**[InSpec - Inspect Your Infrastructure](https://blog.ragnarson.com/2017/03/13/inspec-inspect-your-infrastructure.html)**

We don't have many infrastructure related posts here. We have a few full-time DevOps, it's a part of our normal offer. But, we're busy most of the time. When an opportunity to write a blog post emerged, I knew I've had to write about a thing which changed our workflow. During last two years, there were few ideas which helped us providing better and more reliable services - one of them is infrastructure integration testing.

*Title is stolen from*[*inspec.io*](http://inspec.io/)*, I hope they don't mind.*

Before we start, few notes:

* This is only an introduction to the idea of infrastructure integration testing.
* It's opinionated.
* It's based on my two years experience with [Serverspec](http://serverspec.org/) and few weeks with InSpec.
* By infrastructure, I mean everything, from your local workstation to your >1024 machine cluster.
* Some ideas in this blog post may be closer to acceptance or functional testing. I'm aware of this, it's a thin line in some cases.
* I will omit some InSpec features.
* Tests are hard.

**Why not unit testing?**

In our case, unit testing is strongly coupled with Chef and ChefSpec, which means this blog post would apply to the much narrower audience.

Besides, it's easy to [overtest](https://coderanger.net/overtesting/) in the case of unit tests, where only true logic should be verified.

Integration tests are closer to the infrastructure and I'll focus on InSpec for now.

**What is InSpec?**

*InSpec is an open-source testing framework for infrastructure with a human-readable language for specifying compliance, security and other policy requirements.*

This sentence is borrowed from [inspec.io](http://inspec.io/). Is it true? Let's start with short code sample:

describe file('/etc/myapp.conf') **do**

it { should exist }

its ('mode') { should cmp '0644' }

**end**

describe port(8080) **do**

it { should be\_listening }

**end**

It consists two resources (file and port) with basic matchers verifying file existence, proper mode and listening port. Resources are classes which help with verifying various aspects of infrastructure.

And the output of execution:

$ inspec exec foo.rb

Target: local://

File /etc/myap.conf

✔ should exist

✔ mode should cmp **==** "0644"

Port 8080

∅ should be listening

expected `Port 8080.listening?` to **return** true, got false

Test Summary: 2 successful, 1 failures, 0 skipped

Is it readable? Well, it is, with basic English knowledge it shouldn't be too hard to understand what source code describes and what is the result. It can specify compliance rules and policy requirements.

Also, the code is very similar to RSpec, which is used as the underlying foundation of InSpec. One *novelty* may be the *cmp* matcher. It helps to compare file modes, string, single element array strings and numbers. You can read more about cmp in [InSpec documentation](http://inspec.io/docs/reference/matchers/).

The tool itself doesn't depend on any particular operating system. You can develop your code on almost anything which runs Ruby. For the best experience, you may want to use[ChefDK](https://github.com/chef/chef-dk). Most of the resources will work with common tools available on any system.

I should mention that InSpec is developed by Chef Software Inc., but it's isn't tied to Chef it can be used without it. You don't even have to like Chef (but why wouldn't you?).

**Why InSpec?**

Why not $(another framework)? I assume you have a favorite integration testing tool, so I'll try to convince you to use InSpec.

1. It's open source. I know most of the tools are, but it's a good start.
2. Its development is supported by Chef Software Inc., it probably won't disappear like some nodejs libs in the past and it won't be abandoned.
3. Awesome community. If you have any issues, you can always ask for help on [Chef slack's InSpec channel](http://community-slack.chef.io/), core maintainers are frequently the first responders.
4. Resource-rich. When I'm writing this, there are [over 60 resources](http://inspec.io/docs/reference/resources/) available and this number is growing. If you need something unusual you can always use file/command resources or contribute to InSpec.
5. Can run anywhere. From your local workstation, you can verify machine over ssh, docker or winrm.
6. $ inspec exec test.rb -t ssh://user@hostname
7. $ inspec exec test.rb -t winrm://Administrator@windowshost --password 'your-password'
8. $ inspec exec test.rb -t docker://container\_id
9. It has an interactive shell. I'm so used to Chef Shell, I don't know how I was able to write integration tests without inspec shell.
10. $ inspec shell
11. Welcome to the interactive InSpec Shell
12. To find out how to use it, type: help
13. inspec> describe file**(**'/home'**)** **do**
14. inspec> it **{** should exist **}**
15. inspec> end
16. Profile: inspec-shell
17. Version: unknown
18. File /home
19. ✔ should exist
20. Test Summary: 1 successful, 0 failures, 0 skipped
21. [It works on most popular operating systems](https://github.com/chef/inspec#supported-os). On less popular too!
22. Ruby. As Serverspec/RSpec derivative, it's written in Ruby. If I need to, I can easily jump into the source and verify any doubts.

**How to use InSpec?**

First, we'll focus on possible scenarios.

