IMAGE RECOGNITION SYSTEM USING IBM CLOUD VISUAL RECOGNITION.

Introduction :

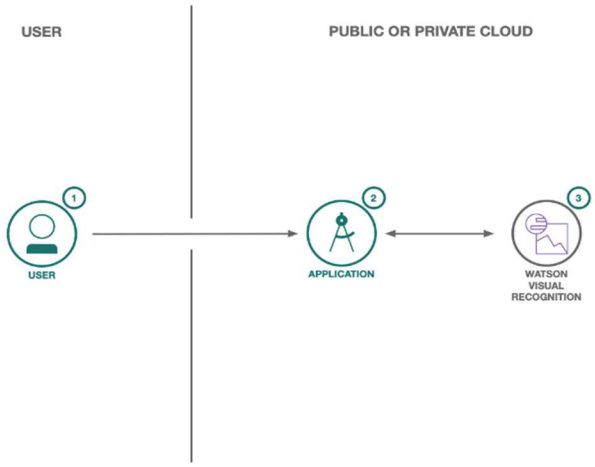
This is an image recognition system using IBM Cloud Visual Recognition. It is a tool that uses deep learning algorithm to analyze images and allow users to automatically identify subjects and objects contained within the image and organize and classify these images into categories.

To create a image recognition using IBM Cloud, including setting up an IBM Cloud account, creating a Visual Recognition service, creating IBM Cloud. In this integration, IBM Cloud Functions serves as the event-driven architecture where you can trigger image recognition functions in response to events such image uploads. When an event is triggered, it invokes a serverless function, which then utilizes Visual Recognition to analyze and interpret the images.

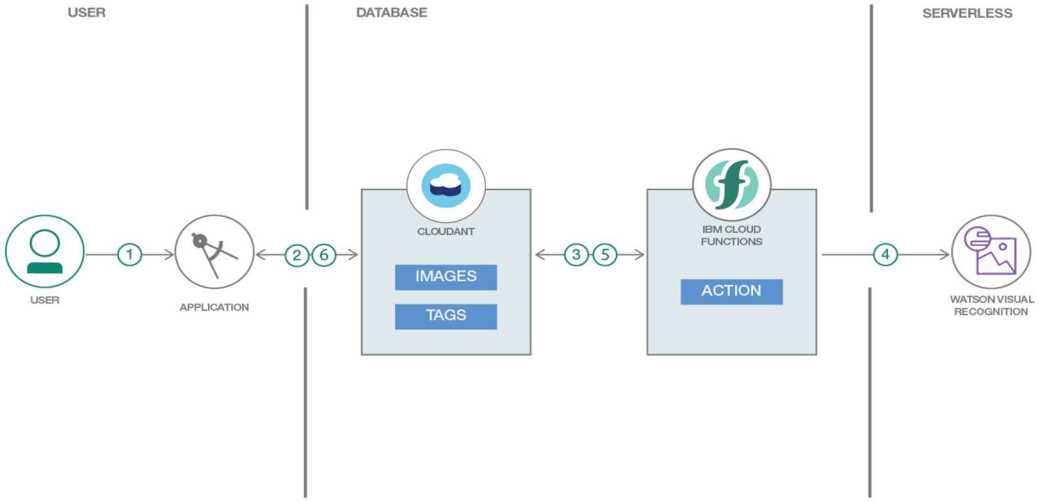
- Create and Deploy Cloud Functions
- Trigger Cloud Functions with Cloudant changes
- Use Watson Image Recognition with Cloud Functions

Flow:

1. User chooses a picture from the gallery.
2. The image is stored in the Cloudant database.
3. Cloud Function is triggered when there's a new image in the database.
4. Cloud Function gets the image and uses Watson Visual Recognition to process the image.
5. Cloud Function stores the results (classes with scores) from Visual Recognition in the database.
6. The user can see the new tags or classes in the image they uploaded.



Architecture:



Included Components:

- IBM Cloud Functions (powered by Apache OpenWhisk): Execute code on demand in a highly scalable, serverless environment.
- Cloudant: A fully managed data layer designed for modern web and mobile applications that leverages a flexible JSON schema.
- Watson Visual Recognition: Visual Recognition understands the contents of images - visual concepts tag the image, find human faces, approximate age and gender, and find similar images in a collection.

Prerequisites:

- IBM Cloud Functions CLI to create cloud functions from the terminal. Make sure you do the test action ibmcloud wsk action invoke /whisk.system/utils/echo -p message hello --result so that your ~/.wskprops is pointing to the right account.
- Whisk Deploy *(wskdeploy)* is a utility to help you describe and deploy any part of the OpenWhisk programming model using a Manifest file written in YAML. You'll use it to deploy all the Cloud Function resources using a single command. You can download it from the releases page and select the appropriate file for your system.

  brew install wskdeploy

- Install Node.js if you want to use Electron.

