

# La reconnaissance d'images à portée de main avec l'**API Cloud Vision** de Google



Aurélie Vache  
Lead Developer chez **Atchik**  
Duchess France & DevFest  
Toulouse Leader  
[@aurelievache](#)

## Que permet la Cloud Vision API ?

L'API Cloud Vision fournit un ensemble de fonctionnalités pour analyser des images. Google Cloud Vision permet aux développeurs d'utiliser le pouvoir du Machine Learning afin de collecter des informations sur une image. Les caractéristiques des images détectées peuvent être par exemple les principales couleurs d'une image, les principaux objets qui la composent, la détection de visages, de logos, de contenus non appropriés à caractère sexuel ou de violence, de localisation, de texte dans les images via l'OCR (Optical Character Recognition).

Si vous appelez l'API REST avec une image d'une pizza par exemple, vous pourrez obtenir comme résultat qu'il s'agit de nourriture (food) et d'une pizza ;-). Par exemple, plus vous demanderiez de LABEL et plus vous obtiendriez de résultats vous permettant de classifier une image. L'API de Vision est utilisée en interne par Google; sans le savoir nous utilisons donc des outils de Google qui utilisent cette API, tels que Street View, Photos, Search Image et Translate. [1]

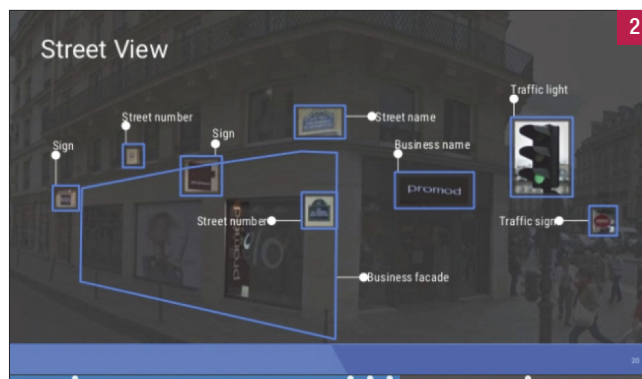
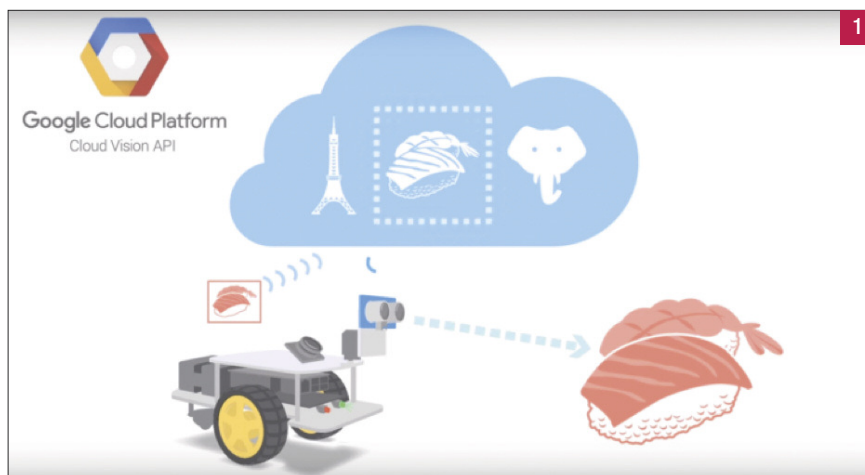
2.png

## REST API

Afin d'appeler l'API REST il suffit de faire une requête POST en utilisant le endpoint `images:annotate` avec comme requête la feature que l'on veut utiliser :

```
POST https://vision.googleapis.com/v1/images:annotate?key=YOUR_API_KEY
{
  "requests": [
    {
      "images": {
        "content": "/9j/7QBEUGhvdG9zaG9...base64-encoded-image-content...fXNW
zvDEeYxxxzj/Coa6Bax/Z"
      },
      "features": [
        {
          "type": "LABEL_DETECTION", //voir la liste des types de features possibles
          plus bas
        }
      ]
    }
  ]
}
```

L'API Cloud Vision de Google, qui est sortie récemment en GA (General Availability) est un nouveau service de reconnaissance d'images, faisant partie de l'offre de Machine Learning (apprentissage automatique) de Google. Grâce à cette API, Google continue de démocratiser les technologies de Machine Learning utilisées en interne et fournit aux développeur.se.s une simple API REST extrêmement puissante. [1]



"maxResults": 3 //par exemple

```
{
  }
  ]
}
```

Les exemples de réponses seront précisés lors du listing des features possibles.

## Librairies

Comme d'habitude, vous pouvez utiliser les librairies fournies par Google dans vos projets en Java, Python, C#, Go, NodeJS, PHP et Ruby. Allons maintenant voir ce que l'API a sous le capot comme fonctionnalités.

