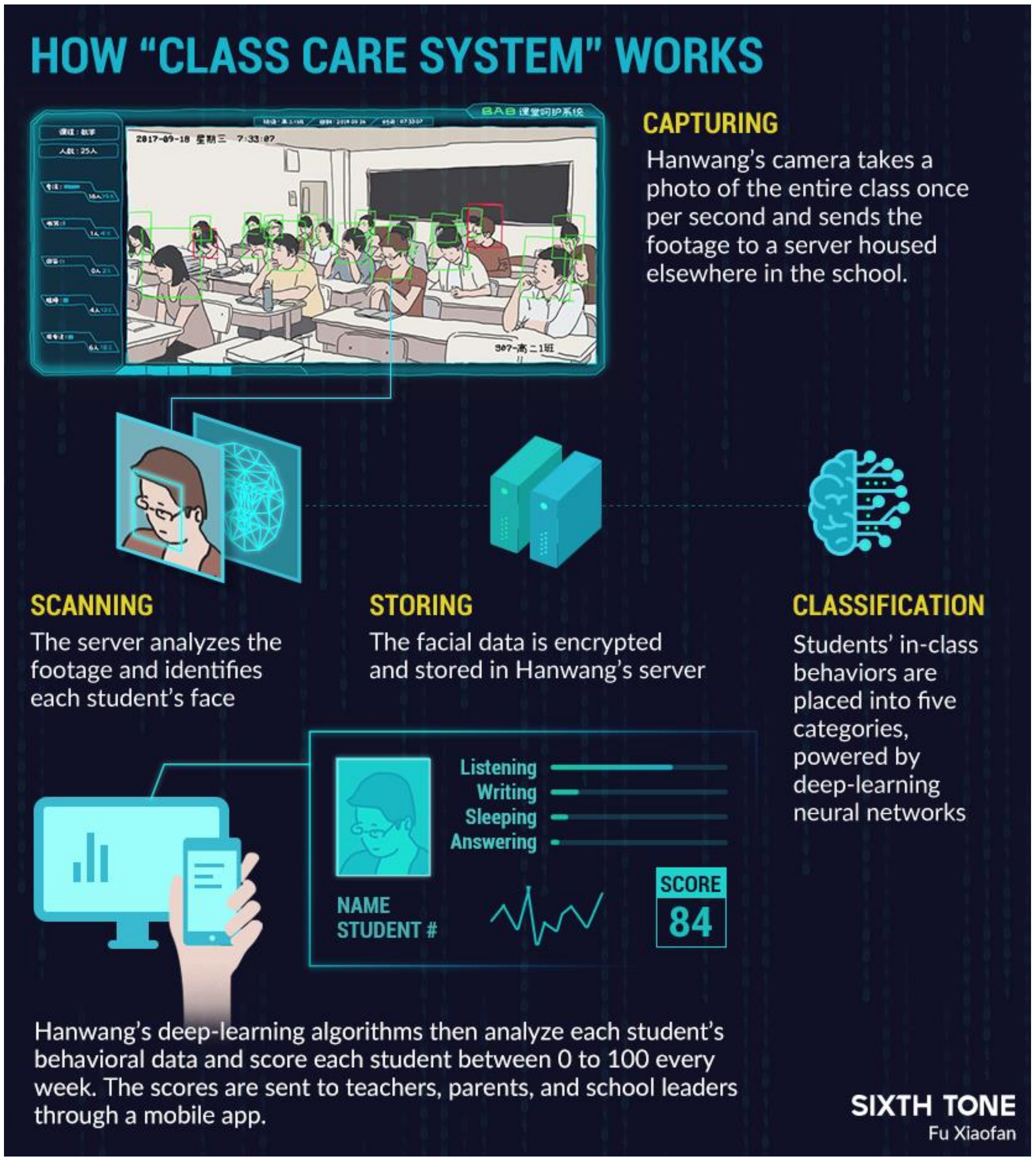
IBM Skills Academy

Lab Guide

How well Can AI Tell Your Age from your Pic?

Lab 3: Face Recognition

Face scanners that log employee work hours or smart billboards that tailor their advertisements to audience gender, how about intelligent education. In July 2017, China's highest governmental body, the State Council, released an ambitious policy initiative called the Next Generation Artificial Intelligence Development Plan (NGAIDP).
Class Care System (CCS) is one of the flagship products of Hanwang Education, a subsidiary of Hanwang Technology. Hanwang is a household name for China's younger generation. Growing up, they watched catchy commercials for the company's text-to-speech reading pen on television. Today, Hanwang builds hardware and software products that provide facial and biometric recognition services and optical character recognition, as well as air quality monitors and purifiers. Hanwang Education was founded in 2014 as part of an initiative to expand the company's market into China's education sector.

