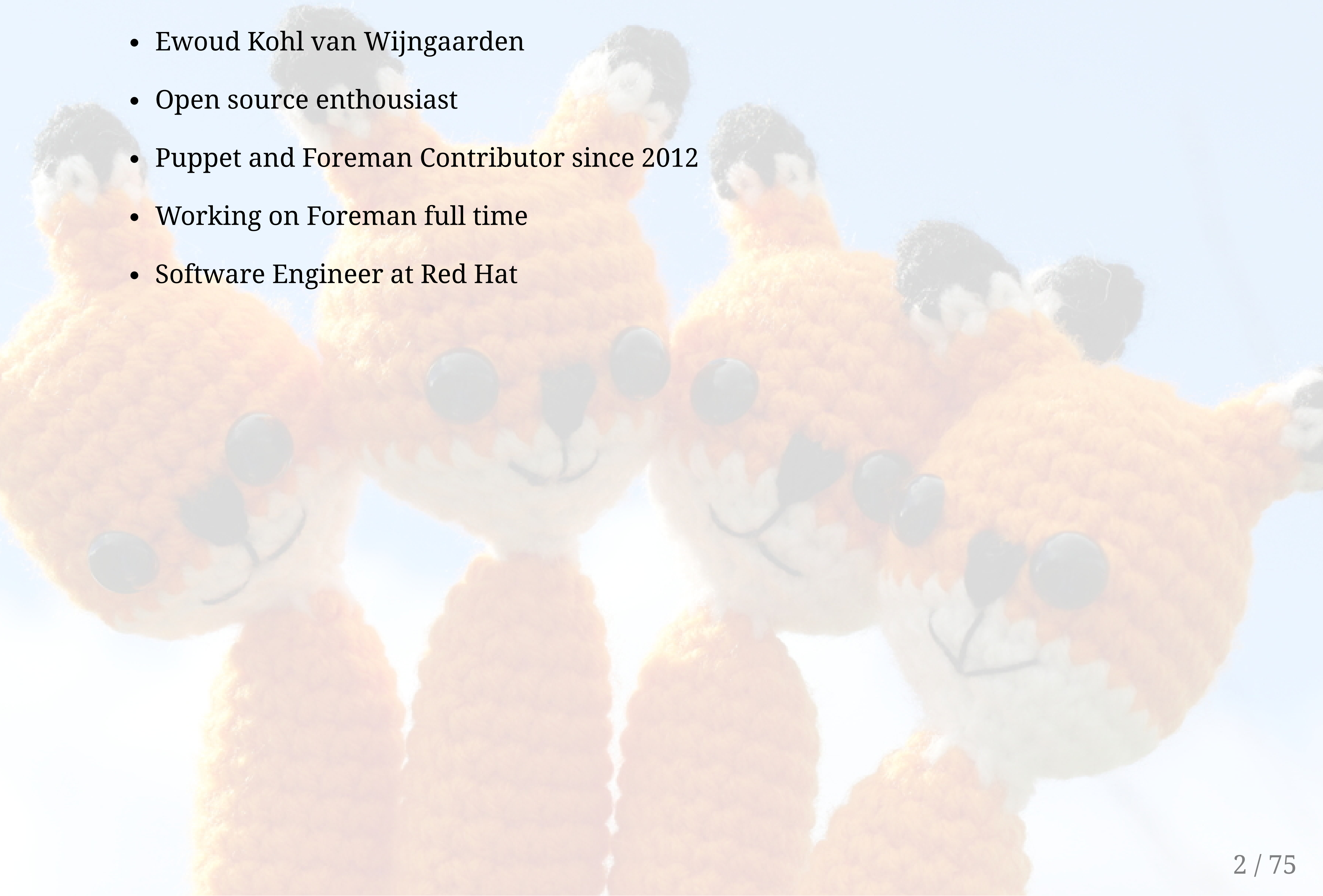
How Vox Pupuli built their Continuous Integration

Understanding how the puzzle pieces fit together



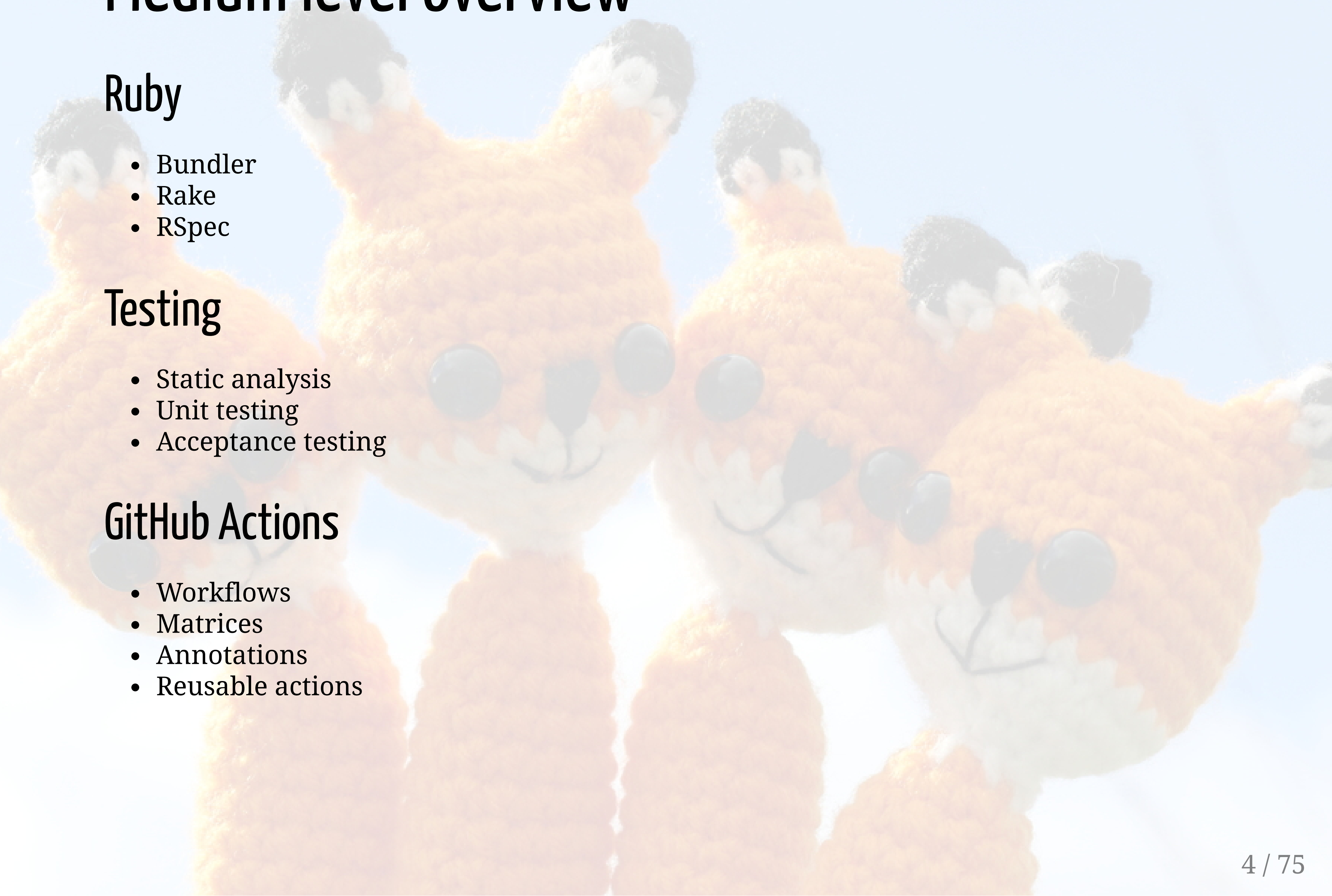
$ whoami

- Ewoud Kohl van Wijngaarden
- Open source enthousiast
- Puppet and Foreman Contributor since 2012
- Working on Foreman full time
- Software Engineer at Red Hat

High level overview

About Ruby, Testing, and GitHub Actions

Medium level overview



Ruby

- Bundler
- Rake
- RSpec

Testing

- Static analysis
- Unit testing
- Acceptance testing

GitHub Actions

- Workflows
- Matrices
- Annotations
- Reusable actions

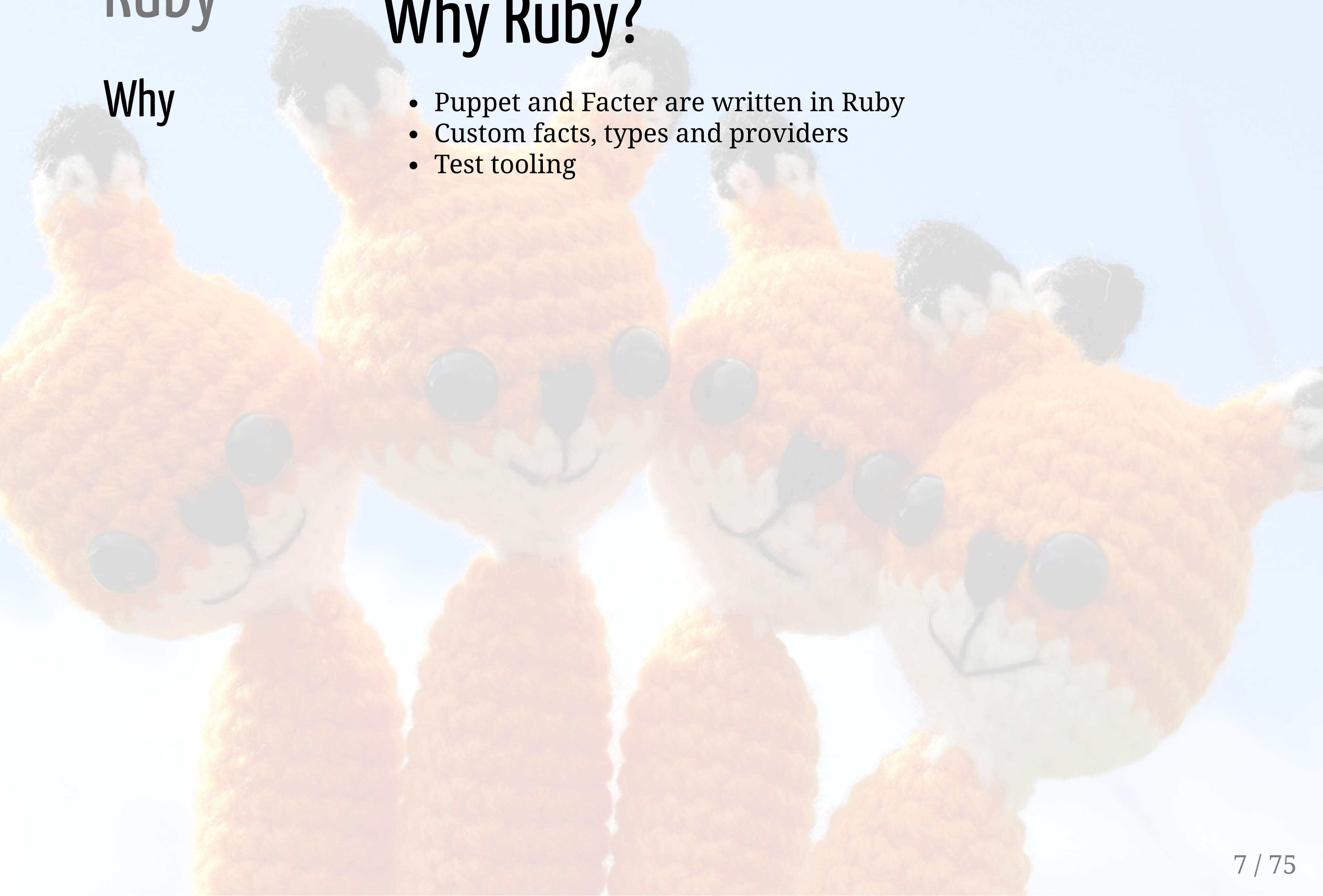
This is about test suites

Not about writing tests

Ruby

Why Ruby?

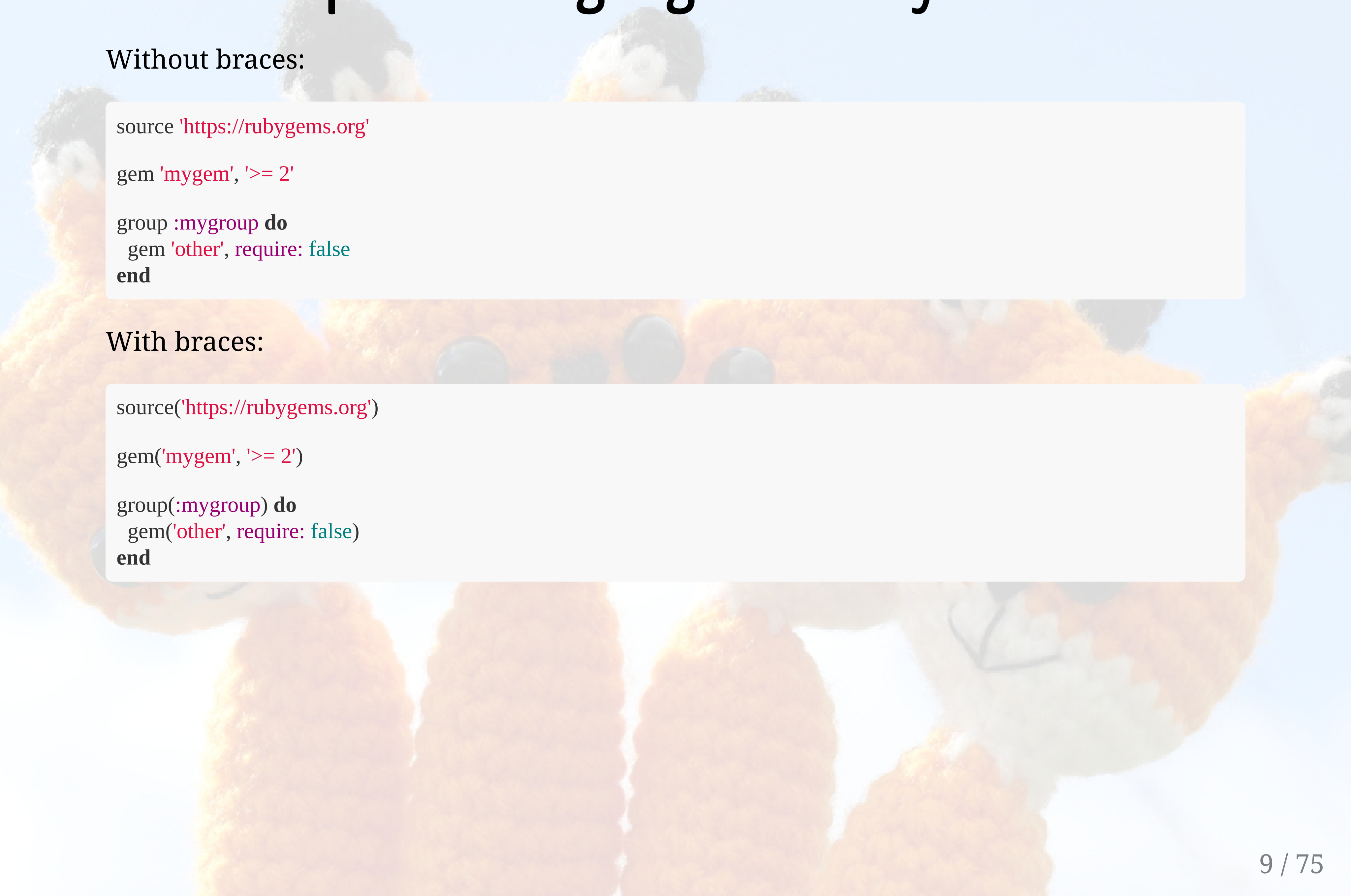
- Puppet and Facter are written in Ruby
- Custom facts, types and providers
- Test tooling



