

Contents

[Documentación sobre Computer Vision API](#)

[Información general](#)

[¿Qué es Computer Vision?](#)

[Guías de inicio rápido](#)

[Uso del SDK de cliente](#)

[.NET SDK](#)

[SDK de Python](#)

[SDK de Java](#)

[SDK de Node.js](#)

[Go SDK](#)

[Uso de la API de REST](#)

[Analizar una imagen remota](#)

[cURL](#)

[Go](#)

[Java](#)

[JavaScript](#)

[Node.js](#)

[Python](#)

[Analizar una imagen local](#)

[C#](#)

[Python](#)

[Generar una miniatura](#)

[C#](#)

[cURL](#)

[Go](#)

[Java](#)

[JavaScript](#)

[Node.js](#)

[Python](#)

Extracción de texto (Read API)

C#

Java

JavaScript

Python

Extracción de texto impreso (OCR)

C#

cURL

Go

Java

JavaScript

Node.js

Python

Usar un modelo de dominio

Python

Tutoriales

Generación de metadatos para las imágenes

Conceptos

Etiquetas de contenido

Detección de objetos

Detección de la marca

Categorización de imágenes

Descripciones de imágenes

Detección de caras

Detección del tipo de imagen

Contenido específico del dominio

Detección de la combinación de colores

Miniaturas de recorte inteligente

Reconocimiento de texto manuscrito e impreso

Detección de contenido para adultos

Guías de procedimientos

Llamada a Computer Vision API

Uso de contenedores

[Instalación y ejecución de contenedores](#)

[Configuración de contenedores](#)

[Uso con Kubernetes y Helm](#)

[Uso de instancias de contenedores](#)

[Usar el servicio conectado de Computer Vision](#)

[Análisis de vídeos en tiempo real](#)

Referencia

[CLI de Azure](#)

[Azure PowerShell](#)

[Computer Vision API v2.1](#)

[Computer Vision API v2.0](#)

[Computer Vision API v1.0](#)

[SDK](#)

[.NET](#)

[Node.js](#)

[Python](#)

[Go](#)

[Java](#)

Recursos

Ejemplos

[Exploración de una aplicación de procesamiento de imágenes](#)

[Otros ejemplos de Computer Vision](#)

[Preguntas más frecuentes](#)

[Taxonomía de categoría](#)

[Compatibilidad con idiomas](#)

[Precios y límites](#)

[UserVoice](#)

[Stack Overflow](#)

[Azure Roadmap](#)

[Disponibilidad regional](#)

[Cumplimiento normativo](#)

¿Qué es Computer Vision?

13/01/2020 • 11 minutes to read • [Edit Online](#)

El servicio Computer Vision de Azure proporciona a los desarrolladores acceso a algoritmos avanzados que procesan imágenes y devuelven información, según las características visuales que le interesen. Por ejemplo, Computer Vision puede determinar si una imagen incluye contenido para adultos o buscar todos los rostros en una imagen.

Puede usar Computer Vision en la aplicación mediante el uso de un SDK nativo o invocar la API de REST directamente. Esta página cubre ampliamente lo que puede hacer con Computer Vision.

Computer Vision para la administración de activos digitales

Computer Vision puede funcionar en muchos escenarios de administración de activos digitales (DAM). DAM es el proceso empresarial de organización, almacenamiento y recuperación de recursos multimedia enriquecidos, y la administración de los permisos y derechos digitales. Por ejemplo, es posible que una empresa desee agrupar e identificar imágenes basadas en logotipos, caras, objetos, colores visibles, etc. O bien, puede que desee [generar automáticamente leyendas para las imágenes](#) y adjuntar palabras clave para que admitan búsquedas. Para una solución DAM todo en uno que use Cognitive Services, Azure Cognitive Search e informes inteligentes, vea la [Guía del acelerador de la solución de minería del conocimiento](#) en GitHub. Para ver otros ejemplos de DAM, consulte el repositorio [plantillas de la solución Computer Vision](#).

Análisis de imágenes para obtener información

Puede analizar imágenes para detectar y proporcionar información detallada acerca de las funciones visuales y las características. Todas las características de la tabla siguiente se proporcionan gracias a la API [de análisis de imágenes](#).

ACCIÓN	DESCRIPCIÓN
Etiquetar características visuales	Identifique y etiquete las características visuales de una imagen a partir de un conjunto de miles de objetos, seres vivos, paisajes y acciones reconocibles. Cuando las etiquetas son ambiguas o no muy comunes, la respuesta de la API contiene "indicaciones" para aclarar el contexto de la etiqueta. El etiquetado no se limita al sujeto principal, como una persona en primer plano, sino que también incluye el entorno (interior o exterior), muebles, herramientas, plantas, animales, accesorios, gadgets, etc.
Detectar objetos	La detección de objetos es similar al etiquetado, pero la API devuelve las coordenadas del rectángulo delimitador para cada etiqueta aplicada. Por ejemplo, si una imagen contiene un perro, un gato y una persona, la operación de detección mostrará esos objetos junto con sus coordenadas en la imagen. Puede usar esta funcionalidad para procesar más las relaciones entre los objetos de una imagen. También permite saber cuando hay varias instancias de la misma etiqueta en una imagen.

ACCIÓN	DESCRIPCIÓN
Detección de las marcas	Identifique las marcas comerciales en imágenes o vídeos desde una base de datos de miles de logotipos globales. Puede usar esta característica, por ejemplo, para detectar qué marcas son más populares en medios sociales o más frecuentes en la ubicación de los productos multimedia.
Clasificar una imagen	Identifique y clasifique toda una imagen mediante una taxonomía de categoría con jerarquías hereditarias de elementos primarios y secundarios. Las categorías se pueden usar solas o con nuestros nuevos modelos de etiquetado. Actualmente, el inglés es el único idioma que se admite para etiquetar y clasificar imágenes.
Describir una imagen	Genere una descripción de toda una imagen en lenguaje natural, con frases completas. Los algoritmos de Computer Vision generan varias descripciones en función de los objetos identificados en la imagen. Cada una de estas descripciones se evalúa y se genera una puntuación de confianza. Después, se devuelve una lista de puntuaciones de confianza ordenadas de más alta a más baja.
Detectar caras	Detecte caras en una imagen y proporcione información acerca de ellas. Computer Vision devuelve las coordenadas, el rectángulo, el género y la edad de los rostros que detecta. Computer Vision proporciona un subconjunto de la funcionalidad del servicio Face . Puede usar el servicio Face para realizar un análisis más detallado, como la identificación facial y la detección de poses.
Detectar tipos de imagen	Detecte las características de una imagen, como por ejemplo, si una imagen es un dibujo lineal o la probabilidad de que sea una imagen prediseñada.
Detectar contenido específico del dominio	Use los modelos de dominio para detectar e identificar el contenido específico del dominio en una imagen, como celebridades y monumentos. Por ejemplo, si una imagen contiene personas, Computer Vision puede usar un modelo de dominio para celebridades para determinar si las personas que se han detectado en la imagen son famosos conocidos.
Detectar la combinación de colores	Analice el uso del color en una imagen. Computer Vision puede determinar si una imagen está en blanco y negro o en color, y en las imágenes de color, identificar los colores dominantes y de énfasis.
Generar una miniatura	Analice el contenido de una imagen para generar una miniatura adecuada de la misma. En primer lugar, Computer Vision genera una miniatura de alta calidad y, después, analiza los objetos de la imagen para determinar el <i>área de interés</i> . Luego, Computer Vision recorta la imagen para ajustarla a los requisitos del área de interés. La miniatura generada se puede presentar con una relación de aspecto diferente de la de la imagen original en función de sus necesidades.

ACCIÓN	DESCRIPCIÓN
Obtener el área de interés	Analice el contenido de una imagen para devolver las coordenadas del <i>área de interés</i> . En lugar de recortar la imagen y generar una miniatura, Computer Vision devuelve las coordenadas del rectángulo delimitador de la región, por lo que la aplicación que realiza la llamada puede modificar la imagen original según sea necesario.

Extracción de texto de las imágenes

Puede usar la [API Read](#) de Computer Vision para extraer texto escrito a mano e impreso de imágenes en una secuencia de caracteres legible por máquina. La API Read utiliza modelos actualizados y funciona con texto sobre superficies y fondos distintos, como recibos, pósteres, tarjetas de visita, cartas y pizarras. Actualmente, el inglés es el único idioma que se admite.

También puede usar la API de [reconocimiento óptico de caracteres \(OCR\)](#) para extraer el texto impreso en varios idiomas. Si es necesario, OCR corrige el giro del texto reconocido y proporciona las coordenadas del marco de cada palabra. El OCR admite 25 idiomas y detecta automáticamente el idioma del texto reconocido.

Moderación del contenido de las imágenes

Puede usar Computer Vision para [detectar contenido para adultos](#) en una imagen y devolver puntuaciones de confianza en las distintas clasificaciones. El umbral para el etiquetado de contenido se puede establecer en una escala deslizante, con el fin de que pueda ajustarlo a sus preferencias.

Uso de contenedores

[Use contenedores de Computer Vision](#) para reconocer texto impreso y manuscrito localmente, mediante la instalación de un contenedor de Docker estándar más cercano a los datos.

Requisitos de imagen

Computer Vision puede analizar las imágenes que cumplan los requisitos siguientes:

- La imagen se debe presentar en formato JPEG, PNG, GIF o BMP
- El tamaño de archivo de la imagen debe ser inferior a 4 megabytes (MB)
- Las dimensiones de la imagen deben ser mayores que 50 x 50 píxeles
 - Para la API Read, las dimensiones de la imagen deben estar entre 50 x 50 y 10 000 x 10 000 píxeles.

Seguridad y privacidad de datos

Al igual que sucede con todas las instancias de Cognitive Services, los desarrolladores que usan el servicio Computer Vision deben estar al tanto de las directivas de Microsoft sobre los datos de clientes. Para más información, consulte la [página de Cognitive Services](#) en Microsoft Trust Center.

Pasos siguientes

Introducción a Computer Vision, con una guía de inicio rápido:

- [Inicio rápido: SDK de Computer Vision para .NET](#)
- [Inicio rápido: SDK de Python para Computer Vision](#)
- [Inicio rápido: SDK de Java para Computer Vision](#)

Inicio rápido: Biblioteca cliente de Computer Vision para .NET

14/01/2020 • 16 minutes to read • [Edit Online](#)

Introducción a la biblioteca cliente de Computer Vision para .NET. Siga estos pasos para instalar el paquete y probar el código de ejemplo para realizar tareas básicas. Computer Vision proporciona acceso a algoritmos avanzados para procesar imágenes y devolver información.

Puede utilizar la biblioteca cliente de Computer Vision para .NET para:

- Analizar una imagen para ver las etiquetas, la descripción de texto, las caras, el contenido para adultos, etc.
- Reconocer texto manuscrito e impreso con la API Read para Batch.

[Documentación de referencia](#) | [Código fuente de la biblioteca](#) | [Paquete \(NuGet\)](#) | [Ejemplos](#)

Prerequisitos

- Una suscripción a Azure: [cree una cuenta gratuita](#)
- La versión actual de [.NET Core](#).

Instalación

Creación de un recurso de Computer Vision en Azure

Los servicios de Azure Cognitive Services se representan por medio de recursos de Azure a los que se suscribe. Cree un recurso para Computer Vision con [Azure Portal](#) o la [CLI de Azure](#) en la máquina local. También puede:

- Obtener una [clave de prueba](#) válida durante siete días de forma gratuita. Después de registrarse, estará disponible en el [sitio web de Azure](#).
- Ver el recurso en [Azure Portal](#)

Después de obtener una clave de la suscripción de evaluación o el recurso, [cree una variable de entorno](#) para la clave y la dirección URL del punto de conexión, denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación de una aplicación de C#

Cree una nueva aplicación de consola de .NET Core en el IDE o editor que prefiera.

En una ventana de consola (por ejemplo, cmd, PowerShell o Bash), use el comando `dotnet new` para crear una nueva aplicación de consola con el nombre `computer-vision-quickstart`. Este comando crea un sencillo proyecto "Hola mundo" de C# con un solo archivo de origen: *Program.cs*.

```
dotnet new console -n computer-vision-quickstart
```

Cambie el directorio a la carpeta de aplicaciones recién creada. Para compilar la aplicación:

```
dotnet build
```

La salida de la compilación no debe contener advertencias ni errores.

```
...
Build succeeded.
    0 Warning(s)
    0 Error(s)
...
```

En el directorio del proyecto, abra el archivo *Program.cs* en el editor o IDE que prefiera. Agregue las siguientes directivas `using` :

```
using System;
using System.Collections.Generic;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;
using System.Threading.Tasks;
using System.IO;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
```

En la clase **Program** de la aplicación, cree variables para el punto de conexión y la clave de Azure del recurso.

```
// Add your Computer Vision subscription key and endpoint to your environment variables.
// Close/reopen your project for them to take effect.
static string subscriptionKey = Environment.GetEnvironmentVariable("COMPUTER_VISION_SUBSCRIPTION_KEY");
static string endpoint = Environment.GetEnvironmentVariable("COMPUTER_VISION_ENDPOINT");
```

Instalación de la biblioteca cliente

Dentro del directorio de aplicaciones, instale la biblioteca cliente de Computer Vision para .NET con el siguiente comando:

```
dotnet add package Microsoft.Azure.CognitiveServices.Vision.ComputerVision --version 5.0.0
```

Si usa el IDE de Visual Studio, la biblioteca cliente estará disponible como un paquete de NuGet descargable.

Modelo de objetos

Las siguientes clases e interfaces controlan algunas de las características principales del SDK de .NET para Computer Vision.

NOMBRE	DESCRIPCIÓN
ComputerVisionClient	Esta clase es necesaria para todas las funcionalidades de Computer Vision. Cree una instancia de ella con la información de suscripción y úsela para realizar la mayoría de las operaciones con imágenes.
ComputerVisionClientExtensions	Esta clase contiene métodos adicionales para ComputerVisionClient .
VisualFeatureTypes	Esta enumeración define los diferentes tipos de análisis de imágenes que se pueden realizar en una operación de análisis estándar. Debe especificar un conjunto de valores de VisualFeatureTypes en función de sus necesidades.

Ejemplos de código

En estos fragmentos de código se muestra cómo realizar las siguientes tareas con la biblioteca cliente de Computer Vision para .NET:

- [Autenticar el cliente](#)
- [Analizar una imagen](#)
- [Leer texto manuscrito e impreso](#)

Autenticar el cliente

NOTE

En este inicio rápido se da por supuesto que ha [creado variables de entorno](#) para la clave de Computer Vision y el punto de conexión, denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT` respectivamente.

En un nuevo método, cree una instancia de un cliente con la clave y el punto de conexión. Cree un objeto [CognitiveServicesCredentials](#) con la clave y úselo con el punto de conexión para crear un objeto [ComputerVisionClient](#).

```
/*
 * AUTHENTICATE
 * Creates a Computer Vision client used by each example.
 */
public static ComputerVisionClient Authenticate(string endpoint, string key)
{
    ComputerVisionClient client =
        new ComputerVisionClient(new ApiKeyServiceClientCredentials(key))
        { Endpoint = endpoint };
    return client;
}
```

Es probable que desee llamar a este método en el método `Main`.

```
// Create a client
ComputerVisionClient client = Authenticate(endpoint, subscriptionKey);
```

Análisis de una imagen

El código siguiente define un método, `AnalyzeImageUrl`, que utiliza el objeto de cliente para analizar una imagen remota e imprimir los resultados. El método devuelve una descripción de texto, categorización, lista de etiquetas, caras detectadas, marcas de contenido para adultos, colores principales y tipo de imagen.

Agregue la llamada al método en el método `Main`.

```
// Analyze an image to get features and other properties.
AnalyzeImageUrl(client, ANALYZE_URL_IMAGE).Wait();
```

Configuración de una imagen de prueba

En la clase **Program**, guarde una referencia a la dirección URL de la imagen que desea analizar.

```
// URL image used for analyzing an image (image of puppy)
private const string ANALYZE_URL_IMAGE =
    "https://moderatorsampleimages.blob.core.windows.net/samples/sample16.png";
```

NOTE

También puede analizar una imagen local. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes locales.

Especificación de características visuales

Defina el nuevo método para el análisis de imágenes. Agregue el código siguiente, que especifica qué características visuales desea extraer en el análisis. Vea la enumeración [VisualFeatureTypes](#) para obtener una lista completa.

```
/*
 * ANALYZE IMAGE - URL IMAGE
 * Analyze URL image. Extracts captions, categories, tags, objects, faces, racy/adult content,
 * brands, celebrities, landmarks, color scheme, and image types.
 */
public static async Task AnalyzeImageUrl(ComputerVisionClient client, string imageUrl)
{
    Console.WriteLine("-----");
    Console.WriteLine("ANALYZE IMAGE - URL");
    Console.WriteLine();

    // Creating a list that defines the features to be extracted from the image.
    List<VisualFeatureTypes> features = new List<VisualFeatureTypes>()
    {
        VisualFeatureTypes.Categories, VisualFeatureTypes.Description,
        VisualFeatureTypes.Faces, VisualFeatureTypes.ImageType,
        VisualFeatureTypes.Tags, VisualFeatureTypes.Adult,
        VisualFeatureTypes.Color, VisualFeatureTypes.Brands,
        VisualFeatureTypes.Objects
    };
};
```

Analizar

El método **AnalyzeImageAsync** devuelve un objeto **ImageAnalysis** que contiene toda la información extraída.

```
Console.WriteLine($"Analyzing the image {Path.GetFileName(imageUrl)}...");
Console.WriteLine();
// Analyze the URL image
ImageAnalysis results = await client.AnalyzeImageAsync(imageUrl, features);
```

En las secciones siguientes se muestra cómo analizar esta información en detalle.

Obtención de la descripción de la imagen

El código siguiente obtiene la lista de títulos generados para la imagen. Consulte [Descripción de imágenes](#) para más detalles.

```
// Summarizes the image content.
Console.WriteLine("Summary:");
foreach (var caption in results.Description.Captions)
{
    Console.WriteLine($"{caption.Text} with confidence {caption.Confidence}");
}
Console.WriteLine();
```

Obtención de la categoría de imagen

El código siguiente obtiene la categoría detectada de la imagen. Consulte [Categorización de imágenes](#) para más detalles.

```
// Display categories the image is divided into.
Console.WriteLine("Categories:");
foreach (var category in results.Categories)
{
    Console.WriteLine($"{category.Name} with confidence {category.Score}");
}
Console.WriteLine();
```

Obtención de etiquetas de imagen

El código siguiente obtiene el conjunto de las etiquetas detectadas en la imagen. Consulte [Etiquetas de contenido](#) para más detalles.

```
// Image tags and their confidence score
Console.WriteLine("Tags:");
foreach (var tag in results.Tags)
{
    Console.WriteLine($"{tag.Name} {tag.Confidence}");
}
Console.WriteLine();
```

Detección de objetos

El código siguiente detecta objetos comunes en la imagen y los imprime en la consola. Consulte [Detección de objetos](#) para más información.

```
// Objects
Console.WriteLine("Objects:");
foreach (var obj in results.Objects)
{
    Console.WriteLine($"{obj.ObjectProperty} with confidence {obj.Confidence} at location {obj.Rectangle.X}, "
+
    $"{obj.Rectangle.X + obj.Rectangle.W}, {obj.Rectangle.Y}, {obj.Rectangle.Y + obj.Rectangle.H}");
}
Console.WriteLine();
```

Detección de marcas

El código siguiente detecta marcas corporativas y logotipos en la imagen y los imprime en la consola. Consulte [Detección de marcas](#) para más información.

```
// Well-known (or custom, if set) brands.
Console.WriteLine("Brands:");
foreach (var brand in results.Brands)
{
    Console.WriteLine($"Logo of {brand.Name} with confidence {brand.Confidence} at location
{brand.Rectangle.X}, " +
    $"{brand.Rectangle.X + brand.Rectangle.W}, {brand.Rectangle.Y}, {brand.Rectangle.Y +
brand.Rectangle.H}");
}
Console.WriteLine();
```

Detección de caras

El código siguiente devuelve las caras detectadas en la imagen con sus coordenadas de rectángulo y selecciona los atributos de cara. Consulte [Detección de caras](#) para más información.

```
// Faces
Console.WriteLine("Faces:");
foreach (var face in results.Faces)
{
    Console.WriteLine($"A {face.Gender} of age {face.Age} at location {face.FaceRectangle.Left}, " +
        $"{face.FaceRectangle.Left}, {face.FaceRectangle.Top + face.FaceRectangle.Width}, " +
        $"{face.FaceRectangle.Top + face.FaceRectangle.Height}");
}
Console.WriteLine();
```

Detección de contenido para adultos, explícito o sangriento

El siguiente código imprime la presencia detectada de contenido para adultos en la imagen. Para más información, consulte [Contenido para adultos, subido de tono y sangriento](#).

```
// Adult or racy content, if any.
Console.WriteLine("Adult:");
Console.WriteLine($"Has adult content: {results.Adult.IsAdultContent} with confidence {results.Adult.AdultScore}");
Console.WriteLine($"Has racy content: {results.Adult.IsRacyContent} with confidence {results.Adult.RacyScore}");
Console.WriteLine();
```

Obtención de la combinación de colores de imagen

El código siguiente imprime los atributos de color detectados en la imagen, como los colores dominantes y el color de énfasis. Consulte [Combinaciones de colores](#) para más detalles.

```
// Identifies the color scheme.
Console.WriteLine("Color Scheme:");
Console.WriteLine("Is black and white?: " + results.Color.IsBWImg);
Console.WriteLine("Accent color: " + results.Color.AccentColor);
Console.WriteLine("Dominant background color: " + results.Color.DominantColorBackground);
Console.WriteLine("Dominant foreground color: " + results.Color.DominantColorForeground);
Console.WriteLine("Dominant colors: " + string.Join(", ", results.Color.DominantColors));
Console.WriteLine();
```

Obtención de contenido específico del dominio

Computer Vision puede usar modelos especializados para realizar análisis adicionales en las imágenes. Consulte [Contenido específico del dominio](#) para más detalles.

En el código siguiente se analizan los datos sobre las celebridades detectadas en la imagen.

```
// Celebrities in image, if any.
Console.WriteLine("Celebrities:");
foreach (var category in results.Categories)
{
    if (category.Detail?.Celebrities != null)
    {
        foreach (var celeb in category.Detail.Celebrities)
        {
            Console.WriteLine($"{celeb.Name} with confidence {celeb.Confidence} at location {celeb.FaceRectangle.Left}, " +
                $"{celeb.FaceRectangle.Top}, {celeb.FaceRectangle.Height}, {celeb.FaceRectangle.Width}");
        }
    }
}
Console.WriteLine();
```

En el código siguiente se analizan los datos sobre los paisajes detectados en la imagen.

```
// Popular landmarks in image, if any.
Console.WriteLine("Landmarks:");
foreach (var category in results.Categories)
{
    if (category.Detail?.Landmarks != null)
    {
        foreach (var landmark in category.Detail.Landmarks)
        {
            Console.WriteLine($"{landmark.Name} with confidence {landmark.Confidence}");
        }
    }
}
Console.WriteLine();
```

Obtención del tipo de imagen

El código siguiente imprime información sobre el tipo de imagen (si es una imagen prediseñada o dibujo lineal).

```
// Detects the image types.
Console.WriteLine("Image Type:");
Console.WriteLine("Clip Art Type: " + results.ImageType.ClipArtType);
Console.WriteLine("Line Drawing Type: " + results.ImageType.LineDrawingType);
Console.WriteLine();
```

Lectura de texto manuscrito e impreso

Computer Vision puede leer texto visible de una imagen y convertirlo en un flujo de caracteres. El código de esta sección define un método, `ExtractTextUrl`, que utiliza el objeto de cliente para detectar y extraer texto impreso o manuscrito de la imagen.

Agregue la llamada al método en el método `Main`.

```
// Read the batch text from an image (handwriting and/or printed).
BatchReadFileUrl(client, EXTRACT_TEXT_URL_IMAGE).Wait();
BatchReadFileLocal(client, EXTRACT_TEXT_LOCAL_IMAGE).Wait();
```

Configuración de una imagen de prueba

En la clase **Program**, guarde una referencia de la dirección URL de la imagen de la que desea extraer texto.

```
private const string EXTRACT_TEXT_URL_IMAGE =
    "https://moderatorsampleimages.blob.core.windows.net/samples/sample2.jpg";
// URL image for OCR (optical character recognition). (Image of motivational meme).
```

NOTE

También puede extraer texto de una imagen local. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes locales.

Llamada a la API Read

Defina el nuevo método para leer texto. Agregue el código siguiente, que llama al método **BatchReadFileAsync** para la imagen especificada. Esto devuelve un identificador de operación e inicia un proceso asíncrono para leer el contenido de la imagen.

```

/*
 * BATCH READ FILE - URL IMAGE
 * Recognizes handwritten text.
 * This API call offers an improvement of results over the Recognize Text calls.
 */
public static async Task BatchReadFileUrl(ComputerVisionClient client, string urlImage)
{
    Console.WriteLine("-----");
    Console.WriteLine("BATCH READ FILE - URL IMAGE");
    Console.WriteLine();

    // Read text from URL
    BatchReadFileHeaders textHeaders = await client.BatchReadFileAsync(urlImage);
    // After the request, get the operation location (operation ID)
    string operationLocation = textHeaders.OperationLocation;
}

```

Obtención de resultados de lectura

A continuación, obtenga el identificador de operación devuelto de la llamada a **BatchReadFileAsync** y úselo para consultar los resultados de la operación en el servicio. El código siguiente comprueba la operación a intervalos de un segundo hasta que se devuelven los resultados. A continuación, imprime los datos de texto extraídos en la consola.

```

// Retrieve the URI where the recognized text will be stored from the Operation-Location header.
// We only need the ID and not the full URL
const int numberOfCharsInOperationId = 36;
string operationId = operationLocation.Substring(operationLocation.Length - numberOfCharsInOperationId);

// Extract the text
// Delay is between iterations and tries a maximum of 10 times.
int i = 0;
int maxRetries = 10;
ReadOperationResult results;
Console.WriteLine($"Extracting text from URL image {Path.GetFileName(urlImage)}...");
Console.WriteLine();
do
{
    {
        results = await client.GetReadOperationResultAsync(operationId);
        Console.WriteLine("Server status: {0}, waiting {1} seconds...", results.Status, i);
        await Task.Delay(1000);
        if (i == 9)
    }
    {
        Console.WriteLine("Server timed out.");
    }
}
while ((results.Status == TextOperationStatusCodes.Running ||
        results.Status == TextOperationStatusCodes.NotStarted) && i++ < maxRetries);

```

Visualización de resultados de lectura

Agregue el código siguiente para analizar y mostrar los datos de texto recuperados y finalice la definición del método.

```
// Display the found text.
Console.WriteLine();
var textRecognitionLocalFileResults = results.RecognitionResults;
foreach (TextRecognitionResult recResult in textRecognitionLocalFileResults)
{
    foreach (Line line in recResult.Lines)
    {
        Console.WriteLine(line.Text);
    }
}
Console.WriteLine();
}
```

Ejecución de la aplicación

Ejecute la aplicación desde el directorio de la aplicación con el comando `dotnet run`.

```
dotnet run
```

Limpieza de recursos

Si quiere limpiar y eliminar una suscripción a Cognitive Services, puede eliminar el recurso o grupo de recursos. Al eliminar el grupo de recursos, también se elimina cualquier otro recurso que esté asociado a él.

- [Portal](#)
- [CLI de Azure](#)

Pasos siguientes

[Referencia de la API Computer Vision \(.NET\)](#)

- [¿Qué es Computer Vision?](#)
- El código fuente de este ejemplo está disponible en [GitHub](#).

Inicio rápido: Biblioteca cliente de Computer Vision para Python

14/01/2020 • 16 minutes to read • [Edit Online](#)

El servicio Computer Vision proporciona a los desarrolladores acceso a algoritmos avanzados para procesar imágenes y devolver información. Los algoritmos de Computer Vision analizan el contenido de una imagen de diferentes formas, en función de las características visuales que le interesen.

Puede utilizar la biblioteca cliente de Computer Vision para Python para:

- Analizar una imagen para ver las etiquetas, la descripción de texto, las caras, el contenido para adultos, etc.
- Reconocer texto manuscrito e impreso con la API Read para Batch.

NOTE

Los escenarios de este inicio rápido usan direcciones URL de imágenes remotas. Para ver un código de ejemplo que realiza las mismas operaciones en imágenes locales, consulte el código en [GitHub](#).

[Documentación de referencia](#) | [Código fuente de la biblioteca](#) | [Paquete \(PiPy\)](#) | [Ejemplos](#)

Prerequisites

- Una suscripción a Azure: [cree una cuenta gratuita](#)
- [Python 3.x](#)

Instalación

Creación de un recurso de Computer Vision en Azure

Los servicios de Azure Cognitive Services se representan por medio de recursos de Azure a los que se suscribe. Cree un recurso para Computer Vision con [Azure Portal](#) o la [CLI de Azure](#) en la máquina local. También puede:

- Obtener una [clave de prueba](#) válida durante siete días de forma gratuita. Después de registrarse, estará disponible en el [sitio web de Azure](#).
- Ver este recurso en [Azure Portal](#).

Después de obtener una clave de la suscripción de evaluación o el recurso, [cree una variable de entorno](#) para la clave y la dirección URL del punto de conexión, denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación de una nueva aplicación de Python

Cree un nuevo script de Python, por ejemplo, `—quickstart-file.py`. Ábralo en el editor o el IDE que prefiera e importe las siguientes bibliotecas.


```

from azure.cognitiveservices.vision.computervision import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models import TextOperationStatusCodes
from azure.cognitiveservices.vision.computervision.models import TextRecognitionMode
from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes
from msrest.authentication import CognitiveServicesCredentials

from array import array
import os
from PIL import Image
import sys
import time

```

A continuación, cree variables para el punto de conexión y la clave de Azure del recurso.

```

# Add your Computer Vision subscription key to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()
# Add your Computer Vision endpoint to your environment variables.
if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']
else:
    print("\nSet the COMPUTER_VISION_ENDPOINT environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

```

NOTE

Si ha creado la variable de entorno después de haber iniciado la aplicación, deberá cerrar y volver a abrir el editor, el IDE o el shell que lo ejecuta para acceder a la variable.

Instalación de la biblioteca cliente

Puede instalar la biblioteca cliente con lo siguiente:

```

pip install --upgrade azure-cognitiveservices-vision-computervision

```

Modelo de objetos

Las siguientes clases e interfaces controlan algunas de las características principales del SDK de Python para Computer Vision.

NOMBRE	DESCRIPCIÓN
ComputerVisionClientOperationsMixin	Esta clase controla directamente todas las operaciones de imagen, como el análisis de imágenes, la detección de texto y la generación de miniaturas.
ComputerVisionClient	Esta clase es necesaria para todas las funcionalidades de Computer Vision. Cree una instancia de ella con la información de suscripción y úsela para generar instancias de otras clases. Implementa ComputerVisionClientOperationsMixin .

NOMBRE	DESCRIPCIÓN
VisualFeatureTypes	Esta enumeración define los diferentes tipos de análisis de imágenes que se pueden realizar en una operación de análisis estándar. Debe especificar un conjunto de valores de VisualFeatureTypes en función de sus necesidades.

Ejemplos de código

En estos fragmentos de código se muestra cómo realizar las siguientes tareas con la biblioteca cliente de Computer Vision para Python:

- [Autenticar el cliente](#)
- [Analizar una imagen](#)
- [Leer texto manuscrito e impreso](#)

Autenticar el cliente

NOTE

En este inicio rápido se da por supuesto que ha [creado una variable de entorno](#) para la clave de Computer Vision, denominada `COMPUTER_VISION_SUBSCRIPTION_KEY`.

Cree una instancia de un cliente con la clave y el punto de conexión. Cree un objeto [CognitiveServicesCredentials](#) con la clave y úselo con el punto de conexión para crear un objeto [ComputerVisionClient](#).

```
computervision_client = ComputerVisionClient(endpoint, CognitiveServicesCredentials(subscription_key))
```

Análisis de una imagen

Guarde una referencia a la dirección URL de una imagen que desee analizar.

```
remote_image_url = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/landmark.jpg"
```

Obtención de la descripción de la imagen

El código siguiente obtiene la lista de títulos generados para la imagen. Consulte [Descripción de imágenes](#) para más detalles.

```
'''
Describe an Image - remote
This example describes the contents of an image with the confidence score.
'''

print("==== Describe an image - remote ====")
# Call API
description_results = computervision_client.describe_image(remote_image_url )

# Get the captions (descriptions) from the response, with confidence level
print("Description of remote image: ")
if (len(description_results.captions) == 0):
    print("No description detected.")
else:
    for caption in description_results.captions:
        print('{}' with confidence {:.2f}%".format(caption.text, caption.confidence * 100))
```

Obtención de la categoría de imagen

El código siguiente obtiene la categoría detectada de la imagen. Consulte [Categorización de imágenes](#) para más detalles.

```
'''
Categorize an Image - remote
This example extracts (general) categories from a remote image with a confidence score.
'''

print("==== Categorize an image - remote ====")
# Select the visual feature(s) you want.
remote_image_features = ["categories"]
# Call API with URL and features
categorize_results_remote = computervision_client.analyze_image(remote_image_url , remote_image_features)

# Print results with confidence score
print("Categories from remote image: ")
if (len(categorize_results_remote.categories) == 0):
    print("No categories detected.")
else:
    for category in categorize_results_remote.categories:
        print('{}' with confidence {:.2f}%".format(category.name, category.score * 100))
```

Obtención de etiquetas de imagen

El código siguiente obtiene el conjunto de las etiquetas detectadas en la imagen. Consulte [Etiquetas de contenido](#) para más detalles.

```
'''
Tag an Image - remote
This example returns a tag (key word) for each thing in the image.
'''

print("==== Tag an image - remote ====")
# Call API with remote image
tags_result_remote = computervision_client.tag_image(remote_image_url )

# Print results with confidence score
print("Tags in the remote image: ")
if (len(tags_result_remote.tags) == 0):
    print("No tags detected.")
else:
    for tag in tags_result_remote.tags:
        print('{}' with confidence {:.2f}%".format(tag.name, tag.confidence * 100))
```

Detección de objetos

El código siguiente detecta objetos comunes en la imagen y los imprime en la consola. Consulte [Detección de](#)

[objetos](#) para más información.

```
'''
Detect Objects - remote
This example detects different kinds of objects with bounding boxes in a remote image.
'''

print("==== Detect Objects - remote ====")
# Get URL image with different objects
remote_image_url_objects = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/objects.jpg"
# Call API with URL
detect_objects_results_remote = computervision_client.detect_objects(remote_image_url_objects)

# Print detected objects results with bounding boxes
print("Detecting objects in remote image:")
if len(detect_objects_results_remote.objects) == 0:
    print("No objects detected.")
else:
    for object in detect_objects_results_remote.objects:
        print("object at location {}, {}, {}, {}".format( \
            object.rectangle.x, object.rectangle.x + object.rectangle.w, \
            object.rectangle.y, object.rectangle.y + object.rectangle.h))
```

Detección de marcas

El código siguiente detecta marcas corporativas y logotipos en la imagen y los imprime en la consola. Consulte [Detección de marcas](#) para más información.

```
'''
Detect Brands - remote
This example detects common brands like logos and puts a bounding box around them.
'''

print("==== Detect Brands - remote ====")
# Get a URL with a brand logo
remote_image_url = "https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/images/gray-shirt-logo.jpg"
# Select the visual feature(s) you want
remote_image_features = ["brands"]
# Call API with URL and features
detect_brands_results_remote = computervision_client.analyze_image(remote_image_url, remote_image_features)

print("Detecting brands in remote image: ")
if len(detect_brands_results_remote.brands) == 0:
    print("No brands detected.")
else:
    for brand in detect_brands_results_remote.brands:
        print("'{}' brand detected with confidence {:.1f}% at location {}, {}, {}, {}".format( \
            brand.name, brand.confidence * 100, brand.rectangle.x, brand.rectangle.x + brand.rectangle.w, \
            brand.rectangle.y, brand.rectangle.y + brand.rectangle.h))
```

Detección de caras

El código siguiente devuelve las caras detectadas en la imagen con sus coordenadas de rectángulo y selecciona los atributos de cara. Consulte [Detección de caras](#) para más información.

```

'''
Detect Faces - remote
This example detects faces in a remote image, gets their gender and age,
and marks them with a bounding box.
'''

print("==== Detect Faces - remote ====")
# Get an image with faces
remote_image_url_faces = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/faces.jpg"
# Select the visual feature(s) you want.
remote_image_features = ["faces"]
# Call the API with remote URL and features
detect_faces_results_remote = computervision_client.analyze_image(remote_image_url_faces,
remote_image_features)

# Print the results with gender, age, and bounding box
print("Faces in the remote image: ")
if (len(detect_faces_results_remote.faces) == 0):
    print("No faces detected.")
else:
    for face in detect_faces_results_remote.faces:
        print("{}' of age {} at location {}, {}, {}, {}".format(face.gender, face.age, \
        face.face_rectangle.left, face.face_rectangle.top, \
        face.face_rectangle.left + face.face_rectangle.width, \
        face.face_rectangle.top + face.face_rectangle.height))

```

Detección de contenido para adultos, explícito o sangriento

El siguiente código imprime la presencia detectada de contenido para adultos en la imagen. Para más información, consulte [Contenido para adultos, subido de tono y sangriento](#).

```

'''
Detect Adult or Racy Content - remote
This example detects adult or racy content in a remote image, then prints the adult/racy score.
The score is ranged 0.0 - 1.0 with smaller numbers indicating negative results.
'''

print("==== Detect Adult or Racy Content - remote ====")
# Select the visual feature(s) you want
remote_image_features = ["adult"]
# Call API with URL and features
detect_adult_results_remote = computervision_client.analyze_image(remote_image_url, remote_image_features)

# Print results with adult/racy score
print("Analyzing remote image for adult or racy content ... ")
print("Is adult content: {} with confidence {:.2f}".format(detect_adult_results_remote.adult.is_adult_content,
detect_adult_results_remote.adult.adult_score * 100))
print("Has racy content: {} with confidence {:.2f}".format(detect_adult_results_remote.adult.is_racy_content,
detect_adult_results_remote.adult.racy_score * 100))

```

Obtención de la combinación de colores de imagen

El código siguiente imprime los atributos de color detectados en la imagen, como los colores dominantes y el color de énfasis. Consulte [Combinaciones de colores](#) para más detalles.

```

'''
Detect Color - remote
This example detects the different aspects of its color scheme in a remote image.
'''

print("==== Detect Color - remote =====")
# Select the feature(s) you want
remote_image_features = ["color"]
# Call API with URL and features
detect_color_results_remote = computervision_client.analyze_image(remote_image_url, remote_image_features)

# Print results of color scheme
print("Getting color scheme of the remote image: ")
print("Is black and white: {}".format(detect_color_results_remote.color.is_bw_img))
print("Accent color: {}".format(detect_color_results_remote.color.accent_color))
print("Dominant background color: {}".format(detect_color_results_remote.color.dominant_color_background))
print("Dominant foreground color: {}".format(detect_color_results_remote.color.dominant_color_foreground))
print("Dominant colors: {}".format(detect_color_results_remote.color.dominant_colors))

```

Obtención de contenido específico del dominio

Computer Vision puede usar un modelo especializado para realizar análisis adicionales en las imágenes. Consulte [Contenido específico del dominio](#) para más detalles.

En el código siguiente se analizan los datos sobre las celebridades detectadas en la imagen.

```

'''
Detect Domain-specific Content - remote
This example detects celebrities and landmarks in remote images.
'''

print("==== Detect Domain-specific Content - remote =====")
# URL of one or more celebrities
remote_image_url_celebs = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/faces.jpg"
# Call API with content type (celebrities) and URL
detect_domain_results_celebs_remote = computervision_client.analyze_image_by_domain("celebrities",
remote_image_url_celebs)

# Print detection results with name
print("Celebrities in the remote image:")
if len(detect_domain_results_celebs_remote.result["celebrities"]) == 0:
    print("No celebrities detected.")
else:
    for celeb in detect_domain_results_celebs_remote.result["celebrities"]:
        print(celeb["name"])

```

En el código siguiente se analizan los datos sobre los paisajes detectados en la imagen.

```

# Call API with content type (landmarks) and URL
detect_domain_results_landmarks = computervision_client.analyze_image_by_domain("landmarks", remote_image_url)
print()

print("Landmarks in the remote image:")
if len(detect_domain_results_landmarks.result["landmarks"]) == 0:
    print("No landmarks detected.")
else:
    for landmark in detect_domain_results_landmarks.result["landmarks"]:
        print(landmark["name"])

```

Obtención del tipo de imagen

El código siguiente imprime información sobre el tipo de imagen (si es una imagen prediseñada o dibujo lineal).

```

'''
Detect Image Types - remote
This example detects an image's type (clip art/line drawing).
'''

print("==== Detect Image Types - remote ====")
# Get URL of an image with a type
remote_image_url_type = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/type-image.jpg"
# Select visual feature(s) you want
remote_image_features = VisualFeatureTypes.image_type
# Call API with URL and features
detect_type_results_remote = computervision_client.analyze_image(remote_image_url_type, remote_image_features)

# Prints type results with degree of accuracy
print("Type of remote image:")
if detect_type_results_remote.image_type.clip_art_type == 0:
    print("Image is not clip art.")
elif detect_type_results_remote.image_type.line_drawing_type == 1:
    print("Image is ambiguously clip art.")
elif detect_type_results_remote.image_type.line_drawing_type == 2:
    print("Image is normal clip art.")
else:
    print("Image is good clip art.")

if detect_type_results_remote.image_type.line_drawing_type == 0:
    print("Image is not a line drawing.")
else:
    print("Image is a line drawing")

```

Lectura de texto manuscrito e impreso

Computer Vision puede leer texto visible de una imagen y convertirlo en un flujo de caracteres. Esto se hace en dos partes.

Llamada a la API Read

En primer lugar, use el siguiente código para llamar al método **batch_read_file** para la imagen especificada. Esto devuelve un identificador de operación e inicia un proceso asíncronico para leer el contenido de la imagen.

```

'''
Batch Read File, recognize printed text - remote
This example will extract printed text in an image, then print results, line by line.
This API call can also recognize handwriting (not shown).
'''

print("==== Batch Read File - remote ====")
# Get an image with printed text
remote_image_printed_text_url = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/printed_text.jpg"

# Call API with URL and raw response (allows you to get the operation location)
recognize_printed_results = computervision_client.batch_read_file(remote_image_printed_text_url, raw=True)

```

Obtención de resultados de lectura

A continuación, obtenga el identificador de operación devuelto de la llamada a **batch_read_file** y úselo para consultar los resultados de la operación en el servicio. El código siguiente comprueba la operación a intervalos de un segundo hasta que se devuelven los resultados. A continuación, imprime los datos de texto extraídos en la consola.

```
# Get the operation location (URL with an ID at the end) from the response
operation_location_remote = recognize_printed_results.headers["Operation-Location"]
# Grab the ID from the URL
operation_id = operation_location_remote.split("/")[-1]

# Call the "GET" API and wait for it to retrieve the results
while True:
    get_printed_text_results = computervision_client.get_read_operation_result(operation_id)
    if get_printed_text_results.status not in ['NotStarted', 'Running']:
        break
    time.sleep(1)

# Print the detected text, line by line
if get_printed_text_results.status == TextOperationStatusCodes.succeeded:
    for text_result in get_printed_text_results.recognition_results:
        for line in text_result.lines:
            print(line.text)
            print(line.bounding_box)
print()
```

Ejecución de la aplicación

Ejecute la aplicación con el comando `python` en el archivo de inicio rápido.

```
python quickstart-file.py
```

Limpieza de recursos

Si quiere limpiar y eliminar una suscripción a Cognitive Services, puede eliminar el recurso o grupo de recursos. Al eliminar el grupo de recursos, también se elimina cualquier otro recurso que esté asociado a él.

- [Portal](#)
- [CLI de Azure](#)

Pasos siguientes

En este inicio rápido, ha aprendido a usar la biblioteca de Computer Vision para Python para realizar tareas básicas. A continuación, consulte la documentación de referencia para más información sobre la biblioteca.

[Referencia de la API Computer Vision \(Python\)](#)

- [¿Qué es Computer Vision?](#)
- El código fuente de este ejemplo está disponible en [GitHub](#).

Inicio rápido: Biblioteca de cliente de Computer Vision para Java

14/01/2020 • 16 minutes to read • [Edit Online](#)

Introducción a la biblioteca de cliente de Computer Vision para Java. Siga estos pasos para instalar el paquete y probar el código de ejemplo para realizar tareas básicas. Computer Vision proporciona acceso a algoritmos avanzados para procesar imágenes y devolver información.

Puede utilizar la biblioteca de cliente de Computer Vision para Java para:

- Analizar una imagen para ver las etiquetas, la descripción de texto, las caras, el contenido para adultos, etc.
- Reconocer texto manuscrito e impreso con la API Read para Batch.

[Documentación de referencia](#) | [Artefacto \(Maven\)](#) | [Ejemplos](#)

Prerequisitos

- Una suscripción a Azure: [cree una cuenta gratuita](#)
- La versión actual de [Java Development Kit \(JDK\)](#)
- La [herramienta de compilación de Gradle](#) u otro administrador de dependencias.

Instalación

Creación de un recurso de Computer Vision en Azure

Los servicios de Azure Cognitive Services se representan por medio de recursos de Azure a los que se suscribe. Cree un recurso para Computer Vision con [Azure Portal](#) o la [CLI de Azure](#) en la máquina local. También puede:

- Obtener una [clave de prueba](#) válida durante siete días de forma gratuita. Después de registrarse, estará disponible en el [sitio web de Azure](#).
- Ver el recurso en [Azure Portal](#)

Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas

`COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación de un proyecto de Gradle

En una ventana de la consola (como cmd, PowerShell o Bash), cree un directorio para la aplicación y vaya a él.

```
mkdir myapp && cd myapp
```

Ejecute el comando `gradle init` desde el directorio de trabajo. Este comando creará archivos de compilación esenciales para Gradle, como *build.gradle.kts*, que se usa en tiempo de ejecución para crear y configurar la aplicación.

```
gradle init --type basic
```

Cuando se le solicite que elija un **DSL**, seleccione **Kotlin**.

Busque *build.gradle.kts* y ábralo con el IDE o el editor de texto que prefiera. A continuación, cópielo en la siguiente configuración de compilación. Esta configuración define el proyecto como una aplicación Java cuyo punto de

entrada es la clase **ComputerVisionQuickstarts**. Importa la biblioteca de Computer Vision.

```
plugins {
    java
    application
}
application {
    mainClassName = "ComputerVisionQuickstarts"
}
repositories {
    mavenCentral()
}
```

En el directorio de trabajo, ejecute el siguiente comando para crear una carpeta de origen del proyecto:

```
mkdir -p src/main/java
```

Vaya a la nueva carpeta y cree un archivo denominado *ComputerVisionQuickstarts.java*. Ábralo en el editor o el IDE que prefiera y agregue las siguientes instrucciones `import` :

```
import com.microsoft.azure.cognitiveservices.vision.computervision.*;
import com.microsoft.azure.cognitiveservices.vision.computervision.models.*;

import java.io.File;
import java.io.FileInputStream;
import java.nio.file.Files;

import java.util.ArrayList;
import java.util.List;
```

A continuación, agregue una definición de clase para **ComputerVisionQuickstarts**.

Instalación de la biblioteca cliente

En este inicio rápido se usa el administrador de dependencias Gradle. Puede encontrar la biblioteca de cliente y la información de otros administradores de dependencias en el [repositorio central de Maven](#).

En el archivo *build.gradle.kts* del proyecto, incluya la biblioteca de cliente de Computer Vision como una dependencia.

```
dependencies {
    compile(group = "com.microsoft.azure.cognitiveservices", name = "azure-cognitiveservices-computervision",
        version = "1.0.2-beta")
}
```

Modelo de objetos

Las siguientes clases e interfaces controlan algunas de las características principales del SDK de Java para Computer Vision.

NOMBRE	DESCRIPCIÓN
ComputerVisionClient	Esta clase es necesaria para todas las funcionalidades de Computer Vision. Cree una instancia de ella con la información de suscripción y úsela para generar instancias de otras clases.

NOMBRE	DESCRIPCIÓN
ComputerVision	Esta clase procede del objeto de cliente y controla directamente todas las operaciones de imagen, como el análisis de imágenes, la detección de texto y la generación de miniaturas.
VisualFeatureTypes	Esta enumeración define los diferentes tipos de análisis de imágenes que se pueden realizar en una operación de análisis estándar. Debe especificar un conjunto de valores de VisualFeatureTypes en función de sus necesidades.

Ejemplos de código

En estos fragmentos de código se muestra cómo realizar las siguientes tareas con la biblioteca de cliente de Computer Vision para Java:

- [Autenticar el cliente](#)
- [Analizar una imagen](#)
- [Leer texto manuscrito e impreso](#)

Autenticar el cliente

NOTE

En este inicio rápido se da por supuesto que ha [creado una variable de entorno](#) para la clave de Computer Vision, denominada `COMPUTER_VISION_SUBSCRIPTION_KEY`.

El código siguiente agrega un método `main` a la clase y se crean variables para el punto de conexión y la clave de Azure del recurso. Tendrá que escribir su propia cadena de punto de conexión, que puede encontrar comprobando la sección de **Introducción** de Azure Portal.

```
public static void main(String[] args) {
    // Add your Computer Vision subscription key and endpoint to your environment
    // variables.
    // After setting, close and then re-open your command shell or project for the
    // changes to take effect.
    String subscriptionKey = System.getenv("COMPUTER_VISION_SUBSCRIPTION_KEY");
    String endpoint = System.getenv("COMPUTER_VISION_ENDPOINT");
```

A continuación, agregue el código siguiente para crear un objeto [ComputerVisionClient](#) y pasarlo a otros métodos, que definirá más adelante.

```
ComputerVisionClient compVisClient =
    ComputerVisionManager.authenticate(subscriptionKey).withEndpoint(endpoint);
// END - Create an authenticated Computer Vision client.

System.out.println("\nAzure Cognitive Services Computer Vision - Java Quickstart Sample");

// Analyze local and remote images
AnalyzeLocalImage(compVisClient);

// Recognize printed text with OCR for a local and remote (URL) image
RecognizeTextOCRLocal(compVisClient);
```

NOTE

Si ha creado la variable de entorno después de haber iniciado la aplicación, tendrá que cerrar y volver a abrir el editor, el IDE o el shell que lo ejecuta para acceder a la variable.

Análisis de una imagen

El código siguiente define un método, `AnalyzeLocalImage`, que utiliza el objeto de cliente para analizar una imagen local e imprimir los resultados. El método devuelve una descripción de texto, categorización, lista de etiquetas, caras detectadas, marcas de contenido para adultos, colores principales y tipo de imagen.

Configuración de una imagen de prueba

En primer lugar, cree una carpeta **recursos/** en la carpeta **src /main/** del proyecto y agregue una imagen que desee analizar. A continuación, agregue la siguiente definición de método a la clase **ComputerVisionQuickstarts**. Si es necesario, cambie el valor de `pathToLocalImage` para que coincida con el archivo de imagen.

```
public static void AnalyzeLocalImage(ComputerVisionClient compVisClient) {  
    /*  
     * Analyze a local image:  
     *  
     * Set a string variable equal to the path of a local image. The image path  
     * below is a relative path.  
     */  
    String pathToLocalImage = "src\\main\\resources\\myImage.jpg";
```

NOTE

También puede analizar una imagen remota mediante su dirección URL. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes remotas.

Especificación de características visuales

A continuación, especifique qué características visuales desea extraer en el análisis. Vea la enumeración [VisualFeatureTypes](#) para obtener una lista completa.

```
// This list defines the features to be extracted from the image.  
List<VisualFeatureTypes> featuresToExtractFromLocalImage = new ArrayList<>();  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.DESCRPTION);  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.CATEGORIES);  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.TAGS);  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.FACES);  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.ADULT);  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.COLOR);  
featuresToExtractFromLocalImage.add(VisualFeatureTypes.IMAGE_TYPE);
```

Analizar

Este método imprime resultados detallados en la consola para cada ámbito del análisis de imágenes. Se recomienda que incluya esta llamada de método en un bloque Try/Catch. El método **analyzeImageInStream** devuelve un objeto **ImageAnalysis** que contiene toda la información extraída.

```
// Need a byte array for analyzing a local image.
File rawImage = new File(pathToLocalImage);
byte[] imageByteArray = Files.readAllBytes(rawImage.toPath());

// Call the Computer Vision service and tell it to analyze the loaded image.
ImageAnalysis analysis = compVisClient.computerVision().analyzeImageInStream().withImage(imageByteArray)
    .withVisualFeatures(featuresToExtractFromLocalImage).execute();
```

En las secciones siguientes se muestra cómo analizar esta información en detalle.