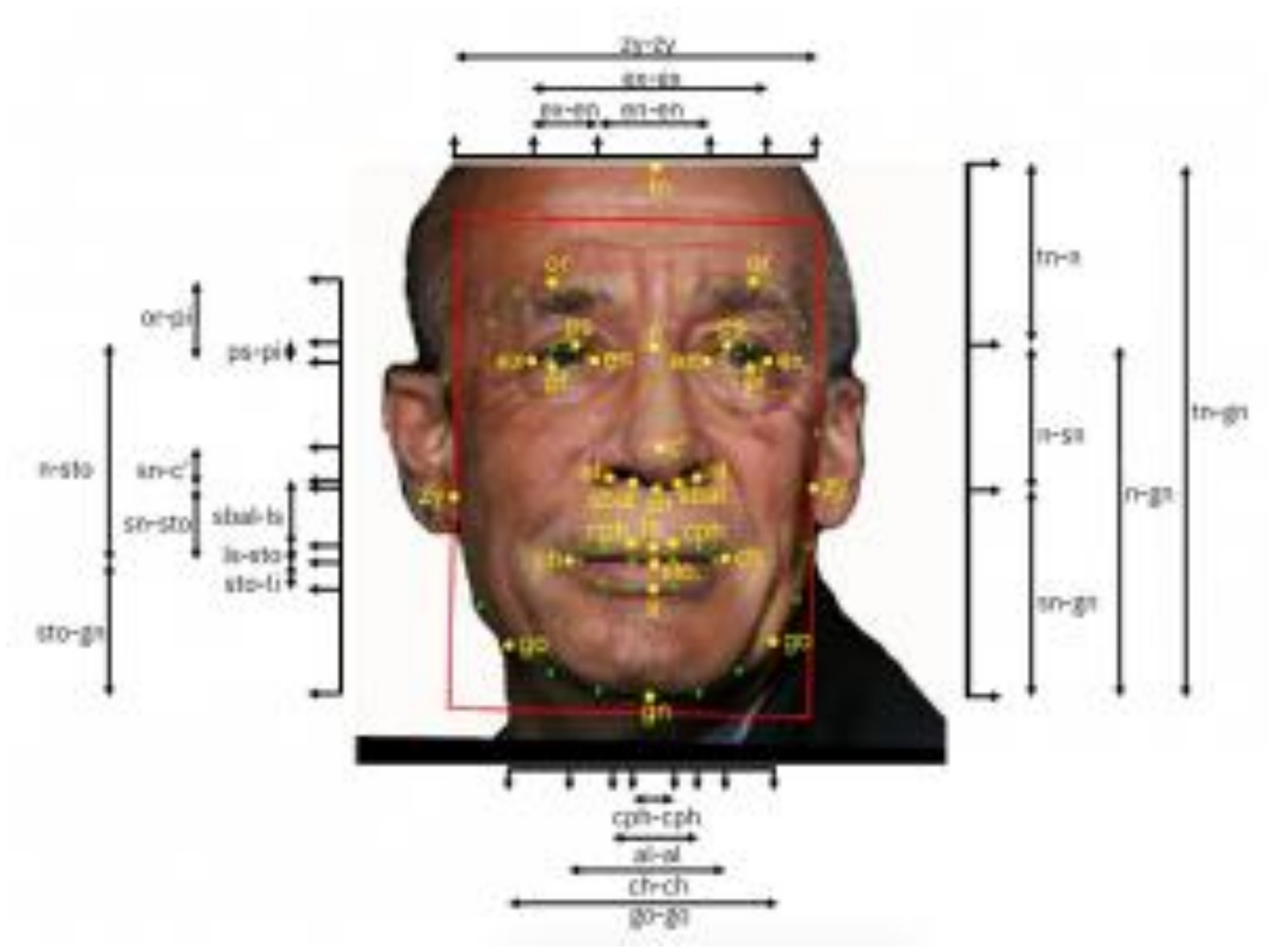

HOW "CLASS CARE SYSTEM" WORKS
CAPTURING
Hanwang's camera takes a photo of the entire class once per second and sends the footage to a server housed elsewhere in the school.
SCANNING
The server analyzes the footage and identifies each student's face
STORING
The facial data is encrypted and stored in Hanwang's server
CLASSIFICATION
Students' in-class behaviors are placed into five categories, powered by deep-learning neural networks
Listening
Writing
Sleeping
Answering
NAME
STUDENT #
SCORE
84
Hanwang's deep-learning algorithms then analyze each student's behavioral data and score each student between 0 to 100 every week. The scores are sent to teachers, parents, and school leaders through a mobile app.
SIXTH TONE
Fu Xiaofan

http://www.sixthtone.com/news/1003759/camera-above-the-classroom?_branch_match_id=443791358268277735#_=_

John R. Smith an IBM Fellow, in his January 29 publication, wrote that much of the power of AI today comes from the use of data-driven deep learning to train increasingly accurate models by using growing amounts of data. However, the strength of these techniques can also be a weakness. The AI systems learn what they're taught, and if they are not taught with robust and diverse datasets, accuracy and fairness could be at risk. For that reason, IBM, along with AI developers and the research community, need to be thoughtful about what data we use for training. IBM remains committed to developing AI systems to make the world more fair.