Bundler

- Isolated enviroments
- Lockfile
- Gemfile

```
source 'https://rubygems.org'

gem 'mygem'

group :mygroup do
  gem 'other'
end
```



Domain Specific Languages in Ruby

Without braces:

```ruby
source 'https://rubygems.org'

gem 'mygem', '>= 2'

group :mygroup do
  gem 'other', require: false
end
```

With braces:

```ruby
source('https://rubygems.org')

gem('mygem', '>= 2')

group(:mygroup) do
  gem('other', require: false)
end
```

Rake

- Just Ruby
- Inspired by make
- Tasks with prequisites

```
desc 'My First Rake task'
task :hello do
  puts 'Hello World'
end

namespace :check do
  task :first { puts 'First' }
  task :second { puts 'Second' }
end

desc 'Run all checks'
task :check => ['check:first', 'check:second']

task :default => [:hello]
```

RSpec

RSpec is a Behaviour-Driven Development tool for Ruby programmers. BDD is an approach to software development that combines Test-Driven Development, Domain Driven Design, and Acceptance Test-Driven Planning. RSpec helps you do the TDD part of that equation, focusing on the documentation and design aspects of TDD.

https://relishapp.com/rspec

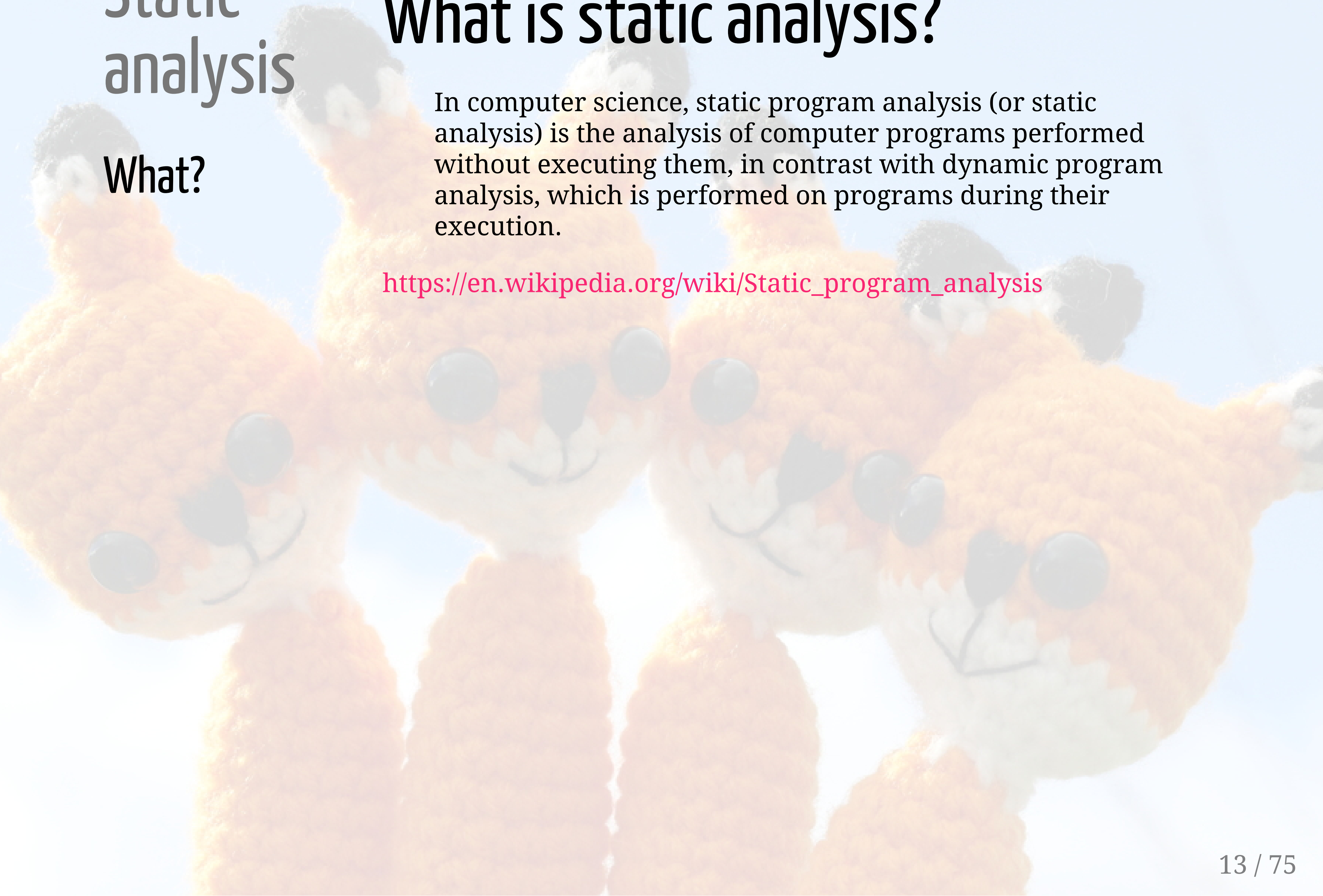
```ruby
RSpec.describe Game do
  describe "#score" do
    it "returns 0 for an all gutter game" do
      game = Game.new
      20.times { game.roll(0) }
      expect(game.score).to eq(0)
    end
  end
end
```

Static analysis

What is static analysis?

> In computer science, static program analysis (or static analysis) is the analysis of computer programs performed without executing them, in contrast with dynamic program analysis, which is performed on programs during their execution.

https://en.wikipedia.org/wiki/Static_program_analysis



Static analysis

What?

Syntax

puppet-syntax

Puppet::Syntax checks for correct syntax in Puppet manifests, templates, and Hiera YAML.

https://github.com/voxpupuli/puppet-syntax

```
$ bundle exec rake syntax
---> syntax:manifests
Could not parse for environment *root*: Syntax error at end of input (file: invalid.pp)
```

metadata-json-lint

- Validates metadata.json against a schema
- Lints
  - Duplicate dependencies
  - Deprecated fields
  - Warn about EOL Puppet version

https://github.com/voxpupuli/metadata-json-lint

```
$ bundle exec metadata-json-lint metadata.json
(ERROR) version: The property 'version' must be a valid semantic version: Unable to parse
  '0.2.1x' as a semantic version identifier
(ERROR) required_fields: The file did not contain a required property of 'name'
Errors found in metadata.json
```

Static analysis

What?

Syntax

Metadata

Lint

puppet-lint

- Check that your Puppet manifests conform to the style guide
- Many checks can autofix
- Many plugins
- Forked to puppetlabs

```
$ bundle exec puppet-lint manifests/dirty.pp
WARNING: class not documented on line 1 (check: documentation)
WARNING: class included by absolute name (::$class) on line 2 (check: relative_classname_inc
WARNING: indent should be 2 chars and is 0 on line 2 (check: strict_indent)
```

https://github.com/puppetlabs/puppet-lint

https://github.com/voxpupuli/voxpupuli-puppet-lint-plugins

http://puppet-lint.com

RuboCop

> RuboCop is a Ruby static code analyzer (a.k.a. linter) and code formatter. Out of the box it will enforce many of the guidelines outlined in the community Ruby Style Guide.

https://rubocop.org/

```
$ bundle exec rubocop
Inspecting 1 file
W

Offenses:

test.rb:1:5: C: Naming/MethodName: Use snake_case for method names.
def badName
    ^^^^^^^
test.rb:4:5: W: Layout/EndAlignment: end at 4, 4 is not aligned with if at 2, 2.
    end
    ^^^

1 file inspected, 2 offenses detected
```

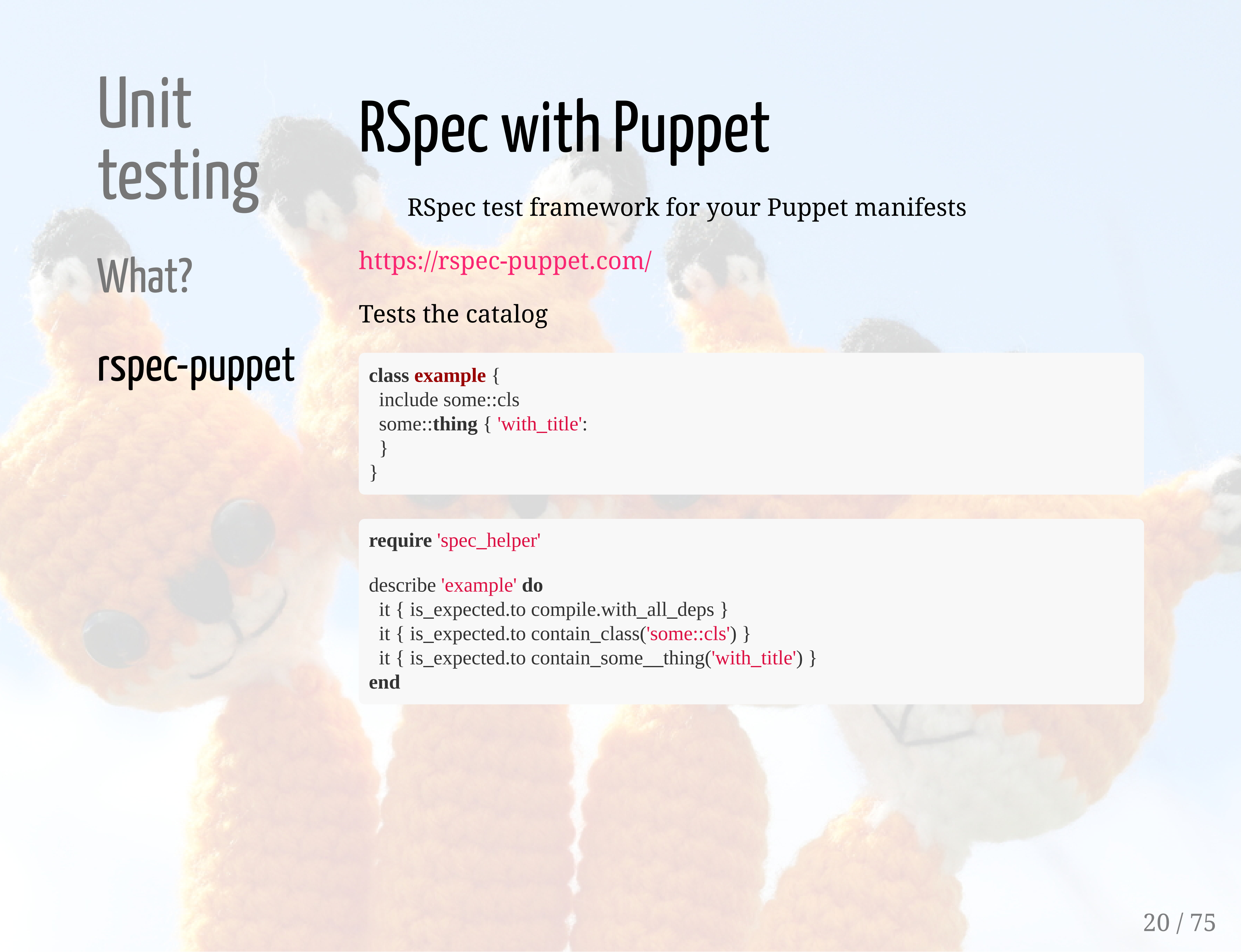
Unit testing

What is unit testing

> In computer programming, unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use.

https://en.wikipedia.org/wiki/Unit_testing

TL;DR: Test smaller parts individually



RSpec with Puppet

RSpec test framework for your Puppet manifests

https://rspec-puppet.com/

Tests the catalog

```
class example {
  include some::cls
  some::thing { 'with_title':
  }
}
```

```
require 'spec_helper'

describe 'example' do
  it { is_expected.to compile.with_all_deps }
  it { is_expected.to contain_class('some::cls') }
  it { is_expected.to contain_some__thing('with_title') }
end
```

Dealing with facts

```
require 'spec_helper'

describe 'example' do
  context 'on Red Hat 9' do
    let(:facts) do
      {
        os: {
          release: {
            major: '9'
          }
        }
      }
    end

    it { is_expected.to compile.with_all_deps }
  end

  context 'on Debian 11' do
    let(:facts) { ... }

    it { is_expected.to compile.with_all_deps }
  end
end
```

Stubbing facts with FacterDB

Simplify your unit tests by looping on every supported Operating System and populating facts.

https://github.com/voxpupuli/rspec-puppet-facts

A Database of OS facts provided by Facter

https://github.com/voxpupuli/facterdb

```ruby
require 'spec_helper'

describe 'example' do
  on_supported_os.each do |os, os_facts|
    context "on #{os}" do
      let(:facts) { os_facts }

      it { is_expected.to compile.with_all_deps }
    end
  end
end
```

Running the test suite

puppet-example's `spec/classes/example_spec`

```
$ bundle exec rspec --format documentation spec/classes/example_spec.rb

example
  on redhat-7-x86_64
    is expected to compile into a catalogue without dependency cycles
    is expected to contain File[/tmp/puppet-example] with content  supplied string
  on redhat-8-x86_64
    is expected to compile into a catalogue without dependency cycles
    is expected to contain File[/tmp/puppet-example] with content  supplied string

Code coverage
  must cover at least 0% of resources


Coverage Report:

Total resources:   1
Touched resources: 1
Resource coverage: 100.00%

Finished in 0.91092 seconds (files took 2.43 seconds to load)
5 examples, 0 failures
```

Acceptance testing

Acceptance testing

What

In engineering and its various subdisciplines, acceptance testing is a test conducted to determine if the requirements of a specification or contract are met. It may involve chemical tests, physical tests, or performance tests.

