DO NOT READ THIS FILE ON GITHUB, GUIDES ARE PUBLISHED ON https://guides.rubyonrails.org.

Upgrading Ruby on Rails

This guide provides steps to be followed when you upgrade your applications to a newer version of Ruby on Rails. These steps are also available in individual release guides.

General Advice

Before attempting to upgrade an existing application, you should be sure you have a good reason to upgrade. You need to balance several factors: the need for new features, the increasing difficulty of finding support for old code, and your available time and skills, to name a few.

Test Coverage

The best way to be sure that your application still works after upgrading is to have good test coverage before you start the process. If you don't have automated tests that exercise the bulk of your application, you'll need to spend time manually exercising all the parts that have changed. In the case of a Rails upgrade, that will mean every single piece of functionality in the application. Do yourself a favor and make sure your test coverage is good *before* you start an upgrade.

Ruby Versions

Rails generally stays close to the latest released Ruby version when it's released:

- Rails 7 requires Ruby 2.7.0 or newer.
- Rails 6 requires Ruby 2.5.0 or newer.
- Rails 5 requires Ruby 2.2.2 or newer.

It's a good idea to upgrade Ruby and Rails separately. Upgrade to the latest Ruby you can first, and then upgrade Rails.

The Upgrade Process

When changing Rails versions, it's best to move slowly, one minor version at a time, in order to make good use of the deprecation warnings. Rails version numbers are in the form Major.Minor.Patch. Major and Minor versions are allowed to make changes to the public API, so this may cause errors in your application. Patch versions only include bug fixes, and don't change any public API.

The process should go as follows:

1. Write tests and make sure they pass.
2. Move to the latest patch version after your current version.
3. Fix tests and deprecated features.
4. Move to the latest patch version of the next minor version.

Repeat this process until you reach your target Rails version.

Moving between versions

To move between versions:

1. Change the Rails version number in the `Gemfile` and run `bundle update`.

2. Change the versions for Rails JavaScript packages in `package.json` and run `yarn install`, if running on Webpacker.
3. Run the Update task.
4. Run your tests.

You can find a list of all released Rails gems here.

The Update Task

Rails provides the `rails app:update` command. After updating the Rails version in the `Gemfile`, run this command. This will help you with the creation of new files and changes of old files in an interactive session.

```
$ bin/rails app:update
       exist  config
    conflict  config/application.rb
Overwrite /myapp/config/application.rb? (enter "h" for help) [Ynaqdh]
       force  config/application.rb
      create  config/initializers/new_framework_defaults_7_0.rb
...
```

Don't forget to review the difference, to see if there were any unexpected changes.

Configure Framework Defaults

The new Rails version might have different configuration defaults than the previous version. However, after following the steps described above, your application would still run with configuration defaults from the *previous* Rails version. That's because the value for `config.load_defaults` in `config/application.rb` has not been changed yet.

To allow you to upgrade to new defaults one by one, the update task has created a file `config/initializers/new_framework_defaults_X.Y.rb` (with the desired Rails version in the filename). You should enable the new configuration defaults by uncommenting them in the file; this can be done gradually over several deployments. Once your application is ready to run with new defaults, you can remove this file and flip the `config.load_defaults` value.

Upgrading from Rails 7.0 to Rails 7.1

Autoloaded paths are no longer in load path

Starting from Rails 7.1, all paths managed by the autoloader will no longer be added to `$LOAD_PATH`. This means it won't be possible to load them with a manual `require` call, the class or module can be referenced instead.

Reducing the size of `$LOAD_PATH` speed-up `require` calls for apps not using `bootsnap`, and reduce the size of the `bootsnap` cache for the others.

`ActiveStorage::BaseController` no longer includes the streaming concern

Application controllers that inherit from `ActiveStorage::BaseController` and use streaming to implement custom file serving logic must now explicitly include the `ActiveStorage::Streaming` module.

New `ActiveSupport::MessageEncryptor` default serializer

As of Rails 7.1, the default serializer in use by the `MessageEncryptor` is `JSON`. This offers a more secure alternative to the current default serializer.

The `MessageEncryptor` offers the ability to migrate the default serializer from `Marshal` to `JSON`.

If you would like to ignore this change in existing applications, set the following: `config.active_support.default_message_encryptor_serializer = :marshal`.

In order to roll out the new default when upgrading from `7.0` to `7.1`, there are three configuration variables to keep in mind.

```
config.active_support.default_message_encryptor_serializer
config.active_support.fallback_to_marshal_deserialization
config.active_support.use_marshal_serialization
```

`default_message_encryptor_serializer` defaults to `:json` as of `7.1` but it offers both a `:hybrid` and `:marshal` option.

In order to migrate an older deployment to `:json`, first ensure that the `default_message_encryptor_serializer` is set to `:marshal`.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_encryptor_serializer = :marshal
```

Once this is deployed on all Rails processes, set `default_message_encryptor_serializer` to `:hybrid` to begin using the `ActiveSupport::JsonWithMarshalFallback` class as the serializer. The defaults for this class are to use `Marshal` as the serializer and to allow the deserialisation of both `Marshal` and `JSON` serialized payloads.

```
config.load_defaults 7.0
config.active_support.default_message_encryptor_serializer = :hybrid
```

Once this is deployed on all Rails processes, set the following configuration options in order to stop the `ActiveSupport::JsonWithMarshalFallback` class from using `Marshal` to serialize new payloads.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_encryptor_serializer = :hybrid
config.active_support.use_marshal_serialization = false
```

Allow this configuration to run on all processes for a considerable amount of time. `ActiveSupport::JsonWithMarshalFallback` logs the following each time the `Marshal` fallback is used:

```
JsonWithMarshalFallback: Marshal load fallback occurred.
```

Once those message stop appearing in your logs and you're confident that all `MessageEncryptor` payloads in transit are `JSON` serialized, the following configuration options will disable the Marshal fallback in `ActiveSupport::JsonWithMarshalFallback`.

```
# config/application.rb
config.load_defaults 7.0
```

```
config.active_support.default_message_encryptor_serializer = :hybrid
config.active_support.use_marshal_serialization = false
config.active_support.fallback_to_marshal_deserialization = false
```

If all goes well, you should now be safe to migrate the Message Encryptor from `ActiveSupport::JsonWithMarshalFallback` to `ActiveSupport::JSON`. To do so, simply swap the `:hybrid` serializer for the `:json` serializer.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_encryptor_serializer = :json
```

Alternatively, you could load defaults for 7.1

```
# config/application.rb
config.load_defaults 7.1
```

New `ActiveSupport::MessageVerifier` default serializer

As of Rails 7.1, the default serializer in use by the `MessageVerifier` is `JSON`. This offers a more secure alternative to the current default serializer.

The `MessageVerifier` offers the ability to migrate the default serializer from `Marshal` to `JSON`.

If you would like to ignore this change in existing applications, set the following: `config.active_support.default_message_verifier_serializer = :marshal`.

In order to roll out the new default when upgrading from `7.0` to `7.1`, there are three configuration variables to keep in mind.

```
config.active_support.default_verifier_serializer
config.active_support.fallback_to_marshal_deserialization
config.active_support.use_marshal_serialization
```

`default_message_verifier_serializer` defaults to `:json` as of `7.1` but it offers both a `:hybrid` and `:marshal` option.

In order to migrate an older deployment to `:json`, first ensure that the `default_message_verifier_serializer` is set to `:marshal`.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_verifier_serializer = :marshal
```

Once this is deployed on all Rails processes, set `default_message_verifier_serializer` to `:hybrid` to begin using the `ActiveSupport::JsonWithMarshalFallback` class as the serializer. The defaults for this class are to use `Marshal` as the serializer and to allow the deserialisation of both `Marshal` and `JSON` serialized payloads.

```
config.load_defaults 7.0
config.active_support.default_message_verifier_serializer = :hybrid
```

Once this is deployed on all Rails processes, set the following configuration options in order to stop the `ActiveSupport::JsonWithMarshalFallback` class from using `Marshal` to serialize new payloads.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_verifier_serializer = :hybrid
config.active_support.use_marshal_serialization = false
```

Allow this configuration to run on all processes for a considerable amount of time.
`ActiveSupport::JsonWithMarshalFallback` logs the following each time the `Marshal` fallback is used:

```
JsonWithMarshalFallback: Marshal load fallback occurred.
```

Once those message stop appearing in your logs and you're confident that all `MessageVerifier` payloads in transit are `JSON` serialized, the following configuration options will disable the Marshal fallback in `ActiveSupport::JsonWithMarshalFallback`.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_verifier_serializer = :hybrid
config.active_support.use_marshal_serialization = false
config.active_support.fallback_to_marshal_deserialization = false
```

If all goes well, you should now be safe to migrate the Message Verifier from `ActiveSupport::JsonWithMarshalFallback` to `ActiveSupport::JSON`. To do so, simply swap the `:hybrid` serializer for the `:json` serializer.

```
# config/application.rb
config.load_defaults 7.0
config.active_support.default_message_verifier_serializer = :json
```

Alternatively, you could load defaults for 7.1

```
# config/application.rb
config.load_defaults 7.1
```

Upgrading from Rails 6.1 to Rails 7.0

`ActionView::Helpers::UrlHelper#button_to` changed behavior

Starting from Rails 7.0 `button_to` renders a `form` tag with `patch` HTTP verb if a persisted Active Record object is used to build button URL. To keep current behavior consider explicitly passing `method:` option:

```
-button_to("Do a POST", [:my_custom_post_action_on_workshop, Workshop.find(1)])
+button_to("Do a POST", [:my_custom_post_action_on_workshop, Workshop.find(1)],
method: :post)
```

or using helper to build the URL:

```
-button_to("Do a POST", [:my_custom_post_action_on_workshop, Workshop.find(1)])
+button_to("Do a POST",
my_custom_post_action_on_workshop_workshop_path(Workshop.find(1)))
```

Spring

If your application uses Spring, it needs to be upgraded to at least version 3.0.0. Otherwise you'll get

```
undefined method `mechanism=' for ActiveSupport::Dependencies:Module
```

Also, make sure `config.cache_classes` is set to `false` in `config/environments/test.rb`.

Sprockets is now an optional dependency

The gem `rails` doesn't depend on `sprockets-rails` anymore. If your application still needs to use Sprockets, make sure to add `sprockets-rails` to your Gemfile.

```
gem "sprockets-rails"
```

Applications need to run in `zeitwerk` mode

Applications still running in `classic` mode have to switch to `zeitwerk` mode. Please check the Classic to Zeitwerk HOWTO guide for details.

The setter `config.autoloader=` has been deleted

In Rails 7 there is no configuration point to set the autoloading mode, `config.autoloader=` has been deleted. If you had it set to `:zeitwerk` for whatever reason, just remove it.