Obtención de la descripción de la imagen

El código siguiente obtiene la lista de títulos generados para la imagen. Para más información, consulte [Descripción de imágenes](#).

```
// Display image captions and confidence values.
System.out.println("\nC captions: ");
for (ImageCaption caption : analysis.description().captions()) {
    System.out.printf("\'%s\' with confidence %f\n", caption.text(), caption.confidence());
}
```

Obtención de la categoría de imagen

El código siguiente obtiene la categoría detectada de la imagen. Para más información, consulte [Categorización de imágenes](#).

```
// Display image category names and confidence values.
System.out.println("\nC categories: ");
for (Category category : analysis.categories()) {
    System.out.printf("\'%s\' with confidence %f\n", category.name(), category.score());
}
```

Obtención de etiquetas de imagen

El código siguiente obtiene el conjunto de las etiquetas detectadas en la imagen. Para más información, consulte [Etiquetas de contenido](#).

```
// Display image tags and confidence values.
System.out.println("\nC tags: ");
for (ImageTag tag : analysis.tags()) {
    System.out.printf("\'%s\' with confidence %f\n", tag.name(), tag.confidence());
}
```

Detección de caras

El código siguiente devuelve las caras detectadas en la imagen con sus coordenadas de rectángulo y selecciona los atributos de cara. Para más información, consulte [Detección de caras](#).

```
// Display any faces found in the image and their location.
System.out.println("\nC faces: ");
for (FaceDescription face : analysis.faces()) {
    System.out.printf("\'%s\' of age %d at location (%d, %d), (%d, %d)\n", face.gender(), face.age(),
        face.faceRectangle().left(), face.faceRectangle().top(),
        face.faceRectangle().left() + face.faceRectangle().width(),
        face.faceRectangle().top() + face.faceRectangle().height());
}
```

Detección de contenido para adultos, explícito o sangriento

El siguiente código imprime la presencia detectada de contenido para adultos en la imagen. Para más información, consulte [Contenido para adultos, subido de tono y sangriento](#).

```
// Display whether any adult or racy content was detected and the confidence
// values.
System.out.println("\nAdult: ");
System.out.printf("Is adult content: %b with confidence %f\n", analysis.adult().isAdultContent(),
    analysis.adult().adultScore());
System.out.printf("Has racy content: %b with confidence %f\n", analysis.adult().isRacyContent(),
    analysis.adult().racyScore());
```

Obtención de la combinación de colores de imagen

El código siguiente imprime los atributos de color detectados en la imagen, como los colores dominantes y el color de énfasis. Para más información, consulte [Combinaciones de colores](#).

```
// Display the image color scheme.
System.out.println("\nColor scheme: ");
System.out.println("Is black and white: " + analysis.color().isBWImg());
System.out.println("Accent color: " + analysis.color().accentColor());
System.out.println("Dominant background color: " + analysis.color().dominantColorBackground());
System.out.println("Dominant foreground color: " + analysis.color().dominantColorForeground());
System.out.println("Dominant colors: " + String.join(", ", analysis.color().dominantColors()));
```

Obtención de contenido específico del dominio

Computer Vision puede usar un modelo especializado para realizar análisis adicionales en las imágenes. Para más información, consulte [Contenido específico del dominio](#).

En el código siguiente se analizan los datos sobre las celebridades detectadas en la imagen.

```
// Display any celebrities detected in the image and their locations.
System.out.println("\nCelebrities: ");
for (Category category : analysis.categories()) {
    if (category.detail() != null && category.detail().celebrities() != null) {
        for (CelebritiesModel celeb : category.detail().celebrities()) {
            System.out.printf("\'%s\' with confidence %f at location (%d, %d), (%d, %d)\n", celeb.name(),
                celeb.confidence(), celeb.faceRectangle().left(), celeb.faceRectangle().top(),
                celeb.faceRectangle().left() + celeb.faceRectangle().width(),
                celeb.faceRectangle().top() + celeb.faceRectangle().height());
        }
    }
}
```

En el código siguiente se analizan los datos sobre los paisajes detectados en la imagen.

```
// Display any landmarks detected in the image and their locations.
System.out.println("\nLandmarks: ");
for (Category category : analysis.categories()) {
    if (category.detail() != null && category.detail().landmarks() != null) {
        for (LandmarksModel landmark : category.detail().landmarks()) {
            System.out.printf("\'%s\' with confidence %f\n", landmark.name(), landmark.confidence());
        }
    }
}
```

Obtención del tipo de imagen

El código siguiente imprime información sobre el tipo de imagen (si es una imagen prediseñada o dibujo lineal).

```
// Display what type of clip art or line drawing the image is.
System.out.println("\nImage type:");
System.out.println("Clip art type: " + analysis.imageType().clipArtType());
System.out.println("Line drawing type: " + analysis.imageType().lineDrawingType());
```

Lectura de texto manuscrito e impreso

Computer Vision puede leer texto visible de una imagen y convertirlo en un flujo de caracteres.

NOTE

También puede leer el texto de una imagen remota mediante su dirección URL. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes remotas.

Llamada a Recognize API

En primer lugar, use el siguiente código para llamar al método **recognizePrintedTextInStream** para la imagen especificada. Al agregar este código al proyecto, debe reemplazar el valor de `localTextImagePath` por la ruta de acceso a la imagen local.

```
// Display what type of clip art or line drawing the image is.
System.out.println("\nImage type:");
System.out.println("Clip art type: " + analysis.imageType().clipArtType());
System.out.println("Line drawing type: " + analysis.imageType().lineDrawingType());
```

Impresión de los resultados del reconocimiento

El siguiente bloque de código procesa el texto devuelto y lo analiza para imprimir la primera palabra de cada línea. Puede usar este código para comprender rápidamente la estructura de una instancia **OcrResult**.

```
// Print results of local image
System.out.println();
System.out.println("Recognizing printed text from a local image with OCR ...");
System.out.println("\nLanguage: " + ocrResultLocal.language());
System.out.printf("Text angle: %1.3f\n", ocrResultLocal.textAngle());
System.out.println("Orientation: " + ocrResultLocal.orientation());

boolean firstWord = true;
// Gets entire region of text block
for (OcrRegion reg : ocrResultLocal.regions()) {
    // Get one line in the text block
    for (OcrLine line : reg.lines()) {
        for (OcrWord word : line.words()) {
            // get bounding box of first word recognized (just to demo)
            if (firstWord) {
                System.out.println("\nFirst word in first line is \"" + word.text()
                    + "\" with bounding box: " + word.boundingBox());
                firstWord = false;
                System.out.println();
            }
            System.out.print(word.text() + " ");
        }
        System.out.println();
    }
}
```

Por último, cierre el bloque try/catch y la definición del método.

```
} catch (Exception e) {  
    System.out.println(e.getMessage());  
    e.printStackTrace();  
}  
}
```

Ejecución de la aplicación

Puede compilar la aplicación con:

```
gradle build
```

Ejecute la aplicación con el comando `gradle run`:

```
gradle run
```

Limpieza de recursos

Si quiere limpiar y eliminar una suscripción a Cognitive Services, puede eliminar el recurso o grupo de recursos. Al eliminar el grupo de recursos, también se elimina cualquier otro recurso que esté asociado a él.

- [Portal](#)
- [CLI de Azure](#)

Pasos siguientes

En este inicio rápido ha aprendido a usar la biblioteca de Computer Vision para Java para realizar tareas básicas. A continuación, consulte la documentación de referencia para más información sobre la biblioteca.

[Referencia de Computer Vision \(Java\)](#)

- [¿Qué es Computer Vision?](#)
- El código fuente de este ejemplo está disponible en [GitHub](#).

Inicio rápido: Biblioteca cliente de Computer Vision para Node.js

14/01/2020 • 16 minutes to read • [Edit Online](#)

Introducción a la biblioteca cliente de Computer Vision para Node.js Siga estos pasos para instalar el paquete y probar el código de ejemplo para realizar tareas básicas.

Use la biblioteca cliente de Computer Vision para Node.js para:

- [Analizar una imagen](#)
- [Leer texto manuscrito e impreso](#)

[Documentación de referencia](#) | [Código fuente de la biblioteca](#) | [Paquete \(npm\)](#) | [Ejemplos](#)

Prerequisites

- Una suscripción a Azure: [cree una cuenta gratuita](#)
- La versión actual de [Node.js](#)

Instalación

Creación de un recurso de Computer Vision en Azure

Los servicios de Azure Cognitive Services se representan por medio de recursos de Azure a los que se suscribe. Cree un recurso para Computer Vision con [Azure Portal](#) o la [CLI de Azure](#) en la máquina local. También puede:

- Obtener una [clave de prueba](#) válida durante siete días de forma gratuita. Después de registrarse, estará disponible en el [sitio web de Azure](#).
- Ver el recurso en [Azure Portal](#)

Después de obtener una clave de la suscripción de prueba o del recurso, [cree las variables de entorno](#) para la clave y la dirección URL del punto de conexión, denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación de una aplicación Node.js

En una ventana de la consola (como cmd, PowerShell o Bash), cree un directorio para la aplicación y vaya a él.

```
mkdir myapp && cd myapp
```

Ejecute el comando `npm init` para crear una aplicación de nodo con un archivo `package.json`.

```
npm init
```

Instalación de la biblioteca cliente

Instale los paquetes `ms-rest-azure` y `azure-cognitiveservices-computervision` de NPM:

```
npm install azure-cognitiveservices-computervision ms-rest-azure
```

el archivo `package.json` de la aplicación se actualizará con las dependencias.

Preparación del script de Node.js

Cree un archivo, *index.js* y ábralo en un editor de texto. Agregue las siguientes instrucciones import.

```
'use strict';

const async = require('async');
const fs = require('fs');
const path = require("path");
const createReadStream = require('fs').createReadStream
const sleep = require('util').promisify(setTimeout);
const ComputerVisionClient = require('@azure/cognitiveservices-computervision').ComputerVisionClient;
const ApiKeyCredentials = require('@azure/ms-rest-js').ApiKeyCredentials;
```

Luego, defina una función `computerVision` y declare una serie asincrónica con la función principal y la función de devolución de llamada. Se agregará el código de inicio rápido a la función principal y se llamará a `computerVision` en la parte inferior del script.

```
function computerVision() {
  async.series([
    async function () {

    },
    function () {
      return new Promise((resolve) => {
        resolve();
      })
    }
  ], (err) => {
    throw (err);
  });
}

computerVision();
```

Modelo de objetos

Las siguientes clases e interfaces controlan algunas de las características principales del SDK de Node.js para Computer Vision.

NOMBRE	DESCRIPCIÓN
ComputerVisionClient	Esta clase es necesaria para todas las funcionalidades de Computer Vision. Cree una instancia de ella con la información de suscripción y úsela para realizar la mayoría de las operaciones con imágenes.
VisualFeatureTypes	Esta enumeración define los diferentes tipos de análisis de imágenes que se pueden realizar en una operación de análisis estándar. Debe especificar un conjunto de valores de VisualFeatureTypes en función de sus necesidades.

Ejemplos de código

En estos fragmentos de código se muestra cómo realizar las siguientes tareas con la biblioteca cliente de Computer Vision para Node.js:

- [Autenticar el cliente](#)
- [Analizar una imagen](#)
- [Leer texto manuscrito e impreso](#)

Autenticar el cliente

Cree variables para el punto de conexión y la clave de Azure del recurso. Si ha creado la variable de entorno después de haber iniciado la aplicación, deberá cerrar y volver a abrir el editor, el IDE o el shell que lo ejecuta para acceder a la variable.

```
/**
 * AUTHENTICATE
 * This single client is used for all examples.
 */
let key = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
let endpoint = process.env['COMPUTER_VISION_ENDPOINT'];
if (!key) { throw new Error('Set your environment variables for your subscription key and endpoint.')} }
```

Cree una instancia de un cliente con la clave y el punto de conexión. Cree un objeto [ApiKeyCredentials](#) con su clave y punto de conexión y úselo para crear un objeto [ComputerVisionClient](#).

```
let computerVisionClient = new ComputerVisionClient(
    new ApiKeyCredentials({inHeader: {'Ocp-Apim-Subscription-Key': key}}), endpoint);
```

Análisis de una imagen

El código de esta sección analiza las imágenes remotas para extraer varias características visuales. Puede realizar estas operaciones como parte del método **analyzeImage** del objeto cliente, o puede llamarlas mediante métodos individuales. Para más información, consulte la [documentación de referencia](#).

NOTE

También puede analizar una imagen local. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes locales.

Obtención de la descripción de la imagen

El código siguiente obtiene la lista de títulos generados para la imagen. Consulte [Descripción de imágenes](#) para más detalles.

En primer lugar, defina la dirección URL de la imagen que se va a analizar:

```
var describeURL = 'https://moderatorsampleimages.blob.core.windows.net/samples/sample1.jpg';
```

Después, agregue el código siguiente para obtener la descripción de la imagen e imprímala en la consola.

```
// Analyze URL image
console.log('Analyzing URL image to describe...', describeURL.split('/').pop());
var caption = (await computerVisionClient.describeImage(describeURL)).captions[0];
console.log(`This may be ${caption.text} (${caption.confidence.toFixed(2)} confidence)`);
```

Obtención de la categoría de imagen

El código siguiente obtiene la categoría detectada de la imagen. Consulte [Categorización de imágenes](#) para más

detalles.

```
const categoryURLImage = 'https://moderatorsampleimages.blob.core.windows.net/samples/sample16.png';

// Analyze URL image
console.log('Analyzing category in image...', categoryURLImage.split('/').pop());
let categories = (await computerVisionClient.analyzeImage(categoryURLImage)).categories;
console.log(`Categories: ${formatCategories(categories)}`);
```

Defina la función auxiliar `formatCategories` :

```
// Formats the image categories
function formatCategories(categories) {
  categories.sort((a, b) => b.score - a.score);
  return categories.map(cat => `${cat.name} (${cat.score.toFixed(2)})`).join(', ');
}
```

Obtención de etiquetas de imagen

El código siguiente obtiene el conjunto de las etiquetas detectadas en la imagen. Consulte [Etiquetas de contenido](#) para más detalles.

```
console.log('-----');
console.log('DETECT TAGS');
console.log();

// Image of different kind of dog.
const tagsURL = 'https://moderatorsampleimages.blob.core.windows.net/samples/sample16.png';

// Analyze URL image
console.log('Analyzing tags in image...', tagsURL.split('/').pop());
let tags = (await computerVisionClient.analyzeImage(tagsURL, {visualFeatures: ['Tags']})).tags;
console.log(`Tags: ${formatTags(tags)}`);
```

Defina la función auxiliar `formatTags` :

```
// Format tags for display
function formatTags(tags) {
  return tags.map(tag => `${tag.name} (${tag.confidence.toFixed(2)})`).join(', ');
}
```

Detección de objetos

El código siguiente detecta objetos comunes en la imagen y los imprime en la consola. Consulte [Detección de objetos](#) para más información.

```
// Image of a dog
const objectURL = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-node-sdk-samples/master/Data/image.jpg';

// Analyze a URL image
console.log('Analyzing objects in image...', objectURL.split('/').pop());
let objects = (await computerVisionClient.analyzeImage(objectURL, {visualFeatures: ['Objects']})).objects;
console.log();

// Print objects bounding box and confidence
if (objects.length) {
  console.log(`${objects.length} object${objects.length == 1 ? '' : 's'} found:`);
  for (let obj of objects) { console.log(`    ${obj.object} (${obj.confidence.toFixed(2)}) at
  ${formatRectObjects(obj.rectangle)}`); }
} else { console.log('No objects found.');
```

Defina la función auxiliar `formatRectObjects` :

```
// Formats the bounding box
function formatRectObjects(rect) {
  return `top=${rect.y}`.padEnd(10) + `left=${rect.x}`.padEnd(10) + `bottom=${rect.y + rect.h}`.padEnd(12)
  + `right=${rect.x + rect.w}`.padEnd(10) + `(${rect.w}x${rect.h})`;
}
```

Detección de marcas

El código siguiente detecta marcas corporativas y logotipos en la imagen y los imprime en la consola. Consulte [Detección de marcas](#) para más información.

```
const brandURLImage = 'https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/images/red-shirt-logo.jpg';

// Analyze URL image
console.log('Analyzing brands in image...', brandURLImage.split('/').pop());
let brands = (await computerVisionClient.analyzeImage(brandURLImage, {visualFeatures: ['Brands']})).brands;

// Print the brands found
if (brands.length) {
  console.log(`${brands.length} brand${brands.length != 1 ? 's' : ''} found:`);
  for (let brand of brands) {
    console.log(`    ${brand.name} (${brand.confidence.toFixed(2)} confidence)`);
  }
} else { console.log(`No brands found.`); }
```

Detección de caras

El código siguiente devuelve las caras detectadas en la imagen con sus coordenadas de rectángulo y selecciona los atributos de cara. Consulte [Detección de caras](#) para más información.

```
const facesImageUrl = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/faces.jpg';

// Analyze URL image.
console.log('Analyzing faces in image...', facesImageUrl.split('/').pop());
// Get the visual feature for 'Faces' only.
let faces = (await computerVisionClient.analyzeImage(facesImageUrl, {visualFeatures: ['Faces']})).faces;

// Print the bounding box, gender, and age from the faces.
if (faces.length) {
  console.log(`${faces.length} face${faces.length == 1 ? '' : 's'} found:`);
  for (let face of faces) { console.log(`    Gender: ${face.gender}`.padEnd(20)
    + ` Age: ${face.age}`.padEnd(10) + `at ${formatRectFaces(face.faceRectangle)}`); }
} else { console.log('No faces found.');
```

Defina la función auxiliar `formatRectFaces`:

```
// Formats the bounding box
function formatRectFaces(rect) {
  return `top=${rect.top}`.padEnd(10) + `left=${rect.left}`.padEnd(10) + `bottom=${rect.top +
rect.height}`.padEnd(12)
  + `right=${rect.left + rect.width}`.padEnd(10) + `(${rect.width}x${rect.height})`;
}
```

Detección de contenido para adultos, explícito o sangriento

El siguiente código imprime la presencia detectada de contenido para adultos en la imagen. Para más información, consulte [Contenido para adultos, subido de tono y sangriento](#).

Defina la dirección URL de la imagen que se va a usar:

```
// The URL image and local images are not racy/adult.
// Try your own racy/adult images for a more effective result.
const adultURLImage = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/celebrities.jpg';
```

Después, agregue el código siguiente para detectar el contenido para adultos e imprima los resultados en la consola.

```
// Function to confirm racy or not
const isIt = flag => flag ? 'is' : 'isn't';

// Analyze URL image
console.log('Analyzing image for racy/adult content...', adultURLImage.split('/').pop());
var adult = (await computerVisionClient.analyzeImage(adultURLImage, {
  visualFeatures: ['Adult']
})).adult;
console.log(`This probably ${isIt(adult.isAdultContent)} adult content (${adult.adultScore.toFixed(4)} score)`);
console.log(`This probably ${isIt(adult.isRacyContent)} racy content (${adult.racyScore.toFixed(4)} score)`);
```

Obtención de la combinación de colores de imagen

El código siguiente imprime los atributos de color detectados en la imagen, como los colores dominantes y el color de énfasis. Consulte [Combinaciones de colores](#) para más detalles.

```
const colorURLImage = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/celebrities.jpg';

// Analyze URL image
console.log('Analyzing image for color scheme...', colorURLImage.split('/').pop());
console.log();
let color = (await computerVisionClient.analyzeImage(colorURLImage, {visualFeatures: ['Color']})).color;
printColorScheme(color);
```

Defina la función auxiliar `printColorScheme` para imprimir los detalles de la combinación de color en la consola.

```
// Print a detected color scheme
function printColorScheme(colors){
  console.log(`Image is in ${colors.isBwImg ? 'black and white' : 'color'}`);
  console.log(`Dominant colors: ${colors.dominantColors.join(', ')}`);
  console.log(`Dominant foreground color: ${colors.dominantColorForeground}`);
  console.log(`Dominant background color: ${colors.dominantColorBackground}`);
  console.log(`Suggested accent color: #${colors.accentColor}`);
}
```

Obtención de contenido específico del dominio

Computer Vision puede usar un modelo especializado para realizar análisis adicionales en las imágenes. Consulte [Contenido específico del dominio](#) para más detalles.

En primer lugar, defina la dirección URL de la imagen que se va a analizar:

```
const domainURLImage = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/landmark.jpg';
```

En el código siguiente se analizan los datos sobre los paisajes detectados en la imagen.

```
// Analyze URL image
console.log('Analyzing image for landmarks...', domainURLImage.split('/').pop());
let domain = (await computerVisionClient.analyzeImageByDomain('landmarks', domainURLImage)).result.landmarks;

// Prints domain-specific, recognized objects
if (domain.length) {
  console.log(`${domain.length} ${domain.length == 1 ? 'landmark' : 'landmarks'} found:`);
  for (let obj of domain) {
    console.log(`    ${obj.name}`.padEnd(20) + `( ${obj.confidence.toFixed(2)} confidence)` .padEnd(20) +
    `${formatRectDomain(obj.faceRectangle)}`);
  }
} else {
  console.log('No landmarks found.');
```

Defina la función auxiliar `formatRectDomain` para analizar los datos de ubicación de los puntos de referencia detectados.

```
// Formats bounding box
function formatRectDomain(rect) {
  if (!rect) return '';
  return `top=${rect.top}`.padEnd(10) + `left=${rect.left}`.padEnd(10) + `bottom=${rect.top +
  rect.height}`.padEnd(12) +
  `right=${rect.left + rect.width}`.padEnd(10) + `(${rect.width}x${rect.height})`;
}
```

Obtención del tipo de imagen

El código siguiente imprime información sobre el tipo de imagen (si es una imagen prediseñada o dibujo lineal).

```
const typeURLImage = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-python-sdk-samples/master/samples/vision/images/make_things_happen.jpg';

// Analyze URL image
console.log('Analyzing type in image...', typeURLImage.split('/').pop());
let types = (await computerVisionClient.analyzeImage(typeURLImage, {visualFeatures: ['ImageType']})).imageType;
console.log(`Image appears to be ${describeType(types)}`);
```

Defina la función auxiliar `describeType`:

```
function describeType(imageType) {
  if (imageType.clipArtType && imageType.clipArtType > imageType.lineDrawingType) return 'clip art';
  if (imageType.lineDrawingType && imageType.clipArtType < imageType.lineDrawingType) return 'a line drawing';
  return 'a photograph';
}
```

Lectura de texto manuscrito e impreso

Computer Vision puede leer texto visible de una imagen y convertirlo en un flujo de caracteres.

NOTE

También puede leer el texto de una imagen local. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes locales.

Configuración de imágenes de prueba

Guarde una referencia de la dirección URL de las imágenes de las que quiere extraer texto.

```
// URL images containing printed and handwritten text
const printedText = 'https://moderatorsampleimages.blob.core.windows.net/samples/sample2.jpg';
const handwrittenText = 'https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/master/ComputerVision/Images/handwritten_text.jpg';
```

Llamada a Recognize API

Agregue el código siguiente, que llama a la función `recognizeText` para las imágenes proporcionadas.

```
// Recognize text in printed image
console.log('Recognizing printed text...', printedText.split('/').pop());
var printed = await recognizeText(computerVisionClient, 'Printed', printedText);
printRecText(printed);

// Recognize text in handwritten image
console.log('\nRecognizing handwritten text...', handwrittenText.split('/').pop());
var handwriting = await recognizeText(computerVisionClient, 'Handwritten', handwrittenText);
printRecText(handwriting);
```

Defina la función `recognizeText`. Esta acción llama al método **recognizeText** en el objeto cliente, que devuelve un identificador de operación e inicia un proceso asíncronico para leer el contenido de la imagen. Luego, usa el identificador de operación para comprobar la operación a intervalos de un segundo hasta que se devuelven los resultados. Después, devuelve los resultados extraídos.


```
// Perform text recognition and await the result
async function recognizeText(client, mode, url) {
  // To recognize text in a local image, replace client.recognizeText() with recognizeTextInStream() as shown:
  // result = await client.recognizeTextInStream(mode, () => createReadStream(localImagePath));
  let result = await client.recognizeText(mode, url);
  // Operation ID is last path segment of operationLocation (a URL)
  let operation = result.operationLocation.split('/').slice(-1)[0];

  // Wait for text recognition to complete
  // result.status is initially undefined, since it's the result of recognizeText
  while (result.status !== 'Succeeded') { await sleep(1000); result = await
client.getTextOperationResult(operation); }
  return result.recognitionResult;
}
```

Seguidamente, defina la función auxiliar `printRecText`, que imprime los resultados de una operación de reconocimiento en la consola.

```
// Prints all text from OCR result
function printRecText(ocr) {
  if (ocr.lines.length) {
    console.log('Recognized text:');
    for (let line of ocr.lines) {
      console.log(line.words.map(w => w.text).join(' '));
    }
  }
  else { console.log('No recognized text.')} }
}
```

Ejecución de la aplicación

Ejecute la aplicación con el comando `node` en el archivo de inicio rápido.

```
node index.js
```

Limpieza de recursos

Si quiere limpiar y eliminar una suscripción a Cognitive Services, puede eliminar el recurso o grupo de recursos. Al eliminar el grupo de recursos, también se elimina cualquier otro recurso que esté asociado a él.

- [Portal](#)
- [CLI de Azure](#)

Pasos siguientes

[Referencia de la API Computer Vision \(Node.js\)](#)

- [¿Qué es Computer Vision?](#)
- El código fuente de este ejemplo está disponible en [GitHub](#).

Inicio rápido: Biblioteca cliente de Computer Vision para Go

14/01/2020 • 20 minutes to read • [Edit Online](#)

Introducción a la biblioteca cliente de Computer Vision para Go. Siga estos pasos para instalar el paquete y probar el código de ejemplo para realizar tareas básicas. Computer Vision proporciona acceso a algoritmos avanzados para procesar imágenes y devolver información.

Puede utilizar la biblioteca cliente de Computer Vision para Go para:

- Analizar una imagen para ver las etiquetas, la descripción de texto, las caras, el contenido para adultos, etc.
- Reconocer texto manuscrito e impreso con la API Read para Batch.

[Documentación de referencia](#) | [Código fuente de la biblioteca](#) | [Paquete](#)

Prerequisitos

- Una suscripción a Azure: [cree una cuenta gratuita](#)
- La versión más reciente de [Go](#).

Instalación

Creación de un recurso de Computer Vision en Azure

Los servicios de Azure Cognitive Services se representan por medio de recursos de Azure a los que se suscribe. Cree un recurso para Computer Vision con [Azure Portal](#) o la [CLI de Azure](#) en la máquina local. También puede:

- Obtener una [clave de prueba](#) válida durante siete días de forma gratuita. Después de registrarse, estará disponible en el [sitio web de Azure](#).
- Ver el recurso en [Azure Portal](#)

Después de obtener una clave de la suscripción de evaluación o el recurso, [cree una variable de entorno](#) para la clave y la dirección URL del punto de conexión, denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación de un directorio del proyecto de Go

En una ventana de consola (cmd, PowerShell, Terminal, Bash), cree una nueva área de trabajo para el proyecto de Go llamado `my-app` y vaya hasta ella.

```
mkdir -p my-app/{src, bin, pkg}
cd my-app
```

El área de trabajo contendrá tres carpetas:

- **src**: este directorio contendrá código fuente y paquetes. Todos los paquetes instalados con el comando `go get` se encontrarán en este directorio.
- **pkg**: este directorio contendrá objetos de paquete de Go compilados. Todos estos archivos tienen la extensión `.a`.
- **bin**: este directorio contendrá los archivos ejecutables binarios que se crean al ejecutar `go install`.

TIP

Para más información sobre la estructura de un área de trabajo de Go, consulte la [documentación del lenguaje Go](#). En esta guía se incluye información para establecer `$GOPATH` y `$GOROOT`.

Instalación de la biblioteca cliente para Go

Instale a continuación la biblioteca cliente para Go:

```
go get -u https://github.com/Azure/azure-sdk-for-go/tree/master/services/cognitiveservices/v2.1/computervision
```

O, si usa dep, dentro de su repositorio ejecute:

```
dep ensure -add https://github.com/Azure/azure-sdk-for-go/tree/master/services/cognitiveservices/v2.1/computervision
```

Creación de una aplicación de Go

A continuación, cree un archivo en el directorio **src** denominado `sample-app.go`:

```
cd src
touch sample-app.go
```

Abra `sample-app.go` en el entorno de desarrollo integrado o editor de texto que prefiera. A continuación, agregue el nombre del paquete e importe las siguientes bibliotecas:

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "github.com/Azure/azure-sdk-for-go/services/cognitiveservices/v2.0/computervision"
    "github.com/Azure/go-autorest/autorest"
    "io"
    "log"
    "os"
    "strings"
    "time"
)
```

Declare también un contexto en la raíz del script. Necesitará este objeto para ejecutar la mayoría de las llamadas a funciones de Computer Vision:

```
// Declare global so don't have to pass it to all of the tasks.
var computerVisionContext context.Context
```

A continuación, empezará a agregar código para llevar a cabo distintas operaciones de Computer Vision.

Modelo de objetos

Las siguientes clases e interfaces controlan algunas de las características principales del SDK de Go para Computer Vision.

NOMBRE	DESCRIPCIÓN
BaseClient	Esta clase es necesaria para todas las funciones de Computer Vision, como el análisis de imágenes y la lectura de texto. Cree una instancia de ella con la información de suscripción y úsela para realizar la mayoría de las operaciones con imágenes.
ImageAnalysis	Este tipo contiene los resultados de una llamada de función AnalyzeImage . Hay tipos similares para cada una de las funciones específicas de categoría.
ReadOperationResult	Este tipo contiene los resultados de una operación de lectura por lotes.
VisualFeatureTypes	Este tipo define las diferentes clases de análisis de imágenes que se pueden realizar en una operación de análisis estándar. Debe especificar un conjunto de valores de VisualFeatureTypes en función de sus necesidades.

Ejemplos de código

En estos fragmentos de código se muestra cómo realizar las siguientes tareas con la biblioteca cliente de Computer Vision para Go:

- [Autenticar el cliente](#)
- [Analizar una imagen](#)
- [Leer texto manuscrito e impreso](#)

Autenticar el cliente

NOTE

En este paso se da por supuesto que ha [creado variables de entorno](#) para la clave de Computer Vision y el punto de conexión, denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT` respectivamente.

Cree una función `main` y agréguele el código siguiente para crear una instancia de un cliente con su punto de conexión y clave.

```

/*
 * Configure the Computer Vision client
 * Set environment variables for COMPUTER_VISION_SUBSCRIPTION_KEY and COMPUTER_VISION_ENDPOINT,
 * then restart your command shell or your IDE for changes to take effect.
 */
computerVisionKey := os.Getenv("COMPUTER_VISION_SUBSCRIPTION_KEY")

if (computerVisionKey == "") {
    log.Fatal("\n\nPlease set a COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n" +
        "***You may need to restart your shell or IDE after it's set.**\n")
}

endpointURL := os.Getenv("COMPUTER_VISION_ENDPOINT")
if (endpointURL == "") {
    log.Fatal("\n\nPlease set a COMPUTER_VISION_ENDPOINT environment variable.\n" +
        "***You may need to restart your shell or IDE after it's set.**")
}

computerVisionClient := computervision.New(endpointURL);
computerVisionClient.Authorizer = autorest.NewCognitiveServicesAuthorizer(computerVisionKey)

computerVisionContext = context.Background()
/*
 * END - Configure the Computer Vision client
 */

```

Análisis de una imagen

El código siguiente utiliza el objeto de cliente para analizar una imagen remota e imprimir los resultados en la consola. Puede obtener una descripción de texto, categorización, lista de etiquetas, objetos detectados, marcas detectadas, caras detectadas, marcas de contenido para adultos, colores principales y tipo de imagen.

Configuración de una imagen de prueba

Guarde primero una referencia a la dirección URL de la imagen que desee analizar. Colóquela dentro de la función

```
main .
```

```

landmarkImageURL := "https://github.com/Azure-Samples/cognitive-services-sample-data-
files/raw/master/ComputerVision/Images/landmark.jpg"

```

NOTE

También puede analizar una imagen local. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes locales.

Especificación de características visuales

Las siguientes llamadas de función extraen diferentes características visuales de la imagen de ejemplo. Definirá estas funciones en las secciones siguientes.

```
// Analyze features of an image, remote
DescribeRemoteImage(computerVisionClient, landmarkImageURL)
CategorizeRemoteImage(computerVisionClient, landmarkImageURL)
TagRemoteImage(computerVisionClient, landmarkImageURL)
DetectFacesRemoteImage(computerVisionClient, facesImageURL)
DetectObjectsRemoteImage(computerVisionClient, objectsImageURL)
DetectBrandsRemoteImage(computerVisionClient, brandsImageURL)
DetectAdultOrRacyContentRemoteImage(computerVisionClient, adultRacyImageURL)
DetectColorSchemeRemoteImage(computerVisionClient, brandsImageURL)
DetectDomainSpecificContentRemoteImage(computerVisionClient, landmarkImageURL)
DetectImageTypesRemoteImage(computerVisionClient, detectTypeImageURL)
GenerateThumbnailRemoteImage(computerVisionClient, adultRacyImageURL)
```

Obtención de la descripción de la imagen

La siguiente función obtiene la lista de títulos generados para la imagen. Para más información acerca de la descripción de la imagen, consulte [Descripción de imágenes](#).

```
func DescribeRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DESCRIBE IMAGE - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    maxNumberDescriptionCandidates := new(int32)
    *maxNumberDescriptionCandidates = 1

    remoteImageDescription, err := client.DescribeImage(
        computerVisionContext,
        remoteImage,
        maxNumberDescriptionCandidates,
        "") // language
    if err != nil { log.Fatal(err) }

    fmt.Println("Captions from remote image: ")
    if len(*remoteImageDescription.Captions) == 0 {
        fmt.Println("No captions detected.")
    } else {
        for _, caption := range *remoteImageDescription.Captions {
            fmt.Printf("%v' with confidence %.2f%%\n", *caption.Text, *caption.Confidence * 100)
        }
    }
    fmt.Println()
}
```

Obtención de la categoría de imagen

La función siguiente obtiene la categoría detectada de la imagen. Para más información, consulte [Categorización de imágenes](#).

```

func CategorizeRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("CATEGORIZE IMAGE - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    features := []computervision.VisualFeatureTypes{computervision.VisualFeatureTypesCategories}
    imageAnalysis, err := client.AnalyzeImage(
        computerVisionContext,
        remoteImage,
        features,
        []computervision.Details{},
        "")
    if err != nil { log.Fatal(err) }

    fmt.Println("Categories from remote image: ")
    if len(*imageAnalysis.Categories) == 0 {
        fmt.Println("No categories detected.")
    } else {
        for _, category := range *imageAnalysis.Categories {
            fmt.Printf("%v' with confidence %.2f%%\n", *category.Name, *category.Score * 100)
        }
    }
    fmt.Println()
}

```

Obtención de etiquetas de imagen

La función siguiente obtiene el conjunto de las etiquetas detectadas en la imagen. Para más información, consulte [Etiquetas de contenido](#).

```

func TagRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("TAG IMAGE - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    remoteImageTags, err := client.TagImage(
        computerVisionContext,
        remoteImage,
        "")
    if err != nil { log.Fatal(err) }

    fmt.Println("Tags in the remote image: ")
    if len(*remoteImageTags.Tags) == 0 {
        fmt.Println("No tags detected.")
    } else {
        for _, tag := range *remoteImageTags.Tags {
            fmt.Printf("%v' with confidence %.2f%%\n", *tag.Name, *tag.Confidence * 100)
        }
    }
    fmt.Println()
}

```

Detección de objetos

La función siguiente detecta objetos comunes en la imagen y los imprime en la consola. Para más información, consulte [Detección de objetos](#).

```

func DetectObjectsRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT OBJECTS - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    imageAnalysis, err := client.DetectObjects(
        computerVisionContext,
        remoteImage,
    )
    if err != nil { log.Fatal(err) }

    fmt.Println("Detecting objects in remote image: ")
    if len(*imageAnalysis.Objects) == 0 {
        fmt.Println("No objects detected.")
    } else {
        // Print the objects found with confidence level and bounding box locations.
        for _, object := range *imageAnalysis.Objects {
            fmt.Printf("'%' with confidence %.2f%% at location (%v, %v), (%v, %v)\n",
                *object.Object, *object.Confidence * 100,
                *object.Rectangle.X, *object.Rectangle.X + *object.Rectangle.W,
                *object.Rectangle.Y, *object.Rectangle.Y + *object.Rectangle.H)
        }
    }
    fmt.Println()
}

```

Detección de marcas

El código siguiente detecta marcas corporativas y logotipos en la imagen y los imprime en la consola. Para más información, consulte [Detección de marcas](#).

En primer lugar, declare una referencia a una nueva imagen dentro de la función `main`.

```

brandsImageURL := "https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/images/gray-shirt-
logo.jpg"

```

El código siguiente define la función de detección de marca.


```

func DetectBrandsRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT BRANDS - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    // Define the kinds of features you want returned.
    features := []computervision.VisualFeatureTypes{computervision.VisualFeatureTypesBrands}

    imageAnalysis, err := client.AnalyzeImage(
        computerVisionContext,
        remoteImage,
        features,
        []computervision.Details{},
        "en")
    if err != nil { log.Fatal(err) }

    fmt.Println("Detecting brands in remote image: ")
    if len(*imageAnalysis.Brands) == 0 {
        fmt.Println("No brands detected.")
    } else {
        // Get bounding box around the brand and confidence level it's correctly identified.
        for _, brand := range *imageAnalysis.Brands {
            fmt.Printf("'%' with confidence %.2f%% at location (%v, %v), (%v, %v)\n",
                *brand.Name, *brand.Confidence * 100,
                *brand.Rectangle.X, *brand.Rectangle.X + *brand.Rectangle.W,
                *brand.Rectangle.Y, *brand.Rectangle.Y + *brand.Rectangle.H)
        }
    }
    fmt.Println()
}

```

DetECCIÓN DE CARAS

La función siguiente devuelve las caras detectadas en la imagen con sus coordenadas de rectángulo y selecciona determinados atributos de cara. Para más información, consulte [Detección de caras](#).

```

func DetectFacesRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT FACES - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    // Define the features you want returned with the API call.
    features := []computervision.VisualFeatureTypes{computervision.VisualFeatureTypesFaces}
    imageAnalysis, err := client.AnalyzeImage(
        computerVisionContext,
        remoteImage,
        features,
        []computervision.Details{},
        "")
    if err != nil { log.Fatal(err) }

    fmt.Println("Detecting faces in a remote image ...")
    if len(*imageAnalysis.Faces) == 0 {
        fmt.Println("No faces detected.")
    } else {
        // Print the bounding box locations of the found faces.
        for _, face := range *imageAnalysis.Faces {
            fmt.Printf("'%' of age %v at location (%v, %v), (%v, %v)\n",
                face.Gender, *face.Age,
                *face.FaceRectangle.Left, *face.FaceRectangle.Top,
                *face.FaceRectangle.Left + *face.FaceRectangle.Width,
                *face.FaceRectangle.Top + *face.FaceRectangle.Height)
        }
    }
    fmt.Println()
}

```

Detección de contenido para adultos, explícito o sangriento

La siguiente función imprime la presencia detectada de contenido para adultos en la imagen. Para más información, consulte [Contenido para adultos, subido de tono y sangriento](#).

```

func DetectAdultOrRacyContentRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT ADULT OR RACY CONTENT - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    // Define the features you want returned from the API call.
    features := []computervision.VisualFeatureTypes{computervision.VisualFeatureTypesAdult}
    imageAnalysis, err := client.AnalyzeImage(
        computerVisionContext,
        remoteImage,
        features,
        []computervision.Details{},
        "") // language, English is default
    if err != nil { log.Fatal(err) }

    // Print whether or not there is questionable content.
    // Confidence levels: low means content is OK, high means it's not.
    fmt.Println("Analyzing remote image for adult or racy content: ");
    fmt.Printf("Is adult content: %v with confidence %.2f%%\n", *imageAnalysis.Adult.IsAdultContent,
        *imageAnalysis.Adult.AdultScore * 100)
    fmt.Printf("Has racy content: %v with confidence %.2f%%\n", *imageAnalysis.Adult.IsRacyContent,
        *imageAnalysis.Adult.RacyScore * 100)
    fmt.Println()
}

```

Obtención de la combinación de colores de imagen

La función siguiente imprime los atributos de color detectados en la imagen, como los colores dominantes y el color de énfasis. Para más información, consulte [Combinaciones de colores](#).

```
func DetectColorSchemeRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT COLOR SCHEME - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    // Define the features you'd like returned with the result.
    features := []computervision.VisualFeatureTypes{computervision.VisualFeatureTypesColor}
    imageAnalysis, err := client.AnalyzeImage(
        computerVisionContext,
        remoteImage,
        features,
        []computervision.Details{},
        "") // language, English is default
    if err != nil { log.Fatal(err) }

    fmt.Println("Color scheme of the remote image: ");
    fmt.Printf("Is black and white: %v\n", *imageAnalysis.Color.IsBWImg)
    fmt.Printf("Accent color: 0x%x\n", *imageAnalysis.Color.AccentColor)
    fmt.Printf("Dominant background color: %v\n", *imageAnalysis.Color.DominantColorBackground)
    fmt.Printf("Dominant foreground color: %v\n", *imageAnalysis.Color.DominantColorForeground)
    fmt.Printf("Dominant colors: %v\n", strings.Join(*imageAnalysis.Color.DominantColors, ", "))
    fmt.Println()
}
```

Obtención de contenido específico del dominio

Computer Vision puede usar modelos especializados para realizar análisis adicionales en las imágenes. Para más información, consulte [Contenido específico del dominio](#).

En el código siguiente se analizan los datos sobre las celebridades detectadas en la imagen.

```

func DetectDomainSpecificContentRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT DOMAIN-SPECIFIC CONTENT - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    fmt.Println("Detecting domain-specific content in the local image ...")

    // Check if there are any celebrities in the image.
    celebrities, err := client.AnalyzeImageByDomain(
        computerVisionContext,
        "celebrities",
        remoteImage,
        "") // language, English is default
    if err != nil { log.Fatal(err) }

    fmt.Println("\nCelebrities: ")

    // Marshal the output from AnalyzeImageByDomain into JSON.
    data, err := json.MarshalIndent(celebrities.Result, "", "\t")

    // Define structs for which to unmarshal the JSON.
    type Celebrities struct {
        Name string `json:"name"`
    }

    type CelebrityResult struct {
        Celebrities []Celebrities `json:"celebrities"`
    }

    var celebrityResult CelebrityResult

    // Unmarshal the data.
    err = json.Unmarshal(data, &celebrityResult)
    if err != nil { log.Fatal(err) }

    // Check if any celebrities detected.
    if len(celebrityResult.Celebrities) == 0 {
        fmt.Println("No celebrities detected.")
    } else {
        for _, celebrity := range celebrityResult.Celebrities {
            fmt.Printf("name: %v\n", celebrity.Name)
        }
    }
}

```

En el código siguiente se analizan los datos sobre los paisajes detectados en la imagen.

```

fmt.Println("\nLandmarks: ")

// Check if there are any landmarks in the image.
landmarks, err := client.AnalyzeImageByDomain(
    computerVisionContext,
    "landmarks",
    remoteImage,
    "")
if err != nil { log.Fatal(err) }

// Marshal the output from AnalyzeImageByDomain into JSON.
data, err = json.MarshalIndent(landmarks.Result, "", "\t")

// Define structs for which to unmarshal the JSON.
type Landmarks struct {
    Name string `json:"name"`
}

type LandmarkResult struct {
    Landmarks []Landmarks `json:"landmarks"`
}

var landmarkResult LandmarkResult

// Unmarshal the data.
err = json.Unmarshal(data, &landmarkResult)
if err != nil { log.Fatal(err) }

// Check if any celebrities detected.
if len(landmarkResult.Landmarks) == 0 {
    fmt.Println("No landmarks detected.")
} else {
    for _, landmark := range landmarkResult.Landmarks {
        fmt.Printf("name: %v\n", landmark.Name)
    }
}
fmt.Println()
}

```

Obtención del tipo de imagen

La función siguiente imprime información sobre el tipo de imagen (si es una imagen prediseñada o dibujo lineal).

```

func DetectImageTypesRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("DETECT IMAGE TYPES - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    features := []computervision.VisualFeatureTypes{computervision.VisualFeatureTypesImageType}

    imageAnalysis, err := client.AnalyzeImage(
        computerVisionContext,
        remoteImage,
        features,
        []computervision.Details{},
        "")
    if err != nil { log.Fatal(err) }

    fmt.Println("Image type of remote image:")

    fmt.Println("\nClip art type: ")
    switch *imageAnalysis.ImageType.ClipArtType {
    case 0:
        fmt.Println("Image is not clip art.")
    case 1:
        fmt.Println("Image is ambiguously clip art.")
    case 2:
        fmt.Println("Image is normal clip art.")
    case 3:
        fmt.Println("Image is good clip art.")
    }

    fmt.Println("\nLine drawing type: ")
    if *imageAnalysis.ImageType.LineDrawingType == 1 {
        fmt.Println("Image is a line drawing.")
    } else {
        fmt.Println("Image is not a line drawing.")
    }
    fmt.Println()
}

```

Lectura de texto manuscrito e impreso

Computer Vision puede leer texto visible de una imagen y convertirlo en un flujo de caracteres. El código de esta sección define una función, `RecognizeTextReadAPIRemoteImage`, que utiliza el objeto de cliente para detectar y extraer texto impreso o manuscrito de la imagen.

Agregue la referencia de imagen de ejemplo y la llamada de función a la función `main`.

```

// Analyze text in an image, remote
BatchReadFileRemoteImage(computerVisionClient, printedImageURL)

```

NOTE

También puede extraer texto de una imagen local. Consulte el código de ejemplo en [GitHub](#) para ver los escenarios que implican imágenes locales.

Llamada a la API Read

Defina la nueva función para leer texto, `RecognizeTextReadAPIRemoteImage`. Agregue el código siguiente, que llama al método **BatchReadFile** para la imagen especificada. Este método devuelve un identificador de operación e inicia

un proceso asíncrono para leer el contenido de la imagen.

```
func BatchReadFileRemoteImage(client computervision.BaseClient, remoteImageURL string) {
    fmt.Println("-----")
    fmt.Println("BATCH READ FILE - remote")
    fmt.Println()
    var remoteImage computervision.ImageURL
    remoteImage.URL = &remoteImageURL

    // The response contains a field called "Operation-Location",
    // which is a URL with an ID that you'll use for GetReadOperationResult to access OCR results.
    textHeaders, err := client.BatchReadFile(computerVisionContext, remoteImage)
    if err != nil { log.Fatal(err) }

    // Use ExtractHeader from the autorest library to get the Operation-Location URL
    operationLocation := autorest.ExtractHeaderValue("Operation-Location", textHeaders.Response)

    numberOfCharsInOperationId := 36
    operationId := string(operationLocation[len(operationLocation)-numberOfCharsInOperationId :
len(operationLocation)])
}
```

Obtención de resultados de lectura

A continuación, obtenga el identificador de operación devuelto de la llamada a **BatchReadFile** y úselo con el método **GetReadOperationResult** para consultar los resultados de la operación en el servicio. El código siguiente comprueba la operación a intervalos de un segundo hasta que se devuelven los resultados. A continuación, imprime los datos de texto extraídos en la consola.

```
readOperationResult, err := client.GetReadOperationResult(computerVisionContext, operationId)
if err != nil { log.Fatal(err) }

// Wait for the operation to complete.
i := 0
maxRetries := 10

fmt.Println("Recognizing text in a remote image with the batch Read API ...")
for readOperationResult.Status != computervision.Failed &&
    readOperationResult.Status != computervision.Succeeded {
    if i >= maxRetries {
        break
    }
    i++

    fmt.Printf("Server status: %v, waiting %v seconds...\n", readOperationResult.Status, i)
    time.Sleep(1 * time.Second)

    readOperationResult, err = client.GetReadOperationResult(computerVisionContext, operationId)
    if err != nil { log.Fatal(err) }
}
```

Visualización de resultados de lectura

Agregue el código siguiente para analizar y mostrar los datos de texto recuperados y finalice la definición de la función.

```
// Display the results.
fmt.Println()
for _, recResult := range *(readOperationResult.RecognitionResults) {
    for _, line := range *recResult.Lines {
        fmt.Println(*line.Text)
    }
}
```

Ejecución de la aplicación

Ejecute la aplicación desde el directorio de la aplicación con el comando `go run`.

```
go run sample-app.go
```

Limpieza de recursos

Si quiere limpiar y eliminar una suscripción a Cognitive Services, puede eliminar el recurso o grupo de recursos. Al eliminar el grupo de recursos, también se elimina cualquier otro recurso que esté asociado a él.

- [Portal](#)
- [CLI de Azure](#)

Pasos siguientes

[Referencia de la API Computer Vision \(Go\)](#)

- [¿Qué es la API Computer Vision?](#)
- El código fuente de este ejemplo está disponible en [GitHub](#).

Inicio rápido: Análisis de imágenes remotas mediante la API REST Computer Vision y cURL

13/01/2020 • 3 minutes to read • [Edit Online](#)

En este inicio rápido, analizará una imagen almacenada de forma remota para extraer características visuales con la API REST de Computer Vision. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [cURL](#).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave.

Creación y ejecución del comando de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el comando siguiente en un editor de texto.
2. Realice los siguientes cambios en el comando donde sea necesario:
 - a. Reemplace el valor de `<subscriptionKey>` por la clave de suscripción.
 - b. Reemplace la primera parte de la dirección URL de la solicitud (`westcentralus`) por el texto de la dirección URL de su punto de conexión.

NOTE

Los nuevos recursos creados después del 1 de julio de 2019 usarán nombres de subdominio personalizados. Para más información y para obtener una lista completa de los puntos de conexión regionales, consulte [Nombres de subdominios personalizados para Cognitive Services](#).

- c. Si lo desea, cambie el parámetro de idioma de la dirección URL de solicitud (`language=en`) para usar otro idioma admitido.
 - d. Si lo desea, cambie la dirección URL de la imagen del cuerpo de la solicitud (`http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg`) por la dirección URL de una imagen diferente que desee analizar.
3. Abra una ventana de símbolo del sistema.
 4. Pegue el comando del editor de texto en la ventana del símbolo del sistema y después ejecute el comando.

