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

Configuring Rails Applications

This guide covers the configuration and initialization features available to Rails applications.

After reading this guide, you will know:

- How to adjust the behavior of your Rails applications.
- How to add additional code to be run at application start time.

Locations for Initialization Code

Rails offers four standard spots to place initialization code:

- `config/application.rb`
- Environment-specific configuration files
- Initializers
- After-initializers

Running Code Before Rails

In the rare event that your application needs to run some code before Rails itself is loaded, put it above the call to `require "rails/all"` in `config/application.rb`.

Configuring Rails Components

In general, the work of configuring Rails means configuring the components of Rails, as well as configuring Rails itself. The configuration file `config/application.rb` and environment-specific configuration files (such as `config/environments/production.rb`) allow you to specify the various settings that you want to pass down to all of the components.

For example, you could add this setting to `config/application.rb` file:

```
config.time_zone = 'Central Time (US & Canada)'
```

This is a setting for Rails itself. If you want to pass settings to individual Rails components, you can do so via the same `config` object in `config/application.rb`:

```
config.active_record.schema_format = :ruby
```

Rails will use that particular setting to configure Active Record.

WARNING: Use the public configuration methods over calling directly to the associated class. e.g. `Rails.application.config.action_mailer.options` instead of `ActionMailer::Base.options`.

NOTE: If you need to apply configuration directly to a class, use a lazy load hook in an initializer to avoid autoloading the class before initialization has completed. This will break because autoloading during initialization cannot be safely repeated when the app reloads.

Versioned Default Values

`config.load_defaults` loads default configuration values for a target version and all versions prior. For example, `config.load_defaults 6.1` will load defaults for all versions up to and including version 6.1.

Below are the default values associated with each target version. In cases of conflicting values, newer versions take precedence over older versions.

Default Values for Target Version 7.1

- `config.action_dispatch.default_headers: { "X-Frame-Options" => "SAMEORIGIN", "X-XSS-Protection" => "0", "X-Content-Type-Options" => "nosniff", "X-Permitted-Cross-Domain-Policies" => "none", "Referrer-Policy" => "strict-origin-when-cross-origin" }`
- `config.add_autoload_paths_to_load_path: false`
- `config.active_support.default_message_encryptor_serializer: :json`
- `config.active_support.default_message_verifier_serializer: :json`
- `config.action_controller.allow_deprecated_parameters_hash_equality: false`

Default Values for Target Version 7.0

- `config.action_controller.raise_on_open_redirects: true`
- `config.action_view.button_to_generates_button_tag: true`
- `config.action_view.apply_stylesheet_media_default: false`
- `config.active_support.key_generator_hash_digest_class: OpenSSL::Digest::SHA256`
- `config.active_support.hash_digest_class: OpenSSL::Digest::SHA256`
- `config.active_support.cache_format_version: 7.0`
- `config.active_support.remove_deprecated_time_with_zone_name: true`
- `config.active_support.executor_around_test_case: true`
- `config.active_support.use_rfc4122_namespaced_uuids: true`
- `config.active_support.disable_to_s_conversion: true`
- `config.action_dispatch.return_only_request_media_type_on_content_type: false`
- `config.action_dispatch.cookies_serializer: :json`

- config.action_mailer.smtp_timeout: 5
- config.active_storage.video_preview_arguments: "-vf 'select=eq(n\\,0)+eq(key\\,1)+gt(sce
  -frames:v 1 -f image2"
- config.active_storage.multiple_file_field_include_hidden:
  true
- config.active_record.automatic_scope_inversing: true
- config.active_record.verify_foreign_keys_for_fixtures: true
- config.active_record.partial_inserts: false
- config.active_storage.variant_processor: :vips
- config.action_controller.wrap_parameters_by_default: true
- config.action_dispatch.default_headers: { "X-Frame-Options"
  => "SAMEORIGIN", "X-XSS-Protection" => "0", "X-Content-Type-Options"
  => "nosniff", "X-Download-Options" => "noopen", "X-Permitted-Cross-Domain-Policies"
  => "none", "Referrer-Policy" => "strict-origin-when-cross-origin"
  }

Default Values for Target Version 6.1

- config.active_record.has_many_inversing: true
- config.active_record.legacy_connection_handling: false
- config.active_storage.track_variants: true
- config.active_storage.queues.analysis: nil
- config.active_storage.queues.purge: nil
- config.action_mailbox.queues.incineration: nil
- config.action_mailbox.queues.routing: nil
- config.action_mailer.deliver_later_queue_name: nil
- config.active_job.retry_jitter: 0.15
- config.action_dispatch.cookies_same_site_protection: :lax
- config.action_dispatch.ssl_default_redirect_status = 308
- ActiveSupport.utc_to_local_returns_utc_offset_times: true
- config.action_controller.urlsafe_csrf_tokens: true
- config.action_view.form_with_generates_remote_forms: false
- config.action_view.preload_links_header: true

Default Values for Target Version 6.0

- config.action_view.default_enforce_utf8: false
- config.action_dispatch.use_cookies_with_metadata: true
- config.action_mailer.delivery_job: "ActionMailer::MailDeliveryJob"
- config.active_storage.queues.analysis: :active_storage_analysis
- config.active_storage.queues.purge: :active_storage_purge
- config.active_storage.replace_on_assign_to_many: true
- config.active_record.collection_cache_versioning: true

Default Values for Target Version 5.2

- config.active_record.cache_versioning: true

- `config.action_dispatch.use_authenticated_cookie_encryption`: `true`
- `config.active_support.use_authenticated_message_encryption`: `true`
- `config.active_support.hash_digest_class`: `OpenSSL::Digest::SHA1`
- `config.action_controller.default_protect_from_forgery`: `true`
- `config.action_view.form_with_generates_ids`: `true`

Default Values for Target Version 5.1

- `config.assets.unknown_asset_fallback`: `false`
- `config.action_view.form_with_generates_remote_forms`: `true`

Default Values for Target Version 5.0

- `config.action_controller.per_form_csrf_tokens`: `true`
- `config.action_controller.forgery_protection_origin_check`: `true`
- `ActiveSupport.to_time_preserves_timezone`: `true`
- `config.active_record.belongs_to_required_by_default`: `true`
- `config.ssl_options`: `{ hsts: { subdomains: true } }`

Rails General Configuration

The following configuration methods are to be called on a `Rails::Railtie` object, such as a subclass of `Rails::Engine` or `Rails::Application`.

`config.after_initialize` Takes a block which will be run *after* Rails has finished initializing the application. That includes the initialization of the framework itself, engines, and all the application's initializers in `config/initializers`. Note that this block *will* be run for rake tasks. Useful for configuring values set up by other initializers:

```
config.after_initialize do
  ActionView::Base.sanitized_allowed_tags.delete 'div'
end
```

`config.asset_host` Sets the host for the assets. Useful when CDNs are used for hosting assets, or when you want to work around the concurrency constraints built-in in browsers using different domain aliases. Shorter version of `config.action_controller.asset_host`.

`config.autoload_once_paths` Accepts an array of paths from which Rails will autoload constants that won't be wiped per request. Relevant if `config.cache_classes` is `false`, which is the default in the development environment. Otherwise, all autoloading happens only once. All elements of this array must also be in `autoload_paths`. Default is an empty array.

`config.autoload_paths` Accepts an array of paths from which Rails will autoload constants. Default is an empty array. Since Rails 6, it is not recommended to adjust this. See Autoloading and Reloading Constants.

`config.add_autoload_paths_to_load_path` Says whether autoload paths have to be added to `$LOAD_PATH`. It is recommended to be set to `false` in `:zeitwerk` mode early, in `config/application.rb`. Zeitwerk uses absolute paths internally, and applications running in `:zeitwerk` mode do not need `require_dependency`, so models, controllers, jobs, etc. do not need to be in `$LOAD_PATH`. Setting this to `false` saves Ruby from checking these directories when resolving `require` calls with relative paths, and saves Bootsnap work and RAM, since it does not need to build an index for them.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
7.1	`false`

`config.cache_classes` Controls whether or not application classes and modules should be reloaded if they change. When the cache is enabled (`true`), reloading will not occur. Defaults to `false` in the development environment, and `true` in production. In the test environment, the default is `false` if Spring is installed, `true` otherwise.

`config.beginning_of_week` Sets the default beginning of week for the application. Accepts a valid day of week as a symbol (e.g. `:monday`).

`config.cache_store` Configures which cache store to use for Rails caching. Options include one of the symbols `:memory_store`, `:file_store`, `:mem_cache_store`, `:null_store`, `:redis_cache_store`, or an object that implements the cache API. Defaults to `:file_store`. See Cache Stores for per-store configuration options.

`config.colorize_logging` Specifies whether or not to use ANSI color codes when logging information. Defaults to `true`.

`config.consider_all_requests_local` Is a flag. If `true` then any error will cause detailed debugging information to be dumped in the HTTP response, and the `Rails::Info` controller will show the application runtime context in `/rails/info/properties`. `true` by default in the development and test environments, and `false` in production. For finer-grained control, set this to `false` and implement `show_detailed_exceptions?` in controllers to specify which requests should provide debugging information on errors.

config.console Allows you to set the class that will be used as console when you run `bin/rails console`. It's best to run it in the `console` block:

```
console do
  # this block is called only when running console,
  # so we can safely require pry here
  require "pry"
  config.console = Pry
end
```

config.disable_sandbox Controls whether or not someone can start a console in sandbox mode. This is helpful to avoid a long running session of sandbox console, that could lead a database server to run out of memory. Defaults to `false`.

config.eager_load When `true`, eager loads all registered `config.eager_load_namespaces`. This includes your application, engines, Rails frameworks, and any other registered namespace.

config.eager_load_namespaces Registers namespaces that are eager loaded when `config.eager_load` is set to `true`. All namespaces in the list must respond to the `eager_load!` method.

config.eager_load_paths Accepts an array of paths from which Rails will eager load on boot if `config.cache_classes` is set to `true`. Defaults to every folder in the `app` directory of the application.

config.enable_dependency_loading When `true`, enables autoloading, even if the application is eager loaded and `config.cache_classes` is set to `true`. Defaults to `false`.

config.encoding Sets up the application-wide encoding. Defaults to UTF-8.

config.exceptions_app Sets the exceptions application invoked by the `ShowException` middleware when an exception happens. Defaults to `ActionDispatch::PublicExceptions.new(Rails.public_path)`.

config.debug_exception_response_format Sets the format used in responses when errors occur in the development environment. Defaults to `:api` for API only apps and `:default` for normal apps.

config.file_watcher Is the class used to detect file updates in the file system when `config.reload_classes_only_on_change` is `true`. Rails ships with `ActiveSupport::FileUpdateChecker`, the default, and `ActiveSupport::EventedFileUpdateChecker` (this one depends on the listen

gem). Custom classes must conform to the `ActiveSupport::FileUpdateChecker` API.

`config.filter_parameters` Used for filtering out the parameters that you don't want shown in the logs, such as passwords or credit card numbers. It also filters out sensitive values of database columns when calling `#inspect` on an Active Record object. By default, Rails filters out passwords by adding the following filters in `config/initializers/filter_parameter_logging.rb`.

```
Rails.application.config.filter_parameters += [
  :passw, :secret, :token, :_key, :crypt, :salt, :certificate, :otp, :ssn
]
```

Parameters filter works by partial matching regular expression.

`config.force_ssl` Forces all requests to be served over HTTPS, and sets "https://" as the default protocol when generating URLs. Enforcement of HTTPS is handled by the `ActionDispatch::SSL` middleware, which can be configured via `config.ssl_options`.

`config.javascript_path` Sets the path where your app's JavaScript lives relative to the `app` directory. The default is `javascript`, used by webpacker. An app's configured `javascript_path` will be excluded from `autoload_paths`.

`config.log_formatter` Defines the formatter of the Rails logger. This option defaults to an instance of `ActiveSupport::Logger::SimpleFormatter` for all environments. If you are setting a value for `config.logger` you must manually pass the value of your formatter to your logger before it is wrapped in an `ActiveSupport::TaggedLogging` instance, Rails will not do it for you.

`config.log_level` Defines the verbosity of the Rails logger. This option defaults to `:debug` for all environments except production, where it defaults to `:info`. The available log levels are: `:debug`, `:info`, `:warn`, `:error`, `:fatal`, and `:unknown`.