Today, IBM Research is releasing a new large and diverse dataset called Diversity in Faces (DiF) to advance the study of fairness and accuracy in facial recognition technology. The first of its kind available to the global research community, DiF provides a dataset of annotations of 1 million human facial images. Using publicly available images from the YFCC-100M Creative Commons data set, we annotated the faces using 10 well-established and independent coding schemes from the scientific literature [3-12]. The coding schemes principally include objective measures of human faces, such as craniofacial features, as well as more subjective annotations, such as human-labeled predictions of age and gender. We believe by extracting and releasing these facial coding scheme annotations on a large dataset of 1 million images of faces, we will accelerate the study of diversity and coverage of data for AI facial recognition systems to ensure more fair and accurate AI systems. Today's release is simply the first step.

We believe the DiF dataset and its 10 coding schemes offer a jumping-off point for researchers around the globe studying the facial recognition technology. The 10 facial coding methods include craniofacial (e.g., head length, nose length, forehead height), facial ratios (symmetry), visual attributes (age, gender), and pose and resolution, among others. These schemes are some of the strongest identified by the scientific literature, building a solid foundation to our collective knowledge.



Our initial analysis has shown that the DiF dataset provides a more balanced distribution and broader coverage of facial images compared to previous datasets. Furthermore, the insights obtained from the statistical analysis of the 10 initial coding schemes on the DiF dataset has furthered our own understanding of what is important for characterizing human faces and enabled us to continue important research into ways to improve facial recognition technology.

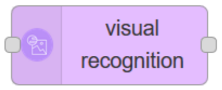
Preface

The IBM Watson™ Visual Recognition service uses deep learning algorithms to analyze images for scenes, objects, faces, and other content. The response includes keywords that provide information about the content.
This lab will show you how to build an application that uses the Visual Recognition service analyzing the contents of an image and extract features from it. The Face Detection service can identify multiple faces within the image and determine their gender and age with a confidence score. The application will be built using an Open Source tool called Node-RED.

You can find more labs to perform by referring to Github:
https://github.com/watson-developer-cloud/node-red-labs

