**IMAGE RECOGNITION SYSTEM USING IB CLOUD VISUAL RECOGNITION.**

**Introduction :**

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.

**Procedure :**

* Setting up IBM Cloud Functions, Signup for an IBM cloud account. Then Install IBM Cloud CLI on your local machine to interact with IBM Cloud services.
* Integrate IBM Cloud Functions with Watson Visual Recognition which will be triggered to perform image recognition using Watson Visual Recognition. Within the function code, the Watson Visual Recognition API to send the image for analysis and receive the classification results. Set up triggers within IBM Cloud Functions to invoke the serverless function whenever an event occurs, such as an image being uploaded to a designated storage.
* Test and Deploy the integration and production, Upload sample images to the designated storage and observe the serverless functions invoking Watson Visual Recognition to analyze the images. Optimize and Fine-tune your serverless function and Watson Visual Recognition integration based on test results and performance. Deploy the integrated system in a production environment and ensure it functions reliably and efficiently.

**Steps to follow:**

* **IBM Cloud Account :**

Sign up for an IBM Cloud account if you not have one already. Then Log in to your IBM Cloud account and navigate to the IBM Dashboard.

* **To create a Visual Recognition Service :**

In “Create Resource” or at “Add Service” from the IBM Cloud Dashboard and select the Visual Recognition service.

* **Obtain API Key :**

Once the Visual Recognition service is created, go to the service instance. Then Obtain the API key for accessing the service.

* **Set up a Node.js Application and Dependencies :**

Create a new Node.js project or use an existing one.

Install the required npm packages, including ‘watson-developer-cloud’ for interacting with the visual recognition service and other necessary modules.

* **Integrate with the Visual Recognition Service :**

Use the API key and service instance credentials to authenticate with the Visual Recognition service.

Utilize the Watson Visual Recognition API to classify images and obtain results.

* **Upload and Display classification :**

Implement functionality to allow users to upload images for classification, Present the classification results obtained from the Visual Recognition service to users in a user-friendly format.

Deploy Node.js application, ensuring it’s accessible over the internet.

* **Test the Image reognition System :**

Upload images to the application and verify that it accurately classifies them using IBM Visual Recognition.

Optimize the application for performance, scalability and the usability based on requirements.

Consider adding features like image tagging, training the model with custom classes, or itegrating the system with other services.

**Installation :**

---

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."

