

AI benefits for the lazy hacker

Get fast usable access to AI services with Node-RED

The purpose of this workshop is to give you quick and easy access to the *IBM Watson Cognitive Services* APIs, and allow you to experiment with

- Image classification
- Speech to text, and text to speech
- Document discovery
- Language identification and translation with little or no code.

This workshop assumes a little programming understanding/experience - an appreciation of procedural logic, data structure, and the use of API-based services.

For those with full-on developer experience, we hope you'll find using the [Node-RED](#) tools a fun and useful addition to your skills kitbag.

What You'll Learn



We will kick off with the basics of running **Node-RED** in the IBM Cloud (previously known as *Bluemix*).

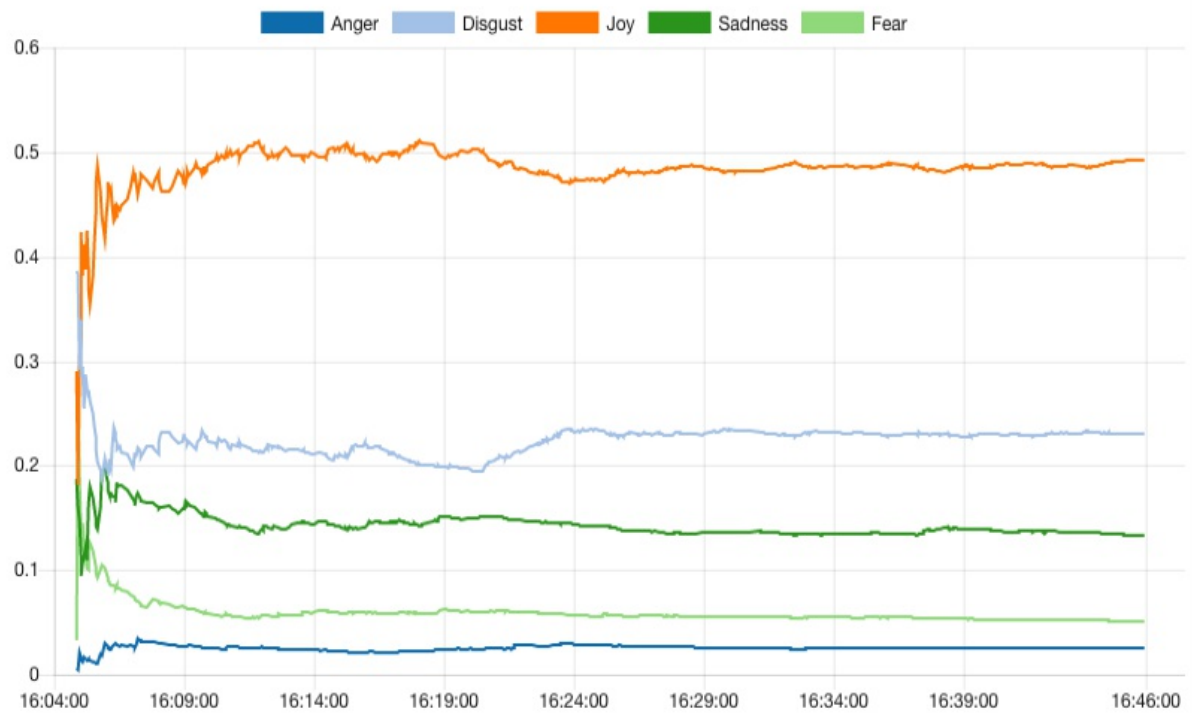
You'll learn how to construct event flows in the Integrated Development Environment (IDE), through simple, and progressively more complex examples.

Then you'll be ready to link into the Watson services, to experiment with off-the-shelf Artificial Intelligence capabilities you can use straight away to build or enhance applications.

Immediate results will be a Node-RED web server app which can display trending emotions associated with a popular Twitter hashtag, and optional generate (in)appropriate responses or replies.

Tone Analyzer

#NationalPetDay Tweets Tone Analyzer



What You'll Need

1. A laptop running Windows, MacOS, or linux, with access to the public internet.
2. A current version of one of the following browsers:
 - [Firefox](#)
 - [Chrome](#)
 - [Safari](#)
3. An IBM Cloud account; if you don't have one already, sign up at [IBM Cloud account setup](#)

For a brand new IBM Cloud account, that's it!

If you have an existing IBM Cloud account, and have existing applications and services, particularly *Cloudant* database instances, you'll possibly need a couple of extra tools:

1. the [Cloud Foundry](#) command line tool `cf`
 - [download and install `cf` from github](#).
2. the [Git Version Control Management](#) command line tool `git`
 - [download and install `git` from git-scm](#)

A note of thanks and appreciation

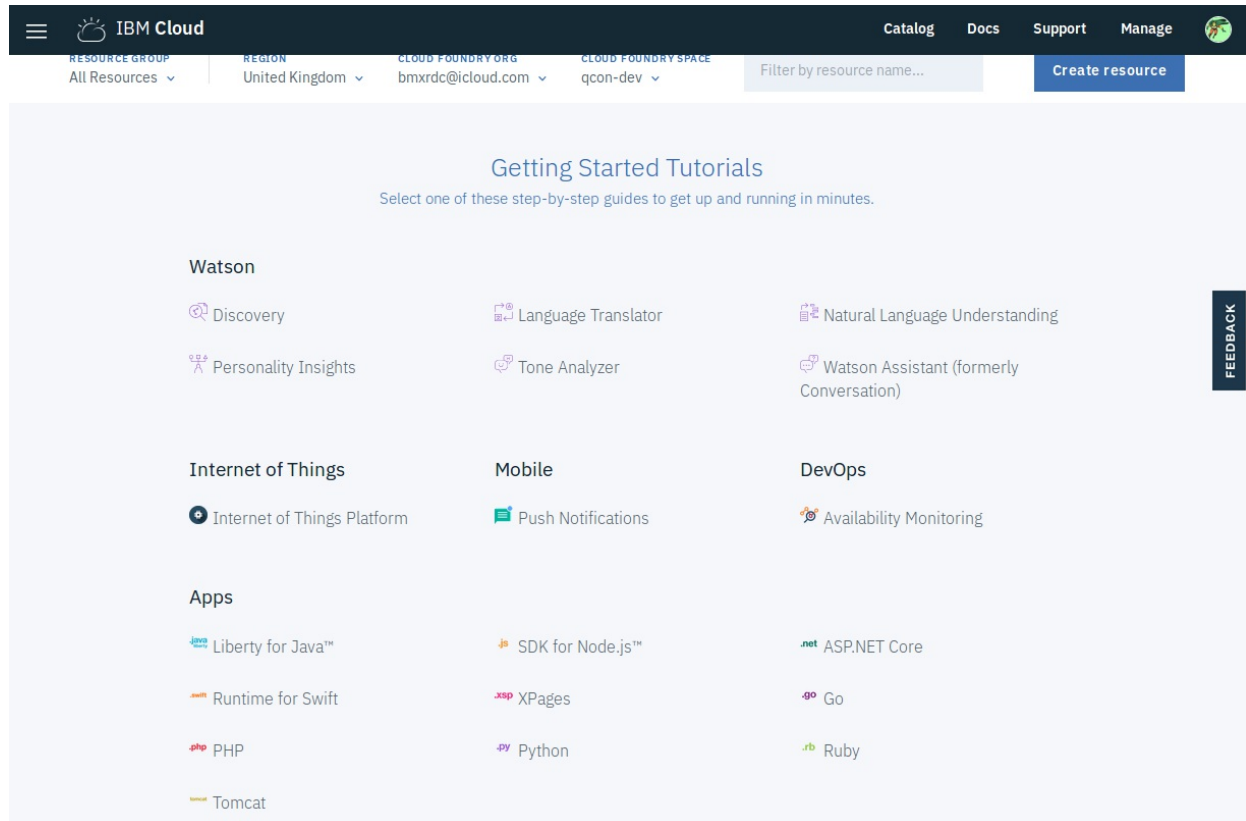
Sections of this workshop take significant inspiration (and a bit of sample code) from the [Watson Tone Analyzer Tutorial](#) by Michael Qiu from the excellent team at [SenseTecnica](#).

Sample data sourced via RESTful API calls is provided by [typicode's JSONPlaceholder](#)

Installation/environment Requirements

Using your IBM Cloud account, login to the IBM Cloud [console](#).

For a brand new IBM Cloud account, you'll see a dashboard similar to this:



We need to establish what type of IBM Cloud you have, so there are a couple of simple steps:

1. at the top right of the dashboard, click on **Manage**
2. click **Billing and Usage**
3. and then click **Billing**

The resulting page

☰ IBM Cloud

Catalog Docs Support Manage

Profile

Platform Notifications

Usage Dashboard

Billing

Cloud Foundry Orgs

Resource Groups

Billing

Account Type

Account

Account Type

256 MB of free memory each month, free Lite plan services to choose from, and easy to upgrade when you're ready for more.

Get a \$200 credit when you upgrade

Enter your credit card to receive a limited offer promotional credit. The credit does not apply to infrastructure and third-party services.

Add Credit Card

Account: ross cruickshank's Account

FEEDBACK

shows the Account Type information.