## Quelles sont les fonctionnalités de Cloud Vision ?

- LABEL\_DETECTION : une analyse performante de l'image afin d'en extraire une large série de catégories sous forme de mots-clés descriptifs.

```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/0bt9lr",
          "description": "dog",
          "score": 0.96969336
        },
        {
          "mid": "/m/04rky",
          "description": "mammal",
          "score": 0.92070323
        },
        {
          "mid": "/m/09686",
          "description": "vertebrate",
          "score": 0.89664793
        }
      ]
    }
  ]
}
```

- TEXT\_DETECTION : grâce à la technologie OCR, l'API va être capable d'extraire les informations textuelles d'une image. On fournit à l'API par exemple une capture écran et l'API y extrait le texte contenu dans celle-ci en y détectant la langue du texte (locale). Cette fonctionnalité peut être très utile pour reconnaître le texte sur une plaque d'immatriculation, sur un panneau de signalisation, sur la plaque du nom des rues.

```
{
  "responses": [
    {
      "textAnnotations": [
        {
          "locale": "fr",
          "description": "Salut, ça va ?\nOui et toi ?",
          "boundingPoly": {
            "vertices": [
              {
                "x": 53,
                "y": 15
              },
              {
                "x": 272,
                "y": 15
              }
            ]
          }
        }
      ]
    }
  ]
}
```

```
"description": "Vive l'extraction de texte dans une capture d'écran !",
"boundingPoly": {
  ...
}
```

- FACE\_DETECTION : détection des visages dans les images incluant une reconnaissance des sentiments (joie, tristesse, etc.).

```
{
  "responses": [
    {
      "faceAnnotations": [
        {
          "boundingPoly": {
            "vertices": [
              {
                "x": 669,
                "y": 324
              },
              {
                "x": 760,
                "y": 324
              }
            ]
          },
          "fdBoundingPoly": {
            ...
            {
              "type": "CHIN_RIGHT_GONION",
              "position": {
                "x": 861.7816,
                "y": 568.26685,
                "z": 50.14255
              }
            }
          ],
          "rollAngle": -5.0866733,
          "panAngle": 1.5809642,
          "tiltAngle": -18.30349,
          "detectionConfidence": 0.9379803,
          "landmarkingConfidence": 0.24303582,
          "joyLikelihood": "VERY_LIKELY",
          "sorrowLikelihood": "VERY_UNLIKELY",
          "angerLikelihood": "VERY_UNLIKELY",
          "surpriseLikelihood": "VERY_UNLIKELY",
          "underExposedLikelihood": "VERY_UNLIKELY",
          "blurredLikelihood": "VERY_UNLIKELY",
          "headwearLikelihood": "VERY_UNLIKELY"
        }
      ],
      "landmarkAnnotations": [
        {
          "mid": "/m/0c7zy",
          "description": "Petra",
          "score": 0.5403372,
          "boundingPoly": {
            ...
          }
        }
      ]
    }
  ]
}
```

```

    "vertices": [
      {
        "x": 153,
        "y": 64
      },
      ...
    ]
  },
  "locations": [
    {
      "latLng": {
        "latitude": 30.323975,
        "longitude": 35.449361
      }
    }
  ]
}

```

- LANDMARK\_DETECTION : détection d'éléments/de localisation géographiques et architecturaux (monument, montagne ...).

```

{
  "responses": [
    {
      "landmarkAnnotations": [
        {
          "mid": "/m/0260wgf",
          "description": "Capitole de Toulouse",
          "score": 0.8286499,
          "boundingPoly": {
            "vertices": [
              {
                "x": 883,
                "y": 899
              },
              {
                "x": 4220,
                "y": 899
              },
              "..."
            ]
          },
          ...
        }
      ],
      "locations": [
        {
          "latLng": {
            "latitude": 43.604416,
            "longitude": 1.443361
          }
        }
      ]
    }
  ]
}

```

```

]

```

- LOGO\_DETECTION : détection des logos d'entreprises et de marques dans des images.

```

{
  "responses": [
    {
      "logoAnnotations": [
        {
          "description": "Programmez",
          "score": 0.58307266,
          "boundingPoly": {
            "vertices": [
              {
                "x": 14,
                "y": 22
              },
              {
                "x": 449,
                "y": 22
              },
              {
                "x": 449,
                "y": 92
              },
              {
                "x": 14,
                "y": 92
              }
            ]
          },
          ...
        }
      ]
    }
  ]
}

```

- SAFE\_SEARCH\_DETECTION : utilisation de la technologie SafeSearch de Google qui permet de déterminer si une image est inappropriée ou explicite (contenu pour adulte, violence ...). SafeSearch est utilisé dans les résultats de recherche de Google.

```

{
  "responses": [
    {
      "safeSearchAnnotation": {
        "adult": "POSSIBLE",
        "spoof": "POSSIBLE",
        "medical": "LIKELY",
        "violence": "POSSIBLE"
      }
    }
  ]
}

```

- IMAGE\_PROPERTIES : calculez un ensemble de propriétés sur l'image

(telles que les couleurs dominantes de l'image).