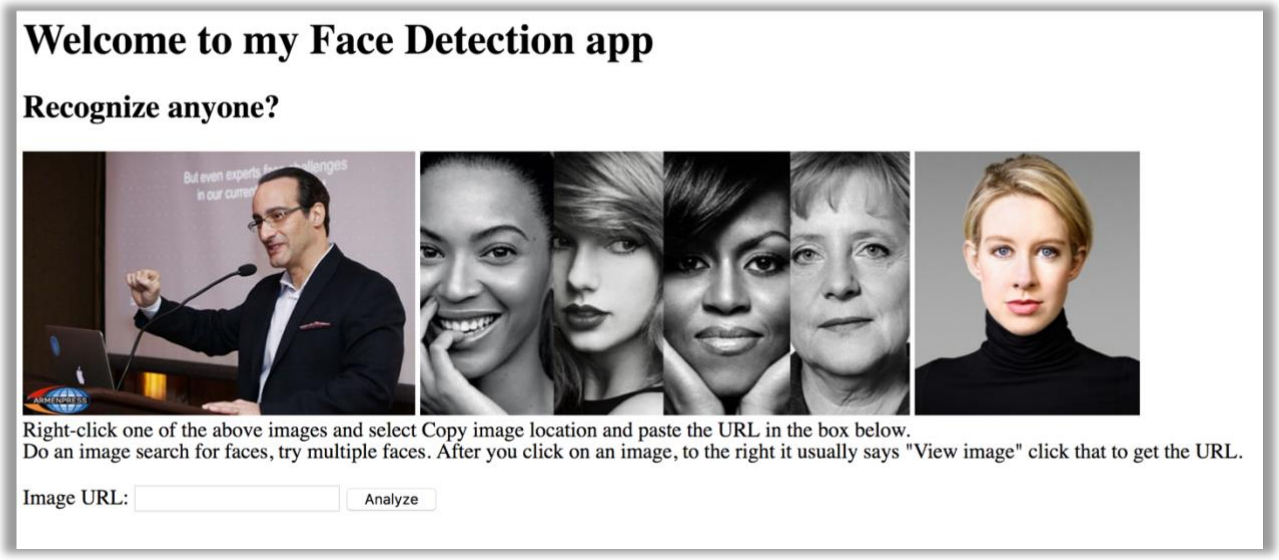
About Node-RED

Node-RED is a visual tool for wiring the Internet of Things. It is easy to connect devices, data and APIs (services). It can also be used for other types of applications to quickly assemble flows of services. Node-RED is available as open source and has been implemented by the IBM Emerging Technology organization. Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click. While Node-Red is based on Node.js, JavaScript functions can be created within the editor using a rich text editor. A built-in library allows you to save useful functions, templates or flows for re-use.
Node-RED is included in the Node-RED starter application in IBM Cloud but you can also deploy it as a stand-alone Node.js application. Node-RED is not just used for IoT applications, but it is a generic event-processing engine. For example, you can use it to listen to events from http, web sockets, TCP, Twitter and more and store this data in databases without having to program much if at all. You can also use it for example to implement simple REST APIs.
The following table explains the nodes that you will use in this lab.

Node name	Description
	The **http in** node provides an input node for http requests, allowing the creation of simple web services. The resulting message has the following properties: msg.req : http request msg.res : http response For POST/PUT requests, the body is available under msg.req.body This uses the Express bodyParser middleware to parse the content to a JSON object. By default, this expects the body of the request to be URL encoded: foo=bar&this=that To send JSON encoded data to the node, the content-type header of the request must be set to application/json. Note: This node does not send any response to the http request use a subsequent HTTP Response node.
	The **http response** node can send responses back to http requests received from an HTTP Input node. The response can be customized using the following message properties: payload is sent as the body of the response StatusCode if set, is used as the response status code (default: 200) headers if set, should be an object containing field/value pairs to be added as response headers.
	With the **change** node you can set, change or delete properties of a message. The node can specify multiple rules that will be applied to the message in turn. The available operations are: Set Sets a property. The to property can either be a string value, or reference another message property by name, for example: msg.topic. Change search & replace parts of the property. If regular expressions are enabled, the replace with property can include capture groups, for example $1 Delete deletes a property.
	The **switch** node is a simple function node that routes messages based on its properties. When a message arrives, the selected property is evaluated against each of the defined rules. The message is then sent to the output of all rules that pass. Note: the otherwise rule applies as a "not any of" the rules preceding it.
	The Visual Recognition node provides a very easy wrapper node that takes an image URL or binary stream as input, and produces an array of detected faces, age, bounding box, gender and name.
	The template node creates a new message based on the provided template. This uses the mustache format. For example, when a template of: Hello {{name}}. Today is {{date}} receives a message containing: { name: "Fred", date: "Monday" payload: ... } The resulting payload will be: Hello Fred. Today is Monday

Section 1. About your application

In this exercise, we will show how to generate the face recognition data from an image URL. The flow will present a simple Web page with a text field where to input the image's URL, then submit it to the service, and output the faces that have been found on the reply Web page.
Use the flow below as a guideline as you drag and drop nodes onto the canvas. Connect the nodes at the end and always refer to the flow depicted below for logical placement of the nodes onto the canvas.
Click this link to view an already built app: https://manyfaces.mybluemix.net/manyofus

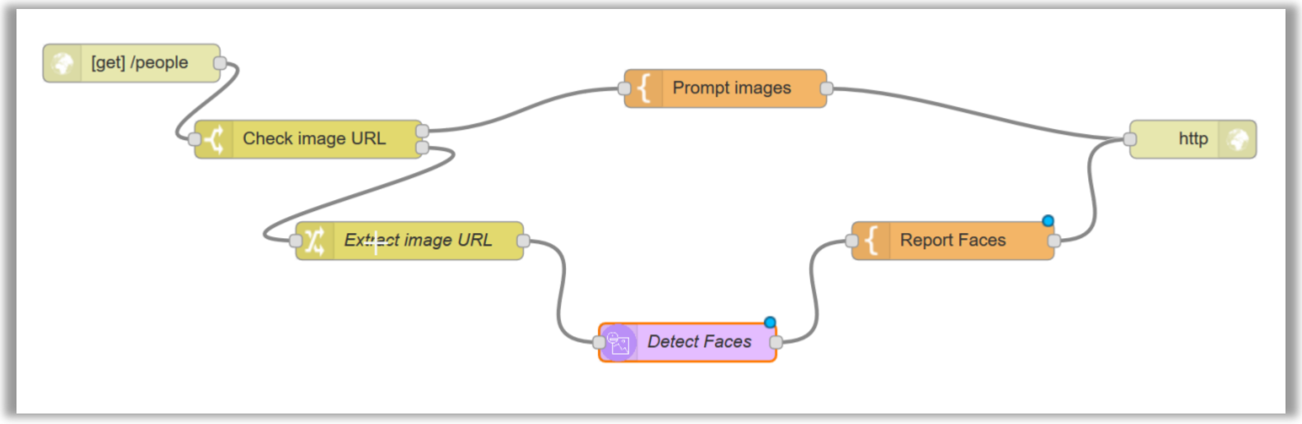


Welcome to my Face Detection app

Recognize anyone?