Note your account type

To return to the dashboard view, click on the ☰ menu icon, and click on Dashboard

✕

Containers

Infrastructure

VMware

Dashboard

APIs Dashboard

Application Services

Apple Development New

Blockchain

Things to know about your account

Lite (Free)

- 256M runtime memory, up to 100 service instances
 - sample applications tend to be created with 256M allocated, so will need adjusted if you want to run more than one application at a time
 - service instances by IBM (database, messaging, Watson, etc) will be created with what is call a *Lite Plan*, only one each of which can defined to your account at any time; for example, trying to create a second Watson Speech to Text service instance will fail
 - third party services which have no IBM Cloud charging plan (twilio, elephantSQL, rediscloud, ...) can be added, subject to the provider's restrictions
 - sample applications available in the catalog (shown as "Boilerplates") may try and create instances of services (usually Cloudant database), regardless of whether a similar instance already exists

Pay-As-You-Go

- 512M runtime memory, up to 100 service instances
 - keep using the free services (*Lite Plan*), as well as adding other services which may incur charges after any free use allocation

Trial (Free)

- you have applied a promotion code to your account, usually to experiment with services for which the `Lite (Free)` option is too restrictive; this type of account is time-limited, and must be converted to `Pay-As-You-Go` (or `Subscription`) by the end of the trial period, or it will be suspended.

Subscription

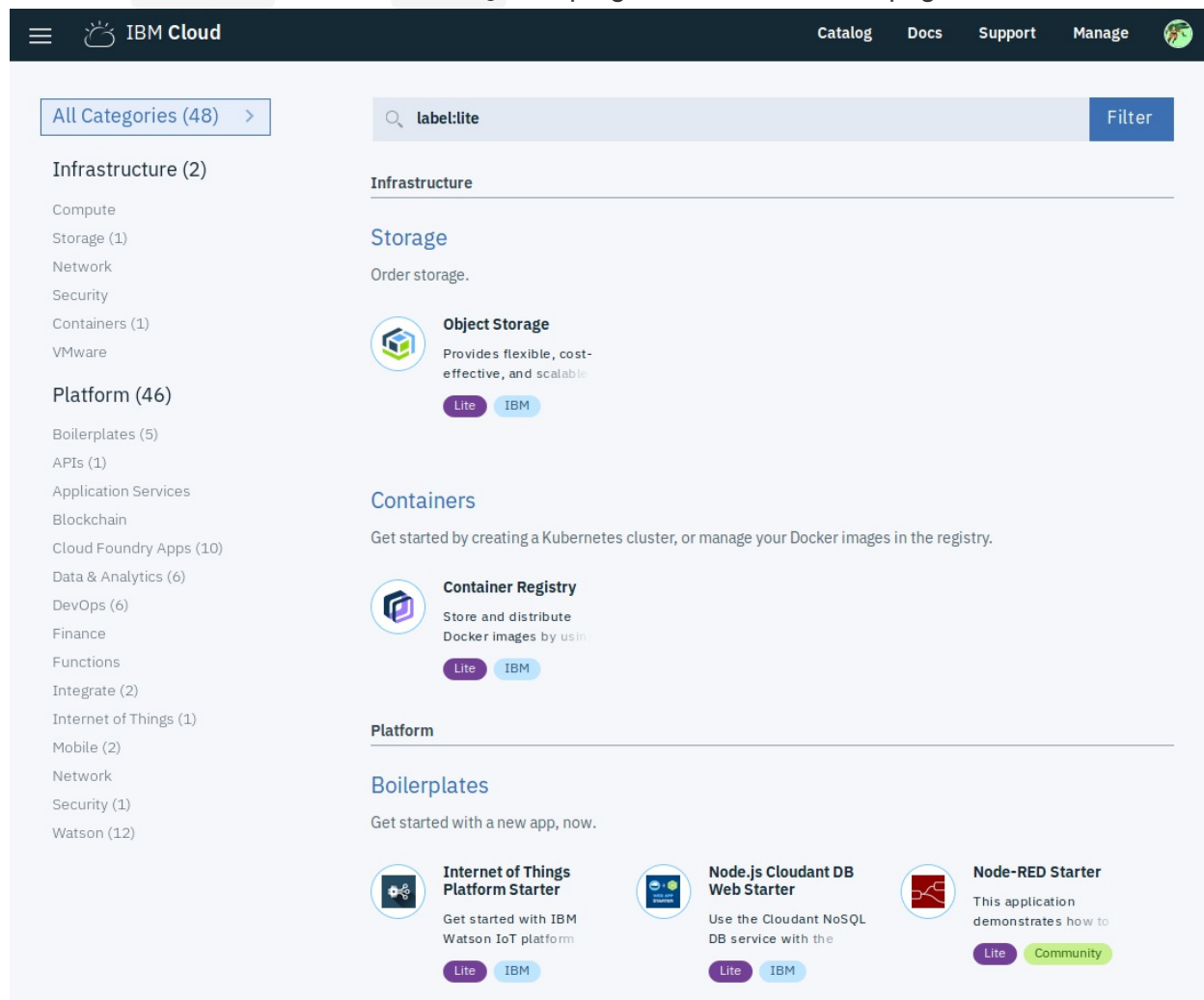
- you've elected to pay monthly upfront - you probably know what you're doing ...

Getting started

Account type = Lite (Free) , no existing applications or services

This is the simplest starting point.

From the **Dashboard** , click on **Catalog** at top-right. You should see a page similar to this:



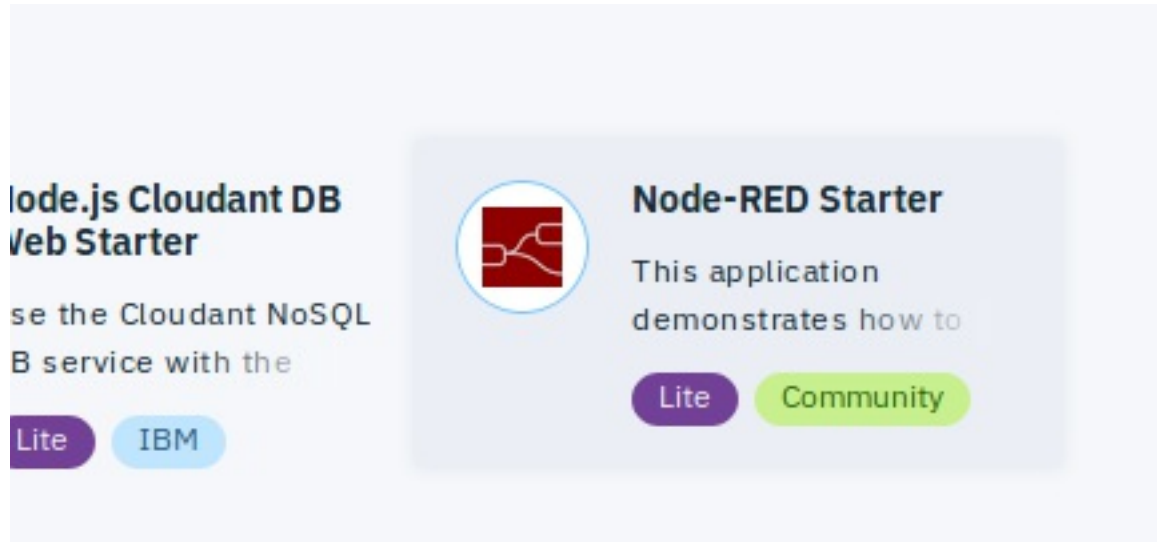
(The categories and numbers vary from time to time, as new offerings are published, and older/less popular services are deprecated).

The area of interest for this workshop is the **Node-RED Starter** Boilerplate, shown here bottom-right.

This boilerplate will create a ready-to-use web-accessible Node-RED application, employing three components:

1. **Cloudbant** nosql database - used to store Node-RED configurations (and application-generated data) - see also [Apache CouchDB](#)
2. Node.js runtime container (**node v6.13.0** at time of writing)
3. Node-RED application package (**node-red v0.18.4** at time of writing)

Hover over the description of the Node-Red Starter until highlighted:



then click.

At this point, you will need to provide a **unique** name for your hosted application - note, you only need to enter a name into the `App name:` field; the `Host name:` field will fill automatically.

A screenshot of the IBM Cloud 'Create a Cloud Foundry App' page. The header shows the IBM Cloud logo and a 'View all' link. The main heading is 'Create a Cloud Foundry App'. Below this, the 'Node-RED Starter' card is shown with its description and 'Lite'/'Community' buttons. To the right, there are input fields for 'App name:' and 'Host name:'. The 'App name:' field contains the text 'QCON-NR-AIWORKSHOP-ross'. The 'Host name:' field also contains 'QCON-NR-AIWORKSHOP-ross'. At the bottom, there are labels 'Choose a region/location to' and 'Choose an organiza'.

(if you're stuck for a name - as a minimum, change the example by overwriting "ross" with some random - alphanumeric - characters)

A brief introduction on some of the Watson Cognitive Services

As you can see, there are quite some nodes in the Watson node modules, and they are all used for different purposes. Each of the Watson nodes is corresponding to a specific Watson service on the IBM Cloud platform. You are able to view the documentation of each service in the info tab of these nodes.

- **Assistant** : helps to interpret human language and host a simple conversation/dialog. This is the Watson component you'd use for building a Chatbot backend service. -- [Driving chatbot demo](#)
- **Discovery** : helps prepare your unstructured data, create a query that will pinpoint the information you need, and then integrate those insights into your application or existing solution. This service also includes the continuously updated **News** service, giving access to current and recent news articles from around the world -- [News demo](#)
- **Natural Language Classifier** : applies cognitive computing techniques to return the best matching classes for a sentence or phrase -- [Classifier demo](#)
- **Natural Language Understanding** : The Natural Language Understanding will analyze a block of text, an article or HTML data to output a JSON object that matches the input data with various index -- [NLU demo](#)
- **Personality insights**: derives insights from user-created text/content (transactional and social media data) to identify psychological traits which might determine purchase decisions, intent and behavioral traits; may be utilized to improve retail conversion rates -- [Personality insights demo](#)
- **Visual Recognition**: Analyze images for scenes, objects, faces, and other content. Choose a default model off the shelf, or create your own custom classifier. Develop smart applications that analyze the visual content of images or video frames to understand what is happening in a scene -- [Visual Recognizer demo](#)
- **Speech to text/Text to speech**: for a growing set of spoken languages, converts from one form to another, allowing speech to be processed by text-oriented services like chatbots, Personality Insights, etc, and use speech synthesis to convert messages and text into audio streams -- [STT demo](#), [TTS demo](#)
- **Tone Analyzer**: The Tone Analyzer service uses linguistic analysis to detect emotional tones, social propensities, and writing styles in written communication. This service will be

used in this workshop to show how Node-RED nodes can be used to easily integrate Watson Cognitive Services into an application, with little or no AI skills -- [Tone Analyzer demo](#)

Labs Overview

In this lab, you will learn how Node-RED can be used as a rapid prototyping server application which can integrate local and remote data, and present information in a variety of easily consumable forms.