```
curl -H "Ocp-Apim-Subscription-Key: <subscriptionKey>" -H "Content-Type: application/json"
"https://westcentralus.api.cognitive.microsoft.com/vision/v2.1/analyze?
visualFeatures=Categories,Description&details=Landmarks&language=en" -d "
{"url": "http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg"}
```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```
{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "requestId": "b6e33879-abb2-43a0-a96e-02cb5ae0b795",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}
```

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Análisis de imágenes remotas mediante la API REST Computer Vision con Go

13/01/2020 • 4 minutes to read • [Edit Online](#)

En este inicio rápido, analizará una imagen almacenada de forma remota para extraer características visuales con la API REST de Computer Vision. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Go](#) instalado.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `imageUrl` por la dirección URL de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con una extensión `.go`. Por ejemplo, `analyze-image.go`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, ejecute el comando `go build` para compilar el paquete desde el archivo. Por ejemplo, `go build analyze-image.go`.
6. En el símbolo del sistema, ejecute el paquete compilado. Por ejemplo, `analyze-image`.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Add your Computer Vision subscription key and endpoint to your environment variables.
    subscriptionKey := os.Getenv("COMPUTER_VISION_SUBSCRIPTION_KEY")
    if (subscriptionKey == "") {
        log.Fatal("\n\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n" +
            "***Restart your shell or IDE for changes to take effect.**\n")
    }

    endpoint := os.Getenv("COMPUTER_VISION_ENDPOINT")
    if ("" == endpoint) {
        log.Fatal("\n\nSet the COMPUTER_VISION_ENDPOINT environment variable.\n" +
```

```

    "**Restart your shell or IDE for changes to take effect.**")
}
const uriBase = endpoint + "vision/v2.1/analyze"
const imageUrl =
    "https://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg"

const params = "?visualFeatures=Description&details=Landmarks&language=en"
const uri = uriBase + params
const imageUrlEnc = "{\"url\":\"" + imageUrl + "\"}"

reader := strings.NewReader(imageUrlEnc)

// Create the HTTP client
client := &http.Client{
    Timeout: time.Second * 2,
}

// Create the POST request, passing the image URL in the request body
req, err := http.NewRequest("POST", uri, reader)
if err != nil {
    panic(err)
}

// Add request headers
req.Header.Add("Content-Type", "application/json")
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

// Send the request and retrieve the response
resp, err := client.Do(req)
if err != nil {
    panic(err)
}

defer resp.Body.Close()

// Read the response body
// Note, data is a byte array
data, err := ioutil.ReadAll(resp.Body)
if err != nil {
    panic(err)
}

// Parse the JSON data from the byte array
var f interface{}
json.Unmarshal(data, &f)

// Format and display the JSON result
jsonFormatted, _ := json.MarshalIndent(f, "", " ")
fmt.Println(string(jsonFormatted))
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```

{
  "categories": [
    {
      "detail": {
        "landmarks": []
      },
      "name": "outdoor_water",
      "score": 0.9921875
    }
  ],
  "description": {
    "captions": [
      {
        "confidence": 0.916458423253597,
        "text": "a large waterfall over a rocky cliff"
      }
    ]
  },
  "tags": [
    "nature",
    "water",
    "waterfall",
    "outdoor",
    "rock",
    "mountain",
    "rocky",
    "grass",
    "hill",
    "covered",
    "hillside",
    "standing",
    "side",
    "group",
    "walking",
    "white",
    "man",
    "large",
    "snow",
    "grazing",
    "forest",
    "slope",
    "herd",
    "river",
    "giraffe",
    "field"
  ]
},
  "metadata": {
    "format": "Jpeg",
    "height": 959,
    "width": 1280
  },
  "requestId": "a92f89ab-51f8-4735-a58d-507da2213fc2"
}

```

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Análisis de imágenes remotas mediante la Java y la API REST Computer Vision

13/01/2020 • 6 minutes to read • [Edit Online](#)

En este inicio rápido, va a analizar una imagen almacenada de forma remota para extraer características visuales mediante el uso de Java y la API REST de Computer Vision. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalados la plataforma de [Java™](#), y el [kit de desarrollo de edición estándar 7 u 8](#) (JDK 7 u 8).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Cree un nuevo proyecto de Java en su IDE o editor favorito. Si la opción está disponible, cree el proyecto de Java desde una plantilla de aplicación de línea de comandos.
2. Importe las bibliotecas siguientes en el proyecto de Java. Si usa Maven, se proporcionan las coordenadas de Maven para cada biblioteca.
 - [Cliente HTTP de Apache](#) (org.apache.httpcomponents:httpclient:4.5.5)
 - [Núcleo del cliente HTTP de Apache](#) (org.apache.httpcomponents:httpcore:4.4.9)
 - [Biblioteca JSON](#) (org.json:json:20180130)
3. Agregue las siguientes instrucciones `import` al archivo que contiene la clase pública `Main` para el proyecto.

```
import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;
```

4. Reemplace la clase pública `Main` por el código siguiente.
5. También puede reemplazar el valor de `imageToAnalyze` por la dirección URL de una imagen diferente que desee analizar.

```

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Add your Computer Vision subscription key and endpoint to your environment variables.
    // After setting, close and then re-open your command shell or project for the changes to take effect.
    String subscriptionKey = System.getenv("COMPUTER_VISION_SUBSCRIPTION_KEY");
    String endpoint = ("COMPUTER_VISION_ENDPOINT");

    private static final String uriBase = endpoint +
        "vision/v2.1/analyze";
    private static final String imageToAnalyze =
        "https://upload.wikimedia.org/wikipedia/commons/" +
        "1/12/Broadway_and_Times_Square_by_night.jpg";

    public static void main(String[] args) {
        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            URIBuilder builder = new URIBuilder(uriBase);

            // Request parameters. All of them are optional.
            builder.setParameter("visualFeatures", "Categories,Description,Color");
            builder.setParameter("language", "en");

            // Prepare the URI for the REST API method.
            URI uri = builder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body.
            StringEntity requestEntity =
                new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
            request.setEntity(requestEntity);

            // Call the REST API method and get the response entity.
            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null) {
                // Format and display the JSON response.
                String jsonString = EntityUtils.toString(entity);
                JSONObject json = new JSONObject(jsonString);
                System.out.println("REST Response:\n");
                System.out.println(json.toString(2));
            }
        } catch (Exception e) {
            // Display error message.
            System.out.println(e.getMessage());
        }
    }
}

```

Compilación y ejecución del programa

1. Guárdela y compile el proyecto de Java.
2. Si usa un IDE, ejecute `Main`.

Como alternativa, si está ejecutando el programa desde una ventana de línea de comandos, ejecute los siguientes comandos. Estos comandos suponen que las bibliotecas están en una carpeta denominada `libs` que se encuentra

en la misma carpeta que `Main.java`; si no, deberá reemplazar `libs` por la ruta de acceso a las bibliotecas.

1. Compile el archivo `Main.java`.

```
javac -cp ".;libs/*" Main.java
```

2. Ejecute el programa. La solicitud se envía a QnA Maker API para crear la base de conocimiento y, luego, se sondean los resultados cada 30 segundos. Cada respuesta se imprime en la ventana de línea de comandos.

```
java -cp ".;libs/*" Main
```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:

REST Response:

```
{
  "metadata": {
    "width": 1826,
    "format": "Jpeg",
    "height": 2436
  },
  "color": {
    "dominantColorForeground": "Brown",
    "isBwImg": false,
    "accentColor": "B74314",
    "dominantColorBackground": "Brown",
    "dominantColors": ["Brown"]
  },
  "requestId": "bbffe1a1-4fa3-4a6b-a4d5-a4964c58a811",
  "description": {
    "captions": [{
      "confidence": 0.8241405091548035,
      "text": "a group of people on a city street filled with traffic at night"
    }],
    "tags": [
      "outdoor",
      "building",
      "street",
      "city",
      "busy",
      "people",
      "filled",
      "traffic",
      "many",
      "table",
      "car",
      "group",
      "walking",
      "bunch",
      "crowded",
      "large",
      "night",
      "light",
      "standing",
      "man",
      "tall",
      "umbrella",
      "riding",
      "sign",
      "crowd"
    ]
  },
  "categories": [{
    "score": 0.625,
    "name": "outdoor_street"
  }]
}
```

Pasos siguientes

Explore una aplicación de Java Swing que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Computer Vision API Java Tutorial](#) (Tutorial de Computer Vision API para Java)

Inicio rápido: Análisis de imágenes remotas mediante la API REST y JavaScript en Computer Vision

13/01/2020 • 4 minutes to read • [Edit Online](#)

En este inicio rápido, analizará una imagen almacenada de forma remota para extraer características visuales con la API REST de Computer Vision. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor del atributo `value` para el control `inputImage` por la dirección URL de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con la extensión `.html`. Por ejemplo, `analyze-image.html`.
4. Abra una ventana del explorador.
5. En el explorador, arrastre y coloque el archivo en la ventana del explorador.
6. Cuando la página web se muestra en el explorador, seleccione el botón **Analizar imagen**.

```
<!DOCTYPE html>
<html>
<head>
  <title>Analyze Sample</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
    let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
    if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and endpoint.');
```

```

        "details": "",
        "language": "en",
    });

    // Display the image.
    var sourceImageUrl = document.getElementById("inputImage").value;
    document.querySelector("#sourceImage").src = sourceImageUrl;

    // Make the REST API call.
    $.ajax({
        url: uriBase + "?" + $.param(params),

        // Request headers.
        beforeSend: function(xhrObj){
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader(
                "Ocp-Apim-Subscription-Key", subscriptionKey);
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'",
    });

    .done(function(data) {
        // Show formatted JSON on webpage.
        $("#responseTextArea").val(JSON.stringify(data, null, 2));
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Display error message.
        var errorString = (errorThrown === "") ? "Error. " :
            errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" :
            jQuery.parseJSON(jqXHR.responseText).message;
        alert(errorString);
    });
    });
</script>

<h1>Analyze image:</h1>
Enter the URL to an image, then click the <strong>Analyze image</strong> button.
<br><br>
Image to analyze:
<input type="text" name="inputImage" id="inputImage"
    value="https://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg" />
<button onclick="processImage()">Analyze image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput"
            style="width:580px; height:400px;"></textarea>
    </div>
    <div id="imageDiv" style="width:420px; display:table-cell;">
        Source image:
        <br><br>
        <img id="sourceImage" width="400" />
    </div>
</div>
</body>
</html>

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del explorador, parecida a la del ejemplo siguiente:

```
{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "color": {
    "dominantColorForeground": "Grey",
    "dominantColorBackground": "Green",
    "dominantColors": [
      "Grey",
      "Green"
    ],
    "accentColor": "4D5E2F",
    "isBwImg": false
  },
  "requestId": "73ef10ce-a4ea-43c6-ae7-70325777e4b3",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}
```

Pasos siguientes

Explore una aplicación de JavaScript que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR); crear miniaturas con recorte inteligente; y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para JavaScript](#)

Inicio rápido: Análisis de imágenes remotas mediante la API REST Computer Vision con Node.js

13/01/2020 • 4 minutes to read • [Edit Online](#)

En este inicio rápido, va a analizar una imagen almacenada de forma remota para extraer características visuales mediante el uso de la API REST de Computer Vision con Node.js. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalado [Node.js](#) 4.x o una versión posterior.
- Debe tener [npm](#) instalado.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Instale el paquete `request` de npm.
 - a. Abra una ventana del símbolo del sistema como administrador.
 - b. Ejecute el siguiente comando:

```
npm install request
```
 - c. Una vez que el paquete se haya instalado correctamente, cierre la ventana del símbolo del sistema.
2. Copie el código siguiente en un editor de texto.
3. También puede reemplazar el valor de `imageUri` por la dirección URL de una imagen diferente que desee analizar.
4. Si lo desea, reemplace el valor del parámetro de la solicitud `language` por un idioma diferente.
5. Guarde el código como un archivo con una extensión `.js`. Por ejemplo, `analyze-image.js`.
6. Abra una ventana de símbolo del sistema.
7. En el símbolo del sistema, utilice el comando `node` para ejecutar el archivo. Por ejemplo,

```
node analyze-image.js
```

```

'use strict';

const request = require('request');

let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and endpoint.')}

var uriBase = endpoint + 'vision/v2.1/analyze';

const imageUrl =
  'https://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg';

// Request parameters.
const params = {
  'visualFeatures': 'Categories,Description,Color',
  'details': '',
  'language': 'en'
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + '"' + imageUrl + '"}',
  headers: {
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key' : subscriptionKey
  }
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
  let jsonResponse = JSON.stringify(JSON.parse(body), null, ' ');
  console.log('JSON Response\n');
  console.log(jsonResponse);
});

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. El ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:


```

{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "color": {
    "dominantColorForeground": "Grey",
    "dominantColorBackground": "Green",
    "dominantColors": [
      "Grey",
      "Green"
    ],
    "accentColor": "4D5E2F",
    "isBwImg": false
  },
  "requestId": "81b4e400-e3c1-41f1-9020-e6871ad9f0ed",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}

```

Limpieza de recursos

Cuando ya no sea necesario, elimine el archivo y después desinstale el paquete `request` de npm. Para desinstalar el paquete, realice los siguientes pasos:

1. Abra una ventana del símbolo del sistema como administrador.
2. Ejecute el siguiente comando:

```
npm uninstall request
```

3. Una vez que el paquete se haya desinstalado correctamente, cierre la ventana del símbolo del sistema.

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar personajes y sitios emblemáticos, crear una miniatura y extraer texto impreso y escrito a mano. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Análisis de imágenes remotas mediante la API REST Computer Vision y Python

13/01/2020 • 4 minutes to read • [Edit Online](#)

En este inicio rápido, analizará una imagen almacenada de forma remota para extraer características visuales con la API REST de Computer Vision. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Puede ejecutar esta guía de inicio rápido paso a paso mediante un cuaderno de Jupyter en [MyBinder](#). Para iniciar Binder, seleccione el botón siguiente:

launch binder

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Python](#) instalado si desea ejecutar el ejemplo localmente.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.
- Debe tener instalados los siguientes paquetes Python: Puede usar [pip](#) para instalar los paquetes de Python.
 - Solicitudes
 - [matplotlib](#)
 - [pillow](#)

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `image_url` por la dirección URL de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `analyze-image.py`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python analyze-image.py`.

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
import json
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

analyze_url = endpoint + "vision/v2.1/analyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/" + \
    "Broadway_and_Times_Square_by_night.jpg/450px-Broadway_and_Times_Square_by_night.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'visualFeatures': 'Categories,Description,Color'}
data = {'url': image_url}
response = requests.post(analyze_url, headers=headers,
                        params=params, json=data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The most
# relevant caption for the image is obtained from the 'description' property.
analysis = response.json()
print(json.dumps(response.json()))
image_caption = analysis["description"][0]["captions"][0]["text"].capitalize()

# Display the image and overlay it with the caption.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)
plt.show()

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```

{
  "categories": [
    {
      "name": "outdoor_",
      "score": 0.00390625,
      "detail": {
        "landmarks": []
      }
    },
    {
      "name": "outdoor_street",
      "score": 0.33984375,
      "detail": {
        "landmarks": []
      }
    }
  ]
}

```

```

    },
    ],
    "description": {
      "tags": [
        "building",
        "outdoor",
        "street",
        "city",
        "people",
        "busy",
        "table",
        "walking",
        "traffic",
        "filled",
        "large",
        "many",
        "group",
        "night",
        "light",
        "crowded",
        "bunch",
        "standing",
        "man",
        "sign",
        "crowd",
        "umbrella",
        "riding",
        "tall",
        "woman",
        "bus"
      ],
    },
    "captions": [
      {
        "text": "a group of people on a city street at night",
        "confidence": 0.9122243847383961
      }
    ]
  },
  "color": {
    "dominantColorForeground": "Brown",
    "dominantColorBackground": "Brown",
    "dominantColors": [
      "Brown"
    ],
    "accentColor": "B54316",
    "isBwImg": false
  },
  "requestId": "c11894eb-de3e-451b-9257-7c8b168073d1",
  "metadata": {
    "height": 600,
    "width": 450,
    "format": "Jpeg"
  }
}

```

Pasos siguientes

Explore una aplicación de Python que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para Python](#)

Inicio rápido: Análisis de una imagen local mediante la API REST Computer Vision y C#

16/01/2020 • 5 minutes to read • [Edit Online](#)

En este inicio rápido, analizará una imagen almacenada localmente para extraer características visuales con la API REST de Computer Vision. Con el método [Analyze Image](#), puede extraer información de las características visuales a partir del contenido de una imagen.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Prerequisites

- Debe tener [Visual Studio 2015](#) o posterior.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear el ejemplo en Visual Studio, siga estos pasos:

1. Cree una solución de Visual Studio en Visual Studio, mediante la plantilla de aplicación de consola de Visual C# (.NET Framework).
2. Instale el paquete NuGet Newtonsoft.Json.
 - a. En el menú, haga clic en **Herramientas**, seleccione **Administrador de paquetes NuGet** y, luego, **Manage NuGet Packages for Solution** (Administrar paquetes NuGet para la solución).
 - b. Haga clic en la pestaña **Examinar** y, en el cuadro **Buscar**, escriba "Newtonsoft.Json".
 - c. Seleccione **Newtonsoft.Json** cuando se muestre, marque la casilla junto al nombre del proyecto y haga clic en **Instalar**.
3. Ejecute el programa.
4. En el símbolo del sistema, escriba la ruta de acceso a una imagen local.

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Add your Computer Vision subscription key and endpoint to your environment variables.
        static string subscriptionKey =
            Environment.GetEnvironmentVariable("COMPUTER_VISION_SUBSCRIPTION_KEY");

        static string endpoint = Environment.GetEnvironmentVariable("COMPUTER_VISION_ENDPOINT");

        // the Analyze method endpoint
```

```

static string uriBase = endpoint + "vision/v2.1/analyze";

static async Task Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Analyze an image:");
    Console.Write(
        "Enter the path to the image you wish to analyze: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Call the REST API method.
        Console.WriteLine("\nWait for the results to appear.\n");
        await MakeAnalysisRequest(imageFilePath);
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets the analysis of the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file to analyze.</param>
static async Task MakeAnalysisRequest(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameters. A third optional parameter is "details".
        // The Analyze Image method returns information about the following
        // visual features:
        // Categories: categorizes image content according to a
        //                 taxonomy defined in documentation.
        // Description: describes the image content with a complete
        //                 sentence in supported languages.
        // Color:       determines the accent color, dominant color,
        //                 and whether an image is black & white.
        string requestParameters =
            "visualFeatures=Categories,Description,Color";

        // Assemble the URI for the REST API method.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Read the contents of the specified local image
        // into a byte array.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        // Add the byte array as an octet stream to the request body.
        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses the "application/octet-stream" content type.
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");
        }
    }
}

```

```

        // Asynchronously call the REST API method.
        response = await client.PostAsync(uri, content);
    }

    // Asynchronously get the JSON response.
    string contentString = await response.Content.ReadAsStringAsync();

    // Display the JSON response.
    Console.WriteLine("\nResponse:\n\n{0}\n",
        JToken.Parse(contentString).ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("\n" + e.Message);
    }
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    // Open a read-only file stream for the specified file.
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        // Read the file's contents into a byte array.
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:


```
{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "others_",
      "score": 0.0234375
    },
    {
      "name": "outdoor_",
      "score": 0.00390625
    }
  ],
  "description": {
    "tags": [
      "road",
      "building",
      "outdoor",
      "street",
      "night",
      "black",
      "city",
      "white",
      "light",
      "sitting",
      "riding",
      "man",
      "side",
      "empty",
      "rain",
      "corner",
      "traffic",
      "lit",
      "hydrant",
      "stop",
      "board",
      "parked",
      "bus",
      "tall"
    ],
    "captions": [
      {
        "text": "a close up of an empty city street at night",
        "confidence": 0.7965622853462756
      }
    ]
  },
  "requestId": "ddd1ac9-7e66-4c47-bdef-222f3fe5aa23",
  "metadata": {
    "width": 3733,
    "height": 1986,
    "format": "Jpeg"
  },
  "color": {
    "dominantColorForeground": "Black",
    "dominantColorBackground": "Black",
    "dominantColors": [
      "Black",
      "Grey"
    ],
    "accentColor": "666666",
    "isBImg": true
  }
}
```

Pasos siguientes

Explore una aplicación básica de Windows que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen.

[Tutorial de C# de Computer Vision API](#)

Inicio rápido: Análisis de una imagen local mediante la API REST Computer Vision y Python

13/01/2020 • 4 minutes to read • [Edit Online](#)

En este inicio rápido, analizará una imagen almacenada localmente para extraer características visuales con la API REST de Computer Vision. Con el [método de análisis de imagen](#), puede extraer características visuales basadas en el contenido de una imagen.

Puede ejecutar esta guía de inicio rápido paso a paso mediante un cuaderno de Jupyter en [MyBinder](#). Para iniciar Binder, seleccione el botón siguiente:

launch binder

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Python](#) instalado si desea ejecutar el ejemplo localmente.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.
- Debe tener instalados los siguientes paquetes Python: Puede usar [pip](#) para instalar los paquetes de Python.
 - Solicitudes
 - [matplotlib](#)
 - [pillow](#)

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `image_path` por la ruta de acceso y el nombre de archivo de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `analyze-local-image.py`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python analyze-local-image.py`.

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

analyze_url = endpoint + "vision/v2.1/analyze"

# Set image_path to the local path of an image that you want to analyze.
image_path = "C:/Documents/ImageToAnalyze.jpg"

# Read the image into a byte array
image_data = open(image_path, "rb").read()
headers = {'Ocp-Apim-Subscription-Key': subscription_key,
           'Content-Type': 'application/octet-stream'}
params = {'visualFeatures': 'Categories,Description,Color'}
response = requests.post(
    analyze_url, headers=headers, params=params, data=image_data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The most
# relevant caption for the image is obtained from the 'description' property.
analysis = response.json()
print(analysis)
image_caption = analysis["description"]["captions"][0]["text"].capitalize()

# Display the image and overlay it with the caption.
image = Image.open(BytesIO(image_data))
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```

{
  "categories": [
    {
      "name": "outdoor_",
      "score": 0.00390625,
      "detail": {
        "landmarks": []
      }
    },
    {
      "name": "outdoor_street",
      "score": 0.33984375,
      "detail": {
        "landmarks": []
      }
    }
  ]
}

```

```

    ],
    "description": {
      "tags": [
        "building",
        "outdoor",
        "street",
        "city",
        "people",
        "busy",
        "table",
        "walking",
        "traffic",
        "filled",
        "large",
        "many",
        "group",
        "night",
        "light",
        "crowded",
        "bunch",
        "standing",
        "man",
        "sign",
        "crowd",
        "umbrella",
        "riding",
        "tall",
        "woman",
        "bus"
      ],
      "captions": [
        {
          "text": "a group of people on a city street at night",
          "confidence": 0.9122243847383961
        }
      ]
    },
    "color": {
      "dominantColorForeground": "Brown",
      "dominantColorBackground": "Brown",
      "dominantColors": [
        "Brown"
      ],
      "accentColor": "B54316",
      "isBwImg": false
    },
    "requestId": "c11894eb-de3e-451b-9257-7c8b168073d1",
    "metadata": {
      "height": 600,
      "width": 450,
      "format": "Jpeg"
    }
  }
}

```

Pasos siguientes

Explore una aplicación de Python que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para Python](#)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision y C#

16/01/2020 • 6 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Con el método [Obtener miniatura](#), puede generar una miniatura de una imagen. Debe especificar el alto y el ancho, que pueden ser diferentes a la relación de aspecto de la imagen de entrada. Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte en función de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Prerequisitos

- Debe tener [Visual Studio 2015](#) o posterior.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear el ejemplo en Visual Studio, siga estos pasos:

1. Cree una solución de Visual Studio en Visual Studio, mediante la plantilla de aplicación de consola de Visual C#.
2. Instale el paquete NuGet Newtonsoft.Json.
 - a. En el menú, haga clic en **Herramientas**, seleccione **Administrador de paquetes NuGet** y, luego, **Manage NuGet Packages for Solution** (Administrar paquetes NuGet para la solución).
 - b. Haga clic en la pestaña **Examinar** y, en el cuadro **Buscar**, escriba "Newtonsoft.Json".
 - c. Seleccione **Newtonsoft.Json** cuando se muestre, marque la casilla junto al nombre del proyecto y haga clic en **Instalar**.
3. Ejecute el programa.
4. En el símbolo del sistema, escriba la ruta de acceso a una imagen local.

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Add your Computer Vision subscription key and endpoint to your environment variables.
        static string subscriptionKey = Environment.GetEnvironmentVariable("COMPUTER_VISION_SUBSCRIPTION_KEY");

        static string endpoint = Environment.GetEnvironmentVariable("COMPUTER_VISION_ENDPOINT");

        // the GenerateThumbnail method endpoint
```

```

static string uriBase = endpoint + "vision/v2.1/generateThumbnail";

static async Task Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Thumbnail:");
    Console.Write(
        "Enter the path to the image you wish to use to create a thumbnail image: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Call the REST API method.
        Console.WriteLine("\nWait a moment for the results to appear.\n");
        await MakeThumbNailRequest(imageFilePath);
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets a thumbnail image from the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file to use to create the thumbnail image.</param>
static async Task MakeThumbNailRequest(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameters.
        // The width and height parameters specify a thumbnail that's
        // 200 pixels wide and 150 pixels high.
        // The smartCropping parameter is set to true, to enable smart cropping.
        string requestParameters = "width=200&height=150&smartCropping=true";

        // Assemble the URI for the REST API method.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Read the contents of the specified local image
        // into a byte array.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        // Add the byte array as an octet stream to the request body.
        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses the "application/octet-stream" content type.
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");

            // Asynchronously call the REST API method.
            response = await client.PostAsync(uri, content);
        }

        // Check the HTTP status code of the response. If successful, display
        // display the response and save the thumbnail.
    }
}

```

```

        if (response.IsSuccessStatusCode)
        {
            // Display the response data.
            Console.WriteLine("\nResponse:\n{0}", response);

            // Get the image data for the thumbnail from the response.
            byte[] thumbnailImageData =
                await response.Content.ReadAsByteArrayAsync();

            // Save the thumbnail to the same folder as the original image,
            // using the original name with the suffix "_thumb".
            // Note: This will overwrite an existing file of the same name.
            string thumbnailFilePath =
                imagePath.Insert(imageFilePath.Length - 4, "_thumb");
            File.WriteAllBytes(thumbnailFilePath, thumbnailImageData);
            Console.WriteLine("\nThumbnail written to: {0}", thumbnailFilePath);
        }
        else
        {
            // Display the JSON error data.
            string errorString = await response.Content.ReadAsStringAsync();
            Console.WriteLine("\n\nResponse:\n{0}\n",
                JToken.Parse(errorString).ToString());
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("\n" + e.Message);
    }
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imagePath)
{
    // Open a read-only file stream for the specified file.
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        // Read the file's contents into a byte array.
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

Examen de la respuesta

Se devuelve una respuesta correcta como datos binarios, que representan los datos de imagen para la miniatura. Si la solicitud se realiza correctamente, la miniatura se guarda en la misma carpeta que la imagen local, mediante el nombre original con el sufijo "_thumb". Si se produce un error en la solicitud, la respuesta contiene un código de error y un mensaje para ayudar a determinar qué salió mal.

La aplicación de ejemplo muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:

Response:

```
Status-Code: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.StreamContent, Headers:
{
  Pragma: no-cache
  apim-request-id: 131eb5b4-5807-466d-9656-4c1ef0a64c9b
  Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
  x-content-type-options: nosniff
  Cache-Control: no-cache
  Date: Tue, 06 Jun 2017 20:54:07 GMT
  X-AspNet-Version: 4.0.30319
  X-Powered-By: ASP.NET
  Content-Length: 5800
  Content-Type: image/jpeg
  Expires: -1
}
```

Pasos siguientes

Explore una aplicación básica de Windows que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con las versiones de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para C#](#)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision y cURL

13/01/2020 • 5 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Debe especificar el alto y el ancho deseados, con una relación de aspecto que puede ser distinta a la de la imagen de entrada. Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte alrededor de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [cURL](#).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave.

Solicitud Get Thumbnail

Con el método [Get Thumbnail](#), puede generar una miniatura de una imagen.

Para ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor.
2. Reemplace `<Subscription Key>` por una clave de suscripción válida.
3. Reemplace `<File>` por la ruta de acceso y el nombre de archivo para guardar la miniatura.
4. Si es necesario, cambie la dirección URL de solicitud (`https://westcentralus.api.cognitive.microsoft.com/vision/v2.1`) para usar la ubicación donde obtuvo las claves de suscripción.
5. Además, puede cambiar la imagen (`{\"url\":\"...\"}`) que se va a analizar.
6. Abra una ventana de comandos en un equipo que tenga instalado cURL.
7. Pegue el código en la ventana y ejecute el comando.

NOTE

Debe utilizar la misma ubicación en la llamada de REST que utilizó para obtener las claves de la suscripción. Por ejemplo, si obtuvo sus claves de suscripción de westus, reemplace "westcentralus" en la siguiente URL por "westus".

Creación y ejecución del comando de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el comando siguiente en un editor de texto.
2. Realice los siguientes cambios en el comando donde sea necesario:
 - a. Reemplace el valor de `<subscriptionKey>` por la clave de suscripción.
 - b. Reemplace el valor de `<thumbnailFile>` con la ruta de acceso y el nombre del archivo donde se va a

guardar la miniatura.

- c. Reemplace la primera parte de la dirección URL de la solicitud (`westcentralus`) por el texto de la dirección URL de su punto de conexión.

NOTE

Los nuevos recursos creados después del 1 de julio de 2019 usarán nombres de subdominio personalizados. Para más información y para obtener una lista completa de los puntos de conexión regionales, consulte [Nombres de subdominios personalizados para Cognitive Services](#).

- d. Si lo desea, cambie la dirección URL de la imagen del cuerpo de la solicitud (

```
https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Shorkie_Poo_Puppy.jpg/1280px-Shorkie_Poo_Puppy.jpg\
```

) por la dirección URL de una imagen diferente de la que desea generar una miniatura.

3. Abra una ventana de símbolo del sistema.

4. Pegue el comando del editor de texto en la ventana del símbolo del sistema y después ejecute el comando.

```
curl -H "Ocp-Apim-Subscription-Key: <subscriptionKey>" -o <thumbnailFile> -H "Content-Type: application/json" "https://westcentralus.api.cognitive.microsoft.com/vision/v2.1/generateThumbnail?width=100&height=100&smartCropping=true" -d "{\n  \"url\": \"https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Shorkie_Poo_Puppy.jpg/1280px-Shorkie_Poo_Puppy.jpg\"\n}"
```

Examen de la respuesta

Una respuesta correcta escribe la imagen en miniatura en el archivo especificado en `<thumbnailFile>`. Si se produce un error en la solicitud, la respuesta contiene un código de error y un mensaje para ayudar a determinar qué salió mal. Si la solicitud parece realizarse correctamente pero la miniatura creada no es un archivo de imagen válido, es posible que su clave de suscripción no sea válida.

Pasos siguientes

Explore las versiones de Computer Vision API para ver cómo analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision con Go

13/01/2020 • 4 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Debe especificar el alto y el ancho, que pueden ser diferentes a la relación de aspecto de la imagen de entrada. Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte en función de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Go](#) instalado.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `imageUrl` por la dirección URL de una imagen diferente de la que desea generar una miniatura.
3. Guarde el código como un archivo con una extensión `.go`. Por ejemplo, `get-thumbnail.go`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, ejecute el comando `go build` para compilar el paquete desde el archivo. Por ejemplo, `go build get-thumbnail.go`.
6. En el símbolo del sistema, ejecute el paquete compilado. Por ejemplo, `get-thumbnail`.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Add your Computer Vision subscription key and endpoint to your environment variables.
    subscriptionKey := os.Getenv("COMPUTER_VISION_SUBSCRIPTION_KEY")
    if (subscriptionKey == "") {
        log.Fatal("\n\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n" +
            "**Restart your shell or IDE for changes to take effect.**\n")
    }

    endpoint := os.Getenv("COMPUTER_VISION_ENDPOINT")
    if (" " == endpoint) {

```

```

11 \ -- endpoint) {
    log.Fatal("\n\nSet the COMPUTER_VISION_ENDPOINT environment variable.\n" +
        "**Restart your shell or IDE for changes to take effect.**")
}
const uriBase = endpoint + "vision/v2.1/generateThumbnail"
const imageUrl =
    "https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg"

const params = "?width=100&height=100&smartCropping=true"
const uri = uriBase + params
const imageUrlEnc = "{\"url\":\"" + imageUrl + "\"}"

reader := strings.NewReader(imageUrlEnc)

// Create the HTTP client
client := &http.Client{
    Timeout: time.Second * 2,
}

// Create the POST request, passing the image URL in the request body
req, err := http.NewRequest("POST", uri, reader)
if err != nil {
    panic(err)
}

// Add headers
req.Header.Add("Content-Type", "application/json")
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

// Send the request and retrieve the response
resp, err := client.Do(req)
if err != nil {
    panic(err)
}

defer resp.Body.Close()

// Read the response body.
// Note, data is a byte array
data, err := ioutil.ReadAll(resp.Body)
if err != nil {
    panic(err)
}

// Parse the JSON data
var f interface{}
json.Unmarshal(data, &f)

// Format and display the JSON result
jsonFormatted, _ := json.MarshalIndent(f, "", " ")
fmt.Println(string(jsonFormatted))
}

```

Examen de la respuesta

Una respuesta correcta contiene los datos binarios de la imagen en miniatura. Si se produce un error en la solicitud, la respuesta contiene un código de error y un mensaje para ayudar a determinar qué salió mal.

Pasos siguientes

Explore Computer Vision API para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision y Java

13/01/2020 • 6 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Debe especificar el alto y el ancho, que pueden ser diferentes a la relación de aspecto de la imagen de entrada. Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte en función de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalados la plataforma de [Java™](#), y el [kit de desarrollo de edición estándar 7 u 8](#) (JDK 7 u 8).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Cree un nuevo proyecto de Java en su IDE o editor favorito. Si la opción está disponible, cree el proyecto de Java desde una plantilla de aplicación de línea de comandos.
2. Importe las bibliotecas siguientes en el proyecto de Java. Si usa Maven, se proporcionan las coordenadas de Maven para cada biblioteca.
 - [Cliente HTTP de Apache](#) (org.apache.httpcomponents:httpclient:4.5.5)
 - [Núcleo del cliente HTTP de Apache](#) (org.apache.httpcomponents:httpcore:4.4.9)
 - [Biblioteca JSON](#) (org.json:json:20180130)
3. Agregue las siguientes instrucciones `import` al archivo que contiene la clase pública `Main` para el proyecto.

```
import java.awt.*;
import javax.swing.*;
import java.net.URI;
import java.io.InputStream;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;
```

4. Reemplace la clase pública `Main` por el código siguiente.

5. También puede reemplazar el valor de `imageToAnalyze` por la dirección URL de una imagen diferente para la que desea generar una miniatura.
6. Guárdela y compile el proyecto de Java.
7. Si usa un IDE, ejecute `Main`. En caso contrario, abra una ventana del símbolo del sistema y, a continuación, utilice el comando `java` para ejecutar la clase compilada. Por ejemplo, `java Main`.

```
// This sample uses the following libraries:
// - Apache HTTP client (org.apache.httpcomponents:httpclient:4.5.5)
// - Apache HTTP core (org.apache.httpcomponents:httpcore:4.4.9)
// - JSON library (org.json:json:20180130).

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Add your Computer Vision subscription key and endpoint to your environment variables.
    // After setting, close and then re-open your command shell or project for the changes to take effect.
    String subscriptionKey = System.getenv("COMPUTER_VISION_SUBSCRIPTION_KEY");
    String endpoint = ("COMPUTER_VISION_ENDPOINT");

    private static final String uriBase = endpoint +
        "vision/v2.1/generateThumbnail";

    private static final String imageToAnalyze =
        "https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg";

    public static void main(String[] args) {
        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            URIBuilder uriBuilder = new URIBuilder(uriBase);

            // Request parameters.
            uriBuilder.setParameter("width", "100");
            uriBuilder.setParameter("height", "150");
            uriBuilder.setParameter("smartCropping", "true");

            // Prepare the URI for the REST API method.
            URI uri = uriBuilder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body.
            StringEntity requestEntity =
                new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
            request.setEntity(requestEntity);

            // Call the REST API method and get the response entity.
            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            // Check for success.
            if (response.getStatusLine().getStatusCode() == 200) {
                // Display the thumbnail.
                System.out.println("\nDisplaying thumbnail.\n");
                displayImage(entity.getContent());
            } else {
                // Format and display the JSON error message.
                String jsonString = EntityUtils.toString(entity);
                JSONObject json = new JSONObject(jsonString);
```



```

        System.out.println("Error:\n");
        System.out.println(json.toString(2));
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}

// Displays the given input stream as an image.
private static void displayImage(InputStream inputStream) {
    try {
        BufferedImage bufferedImage = ImageIO.read(inputStream);

        ImageIcon imageIcon = new ImageIcon(bufferedImage);

        JLabel jLabel = new JLabel();
        jLabel.setIcon(imageIcon);

        JFrame jFrame = new JFrame();
        jFrame.setLayout(new FlowLayout());
        jFrame.setSize(100, 150);

        jFrame.add(jLabel);
        jFrame.setVisible(true);
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
}
}

```

Examen de la respuesta

Se devuelve una respuesta correcta como datos binarios, que representan los datos de imagen para la miniatura. Si la solicitud se realiza correctamente, la miniatura se genera a partir de los datos binarios de la respuesta y se muestra en una ventana independiente creada por la aplicación de ejemplo. Si se produce un error en la solicitud, la respuesta se muestra en la ventana de consola. La respuesta de la solicitud con error contiene un código de error y un mensaje para ayudar a determinar qué salió mal.

Pasos siguientes

Explore una aplicación de Java Swing que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Computer Vision API Java Tutorial](#) (Tutorial de Computer Vision API para Java)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision y JavaScript

13/01/2020 • 5 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Debe especificar el alto y el ancho, que pueden ser diferentes a la relación de aspecto de la imagen de entrada. Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte en función de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor del atributo `value` para el control `inputImage` por la dirección URL de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con la extensión `.html`. Por ejemplo, `get-thumbnail.html`.
4. Abra una ventana del explorador.
5. En el explorador, arrastre y coloque el archivo en la ventana del explorador.
6. Cuando la página web se muestra en el explorador, seleccione el botón **Generar miniatura**.

```
<!DOCTYPE html>
<html>
<head>
  <title>Thumbnail Sample</title>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
    let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
    if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and endpoint.'); }

    var uriBase = endpoint + "vision/v2.1/generateThumbnail";

    // Request parameters.
    var params = "?width=100&height=150&smartCropping=true";
```

```

// Display the source image.
var sourceImageUrl = document.getElementById("inputImage").value;
document.querySelector("#sourceImage").src = sourceImageUrl;

// Prepare the REST API call:

// Create the HTTP Request object.
var xhr = new XMLHttpRequest();

// Identify the request as a POST, with the URL and parameters.
xhr.open("POST", uriBase + params);

// Add the request headers.
xhr.setRequestHeader("Content-Type","application/json");
xhr.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

// Set the response type to "blob" for the thumbnail image data.
xhr.responseType = "blob";

// Process the result of the REST API call.
xhr.onreadystatechange = function(e) {
    if(xhr.readyState === XMLHttpRequest.DONE) {

        // Thumbnail successfully created.
        if (xhr.status === 200) {
            // Show response headers.
            var s = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);
            document.getElementById("responseTextArea").value =
                JSON.stringify(xhr.getAllResponseHeaders(), null, 2);

            // Show thumbnail image.
            var urlCreator = window.URL || window.webkitURL;
            var imageUrl = urlCreator.createObjectURL(this.response);
            document.querySelector("#thumbnailImage").src = imageUrl;
        } else {
            // Display the error message. The error message is the response
            // body as a JSON string. The code in this code block extracts
            // the JSON string from the blob response.
            var reader = new FileReader();

            // This event fires after the blob has been read.
            reader.addEventListener('loadend', (e) => {
                document.getElementById("responseTextArea").value =
                    JSON.stringify(JSON.parse(e.srcElement.result), null, 2);
            });

            // Start reading the blob as text.
            reader.readAsText(xhr.response);
        }
    }
};

// Make the REST API call.
xhr.send('{"url": ' + '"' + sourceImageUrl + '"}');
};
</script>

```

<h1>Generate thumbnail image:</h1>

Enter the URL to an image to use in creating a thumbnail image,
then click the Generate thumbnail button.

Image for thumbnail:

<input type="text" name="inputImage" id="inputImage"

value="https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Shorkie_Poo_Puppy.jpg/1280px-Shorkie_Poo_Puppy.jpg" />

<button onclick="processImage()">Generate thumbnail</button>

<div id="wrapper" style="width:1160px; display:table;">

```
<div id="jsonOutput" style="width:600px; display:table-cell;">
  Response:
  <br><br>
  <textarea id="responseTextArea" class="UIInput"
    style="width:580px; height:400px;"></textarea>
</div>
<div id="imageDiv" style="width:420px; display:table-cell;">
  Source image:
  <br><br>
  <img id="sourceImage" width="400" />
</div>
<div id="thumbnailDiv" style="width:140px; display:table-cell;">
  Thumbnail:
  <br><br>
  <img id="thumbnailImage" />
</div>
</div>
</body>
</html>
```

Examen de la respuesta

Se devuelve una respuesta correcta como datos binarios, que representan los datos de imagen para la miniatura. Si la solicitud se realiza correctamente, la miniatura se genera a partir de los datos binarios de la respuesta y se muestra en la ventana del explorador. Si se produce un error en la solicitud, la respuesta se muestra en la ventana de consola. La respuesta de la solicitud con error contiene un código de error y un mensaje para ayudar a determinar qué salió mal.

Pasos siguientes

Explore una aplicación de JavaScript que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR); crear miniaturas con recorte inteligente; y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para JavaScript](#)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision y Node.js

13/01/2020 • 4 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Con el método [Obtener miniatura](#), puede generar una miniatura de una imagen. Debe especificar el alto y el ancho, que pueden ser diferentes a la relación de aspecto de la imagen de entrada. Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte en función de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalado [Node.js](#) 4.x o una versión posterior.
- Debe tener [npm](#) instalado.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Instale el paquete `request` de npm.
 - a. Abra una ventana del símbolo del sistema como administrador.
 - b. Ejecute el siguiente comando:

```
npm install request
```
 - c. Una vez que el paquete se haya instalado correctamente, cierre la ventana del símbolo del sistema.
2. Copie el código siguiente en un editor de texto.
3. También puede reemplazar el valor de `imageUrl` por la dirección URL de una imagen diferente que desee analizar.
4. Guarde el código como un archivo con una extensión `.js`. Por ejemplo, `get-thumbnail.js`.
5. Abra una ventana de símbolo del sistema.
6. En el símbolo del sistema, utilice el comando `node` para ejecutar el archivo. Por ejemplo,

```
node get-thumbnail.js
```

.

```

'use strict';

const request = require('request');

let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and endpoint.')}

var uriBase = endpoint + 'vision/v2.1/generateThumbnail';

const imageUrl =
  'https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg';

// Request parameters.
const params = {
  'width': '100',
  'height': '100',
  'smartCropping': 'true'
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + '"' + imageUrl + '"}',
  headers: {
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key' : subscriptionKey
  }
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
});

```

Examen de la respuesta

Se devuelve una respuesta correcta como datos binarios, que representan los datos de imagen para la miniatura. Si se produce un error en la solicitud, la respuesta se muestra en la ventana de consola. La respuesta de la solicitud con error contiene un código de error y un mensaje para ayudar a determinar qué salió mal.

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Generación de una miniatura mediante la API REST Computer Vision y Python

13/01/2020 • 3 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, generará una miniatura de una imagen mediante la API REST de Computer Vision. Con el método [Get Thumbnail](#), puede especificar el alto y el ancho, y Computer Vision usa el recorte inteligente para identificar el área de interés de forma inteligente y generar coordenadas de recorte en función de esa región.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.
- Un editor de código como [Visual Studio Code](#).

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, copie el código siguiente en el editor de código.

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

thumbnail_url = endpoint + "vision/v2.1/generateThumbnail"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'width': '50', 'height': '50', 'smartCropping': 'true'}
data = {'url': image_url}
response = requests.post(thumbnail_url, headers=headers,
                        params=params, json=data)
response.raise_for_status()

thumbnail = Image.open(BytesIO(response.content))

# Display the thumbnail.
plt.imshow(thumbnail)
plt.axis("off")

# Verify the thumbnail size.
print("Thumbnail is {0}-by-{1}".format(*thumbnail.size))

```

A continuación, haga lo siguiente:

1. También puede reemplazar el valor de `image_url` por la dirección URL de una imagen diferente para la que desea generar una miniatura.
2. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `get-thumbnail.py`.
3. Abra una ventana de símbolo del sistema.
4. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python get-thumbnail.py`.

Examen de la respuesta

Se devuelve una respuesta correcta como datos binarios, que representan los datos de imagen para la miniatura. El ejemplo debe mostrar esta imagen. Si se produce un error en la solicitud, la respuesta se muestra en la ventana del símbolo del sistema y debería contener un código de error.

Ejecución en Jupyter (opcional)

Puede ejecutar este inicio rápido paso a paso mediante un cuaderno de Jupyter en [MyBinder](#). Para iniciar Binder, seleccione el botón siguiente:



Pasos siguientes

A continuación, obtenga información más detallada acerca de la característica de generación de miniaturas.

[Generación de miniaturas](#)

Inicio rápido: Extracción de texto impreso y manuscrito mediante la API REST Computer Vision y C#

16/01/2020 • 8 minutes to read • [Edit Online](#)

En este inicio rápido, extraerá texto impreso y manuscrito de una imagen mediante la API REST de Computer Vision. Los métodos [Batch Read](#) y [Read Operation Result](#) permiten detectar texto de una imagen y extraer los caracteres reconocidos en una secuencia de caracteres de lectura mecánica. La API determinará qué modelo de reconocimiento se usará para cada línea de texto, ya que admite imágenes tanto con texto impreso como manuscrito.

IMPORTANT

El método [Batch Read](#) se ejecuta de forma asincrónica. Este método no devuelve ninguna información en el cuerpo de una respuesta correcta. En su lugar, el método [Read Operation Result](#) devuelve un identificador URI en el campo del encabezado de respuesta `Operation-Location`. A continuación, puede usar este identificador URI, que representa el método [Read Operation Result](#), para comprobar el estado y devolver los resultados de la llamada al método [Batch Read](#).

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Prerequisites

- Debe tener [Visual Studio 2015 o posterior](#).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear el ejemplo en Visual Studio, siga estos pasos:

1. Cree una solución de Visual Studio en Visual Studio, mediante la plantilla de aplicación de consola de Visual C#.
2. Instale el paquete NuGet Newtonsoft.Json.
 - a. En el menú, haga clic en **Herramientas**, seleccione **Administrador de paquetes NuGet** y, luego, **Manage NuGet Packages for Solution** (Administrar paquetes NuGet para la solución).
 - b. Haga clic en la pestaña **Examinar** y, en el cuadro **Buscar**, escriba "Newtonsoft.Json".
 - c. Seleccione **Newtonsoft.Json** cuando se muestre, marque la casilla junto al nombre del proyecto y haga clic en **Instalar**.
3. Ejecute el programa.
4. En el símbolo del sistema, escriba la ruta de acceso a una imagen local.

```
using Newtonsoft.Json.Linq;  
using System;  
using System.IO;  
using System.Linq;  
using System.Net.Http;
```

```

using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Add your Computer Vision subscription key and endpoint to your environment variables.
        static string subscriptionKey = Environment.GetEnvironmentVariable("COMPUTER_VISION_SUBSCRIPTION_KEY");

        static string endpoint = Environment.GetEnvironmentVariable("COMPUTER_VISION_ENDPOINT");

        // the Batch Read method endpoint
        static string uriBase = endpoint + "vision/v2.1/read/core/asyncBatchAnalyze";

        static async Task Main()
        {
            // Get the path and filename to process from the user.
            Console.WriteLine("Text Recognition:");
            Console.Write(
                "Enter the path to an image with text you wish to read: ");
            string imageFilePath = Console.ReadLine();

            if (File.Exists(imageFilePath))
            {
                // Call the REST API method.
                Console.WriteLine("\nWait a moment for the results to appear.\n");
                await ReadText(imageFilePath);
            }
            else
            {
                Console.WriteLine("\nInvalid file path");
            }
            Console.WriteLine("\nPress Enter to exit...");
            Console.ReadLine();
        }

        /// <summary>
        /// Gets the text from the specified image file by using
        /// the Computer Vision REST API.
        /// </summary>
        /// <param name="imageFilePath">The image file with text.</param>
        static async Task ReadText(string imageFilePath)
        {
            try
            {
                HttpClient client = new HttpClient();

                // Request headers.
                client.DefaultRequestHeaders.Add(
                    "Ocp-Apim-Subscription-Key", subscriptionKey);

                // Assemble the URI for the REST API method.
                string uri = uriBase;

                HttpResponseMessage response;

                // Two REST API methods are required to extract text.
                // One method to submit the image for processing, the other method
                // to retrieve the text found in the image.

                // operationLocation stores the URI of the second REST API method,
                // returned by the first REST API method.
                string operationLocation;

                // Reads the contents of the specified local image
                // into a byte array.
                byte[] byteData = GetImageAsByteArray(imageFilePath);
            }
        }
    }
}

```

```

// Adds the byte array as an octet stream to the request body.
using (ByteArrayContent content = new ByteArrayContent(byteData))
{
    // This example uses the "application/octet-stream" content type.
    // The other content types you can use are "application/json"
    // and "multipart/form-data".
    content.Headers.ContentType =
        new MediaTypeHeaderValue("application/octet-stream");

    // The first REST API method, Batch Read, starts
    // the async process to analyze the written text in the image.
    response = await client.PostAsync(uri, content);
}

// The response header for the Batch Read method contains the URI
// of the second method, Read Operation Result, which
// returns the results of the process in the response body.
// The Batch Read operation does not return anything in the response body.
if (response.IsSuccessStatusCode)
    operationLocation =
        response.Headers.GetValues("Operation-Location").FirstOrDefault();
else
{
    // Display the JSON error data.
    string errorString = await response.Content.ReadAsStringAsync();
    Console.WriteLine("\n\nResponse:\n{0}\n",
        JToken.Parse(errorString).ToString());
    return;
}

// If the first REST API method completes successfully, the second
// REST API method retrieves the text written in the image.
//
// Note: The response may not be immediately available. Text
// recognition is an asynchronous operation that can take a variable
// amount of time depending on the length of the text.
// You may need to wait or retry this operation.
//
// This example checks once per second for ten seconds.
string contentString;
int i = 0;
do
{
    System.Threading.Thread.Sleep(1000);
    response = await client.GetAsync(operationLocation);
    contentString = await response.Content.ReadAsStringAsync();
    ++i;
}
while (i < 10 && contentString.IndexOf("\"status\": \"Succeeded\"") == -1);

if (i == 10 && contentString.IndexOf("\"status\": \"Succeeded\"") == -1)
{
    Console.WriteLine("\nTimeout error.\n");
    return;
}

// Display the JSON response.
Console.WriteLine("\nResponse:\n\n{0}\n",
    JToken.Parse(contentString).ToString());
}
catch (Exception e)
{
    Console.WriteLine("\n" + e.Message);
}
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>

```

```

    /// </summary>
    /// <param name="imageFilePath">The image file to read.</param>
    /// <returns>The byte array of the image data.</returns>
    static byte[] GetImageAsByteArray(string imageFilePath)
    {
        // Open a read-only file stream for the specified file.
        using (FileStream fileStream =
            new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
        {
            // Read the file's contents into a byte array.
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }
    }
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:

```

{
  "status": "Succeeded",
  "recognitionResults": [
    {
      "page": 1,
      "clockwiseOrientation": 349.59,
      "width": 3200,
      "height": 3200,
      "unit": "pixel",
      "lines": [
        {
          "boundingBox": [202,618,2047,643,2046,840,200,813],
          "text": "Our greatest glory is not",
          "words": [
            {
              "boundingBox": [204,627,481,628,481,830,204,829],
              "text": "Our"
            },
            {
              "boundingBox": [519,628,1057,630,1057,832,518,830],
              "text": "greatest"
            },
            {
              "boundingBox": [1114,630,1549,631,1548,833,1114,832],
              "text": "glory"
            },
            {
              "boundingBox": [1586,631,1785,632,1784,834,1586,833],
              "text": "is"
            },
            {
              "boundingBox": [1822,632,2115,633,2115,835,1822,834],
              "text": "not"
            }
          ]
        },
        {
          "boundingBox": [420,1273,2954,1250,2958,1488,422,1511],
          "text": "but in rising every time we fall",
          "words": [
            {
              "boundingBox": [423,1269,634,1268,635,1507,424,1508],
              "text": "but"
            },

```

```

    {
      "boundingBox": [667,1268,808,1268,809,1506,668,1507],
      "text": "in"
    },
    {
      "boundingBox": [874,1267,1289,1265,1290,1504,875,1506],
      "text": "rising"
    },
    {
      "boundingBox": [1331,1265,1771,1263,1772,1502,1332,1504],
      "text": "every"
    },
    {
      "boundingBox": [1812, 1263, 2178, 1261, 2179, 1500, 1813, 1502],
      "text": "time"
    },
    {
      "boundingBox": [2219, 1261, 2510, 1260, 2511, 1498, 2220, 1500],
      "text": "we"
    },
    {
      "boundingBox": [2551, 1260, 3016, 1258, 3017, 1496, 2552, 1498],
      "text": "fall"
    }
  ]
},
{
  "boundingBox": [1612, 903, 2744, 935, 2738, 1139, 1607, 1107],
  "text": "in never failing ,",
  "words": [
    {
      "boundingBox": [1611, 934, 1707, 933, 1708, 1147, 1613, 1147],
      "text": "in"
    },
    {
      "boundingBox": [1753, 933, 2132, 930, 2133, 1144, 1754, 1146],
      "text": "never"
    },
    {
      "boundingBox": [2162, 930, 2673, 927, 2674, 1140, 2164, 1144],
      "text": "failing"
    },
    {
      "boundingBox": [2703, 926, 2788, 926, 2790, 1139, 2705, 1140],
      "text": ",,",
      "confidence": "Low"
    }
  ]
}
]
}
]
}
]
}

```

Limpieza de recursos

Cuando ya no la necesite, elimine la solución de Visual Studio. Para ello, abra el Explorador de archivos, vaya a la carpeta en la que creó la solución de Visual Studio y elimine la carpeta.