`ActiveSupport::Dependencies` private API has been deleted

The private API of `ActiveSupport::Dependencies` has been deleted. That includes methods like `hook!`, `unhook!`, `depend_on`, `require_or_load`, `mechanism`, and many others.

A few of highlights:

- If you used `ActiveSupport::Dependencies.constantize` or `ActiveSupport::Dependencies.safe_constantize`, just change them to `String#constantize` or `String#safe_constantize`.

  ```
  ActiveSupport::Dependencies.constantize("User") # NO LONGER POSSIBLE
  "User".constantize # 👍
  ```

- Any usage of `ActiveSupport::Dependencies.mechanism`, reader or writer, has to be replaced by accessing `config.cache_classes` accordingly.
- If you want to trace the activity of the autoloader, `ActiveSupport::Dependencies.verbose=` is no longer available, just throw `Rails.autoloaders.log!` in `config/application.rb`.

Auxiliary internal classes or modules are also gone, like `ActiveSupport::Dependencies::Reference`, `ActiveSupport::Dependencies::Blamable`, and others.

Autoloading during initialization

Applications that autoloaded reloadable constants during initialization outside of `to_prepare` blocks got those constants unloaded and had this warning issued since Rails 6.0:

```
DEPRECATION WARNING: Initialization autoloaded the constant ....

Being able to do this is deprecated. Autoloading during initialization is going
to be an error condition in future versions of Rails.

...
```

If you still get this warning in the logs, please check the section about autoloading when the application boots in the autoloading guide. You'd get a `NameError` in Rails 7 otherwise.

Ability to configure `config.autoload_once_paths`

`config.autoload_once_paths` can be set in the body of the application class defined in `config/application.rb` or in the configuration for environments in `config/environments/*`.

Similarly, engines can configure that collection in the class body of the engine class or in the configuration for environments.

After that, the collection is frozen, and you can autoload from those paths. In particular, you can autoload from there during initialization. They are managed by the `Rails.autoloaders.once` autoloader, which does not reload, only autoloads/eager loads.

If you configured this setting after the environments configuration has been processed and are getting `FrozenError`, please just move the code.

`ActionDispatch::Request#content_type` now returned Content-Type header as it is.

Previously, `ActionDispatch::Request#content_type` returned value does NOT contain charset part. This behavior changed to returned Content-Type header containing charset part as it is.

If you want just MIME type, please use `ActionDispatch::Request#media_type` instead.

Before:

```ruby
request = ActionDispatch::Request.new("CONTENT_TYPE" => "text/csv; header=present;
charset=utf-16", "REQUEST_METHOD" => "GET")
request.content_type #=> "text/csv"
```

After:

```
request = ActionDispatch::Request.new("Content-Type" => "text/csv; header=present;
charset=utf-16", "REQUEST_METHOD" => "GET")
request.content_type #=> "text/csv; header=present; charset=utf-16"
request.media_type   #=> "text/csv"
```

Key generator digest class changing to use SHA256

The default digest class for the key generator is changing from SHA1 to SHA256. This has consequences in any encrypted message generated by Rails, including encrypted cookies.

In order to be able to read messages using the old digest class it is necessary to register a rotator.

The following is an example for rotator for the encrypted cookies.

```
# config/initializers/cookie_rotator.rb
Rails.application.config.after_initialize do
  Rails.application.config.action_dispatch.cookies_rotations.tap do |cookies|
    salt =
Rails.application.config.action_dispatch.authenticated_encrypted_cookie_salt
    secret_key_base = Rails.application.secrets.secret_key_base

    key_generator = ActiveSupport::KeyGenerator.new(
      secret_key_base, iterations: 1000, hash_digest_class: OpenSSL::Digest::SHA1
    )
    key_len = ActiveSupport::MessageEncryptor.key_len
    secret = key_generator.generate_key(salt, key_len)

    cookies.rotate :encrypted, secret
  end
end
```

Digest class for ActiveSupport::Digest changing to SHA256

The default digest class for ActiveSupport::Digest is changing from SHA1 to SHA256. This has consequences for things like Etags that will change and cache keys as well. Changing these keys can have impact on cache hit rates, so be careful and watch out for this when upgrading to the new hash.

New ActiveSupport::Cache serialization format

A faster and more compact serialization format was introduced.

To enable it you must set `config.active_support.cache_format_version = 7.0`:

```
# config/application.rb

config.load_defaults 6.1
config.active_support.cache_format_version = 7.0
```

Or simply:

```
# config/application.rb

config.load_defaults 7.0
```

However Rails 6.1 applications are not able to read this new serialization format, so to ensure a seamless upgrade you must first deploy your Rails 7.0 upgrade with `config.active_support.cache_format_version = 6.1`, and then only once all Rails processes have been updated you can set `config.active_support.cache_format_version = 7.0`.

Rails 7.0 is able to read both formats so the cache won't be invalidated during the upgrade.

Active Storage video preview image generation

Video preview image generation now uses FFmpeg's scene change detection to generate more meaningful preview images. Previously the first frame of the video would be used and that caused problems if the video faded in from black. This change requires FFmpeg v3.4+.

Active Storage default variant processor changed to `:vips`

For new apps, image transformation will use libvips instead of ImageMagick. This will reduce the time taken to generate variants as well as CPU and memory usage, improving response times in apps that rely on Active Storage to serve their images.

The `:mini_magick` option is not being deprecated, so it is fine to keep using it.

To migrate an existing app to libvips, set:

```
Rails.application.config.active_storage.variant_processor = :vips
```

You will then need to change existing image transformation code to the `image_processing` macros, and replace ImageMagick's options with libvips' options.

Replace resize with resize_to_limit

```
- variant(resize: "100x")
+ variant(resize_to_limit: [100, nil])
```

If you don't do this, when you switch to vips you will see this error: `no implicit conversion to float from string`.

Use an array when cropping

```
- variant(crop: "1920x1080+0+0")
+ variant(crop: [0, 0, 1920, 1080])
```

If you don't do this when migrating to vips, you will see the following error: `unable to call crop: you supplied 2 arguments, but operation needs 5`.

Clamp your crop values:

Vips is more strict than ImageMagick when it comes to cropping:

1. It will not crop if `x` and/or `y` are negative values. e.g.: `[-10, -10, 100, 100]`
2. It will not crop if position (`x` or `y`) plus crop dimension (`width` , `height`) is larger than the image.
   e.g.: a 125x125 image and a crop of `[50, 50, 100, 100]`

If you don't do this when migrating to vips, you will see the following error: `extract_area: bad extract area`

Adjust the background color used for `resize_and_pad`

Vips uses black as the default background color `resize_and_pad`, instead of white like ImageMagick. Fix that by using the `background` option:

```
- variant(resize_and_pad: [300, 300])
+ variant(resize_and_pad: [300, 300, background: [255]])
```

Remove any EXIF based rotation

Vips will auto rotate images using the EXIF value when processing variants. If you were storing rotation values from user uploaded photos to apply rotation with ImageMagick, you must stop doing that:

```
- variant(format: :jpg, rotate: rotation_value)
+ variant(format: :jpg)
```

Replace monochrome with colourspace

Vips uses a different option to make monochrome images:

```
- variant(monochrome: true)
+ variant(colourspace: "b-w")
```

Switch to libvips options for compressing images

JPEG

```
- variant(strip: true, quality: 80, interlace: "JPEG", sampling_factor: "4:2:0",
colorspace: "sRGB")
+ variant(saver: { strip: true, quality: 80, interlace: true })
```

PNG

```
- variant(strip: true, quality: 75)
+ variant(saver: { strip: true, compression: 9 })
```

WEBP

```
- variant(strip: true, quality: 75, define: { webp: { lossless: false,
alpha_quality: 85, thread_level: 1 } })
+ variant(saver: { strip: true, quality: 75, lossless: false, alpha_q: 85,
reduction_effort: 6, smart_subsample: true })
```

GIF

```
- variant(layers: "Optimize")
+ variant(saver: { optimize_gif_frames: true, optimize_gif_transparency: true })
```

Deploy to production

Active Storage encodes into the url for the image the list of transformations that must be performed. If your app is caching these urls, your images will break after you deploy the new code to production. Because of this you must manually invalidate your affected cache keys.

For example, if you have something like this in a view:

```
<% @products.each do |product| %>
  <% cache product do %>
    <%= image_tag product.cover_photo.variant(resize: "200x") %>
  <% end %>
<% end %>
```

You can invalidate the cache either by touching the product, or changing the cache key:

```
<% @products.each do |product| %>
  <% cache ["v2", product] do %>
    <%= image_tag product.cover_photo.variant(resize_to_limit: [200, nil]) %>
  <% end %>
<% end %>
```

Rails version is now included in the Active Record schema dump

Rails 7.0 changed some default values for some column types. To avoid that application upgrading from 6.1 to 7.0 load the current schema using the new 7.0 defaults, Rails now includes the version of the framework in the schema dump.

Before loading the schema for the first time in Rails 7.0, make sure to run `rails app:update` to ensure that the version of the schema is included in the schema dump.

The schema file will look like this:

```
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
#
# This file is the source Rails uses to define your schema when running `bin/rails
# db:schema:load`. When creating a new database, `bin/rails db:schema:load` tends to
# be faster and is potentially less error prone than running all of your
# migrations from scratch. Old migrations may fail to apply correctly if those
# migrations use external dependencies or application code.
#
# It's strongly recommended that you check this file into your version control
system.

ActiveRecord::Schema[6.1].define(version: 2022_01_28_123512) do
```

NOTE: The first time you dump the schema with Rails 7.0, you will see many changes to that file, including some column information. Make sure to review the new schema file content and commit it to your repository.

Upgrading from Rails 6.0 to Rails 6.1

For more information on changes made to Rails 6.1 please see the release notes.

`Rails.application.config_for` return value no longer supports access with String keys.

Given a configuration file like this:

```
# config/example.yml
development:
  options:
    key: value
```

```
Rails.application.config_for(:example).options
```

This used to return a hash on which you could access values with String keys. That was deprecated in 6.0, and now doesn't work anymore.

You can call `with_indifferent_access` on the return value of `config_for` if you still want to access values with String keys, e.g.:

```
Rails.application.config_for(:example).with_indifferent_access.dig('options', 'key')
```

Response's Content-Type when using `respond_to#any`

The Content-Type header returned in the response can differ from what Rails 6.0 returned, more specifically if your application uses `respond_to { |format| format.any }`. The Content-Type will now be based on the given block rather than the request's format.

Example:

```
def my_action
  respond_to do |format|
    format.any { render(json: { foo: 'bar' }) }
  end
end
```

```
get('my_action.csv')
```

Previous behaviour was returning a `text/csv` response's Content-Type which is inaccurate since a JSON response is being rendered. Current behaviour correctly returns a `application/json` response's Content-Type.