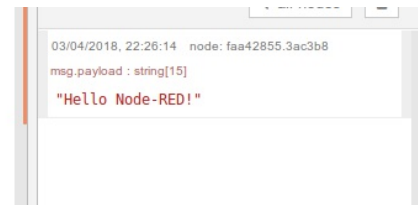
The first couple of activities help set up Node-RED as a web application service.

1. Initially, simple application-generated data will be sent to the requester (a browser).
2. Then remote data will be requested, reformatted and returned to the requester.
3. Next, your application will use the Watson Tone Analyzer service referred to earlier to quickly and easily analyze the content of Twitter messages and graphically display the trending tones/sentiments embodied in the messages.
4. Finally, select tweets will be exchanged with an example Watson Conversation Agent (Chatbot) and the responses sent to the requester.

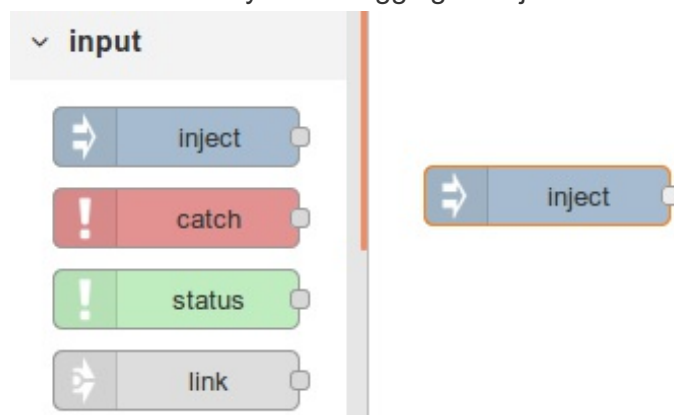
Lab - basic user response

Step one for any first-time exercise with a new programming tool is `Hello World!`

The most basic option for this in Node-RED is a combination of an `Inject` and a `Debug` node; for Node-RED this looks like



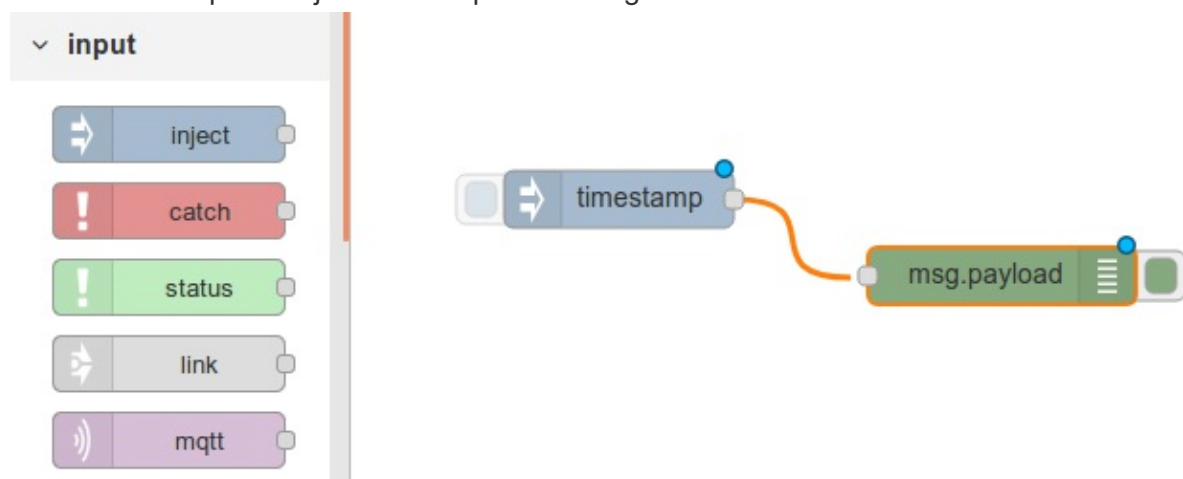
This is achieved by click/dragging an `Inject` node from the palette on the left, onto the canvas:



then do the same for the `Debug` node:



and link the output of `Inject` to the input of `Debug`:



Click-and-hold the output connector of the `Inject` node, drag the connection line to the `Debug` node input connector, ensuring the target connector turns orange; then release.

Review the `Debug` node by double-clicking in the body of the node, and you will see the configuration panel:

Edit debug node

Delete

Cancel

Done

▼ node properties

☰ Output

▼ msg. payload

⌕ To

☒ debug window

☐ system console

☐ node status (32 characters)

🏷 Name

Name

Click

Cancel

Review the Inject node by double-clicking in the body of the node, and you will see its configuration panel:

Edit inject node

Delete

Cancel

Done

▼ node properties

✉ Payload

▼ timestamp

☰ Topic

☐ Inject once after

0.1

seconds, then

🔄 Repeat

none

🏷 Name

Name

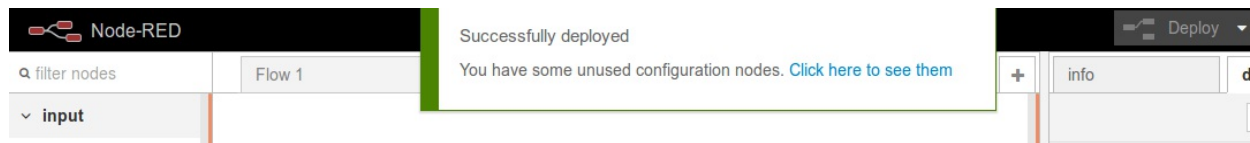
Note: "interval between times" and "at a specific time" will use cron.
"interval" should be less than 596 hours.
See info box for details.

Click on the black triangle next to `timestamp` , select the `string` option, and type into the new text entry field the message you would like to see in the debug window. Then click `Done` .

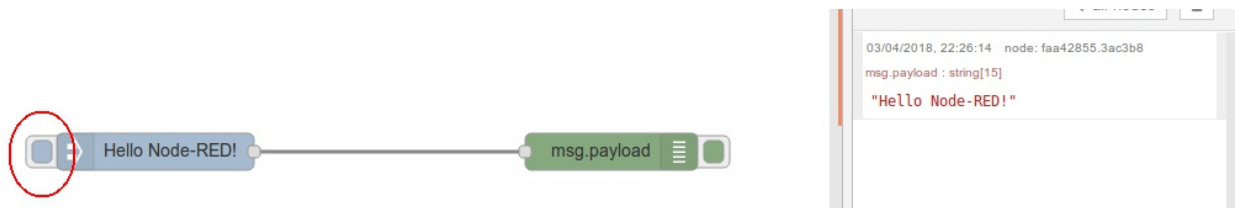
You should see that the Inject node has a blue dot, indicating it has been modified:

also observe that the top-right menu now has a red `Deploy` button:

This indicates the Node-RED configuration has changed and does not match the running application. Click **Deploy** to add the flow you just created into Node-RED runtime.



Click on the button on the left side of the **Inject** node and observe the message displayed in the debug window:



Congratulations! - you have established a working Node-RED live application, and successfully generated data.

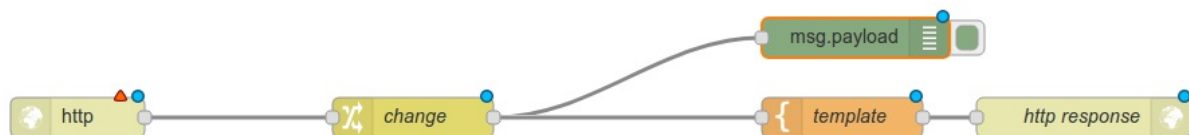
Lab - enable Node-RED as a server application

Now you have a working Node-RED instance, you can add support for external applications (browsers, other applications on mobile devices, cloud apps, etc) to invoke services within Node-RED, to exchange information.

In this Lab, the goal will be to add support for incoming HTTP requests, from a browser, to request data be returned to display in the browser (i.e basic HTML).

1. Add the necessary nodes
2. Configure the nodes to support inbound HTTP and return content
3. Deploy
4. Test and validate via ***https://{{your-node-red-hostname}}/stage1***

Using the techniques from the previous Lab, add a new flow to the canvas:



Configure the `http` (inbound) node to accept the url `/stage1`:

Edit http in node

Delete

Cancel

Done

▼ node properties

Method

GET

URL

/stage1

Name

Name


Configure the `change` node to set the response text, by changing `msg.payload` to the desired


message.

Edit change node

DeleteCancelDone

▼ node properties

 Name

 Rules

Set

▼ msg. payload

to

▼ a_z

Hello World! (Stage 1)|

Configure the `template` node to be some simple HTML, and imbed the msg content from the flow, using [mustache templating](#):

Edit template node

Delete

Cancel

Done

node properties

Name

Name

Set property

msg. payload

Format

Mustache template

Template

Syntax Highlight: mustache

1<h2>{{payload}}</h2>

2

3Show me more

4

The `http response` node does not usually need configuration, but is required as a final stage for returning data to a waiting http requester.

