Active Storage

Active Storage makes it simple to upload and reference files in cloud services like Amazon S3, Google Cloud Storage, or Microsoft Azure Storage, and attach those files to Active Records. Supports having one main service and mirrors in other services for redundancy. It also provides a disk service for testing or local deployments, but the focus is on cloud storage.

Files can be uploaded from the server to the cloud or directly from the client to the cloud.

Image files can furthermore be transformed using on-demand variants for quality, aspect ratio, size, or any other MiniMagick or Vips supported transformation.

You can read more about Active Storage in the Active Storage Overview guide.

Compared to other storage solutions

A key difference to how Active Storage works compared to other attachment solutions in Rails is through the use of built-in Blob and Attachment models (backed by Active Record). This means existing application models do not need to be modified with additional columns to associate with files. Active Storage uses polymorphic associations via the `Attachment` join model, which then connects to the actual `Blob`.

`Blob` models store attachment metadata (filename, content-type, etc.), and their identifier key in the storage service. Blob models do not store the actual binary data. They are intended to be immutable in spirit. One file, one blob. You can associate the same blob with multiple application models as well. And if you want to do transformations of a given `Blob`, the idea is that you'll simply create a new one, rather than attempt to mutate the existing one (though of course you can delete the previous version later if you don't need it).

Installation

Run `bin/rails active_storage:install` to copy over active_storage migrations.

NOTE: If the task cannot be found, verify that `require "active_storage/engine"` is present in `config/application.rb`.

Examples

One attachment:

```ruby
class User < ApplicationRecord
  # Associates an attachment and a blob. When the user is destroyed they are
  # purged by default (models destroyed, and resource files deleted).
  has_one_attached :avatar
```

```ruby
end

# Attach an avatar to the user.
user.avatar.attach(io: File.open("/path/to/face.jpg"), filename: "face.jpg", content_type:

# Does the user have an avatar?
user.avatar.attached? # => true

# Synchronously destroy the avatar and actual resource files.
user.avatar.purge

# Destroy the associated models and actual resource files async, via Active Job.
user.avatar.purge_later

# Does the user have an avatar?
user.avatar.attached? # => false

# Generate a permanent URL for the blob that points to the application.
# Upon access, a redirect to the actual service endpoint is returned.
# This indirection decouples the public URL from the actual one, and
# allows for example mirroring attachments in different services for
# high-availability. The redirection has an HTTP expiration of 5 min.
url_for(user.avatar)

class AvatarsController < ApplicationController
  def update
    # params[:avatar] contains an ActionDispatch::Http::UploadedFile object
    Current.user.avatar.attach(params.require(:avatar))
    redirect_to Current.user
  end
end
```

Many attachments:

```ruby
class Message < ApplicationRecord
  has_many_attached :images
end
```

```erb
<%= form_with model: @message, local: true do |form| %>
  <%= form.text_field :title, placeholder: "Title" %><br>
  <%= form.text_area :content %><br><br>

  <%= form.file_field :images, multiple: true %><br>
  <%= form.submit %>
<% end %>
```

```ruby
class MessagesController < ApplicationController
  def index
```

```
    # Use the built-in with_attached_images scope to avoid N+1
    @messages = Message.all.with_attached_images
  end

  def create
    message = Message.create! params.require(:message).permit(:title, :content, images: [])
    redirect_to message
  end

  def show
    @message = Message.find(params[:id])
  end
end
```

Variation of image attachment:

```
<%# Hitting the variant URL will lazy transform the original blob and then redirect to its r
<%= image_tag user.avatar.variant(resize_to_limit: [100, 100]) %>
```

File serving strategies

Active Storage supports two ways to serve files: redirecting and proxying.

Redirecting

Active Storage generates stable application URLs for files which, when accessed, redirect to signed, short-lived service URLs. This relieves application servers of the burden of serving file data. It is the default file serving strategy.

When the application is configured to proxy files by default, use the `rails_storage_redirect_path` and `_url` route helpers to redirect instead:

```
<%= image_tag rails_storage_redirect_path(@user.avatar) %>
```

Proxying

Optionally, files can be proxied instead. This means that your application servers will download file data from the storage service in response to requests. This can be useful for serving files from a CDN.

You can configure Active Storage to use proxying by default:

```
# config/initializers/active_storage.rb
Rails.application.config.active_storage.resolve_model_to_route = :rails_storage_proxy
```

Or if you want to explicitly proxy specific attachments there are URL helpers you can use in the form of `rails_storage_proxy_path` and `rails_storage_proxy_url`.

```
<%= image_tag rails_storage_proxy_path(@user.avatar) %>
```

Direct uploads

Active Storage, with its included JavaScript library, supports uploading directly from the client to the cloud.

Direct upload installation

1. Include the Active Storage JavaScript in your application's JavaScript bundle or reference it directly.

   Requiring directly without bundling through the asset pipeline in the application html with autostart: `html <%= javascript_include_tag "activestorage" %>` Requiring via importmap-rails without bundling through the asset pipeline in the application html without autostart as ESM: `ruby # config/importmap.rb pin "@rails/activestorage", to: "activestorage.esm.js"` `html <script type="module-shim"> import * as ActiveStorage from "@rails/activestorage" ActiveStorage.start() </script>` Using the asset pipeline: `js //= require activestorage` Using the npm package: `js import * as ActiveStorage from "@rails/activestorage" ActiveStorage.start()`

2. Annotate file inputs with the direct upload URL.

   ```
   <%= form.file_field :attachments, multiple: true, direct_upload: true %>
   ```

3. That's it! Uploads begin upon form submission.

Direct upload JavaScript events

Event name	Event target	Event data (`event.detail`)	Description
`direct-uploads:start`	`<form>`	None	A form containing files for direct upload fields was submitted.
`direct-upload:initialize`	`<input>`	`{id, file}`	Dispatched for every file after form submission.
`direct-upload:start`	`<input>`	`{id, file}`	A direct upload is starting.
`direct-upload:before-blob-request`	`<input>`	`{id, file, xhr}`	Before making a request to your application for direct upload metadata.

Event name	Event target	Event data (`event.detail`)	Description
`direct-upload:before-storage-request`	`<input>`	`{id, file, xhr}`	Before making a request to store a file.
`direct-upload:progress`	`<input>`	`{id, file, progress}`	As requests to store files progress.
`direct-upload:error`	`<input>`	`{id, file, error}`	An error occurred. An `alert` will display unless this event is canceled.
`direct-upload:end`	`<input>`	`{id, file}`	A direct upload has ended.
`direct-uploads:end`	`<form>`	None	All direct uploads have ended.

License

Active Storage is released under the MIT License.

Support

API documentation is at:

- https://api.rubyonrails.org

Bug reports for the Ruby on Rails project can be filed here:

- https://github.com/rails/rails/issues

Feature requests should be discussed on the rails-core mailing list here:

- https://discuss.rubyonrails.org/c/rubyonrails-core