Right-click one of the above images and select Copy image location and paste the URL in the box below.
Do an image search for faces, try multiple faces. After you click on an image, to the right it usually says "View image" click that to get the URL.

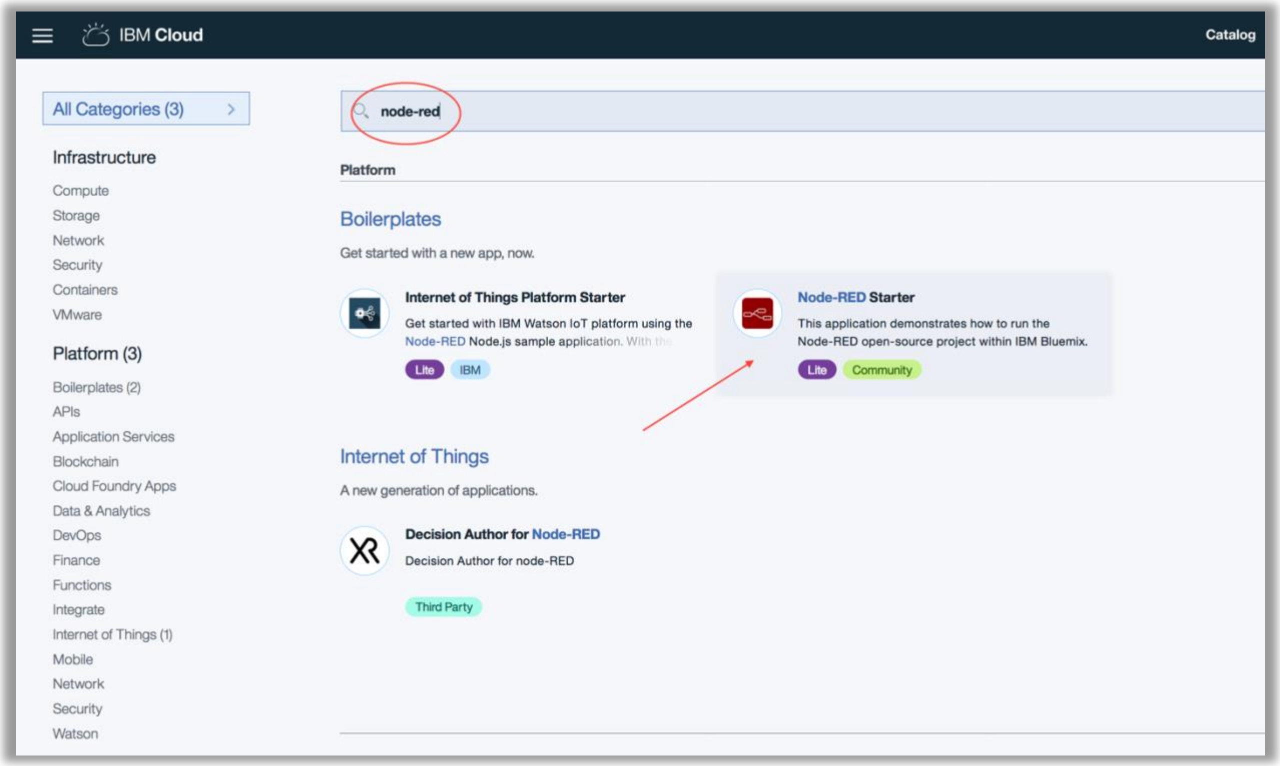
You will build the app from scratch using Node-RED as depicted in the image below:



Build a Node-RED Starter Boilerplate

This lab assumes that you have a IBM Cloud account, you have signed in and applied the promocode. You can register for IBM Cloud by clicking the **SIGN UP** button in the upper right corner of the page at console.ng.bluemix.net. After registering, you will receive an email message that requires you to confirm your registration.

1. Sign into IBM Cloud: https://console.bluemix.net/catalog/
2. Obtain and apply the promo code to your IBM Cloud.
3. From the IBM Cloud console, access the **Catalog** tab and search for **Node-RED Starter**.



Boilerplates

Get started with a new app, now.