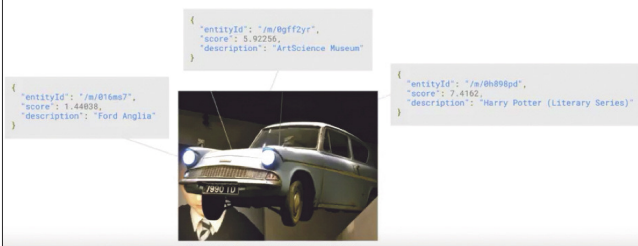
```
{
  "responses": [
    {
      "imagePropertiesAnnotation": {
        "dominantColors": {
          "colors": [
            {
              "color": {
                "red": 69,
                "green": 42,
                "blue": 27
              },
              "score": 0.15197733,
              "pixelFraction": 0.14140345
            },
            ...
          ]
        }
      }
    },
    ...
  ]
}
```

### Nouveautés de la version 1.1 (sortie lors du Cloud Next 17) :

- WEB\_DETECTION : la fonctionnalité va s'appuyer sur la puissance de Google Image Search pour rechercher des informations sur l'image demandée, cela permet également d'obtenir l'url des images qui sont similaires à celle recherchée.

```
{
  "responses": [
    {
      "webDetection": {
        "webEntities": [
          {
            "entityId": "/m/0cbhh",
            "score": 8.57296,
            "description": "Toulouse"
          },
          {
            "entityId": "/m/0404s6",
            "score": 6.7826004,
            "description": "Arno"
          },
          {
            "entityId": "/m/0dy6rb",
            "score": 2.98696,
            "description": "Sée"
          }
        ]
      }
    }
  ]
}
```

#### Web annotations




2b

```
}
],
"fullMatchingImages": [
  {
    "url": "https://pbs.twimg.com/media/CwWuUI4XAAEatX2.jpg"
  },
  {
    "url": "https://pbs.twimg.com/media/Cx72uZLXcAlxpRg.jpg"
  },
  ...
],
"partialMatchingImages": [
  {
    "url": "https://pbs.twimg.com/media/CwWuUI4XAAEatX2.jpg"
  },
  ...
]
```

Les objets *fullMatchingImages*, *partialMatchingImages* et *pagesWithMatchingImages* vont permettre de savoir où est-ce que cette image existe/est hébergée sur Internet. [2b] [3]

- CROP\_HINTS : pour savoir comment "cropper"/couper de manière optimale son image

```
{
  "responses": [
    {
      "cropHintsAnnotation": {
        "cropHints": [
          {
            "boundingPoly": {
              "vertices": [
                {
                  "x": 240
                },
                {
                  "x": 240,
                  "y": 65
                },
                {
                  "x": 240,
                  "y": 65
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
```



#### Web Entities

Toulouse	8.57296
Arno	6.7826
Sée	2.98696
Twitter	0.75889
Software Developer	0.74142
MySQL	0.48484
Image	0.38213
Hashtag	0.38201
Media	0.37943
Craft	0.37881
technics	0.37127
PHP	0.34595
France	0.31021
	0.30375
	0.00013

#### Pages with Matched Images

3

```

    ]
  },
  "confidence": 0.79999995,
  "importanceFraction": 1
}
]
}
}
]
}
}

```

- DOCUMENT\_TEXT\_DETECTION : détection de larges blocs de texte dans une image. Il s'agit d'une optimisation de la feature TEXT\_DOCUMENT utilisant l'OCR de façon plus poussée.

```

{
  "responses": [
    {
      "textAnnotations": [
        {
          "locale": "en",
          "description": "\n",
          "boundingPoly": {
            "vertices": [
              {
                "x": 577,
                "y": 104
              },
              {
                "x": 578,
                "y": 104
              },
              {
                "x": 578,
                "y": 108
              },
              {
                "x": 577,
                "y": 108
              }
            ]
          }
        }
      ]
    }
  ],
  ...
}

```

Feature	1 - 1000 units/month	1001 - 1,000,000 units/month	1,000,001 to 5,000,000 units/month	5,000,001 - 20,000,000 units/month
Label Detection	Free	\$1.50	\$1.50	\$1.00
OCR	Free	\$1.50	\$1.50	\$0.60
Explicit Content Detection	Free		Now free with Label Detection*	
Facial Detection	Free	\$1.50	\$1.50	\$0.60
Landmark Detection	Free	\$1.50	\$1.50	\$0.60
Logo Detection	Free	\$1.50	\$1.50	\$0.60
Image Properties	Free	\$1.50	\$1.50	\$0.60
Web Detection	Free	\$3.50	\$3.50	Contact Google for more information
Document Text Detection	Free	\$3.50	\$3.50	Contact Google for more information