If your application relies on the previous incorrect behaviour, you are encouraged to specify which formats your action accepts, i.e.

```
format.any(:xml, :json) { render request.format.to_sym => @people }
```

`ActiveSupport::Callbacks#halted_callback_hook` now receive a second argument

Active Support allows you to override the `halted_callback_hook` whenever a callback halts the chain. This method now receives a second argument which is the name of the callback being halted. If you have classes that override this method, make sure it accepts two arguments. Note that this is a breaking change without a prior deprecation cycle (for performance reasons).

Example:

```
class Book < ApplicationRecord
  before_save { throw(:abort) }
  before_create { throw(:abort) }

  def halted_callback_hook(filter, callback_name) # => This method now accepts 2
arguments instead of 1
    Rails.logger.info("Book couldn't be #{callback_name}d")
  end
end
```

The `helper` class method in controllers uses `String#constantize`

Conceptually, before Rails 6.1

```
helper "foo/bar"
```

resulted in

```
require_dependency "foo/bar_helper"
module_name = "foo/bar_helper".camelize
module_name.constantize
```

Now it does this instead:

```
prefix = "foo/bar".camelize
"#{prefix}Helper".constantize
```

This change is backwards compatible for the majority of applications, in which case you do not need to do anything.

Technically, however, controllers could configure `helpers_path` to point to a directory in `$LOAD_PATH` that was not in the autoload paths. That use case is no longer supported out of the box. If the helper module is not autoloadable, the application is responsible for loading it before calling `helper`.

Redirection to HTTPS from HTTP will now use the 308 HTTP status code

The default HTTP status code used in `ActionDispatch::SSL` when redirecting non-GET/HEAD requests from HTTP to HTTPS has been changed to `308` as defined in https://tools.ietf.org/html/rfc7538.

Active Storage now requires Image Processing

When processing variants in Active Storage, it's now required to have the image_processing gem bundled instead of directly using `mini_magick`. Image Processing is configured by default to use `mini_magick` behind the scenes, so the easiest way to upgrade is by replacing the `mini_magick` gem for the `image_processing` gem and making sure to remove the explicit usage of `combine_options` since it's no longer needed.

For readability, you may wish to change raw `resize` calls to `image_processing` macros. For example, instead of:

```
video.preview(resize: "100x100")
video.preview(resize: "100x100>")
video.preview(resize: "100x100^")
```

you can respectively do:

```
video.preview(resize_to_fit: [100, 100])
video.preview(resize_to_limit: [100, 100])
video.preview(resize_to_fill: [100, 100])
```

Upgrading from Rails 5.2 to Rails 6.0

For more information on changes made to Rails 6.0 please see the release notes.

Using Webpacker

Webpacker is the default JavaScript compiler for Rails 6. But if you are upgrading the app, it is not activated by default. If you want to use Webpacker, then include it in your Gemfile and install it:

```
gem "webpacker"
```

```
$ bin/rails webpacker:install
```

Force SSL

The `force_ssl` method on controllers has been deprecated and will be removed in Rails 6.1. You are encouraged to enable `config.force_ssl` to enforce HTTPS connections throughout your application. If you need to exempt certain endpoints from redirection, you can use `config.ssl_options` to configure that behavior.

Purpose and expiry metadata is now embedded inside signed and encrypted cookies for increased security

To improve security, Rails embeds the purpose and expiry metadata inside encrypted or signed cookies value.

Rails can then thwart attacks that attempt to copy the signed/encrypted value of a cookie and use it as the value of another cookie.

This new embed metadata make those cookies incompatible with versions of Rails older than 6.0.

If you require your cookies to be read by Rails 5.2 and older, or you are still validating your 6.0 deploy and want to be able to rollback set `Rails.application.config.action_dispatch.use_cookies_with_metadata` to `false`.

All npm packages have been moved to the `@rails` scope

If you were previously loading any of the `actioncable`, `activestorage`, or `rails-ujs` packages through npm/yarn, you must update the names of these dependencies before you can upgrade them to `6.0.0`:

```
actioncable   → @rails/actioncable
activestorage → @rails/activestorage
rails-ujs     → @rails/ujs
```

Action Cable JavaScript API Changes

The Action Cable JavaScript package has been converted from CoffeeScript to ES2015, and we now publish the source code in the npm distribution.

This release includes some breaking changes to optional parts of the Action Cable JavaScript API:

- Configuration of the WebSocket adapter and logger adapter have been moved from properties of `ActionCable` to properties of `ActionCable.adapters`. If you are configuring these adapters you will need to make these changes:

```
-    ActionCable.WebSocket = MyWebSocket
+    ActionCable.adapters.WebSocket = MyWebSocket
```

```
-    ActionCable.logger = myLogger
+    ActionCable.adapters.logger = myLogger
```

- The `ActionCable.startDebugging()` and `ActionCable.stopDebugging()` methods have been removed and replaced with the property `ActionCable.logger.enabled`. If you are using these methods you will need to make these changes:

```
-    ActionCable.startDebugging()
+    ActionCable.logger.enabled = true
```

```
-    ActionCable.stopDebugging()
+    ActionCable.logger.enabled = false
```

`ActionDispatch::Response#content_type` now returns the Content-Type header without modification

Previously, the return value of `ActionDispatch::Response#content_type` did NOT contain the charset part. This behavior has changed to include the previously omitted charset part as well.

If you want just the MIME type, please use `ActionDispatch::Response#media_type` instead.

Before:

```
resp = ActionDispatch::Response.new(200, "Content-Type" => "text/csv;
header=present; charset=utf-16")
resp.content_type #=> "text/csv; header=present"
```

After:

```
resp = ActionDispatch::Response.new(200, "Content-Type" => "text/csv;
header=present; charset=utf-16")
resp.content_type #=> "text/csv; header=present; charset=utf-16"
resp.media_type   #=> "text/csv"
```

Autoloading

The default configuration for Rails 6

```
# config/application.rb

config.load_defaults 6.0
```

enables `zeitwerk` autoloading mode on CRuby. In that mode, autoloading, reloading, and eager loading are managed by Zeitwerk.

If you are using defaults from a previous Rails version, you can enable zeitwerk like so:

```
# config/application.rb

config.autoloader = :zeitwerk
```

Public API

In general, applications do not need to use the API of Zeitwerk directly. Rails sets things up according to the existing contract: `config.autoload_paths`, `config.cache_classes`, etc.

While applications should stick to that interface, the actual Zeitwerk loader object can be accessed as

```
Rails.autoloaders.main
```

That may be handy if you need to preload Single Table Inheritance (STI) classes or configure a custom inflector, for example.

Project Structure

If the application being upgraded autoloads correctly, the project structure should be already mostly compatible.

However, `classic` mode infers file names from missing constant names (`underscore`), whereas `zeitwerk` mode infers constant names from file names (`camelize`). These helpers are not always inverse of each other, in particular if acronyms are involved. For instance, `"FOO".underscore` is `"foo"`, but `"foo".camelize` is `"Foo"`, not `"FOO"`.

Compatibility can be checked with the `zeitwerk:check` task:

```
$ bin/rails zeitwerk:check
Hold on, I am eager loading the application.
All is good!
```

require_dependency

All known use cases of `require_dependency` have been eliminated, you should grep the project and delete them.

If your application uses Single Table Inheritance, please see the Single Table Inheritance section of the Autoloading and Reloading Constants (Zeitwerk Mode) guide.

Qualified names in class and module definitions

You can now robustly use constant paths in class and module definitions:

```
# Autoloading in this class' body matches Ruby semantics now.
class Admin::UsersController < ApplicationController
  # ...
end
```

A gotcha to be aware of is that, depending on the order of execution, the classic autoloader could sometimes be able to autoload `Foo::Wadus` in

```
class Foo::Bar
  Wadus
end
```

That does not match Ruby semantics because `Foo` is not in the nesting, and won't work at all in `zeitwerk` mode. If you find such corner case you can use the qualified name `Foo::Wadus`:

```
class Foo::Bar
  Foo::Wadus
end
```

or add `Foo` to the nesting:

```
module Foo
  class Bar
    Wadus
  end
end
```

Concerns

You can autoload and eager load from a standard structure like

```
app/models
app/models/concerns
```

In that case, `app/models/concerns` is assumed to be a root directory (because it belongs to the autoload paths), and it is ignored as namespace. So, `app/models/concerns/foo.rb` should define `Foo`, not `Concerns::Foo`.

The `Concerns::` namespace worked with the classic autoloader as a side-effect of the implementation, but it was not really an intended behavior. An application using `Concerns::` needs to rename those classes and modules to be able to run in `zeitwerk` mode.

Having `app` in the autoload paths

Some projects want something like `app/api/base.rb` to define `API::Base`, and add `app` to the autoload paths to accomplish that in `classic` mode. Since Rails adds all subdirectories of `app` to the autoload paths automatically, we have another situation in which there are nested root directories, so that setup no longer works. Similar principle we explained above with `concerns`.

If you want to keep that structure, you'll need to delete the subdirectory from the autoload paths in an initializer:

```
ActiveSupport::Dependencies.autoload_paths.delete("#{Rails.root}/app/api")
```

Autoloaded Constants and Explicit Namespaces

If a namespace is defined in a file, as `Hotel` is here:

```
app/models/hotel.rb         # Defines Hotel.
app/models/hotel/pricing.rb # Defines Hotel::Pricing.
```

the `Hotel` constant has to be set using the `class` or `module` keywords. For example:

```
class Hotel
end
```

is good.

Alternatives like

```
Hotel = Class.new
```

or

```
Hotel = Struct.new
```

won't work, child objects like `Hotel::Pricing` won't be found.

This restriction only applies to explicit namespaces. Classes and modules not defining a namespace can be defined using those idioms.

One file, one constant (at the same top-level)

In `classic` mode you could technically define several constants at the same top-level and have them all reloaded. For example, given

```
# app/models/foo.rb

class Foo
end
```

```
class Bar
end
```

while `Bar` could not be autoloaded, autoloading `Foo` would mark `Bar` as autoloaded too. This is not the case in `zeitwerk` mode, you need to move `Bar` to its own file `bar.rb`. One file, one constant.

This only applies to constants at the same top-level as in the example above. Inner classes and modules are fine. For example, consider

```
# app/models/foo.rb

class Foo
  class InnerClass
  end
end
```

If the application reloads `Foo`, it will reload `Foo::InnerClass` too.

Spring and the `test` Environment

Spring reloads the application code if something changes. In the `test` environment you need to enable reloading for that to work:

```
# config/environments/test.rb

config.cache_classes = false
```

Otherwise you'll get this error:

```
reloading is disabled because config.cache_classes is true
```

Bootsnap

Bootsnap should be at least version 1.4.2.

In addition to that, Bootsnap needs to disable the iseq cache due to a bug in the interpreter if running Ruby 2.5. Please make sure to depend on at least Bootsnap 1.4.4 in that case.