Now `Deploy` the changes.

To test this flow, open a new browser tab or window and load **`https://{{your-node-red-hostname}}/stage1`**.

All being well, you should receive the following response:

Hello World!

[Show me more](#)

Note that if you click on the `Show me more` link, you will receive a response like `Cannot GET /stage2` - that's because "stage2" will be added in the next Lab.

Congratulations! - you have completed the first stage of establishing Node-RED as a web server application, and responding to HTTP requests with static data.

Lab - remote data presentation

Now your Node-RED application can service web requests, you can add support for integrating local and remote data into your responses.

In this Lab, the goal will be to take an incoming HTTP request, from a browser, retrieve data from a remote application API, and return part of the data (reformatted) to display in the browser (i.e basic HTML).

1. Add the necessary nodes
2. Configure the nodes to support inbound HTTP, invoke API, and return content
3. Deploy
4. Test and validate via ***<https://{{your-node-red-hostname}}/stage2>***

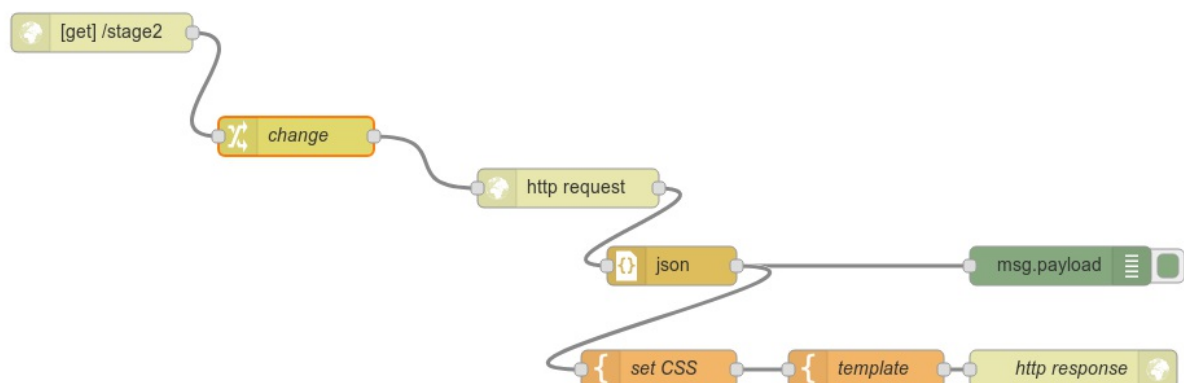
The source for the remote data is a very handy, simple JSON API server provided by

[@typicode](#)

<https://jsonplaceholder.typicode.com/> - the /users API returns a reasonable amount of data for demonstration purposes:

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
```

As before, add a new flow to the canvas:



A small, but important difference to note this time - instead of setting the `msg.payload` property, the `change` node is placing the target API url into the `msg.url` property. This is used to dynamically configure the `http request` node.

Edit change node

Delete Cancel Done

▼ **node properties**

📌 Name

☰ Rules

Set ▼ msg.url

to ▼ a_z https://jsonplaceholder.typicode.com/users

x

The `http request` node does not need any specific configuration for this flow.

The response from <https://jsonplaceholder.typicode.com/users> is a string; to process more easily and efficiently in Node-RED, we pass it through a `JSON` node which will parse the string into a JSON object (in this case an array of *user* objects).

To add a little sophistication to the response content, the two `template` nodes allow generation of dynamic HTML, and support the creation of inline CSS.

The first `template` sets a msg property `msg.css` to some simple CSS for imbedding in the returned HTML:

Edit template node

Delete

Cancel

Done

▼ node properties

📌 Name

set CSS



✎ Set property

▼ msg. **CSS**

</> Format

Mustache template

📄 Template

Syntax Highlight: mustache

```
1 .col {
2     display: inline-block;
3     width: 150px;
4     vertical-align: top;
5 }
6 .address {
7     display: inline-block;
8     width: 200px;
9 }
10 .title {
11     font-weight: bold;
12 }
```

```
.col {
  display: inline-block;
  width: 150px;
  vertical-align: top;
}
.address {
  display: inline-block;
  width: 200px;
}
.title {
  font-weight: bold;
}
```

The second `template` node creates a tabular response from the users array, and imbeds the CSS:

Delete

Cancel

Done

node properties

Name

template

Set property

msg. payload

Format

Mustache template

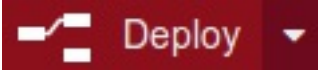
Template

Syntax Highlight: mustache

```
1 <style> {{{css}}} </style> <!-- imbed styling from msg.css -->
2 <div>
3   <span class="col title user">Username</span>
4   <span class="col title address">Location</span>
5   <span class="col title website">Website</span>
6 </div>
7 {{{payload}}}
8 <div>
9   <span class="col user">{{username}}</span>
10  <span class="col address">
11    {{{address}}}
12    <details>
13      <summary>
14        <a target="_blank" href="https://www.google.com/maps/search/?api=1&query={{geo.lat}},{{geo.lng}}">{{city}}</a>
15      </summary>
16      {{{street}}<br>{{city}}<br>{{zipcode}}
17    </details>
18    {{{/address}}}
19  </span>
20  <span class="col website"><a href="http://{{{website}}}">{{website}}</a></span>
21 </div>
22 {{{/payload}}}
```