4

## Limites

Qui dit API dit généralement limitation, les voici :

- 4Mo par image ;
- 8Mo par requête ;
- 10 requêtes par seconde ;
- 700 000 requêtes par feature par jour ;
- 20 000 000 requêtes par feature par mois ;
- 8 images par seconde ;
- 16 images par requête ;

Comme d'habitude avec les services de Google, les limitations évoluent et elles ont tendance à augmenter dans le temps. Je vous conseille d'aller les regarder de temps en temps : <https://cloud.google.com/vision/docs/limits>

## Coût :

Une des philosophie des service de Google est de ne payer que pour l'utilisation que vous en faites, tout comme avec BigQuery. Voici le tableau des coûts : [4]

Pour réaliser un POC (Proof Of Concept), pas de soucis il n'y aura pas de coût mais une fois dépassées les 1000 unités, la facture risque de grimper vite. Quand je pense que le stockage et l'analyse de nos données mensuelles chez Atchik coûtent une vingtaine de centimes de dollars par mois, je me dis que la note risque de devenir salée si on utilise plusieurs features pour un grand volume d'images.

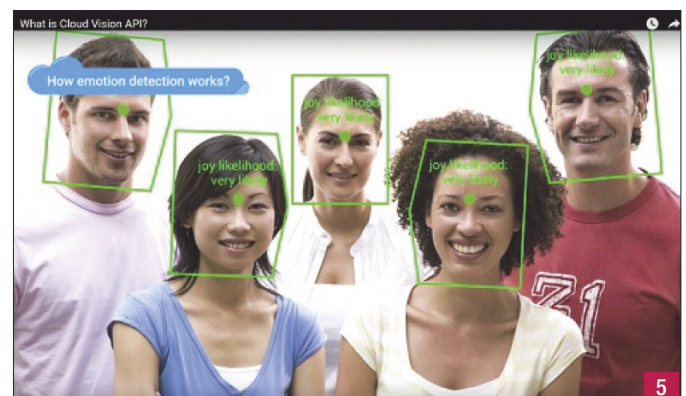
Plus d'information sur les coûts : <https://cloud.google.com/vision/docs/pricing>

## Quelle utilisation peut-on faire de cette API ?

L'utilisation d'une telle API permet d'ouvrir le champ des possibles :

- La reconnaissance de **label** (de mots-clés) dans une image permet plusieurs possibilités comme l'amélioration de l'indexation et la catégorisation de ces dernières, ainsi que la génération automatique de nuage de tags.
- La reconnaissance **faciale** permet de lister les images contenant des personnes, de connaître l'état émotionnel d'une personne et de faire de l'analyse de sentiment sur des photos. [5]
- La reconnaissance de **texte** dans des images pourrait être intégrée dans une voiture connectée par exemple. On pourrait envisager d'analyser les photos prises par un capteur relié à la voiture, la reconnaissance de texte sur les panneaux environnant la voiture pourrait permettre au conducteur d'être mieux guidé qu'un GPS standard par exemple. [6]
- La reconnaissance de **logo** est très utile pour les marques qui vont pouvoir analyser un énorme échantillon de photo et analyser l'utilisation de leurs produits, l'affichage de leur marque.

Analyser un texte est généralement plus facile qu'analyser du texte



5



dans une image, grâce à cette API les deux sont désormais de complexité équivalente. Les services de traduction vont notamment pouvoir être améliorés grâce à cette API.

- La reconnaissance **SafeSearch** est la raison pour laquelle je me suis intéressée à la Cloud Vision API. Savoir si une image est sensible ou non est une donnée qui peut être très utile pour améliorer nos filtres de modération automatisée et intelligente. Les filtres sont créés pour aider les humains afin de traiter efficacement et rapidement un message. On peut également utiliser cette feature dans une application qui permettrait à ses utilisateurs d'uploader un avatar pour refuser automatiquement les avatars sensibles, de type violent ou à caractère sexuel.
- La reconnaissance **Web** peut permettre de lister tous les sites qui hébergent votre image par exemple, et qui font du plagiat; donc pour effectuer de la détection de copyright c'est parfait.

## Cas pratique

### Création d'un compte Google

Si vous n'avez pas encore de compte Google, vous devez en créer un. Connectez-vous sur la console de Google Cloud ([console.cloud.google.com](https://console.cloud.google.com)) et créez un projet.

Notez l'ID du projet, vous en aurez besoin plus loin.

Ensuite il faut activer le paiement (billing) dans la console pour développeurs afin de pouvoir utiliser les ressources de Google Cloud telles que le Datastore et le Storage.

Les nouveaux utilisateurs de la plateforme Cloud de Google peuvent avoir accès à un free trial de 300€ (<https://cloud.google.com/free-trial>).