Pasos siguientes

Explore una aplicación básica de Windows que utiliza Computer Vision para realizar el reconocimiento óptico de caracteres (OCR). Cree miniaturas de recorte inteligente; además de detectar, clasificar, etiquetar y describir las características visuales de una imagen, incluidas las caras.

Inicio rápido: Extracción de texto impreso y manuscrito mediante la API REST Computer Vision y Java

13/01/2020 • 8 minutes to read • [Edit Online](#)

En este inicio rápido, extraerá texto impreso y manuscrito de una imagen mediante la API REST de Computer Vision. Los métodos [Batch Read](#) y [Read Operation Result](#) permiten detectar texto de una imagen y extraer los caracteres reconocidos en una secuencia de caracteres de lectura mecánica. La API determinará qué modelo de reconocimiento se usará para cada línea de texto, ya que admite imágenes tanto con texto impreso como manuscrito.

IMPORTANT

A diferencia del método [OCR](#), el método [Batch Read](#) se ejecuta de forma asincrónica. Este método no devuelve ninguna información en el cuerpo de una respuesta correcta. En su lugar, el método Batch Read devuelve un identificador URI en el valor del campo del encabezado de respuesta `Operation-Content`. A continuación, puede llamar a este identificador URI, que representa el método [Read Operation Result](#), para comprobar el estado y devolver los resultados de la llamada al método Batch Read.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalados la plataforma de [Java™](#), y el [kit de desarrollo de edición estándar 7 u 8](#) (JDK 7 u 8).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Cree un nuevo proyecto de Java en su IDE o editor favorito. Si la opción está disponible, cree el proyecto de Java desde una plantilla de aplicación de línea de comandos.
2. Importe las bibliotecas siguientes en el proyecto de Java. Si usa Maven, se proporcionan las coordenadas de Maven para cada biblioteca.
 - [Cliente HTTP de Apache](#) (org.apache.httpcomponents:httpclient:4.5.5)
 - [Núcleo del cliente HTTP de Apache](#) (org.apache.httpcomponents:httpcore:4.4.9)
 - [Biblioteca JSON](#) (org.json:json:20180130)
3. Agregue las siguientes instrucciones `import` al archivo que contiene la clase pública `Main` para el proyecto.


```
import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.apache.http.Header;
import org.json.JSONObject;
```

4. Reemplace la clase pública `Main` por el código siguiente.
5. También puede reemplazar el valor de `imageToAnalyze` por la dirección URL de una imagen diferente desde la que desea extraer el texto.
6. Guárdela y compile el proyecto de Java.
7. Si usa un IDE, ejecute `Main`. En caso contrario, abra una ventana del símbolo del sistema y, a continuación, utilice el comando `java` para ejecutar la clase compilada. Por ejemplo, `java Main`.

```
public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Add your Computer Vision subscription key and endpoint to your environment variables.
    // After setting, close and then re-open your command shell or project for the changes to take effect.
    String subscriptionKey = System.getenv("COMPUTER_VISION_SUBSCRIPTION_KEY");
    String endpoint = ("COMPUTER_VISION_ENDPOINT");

    private static final String uriBase = endpoint +
        "vision/v2.1/read/core/asyncBatchAnalyze";

    private static final String imageToAnalyze =
        "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/" +
        "Cursive_Writing_on_Notebook_paper.jpg/800px-Cursive_Writing_on_Notebook_paper.jpg";

    public static void main(String[] args) {
        CloseableHttpClient httpTextClient = HttpClientBuilder.create().build();
        CloseableHttpClient httpResultClient = HttpClientBuilder.create().build();

        try {
            // This operation requires two REST API calls. One to submit the image
            // for processing, the other to retrieve the text found in the image.

            URIBuilder builder = new URIBuilder(uriBase);

            // Prepare the URI for the REST API method.
            URI uri = builder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body.
            StringEntity requestEntity =
                new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
            request.setEntity(requestEntity);

            // Two REST API methods are required to extract text.
            // One method to submit the image for processing, the other method
```

```

// to retrieve the text found in the image.

// Call the first REST API method to detect the text.
HttpResponse response = httpTextClient.execute(request);

// Check for success.
if (response.getStatusLine().getStatusCode() != 202) {
    // Format and display the JSON error message.
    HttpEntity entity = response.getEntity();
    String jsonString = EntityUtils.toString(entity);
    JSONObject json = new JSONObject(jsonString);
    System.out.println("Error:\n");
    System.out.println(json.toString(2));
    return;
}

// Store the URI of the second REST API method.
// This URI is where you can get the results of the first REST API method.
String operationLocation = null;

// The 'Operation-Location' response header value contains the URI for
// the second REST API method.
Header[] responseHeaders = response.getAllHeaders();
for (Header header : responseHeaders) {
    if (header.getName().equals("Operation-Location")) {
        operationLocation = header.getValue();
        break;
    }
}

if (operationLocation == null) {
    System.out.println("\nError retrieving Operation-Location.\nExiting.");
    System.exit(1);
}

// If the first REST API method completes successfully, the second
// REST API method retrieves the text written in the image.
//
// Note: The response may not be immediately available. Text
// recognition is an asynchronous operation that can take a variable
// amount of time depending on the length of the text.
// You may need to wait or retry this operation.

System.out.println("\nText submitted.\n" +
    "Waiting 10 seconds to retrieve the recognized text.\n");
Thread.sleep(10000);

// Call the second REST API method and get the response.
HttpGet resultRequest = new HttpGet(operationLocation);
resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

HttpResponse resultResponse = httpResultClient.execute(resultRequest);
HttpEntity responseEntity = resultResponse.getEntity();

if (responseEntity != null) {
    // Format and display the JSON response.
    String jsonString = EntityUtils.toString(responseEntity);
    JSONObject json = new JSONObject(jsonString);
    System.out.println("Text recognition result response: \n");
    System.out.println(json.toString(2));
}
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:

Text submitted. Waiting 10 seconds to retrieve the recognized text.

Text recognition result response:

```
{
  "status": "Succeeded",
  "recognitionResults": [
    {
      "page": 1,
      "clockwiseOrientation": 349.59,
      "width": 3200,
      "height": 3200,
      "unit": "pixel",
      "lines": [
        {
          "boundingBox": [202,618,2047,643,2046,840,200,813],
          "text": "Our greatest glory is not",
          "words": [
            {
              "boundingBox": [204,627,481,628,481,830,204,829],
              "text": "Our"
            },
            {
              "boundingBox": [519,628,1057,630,1057,832,518,830],
              "text": "greatest"
            },
            {
              "boundingBox": [1114,630,1549,631,1548,833,1114,832],
              "text": "glory"
            },
            {
              "boundingBox": [1586,631,1785,632,1784,834,1586,833],
              "text": "is"
            },
            {
              "boundingBox": [1822,632,2115,633,2115,835,1822,834],
              "text": "not"
            }
          ]
        },
        {
          "boundingBox": [420,1273,2954,1250,2958,1488,422,1511],
          "text": "but in rising every time we fall",
          "words": [
            {
              "boundingBox": [423,1269,634,1268,635,1507,424,1508],
              "text": "but"
            },
            {
              "boundingBox": [667,1268,808,1268,809,1506,668,1507],
              "text": "in"
            },
            {
              "boundingBox": [874,1267,1289,1265,1290,1504,875,1506],
              "text": "rising"
            },
            {
              "boundingBox": [1331,1265,1771,1263,1772,1502,1332,1504],
              "text": "every"
            },
            {
              "boundingBox": [1812, 1263, 2178, 1261, 2179, 1500, 1813, 1502],

```

```

        "text": "time"
    },
    {
        "boundingBox": [2219, 1261, 2510, 1260, 2511, 1498, 2220, 1500],
        "text": "we"
    },
    {
        "boundingBox": [2551, 1260, 3016, 1258, 3017, 1496, 2552, 1498],
        "text": "fall"
    }
]
},
{
    "boundingBox": [1612, 903, 2744, 935, 2738, 1139, 1607, 1107],
    "text": "in never failing ,",
    "words": [
        {
            "boundingBox": [1611, 934, 1707, 933, 1708, 1147, 1613, 1147],
            "text": "in"
        },
        {
            "boundingBox": [1753, 933, 2132, 930, 2133, 1144, 1754, 1146],
            "text": "never"
        },
        {
            "boundingBox": [2162, 930, 2673, 927, 2674, 1140, 2164, 1144],
            "text": "failing"
        },
        {
            "boundingBox": [2703, 926, 2788, 926, 2790, 1139, 2705, 1140],
            "text": ",",
            "confidence": "Low"
        }
    ]
}
]
}
]
}
}

```

Limpieza de recursos

Cuando ya no lo necesite, elimine el proyecto de Java, incluidas las bibliotecas importadas y de clase compilada.

Pasos siguientes

Explore una aplicación de Java Swing que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Computer Vision API Java Tutorial](#) (Tutorial de Computer Vision API para Java)

Inicio rápido: Extracción de texto impreso y manuscrito mediante la API REST Computer Vision y JavaScript

13/01/2020 • 7 minutes to read • [Edit Online](#)

En este inicio rápido, extraerá texto impreso y manuscrito de una imagen mediante la API REST de Computer Vision. Los métodos [Batch Read](#) y [Read Operation Result](#) permiten detectar texto de una imagen y extraer los caracteres reconocidos en una secuencia de caracteres de lectura mecánica. La API determinará qué modelo de reconocimiento se usará para cada línea de texto, ya que admite imágenes tanto con texto impreso como manuscrito.

IMPORTANT

A diferencia del método [OCR](#), el método [Batch Read](#) se ejecuta de forma asincrónica. Este método no devuelve ninguna información en el cuerpo de una respuesta correcta. En su lugar, el método Batch Read devuelve un identificador URI en el valor del campo del encabezado de respuesta `Operation-Content`. A continuación, puede llamar a este identificador URI, que representa el método [Read Operation Result](#), para comprobar el estado y devolver los resultados de la llamada al método Batch Read.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor del atributo `value` para el control de `inputImage` por la dirección URL de una imagen diferente desde la que desea extraer el texto.
3. Guarde el código como un archivo con la extensión `.html`. Por ejemplo, `get-text.html`.
4. Abra una ventana del explorador.
5. En el explorador, arrastre y coloque el archivo en la ventana del explorador.
6. Cuando la página web se muestra en el explorador, seleccione el botón **Read image** (Leer imagen).

```
<!DOCTYPE html>
<html>
<head>
  <title>Text Recognition Sample</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>
```

```

<script type="text/javascript">
    function processImage() {
        // *****
        // *** Update or verify the following values. ***
        // *****

        let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
        let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
        if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and
endpoint.');
```

```

        }; }

        var uriBase = endpoint + "vision/v2.1/read/core/asyncBatchAnalyze";

        // Display the image.
        var sourceImageUrl = document.getElementById("inputImage").value;
        document.querySelector("#sourceImage").src = sourceImageUrl;

        // This operation requires two REST API calls. One to submit the image
        // for processing, the other to retrieve the text found in the image.
        //
        // Make the first REST API call to submit the image for processing.
        $.ajax({
            url: uriBase,

            // Request headers.
            beforeSend: function(jqXHR){
                jqXHR.setRequestHeader("Content-Type","application/json");
                jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
            },

            type: "POST",

            // Request body.
            data: '{"url": ' + "'" + sourceImageUrl + "'",
        })

        .done(function(data, textStatus, jqXHR) {
            // Show progress.
            $("#responseTextArea").val("Text submitted. " +
                "Waiting 10 seconds to retrieve the recognized text.");

            // Note: The response may not be immediately available. Text
            // recognition is an asynchronous operation that can take a variable
            // amount of time depending on the length of the text you want to
            // recognize. You may need to wait or retry the GET operation.
            //
            // Wait ten seconds before making the second REST API call.
            setTimeout(function () {
                // "Operation-Location" in the response contains the URI
                // to retrieve the recognized text.
                var operationLocation = jqXHR.getResponseHeader("Operation-Location");

                // Make the second REST API call and get the response.
                $.ajax({
                    url: operationLocation,

                    // Request headers.
                    beforeSend: function(jqXHR){
                        jqXHR.setRequestHeader("Content-Type","application/json");
                        jqXHR.setRequestHeader(
                            "Ocp-Apim-Subscription-Key", subscriptionKey);
                    },

                    type: "GET",
                })

                .done(function(data) {
                    // Show formatted JSON on webpage.
                    $("#responseTextArea").val(JSON.stringify(data, null, 2));
                });
            }, 10000);
        });
    }

```

```

    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Display error message.
        var errorString = (errorThrown === "") ? "Error. " :
            errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" :
            (jQuery.parseJSON(jqXHR.responseText).message) ?
                jQuery.parseJSON(jqXHR.responseText).message :
                jQuery.parseJSON(jqXHR.responseText).error.message;
        alert(errorString);
    });
}, 10000);
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Put the JSON description into the text area.
    $("#responseTextArea").val(JSON.stringify(jqXHR, null, 2));

    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " :
        errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" :
        (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message :
            jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
};
</script>
<h1>Read text from image:</h1>
Enter the URL to an image of text, then click
the <strong>Read image</strong> button.
<br><br>
Image to read:
<input type="text" name="inputImage" id="inputImage"

value="https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Cursive_Writing_on_Notebook_paper.jpg/800px-
Cursive_Writing_on_Notebook_paper.jpg" />
<button onclick="processImage()">Read image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput"
            style="width:580px; height:400px;"></textarea>
    </div>
    <div id="imageDiv" style="width:420px; display:table-cell;">
        Source image:
        <br><br>
        <img id="sourceImage" width="400" />
    </div>
</div>
</body>
</html>

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del explorador, parecida a la del ejemplo siguiente:

```

{
  "status": "Succeeded",
  "recognitionResults": [
    ,

```

```

{
  "page": 1,
  "clockwiseOrientation": 349.59,
  "width": 3200,
  "height": 3200,
  "unit": "pixel",
  "lines": [
    {
      "boundingBox": [202,618,2047,643,2046,840,200,813],
      "text": "Our greatest glory is not",
      "words": [
        {
          "boundingBox": [204,627,481,628,481,830,204,829],
          "text": "Our"
        },
        {
          "boundingBox": [519,628,1057,630,1057,832,518,830],
          "text": "greatest"
        },
        {
          "boundingBox": [1114,630,1549,631,1548,833,1114,832],
          "text": "glory"
        },
        {
          "boundingBox": [1586,631,1785,632,1784,834,1586,833],
          "text": "is"
        },
        {
          "boundingBox": [1822,632,2115,633,2115,835,1822,834],
          "text": "not"
        }
      ]
    },
    {
      "boundingBox": [420,1273,2954,1250,2958,1488,422,1511],
      "text": "but in rising every time we fall",
      "words": [
        {
          "boundingBox": [423,1269,634,1268,635,1507,424,1508],
          "text": "but"
        },
        {
          "boundingBox": [667,1268,808,1268,809,1506,668,1507],
          "text": "in"
        },
        {
          "boundingBox": [874,1267,1289,1265,1290,1504,875,1506],
          "text": "rising"
        },
        {
          "boundingBox": [1331,1265,1771,1263,1772,1502,1332,1504],
          "text": "every"
        },
        {
          "boundingBox": [1812, 1263, 2178, 1261, 2179, 1500, 1813, 1502],
          "text": "time"
        },
        {
          "boundingBox": [2219, 1261, 2510, 1260, 2511, 1498, 2220, 1500],
          "text": "we"
        },
        {
          "boundingBox": [2551, 1260, 3016, 1258, 3017, 1496, 2552, 1498],
          "text": "fall"
        }
      ]
    }
  ],
  {
    "boundingBox": [1612, 903, 2744, 935, 2738, 1139, 1607, 1107],

```



```
"text": "in never failing ,",
"words": [
  {
    "boundingBox": [1611, 934, 1707, 933, 1708, 1147, 1613, 1147],
    "text": "in"
  },
  {
    "boundingBox": [1753, 933, 2132, 930, 2133, 1144, 1754, 1146],
    "text": "never"
  },
  {
    "boundingBox": [2162, 930, 2673, 927, 2674, 1140, 2164, 1144],
    "text": "failing"
  },
  {
    "boundingBox": [2703, 926, 2788, 926, 2790, 1139, 2705, 1140],
    "text": ",,",
    "confidence": "Low"
  }
]
}
]
```

Pasos siguientes

Explore una aplicación de JavaScript que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR); crear miniaturas con recorte inteligente; y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para JavaScript](#)

Inicio rápido: Extracción de texto impreso y manuscrito mediante la API REST Computer Vision y Python

13/01/2020 • 7 minutes to read • [Edit Online](#)

En este inicio rápido, extraerá texto impreso y manuscrito de una imagen mediante la API REST de Computer Vision. Los métodos [Batch Read](#) y [Read Operation Result](#) permiten detectar texto de una imagen y extraer los caracteres reconocidos en una secuencia de caracteres de lectura mecánica. La API determinará qué modelo de reconocimiento se usará para cada línea de texto, ya que admite imágenes tanto con texto impreso como manuscrito.

IMPORTANT

A diferencia del método [OCR](#), el método [Batch Read](#) se ejecuta de forma asincrónica. Este método no devuelve ninguna información en el cuerpo de una respuesta correcta. En su lugar, el método Batch Read devuelve un identificador URI en el valor del campo del encabezado de respuesta `Operation-Content`. A continuación, puede llamar a este identificador URI, que representa a [Read Operation Result](#) API, para comprobar el estado y devolver los resultados de la llamada al método Batch Read.

Puede ejecutar esta guía de inicio rápido paso a paso mediante un cuaderno de Jupyter en [MyBinder](#). Para iniciar Binder, seleccione el botón siguiente:

[launch](#) [binder](#)

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Python](#) instalado si desea ejecutar el ejemplo localmente.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `image_url` por la dirección URL de una imagen diferente desde la que desea extraer el texto.
3. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `get-text.py`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python get-text.py`.

```
import requests
```

```

import time
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

text_recognition_url = endpoint + "vision/v2.1/read/core/asyncBatchAnalyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/d/dd/Cursive_Writing_on_Notebook_paper.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
data = {'url': image_url}
response = requests.post(
    text_recognition_url, headers=headers, json=data)
response.raise_for_status()

# Extracting text requires two API calls: One call to submit the
# image for processing, the other to retrieve the text found in the image.

# Holds the URI used to retrieve the recognized text.
operation_url = response.headers["Operation-Location"]

# The recognized text isn't immediately available, so poll to wait for completion.
analysis = {}
poll = True
while (poll):
    response_final = requests.get(
        response.headers["Operation-Location"], headers=headers)
    analysis = response_final.json()
    print(analysis)
    time.sleep(1)
    if ("recognitionResults" in analysis):
        poll = False
    if ("status" in analysis and analysis['status'] == 'Failed'):
        poll = False

polygons = []
if ("recognitionResults" in analysis):
    # Extract the recognized text, with bounding boxes.
    polygons = [(line["boundingBox"], line["text"])
                for line in analysis["recognitionResults"][0]["lines"]]

# Display the image and overlay it with the extracted text.
plt.figure(figsize=(15, 15))
image = Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image)
for polygon in polygons:
    vertices = [(polygon[0][i], polygon[0][i+1])
                for i in range(0, len(polygon[0]), 2)]
    text = polygon[1]
    patch = Polygon(vertices, closed=True, fill=False, linewidth=2, color='y')
    ax.axes.add_patch(patch)
    plt.text(vertices[0][0], vertices[0][1], text, fontsize=20, va="top")

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```
{
  "status": "Succeeded",
  "recognitionResult": {
    "lines": [
      {
        "boundingBox": [
          2,
          52,
          65,
          46,
          69,
          89,
          7,
          95
        ],
        "text": "dog",
        "words": [
          {
            "boundingBox": [
              0,
              59,
              63,
              43,
              77,
              86,
              3,
              102
            ],
            "text": "dog"
          }
        ]
      },
      {
        "boundingBox": [
          6,
          2,
          771,
          13,
          770,
          75,
          5,
          64
        ],
        "text": "The quick brown fox jumps over the lazy",
        "words": [
          {
            "boundingBox": [
              0,
              4,
              92,
              5,
              77,
              71,
              0,
              71
            ],
            "text": "The"
          }
        ]
      },
      {
        "boundingBox": [
          74,
          4,
```

```
189,  
5,  
174,  
72,  
60,  
71  
],  
"text": "quick"  
},  
{  
  "boundingBox": [  
    176,  
    5,  
    321,  
    6,  
    306,  
    73,  
    161,  
    72  
  ],  
  "text": "brown"  
},  
{  
  "boundingBox": [  
    308,  
    6,  
    387,  
    6,  
    372,  
    73,  
    293,  
    73  
  ],  
  "text": "fox"  
},  
{  
  "boundingBox": [  
    382,  
    6,  
    506,  
    7,  
    491,  
    74,  
    368,  
    73  
  ],  
  "text": "jumps"  
},  
{  
  "boundingBox": [  
    492,  
    7,  
    607,  
    8,  
    592,  
    75,  
    478,  
    74  
  ],  
  "text": "over"  
},  
{  
  "boundingBox": [  
    589,  
    8,  
    673,  
    8,  
    658,  
    75,
```

```

        575,
        75
    ],
    "text": "the"
},
{
    "boundingBox": [
        660,
        8,
        783,
        9,
        768,
        76,
        645,
        75
    ],
    "text": "lazy"
}
]
},
{
    "boundingBox": [
        2,
        84,
        783,
        96,
        782,
        154,
        1,
        148
    ],
    "text": "Pack my box with five dozen liquor jugs",
    "words": [
        {
            "boundingBox": [
                0,
                86,
                94,
                87,
                72,
                151,
                0,
                149
            ],
            "text": "Pack"
        },
        {
            "boundingBox": [
                76,
                87,
                164,
                88,
                142,
                152,
                54,
                150
            ],
            "text": "my"
        },
        {
            "boundingBox": [
                155,
                88,
                243,
                89,
                222,
                152,
                134,
                151
            ],

```

```

    ],
    "text": "box"
  },
  {
    "boundingBox": [
      226,
      89,
      344,
      90,
      323,
      154,
      204,
      152
    ],
    "text": "with"
  },
  {
    "boundingBox": [
      336,
      90,
      432,
      91,
      411,
      154,
      314,
      154
    ],
    "text": "five"
  },
  {
    "boundingBox": [
      419,
      91,
      538,
      92,
      516,
      154,
      398,
      154
    ],
    "text": "dozen"
  },
  {
    "boundingBox": [
      547,
      92,
      701,
      94,
      679,
      154,
      525,
      154
    ],
    "text": "liquor"
  },
  {
    "boundingBox": [
      696,
      94,
      800,
      95,
      780,
      154,
      675,
      154
    ],
    "text": "jugs"
  }
}

```

```
}  
}  
}  
}
```

Pasos siguientes

Explore una aplicación de Python que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para Python](#)

Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision y C#

16/01/2020 • 5 minutes to read • [Edit Online](#)

NOTE

Si va a extraer texto en idioma inglés, considere la posibilidad de usar la nueva [operación de lectura](#). Hay disponible una [guía de inicio rápido para C#](#).

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con la característica [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Prerequisites

- Debe tener [Visual Studio 2015](#) o posterior.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear el ejemplo en Visual Studio, siga estos pasos:

1. Cree una solución de Visual Studio en Visual Studio, mediante la plantilla de aplicación de consola de Visual C#.
2. Instale el paquete NuGet Newtonsoft.Json.
 - a. En el menú, haga clic en **Herramientas**, seleccione **Administrador de paquetes NuGet** y, luego, **Manage NuGet Packages for Solution** (Administrar paquetes NuGet para la solución).
 - b. Haga clic en la pestaña **Examinar** y, en el cuadro **Buscar**, escriba "Newtonsoft.Json".
 - c. Seleccione **Newtonsoft.Json** cuando se muestre, marque la casilla junto al nombre del proyecto y haga clic en **Instalar**.
3. Ejecute el programa.
4. En el símbolo del sistema, escriba la ruta de acceso a una imagen local.

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
```

```

// Add your Computer Vision subscription key and endpoint to your environment variables.
static string subscriptionKey =
Environment.GetEnvironmentVariable("COMPUTER_VISION_SUBSCRIPTION_KEY");

static string endpoint = Environment.GetEnvironmentVariable("COMPUTER_VISION_ENDPOINT");

// the OCR method endpoint
static string uriBase = endpoint + "vision/v2.1/ocr";

static async Task Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Optical Character Recognition:");
    Console.Write("Enter the path to an image with text you wish to read: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Call the REST API method.
        Console.WriteLine("\nWait a moment for the results to appear.\n");
        await MakeOCRRequest(imageFilePath);
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets the text visible in the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file with printed text.</param>
static async Task MakeOCRRequest(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameters.
        // The language parameter doesn't specify a language, so the
        // method detects it automatically.
        // The detectOrientation parameter is set to true, so the method detects and
        // and corrects text orientation before detecting text.
        string requestParameters = "language=unk&detectOrientation=true";

        // Assemble the URI for the REST API method.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Read the contents of the specified local image
        // into a byte array.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        // Add the byte array as an octet stream to the request body.
        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses the "application/octet-stream" content type.
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");

```

```

        // Asynchronously call the REST API method.
        response = await client.PostAsync(uri, content);
    }

    // Asynchronously get the JSON response.
    string contentString = await response.Content.ReadAsStringAsync();

    // Display the JSON response.
    Console.WriteLine("\nResponse:\n\n{0}\n",
        JToken.Parse(contentString).ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("\n" + e.Message);
    }
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    // Open a read-only file stream for the specified file.
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        // Read the file's contents into a byte array.
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:

```

{
  "language": "en",
  "textAngle": -1.5000000000000335,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "154,49,351,575",
      "lines": [
        {
          "boundingBox": "165,49,340,117",
          "words": [
            {
              "boundingBox": "165,49,63,109",
              "text": "A"
            },
            {
              "boundingBox": "261,50,244,116",
              "text": "GOAL"
            }
          ]
        }
      ],
      {
        "boundingBox": "165,169,339,93",
        "words": [

```


Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision y cURL

13/01/2020 • 3 minutes to read • [Edit Online](#)

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con el método [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [cURL](#).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave.

Creación y ejecución del comando de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el comando siguiente en un editor de texto.
2. Realice los siguientes cambios en el comando donde sea necesario:
 - a. Reemplace el valor de `<subscriptionKey>` por la clave de suscripción.
 - b. Reemplace la primera parte de la dirección URL de la solicitud (`westcentralus`) por el texto de la dirección URL de su punto de conexión.

NOTE

Los nuevos recursos creados después del 1 de julio de 2019 usarán nombres de subdominio personalizados. Para más información y para obtener una lista completa de los puntos de conexión regionales, consulte [Nombres de subdominios personalizados para Cognitive Services](#).

- c. Si lo desea, cambie la dirección URL de la imagen del cuerpo de la solicitud (`https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png\`) por la dirección URL de una imagen diferente que desee analizar.
3. Abra una ventana de símbolo del sistema.
 4. Pegue el comando del editor de texto en la ventana del símbolo del sistema y después ejecute el comando.

```
curl -H "Ocp-Apim-Subscription-Key: <subscriptionKey>" -H "Content-Type: application/json"
"https://westcentralus.api.cognitive.microsoft.com/vision/v2.1/ocr?language=unk&detectOrientation=true" -d "{
  \"url\": \"https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png\"}"
```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```
{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        },
        {
          "boundingBox": "27,66,283,52",
          "words": [
            {
              "boundingBox": "27,66,283,52",
              "text": "EXISTS"
            }
          ]
        },
        {
          "boundingBox": "27,128,292,49",
          "words": [
            {
              "boundingBox": "27,128,292,49",
              "text": "EXCEPT"
            }
          ]
        },
        {
          "boundingBox": "24,188,292,54",
          "words": [
            {
              "boundingBox": "24,188,292,54",
              "text": "ATOMS"
            }
          ]
        },
        {
          "boundingBox": "22,253,297,32",
          "words": [
            {
              "boundingBox": "22,253,105,32",
              "text": "AND"
            },
            {
              "boundingBox": "144,253,175,32",
              "text": "EMPTY"
            }
          ]
        },
        {
          "boundingBox": "21,298,304,60",
          "words": [
            {
              "boundingBox": "21,298,304,60",
              "text": "SPACE."
            }
          ]
        },
        {
          "boundingBox": "26,387,294,37",
```

```
    "words": [
      {
        "boundingBox": "26,387,210,37",
        "text": "Everything"
      },
      {
        "boundingBox": "249,389,71,27",
        "text": "else"
      }
    ]
  },
  {
    "boundingBox": "127,431,198,36",
    "words": [
      {
        "boundingBox": "127,431,31,29",
        "text": "is"
      },
      {
        "boundingBox": "172,431,153,36",
        "text": "opinion."
      }
    ]
  }
]
}
```

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision con Go

13/01/2020 • 4 minutes to read • [Edit Online](#)

NOTE

Si va a extraer texto en idioma inglés, considere la posibilidad de usar la nueva [operación de lectura](#). Hay disponible una [guía de inicio rápido para Go](#).

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con el método [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Go](#) instalado.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `imageUrl` por la dirección URL de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con una extensión `.go`. Por ejemplo, `get-printed-text.go`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, ejecute el comando `go build` para compilar el paquete desde el archivo. Por ejemplo, `go build get-printed-text.go`.
6. En el símbolo del sistema, ejecute el paquete compilado. Por ejemplo, `get-printed-text`.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Add your Computer Vision subscription key and endpoint to your environment variables.
    subscriptionKey := os.Getenv("COMPUTER_VISION_SUBSCRIPTION_KEY")
```



```

subscriptionKey := os.Getenv("COMPUTER_VISION_SUBSCRIPTION_KEY")
if (subscriptionKey == "") {
    log.Fatal("\n\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n" +
        "**Restart your shell or IDE for changes to take effect.**\n")

    endpoint := os.Getenv("COMPUTER_VISION_ENDPOINT")
    if ("" == endpoint) {
        log.Fatal("\n\nSet the COMPUTER_VISION_ENDPOINT environment variable.\n" +
            "**Restart your shell or IDE for changes to take effect.**")
    }
    const uriBase = endpoint + "vision/v2.1/ocr"
    const imageUrl = "https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/" +
        "Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png"

    const params = "?language=unk&detectOrientation=true"
    const uri = uriBase + params
    const imageUrlEnc = "{\"url\":\"" + imageUrl + "\"}"

    reader := strings.NewReader(imageUrlEnc)

    // Create the Http client
    client := &http.Client{
        Timeout: time.Second * 2,
    }

    // Create the Post request, passing the image URL in the request body
    req, err := http.NewRequest("POST", uri, reader)
    if err != nil {
        panic(err)
    }

    // Add headers
    req.Header.Add("Content-Type", "application/json")
    req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

    // Send the request and retrieve the response
    resp, err := client.Do(req)
    if err != nil {
        panic(err)
    }

    defer resp.Body.Close()

    // Read the response body.
    // Note, data is a byte array
    data, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        panic(err)
    }

    // Parse the Json data
    var f interface{}
    json.Unmarshal(data, &f)

    // Format and display the Json result
    jsonFormatted, _ := json.MarshalIndent(f, "", " ")
    fmt.Println(string(jsonFormatted))
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```

{
  "language": "en",

```

```
"orientation": "Up",
"regions": [
  {
    "boundingBox": "21,16,304,451",
    "lines": [
      {
        "boundingBox": "28,16,288,41",
        "words": [
          {
            "boundingBox": "28,16,288,41",
            "text": "NOTHING"
          }
        ]
      },
      {
        "boundingBox": "27,66,283,52",
        "words": [
          {
            "boundingBox": "27,66,283,52",
            "text": "EXISTS"
          }
        ]
      },
      {
        "boundingBox": "27,128,292,49",
        "words": [
          {
            "boundingBox": "27,128,292,49",
            "text": "EXCEPT"
          }
        ]
      },
      {
        "boundingBox": "24,188,292,54",
        "words": [
          {
            "boundingBox": "24,188,292,54",
            "text": "ATOMS"
          }
        ]
      },
      {
        "boundingBox": "22,253,297,32",
        "words": [
          {
            "boundingBox": "22,253,105,32",
            "text": "AND"
          },
          {
            "boundingBox": "144,253,175,32",
            "text": "EMPTY"
          }
        ]
      },
      {
        "boundingBox": "21,298,304,60",
        "words": [
          {
            "boundingBox": "21,298,304,60",
            "text": "SPACE."
          }
        ]
      },
      {
        "boundingBox": "26,387,294,37",
        "words": [
          {
            "boundingBox": "26,387,210,37",
            "text": "Everything"
          }
        ]
      }
    ]
  }
]
```

```
    },
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ]
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion."
    }
  ]
}
],
"textAngle": 0
}
```

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision y Java

13/01/2020 • 5 minutes to read • [Edit Online](#)

NOTE

Si va a extraer texto en idioma inglés, considere la posibilidad de usar la nueva [operación de lectura](#). Hay disponible una [guía de inicio rápido para Java](#).

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con el método [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalados la plataforma de [Java™](#), y el [kit de desarrollo de edición estándar 7 u 8](#) (JDK 7 u 8).
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Prueba Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución de la aplicación de ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Cree un nuevo proyecto de Java en su IDE o editor favorito. Si la opción está disponible, cree el proyecto de Java desde una plantilla de aplicación de línea de comandos.
2. Importe las bibliotecas siguientes en el proyecto de Java. Si usa Maven, se proporcionan las coordenadas de Maven para cada biblioteca.
 - [Cliente HTTP de Apache](#) (org.apache.httpcomponents:httpclient:4.5.5)
 - [Núcleo del cliente HTTP de Apache](#) (org.apache.httpcomponents:httpcore:4.4.9)
 - [Biblioteca JSON](#) (org.json:json:20180130)
3. Agregue las siguientes instrucciones `import` al archivo que contiene la clase pública `Main` para el proyecto.

```
import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;
```

4. Reemplace la clase pública `Main` por el código siguiente.
5. También puede reemplazar el valor de `imageToAnalyze` por la dirección URL de una imagen diferente desde la que desea extraer el texto impreso.
6. Guárdela y compile el proyecto de Java.
7. Si usa un IDE, ejecute `Main`. En caso contrario, abra una ventana del símbolo del sistema y, a continuación, utilice el comando `java` para ejecutar la clase compilada. Por ejemplo, `java Main`.

```

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Add your Computer Vision subscription key and endpoint to your environment variables.
    // After setting, close and then re-open your command shell or project for the changes to take effect.
    String subscriptionKey = System.getenv("COMPUTER_VISION_SUBSCRIPTION_KEY");
    String endpoint = ("COMPUTER_VISION_ENDPOINT");

    private static final String uriBase = endpoint +
        "vision/v2.1/ocr";

    private static final String imageToAnalyze =
        "https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/" +
        "Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png";

    public static void main(String[] args) {
        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            URIBuilder uriBuilder = new URIBuilder(uriBase);

            uriBuilder.setParameter("language", "unk");
            uriBuilder.setParameter("detectOrientation", "true");

            // Request parameters.
            URI uri = uriBuilder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body.
            StringEntity requestEntity =
                new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
            request.setEntity(requestEntity);

            // Call the REST API method and get the response entity.
            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null) {
                // Format and display the JSON response.
                String jsonString = EntityUtils.toString(entity);
                JSONObject json = new JSONObject(jsonString);
                System.out.println("REST Response:\n");
                System.out.println(json.toString(2));
            }
        } catch (Exception e) {
            // Display error message.
            System.out.println(e.getMessage());
        }
    }
}

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La aplicación de ejemplo analiza y muestra una respuesta correcta en la ventana de la consola, parecida a la del ejemplo siguiente:

```

REST Response:

```

```

{

```

```
"orientation": "Up",
"regions": [{
  "boundingBox": "21,16,304,451",
  "lines": [
    {
      "boundingBox": "28,16,288,41",
      "words": [{
        "boundingBox": "28,16,288,41",
        "text": "NOTHING"
      }]
    },
    {
      "boundingBox": "27,66,283,52",
      "words": [{
        "boundingBox": "27,66,283,52",
        "text": "EXISTS"
      }]
    },
    {
      "boundingBox": "27,128,292,49",
      "words": [{
        "boundingBox": "27,128,292,49",
        "text": "EXCEPT"
      }]
    },
    {
      "boundingBox": "24,188,292,54",
      "words": [{
        "boundingBox": "24,188,292,54",
        "text": "ATOMS"
      }]
    },
    {
      "boundingBox": "22,253,297,32",
      "words": [
        {
          "boundingBox": "22,253,105,32",
          "text": "AND"
        },
        {
          "boundingBox": "144,253,175,32",
          "text": "EMPTY"
        }
      ]
    },
    {
      "boundingBox": "21,298,304,60",
      "words": [{
        "boundingBox": "21,298,304,60",
        "text": "SPACE."
      }]
    },
    {
      "boundingBox": "26,387,294,37",
      "words": [
        {
          "boundingBox": "26,387,210,37",
          "text": "Everything"
        },
        {
          "boundingBox": "249,389,71,27",
          "text": "else"
        }
      ]
    },
    {
      "boundingBox": "127,431,198,36",
      "words": [
```

```
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion."
    }
  ]
}
]
}],
"textAngle": 0,
"language": "en"
}
```

Pasos siguientes

Explore una aplicación de Java Swing que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Computer Vision API Java Tutorial](#) (Tutorial de Computer Vision API para Java)

Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision y JavaScript

13/01/2020 • 4 minutes to read • [Edit Online](#)

NOTE

Si va a extraer texto en idioma inglés, considere la posibilidad de usar la nueva [operación de lectura](#). Hay disponible una [guía de inicio rápido para JavaScript](#).

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con el método [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor del atributo `value` para el control `inputImage` por la dirección URL de una imagen diferente que desee analizar.
3. Guarde el código como un archivo con la extensión `.html`. Por ejemplo, `get-printed-text.html`.
4. Abra una ventana del explorador.
5. En el explorador, arrastre y coloque el archivo en la ventana del explorador.
6. Cuando la página web se muestra en el explorador, seleccione el botón **Read image** (Leer imagen).

```
<!DOCTYPE html>
<html>
<head>
  <title>OCR Sample</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
    let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
    if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and
```

```
if (!subscriptionKey) { throw new Error("Set your environment variables for your subscription key and endpoint."); }
```

```
var uriBase = endpoint + "vision/v2.1/ocr";
```

```
// Request parameters.
```

```
var params = {
  "language": "unk",
  "detectOrientation": "true",
};
```

```
// Display the image.
```

```
var sourceImageUrl = document.getElementById("inputImage").value;
document.querySelector("#sourceImage").src = sourceImageUrl;
```

```
// Perform the REST API call.
```

```
$.ajax({
  url: uriBase + "?" + $.param(params),

  // Request headers.
  beforeSend: function(jqXHR){
    jqXHR.setRequestHeader("Content-Type","application/json");
    jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
  },

  type: "POST",

  // Request body.
  data: '{"url": ' + '"' + sourceImageUrl + '"}',
})
```

```
.done(function(data) {
  // Show formatted JSON on webpage.
  $("#responseTextArea").val(JSON.stringify(data, null, 2));
})
```

```
.fail(function(jqXHR, textStatus, errorThrown) {
  // Display error message.
  var errorString = (errorThrown === "") ?
    "Error. " : errorThrown + " (" + jqXHR.status + "): ";
  errorString += (jqXHR.responseText === "") ? "" :
    (jQuery.parseJSON(jqXHR.responseText).message) ?
      jQuery.parseJSON(jqXHR.responseText).message :
      jQuery.parseJSON(jqXHR.responseText).error.message;
  alert(errorString);
});
```

```
});
```

```
</script>
```

```
<h1>Optical Character Recognition (OCR)</h1>
```

```
Enter the URL to an image of printed text, then
```

```
click the <strong>Read image</strong> button.
```

```
<br><br>
```

```
Image to read:
```

```
<input type="text" name="inputImage" id="inputImage"
```

```
  value="https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-
```

```
Atomist_quote_from_Democritus.png" />
```

```
<button onclick="processImage()">Read image</button>
```

```
<br><br>
```

```
<div id="wrapper" style="width:1020px; display:table;">
```

```
  <div id="jsonOutput" style="width:600px; display:table-cell;">
```

```
    Response:
```

```
    <br><br>
```

```
    <textarea id="responseTextArea" class="UIInput"
```

```
      style="width:580px; height:400px;"></textarea>
```

```
</div>
```

```
<div id="imageDiv" style="width:420px; display:table-cell;">
```

```
  Source image:
```

```
  <br><br>
```

```
<img id="sourceImage" width="400" />
</div>
</div>
</body>
</html>
```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del explorador, parecida a la del ejemplo siguiente:

```
{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        },
        {
          "boundingBox": "27,66,283,52",
          "words": [
            {
              "boundingBox": "27,66,283,52",
              "text": "EXISTS"
            }
          ]
        },
        {
          "boundingBox": "27,128,292,49",
          "words": [
            {
              "boundingBox": "27,128,292,49",
              "text": "EXCEPT"
            }
          ]
        },
        {
          "boundingBox": "24,188,292,54",
          "words": [
            {
              "boundingBox": "24,188,292,54",
              "text": "ATOMS"
            }
          ]
        },
        {
          "boundingBox": "22,253,297,32",
          "words": [
            {
              "boundingBox": "22,253,105,32",
              "text": "AND"
            },
            {
              "boundingBox": "144,253,175,32",
              "text": "EMPTY"
            }
          ]
        }
      ]
    }
  ]
}
```

```

    ]
  },
  {
    "boundingBox": "21,298,304,60",
    "words": [
      {
        "boundingBox": "21,298,304,60",
        "text": "SPACE."
      }
    ]
  },
  {
    "boundingBox": "26,387,294,37",
    "words": [
      {
        "boundingBox": "26,387,210,37",
        "text": "Everything"
      },
      {
        "boundingBox": "249,389,71,27",
        "text": "else"
      }
    ]
  },
  {
    "boundingBox": "127,431,198,36",
    "words": [
      {
        "boundingBox": "127,431,31,29",
        "text": "is"
      },
      {
        "boundingBox": "172,431,153,36",
        "text": "opinion."
      }
    ]
  }
]
}
]
}
}

```

Pasos siguientes

Explore una aplicación de JavaScript que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR); crear miniaturas con recorte inteligente; y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para JavaScript](#)

Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision y Node.js

13/01/2020 • 5 minutes to read • [Edit Online](#)

NOTE

Si va a extraer texto en idioma inglés, considere la posibilidad de usar la nueva [operación de lectura](#).

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con el método [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener instalado [Node.js](#) 4.x o una versión posterior.
- Debe tener [npm](#) instalado.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Instale el paquete `request` de npm.
 - a. Abra una ventana del símbolo del sistema como administrador.
 - b. Ejecute el siguiente comando:

```
npm install request
```
 - c. Una vez que el paquete se haya instalado correctamente, cierre la ventana del símbolo del sistema.
2. Copie el código siguiente en un editor de texto.
3. También puede reemplazar el valor de `imageUrl` por la dirección URL de una imagen diferente desde la que desea extraer el texto impreso.
4. Guarde el código como un archivo con una extensión `.js`. Por ejemplo, `get-printed-text.js`.
5. Abra una ventana de símbolo del sistema.
6. En el símbolo del sistema, utilice el comando `node` para ejecutar el archivo. Por ejemplo,

```
node get-printed-text.js
```

```

'use strict';

const request = require('request');

let subscriptionKey = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
if (!subscriptionKey) { throw new Error('Set your environment variables for your subscription key and endpoint.')}

var uriBase = endpoint + 'vision/v2.1/ocr';

const imageUrl = 'https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/' +
  'Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png';

// Request parameters.
const params = {
  'language': 'unk',
  'detectOrientation': 'true',
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + '"' + imageUrl + '"}',
  headers: {
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key' : subscriptionKey
  }
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
  let jsonResponse = JSON.stringify(JSON.parse(body), null, ' ');
  console.log('JSON Response\n');
  console.log(jsonResponse);
});

```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. El ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```

{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        },
        {
          "boundingBox": "27,66,283,52",
          "words": [

```

```

    {
      "boundingBox": "27,66,283,52",
      "text": "EXISTS"
    }
  ],
},
{
  "boundingBox": "27,128,292,49",
  "words": [
    {
      "boundingBox": "27,128,292,49",
      "text": "EXCEPT"
    }
  ]
},
{
  "boundingBox": "24,188,292,54",
  "words": [
    {
      "boundingBox": "24,188,292,54",
      "text": "ATOMS"
    }
  ]
},
{
  "boundingBox": "22,253,297,32",
  "words": [
    {
      "boundingBox": "22,253,105,32",
      "text": "AND"
    },
    {
      "boundingBox": "144,253,175,32",
      "text": "EMPTY"
    }
  ]
},
{
  "boundingBox": "21,298,304,60",
  "words": [
    {
      "boundingBox": "21,298,304,60",
      "text": "SPACE."
    }
  ]
},
{
  "boundingBox": "26,387,294,37",
  "words": [
    {
      "boundingBox": "26,387,210,37",
      "text": "Everything"
    },
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ]
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion "
    }
  ]
}

```

```
    text : opinion.  
  }  
}  
}  
}  
}  
}  
}
```

Limpieza de recursos

Cuando ya no sea necesario, elimine el archivo y después desinstale el paquete `request` de npm. Para desinstalar el paquete, realice los siguientes pasos:

1. Abra una ventana del símbolo del sistema como administrador.
2. Ejecute el siguiente comando:

```
npm uninstall request
```

3. Una vez que el paquete se haya desinstalado correctamente, cierre la ventana del símbolo del sistema.

Pasos siguientes

Explore las versiones de Computer Vision API que se usan para analizar una imagen, detectar celebridades y sitios emblemáticos, crear una miniatura y extraer texto impreso y manuscrito. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Explore Computer Vision API](#)

Inicio rápido: Extracción de texto impreso (OCR) mediante la API REST Computer Vision y Python

13/01/2020 • 5 minutes to read • [Edit Online](#)

NOTE

Si va a extraer texto en idioma inglés, considere la posibilidad de usar la nueva [operación de lectura](#). Hay disponible una [guía de inicio rápido para Python](#).

En este inicio rápido, extraerá texto impreso con el reconocimiento óptico de caracteres (OCR) de una imagen con la API REST de Computer Vision. Con el método [OCR](#), puede detectar texto impreso en cualquier imagen y extraer los caracteres reconocidos en una secuencia de caracteres que pueda usar una máquina.

Puede ejecutar esta guía de inicio rápido paso a paso mediante un cuaderno de Jupyter en [MyBinder](#). Para iniciar Binder, seleccione el botón siguiente:

launch [binder](#)

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Prerequisites

- Debe tener [Python](#) instalado si desea ejecutar el ejemplo localmente.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo

Para crear y ejecutar el ejemplo, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `image_url` por la dirección URL de una imagen diferente desde la que desea extraer el texto impreso.
3. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `get-printed-text.py`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python get-printed-text.py`.

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

ocr_url = endpoint + "vision/v2.1/ocr"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/" + \
    "Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'language': 'unk', 'detectOrientation': 'true'}
data = {'url': image_url}
response = requests.post(ocr_url, headers=headers, params=params, json=data)
response.raise_for_status()

analysis = response.json()

# Extract the word bounding boxes and text.
line_infos = [region["lines"] for region in analysis["regions"]]
word_infos = []
for line in line_infos:
    for word_metadata in line:
        for word_info in word_metadata["words"]:
            word_infos.append(word_info)

word_infos

# Display the image and overlay it with the extracted text.
plt.figure(figsize=(5, 5))
image = Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image, alpha=0.5)
for word in word_infos:
    bbox = [int(num) for num in word["boundingBox"].split(",")]
    text = word["text"]
    origin = (bbox[0], bbox[1])
    patch = Rectangle(origin, bbox[2], bbox[3],
                      fill=False, linewidth=2, color='y')
    ax.axes.add_patch(patch)
    plt.text(origin[0], origin[1], text, fontsize=20, weight="bold", va="top")
plt.axis("off")

```

Carga de la imagen desde el almacenamiento local

Si desea analizar una imagen local, establezca el encabezado Content-Type en application/octet-stream y el cuerpo de la solicitud en una matriz de bytes, en lugar de datos JSON.

```
image_path = "<path-to-local-image-file>"
# Read the image into a byte array
image_data = open(image_path, "rb").read()
# Set Content-Type to octet-stream
headers = {'Ocp-Apim-Subscription-Key': subscription_key, 'Content-Type': 'application/octet-stream'}
# put the byte array into your post request
response = requests.post(ocr_url, headers=headers, params=params, data = image_data)
```

Examen de la respuesta

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```
{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        },
        {
          "boundingBox": "27,66,283,52",
          "words": [
            {
              "boundingBox": "27,66,283,52",
              "text": "EXISTS"
            }
          ]
        },
        {
          "boundingBox": "27,128,292,49",
          "words": [
            {
              "boundingBox": "27,128,292,49",
              "text": "EXCEPT"
            }
          ]
        },
        {
          "boundingBox": "24,188,292,54",
          "words": [
            {
              "boundingBox": "24,188,292,54",
              "text": "ATOMS"
            }
          ]
        },
        {
          "boundingBox": "22,253,297,32",
          "words": [
            {
              "boundingBox": "22,253,105,32",
              "text": "AND"
            }
          ]
        }
      ]
    }
  ]
}
```

```

        "boundingBox": "144,253,175,32",
        "text": "EMPTY"
    }
]
},
{
    "boundingBox": "21,298,304,60",
    "words": [
        {
            "boundingBox": "21,298,304,60",
            "text": "SPACE."
        }
    ]
},
{
    "boundingBox": "26,387,294,37",
    "words": [
        {
            "boundingBox": "26,387,210,37",
            "text": "Everything"
        },
        {
            "boundingBox": "249,389,71,27",
            "text": "else"
        }
    ]
},
{
    "boundingBox": "127,431,198,36",
    "words": [
        {
            "boundingBox": "127,431,31,29",
            "text": "is"
        },
        {
            "boundingBox": "172,431,153,36",
            "text": "opinion."
        }
    ]
}
]
}
]
}
}

```

Pasos siguientes

Explore una aplicación de Python que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para Python](#)

Inicio rápido: Uso de un modelo de dominio mediante la API REST y Python en Computer Vision

13/01/2020 • 7 minutes to read • [Edit Online](#)

En esta guía de inicio rápido, usará un modelo de dominio para identificar puntos de referencia o, si lo desea, celebridades en una imagen almacenada de forma remota mediante la API REST de Computer Vision. Con el método [Reconocer contenido específico de dominio](#), puede aplicar un modelo específico de dominio para reconocer contenido en una imagen.

Puede ejecutar esta guía de inicio rápido paso a paso mediante un cuaderno de Jupyter en [MyBinder](#). Para iniciar Binder, seleccione el botón siguiente:

[launch](#) [binder](#)

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- Debe tener [Python](#) instalado si desea ejecutar el ejemplo localmente.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Después, [cree variables de entorno](#) para la cadena de punto de conexión del servicio y la clave denominadas `COMPUTER_VISION_SUBSCRIPTION_KEY` y `COMPUTER_VISION_ENDPOINT`, respectivamente.

Creación y ejecución del ejemplo de puntos de referencia

Para crear y ejecutar el ejemplo de puntos de referencia, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. También puede reemplazar el valor de `image_url` por la dirección URL de una imagen diferente en la que desea detectar puntos de referencia.
3. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `get-landmarks.py`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python get-landmarks.py`.

