Contributing to cookbook-elasticsearch

======================================

### General philosophy

We try, as much as possible, to mirror the upstream experience of installing, configuring, and running Elasticsearch. Sometimes, the upstream project won't expose certain settings, or provide defaults that everyone agrees with. Wherever possible, we will:

- follow the upstream standards for specific config files

- include any files shipped with packages upstream in this cookbook as-is

- expose a template and cookbook setting for any file that you might want to modify

- recommend major changes to standard files be sent upstream first

- minimize the number of exceptions and maintained "workarounds" in this cookbook

### Workflow for contributing

1. Create a branch directly in this repo or a fork (if you don't have push access). Please name branches within this repository `<github username>/<description>`. For example, something like karmi/install\_from\_deb.

1. Create an issue or open a PR. If you aren't sure your PR will solve the issue, or may be controversial, we commend opening an issue separately and linking to it in your PR, so that if the PR is not accepted, the issue will remain and be tracked.

1. Close (and reference) issues by the `closes #XXX` or `fixes #XXX` notation in the commit message. Please use a descriptive, useful commit message that could be used to understand why a particular change was made.

1. Keep pushing commits to the initial branch, `--amend`-ing if necessary. Please don't mix fixing unrelated issues in a single branch.

1. When everything is ready for merge, clean up the branch (rebase with master to synchronize, squash, edit commits, etc) to prepare for it to be merged.

### Merging contributions

1. After reviewing commits for documentation, passing CI tests, and good descriptive commit messages, merge it with --no-ff switch, so it's indicated in the Git history

1. Do not use the Github "merge button", since it doesn't do a fast-forward merge (see previous item).

### Testing

This cookbook is equipped with both unit tests (chefspec) and integration tests

(test-kitchen and serverspec). It also comes with rubocop and foodcritic tasks

in the supplied Rakefile. Contributions to this cookbook should include tests

for new features or bugfixes, with a preference for unit tests over integration

tests to ensure speedy testing runs. \*\*\*All tests and most other commands here

should be run using bundler\*\*\* and our standard Gemfile. This ensures that

contributions and changes are made in a standardized way against the same

versions of gems. We recommend installing rubygems-bundler so that bundler is

automatically inserting `bundle exec` in front of commands run in a directory

that contains a Gemfile.

A full test run of all tests and style checks would look like:

```bash

$ bundle exec rake style

$ bundle exec rake spec

$ bundle exec rake integration

$ bundle exec rake destroy

```

The final destroy is intended to clean up any systems that failed a test, and is

mostly useful when running with kitchen drivers for cloud providers, so that no

machines are left orphaned and costing you money.

#### Fixtures

This cookbook supplies a few different test fixtures (under `test/fixtures/`)

that can be shared amongst any number of unit or integration tests: cookbooks,

environments, and nodes. Environments and nodes are automatically loaded into

chef-zero for both chefspec tests that run locally and serverspec tests that run

from test-kitchen.

It also contains 'platform data' that can be used to drive unit testing, for

example, you might read `httpd` for some platforms and `apache2` for others,

allowing you to write a single test for the Apache webserver. Unfortunately,

without further modifications to `busser` and `busser-serverspec`, the platform

data will not be available to serverspec tests.

#### Style and Best Practices

Rubocop and Foodcritic evaluations may be made by running `rake style`. There

are no overrides for foodcritic rules, however the adjustments to

rubocop are made using the supplied `.rubocop.yml` file and have been documented

by comments within. Most notably, rubocop has been restricted to only apply to

`.rb` files.

Rubocop and foodcritic tests can be executed using `rake style`.

#### Unit testing

Unit testing is done using the latest versions of Chefspec. The current default

test layout includes running against all supported platforms, as well as

stubbing data into chef-zero. This allows us to also test against chef search.

As is currently a best practice in the community, we will avoid the use of

chef-solo, but not create barriers to explicitly fail for chef-solo.

Unit tests can be executed using `rake spec`.

#### Integration testing

Integration testing is accomplished using the latest versions of test-kitchen

and serverspec. Currently, this cookbook uses the busser-serverspec plugin for

copying serverspec files to the system being tested. There is some debate in the

community about whether this should be done using busser-rspec instead, and each

busser plugin has a slightly different feature set.

While the default test-kitchen configuration uses the vagrant driver, you may

override this using `~/.kitchen/config.yml` or by placing a `.kitchen.local.yml`

in the current directory. This allows you to run these integration tests using

any supported test-kitchen driver (ec2, rackspace, docker, etc).

Integration tests can be executed using `rake integration` or `kitchen test`.

### Releasing

1. Create/update the changelog. We are using the `github\_changelog\_generator`

gem.

1. We highly recommend using the `stove` project, which pushes cookbooks to

Supermarket and tags to Github.