**Program :**

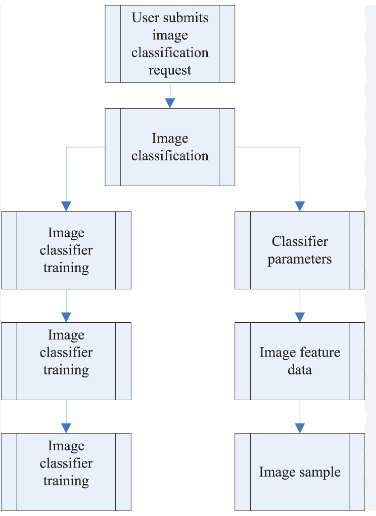
**FRONT END:**

|  |
| --- |
| <!DOCTYPE html> |
|  |  | <html lang="en"> |
|  |  | <head> |
|  |  | <meta charset="utf-8" /> |
|  |  | <meta http-equiv="X-UA-Compatible" content="IE=edge" /> |
|  |  | <meta name="viewport" content="width=device-width, initial-scale=1" /> |
|  |  | <title>Image classification with Code Engine</title> |
|  |  | <link |
|  |  | rel="stylesheet" |
|  |  | href="https://cdn.jsdelivr.net/npm/bulma@0.8.0/css/bulma.min.css" |
|  |  | /> |
|  |  | <link href="https://fonts.googleapis.com/css2?family=IBM+Plex+Sans&family=  IBM+Plex+Mono&display=swap"  rel="stylesheet"> |
|  |  | <script |
|  |  | defer |
|  |  | src="https://use.fontawesome.com/releases/v5.3.1/js/all.js" |
|  |  | ></script> |
|  |  | </head> |
|  |  |  |
|  |  | <body> |
|  |  | <section class="hero is-fullheight is-default is-bold"> |
|  |  | <div class="hero-head"> |
|  |  | <nav class="navbar"> |
|  |  | <div class="container"> |
|  |  | <div class="navbar-brand"> |
|  |  | <h1 class="navbar-item title"> |
|  |  | Image Classification with Code Engine |
|  |  | </h1> |
|  |  | <span class="navbar-burger burger" data-target="navbarMenu"> |
|  |  | <span></span> |
|  |  | <span></span> |
|  |  | <span></span> |
|  |  | </span> |
|  |  | </div> |
|  |  | <div id="navbarMenu" class="navbar-menu"> |
|  |  | <div class="navbar-end"> |
|  |  | <div class="navbar-item"> |
|  |  | <div class="field is-grouped"> |
|  |  | <p class="control"> |
|  |  | <a target="\_blank" href="https://cloud.ibm.com/docs/codeengine"> |
|  |  | <img src="./code-engine-logo.svg" alt="Code Engine Logo" width="16" height="16"> |
|  |  | Code-Engine |
|  |  | </a> |
|  |  | </p> |
|  |  | </div> |
|  |  | </div> |
|  |  | <div class="navbar-item"> |
|  |  | <div class="field is-grouped"> |
|  |  | <p class="control"> |
|  |  | <a target="\_blank" href="https://cloud.ibm.com/docs/  solution-tutorials?topic=solution-tutorials-image\_classification\_code\_engine"> |
|  |  | <a target="\_blank" href="https://cloud.ibm.com/docs/  solution-tutorials?topic=solution-tutorials-text-analysis-code-engine"> |
|  |  | <i class="fab fa-readme" ></i> |
|  |  | Solution-tutorial |
|  |  | </a> |
|  | | @@ -56,7 +56,7 @@ <h1 class="navbar-item title"> |
|  |  | <div class="navbar-item"> |
|  |  | <div class="field is-grouped"> |
|  |  | <p class="control"> |
|  |  | <a target="\_blank" href="https://github.com/IBM-Cloud/  code-engine-image-classification"> |
|  |  | <a target="\_blank" href="https://github.com/VidyasagarMSC/  image-classification-code-engine"> |
|  |  | <i class="fab fa-github"></i> |
|  |  | Code |
|  |  | </a> |
|  |  | </p> |
|  |  | </div> |
|  |  | </div> |
|  |  | </div> |
|  |  | </div> |
|  |  | </div> |
|  |  | </nav> |
|  |  | </div> |
|  |  | <div class="hero-body"> |
|  |  | <div class="container has-text-centered"> |
|  |  | <section class="section"> |
|  |  | <div class="container"> |
|  |  | <div class="columns is-vcentered"> |
|  |  | <div class="column has-text-centered"> |
|  |  | <h1 class="title">Congratulations! Your Frontend is working</h1> |
|  |  | <p class="subtitle">Oops!! Looks like the Connection to the backend is failing. Time to add a backend</p> |
|  |  | </div> |
|  |  | <div class="column has-text-centered"> |
|  |  | <img src="./robot-2.png" width="400" height="400" /> |
|  |  | </div> |
|  |  | </div> |
|  |  | </div> |
|  |  | </section> |
|  |  | </div> |
|  |  | </div> |
|  |  |  |
|  |  | <div class="hero-foot"> |
|  |  | <div class="container"> |
|  |  | <div class="tabs is-centered"> |
|  |  | <ul> |
|  |  | <li class="is-active"><a href="https://www.ibm.com/cloud" target="\_blank">  <img  src="./cloud.png" alt="IBM Cloud logo" width="16" height="16">&nbsp;IBM Cloud  </a></li> |
|  |  | </ul> |
|  |  | </div> |
|  |  | </div> |
|  |  | </div> |
|  |  | </section> |
|  |  | <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"><  /script> |
|  |  | <script src="http://cdnjs.cloudflare.com/ajax/libs/jquery.form/3.51  /jquery.form.min.js"></script> |
|  |  | <script src="./index.js" type="text/javascript"></script> |
|  |  | <link rel="stylesheet" type="text/css" href="./style.css"/> |
|  |  | </body> |
|  |  | </html> |