`config.log_tags` Accepts a list of methods that the `request` object responds to, a `Proc` that accepts the `request` object, or something that responds to `to_s`. This makes it easy to tag log lines with debug information like subdomain and request id - both very helpful in debugging multi-user production applications.

`config.logger` Is the logger that will be used for `Rails.logger` and any related Rails logging such as `ActiveRecord::Base.logger`. It defaults to an instance of `ActiveSupport::TaggedLogging` that wraps an instance of `ActiveSupport::Logger` which outputs a log to the `log/` directory. You can supply a custom logger, to get full compatibility you must follow these guidelines:

- To support a formatter, you must manually assign a formatter from the `config.log_formatter` value to the logger.
- To support tagged logs, the log instance must be wrapped with `ActiveSupport::TaggedLogging`.
- To support silencing, the logger must include `ActiveSupport::LoggerSilence` module. The `ActiveSupport::Logger` class already includes these modules.

```
class MyLogger < ::Logger
  include ActiveSupport::LoggerSilence
end

mylogger           = MyLogger.new(STDOUT)
mylogger.formatter = config.log_formatter
config.logger      = ActiveSupport::TaggedLogging.new(mylogger)
```

`config.middleware` Allows you to configure the application's middleware. This is covered in depth in the Configuring Middleware section below.

`config.rake_eager_load` When `true`, eager load the application when running Rake tasks. Defaults to `false`.

`config.reload_classes_only_on_change` Enables or disables reloading of classes only when tracked files change. By default tracks everything on autoload paths and is set to `true`. If `config.cache_classes` is `true`, this option is ignored.

`config.credentials.content_path` Configures lookup path for encrypted credentials.

`config.credentials.key_path` Configures lookup path for encryption key.

`secret_key_base` Is used for specifying a key which allows sessions for the application to be verified against a known secure key to prevent tampering. Applications get a random generated key in test and development environments, other environments should set one in `config/credentials.yml.enc`.

`config.require_master_key` Causes the app to not boot if a master key hasn't been made available through `ENV["RAILS_MASTER_KEY"]` or the `config/master.key` file.

`config.public_file_server.enabled` Configures Rails to serve static files from the public directory. This option defaults to `true`, but in the production environment it is set to `false` because the server software (e.g. NGINX or Apache) used to run the application should serve static files instead. If you are

running or testing your app in production using WEBrick (it is not recommended to use WEBrick in production) set the option to `true`. Otherwise, you won't be able to use page caching and request for files that exist under the public directory.

`config.session_store` Specifies what class to use to store the session. Possible values are `:cache_store`, `:cookie_store`, `:mem_cache_store`, a custom store, or `:disabled`. `:disabled` tells Rails not to deal with sessions.

This setting is configured via a regular method call, rather than a setter. This allows additional options to be passed:

```
config.session_store :cookie_store, key: "_your_app_session"
```

If a custom store is specified as a symbol, it will be resolved to the `ActionDispatch::Session` namespace:

```
# use ActionDispatch::Session::MyCustomStore as the session store
config.session_store :my_custom_store
```

The default store is a cookie store with the application name as the session key.

`config.ssl_options` Configuration options for the `ActionDispatch::SSL` middleware.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`{}`
5.0	`{ hsts: { subdomains: true } }`

`config.time_zone` Sets the default time zone for the application and enables time zone awareness for Active Record.

Configuring Assets

`config.assets.css_compressor` Defines the CSS compressor to use. It is set by default by `sass-rails`. The unique alternative value at the moment is `:yui`, which uses the `yui-compressor` gem.

`config.assets.js_compressor` Defines the JavaScript compressor to use. Possible values are `:terser`, `:closure`, `:uglifier`, and `:yui`, which require the use of the `terser`, `closure-compiler`, `uglifier`, or `yui-compressor` gems respectively.

`config.assets.gzip` A flag that enables the creation of gzipped version of compiled assets, along with non-gzipped assets. Set to `true` by default.

`config.assets.paths` Contains the paths which are used to look for assets. Appending paths to this configuration option will cause those paths to be used in the search for assets.

`config.assets.precompile` Allows you to specify additional assets (other than `application.css` and `application.js`) which are to be precompiled when `rake assets:precompile` is run.

`config.assets.unknown_asset_fallback` Allows you to modify the behavior of the asset pipeline when an asset is not in the pipeline, if you use sprockets-rails 3.2.0 or newer.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
5.1	`false`

`config.assets.prefix` Defines the prefix where assets are served from. Defaults to `/assets`.

`config.assets.manifest` Defines the full path to be used for the asset precompiler's manifest file. Defaults to a file named `manifest-<random>.json` in the `config.assets.prefix` directory within the public folder.

`config.assets.digest` Enables the use of SHA256 fingerprints in asset names. Set to `true` by default.

`config.assets.debug` Disables the concatenation and compression of assets. Set to `true` by default in `development.rb`.

`config.assets.version` Is an option string that is used in SHA256 hash generation. This can be changed to force all files to be recompiled.

`config.assets.compile` Is a boolean that can be used to turn on live Sprockets compilation in production.

`config.assets.logger` Accepts a logger conforming to the interface of Log4r or the default Ruby `Logger` class. Defaults to the same configured at `config.logger`. Setting `config.assets.logger` to `false` will turn off served assets logging.

`config.assets.quiet` Disables logging of assets requests. Set to `true` by default in `development.rb`.

Configuring Generators

Rails allows you to alter what generators are used with the `config.generators` method. This method takes a block:

```
config.generators do |g|
  g.orm :active_record
  g.test_framework :test_unit
end
```

The full set of methods that can be used in this block are as follows:

- `force_plural` allows pluralized model names. Defaults to `false`.
- `helper` defines whether or not to generate helpers. Defaults to `true`.
- `integration_tool` defines which integration tool to use to generate integration tests. Defaults to `:test_unit`.
- `system_tests` defines which integration tool to use to generate system tests. Defaults to `:test_unit`.
- `orm` defines which orm to use. Defaults to `false` and will use Active Record by default.
- `resource_controller` defines which generator to use for generating a controller when using `bin/rails generate resource`. Defaults to `:controller`.
- `resource_route` defines whether a resource route definition should be generated or not. Defaults to `true`.
- `scaffold_controller` different from `resource_controller`, defines which generator to use for generating a *scaffolded* controller when using `bin/rails generate scaffold`. Defaults to `:scaffold_controller`.
- `test_framework` defines which test framework to use. Defaults to `false` and will use minitest by default.
- `template_engine` defines which template engine to use, such as ERB or Haml. Defaults to `:erb`.

Configuring Middleware

Every Rails application comes with a standard set of middleware which it uses in this order in the development environment:

`ActionDispatch::HostAuthorization` Prevents against DNS rebinding and other `Host` header attacks. It is included in the development environment by default with the following configuration:

```
Rails.application.config.hosts = [
  IPAddr.new("0.0.0.0/0"),         # All IPv4 addresses.
  IPAddr.new("::/0"),              # All IPv6 addresses.
  "localhost",                     # The localhost reserved domain.
  ENV["RAILS_DEVELOPMENT_HOSTS"]   # Additional comma-separated hosts for development.
]
```

In other environments `Rails.application.config.hosts` is empty and no `Host` header checks will be done. If you want to guard against header attacks on production, you have to manually permit the allowed hosts with:

```
Rails.application.config.hosts << "product.com"
```

The host of a request is checked against the `hosts` entries with the case operator (`#===`), which lets `hosts` support entries of type `Regexp`, `Proc` and `IPAddr` to name a few. Here is an example with a regexp.

```
# Allow requests from subdomains like `www.product.com` and
# `beta1.product.com`.
Rails.application.config.hosts << /.*\.product\.com/
```

The provided regexp will be wrapped with both anchors (`\A` and `\z`) so it must match the entire hostname. `/product.com/`, for example, once anchored, would fail to match `www.product.com`.

A special case is supported that allows you to permit all sub-domains:

```
# Allow requests from subdomains like `www.product.com` and
# `beta1.product.com`.
Rails.application.config.hosts << ".product.com"
```

You can exclude certain requests from Host Authorization checks by setting `config.host_configuration.exclude`:

```
# Exclude requests for the /healthcheck/ path from host checking
Rails.application.config.host_configuration = {
  exclude: ->(request) { request.path =~ /healthcheck/ }
}
```

When a request comes to an unauthorized host, a default Rack application will run and respond with `403 Forbidden`. This can be customized by setting `config.host_configuration.response_app`. For example:

```
Rails.application.config.host_configuration = {
  response_app: -> env do
    [400, { "Content-Type" => "text/plain" }, ["Bad Request"]]
  end
}
```

`ActionDispatch::SSL` Forces every request to be served using HTTPS. Enabled if `config.force_ssl` is set to `true`. Options passed to this can be configured by setting `config.ssl_options`.

`ActionDispatch::Static` Is used to serve static assets. Disabled if `config.public_file_server.enabled` is `false`. Set `config.public_file_server.index_name` if you need to serve a static directory index file that is not named `index`. For

example, to serve `main.html` instead of `index.html` for directory requests, set `config.public_file_server.index_name` to `"main"`.

`ActionDispatch::Executor` Allows thread safe code reloading. Disabled if `config.allow_concurrency` is `false`, which causes `Rack::Lock` to be loaded. `Rack::Lock` wraps the app in mutex so it can only be called by a single thread at a time.

`ActiveSupport::Cache::Strategy::LocalCache` Serves as a basic memory backed cache. This cache is not thread safe and is intended only for serving as a temporary memory cache for a single thread.

`Rack::Runtime` Sets an `X-Runtime` header, containing the time (in seconds) taken to execute the request.

`Rails::Rack::Logger` Notifies the logs that the request has begun. After request is complete, flushes all the logs.

`ActionDispatch::ShowExceptions` Rescues any exception returned by the application and renders nice exception pages if the request is local or if `config.consider_all_requests_local` is set to `true`. If `config.action_dispatch.show_exceptions` is set to `false`, exceptions will be raised regardless.

`ActionDispatch::RequestId` Makes a unique X-Request-Id header available to the response and enables the `ActionDispatch::Request#uuid` method. Configurable with `config.action_dispatch.request_id_header`.

`ActionDispatch::RemoteIp` Checks for IP spoofing attacks and gets valid `client_ip` from request headers. Configurable with the `config.action_dispatch.ip_spoofing_check`, and `config.action_dispatch.trusted_proxies` options.

`Rack::Sendfile` Intercepts responses whose body is being served from a file and replaces it with a server specific X-Sendfile header. Configurable with `config.action_dispatch.x_sendfile_header`.

`ActionDispatch::Callbacks` Runs the prepare callbacks before serving the request.

`ActionDispatch::Cookies` Sets cookies for the request.

`ActionDispatch::Session::CookieStore` Is responsible for storing the session in cookies. An alternate middleware can be used for this by changing `config.session_store`.

`ActionDispatch::Flash` Sets up the `flash` keys. Only available if `config.session_store` is set to a value.

`Rack::MethodOverride` Allows the method to be overridden if `params[:_method]` is set. This is the middleware which supports the PATCH, PUT, and DELETE HTTP method types.

`Rack::Head` Converts HEAD requests to GET requests and serves them as so.

Adding Custom Middleware Besides these usual middleware, you can add your own by using the `config.middleware.use` method:

```
config.middleware.use Magical::Unicorns
```

This will put the `Magical::Unicorns` middleware on the end of the stack. You can use `insert_before` if you wish to add a middleware before another.

```
config.middleware.insert_before Rack::Head, Magical::Unicorns
```

Or you can insert a middleware to exact position by using indexes. For example, if you want to insert `Magical::Unicorns` middleware on top of the stack, you can do it, like so:

```
config.middleware.insert_before 0, Magical::Unicorns
```

There's also `insert_after` which will insert a middleware after another:

```
config.middleware.insert_after Rack::Head, Magical::Unicorns
```

Middlewares can also be completely swapped out and replaced with others:

```
config.middleware.swap ActionController::Failsafe, Lifo::Failsafe
```

Middlewares can be moved from one place to another:

```
config.middleware.move_before ActionDispatch::Flash, Magical::Unicorns
```

This will move the `Magical::Unicorns` middleware before `ActionDispatch::Flash`. You can also move it after:

```
config.middleware.move_after ActionDispatch::Flash, Magical::Unicorns
```

They can also be removed from the stack completely:

```
config.middleware.delete Rack::MethodOverride
```

Configuring i18n

All these configuration options are delegated to the `I18n` library.

