Vertex AI and Google Vision API Integration Guide

Step 1: Prepare Google Cloud Environment  
1. \*\*Enable APIs\*\*: Ensure that the Google Sheets API, Google Cloud Storage, and Google Vision API are enabled in your Google Cloud project.  
2. \*\*Create a Google Cloud Storage Bucket\*\*: Store images uploaded from Google Sheets for processing by the Google Vision API.  
3. \*\*Service Account\*\*: Create a service account with permissions to access Google Sheets, Google Cloud Storage, and Google Vision API.  
  
Step 2: Google Sheets to Google Cloud Storage Integration  
1. Use Google Apps Script linked to your Google Sheet to detect changes and upload images to Google Cloud Storage.  
 - Go to Extensions > Apps Script in your Google Sheet.  
 - Use the `onEdit` or `onChange` trigger to detect new images.  
 - Write a script to upload the image to Google Cloud Storage.  
  
Step 3: Trigger Google Cloud Function on Image Upload  
1. \*\*Create a Google Cloud Function\*\*: This function is triggered whenever a new image is uploaded to your specified Google Cloud Storage bucket.  
2. \*\*Function Logic\*\*: The function sends the image to the Google Vision API for analysis and then updates the Google Sheet with the response.  
  
Step 4: Vertex AI Prompt Template  
Below is a template for a Vertex AI (or any AI/automation tool) prompt to automate the Google Vision API request. This assumes you have a mechanism (like a Cloud Function) to execute this code based on the trigger:  
  
When a new image is added to Google Cloud Storage Bucket {[sodium-stage-411504.appspot.com](https://console.cloud.google.com/storage/browser/sodium-stage-411504.appspot.com;tab=objects?forceOnBucketsSortingFiltering=false&project=sodium-stage-411504)

},   
trigger a Cloud Function that performs the following actions:  
  
1. Extract the image file path from the trigger event.  
2. Use the Google Vision API to analyze the image for {desired\_features, e.g., label detection, object detection, etc.}.  
3. Format the Google Vision API response to extract relevant information.  
4. Update a specific Google Sheets document, inserting the analysis results into the corresponding row and cell.  
5. Ensure error handling is in place for failed API requests or issues updating the Google Sheet.  
  
Code snippet for Cloud Function (Node.js example):  
  
```javascript  
const vision = require('@google-cloud/vision');  
const {GoogleSpreadsheet} = require('google-spreadsheet');  
const {Storage} = require('@google-cloud/storage');  
  
// Google Cloud Function entry point  
exports.analyzeImageAndUpdateSheet = async (event, context) => {  
 const storage = new Storage();  
 const filePath = event.name;  
 const bucketName = event.bucket;  
 const doc = new GoogleSpreadsheet('{Your\_Sheet\_ID}');  
   
 // Initialize Google Vision client  
 const client = new vision.ImageAnnotatorClient();  
  
 try {  
 // Call the Google Vision API  
 const [result] = await client.labelDetection(`gs://${[sodium-stage-411504.appspot.com](https://console.cloud.google.com/storage/browser/sodium-stage-411504.appspot.com;tab=objects?forceOnBucketsSortingFiltering=false&project=sodium-stage-411504)

}/${filePath}`);  
 const labels = result.labelAnnotations.map(label => label.description).join(', ');  
  
 // Authenticate and update the Google Sheet  
 await doc.useServiceAccountAuth(require('./credentials.json'));  
 await doc.loadInfo();  
 const sheet = doc.sheetsByIndex[0]; // or select by sheet name  
 await sheet.addRow({Image: filePath, Labels: labels}); // Customize based on your sheet's structure  
 } catch (error) {  
 console.error('Failed to process image:', error);  
 throw error;  
 }  
};  
```  
  
Replace `{Your\_Bucket\_Name}`, `{Your\_Sheet\_ID}`, and `{desired\_features}` with your specific details and desired Google Vision API features.