Node.js image and video upload

Cloudinary provides an API for uploading images, videos, and any other kind of file to the cloud. Files uploaded to Cloudinary are stored safely in the cloud with secure backups and revision history. Cloudinary's APIs allow secure uploading from your servers, directly from your visitors' browsers or mobile applications, or fetched via remote public URLs.

Cloudinary's Node.js SDK wraps Cloudinary's upload API and simplifies the integration. Node.js methods are available for easily performing Node.js image and video uploads to the cloud and Node.js helper methods are available for uploading directly from a browser to Cloudinary.

This page covers common usage patterns for Node.js image and video upload with Cloudinary.

For details on all available upload options and parameters, see the Upload images (upload_images) and Upload videos (upload_videos) documentation, and the Upload (image_upload_api_reference#upload_method) method of the Upload API Reference.

On this page:

Upload widget

Cloudinary's Upload widget (upload_widget) is an interactive, feature rich, simple-to-integrate user interface that enables you to add Cloudinary upload support to your website. The widget can be easily embedded in your web application with just a few lines of JavaScript code, eliminating the need to develop in-house interactive upload capabilities. See the Upload widget (upload_widget) documentation for detailed information.

Server-side upload

You can upload images, videos, or any other raw file to Cloudinary from your Node.js code. Uploading is done over HTTPS using a secure protocol based on your account's `api_key` and `api_secret` parameters.

Node.js image upload

The following Node.js method uploads an image to the cloud:

```
function upload(file, options, callback)
```

For example, uploading a local image file named 'my_image.jpg':

```
cloudinary.v2.uploader.upload("/home/my_image.jpg",
    function(error, result) {console.log(result, error)});
```

The file to upload can be specified as a local path, a remote HTTP or HTTPS URL, a whitelisted storage bucket (S3 or Google Storage) URL, a base-64 data URI, or an FTP URL. For details, see Data upload options (upload_images#data_upload_options).

For details on all available upload options and parameters, see the Upload images (upload_images) and Upload videos (upload_videos) documentation, and the Upload (image_upload_api_reference#upload_method) method of the Upload API Reference.

Node.js video upload

You upload videos in exactly the same way as images. However, the `upload` method supports uploading files only up to 100 MB. To upload larger videos, use the upload_large (upload_videos#chunked_video_upload) method, which uploads large files to the cloud in chunks.

The `upload_large` method has the identical signature and options as the `upload` method, with the addition of an optional `chunk_size` parameter (default 20MB).

The following example uploads `dog.mp4` to Cloudinary and stores it in a bi-level folder structure with the public ID `dog_closeup`. It also performs two eager transformations that resize the video to a square and a small rectangle.

```
cloudinary.v2.uploader.upload("dog.mp4",
  { resource_type: "video",
    public_id: "my_folder/my_sub_folder/dog_closeup",
    chunk_size: 6000000,
    eager: [
      { width: 300, height: 300, crop: "pad", audio_codec: "none" },
      { width: 160, height: 100, crop: "crop", gravity: "south", audio_codec: "none" } ],
    eager_async: true,
    eager_notification_url: "https://mysite.example.com/notify_endpoint" },
  function(error, result) {console.log(result, error)});
```

Upload response

By default, uploading is performed synchronously. Once finished, the uploaded image or video is immediately available for manipulation and delivery. An upload call returns a Hash with content similar to the following:

```
{
  public_id: 'cr4mxeqx5zb8rlakpfkg',
  version: 1571218330,
  signature: '63bfbca643baa9c86b7d2921d776628ac83a1b6e',
  width: 864,
  height: 576,
  format: 'jpg',
  resource_type: 'image',
  created_at: '2017-06-26T19:46:03Z',
  bytes: 120253,
  type: 'upload',
  url: 'http://res.cloudinary.com/demo/image/upload/v1571218330/cr4mxeqx5zb8rlakpfkg.jpg',
  secure_url: 'https://res.cloudinary.com/demo/image/upload/v1571218330/cr4mxeqx5zb8rlakpfkg.jpg'
}
```

The response includes HTTP and HTTPS URLs for accessing the uploaded media asset as well as additional information regarding the uploaded asset: The Public ID, resource type, width and height, file format, file size in bytes, a signature for verifying the response and more.

Related topics

→ For more information on uploading media assets, see the Upload Images (upload_images) and Upload Videos (upload_videos) documentation.

→ For details on all available upload parameters, see the Upload (image_upload_api_reference#upload_method) method of the Upload API Reference.

Direct uploading from the browser

The upload sample mentioned above allows your server-side Node.js code to upload media assets to Cloudinary. In this flow, if you have a web form that allows your users to upload images or videos, the media file's data is first sent to your server and only then uploaded to Cloudinary. A more efficient and powerful option is to allow your users to upload images and videos directly from the browser to Cloudinary instead of going through your servers. This method allows for faster uploading and a better user experience. It also reduces load from your servers and reduces the complexity of your Node.js applications.

Uploading directly from the browser is done using Cloudinary's **jQuery plugin**. To ensure that all uploads were authorized by your application, a secure signature must first be generated in your server-side Node.js code.

Direct uploading environment setup

Start by including the required Javascript files: Cloudinary's jQuery plugin as well as the jQuery-File-Upload plugin it depends on. These are available in the **js** folder of Cloudinary's Javascript library (https://github.com/cloudinary/cloudinary_js/tree/master/js).