```
<style> {{{css}}} </style> <!-- imbed styling from msg.css -->
<div>
  <span class="col title user">Username</span>
  <span class="col title address">Location</span>
  <span class="col title website">Website</span>
</div>
{{{payload}}}
<div>
  <span class="col user">{{username}}</span>
  <span class="col address">
    {{{address}}}
    <details>
      <summary>
        <a target="_blank" href="https://www.google.com/maps/search/?api=1&query=
      </summary>
      {{{street}}}<br>{{city}}<br>{{zipcode}}
    </details>
    {{{/address}}}
  </span>
  <span class="col website"><a href="http://{{{website}}}">{{website}}</a></span>
</div>
{{{/payload}}}
```

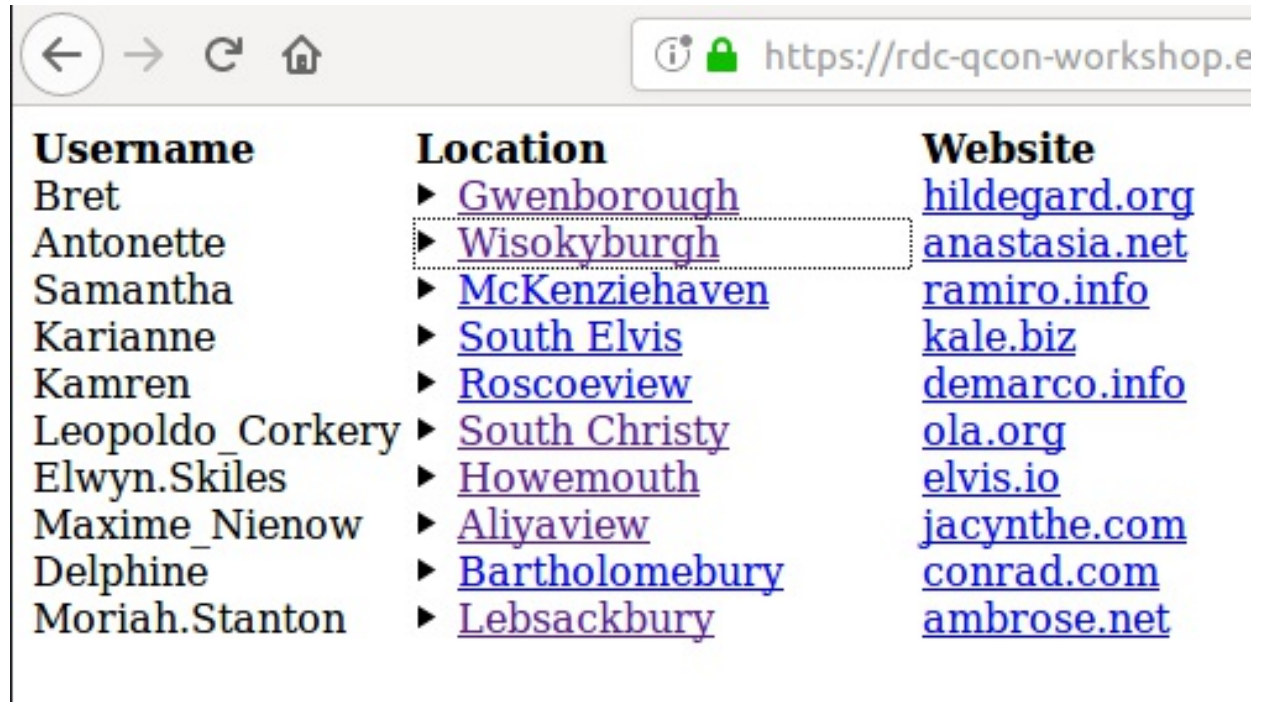
Note that although the response will generate links to **Google Maps** locations, the data appears to point to random places.

And again, use the  button to update the runtime.

Now to test ...

https://{{your-node-red-hostname}}/stage2

You should see a response like:



Username	Location	Website
Bret	▶ Gwenborough	hildegard.org
Antonette	▶ Wisokyburgh	anastasia.net
Samantha	▶ McKenziehaven	ramiro.info
Karianne	▶ South Elvis	kale.biz
Kamren	▶ Roscoeview	demarco.info
Leopoldo_Corkery	▶ South Christy	ola.org
Elwyn.Skiles	▶ Howemouth	elvis.io
Maxime_Nienow	▶ Aliyaview	jacynthe.com
Delphine	▶ Bartholomebury	conrad.com
Moriah.Stanton	▶ Lebsackbury	ambrose.net

Congratulations! - you have completed the second stage of establishing Node-RED as a web server application, and responding to HTTP requests with dynamically sourced data.

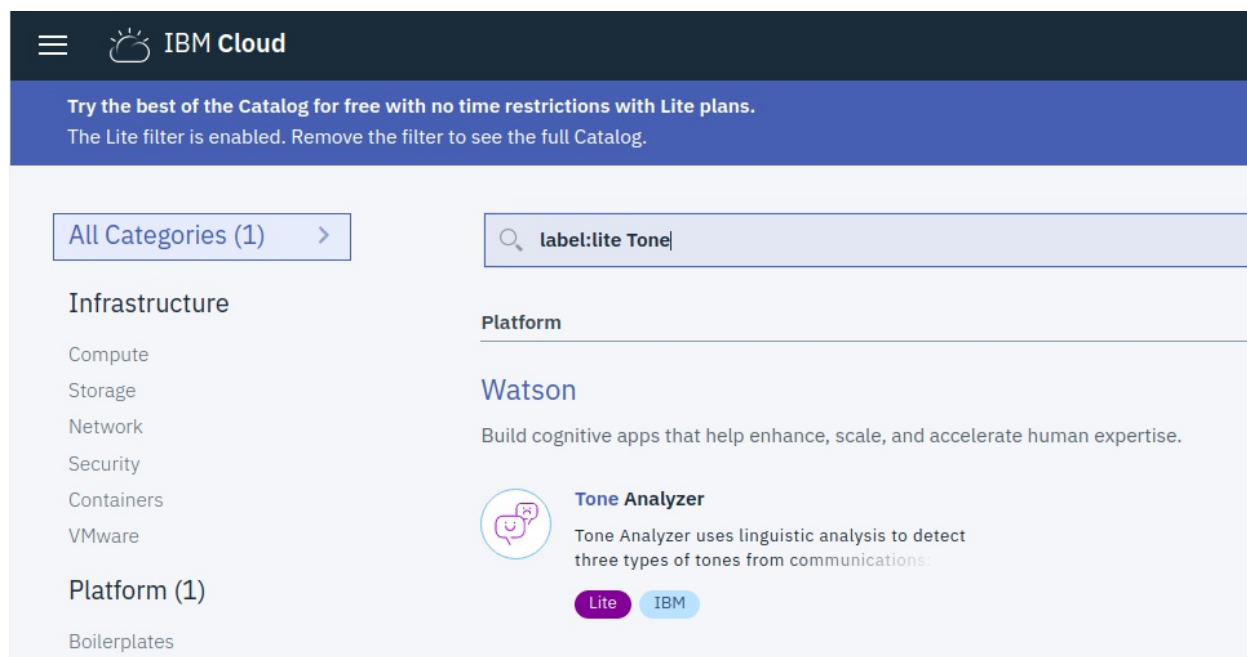
The next Lab will build on your new Node-RED skills to pull in tweets from [Twitter](#), analyze with the Watson AI language services, and generate a dashboard showing trending emotional response.