### Authentification (credentials)

Afin d'utiliser l'API dans une application, il faut que cette dernière soit authentifiée. Le mécanisme d'authentification le plus simple implique le passage d'une clé API directement au service. Lors de l'utilisation de l'API Vision, il est recommandé d'activer une clé API à des fins de test et un compte de service pour l'utilisation de l'API en production.

- Allez sur la console de google pour développeurs (<https://console.cloud.google.com>) ;
- Sélectionnez le projet que vous avez créé ;
- A gauche, cliquez sur le menu hamburger puis sur Gestionnaire d'API > Identifiants ;
- Cliquez sur Créer des identifiants > Clé API ;
- Sélectionnez Clé de navigateur pour cette clé puis cliquez sur Créer ;
- Copiez cette clé, vous allez en avoir besoin ;-).

### Activer la Cloud Vision API

La clé que vous avez créée permet à votre application de s'authentifier auprès de l'API Google Cloud Vision. Pour activer l'API :

Cliquez sur le menu hamburger puis sur Gestionnaire d'API > Activer une API ;

- Cherchez "Cloud Vision API" ;
- Cliquez sur la Cloud Vision API ;
- Puis sur Activer l'API.

### Trève de bla bla, place au code !

Afin de tester ce qu'avait dans le ventre cette API, j'ai réalisé une petite classe Java permettant de tester les différentes features.

On va commencer par exporter sa clé API dans son environnement (on pourrait l'ajouter dans son fichier .bashrc pour ne pas avoir à le répéter à chaque fois par exemple) :



```
$ export GOOGLE_APPLICATION_CREDENTIALS=<path_to_service_account_file>
```

Ensuite, si vous utilisez Maven, il faudra ajouter la dépendance :

```
<!-- https://mvnrepository.com/artifact/com.google.apis/google-api-services-vision -->
<dependency>
  <groupId>com.google.apis</groupId>
  <artifactId>google-api-services-vision</artifactId>
  <version>v1-rev347-1.22.0</version>
</dependency>
```

On va maintenant écrire une petite classe Java permettant d'utiliser toutes les features que l'API permet au moment où je rédige cet article.

```
public class GoogleTest {
    //TODO change your home
    private static final String TMP_REPOSITORY = "/home/aurelie/tmp/";
    //TODO remove your token.properties file if you want to ask a new token to
    Google with new rights/ACL
    private static final String TMP_TOKEN_PROPERTIES = TMP_REPOSITORY +
    "token.properties";

    private static final String TABLE_ID = "table_id";
    private static final String DATASET_ID = "dataset_id";
    private static final String CLIENT_ID = "client_id";
    private static final String CLIENT_SECRET = "client_secret";
    private static final String ACCESS_TOKEN = "access_token";
    private static final String REFRESH_TOKEN = "refresh_token";
    private static final String USER_EMAIL = "toto@gmail.com";

    // Your Google Developer Project number
    private static final String PROJECT_NUMBER = "project_number";

    private static final int MAX_LABELS = 10;

    // Load Client ID/secret from client_secrets.json file
    private static final String CLIENTSECRETS_LOCATION = "client_secrets.json";
    static GoogleClientSecrets clientSecrets = loadClientSecrets();

    static GoogleClientSecrets loadClientSecrets() {
        try {
```

```

        GoogleClientSecrets clientSecrets =
            GoogleClientSecrets.load(new JacksonFactory(),
                new InputStreamReader(GoogleTest.class.getResourceAsStream(
                    CLIENTSECRETS_LOCATION)));
        return clientSecrets;
    } catch (Exception e) {
        System.out.println("Could not load client_secrets.json");
        e.printStackTrace();
        return null;
    }
}

// For installed applications, use the redirect URI "urn:ietf:wg:oauth:2.0:oob"
private static final String REDIRECT_URI = "urn:ietf:wg:oauth:2.0:oob";

// Objects for handling HTTP transport and JSON formatting of API calls
private static final HttpTransport HTTP_TRANSPORT = new NetHttpTransport();
private static final JsonFactory JSON_FACTORY = new JacksonFactory();

public GoogleTest() {
}

/**
 * @param args
 * @throws IOException
 */
public static void main(String[] args) throws IOException {
    //Test Cloud Vision API
    Credential credential = getInteractiveAndStoreGoogleCredential();
    cloudVisionTest(credential);
}

public static void cloudVisionTest(final Credential credential) throws IOException {
    Vision vision = new Vision.Builder(HTTP_TRANSPORT, JSON_FACTORY, credential)
        .setHttpRequestInitializer(new HttpRequestInitializer() {
            public void initialize(HttpRequest httpRequest) throws IOException {
                credential.initialize(httpRequest);
                httpRequest.setConnectTimeout(3 * 60000); // 3 minutes connect timeout
                httpRequest.setReadTimeout(3 * 60000); // 3 minutes read timeout
            }
        })
        .setApplicationName("CloudVisionTest")
        .build();

    // In local - The path to the image file to annotate
    // String fileName = "/home/toto/tmp/selfie.jpeg";

    // Reads the image file into memory
    // Path path = Paths.get(fileName);
    // byte[] data = Files.readAllBytes(path);

    //Google Cloud Storage - images are in your gcs bucket
    String gcsPath = "gs://your_bucket/devfestoulouse.jpg";

    // AnnotateImageRequest request = new AnnotateImageRequest()
    //     .setImage(new Image().encodeContent(data))
    //     .setFeatures(ImmutableList.of(