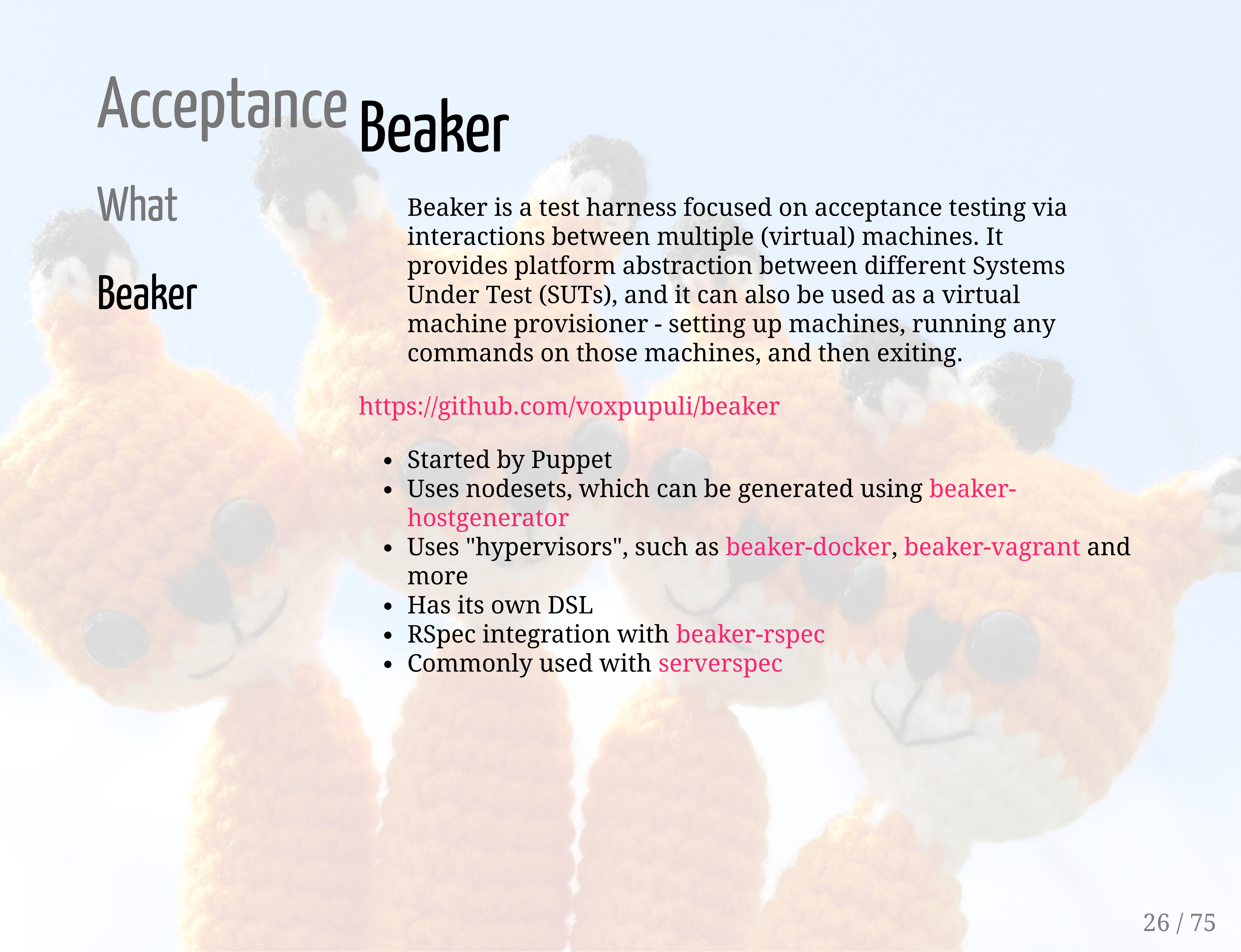
In systems engineering, it may involve black-box testing performed on a system (for example: a piece of software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery.

In software testing, the ISTQB defines acceptance testing as:

> Formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether to accept the system. — Standard Glossary of Terms used in Software Testing

https://en.wikipedia.org/wiki/Acceptance_testing

TL;DR: Real world tests



Beaker

Beaker is a test harness focused on acceptance testing via interactions between multiple (virtual) machines. It provides platform abstraction between different Systems Under Test (SUTs), and it can also be used as a virtual machine provisioner - setting up machines, running any commands on those machines, and then exiting.

https://github.com/voxpupuli/beaker

- Started by Puppet
- Uses nodesets, which can be generated using beaker-hostgenerator
- Uses "hypervisors", such as beaker-docker, beaker-vagrant and more
- Has its own DSL
- RSpec integration with beaker-rspec
- Commonly used with serverspec

Acceptance

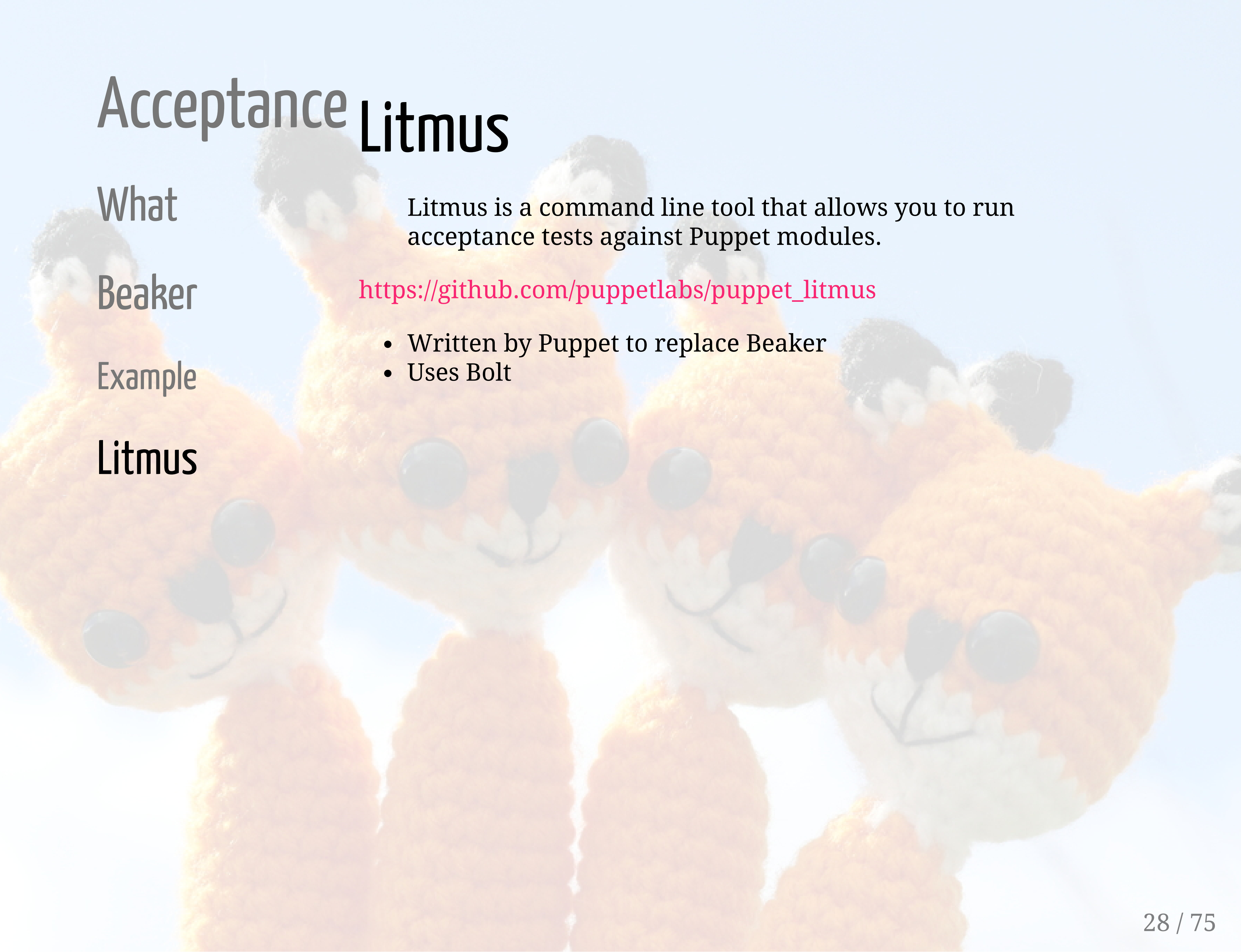
Beaker example

What

Beaker

Example

```
require 'spec_helper_acceptance'

describe 'example' do
  let(:manifest) { 'include example' }

  it 'applies successfully' do
    apply_manifest(manifest, catch_failures: true)
  end

  it 'applies idempotently' do
    apply_manifest(manifest, catch_changes: true)
  end

  it 'creates a file' do
    expect(file('/tmp/example')).to be_file
  end
end
```



Litmus

Litmus is a command line tool that allows you to run acceptance tests against Puppet modules.

https://github.com/puppetlabs/puppet_litmus

- Written by Puppet to replace Beaker
- Uses Bolt

Analytics collection is not normal

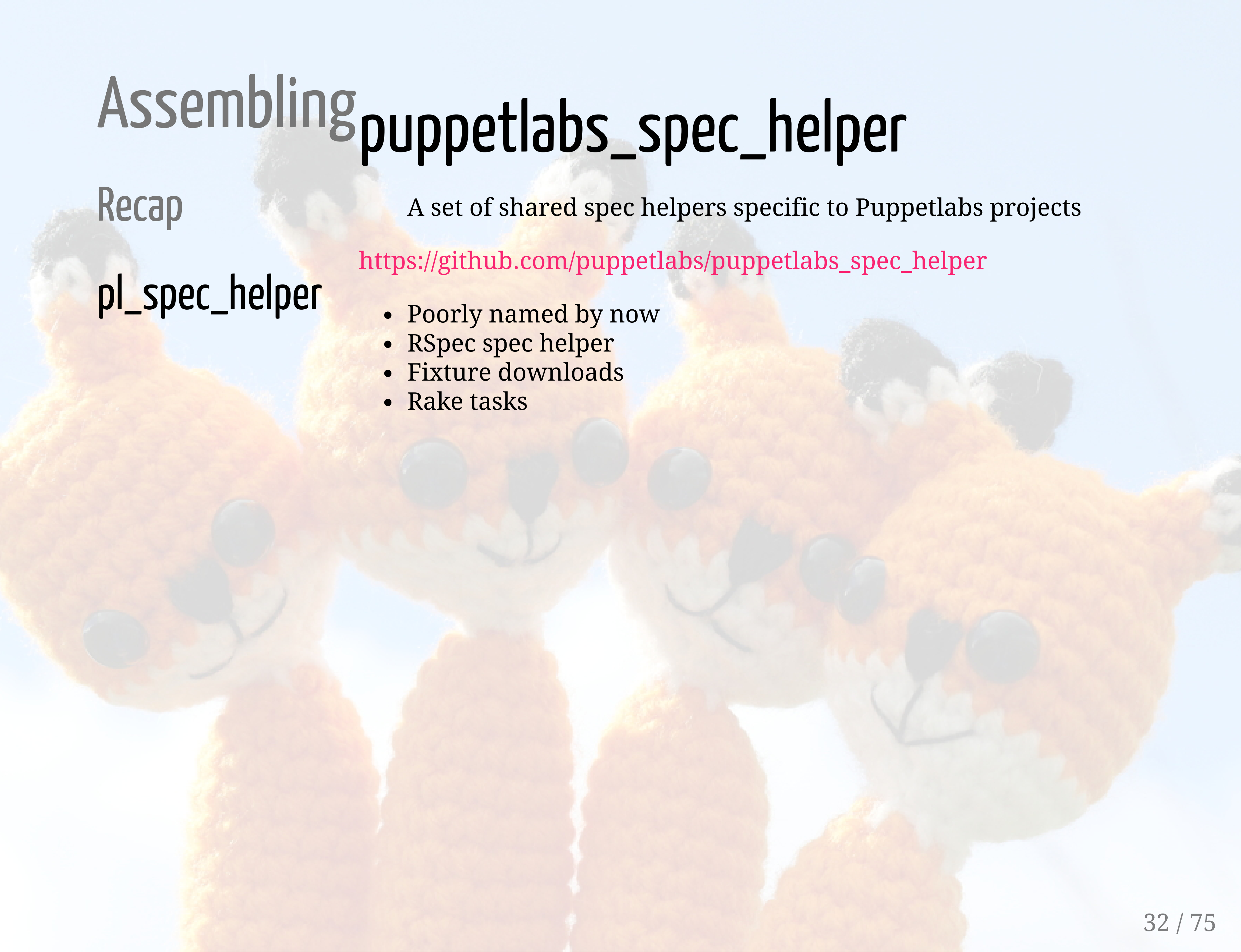
Don't pretend it is

Putting it together

Recapping what we just learned

Recap

- Static analysis
  - Syntax
  - Metadata
  - Lint
  - RuboCop
- Unit testing
  - RSpec
- Acceptance testing
  - RSpec