`config.add_autoload_paths_to_load_path`

The new configuration point `config.add_autoload_paths_to_load_path` is `true` by default for backwards compatibility, but allows you to opt-out from adding the autoload paths to `$LOAD_PATH`.

This makes sense in most applications, since you never should require a file in `app/models`, for example, and Zeitwerk only uses absolute file names internally.

By opting-out you optimize `$LOAD_PATH` lookups (less directories to check), and save Bootsnap work and memory consumption, since it does not need to build an index for these directories.

Thread-safety

In classic mode, constant autoloading is not thread-safe, though Rails has locks in place for example to make web requests thread-safe when autoloading is enabled, as it is common in the development environment.

Constant autoloading is thread-safe in `zeitwerk` mode. For example, you can now autoload in multi-threaded scripts executed by the `runner` command.

Globs in config.autoload_paths

Beware of configurations like

```
config.autoload_paths += Dir["#{config.root}/lib/**/"]
```

Every element of `config.autoload_paths` should represent the top-level namespace (`Object`) and they cannot be nested in consequence (with the exception of `concerns` directories explained above).

To fix this, just remove the wildcards:

```
config.autoload_paths << "#{config.root}/lib"
```

Eager loading and autoloading are consistent

In `classic` mode, if `app/models/foo.rb` defines `Bar` , you won't be able to autoload that file, but eager loading will work because it loads files recursively blindly. This can be a source of errors if you test things first eager loading, execution may fail later autoloading.

In `zeitwerk` mode both loading modes are consistent, they fail and err in the same files.

How to Use the Classic Autoloader in Rails 6

Applications can load Rails 6 defaults and still use the classic autoloader by setting `config.autoloader` this way:

```
# config/application.rb

config.load_defaults 6.0
config.autoloader = :classic
```

When using the Classic Autoloader in Rails 6 application it is recommended to set concurrency level to 1 in development environment, for the web servers and background processors, due to the thread-safety concerns.

Active Storage assignment behavior change

With the configuration defaults for Rails 5.2, assigning to a collection of attachments declared with `has_many_attached` appends new files:

```
class User < ApplicationRecord
  has_many_attached :highlights
end

user.highlights.attach(filename: "funky.jpg", ...)
user.highlights.count # => 1

blob = ActiveStorage::Blob.create_after_upload!(filename: "town.jpg", ...)
```

```
user.update!(highlights: [ blob ])

user.highlights.count # => 2
user.highlights.first.filename # => "funky.jpg"
user.highlights.second.filename # => "town.jpg"
```

With the configuration defaults for Rails 6.0, assigning to a collection of attachments replaces existing files instead of appending to them. This matches Active Record behavior when assigning to a collection association:

```
user.highlights.attach(filename: "funky.jpg", ...)
user.highlights.count # => 1

blob = ActiveStorage::Blob.create_after_upload!(filename: "town.jpg", ...)
user.update!(highlights: [ blob ])

user.highlights.count # => 1
user.highlights.first.filename # => "town.jpg"
```

`#attach` can be used to add new attachments without removing the existing ones:

```
blob = ActiveStorage::Blob.create_after_upload!(filename: "town.jpg", ...)
user.highlights.attach(blob)

user.highlights.count # => 2
user.highlights.first.filename # => "funky.jpg"
user.highlights.second.filename # => "town.jpg"
```

Existing applications can opt in to this new behavior by setting `config.active_storage.replace_on_assign_to_many` to `true`. The old behavior will be deprecated in Rails 7.0 and removed in Rails 7.1.

Upgrading from Rails 5.1 to Rails 5.2

For more information on changes made to Rails 5.2 please see the release notes.

Bootsnap

Rails 5.2 adds bootsnap gem in the newly generated app's Gemfile. The `app:update` command sets it up in `boot.rb`. If you want to use it, then add it in the Gemfile:

```
# Reduces boot times through caching; required in config/boot.rb
gem 'bootsnap', require: false
```

Otherwise change the `boot.rb` to not use bootsnap.

Expiry in signed or encrypted cookie is now embedded in the cookies values

To improve security, Rails now embeds the expiry information also in encrypted or signed cookies value.

This new embedded information makes those cookies incompatible with versions of Rails older than 5.2.

If you require your cookies to be read by 5.1 and older, or you are still validating your 5.2 deploy and want to allow you to rollback set `Rails.application.config.action_dispatch.use_authenticated_cookie_encryption` to `false`.

Upgrading from Rails 5.0 to Rails 5.1

For more information on changes made to Rails 5.1 please see the release notes.

Top-level `HashWithIndifferentAccess` is soft-deprecated

If your application uses the top-level `HashWithIndifferentAccess` class, you should slowly move your code to instead use `ActiveSupport::HashWithIndifferentAccess`.

It is only soft-deprecated, which means that your code will not break at the moment and no deprecation warning will be displayed, but this constant will be removed in the future.

Also, if you have pretty old YAML documents containing dumps of such objects, you may need to load and dump them again to make sure that they reference the right constant, and that loading them won't break in the future.

`application.secrets` now loaded with all keys as symbols

If your application stores nested configuration in `config/secrets.yml`, all keys are now loaded as symbols, so access using strings should be changed.

From:

```
Rails.application.secrets[:smtp_settings]["address"]
```

To:

```
Rails.application.secrets[:smtp_settings][:address]
```

Removed deprecated support to `:text` and `:nothing` in `render`

If your controllers are using `render :text`, they will no longer work. The new method of rendering text with MIME type of `text/plain` is to use `render :plain`.

Similarly, `render :nothing` is also removed and you should use the `head` method to send responses that contain only headers. For example, `head :ok` sends a 200 response with no body to render.

Removed deprecated support of `redirect_to :back`

In Rails 5.0, `redirect_to :back` was deprecated. In Rails 5.1, it was removed completely.

As an alternative, use `redirect_back`. It's important to note that `redirect_back` also takes a `fallback_location` option which will be used in case the `HTTP_REFERER` is missing.

```
redirect_back(fallback_location: root_path)
```

Upgrading from Rails 4.2 to Rails 5.0

For more information on changes made to Rails 5.0 please see the release notes.

Ruby 2.2.2+ required

From Ruby on Rails 5.0 onwards, Ruby 2.2.2+ is the only supported Ruby version. Make sure you are on Ruby 2.2.2 version or greater, before you proceed.

Active Record Models Now Inherit from ApplicationRecord by Default

In Rails 4.2, an Active Record model inherits from `ActiveRecord::Base`. In Rails 5.0, all models inherit from `ApplicationRecord`.

`ApplicationRecord` is a new superclass for all app models, analogous to app controllers subclassing `ApplicationController` instead of `ActionController::Base`. This gives apps a single spot to configure app-wide model behavior.

When upgrading from Rails 4.2 to Rails 5.0, you need to create an `application_record.rb` file in `app/models/` and add the following content:

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Then make sure that all your models inherit from it.

Halting Callback Chains via `throw(:abort)`

In Rails 4.2, when a 'before' callback returns `false` in Active Record and Active Model, then the entire callback chain is halted. In other words, successive 'before' callbacks are not executed, and neither is the action wrapped in callbacks.

In Rails 5.0, returning `false` in an Active Record or Active Model callback will not have this side effect of halting the callback chain. Instead, callback chains must be explicitly halted by calling `throw(:abort)`.

When you upgrade from Rails 4.2 to Rails 5.0, returning `false` in those kind of callbacks will still halt the callback chain, but you will receive a deprecation warning about this upcoming change.

When you are ready, you can opt into the new behavior and remove the deprecation warning by adding the following configuration to your `config/application.rb`:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

Note that this option will not affect Active Support callbacks since they never halted the chain when any value was returned.

See #17227 for more details.

ActiveJob Now Inherits from ApplicationJob by Default

In Rails 4.2, an Active Job inherits from `ActiveJob::Base`. In Rails 5.0, this behavior has changed to now inherit from `ApplicationJob`.

When upgrading from Rails 4.2 to Rails 5.0, you need to create an `application_job.rb` file in `app/jobs/` and add the following content:

```
class ApplicationJob < ActiveJob::Base
end
```

Then make sure that all your job classes inherit from it.

See #19034 for more details.

Rails Controller Testing

Extraction of some helper methods to `rails-controller-testing`

`assigns` and `assert_template` have been extracted to the `rails-controller-testing` gem. To continue using these methods in your controller tests, add `gem 'rails-controller-testing'` to your `Gemfile`.

If you are using RSpec for testing, please see the extra configuration required in the gem's documentation.

New behavior when uploading files

If you are using `ActionDispatch::Http::UploadedFile` in your tests to upload files, you will need to change to use the similar `Rack::Test::UploadedFile` class instead.

See #26404 for more details.

Autoloading is Disabled After Booting in the Production Environment

Autoloading is now disabled after booting in the production environment by default.

Eager loading the application is part of the boot process, so top-level constants are fine and are still autoloaded, no need to require their files.

Constants in deeper places only executed at runtime, like regular method bodies, are also fine because the file defining them will have been eager loaded while booting.

For the vast majority of applications this change needs no action. But in the very rare event that your application needs autoloading while running in production, set `Rails.application.config.enable_dependency_loading` to true.

XML Serialization

`ActiveModel::Serializers::Xml` has been extracted from Rails to the `activemodel-serializers-xml` gem. To continue using XML serialization in your application, add `gem 'activemodel-serializers-xml'` to your `Gemfile`.

Removed Support for Legacy `mysql` Database Adapter

Rails 5 removes support for the legacy `mysql` database adapter. Most users should be able to use `mysql2` instead. It will be converted to a separate gem when we find someone to maintain it.

Removed Support for Debugger

`debugger` is not supported by Ruby 2.2 which is required by Rails 5. Use `byebug` instead.

Use `bin/rails` for running tasks and tests

Rails 5 adds the ability to run tasks and tests through `bin/rails` instead of rake. Generally these changes are in parallel with rake, but some were ported over altogether.

To use the new test runner simply type `bin/rails test`.

`rake dev:cache` is now `bin/rails dev:cache`.

Run `bin/rails` inside your application's root directory to see the list of commands available.

`ActionController::Parameters` No Longer Inherits from `HashWithIndifferentAccess`

Calling `params` in your application will now return an object instead of a hash. If your parameters are already permitted, then you will not need to make any changes. If you are using `map` and other methods that depend on being able to read the hash regardless of `permitted?` you will need to upgrade your application to first permit and then convert to a hash.

```
params.permit([:proceed_to, :return_to]).to_h
```

`protect_from_forgery` Now Defaults to `prepend: false`

`protect_from_forgery` defaults to `prepend: false` which means that it will be inserted into the callback chain at the point in which you call it in your application. If you want `protect_from_forgery` to always run first, then you should change your application to use `protect_from_forgery prepend: true`.

Default Template Handler is Now RAW

Files without a template handler in their extension will be rendered using the raw handler. Previously Rails would render files using the ERB template handler.