You can directly include the Javascript files:

```
<script type="text/javascript" src="jquery.min.js"></script>
<script type="text/javascript" src="jquery.ui.widget.js"></script>
<script type="text/javascript" src="jquery.iframe-transport.js"></script>
<script type="text/javascript" src="jquery.fileupload.js"></script>
<script type="text/javascript" src="jquery.cloudinary.js"></script>
```

Cloudinary's jQuery plugin requires your `cloud_name` and additional configuration parameters.

Important: Never expose your `api_secret` in public client-side code.

To automatically set-up Cloudinary's configuration, include the following line in your view or layout:

```
cloudinary.cloudinary_js_config()
```

The Cloudinary jQuery library utilizes the **Blueimp File Upload** library to support uploading media directly from the browser. You must explicitly initialize this library:

```
$(function() {
  if($.fn.cloudinary_fileupload !== undefined) {
    $("input.cloudinary-fileupload[type=file]").cloudinary_fileupload();
  }
});
```

Direct uploading from the browser is performed using XHR (Ajax XMLHttpRequest) CORS (Cross Origin Resource Sharing) requests. To support older browsers that do not support CORS, the jQuery plugin gracefully degrades to an iframe based solution. This solution requires placing `cloudinary_cors.html` in the static folder of your Node.js application. This file is available in the **html** folder of Cloudinary's Javascript library (https://github.com/cloudinary/cloudinary_js/tree/master/html). The following code builds a URL of the local `cloudinary_cors.html` file:

```
var cloudinary_cors = "https://" + request.headers.host + "/cloudinary_cors.html";
```

Direct upload file tag

Embed a file input tag in your HTML pages using the `image_upload_tag` helper method. The following example adds a file input field to your form. Selecting or dragging a file to this input field automatically initiates uploading from the browser to Cloudinary.

```
cloudinary.v2.uploader.image_upload_tag('image_id', {callback: cloudinary_cors});
```

When uploading is completed, the identifier of the uploaded image is set as the value of a hidden input field of your selected name (e.g., 'image_id' in the example above). You can then process the identifier received by your Node.js code and store it in your model for future use, exactly as if you're using standard server side uploading.

The following Javascript code processes the received identifier, verifies the signature (concatenated to the identifier) and updates a model entity with the identifiers of the uploaded image (i.e., the Public ID and version of the image).

```
var url = require('url');
var url_parts = url.parse(request.url, true);
var query = url_parts.query;

var preloaded_file = new cloudinary.PreloadedFile(query.image_id);
if (preloaded_file.is_valid()) {
  photo.image_id = preloaded_file.identifier();
} else {
  throw("Invalid upload signature");
}
```

You can now display a directly uploaded image in the same way you would display any other Cloudinary hosted image:

```
cloudinary.image(photo.public_id, { format: "jpg", crop: "fill", width: 120, height: 80 });
```

Directly uploading a video

The following example renders a direct file upload input field using the `image_upload_tag` helper method. Although the default `resource_type` for this method is `auto`, the `video` type is explicitly defined, and asynchronous eager transformations are used to generate adaptive bitrate streaming (video_manipulation_and_delivery#adaptive_bitrate_streaming_hls_and_mpeg_dash) content. The `html` parameter is used to include standard HTML parameters (in this case, an `id` attribute) in the generated tag.

```
cloudinary.v2.uploader.image_upload_tag('image_id',
  { resource_type: "video",
    eager: [{streaming_profile:"full_hd", format:"m3u8"}],
    eager_async: true,
    eager_notification_url: "https://mysite.example.com/notify_endpoint",
    html: {id: "my_upload_tag" } },
  function(error, result) {console.log(result, error)}
);
```

Additional jQuery library features

Cloudinary's jQuery library also enables an enhanced uploading experience with options like showing a progress bar, displaying a thumbnail of the uploaded image, drag & drop support, uploading multiple files and more.

For example, bind to Cloudinary's `cloudinarydone` event if you want to be notified when an upload to Cloudinary has completed. You will have access to the full details of the uploaded image and you can display a cloud-generated thumbnail of the uploaded images using Cloudinary's jQuery plugin. The following code creates a 150x100 thumbnail of an uploaded image and updates an input field with the public ID of this image.

```
$('.cloudinary-fileupload').bind('cloudinarydone', function(e, data) {
  $('.preview').html(
    $.cloudinary.image(data.result.public_id,
      { format: data.result.format, version: data.result.version,
        crop: 'fill', width: 150, height: 100 })
  );
  $('.image_public_id').val(data.result.public_id);
  return true;
});
```

You can find more details and options in the jQuery (jquery_image_and_video_upload) documentation.

Related topics

→ For more information on uploading media assets, see the Upload Images (upload_images) and Upload Videos (upload_videos) documentation.

→ For details on all available upload parameters, see the Upload (image_upload_api_reference#upload_method) method of the Upload API Reference.

← Node.js SDK(/documentation/node_integration)

Node.js image manipulation(/documentation/node_image_manipulation) →

Service status

(https://status.cloudinary.com)

Connect

(https://www.facebook.com/Cloudinary)

(https://twitter.com/cloudinary)

(https://www.linkedin.com/company/cloudinary/)

Terms of Use (https://cloudinary.com/tos) Privacy Policy (https://cloudinary.com/privacy) DMCA Notice (https://cloudinary.com/dmca)