Steps :

➢ **To start with Image Recognition, you will need:**

- An IBM Cloud account (you can sign up here)
- Python 3.8 and pip

In this part, you will learn to run image recognition code on your machine and extract the required result from the JSON response.

➢ **Clone the repo**

$ git clone https://github.com/IBM/serverless-image- recognition

➢ **Create an IBM Cloud services**

Create a **Cloudant** instance and choose Use both legacy credentials and IAM for the *Available authentication method* option.

- Create credentials for this instance and copy the username and password in the local.env file in the value of CLOUDANT_USERNAME and CLOUDANT_PASSWORD.
- Launch the Cloudant web console and create a database named images and tags. Create Cloudant credentials using the IBM Cloud dashboard and place them in the local.env file.

Create a Watson Visual Recognition instance.

- Copy the API Key in the Credentials section and paste it in the local.env file in the value of WATSON_VISUAL_APIKEY

➢ **Deploy Cloud Functions**

Create 2 databases in cloudant:

1. images
2. tags

Deploy through the IBM Cloud Functions console user interface

Choose "Start Creating" in the IBM Cloud Functions Dashboard. Then proceed to this deployment instructions using the UI.

You can also deploy them directly from the CLI by following the steps in the next section.

Deploy using the wskdeploy command line tool

This approach deploy the Cloud Functions with one command driven by the runtime-specific manifest file available in this repository.

Make sure you have the right environment variables in the local.env file. Export them in your terminal then deploy the Cloud Functions using wskdeploy. This uses the manifest.yaml file in this root directory.

```
$ source local.env
$ wskdeploy
```

Launch Application

Configure web/scripts/upload.js. Modify the lines for your Cloudant credentials.

```
let usernameCloudant = "YOUR_CLOUDANT_USERNAME"
let passwordCloudant = "YOUR_CLOUDANT_PASSWORD"
```

Run the Electron app or open the html file

- Electron:

```
- $ npm install
- $ npm start
```

Tests :

Unit test

Run unit tests with:

```
npm run test:components
```

Integration tests

First you have to make sure your code is built:

```
npm run build
```

Then run integration tests with:

```
npm run test:integration
```

Program:

cache: pip

```
before_install:
  - sudo apt-get install shellcheck
  - sudo pip install yamllint
  - git clone https://github.com/IBM/pattern-ci

before_script:
  - "./pattern-ci/tests/shellcheck-lint.sh"
  - "./pattern-ci/tests/yaml-lint.sh"
```

```
    jobs:
      include:
  - script: echo "Lints passed."
```

local :

```
    # Customize as needed
    export CLOUDANT_IMAGE_DATABASE=images
    export CLOUDANT_TAGS_DATABASE=tags
    export CLOUDANT_USERNAME=
    export CLOUDANT_PASSWORD=
export WATSON_VISUAL_APIKEY=
```