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

landmark_analyze_url = endpoint + "vision/v2.1/models/landmarks/analyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/f/f6/" + \
    "Bunker_Hill_Monument_2005.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'model': 'landmarks'}
data = {'url': image_url}
response = requests.post(
    landmark_analyze_url, headers=headers, params=params, json=data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The
# most relevant landmark for the image is obtained from the 'result' property.
analysis = response.json()
assert analysis["result"]["landmarks"] is not []
print(analysis)
landmark_name = analysis["result"]["landmarks"][0]["name"].capitalize()

# Display the image and overlay it with the landmark name.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(landmark_name, size="x-large", y=-0.1)

```

Análisis de la respuesta del ejemplo de puntos de referencia

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:

```
{
  "result": {
    "landmarks": [
      {
        "name": "Bunker Hill Monument",
        "confidence": 0.9768505096435547
      }
    ]
  },
  "requestId": "659a10cd-44bb-44db-9147-a295b853b2b8",
  "metadata": {
    "height": 1600,
    "width": 1200,
    "format": "Jpeg"
  }
}
```

Creación y ejecución del ejemplo de celebridades

Para crear y ejecutar el ejemplo de puntos de referencia, siga estos pasos:

1. Copie el código siguiente en un editor de texto.
2. Realice los siguientes cambios en el código donde sea necesario:
 - a. Reemplace el valor de `subscription_key` por la clave de suscripción.
 - b. Reemplace el valor de `vision_base_url` por la dirección URL del punto de conexión del recurso de Computer Vision en la región de Azure donde obtuvo las claves de suscripción, si es necesario.
 - c. También puede reemplazar el valor de `image_url` por la dirección URL de una imagen diferente en la que desea detectar celebridades.
3. Guarde el código como un archivo con la extensión `.py`. Por ejemplo, `get-celebrities.py`.
4. Abra una ventana de símbolo del sistema.
5. En el símbolo del sistema, utilice el comando `python` para ejecutar el ejemplo. Por ejemplo, `python get-celebrities.py`.

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.1/"

celebrity_analyze_url = vision_base_url + "models/celebrities/analyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/d/d9/" + \
    "Bill_gates_portrait.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'model': 'celebrities'}
data = {'url': image_url}
response = requests.post(
    celebrity_analyze_url, headers=headers, params=params, json=data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The
# most relevant celebrity for the image is obtained from the 'result' property.
analysis = response.json()
assert analysis["result"]["celebrities"] is not []
print(analysis)
celebrity_name = analysis["result"]["celebrities"][0]["name"].capitalize()

# Display the image and overlay it with the celebrity name.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(celebrity_name, size="x-large", y=-0.1)

```

Análisis de la respuesta del ejemplo de celebridades

Se devuelve una respuesta correcta en JSON. La página web de ejemplo analiza y muestra una respuesta correcta en la ventana del símbolo del sistema, parecida a la del ejemplo siguiente:


```
{
  "result": {
    "celebrities": [
      {
        "faceRectangle": {
          "top": 123,
          "left": 156,
          "width": 187,
          "height": 187
        },
        "name": "Bill Gates",
        "confidence": 0.9993845224380493
      }
    ]
  },
  "requestId": "f14ec1d0-62d4-4296-9ceb-6b5776dc2020",
  "metadata": {
    "height": 521,
    "width": 550,
    "format": "Jpeg"
  }
}
```

Limpieza de recursos

Cuando ya no los necesite, elimine los archivos de los dos ejemplos.

Pasos siguientes

Explore una aplicación de Python que usa Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas con recorte inteligente, y detectar, clasificar, etiquetar y describir características visuales, como caras, en una imagen. Para experimentar rápidamente con la versión de Computer Vision API, pruebe la [consola de pruebas de Open API](#).

[Tutorial de Computer Vision API para Python](#)

Tutorial: Uso de Computer Vision para generar metadatos de imágenes en Azure Storage

13/01/2020 • 12 minutes to read • [Edit Online](#)

En este tutorial, aprenderá a integrar el servicio Azure Computer Vision en una aplicación web para generar metadatos para las imágenes cargadas. Esto resulta útil para los escenarios de [administración de activos digitales \(DAM\)](#); por ejemplo, si una empresa quiere generar rápidamente leyendas descriptivas o palabras clave de búsqueda para todas sus imágenes.

En [Laboratorio de Azure Storage y Cognitive Services](#), de GitHub, se puede encontrar una guía de aplicaciones completa y en este tutorial se trata esencialmente el ejercicio 5 de dicho laboratorio. Puede que desee crear la aplicación completa siguiendo cada paso, pero si solo desea obtener información sobre cómo integrar Computer Vision en una aplicación web existente, aquí puede leerlo.

En este tutorial se muestra cómo realizar las siguientes acciones:

- Crear un recurso de Computer Vision en Azure
- Realizar análisis de imágenes en imágenes de Azure Storage
- Adjuntar metadatos a imágenes de Azure Storage
- Comprobar metadatos de imágenes desde el Explorador de Azure Storage

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

- [Visual Studio 2017 Community Edition](#), o cualquier versión superior, con las cargas de trabajo "Desarrollo de ASP.NET y web" y "Desarrollo de Azure" instaladas.
- Una cuenta de Azure Storage con un contenedor de blobs configurado para el almacenamiento de imágenes (siga el [ejercicio 1 del laboratorio de Azure Storage](#) si necesita ayuda con este paso).
- La herramienta Explorador de Azure Storage (siga el [ejercicio 2 del laboratorio de Azure Storage](#) si necesita ayuda con este paso).
- Una aplicación web ASP.NET con acceso a Azure Storage (siga el [ejercicio 3 del laboratorio de Azure Storage](#) para crear la aplicación rápidamente).

Creación de un recurso de Computer Vision

Deberá crear un recurso de Computer Vision para su cuenta de Azure; este recurso administra el acceso al servicio Computer Vision de Azure.

1. Siga las instrucciones que se indican en [Creación de un recurso de Azure Cognitive Services](#) para crear un recurso de Computer Vision.
2. Luego, vuelva al menú del grupo de recursos y haga clic en la suscripción de Computer Vision API que acaba de crear. Copie la dirección URL de **Punto de conexión** a cualquier lugar en que pueda recuperarla fácilmente en un momento. Luego, haga clic en **Show access keys** (Mostrar claves de acceso).

Delete

Essentials

Resource group (change)	API type
IntellipixResources	Computer Vision API
Status	Pricing tier
Active	Free
Location	Endpoint
South Central US	https://southcentralus.api.cognitive.microsoft.com/
Subscription name (change)	Manage keys
Subscription ID	Show access keys ...

NOTE

Los nuevos recursos creados después del 1 de julio de 2019 usarán nombres de subdominio personalizados. Para más información y para obtener una lista completa de los puntos de conexión regionales, consulte [Nombres de subdominios personalizados para Cognitive Services](#).

- En la ventana siguiente, copie el valor de **KEY 1** en el Portapapeles.

Manage keys
vision-api-key

Regenerate Key1
Regenerate Key2

Notice: It may take up to 10 minutes for the newly (re)generated keys to take effect.

NAME
vision-api-key

KEY 1

KEY 2

Adición de credenciales de Computer Vision

A continuación, agregará las credenciales necesarias a la aplicación para que pueda acceder a los recursos de Computer Vision.

Abra la aplicación web ASP.NET en Visual Studio y vaya al archivo **Web.config**, que se encuentra en la raíz del proyecto. Agregue las siguientes instrucciones a la sección `<appSettings>` del archivo, pero reemplace `VISION_KEY` por la clave que copió en el paso anterior y `VISION_ENDPOINT` por la dirección URL que guardó en el paso anterior.

```
<add key="SubscriptionKey" value="VISION_KEY" />
<add key="VisionEndpoint" value="VISION_ENDPOINT" />
```

Luego, en el Explorador de soluciones, haga clic con el botón derecho en el proyecto y use el comando **Administrar paquetes NuGet** para instalar el paquete **Microsoft.Azure.CognitiveServices.Vision.ComputerVision**. Dicho paquete contiene los tipos necesarios para llamar a Computer Vision API.

Incorporación del código de generación de metadatos

A continuación, agregará el código que realmente saca provecho del servicio Computer Vision para crear metadatos para las imágenes. Estos pasos se aplicarán a la aplicación ASP.NET del laboratorio, pero puede adaptarlos a su propia aplicación. Lo importante es que tenga una aplicación web ASP.NET que puede cargar imágenes en un contenedor de Azure Storage, leer imágenes del mismo y mostrarlas en la vista. Si no está seguro de este paso, es mejor que siga el [ejercicio 3 del laboratorio de Azure Storage](#).

1. Abra el archivo *HomeController.cs*, que encontrará en la carpeta **Controllers** del proyecto, y agregue las siguientes `using` instrucciones al principio del mismo:

```
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;
```

2. A continuación, vaya al método **Upload**; este método convierte y carga imágenes en Blob Storage. Agregue el código siguiente inmediatamente después del bloque que comienza por `// Generate a thumbnail` (o al final del proceso de creación de blobs de imágenes). Este código toma el blob que contiene la imagen (`photo`) y utiliza Computer Vision para generar una descripción de ella. Computer Vision API también genera una lista de palabras clave que se aplican a la imagen. Tanto la descripción como las palabras clave generadas se almacenan en los metadatos del blob para que se puedan recuperar más adelante.

```
// Submit the image to Azure's Computer Vision API
ComputerVisionClient vision = new ComputerVisionClient(
    new ApiKeyServiceClientCredentials(ConfigurationManager.AppSettings["SubscriptionKey"]),
    new System.Net.Http.DelegatingHandler[] { });
vision.Endpoint = ConfigurationManager.AppSettings["VisionEndpoint"];

VisualFeatureTypes[] features = new VisualFeatureTypes[] { VisualFeatureTypes.Description };
var result = await vision.AnalyzeImageAsync(photo.Uri.ToString(), features);

// Record the image description and tags in blob metadata
photo.Metadata.Add("Caption", result.Description.Captions[0].Text);

for (int i = 0; i < result.Description.Tags.Count; i++)
{
    string key = String.Format("Tag{0}", i);
    photo.Metadata.Add(key, result.Description.Tags[i]);
}

await photo.SetMetadataAsync();
```

3. A continuación, vaya al método **Index** en el mismo archivo. Este método enumera los blobs de imágenes almacenados en el contenedor de blobs de destino (como instancias de **IListBlobItem**) y los pasa a la vista de aplicación. Reemplace el bloque `foreach` de este método por el siguiente código. Este código llama a **CloudBlockBlob.FetchAttributes** para obtener los metadatos adjuntos de cada blob. Extrae la descripción generada por el equipo (`caption`) de los metadatos y la agrega al objeto **BlobInfo**, que se pasa a la vista.

```

foreach (IListBlobItem item in container.ListBlobs())
{
    var blob = item as CloudBlockBlob;

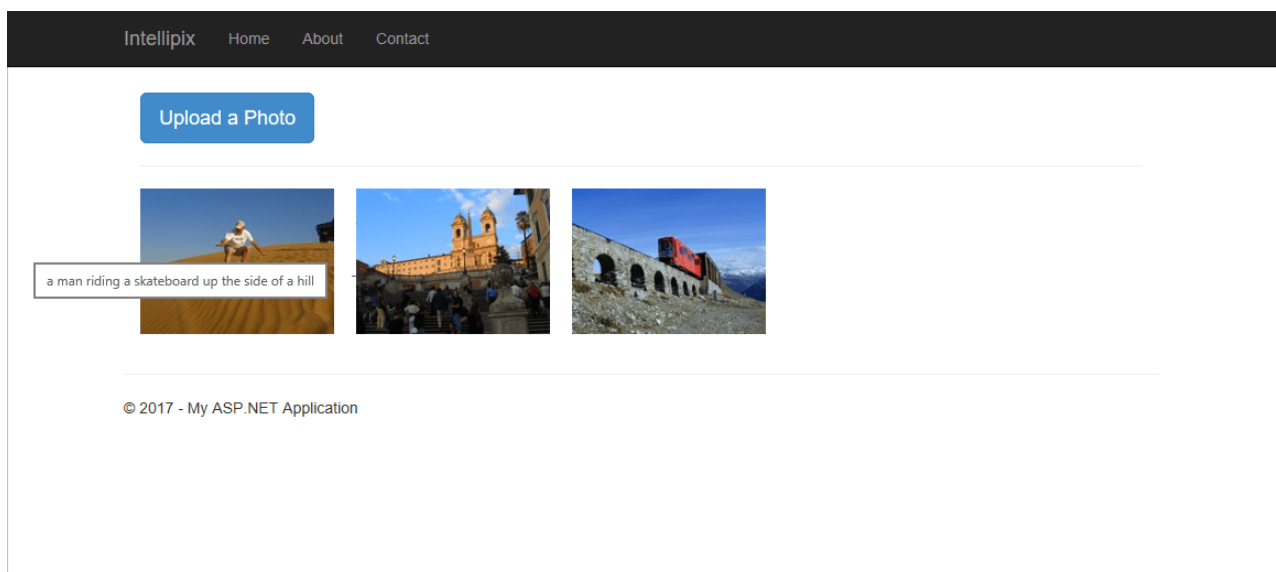
    if (blob != null)
    {
        blob.FetchAttributes(); // Get blob metadata
        var caption = blob.Metadata.ContainsKey("Caption") ? blob.Metadata["Caption"] : blob.Name;

        blobs.Add(new BlobInfo()
        {
            ImageUri = blob.Uri.ToString(),
            ThumbnailUri = blob.Uri.ToString().Replace("/photos/", "/thumbnails/"),
            Caption = caption
        });
    }
}

```

Prueba de la aplicación

Guarde los cambios en Visual Studio y presione **Ctrl+F5** para iniciar la aplicación en el explorador. Use la aplicación para cargar algunas imágenes desde la carpeta "photos" de los recursos del laboratorio o desde su propia carpeta. Cuando mueva el puntero sobre cualquiera de las imágenes en la vista, debería aparecer una ventana de información sobre herramientas que debe aparecer y mostrar la leyenda que ha generado el equipo de la imagen.



Para ver todos los metadatos adjuntos, use el Explorador de Azure Storage para ver el contenedor de almacenamiento que usa para las imágenes. Haga clic con el botón derecho en cualquiera de los blobs del mismo y seleccione **Propiedades**. En el cuadro de diálogo, verá una lista de pares clave-valor. La descripción de la imagen generada por el equipo se almacena en el elemento "Caption" y las palabras clave de búsqueda se almacenan en "Tag0", "Tag1", y así sucesivamente. Cuando haya terminado, haga clic en **Cancelar** para cerrar el cuadro de diálogo.

Properties

Name	Dubai.jpg
Container	photos
Etag	"0x8D515A95F092484"
LastModified	Tue, 17 Oct 2017 21:52:33 GMT
Size	1980812
BlobType	BlockBlob
LeaseState	available

Metadata

Caption	a man riding a skateboard up the side of a hill	✕
Tag0	outdoor	✕
Tag1	nature	✕
Tag2	man	✕
Tag3		✕

Add metadata

Save

Cancel

Limpieza de recursos

Si desea seguir trabajando en la aplicación web, consulte la sección [Pasos siguientes](#). Si no planea seguir usando esta aplicación, debe eliminar todos los recursos específicos de la misma. Para ello, puede eliminar el grupo de recursos que contiene la suscripción de Azure Storage y el recurso de Computer Vision. De esta forma se quita la cuenta de almacenamiento, los blobs que se han cargado en ella y el recurso de App Service necesario para conectarse con la aplicación web ASP.NET.

Para eliminar el grupo de recursos, abra la pestaña **Grupos de recursos** del portal, vaya al grupo de recursos que usó para el proyecto y haga clic en **Eliminar grupo de recursos** en la parte superior de la vista. Se le pedirá que escriba el nombre del grupo de recursos para confirmar que desea eliminarlo, porque una vez eliminados, los grupos de recursos no se pueden recuperar.

Pasos siguientes

En este tutorial, ha configurado el servicio Computer Vision de Azure en una aplicación web existente para que se generen automáticamente leyendas y palabras clave para las imágenes de los blobs en el momento en que se cargan. Luego, consulte el ejercicio 6 del laboratorio de Azure Storage, para aprender a agregar la funcionalidad de búsqueda a una aplicación web. Así se saca provecho de las palabras clave de búsqueda que genera el servicio Computer Vision.

[Agregar un cuadro de búsqueda a la aplicación](#)

Aplicación de etiquetas de contenido a imágenes

13/01/2020 • 2 minutes to read • [Edit Online](#)

Computer Vision devuelve etiquetas en función de miles de objetos, seres vivos, paisajes y acciones reconocibles. Cuando las etiquetas son ambiguas o no muy comunes, la respuesta de la API contiene «indicaciones» para aclarar el significado de la etiqueta en el contexto de un entorno conocido. Las etiquetas no están organizadas por su taxonomía y no existen jerarquías de herencia. Una colección de etiquetas de contenido es la base de la "descripción" de una imagen que se muestra en lenguaje natural con formato de oraciones completas. Tenga en cuenta que, en este momento, el inglés es el único idioma admitido para la descripción de la imagen.

Después de cargar una imagen o especificar la URL de una imagen, los algoritmos de Computer Vision devuelven como salida etiquetas basadas en los objetos, seres vivos, paisajes y acciones identificados en la imagen. El etiquetado no se limita al sujeto principal, como una persona en primer plano, sino que también incluye el entorno (interior o exterior), muebles, herramientas, plantas, animales, accesorios, gadgets, etc.

Ejemplo de etiquetado de imágenes

La siguiente respuesta JSON muestra lo que devuelve Computer Vision al etiquetar las características visuales detectadas en la imagen de ejemplo.



```

{
  "tags": [
    {
      "name": "grass",
      "confidence": 0.9999995231628418
    },
    {
      "name": "outdoor",
      "confidence": 0.99992108345031738
    },
    {
      "name": "house",
      "confidence": 0.99685388803482056
    },
    {
      "name": "sky",
      "confidence": 0.99532157182693481
    },
    {
      "name": "building",
      "confidence": 0.99436837434768677
    },
    {
      "name": "tree",
      "confidence": 0.98880356550216675
    },
    {
      "name": "lawn",
      "confidence": 0.788884699344635
    },
    {
      "name": "green",
      "confidence": 0.71250593662261963
    },
    {
      "name": "residential",
      "confidence": 0.70859086513519287
    },
    {
      "name": "grassy",
      "confidence": 0.46624681353569031
    }
  ],
  "requestId": "06f39352-e445-42dc-96fb-0a1288ad9cf1",
  "metadata": {
    "height": 200,
    "width": 300,
    "format": "Jpeg"
  }
}

```

Pasos siguientes

Conozca los conceptos de [categorización de imágenes](#) y de [descripción de imágenes](#).

Detección de objetos comunes en imágenes

13/01/2020 • 3 minutes to read • [Edit Online](#)

La detección de objetos es similar al [etiquetado](#), pero la API devuelve las coordenadas del rectángulo (en píxeles) que delimita cada objeto encontrado. Por ejemplo, si una imagen contiene un perro, un gato y una persona, la operación de detección mostrará estos objetos junto con sus coordenadas en la imagen. Puede usar esta funcionalidad para procesar las relaciones entre los objetos de una imagen. También le permite saber si hay varias instancias de la misma etiqueta en una imagen.

La API de detección aplica etiquetas en función de los objetos o seres vivos que se identifiquen en la imagen. Actualmente no hay ninguna relación formal entre la taxonomía de etiquetado y la taxonomía de detección de objetos. Desde un punto de vista conceptual, la API de detección solo busca objetos y seres vivos, mientras que la API de etiquetado también puede incluir términos contextuales, como "interior", que no pueden localizarse con rectángulos de selección.

Ejemplo de detección de objetos

En la siguiente respuesta JSON, se ilustra qué devuelve Computer Vision cuando se detectan los objetos de la imagen de ejemplo.



```

{
  "objects":[
    {
      "rectangle":{
        "x":730,
        "y":66,
        "w":135,
        "h":85
      },
      "object":"kitchen appliance",
      "confidence":0.501
    },
    {
      "rectangle":{
        "x":523,
        "y":377,
        "w":185,
        "h":46
      },
      "object":"computer keyboard",
      "confidence":0.51
    },
    {
      "rectangle":{
        "x":471,
        "y":218,
        "w":289,
        "h":226
      },
      "object":"Laptop",
      "confidence":0.85,
      "parent":{
        "object":"computer",
        "confidence":0.851
      }
    },
    {
      "rectangle":{
        "x":654,
        "y":0,
        "w":584,
        "h":473
      },
      "object":"person",
      "confidence":0.855
    }
  ],
  "requestId":"a7fde8fd-cc18-4f5f-99d3-897dcd07b308",
  "metadata":{
    "width":1260,
    "height":473,
    "format":"Jpeg"
  }
}

```

Limitaciones

Es importante tener en cuenta las limitaciones de la detección de objetos para que pueda evitar o mitigar los efectos de los falsos negativos (objetos que faltan) y los detalles limitados.

- Por lo general, los objetos no se detectan si son muy pequeños (menores del 5 % de la imagen).
- Los objetos no se suelen detectar si están cerca (en una pila de platos, por ejemplo).
- Los objetos no se diferencian por la marca ni los nombres de productos (tipos diferentes de los refrescos en una estantería de un almacén, por ejemplo). Sin embargo, puede obtener información de la marca de una

imagen mediante la característica [de detección de la marca](#).

Uso de la API

La característica de detección de objetos forma parte de la API [de análisis de imágenes](#). Puede llamar a esta API mediante una SDK nativa o con llamadas a REST. Incluya `objects` en el parámetro de consulta **visualFeatures**. Después, cuando llegue la respuesta JSON completa, simplemente analice la cadena con el contenido de la sección `"objects"`.

- [Inicio rápido: SDK de Computer Vision para .NET](#)
- [Inicio rápido: Análisis de imágenes \(API REST\)](#)

Detección de marcas populares en las imágenes

13/01/2020 • 2 minutes to read • [Edit Online](#)

La detección de la marca es un modo especializado de [detección de objetos](#) que usa una base de datos de miles de logotipos mundiales para identificar las marcas comerciales en imágenes o vídeos. Puede usar esta característica, por ejemplo, para detectar qué marcas son más populares en medios sociales o más frecuentes en la ubicación de los productos multimedia.

El servicio Computer Vision detecta si existen logotipos de marca en una imagen determinada; si es así, devuelve el nombre de la marca, una puntuación de confianza y las coordenadas del rectángulo delimitador del logotipo.

La base de datos de logotipos integrada cubre marcas populares de electrónica de consumo, vestimenta y mucho más. Si encuentra que el servicio Computer Vision no detecta la marca que está buscando, puede ser conveniente crear y entrenar su propio detector de logotipos con el servicio [Custom Vision](#).

Ejemplo de detección de marcas

En la siguiente respuesta JSON, se ilustra qué devuelve Computer Vision cuando se detectan marcas de las imágenes de ejemplo.



```
"brands": [
  {
    "name": "Microsoft",
    "rectangle": {
      "x": 20,
      "y": 97,
      "w": 62,
      "h": 52
    }
  }
]
```

En algunos casos, el detector de marcas captará la imagen del logotipo y el nombre estilizado de la marca como dos logotipos independientes.



```
"brands":[
  {
    "name":"Microsoft",
    "rectangle":{
      "x":58,
      "y":106,
      "w":55,
      "h":46
    }
  },
  {
    "name":"Microsoft",
    "rectangle":{
      "x":58,
      "y":86,
      "w":202,
      "h":63
    }
  }
]
```

Uso de la API

La característica de detección de marcas forma parte de la API [Analyze Image](#). Puede llamar a esta API mediante una SDK nativa o con llamadas a REST. Incluya `Brands` en el parámetro de consulta **visualFeatures**. Después, cuando llegue la respuesta JSON completa, simplemente analice la cadena con el contenido de la sección `"brands"`.

- [Inicio rápido: SDK de Computer Vision para .NET](#)
- [Inicio rápido: Análisis de imágenes \(API REST\)](#)

Categorización de las imágenes por materia

13/01/2020 • 2 minutes to read • [Edit Online](#)

Además de etiquetas y descripciones, Computer Vision devuelve las categorías basadas en la taxonomía que se detectan en una imagen. A diferencia de las etiquetas, las categorías se organizan en una jerarquía hereditaria de elementos primarios y secundarios, y son menos numerosas (86 en lugar de miles de etiquetas). Todos los nombres de categorías están en inglés. La categorización puede realizarse en solitario o junto con un nuevo modelo de etiquetas.

El concepto de las 86 categorías

Computer Vision puede categorizar una imagen de manera general o específica utilizando las 86 categorías del diagrama siguiente. Consulte la taxonomía completa en formato de texto en [Taxonomía de las categorías](#).



Ejemplos de categorización de imágenes

La siguiente respuesta JSON muestra lo que devuelve Computer Vision al categorizar la imagen de ejemplo según sus características visuales.



```
{
  "categories": [
    {
      "name": "people_",
      "score": 0.81640625
    }
  ],
  "requestId": "bae7f76a-1cc7-4479-8d29-48a694974705",
  "metadata": {
    "height": 200,
    "width": 300,
    "format": "Jpeg"
  }
}
```

En la tabla siguiente se muestra un conjunto típico de imágenes y la categoría devuelta por Computer Vision para cada imagen.





IMAGEN	CATEGORÍA
	people_group
	animal_dog
	outdoor_mountain

IMAGEN	CATEGORÍA
	food_bread

Pasos siguientes

Conozca los conceptos de [etiquetado de imágenes](#) y de [descripción de imágenes](#).

Descripción de imágenes con lenguaje natural

13/01/2020 • 2 minutes to read • [Edit Online](#)

Computer Vision puede analizar una imagen y generar una frase inteligible que describa su contenido. El algoritmo realmente devuelve varias descripciones según diferentes características visuales y cada descripción tiene una puntuación de confianza. El resultado final es una lista de descripciones ordenadas de mayor a menor confianza.

Ejemplo de descripción de imagen

La siguiente respuesta JSON muestra lo que devuelve Computer Vision al describir la imagen de ejemplo según sus características visuales.



```
{
  "description": {
    "tags": ["outdoor", "building", "photo", "city", "white", "black", "large", "sitting", "old",
"water", "skyscraper", "many", "boat", "river", "group", "street", "people", "field", "tall", "bird",
"standing"],
    "captions": [
      {
        "text": "a black and white photo of a city",
        "confidence": 0.95301952483304808
      },
      {
        "text": "a black and white photo of a large city",
        "confidence": 0.94085190563213816
      },
      {
        "text": "a large white building in a city",
        "confidence": 0.93108362931954824
      }
    ]
  },
  "requestId": "b20bfc83-fb25-4b8d-a3f8-b2a1f084b159",
  "metadata": {
    "height": 300,
    "width": 239,
    "format": "Jpeg"
  }
}
```

Pasos siguientes

Conozca los conceptos de [etiquetado de imágenes](#) y de [categorización de imágenes](#).

Detección de caras con Computer Vision

13/01/2020 • 2 minutes to read • [Edit Online](#)

Computer Vision puede detectar caras humanas en una imagen y generar la edad, el sexo y el rectángulo de cada rostro detectado.

NOTE

Esta característica también está disponible en el servicio Azure [Face](#). Utilice esta alternativa para realizar un análisis más detallado de los rostros que incluya la identificación de la cara y la detección de la pose.

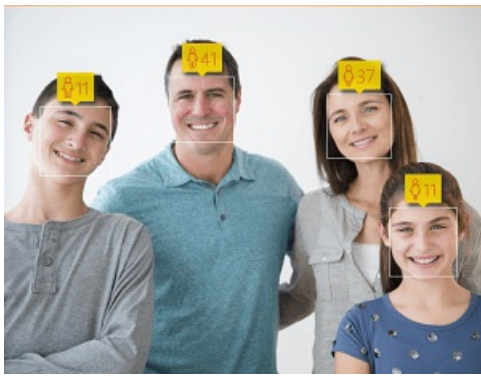
Ejemplos de detección de caras

En el ejemplo siguiente, se muestra la respuesta JSON devuelta por Computer Vision en una imagen con un solo rostro humano.



```
{
  "faces": [
    {
      "age": 23,
      "gender": "Female",
      "faceRectangle": {
        "top": 45,
        "left": 194,
        "width": 44,
        "height": 44
      }
    }
  ],
  "requestId": "8439ba87-de65-441b-a0f1-c85913157ecd",
  "metadata": {
    "height": 200,
    "width": 300,
    "format": "Png"
  }
}
```

En el ejemplo siguiente, se muestra la respuesta JSON devuelta en una imagen con varios rostros humanos.



```
{
  "faces": [
    {
      "age": 11,
      "gender": "Male",
      "faceRectangle": {
        "top": 62,
        "left": 22,
        "width": 45,
        "height": 45
      }
    },
    {
      "age": 11,
      "gender": "Female",
      "faceRectangle": {
        "top": 127,
        "left": 240,
        "width": 42,
        "height": 42
      }
    },
    {
      "age": 37,
      "gender": "Female",
      "faceRectangle": {
        "top": 55,
        "left": 200,
        "width": 41,
        "height": 41
      }
    },
    {
      "age": 41,
      "gender": "Male",
      "faceRectangle": {
        "top": 45,
        "left": 103,
        "width": 39,
        "height": 39
      }
    }
  ],
  "requestId": "3a383cbe-1a05-4104-9ce7-1b5cf352b239",
  "metadata": {
    "height": 230,
    "width": 300,
    "format": "Png"
  }
}
```

Pasos siguientes

Para más información sobre el uso de la característica de detección de rostros, consulte la documentación de referencia de [Analyze Image](#) (Análisis de imágenes).

Detección de tipos de imagen con Computer Vision

13/01/2020 • 2 minutes to read • [Edit Online](#)

Con la API [Analyze Image](#), Computer Vision puede analizar el tipo de contenido de las imágenes, que indica si se trata de una imagen prediseñada o un dibujo lineal.

Detección de imágenes prediseñadas

Computer Vision analiza una imagen y evalúa la probabilidad de que sea una imagen prediseñada en una escala de 0 a 3, tal y como se describe en la tabla siguiente.

VALOR	SIGNIFICADO
0	No es imagen prediseñada
1	Ambigua
2	Imagen prediseñada normal
3	Imagen prediseñada buena

Ejemplos de detección de imágenes prediseñadas

Las respuestas JSON siguientes muestran lo que devuelve Computer Vision cuando evalúa la probabilidad de que las imágenes de ejemplo sean imágenes prediseñadas.



```
{
  "imageType": {
    "clipArtType": 3,
    "lineDrawingType": 0
  },
  "requestId": "88c48d8c-80f3-449f-878f-6947f3b35a27",
  "metadata": {
    "height": 225,
    "width": 300,
    "format": "Jpeg"
  }
}
```



```
{
  "imageType": {
    "clipArtType": 0,
    "lineDrawingType": 0
  },
  "requestId": "a9c8490a-2740-4e04-923b-e8f4830d0e47",
  "metadata": {
    "height": 200,
    "width": 300,
    "format": "Jpeg"
  }
}
```

Detección de dibujos lineales

Computer Vision analiza una imagen y devuelve un valor booleano que indica si la imagen es un dibujo lineal.

Ejemplos de detección de dibujos lineales

Las respuestas JSON siguientes muestran lo que devuelve Computer Vision al informar si las imágenes de ejemplo son dibujos lineales.



```
{
  "imageType": {
    "clipArtType": 2,
    "lineDrawingType": 1
  },
  "requestId": "6442dc22-476a-41c4-aa3d-9ceb15172f01",
  "metadata": {
    "height": 268,
    "width": 300,
    "format": "Jpeg"
  }
}
```



```
{
  "imageType": {
    "clipArtType": 0,
    "lineDrawingType": 0
  },
  "requestId": "98437d65-1b05-4ab7-b439-7098b5dfdcbf",
  "metadata": {
    "height": 200,
    "width": 300,
    "format": "Jpeg"
  }
}
```

Pasos siguientes

Consulte la documentación de referencia sobre el [análisis de imagen](#) para obtener información sobre cómo detectar los tipos de imagen.

Detectar contenido específico del dominio

13/01/2020 • 3 minutes to read • [Edit Online](#)

Además de la categorización de alto nivel y el etiquetado, Computer Vision también admite un análisis más refinado y específico del dominio con modelos que se hayan entrenado en datos especializados.

Hay dos maneras de utilizar los modelos específicos del dominio: solos (análisis con ámbito) o como una mejora de la característica de categorización.

Análisis con ámbito

Puede analizar una imagen usando solo el modelo específico de dominio elegido mediante una llamada a la API [Models/<model>/Analyze](#).

El siguiente es una muestra de respuesta JSON devuelta por la API **models/celebrities/analyze** para la imagen especificada:



```
{
  "result": {
    "celebrities": [{
      "faceRectangle": {
        "top": 391,
        "left": 318,
        "width": 184,
        "height": 184
      },
      "name": "Satya Nadella",
      "confidence": 0.99999856948852539
    }]
  },
  "requestId": "8217262a-1a90-4498-a242-68376a4b956b",
  "metadata": {
    "width": 800,
    "height": 1200,
    "format": "Jpeg"
  }
}
```

Análisis de categorización mejorada

También puede usar modelos específicos del dominio para complementar los análisis de imágenes generales. Esto forma parte de la [categorización de alto nivel](#) mediante la especificación de los modelos específicos del dominio en la llamada al parámetro *details* de la llamada a la API [Analyze](#).

En este caso, se llama primero al clasificador de la taxonomía de las 86 categorías. Si alguna de las categorías detectadas tiene un modelo específico de dominio coincidente, la imagen se pasa por ese modelo y se agregan los resultados.

La siguiente respuesta JSON muestra el modo en que el análisis específico del dominio puede incluirse como el nodo `detail` en un análisis de categorización más amplio.

```
"categories":[
  {
    "name":"abstract_",
    "score":0.00390625
  },
  {
    "name":"people_",
    "score":0.83984375,
    "detail":{"
      "celebrities":[
        {
          "name":"Satya Nadella",
          "faceRectangle":{"
            "left":597,
            "top":162,
            "width":248,
            "height":248
          },
          "confidence":0.999028444
        }
      ],
      "landmarks":[
        {
          "name":"Forbidden City",
          "confidence":0.9978346
        }
      ]
    }
  }
]
```

Enumeración de modelos específicos de dominio

Actualmente, Computer Vision admite los siguientes modelos específicos de dominio:

NOMBRE	DESCRIPCIÓN
celebrities	Reconocimiento de celebridades, compatible con imágenes clasificadas en la categoría <code>people_</code> .
landmarks	Reconocimiento de puntos de referencia, compatible con imágenes clasificadas en las categorías <code>outdoor_</code> o <code>building_</code> .

Una llamada a la API [Models](#) devolverá esta información junto con las categorías a las que se puede aplicar cada modelo:

```
{
  "models":[
    {
      "name":"celebrities",
      "categories":[
        "people_",
        "人_",
        "pessoas_",
        "gente_"
      ]
    },
    {
      "name":"landmarks",
      "categories":[
        "outdoor_",
        "户外_",
        "屋外_",
        "aoarlivre_",
        "alairelibre_",
        "building_",
        "建筑_",
        "建物_",
        "edifício_"
      ]
    }
  ]
}
```

Pasos siguientes

Conozca los conceptos de [categorización de imágenes](#).

Detección de las combinaciones de colores de las imágenes

13/01/2020 • 3 minutes to read • [Edit Online](#)

Computer Vision analiza los colores de una imagen para proporcionar tres atributos diferentes: el color de primer plano predominante, el color de fondo predominante y el conjunto de colores predominantes de la imagen como un todo. Los colores devueltos pertenecen al conjunto negro, azul, marrón, gris, verde, naranja, rosa, púrpura, rojo, verde azulado, blanco y amarillo.

Computer Vision extrae también un color de énfasis, que representa el color más brillantes de la imagen, basándose en una combinación de colores dominantes y en la saturación. El color de énfasis se devuelve como un código de color HTML hexadecimal.

Computer Vision también devuelve un valor booleano que indica si una imagen está en blanco y negro.

Ejemplos de detección de la combinación de colores



El ejemplo siguiente muestra la respuesta JSON devuelta por Computer Vision durante la detección de la combinación de colores de la imagen de ejemplo. En este caso, la imagen de ejemplo no es una imagen en blanco y negro, pero el color predominante de primer plano y de fondo es el negro, y los colores predominantes de la imagen como un todo son el blanco y el negro.



```
{
  "color": {
    "dominantColorForeground": "Black",
    "dominantColorBackground": "Black",
    "dominantColors": ["Black", "White"],
    "accentColor": "BB6D10",
    "isBwImg": false
  },
  "requestId": "0dc394bf-db50-4871-bdcc-13707d9405ea",
  "metadata": {
    "height": 202,
    "width": 300,
    "format": "Jpeg"
  }
}
```

Ejemplos de color predominante

En la tabla siguiente se muestra el color de primer plano, el color de fondo y los colores de la imagen devueltos para cada imagen de ejemplo.

IMAGEN	COLORES PREDOMINANTES
	Primer plano: Negro Fondo: Blanco Colores: negro, blanco y verde
	Primer plano: Negro Fondo: Negro Colores: Negro

Ejemplos de color de énfasis

En la tabla siguiente se muestran los colores de énfasis devueltos, como un valor hexadecimal de HTML, para cada imagen de ejemplo.






IMAGEN	COLOR DE ÉNFASIS
	#BB6D10
	#C6A205

IMAGEN	COLOR DE ÉNFASIS
	#474A84

Ejemplos de detección en blanco y negro

La siguiente tabla muestra la evaluación en blanco y negro de Computer Vision en las imágenes de ejemplo.

IMAGEN	¿BLANCO Y NEGRO?
	true
	false

Pasos siguientes

Conozca los conceptos sobre [detectar tipos de imagen](#).

Generación de miniaturas de recorte inteligente con Computer Vision

13/01/2020 • 2 minutes to read • [Edit Online](#)

Una miniatura es una representación reducida de una imagen. Las miniaturas se utilizan para representar imágenes y otros datos de un modo más económico y más compatible con el diseño. Computer Vision API utiliza la funcionalidad de recorte inteligente junto con el cambio de tamaño para crear miniaturas intuitivas de una determinada imagen.

El algoritmo de generación de miniaturas de Computer Vision funciona del modo siguiente:

1. Quita los elementos prescindibles de la imagen e identifica el *área de interés*—; es decir, el área de la imagen en la que aparecen los objetos principales.
2. Recorta la imagen en función del *área de interés* identificada.
3. Cambia la relación de aspecto para adaptarla a las dimensiones de la miniatura de destino.

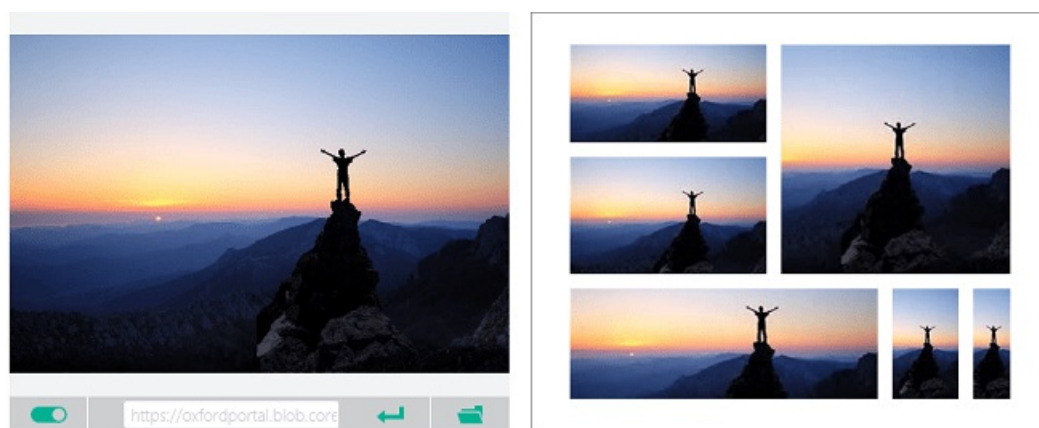
Área de interés

Cuando se carga una imagen, Computer Vision API la analiza para determinar el *área de interés*. También puede usar esta región para determinar cómo debe recortarse la imagen. Sin embargo, la operación de recorte siempre se adaptará a la relación de aspecto deseada, si se especifica una.







También puede obtener las coordenadas del rectángulo de selección de esta misma *área de interés* llamando a **areaOfInterest** API en su lugar. Puede utilizar esta información para modificar la imagen original como desee.

Ejemplos

La miniatura generada puede variar enormemente en función de la altura, anchura y la función de recorte inteligente que se especifiquen, tal como se muestra en la siguiente imagen.



En la tabla siguiente se muestran las típicas miniaturas que se pueden generar con Computer Vision, teniendo en cuenta las imágenes de ejemplo. Las miniaturas se crearon para un destino con una altura y anchura específicas de 50 píxeles, y tienen la función de recorte inteligente habilitada.

IMAGEN	MINIATURA
	
	
	

Pasos siguientes

Mas información sobre el [etiquetado de imágenes](#) y la [categorización de imágenes](#).

Reconocimiento de texto manuscrito e impreso

13/01/2020 • 7 minutes to read • [Edit Online](#)

Computer Vision proporciona varios servicios que detectan y extraen texto manuscrito o impreso que aparece en las imágenes. Esto es útil en una variedad de escenarios, como la toma de notas, los registros médicos, seguridad y banca. Las tres secciones siguientes detallan tres API de reconocimiento de texto diferentes, cada una optimizada para distintos casos de uso.

Read API

Read API detecta el contenido de texto de una imagen con los últimos modelos de reconocimiento y convierte el texto identificado en una secuencia de caracteres legible por una máquina. Está optimizada para imágenes con gran cantidad de texto (por ejemplo, documentos que han sido digitalizados) y para imágenes con mucho ruido visual. Determinará qué modelo de reconocimiento se usará para cada línea de texto y admite imágenes con texto escrito a mano e impreso. Read API se ejecuta de forma asíncrona porque los documentos más grandes pueden tardar varios minutos en devolver un resultado.

La operación de lectura mantiene las agrupaciones de líneas originales de palabras reconocidas en la salida. Cada línea incluye las coordenadas del rectángulo delimitador y cada palabra dentro de la línea también tiene sus propias coordenadas. Si una palabra se ha reconocido con confianza baja, dicha información se muestra también. Consulte los [documentos de referencia de Read API](#) para más información.

NOTE

Esta característica solo está disponible para texto en inglés.

Requisitos de imagen

Read API funciona con imágenes que cumplan los requisitos siguientes:

- La imagen debe estar en formato JPEG, PNG, BMP, PDF o TIFF.
- El tamaño de la imagen debe estar entre 50 x 50 y 10 000 x 10 000 píxeles. Las páginas PDF deben ser de 17 x 17 pulgadas o más pequeñas.
- El tamaño del archivo de la imagen debe ser menor de 20 megabytes (MB).

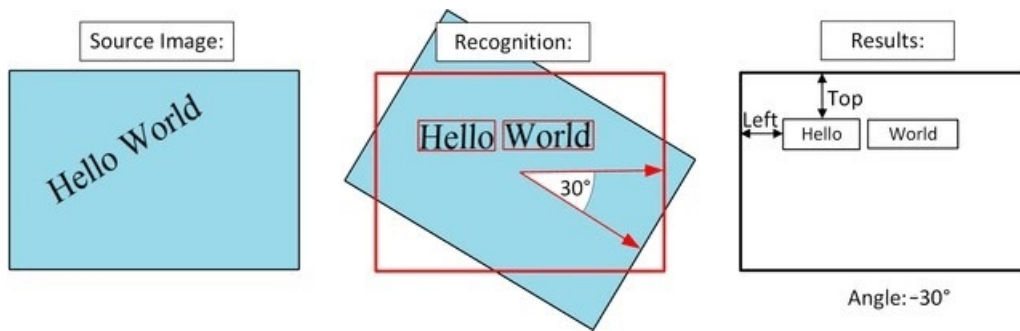
Limitaciones

Si usa una suscripción gratuita, Read API solo procesará las dos primeras páginas de un documento PDF o TIFF. Con una suscripción de pago, procesará hasta 200 páginas. Tenga en cuenta también que la API detectará un máximo de 300 líneas por página.

OCR API (reconocimiento óptico de caracteres)

La API de reconocimiento óptico de caracteres (OCR) de Computer Vision es similar a Read API, pero se ejecuta de forma sincrónica y no está optimizada para documentos de gran tamaño. Usa un modelo de reconocimiento anterior, pero funciona con más idiomas; consulte [Idiomas admitidos](#) para obtener una lista completa de los idiomas admitidos.

Si es necesario, OCR corrige el giro del texto reconocido devolviendo el desplazamiento de rotación en grados alrededor del eje horizontal de la imagen. OCR también proporciona las coordenadas del marco de cada palabra, como se observa en la ilustración siguiente.



Consulte los [documentos de referencia de OCR](#) para más información.

Requisitos de imagen

OCR API funciona con imágenes que cumplan los requisitos siguientes:

- La imagen debe estar en formato JPEG, PNG, GIF o BMP.
- El tamaño de la imagen de entrada debe estar entre 50 x 50 y 4200 x 4200 píxeles.
- El texto de la imagen se puede girar cualquier múltiplo de 90 grados, además de un pequeño ángulo de hasta 40 grados.

Limitaciones

En las fotografías en las que predomina el texto, las palabras que se reconocen de forma parcial pueden producir falsos positivos. En algunas fotografías, especialmente las que no tienen texto, la precisión puede variar según el tipo de imagen.

Recognize Text API

NOTE

Recognize Text API está en desuso en favor de Read API. Read API tiene funcionalidades similares y se ha actualizado para trabajar con archivos de varias páginas, TIFF y PDF.

Recognize Text API es similar a OCR, pero se ejecuta de forma asincrónica y usa modelos de reconocimiento actualizados. Consulte los [documentos de referencia de Recognize Text API](#) para más información.

Requisitos de imagen

Recognize Text API funciona con imágenes que cumplan los requisitos siguientes:

- La imagen debe estar en formato JPEG, PNG o BMP.
- El tamaño de la imagen debe estar entre 50 x 50 y 4200 x 4200 píxeles.
- El tamaño de archivo de la imagen debe ser inferior a 4 megabytes (MB).

Limitaciones

La precisión de las operaciones de reconocimiento de texto depende de la calidad de las imágenes. Los siguientes factores pueden provocar una lectura imprecisa:

- Imágenes borrosas.
- Texto escrito a mano o en cursiva.
- Estilos de fuente artísticos.
- Tamaño de texto pequeño.
- Fondos complejos, sombras, brillo sobre el texto o distorsión de la perspectiva.
- Letra mayúscula grande o falta de letras mayúsculas iniciales.
- Texto de superíndice, subíndice o tachado.

Pasos siguientes

Siga la guía de inicio rápido [Extracción de texto impreso \(OCR\)](#) para implementar el reconocimiento de texto en una sencilla aplicación de C#.

Detección de contenido para adultos

13/01/2020 • 2 minutes to read • [Edit Online](#)

Computer Vision puede detectar material para adultos en imágenes para que los programadores puedan restringir la presentación de dichas imágenes en su software. Las marcas de contenido se aplican con una puntuación entre cero y uno para que los desarrolladores puedan interpretar los resultados según sus preferencias.

NOTE

Una gran parte de esta funcionalidad está disponible en el servicio [Azure Content Moderator](#). Consulte esta alternativa para las soluciones en escenarios de moderación de contenido más rigurosas, como la moderación de texto y los flujos de trabajo de revisión humana.

Definiciones de las marcas de contenido

Dentro de la clasificación "adulto" hay varias categorías diferentes:

- Las imágenes **para adultos** se definen como aquellas de naturaleza explícitamente sexual que suelen representar desnudos y actos sexuales.
- Las imágenes **subidas de tono** se definen como las imágenes de naturaleza sexualmente sugerente y que a menudo contienen material sexual menos explícito que las imágenes etiquetadas como **para adultos**.
- Las imágenes **gore** son las que representan experiencias de ese tipo.

Uso de la API

Puede detectar contenido para adultos con la API [Analyze Image](#). Al agregar el valor de `Adult` al parámetro de consulta **visualFeatures**, la API devuelve tres propiedades booleanas — `isAdultContent`, `isRacyContent` y `isGoryContent` — en su respuesta JSON. El método también devuelve las propiedades correspondientes — `adultScore`, `racyScore` y `goreScore` — que representan las puntuaciones de confianza entre cero y uno para cada categoría correspondiente.

- [Inicio rápido: Análisis de imágenes \(SDK .NET\)](#)
- [Inicio rápido: Análisis de imágenes \(API REST\)](#)

Llamada a Computer Vision API

13/01/2020 • 10 minutes to read • [Edit Online](#)

En este artículo se muestra cómo llamar a la API Computer Vision mediante la API REST. Los ejemplos están escritos en C# mediante la biblioteca cliente de la API Computer Vision y como llamadas HTTP POST o GET. Este artículo se centra en:

- Obtención de etiquetas, descripciones y categorías
- Obtención de información específica de dominio o "celebridades"

Requisitos previos

- Una dirección URL de la imagen o ruta de acceso a la imagen almacenada localmente
- Métodos de entrada admitidos: archivos binarios de imagen raw en forma de application/octet-stream o URL de imagen
- Formatos de archivos de imagen admitidos: JPEG, PNG, GIF y BMP
- Tamaño del archivo de imagen: 4 MB o menos
- Dimensiones de la imagen: 50 × 50 píxeles o superior

En los ejemplos de este artículo se muestran las características siguientes:

- Análisis de una imagen para devolver una matriz de etiquetas y una descripción
- Análisis de una imagen con un modelo específico de dominio (en concreto, el modelo "celebridades") para devolver el resultado correspondiente en JSON

Las características ofrecen las siguientes opciones:

- **Opción 1:** Análisis con ámbito. Análisis de solo un modelo determinado
- **Opción 2:** Análisis mejorado. Análisis para ofrecer detalles adicionales con una [taxonomía de 86 categorías](#)

Autorización de la llamada a la API

Cada llamada a Computer Vision API requiere una clave de suscripción. Esta clave debe pasarse a través de un parámetro de cadena de consulta o especificarse en la cabecera de la solicitud.

Para obtener una clave de evaluación gratuita, realice una de las siguientes acciones:

- Vaya a la página [Pruebe Cognitive Services](#).
- Vaya a la página de [creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision.

Para pasar la clave de suscripción, realice una de las acciones siguientes:

- Pásela mediante una cadena de consulta, como en este ejemplo de la API Computer Vision:

```
https://westus.api.cognitive.microsoft.com/vision/v2.1/analyze?
visualFeatures=Description,Tags&subscription-key=<Your subscription key>
```

- Especificuella en el encabezado de solicitud HTTP:

```
ocp-apim-subscription-key: <Your subscription key>
```

- Si se usa la biblioteca de cliente, pase la clave mediante el constructor de ComputerVisionClient y especifique la región en una propiedad del cliente:

```
var visionClient = new ComputerVisionClient(new ApiKeyServiceClientCredentials("Your subscriptionKey"))
{
    Endpoint = "https://westus.api.cognitive.microsoft.com"
}
```

Carga de una imagen en el servicio de la API Computer Vision

La forma básica para realizar la llamada a la API Computer Vision es cargar directamente una imagen para devolver etiquetas, una descripción y las celebridades. Para ello, envíe una solicitud "POST" con la imagen binaria en el cuerpo HTTP junto con los datos leídos de la imagen. El método de carga es el mismo para todas las llamadas a la API Computer Vision. La única diferencia radicarán en los parámetros de consulta que se especifican.

En el caso de una imagen especificada, obtenga etiquetas y una descripción mediante cualquiera de las siguientes opciones:

Opción 1: Obtención de una lista de etiquetas y una descripción

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.1/analyze?
visualFeatures=Description,Tags&subscription-key=<Your subscription key>
```

```
using System.IO;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;

ImageAnalysis imageAnalysis;
var features = new VisualFeatureTypes[] { VisualFeatureTypes.Tags, VisualFeatureTypes.Description };

using (var fs = new FileStream(@"C:\Vision\Sample.jpg", FileMode.Open))
{
    imageAnalysis = await visionClient.AnalyzeImageInStreamAsync(fs, features);
}
```

Opción 2: Obtención de una lista solo de etiquetas o de solo una descripción

Solo para las etiquetas, ejecute:

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.1/tag?subscription-key=<Your subscription key>
var tagResults = await visionClient.TagImageAsync("http://contoso.com/example.jpg");
```

Solo para una descripción, ejecute:

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.1/describe?subscription-key=<Your subscription key>
using (var fs = new FileStream(@"C:\Vision\Sample.jpg", FileMode.Open))
{
    imageDescription = await visionClient.DescribeImageInStreamAsync(fs);
}
```

Obtiene un análisis específico del dominio (celebridades)

Opción 1: Análisis con ámbito. Análisis de solo un modelo determinado


```
POST https://westus.api.cognitive.microsoft.com/vision/v2.1/models/celebrities/analyze
var celebritiesResult = await visionClient.AnalyzeImageInDomainAsync(url, "celebrities");
```

Para esta opción, todos los demás parámetros de consulta {visualFeatures, details} no son válidos. Si desea ver todos los modelos compatibles, use:

```
GET https://westus.api.cognitive.microsoft.com/vision/v2.1/models
var models = await visionClient.ListModelsAsync();
```

Opción 2: Análisis mejorado. Análisis para ofrecer detalles adicionales con una taxonomía de 86 categorías

Para aplicaciones donde desea obtener un análisis genérico de imágenes adicional a los detalles de uno o más modelos específicos del dominio, amplíe la API v1 mediante el parámetro de consulta de modelos.

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.1/analyze?details=celebrities
```

Al invocar este método, primero se llama al clasificador [86-category](#). Si ninguna de las categorías concuerda con los modelos conocidos o coincidentes, se procede a una segunda pasada de invocaciones de clasificador. Por ejemplo, si "details=all" o "details" incluye "celebrities", llame al modelo de celebridades después de llamar al clasificador 86-category. El resultado incluye la categoría "person". A diferencia de la opción 1, este método aumenta la latencia de los usuarios que están interesados en celebridades.