Assembling

puppetlabs_spec_helper

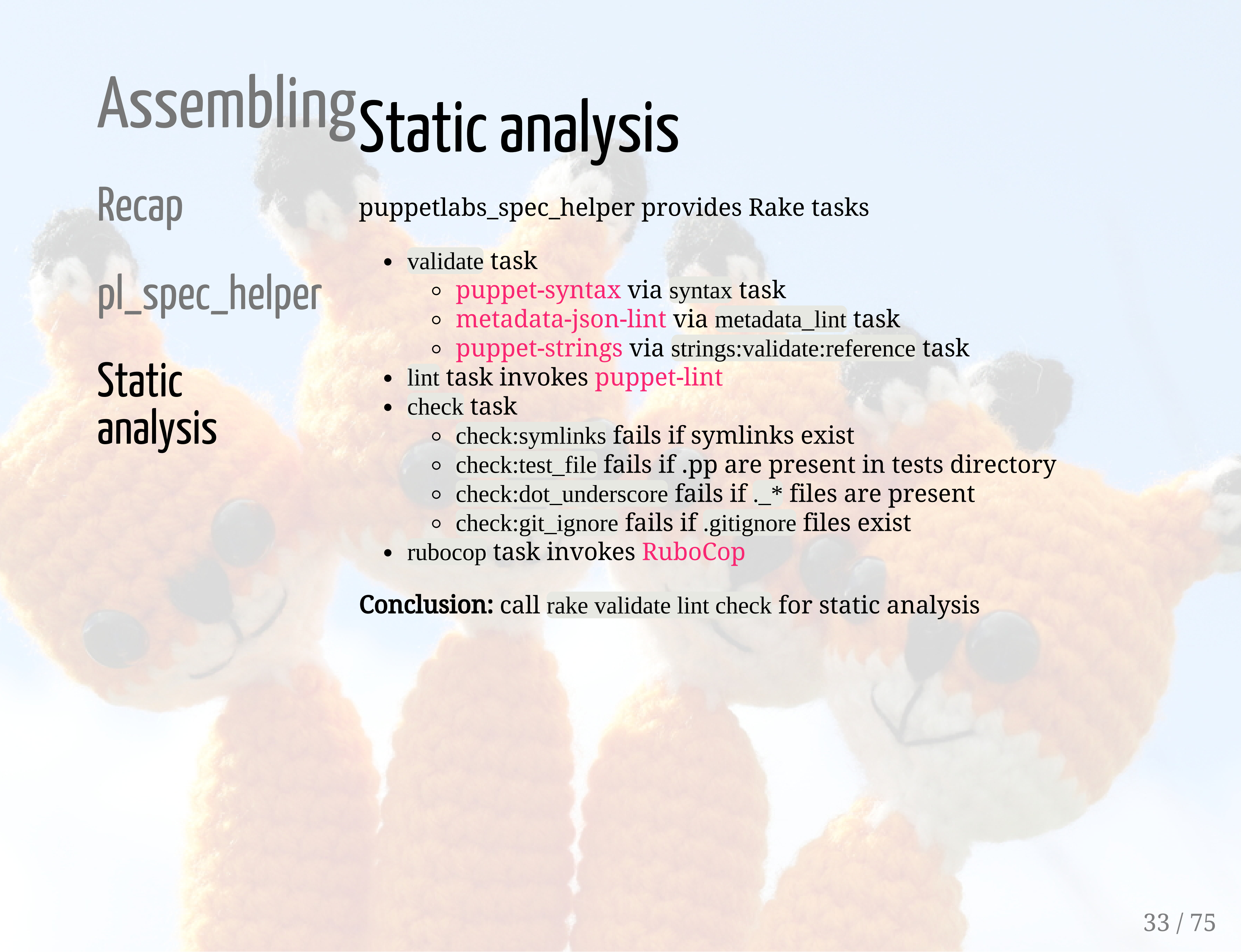
Recap

pl_spec_helper

A set of shared spec helpers specific to Puppetlabs projects

https://github.com/puppetlabs/puppetlabs_spec_helper

- Poorly named by now
- RSpec spec helper
- Fixture downloads
- Rake tasks

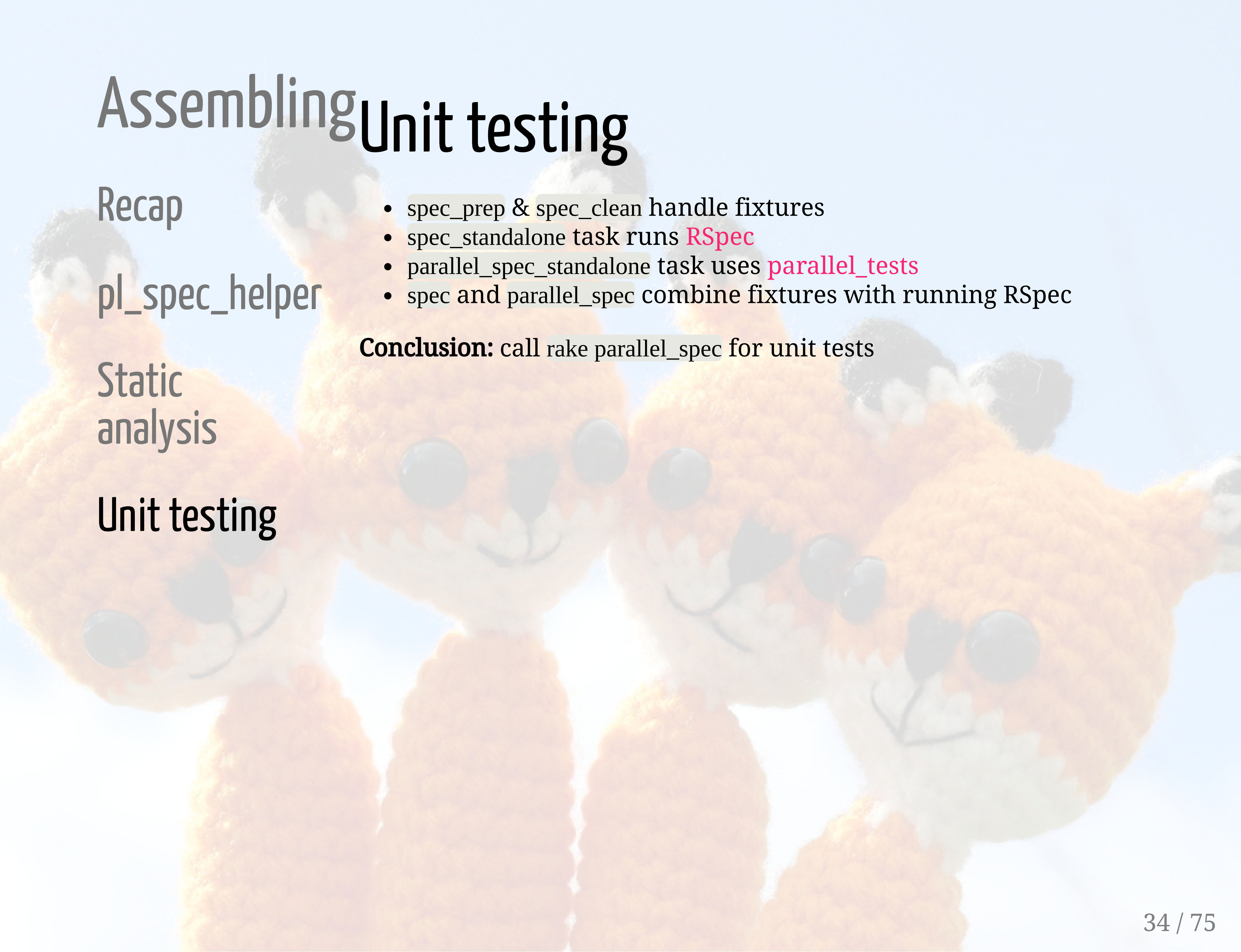


Static analysis

puppetlabs_spec_helper provides Rake tasks

- `validate` task
  - puppet-syntax via `syntax` task
  - metadata-json-lint via `metadata_lint` task
  - puppet-strings via `strings:validate:reference` task
- `lint` task invokes puppet-lint
- `check` task
  - `check:symlinks` fails if symlinks exist
  - `check:test_file` fails if .pp are present in tests directory
  - `check:dot_underscore` fails if ._* files are present
  - `check:git_ignore` fails if `.gitignore` files exist
- `rubocop` task invokes RuboCop

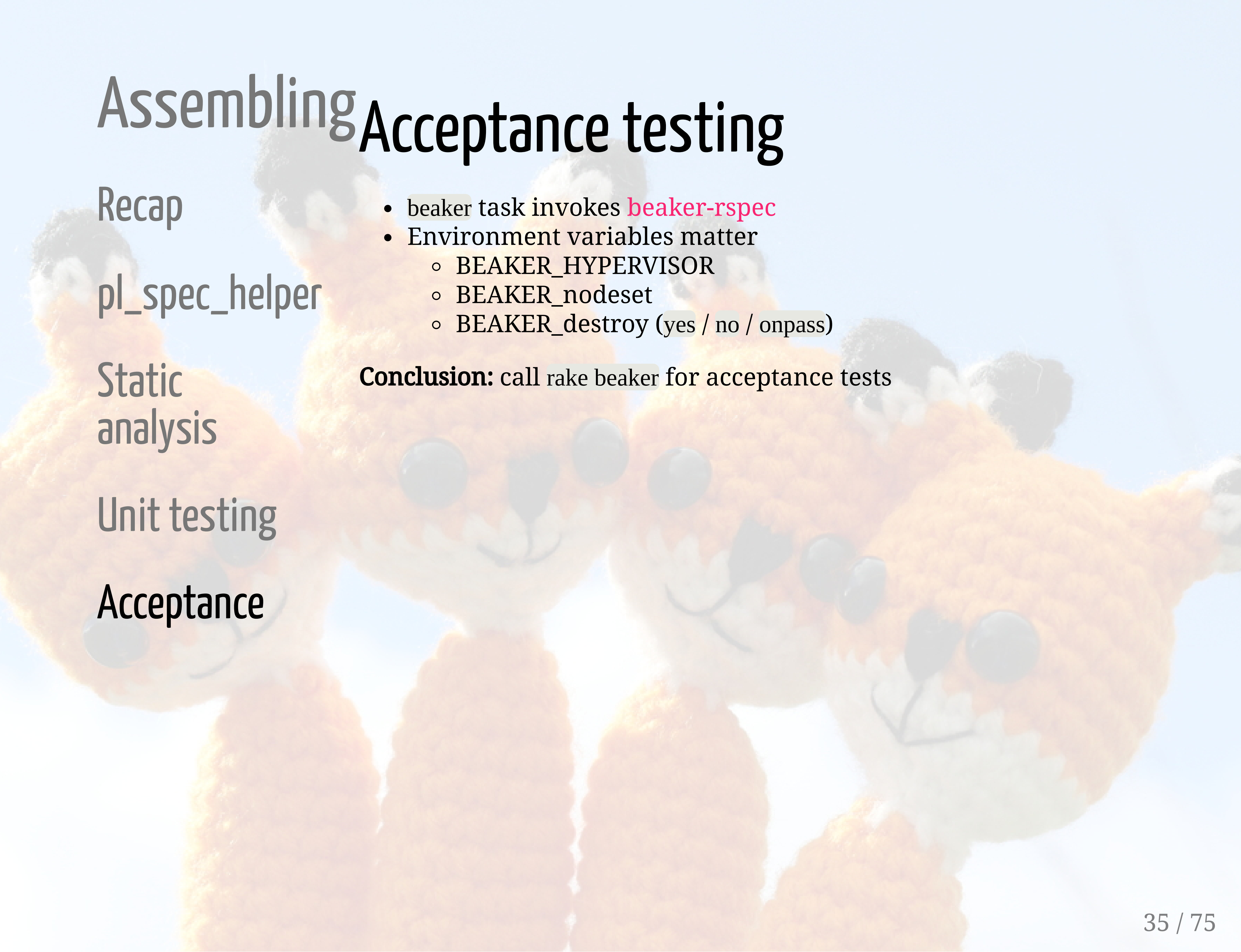
Conclusion: call `rake validate lint check` for static analysis



Unit testing

- `spec_prep` & `spec_clean` handle fixtures
- `spec_standalone` task runs RSpec
- `parallel_spec_standalone` task uses parallel_tests
- `spec` and `parallel_spec` combine fixtures with running RSpec

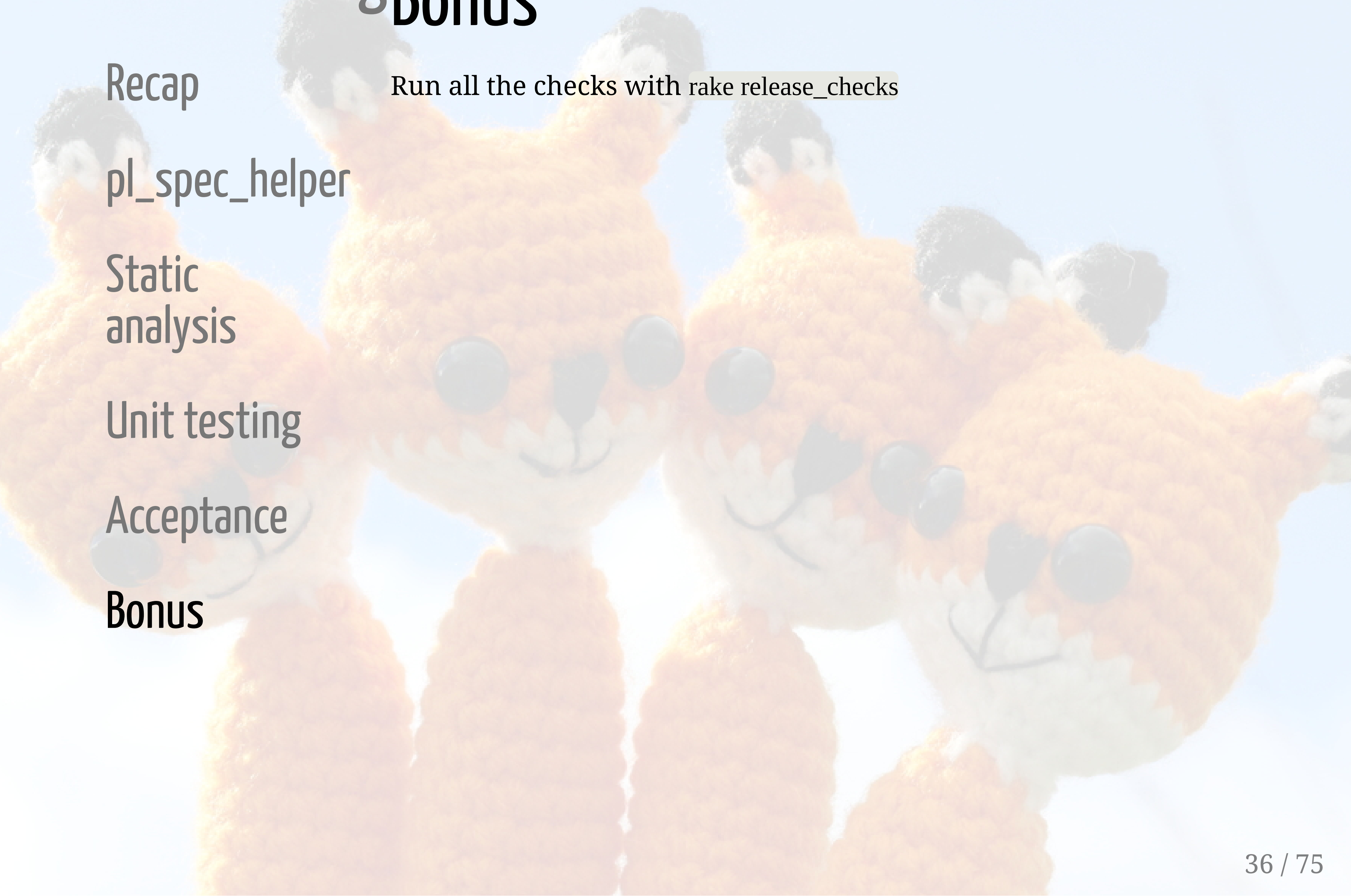
Conclusion: call `rake parallel_spec` for unit tests



Acceptance testing

- `beaker` task invokes beaker-rspec
- Environment variables matter
  - BEAKER_HYPERVISOR
  - BEAKER_nodeset
  - BEAKER_destroy (`yes` / `no` / `onpass`)

Conclusion: call `rake beaker` for acceptance tests



Bonus

Run all the checks with `rake release_checks`

GitHub Actions

GitHub Actions

What

> Automate, customize, and execute your software development workflows right in your repository with GitHub Actions. You can discover, create, and share actions to perform any job you'd like, including CI/CD, and combine actions in a completely customized workflow.