Lab - twitter sentiment

This time, you can build a fun, simple application that uses the Watson Tone Analyzer service.

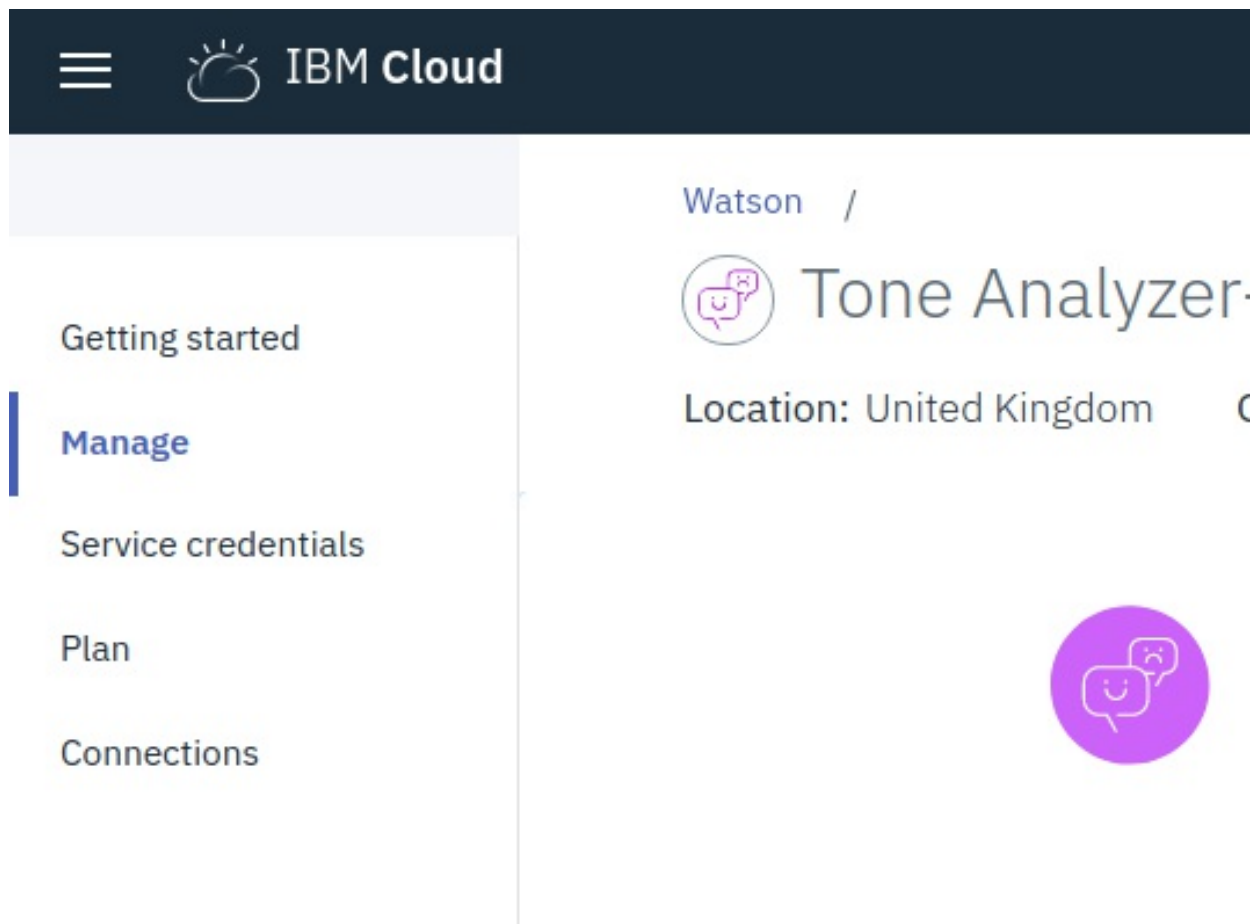
You will need an instance of the Watson Tone Analyzer service to be created from the IBM Cloud catalog.

Switch to the IBM Cloud dashboard, select `catalog` from the menu bar (top-right), and add "Tone" to the filter search argument.

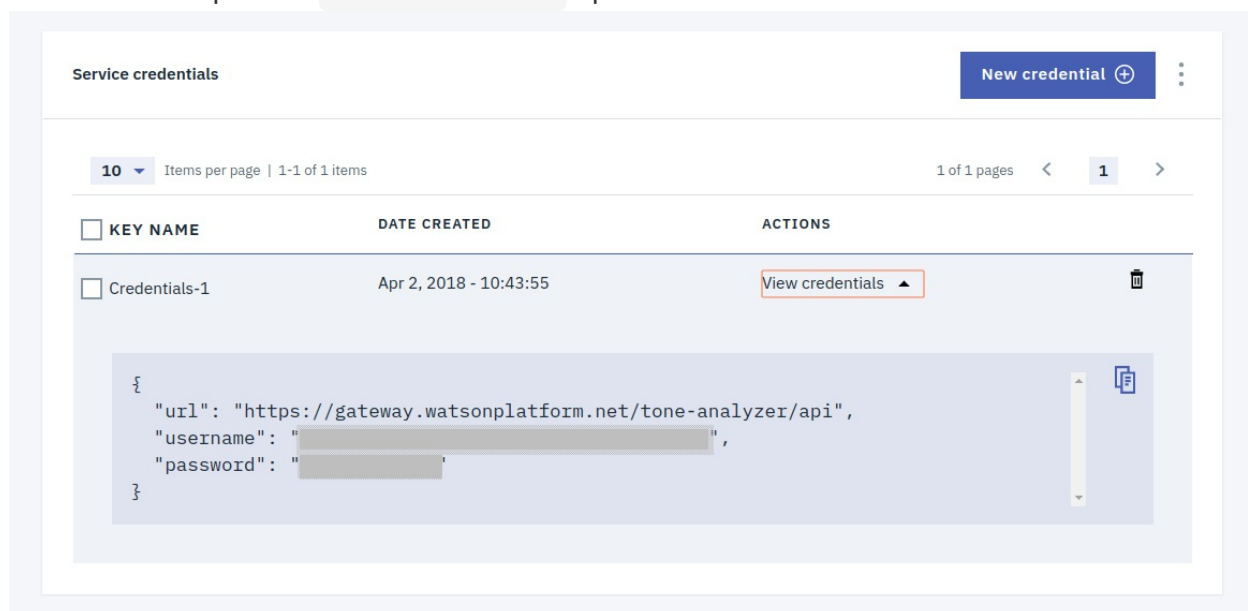


Click on `Tone Analyzer` and create a new instance, accepting all defaults.

When the instance has been provisioned, you'll be presented with the service management panel -- select the `Service credentials` option from the left navigator.



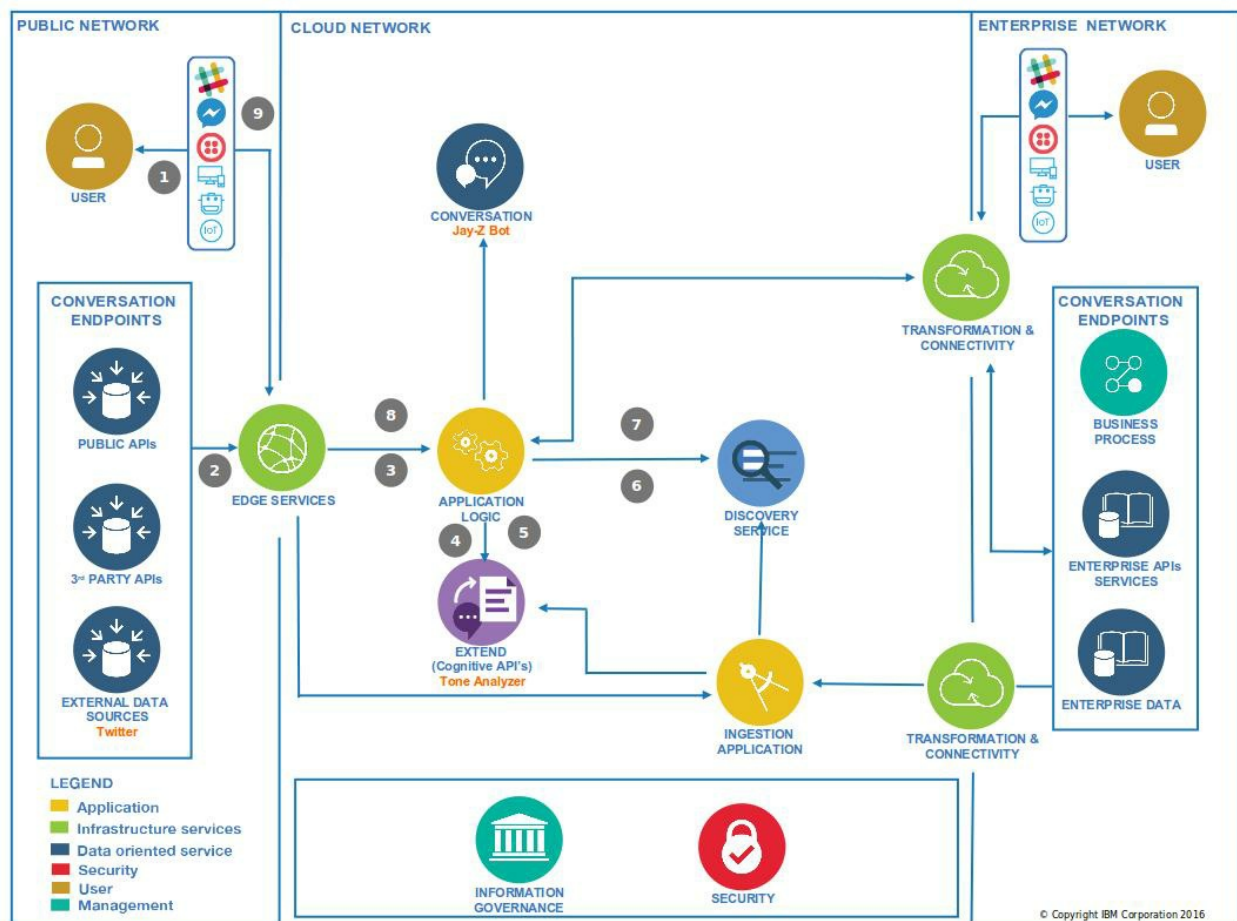
Then find and open the `View credentials` option:



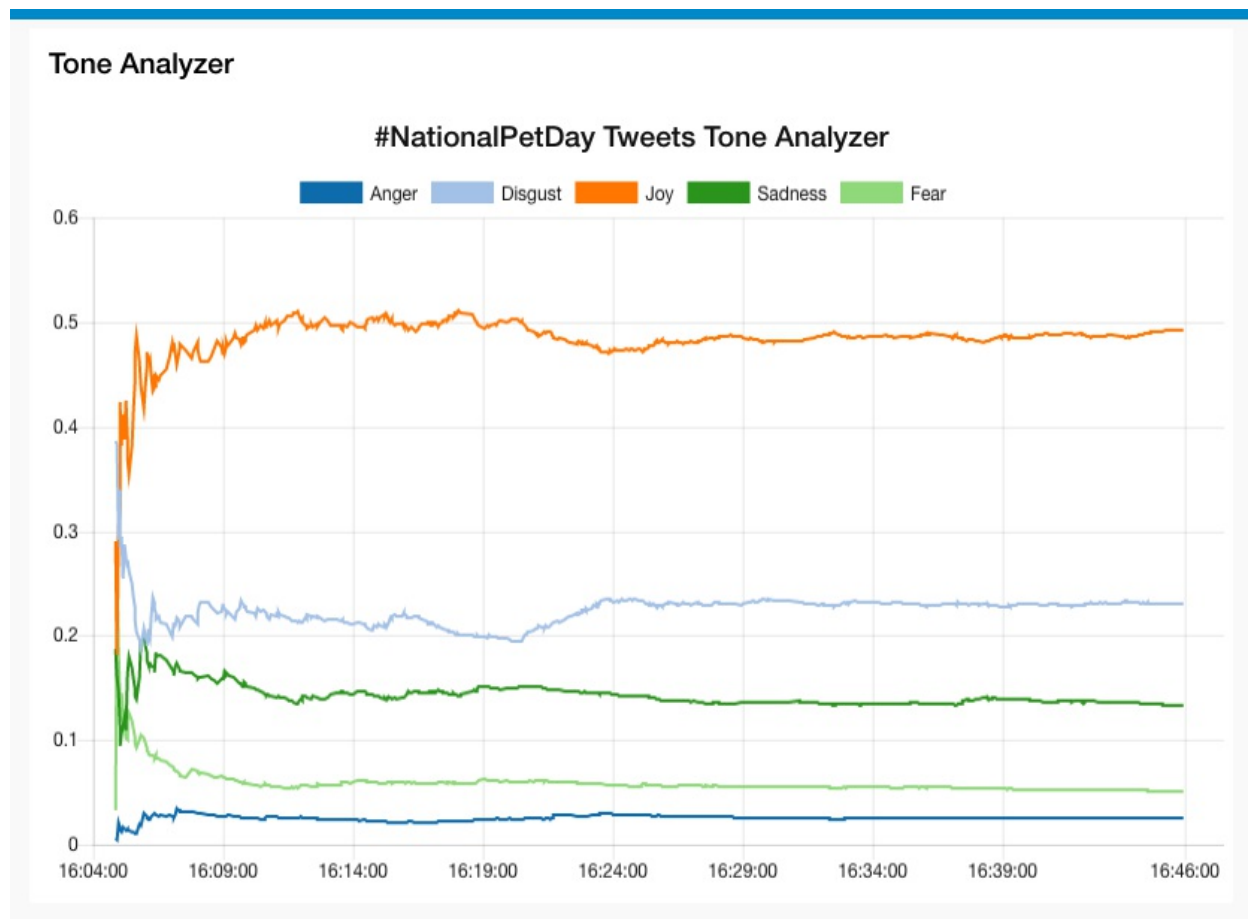
Take a note of where the `username` and `password` fields are, as you will need to copy/paste these values into the Node-RED Tone Analyzer node, shortly.

The basic application flow is based on the following Cognitive Application Reference

Architecture:



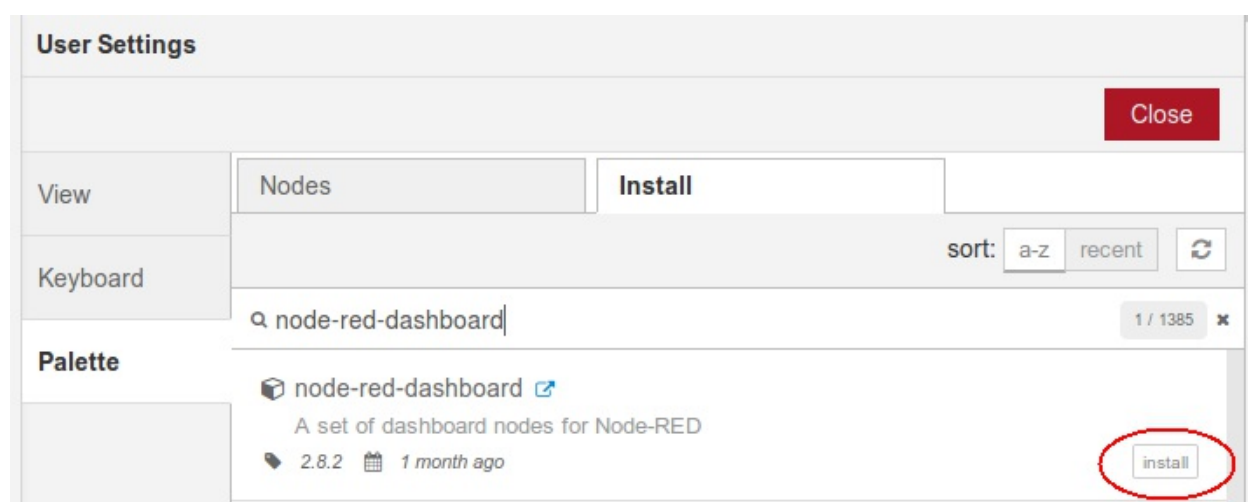
Let's say we want to know about people's general opinion on a specific topic on Twitter. We can use a `twitter in` node to obtain tweet feeds, then send the content to Watson Tone Analyzer service. When we get the analytic result, we will keep track of the result over time and visualize the average score of each index - similar to the following:



To produce the sample chart, you will need an additional package to be added to the Node-RED palette - `node-red-dashboard`.

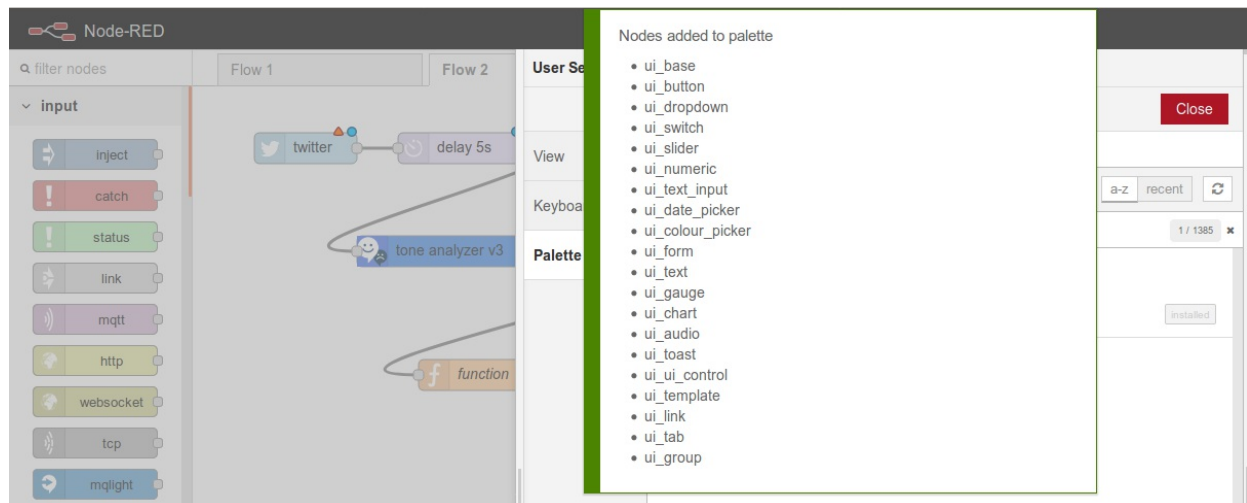
From the Node-RED menu  (top-right), select `Manage palette`.