En este caso, todos los parámetros de consulta de v1 se comportarán igual. Si no se especifica visualFeatures=categories, se habilita implícitamente.

Recuperar y describir la salida JSON para el análisis

Este es un ejemplo:

```
{
  "tags": [
    {
      "name": "outdoor",
      "score": 0.976
    },
    {
      "name": "bird",
      "score": 0.95
    }
  ],
  "description": {
    "tags": [
      "outdoor",
      "bird"
    ],
    "captions": [
      {
        "text": "partridge in a pear tree",
        "confidence": 0.96
      }
    ]
  }
}
```

CAMPO	TYPE	CONTENIDO
Etiquetas	object	Objeto de nivel superior de una matriz de etiquetas
tags[].Name	string	Palabra clave del clasificador de etiquetas
tags[].Score	number	Puntuación de confianza, entre 0 y 1
description	object	Objeto de nivel superior de la descripción
description.tags[]	string	Lista de etiquetas. Si la confianza es insuficiente para poder producir un título, las etiquetas pueden ser la única información disponible para el llamador.
description.captions[].text	string	Una frase que describe la imagen
description.captions[].confidence	number	Puntuación de confianza para la frase

Recuperar y describir la salida JSON de modelos específicos del dominio

Opción 1: Análisis con ámbito. Análisis de solo un modelo determinado

La salida es una matriz de etiquetas, tal como se muestra en el ejemplo siguiente:

```
{
  "result": [
    {
      "name": "golden retriever",
      "score": 0.98
    },
    {
      "name": "Labrador retriever",
      "score": 0.78
    }
  ]
}
```

Opción 2: Análisis mejorado. Análisis para ofrecer detalles adicionales con una taxonomía de 86 categorías

Para modelos específicos del dominio con la opción 2 (Análisis mejorado), el tipo de devolución de categorías se amplía, tal como se muestra en el ejemplo siguiente:

```
{
  "requestId": "87e44580-925a-49c8-b661-d1c54d1b83b5",
  "metadata": {
    "width": 640,
    "height": 430,
    "format": "Jpeg"
  },
  "result": {
    "celebrities": [
      {
        "name": "Richard Nixon",
        "faceRectangle": {
          "left": 107,
          "top": 98,
          "width": 165,
          "height": 165
        },
        "confidence": 0.9999827
      }
    ]
  }
}
```

El campo de categorías es una lista de una o varias de las [86 categorías](#) de la taxonomía original. Las categorías que terminan con un guión bajo coincidirán con esa categoría y sus elementos secundarios (por ejemplo, "people_" o "people_group", para el modelo de celebridades).

CAMPO	TYPE	CONTENIDO
categories	object	Objeto de nivel superior
categories[].name	string	Nombre de la lista de taxonomía de 86 categorías
categories[].score	number	Puntuación de confianza, entre 0 y 1
categories[].detail	object?	(Opcional) Objeto de detalle.

Si varias categorías coinciden (por ejemplo, el clasificador "86-category" devuelve una puntuación para "people_" y "people_young" si model=celebrities), los detalles se asocian a la coincidencia de nivel más general ("people_" en dicho ejemplo).

Respuestas de errores

Estos errores son idénticos a los de vision.analyze, con el error adicional de NotSupportedModel (HTTP 400), que puede devolverse en escenarios de la opción 1 y la opción 2. Para la opción 2 (análisis mejorado), si no se reconoce ninguno de los modelos especificados en los detalles, la API devolverá NotSupportedModel, incluso si uno o varios de ellos son válidos. Para averiguar qué modelos se admiten, se puede llamar a listModels.

Pasos siguientes

Para usar la API de REST, vaya a la [referencia de Computer Vision API](#).

Instalación y ejecución de los contenedores de Read (versión preliminar)

14/01/2020 • 25 minutes to read • [Edit Online](#)

Los contenedores le permiten ejecutar las API de Computer Vision en su propio entorno. Los contenedores son excelentes para requisitos específicos de control de datos y seguridad. En este artículo, aprenderá a descargar, instalar y ejecutar un contenedor de Computer Vision.

Un único contenedor Docker, *Read* está disponible para Computer Vision: El contenedor *Read* le permite detectar y extraer *texto impreso* de imágenes que muestren diversos objetos con diferentes superficies y fondos, como recibos, pósteres y tarjetas de visita. Además, el contenedor *Read* también detecta *texto manuscrito* en las imágenes y admite los formatos PDF, TIFF y multipágina. Para obtener más información, consulte la documentación de la [API Read](#).

Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.

Requisitos previos

Debe cumplir los siguientes requisitos previos para poder usar los contenedores:

OBLIGATORIO	PROPÓSITO
Motor de Docker	<p>Necesita que el motor de Docker esté instalado en un equipo host. Docker dispone de paquetes que configuran el entorno de Docker en macOS, Windows y Linux. Para conocer los principios básicos de Docker y de los contenedores, consulte Introducción a Docker.</p> <p>Docker debe configurarse para permitir que los contenedores se conecten con Azure y envíen datos de facturación a dicho servicio.</p> <p>En Windows, Docker también debe estar configurado de forma que admita los contenedores de Linux.</p>
Conocimientos sobre Docker	<p>Debe tener conocimientos básicos sobre los conceptos de Docker, como los registros, los repositorios, los contenedores y las imágenes de contenedor, así como conocer los comandos <code>docker</code> básicos.</p>
Recurso de Computer Vision	<p>Para poder usar el contenedor, debe tener:</p> <p>Un recurso de Computer Vision de Azure y la clave de API asociada con el URI del punto de conexión. Ambos valores están disponibles en las páginas de introducción y claves del recurso y son necesarios para iniciar el contenedor.</p> <p>{API_KEY} : una de las dos claves de recurso disponibles en la página Claves</p> <p>{ENDPOINT_URI} : el punto de conexión tal como se proporciona en la página de Introducción.</p>

Solicitud de acceso al registro de contenedor privado

Rellene y envíe el [formulario de solicitud de contenedores de visión de Cognitive Services](#) para solicitar acceso al contenedor. El formulario solicita información acerca del usuario y de su empresa, así como del escenario de usuario para el que se va a usar el contenedor. Después de enviar el formulario, el equipo de Azure Cognitive Services lo revisa para asegurarse de que cumple los criterios de acceso al registro de contenedor privado.

IMPORTANT

Debe usar una dirección de correo electrónico asociada con una cuenta de Microsoft (MSA) o de Azure Active Directory (Azure AD) en el formulario.

Si se aprueba la solicitud, recibirá un correo electrónico con instrucciones que describen cómo obtener las credenciales y acceder al registro de contenedor privado.

Inicio de sesión en el registro de contenedor privado

Hay varias maneras de autenticarse con el registro de contenedor privado para los contenedores de Cognitive Services. Se recomienda que utilice el método de línea de comandos mediante la [CLI de Docker](#).

Use el comando [docker login](#), como se muestra en el ejemplo siguiente, para iniciar sesión en `containerpreview.azurecr.io`, que es el registro de contenedor privado de los contenedores de Cognitive Services. Reemplace `<username>` por el nombre de usuario y `<password>` por la contraseña proporcionada en las credenciales que recibió del equipo de Azure Cognitive Services.

```
docker login containerpreview.azurecr.io -u <username> -p <password>
```

Si ha protegido las credenciales en un archivo de texto, puede concatenar el contenido del archivo de texto con el comando `docker login`. Use el comando `cat`, como se muestra en el siguiente ejemplo: Sustituya `<passwordFile>` por la ruta y el nombre del archivo de texto que contiene la contraseña. Reemplace `<username>` por el nombre de usuario proporcionado en las credenciales.

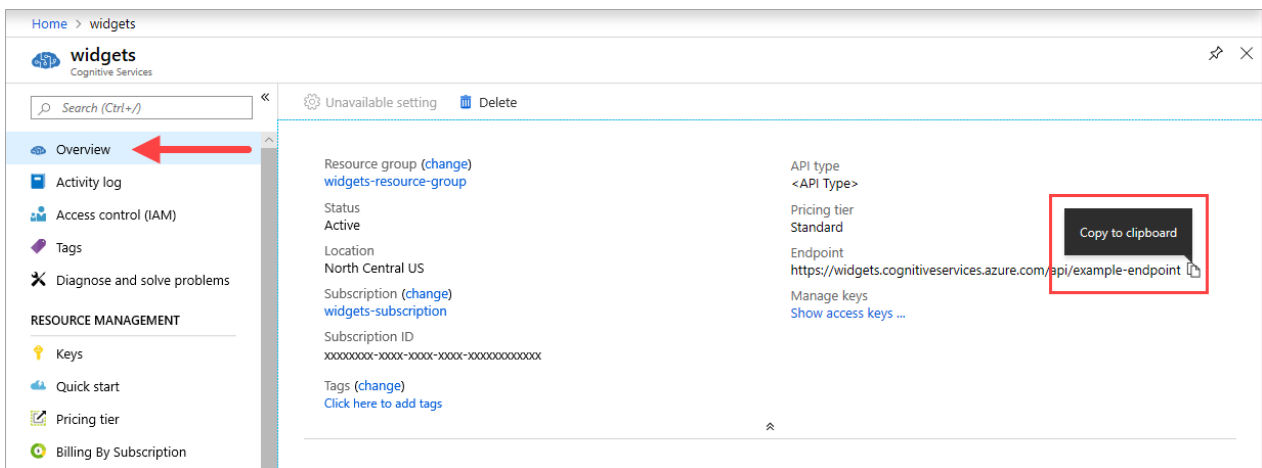
```
cat <passwordFile> | docker login containerpreview.azurecr.io -u <username> --password-stdin
```

Recopilación de los parámetros obligatorios

Hay tres parámetros principales para todos los contenedores de Cognitive Services que son necesarios. El contrato de licencia para el usuario final (CLUF) debe estar presente con un valor de `accept`. Además, se necesitan una dirección URL de punto de conexión y una clave de API.

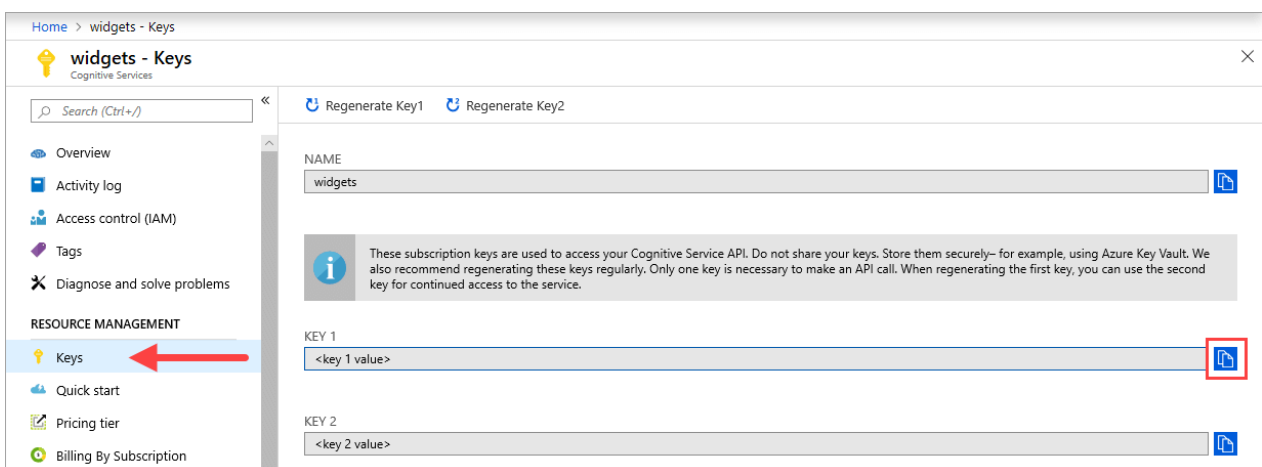
URI de punto de conexión `{ENDPOINT_URI}`

El valor del URI del **punto de conexión** está disponible en la página *Información general* de Azure Portal del recurso de Cognitive Services correspondiente. Vaya a la página *Información general*, mantenga el cursor sobre el punto de conexión y aparecerá un icono `Copy to clipboard`. Cópielo y utilícelo cuando sea necesario.



Claves {API_KEY}

Esta clave se usa para iniciar el contenedor y está disponible en la página de claves de Azure Portal del recurso de Cognitive Services correspondiente. Vaya a la página *Claves* y haga clic en el icono `Copy to clipboard`.



IMPORTANT

Estas claves de suscripción se usan para tener acceso a la API de Cognitive Services. No comparta las claves. Almacénelas de forma segura, por ejemplo, con Azure Key Vault. También se recomienda regenerar estas claves periódicamente. Solo se necesita una clave para realizar una llamada API. Al volver a generar la primera clave, puede usar la segunda clave para seguir teniendo acceso al servicio.

El equipo host

El host es un equipo basado en x64 que ejecuta el contenedor de Docker. Puede ser un equipo del entorno local o un servicio de hospedaje de Docker incluido en Azure, como:

- [Azure Kubernetes Service](#).
- [Azure Container Instances](#).
- Un clúster de [Kubernetes](#) implementado en [Azure Stack](#). Para obtener más información, consulte [Implementación de Kubernetes en Azure Stack](#).

Recomendaciones y requisitos del contenedor

NOTE

Los requisitos y las recomendaciones se basan en pruebas comparativas con una única solicitud por segundo, con una imagen de 8 MB de una carta comercial digitalizada que contiene 29 líneas y un total de 803 caracteres.

En la tabla siguiente se describe la asignación mínima y recomendada de recursos para cada contenedor de

lectura.

CONTENEDOR	MÍNIMA	RECOMENDADO	TPS (MÍNIMO, MÁXIMO)
Lectura	1 núcleo, 8 GB de memoria, 0,24 TPS	8 núcleos, 16 GB de memoria, 1,17 TPS	0,24, 1,17

- Cada núcleo debe ser de 2,6 gigahercios (GHz) como mínimo.
- TPS: transacciones por segundo

El núcleo y la memoria se corresponden con los valores de `--cpus` y `--memory` que se usan como parte del comando `docker run`.

Obtención de la imagen del contenedor con `docker pull`

Hay imágenes de contenedor para Leer disponibles.

CONTENEDOR	CONTAINER REGISTRY/REPOSITORIO/NOMBRE DE IMAGEN
Lectura	<code>containerpreview.azurecr.io/microsoft/cognitive-services-read:latest</code>

Use el comando `docker pull` para descargar una imagen de contenedor.

Docker pull para el contenedor Leer

```
docker pull containerpreview.azurecr.io/microsoft/cognitive-services-read:latest
```

TIP

Puede usar el comando `docker images` para enumerar las imágenes de contenedor descargadas. Por ejemplo, el comando siguiente muestra el id., el repositorio y la etiqueta de cada imagen de contenedor descargada, con formato de tabla:

```
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
```

IMAGE ID	REPOSITORY	TAG
<image-id>	<repository-path/name>	<tag-name>

Uso del contenedor

Una vez que el contenedor esté en el [equipo host](#), utilice el siguiente proceso para trabajar con el contenedor.

1. [Ejecute el contenedor](#) con la configuración de facturación requerida. Hay más [ejemplos](#) del comando `docker run` disponibles.
2. [Consulta del punto de conexión de predicción del contenedor](#).

Ejecute el contenedor con `docker run`.

Utilice el comando `docker run` para ejecutar el contenedor. Consulte [Recopilación de los parámetros obligatorios](#) para más información sobre cómo obtener los valores de `{ENDPOINT_URI}` y `{API_KEY}`.

Hay disponibles [ejemplos](#) del comando `docker run`.

```
docker run --rm -it -p 5000:5000 --memory 16g --cpus 8 \
containerpreview.azurecr.io/microsoft/cognitive-services-read \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

Este comando:

- Ejecuta el contenedor Leer desde la imagen de contenedor.
- Asigna un núcleo de 8 CPU y 16 gigabytes (GB) de memoria.
- Expone el puerto TCP 5000 y asigna un pseudo-TTY para el contenedor.
- Una vez que se produce la salida, quita automáticamente el contenedor. La imagen del contenedor sigue estando disponible en el equipo host.

Hay más [ejemplos](#) del comando `docker run` disponibles.

IMPORTANT

Para poder ejecutar el contenedor, las opciones `Eula`, `Billing` y `ApiKey` deben estar especificadas; de lo contrario, el contenedor no se iniciará. Para obtener más información, vea [Facturación](#).

Ejecución de varios contenedores en el mismo host

Si tiene pensado ejecutar varios contenedores con puertos expuestos, asegúrese de que ejecuta cada contenedor con un puerto expuesto diferente. Por ejemplo, ejecute el primer contenedor en el puerto 5000 y el segundo en el puerto 5001.

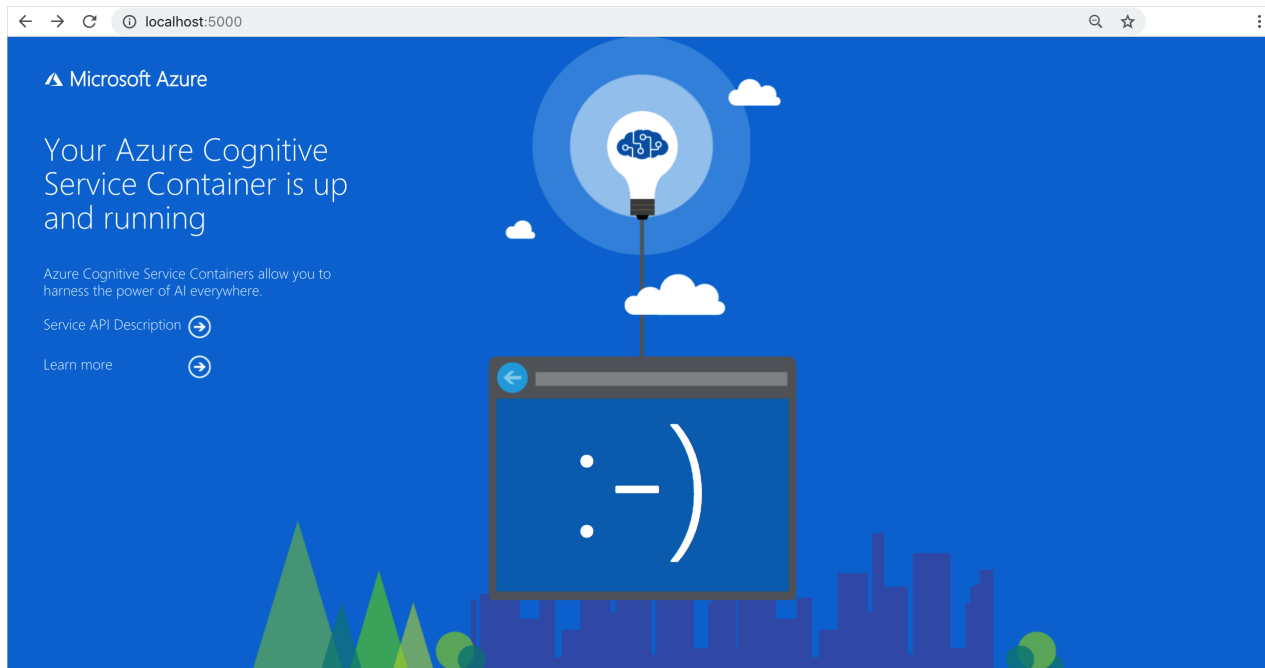
Puede tener este contenedor y un contenedor de Azure Cognitive Services diferente en ejecución simultáneamente en el HOST. También puede tener varios contenedores del mismo contenedor de Cognitive Services en ejecución.

Comprobación de que un contenedor está en ejecución

Hay varias maneras de comprobar que el contenedor está en ejecución. Busque la dirección *IP externa* y el puerto expuesto del contenedor en cuestión y abra el explorador web que prefiera. Use las distintas direcciones URL de solicitud para validar que el contenedor se está ejecutando. Las direcciones URL de solicitud de ejemplo que se enumeran a continuación son `http://localhost:5000`, pero el contenedor específico puede variar. Tenga en cuenta que va a confiar en la *dirección IP externa* y el puerto expuesto del contenedor.

URL DE LA SOLICITUD	PROPÓSITO
<code>http://localhost:5000/</code>	El contenedor ofrece una página principal.
<code>http://localhost:5000/status</code>	Se solicitó con HTTP GET, para comprobar que el contenedor está en ejecución sin causar una consulta al punto de conexión. Esta solicitud se puede usar con los sondeos de ejecución y preparación de Kubernetes.

URL DE LA SOLICITUD	PROPÓSITO
<code>http://localhost:5000/swagger</code>	El contenedor cuenta con un completo conjunto de documentación sobre los puntos de conexión y una característica de prueba . Esta característica le permite especificar la configuración en un formulario HTML basado en web y realizar la consulta sin necesidad de escribir código. Una vez que la consulta devuelve resultados, se proporciona un ejemplo del comando CURL para mostrar los encabezados HTTP y el formato de cuerpo requeridos.



Consulta del punto de conexión de predicción del contenedor

El contenedor proporciona varias API de puntos de conexión de predicción de consultas basadas en REST.

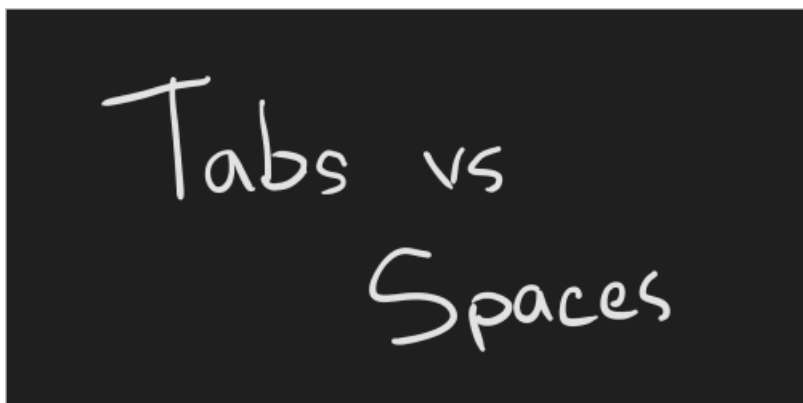
Utilice el host, `http://localhost:5000`, con las API de contenedor.

Lectura asincrónica

Puede usar las operaciones `POST /vision/v2.0/read/core/asyncBatchAnalyze` y

`GET /vision/v2.0/read/operations/{operationId}` conjuntamente para leer una imagen asincrónicamente, de manera similar a cómo el servicio Computer Vision usa las operaciones de REST correspondientes. El método POST asincrónico devolverá un valor `operationId` que se usa como identificador de la solicitud HTTP GET.

En la interfaz de usuario de Swagger, seleccione `asyncBatchAnalyze` para expandirlo en el explorador. A continuación, seleccione **Probarlo** > **Elegir archivo**. En este ejemplo, usaremos la imagen siguiente:



Una vez ejecutado correctamente el método POST, devuelve un código de estado **HTTP 202**. Como parte de la respuesta, hay un encabezado `operation-location` que contiene el punto de conexión del resultado de la solicitud.

```
content-length: 0
date: Fri, 13 Sep 2019 16:23:01 GMT
operation-location: http://localhost:5000/vision/v2.0/read/operations/a527d445-8a74-4482-8cb3-c98a65ec7ef9
server: Kestrel
```

El elemento `operation-location` es la dirección URL completa y se obtiene acceso a ella a través de HTTP GET. Esta es la respuesta JSON a la ejecución de la dirección URL `operation-location` desde la imagen anterior:

```
{
  "status": "Succeeded",
  "recognitionResults": [
    {
      "page": 1,
      "clockwiseOrientation": 2.42,
      "width": 502,
      "height": 252,
      "unit": "pixel",
      "lines": [
        {
          "boundingBox": [
            56,
            39,
            317,
            50,
            313,
            134,
            53,
            123
          ],
          "text": "Tabs VS",
          "words": [
            {
              "boundingBox": [
                90,
                43,
                243,
                53,
                243,
                123,
                94,
                125
              ],
              "text": "Tabs",
              "confidence": "Low"
            },
            {
              "boundingBox": [
                259,
                55,
                313,
                62,
                313,
                122,
                259,
                123
              ],
              "text": "VS"
            }
          ]
        }
      ]
    }
  ],
  {
    "boundingBox": [
```

```

        221,
        148,
        417,
        146,
        417,
        206,
        227,
        218
    ],
    "text": "Spaces",
    "words": [
        {
            "boundingBox": [
                230,
                148,
                416,
                141,
                419,
                211,
                232,
                218
            ],
            "text": "Spaces"
        }
    ]
}

```

Lectura sincrónica

Puede usar la operación `POST /vision/v2.0/read/core/Analyze` para leer sincrónicamente una imagen. Una vez leída la imagen en su totalidad, entonces, y solo entonces, devuelve la API una respuesta JSON. La única excepción a esto es un escenario de error. Cuando se produce un error, se devuelve el siguiente código JSON:

```

{
    status: "Failed"
}

```

El objeto de respuesta JSON tiene el mismo gráfico de objetos que la versión asincrónica. Si es un usuario de JavaScript y quiere seguridad de tipos, puede usar los tipos siguientes para convertir la respuesta JSON como un objeto `AnalyzeResult`.

```

export interface AnalyzeResult {
    status: Status;
    recognitionResults?: RecognitionResult[] | null;
}

export enum Status {
    NotStarted = 0,
    Running = 1,
    Failed = 2,
    Succeeded = 3
}

export enum Unit {
    Pixel = 0,
    Inch = 1
}

export interface RecognitionResult {
    page?: number | null;
    clockwiseOrientation?: number | null;
    width?: number | null;
    height?: number | null;
    unit?: Unit | null;
    lines?: Line[] | null;
}

export interface Line {
    boundingBox?: number[] | null;
    text: string;
    words?: Word[] | null;
}

export enum Confidence {
    High = 0,
    Low = 1
}

export interface Word {
    boundingBox?: number[] | null;
    text: string;
    confidence?: Confidence | null;
}

```

Para ver un caso de uso de ejemplo, consulte el [espacio aislado de TypeScript aquí](#) y seleccione **Ejecutar** para visualizar su facilidad de uso.

Detención del contenedor

Para apagar el contenedor, en el entorno de la línea de comandos donde se ejecuta el contenedor, seleccione **Ctrl + C**.

solución de problemas

Si ejecuta el contenedor con un [montaje](#) de salida y el registro habilitados, el contenedor genera archivos de registro que resultan útiles para solucionar problemas que se producen al iniciar o ejecutar el contenedor.

TIP

Para más información e instrucciones para solución de problemas, vea [Preguntas frecuentes de los contenedores de Cognitive Services](#).

Facturación

Los contenedores de Cognitive Services envían información de facturación a Azure mediante el recurso correspondiente de la cuenta de Azure.

Las consultas en el contenedor se facturan con el plan de tarifa del recurso de Azure que se usa para `<ApiKey>`.

Los contenedores de Azure Cognitive Services no tienen licencia para ejecutarse si no están conectados al punto de conexión de facturación para las mediciones. Debe habilitar los contenedores para que comuniquen la información de facturación al punto de conexión de facturación en todo momento. Los contenedores de Cognitive Services no envían datos de los clientes (por ejemplo, la imagen o el texto que se está analizando) a Microsoft.

Conexión con Azure

El contenedor necesita que se ejecuten los valores del argumento de facturación. Estos valores permiten al contenedor conectarse al punto de conexión de facturación. El contenedor informa sobre el uso cada 10 a 15 minutos. Si el contenedor no se conecta a Azure en la ventana de tiempo permitida, continuará ejecutándose pero no atenderá las consultas hasta que se restaure el punto de conexión de facturación. Se intenta 10 veces la conexión en el mismo intervalo de tiempo de 10 a 15 minutos. Si no se puede conectar con el punto de conexión de facturación en esos 10 intentos, el contenedor deja de ejecutarse.

Argumentos de facturación

Para que el comando `docker run` inicie el contenedor, se deben especificar las tres opciones siguientes se deben especificar con valores válidos:

OPCIÓN	DESCRIPCIÓN
<code>ApiKey</code>	La clave de API del recurso de Cognitive Services que se usa para realizar un seguimiento de la información de facturación. El valor de esta opción se debe establecer en una clave de API para el recurso aprovisionado que se especifica en <code>Billing</code> .
<code>Billing</code>	El punto de conexión del recurso de Cognitive Services que se usa para realizar el seguimiento de la información de facturación. El valor de esta opción debe establecerse en el URI del punto de conexión de un recurso aprovisionado de Azure.
<code>Eula</code>	Indica que ha aceptado la licencia del contenedor. El valor de esta opción debe establecerse en accept .

Para obtener más información acerca de estas opciones, consulte [Configure containers](#) (Configuración de contenedores).

Publicaciones de blog

- [Ejecución de contenedores de Cognitive Services](#)
- [Azure Cognitive Services](#)

Ejemplos para desarrolladores

Hay ejemplos para desarrolladores disponibles en nuestro [repositorio de GitHub](#).

Ver seminario web

Únase al [seminario web](#) para más información sobre:

- Implementación de Cognitive Services en cualquier máquina con Docker
- Implementación de Cognitive Services en AKS

Resumen

En este artículo, ha aprendido los conceptos y el flujo de trabajo para la descarga, instalación y ejecución de contenedores de Computer Vision. En resumen:

- Computer Vision proporciona un contenedor de Linux para Docker que incluye Leer.
- Las imágenes de contenedor se descargan desde el registro de contenedores "Contenedor (versión preliminar)" en Azure.
- Las imágenes de contenedor se ejecutan en Docker.
- Puede usar la API REST o el SDK para llamar a operaciones en contenedores de Read mediante la especificación del URI del host del contenedor.
- Debe especificar la información de facturación al crear una instancia de un contenedor.

IMPORTANT

Los contenedores de Cognitive Services no tienen licencia para ejecutarse sin estar conectados a Azure para realizar mediciones. Los clientes tienen que habilitar los contenedores para comunicar la información de facturación con el servicio de medición en todo momento. Los contenedores de Cognitive Services no envían datos de los clientes (por ejemplo, la imagen o el texto que se está analizando) a Microsoft.

Pasos siguientes

- Revise [Configure containers](#) (Configuración de contenedores) para ver las opciones de configuración.
- Revise [Introducción a Computer Vision](#) para obtener más información sobre el reconocimiento de texto escrito a mano e impreso.
- Consulte [Computer Vision API](#) para obtener más información acerca de los métodos que admite el contenedor.
- Consulte [Preguntas más frecuentes \(P+F\)](#) para resolver problemas relacionados con la funcionalidad de Computer Vision.
- Use más [contenedores de Cognitive Services](#)

Configuración de contenedores de Docker de Computer Vision

13/01/2020 • 16 minutes to read • [Edit Online](#)

El entorno de ejecución del contenedor de Computer Vision se configura mediante los argumentos del comando `docker run`. Este contenedor tiene varias opciones de configuración necesarias, así como otras opcionales. Hay disponibles varios [ejemplos](#) del comando. La configuración específica del contenedor es la configuración de facturación.

Valores de configuración

Este contenedor tiene las siguientes opciones de configuración:

OBLIGATORIO	CONFIGURACIÓN	PROPÓSITO
Sí	ApiKey	Realiza el seguimiento de la información de facturación.
Sin	Application Insights	Permite agregar compatibilidad con los datos de telemetría de Azure Application Insights al contenedor.
Sí	Facturación	Especifica el URI del punto de conexión del recurso de servicio en Azure.
Sí	Eula	Indica que ha aceptado la licencia del contenedor.
Sin	Fluentd	Escribe el registro y, opcionalmente, los datos de métricas en un servidor de Fluentd.
Sin	Proxy HTTP	Configura un proxy HTTP para realizar solicitudes de salida.
Sin	Logging	Proporciona compatibilidad con el registro de ASP.NET Core al contenedor.
Sin	Mounts	Lee y escribe los datos desde el equipo host al contenedor y del contenedor de nuevo al equipo host.

IMPORTANT

Las opciones [ApiKey](#), [Billing](#) y [Eula](#) se usan en conjunto y debe proporcionar valores válidos para las tres; en caso contrario, no se inicia el contenedor. Para obtener más información sobre el uso de estas opciones de configuración para crear instancias de un contenedor, consulte [Facturación](#).

Opción de configuración ApiKey

La opción de configuración `ApiKey` especifica la clave de recurso de Azure `Cognitive Services` usada para realizar un seguimiento de la información de facturación del contenedor. Debe especificar un valor para `ApiKey` que debe ser una clave válida para el recurso de *Cognitive Services* especificado para la opción de configuración `Billing`.

Este valor se puede encontrar en el siguiente lugar:

- Azure Portal: Administración de recursos de **Cognitivas Services**, en **Claves**.

Opción de configuración ApplicationInsights

La opción de configuración `ApplicationInsights` le permite agregar compatibilidad con los datos de telemetría de [Azure Application Insights](#) al contenedor. Application Insights proporciona una supervisión detallada del contenedor. Puede supervisar fácilmente la disponibilidad, el rendimiento y el uso del contenedor. También puede identificar y diagnosticar errores en el contenedor rápidamente.

En la tabla siguiente se describen las opciones de configuración compatibles en la sección `ApplicationInsights`.

OBLIGATORIO	NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
-------------	--------	---------------	-------------

OBLIGATORIO	NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
Sin	InstrumentationKey	Cadena	Clave de instrumentación de la instancia de Application Insights para la que se envían los datos de telemetría del contenedor. Para más información, consulte Application Insights para ASP.NET Core . Ejemplo: InstrumentationKey=123456789

Opción de configuración Billing

La opción de configuración `Billing` especifica el URI del punto de conexión del recurso de *Cognitive Services* que se usa para medir la información de facturación del contenedor. Debe especificar un valor para esta opción de configuración, que debe ser un URI de punto de conexión válido para un recurso de *Cognitive Services* en Azure. El contenedor informa sobre el uso cada 10 a 15 minutos.

Este valor se puede encontrar en el siguiente lugar:

- Azure Portal: Información general de **Cognitivas Services**, con la etiqueta `Endpoint`

No olvide agregar la ruta `vision/v1.0` al URI de punto de conexión, tal como se muestra en la tabla siguiente.

OBLIGATORIO	NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
Sí	Billing	Cadena	Identificador URI del punto de conexión de facturación Ejemplo: Billing=https://westcentralus.api.cognitiv

Opción de configuración Eula

La opción de configuración `Eula` indica que ha aceptado la licencia del contenedor. Debe especificar un valor para esta opción de configuración y el valor debe establecerse en `accept`.

OBLIGATORIO	NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
Sí	Eula	Cadena	Aceptación de la licencia Ejemplo: Eula=accept

Los contenedores de Cognitive Services tienen una licencia sujeta al [contrato](#) que rige el uso de Azure. Si no tiene ningún contrato que rija el uso de Azure, acepta que el contrato que rige el uso de Azure es el [Contrato Microsoft Online Subscription](#), que incorpora los [Términos de Online Services](#). En el caso de las versiones preliminares, acepta también los [Términos de uso complementarios para las versiones preliminares de Microsoft Azure](#). Al usar el contenedor, acepta estos términos.

Opción de configuración Fluentd

Fluentd es un recopilador de datos de código abierto para el registro unificado. La opción de configuración `Fluentd` administra la conexión del contenedor a un servidor de [Fluentd](#). En el contenedor, se incluye un proveedor de registros de Fluentd que permite que el contenedor escriba los registros y, opcionalmente, los datos de métricas en un servidor de Fluentd.

En la tabla siguiente se describen las opciones de configuración compatibles en la sección `Fluentd`.

NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
Host	Cadena	Dirección IP o nombre de host DNS del servidor de Fluentd.
Port	Entero	Puerto del servidor de Fluentd. El valor predeterminado es 24 224.
HeartbeatMs	Entero	Intervalo de latidos (en milisegundos). Si no se envía ningún tráfico de evento antes de que este intervalo expire, se envía un latido al servidor de Fluentd. El valor predeterminado es 60 000 milisegundos (1 minuto).

NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
<code>SendBufferSize</code>	Entero	Espacio en búfer de red (en bytes) asignado para las operaciones de envío. El valor predeterminado es 32 768 bytes (32 kilobytes).
<code>TlsConnectionEstablishmentTimeoutMs</code>	Entero	Tiempo de expiración (en milisegundos) para establecer una conexión SSL/TLS con el servidor de Fluentd. El valor predeterminado es 10 000 milisegundos (10 segundos). Si <code>UseTLS</code> se establece en false, este valor se ignora.
<code>UseTLS</code>	Boolean	Indica si el contenedor debe utilizar SSL/TLS para comunicarse con el servidor de Fluentd. El valor predeterminado es false.

Configuración de las credenciales del proxy HTTP

Si necesita configurar un proxy HTTP para realizar solicitudes de salida, use estos dos argumentos:

NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
HTTP_PROXY	string	El proxy que se va a utilizar, por ejemplo, <code>http://proxy:8888</code> <proxy-url>
HTTP_PROXY_CREDS	string	Las credenciales necesarias para autenticarse en el proxy, por ejemplo, nombre de usuario:contraseña.
<proxy-user>	string	El usuario del proxy.
<proxy-password>	string	La contraseña asociada con <proxy-user> para el proxy.

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
HTTP_PROXY=<proxy-url> \
HTTP_PROXY_CREDS=<proxy-user>:<proxy-password> \
```

Opción de configuración Logging

La opción de configuración `Logging` administra la compatibilidad con el registro de ASP.NET Core del contenedor. Puede usar los mismos valores y opciones de configuración para el contenedor que los que usa para una aplicación ASP.NET Core.

Los siguientes proveedores de registro son compatibles con el contenedor:

PROVEEDOR	PROPÓSITO
Console	Proveedor de registro de <code>Console</code> de ASP.NET Core. Se admiten todos los valores predeterminados y las opciones de configuración de ASP.NET Core para este proveedor de registro.
Depuración	Proveedor de registro de <code>Debug</code> de ASP.NET Core. Se admiten todos los valores predeterminados y las opciones de configuración de ASP.NET Core para este proveedor de registro.
Disco	Proveedor de registro JSON. Este proveedor de registro escribe datos de registro para el montaje de salida.

Este comando de contenedor almacena información de registro en formato JSON en el montaje de salida:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Disk:Format=json
```

Este comando de contenedor muestra información de depuración, con el prefijo `debug` mientras el contenedor se ejecuta:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Console:LogLevel:Default=Debug
```

Registro del disco

El proveedor de registro `Disk` admite la configuración siguiente:

NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
<code>Format</code>	Cadena	Formato de salida de los archivos de registro. Nota: Este valor debe establecerse en <code>json</code> para habilitar el proveedor de registro. Si se especifica este valor sin especificar también un montaje de salida al crear una instancia de un contenedor, se produce un error.
<code>MaxFileSize</code>	Entero	Tamaño máximo en megabytes (MB) de un archivo de registro. Cuando el tamaño del archivo de registro actual cumple este valor o lo supera, el proveedor de registro inicia un nuevo archivo de registro. Si se especifica -1, el tamaño del archivo de registro solo está limitado por el tamaño máximo de archivo, si existe, para el montaje de salida. El valor predeterminado es 1.

Para obtener más información acerca de cómo configurar la compatibilidad con el registro de ASP.NET Core, consulte [Configuración del archivo de configuración](#).

Configuración de montaje

Utilice montajes de enlace para leer y escribir datos hacia y desde el contenedor. Puede especificar un montaje de entrada o un montaje de salida mediante la opción `--mount` del comando `docker run`.

El contenedor de Computer Vision no usa los montajes de entrada o salida para almacenar datos de entrenamiento o del servicio.

La sintaxis exacta de la ubicación de montaje del host varía según el sistema operativo del host. Además, la ubicación de montaje del [equipo host](#) puede no ser accesible debido a un conflicto entre los permisos que usa la cuenta de servicio de Docker y los permisos de la ubicación de montaje del host.

OPCIONAL	NOMBRE	TIPO DE DATOS	DESCRIPCIÓN
No permitida	<code>Input</code>	Cadena	Los contenedores de Computer Vision no usan esto.
Opcional	<code>Output</code>	Cadena	Destino del montaje de salida. El valor predeterminado es <code>/output</code> . Esta es la ubicación de los registros. Esto incluye los registros de contenedor. Ejemplo: <code>--mount type=bind,src=c:\output,target=/output</code>

Comandos de ejemplo de docker run

Los ejemplos siguientes usan las opciones de configuración para ilustrar cómo escribir y usar comandos `docker run`. Una vez que se está ejecutando, el contenedor continúa ejecutándose hasta que lo [detenga](#).

- **Carácter de continuación de línea:** Los comandos de Docker de las secciones siguientes usan la barra diagonal inversa (`\`) como un carácter de continuación de línea. Puede quitarla o reemplazarla en función de los requisitos del sistema operativo del host.

- **Orden de los argumentos:** No cambie el orden de los argumentos a menos que esté muy familiarizado con los contenedores de Docker.

Reemplace `{argument_name}` por sus propios valores:

MARCADOR DE POSICIÓN	VALOR	FORMATO O EJEMPLO
{CLAVE_API}	La clave del punto de conexión del recurso <code>Computer Vision</code> en la página Claves de <code>Computer Vision</code> de Azure.	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
{URI_PUNTODECONEXIÓN}	El valor del punto de conexión de facturación está disponible en la página Información general de Azure <code>Computer Vision</code> .	Consulte el apartado de recopilación de los parámetros necesarios para ejemplos explícitos.

NOTE

Los nuevos recursos creados después del 1 de julio de 2019 usarán nombres de subdominio personalizados. Para más información y para obtener una lista completa de los puntos de conexión regionales, consulte [Nombres de subdominios personalizados para Cognitive Services](#).

IMPORTANT

Para poder ejecutar el contenedor, las opciones `Eula`, `Billing` y `ApiKey` deben estar especificadas; de lo contrario, el contenedor no se iniciará. Para obtener más información, vea [Facturación](#). El valor de `ApiKey` es la **clave** de la página de claves de recursos de Azure `Cognitive Services`.

Ejemplos de contenedor de Docker

Los siguientes ejemplos de Docker son del contenedor Lectura.

Ejemplo básico

```
docker run --rm -it -p 5000:5000 --memory 16g --cpus 8 \
  containerpreview.azurecr.io/microsoft/cognitive-services-read \
  Eula=accept \
  Billing={ENDPOINT_URI} \
  ApiKey={API_KEY}
```

Ejemplo de registro

```
docker run --rm -it -p 5000:5000 --memory 16g --cpus 8 \
  containerpreview.azurecr.io/microsoft/cognitive-services-read \
  Eula=accept \
  Billing={ENDPOINT_URI} \
  ApiKey={API_KEY} \
  Logging:Console:LogLevel:Default=Information
```

Pasos siguientes

- Consulte [Instalación y ejecución de contenedores](#).

Uso de un contenedor de Computer Vision con Kubernetes y Helm

14/01/2020 • 14 minutes to read • [Edit Online](#)

Una opción para administrar los contenedores de Computer Vision es usar Kubernetes y Helm. Vamos a crear un paquete de Kubernetes usando Kubernetes y Helm para definir una imagen de contenedor de Computer Vision. Este paquete se va a implementar en un clúster de Kubernetes en el entorno local. Por último, exploraremos cómo probar los servicios implementados. Para más información sobre la ejecución de contenedores de Docker sin orquestación de Kubernetes, consulte [Instalación y ejecución de contenedores de Computer Vision](#).

Requisitos previos

Estos son los requisitos previos para poder usar contenedores de Computer Vision en el entorno local:

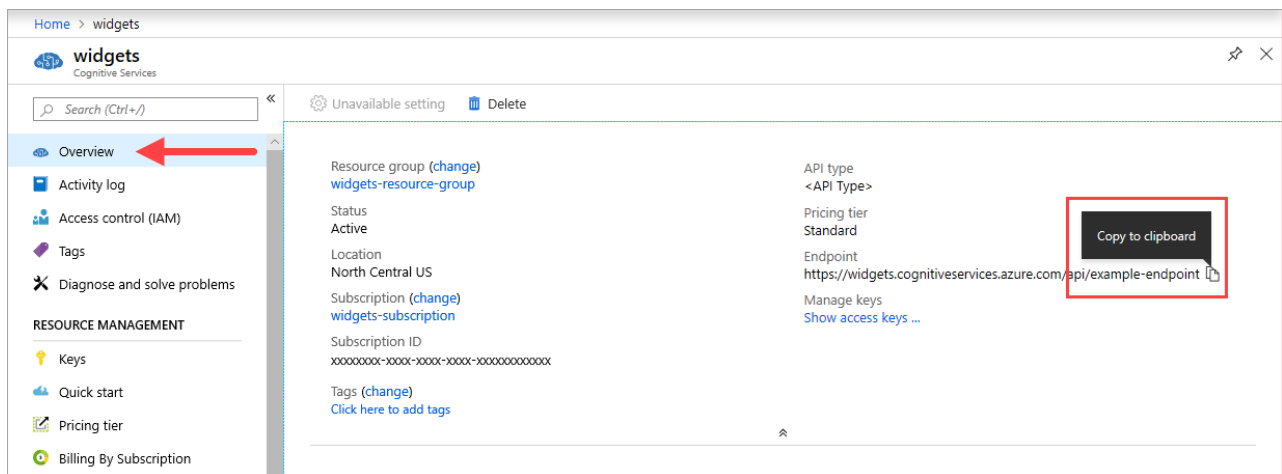
OBLIGATORIO	PROPÓSITO
Cuenta de Azure	Si no tiene una suscripción a Azure, cree una cuenta gratuita antes de empezar.
CLI de Kubernetes	La CLI de Kubernetes es necesaria para administrar las credenciales compartidas desde el registro de contenedor. Kubernetes también se necesita antes que Helm, que es el administrador de paquetes de Kubernetes.
CLI de Helm	Como parte de la instalación de la CLI de Helm , también tendrá que inicializar Helm, que instalará Tiller .
Recurso de Computer Vision	<p>Para poder usar el contenedor, debe tener:</p> <p>Un recurso de Computer Vision de Azure y la clave de API asociada con el URI del punto de conexión. Ambos valores están disponibles en las páginas de introducción y claves del recurso y son necesarios para iniciar el contenedor.</p> <p>{API_KEY} : una de las dos claves de recurso disponibles en la página Claves</p> <p>{ENDPOINT_URI} : el punto de conexión tal como se proporciona en la página de Información general.</p>

Recopilación de los parámetros obligatorios

Hay tres parámetros principales para todos los contenedores de Cognitive Services que son necesarios. El contrato de licencia para el usuario final (CLUF) debe estar presente con un valor de `accept`. Además, se necesitan una dirección URL de punto de conexión y una clave de API.

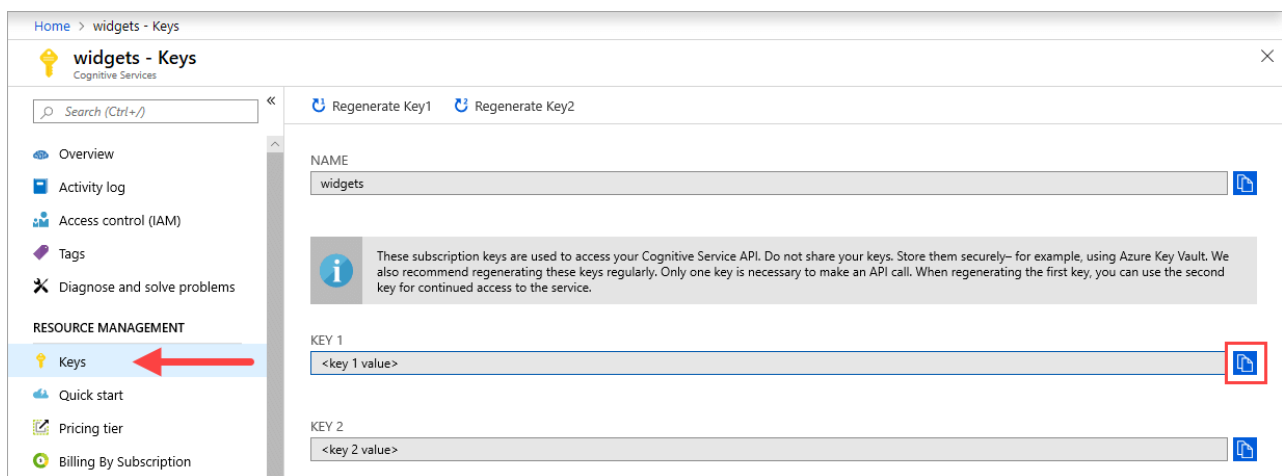
URI de punto de conexión `{ENDPOINT_URI}`

El valor del URI del **punto de conexión** está disponible en la página *Información general* de Azure Portal del recurso de Cognitive Services correspondiente. Vaya a la página *Información general*, mantenga el cursor sobre el punto de conexión y aparecerá un icono `Copy to clipboard`. Cópielo y utilícelo cuando sea necesario.



Claves {API_KEY}

Esta clave se usa para iniciar el contenedor y está disponible en la página de claves de Azure Portal del recurso de Cognitive Services correspondiente. Vaya a la página *Claves* y haga clic en el icono `Copy to clipboard` .



IMPORTANT

Estas claves de suscripción se usan para tener acceso a la API de Cognitive Services. No comparta las claves. Almacénelas de forma segura, por ejemplo, con Azure Key Vault. También se recomienda regenerar estas claves periódicamente. Solo se necesita una clave para realizar una llamada API. Al volver a generar la primera clave, puede usar la segunda clave para seguir teniendo acceso al servicio.

El equipo host

El host es un equipo basado en x64 que ejecuta el contenedor de Docker. Puede ser un equipo del entorno local o un servicio de hospedaje de Docker incluido en Azure, como:

- [Azure Kubernetes Service](#).
- [Azure Container Instances](#).
- Un clúster de [Kubernetes](#) implementado en [Azure Stack](#). Para obtener más información, consulte [Implementación de Kubernetes en Azure Stack](#).

Recomendaciones y requisitos del contenedor

NOTE

Los requisitos y las recomendaciones se basan en pruebas comparativas con una única solicitud por segundo, con una imagen de 8 MB de una carta comercial digitalizada que contiene 29 líneas y un total de 803 caracteres.

En la tabla siguiente se describe la asignación mínima y recomendada de recursos para cada contenedor de lectura.

CONTENEDOR	MÍNIMA	RECOMENDADO	TPS (MÍNIMO, MÁXIMO)
Lectura	1 núcleo, 8 GB de memoria, 0,24 TPS	8 núcleos, 16 GB de memoria, 1,17 TPS	0,24, 1,17

- Cada núcleo debe ser de 2,6 gigahercios (GHz) como mínimo.
- TPS: transacciones por segundo

El núcleo y la memoria se corresponden con los valores de `--cpus` y `--memory` que se usan como parte del comando `docker run`.

Conectar al clúster de Kubernetes

Se espera que el equipo host tenga un clúster de Kubernetes disponible. Vea este tutorial sobre la [implementación de un clúster de Kubernetes](#) para lograr un reconocimiento conceptual de cómo implementar un clúster de Kubernetes en un equipo host.

Uso compartido de credenciales de Docker con el clúster de Kubernetes

Para permitir que el clúster de Kubernetes `docker pull` las imágenes configuradas del registro de contenedor `containerpreview.azurecr.io`, debe transferir las credenciales de Docker al clúster. Ejecute el comando `kubect1 create` siguiente para crear un *secreto de registro de Docker* basado en las credenciales proporcionadas en el requisito previo de acceso al registro de contenedor.