If you do not want your file to be handled via the raw handler, you should add an extension to your file that can be parsed by the appropriate template handler.

Added Wildcard Matching for Template Dependencies

You can now use wildcard matching for your template dependencies. For example, if you were defining your templates as such:

```
<% # Template Dependency: recordings/threads/events/subscribers_changed %>
<% # Template Dependency: recordings/threads/events/completed %>
<% # Template Dependency: recordings/threads/events/uncompleted %>
```

You can now just call the dependency once with a wildcard.

```
<% # Template Dependency: recordings/threads/events/* %>
```

`ActionView::Helpers::RecordTagHelper` moved to external gem (record_tag_helper)

`content_tag_for` and `div_for` have been removed in favor of just using `content_tag`. To continue using the older methods, add the `record_tag_helper` gem to your `Gemfile`:

```
gem 'record_tag_helper', '~> 1.0'
```

See #18411 for more details.

Removed Support for `protected_attributes` Gem

The `protected_attributes` gem is no longer supported in Rails 5.

Removed support for `activerecord-deprecated_finders` gem

The `activerecord-deprecated_finders` gem is no longer supported in Rails 5.

`ActiveSupport::TestCase` Default Test Order is Now Random

When tests are run in your application, the default order is now `:random` instead of `:sorted`. Use the following config option to set it back to `:sorted`.

```
# config/environments/test.rb
Rails.application.configure do
  config.active_support.test_order = :sorted
end
```

`ActionController::Live` became a `Concern`

If you include `ActionController::Live` in another module that is included in your controller, then you should also extend the module with `ActiveSupport::Concern`. Alternatively, you can use the `self.included` hook to include `ActionController::Live` directly to the controller once the `StreamingSupport` is included.

This means that if your application used to have its own streaming module, the following code would break in production:

```
# This is a work-around for streamed controllers performing authentication with
Warden/Devise.
# See https://github.com/plataformatec/devise/issues/2332
# Authenticating in the router is another solution as suggested in that issue
class StreamingSupport
  include ActionController::Live # this won't work in production for Rails 5
  # extend ActiveSupport::Concern # unless you uncomment this line.

  def process(name)
    super(name)
  rescue ArgumentError => e
    if e.message == 'uncaught throw :warden'
      throw :warden
    else
      raise e
    end
  end
end
```

New Framework Defaults

Active Record `belongs_to` Required by Default Option

`belongs_to` will now trigger a validation error by default if the association is not present.

This can be turned off per-association with `optional: true`.

This default will be automatically configured in new applications. If an existing application wants to add this feature it will need to be turned on in an initializer:

```
config.active_record.belongs_to_required_by_default = true
```

The configuration is by default global for all your models, but you can override it on a per model basis. This should help you migrate all your models to have their associations required by default.

```
class Book < ApplicationRecord
  # model is not yet ready to have its association required by default

  self.belongs_to_required_by_default = false
  belongs_to(:author)
end

class Car < ApplicationRecord
  # model is ready to have its association required by default

  self.belongs_to_required_by_default = true
  belongs_to(:pilot)
end
```

Per-form CSRF Tokens

Rails 5 now supports per-form CSRF tokens to mitigate against code-injection attacks with forms created by JavaScript. With this option turned on, forms in your application will each have their own CSRF token that is specific to the action and method for that form.

```
config.action_controller.per_form_csrf_tokens = true
```

Forgery Protection with Origin Check

You can now configure your application to check if the HTTP `Origin` header should be checked against the site's origin as an additional CSRF defense. Set the following in your config to true:

```
config.action_controller.forgery_protection_origin_check = true
```

Allow Configuration of Action Mailer Queue Name

The default mailer queue name is `mailers`. This configuration option allows you to globally change the queue name. Set the following in your config:

```
config.action_mailer.deliver_later_queue_name = :new_queue_name
```

Support Fragment Caching in Action Mailer Views

Set `config.action_mailer.perform_caching` in your config to determine whether your Action Mailer views should support caching.

```
config.action_mailer.perform_caching = true
```

Configure the Output of `db:structure:dump`

If you're using `schema_search_path` or other PostgreSQL extensions, you can control how the schema is dumped. Set to `:all` to generate all dumps, or to `:schema_search_path` to generate from schema search path.

```
config.active_record.dump_schemas = :all
```

Configure SSL Options to Enable HSTS with Subdomains

Set the following in your config to enable HSTS when using subdomains:

```
config.ssl_options = { hsts: { subdomains: true } }
```

Preserve Timezone of the Receiver

When using Ruby 2.4, you can preserve the timezone of the receiver when calling `to_time`.

```
ActiveSupport.to_time_preserves_timezone = false
```

Changes with JSON/JSONB serialization

In Rails 5.0, how JSON/JSONB attributes are serialized and deserialized changed. Now, if you set a column equal to a `String`, Active Record will no longer turn that string into a `Hash`, and will instead only return the string. This is not limited to code interacting with models, but also affects `:default` column settings in `db/schema.rb`. It is recommended that you do not set columns equal to a `String`, but pass a `Hash` instead, which will be converted to and from a JSON string automatically.

Upgrading from Rails 4.1 to Rails 4.2

Web Console

First, add `gem 'web-console', '~> 2.0'` to the `:development` group in your `Gemfile` and run `bundle install` (it won't have been included when you upgraded Rails). Once it's been installed, you can simply drop a reference to the console helper (i.e., `<%= console %>`) into any view you want to enable it for. A console will also be provided on any error page you view in your development environment.

Responders

`respond_with` and the class-level `respond_to` methods have been extracted to the `responders` gem. To use them, simply add `gem 'responders', '~> 2.0'` to your `Gemfile`. Calls to `respond_with` and `respond_to` (again, at the class level) will no longer work without having included the `responders` gem in your dependencies:

```ruby
# app/controllers/users_controller.rb

class UsersController < ApplicationController
  respond_to :html, :json

  def show
    @user = User.find(params[:id])
    respond_with @user
  end
end
```

Instance-level `respond_to` is unaffected and does not require the additional gem:

```ruby
# app/controllers/users_controller.rb

class UsersController < ApplicationController
  def show
    @user = User.find(params[:id])
    respond_to do |format|
      format.html
      format.json { render json: @user }
    end
  end
end
```

See #16526 for more details.

Error handling in transaction callbacks

Currently, Active Record suppresses errors raised within `after_rollback` or `after_commit` callbacks and only prints them to the logs. In the next version, these errors will no longer be suppressed. Instead, the errors will propagate normally just like in other Active Record callbacks.

When you define an `after_rollback` or `after_commit` callback, you will receive a deprecation warning about this upcoming change. When you are ready, you can opt into the new behavior and remove the deprecation warning by adding following configuration to your `config/application.rb`:

```ruby
config.active_record.raise_in_transactional_callbacks = true
```

See #14488 and #16537 for more details.

Ordering of test cases

In Rails 5.0, test cases will be executed in random order by default. In anticipation of this change, Rails 4.2 introduced a new configuration option `active_support.test_order` for explicitly specifying the test ordering. This allows you to either lock down the current behavior by setting the option to `:sorted`, or opt into the future behavior by setting the option to `:random`.

If you do not specify a value for this option, a deprecation warning will be emitted. To avoid this, add the following line to your test environment:

```
# config/environments/test.rb
Rails.application.configure do
  config.active_support.test_order = :sorted # or `:random` if you prefer
end
```

Serialized attributes

When using a custom coder (e.g. `serialize :metadata, JSON`), assigning `nil` to a serialized attribute will save it to the database as `NULL` instead of passing the `nil` value through the coder (e.g. `"null"` when using the `JSON` coder).

Production log level

In Rails 5, the default log level for the production environment will be changed to `:debug` (from `:info`). To preserve the current default, add the following line to your `production.rb`:

```
# Set to `:info` to match the current default, or set to `:debug` to opt-into
# the future default.
config.log_level = :info
```

`after_bundle` in Rails templates

If you have a Rails template that adds all the files in version control, it fails to add the generated binstubs because it gets executed before Bundler:

```
# template.rb
generate(:scaffold, "person name:string")
route "root to: 'people#index'"
rake("db:migrate")

git :init
git add: "."
git commit: %Q{ -m 'Initial commit' }
```

You can now wrap the `git` calls in an `after_bundle` block. It will be run after the binstubs have been generated.

```
# template.rb
generate(:scaffold, "person name:string")
route "root to: 'people#index'"
rake("db:migrate")

after_bundle do
  git :init
  git add: "."
  git commit: %Q{ -m 'Initial commit' }
end
```

Rails HTML Sanitizer

There's a new choice for sanitizing HTML fragments in your applications. The venerable html-scanner approach is now officially being deprecated in favor of `Rails HTML Sanitizer`.

This means the methods `sanitize`, `sanitize_css`, `strip_tags` and `strip_links` are backed by a new implementation.

This new sanitizer uses Loofah internally. Loofah in turn uses Nokogiri, which wraps XML parsers written in both C and Java, so sanitization should be faster no matter which Ruby version you run.

The new version updates `sanitize`, so it can take a `Loofah::Scrubber` for powerful scrubbing. See some examples of scrubbers here.

Two new scrubbers have also been added: `PermitScrubber` and `TargetScrubber`. Read the gem's readme for more information.

The documentation for `PermitScrubber` and `TargetScrubber` explains how you can gain complete control over when and how elements should be stripped.

If your application needs to use the old sanitizer implementation, include `rails-deprecated_sanitizer` in your `Gemfile`:

```
gem 'rails-deprecated_sanitizer'
```

Rails DOM Testing

The `TagAssertions` module (containing methods such as `assert_tag`), has been deprecated in favor of the `assert_select` methods from the `SelectorAssertions` module, which has been extracted into the rails-dom-testing gem.

Masked Authenticity Tokens

In order to mitigate SSL attacks, `form_authenticity_token` is now masked so that it varies with each request. Thus, tokens are validated by unmasking and then decrypting. As a result, any strategies for verifying requests from non-rails forms that relied on a static session CSRF token have to take this into account.

Action Mailer

Previously, calling a mailer method on a mailer class will result in the corresponding instance method being executed directly. With the introduction of Active Job and `#deliver_later`, this is no longer true. In Rails 4.2, the invocation of the instance methods are deferred until either `deliver_now` or `deliver_later` is called. For example:

```
class Notifier < ActionMailer::Base
  def notify(user, ...)
    puts "Called"
    mail(to: user.email, ...)
  end
end
```

```
mail = Notifier.notify(user, ...) # Notifier#notify is not yet called at this point
mail = mail.deliver_now           # Prints "Called"
```

This should not result in any noticeable differences for most applications. However, if you need some non-mailer methods to be executed synchronously, and you were previously relying on the synchronous proxying behavior, you should define them as class methods on the mailer class directly:

```
class Notifier < ActionMailer::Base
  def self.broadcast_notifications(users, ...)
    users.each { |user| Notifier.notify(user, ...) }
  end
end
```

Foreign Key Support

The migration DSL has been expanded to support foreign key definitions. If you've been using the Foreigner gem, you might want to consider removing it. Note that the foreign key support of Rails is a subset of Foreigner. This means that not every Foreigner definition can be fully replaced by its Rails migration DSL counterpart.