manifest :

```
    # Manifest for serverless-image-recognition
    # Repo is located a t
```
https://github.com/IBM/serverless-image-recognition

```
    # Deployment using this manifest file creates following OpenWhisk components:
    #   Package:    serverless-image-recognition
    #   Action:     update-document-with-watson
    #   Trigger:    update-trigger
    # This manifest file expects following env. variables:
    #   CLOUDANT_IMAGE_DATABASE
    #   CLOUDANT_TAGS_DATABASE
    #   CLOUDANT_USERNAME
    #   CLOUDANT_PASSWORD
    #   WATSON_VISUAL_APIKEY
```

```
    packages:
      serverless-image-recognition:
        dependencies:
          # binding cloudant package named openwhisk-cloudant
          serverless-pattern-cloudant-package:
```

```
        location: /whisk.system/cloudant
        inputs:
          username: $CLOUDANT_USERNAME
          password: $CLOUDANT_PASSWORD
          host: ${CLOUDANT_USERNAME}.cloudant.com
      actions:
        # Action named "update-document-with-watson"
        # Creating action as a regular Node.js action
        update-document-with-watson:
          function: actions/updateDocumentWithWatson.js
          runtime: nodejs:8
          inputs:
            USERNAME: $CLOUDANT_USERNAME
            PASSWORD: $CLOUDANT_PASSWORD
    DBNAME: $CLOUDANT_IMAGE_DATABASE
            DBNAME_PROCESSED: $CLOUDANT_TAGS_DATABASE
            WATSON_VR_APIKEY: $WATSON_VISUAL_APIKEY
      triggers:
        # Creating the update-trigger trigger"
        update-trigger:
          feed: serverless-pattern-cloudant-package/changes
          inputs:
            dbname: $CLOUDANT_IMAGE_DATABASE
      rules:
        # Rule named "update-trigger-rule"
        # Creating the rule that links the trigger to the sequence
        update-trigger-rule:
          trigger: update-trigger
  action: update-document-with-watson
```

```
  var Cloudant = require('@cloudant/cloudant');

  var fs = require('fs');

  var cloudant;

  var dbName;
```

```
var dbNameProcessed;
function main(params) {
   cloudant = Cloudant({account:params.USERNAME,
password:params.PASSWORD});
   dbName = params.DBNAME;
   dbNameProcessed = params.DBNAME_PROCESSED;
   return new Promise(function(resolve, reject) {
      let mydb = cloudant.db.use(dbName);
      mydb.attachment.get(params.id, 'image', function(err, data) {
         if (err) {
            reject(err);
         } else {
            console.log(params)
            console.log(data)
resolve(processImageToWatson(data,params.id,params.WATSON_VR_
APIKEY)); }  }));  }); }
 function processImageToWatson(data,id,apikey) {
   let filename = __dirname + '/' + id;
   fs.writeFileSync(filename, data)
   let VisualRecognitionV3 = require('watson-developer-cloud/visual-
recognition/v3');
   var visualRecognition = new VisualRecognitionV3({
      version: '2018-03-19',
      iam_apikey: apikey, });
   var watsonvrparams = {
      images_file: fs.createReadStream(filename) };
```

```
    return new Promise(function(resolve, reject) {
      visualRecognition.classify(watsonvrparams, function(err, res) {
        if (err) {
          console.log(err);
         reject(err);
        } else {
          resolve(updateDocument(res,id));          } });});
function updateDocument(watsonResult,id) {
    return new Promise(function(resolve, reject) {
      let mydb = cloudant.db.use(dbNameProcessed);
      var doc = {};
      doc._id = id
      doc.watsonResults = watsonResult.images[0].classifiers;
      mydb.insert(doc, function(err,body) {
        if (err) {
          reject(err);
        } else {
          console.log(body);
          resolve(body); }});   });}
```

Loadimage.js

```
var rateLimit = 0;

function getAllImageDocuments() {
 $.ajax({
    url: cloudantURL.origin + "/" + imageDatabase + "/_all_docs",
    type: "GET",
```

```
    headers: {
      "Authorization": "Basic " + btoa(cloudantURL.username + ":" +
cloudantURL.password)     },
    success: function (data) {
      for (var index in data.rows) {
        getImage(data.rows[index].id) }  },
    error: function (jqXHR, textStatus, errorThrown) {
      console.log(errorThrown);
      console.log(textStatus);
      console.log(jqXHR); }});}

function getImage(id) {
 var image = new Image();
 var imageSection = document.createElement('div');
 var imageHolder = document.createElement('div');
 image.className = "uploadedImage";
 loadAttachment(id,image);
 imageSection.id = id
 imageSection.className = "imageSection";
 imageHolder.className = "imageHolder";
 imageHolder.appendChild(image);
 imageSection.appendChild(imageHolder);
 uploadedImages.prepend(imageSection);
 getDocumentWithId(id, imageSection, 0); }

 function loadAttachment(id, dom) {

 if (rateLimit >= 5) {
  setTimeout(function() {
   loadAttachment(id,dom);
  }, 1000);
 } else {
  rateLimit++;
  console.log(rateLimit);
  $.ajax({
```

```
        url: cloudantURL.origin + "/" + imageDatabase + "/" + id +
    "/image",
        type: "GET",
        headers: {
          "Authorization": "Basic " + btoa(cloudantURL.username + ":" +
    cloudantURL.password)
        },
        xhr:function(){
          var xhr = new XMLHttpRequest();
          xhr.responseType= 'blob'
          return xhr;
        },
        success: function (data) {
          let url = window.URL || window.webkitURL;
          dom.src = url.createObjectURL(data);
          rateLimit--;
        },
        error: function (jqXHR, textStatus, errorThrown) {
          console.log(errorThrown);
          console.log(textStatus);
          console.log(jqXHR);
          rateLimit--;  } });  }

    }getAllImageDocuments();

bluemix:fa2971ea3c351e710593bd1fb85d6b714dd5d2c9cdc03a49568f58fd8
874cb1f@d1dda683-a71d-43ca-9c92-bf111700dc00-
bluemix.cloudant.com/test");

    //   console.log("error");
    //   console.log(err);
    // });;
```