```

```

        // new Feature()
        //     .setType("LABEL_DETECTION")
        //     .setMaxResults(MAX_LABELS));

    ArrayList<Feature> features = new ArrayList<Feature>();
    features.add(new Feature()
        .setType("LABEL_DETECTION")
        .setMaxResults(MAX_LABELS));
    features.add(new Feature()
        .setType("SAFE_SEARCH_DETECTION"));
    features.add(new Feature()
        .setType("TEXT_DETECTION")
        .setMaxResults(MAX_TEXTS));
    features.add(new Feature()
        .setType("FACE_DETECTION")
        .setMaxResults(MAX_FACES));
    features.add(new Feature()
        .setType("LANDMARK_DETECTION"));
    features.add(new Feature()
        .setType("LOGO_DETECTION"));
    features.add(new Feature()
        .setType("IMAGE_PROPERTIES"));
    features.add(new Feature()
        .setType("WEB_DETECTION"));
    features.add(new Feature()
        .setType("CROP_HINTS"));
    features.add(new Feature()
        .setType("DOCUMENT_TEXT_DETECTION"));

    AnnotateImageRequest request = new AnnotateImageRequest()
        //     .setImage(new Image().encodeContent(data))
        .setImage(img)
        .setFeatures(features);

    Vision.Images.Annotate annotate =
        vision.images()
            .annotate(new BatchAnnotateImagesRequest().setRequests(Immutable
                List.of(request)));
    // Due to a bug: requests to Vision API containing large images fail when GZipped.
    annotate.setDisableGZipContent(true);

    BatchAnnotateImagesResponse batchResponse = annotate.execute();
    assert batchResponse.getResponses().size() == 1;
    AnnotateImageResponse response = batchResponse.getResponses().get(0);
    if (response.getLabelAnnotations() == null) {
        throw new IOException(
            response.getError() != null
                ? response.getError().getMessage()
                : "Unknown error getting image annotations");
    }

    if (response.getLabelAnnotations() != null) {
        System.out.println("- LABEL DETECTION -");
        for (EntityAnnotation annotation : response.getLabelAnnotations()) {
            System.out.println(annotation.toString());
        }
    }
}

```

```

if(response.getSafeSearchAnnotation() != null) {
    System.out.println("- SAFE SEARCH DETECTION -");
    System.out.println(response.getSafeSearchAnnotation().toString());
}

System.out.println("- TEXT DETECTION -");
if (response.getTextAnnotations() != null) {
    for(EntityAnnotation annotation : response.getTextAnnotations()) {
        System.out.println(annotation.toString());
        System.out.println("description: " + annotation.getDescription());
    }
} else {
    System.out.println("L'image ne comporte pas de texte !");
}

if(response.getFaceAnnotations() != null) {
    System.out.println("- FACE DETECTION -");
    for(FaceAnnotation annotation : response.getFaceAnnotations()) {
        System.out.println(annotation.toString());

        System.out.printf("anger: %s\njoy: %s\nsurprise: %s\nposition: %s",
            annotation.getAngerLikelihood(),
            annotation.getJoyLikelihood(),
            annotation.getSurpriseLikelihood(),
            annotation.getBoundingPoly());
        System.out.println("\n");
    }
}

if(response.getLandmarkAnnotations() != null) {
    System.out.println("- LANDMARK DETECTION -");
    for (EntityAnnotation annotation : response.getLandmarkAnnotations()) {
        LocationInfo info = annotation.getLocations().listIterator().next();
        System.out.printf("Landmark: %s\n %s\n", annotation.getDescription(),
            info.getLatLng());
        System.out.println("\n");
    }
}

if(response.getLogoAnnotations() != null) {
    System.out.println("- LOGO DETECTION -");
    for(EntityAnnotation annotation : response.getLogoAnnotations()) {
        System.out.println(annotation.toString());
    }
}

if(response.getImagePropertiesAnnotation() != null) {
    System.out.println("- IMAGE PROPERTIES -");
    DominantColorsAnnotation colors = response.getImageProperties
Annotation().getDominantColors();
    for (ColorInfo color : colors.getColors()) {
        System.out.printf("fraction: %f, g: %f, b: %f\n",
            color.getPixelFraction(),
            color.getColor().getRed(),
            color.getColor().getGreen(),
            color.getColor().getBlue());
    }
}

