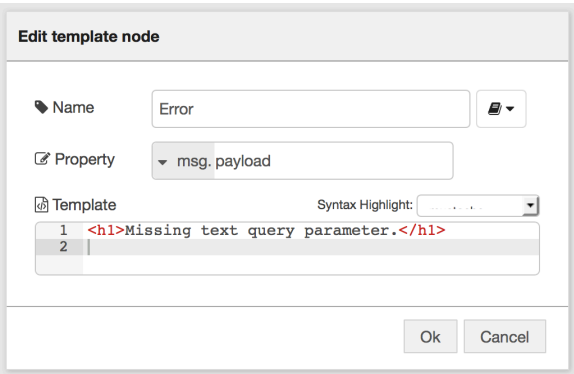
Rule 2: otherwise → 2

+ rule

stopping after first match

Ok Cancel

9. You'll notice that adding the second otherwise rule has created a second output handle for the switch node. Add a  node with the error message shown below.



Edit template node

Name: Error

Property: msg.payload

Template


```
1 <h1>Missing text query parameter.</h1>
2
```

Syntax Highlight:

Ok Cancel

10. Connect the nodes together as shown below:



11. Click on  to save the changes. Test the flow by opening a browser and going to the application URL, appended with the following endpoints:

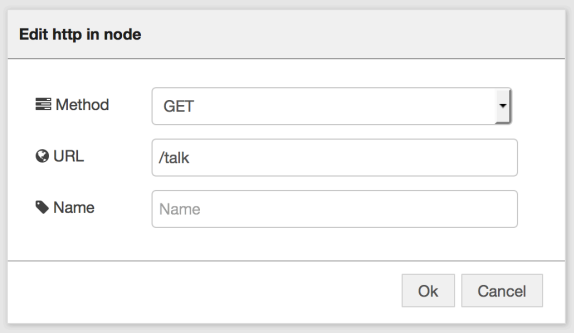
https://_____.mybluemix.net/talk/sayit?text=Hello
 https://_____.mybluemix.net/talk/sayit

(plays the audio with Hello)
 (displays error message)

Creating an Interactive Web UI

In this section we will create a simple webpage that displays an input textbox for the user to enter text. When the form is submitted, the text will be played via audio.

1. Add a  node with the following:



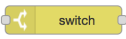
Edit http in node

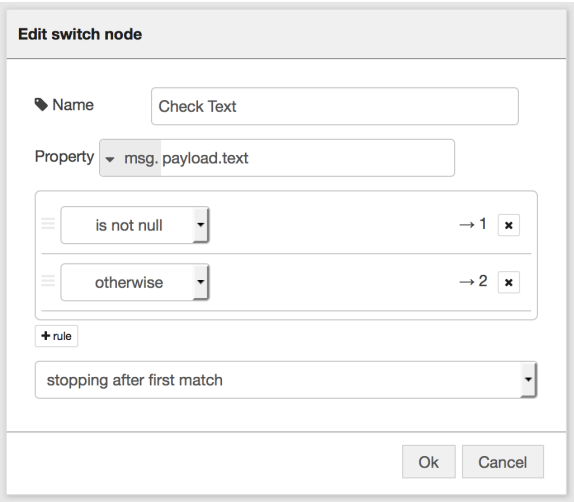
Method: GET

URL: /talk

Name: Name

Ok Cancel

2. Add a  node to branch the flow depending whether text input is submitted.



Edit switch node

Name: Check Text

Property: msg.payload.text

is not null → 1

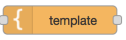
otherwise → 2

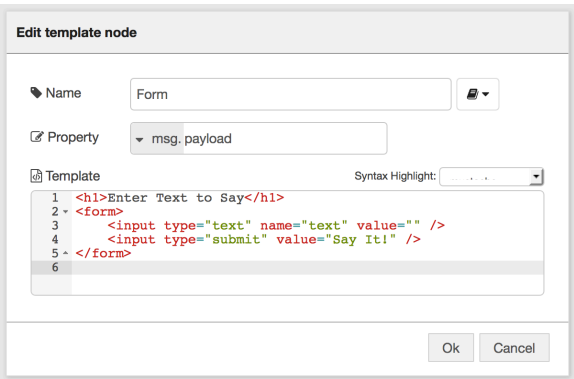
+ rule

stopping after first match

Ok Cancel

When the form is submitted and the text parameter is present, proceed with the first flow. Otherwise, the blank form (flow #2) will be used.

3. Add a  node with the HTML shown below:



Edit template node

Name: Form


Property: msg.payload

Template

```
1 <h1>Enter Text to Say</h1>
2 <form>
3   <input type="text" name="text" value="" />
4   <input type="submit" value="Say It!" />
5 </form>
6
```

Syntax Highlight: . . .

Ok Cancel

4. Add a  node with the HTML shown below:

Edit template node

Name:

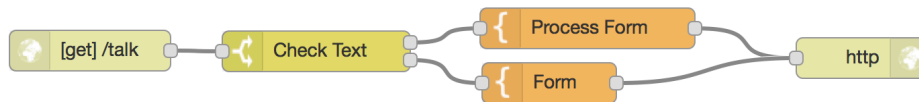
Property:


Template

```
1 <h1>You Want To Say</h1>
2 <p><q>{{payload.text}}</q></p>
3
4 <h1>Hear It</h1>
5 <audio controls autoplay>
6   <source src="/talk/sayit?text={{payload.text}}" type="audio/wav">
7   Your browser does not support the audio element.
8 </audio>
9
10 <h1>Enter Text to Say</h1>
11 <form>
12   <input type="text" name="text" value="{{payload.text}}" />
13   <input type="submit" value="Say It!" />
14 </form>
15
```

Ok Cancel

5. Add a  and connect the nodes together as follows:



6. Click  to save the changes. Test the flow by opening the application URL in the browser, appended with /talk.

https://_____.mybluemix.net/talk