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 :

```
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);
            }
        });
    });
}
```

Node.js :

```
# Node.js
# Logs
logs
*.log
npm-debug.log*
```

```
yarn-debug.log*
yarn-error.log*

# Runtime data
pids
*.pid
*.seed
*.pid.lock

# Directory for instrumented libs generated by jscoverage/JSCover
lib-cov


.grunt
.env
.vuepress/dist

# Serverless directories
.serverless
build/
DerivedData/

## Various settings
*.pbxuser
```

```
!default.pbxuser
*.mode1v3
!default.mode1v3
*.mode2v3
!default.mode2v3
*.perspectivev3
!default.perspectivev3
xcuserdata/

## Other
*.moved-aside
*.xccheckout
*.xcscmblueprint

## Obj-C/Swift specific
*.hmap
*.ipa
*.dSYM.zip
*.dSYM

## Playgrounds
timeline.xctimeline
playground.xcworkspace
```

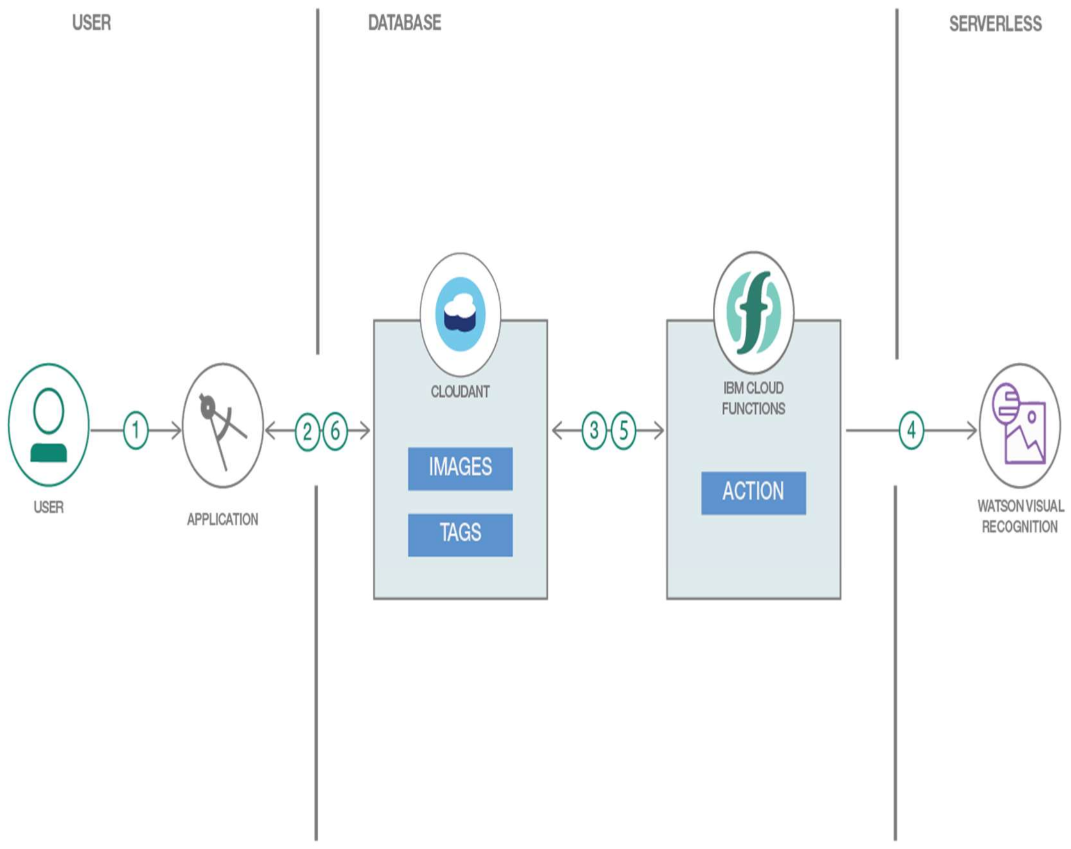
```
# Swift Package Manager

# Add this line if you want to avoid checking in source code from Swift
Package Manager dependencies.

# Packages/

# Package.pins

# Package.resolved

.build/

# CocoaPods

# We recommend against adding the Pods directory to your .gitignore.
However

# you should judge for yourself, the pros and cons are mentioned at:

# https://guides.cocoapods.org/using/using-cocoapods.html#should-i-
check-the-pods-directory-into-source-control


# Pods/

# Carthage

# Add this line if you want to avoid checking in source code from
Carthage dependencies.

# Carthage/Checkouts


Carthage/Build


fastlane/report.xml
```

fastlane/Preview.html

fastlane/screenshots/**/*.png

fastlane/test_output

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.