Ejecute el siguiente comando desde la interfaz de la línea de comandos que prefiera. Asegúrese de reemplazar `<username>`, `<password>` y `<email-address>` por las credenciales del registro de contenedor.

```
kubect1 create secret docker-registry containerpreview \  
  --docker-server=containerpreview.azurecr.io \  
  --docker-username=<username> \  
  --docker-password=<password> \  
  --docker-email=<email-address>
```

NOTE

Si ya tiene acceso al registro de contenedor `containerpreview.azurecr.io`, puede crear un secreto de Kubernetes con la marca genérica en su lugar. Tenga en cuenta el siguiente comando que se ejecuta en el JSON de configuración de Docker.

```
kubect1 create secret generic containerpreview \  
  --from-file=.dockerconfigjson=~/.docker/config.json \  
  --type=kubernetes.io/dockerconfigjson
```

El siguiente resultado se imprime en la consola una vez que el secreto se ha creado correctamente.

```
secret "containerpreview" created
```

Para comprobar que el secreto se ha creado, ejecute `kubect1 get` con la marca `secrets`.

```
kubect1 get secrets
```

Al ejecutar `kubect1 get secrets`, se imprimen todos los secretos configurados.

NAME	TYPE	DATA	AGE
containerpreview	kubernetes.io/dockerconfigjson	1	30s

Configuración de los valores del gráfico de Helm para la implementación

Para empezar, cree una carpeta llamada *lectura* y pegue el siguiente contenido de YAML en un nuevo archivo denominado *Chart.yml*.

```
apiVersion: v1
name: read
version: 1.0.0
description: A Helm chart to deploy the microsoft/cognitive-services-read to a Kubernetes cluster
```

Para configurar los valores predeterminados del gráfico de Helm, copie y pegue el contenido de YAML en un archivo denominado `values.yaml`. Reemplace los comentarios `# {ENDPOINT_URI}` y `# {API_KEY}` por sus propios valores.

```
# These settings are deployment specific and users can provide customizations

read:
  enabled: true
  image:
    name: cognitive-services-read
    registry: containerpreview.azurecr.io/
    repository: microsoft/cognitive-services-read
    tag: latest
    pullSecret: containerpreview # Or an existing secret
  args:
    eula: accept
    billing: # {ENDPOINT_URI}
    apikey: # {API_KEY}
```

IMPORTANT

Si no se proporcionan los valores `billing` y `apikey`, los servicios expiran pasados 15 minutos. Del mismo modo, se produce un error de comprobación, ya que los servicios no están disponibles.

Cree una carpeta *plantillas* en el directorio *lectura*. Copie y pegue el siguiente YAML en un archivo denominado `deployment.yaml`. El archivo `deployment.yaml` servirá como una plantilla de Helm.

Las plantillas generan archivos de manifiesto, que son descripciones de recursos con formato YAML que Kubernetes puede entender. [-Guía de plantilla de gráfico de Helm](#)

```

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: read
spec:
  template:
    metadata:
      labels:
        app: read-app
    spec:
      containers:
        - name: {{.Values.read.image.name}}
          image: {{.Values.read.image.registry}}{{.Values.read.image.repository}}
          ports:
            - containerPort: 5000
          env:
            - name: EULA
              value: {{.Values.read.image.args.eula}}
            - name: billing
              value: {{.Values.read.image.args.billing}}
            - name: apikey
              value: {{.Values.read.image.args.apikey}}
          imagePullSecrets:
            - name: {{.Values.read.image.pullSecret}}

---
apiVersion: v1
kind: Service
metadata:
  name: read
spec:
  type: LoadBalancer
  ports:
    - port: 5000
  selector:
    app: read-app

```

La plantilla especifica un servicio de equilibrador de carga y la implementación del contenedor o la imagen para lectura.

Paquete de Kubernetes (gráfico de Helm)

El *gráfico de Helm* contiene la configuración de las imágenes de Docker que se van a extraer del registro de contenedor `containerpreview.azurecr.io`.

Un [gráfico de Helm](#) es una colección de archivos que describen un conjunto relacionado de recursos de Kubernetes. Un solo gráfico se podría usar para implementar algo sencillo, como un pod almacenado en memoria, o complejo, como una pila de aplicación web completa con servidores HTTP, bases de datos, memorias caché, etc.

Los *gráficos de Helm* proporcionados extraen las imágenes de Docker del servicio Computer Vision y el servicio correspondiente del registro de contenedor `containerpreview.azurecr.io`.

Instalación del gráfico de Helm en el clúster de Kubernetes

Para instalar el *gráfico de Helm*, es necesario ejecutar el comando `helm install`. Asegúrese de ejecutar el comando de instalación desde el directorio situado encima de la carpeta `read`.

```
helm install read --name read
```


Este es el resultado de ejemplo que se puede ver tras una ejecución de instalación correcta:

```
NAME: read
LAST DEPLOYED: Thu Sep 04 13:24:06 2019
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME                READY  STATUS             RESTARTS  AGE
read-57cb76bcf7-45sdh  0/1    ContainerCreating  0          0s

==> v1/Service
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
read     LoadBalancer 10.110.44.86   localhost    5000:31301/TCP   0s

==> v1beta1/Deployment
NAME      READY  UP-TO-DATE  AVAILABLE  AGE
read     0/1    1           0          0s
```

La implementación de Kubernetes puede tardar varios minutos en completarse. Para confirmar que los pods y los servicios se han implementado correctamente y están disponibles, ejecute el siguiente comando:

```
kubectl get all
```

Debería ver algo parecido al resultado siguiente:

```
kubectl get all
NAME                READY  STATUS             RESTARTS  AGE
pod/read-57cb76bcf7-45sdh  1/1    Running            0          17s

NAME                TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
service/kubernetes  ClusterIP    10.96.0.1     <none>       443/TCP          45h
service/read        LoadBalancer 10.110.44.86   localhost    5000:31301/TCP   17s

NAME                READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/read  1/1    1           1          17s

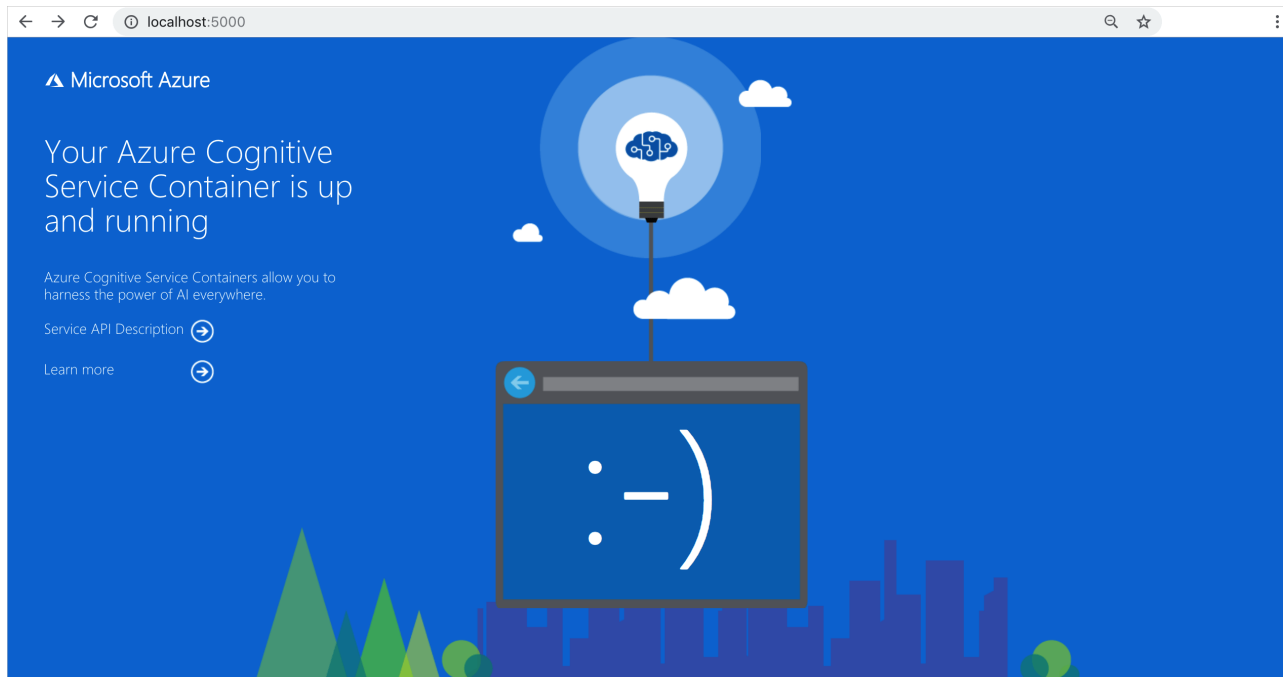
NAME                DESIRED  CURRENT  READY  AGE
replicaset.apps/read-57cb76bcf7  1        1        1      17s
```

Comprobación de que un contenedor está en ejecución

Hay varias maneras de comprobar que el contenedor está en ejecución. Busque la dirección *IP externa* y el puerto expuesto del contenedor en cuestión y abra el explorador web que prefiera. Use las distintas direcciones URL de solicitud para validar que el contenedor se está ejecutando. Las direcciones URL de solicitud de ejemplo que se enumeran a continuación son `http://localhost:5000`, pero el contenedor específico puede variar. Tenga en cuenta que va a confiar en la *dirección IP externa* y el puerto expuesto del contenedor.

URL DE LA SOLICITUD	PROPÓSITO
<code>http://localhost:5000/</code>	El contenedor ofrece una página principal.
<code>http://localhost:5000/status</code>	Se solicitó con HTTP GET, para comprobar que el contenedor está en ejecución sin causar una consulta al punto de conexión. Esta solicitud se puede usar con los sondeos de ejecución y preparación de Kubernetes.

URL DE LA SOLICITUD	PROPÓSITO
<code>http://localhost:5000/swagger</code>	El contenedor cuenta con un completo conjunto de documentación sobre los puntos de conexión y una característica de prueba . Esta característica le permite especificar la configuración en un formulario HTML basado en web y realizar la consulta sin necesidad de escribir código. Una vez que la consulta devuelve resultados, se proporciona un ejemplo del comando CURL para mostrar los encabezados HTTP y el formato de cuerpo requeridos.



Pasos siguientes

Para obtener más información sobre la instalación de aplicaciones con Helm en Azure Kubernetes Service (AKS), [visite esta página](#).

[Contenedores de Cognitive Services](#)

Implementar un contenedor de Computer Vision en Azure Container Instances

16/01/2020 • 10 minutes to read • [Edit Online](#)

Aprenda a implementar el contenedor de [Computer Vision](#) de Cognitive Services en una instancia de [Azure Container Instances](#). Este procedimiento muestra la creación del recurso de Computer Vision. Luego se trata la extracción de la imagen de contenedor asociada. Por último, se resalta la posibilidad de aprovechar la orquestación de los dos desde un explorador. El uso de contenedores puede desviar la atención de los desarrolladores de la administración de la infraestructura y centrarla en el desarrollo de aplicaciones.

Requisitos previos

- Use una suscripción de Azure. Si no tiene una suscripción a Azure, cree una [cuenta gratuita](#) antes de empezar.
- Instale la [CLI de Azure](#) (az).
- [Motor de docker](#) y se asegura de que la CLI de Docker funciona en una ventana de consola.

Solicitud de acceso al registro de contenedor privado

Rellene y envíe el [formulario de solicitud de contenedores de visión de Cognitive Services](#) para solicitar acceso al contenedor. El formulario solicita información acerca del usuario y de su empresa, así como del escenario de usuario para el que se va a usar el contenedor. Después de enviar el formulario, el equipo de Azure Cognitive Services lo revisa para asegurarse de que cumple los criterios de acceso al registro de contenedor privado.

IMPORTANT

Debe usar una dirección de correo electrónico asociada con una cuenta de Microsoft (MSA) o de Azure Active Directory (Azure AD) en el formulario.

Si se aprueba la solicitud, recibirá un correo electrónico con instrucciones que describen cómo obtener las credenciales y acceder al registro de contenedor privado.

Inicio de sesión en el registro de contenedor privado

Hay varias maneras de autenticarse con el registro de contenedor privado para los contenedores de Cognitive Services. Se recomienda que utilice el método de línea de comandos mediante la [CLI de Docker](#).

Use el comando [docker login](#), como se muestra en el ejemplo siguiente, para iniciar sesión en

`containerpreview.azurecr.io`, que es el registro de contenedor privado de los contenedores de Cognitive Services.

Reemplace `<username>` por el nombre de usuario y `<password>` por la contraseña proporcionada en las credenciales que recibió del equipo de Azure Cognitive Services.

```
docker login containerpreview.azurecr.io -u <username> -p <password>
```

Si ha protegido las credenciales en un archivo de texto, puede concatenar el contenido del archivo de texto con el comando `docker login`. Use el comando `cat`, como se muestra en el siguiente ejemplo: Sustituya `<passwordFile>` por la ruta y el nombre del archivo de texto que contiene la contraseña. Reemplace `<username>` por el nombre de usuario proporcionado en las credenciales.

```
cat <passwordFile> | docker login containerpreview.azurecr.io -u <username> --password-stdin
```

Creación de un recurso de Computer Vision

1. Inicie sesión en el [Portal de Azure](#).
2. Haga clic en [Crear un recurso de Computer Vision](#).
3. Establezca todas las opciones de configuración necesarias:

CONFIGURACIÓN	VALOR
NOMBRE	Nombre que quiera (2-64 caracteres).
Subscription	Seleccione una suscripción adecuada.
Location	Seleccione cualquier ubicación disponible cercana.
Nivel de precios	<input type="text" value="F0"/> : el plan de tarifa mínimo.
Grupo de recursos	Seleccione un grupo de recursos disponible.

4. Haga clic en **Crear** y espere a que el recurso se cree. Una vez creado, vaya a la página de recursos.
5. Recopile los elementos `{ENDPOINT_URI}` y `{API_KEY}` configurados. Consulte [recopilación los parámetros obligatorios](#) para obtener más información.

Creación de un recurso de instancia de contenedor de Azure con la CLI de Azure

El siguiente código YAML define el recurso de instancia de contenedor de Azure. Copie y pegue el contenido en un nuevo archivo llamado `my-aci.yaml` y reemplace los valores comentados por los suyos propios. Consulte el [formato de plantilla](#) para ver el código YAML válido. Consulte las [imágenes y los repositorios de contenedor](#) para ver los nombres de imagen disponibles y su repositorio correspondiente. Para más información sobre la referencia de YAML para las instancias de contenedor, vea [referencia de YAML: Azure Container Instances](#).

```

apiVersion: 2018-10-01
location: # < Valid location >
name: # < Container Group name >
imageRegistryCredentials: # This is required when pulling a non-public image
  - server: containerpreview.azurecr.io
    username: # < The username for the preview container registry >
    password: # < The password for the preview container registry >
properties:
  containers:
  - name: # < Container name >
    properties:
      image: # < Repository/Image name >
      environmentVariables: # These env vars are required
        - name: eula
          value: accept
        - name: billing
          value: # < Service specific Endpoint URL >
        - name: apikey
          value: # < Service specific API key >
      resources:
        requests:
          cpu: 4 # Always refer to recommended minimal resources
          memoryInGb: 8 # Always refer to recommended minimal resources
      ports:
        - port: 5000
    osType: Linux
    volumes: # This node, is only required for container instances that pull their model in at runtime, such as
      LUIS.
      - name: aci-file-share
        azureFile:
          shareName: # < File share name >
          storageAccountName: # < Storage account name>
          storageAccountKey: # < Storage account key >
    restartPolicy: OnFailure
    ipAddress:
      type: Public
      ports:
        - protocol: tcp
          port: 5000
    tags: null
type: Microsoft.ContainerInstance/containerGroups

```

NOTE

No todas las ubicaciones tienen la misma disponibilidad de CPU y memoria. Consulte la tabla [ubicación y recursos](#) para obtener una lista de los recursos disponibles para los contenedores por ubicación y sistema operativo.

Nos basaremos en el archivo YAML que hemos creado para el comando `az container create`. En la CLI de Azure, ejecute el comando `az container create` reemplazando `<resource-group>` por el suyo propio. Además, para proteger los valores dentro de una implementación de YAML, consulte cómo [proteger los valores](#).

```
az container create -g <resource-group> -f my-aci.yaml
```

La salida del comando es `Running...` si es válido. Después de unos minutos, la salida cambia a una cadena JSON que representa el recurso ACI recién creado. Es muy probable que la imagen de contenedor no esté disponible durante un tiempo, pero el recurso ya está implementado.

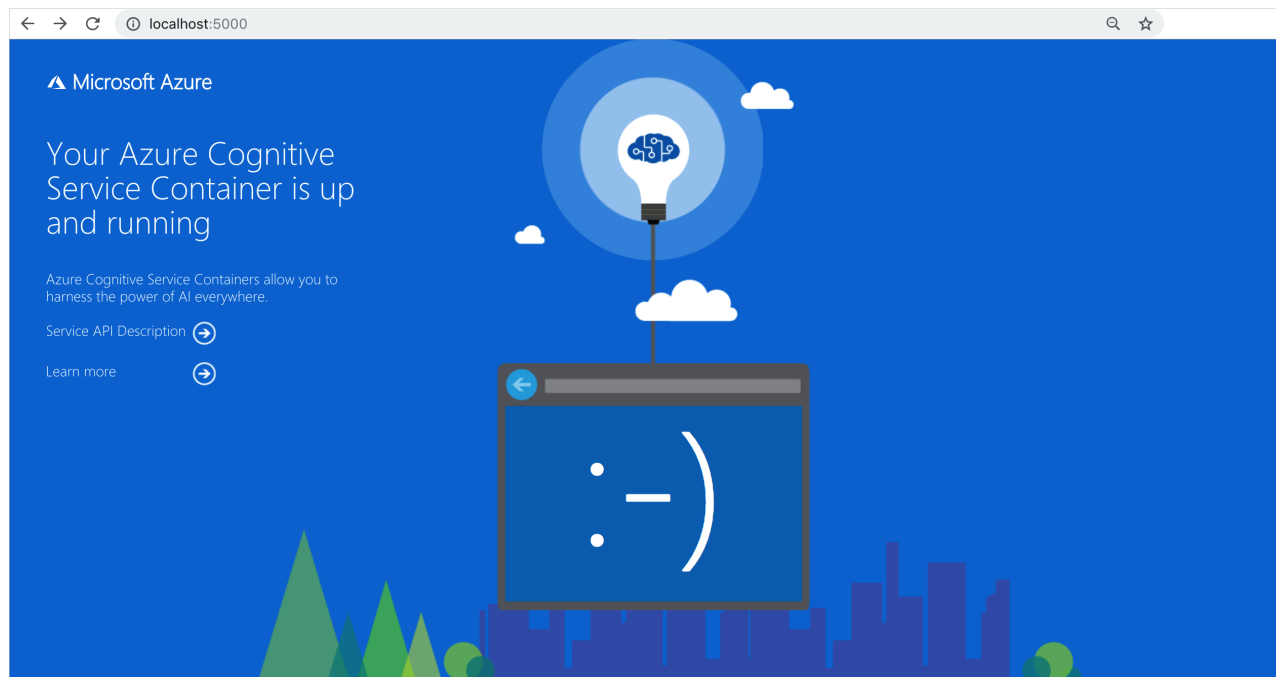
TIP

Preste mucha atención a las ubicaciones de las ofertas de la versión preliminar pública de Azure Cognitive Services, ya que el código YAML tendrá que ajustarse según la ubicación.

Comprobación de que un contenedor está en ejecución

Hay varias maneras de comprobar que el contenedor está en ejecución. Busque la dirección *IP externa* y el puerto expuesto del contenedor en cuestión y abra el explorador web que prefiera. Use las distintas direcciones URL de solicitud para validar que el contenedor se está ejecutando. Las direcciones URL de solicitud de ejemplo que se enumeran a continuación son `http://localhost:5000`, pero el contenedor específico puede variar. Tenga en cuenta que va a confiar en la *dirección IP externa* y el puerto expuesto del contenedor.

URL DE LA SOLICITUD	PROPÓSITO
<code>http://localhost:5000/</code>	El contenedor ofrece una página principal.
<code>http://localhost:5000/status</code>	Se solicitó con HTTP GET, para comprobar que el contenedor está en ejecución sin causar una consulta al punto de conexión. Esta solicitud se puede usar con los sondeos de ejecución y preparación de Kubernetes.
<code>http://localhost:5000/swagger</code>	El contenedor cuenta con un completo conjunto de documentación sobre los puntos de conexión y una característica de prueba . Esta característica le permite especificar la configuración en un formulario HTML basado en web y realizar la consulta sin necesidad de escribir código. Una vez que la consulta devuelve resultados, se proporciona un ejemplo del comando CURL para mostrar los encabezados HTTP y el formato de cuerpo requeridos.



Pasos siguientes

Vamos a seguir trabajando con los contenedores de Azure Cognitive Services.

[Usar otros contenedores de Cognitive Services](#)

Uso de servicios conectados en Visual Studio para conectarse a Computer Vision API

13/01/2020 • 9 minutes to read • [Edit Online](#)

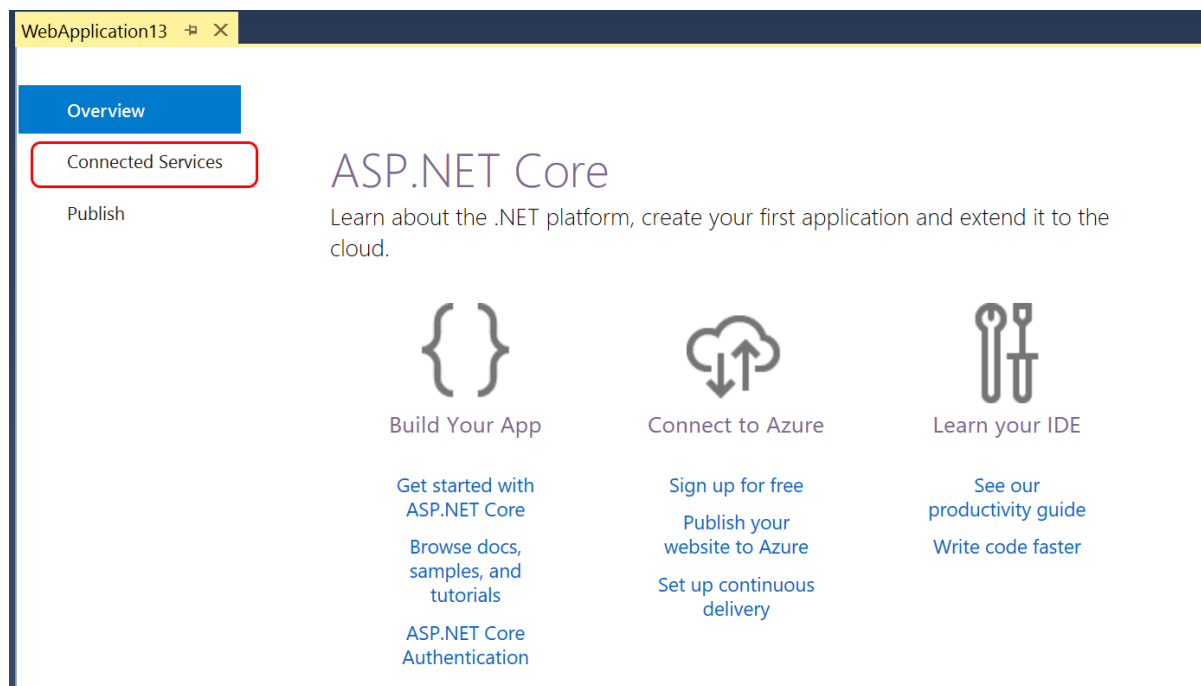
En este artículo y sus artículos complementarios se proporcionan detalles sobre el uso de la característica Servicio conectado de Visual Studio para Computer Vision API de Cognitive Services. La funcionalidad está disponible en Visual Studio 2017 15.7 o posterior, con la extensión Cognitive Services instalada.

Requisitos previos

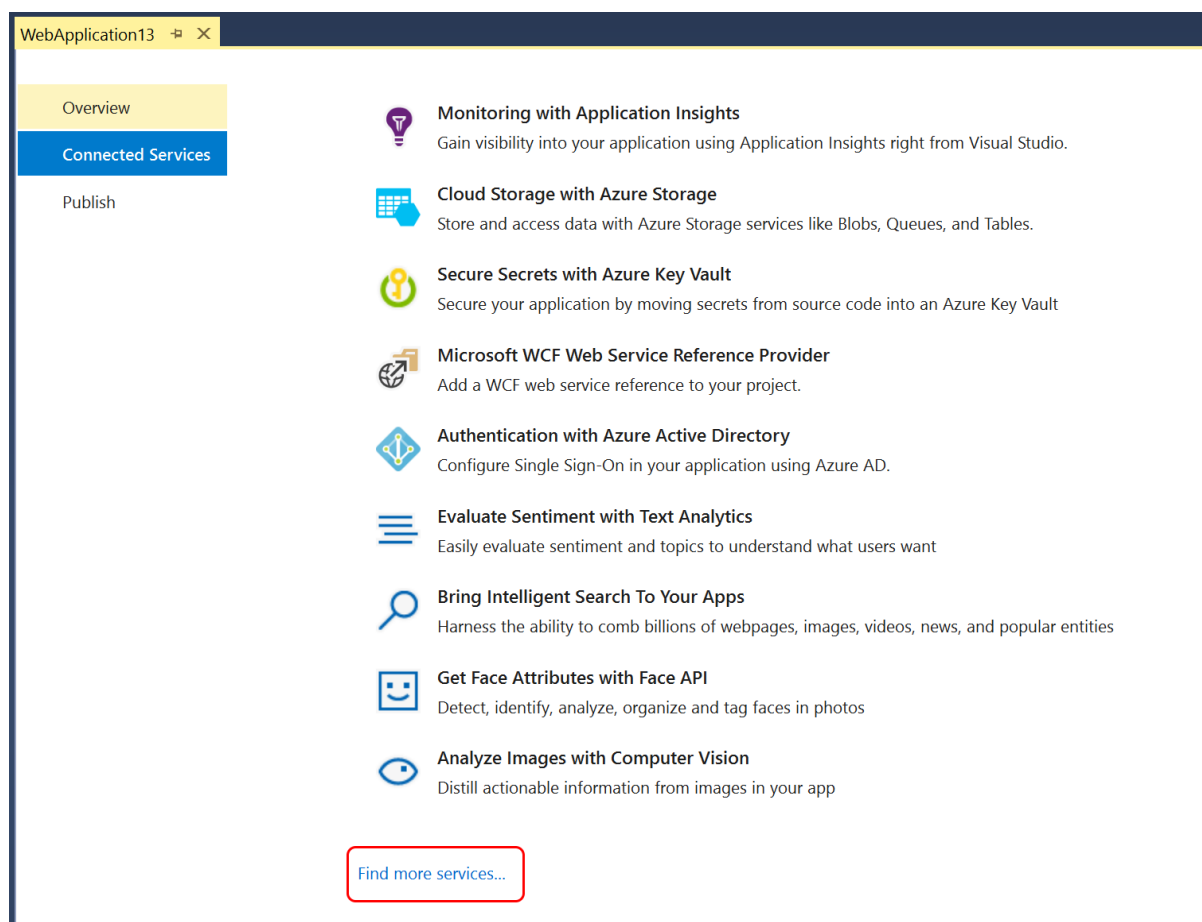
- Una suscripción de Azure. Si no tiene, puede registrarse para obtener una [cuenta gratuita](#).
- Visual Studio 2017, versión 15.7 o posterior, con la carga de trabajo **Desarrollo web** instalada. [Descárguelo ahora](#).

Instalar la extensión de Cognitive Services VSIX

1. Después de abrir el proyecto web en Visual Studio, elija la pestaña **Servicios conectados**. La pestaña está disponible en la página principal que aparece al abrir un nuevo proyecto. Si no ve la pestaña, seleccione **Servicios conectados** en el proyecto en el Explorador de soluciones.

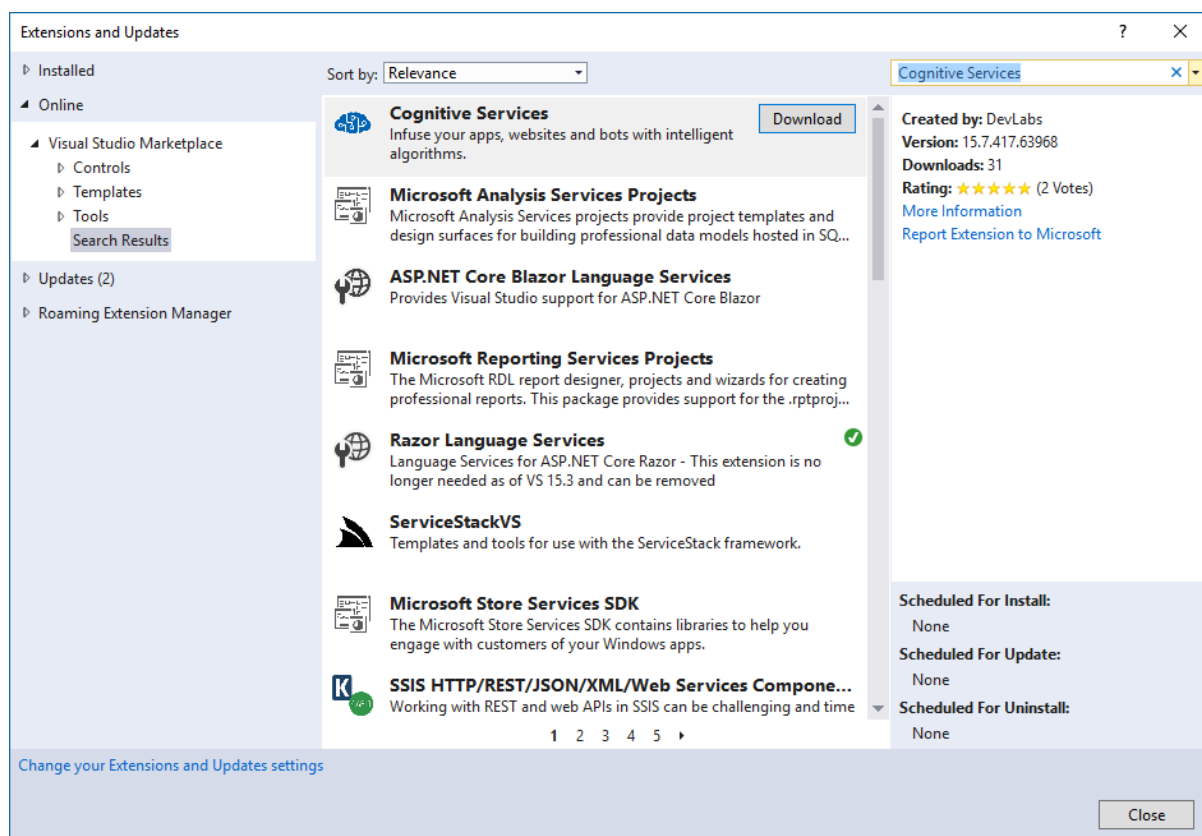


2. Desplácese hasta la parte inferior de la lista de servicios y seleccione **Find more services** (Buscar más servicios).



Aparecerá el cuadro de diálogo **Extensiones y actualizaciones**.

3. En el cuadro de diálogo **Extensiones y actualizaciones**, busque **Cognitive Services** y, luego, descargue e instale el paquete Cognitive Services VSIX.



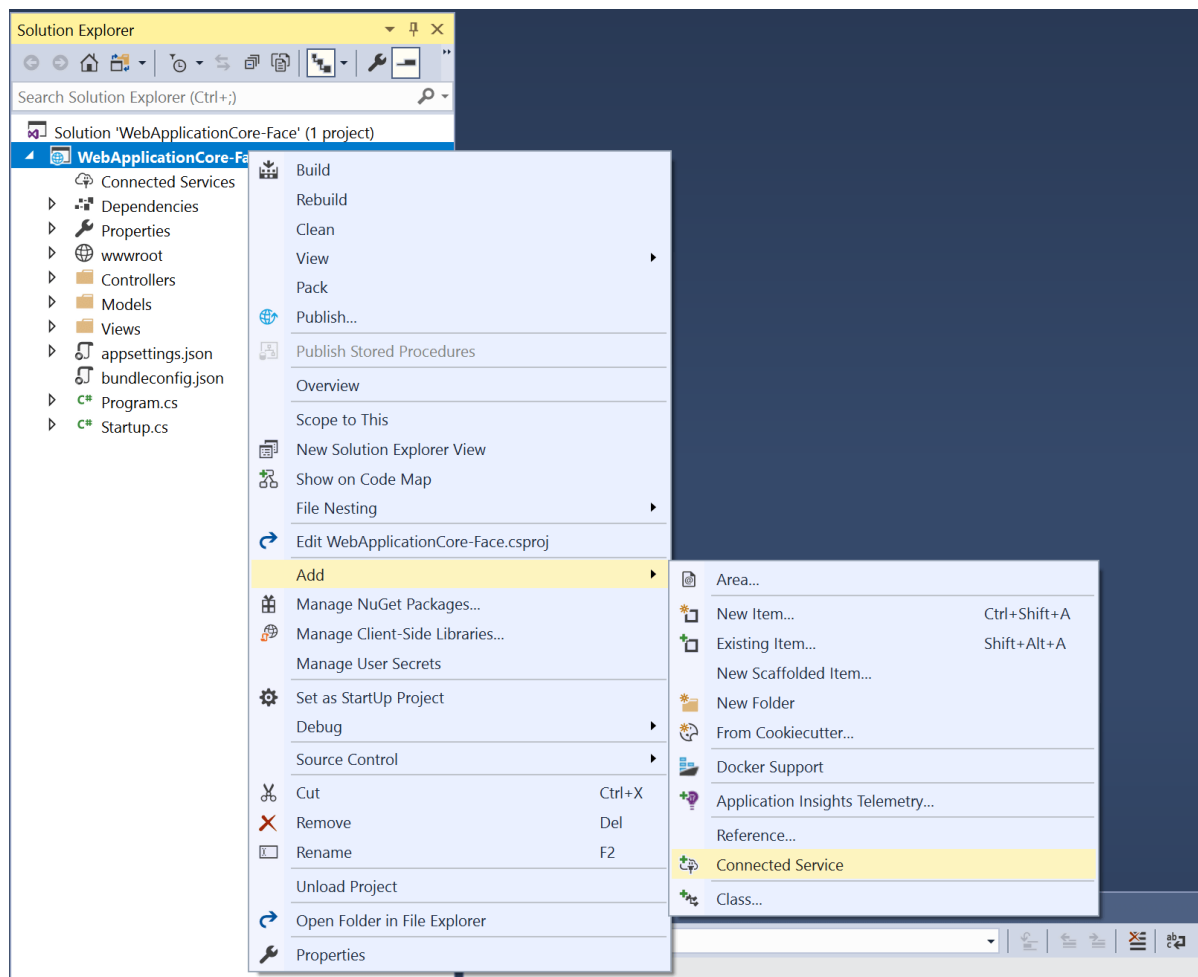
La instalación de una extensión requiere un reinicio del entorno de desarrollo integrado (IDE).

4. Reinicie Visual Studio. La extensión se instala cuando cierra Visual Studio y está disponible la próxima vez

que inicia el IDE.

Adición de compatibilidad al proyecto para Computer Vision API de Cognitive Services

1. Cree un proyecto web ASP.NET Core. Use la plantilla de proyecto Vacío.
2. En el **Explorador de soluciones**, elija **Agregar > Servicio conectado**. Aparece la página del servicio conectado con los servicios que puede agregar al proyecto.



3. En el menú de servicios disponibles, elija **Cognitive Services Computer Vision API**.

WebApplication-Core-ComputerVision










Overview

Connected Services

Publish

Connected Services

Add code and dependencies for one of these services to your application

-  **Monitoring with Application Insights**
Gain visibility into your application using Application Insights right from Visual Studio.
-  **Cloud Storage with Azure Storage**
Store and access data with Azure Storage services like Blobs, Queues, and Tables.
-  **Secure Secrets with Azure Key Vault**
Secure your application by moving secrets from source code into an Azure Key Vault
-  **Microsoft WCF Web Service Reference Provider**
Add a WCF web service reference to your project.
-  **Analyze Images with Computer Vision**
Distill actionable information from images in your app.
-  **Evaluate Sentiment with Text Analytics**
Easily evaluate sentiment and topics to understand what users want
-  **Bring Intelligent Search To Your Apps**
Harness the ability to comb billions of webpages, images, videos, news, and popular entities.
-  **Get Face Attributes with Face API**
Detect, identify, analyze, organize and tag faces in photos
-  **Translate Content with Speech Service**
Convert spoken audio into text, use voice for verification, or add speaker recognition to your app.

Si ha iniciado sesión en Visual Studio y tiene una suscripción de Azure asociada a su cuenta, aparece una página con una lista desplegable con las suscripciones.

WebApplicationCor...omputer Vision API WebApplicationCore-Cog1

Computer Vision API

Distill actionable information from images in your app.

Subscription: **1** Microsoft Azure Internal Consumption

Name: WebApplicationCore-Cog1_ComputerVisionAPI (ne **2** Edit...

Adding an Azure Computer Vision API will:

- Create a new Computer Vision API in resource group 'WebApplicationCore-Cog1_rg' in Australia East.
- Create a new resource group to host your Computer Vision API
- Use the pricing tier F0 - Free
- <What you can do with a cognitive service>

[More About Computer Vision API](#)

[Review pricing](#)

Add

4. Seleccione la suscripción que desea utilizar y, a continuación, elija un nombre para Computer Vision API, o elija el vínculo Editar para modificar el nombre generado automáticamente, elija el grupo de recursos y el plan de tarifa.

✕

Edit Azure Computer Vision API

Name:

WebApplicationCore-Cog1_ComputerVisionAPI

Resource Group:

WebApplicationCore-Cog1_rg (new) ▾

Location:

Australia East ▾

Pricing tier:

F0 - Free ▾

[Review pricing](#)

OK

Siga el vínculo para obtener más información sobre los planes de tarifa.

5. Elija Agregar para agregar compatibilidad con el servicio conectado. Visual Studio modifica su proyecto para agregar paquetes NuGet, las entradas del archivo de configuración y otros cambios para admitir una conexión con Computer Vision API. La Ventana de salida muestra el registro de lo que sucede en el proyecto. Debe ver algo parecido a lo siguiente:

```
[4/26/2018 5:15:31.664 PM] Adding Computer Vision API to the project.
[4/26/2018 5:15:32.084 PM] Creating new ComputerVision...
[4/26/2018 5:15:32.153 PM] Creating new Resource Group...
[4/26/2018 5:15:40.286 PM] Installing NuGet package
'Microsoft.Azure.CognitiveServices.Vision.ComputerVision' version 2.1.0.
[4/26/2018 5:15:44.117 PM] Retrieving keys...
[4/26/2018 5:15:45.602 PM] Changing appsettings.json setting: ComputerVisionAPI_ServiceKey=<service key>
[4/26/2018 5:15:45.606 PM] Changing appsettings.json setting:
ComputerVisionAPI_ServiceEndPoint=https://australiaeast.api.cognitive.microsoft.com/vision/v2.1
[4/26/2018 5:15:45.609 PM] Changing appsettings.json setting: ComputerVisionAPI_Name=WebApplication-
Core-ComputerVision_ComputerVisionAPI
[4/26/2018 5:15:46.747 PM] Successfully added Computer Vision API to the project.
```

Uso de Computer Vision API para detectar los atributos de una imagen

1. Agregue lo siguiente usando las instrucciones de Startup.cs.

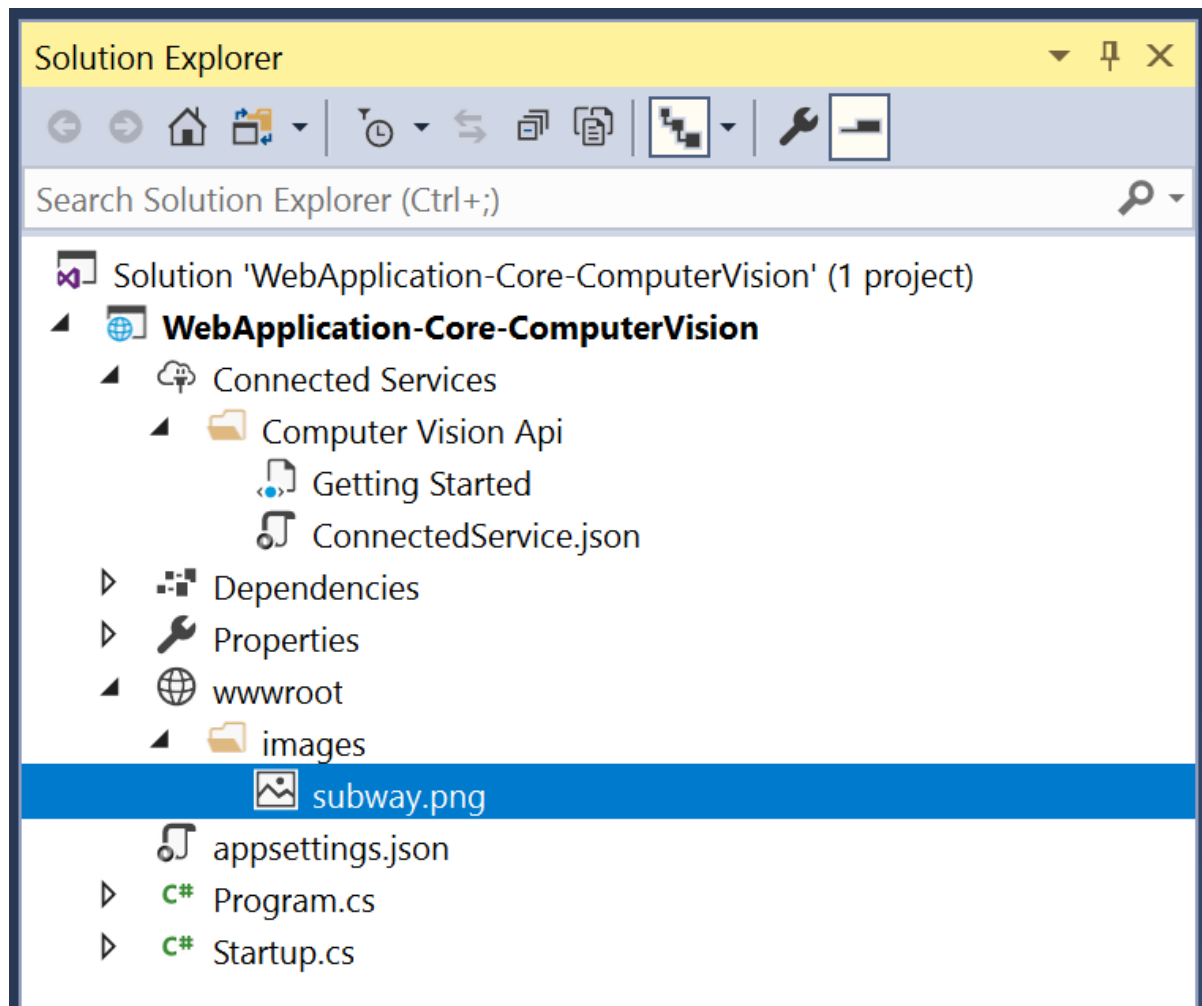
```
using System.IO;
using System.Text;
using Microsoft.Extensions.Configuration;
using System.Net.Http;
using System.Net.Http.Headers;
```

2. Agregue un campo de configuración y después un constructor que inicializa el campo de configuración de la clase `Startup` para habilitar la configuración del programa.

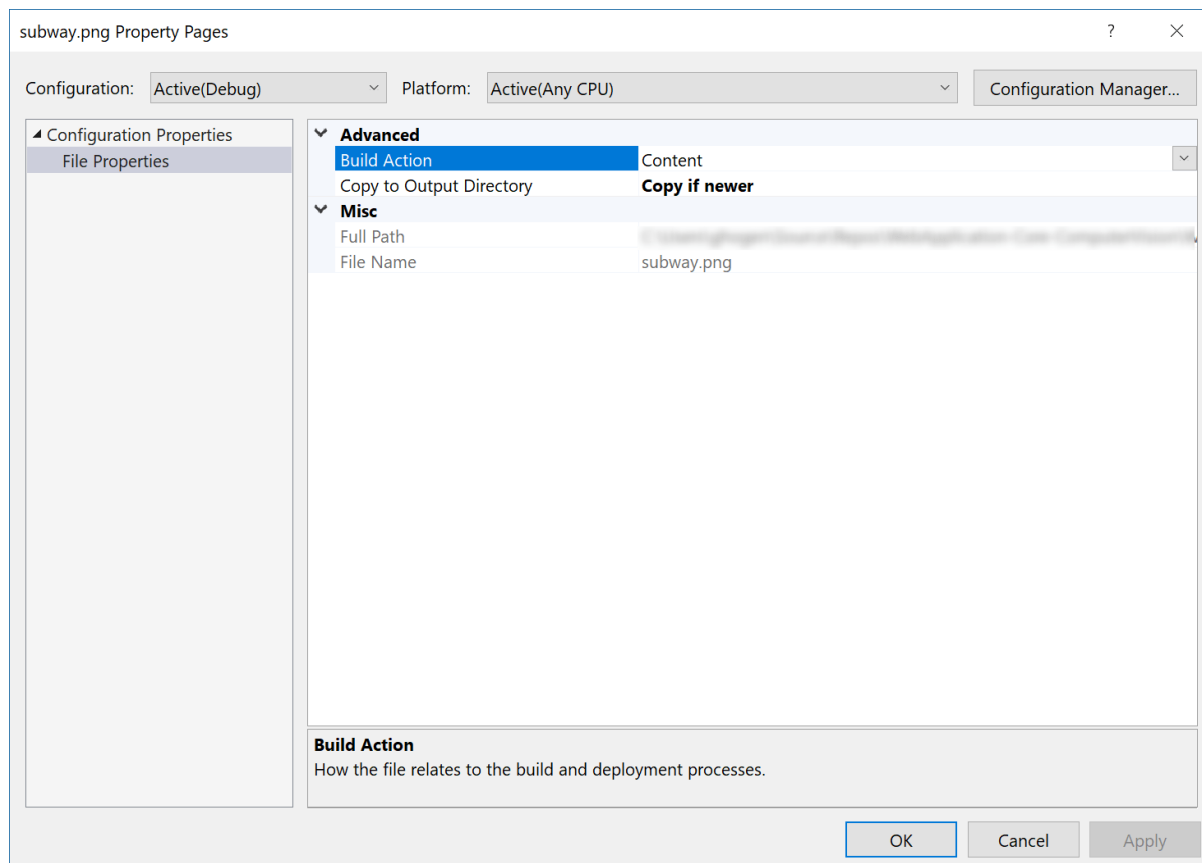
```
private IConfiguration configuration;

public Startup(IConfiguration configuration)
{
    this.configuration = configuration;
}
```

3. En la carpeta wwwroot del proyecto, agregue una carpeta de imágenes y agregue un archivo de imagen a la carpeta wwwroot. Por ejemplo, puede usar una de las imágenes en esta [página de Computer Vision API](#). Haga clic con el botón derecho en una de las imágenes, guárdela en la unidad de disco duro local y, a continuación, en el Explorador de soluciones, haga clic con el botón derecho en la carpeta imágenes y seleccione **Agregar > Elemento existente** para agregarlo al proyecto. El proyecto debe tener un aspecto similar al siguiente en el Explorador de soluciones:



4. Haga clic con el botón derecho en el archivo de imagen, elija Propiedades y luego seleccione **Copiar si es posterior**.



5. Reemplace el método de configuración con el código siguiente para acceder a Computer Vision API y probar una imagen.

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // TODO: Change this to your image's path on your site.
    string imagePath = @"images/subway.png";

    // Enable static files such as image files.
    app.UseStaticFiles();

    string visionApiKey = this.configuration["ComputerVisionAPI_ServiceKey"];
    string visionApiEndPoint = this.configuration["ComputerVisionAPI_ServiceEndPoint"];

    HttpClient client = new HttpClient();

    // Request headers.
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", visionApiKey);

    // Request parameters. A third optional parameter is "details".
    string requestParameters = "visualFeatures=Categories,Description,Color&language=en";

    // Assemble the URI for the REST API Call.
    string uri = visionApiEndPoint + "/analyze" + "?" + requestParameters;

    HttpResponseMessage response;

    // Request body. Posts an image you've added to your site's images folder.
    var fileInfo = env.WebRootFileProvider.GetFileInfo(imagePath);
    byte[] byteData = GetImageAsByteArray(fileInfo.PhysicalPath);

    string contentString = string.Empty;
    using (ByteArrayContent content = new ByteArrayContent(byteData))
    {
        // This example uses content type "application/octet-stream".
        // The other content types you can use are "application/json" and "multipart/form-data".
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

        // Execute the REST API call.
        response = client.PostAsync(uri, content).Result;

        // Get the JSON response.
        contentString = response.Content.ReadAsStringAsync().Result;
    }

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("<h1>Cognitive Services Demo</h1>");
        await context.Response.WriteAsync($"<p><b>Test Image:</b></p>");
        await context.Response.WriteAsync($"<div><img src=\"\" + imagePath + "\" /></div>");
        await context.Response.WriteAsync($"<p><b>Computer Vision API results:</b></p>");
        await context.Response.WriteAsync("<p>");
        await context.Response.WriteAsync(JsonPrettyPrint(contentString));
        await context.Response.WriteAsync("<p>");
    });
}
```

Este código construye una solicitud HTTP con el URI y la imagen como contenido binario de una llamada a Computer Vision API de REST.

6. Agregue las funciones auxiliares GetImageAsByteArray y JsonPrettyPrint.