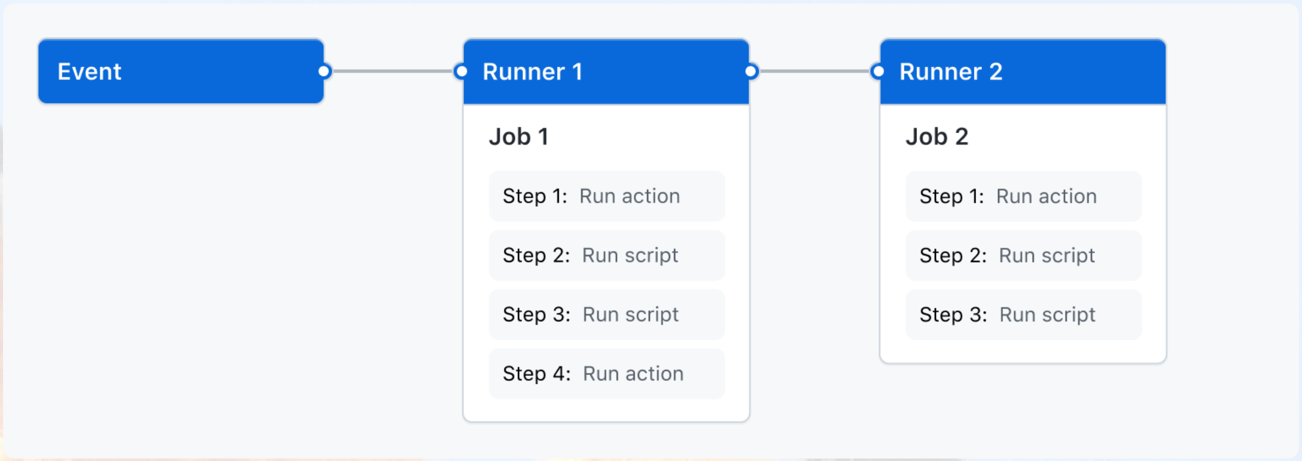
https://docs.github.com/en/actions

- Free up to a certain point
- Vox Pupuli is on a sponsored plan by GitHub
- Workflows in YAML
- Support for (dynamic) matrices

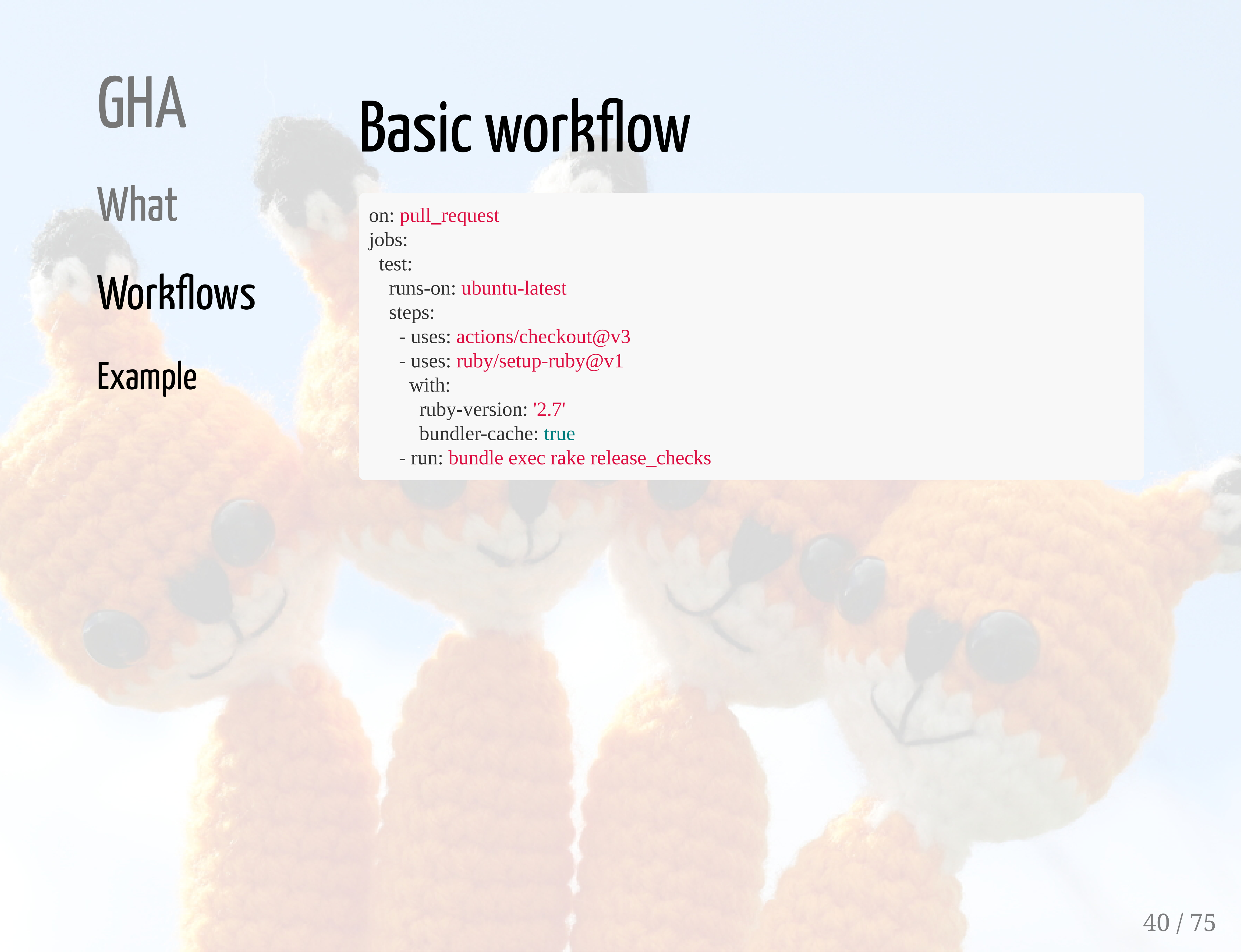
GHA

What

Workflows



https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions



Basic workflow

```
on: pull_request
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: ruby/setup-ruby@v1
        with:
          ruby-version: '2.7'
          bundler-cache: true
      - run: bundle exec rake release_checks
```

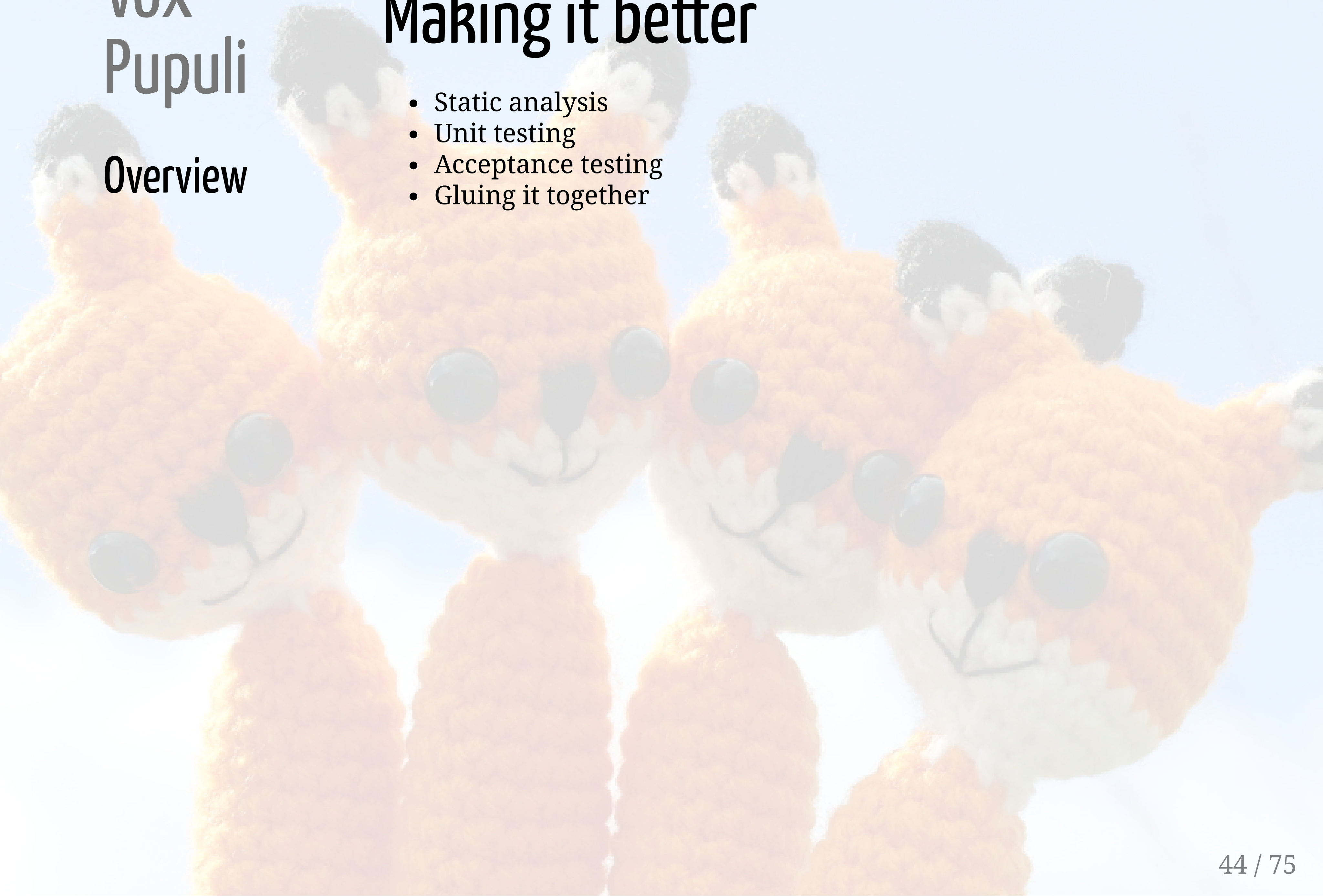
Welcome to the matrix

```
on:
 - pull_request
 - push
jobs:
 test:
   runs-on: ubuntu-latest
   strategy:
     matrix:
       ruby:
         - '2.5'
         - '2.7'
     fail-fast: false
   steps:
     - uses: actions/checkout@v3
     - uses: ruby/setup-ruby@v1
       with:
         ruby-version: ${{ matrix.ruby }}
         bundler-cache: true
     - run: bundle exec rake release_checks
```

Problems with this

- Stored in each repository is a lot of duplication
- Static in what it tests
- Haven't even touched acceptance testing

Vox Pupuli's "secret" sauce



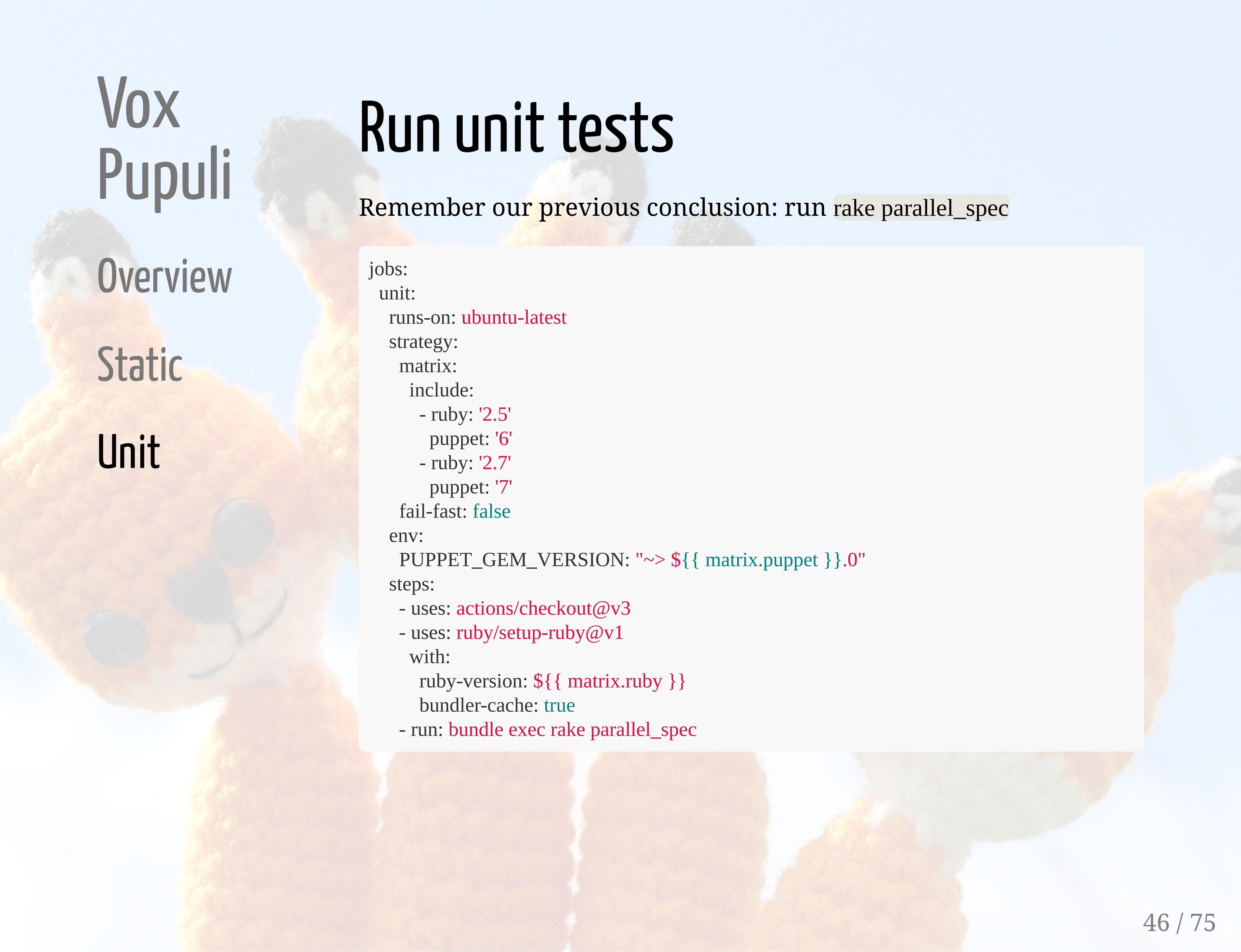
Making it better

- Static analysis
- Unit testing
- Acceptance testing
- Gluing it together

Run static validations

Remember our previous conclusion: run `rake validate check lint`

```
jobs:
  static:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: ruby/setup-ruby@v1
        with:
          ruby-version: '2.7'
          bundler-cache: true
      - run: bundle exec rake validate check lint
```



Run unit tests

Remember our previous conclusion: run `rake parallel_spec`

```
jobs:
  unit:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        include:
          - ruby: '2.5'
            puppet: '6'
          - ruby: '2.7'
            puppet: '7'
      fail-fast: false
    env:
      PUPPET_GEM_VERSION: "~> ${{ matrix.puppet }}.0"
    steps:
      - uses: actions/checkout@v3
      - uses: ruby/setup-ruby@v1
        with:
          ruby-version: ${{ matrix.ruby }}
          bundler-cache: true
      - run: bundle exec rake parallel_spec
```

Run acceptance tests

Remember our previous conclusion: run `rake beaker`

```
jobs:
  acceptance:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        puppet:
          - '6'
          - '7'
        setfile:
          - centos8
          - debian11
      fail-fast: false
    env:
      BEAKER_PUPPET_COLLECTION: "puppet${{ matrix.puppet }}"
      BEAKER_setfile: "${{ matrix.setfile }}-64"
    steps:
      - uses: actions/checkout@v3
      - uses: ruby/setup-ruby@v1
        with:
          ruby-version: '2.7'
          bundler-cache: true
      - run: bundle exec rake beaker
```