The migration procedure is as follows:

1. remove `gem "foreigner"` from the `Gemfile`.
2. run `bundle install`.
3. run `bin/rake db:schema:dump`.
4. make sure that `db/schema.rb` contains every foreign key definition with the necessary options.

Upgrading from Rails 4.0 to Rails 4.1

CSRF protection from remote `<script>` tags

Or, "whaaat my tests are failing!!!?" or "my `<script>` widget is busted!!"

Cross-site request forgery (CSRF) protection now covers GET requests with JavaScript responses, too. This prevents a third-party site from remotely referencing your JavaScript with a `<script>` tag to extract sensitive data.

This means that your functional and integration tests that use

```
get :index, format: :js
```

will now trigger CSRF protection. Switch to

```
xhr :get, :index, format: :js
```

to explicitly test an `XmlHttpRequest`.

NOTE: Your own `<script>` tags are treated as cross-origin and blocked by default, too. If you really mean to load JavaScript from `<script>` tags, you must now explicitly skip CSRF protection on those actions.

Spring

If you want to use Spring as your application preloader you need to:

1. Add `gem 'spring', group: :development` to your `Gemfile`.
2. Install spring using `bundle install`.
3. Generate the Spring binstub with `bundle exec spring binstub`.

NOTE: User defined rake tasks will run in the `development` environment by default. If you want them to run in other environments consult the Spring README.

`config/secrets.yml`

If you want to use the new `secrets.yml` convention to store your application's secrets, you need to:

1. Create a `secrets.yml` file in your `config` folder with the following content:

```
development:
  secret_key_base:

test:
  secret_key_base:

production:
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

2. Use your existing `secret_key_base` from the `secret_token.rb` initializer to set the `SECRET_KEY_BASE` environment variable for whichever users running the Rails application in production. Alternatively, you can simply copy the existing `secret_key_base` from the `secret_token.rb` initializer to `secrets.yml` under the `production` section, replacing `<%= ENV["SECRET_KEY_BASE"] %>`.

3. Remove the `secret_token.rb` initializer.

4. Use `rake secret` to generate new keys for the `development` and `test` sections.

5. Restart your server.

Changes to test helper

If your test helper contains a call to `ActiveRecord::Migration.check_pending!` this can be removed. The check is now done automatically when you `require "rails/test_help"`, although leaving this line in your helper is not harmful in any way.

Cookies serializer

Applications created before Rails 4.1 uses `Marshal` to serialize cookie values into the signed and encrypted cookie jars. If you want to use the new `JSON`-based format in your application, you can add an initializer file with the following content:

```
Rails.application.config.action_dispatch.cookies_serializer = :hybrid
```

This would transparently migrate your existing `Marshal`-serialized cookies into the new `JSON`-based format.

When using the `:json` or `:hybrid` serializer, you should beware that not all Ruby objects can be serialized as JSON. For example, `Date` and `Time` objects will be serialized as strings, and `Hash`es will have their keys stringified.

```ruby
class CookiesController < ApplicationController
  def set_cookie
    cookies.encrypted[:expiration_date] = Date.tomorrow # => Thu, 20 Mar 2014
    redirect_to action: 'read_cookie'
  end

  def read_cookie
    cookies.encrypted[:expiration_date] # => "2014-03-20"
  end
end
```

It's advisable that you only store simple data (strings and numbers) in cookies. If you have to store complex objects, you would need to handle the conversion manually when reading the values on subsequent requests.

If you use the cookie session store, this would apply to the `session` and `flash` hash as well.

Flash structure changes

Flash message keys are normalized to strings. They can still be accessed using either symbols or strings. Looping through the flash will always yield string keys:

```ruby
flash["string"] = "a string"
flash[:symbol] = "a symbol"

# Rails < 4.1
flash.keys # => ["string", :symbol]

# Rails >= 4.1
flash.keys # => ["string", "symbol"]
```

Make sure you are comparing Flash message keys against strings.

Changes in JSON handling

There are a few major changes related to JSON handling in Rails 4.1.

MultiJSON removal

MultiJSON has reached its end-of-life and has been removed from Rails.

If your application currently depends on MultiJSON directly, you have a few options:

1. Add 'multi_json' to your `Gemfile`. Note that this might cease to work in the future
2. Migrate away from MultiJSON by using `obj.to_json`, and `JSON.parse(str)` instead.

WARNING: Do not simply replace `MultiJson.dump` and `MultiJson.load` with `JSON.dump` and `JSON.load`. These JSON gem APIs are meant for serializing and deserializing arbitrary Ruby objects and are generally unsafe.

JSON gem compatibility

Historically, Rails had some compatibility issues with the JSON gem. Using `JSON.generate` and `JSON.dump` inside a Rails application could produce unexpected errors.

Rails 4.1 fixed these issues by isolating its own encoder from the JSON gem. The JSON gem APIs will function as normal, but they will not have access to any Rails-specific features. For example:

```
class FooBar
  def as_json(options = nil)
    { foo: 'bar' }
  end
end
```

```
irb> FooBar.new.to_json
=> "{\"foo\":\"bar\"}"
irb> JSON.generate(FooBar.new, quirks_mode: true)
=> "\"#<FooBar:0x007fa80a481610>\""
```

New JSON encoder

The JSON encoder in Rails 4.1 has been rewritten to take advantage of the JSON gem. For most applications, this should be a transparent change. However, as part of the rewrite, the following features have been removed from the encoder:

1. Circular data structure detection
2. Support for the `encode_json` hook
3. Option to encode `BigDecimal` objects as numbers instead of strings

If your application depends on one of these features, you can get them back by adding the `activesupport-json_encoder` gem to your `Gemfile`.

JSON representation of Time objects

`#as_json` for objects with time component (`Time`, `DateTime`, `ActiveSupport::TimeWithZone`) now returns millisecond precision by default. If you need to keep old behavior with no millisecond precision, set the following in an initializer:

```
ActiveSupport::JSON::Encoding.time_precision = 0
```

Usage of `return` within inline callback blocks

Previously, Rails allowed inline callback blocks to use `return` this way:

```
class ReadOnlyModel < ActiveRecord::Base
  before_save { return false } # BAD
end
```

This behavior was never intentionally supported. Due to a change in the internals of `ActiveSupport::Callbacks`, this is no longer allowed in Rails 4.1. Using a `return` statement in an inline callback block causes a `LocalJumpError` to be raised when the callback is executed.

Inline callback blocks using `return` can be refactored to evaluate to the returned value:

```ruby
class ReadOnlyModel < ActiveRecord::Base
  before_save { false } # GOOD
end
```

Alternatively, if `return` is preferred it is recommended to explicitly define a method:

```ruby
class ReadOnlyModel < ActiveRecord::Base
  before_save :before_save_callback # GOOD

  private
    def before_save_callback
      return false
    end
end
```

This change applies to most places in Rails where callbacks are used, including Active Record and Active Model callbacks, as well as filters in Action Controller (e.g. `before_action`).

See this pull request for more details.

Methods defined in Active Record fixtures

Rails 4.1 evaluates each fixture's ERB in a separate context, so helper methods defined in a fixture will not be available in other fixtures.

Helper methods that are used in multiple fixtures should be defined on modules included in the newly introduced `ActiveRecord::FixtureSet.context_class`, in `test_helper.rb`.

```ruby
module FixtureFileHelpers
  def file_sha(path)
    OpenSSL::Digest::SHA256.hexdigest(File.read(Rails.root.join('test/fixtures',
path)))
  end
end

ActiveRecord::FixtureSet.context_class.include FixtureFileHelpers
```

I18n enforcing available locales

Rails 4.1 now defaults the I18n option `enforce_available_locales` to `true`. This means that it will make sure that all locales passed to it must be declared in the `available_locales` list.

To disable it (and allow I18n to accept *any* locale option) add the following configuration to your application:

```ruby
config.i18n.enforce_available_locales = false
```

Note that this option was added as a security measure, to ensure user input cannot be used as locale information unless it is previously known. Therefore, it's recommended not to disable this option unless you have a strong reason for doing so.

Mutator methods called on Relation

`Relation` no longer has mutator methods like `#map!` and `#delete_if`. Convert to an `Array` by calling `#to_a` before using these methods.

It intends to prevent odd bugs and confusion in code that call mutator methods directly on the `Relation`.

```
# Instead of this
Author.where(name: 'Hank Moody').compact!

# Now you have to do this
authors = Author.where(name: 'Hank Moody').to_a
authors.compact!
```

Changes on Default Scopes

Default scopes are no longer overridden by chained conditions.

In previous versions when you defined a `default_scope` in a model it was overridden by chained conditions in the same field. Now it is merged like any other scope.

Before:

```
class User < ActiveRecord::Base
  default_scope { where state: 'pending' }
  scope :active, -> { where state: 'active' }
  scope :inactive, -> { where state: 'inactive' }
end

User.all
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending'

User.active
# SELECT "users".* FROM "users" WHERE "users"."state" = 'active'

User.where(state: 'inactive')
# SELECT "users".* FROM "users" WHERE "users"."state" = 'inactive'
```

After:

```
class User < ActiveRecord::Base
  default_scope { where state: 'pending' }
  scope :active, -> { where state: 'active' }
  scope :inactive, -> { where state: 'inactive' }
end

User.all
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending'

User.active
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending' AND
```

```
"users"."state" = 'active'

User.where(state: 'inactive')
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending' AND
"users"."state" = 'inactive'
```

To get the previous behavior it is needed to explicitly remove the `default_scope` condition using `unscoped`, `unscope`, `rewhere` or `except`.

```
class User < ActiveRecord::Base
  default_scope { where state: 'pending' }
  scope :active, -> { unscope(where: :state).where(state: 'active') }
  scope :inactive, -> { rewhere state: 'inactive' }
end

User.all
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending'

User.active
# SELECT "users".* FROM "users" WHERE "users"."state" = 'active'

User.inactive
# SELECT "users".* FROM "users" WHERE "users"."state" = 'inactive'
```

Rendering content from string

Rails 4.1 introduces `:plain`, `:html`, and `:body` options to `render`. Those options are now the preferred way to render string-based content, as it allows you to specify which content type you want the response sent as.

- `render :plain` will set the content type to `text/plain`
- `render :html` will set the content type to `text/html`
- `render :body` will *not* set the content type header.

From the security standpoint, if you don't expect to have any markup in your response body, you should be using `render :plain` as most browsers will escape unsafe content in the response for you.

We will be deprecating the use of `render :text` in a future version. So please start using the more precise `:plain`, `:html`, and `:body` options instead. Using `render :text` may pose a security risk, as the content is sent as `text/html`.

PostgreSQL json and hstore datatypes

Rails 4.1 will map `json` and `hstore` columns to a string-keyed Ruby `Hash`. In earlier versions, a `HashWithIndifferentAccess` was used. This means that symbol access is no longer supported. This is also the case for `store_accessors` based on top of `json` or `hstore` columns. Make sure to use string keys consistently.

Explicit block use for `ActiveSupport::Callbacks`

Rails 4.1 now expects an explicit block to be passed when calling `ActiveSupport::Callbacks.set_callback`. This change stems from `ActiveSupport::Callbacks` being largely rewritten for the 4.1 release.