`config.i18n.available_locales` Defines the permitted available locales for the app. Defaults to all locale keys found in locale files, usually only `:en` on a new application.

`config.i18n.default_locale` Sets the default locale of an application used for i18n. Defaults to `:en`.

`config.i18n.enforce_available_locales` Ensures that all locales passed through i18n must be declared in the `available_locales` list, raising an `I18n::InvalidLocale` exception when setting an unavailable locale. Defaults to `true`. It is recommended not to disable this option unless strongly required, since this works as a security measure against setting any invalid locale from user input.

`config.i18n.load_path` Sets the path Rails uses to look for locale files. Defaults to `config/locales/**/*.{yml,rb}`.

`config.i18n.raise_on_missing_translations` Determines whether an error should be raised for missing translations in controllers and views. This defaults to `false`.

`config.i18n.fallbacks` Sets fallback behavior for missing translations. Here are 3 usage examples for this option:

- You can set the option to `true` for using default locale as fallback, like so:

  ```
  config.i18n.fallbacks = true
  ```

- Or you can set an array of locales as fallback, like so:

  ```
  config.i18n.fallbacks = [:tr, :en]
  ```

- Or you can set different fallbacks for locales individually. For example, if you want to use `:tr` for `:az` and `:de`, `:en` for `:da` as fallbacks, you can do it, like so:

  ```
  config.i18n.fallbacks = { az: :tr, da: [:de, :en] }
  #or
  config.i18n.fallbacks.map = { az: :tr, da: [:de, :en] }
  ```

Configuring Active Model

`config.active_model.i18n_customize_full_message` Is a boolean value which controls whether the `full_message` error format can be overridden at the attribute or model level in the locale files. This is `false` by default.

Configuring Active Record

`config.active_record` includes a variety of configuration options:

`config.active_record.logger` Accepts a logger conforming to the interface of Log4r or the default Ruby Logger class, which is then passed on to any new database connections made. You can retrieve this logger by calling `logger` on either an Active Record model class or an Active Record model instance. Set to `nil` to disable logging.

`config.active_record.primary_key_prefix_type` Lets you adjust the naming for primary key columns. By default, Rails assumes that primary key columns are named `id` (and this configuration option doesn't need to be set). There are two other choices:

- `:table_name` would make the primary key for the Customer class `customerid`.
- `:table_name_with_underscore` would make the primary key for the Customer class `customer_id`.

`config.active_record.table_name_prefix` Lets you set a global string to be prepended to table names. If you set this to `northwest_`, then the Customer class will look for `northwest_customers` as its table. The default is an empty string.

`config.active_record.table_name_suffix` Lets you set a global string to be appended to table names. If you set this to `_northwest`, then the Customer class will look for `customers_northwest` as its table. The default is an empty string.

`config.active_record.schema_migrations_table_name` Lets you set a string to be used as the name of the schema migrations table.

`config.active_record.internal_metadata_table_name` Lets you set a string to be used as the name of the internal metadata table.

`config.active_record.protected_environments` Lets you set an array of names of environments where destructive actions should be prohibited.

`config.active_record.pluralize_table_names` Specifies whether Rails will look for singular or plural table names in the database. If set to `true` (the default), then the Customer class will use the `customers` table. If set to `false`, then the Customer class will use the `customer` table.

`config.active_record.default_timezone` Determines whether to use `Time.local` (if set to `:local`) or `Time.utc` (if set to `:utc`) when pulling dates and times from the database. The default is `:utc`.

`config.active_record.schema_format` Controls the format for dumping the database schema to a file. The options are `:ruby` (the default) for a database-independent version that depends on migrations, or `:sql` for a set of (potentially database-dependent) SQL statements.

`config.active_record.error_on_ignored_order` Specifies if an error should be raised if the order of a query is ignored during a batch query. The options are `true` (raise error) or `false` (warn). Default is `false`.

`config.active_record.timestamped_migrations` Controls whether migrations are numbered with serial integers or with timestamps. The default is `true`, to use timestamps, which are preferred if there are multiple developers working on the same application.

`config.active_record.lock_optimistically` Controls whether Active Record will use optimistic locking and is `true` by default.

`config.active_record.cache_timestamp_format` Controls the format of the timestamp value in the cache key. Default is `:usec`.

`config.active_record.record_timestamps` Is a boolean value which controls whether or not timestamping of `create` and `update` operations on a model occur. The default value is `true`.

`config.active_record.partial_inserts` Is a boolean value and controls whether or not partial writes are used when creating new records (i.e. whether inserts only set attributes that are different from the default).

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
7.0	`false`

`config.active_record.partial_updates` Is a boolean value and controls whether or not partial writes are used when updating existing records (i.e. whether updates only set attributes that are dirty). Note that when using partial updates, you should also use optimistic locking `config.active_record.lock_optimistically` since concurrent updates may write attributes based on a possibly stale read state. The default value is `true`.

`config.active_record.maintain_test_schema` Is a boolean value which controls whether Active Record should try to keep your test database schema up-to-date with `db/schema.rb` (or `db/structure.sql`) when you run your tests. The default is `true`.

`config.active_record.dump_schema_after_migration` Is a flag which controls whether or not schema dump should happen (`db/schema.rb` or `db/structure.sql`) when you run migrations. This is set to `false` in `config/environments/production.rb` which is generated by Rails. The default value is `true` if this configuration is not set.

`config.active_record.dump_schemas` Controls which database schemas will be dumped when calling `db:schema:dump`. The options are `:schema_search_path` (the default) which dumps any schemas listed in `schema_search_path`, `:all` which always dumps all schemas regardless of the `schema_search_path`, or a string of comma separated schemas.

`config.active_record.belongs_to_required_by_default` Is a boolean value and controls whether a record fails validation if `belongs_to` association is not present.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`nil`
5.0	`true`

`config.active_record.action_on_strict_loading_violation` Enables raising or logging an exception if strict_loading is set on an association. The default value is `:raise` in all environments. It can be changed to `:log` to send violations to the logger instead of raising.

`config.active_record.strict_loading_by_default` Is a boolean value that either enables or disables strict_loading mode by default. Defaults to `false`.

`config.active_record.warn_on_records_fetched_greater_than` Allows setting a warning threshold for query result size. If the number of records returned by a query exceeds the threshold, a warning is logged. This can be used to identify queries which might be causing a memory bloat.

`config.active_record.index_nested_attribute_errors` Allows errors for nested `has_many` relationships to be displayed with an index as well as the error. Defaults to `false`.

`config.active_record.use_schema_cache_dump` Enables users to get schema cache information from `db/schema_cache.yml` (generated by `bin/rails db:schema:cache:dump`), instead of having to send a query to the database to get this information. Defaults to `true`.

`config.active_record.cache_versioning` Indicates whether to use a stable `#cache_key` method that is accompanied by a changing version in the `#cache_version` method.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.2	`true`

`config.active_record.collection_cache_versioning` Enables the same cache key to be reused when the object being cached of type `ActiveRecord::Relation` changes by moving the volatile information (max updated at and count) of the relation's cache key into the cache version to support recycling cache key.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
6.0	`true`

`config.active_record.has_many_inversing` Enables setting the inverse record when traversing `belongs_to` to `has_many` associations.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
6.1	`true`

`config.active_record.automatic_scope_inversing` Enables automatically inferring the `inverse_of` for associations with a scope.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.active_record.legacy_connection_handling` Allows to enable new connection handling API. For applications using multiple databases, this new API provides support for granular connection swapping.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
6.1	`false`

`config.active_record.destroy_association_async_job` Allows specifying the job that will be used to destroy the associated records in background. It defaults to `ActiveRecord::DestroyAssociationAsyncJob`.

`config.active_record.destroy_association_async_batch_size` Allows specifying the maximum number of records that will be destroyed in a background job by the `dependent: :destroy_async` association option. All else equal, a lower batch size will enqueue more, shorter-running background jobs, while a higher batch size will enqueue fewer, longer-running background jobs. This option defaults to `nil`, which will cause all dependent records for a given association to be destroyed in the same background job.

`config.active_record.queues.destroy` Allows specifying the Active Job queue to use for destroy jobs. When this option is `nil`, purge jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`). It defaults to `nil`.

`config.active_record.enumerate_columns_in_select_statements`
When `true`, will always include column names in `SELECT` statements, and avoid wildcard `SELECT * FROM ...` queries. This avoids prepared statement cache errors when adding columns to a PostgreSQL database for example. Defaults to `false`.

`config.active_record.verify_foreign_keys_for_fixtures` Ensures all foreign key constraints are valid after fixtures are loaded in tests. Supported by PostgreSQL and SQLite only.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.active_record.query_log_tags_enabled` Specifies whether or not to enable adapter-level query comments. Defaults to `false`.

`config.active_record.query_log_tags` Define an `Array` specifying the key/value tags to be inserted in an SQL comment. Defaults to `[ :application ]`, a predefined tag returning the application name.

`config.active_record.cache_query_log_tags` Specifies whether or not to enable caching of query log tags. For applications that have a large number of queries, caching query log tags can provide a performance benefit when the context does not change during the lifetime of the request or job execution. Defaults to `false`.

`config.active_record.schema_cache_ignored_tables` Define the list of table that should be ignored when generating the schema cache. It accepts an `Array` of strings, representing the table names, or regular expressions.

`config.active_record.verbose_query_logs` Specifies if source locations of methods that call database queries should be logged below relevant queries. By default, the flag is `true` in development and `false` in all other environments.

`config.active_record.async_query_executor` Specifies how asynchronous queries are pooled.

It defaults to `nil`, which means `load_async` is disabled and instead directly executes queries in the foreground. For queries to actually be performed asynchronously, it must be set to either `:global_thread_pool` or `:multi_thread_pool`.

`:global_thread_pool` will use a single pool for all databases the application connects to. This is the preferred configuration for applications with only a single database, or applications which only ever query one database shard at a time.

`:multi_thread_pool` will use one pool per database, and each pool size can be configured individually in `database.yml` through the `max_threads` and `min_thread` properties. This can be useful to applications regularly querying multiple databases at a time, and that need to more precisely define the max concurrency.

`config.active_record.global_executor_concurrency` Used in conjunction with `config.active_record.async_query_executor = :global_thread_pool`, defines how many asynchronous queries can be executed concurrently.

Defaults to `4`.

This number must be considered in accordance with the database pool size configured in `database.yml`. The connection pool should be large enough to accommodate both the foreground threads (.e.g web server or job worker threads) and background threads.

`ActiveRecord::ConnectionAdapters::Mysql2Adapter.emulate_booleans` Controls whether the Active Record MySQL adapter will consider all `tinyint(1)` columns as booleans. Defaults to `true`.

`ActiveRecord::ConnectionAdapters::PostgreSQLAdapter.create_unlogged_tables` Controls whether database tables created by PostgreSQL should be "unlogged", which can speed up performance but adds a risk of data loss if the database crashes. It is highly recommended that you do not enable this in a production environment. Defaults to `false` in all environments.

`ActiveRecord::ConnectionAdapters::PostgreSQLAdapter.datetime_type` Controls what native type the Active Record PostgreSQL adapter should use when you call `datetime` in a migration or schema. It takes a symbol which must correspond to one of the configured `NATIVE_DATABASE_TYPES`. The default is `:timestamp`, meaning `t.datetime` in a migration will create a "timestamp without time zone" column. To use "timestamp with time zone", change this to `:timestamptz` in an initializer. You should run `bin/rails db:migrate` to rebuild your schema.rb if you change this.

`ActiveRecord::SchemaDumper.ignore_tables` Accepts an array of tables that should *not* be included in any generated schema file.

`ActiveRecord::SchemaDumper.fk_ignore_pattern` Allows setting a different regular expression that will be used to decide whether a foreign key's name should be dumped to db/schema.rb or not. By default, foreign key names starting with `fk_rails_` are not exported to the database schema dump. Defaults to `/^fk_rails_[0-9a-f]{10}$/`.