```

```

    }
}

if(response.getWebDetection() != null) {
    System.out.println("- WEB DETECTION -");
    WebDetection annotation = response.getWebDetection();
    for(WebEntity entity : annotation.getWebEntities()) {
        System.out.println(entity.toString());
    }

    System.out.println("Pages with matching images:");
    for (WebPage page : annotation.getPagesWithMatchingImages()) {
        System.out.println(page.getUrl() + " : " + page.getScore());
    }
    System.out.println("Pages with partially matching images:");
    for (WebImage image : annotation.getPartialMatchingImages()) {
        System.out.println(image.getUrl() + " : " + image.getScore());
    }
    System.out.println("Pages with fully matching images:");
    for (WebImage image : annotation.getFullMatchingImages()) {
        System.out.println(image.getUrl() + " : " + image.getScore());
    }
}

if (response.getTextAnnotations() != null) {
    System.out.println("- DOCUMENT TEXT DETECTION -");
    for(EntityAnnotation annotation : response.getTextAnnotations()) {
        System.out.println(annotation.toString());
        System.out.println("description: " + annotation.getDescription());
    }
} else {
    System.out.println("L'image ne comporte pas de texte !");
}

if(response.getCropHintsAnnotation() != null) {
    System.out.println("- CROP HINTS -");
    CropHintsAnnotation annotation = response.getCropHintsAnnotation();
    for (CropHint hint : annotation.getCropHints()) {
        System.out.println(hint.getBoundingPoly()
            + ", confidence: " + java.lang.Math.round(hint.getConfidence() * 100) + "%");
    }
}

}

public static Credential getGoogleCredentialFromRefresh() throws IOException {
    File tokenFile = new File(TMP_REPOSITORY);
    DataStoreFactory credentialStore = new FileDataStoreFactory(tokenFile);
    Credential credential = new GoogleCredential.Builder()
        .setTransport(HTTP_TRANSPORT)
        .setJsonFactory(JSON_FACTORY)
        .setClientSecrets(CLIENT_ID, CLIENT_SECRET)
        .addRefreshListener(new DataStoreCredentialRefreshListener(USER_
EMAIL, credentialStore))
        .build()
        .setAccessToken(ACCESS_TOKEN)
        .setRefreshToken(REFRESH_TOKEN);
}

```



```

    return credential;
}

public static Credential getInteractiveAndStoreGoogleCredential() throws IOException {
    Credential credential = null;
    // Attempt to Load existing Refresh Token
    String storedRefreshToken = loadRefreshToken();
    // Check to see if the an existing refresh token was loaded.
    // If so, create a credential and call refreshToken() to get a new
    // access token.
    if (storedRefreshToken != null) {
        // Request a new Access token using the refresh token.
        credential = new GoogleCredential.Builder().setTransport(HTTP_TRANSPORT)
            .setJsonFactory(JSON_FACTORY)
            .setClientSecrets(clientSecrets)
            .build()
            .setFromTokenResponse(new TokenResponse().setRefreshToken(storedRefreshToken));
        credential.refreshToken();

        System.out.println("access token (readKey) = "+credential.getAccessToken());
        System.out.println("refresh token (deleteKey) = "+credential.getRefreshToken());
    } else {
        // Exchange the auth code for an access token and refresh token
        credential = getInteractiveGoogleCredential();
        System.out.println("access token= "+credential.getAccessToken());
        // Store the refresh token for future use.
        storeRefreshToken(credential.getRefreshToken());
    }
    return credential;
}

/**
 * Gets a credential by prompting the user to access a URL and copy/paste the token
 * @return
 * @throws IOException
 */
public static Credential getInteractiveGoogleCredential() throws IOException {
    List<String> acl = new ArrayList<String>();
    acl.add(VisionScopes.CLOUD_PLATFORM);

    // Create a URL to request that the user provide access to the API
    String authorizeUrl = new GoogleAuthorizationCodeRequestUrl(
        clientSecrets,
        REDIRECT_URI,
        acl).build();

    // Prompt the user to visit the authorization URL, and retrieve the provided
    authorization code
    System.out.println("Paste this URL into a web browser to authorize Vision:\n"
+ authorizeUrl);

    System.out.println("... and paste the code you received here: ");
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String authorizationCode = in.readLine();

    // Create a Authorization flow object