**Backend:**

|  |
| --- |
| "use strict"; |
|  |  | const myCOS = require("ibm-cos-sdk"); |
|  |  | const multer = require("multer"); |
|  |  | const multerS3 = require("multer-s3"); |
|  |  | const express = require("express"); |
|  |  | const app = express(); |
|  |  | const path = require("path"); |
|  |  | require("dotenv").config({ |
|  |  | silent: true |
|  |  | }); |
|  |  |  |
|  |  | const cors = require("cors"); |
|  |  | app.use(cors()); |
|  |  |  |
|  |  | const port = process.env.PORT || 3001; |
|  |  | /\*\* |
|  |  | \*Define Cloud OBject Storage client configuration |
|  |  | \* |
|  |  | \* @return {\*} cosCLient |
|  |  | \*/ |
|  |  | function getCosClient() { |
|  |  | var config = { |
|  |  | endpoint: |
|  |  | process.env.COS\_ENDPOINT || |
|  |  | "s3.us-south.cloud-object-storage.appdomain.cloud", |
|  |  | apiKeyId: process.env.COS\_SECRET\_APIKEY, |
|  |  | apiKeyId: process.env.COS\_APIKEY, |
|  |  | ibmAuthEndpoint: "https://iam.cloud.ibm.com/identity/token", |
|  |  | serviceInstanceId: process.env.COS\_SECRET\_RESOURCE\_INSTANCE\_ID, |
|  |  | serviceInstanceId: process.env.COS\_RESOURCE\_INSTANCE\_ID, |
|  |  | }; |
|  |  |  |
|  |  | //console.log(process.env); |
|  |  | var cosClient = new myCOS.S3(config); |
|  |  | return cosClient; |
|  |  | } |
|  |  |  |
|  |  | /\*\* |
|  |  | \* Upload images to COS Bucket |
|  |  | \* |
|  |  | \* @param {\*} req |
|  |  | \* @param {\*} res |
|  |  | \* @param {\*} next |
|  |  | \*/ |
|  |  | function uploadFilesToCOS(req, res, next) { |
|  |  | var upload = multer({ |
|  |  | storage: multerS3({ |
|  |  | s3: getCosClient(), |
|  |  | bucket: process.env.COS\_BUCKETNAME + "/images", |
|  |  | metadata: function (req, file, cb) { |
|  |  | cb(null, { fieldName: file.fieldname }); |
|  |  | }, |
|  |  | key: function (req, file, cb) { |
|  |  | cb(null, file.originalname); |
|  |  | }, |
|  |  | }), |
|  |  | }).array("files", 10); |
|  |  |  |
|  |  | upload(req, res, function (err) { |
|  |  | if (err) { |
|  |  | return next(err); |
|  |  | } |
|  |  | if (req.files.length === 0) { |
|  |  | return res.send("Upload an image..."); |
|  |  | } else if (req.files.length > 1) { |
|  |  | return res.send( |
|  |  | "Successfully uploaded " + req.files.length + " images to Object Storage" |
|  |  | ); |
|  |  | } else { |
|  |  | return res.send( |
|  |  | "Successfully uploaded " + req.files.length + " image to Object Storage" |
|  |  | ); |
|  |  | } |
|  |  | }); |
|  |  | } |
|  |  | /\*\* |
|  |  | \*Get COS bucket contents (images) |
|  |  | \* |
|  |  | \* @param {\*} req |
|  |  | \* @param {\*} res |
|  |  | \* @param {\*} next |
|  |  | \* @param {\*} prefix |
|  |  | \* @return {\*} result dictionary |
|  |  | \*/ |
|  |  | async function getBucketContents(req, res, next, prefix) { |
|  |  | try { |
|  |  | let cos = getCosClient(); |
|  |  | let bucketName = process.env.COS\_BUCKETNAME; |
|  |  | var resultDict = {}; |
|  |  | var result; |
|  |  | console.log(`Retrieving bucket contents from: ${bucketName}`); |
|  |  |  |
|  |  | const data = await cos |
|  |  | .listObjects({ |
|  |  | Bucket: bucketName, |
|  |  | Prefix: prefix, |
|  |  | }) |
|  |  | .promise(); |
|  |  | if (data != null && data.Contents != null) { |
|  |  | for (var i = 0; i < data.Contents.length; i++) { |
|  |  | var itemKey = data.Contents[i].Key; |
|  |  | var itemSize = data.Contents[i].Size; |
|  |  | console.log(`Item: ${itemKey} (${itemSize} bytes).`); |
|  |  | result = await getItem(bucketName, itemKey, prefix); |
|  |  | resultDict[itemKey] = result; |
|  |  | } |
|  |  | res.send(resultDict); |
|  |  | } |
|  |  | } catch (e) { |
|  |  | console.error(`ERROR: ${e.code} - ${e.message}\n`); |
|  |  | return next(e.message); |
|  |  | } |
|  |  | } |
|  |  | /\*\* |
|  |  | \* Get each item in a COS Bucket |
|  |  | \* |
|  |  | \* @param {\*} bucketName |
|  |  | \* @param {\*} itemName |
|  |  | \* @param {\*} prefix |
|  |  | \* @return {\*} |
|  |  | \*/ |
|  |  | async function getItem(bucketName, itemName, prefix) { |
|  |  | let cos = getCosClient(); |
|  |  | console.log(`Retrieving item from bucket: ${bucketName}, key: ${itemName}`); |
|  |  | try { |
|  |  | const data = await cos |
|  |  | .getObject({ |