Configuring Action Controller

`config.action_controller` includes a number of configuration settings:

`config.action_controller.asset_host` Sets the host for the assets. Useful when CDNs are used for hosting assets rather than the application server itself.

You should only use this if you have a different configuration for Action Mailer, otherwise use `config.asset_host`.

`config.action_controller.perform_caching` Configures whether the application should perform the caching features provided by the Action Controller component or not. Set to `false` in the development environment, `true` in production. If it's not specified, the default will be `true`.

`config.action_controller.default_static_extension` Configures the extension used for cached pages. Defaults to `.html`.

`config.action_controller.include_all_helpers` Configures whether all view helpers are available everywhere or are scoped to the corresponding controller. If set to `false`, `UsersHelper` methods are only available for views rendered as part of `UsersController`. If `true`, `UsersHelper` methods are available everywhere. The default configuration behavior (when this option is not explicitly set to `true` or `false`) is that all view helpers are available to each controller.

`config.action_controller.logger` Accepts a logger conforming to the interface of Log4r or the default Ruby Logger class, which is then used to log information from Action Controller. Set to `nil` to disable logging.

`config.action_controller.request_forgery_protection_token` Sets the token parameter name for RequestForgery. Calling `protect_from_forgery` sets it to `:authenticity_token` by default.

`config.action_controller.allow_forgery_protection` Enables or disables CSRF protection. By default this is `false` in the test environment and `true` in all other environments.

`config.action_controller.forgery_protection_origin_check` Configures whether the HTTP `Origin` header should be checked against the site's origin as an additional CSRF defense.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.0	`true`

`config.action_controller.per_form_csrf_tokens` Configures whether CSRF tokens are only valid for the method/action they were generated for.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.0	`true`

`config.action_controller.default_protect_from_forgery` Determines whether forgery protection is added on `ActionController::Base`.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.2	`true`

`config.action_controller.urlsafe_csrf_tokens` Configures whether generated CSRF tokens are URL-safe.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
6.1	`true`

`config.action_controller.relative_url_root` Can be used to tell Rails that you are deploying to a subdirectory. The default is `ENV['RAILS_RELATIVE_URL_ROOT']`.

`config.action_controller.permit_all_parameters` Sets all the parameters for mass assignment to be permitted by default. The default value is `false`.

`config.action_controller.action_on_unpermitted_parameters` Controls behavior when parameters that are not explicitly permitted are found. The default value is `:log` in test and development environments, `false` otherwise. The values can be:

- `false` to take no action
- `:log` to emit an `ActiveSupport::Notifications.instrument` event on the `unpermitted_parameters.action_controller` topic and log at the DEBUG level

- `:raise` to raise a `ActionController::UnpermittedParameters` exception

`config.action_controller.always_permitted_parameters` Sets a list of permitted parameters that are permitted by default. The default values are `['controller', 'action']`.

`config.action_controller.enable_fragment_cache_logging` Determines whether to log fragment cache reads and writes in verbose format as follows:

```
Read fragment views/v1/2914079/v1/2914079/recordings/70182313-20160225015037000000/d0bdf2974
Rendered messages/_message.html.erb in 1.2 ms [cache hit]
Write fragment views/v1/2914079/v1/2914079/recordings/70182313-20160225015037000000/3b4e249a
Rendered recordings/threads/_thread.html.erb in 1.5 ms [cache miss]
```

By default it is set to `false` which results in following output:

```
Rendered messages/_message.html.erb in 1.2 ms [cache hit]
Rendered recordings/threads/_thread.html.erb in 1.5 ms [cache miss]
```

`config.action_controller.raise_on_open_redirects` Raises an `ArgumentError` when an unpermitted open redirect occurs.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.action_controller.log_query_tags_around_actions` Determines whether controller context for query tags will be automatically updated via an `around_filter`. The default value is `true`.

`config.action_controller.wrap_parameters_by_default` Configures the `ParamsWrapper` to wrap json request by default.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`ActionController::Base.wrap_parameters` Configures the `ParamsWrapper`. This can be called at the top level, or on individual controllers.

`config.action_controller.allow_deprecated_parameters_hash_equality` Controls behaviour of `ActionController::Parameters#==` with `Hash` arguments. Value of the setting determines whether an `ActionController::Parameters` instance is equal to an equivalent `Hash`.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
7.1	`false`

Configuring Action Dispatch

`config.action_dispatch.cookies_serializer` Specifies which serializer to use for cookies. For more information, see Action Controller Cookies.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:marshal`
7.0	`:json`

`config.action_dispatch.default_headers` Is a hash with HTTP headers that are set by default in each response.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	
7.0	
7.1	

`config.action_dispatch.default_charset` Specifies the default character set for all renders. Defaults to `nil`.

`config.action_dispatch.tld_length` Sets the TLD (top-level domain) length for the application. Defaults to `1`.

`config.action_dispatch.ignore_accept_header` Is used to determine whether to ignore accept headers from a request. Defaults to `false`.

`config.action_dispatch.x_sendfile_header` Specifies server specific X-Sendfile header. This is useful for accelerated file sending from server. For example it can be set to 'X-Sendfile' for Apache.

`config.action_dispatch.http_auth_salt` Sets the HTTP Auth salt value. Defaults to `'http authentication'`.

`config.action_dispatch.signed_cookie_salt` Sets the signed cookies salt value. Defaults to `'signed cookie'`.

`config.action_dispatch.encrypted_cookie_salt` Sets the encrypted cookies salt value. Defaults to `'encrypted cookie'`.

`config.action_dispatch.encrypted_signed_cookie_salt` Sets the signed encrypted cookies salt value. Defaults to `'signed encrypted cookie'`.

`config.action_dispatch.authenticated_encrypted_cookie_salt` Sets the authenticated encrypted cookie salt. Defaults to `'authenticated encrypted cookie'`.

`config.action_dispatch.encrypted_cookie_cipher` Sets the cipher to be used for encrypted cookies. This defaults to `"aes-256-gcm"`.

`config.action_dispatch.signed_cookie_digest` Sets the digest to be used for signed cookies. This defaults to `"SHA1"`.

`config.action_dispatch.cookies_rotations` Allows rotating secrets, ciphers, and digests for encrypted and signed cookies.

`config.action_dispatch.use_authenticated_cookie_encryption` Controls whether signed and encrypted cookies use the AES-256-GCM cipher or the older AES-256-CBC cipher.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.2	`true`

`config.action_dispatch.use_cookies_with_metadata` Enables writing cookies with the purpose metadata embedded.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`

Starting with version	The default value is
6.0	`true`

`config.action_dispatch.perform_deep_munge` Configures whether `deep_munge` method should be performed on the parameters. See Security Guide for more information. It defaults to `true`.

`config.action_dispatch.rescue_responses` Configures what exceptions are assigned to an HTTP status. It accepts a hash and you can specify pairs of exception/status. By default, this is defined as:

```
config.action_dispatch.rescue_responses = {
  'ActionController::RoutingError'
    => :not_found,
  'AbstractController::ActionNotFound'
    => :not_found,
  'ActionController::MethodNotAllowed'
    => :method_not_allowed,
  'ActionController::UnknownHttpMethod'
    => :method_not_allowed,
  'ActionController::NotImplemented'
    => :not_implemented,
  'ActionController::UnknownFormat'
    => :not_acceptable,
  'ActionController::InvalidAuthenticityToken'
    => :unprocessable_entity,
  'ActionController::InvalidCrossOriginRequest'
    => :unprocessable_entity,
  'ActionDispatch::Http::Parameters::ParseError'
    => :bad_request,
  'ActionController::BadRequest'
    => :bad_request,
  'ActionController::ParameterMissing'
    => :bad_request,
  'Rack::QueryParser::ParameterTypeError'
    => :bad_request,
  'Rack::QueryParser::InvalidParameterError'
    => :bad_request,
  'ActiveRecord::RecordNotFound'
    => :not_found,
  'ActiveRecord::StaleObjectError'
    => :conflict,
  'ActiveRecord::RecordInvalid'
    => :unprocessable_entity,
  'ActiveRecord::RecordNotSaved'
```

```
    => :unprocessable_entity
}
```

Any exceptions that are not configured will be mapped to 500 Internal Server Error.

`config.action_dispatch.return_only_request_media_type_on_content_type` Change the return value of `ActionDispatch::Request#content_type` to the Content-Type header without modification.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
7.0	`false`

`config.action_dispatch.cookies_same_site_protection` Configures the default value of the `SameSite` attribute when setting cookies. When set to `nil`, the `SameSite` attribute is not added. To allow the value of the `SameSite` attribute to be configured dynamically based on the request, a proc may be specified. For example:

```
config.action_dispatch.cookies_same_site_protection = ->(request) do
  :strict unless request.user_agent == "TestAgent"
end
```

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`nil`
6.1	`:lax`

`config.action_dispatch.ssl_default_redirect_status` Configures the default HTTP status code used when redirecting non-GET/HEAD requests from HTTP to HTTPS in the `ActionDispatch::SSL` middleware.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	307
6.1	308

`config.action_dispatch.log_rescued_responses` Enables logging those unhandled exceptions configured in `rescue_responses`. It defaults to `true`.

`ActionDispatch::Callbacks.before` Takes a block of code to run before the request.

`ActionDispatch::Callbacks.after` Takes a block of code to run after the request.

Configuring Action View

`config.action_view` includes a small number of configuration settings:

`config.action_view.cache_template_loading` Controls whether or not templates should be reloaded on each request. Defaults to whatever is set for `config.cache_classes`.

`config.action_view.field_error_proc` Provides an HTML generator for displaying errors that come from Active Model. The block is evaluated within the context of an Action View template. The default is

```
Proc.new { |html_tag, instance| content_tag :div, html_tag, class: "field_with_errors" }
```

`config.action_view.default_form_builder` Tells Rails which form builder to use by default. The default is `ActionView::Helpers::FormBuilder`. If you want your form builder class to be loaded after initialization (so it's reloaded on each request in development), you can pass it as a `String`.

`config.action_view.logger` Accepts a logger conforming to the interface of Log4r or the default Ruby Logger class, which is then used to log information from Action View. Set to `nil` to disable logging.

`config.action_view.erb_trim_mode` Gives the trim mode to be used by ERB. It defaults to `'-'`, which turns on trimming of tail spaces and newline when using `<%= -%>` or `<%= =%>`. See the Erubis documentation for more information.

`config.action_view.frozen_string_literal` Compiles the ERB template with the `# frozen_string_literal: true` magic comment, making all string literals frozen and saving allocations. Set to `true` to enable it for all views.

`config.action_view.embed_authenticity_token_in_remote_forms` Allows you to set the default behavior for `authenticity_token` in forms with `remote: true`. By default it's set to `false`, which means that remote forms will not include `authenticity_token`, which is helpful when you're fragment-caching the form. Remote forms get the authenticity from the `meta` tag, so embedding is unnecessary unless you support browsers without JavaScript. In such case you can either pass `authenticity_token: true` as a form option or set this config setting to `true`.

`config.action_view.prefix_partial_path_with_controller_namespace` Determines whether or not partials are looked up from a subdirectory in templates rendered from namespaced controllers. For example, consider a controller named `Admin::ArticlesController` which renders this template:

```
<%= render @article %>
```

The default setting is `true`, which uses the partial at `/admin/articles/_article.erb`. Setting the value to `false` would render `/articles/_article.erb`, which is the same behavior as rendering from a non-namespaced controller such as `ArticlesController`.

`config.action_view.automatically_disable_submit_tag` Determines whether `submit_tag` should automatically disable on click, this defaults to `true`.

`config.action_view.debug_missing_translation` Determines whether to wrap the missing translations key in a `<span>` tag or not. This defaults to `true`.

`config.action_view.form_with_generates_remote_forms` Determines whether `form_with` generates remote forms or not.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
5.1	`true`
6.1	`false`

`config.action_view.form_with_generates_ids` Determines whether `form_with` generates ids on inputs.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.2	`true`

`config.action_view.default_enforce_utf8` Determines whether forms are generated with a hidden tag that forces older versions of Internet Explorer to submit forms encoded in UTF-8.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
6.0	`false`

`config.action_view.image_loading` Specifies a default value for the `loading` attribute of `<img>` tags rendered by the `image_tag` helper. For example, when set to `"lazy"`, `<img>` tags rendered by `image_tag` will include `loading="lazy"`, which instructs the browser to wait until an image is near the viewport to load it. (This value can still be overridden per image by passing e.g. `loading: "eager"` to `image_tag`.) Defaults to `nil`.