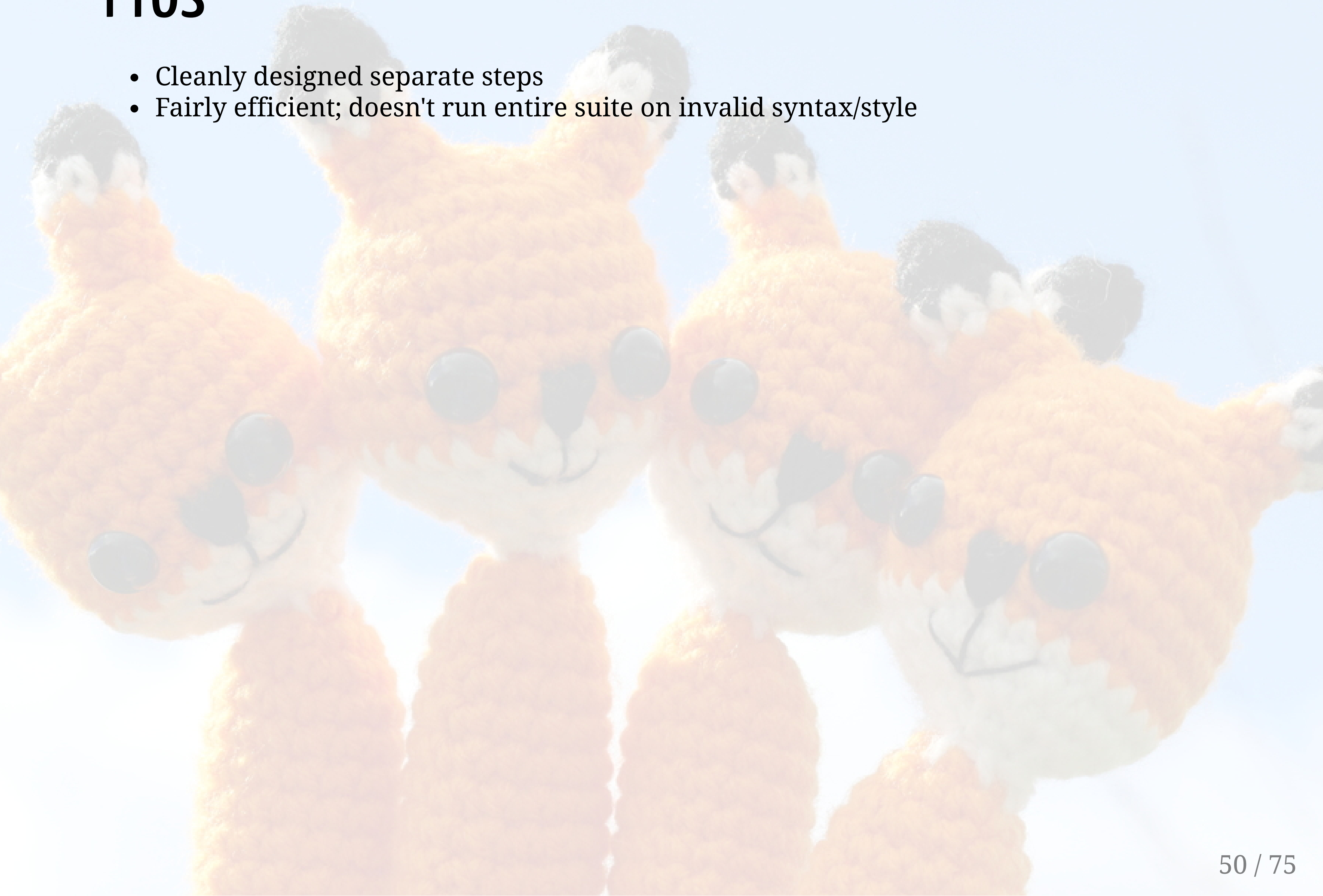
Putting all the pieces together

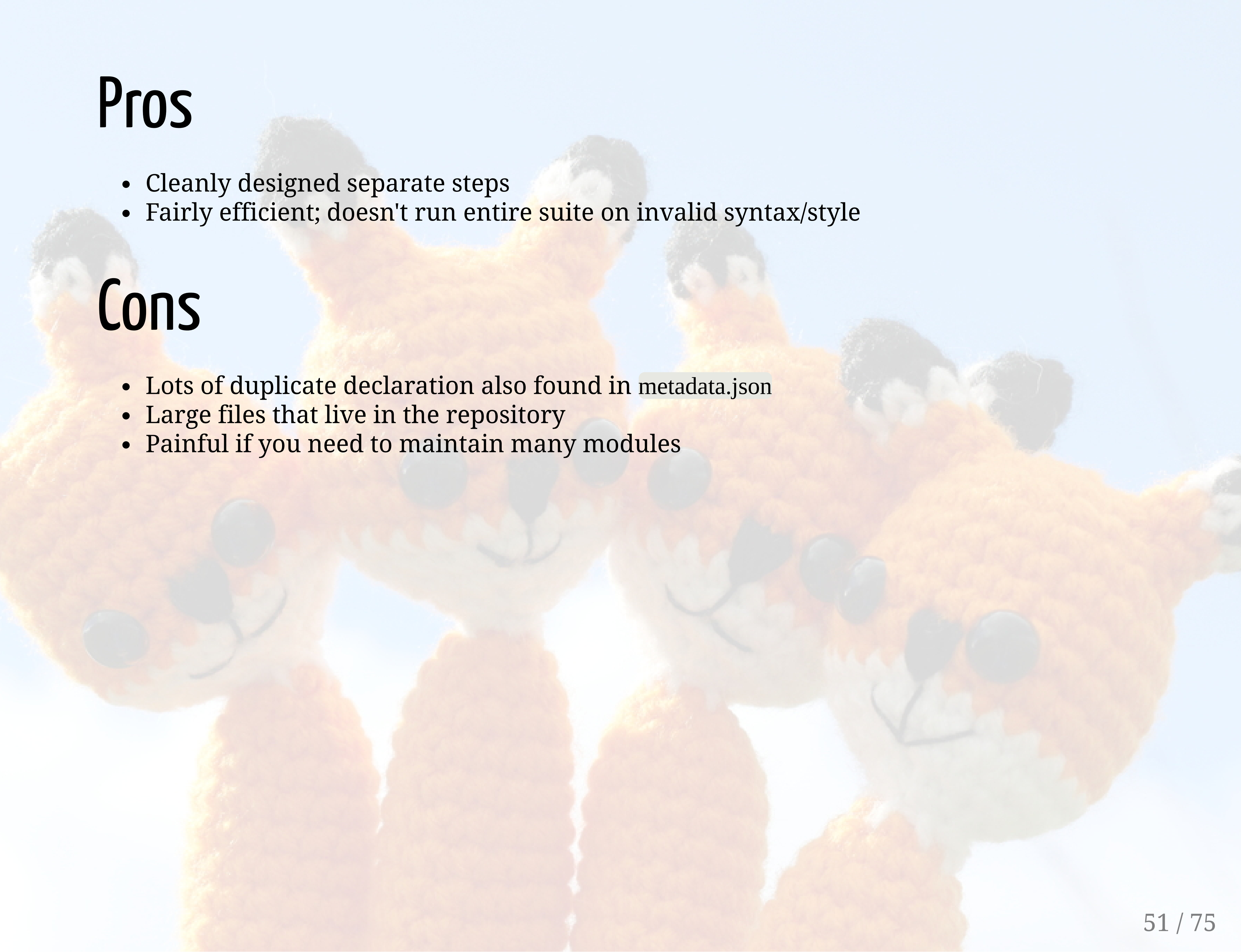
```
jobs:
  static:
    steps:
      - run: bundle exec rake syntax validate lint

  unit:
    needs: static
    steps:
      - run: bundle exec rake parallel_spec

  acceptance:
    needs: static
    steps:
      - run: bundle exec rake beaker

  tests:
    needs:
      - unit
      - acceptance
    steps:
      - run: echo Test suite completed
```

Are we happy?

Pros

- Cleanly designed separate steps
- Fairly efficient; doesn't run entire suite on invalid syntax/style





Pros

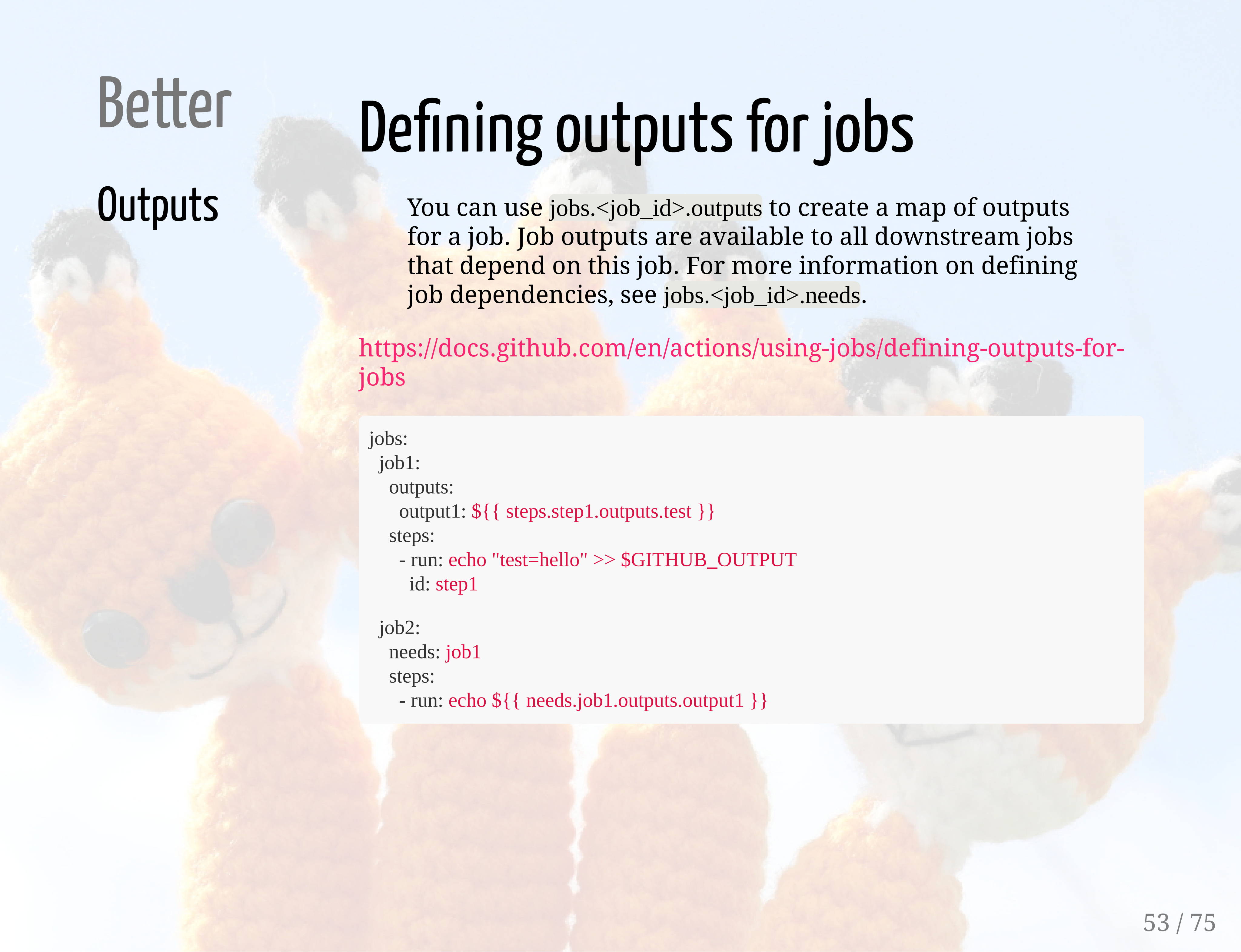
- Cleanly designed separate steps
- Fairly efficient; doesn't run entire suite on invalid syntax/style

Cons

- Lots of duplicate declaration also found in `metadata.json`
- Large files that live in the repository
- Painful if you need to maintain many modules

There must be a better way

- Raymond Hettinger



Outputs

Defining outputs for jobs

> You can use `jobs.<job_id>.outputs` to create a map of outputs for a job. Job outputs are available to all downstream jobs that depend on this job. For more information on defining job dependencies, see `jobs.<job_id>.needs`.

https://docs.github.com/en/actions/using-jobs/defining-outputs-for-jobs

```
jobs:
  job1:
    outputs:
      output1: ${{ steps.step1.outputs.test }}
    steps:
      - run: echo "test=hello" >> $GITHUB_OUTPUT
        id: step1

  job2:
    needs: job1
    steps:
      - run: echo ${{ needs.job1.outputs.output1 }}
```

Better

Outputs

puppet_metadata

https://docs.github.com/en/actions/using-jobs/defining-outputs-for-jobs

puppet_metadata

The gem intends to provide an abstraction over Puppet's metadata.json file. Its API allow easy iteration over its illogical data structures.

https://github.com/voxpupuli/puppet_metadata

```
$ metadata2gha
puppet_major_versions=[{"name":"Puppet 7","value":7,"collection":"puppet7"},{"name":"Pupp
puppet_unit_test_matrix=[{"puppet":7,"ruby":"2.7"},{"puppet":6,"ruby":"2.5"}]
github_action_test_matrix=[{"setfile":{"name":"Debian 11","value":"debian11-64"},"puppet":{
```