|  |  | Bucket: bucketName, |
|  |  | Key: itemName, |
|  |  | }) |
|  |  | .promise(); |
|  |  | if (data != null) { |
|  |  | if (prefix === "results") { |
|  |  | return JSON.parse(data.Body); |
|  |  | } else { |
|  |  | return Buffer.from(data.Body).toString("base64"); |
|  |  | } |
|  |  | } |
|  |  | } catch (e) { |
|  |  | console.error(`ERROR: ${e.code} - ${e.message}\n`); |
|  |  | } |
|  |  | } |
|  |  |  |
|  |  | async function deleteItem(req, res, next, bucketName, itemName, prefix) { |
|  |  | let cos = getCosClient(); |
|  |  | let bucketname = process.env.COS\_BUCKETNAME; |
|  |  | itemName = prefix + "/" + itemName; |
|  |  | if (prefix === "results") { |
|  |  | itemName = itemName + ".json"; |
|  |  | } |
|  |  | console.log(`Deleting item: ${itemName}`); |
|  |  | try { |
|  |  | await cos |
|  |  | .deleteObject({ |
|  |  | Bucket: bucketname, |
|  |  | Key: itemName, |
|  |  | }) |
|  |  | .promise(); |
|  |  | console.log(`Item: ${itemName} deleted!`); |
|  |  | res.send(`Item: ${itemName} deleted!`); |
|  |  | } catch (e) { |
|  |  | console.error(`ERROR: ${e.code} - ${e.message}\n`); |
|  |  | } |
|  |  | } |
|  |  |  |
|  |  | /\* |
|  |  | \* Default route for the web app |
|  |  | \*/ |
|  |  | app.get("/", function (req, res, next) { |
|  |  | res.send("Hello World! from backend"); |
|  |  | }); |
|  |  |  |
|  |  | app.get("/items", async (req, res, next) => { |
|  |  | try { |
|  |  | var prefix = req.query.prefix; |
|  |  | // console.log(prefix); |
|  |  | await getBucketContents(req, res, next, prefix); |
|  |  | } catch (error) { |
|  |  | // Passes errors into the error handler |
|  |  | return next(error); |
|  |  | } |
|  |  | }); |
|  |  | /\* |
|  |  | \* Upload an image for Image classification |
|  |  | \*/ |
|  |  | app.post("/images", uploadFilesToCOS, function (req, res, next) {}); |
|  |  |  |
|  |  | /\*\* |
|  |  | \* Get the JSON from the results folder of COS Bucket |
|  |  | \*/ |
|  |  | app.post("/results", async (req, res, next) => { |
|  |  | try { |
|  |  | await getBucketContents(req, res, next, "results"); |
|  |  | } catch (error) { |
|  |  | // Passes errors into the error handler |
|  |  | console.log(error); |
|  |  | return next(error); |
|  |  | } |
|  |  | }); |
|  |  |  |
|  |  | /\*\* |
|  |  | \* Delete an item from the COS Bucket |
|  |  | \*/ |
|  |  | app.delete("/item", async (req, res, next) => { |
|  |  | var itemName = req.query.filename; |
|  |  | console.log(itemName); |
|  |  | await deleteItem(req, res, next, null, itemName, "images"); |
|  |  | await deleteItem(req, res, next, null, itemName, "results"); |
|  |  | }); |
|  |  |  |
|  |  | /\*\* |
|  |  | \* Middleware to handle not supported routes |
|  |  | \*/ |
|  |  | app.use((req, res, next) => { |
|  |  | const error = new Error("Not found"); |
|  |  | error.status = 404; |
|  |  | next(error); |
|  |  | }); |
|  |  |  |
|  |  | // error handler middleware |
|  |  | app.use((error, req, res, next) => { |
|  |  | console.log(error); |
|  |  | if (res.headersSent) { |
|  |  | return next(error); |
|  |  | } |
|  |  | res.status(error.status || 500).send({ |
|  |  | error: { |
|  |  | status: error.status || 500, |
|  |  | message: error.message || "Internal Server error", |
|  |  | }, |
|  |  | }); |
|  |  | }); |
|  |  |  |
|  |  | app.listen(port, () => console.log(`App listening on port ${port}!`)); |

**Architecture :**

****

**Conclusion :**

Creating an image recognition system using the IBM Clous Visual Recognition in a cloud computing project allows for powerful image analysis capabilities that can be integrated into various applications and solutions.

Image recognition is a crucial application of artificial intelligence that involves the interpretation of images to extract meaningful information. IBM Cloud Visual Recognition is a state-of-the-art AI service that allows for image analysis, classification, and extraction of important features from images.

In conclusion, implementing an image recognition system using IBM Cloud Visual Recognition in a cloud computing project offers a powerful solution for automating image analysis, which can find applications in a wide range of domains, ultimately improving processes, decision-making and the user experiences.