`config.action_view.image_decoding` Specifies a default value for the `decoding` attribute of `<img>` tags rendered by the `image_tag` helper. Defaults to `nil`.

`config.action_view.annotate_rendered_view_with_filenames` Determines whether to annotate rendered view with template file names. This defaults to `false`.

`config.action_view.preload_links_header` Determines whether `javascript_include_tag` and `stylesheet_link_tag` will generate a `Link` header that preload assets.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`nil`
6.1	`true`

`config.action_view.button_to_generates_button_tag` Determines whether `button_to` will render `<button>` element, regardless of whether or not the content is passed as the first argument or as a block.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.action_view.apply_stylesheet_media_default` Determines whether `stylesheet_link_tag` will render `screen` as the default value for the attribute `media` when it's not provided.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`true`
7.0	`false`

Configuring Action Mailbox

`config.action_mailbox` provides the following configuration options:

`config.action_mailbox.logger` Contains the logger used by Action Mailbox. It accepts a logger conforming to the interface of Log4r or the default Ruby Logger class. The default is `Rails.logger`.

```
config.action_mailbox.logger = ActiveSupport::Logger.new(STDOUT)
```

`config.action_mailbox.incinerate_after` Accepts an `ActiveSupport::Duration` indicating how long after processing `ActionMailbox::InboundEmail` records should be destroyed. It defaults to `30.days`.

```
# Incinerate inbound emails 14 days after processing.
config.action_mailbox.incinerate_after = 14.days
```

`config.action_mailbox.queues.incineration` Accepts a symbol indicating the Active Job queue to use for incineration jobs. When this option is `nil`, incineration jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`).

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:action_mailbox_incineration`
6.1	`nil`

`config.action_mailbox.queues.routing` Accepts a symbol indicating the Active Job queue to use for routing jobs. When this option is `nil`, routing jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`).

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:action_mailbox_routing`

Starting with version	The default value is
6.1	`nil`

`config.action_mailbox.storage_service` Accepts a symbol indicating the Active Storage service to use for uploading emails. When this option is `nil`, emails are uploaded to the default Active Storage service (see `config.active_storage.service`).

Configuring Action Mailer

There are a number of settings available on `config.action_mailer`:

`config.action_mailer.asset_host` Sets the host for the assets. Useful when CDNs are used for hosting assets rather than the application server itself. You should only use this if you have a different configuration for Action Controller, otherwise use `config.asset_host`.

`config.action_mailer.logger` Accepts a logger conforming to the interface of Log4r or the default Ruby Logger class, which is then used to log information from Action Mailer. Set to `nil` to disable logging.

`config.action_mailer.smtp_settings` Allows detailed configuration for the `:smtp` delivery method. It accepts a hash of options, which can include any of these options:

- `:address` - Allows you to use a remote mail server. Just change it from its default "localhost" setting.
- `:port` - On the off chance that your mail server doesn't run on port 25, you can change it.
- `:domain` - If you need to specify a HELO domain, you can do it here.
- `:user_name` - If your mail server requires authentication, set the username in this setting.
- `:password` - If your mail server requires authentication, set the password in this setting.
- `:authentication` - If your mail server requires authentication, you need to specify the authentication type here. This is a symbol and one of `:plain`, `:login`, `:cram_md5`.
- `:enable_starttls` - Use STARTTLS when connecting to your SMTP server and fail if unsupported. It defaults to `false`.
- `:enable_starttls_auto` - Detects if STARTTLS is enabled in your SMTP server and starts to use it. It defaults to `true`.
- `:openssl_verify_mode` - When using TLS, you can set how OpenSSL checks the certificate. This is useful if you need to validate a self-signed and/or a wildcard certificate. This can be one of the

OpenSSL verify constants, `:none` or `:peer` – or the constant directly `OpenSSL::SSL::VERIFY_NONE` or `OpenSSL::SSL::VERIFY_PEER`, respectively.
- `:ssl/:tls` - Enables the SMTP connection to use SMTP/TLS (SMTPS: SMTP over direct TLS connection).
- `:open_timeout` - Number of seconds to wait while attempting to open a connection.
- `:read_timeout` - Number of seconds to wait until timing-out a read(2) call.

Additionally, it is possible to pass any configuration option `Mail::SMTP` respects.

`config.action_mailer.smtp_timeout` Allows to configure both the `:open_timeout` and `:read_timeout` values for `:smtp` delivery method.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`nil`
7.0	`5`

`config.action_mailer.sendmail_settings` Allows detailed configuration for the `sendmail` delivery method. It accepts a hash of options, which can include any of these options:

- `:location` - The location of the sendmail executable. Defaults to `/usr/sbin/sendmail`.
- `:arguments` - The command line arguments. Defaults to `-i`.

`config.action_mailer.raise_delivery_errors` Specifies whether to raise an error if email delivery cannot be completed. It defaults to `true`.

`config.action_mailer.delivery_method` Defines the delivery method and defaults to `:smtp`. See the configuration section in the Action Mailer guide for more info.

`config.action_mailer.perform_deliveries` Specifies whether mail will actually be delivered and is `true` by default. It can be convenient to set it to `false` for testing.

`config.action_mailer.default_options` Configures Action Mailer defaults. Use to set options like `from` or `reply_to` for every mailer. These default to:

```
mime_version:  "1.0",
charset:       "UTF-8",
```

```
content_type: "text/plain",
parts_order:  ["text/plain", "text/enriched", "text/html"]
```

Assign a hash to set additional options:

```
config.action_mailer.default_options = {
  from: "noreply@example.com"
}
```

config.action_mailer.observers Registers observers which will be notified when mail is delivered.

```
config.action_mailer.observers = ["MailObserver"]
```

config.action_mailer.interceptors Registers interceptors which will be called before mail is sent.

```
config.action_mailer.interceptors = ["MailInterceptor"]
```

config.action_mailer.preview_interceptors Registers interceptors which will be called before mail is previewed.

```
config.action_mailer.preview_interceptors = ["MyPreviewMailInterceptor"]
```

config.action_mailer.preview_path Specifies the location of mailer previews.

```
config.action_mailer.preview_path = "#{Rails.root}/lib/mailer_previews"
```

config.action_mailer.show_previews Enable or disable mailer previews. By default this is `true` in development.

```
config.action_mailer.show_previews = false
```

config.action_mailer.deliver_later_queue_name Specifies the Active Job queue to use for delivery jobs. When this option is set to `nil`, delivery jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`). Make sure that your Active Job adapter is also configured to process the specified queue, otherwise delivery jobs may be silently ignored.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:mailers`
6.1	`nil`

`config.action_mailer.perform_caching` Specifies whether the mailer templates should perform fragment caching or not. If it's not specified, the default will be `true`.

`config.action_mailer.delivery_job` Specifies delivery job for mail.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`ActionMailer::MailDeliveryJob`
6.0	`"ActionMailer::MailDeliveryJob"`

Configuring Active Support

There are a few configuration options available in Active Support:

`config.active_support.bare` Enables or disables the loading of `active_support/all` when booting Rails. Defaults to `nil`, which means `active_support/all` is loaded.

`config.active_support.test_order` Sets the order in which the test cases are executed. Possible values are `:random` and `:sorted`. Defaults to `:random`.

`config.active_support.escape_html_entities_in_json` Enables or disables the escaping of HTML entities in JSON serialization. Defaults to `true`.

`config.active_support.use_standard_json_time_format` Enables or disables serializing dates to ISO 8601 format. Defaults to `true`.

`config.active_support.time_precision` Sets the precision of JSON encoded time values. Defaults to `3`.

`config.active_support.hash_digest_class` Allows configuring the digest class to use to generate non-sensitive digests, such as the ETag header.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`OpenSSL::Digest::MD5`
5.2	`OpenSSL::Digest::SHA1`
7.0	`OpenSSL::Digest::SHA256`

`config.active_support.key_generator_hash_digest_class` Allows configuring the digest class to use to derive secrets from the configured secret base, such as for encrypted cookies.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`OpenSSL::Digest::SHA1`
7.0	`OpenSSL::Digest::SHA256`

`config.active_support.use_authenticated_message_encryption` Specifies whether to use AES-256-GCM authenticated encryption as the default cipher for encrypting messages instead of AES-256-CBC.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.2	`true`

`config.active_support.cache_format_version` Specifies which version of the cache serializer to use. Possible values are `6.1` and `7.0`.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`6.1`
7.0	`7.0`

`config.active_support.deprecation` Configures the behavior of deprecation warnings. The options are `:raise`, `:stderr`, `:log`, `:notify`, or `:silence`. The default is `:stderr`. Alternatively, you can set `ActiveSupport::Deprecation.behavior`.

`config.active_support.disallowed_deprecation` Configures the behavior of disallowed deprecation warnings. The options are `:raise`, `:stderr`, `:log`, `:notify`, or `:silence`. The default is `:raise`. Alternatively, you can set `ActiveSupport::Deprecation.disallowed_behavior`.

`config.active_support.disallowed_deprecation_warnings` Configures deprecation warnings that the Application considers disallowed. This allows, for example, specific deprecations to be treated as hard failures. Alternatively, you can set `ActiveSupport::Deprecation.disallowed_warnings`.

`config.active_support.report_deprecations` Allows you to disable all deprecation warnings (including disallowed deprecations); it makes `ActiveSupport::Deprecation.warn` a no-op. This is enabled by default in production.

`config.active_support.remove_deprecated_time_with_zone_name` Specifies whether to remove the deprecated override of the `ActiveSupport::TimeWithZone.name` method, to avoid triggering its deprecation warning.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`nil`
7.0	`true`

`config.active_support.isolation_level` Configures the locality of most of Rails internal state. If you use a fiber based server or job processor (e.g. `falcon`), you should set it to `:fiber`. Otherwise it is best to use `:thread` locality. Defaults to `:thread`.

`config.active_support.use_rfc4122_namespaced_uuids` Specifies whether generated namespaced UUIDs follow the RFC 4122 standard for namespace IDs provided as a `String` to `Digest::UUID.uuid_v3` or `Digest::UUID.uuid_v5` method calls.

If set to `true`:

- Only UUIDs are allowed as namespace IDs. If a namespace ID value provided is not allowed, an `ArgumentError` will be raised.
- No deprecation warning will be generated, no matter if the namespace ID used is one of the constants defined on `Digest::UUID` or a `String`.
- Namespace IDs are case-insensitive.
- All generated namespaced UUIDs should be compliant to the standard.

If set to `false`:

- Any `String` value can be used as namespace ID (although not recommended). No `ArgumentError` will be raised in this case in order to preserve backwards-compatibility.
- A deprecation warning will be generated if the namespace ID provided is not one of the constants defined on `Digest::UUID`.
- Namespace IDs are case-sensitive.
- Only namespaced UUIDs generated using one of the namespace ID constants defined on `Digest::UUID` are compliant to the standard.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.active_support.executor_around_test_case` Configure the test suite to call `Rails.application.executor.wrap` around test cases. This makes test cases behave closer to an actual request or job. Several features that are normally disabled in test, such as Active Record query cache and asynchronous queries will then be enabled.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.active_support.disable_to_s_conversion` Disables the override of the `#to_s` methods in some Ruby core classes. This config is for applications that want to take advantage early of a Ruby 3.1 optimization. This configuration needs to be set in `config/application.rb` inside the application class, otherwise it will not take effect.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`ActiveSupport::Logger.silencer` Is set to `false` to disable the ability to silence logging in a block. The default is `true`.

`ActiveSupport::Cache::Store.logger` Specifies the logger to use within cache store operations.

`ActiveSupport.to_time_preserves_timezone` Specifies whether `to_time` methods preserve the UTC offset of their receivers. If `false`, `to_time` methods will convert to the local system UTC offset instead.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
5.0	`true`

`ActiveSupport.utc_to_local_returns_utc_offset_times` Configures `ActiveSupport::TimeZone.utc_to_local` to return a time with a UTC offset instead of a UTC time incorporating that offset.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
6.1	`true`

`config.active_support.default_message_encryptor_serializer` Specifies what serializer the `MessageEncryptor` class will use by default.