Better

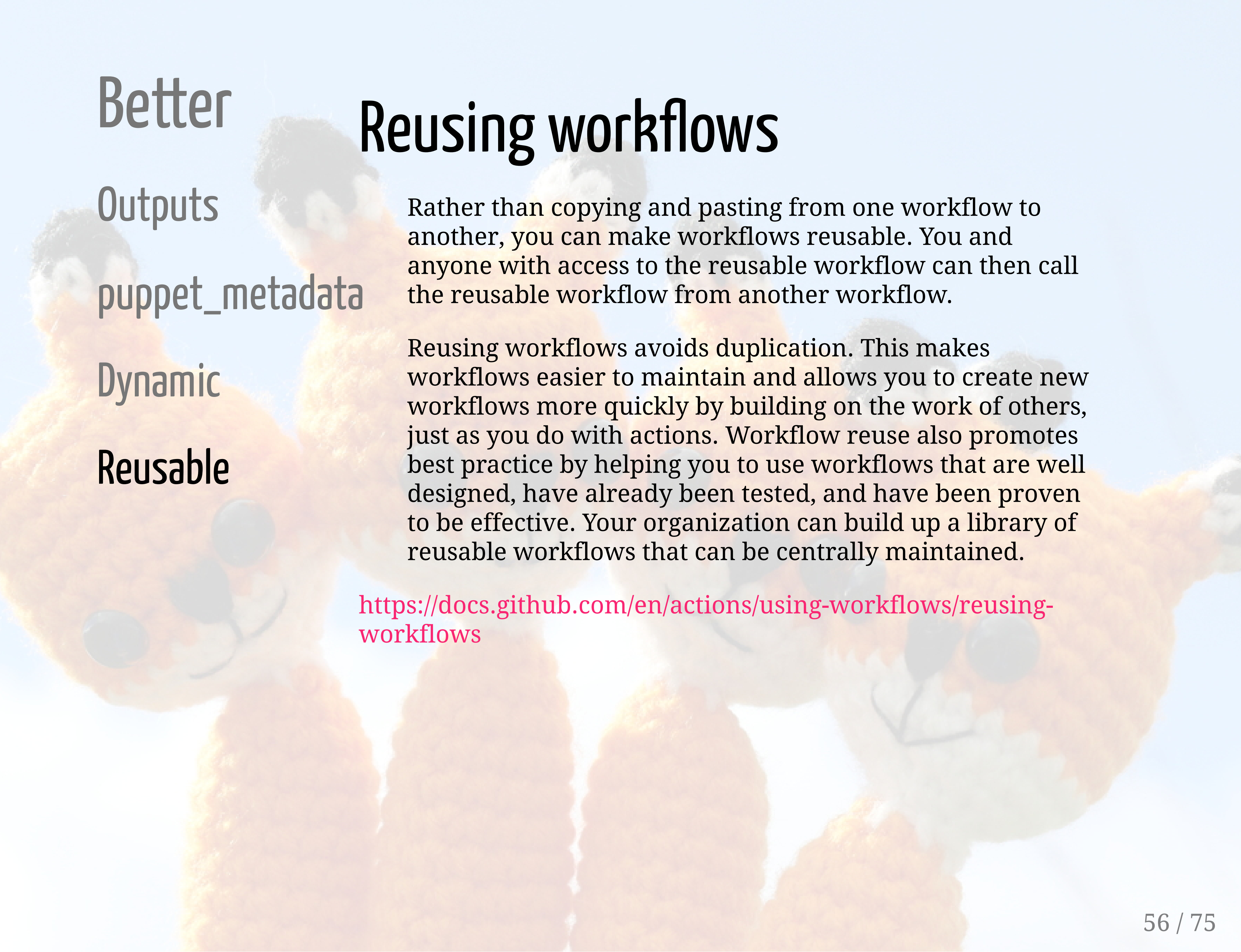
Outputs

puppet_metadata

Dynamic

Dynamic workflows with puppet_metadata

```
jobs:
  static:
    outputs:
      puppet_unit_test_matrix: ${{ steps.metadata.outputs.puppet_unit_test_matrix }}
    steps:
      - run: bundle exec rake syntax validate lint
      - run: bundle exec metadata2gha
        id: metadata

  unit:
    needs: static
    strategy:
      matrix:
        include: ${{fromJson(needs.static.outputs.puppet_unit_test_matrix)}}
    steps:
      - run: bundle exec rake parallel_spec

  tests:
    needs:
      - acceptance
    steps:
      - run: echo Test suite completed
```



Better

Outputs

puppet_metadata

Dynamic

Reusable

Reusing workflows

Rather than copying and pasting from one workflow to another, you can make workflows reusable. You and anyone with access to the reusable workflow can then call the reusable workflow from another workflow.

Reusing workflows avoids duplication. This makes workflows easier to maintain and allows you to create new workflows more quickly by building on the work of others, just as you do with actions. Workflow reuse also promotes best practice by helping you to use workflows that are well designed, have already been tested, and have been proven to be effective. Your organization can build up a library of reusable workflows that can be centrally maintained.

https://docs.github.com/en/actions/using-workflows/reusing-workflows

Puppet GitHub Actions

Reusable workflows to run Puppet tests within GitHub Actions.

https://github.com/voxpupuli/gha-puppet

- Provides both `basic` and `beaker` workflows
- Various options to tune behavior

Better

Outputs

puppet_metadata

Dynamic

Reusable

gha-puppet

Basic

Basic: static analysis and units

```
name: CI

on: pull_request

concurrency:
  group: ${{ github.ref_name }}
  cancel-in-progress: true

jobs:
  puppet:
    name: Puppet
    uses: voxpupuli/gha-puppet/.github/workflows/basic.yml@v1
```

Beaker: basic + acceptance

```yaml
name: CI

on: pull_request

concurrency:
  group: ${{ github.ref_name }}
  cancel-in-progress: true

jobs:
  puppet:
    name: Puppet
    uses: voxpupuli/gha-puppet/.github/workflows/beaker.yml@v1
```

Harder, Better, Faster, Stronger?

But wait, there's more

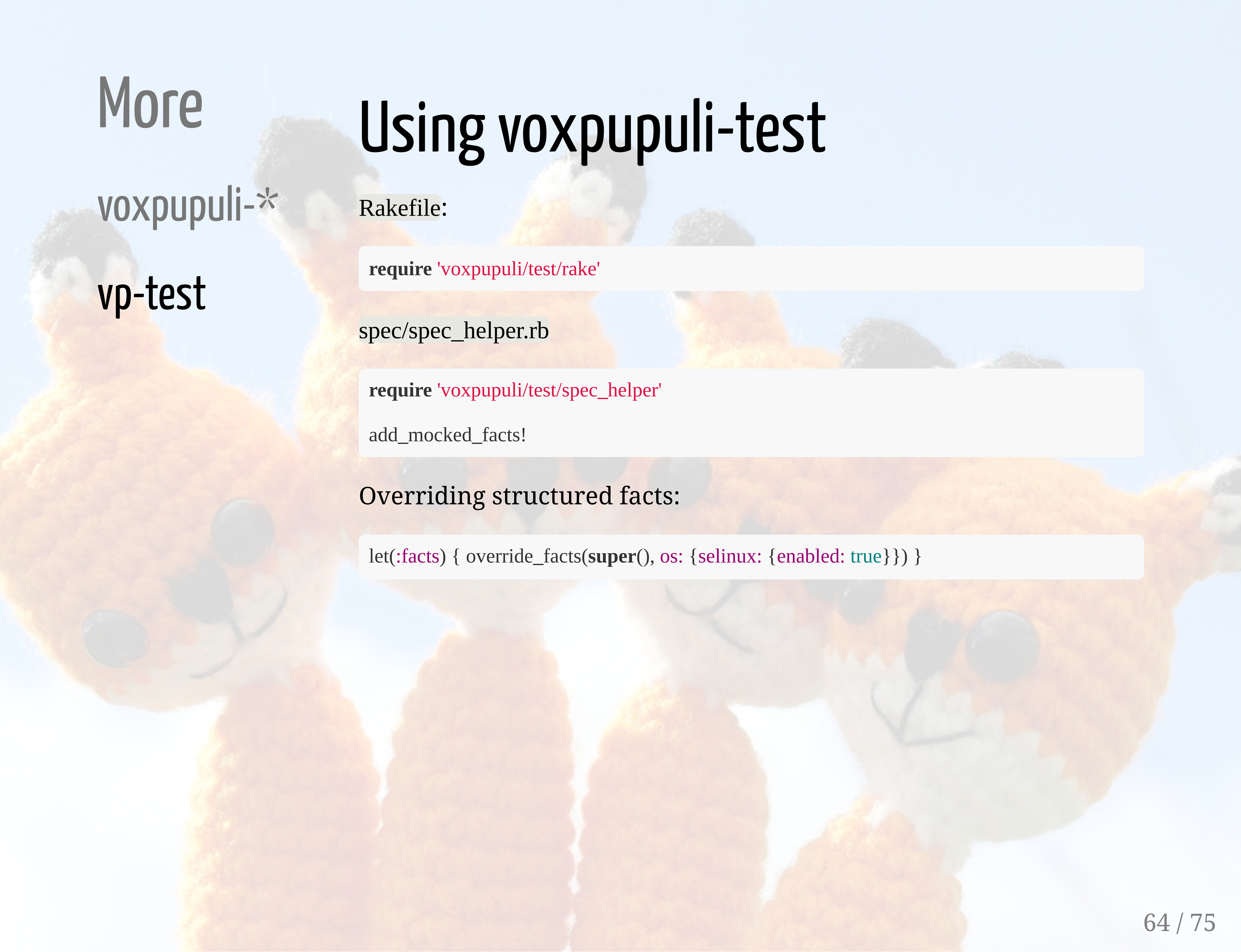
voxpupuli-test and voxpupuli-acceptance

> This is a helper Gem to test the various Vox Pupuli Puppet modules. This Gem provides common functionality for rspec-puppet based testing. The aim is to reduce the boiler plate and need for modulesync.

https://github.com/voxpupuli/voxpupuli-test

> This is a helper Gem to acceptance test the various Vox Pupuli Puppet modules using beaker. This Gem provides common functionality for all beaker based acceptance testing. The aim is to reduce the boiler plate and need for modulesync.

https://github.com/voxpupuli/voxpupuli-acceptance



Using voxpupuli-test

Rakefile:

```
require 'voxpupuli/test/rake'
```

spec/spec_helper.rb

```
require 'voxpupuli/test/spec_helper'

add_mocked_facts!
```

Overriding structured facts:

```
let(:facts) { override_facts(super(), os: {selinux: {enabled: true}}) }
```

Using voxpupuli-acceptance

Rakefile:

```
require 'voxpupuli/acceptance/rake'
```

spec/spec_helper_acceptance.rb

```
require 'voxpupuli/acceptance/spec_helper_acceptance'

configure_beaker
```

Module installation:

```
configure_beaker(modules: :metadata)
configure_beaker(modules: :fixtures)
```

Provide facts with BEAKER_FACTER_ environment variables:

```
$ BEAKER_FACTER_MYMODULE_VERSION=1.0 bundle exec rake beaker
```

Applies spec/setup_acceptance_node.pp

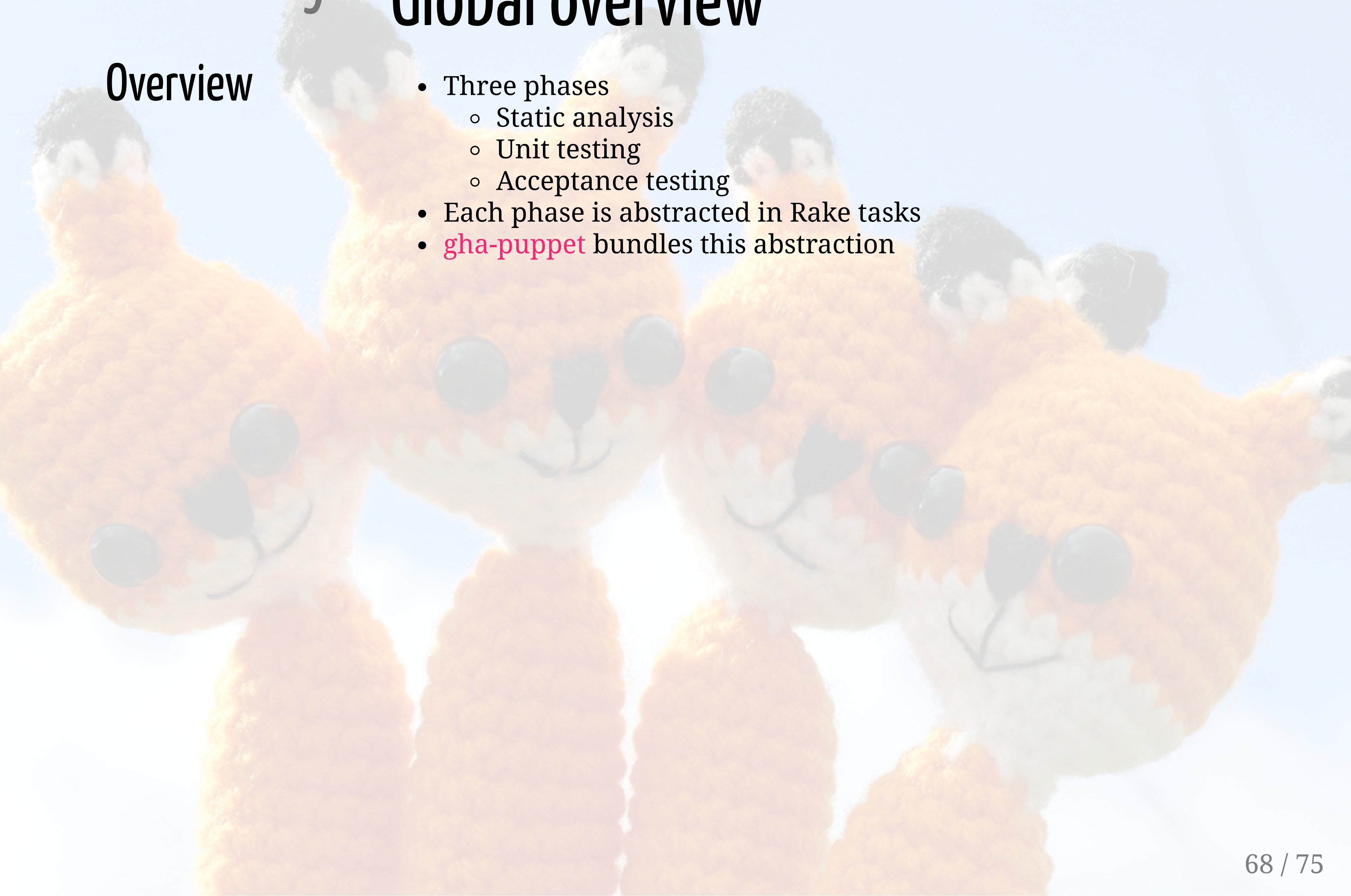
Are we there yet?