Internet of Things Platform Starter

Get started with IBM Watson IoT platform using the Node-RED Node.js sample application. With th

Node-RED Starter

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

Internet of Things

A new generation of applications.

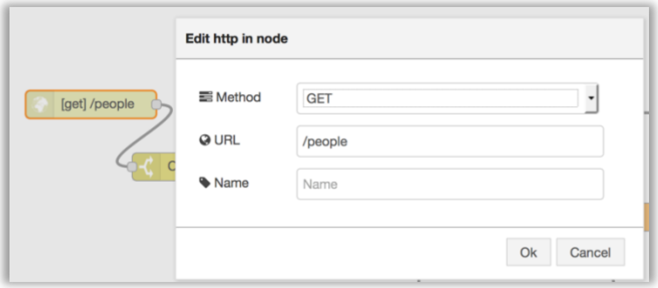
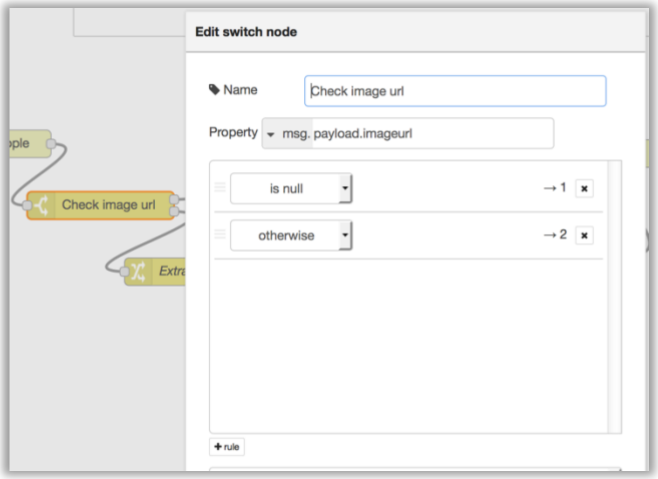
Decision Author for Node-RED

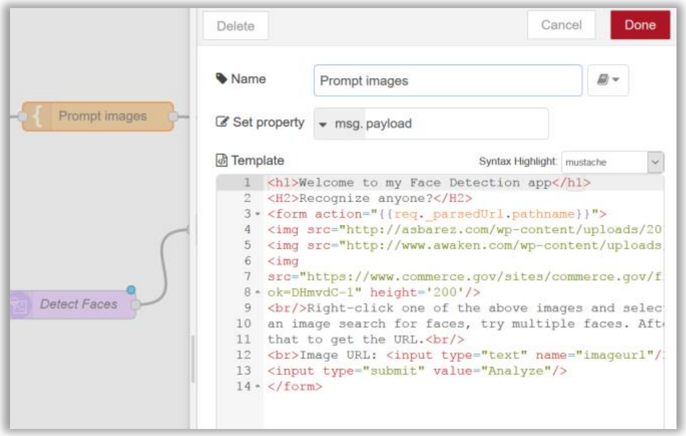
Decision Author for node-RED

4. Specify a unique name for your app (in this example, **manyfaces**) and click **Create**. Allow enough time for the app to stage and start. This may take a few minutes.
5. Click the URL in top left, in this example, it is *manyfaces.mybluenix.net*
6. In the ensuing page, advance through the wizard (give it a username/password).
7. Click **Next** without selecting any of the provided starters; and click **Finish**.
8. In the ensuing page, click **Go to your Node-Red flow editor**.
9. Login with the username/password you created earlier in Step 6.

Section 2. Populate the Node-RED canvas

The remaining steps that pertain to building your node-RED canvas are outlined in the table below.

Steps	Example screen capture
10. Drag and drop an **http in** node. Keep an eye on the completed flow in the previous page as you add nodes. It can help you identify the nodes easily and roughly where to place them on the canvas. 11. For the Method, select **GET** and for the URL, specify any context name, in this example **/people** (ensure to place the **/** before the context name). 12. Click **OK.**	 Edit http in node Method: GET URL: /people Name: Name
13. Drag and drop a **switch** node, which will test for the presence of the **imageurl** query parameter. Use the search box to find these nodes easily. 14. For name, specify any value, in this example, **Check image url**. 15. In the property box, append **imageurl** after payload 16. Set two conditions from the drop-down list, first **is null** and second is **otherwise**. 17. Click **OK**.	 Edit switch node Name: Check image url Property: msg. payload.imageurl is null → 1 otherwise → 2