Options are `:json`, `:hybrid`, and `:marshal`. `:hybrid` uses the `JsonWithMarshalFallback` class.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:marshal`
7.1	`:json`

`config.active_support.fallback_to_marshal_deserialization` Specifies if the `ActiveSupport::JsonWithMarshalFallback` class will fallback to `Marshal` when it encounters a `::JSON::ParserError`.

Defaults to `true`.

`config.active_support.use_marshal_serialization` Specifies if the `ActiveSupport::JsonWithMarshalFallback` class will use `Marshal` to serialize payloads.

If this is set to `false`, it will use `JSON` to serialize payloads.

Used to help migrate apps from `Marshal` to `JSON` as the default serializer for the `MessageEncryptor` class.

Defaults to `true`.

config.active_support.default_message_verifier_serializer Specifies what serializer the `MessageVerifier` class will use by default.

Options are `:json`, `:hybrid`, and `:marshal`. `:hybrid` uses the `JsonWithMarshalFallback` class.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:marshal`
7.1	`:json`

Configuring Active Job

`config.active_job` provides the following configuration options:

config.active_job.queue_adapter Sets the adapter for the queuing backend. The default adapter is `:async`. For an up-to-date list of built-in adapters see the ActiveJob::QueueAdapters API documentation.

```
# Be sure to have the adapter's gem in your Gemfile
# and follow the adapter's specific installation
# and deployment instructions.
config.active_job.queue_adapter = :sidekiq
```

config.active_job.default_queue_name Can be used to change the default queue name. By default this is `"default"`.

```
config.active_job.default_queue_name = :medium_priority
```

config.active_job.queue_name_prefix Allows you to set an optional, non-blank, queue name prefix for all jobs. By default it is blank and not used.

The following configuration would queue the given job on the `production_high_priority` queue when run in production:

```
config.active_job.queue_name_prefix = Rails.env
```

```
class GuestsCleanupJob < ActiveJob::Base
  queue_as :high_priority
  #....
end
```

config.active_job.queue_name_delimiter Has a default value of `'_'`. If `queue_name_prefix` is set, then `queue_name_delimiter` joins the prefix and the non-prefixed queue name.

The following configuration would queue the provided job on the `video_server.low_priority` queue:

```
# prefix must be set for delimiter to be used
config.active_job.queue_name_prefix = 'video_server'
config.active_job.queue_name_delimiter = '.'

class EncoderJob < ActiveJob::Base
  queue_as :low_priority
  #....
end
```

`config.active_job.logger` Accepts a logger conforming to the interface of Log4r or the default Ruby Logger class, which is then used to log information from Active Job. You can retrieve this logger by calling `logger` on either an Active Job class or an Active Job instance. Set to `nil` to disable logging.

`config.active_job.custom_serializers` Allows to set custom argument serializers. Defaults to `[]`.

`config.active_job.log_arguments` Controls if the arguments of a job are logged. Defaults to `true`.

`config.active_job.retry_jitter` Controls the amount of "jitter" (random variation) applied to the delay time calculated when retrying failed jobs.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`0.0`
6.1	`0.15`

`config.active_job.log_query_tags_around_perform` Determines whether job context for query tags will be automatically updated via an `around_perform`. The default value is `true`.

Configuring Action Cable

`config.action_cable.url` Accepts a string for the URL for where you are hosting your Action Cable server. You would use this option if you are running Action Cable servers that are separated from your main application.

`config.action_cable.mount_path` Accepts a string for where to mount Action Cable, as part of the main server process. Defaults to `/cable`. You can set this as nil to not mount Action Cable as part of your normal Rails server.

You can find more detailed configuration options in the Action Cable Overview.

`config.action_cable.precompile_assets` Determines whether the Action Cable assets should be added to the asset pipeline precompilation. It has no effect if Sprockets is not used. The default value is `true`.

Configuring Active Storage

`config.active_storage` provides the following configuration options:

`config.active_storage.variant_processor` Accepts a symbol `:mini_magick` or `:vips`, specifying whether variant transformations and blob analysis will be performed with MiniMagick or ruby-vips.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`:mini_magick`
7.0	`:vips`

`config.active_storage.analyzers` Accepts an array of classes indicating the analyzers available for Active Storage blobs. By default, this is defined as:

```
config.active_storage.analyzers = [ActiveStorage::Analyzer::ImageAnalyzer::Vips, ActiveStora
```

The image analyzers can extract width and height of an image blob; the video analyzer can extract width, height, duration, angle, aspect ratio, and presence/absence of video/audio channels of a video blob; the audio analyzer can extract duration and bit rate of an audio blob.

`config.active_storage.previewers` Accepts an array of classes indicating the image previewers available in Active Storage blobs. By default, this is defined as:

```
config.active_storage.previewers = [ActiveStorage::Previewer::PopplerPDFPreviewer, ActiveSto
```

`PopplerPDFPreviewer` and `MuPDFPreviewer` can generate a thumbnail from the first page of a PDF blob; `VideoPreviewer` from the relevant frame of a video blob.

config.active_storage.paths Accepts a hash of options indicating the locations of previewer/analyzer commands. The default is `{}`, meaning the commands will be looked for in the default path. Can include any of these options:

- `:ffprobe` - The location of the ffprobe executable.
- `:mutool` - The location of the mutool executable.
- `:ffmpeg` - The location of the ffmpeg executable.

```
config.active_storage.paths[:ffprobe] = '/usr/local/bin/ffprobe'
```

config.active_storage.variable_content_types Accepts an array of strings indicating the content types that Active Storage can transform through ImageMagick. By default, this is defined as:

```
config.active_storage.variable_content_types = %w(image/png image/gif image/jpeg image/tiff
```

config.active_storage.web_image_content_types Accepts an array of strings regarded as web image content types in which variants can be processed without being converted to the fallback PNG format. If you want to use `WebP` or `AVIF` variants in your application you can add `image/webp` or `image/avif` to this array. By default, this is defined as:

```
config.active_storage.web_image_content_types = %w(image/png image/jpeg image/gif)
```

config.active_storage.content_types_to_serve_as_binary Accepts an array of strings indicating the content types that Active Storage will always serve as an attachment, rather than inline. By default, this is defined as:

```
config.active_storage.content_types_to_serve_as_binary = %w(text/html image/svg+xml applicat
```

config.active_storage.content_types_allowed_inline Accepts an array of strings indicating the content types that Active Storage allows to serve as inline. By default, this is defined as:

```
config.active_storage.content_types_allowed_inline` = %w(image/png image/gif image/jpeg imag
```

config.active_storage.silence_invalid_content_types_warning Since Rails 7, Active Storage will warn if you use an invalid content type that was incorrectly supported in Rails 6. You can use this config to turn the warning off.

```
config.active_storage.silence_invalid_content_types_warning = false
```

config.active_storage.queues.analysis Accepts a symbol indicating the Active Job queue to use for analysis jobs. When this option is `nil`, analysis jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`).

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
6.0	`:active_storage_analysis`
6.1	`nil`

`config.active_storage.queues.purge` Accepts a symbol indicating the Active Job queue to use for purge jobs. When this option is `nil`, purge jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`).

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
6.0	`:active_storage_purge`
6.1	`nil`

`config.active_storage.queues.mirror` Accepts a symbol indicating the Active Job queue to use for direct upload mirroring jobs. When this option is `nil`, mirroring jobs are sent to the default Active Job queue (see `config.active_job.default_queue_name`). The default is `nil`.

`config.active_storage.logger` Can be used to set the logger used by Active Storage. Accepts a logger conforming to the interface of Log4r or the default Ruby Logger class.

```
config.active_storage.logger = ActiveSupport::Logger.new(STDOUT)
```

`config.active_storage.service_urls_expire_in` Determines the default expiry of URLs generated by:

- `ActiveStorage::Blob#url`
- `ActiveStorage::Blob#service_url_for_direct_upload`
- `ActiveStorage::Variant#url`

The default is 5 minutes.

`config.active_storage.urls_expire_in` Determines the default expiry of URLs in the Rails application generated by Active Storage. The default is nil.

`config.active_storage.routes_prefix` Can be used to set the route prefix for the routes served by Active Storage. Accepts a string that will be prepended to the generated routes.

```
config.active_storage.routes_prefix = '/files'
```

The default is `/rails/active_storage`.

`config.active_storage.replace_on_assign_to_many` Determines whether assigning to a collection of attachments declared with `has_many_attached` replaces any existing attachments or appends to them.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
6.0	`true`

`config.active_storage.track_variants` Determines whether variants are recorded in the database.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
6.1	`true`

`config.active_storage.draw_routes` Can be used to toggle Active Storage route generation. The default is `true`.

`config.active_storage.resolve_model_to_route` Can be used to globally change how Active Storage files are delivered.

Allowed values are:

- `:rails_storage_redirect`: Redirect to signed, short-lived service URLs.
- `:rails_storage_proxy`: Proxy files by downloading them.

The default is `:rails_storage_redirect`.

`config.active_storage.video_preview_arguments` Can be used to alter the way ffmpeg generates video preview images.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`"-y -vframes 1 -f image2"`
7.0	`"-vf 'select=eq(n\\,0)+eq(key\\,1)+gt(scene\\,0.015)"1 + ",loop=loop=-1:size=2,trim=start_frame=1'"2 + " -frames:v 1 -f image2"`

`config.active_storage.multiple_file_field_include_hidden` In Rails 7.1 and beyond, Active Storage `has_many_attached` relationships will default to *replacing* the current collection instead of *appending* to it. Thus to support submitting an *empty* collection, when `multiple_file_field_include_hidden` is `true`, the `file_field` helper will render an auxiliary hidden field, similar to the auxiliary field rendered by the `check_box` helper.

The default value depends on the `config.load_defaults` target version:

Starting with version	The default value is
(original)	`false`
7.0	`true`

`config.active_storage.precompile_assets` Determines whether the Active Storage assets should be added to the asset pipeline precompilation. It has no effect if Sprockets is not used. The default value is `true`.

Configuring Action Text

`config.action_text.attachment_tag_name` Accepts a string for the HTML tag used to wrap attachments. Defaults to `"action-text-attachment"`.

Configuring a Database

Just about every Rails application will interact with a database. You can connect to the database by setting an environment variable `ENV['DATABASE_URL']` or by using a configuration file called `config/database.yml`.

Using the `config/database.yml` file you can specify all the information needed to access your database:

```
development:
  adapter: postgresql
  database: blog_development
  pool: 5
```

This will connect to the database named `blog_development` using the `postgresql` adapter. This same information can be stored in a URL and provided via an environment variable like this:

```
ENV['DATABASE_URL'] # => "postgresql://localhost/blog_development?pool=5"
```

The `config/database.yml` file contains sections for three different environments in which Rails can run by default:

- The `development` environment is used on your development/local computer as you interact manually with the application.
- The `test` environment is used when running automated tests.

- The `production` environment is used when you deploy your application for the world to use.

If you wish, you can manually specify a URL inside of your `config/database.yml`

```
development:
  url: postgresql://localhost/blog_development?pool=5
```

The `config/database.yml` file can contain ERB tags `<%= %>`. Anything in the tags will be evaluated as Ruby code. You can use this to pull out data from an environment variable or to perform calculations to generate the needed connection information.

TIP: You don't have to update the database configurations manually. If you look at the options of the application generator, you will see that one of the options is named `--database`. This option allows you to choose an adapter from a list of the most used relational databases. You can even run the generator repeatedly: `cd .. && rails new blog --database=mysql`. When you confirm the overwriting of the `config/database.yml` file, your application will be configured for MySQL instead of SQLite. Detailed examples of the common database connections are below.

Connection Preference

Since there are two ways to configure your connection (using `config/database.yml` or using an environment variable) it is important to understand how they can interact.

If you have an empty `config/database.yml` file but your `ENV['DATABASE_URL']` is present, then Rails will connect to the database via your environment variable:

```
$ cat config/database.yml

$ echo $DATABASE_URL
postgresql://localhost/my_database
```

If you have a `config/database.yml` but no `ENV['DATABASE_URL']` then this file will be used to connect to your database:

```
$ cat config/database.yml
development:
  adapter: postgresql
  database: my_database
  host: localhost

$ echo $DATABASE_URL
```

If you have both `config/database.yml` and `ENV['DATABASE_URL']` set then Rails will merge the configuration together. To better understand this we must see some examples.