```
/// <summary>
```

```

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}

/// <summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "    ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {
            case '{':
            case '[':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                }
                break;
            case '}':
            case ']':
                if (!quoted)
                {
                    sb.AppendLine();
                }
                sb.Append(ch);
                break;
            case '"':
                sb.Append(ch);
                bool escaped = false;
                var index = i;
                while (index > 0 && json[--index] == '\\')
                    escaped = !escaped;
                if (!escaped)
                    quoted = !quoted;
                break;
            case ',':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                }
                break;
            case ':':
                sb.Append(ch);
                if (!quoted)
                    sb.Append(" ");
                break;
        }
    }
    return sb.ToString();
}

```

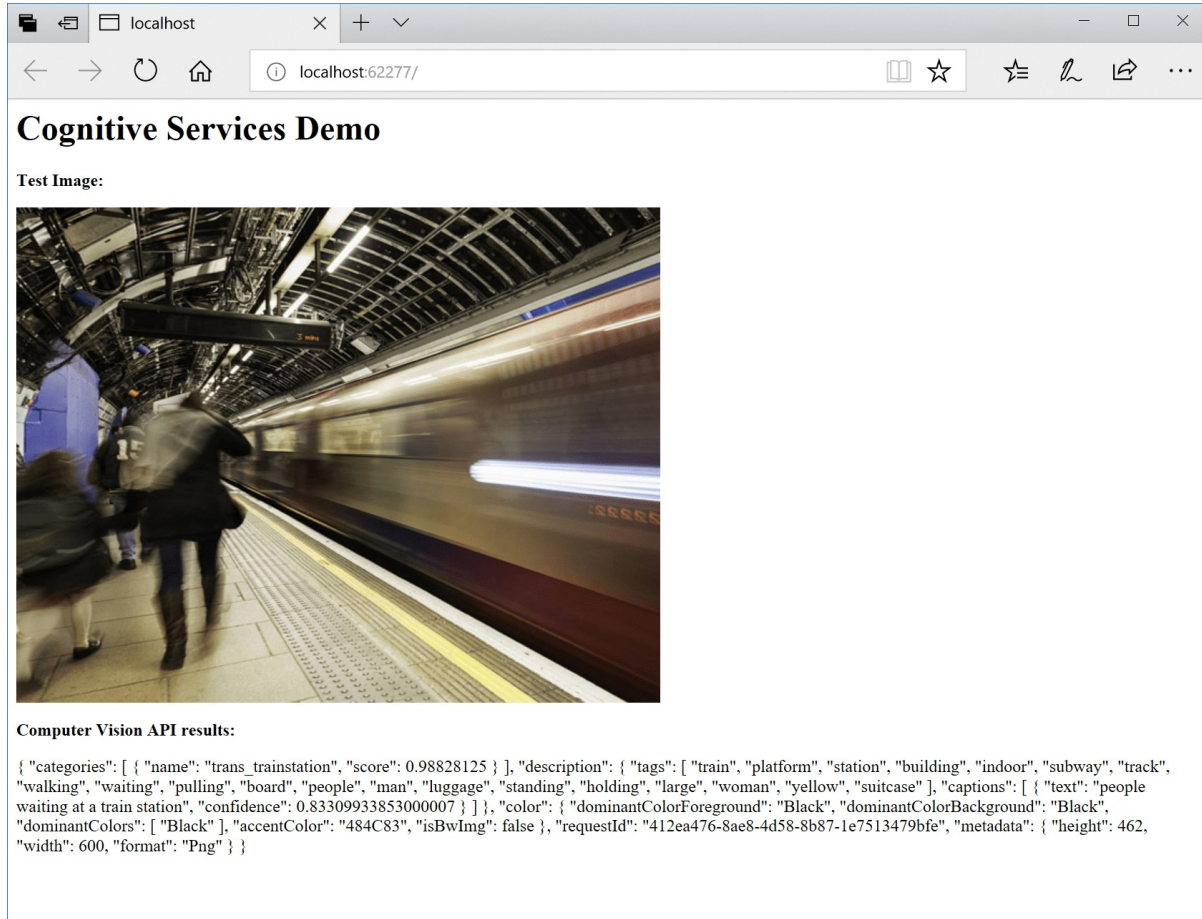


```

        break;
    default:
        sb.Append(ch);
        break;
    }
}
return sb.ToString();
}


```

7. Ejecute la aplicación web y vea qué encontró Computer Vision API en la imagen.



Cognitive Services Demo

Test Image:



Computer Vision API results:

```

{
  "categories": [ { "name": "trans_trainstation", "score": 0.98828125 } ],
  "description": {
    "tags": [ "train", "platform", "station", "building", "indoor", "subway", "track",
    "walking", "waiting", "pulling", "board", "people", "man", "luggage", "standing", "holding", "large", "woman", "yellow", "suitcase" ],
    "captions": [ { "text": "people waiting at a train station", "confidence": 0.83309933853000007 } ] },
  "color": {
    "dominantColorForeground": "Black",
    "dominantColorBackground": "Black",
    "dominantColors": [ "Black" ],
    "accentColor": "484C83",
    "isBwImg": false },
  "requestId": "412ea476-8ae8-4d58-8b87-1e7513479bfe",
  "metadata": { "height": 462, "width": 600, "format": "Png" } }

```

Limpieza de recursos

Cuando ya no necesite el grupo de recursos, elimínelo. De esta forma, se eliminan la instancia de Cognitive Services y los recursos relacionados. Para eliminar el grupo de recursos mediante el portal:

1. Escriba el nombre del grupo de recursos en el cuadro de búsqueda de la parte superior del portal. Cuando vea el grupo de recursos que se utiliza en esta guía de inicio rápido en los resultados de búsqueda, selecciónelo.
2. Seleccione **Eliminar grupo de recursos**.
3. En el cuadro **ESCRIBA EL NOMBRE DEL GRUPO DE RECURSOS**: escriba el nombre del grupo de recursos y seleccione **Eliminar**.

Pasos siguientes

Para más información sobre Computer Vision API, lea la [documentación de Computer Vision API](#).

Análisis de vídeo casi en tiempo real

16/01/2020 • 13 minutes to read • [Edit Online](#)

En este artículo se muestra cómo realizar análisis casi en tiempo real de fotogramas procedentes de una secuencia de vídeo en directo mediante la API Computer Vision. Los elementos básicos de este análisis son:

- Obtención de fotogramas desde un origen de vídeo.
- Selección de los fotogramas que se van a analizar.
- Envío de estos fotogramas a la API.
- Consumo de todos los resultados del análisis que se devuelven de la llamada API.

Los ejemplos de este artículo están escritos en C#. Para acceder al código, vaya a la página [Ejemplo de análisis de fotogramas de vídeo](#) en GitHub.

Enfoques para ejecutar análisis casi en tiempo real

Se pueden usar varios enfoques para solucionar el problema de la ejecución de análisis casi en tiempo real en secuencias de vídeo. En este artículo se describen tres de ellos, de menos a más sofisticado.

Diseño de un bucle infinito

El diseño más sencillo para el análisis casi en tiempo real es un bucle infinito. En cada iteración de este bucle, se obtiene un fotograma, se analiza y, después, se consume el resultado:

```
while (true)
{
    Frame f = GrabFrame();
    if (ShouldAnalyze(f))
    {
        AnalysisResult r = await Analyze(f);
        ConsumeResult(r);
    }
}
```

Si su análisis tuviera que consistir en un algoritmo de cliente ligero, este enfoque sería adecuado. Sin embargo, cuando el análisis se produce en la nube, la latencia resultante implica que una llamada API podría tardar varios segundos. Durante este tiempo no se capturan imágenes y el subproceso básicamente no realiza ninguna acción. La velocidad de fotogramas máxima está limitada por la latencia de las llamadas API.

Permitir la ejecución en paralelo de las llamadas API

Aunque se puede usar un bucle simple de un único subproceso en un algoritmo de cliente ligero, no se ajusta bien a la latencia de una llamada API en la nube. La solución a este problema es permitir que la llamada API de ejecución prolongada se ejecute en paralelo con la captura de los fotogramas. En C#, esto se puede hacer mediante un paralelismo basado en tareas. Por ejemplo, puede ejecutar el siguiente código:

```
while (true)
{
    Frame f = GrabFrame();
    if (ShouldAnalyze(f))
    {
        var t = Task.Run(async () =>
        {
            AnalysisResult r = await Analyze(f);
            ConsumeResult(r);
        })
    }
}
```

Con este enfoque, cada análisis se inicia en una tarea independiente. La tarea puede ejecutarse en segundo plano mientras se siguen capturando nuevos fotogramas. El enfoque evita el bloqueo del subproceso principal mientras se espera la devolución de una llamada API. Sin embargo, el enfoque puede presentar ciertas desventajas:

- Desaparecen algunas de las garantías que proporcionaba la versión simple. En otras palabras, se pueden producir varias llamadas API en paralelo y los resultados pueden devolverse en el orden equivocado.
- También podría suceder que varios subprocesos especificaran la función `ConsumeResult()` al mismo tiempo, lo que podría ser peligroso si la función no es segura para subprocesos.
- Por último, este código simple no realiza un seguimiento de las tareas que se crean, por lo que las excepciones desaparecen de manera silenciosa. Por consiguiente, es preciso agregar un subproceso "consumidor" que realice un seguimiento de las tareas del análisis, genere excepciones, termine tareas de larga duración y garantice que los resultados se consumen en el orden correcto y de uno en uno.

Diseño de un sistema productor-consumidor

Para el enfoque final, el diseño de un sistema "productor-consumidor", se crea un subproceso de productor que es similar al bucle infinito mencionado anteriormente. Sin embargo, en lugar de consumir los resultados del análisis en cuanto están disponibles, el productor simplemente coloca las tareas en una cola para hacer un seguimiento de ellas.

```

// Queue that will contain the API call tasks.
var taskQueue = new BlockingCollection<Task<ResultWrapper>>();

// Producer thread.
while (true)
{
    // Grab a frame.
    Frame f = GrabFrame();

    // Decide whether to analyze the frame.
    if (ShouldAnalyze(f))
    {
        // Start a task that will run in parallel with this thread.
        var analysisTask = Task.Run(async () =>
        {
            // Put the frame, and the result/exception into a wrapper object.
            var output = new ResultWrapper(f);
            try
            {
                output.Analysis = await Analyze(f);
            }
            catch (Exception e)
            {
                output.Exception = e;
            }
            return output;
        })

        // Push the task onto the queue.
        taskQueue.Add(analysisTask);
    }
}

```

También se crea un subproceso de consumidor, que saca las tareas de la cola, espera a que finalicen y muestra el resultado, o bien genera la excepción que se ha iniciado. El uso de la cola permite garantizar que los resultados se consumen uno a uno, en el orden correcto y sin limitar la velocidad de fotogramas máxima del sistema.

```

// Consumer thread.
while (true)
{
    // Get the oldest task.
    Task<ResultWrapper> analysisTask = taskQueue.Take();

    // Wait until the task is completed.
    var output = await analysisTask;

    // Consume the exception or result.
    if (output.Exception != null)
    {
        throw output.Exception;
    }
    else
    {
        ConsumeResult(output.Analysis);
    }
}

```

Implementación de la solución

Introducción rápida

Para ayudarle a poner en marcha su aplicación lo antes posible, hemos implementado el sistema que se describe en la sección anterior. Se pretende que tenga la flexibilidad suficiente para albergar muchos escenarios, aunque es

fácil de usar. Para acceder al código, vaya a la página [Ejemplo de análisis de fotogramas de vídeo](#) en GitHub.

La biblioteca contiene la clase `FrameGrabber`, que implementa el sistema productor-consumidor que se ha descrito anteriormente para procesar los fotogramas de vídeo de una cámara web. Los usuarios pueden especificar la forma exacta de la llamada API y la clase usa eventos para que el código de la llamada sepa cuándo se adquiere un nuevo fotograma o cuándo hay disponible un nuevo resultado del análisis.

Para ilustrar algunas de las posibilidades, se proporcionan dos aplicaciones de ejemplo que utilizan la biblioteca.

La primera es una aplicación de consola sencilla que captura fotogramas de la cámara web predeterminada y los envía a Face API para la detección de caras. En el código siguiente se reproduce una versión simplificada de la aplicación:

```
using System;
using System.Linq;
using Microsoft.Azure.CognitiveServices.Vision.Face;
using Microsoft.Azure.CognitiveServices.Vision.Face.Models;
using VideoFrameAnalyzer;

namespace BasicConsoleSample
{
    internal class Program
    {
        const string ApiKey = "<your API key>";
        const string Endpoint = "https://<your API region>.api.cognitive.microsoft.com";

        private static async Task Main(string[] args)
        {
            // Create grabber.
            FrameGrabber<DetectedFace[]> grabber = new FrameGrabber<DetectedFace[]>();

            // Create Face API Client.
            FaceClient faceClient = new FaceClient(new ApiKeyServiceClientCredentials(ApiKey))
            {
                Endpoint = Endpoint
            };

            // Set up a listener for when we acquire a new frame.
            grabber.NewFrameProvided += (s, e) =>
            {
                Console.WriteLine($"New frame acquired at {e.Frame.Metadata.Timestamp}");
            };

            // Set up a Face API call.
            grabber.AnalysisFunction = async frame =>
            {
                Console.WriteLine($"Submitting frame acquired at {frame.Metadata.Timestamp}");
                // Encode image and submit to Face API.
                return (await
                    faceClient.Face.DetectWithStreamAsync(frame.Image.ToMemoryStream(".jpg"))).ToArray();
            };

            // Set up a listener for when we receive a new result from an API call.
            grabber.NewResultAvailable += (s, e) =>
            {
                if (e.TimedOut)
                    Console.WriteLine("API call timed out.");
                else if (e.Exception != null)
                    Console.WriteLine("API call threw an exception.");
                else
                    Console.WriteLine($"New result received for frame acquired at {e.Frame.Metadata.Timestamp}.
                    {e.Analysis.Length} faces detected");
            };

            // Tell grabber when to call the API.
            // See also TriggerAnalysisOnPredicate
        }
    }
}
```

```

grabber.TriggerAnalysisOnInterval(TimeSpan.FromMilliseconds(3000));

// Start running in the background.
await grabber.StartProcessingCameraAsync();

// Wait for key press to stop.
Console.WriteLine("Press any key to stop..");
Console.ReadKey();

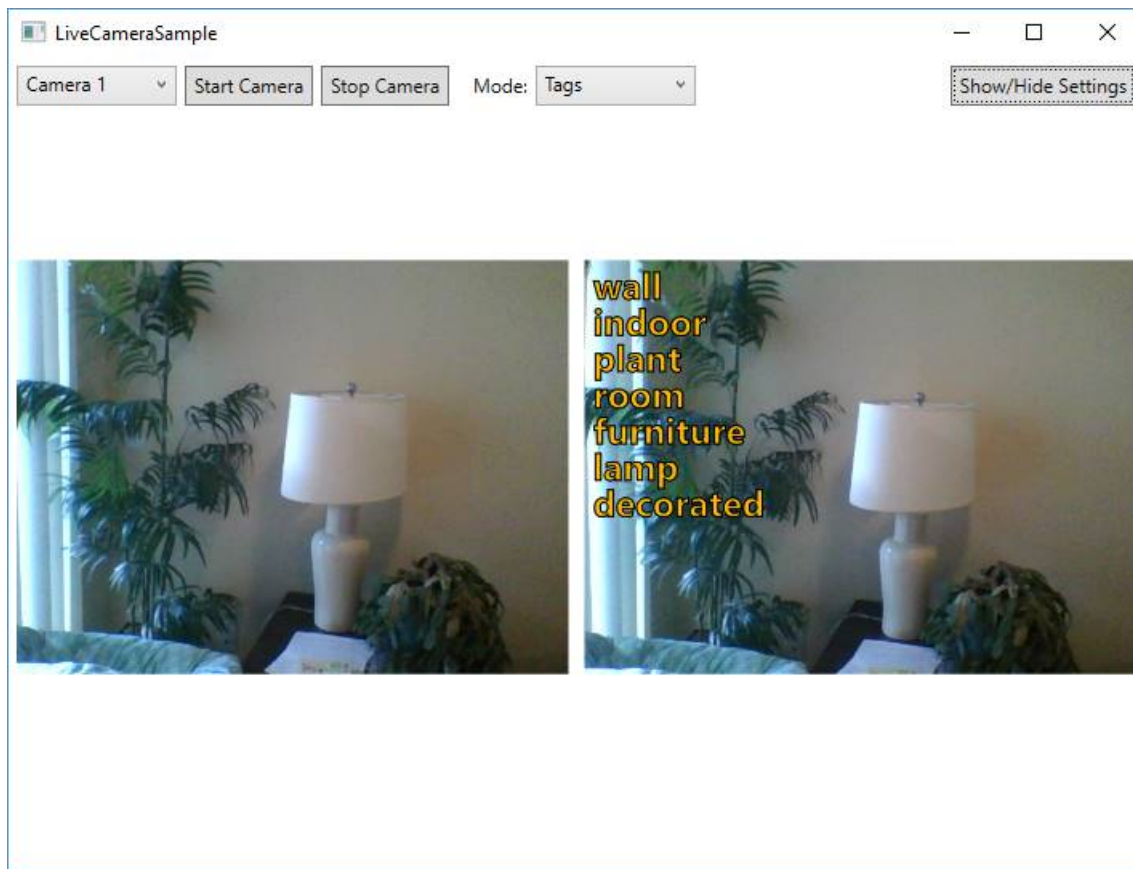
// Stop, blocking until done.
await grabber.StopProcessingAsync();
    }
}
}

```

La segunda aplicación de ejemplo es algo más interesante. Permite elegir la API a la que se llama en los fotogramas de vídeo. A la izquierda, la aplicación muestra una vista previa del vídeo en directo. A la derecha, superpone el resultado de la API más reciente en el fotograma correspondiente.

En la mayoría de los modos, hay retraso visible entre el vídeo en directo de la izquierda y el análisis que se ve a la derecha. Este retraso es el tiempo que tarda en realizarse la llamada API. Una excepción es el modo "EmotionsWithClientFaceDetect", que realiza la detección de caras localmente en el equipo cliente mediante OpenCV antes de enviar las imágenes a Azure Cognitive Services.

Con este enfoque, puede visualizar la cara detectada de inmediato. Luego, puede actualizar las emociones, después de que se devuelva la llamada API. Esto demuestra la posibilidad de un enfoque "híbrido", es decir, se puede realizar un procesamiento simple en el cliente y, después, pueden usarse Cognitive Services APIs para intensificar este procesamiento con un análisis más avanzado si es necesario.



Integración de los ejemplos en el código base

Para empezar con este ejemplo, realice estas acciones:

1. Obtenga las claves de la API de Vision API de [Suscripciones](#). Para el análisis de fotogramas de vídeo, las API correspondientes son:

- [API Computer Vision](#)
- [API Face](#)

2. Clone el repositorio [Cognitive-Samples-VideoFrameAnalysis](#) de GitHub.
3. Abra el ejemplo en Visual Studio 2015, o cualquier versión posterior, y después, compile y ejecute las aplicaciones de ejemplo:
 - Para BasicConsoleSample, la clave de Face API está codificado de forma rígida directamente en [BasicConsoleSample/Program.cs](#).
 - En el caso de LiveCameraSample, escriba las claves en el panel **Configuración** de la aplicación. Las claves se mantienen de una sesión a otra como datos de usuario.

Cuando esté listo para integrar los ejemplos, haga referencia a la biblioteca VideoFrameAnalyzer desde sus propios proyectos.

Las funcionalidades de comprensión de imágenes, voces, vídeo y texto de VideoFrameAnalyzer utilizan Azure Cognitive Services. Microsoft recibirá las imágenes, el audio, el vídeo y los demás datos que cargue (a través de esta aplicación) y puede usarlos para mejorar el servicio. Pedimos su ayuda para proteger a las personas cuyos datos envíe la aplicación a Azure Cognitive Services.

Resumen

En este artículo, ha aprendido a ejecutar análisis casi en tiempo real de secuencias de vídeo en directo mediante API Face y API Computer Vision. También ha aprendido a usar el código de ejemplo para empezar. Para empezar a compilar su aplicación mediante las claves de API gratuitas, vaya a la [página de registro de Azure Cognitive Services](#).

No dude en realizar comentarios y sugerencias en el [repositorio de GitHub](#). Para proporcionar comentarios de API más amplios, visite nuestro [sitio de UserVoice](#).

Sample: Exploración de una aplicación de procesamiento de imágenes con C#

13/01/2020 • 39 minutes to read • [Edit Online](#)

Explore una aplicación Windows básica que utiliza Computer Vision para realizar el reconocimiento óptico de caracteres (OCR), crear miniaturas de recorte inteligente, además de detectar, clasificar, etiquetar y describir las características visuales, incluidas las caras, en una imagen. El ejemplo siguiente le permite enviar la dirección URL de una imagen o un archivo almacenado localmente. Puede usar este ejemplo de código abierto como plantilla para crear su propia aplicación para Windows con Computer Vision API y Windows Presentation Foundation (WPF), una parte de .NET Framework.

- Obtener la aplicación de ejemplo de GitHub
- Abrir y compilar la aplicación de ejemplo en Visual Studio
- Ejecutar la aplicación de ejemplo e interactuar con ella para desarrollar diversos escenarios
- Explorar los distintos escenarios incluidos con la aplicación de ejemplo

Requisitos previos

Antes de explorar la aplicación de ejemplo, asegúrese de haber cumplido los requisitos previos siguientes:

- Debe tener [Visual Studio 2015](#) o posterior.
- Debe tener una clave de suscripción para Computer Vision. Puede obtener una clave de evaluación gratuita en la página [Pruebe Cognitive Services](#). O bien, siga las instrucciones que se indican en [Creación de una cuenta de Cognitive Services](#) para suscribirse a Computer Vision y obtener su clave. Tome nota también de la dirección URL del punto de conexión del servicio.

Obtención de la aplicación de ejemplo

La aplicación de ejemplo de Computer Vision está disponible en GitHub desde el repositorio

`Microsoft/Cognitive-Vision-Windows`. Este repositorio también incluye el repositorio

`Microsoft/Cognitive-Common-Windows` como submódulo de GIT. También puede clonar de forma recursiva este repositorio, incluido el submódulo, ya sea mediante el comando `git clone --recurse-submodules` desde la línea de comandos, o mediante el escritorio de GitHub.

Por ejemplo, para clonar de forma recursiva el repositorio para la aplicación de ejemplo de Computer Vision desde un símbolo del sistema, ejecute el siguiente comando:

```
git clone --recurse-submodules https://github.com/Microsoft/Cognitive-Vision-Windows.git
```

IMPORTANT

No descargue este repositorio como un archivo ZIP. GIT no incluye submódulos cuando descarga un repositorio como un archivo ZIP.

Obtener imágenes de ejemplo opcionales

Si quiere, puede usar las imágenes de ejemplo incluidas con la aplicación de ejemplo de [Face](#), disponible en GitHub desde el repositorio `Microsoft/Cognitive-Face-Windows`. Esa aplicación de ejemplo incluye una carpeta, `/Data`, que contiene varias imágenes de personas. Así mismo, puede clonar de forma recursiva este repositorio

mediante los métodos descritos para la aplicación de ejemplo de Computer Vision.

Por ejemplo, para clonar de forma recursiva el repositorio para la aplicación de ejemplo de Face desde un símbolo del sistema, ejecute el siguiente comando:

```
git clone --recurse-submodules https://github.com/Microsoft/Cognitive-Face-Windows.git
```

Abrir y compilar la aplicación de ejemplo en Visual Studio

En primer lugar, debe compilar la aplicación de ejemplo para que Visual Studio pueda resolver las dependencias, antes de ejecutar o explorar la aplicación de ejemplo. Para abrir y compilar la aplicación de ejemplo, realice los pasos siguientes:

1. Abra el archivo de solución de Visual Studio, `/Sample-WPF/VisionAPI-WPF-Samples.sln`, en Visual Studio.
2. Asegúrese de que la solución de Visual Studio contenga dos proyectos:
 - SampleUserControlLibrary
 - VisionAPI-WPF-Samples

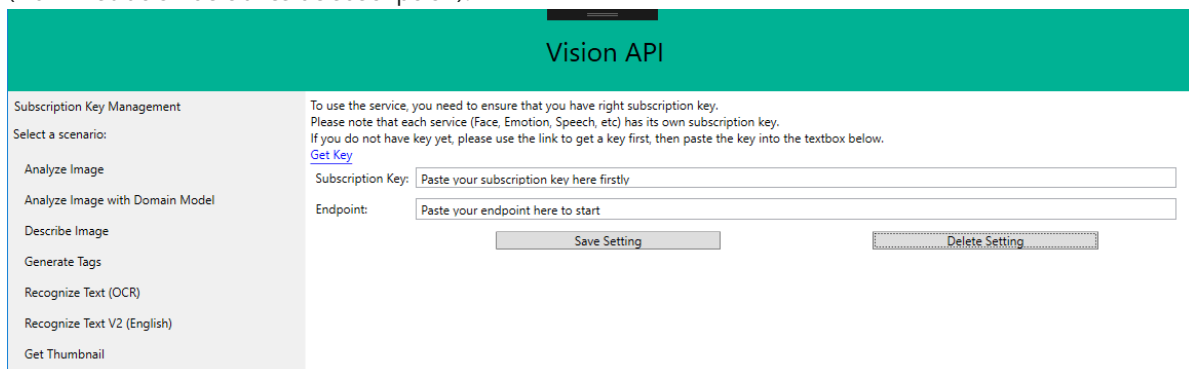
Si no está disponible el proyecto SampleUserControlLibrary, confirme que ha clonado de forma recursiva el repositorio `Microsoft/Cognitive-Vision-Windows`.

3. En Visual Studio, presione Control + Mayús + B o elija la opción **Compilar** en el menú de la cinta de opciones y, a continuación, elija **Compilar solución** para compilar la solución.

Ejecutar e interactuar con la aplicación de ejemplo

Puede ejecutar la aplicación de ejemplo para ver cómo interactúa con usted y con la biblioteca cliente de Computer Vision al realizar varias tareas, como generar miniaturas o etiquetar imágenes. Para ejecutar e interactuar con la aplicación de ejemplo, realice los pasos siguientes:

1. Una vez completada la compilación, presione **F5** o elija **Depurar** en el menú de la cinta de opciones y, a continuación, elija **Iniciar depuración** para ejecutar la aplicación de ejemplo.
2. Cuando se muestre la aplicación de ejemplo, elija **Subscription Key Management** (Administración de claves de suscripción) en el panel de navegación para mostrar la página Subscription Key Management (Administración de claves de suscripción).



(Administración de claves de suscripción)

3. Escriba la clave de suscripción en **Clave de suscripción**.
4. Escriba la dirección URL del punto de conexión en **Punto de conexión**.
Por ejemplo, si va a usar la clave de suscripción de la evaluación gratuita de Computer Vision, escriba la siguiente dirección URL del punto de conexión: `https://westcentralus.api.cognitive.microsoft.com`.

NOTE

Los nuevos recursos creados después del 1 de julio de 2019 usarán nombres de subdominio personalizados. Para más información y para obtener una lista completa de los puntos de conexión regionales, consulte [Nombres de subdominios personalizados para Cognitive Services](#).

- Si no quiere escribir la clave de suscripción y la URL del punto de conexión la próxima vez que ejecute la aplicación de ejemplo, elija **Guardar configuración** para guardar la clave de suscripción y la URL del punto de conexión en el equipo. Si quiere eliminar la dirección URL del punto de conexión y la clave de suscripción que guardó previamente, elija **Eliminar configuración**.

NOTE

La aplicación de ejemplo usa el almacenamiento aislado y `System.IO.IsolatedStorage` para almacenar la dirección URL del punto de conexión y la clave de suscripción.

- En la opción **Seleccione un escenario** del panel de navegación, seleccione uno de los escenarios que actualmente se incluyen con la aplicación de ejemplo:

ESCENARIO	DESCRIPCIÓN
Analyze Image	Usa la operación Analyze Image para analizar una imagen local o remota. Puede elegir las características visuales y el idioma para el análisis y ver la imagen y los resultados.
Analyze Image with Domain Model	Usa la operación List Domain Specific Models para enumerar los modelos de dominio entre los que puede elegir, y la operación Recognize Domain Specific Content para analizar una imagen local o remota mediante el modelo de dominio seleccionado. También puede elegir el idioma para el análisis.
Describe Image	Usa la operación Describe Image para crear una descripción legible de una imagen local o remota. También puede elegir el idioma para la descripción.
Generate Tags	Usa la operación Tag Image para etiquetar características visuales de una imagen local o remota. También puede elegir el idioma para las etiquetas.
Recognize Text (OCR)	Usa la operación OCR para reconocer y extraer texto impreso de una imagen. Puede elegir el idioma que se usará o permitir que Computer Vision lo detecte automáticamente.
Recognize Text V2 (English)	Usa las operaciones Recognize Text y Get Recognize Text Operation Result operaciones para reconocer y extraer texto manuscrito o impreso de una imagen de forma asíncrona.
Get Thumbnail	Usa la operación Get Thumbnail para generar una miniatura de una imagen local o remota.

En la captura de pantalla siguiente se muestra la página proporcionada para el escenario Analyze Image, después de analizar una imagen de ejemplo.

Vision API

Subscription Key Management

Select a scenario:

Analyze Image

Analyze Image with Domain Model

Describe Image

Generate Tags

Recognize Text (OCR)

Get Thumbnail

Analyze an Image

Please click either [Load Image] or paste in an image url and click [Analyze]

Load Image

<https://oxfordportal.blob.core.windows.net/vision/Analysis/1-1.jpg>

Analyze

Analyzing Done



[21:52:43.310111]: Description :
[21:52:43.386402]: Caption : a man swimming in a pool of water; Confidence : 0.752564820236237
[21:52:43.386402]: Tags : water, person, sport, swimming, pool,
[21:52:43.402029]: Tags :
[21:52:43.402029]: Name : water; Confidence : 0.999414682388306; Hint :
[21:52:43.402029]: Name : person; Confidence : 0.936775147914886; Hint :
[21:52:43.417652]: Name : sport; Confidence : 0.848687767982483; Hint :
[21:52:43.417652]: Name : swimming; Confidence : 0.845447421073914; Hint : sport
[21:52:43.433278]: Name : water sport; Confidence : 0.827535569667816; Hint : sport
[21:52:43.433278]: Name : pool; Confidence : 0.805495202541351; Hint :

Explorar la aplicación de ejemplo

La solución de Visual Studio para la aplicación de ejemplo de Computer Vision contiene dos proyectos:

- SampleUserControlLibrary

El proyecto SampleUserControlLibrary proporciona funcionalidades compartidas por varios ejemplos de Cognitive Services. El proyecto contiene lo siguiente:

- SampleScenarios

Control de usuario que proporciona una presentación estandarizada, como la barra de título, el panel de navegación y el panel de contenido, para obtener ejemplos. La aplicación de ejemplo de Computer Vision usa este control en la ventana de MainWindow.xaml para mostrar las páginas del escenario y acceder a la información que se comparte entre los escenarios, como la dirección URL del punto de conexión y la clave de suscripción.

- SubscriptionKeyPage

Página que proporciona un diseño estandarizado para escribir una clave de suscripción y una dirección URL de punto de conexión para la aplicación de ejemplo. La aplicación de ejemplo de Computer Vision usa esta página para administrar la clave de suscripción y la dirección URL de punto de conexión que se usan en las páginas del escenario.

- VideoResultControl

Control de usuario que proporciona una presentación estandarizada de la información de vídeo. La aplicación de ejemplo de Computer Vision no usa este control.

- VisionAPI-WPF-Samples

Proyecto principal de la aplicación de ejemplo de Computer Vision que contiene todas las funcionalidades interesantes para Computer Vision. El proyecto contiene lo siguiente:

- AnalyzeInDomainPage.xaml

Página del escenario Analyze Image con el escenario de modelo de dominio.

- AnalyzeImage.xaml
Página del escenario Analyze Image.
- DescribePage.xaml
Página del escenario Describe Image.
- ImageScenarioPage.cs
Clase ImageScenarioPage, de la que se derivan todas las páginas de escenario de la aplicación de ejemplo. Esta clase administra funcionalidades, como el suministro de credenciales y el formato de salida, que se comparten en todas las páginas de escenarios.
- MainWindow.xaml
Ventana principal de la aplicación de ejemplo. Utiliza el control SampleScenarios para presentar las páginas SubscriptionKeyPage y de escenarios.
- OCRPage.xaml
Página del escenario Recognize Text (OCR).
- RecognizeLanguage.cs
Clase RecognizeLanguage, que proporciona información acerca de los idiomas que admiten los distintos métodos de la aplicación de ejemplo.
- TagsPage.xaml
Página del escenario Generate Tags.
- TextRecognitionPage.xaml
Página del escenario Recognize Text V2 (OCR).
- ThumbnailPage.xaml
Página del escenario Get Thumbnail.

Exploración del código de ejemplo

Las partes importantes del código de ejemplo se enmarcan con los bloques de comentarios que comienzan por `KEY SAMPLE CODE STARTS HERE` y terminan por `KEY SAMPLE CODE ENDS HERE`, para que pueda explorar la aplicación de ejemplo con mayor facilidad. Estas partes importantes del código de ejemplo contienen el código pertinente para aprender a realizar diversas tareas con la biblioteca cliente de Computer Vision API. Puede buscar `KEY SAMPLE CODE STARTS HERE` en Visual Studio para moverse entre las secciones de código de la aplicación de ejemplo de Computer Vision más pertinentes.

Por ejemplo, en el método `UploadAndAnalyzeImageAsync`, que aparece a continuación y se incluye en `AnalyzePage.xaml`, se muestra cómo usar la biblioteca cliente para analizar una imagen local al invocar el método `ComputerVisionClient.AnalyzeImageInStreamAsync`.

```

private async Task<ImageAnalysis> UploadAndAnalyzeImageAsync(string imagePath)
{
    // -----
    // KEY SAMPLE CODE STARTS HERE
    // -----

    //
    // Create Cognitive Services Vision API Service client.
    //
    using (var client = new ComputerVisionClient(Credentials) { Endpoint = Endpoint })
    {
        Log("ComputerVisionClient is created");

        using (Stream imageFileStream = File.OpenRead(imageFilePath))
        {
            //
            // Analyze the image for all visual features.
            //
            Log("Calling ComputerVisionClient.AnalyzeImageInStreamAsync()...");
            VisualFeatureTypes[] visualFeatures = GetSelectedVisualFeatures();
            string language = (_language.SelectedItem as RecognizeLanguage).ShortCode;
            ImageAnalysis analysisResult = await client.AnalyzeImageInStreamAsync(imageFileStream,
visualFeatures, null, language);
            return analysisResult;
        }
    }

    // -----
    // KEY SAMPLE CODE ENDS HERE
    // -----
}

```

Explorar la biblioteca cliente

Esta aplicación de ejemplo usa la biblioteca cliente de Computer Vision API, un contenedor cliente de C# fino para Computer Vision API en Azure Cognitive Services. La biblioteca cliente está disponible en NuGet en el paquete [Microsoft.Azure.CognitiveServices.Vision.ComputerVision](#). Cuando compiló la aplicación de Visual Studio, recuperó la biblioteca cliente desde el paquete NuGet correspondiente. También puede ver el código fuente de la biblioteca cliente en la carpeta `/ClientLibrary` del repositorio `Microsoft/Cognitive-Vision-Windows`.

La funcionalidad de la biblioteca cliente se centra en la clase `ComputerVisionClient`, en el espacio de nombres `Microsoft.Azure.CognitiveServices.Vision.ComputerVision`, mientras que los modelos usados por la clase `ComputerVisionClient` al interactuar con Computer Vision se encuentran en el espacio de nombres `Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models`. En las distintas páginas de escenarios XAML incluidas con la aplicación de ejemplo, encontrará las siguientes directivas `using` para esos espacios de nombres:

```

// -----
// KEY SAMPLE CODE STARTS HERE
// Use the following namespace for ComputerVisionClient.
// -----
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;
// -----
// KEY SAMPLE CODE ENDS HERE
// -----

```

Obtendrá más información sobre los distintos métodos que se incluyen con la clase `ComputerVisionClient` a medida que explore los escenarios incluidos con la aplicación de ejemplo de Computer Vision.

Explorar el escenario Analyze Image

La página `AnalyzePage.xaml` administra este escenario. Puede elegir las características visuales y el idioma para el análisis y ver la imagen y los resultados. Para ello, la página del escenario usa uno de los métodos siguientes, en función del origen de la imagen:

- `UploadAndAnalyzeImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `ComputerVisionClient.AnalyzeImageInStreamAsync`.
- `AnalyzeUrlAsync`
Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `ComputerVisionClient.AnalyzeImageAsync`.

El método `UploadAndAnalyzeImageAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Dado que la aplicación de ejemplo analiza una imagen local, debe enviar el contenido de esa imagen a Computer Vision. Abre el archivo local especificado en `imageFilePath` para la lectura como `Stream` y, a continuación, obtiene las características visuales y el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.AnalyzeImageInStreamAsync`, en que transfiere `Stream` para el archivo, las características visuales y el idioma y, a continuación, devuelve el resultado como una instancia `ImageAnalysis`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `AnalyzeUrlAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Obtiene las características visuales y el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.AnalyzeImageInStreamAsync`, en que transfiere la URL de imagen, las características visuales y el idioma y, a continuación, devuelve el resultado como una instancia `ImageAnalysis`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

Explorar Analyze Image con el escenario de modelo de dominio.

La página `AnalyzeInDomainPage.xaml` administra este escenario. Puede elegir un modelo de dominio, como `celebrities` o `landmarks`, y el idioma para realizar un análisis específico del dominio de la imagen y ver la imagen y los resultados. La página del escenario usa los métodos siguientes, en función del origen de la imagen:

- `GetAvailableDomainModelsAsync`
Este método obtiene la lista de modelos de dominio disponibles de Computer Vision y rellena el control `ComboBox` `_domainModelComboBox` en la página, mediante el método `ComputerVisionClient.ListModelsAsync`.
- `UploadAndAnalyzeInDomainImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `ComputerVisionClient.AnalyzeImageByDomainInStreamAsync`.
- `AnalyzeInDomainUrlAsync`
Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `ComputerVisionClient.AnalyzeImageByDomainAsync`.

El método `UploadAndAnalyzeInDomainImageAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Dado que la aplicación de ejemplo analiza una imagen local, debe enviar el contenido de esa imagen a Computer Vision. Abre el archivo local especificado en `imageFilePath` para la lectura como `Stream` y, a continuación, obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.AnalyzeImageByDomainInStreamAsync`, en que transfiere `Stream` para el archivo, el nombre del modelo de dominio y el idioma y, a continuación, devuelve el resultado como una instancia `DomainModelResults`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `AnalyzeInDomainUrlAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL

del punto de conexión y la clave de suscripción especificadas. Obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.AnalyzeImageByDomainAsync`, en que transfiere la URL de imagen, las características visuales y el idioma y, a continuación, devuelve el resultado como una instancia `DomainModelResults`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

Explorar el escenario Describe Image

La página `DescribePage.xaml` administra este escenario. Puede elegir un idioma para crear una descripción inteligible de la imagen y ver la imagen y los resultados. La página del escenario usa los métodos siguientes, en función del origen de la imagen:

- `UploadAndDescribeImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `ComputerVisionClient.DescribeImageInStreamAsync`.
- `DescribeUrlAsync`
Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `ComputerVisionClient.DescribeImageAsync`.

El método `UploadAndDescribeImageAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Dado que la aplicación de ejemplo analiza una imagen local, debe enviar el contenido de esa imagen a Computer Vision. Abre el archivo local especificado en `imageFilePath` para la lectura como `Stream` y, a continuación, obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.DescribeImageInStreamAsync`, en que transfiere `Stream` para el archivo, el número máximo de candidatos (en este caso, 3) y el idioma y, a continuación, devuelve el resultado como una instancia `ImageDescription`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `DescribeUrlAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.DescribeImageAsync`, en que transfiere la URL de la imagen, el número máximo de candidatos (en este caso, 3) y el idioma y, a continuación, devuelve el resultado como una instancia `ImageDescription`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

Explorar el escenario Generate Tags

La página `TagsPage.xaml` administra este escenario. Puede elegir un idioma para etiquetar las características visuales de una imagen y ver la imagen y los resultados. La página del escenario usa los métodos siguientes, en función del origen de la imagen:

- `UploadAndGetTagsForImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `ComputerVisionClient.TagImageInStreamAsync`.
- `GenerateTagsForUrlAsync`
Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `ComputerVisionClient.TagImageAsync`.

El método `UploadAndGetTagsForImageAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Dado que la aplicación de ejemplo analiza una imagen local, debe enviar el contenido de esa imagen a Computer Vision. Abre el archivo local especificado en `imageFilePath` para la lectura como `Stream` y, a continuación, obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.TagImageInStreamAsync`, en que

transfiere `Stream` para el archivo y el idioma y, a continuación, devuelve el resultado como una instancia `TagResult`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `GenerateTagsForUrlAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.TagImageAsync`, en que transfiere la URL de la imagen y el idioma y, a continuación, devuelve el resultado como una instancia `TagResult`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

Explorar el escenario Recognize Text (OCR)

La página `OCRPage.xaml` administra este escenario. Puede elegir un idioma para reconocer y extraer texto impreso de una imagen y ver la imagen y los resultados. La página del escenario usa los métodos siguientes, en función del origen de la imagen:

- `UploadAndRecognizeImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `ComputerVisionClient.RecognizePrintedTextInStreamAsync`.
- `RecognizeUrlAsync`
Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `ComputerVisionClient.RecognizePrintedTextAsync`.

El método `UploadAndRecognizeImageAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Dado que la aplicación de ejemplo analiza una imagen local, debe enviar el contenido de esa imagen a Computer Vision. Abre el archivo local especificado en `imageFilePath` para la lectura como `Stream` y, a continuación, obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.RecognizePrintedTextInStreamAsync`, en que indica que la orientación no se detectó y transfiere `Stream` para el archivo y el idioma y, a continuación, devuelve el resultado como una instancia `OcrResult`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `RecognizeUrlAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Obtiene el idioma seleccionado en la página del escenario. Realiza una llamada al método `ComputerVisionClient.RecognizePrintedTextAsync`, en que indica que la orientación no se detectó y transfiere la URL de la imagen y el idioma y, a continuación, devuelve el resultado como una instancia `OcrResult`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

Explorar el escenario Recognize Text V2 (English)

La página `TextRecognitionPage.xaml` administra este escenario. Puede elegir el modo de reconocimiento y un idioma para reconocer y extraer texto manuscrito o impreso de una imagen de forma asíncrona y ver la imagen y los resultados. La página del escenario usa los métodos siguientes, en función del origen de la imagen:

- `UploadAndRecognizeImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `RecognizeAsync` y la transferencia de un delegado parametrizado para el método `ComputerVisionClient.RecognizeTextInStreamAsync`.
- `RecognizeUrlAsync`
Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `RecognizeAsync` y la transferencia de un delegado parametrizado para el método `ComputerVisionClient.RecognizeTextAsync`.

- `RecognizeAsync` Este método administra la llamada asincrónica para los métodos `UploadAndRecognizeImageAsync` y `RecognizeUrlAsync`, así como el sondeo de resultados mediante una llamada al método `ComputerVisionClient.GetTextOperationResultAsync`.

A diferencia de los otros escenarios incluidos en la aplicación de ejemplo de Computer Vision, este escenario es asincrónico, ya que se llama a un método para iniciar el proceso, pero se llama a otro método para comprobar el estado y devolver los resultados de ese proceso. El flujo lógico de este escenario es un poco distinto del de los otros escenarios.

El método `UploadAndRecognizeImageAsync` abre el archivo local especificado en `imageFilePath` para la lectura como `Stream` y, a continuación, realiza una llamada al método `RecognizeAsync`, en que transfiere:

- Una expresión lambda para un delegado asincrónico parametrizado del método `ComputerVisionClient.RecognizeTextInStreamAsync`, con el elemento `Stream` para el archivo y el modo de reconocimiento como parámetros, en `GetHeadersAsyncFunc`.
- Una expresión lambda para un delegado para obtener el valor de encabezado de respuesta `Operation-Location`, en `GetOperationUrlFunc`.

El método `RecognizeUrlAsync` realiza una llamada al método `RecognizeAsync`, en que transfiere:

- Una expresión lambda para un delegado asincrónico parametrizado del método `ComputerVisionClient.RecognizeTextAsync`, con la URL de la imagen remota y el modo de reconocimiento como parámetros, en `GetHeadersAsyncFunc`.
- Una expresión lambda para un delegado para obtener el valor de encabezado de respuesta `Operation-Location`, en `GetOperationUrlFunc`.

Cuando el método `RecognizeAsync` se completa, los métodos `UploadAndRecognizeImageAsync` y `RecognizeUrlAsync` devuelven el resultado como una instancia de `TextOperationResult`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `RecognizeAsync` realiza una llamada al delegado parametrizado para el método `ComputerVisionClient.RecognizeTextInStreamAsync` o `ComputerVisionClient.RecognizeTextAsync` transferido en `GetHeadersAsyncFunc` y espera la respuesta. A continuación, el método llama al delegado transferido en `GetOperationUrlFunc` para obtener el valor del encabezado de respuesta `Operation-Location` de la respuesta. Este valor es la dirección URL utilizada para recuperar los resultados del método transferido en `GetHeadersAsyncFunc` de Computer Vision.

A continuación, el método `RecognizeAsync` realiza una llamada al método `ComputerVisionClient.GetTextOperationResultAsync`, en que transfiere la URL recuperada del encabezado de respuesta `Operation-Location`, a fin de obtener el estado y el resultado del método transferido en `GetHeadersAsyncFunc`. Si el estado no indica que el método se completó, de manera correcta o incorrecta, el método `RecognizeAsync` realiza una llamada a `ComputerVisionClient.GetTextOperationResultAsync` 3 veces más, con un tiempo de espera de 3 segundos entre las llamadas. El método `RecognizeAsync` devuelve los resultados al método que lo llamó.

Explorar el escenario Get Thumbnail

La página `ThumbnailPage.xaml` administra este escenario. Puede indicar si quiere usar el recorte inteligente, especificar la altura y la anchura que quiera a fin de generar una miniatura de una imagen y ver la imagen y los resultados. La página del escenario usa los métodos siguientes, en función del origen de la imagen:

- `UploadAndThumbnailImageAsync`
Este método se usa para las imágenes locales, en las que la imagen se debe codificar como `Stream` y enviar a Computer Vision mediante una llamada al método `ComputerVisionClient.GenerateThumbnailInStreamAsync`.

- `ThumbnailUrlAsync`

Este método se usa para las imágenes remotas, en las que la URL de la imagen se envía a Computer Vision mediante una llamada al método `ComputerVisionClient.GenerateThumbnailAsync`.

El método `UploadAndThumbnailImageAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Dado que la aplicación de ejemplo analiza una imagen local, debe enviar el contenido de esa imagen a Computer Vision. Abre el archivo local especificado en `imageFilePath` para la lectura como `Stream`. Realiza una llamada al método `ComputerVisionClient.GenerateThumbnailInStreamAsync`, en que se transfiere la anchura, la altura, el elemento `Stream` para el archivo y si se utiliza el recorte inteligente y, a continuación, devuelve el resultado como `Stream`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

El método `RecognizeUrlAsync` crea una nueva instancia de `ComputerVisionClient` mediante la dirección URL del punto de conexión y la clave de suscripción especificadas. Realiza una llamada al método `ComputerVisionClient.GenerateThumbnailAsync`, en que se transfiere la anchura, la altura, la URL de la imagen y si se utiliza el recorte inteligente y, a continuación, devuelve el resultado como `Stream`. Los métodos heredados de la clase `ImageScenarioPage` presentan los resultados devueltos en la página del escenario.

Limpieza de recursos

Cuando ya no necesite la carpeta en que ha clonado el repositorio `Microsoft/Cognitive-Vision-Windows`, elimínela. Si optó por usar las imágenes de ejemplo, elimine también la carpeta en la que clonó el repositorio `Microsoft/Cognitive-Face-Windows`.

Pasos siguientes

[Introducción a Face API](#)

Preguntas frecuentes sobre Computer Vision API

13/01/2020 • 3 minutes to read • [Edit Online](#)

TIP

Si no puede encontrar respuestas a sus preguntas en estas P+F, puede plantearlas en la comunidad de Computer Vision API en [StackOverflow](#) o ponerse en contacto con [Ayuda y soporte técnico en UserVoice](#).

Pregunta: *¿Puedo entrenar a Computer Vision API para usar etiquetas personalizadas? Por ejemplo, me gustaría introducir imágenes de razas de gatos para "entrenar" a AI y luego recibir el valor de raza en una solicitud de AI.*

Respuesta: Esta función no está disponible actualmente. Sin embargo, nuestros ingenieros están trabajando para incorporar esta funcionalidad a Computer Vision.

Pregunta: *¿Se puede usar Computer Vision localmente sin una conexión a Internet?*

Respuesta: Actualmente no ofrecemos una solución local.

Pregunta: *¿Se puede usar Computer Vision para leer matrículas de entidad de almacén?*

Respuesta: Vision API ofrece una buena detección de texto con OCR, pero en este momento no está actualizada para matrículas de entidad de almacén. En nuestro esfuerzo constante por mejorar nuestros servicios, hemos agregado OCR para el reconocimiento automático de matrículas a nuestra lista de solicitudes de características.

Pregunta: *¿Qué tipos de superficies se admiten para el reconocimiento de escritura a mano?*

Respuesta: La tecnología funciona con distintas clases de superficies, como pizarras, notas del producto y notas rápidas.

Pregunta: *¿Cuánto tarda la operación de reconocimiento de escritura a mano?*

Respuesta: Depende de la longitud del texto. Con textos más largos, puede tardar hasta varios segundos. Por lo tanto, una vez finalizada la operación de reconocimiento de texto escrito a mano, puede que deba esperar antes de poder recuperar los resultados mediante la operación de obtención de resultados de la operación de texto escrito a mano.

Pregunta: *¿Cómo trata la tecnología de reconocimiento de escritura a mano el texto que se ha insertado mediante un signo de intercalación en medio de una línea?*

Respuesta: La operación de reconocimiento de escritura a mano devuelve dicho texto como una línea independiente.

Pregunta: *¿Cómo trata la tecnología de reconocimiento de escritura a mano las palabras o líneas tachadas?*

Respuesta: Si las palabras están tachadas con varias líneas para presentarlas como irreconocibles, la operación de reconocimiento de escritura a mano no las selecciona. Sin embargo, si las palabras están tachadas con una sola línea, ese tachado se trata como ruido y la operación de reconocimiento de escritura a mano sigue seleccionando las palabras.

Pregunta: *¿Qué orientación de texto se admite en la tecnología de reconocimiento de escritura a mano?*

Respuesta: La operación de reconocimiento de escritura a mano puede seleccionar texto orientado en ángulos de

entre 30 y 40 grados.

Taxonomía de 86 categorías de Computer Vision

13/01/2020 • 2 minutes to read • [Edit Online](#)

abstract_

abstract_net

abstract_nonphoto

abstract_rect

abstract_shape

abstract_texture

animal_

animal_bird

animal_cat

animal_dog

animal_horse

animal_panda

building_

building_arch

building_brickwall

building_church

building_corner

building_doorwindows

building_pillar

building_stair

building_street

dark_

drink_

drink_can

dark_fire

dark_fireworks

sky_object

food_

food_bread

food_fastfood
food_grilled
food_pizza
indoor_
indoor_churchwindow
indoor_court
indoor_doorwindows
indoor_marketstore
indoor_room
indoor_venue
dark_light
others_
outdoor_
outdoor_city
outdoor_field
outdoor_grass
outdoor_house
outdoor_mountain
outdoor_oceanbeach
outdoor_playground
outdoor_railway
outdoor_road
outdoor_sportsfield
outdoor_stonerock
outdoor_street
outdoor_water
outdoor_waterside
people_
people_baby
people_crowd
people_group
people_hand
people_many

people_portrait

people_show

people_tattoo

people_young

plant_

plant_branch

plant_flower

plant_leaves

plant_tree

object_screen

object_sculpture

sky_cloud

sky_sun

people_swimming

outdoor_pool

text_

text_mag

text_map

text_menu

text_sign

trans_bicycle

trans_bus

trans_car

trans_trainstation

Compatibilidad con idiomas para Computer Vision

13/01/2020 • 3 minutes to read • [Edit Online](#)

Algunas características de Computer Vision admiten varios idiomas; las características que no se mencionan aquí solo admiten inglés.

Reconocimiento de texto

Computer Vision puede reconocer texto en varios idiomas. Específicamente, la API de [OCR](#) admite varios idiomas, mientras que la API de [Leer](#) y [Reconocer texto](#) solo admiten el inglés. Consulte [Reconocimiento de texto manuscrito e impreso](#) para obtener más información sobre esta funcionalidad y las ventajas de cada API.

OCR detecta automáticamente el idioma del material de entrada, por lo que no es necesario especificar un código de idioma en la llamada a la API. Sin embargo, siempre se devuelven los códigos de idioma como el valor del nodo "language" en la respuesta JSON.

IDIOMA	CÓDIGO DE IDIOMA	API DE OCR
Árabe	ar	✓
Chino (simplificado)	zh-Hans	✓
Chino (tradicional)	zh-Hant	✓
Checo	cs	✓
Danés	da	✓
Neerlandés	nl	✓
English	en	✓
Finés	fi	✓
Francés	fr	✓
Alemán	de	✓
Griego	el	✓
Húngaro	hu	✓
Italiano	it	✓
Japonés	ja	✓
Coreano	ko	✓

IDIOMA	CÓDIGO DE IDIOMA	API DE OCR
Noruego	nb	✓
Polaco	pl	✓
Portugués	pt	✓
Rumano	ro	✓
Ruso	ru	✓
Serbio (cirílico)	sr-Cyr1	✓
Serbio (latino)	sr-Latn	✓
Eslovaco	sk	✓
Español	es	✓
Sueco	sw	✓
Turco	tr	✓

Análisis de imágenes

Algunas acciones de la API [analizar: imagen](#) pueden devolver resultados en otros idiomas, especificados con el parámetro de consulta `language`. Otras acciones devuelven resultados en inglés, independientemente del idioma que se especifique, y otros generan una excepción para los idiomas no admitidos. Las acciones se especifican con los parámetros de consulta `visualFeatures` y `details`; vea la [información general](#) para obtener una lista de todas las acciones que puede hacer con el análisis de imágenes.

IDIO MA	CÓDI GO DE IDIO MA	CATE GORÍ AS	ETIQU ETAS	DESC RIPC IÓN	ADUL TOS	MARC AS	COLO R	CARA S	IMAG ETYPE	OBJET OS	CELEB RIDA DES	PUNT OS DE REFER ENCIA
Chino	zh	✓	✓	✓	-	-	-	-	-	□	✓	✓
English	en	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Japonés	ja	✓	✓	✓	-	-	-	-	-	□	✓	✓
Portugués	pt	✓	✓	✓	-	-	-	-	-	□	✓	✓
Español	es	✓	✓	✓	-	-	-	-	-	□	✓	✓

Pasos siguientes

Conozca el uso de las características de Computer Vision mencionadas en esta guía.

- [Análisis de imágenes locales \(REST\)](#)
- [Extracción de texto impreso \(REST\)](#)