```

```

GoogleAuthorizationCodeFlow flow =
    new GoogleAuthorizationCodeFlow.Builder(HTTP_TRANSPORT,
        JSON_FACTORY,
        clientSecrets,
        acl)
        .setAccessType("offline")
        .setApprovalPrompt("force")
        .build();
// Exchange the access code for a credential authorizing access
GoogleTokenResponse response = flow.newTokenRequest(authorizationCode)
    .setRedirectUri(REDIRECT_URI).execute();
Credential credential = flow.createAndStoreCredential(response, null);
return credential;
}

/**
 * Helper to store a new refresh token in token.properties file.
 */
private static void storeRefreshToken(String refresh_token) {
    Properties properties = new Properties();
    properties.setProperty("refresh_token", refresh_token);
    System.out.println("refresh token= " + properties.get("refresh_token"));
    try {
        properties.store(new FileOutputStream(TMP_TOKEN_PROPERTIES), null);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Helper to load refresh token from the token.properties file.
 */
private static String loadRefreshToken(){
    Properties properties = new Properties();
    try {
        properties.load(new FileInputStream(TMP_TOKEN_PROPERTIES));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return (String) properties.get("refresh_token");
}
}

```

Si vous n'êtes pas trop Java, vous pouvez tester l'API via le Google Cloud Shell (<https://cloud.google.com/shell/docs/> & [https://console.cloud.google.com/cloud-shell?project=api-project-<votre\\_id\\_projet>&hl=fr](https://console.cloud.google.com/cloud-shell?project=api-project-<votre_id_projet>&hl=fr)) :

On commence par exporter sa clé API pour y accéder dans le shell

```
scraly@api-project-666:~$ export API_KEY=xxx
```

On va ensuite se créer un fichier JSON en initialisant une requête en appelant la fonctionnalité détection de Label sur une image que l'on met dans notre bucket sur le Cloud Storage :

```
scraly@api-project-666:~$ vi request.json

{
  "requests": [
    {
      "image": {
        "source": {
          "gsImageUri": "gs://your_bucket/demo-image.jpg"
        }
      },
      "features": [
        {
          "type": "LABEL_DETECTION",
          "maxResults": 10
        }
      ]
    }
  ]
}
```

On donne les droits nécessaires pour accéder à l'image et y récupérer des informations.

```
$ gsutil acl ch -g AllUsers:R gs://your_bucket/demo-image.jpg
```

Si on n'exécute pas cette commande, on obtiendra l'erreur suivante :

```
image-annotator::User lacks permission.: Can not open file: gs://your_buket/demo-image.jpg
```

On appelle l'API de vision avec la requête précédemment paramétrée

```
$ curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
```

On obtient la réponse au format JSON contenant des labelAnnotations :

```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/02wbm",
          "description": "food",
          "score": 0.96290076
        },
        {
          "mid": "/m/02q08p0",
          "description": "dish",

```

```
      "score": 0.9487846
    },
    {
      "mid": "/m/068_x",
      "description": "pizza cheese",
      "score": 0.9431161
    }
  ]
}
```

```
$ gsutil acl ch -g AllUsers:R gs://your_bucket/selfie.jpeg
Updated ACL on gs://your_bucket/selfie.jpeg
```

```
$ vi selfierequest.json
```

```
$ curl -s -X POST -H "Content-Type: application/json" --data-binary @selfierequest.json https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
```

Avec **SAFE\_SEARCH**, une fonctionnalité que j'aime bien, on peut très facilement et rapidement savoir si une photo est du contenu pour les adultes, montrant de la violence, dans le domaine médical ou catégorisé comme spoof. [7]

On peut également interagir avec l'API avec le CLI de Google :

```
gcloud beta ml vision detect-documents
gcloud beta ml vision detect-faces
gcloud beta ml vision detect-image-properties
gcloud beta ml vision detect-labels
gcloud beta ml vision detect-landmarks
gcloud beta ml vision detect-logos
gcloud beta ml vision detect-safe-search
gcloud beta ml vision detect-text
gcloud beta ml vision detect-web
```

## CONCLUSION

En démocratisant l'API Cloud Vision, Google permet encore une fois aux développeurs d'accéder à des algorithmes extrêmement puissants et performants avec une simple API REST. Y accéder et la coupler avec la Natural Language API, stocker les données dans BigQuery et les visualiser dans Data Studio est à la portée de tous. La prochaine étape pour ma part va être d'estimer le coût que cela peut avoir pour utiliser Vision en production et quels avantages productifs cela peut nous donner.

Vous pouvez retrouver l'intégralité du code source sur mon repository Github : <https://github.com/scraly/java-cloud-vision-api>

```
api-project-769082761863 x +
scraly@api-project-769082761863:~$
scraly@api-project-769082761863:~$
scraly@api-project-769082761863:~$
scraly@api-project-769082761863:~$ curl -s -X POST -H "Content-Type: application/json" --data-binary @saferequest.json https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}
{
  "responses": [
    {
      "safeSearchAnnotation": {
        "adult": "POSSIBLE",
        "spoof": "POSSIBLE",
        "medical": "POSSIBLE",
        "violence": "POSSIBLE"
      }
    }
  ]
}
scraly@api-project-769082761863:~$
scraly@api-project-769082761863:~$
scraly@api-project-769082761863:~$
scraly@api-project-769082761863:~$
```