When duplicate connection information is provided the environment variable will take precedence:

```
$ cat config/database.yml
development:
  adapter: sqlite3
  database: NOT_my_database
  host: localhost

$ echo $DATABASE_URL
postgresql://localhost/my_database

$ bin/rails runner 'puts ActiveRecord::Base.configurations'
#<ActiveRecord::DatabaseConfigurations:0x00007fd50e209a28>

$ bin/rails runner 'puts ActiveRecord::Base.configurations.inspect'
#<ActiveRecord::DatabaseConfigurations:0x00007fc8eab02880 @configurations=[
  #<ActiveRecord::DatabaseConfigurations::UrlConfig:0x00007fc8eab020b0
    @env_name="development", @spec_name="primary",
    @config={"adapter"=>"postgresql", "database"=>"my_database", "host"=>"localhost"}
    @url="postgresql://localhost/my_database">
  ]
```

Here the adapter, host, and database match the information in `ENV['DATABASE_URL']`.

If non-duplicate information is provided you will get all unique values, environment variable still takes precedence in cases of any conflicts.

```
$ cat config/database.yml
development:
  adapter: sqlite3
  pool: 5

$ echo $DATABASE_URL
postgresql://localhost/my_database

$ bin/rails runner 'puts ActiveRecord::Base.configurations'
#<ActiveRecord::DatabaseConfigurations:0x00007fd50e209a28>

$ bin/rails runner 'puts ActiveRecord::Base.configurations.inspect'
#<ActiveRecord::DatabaseConfigurations:0x00007fc8eab02880 @configurations=[
  #<ActiveRecord::DatabaseConfigurations::UrlConfig:0x00007fc8eab020b0
    @env_name="development", @spec_name="primary",
    @config={"adapter"=>"postgresql", "database"=>"my_database", "host"=>"localhost", "pool"
    @url="postgresql://localhost/my_database">
  ]
```

Since pool is not in the `ENV['DATABASE_URL']` provided connection in-

formation its information is merged in. Since `adapter` is duplicate, the `ENV['DATABASE_URL']` connection information wins.

The only way to explicitly not use the connection information in `ENV['DATABASE_URL']` is to specify an explicit URL connection using the `"url"` sub key:

```
$ cat config/database.yml
development:
  url: sqlite3:NOT_my_database

$ echo $DATABASE_URL
postgresql://localhost/my_database

$ bin/rails runner 'puts ActiveRecord::Base.configurations'
#<ActiveRecord::DatabaseConfigurations:0x00007fd50e209a28>

$ bin/rails runner 'puts ActiveRecord::Base.configurations.inspect'
#<ActiveRecord::DatabaseConfigurations:0x00007fc8eab02880 @configurations=[
  #<ActiveRecord::DatabaseConfigurations::UrlConfig:0x00007fc8eab020b0
    @env_name="development", @spec_name="primary",
    @config={"adapter"=>"sqlite3", "database"=>"NOT_my_database"}
    @url="sqlite3:NOT_my_database">
  ]
```

Here the connection information in `ENV['DATABASE_URL']` is ignored, note the different adapter and database name.

Since it is possible to embed ERB in your `config/database.yml` it is best practice to explicitly show you are using the `ENV['DATABASE_URL']` to connect to your database. This is especially useful in production since you should not commit secrets like your database password into your source control (such as Git).

```
$ cat config/database.yml
production:
  url: <%= ENV['DATABASE_URL'] %>
```

Now the behavior is clear, that we are only using the connection information in `ENV['DATABASE_URL']`.

Configuring an SQLite3 Database Rails comes with built-in support for SQLite3, which is a lightweight serverless database application. While a busy production environment may overload SQLite, it works well for development and testing. Rails defaults to using an SQLite database when creating a new project, but you can always change it later.

Here's the section of the default configuration file (`config/database.yml`) with connection information for the development environment:

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
```

NOTE: Rails uses an SQLite3 database for data storage by default because it is a zero configuration database that just works. Rails also supports MySQL (including MariaDB) and PostgreSQL "out of the box", and has plugins for many database systems. If you are using a database in a production environment Rails most likely has an adapter for it.

Configuring a MySQL or MariaDB Database If you choose to use MySQL or MariaDB instead of the shipped SQLite3 database, your `config/database.yml` will look a little different. Here's the development section:

```
development:
  adapter: mysql2
  encoding: utf8mb4
  database: blog_development
  pool: 5
  username: root
  password:
  socket: /tmp/mysql.sock
```

If your development database has a root user with an empty password, this configuration should work for you. Otherwise, change the username and password in the `development` section as appropriate.

NOTE: If your MySQL version is 5.5 or 5.6 and want to use the `utf8mb4` character set by default, please configure your MySQL server to support the longer key prefix by enabling `innodb_large_prefix` system variable.

Advisory Locks are enabled by default on MySQL and are used to make database migrations concurrent safe. You can disable advisory locks by setting `advisory_locks` to `false`:

```
production:
  adapter: mysql2
  advisory_locks: false
```

Configuring a PostgreSQL Database If you choose to use PostgreSQL, your `config/database.yml` will be customized to use PostgreSQL databases:

```
development:
  adapter: postgresql
  encoding: unicode
```

```
  database: blog_development
  pool: 5
```

By default Active Record uses database features like prepared statements and advisory locks. You might need to disable those features if you're using an external connection pooler like PgBouncer:

```
production:
  adapter: postgresql
  prepared_statements: false
  advisory_locks: false
```

If enabled, Active Record will create up to `1000` prepared statements per database connection by default. To modify this behavior you can set `statement_limit` to a different value:

```
production:
  adapter: postgresql
  statement_limit: 200
```

The more prepared statements in use: the more memory your database will require. If your PostgreSQL database is hitting memory limits, try lowering `statement_limit` or disabling prepared statements.

Configuring an SQLite3 Database for JRuby Platform If you choose to use SQLite3 and are using JRuby, your `config/database.yml` will look a little different. Here's the development section:

```
development:
  adapter: jdbcsqlite3
  database: db/development.sqlite3
```

Configuring a MySQL or MariaDB Database for JRuby Platform If you choose to use MySQL or MariaDB and are using JRuby, your `config/database.yml` will look a little different. Here's the development section:

```
development:
  adapter: jdbcmysql
  database: blog_development
  username: root
  password:
```

Configuring a PostgreSQL Database for JRuby Platform If you choose to use PostgreSQL and are using JRuby, your `config/database.yml` will look a little different. Here's the development section:

```
development:
  adapter: jdbcpostgresql
```

```
  encoding: unicode
  database: blog_development
  username: blog
  password:
```

Change the username and password in the `development` section as appropriate.

Configuring Metadata Storage By default Rails will store information about your Rails environment and schema in an internal table named `ar_internal_metadata`.

To turn this off per connection, set `use_metadata_table` in your database configuration. This is useful when working with a shared database and/or database user that cannot create tables.

```
development:
  adapter: postgresql
  use_metadata_table: false
```

Creating Rails Environments

By default Rails ships with three environments: "development", "test", and "production". While these are sufficient for most use cases, there are circumstances when you want more environments.

Imagine you have a server which mirrors the production environment but is only used for testing. Such a server is commonly called a "staging server". To define an environment called "staging" for this server, just create a file called `config/environments/staging.rb`. Please use the contents of any existing file in `config/environments` as a starting point and make the necessary changes from there.

That environment is no different than the default ones, start a server with `bin/rails server -e staging`, a console with `bin/rails console -e staging`, `Rails.env.staging?` works, etc.

Deploy to a Subdirectory (relative URL root)

By default Rails expects that your application is running at the root (e.g. `/`). This section explains how to run your application inside a directory.

Let's assume we want to deploy our application to "/app1". Rails needs to know this directory to generate the appropriate routes:

```
config.relative_url_root = "/app1"
```

alternatively you can set the `RAILS_RELATIVE_URL_ROOT` environment variable.

Rails will now prepend "/app1" when generating links.

Using Passenger Passenger makes it easy to run your application in a subdirectory. You can find the relevant configuration in the Passenger manual.

Using a Reverse Proxy Deploying your application using a reverse proxy has definite advantages over traditional deploys. They allow you to have more control over your server by layering the components required by your application.

Many modern web servers can be used as a proxy server to balance third-party elements such as caching servers or application servers.

One such application server you can use is Unicorn to run behind a reverse proxy.

In this case, you would need to configure the proxy server (NGINX, Apache, etc) to accept connections from your application server (Unicorn). By default Unicorn will listen for TCP connections on port 8080, but you can change the port or configure it to use sockets instead.

You can find more information in the Unicorn readme and understand the philosophy behind it.

Once you've configured the application server, you must proxy requests to it by configuring your web server appropriately. For example your NGINX config may include:

```
upstream application_server {
  server 0.0.0.0:8080;
}

server {
  listen 80;
  server_name localhost;

  root /root/path/to/your_app/public;

  try_files $uri/index.html $uri.html @app;

  location @app {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass http://application_server;
  }

  # some other configuration
}
```

Be sure to read the NGINX documentation for the most up-to-date information.

Rails Environment Settings

Some parts of Rails can also be configured externally by supplying environment variables. The following environment variables are recognized by various parts of Rails:

- `ENV["RAILS_ENV"]` defines the Rails environment (production, development, test, and so on) that Rails will run under.
- `ENV["RAILS_RELATIVE_URL_ROOT"]` is used by the routing code to recognize URLs when you deploy your application to a subdirectory.
- `ENV["RAILS_CACHE_ID"]` and `ENV["RAILS_APP_VERSION"]` are used to generate expanded cache keys in Rails' caching code. This allows you to have multiple separate caches from the same application.

Using Initializer Files

After loading the framework and any gems in your application, Rails turns to loading initializers. An initializer is any Ruby file stored under `config/initializers` in your application. You can use initializers to hold configuration settings that should be made after all of the frameworks and gems are loaded, such as options to configure settings for these parts.

The files in `config/initializers` (and any subdirectories of `config/initializers`) are sorted and loaded one by one as part of the `load_config_initializers` initializer.

If an initializer has code that relies on code in another initializer, you can combine them into a single initializer instead. This makes the dependencies more explicit, and can help surface new concepts within your application. Rails also supports numbering of initializer file names, but this can lead to file name churn. Explicitly loading initializers with `require` is not recommended, since it will cause the initializer to get loaded twice.

NOTE: There is no guarantee that your initializers will run after all the gem initializers, so any initialization code that depends on a given gem having been initialized should go into a `config.after_initialize` block.

Initialization events

Rails has 5 initialization events which can be hooked into (listed in the order that they are run):

- `before_configuration`: This is run as soon as the application constant inherits from `Rails::Application`. The `config` calls are evaluated before this happens.
- `before_initialize`: This is run directly before the initialization process of the application occurs with the `:bootstrap_hook` initializer near the

beginning of the Rails initialization process.

- `to_prepare`: Run after the initializers are run for all Railties (including the application itself), but before eager loading and the middleware stack is built. More importantly, will run upon every code reload in `development`, but only once (during boot-up) in `production` and `test`.
- `before_eager_load`: This is run directly before eager loading occurs, which is the default behavior for the `production` environment and not for the `development` environment.
- `after_initialize`: Run directly after the initialization of the application, after the application initializers in `config/initializers` are run.

To define an event for these hooks, use the block syntax within a `Rails::Application`, `Rails::Railtie` or `Rails::Engine` subclass:

```
module YourApp
  class Application < Rails::Application
    config.before_initialize do
      # initialization code goes here
    end
  end
end
```

Alternatively, you can also do it through the `config` method on the `Rails.application` object:

```
Rails.application.config.before_initialize do
  # initialization code goes here
end
```

WARNING: Some parts of your application, notably routing, are not yet set up at the point where the `after_initialize` block is called.

`Rails::Railtie#initializer`

Rails has several initializers that run on startup that are all defined by using the `initializer` method from `Rails::Railtie`. Here's an example of the `set_helpers_path` initializer from Action Controller:

```
initializer "action_controller.set_helpers_path" do |app|
  ActionController::Helpers.helpers_path = app.helpers_paths
end
```

The `initializer` method takes three arguments with the first being the name for the initializer and the second being an options hash (not shown here) and the third being a block. The `:before` key in the options hash can be specified to specify which initializer this new initializer must run before, and the `:after` key will specify which initializer to run this initializer *after*.

Initializers defined using the `initializer` method will be run in the order they are defined in, with the exception of ones that use the `:before` or `:after` methods.

WARNING: You may put your initializer before or after any other initializer in the chain, as long as it is logical. Say you have 4 initializers called "one" through "four" (defined in that order) and you define "four" to go *before* "two" but *after* "three", that just isn't logical and Rails will not be able to determine your initializer order.

The block argument of the `initializer` method is the instance of the application itself, and so we can access the configuration on it by using the `config` method as done in the example.

Because `Rails::Application` inherits from `Rails::Railtie` (indirectly), you can use the `initializer` method in `config/application.rb` to define initializers for the application.

Initializers

Below is a comprehensive list of all the initializers found in Rails in the order that they are defined (and therefore run in, unless otherwise stated).

- `load_environment_hook`: Serves as a placeholder so that `:load_environment_config` can be defined to run before it.
- `load_active_support`: Requires `active_support/dependencies` which sets up the basis for Active Support. Optionally requires `active_support/all` if `config.active_support.bare` is un-truthful, which is the default.
- `initialize_logger`: Initializes the logger (an `ActiveSupport::Logger` object) for the application and makes it accessible at `Rails.logger`, provided that no initializer inserted before this point has defined `Rails.logger`.
- `initialize_cache`: If `Rails.cache` isn't set yet, initializes the cache by referencing the value in `config.cache_store` and stores the outcome as `Rails.cache`. If this object responds to the `middleware` method, its middleware is inserted before `Rack::Runtime` in the middleware stack.
- `set_clear_dependencies_hook`: This initializer - which runs only if `cache_classes` is set to `false` - uses `ActionDispatch::Callbacks.after` to remove the constants which have been referenced during the request from the object space so that they will be reloaded during the following request.
- `bootstrap_hook`: Runs all configured `before_initialize` blocks.
- `i18n.callbacks`: In the development environment, sets up a `to_prepare` callback which will call `I18n.reload!` if any of the locales have changed

since the last request. In production this callback will only run on the first request.

- `active_support.deprecation_behavior`: Sets up deprecation reporting for environments, defaulting to `:log` for development, `:silence` for production, and `:stderr` for test. Can be set to an array of values. This initializer also sets up behaviors for disallowed deprecations, defaulting to `:raise` for development and test and `:silence` for production. Disallowed deprecation warnings default to an empty array.
- `active_support.initialize_time_zone`: Sets the default time zone for the application based on the `config.time_zone` setting, which defaults to "UTC".
- `active_support.initialize_beginning_of_week`: Sets the default beginning of week for the application based on `config.beginning_of_week` setting, which defaults to `:monday`.
- `active_support.set_configs`: Sets up Active Support by using the settings in `config.active_support` by `send`'ing the method names as setters to `ActiveSupport` and passing the values through.
- `action_dispatch.configure`: Configures the `ActionDispatch::Http::URL.tld_length` to be set to the value of `config.action_dispatch.tld_length`.
- `action_view.set_configs`: Sets up Action View by using the settings in `config.action_view` by `send`'ing the method names as setters to `ActionView::Base` and passing the values through.
- `action_controller.assets_config`: Initializes the `config.action_controller.assets_dir` to the app's public directory if not explicitly configured.
- `action_controller.set_helpers_path`: Sets Action Controller's `helpers_path` to the application's `helpers_path`.
- `action_controller.parameters_config`: Configures strong parameters options for `ActionController::Parameters`.
- `action_controller.set_configs`: Sets up Action Controller by using the settings in `config.action_controller` by `send`'ing the method names as setters to `ActionController::Base` and passing the values through.
- `action_controller.compile_config_methods`: Initializes methods for the config settings specified so that they are quicker to access.
- `active_record.initialize_timezone`: Sets `ActiveRecord::Base.time_zone_aware_attributes` to `true`, as well as setting `ActiveRecord::Base.default_timezone` to UTC. When attributes are read from the database, they will be converted into the time zone specified by `Time.zone`.
- `active_record.logger`: Sets `ActiveRecord::Base.logger` - if it's not already set - to `Rails.logger`.

- `active_record.migration_error`: Configures middleware to check for pending migrations.
- `active_record.check_schema_cache_dump`: Loads the schema cache dump if configured and available.
- `active_record.warn_on_records_fetched_greater_than`: Enables warnings when queries return large numbers of records.
- `active_record.set_configs`: Sets up Active Record by using the settings in `config.active_record` by `send`'ing the method names as setters to `ActiveRecord::Base` and passing the values through.
- `active_record.initialize_database`: Loads the database configuration (by default) from `config/database.yml` and establishes a connection for the current environment.
- `active_record.log_runtime`: Includes `ActiveRecord::Railties::ControllerRuntime` which is responsible for reporting the time taken by Active Record calls for the request back to the logger.
- `active_record.set_reloader_hooks`: Resets all reloadable connections to the database if `config.cache_classes` is set to `false`.
- `active_record.add_watchable_files`: Adds `schema.rb` and `structure.sql` files to watchable files.
- `active_job.logger`: Sets `ActiveJob::Base.logger` - if it's not already set - to `Rails.logger`.
- `active_job.set_configs`: Sets up Active Job by using the settings in `config.active_job` by `send`'ing the method names as setters to `ActiveJob::Base` and passing the values through.
- `action_mailer.logger`: Sets `ActionMailer::Base.logger` - if it's not already set - to `Rails.logger`.
- `action_mailer.set_configs`: Sets up Action Mailer by using the settings in `config.action_mailer` by `send`'ing the method names as setters to `ActionMailer::Base` and passing the values through.
- `action_mailer.compile_config_methods`: Initializes methods for the config settings specified so that they are quicker to access.
- `set_load_path`: This initializer runs before `bootstrap_hook`. Adds paths specified by `config.load_paths` and all autoload paths to `$LOAD_PATH`.
- `set_autoload_paths`: This initializer runs before `bootstrap_hook`. Adds all sub-directories of `app` and paths specified by `config.autoload_paths`, `config.eager_load_paths` and `config.autoload_once_paths` to `ActiveSupport::Dependencies.autoload_paths`.

- `add_routing_paths`: Loads (by default) all `config/routes.rb` files (in the application and railties, including engines) and sets up the routes for the application.
- `add_locales`: Adds the files in `config/locales` (from the application, railties, and engines) to `I18n.load_path`, making available the translations in these files.
- `add_view_paths`: Adds the directory `app/views` from the application, railties, and engines to the lookup path for view files for the application.
- `load_environment_config`: Loads the `config/environments` file for the current environment.
- `prepend_helpers_path`: Adds the directory `app/helpers` from the application, railties, and engines to the lookup path for helpers for the application.
- `load_config_initializers`: Loads all Ruby files from `config/initializers` in the application, railties, and engines. The files in this directory can be used to hold configuration settings that should be made after all of the frameworks are loaded.
- `engines_blank_point`: Provides a point-in-initialization to hook into if you wish to do anything before engines are loaded. After this point, all railtie and engine initializers are run.
- `add_generator_templates`: Finds templates for generators at `lib/templates` for the application, railties, and engines, and adds these to the `config.generators.templates` setting, which will make the templates available for all generators to reference.
- `ensure_autoload_once_paths_as_subset`: Ensures that the `config.autoload_once_paths` only contains paths from `config.autoload_paths`. If it contains extra paths, then an exception will be raised.
- `add_to_prepare_blocks`: The block for every `config.to_prepare` call in the application, a railtie, or engine is added to the `to_prepare` callbacks for Action Dispatch which will be run per request in development, or before the first request in production.
- `add_builtin_route`: If the application is running under the development environment then this will append the route for `rails/info/properties` to the application routes. This route provides the detailed information such as Rails and Ruby version for `public/index.html` in a default Rails application.
- `build_middleware_stack`: Builds the middleware stack for the application, returning an object which has a `call` method which takes a Rack environment object for the request.

- `eager_load!`: If `config.eager_load` is `true`, runs the `config.before_eager_load` hooks and then calls `eager_load!` which will load all `config.eager_load_namespaces`.
- `finisher_hook`: Provides a hook for after the initialization of process of the application is complete, as well as running all the `config.after_initialize` blocks for the application, railties, and engines.
- `set_routes_reloader_hook`: Configures Action Dispatch to reload the routes file using `ActiveSupport::Callbacks.to_run`.
- `disable_dependency_loading`: Disables the automatic dependency loading if the `config.eager_load` is set to `true`.

Database pooling

Active Record database connections are managed by `ActiveRecord::ConnectionAdapters::ConnectionPool` which ensures that a connection pool synchronizes the amount of thread access to a limited number of database connections. This limit defaults to 5 and can be configured in `database.yml`.

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
```

Since the connection pooling is handled inside of Active Record by default, all application servers (Thin, Puma, Unicorn, etc.) should behave the same. The database connection pool is initially empty. As demand for connections increases it will create them until it reaches the connection pool limit.

Any one request will check out a connection the first time it requires access to the database. At the end of the request it will check the connection back in. This means that the additional connection slot will be available again for the next request in the queue.

If you try to use more connections than are available, Active Record will block you and wait for a connection from the pool. If it cannot get a connection, a timeout error similar to that given below will be thrown.

```
ActiveRecord::ConnectionTimeoutError - could not obtain a database connection within 5.000 s
```

If you get the above error, you might want to increase the size of the connection pool by incrementing the `pool` option in `database.yml`

NOTE. If you are running in a multi-threaded environment, there could be a chance that several threads may be accessing multiple connections simultaneously. So depending on your current request load, you could very well have multiple threads contending for a limited number of connections.

Custom configuration

You can configure your own code through the Rails configuration object with custom configuration under either the `config.x` namespace, or `config` directly. The key difference between these two is that you should be using `config.x` if you are defining *nested* configuration (ex: `config.x.nested.hi`), and just `config` for *single level* configuration (ex: `config.hello`).

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries  = 3
config.super_debugger = true
```

These configuration points are then available through the configuration object:

```
Rails.configuration.x.payment_processing.schedule # => :daily
Rails.configuration.x.payment_processing.retries  # => 3
Rails.configuration.x.payment_processing.not_set  # => nil
Rails.configuration.super_debugger                # => true
```

You can also use `Rails::Application.config_for` to load whole configuration files:

```
# config/payment.yml
production:
  environment: production
  merchant_id: production_merchant_id
  public_key:  production_public_key
  private_key: production_private_key

development:
  environment: sandbox
  merchant_id: development_merchant_id
  public_key:  development_public_key
  private_key: development_private_key
```

```
# config/application.rb
module MyApp
  class Application < Rails::Application
    config.payment = config_for(:payment)
  end
end
```

```
Rails.configuration.payment['merchant_id'] # => production_merchant_id or development_mercha
```

`Rails::Application.config_for` supports a `shared` configuration to group common configurations. The shared configuration will be merged into the environment configuration.

```
# config/example.yml
shared:
```

```
  foo:
    bar:
      baz: 1

development:
  foo:
    bar:
      qux: 2
```

```
# development environment
Rails.application.config_for(:example)[:foo][:bar] #=> { baz: 1, qux: 2 }
```

Search Engines Indexing

Sometimes, you may want to prevent some pages of your application to be visible on search sites like Google, Bing, Yahoo, or Duck Duck Go. The robots that index these sites will first analyze the `http://your-site.com/robots.txt` file to know which pages it is allowed to index.

Rails creates this file for you inside the `/public` folder. By default, it allows search engines to index all pages of your application. If you want to block indexing on all pages of your application, use this:

```
User-agent: *
Disallow: /
```

To block just specific pages, it's necessary to use a more complex syntax. Learn it on the official documentation.

Evented File System Monitor

If the listen gem is loaded Rails uses an evented file system monitor to detect changes when `config.cache_classes` is `false`:

```
group :development do
  gem 'listen', '~> 3.3'
end
```

Otherwise, in every request Rails walks the application tree to check if anything has changed.

On Linux and macOS no additional gems are needed, but some are required for *BSD and for Windows.

Note that some setups are unsupported.