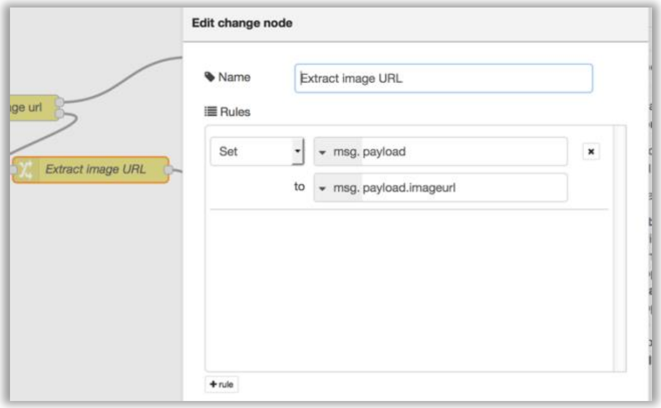
18. Drag and drop the **template** node, configured to output an HTML input field and suggest a few selected images taken from official sources.
19. Specify a name, for example: **Prompt images**
20. Set the property to **msg.payload** (if does not appear as such by default. Use the drop-down list instead of typing.
21. Type **payload** after the msg (no spaces, again if it does not appear as such by default).
 	
The Function node allows JavaScript code to run against the messages that are passed in and then return zero or more messages to continue the flow.
	
	Name: Prompt images
	Set property: msg. payload
	Template — Syntax Highlight: mustache
	`<h1>Welcome to my Face Detection app</h1>`
	`<H2>Recognize anyone?</H2>`
	`<form action="{{req._parsedUrl.pathname}}">`
	`<img src="http://asbarez.com/wp-content/uploads/20`
	`<img src="http://www.awaken.com/wp-content/uploads`
	`<img`
	`src="https://www.commerce.gov/sites/commerce.gov/f`
	`ok=DHmvdC-1" height='200'/>`
	`<br/>Right-click one of the above images and selec`
	`an image search for faces, try multiple faces. Aft`
	`that to get the URL.<br/>`
	`
	Image URL: <input type="text" name="imageurl"/`
	`<input type="submit" value="Analyze"/>`
	`</form>`

The message is passed in as an object called msg. By convention it will have a msg.payload property containing the body of the message.

22. Copy/paste this code in the body of the template overriding the existing template line of code:

```
<h1>Welcome to my Face Detection app</h1>
<H2>Recognize anyone?</H2>
<form action="{{req._parsedUrl.pathname}}">
   <img src="http://asbarez.com/wp-content/uploads/2016/02/ibmwatson.jpg" height='200'/>
   <img src="http://www.awaken.com/wp-content/uploads/2015/05/forbes.jpg" height='200'/>
   <img src="https://www.commerce.gov/sites/commerce.gov/files/styles/scale_250w/public/media/images/profile/elizabethholmes.jpg?itok=DHmvdC-1" height='200'/>
      <br/>Right-click one of the above images and select Copy image location and paste the URL in the box below.<br>Do an image search for faces, try multiple faces. After you click on an image, to the right notice: "View image" click that to get the URL.<br/>
   <br>Image URL: <input type="text" name="imageurl"/>
   <input type="submit" value="Analyze"/>
</form>
```

23. Drag and Drop a **change** node to extract the **imageurl** query parameter from the web request and assign it to the payload to be provided as input to the Visual Recognition node.
24. Specify a name.
25. Select **Set** for rule; the first rule is **msg.payload**, and the second rule is **msg.payload.imageurl** (use the drop-down list to select the **msg** part and type the rest).

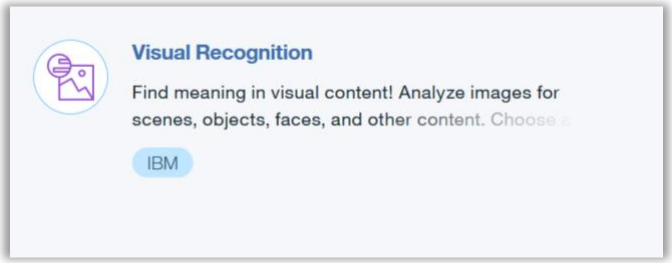

Edit change node

Name: Extract image URL

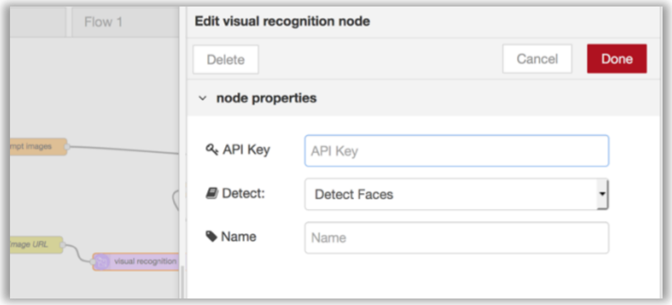
Rules: Set msg. payload to msg. payload.imageurl

If you applied the promo code, then follow these steps:

1. Drag and drop a **Visual Recognition** node.
2. Go to IBM Cloud, search for the **Visual Recognition** service.
3. Give the service a unique name after the dash. For example, **Visual Recognition-<*unique_name*>**
4. Click **Create**. Allow enough time.
5. Click **Service Credentials** from the left pane.
6. Click **View credentials**.
7. Copy the api_key value (without the double quotes).
8. Back to the Node-RED canvas
   a. Paste the API key.
   b. For Detect annotators, select **Detect Faces**.
   c. For name, enter **Detect Faces** (optional)
9. Click **Done**.


