## Overview

This project \*\*does not\*\* use GitHub issues for questions, investigations, discussions, and so on.

Issues are appropriate for something specific enough for a maintainer or contributor to work on:

\* There should be enough information to reproduce the behavior observed in a reasonable amount of time

\* It should be reasonably clear why the behavior should be changed and why this cannot or should not be addressed

in application code, a separate library and so on

All issues that do not satisfy the above properties belong to the [Ruby RabbitMQ clients mailing list](http://groups.google.com/forum/#!forum/ruby-amqp). Pull request that do not satisfy them have a high chance

of being closed.

## Submitting a Pull Request

Please read the sections below to get an idea about how to run Bunny test suites first. Successfully

running all tests, at least with `CI` environment variable exported to `true`, is an important

first step for any contributor.

Once you have a passing test suite, create a branch and make your changes on it.

When you are done with your changes and all

tests pass, write a [good, detailed commit message](http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html) submit a pull request on GitHub.

## Pre-requisites

The project uses Bundler for dependency management and requires RabbitMQ `3.5+` to be running

locally with the `rabbitmq-management` and `rabbitmq\_consistent\_hash\_exchange` plugins enabled.

### Running the Specs

The specs require RabbitMQ to be running locally with a specific set of virtual hosts

and users. RabbitMQ can be provisioned and started any way that's convenient to you

as long as it has a suitable TLS keys configuration and management plugin enabled.

Make sure you have a recent version of RabbitMQ (> `3.7.10`).

The test suite can either use a locally available RabbitMQ node ([generic binary builds](http://www.rabbitmq.com/install-generic-unix.html)

are an option that works well) or by running a RabbitMQ server in a Docker container.

### Using a locally installed RabbitMQ node

It is possible to start a local RabbitMQ node from the repository root. It is not necessarily

optimal but can be a good starting point but is a useful example:

```

RABBITMQ\_NODENAME=bunny RABBITMQ\_CONFIG\_FILE=./spec/config/rabbitmq.conf RABBITMQ\_ENABLED\_PLUGINS\_FILE=./spec/config/enabled\_plugins rabbitmq-server

```

The specs need the RabbitMQ management plugin to be enabled and include TLS connectivity tests,

so the node must be configured to use a [certificate and key pair](http://www.rabbitmq.com/ssl.html#certificates-and-keys).

The config and enabled plugin files in the spec/config directory take care of that

but certificates must be provisioned locally. By default there's a set of CA, server, and client certificates pre-generated at `spec/tls`.

The `BUNNY\_CERTIFICATE\_DIR` environment variable can be used to a directory containing a CA certificate

and a certificate/key pair to be used by the server. The directory can be generated using

[tls-gen](https://github.com/michaelklishin/tls-gen)'s basic profile. This option is recommended.

`BUNNY\_RABBITMQ\_HOSTNAME` can be used to override the expected server hostname for [peer verification](http://www.rabbitmq.com/ssl.html#peer-verification) in the TLS test suite:

```

BUNNY\_CERTIFICATE\_DIR="/path/to/tls-gen/basic/result" BUNNY\_RABBITMQ\_HOSTNAME="mayflower" bundle exec rspec

```

Certificates can be generated with [tls-gen](https://github.com/michaelklishin/tls-gen)'s basic profile.

In that case they include a Subject Alternative Name of `localhost` for improved portability.

### Node Setup

There is also a script that preconfigured the node for Bunny tests. It is sufficient to run

it once but if RabbitMQ is reset it has to be executed again:

```

RABBITMQ\_NODENAME=bunny ./bin/ci/before\_build

```

The script uses `rabbitmqctl` and `rabbitmq-plugins`

to set up RabbitMQ in a way that Bunny test suites expect. Two environment variables,

`RABBITMQCTL` and `RABBITMQ\_PLUGINS`, are available to control what `rabbitmqctl` and

`rabbitmq-plugins` commands will be used. By default they are taken from `PATH`

and prefixed with `sudo`.

And then run the core integration suite:

```

RABBITMQ\_NODENAME=bunny CI=true rspec

```

#### Running a RabbitMQ server in a Docker container

First off you have to [install Docker Compose](https://docker.github.io/compose/install/) (and by proxy Docker).

Version >= 1.6.0+ is required for compose version 2 syntax.

After those have been installed (and the `docker-compose` command is available on your command line path), run

```

docker-compose build && docker-compose run --service-ports rabbitmq

```

The first time you do this, it will take some time, since it has to download everything it needs

to build the Docker image.

The RabbitMQ server will run in the foreground in the terminal where you started it. You can stop

it by pressing CTRL+C. If you want to run it in the background, pass `-d` to `docker-compose`.

### Toxiproxy

If Toxiproxy is running locally on standard ports or started via Docker:

```

docker-compose run --service-ports toxiproxy

```

then Bunny will run additional resiliency tests.

### Running Test Suites

Prior to running the tests, configure the RabbitMQ permissions by running `./bin/ci/before\_build`

if you have RabbitMQ locally installed, if you are running RabbitMQ via Docker as above this step

is not required as the setup is baked in.

Make sure you have those two installed and then run integration tests:

bundle install

rake integration

It is possible to run all tests:

bundle exec rspec

It is possible to run only integration and regression tests but exclude unit and stress tests:

CI=true bundle exec rspec spec/higher\_level\_api/ spec/lower\_level\_api spec/issues spec/higher\_level\_api/integration/connection\_recovery\_spec.rb