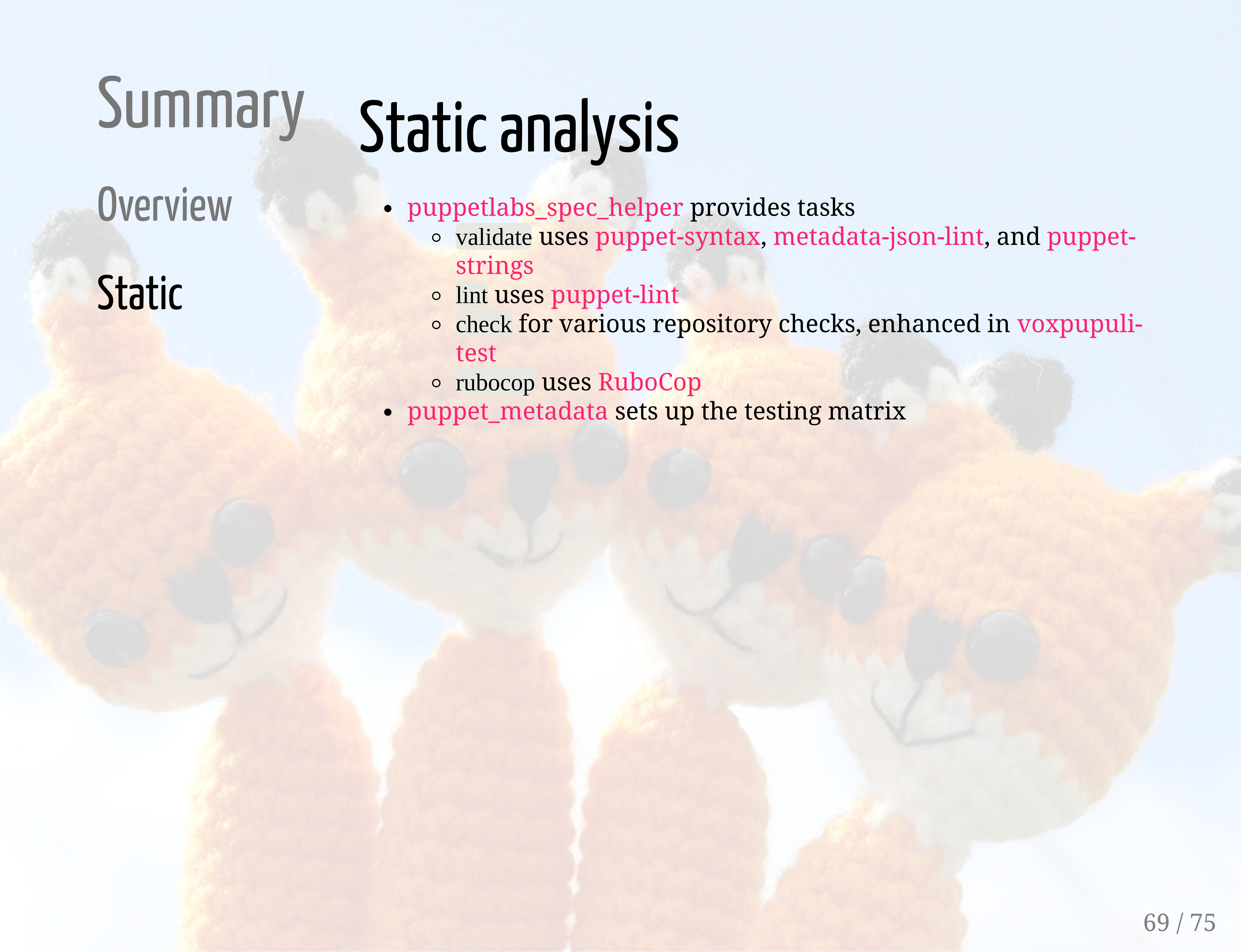
Summarizing

Global overview

Overview

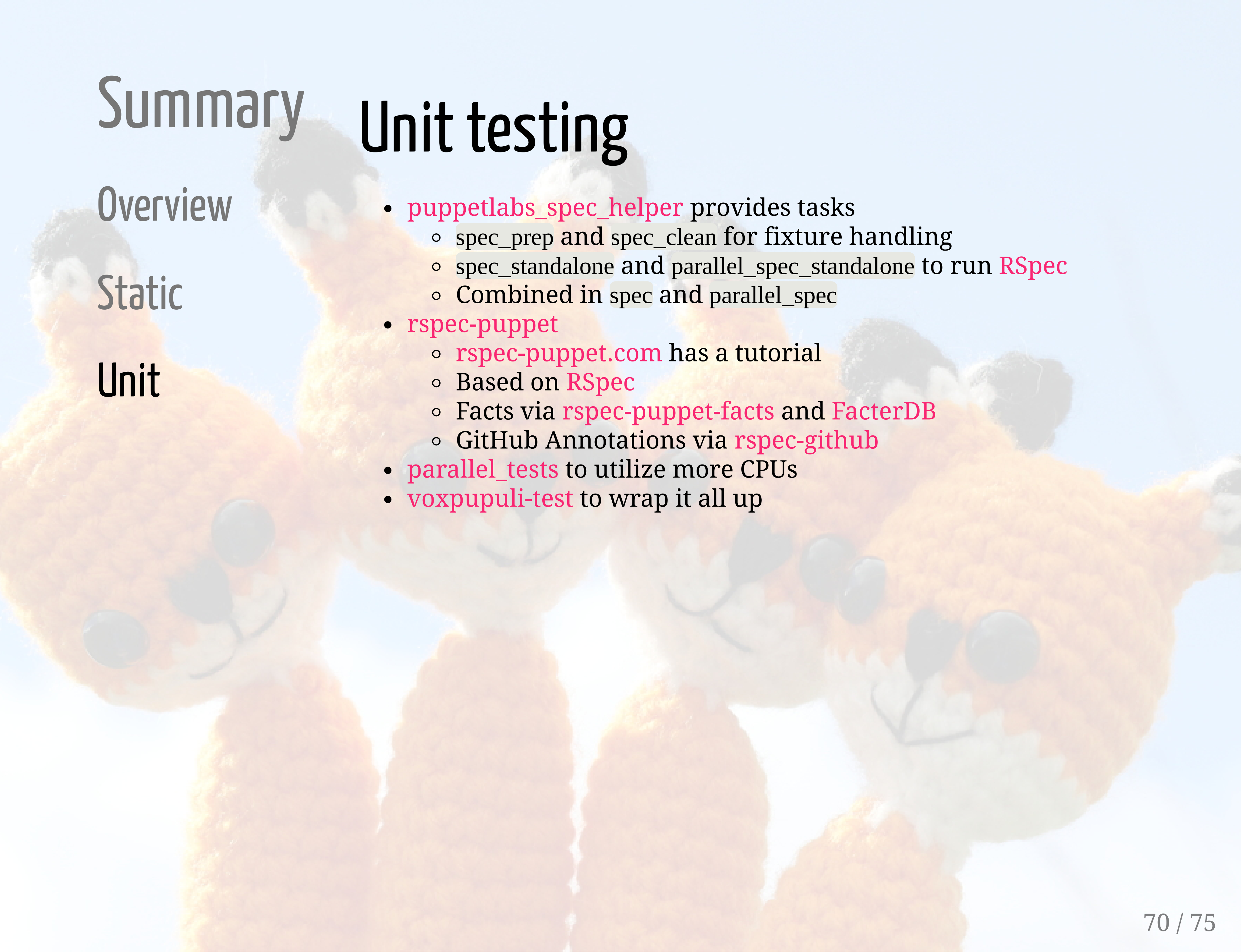
- Three phases
  - Static analysis
  - Unit testing
  - Acceptance testing
- Each phase is abstracted in Rake tasks
- gha-puppet bundles this abstraction





Static analysis

- puppetlabs_spec_helper provides tasks
  - `validate` uses puppet-syntax, metadata-json-lint, and puppet-strings
  - `lint` uses puppet-lint
  - `check` for various repository checks, enhanced in voxpupuli-test
  - `rubocop` uses RuboCop
- puppet_metadata sets up the testing matrix



Unit testing

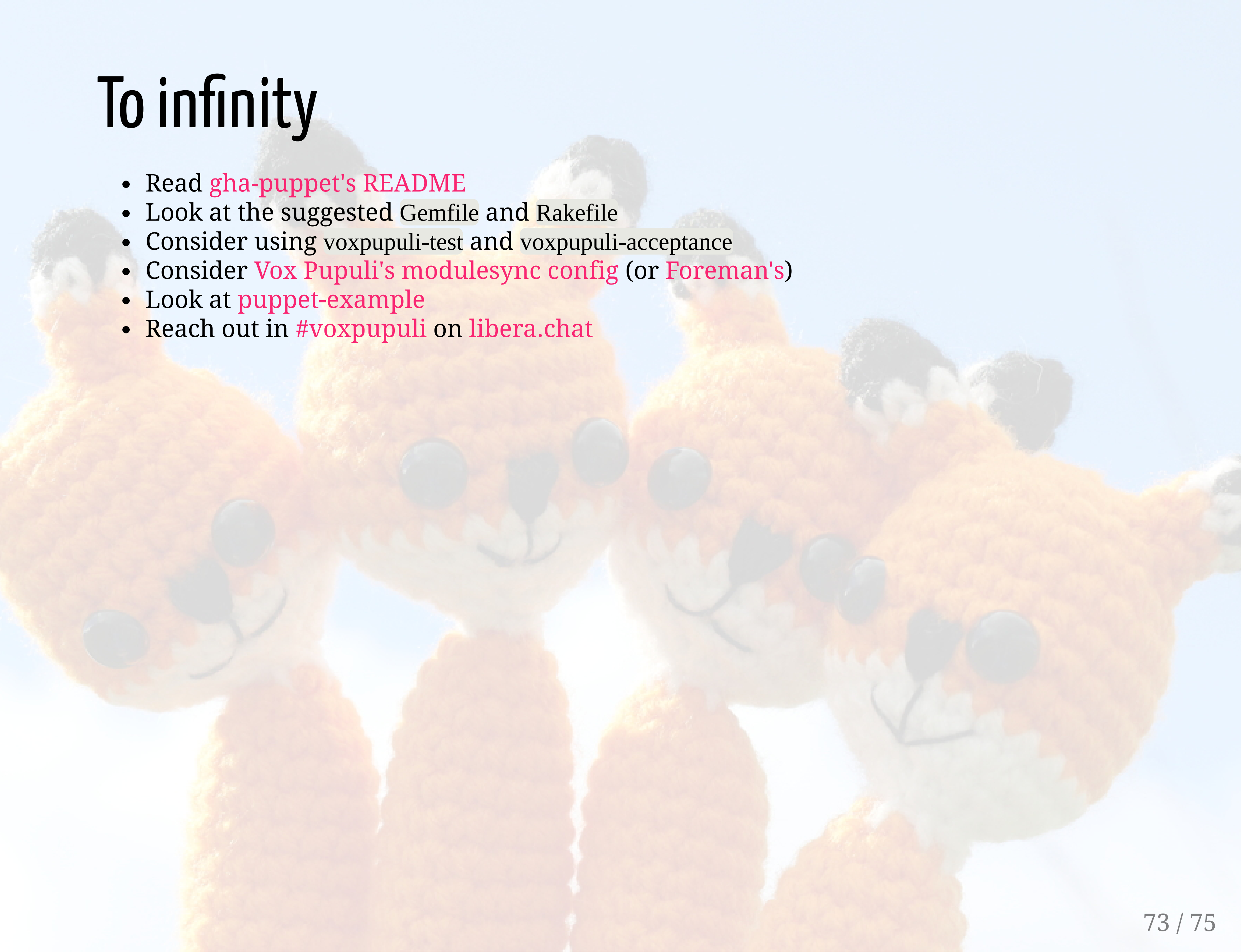
- puppetlabs_spec_helper provides tasks
  - `spec_prep` and `spec_clean` for fixture handling
  - `spec_standalone` and `parallel_spec_standalone` to run RSpec
  - Combined in `spec` and `parallel_spec`
- rspec-puppet
  - rspec-puppet.com has a tutorial
  - Based on RSpec
  - Facts via rspec-puppet-facts and FacterDB
  - GitHub Annotations via rspec-github
- parallel_tests to utilize more CPUs
- voxpupuli-test to wrap it all up

Acceptance testing

- Beaker based
  - RSpec integration via beaker-rspec
  - GitHub Annotations via rspec-github
  - Puppet helpers via beaker-puppet
  - Hypervisors like beaker-docker, beaker-vagrant, and more
  - Nodesets generated via beaker-hostgenerator
- Use serverspec to write expectations
- voxpupuli-acceptance to wrap it all up

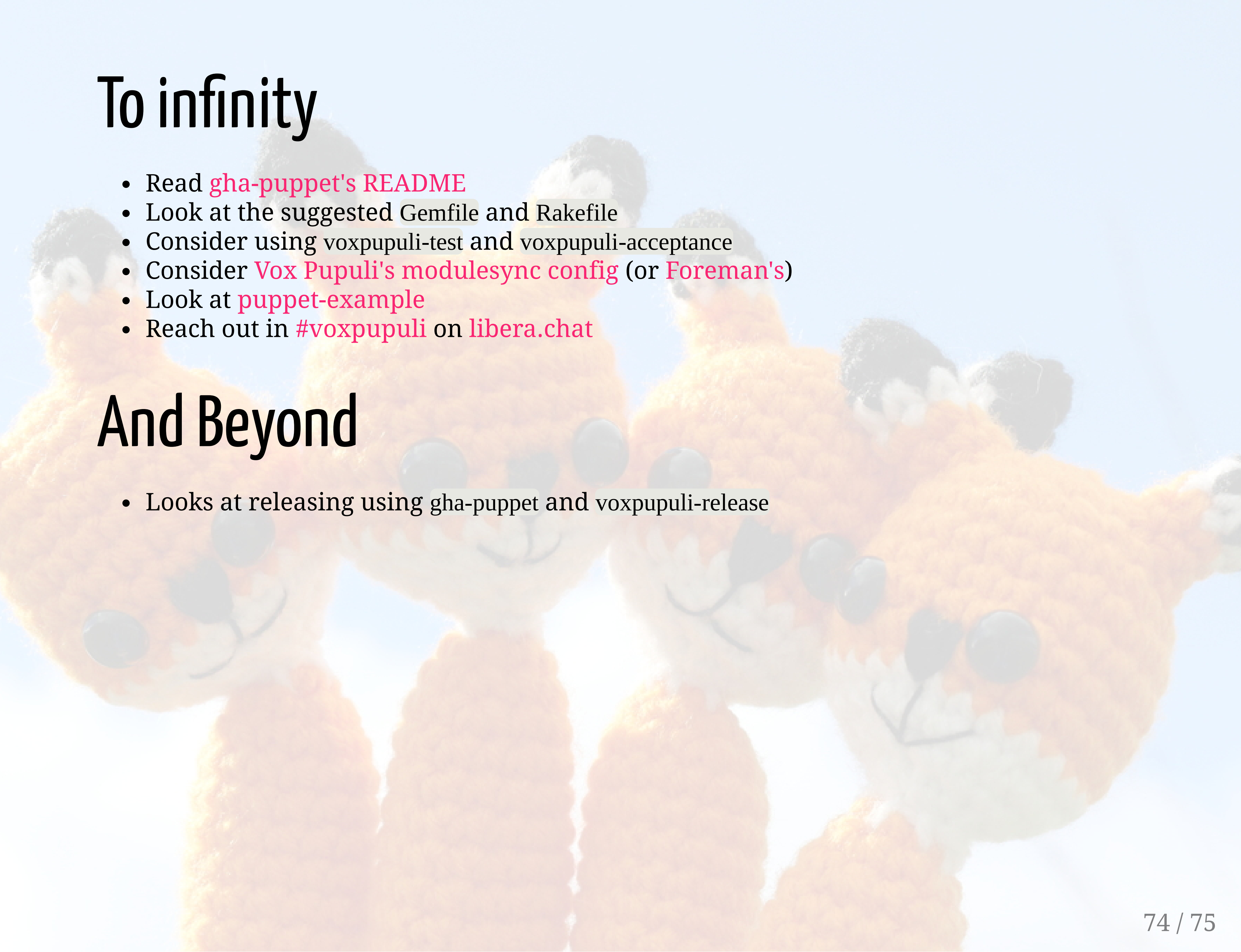
Using GitHub Actions to use it all

- gha-puppet provides reusable workflows



To infinity

- Read gha-puppet's README
- Look at the suggested `Gemfile` and `Rakefile`
- Consider using `voxpupuli-test` and `voxpupuli-acceptance`
- Consider Vox Pupuli's modulesync config (or Foreman's)
- Look at puppet-example
- Reach out in #voxpupuli on libera.chat



To infinity

- Read gha-puppet's README
- Look at the suggested `Gemfile` and `Rakefile`
- Consider using `voxpupuli-test` and `voxpupuli-acceptance`
- Consider Vox Pupuli's modulesync config (or Foreman's)
- Look at puppet-example
- Reach out in #voxpupuli on libera.chat

And Beyond

- Looks at releasing using `gha-puppet` and `voxpupuli-release` fin