Visual Recognition

Find meaning in visual content! Analyze images for scenes, objects, faces, and other content. Choose a


Edit visual recognition node

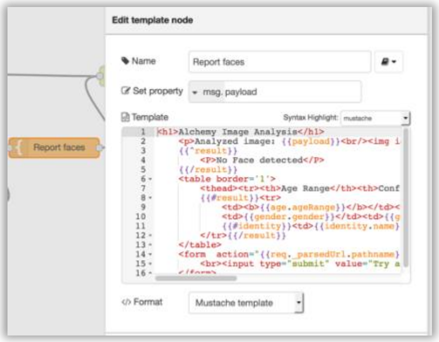
node properties

API Key: API Key

Detect: Detect Faces

Name: Name

10. Drag and drop a **template** node with the following content, which will format the output returned from the Image Analysis node into an HTML table for easier reading. Override the existing line of code.



Edit template node

Name: Report faces

Set property: msg. payload

Template — Syntax Highlight: mustache

```
<h1>Alchemy Image Analysis</h1>
    <p>Analyzed image: {{payload}}<br/><img i
    {{^result}}
        <P>No Face detected</P>
    {{/result}}
    <table border='1'>
        <thead><tr><th>Age Range</th><th>Conf
        {{#result}}<tr>
            <td><b>{{age.ageRange}}</b></td><
            <td>{{gender.gender}}</td><td>{{g
            {{#identity}}<td>{{identity.name}
        </tr>{{/result}}
    </table>
    <form  action="{{req._parsedUrl.pathname}
        <br><input type="submit" value="Try a
```

Format: Mustache template

11. Copy/paste this code inside the template frame by replacing the existing: This is the payload: {{payload}} ! code.

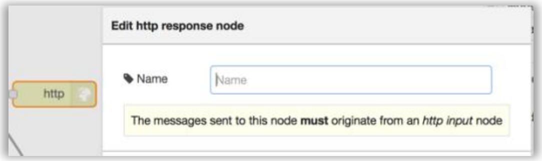
```
<h1>Visual Recognition</h1>

<!-- parse all images -->
{{#result.images}}
<p>Analyzed image: {{source_url}}<br/><img id="alchemy_image" src="{{source_url}}" height="200"/></p>

{{^faces}}
<P>No Face detected</P>
{{/faces}}

<table border='1'>
<thead><tr><th>Age Range</th><th>Confidence</th><th>Gender</th><th>Confidence</th><th>Name</th></tr></thead>
<!-- parse all faces -->
{{#faces}}
<tr>
<td><b>{{age.min}} - {{age.max}}</b></td><td><i>{{age.score}}</i></td>
<td>{{gender.gender}}</td><td>{{gender.score}}</td>
{{#identity}}<td>{{identity.name}} ({{identity.score}})</td>{{/identity}}
</tr>{{/faces}}
</table>
{{/result.images}}
<form action="{{req._parsedUrl.pathname}}">
<br><input type="submit" value="Try again or go back to the home page"/>
</form>
```

12. End the flow with the **http response** node and connect the nodes as you see depicted on page 4.

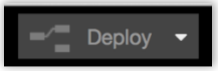


Edit http response node

Name: Name

The messages sent to this node **must** originate from an *http input* node

13. Click **Deploy**.



14. To run the web page, copy and paste the entire URL as it appears on your Node-RED web page onto another tab in your browser.
15. In the URL, backspace all the way until the .net/
16. Append the context name that you specified in the **Http In** node as a GET function, in this example: *manyofus* https://manyfaces.mybluemix.net/**manyofus**
17. Press Enter and you should see the web page.

18. From Google Images, select a person or group of people. Note: Click **View image** to obtain the exact link to that image. 19. Copy the address location that ends with jpg or png image file onto the Submit box of your app. Notice, that the Face Recognition app reveals the age range and sex of the person with corresponding confidence values. Congratulation, you have just built a face recognition app and now, take your time and edit the front page of your app to depict your words and your images, or just a single image. Hint, that is the template node, Step 22.	