Click on the `Install` tab and enter "node-red-dashboard" in the `search modules` field.



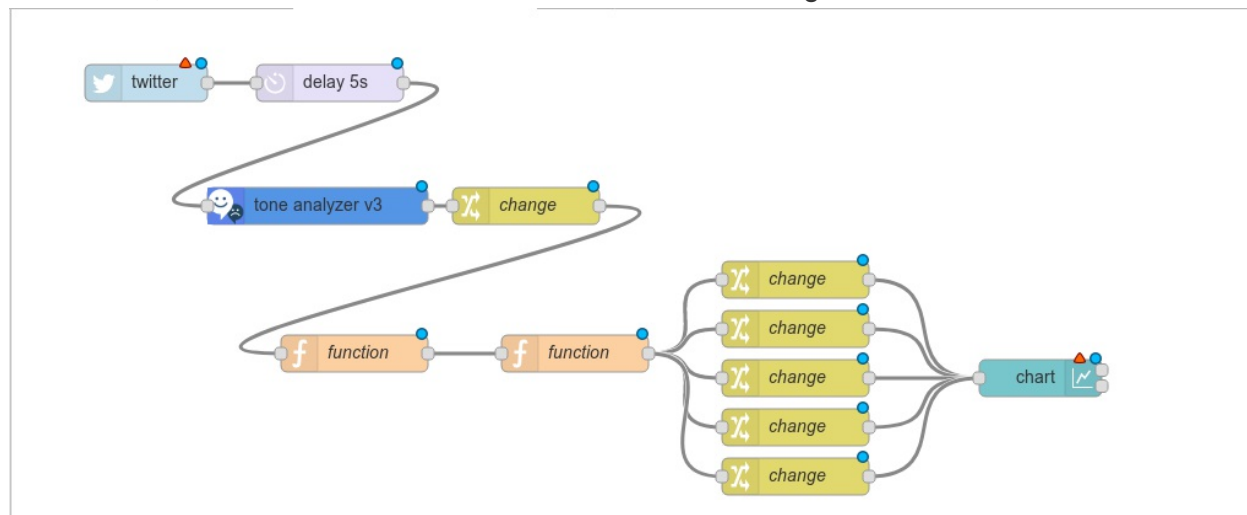
Click the `install` button.

All being well, you should see the additional nodes being dynamically added to the list of available nodes:



You'll find these new nodes towards the bottom of the left-side palette menu.

Once more, create a new flow on the canvas with the following nodes:



For this Lab we will use **#Trump** as the Twitter topic to monitor.

Since this topic is VERY active at the moment, we have to avoid the flood of the incoming tweets overwhelming the Node-RED instance. Add a `delay` node after the `twitter` node and setting the node action to "Limit rate to" 1 message per 1 seconds - this should leave sufficient time for the Watson Tone Analyzer service to handle each request.

You will need to fill in the service credentials (username and password) from the Watson Tone Analyzer service that you created earlier.

In this example, the purpose is to extract emotion tones, so select the `Emotion` option in the `Tones` field.


Edit tone analyzer v3 node


Delete


Cancel

Done


▼ node properties

 Name

 Username


 Password

☒ Use Default Service Endpoint

 Method:


General Tone

▼

 version_date:


Multiple Tones

▼

 Tones


Emotion

▼

 Sentences


True

▼

 Content type

Text

▼

 Input Text Language

English

▼

Since the result from the Watson Tone Analyzer will be in `msg.response` , we can add a `change` node to move `msg.response` to `msg.payload` to make it easier to process in subsequent nodes.

In a `function` node named “Add tweet scores to total”, the scores can be accumulated using the following code:


```

var defaultResult = {
  "emotion_tone":{
    "Anger":0,
    "Disgust":0,
    "Joy":0,
    "Sadness":0,
    "Fear":0
  },
  "count":0
}

if(msg.payload) {

  var result = context.get('twitterAnalysis')||defaultResult;

  msg.payload.document_tone.tone_categories.forEach(function(toneCategory){
    if(toneCategory.tones){
      toneCategory.tones.forEach(function(tone){
        result[toneCategory.category_id][tone.tone_name] += tone.score;
      })
    }
  })
  result.count += 1;

  context.set('twitterAnalysis', result);

  return {payload:result};
}

```

Here, we add the scores of each tone and then save into a context variable named “twitterAnalysis”. A track of the count is kept, for the next node to use in calculating averages.

The next `function` node calculates averages with this code:

```

if(msg.payload.count){

  msg.type = "newMsg";

  for (var toneCategory in msg.payload) {
    if (!msg.payload.hasOwnProperty(toneCategory)) continue;
    var obj = msg.payload[toneCategory];
    for (var prop in obj) {
      if(!obj.hasOwnProperty(prop)) continue;
      obj[prop] = obj[prop]/msg.payload.count;
    }
  }
}

return msg;

```


In order to show the scores of each tone, each score needs to be moved into different topics before sending to the dashboard `chart` node. We use the `change` node to do this; here is an example for the "Anger" score:

The screenshot shows the 'Edit change node' window. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below this is a section titled 'node properties' with a dropdown arrow. Under 'node properties', there is a 'Name' field with the value 'Show Anger'. Below the name field is a 'Rules' section with a list icon. The 'Rules' section contains two rule entries. Each entry has a 'Set' dropdown, a 'to' label, and a target field. The first rule has 'msg. topic' as the source and 'a_z Anger' as the target. The second rule has 'msg. payload' as the source and 'msg. payload.emotion_tone.Anger' as the target. At the bottom left of the 'Rules' section, there is a '+ add' button circled in red.

Note that the second `set` section has been added using the `+ add` button lower-left.

Modify the other `change` nodes to reflect "Joy", "Sadness", "Disgust" and "Fear"

Lab - twitter chatbot setup

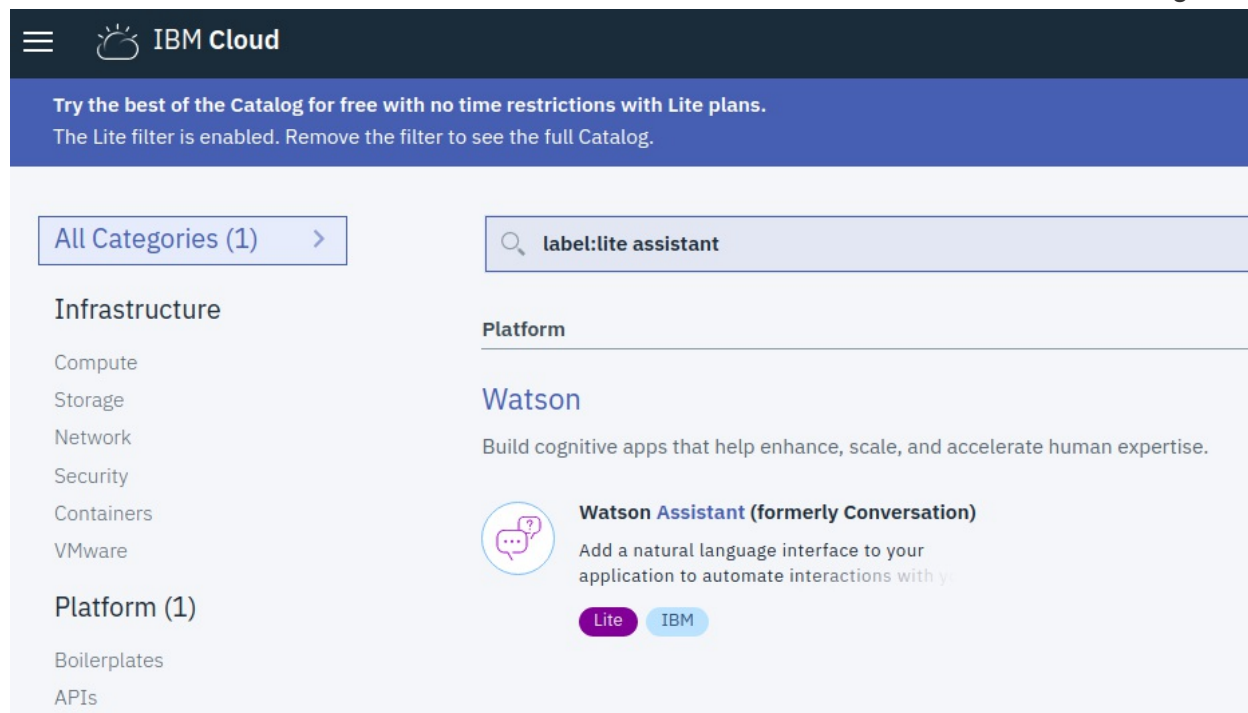
Now, a selection of the tweets can be directed to a chatbot service to trigger a response (based on the Watson Assistant service).

Firstly, get hold of an existing Chatbot configuration; for this lab, we will be using one of the Bot definitions in the [IBM Bot Asset Exchange](#)

This bot responds to queries with the lyrics and wisdom of renowned artists [Jay-Z](#).

Click the `Get this bot` option and save the resulting JSON string to a local file. This file will be imported into the Watson Assistant tool shortly.

You will need to create an instance of the Watson Assistant service in the IBM Cloud catalog -



As before, leave the name to default, create the instance, and then use the `Launch Tool` option to begin creating a Conversation workspace.

Catalog Docs

Manage
Service credentials
Plan
Connections

Watson /

Watson Assistant (formerly Conversation)-yb

Location: United Kingdom
Org: bmxrdc@icloud.com
Space: dev

Watson Assistant

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.

Launch tool

Developer resources:

- Getting started tutorial
- Demo

Select the **Create a Workspace** option

IBM Watson Assistant

Home
Workspaces

Introducing

IBM Watson Assistant

Watson Conversation is evolving to simplify how you build and scale virtual assistants. [See what's new](#)

Three easy steps

Follow these steps to create a virtual assistant.

1

Create intents and entities

Determine what your virtual assistant will understand by providing training examples so Watson can learn.

[Learn more](#)

2

Build your dialog

Utilize the intents and entities you created, plus context from the application, so your virtual assistant responds appropriately.

[Learn more](#)

Get started now

Create a Workspace

And *carefully* **NOT** clicking on the **Create** option, select the **Import workspace** icon



and navigate to your saved Jay-Z bot JSON file.



Import a workspace

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

Choose a file	<code>jayz-bot-1.json</code>
---------------	------------------------------

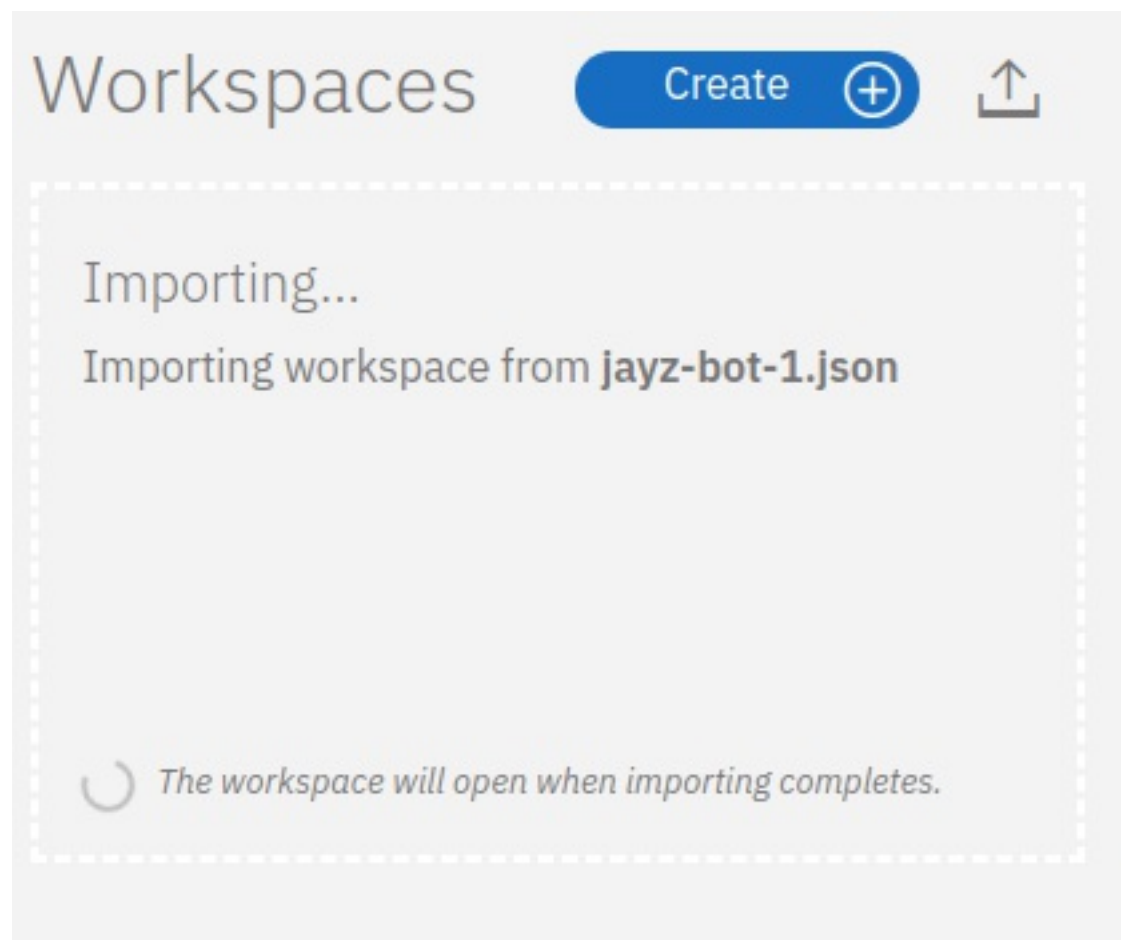
Import

☒ Everything (Intents, Entities, and Dialog)

☐ Intents and Entities

Import

This should result in a new workspace being created, using the definitions in the JSON file



This will create the 3 main areas of chatbot server configuration:

1. Intents:



[Workspaces](#) / Jay-Z Bot / Build



Intents

Entities

Dialog

Content Catalog



Add intent



☐ Intent (6) ▼

☐ #advice

☐ #Goodbye

☐ #greetings

☐ #success

☐ #where

☐ #who

2. Entities:

[Workspaces](#) / Jay-Z Bot / Build

Intents

Entities

Dialog

Content Catalog

**My entities**

System entities

Add entity



Entity (8) ▼

Values



@background

from



@business

business



@kids

child, fatherhood



@life

life



@lyrically

skills, flow, lyrically



@relationship

partner



@society

society, taxes



@wealth

wealth

3. Dialog:

IBM Watson Assistant

Workspaces / Jay-Z Bot / Build

IntentsEntitiesDialogContent Catalog

Add nodeAdd child nodeAdd folder

Jay-Z Bot

Welcome
welcome
1 Response / 0 Context set / Does not return

Conversation end
#Goodbye
1 Response / 0 Context set / Does not return

life
@life
1 Response / 0 Context set / Does not return

relationship
@relationship
1 Response / 0 Context set / Does not return

business
@business
1 Response / 0 Context set / Does not return

Using the existing Node-RED flow, select one of the tone category streams from the twitter analysis [Joy, Anger, Disgust, Sadness, Fear], to direct into the Chatbot, using the Watson



Conversation node

You will need the credentials for the Watson Assistant instance, to plug into the node configuration menu. Either:

- make a connection between the Watson Assistant instance, and your Node-Red application, and after re-stage, the credentials will automatically populate the Watson Conversation nodes
- copy the credentials from the Watson Assistant instance, and apply directly to the node configuration, and use straight away.

The screenshot shows the IBM Cloud console interface for a Watson Assistant instance. The left sidebar contains navigation links: Manage, Service credentials (selected), Plan, and Connections. The main content area is titled 'Watson Assistant (formerly Conversation)-yb' and shows the instance details: Location: United Kingdom, Org: bmxrdc@icloud.com, Space: dev.

Under the 'Service credentials' section, there is a description: '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.' A 'View More' link is present.

Below this, there is a table of service credentials. The table has columns: KEY NAME, DATE CREATED, and ACTIONS. There is one entry with the key name 'conversation_tooling_key1522769880516' and a date of 'Apr 3, 2018 - 04:39:01'. The ACTIONS column for this entry has a 'View credentials' link.

The 'View credentials' link opens a modal window showing a JSON snippet:

```
{
  "url": "https://gateway.watsonplatform.net/assistant/api",
  "username": "f70becdc-a251-4d96-bf06-653481bd2c13",
  "password": "dzFdpMTYCuHT"
}
```

A 'Let's talk' button is visible in the bottom right corner of the modal.

Challenge

Congrats!