Upload.js

```
let uplaodButton = document.getElementById("uploadImage");

let selectImage = document.getElementById("selectImage");
```

```
let form = document.forms['upload'];
let imageDatabase = "images"
let tagsDatabase = "tags"
let uploadedImages = document.getElementById("uploadedImages");
let usernameCloudant = "d1dda683-a71d-43ca-9c92-bf111700dc00-bluemix"
let passwordCloudant = "fa2971ea3c351e710593bd1fb85d6b714dd5d2c9cdc03a49568f58fd8874cb1f
"const cloudantURL = new URL("https://" + usernameCloudant + ":" + passwordCloudant + "@" + usernameCloudant + ".cloudant.com");
function uploadImage() {
  var image = selectImage.files[0];
  // name is .name, type is .type
  var fileReader = new FileReader();
  fileReader.readAsDataURL(selectImage.files[0]);
  fileReader.onload = function (readerEvent) {
    // console.log(readerEvent.target.result.split(',')[1])
    var cloudantDocument = {
      "name": image.name,
      "_attachments": {}    };
     var attachment = {}
    attachment.content_type = image.type
    attachment.data = readerEvent.target.result.split(',')[1]
    cloudantDocument._attachments.image = attachment
    console.log(cloudantDocument);
    loadImageToBrowser(cloudantDocument, selectImage.files[0]); } }
function loadImageToBrowser(doc, imageToLoad) {
```

```
  var fileReader = new FileReader();
  var image = new Image();
  var imageSection = document.createElement('div');
  var imageHolder = document.createElement('div');
  imageSection.className = "imageSection"
  imageHolder.className = "imageHolder"
  fileReader.readAsDataURL(imageToLoad);
  fileReader.onload = function (readerEvent) {
    image.src = readerEvent.target.result
    image.className = "uploadedImage";
    imageHolder.appendChild(image);
    imageSection.appendChild(imageHolder);
    uploadedImages.prepend(imageSection);
    uploadToCloudant(doc, imageSection); } }
    function uploadToCloudant(doc, dom) {
  console.log(doc)
  console.log(cloudantURL.origin)
  $.ajax({
      url: cloudantURL.origin + "/" + imageDatabase,
      type: "POST",
      data: JSON.stringify(doc),
      headers: {
        "Authorization": "Basic "+ btoa(cloudantURL.username + ":" +       },
cloudantURL.password)
      dataType: 'json',
```

```
    contentType: 'application/json',
    success: function (data) {
      // add 1.5s delay to give time for serverless function to execute
      setTimeout(function () {
        getDocumentWithId(data.id, dom, 0);
      }, 1500); },
    error: function (jqXHR, textStatus, errorThrown) {
      console.log(errorThrown);
      console.log(textStatus);
      console.log(jqXHR); }  });  }
function getDocumentWithId(id, dom, tries) {
  $.ajax({
    url: cloudantURL.origin + "/" + tagsDatabase + "/" + id,
    type: "GET",
    headers: {
      "Authorization": "Basic " + btoa(cloudantURL.username + ":" +
cloudantURL.password) },
   success: function (data) {
      displayTags(data, dom) },
   error: function (jqXHR, textStatus, errorThrown) {
      console.log(errorThrown);
      console.log(textStatus);
      console.log(jqXHR);
      if (tries+1 < 20) {
        // try again in 3 seconds
```

```
        setTimeout(function() {
          getDocumentWithId(id, dom, tries+1)
        },3000);
      } else {
        console.log("No document found after 20 tries")  }     } }); }
function displayTags(data, dom) {
  dom.id = data._id;
  var tags = document.createElement('div');
  tags.className = "imageLabels";
  for (var index in data.watsonResults[0].classes) {
    var tag = document.createElement('div');
    tag.className = "imageLabel";
    tag.innerHTML = data.watsonResults[0].classes[index].class
    tags.appendChild(tag); }
  dom.appendChild(tags) }
form.onsubmit = function() {
  uploadImage()
  return false; }
// getDocumentWithId("0724dabd80bc2102e8e5e1f9fdbb3a60",0);
```

Directory structure

```
.
├── app.js                  // express routes
├── config                  // express configuration
│   ├── error-handler.js
│   ├── express.js
│   └── security.js
├── package.json
├── public                  // static resources
├── server.js               // entry point
├── test                    // integration tests
└── src                     // react client
    ├── __test__            // unit tests
    └── index.js            // app entry point
```

Conclusion :

In conclusion, IBM Cloud Visual Recognition is a powerful tool that uses deep learning algorithms to analyze images and identify subjects and objects contained within the image, and organize and classify these images into categories. To create an image recognition system using IBM Cloud Visual Recognition, you need to create an IBM Cloud account, create a Visual Recognition service, and upload and classify your images using the built-in image classifiers. You can also enable image recognition in your app by creating a Watson Visual Recognition instance. . Image recognition is a powerful tool that is being embedded in a heap of new fields including self-driving trucks, product usage analytics, and satellite imagery processing.