```
# Previously in Rails 4.0
set_callback :save, :around, ->(r, &block) { stuff; result = block.call; stuff }

# Now in Rails 4.1
set_callback :save, :around, ->(r, block) { stuff; result = block.call; stuff }
```

Upgrading from Rails 3.2 to Rails 4.0

If your application is currently on any version of Rails older than 3.2.x, you should upgrade to Rails 3.2 before attempting one to Rails 4.0.

The following changes are meant for upgrading your application to Rails 4.0.

HTTP PATCH

Rails 4 now uses `PATCH` as the primary HTTP verb for updates when a RESTful resource is declared in `config/routes.rb`. The `update` action is still used, and `PUT` requests will continue to be routed to the `update` action as well. So, if you're using only the standard RESTful routes, no changes need to be made:

```
resources :users
```

```
<%= form_for @user do |f| %>
```

```
class UsersController < ApplicationController
  def update
    # No change needed; PATCH will be preferred, and PUT will still work.
  end
end
```

However, you will need to make a change if you are using `form_for` to update a resource in conjunction with a custom route using the `PUT` HTTP method:

```
resources :users do
  put :update_name, on: :member
end
```

```
<%= form_for [ :update_name, @user ] do |f| %>
```

```
class UsersController < ApplicationController
  def update_name
    # Change needed; form_for will try to use a non-existent PATCH route.
  end
end
```

If the action is not being used in a public API and you are free to change the HTTP method, you can update your route to use `patch` instead of `put`:

```
resources :users do
  patch :update_name, on: :member
end
```

`PUT` requests to `/users/:id` in Rails 4 get routed to `update` as they are today. So, if you have an API that gets real PUT requests it is going to work. The router also routes `PATCH` requests to `/users/:id` to the `update` action.

If the action is being used in a public API and you can't change to HTTP method being used, you can update your form to use the `PUT` method instead:

```
<%= form_for [ :update_name, @user ], method: :put do |f| %>
```

For more on PATCH and why this change was made, see this post on the Rails blog.

A note about media types

The errata for the `PATCH` verb specifies that a 'diff' media type should be used with `PATCH`. One such format is JSON Patch. While Rails does not support JSON Patch natively, it's easy enough to add support:

```
# in your controller:
def update
  respond_to do |format|
    format.json do
      # perform a partial update
      @article.update params[:article]
    end

    format.json_patch do
      # perform sophisticated change
    end
  end
end
```

```
# config/initializers/json_patch.rb
Mime::Type.register 'application/json-patch+json', :json_patch
```

As JSON Patch was only recently made into an RFC, there aren't a lot of great Ruby libraries yet. Aaron Patterson's hana is one such gem, but doesn't have full support for the last few changes in the specification.

Gemfile

Rails 4.0 removed the `assets` group from `Gemfile`. You'd need to remove that line from your `Gemfile` when upgrading. You should also update your application file (in `config/application.rb`):

```
# Require the gems listed in Gemfile, including any gems
# you've limited to :test, :development, or :production.
Bundler.require(*Rails.groups)
```

vendor/plugins

Rails 4.0 no longer supports loading plugins from `vendor/plugins`. You must replace any plugins by extracting them to gems and adding them to your `Gemfile`. If you choose not to make them gems, you can move them into, say, `lib/my_plugin/*` and add an appropriate initializer in `config/initializers/my_plugin.rb`.

Active Record

- Rails 4.0 has removed the identity map from Active Record, due to some inconsistencies with associations. If you have manually enabled it in your application, you will have to remove the following config that has no effect anymore: `config.active_record.identity_map`.
- The `delete` method in collection associations can now receive `Integer` or `String` arguments as record ids, besides records, pretty much like the `destroy` method does. Previously it raised `ActiveRecord::AssociationTypeMismatch` for such arguments. From Rails 4.0 on `delete` automatically tries to find the records matching the given ids before deleting them.
- In Rails 4.0 when a column or a table is renamed the related indexes are also renamed. If you have migrations which rename the indexes, they are no longer needed.
- Rails 4.0 has changed `serialized_attributes` and `attr_readonly` to class methods only. You shouldn't use instance methods since it's now deprecated. You should change them to use class methods, e.g. `self.serialized_attributes` to `self.class.serialized_attributes`.
- When using the default coder, assigning `nil` to a serialized attribute will save it to the database as `NULL` instead of passing the `nil` value through YAML (`"--- \n...\n"`).
- Rails 4.0 has removed `attr_accessible` and `attr_protected` feature in favor of Strong Parameters. You can use the Protected Attributes gem for a smooth upgrade path.
- If you are not using Protected Attributes, you can remove any options related to this gem such as `whitelist_attributes` or `mass_assignment_sanitizer` options.
- Rails 4.0 requires that scopes use a callable object such as a Proc or lambda:

```
scope :active, where(active: true)

# becomes
scope :active, -> { where active: true }
```

- Rails 4.0 has deprecated `ActiveRecord::Fixtures` in favor of `ActiveRecord::FixtureSet`.
- Rails 4.0 has deprecated `ActiveRecord::TestCase` in favor of `ActiveSupport::TestCase`.
- Rails 4.0 has deprecated the old-style hash-based finder API. This means that methods which previously accepted "finder options" no longer do. For example, `Book.find(:all, conditions: { name: '1984' })` has been deprecated in favor of `Book.where(name: '1984')`
- All dynamic methods except for `find_by_...` and `find_by_...!` are deprecated. Here's how you can handle the changes:

```
* `find_all_by_...`           becomes `where(...)`.
* `find_last_by_...`          becomes `where(...).last`.
```

```
* `scoped_by_...`             becomes `where(...)`.
* `find_or_initialize_by_...` becomes `find_or_initialize_by(...)`.
* `find_or_create_by_...`     becomes `find_or_create_by(...)`.
```

- Note that `where(...)` returns a relation, not an array like the old finders. If you require an `Array`, use `where(...).to_a`.
- These equivalent methods may not execute the same SQL as the previous implementation.
- To re-enable the old finders, you can use the activerecord-deprecated_finders gem.
- Rails 4.0 has changed to default join table for `has_and_belongs_to_many` relations to strip the common prefix off the second table name. Any existing `has_and_belongs_to_many` relationship between models with a common prefix must be specified with the `join_table` option. For example:

```
CatalogCategory < ActiveRecord::Base
  has_and_belongs_to_many :catalog_products, join_table:
'catalog_categories_catalog_products'
end

CatalogProduct < ActiveRecord::Base
  has_and_belongs_to_many :catalog_categories, join_table:
'catalog_categories_catalog_products'
end
```

- Note that the prefix takes scopes into account as well, so relations between `Catalog::Category` and `Catalog::Product` or `Catalog::Category` and `CatalogProduct` need to be updated similarly.

Active Resource

Rails 4.0 extracted Active Resource to its own gem. If you still need the feature you can add the Active Resource gem in your `Gemfile`.

Active Model

- Rails 4.0 has changed how errors attach with the `ActiveModel::Validations::ConfirmationValidator`. Now when confirmation validations fail, the error will be attached to `:#{attribute}_confirmation` instead of `attribute`.
- Rails 4.0 has changed `ActiveModel::Serializers::JSON.include_root_in_json` default value to `false`. Now, Active Model Serializers and Active Record objects have the same default behavior. This means that you can comment or remove the following option in the `config/initializers/wrap_parameters.rb` file:

```
# Disable root element in JSON by default.
# ActiveSupport.on_load(:active_record) do
#   self.include_root_in_json = false
# end
```

Action Pack

- Rails 4.0 introduces `ActiveSupport::KeyGenerator` and uses this as a base from which to generate and verify signed cookies (among other things). Existing signed cookies generated with Rails 3.x will be transparently upgraded if you leave your existing `secret_token` in place and add the new `secret_key_base`.

  ```
  # config/initializers/secret_token.rb
  Myapp::Application.config.secret_token = 'existing secret token'
  Myapp::Application.config.secret_key_base = 'new secret key base'
  ```

  Please note that you should wait to set `secret_key_base` until you have 100% of your userbase on Rails 4.x and are reasonably sure you will not need to rollback to Rails 3.x. This is because cookies signed based on the new `secret_key_base` in Rails 4.x are not backwards compatible with Rails 3.x. You are free to leave your existing `secret_token` in place, not set the new `secret_key_base`, and ignore the deprecation warnings until you are reasonably sure that your upgrade is otherwise complete.

  If you are relying on the ability for external applications or JavaScript to be able to read your Rails app's signed session cookies (or signed cookies in general) you should not set `secret_key_base` until you have decoupled these concerns.

- Rails 4.0 encrypts the contents of cookie-based sessions if `secret_key_base` has been set. Rails 3.x signed, but did not encrypt, the contents of cookie-based session. Signed cookies are "secure" in that they are verified to have been generated by your app and are tamper-proof. However, the contents can be viewed by end users, and encrypting the contents eliminates this caveat/concern without a significant performance penalty.

  Please read Pull Request #9978 for details on the move to encrypted session cookies.

- Rails 4.0 removed the `ActionController::Base.asset_path` option. Use the assets pipeline feature.

- Rails 4.0 has deprecated `ActionController::Base.page_cache_extension` option. Use `ActionController::Base.default_static_extension` instead.

- Rails 4.0 has removed Action and Page caching from Action Pack. You will need to add the `actionpack-action_caching` gem in order to use `caches_action` and the `actionpack-page_caching` to use `caches_page` in your controllers.

- Rails 4.0 has removed the XML parameters parser. You will need to add the `actionpack-xml_parser` gem if you require this feature.

- Rails 4.0 changes the default `layout` lookup set using symbols or procs that return nil. To get the "no layout" behavior, return false instead of nil.

- Rails 4.0 changes the default memcached client from `memcache-client` to `dalli`. To upgrade, simply add `gem 'dalli'` to your `Gemfile`.

- Rails 4.0 deprecates the `dom_id` and `dom_class` methods in controllers (they are fine in views). You will need to include the `ActionView::RecordIdentifier` module in controllers requiring this feature.

- Rails 4.0 deprecates the `:confirm` option for the `link_to` helper. You should instead rely on a data attribute (e.g. `data: { confirm: 'Are you sure?' }`). This deprecation also concerns the helpers based on this one (such as `link_to_if` or `link_to_unless`).

- Rails 4.0 changed how `assert_generates`, `assert_recognizes`, and `assert_routing` work. Now all these assertions raise `Assertion` instead of `ActionController::RoutingError`.

- Rails 4.0 raises an `ArgumentError` if clashing named routes are defined. This can be triggered by explicitly defined named routes or by the `resources` method. Here are two examples that clash with routes named `example_path`:

```
get 'one' => 'test#example', as: :example
get 'two' => 'test#example', as: :example
```

```
resources :examples
get 'clashing/:id' => 'test#example', as: :example
```

  In the first case, you can simply avoid using the same name for multiple routes. In the second, you can use the `only` or `except` options provided by the `resources` method to restrict the routes created as detailed in the Routing Guide.

- Rails 4.0 also changed the way unicode character routes are drawn. Now you can draw unicode character routes directly. If you already draw such routes, you must change them, for example:

```
get Rack::Utils.escape('こんにちは'), controller: 'welcome', action: 'index'
```

  becomes

```
get 'こんにちは', controller: 'welcome', action: 'index'
```

- Rails 4.0 requires that routes using `match` must specify the request method. For example:

```
  # Rails 3.x
  match '/' => 'root#index'

  # becomes
  match '/' => 'root#index', via: :get

  # or
  get '/' => 'root#index'
```

- Rails 4.0 has removed `ActionDispatch::BestStandardsSupport` middleware, `<!DOCTYPE html>` already triggers standards mode per https://msdn.microsoft.com/en-us/library/jj676915(v=vs.85).aspx and ChromeFrame header has been moved to `config.action_dispatch.default_headers`.

  Remember you must also remove any references to the middleware from your application code, for example:

```
# Raise exception
config.middleware.insert_before(Rack::Lock,
ActionDispatch::BestStandardsSupport)
```

Also check your environment settings for `config.action_dispatch.best_standards_support` and remove it if present.

- Rails 4.0 allows configuration of HTTP headers by setting `config.action_dispatch.default_headers`. The defaults are as follows:

```
config.action_dispatch.default_headers = {
  'X-Frame-Options' => 'SAMEORIGIN',
  'X-XSS-Protection' => '1; mode=block'
}
```

  Please note that if your application is dependent on loading certain pages in a `<frame>` or `<iframe>`, then you may need to explicitly set `X-Frame-Options` to `ALLOW-FROM ...` or `ALLOWALL`.

- In Rails 4.0, precompiling assets no longer automatically copies non-JS/CSS assets from `vendor/assets` and `lib/assets`. Rails application and engine developers should put these assets in `app/assets` or configure `config.assets.precompile`.

- In Rails 4.0, `ActionController::UnknownFormat` is raised when the action doesn't handle the request format. By default, the exception is handled by responding with 406 Not Acceptable, but you can override that now. In Rails 3, 406 Not Acceptable was always returned. No overrides.

- In Rails 4.0, a generic `ActionDispatch::ParamsParser::ParseError` exception is raised when `ParamsParser` fails to parse request params. You will want to rescue this exception instead of the low-level `MultiJson::DecodeError`, for example.

- In Rails 4.0, `SCRIPT_NAME` is properly nested when engines are mounted on an app that's served from a URL prefix. You no longer have to set `default_url_options[:script_name]` to work around overwritten URL prefixes.

- Rails 4.0 deprecated `ActionController::Integration` in favor of `ActionDispatch::Integration`.

- Rails 4.0 deprecated `ActionController::IntegrationTest` in favor of `ActionDispatch::IntegrationTest`.

- Rails 4.0 deprecated `ActionController::PerformanceTest` in favor of `ActionDispatch::PerformanceTest`.

- Rails 4.0 deprecated `ActionController::AbstractRequest` in favor of `ActionDispatch::Request`.

- Rails 4.0 deprecated `ActionController::Request` in favor of `ActionDispatch::Request`.

- Rails 4.0 deprecated `ActionController::AbstractResponse` in favor of `ActionDispatch::Response`.

- Rails 4.0 deprecated `ActionController::Response` in favor of `ActionDispatch::Response`.

- Rails 4.0 deprecated `ActionController::Routing` in favor of `ActionDispatch::Routing`.

Active Support

Rails 4.0 removes the `j` alias for `ERB::Util#json_escape` since `j` is already used for `ActionView::Helpers::JavaScriptHelper#escape_javascript`.

Cache

The caching method changed between Rails 3.x and 4.0. You should change the cache namespace and roll out with a cold cache.

Helpers Loading Order

The order in which helpers from more than one directory are loaded has changed in Rails 4.0. Previously, they were gathered and then sorted alphabetically. After upgrading to Rails 4.0, helpers will preserve the order of loaded directories and will be sorted alphabetically only within each directory. Unless you explicitly use the `helpers_path` parameter, this change will only impact the way of loading helpers from engines. If you rely on the ordering, you should check if correct methods are available after upgrade. If you would like to change the order in which engines are loaded, you can use `config.railties_order=` method.

Active Record Observer and Action Controller Sweeper

`ActiveRecord::Observer` and `ActionController::Caching::Sweeper` have been extracted to the `rails-observers` gem. You will need to add the `rails-observers` gem if you require these features.

sprockets-rails

- `assets:precompile:primary` and `assets:precompile:all` have been removed. Use `assets:precompile` instead.
- The `config.assets.compress` option should be changed to `config.assets.js_compressor` like so for instance:

```
config.assets.js_compressor = :uglifier
```

sass-rails

- `asset-url` with two arguments is deprecated. For example: `asset-url("rails.png", image)` becomes `asset-url("rails.png")`.

Upgrading from Rails 3.1 to Rails 3.2

If your application is currently on any version of Rails older than 3.1.x, you should upgrade to Rails 3.1 before attempting an update to Rails 3.2.

The following changes are meant for upgrading your application to the latest 3.2.x version of Rails.

Gemfile

Make the following changes to your `Gemfile`.

```
gem 'rails', '3.2.21'

group :assets do
  gem 'sass-rails',   '~> 3.2.6'
  gem 'coffee-rails', '~> 3.2.2'
```

```
  gem 'uglifier',     '>= 1.0.3'
end
```

config/environments/development.rb

There are a couple of new configuration settings that you should add to your development environment:

```
# Raise exception on mass assignment protection for Active Record models
config.active_record.mass_assignment_sanitizer = :strict

# Log the query plan for queries taking more than this (works
# with SQLite, MySQL, and PostgreSQL)
config.active_record.auto_explain_threshold_in_seconds = 0.5
```

config/environments/test.rb

The `mass_assignment_sanitizer` configuration setting should also be added to `config/environments/test.rb`:

```
# Raise exception on mass assignment protection for Active Record models
config.active_record.mass_assignment_sanitizer = :strict
```

vendor/plugins

Rails 3.2 deprecates `vendor/plugins` and Rails 4.0 will remove them completely. While it's not strictly necessary as part of a Rails 3.2 upgrade, you can start replacing any plugins by extracting them to gems and adding them to your `Gemfile`. If you choose not to make them gems, you can move them into, say, `lib/my_plugin/*` and add an appropriate initializer in `config/initializers/my_plugin.rb`.

Active Record

Option `:dependent => :restrict` has been removed from `belongs_to`. If you want to prevent deleting the object if there are any associated objects, you can set `:dependent => :destroy` and return `false` after checking for existence of association from any of the associated object's destroy callbacks.

Upgrading from Rails 3.0 to Rails 3.1

If your application is currently on any version of Rails older than 3.0.x, you should upgrade to Rails 3.0 before attempting an update to Rails 3.1.

The following changes are meant for upgrading your application to Rails 3.1.12, the last 3.1.x version of Rails.

Gemfile

Make the following changes to your `Gemfile`.

```
gem 'rails', '3.1.12'
gem 'mysql2'

# Needed for the new asset pipeline
group :assets do
```

```
  gem 'sass-rails',   '~> 3.1.7'
  gem 'coffee-rails', '~> 3.1.1'
  gem 'uglifier',     '>= 1.0.3'
end

# jQuery is the default JavaScript library in Rails 3.1
gem 'jquery-rails'
```

config/application.rb

The asset pipeline requires the following additions:

```
config.assets.enabled = true
config.assets.version = '1.0'
```

If your application is using an "/assets" route for a resource you may want to change the prefix used for assets to avoid conflicts:

```
# Defaults to '/assets'
config.assets.prefix = '/asset-files'
```

config/environments/development.rb

Remove the RJS setting `config.action_view.debug_rjs = true`.

Add these settings if you enable the asset pipeline:

```
# Do not compress assets
config.assets.compress = false

# Expands the lines which load the assets
config.assets.debug = true
```

config/environments/production.rb

Again, most of the changes below are for the asset pipeline. You can read more about these in the Asset Pipeline guide.

```
# Compress JavaScripts and CSS
config.assets.compress = true

# Don't fallback to assets pipeline if a precompiled asset is missed
config.assets.compile = false

# Generate digests for assets URLs
config.assets.digest = true

# Defaults to Rails.root.join("public/assets")
# config.assets.manifest = YOUR_PATH
```

```
# Precompile additional assets (application.js, application.css, and all non-JS/CSS
are already added)
# config.assets.precompile += %w( admin.js admin.css )

# Force all access to the app over SSL, use Strict-Transport-Security, and use
secure cookies.
# config.force_ssl = true
```

config/environments/test.rb

You can help test performance with these additions to your test environment:

```
# Configure static asset server for tests with Cache-Control for performance
config.public_file_server.enabled = true
config.public_file_server.headers = {
  'Cache-Control' => 'public, max-age=3600'
}
```

config/initializers/wrap_parameters.rb

Add this file with the following contents, if you wish to wrap parameters into a nested hash. This is on by default in new applications.

```
# Be sure to restart your server when you modify this file.
# This file contains settings for ActionController::ParamsWrapper which
# is enabled by default.

# Enable parameter wrapping for JSON. You can disable this by setting :format to an
empty array.
ActiveSupport.on_load(:action_controller) do
  wrap_parameters format: [:json]
end

# Disable root element in JSON by default.
ActiveSupport.on_load(:active_record) do
  self.include_root_in_json = false
end
```

config/initializers/session_store.rb

You need to change your session key to something new, or remove all sessions:

```
# in config/initializers/session_store.rb
AppName::Application.config.session_store :cookie_store, key: 'SOMETHINGNEW'
```

or

```
$ bin/rake db:sessions:clear
```

Remove :cache and :concat options in asset helpers references in views

- With the Asset Pipeline the :cache and :concat options aren't used anymore, delete these options from your views.