1. We have this awesome automation tool, it does things. Does it? Most basic usage, verify your provisioner - Chef, Puppet, Salt, Ansible, shell scripts.
2. Our app/code has to run in various environments in a mix of conditions. Multiple Linux distributions, Docker, Windows.
3. Operating system migration. It's similar to previous one, but it's worth separate mention. Ubuntu 14.04 LTS will be supported until April 2019. Get ahead and start testing your provisioning on next LTS before it will be too late.
4. Security compliance. There are sets of InSpec profiles linked on [Chef Supermarket](https://supermarket.chef.io/tools?type=compliance_profile)which can help you verify if your operating system is running under sane security settings.
5. Cheap reporting. Describe your whole infrastructure in InSpec, run it periodically and store the reports for better times. Another department may ask for it, someone may ask if your environments are really the same, you may need to prepare for an external audit.

**Examples**

Time for some coding. I would like to show very basic examples of usage. Everything I'll show is available at [a Git repository](https://gitlab.com/szymon.szypulski/inspec-blog-post).

**Tools**

For running this demo I'll use a small cookbook which will install web/database server, everything will be tied with [Test Kitchen](http://kitchen.ci/). For running the code you will need Ruby with Bundler or you can use ChefDK.

As mentioned, the whole thing will be driven by Test Kitchen. Everything is configured in[.kitchen.yml](https://gitlab.com/szymon.szypulski/inspec-blog-post/blob/master/.kitchen.yml).

---

driver:

name: vagrant

provision: true

vm\_hostname: inspec.ragnarson.com

provisioner:

name: chef\_zero

client\_rb:

node\_name: inspec.ragnarson.com

require\_chef\_omnibus: "12.18.31"

verifier:

name: inspec

platforms:

- name: ubuntu-14.04

- name: ubuntu-16.04

Vagrant will be our VM provider. Chef will provision the recipes. Everything will be verified by InSpec and we've two versions of Ubuntu at our disposal. List of all instances used for testing is available after executing kitchen list:

$ kitchen list

Instance Driver Provisioner Verifier Transport Last Action Last Error

web-ubuntu-1404 Vagrant ChefZero Inspec Ssh <Not Created> <None>

web-ubuntu-1604 Vagrant ChefZero Inspec Ssh <Not Created> <None>

database-ubuntu-1604 Vagrant ChefZero Inspec Ssh <Not Created> <None>

ssh-hardening-ubuntu-1604 Vagrant ChefZero Inspec Ssh <Not Created> <None>

**Web**

Let's start with ‘web' server. [Code is very simple, get me nginx](https://gitlab.com/szymon.szypulski/inspec-blog-post/blob/master/recipes/web.rb).

package 'nginx'

First, we've to provision instance by running kitchen converge web-ubuntu-1604. Instance names are a combination of test suites and available platforms. Under suites in*.kitchen.yml* we've a definition of *web* suite.

suites:

- name: web

run\_list:

- recipe[inspec-blog-post::web]

While the machine is provisioning we can take a look at test file located in test/integration/<suite name>/web\_spec.rb (there may be multiple test files for one suite). We'll verify few things.

1. Is package installed?
2. describe package('nginx') **do**
3. it { should be\_installed }
4. **end**
5. Is service running? If we would have monit installed we could also check if it's monitored.
6. describe service('nginx') **do**
7. it { should be\_running }
8. *# it { should be\_monitored.by("monit") }*
9. **end**
10. Is Nginx configuration file present and it has proper ssl\_protocols set.
11. describe file('/etc/nginx/nginx.conf') **do**
12. it { should exist }
13. its(:content) { should match(/ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;/) }
14. **end**
15. Our special CVE-2016-1247 case, on some systems it was possible to escalate from www-data to root user when log directory was owned by the www-data user. We can test against this in InSpec.
16. *# CVE-2016-1247*
17. describe file('/var/log/nginx') **do**
18. it { should\_not be\_owned\_by('www-data') }
19. **end**

To verify if our server does follow the rules run kitchen verify web-ubuntu-1604. You should see all green output, our instance state is matching our expectations.

System Package

✔ nginx should be installed

Service nginx

✔ should be running

File /etc/nginx/nginx.conf

✔ should exist

✔ content should match /ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;/

File /var/log/nginx

✔ should not be owned by "www-data"

Test Summary: 5 successful, 0 failures, 0 skipped

Let's see what will happen if we'll run the same test suite on Ubuntu 14.04. Execute kitchen verify web-ubuntu-1404. If the instance isn't converged it will be, as part of verify task.

...

∅ content should match /ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;/

...

Test Summary: 4 successful, 1 failures, 0 skipped

The command should fail, because Ubuntu 14.04 doesn't have default SSL protocols list, well it doesn't even have SSL enabled. I know it's a trivial test, but it shows how you can use InSpec.

**Database**

Our *database* recipe is quite simple as well, it should just install PostgreSQL.

package 'postgresql'

Lets verify our test suite kitchen verify database-ubuntu-1604. In the meantime, look at our test suite code.

1. As previously, is package installed?
2. describe package('postgresql') **do**
3. it { should be\_installed }
4. **end**
5. Does it run?
6. describe service('postgresql') **do**
7. it { should be\_running }
8. **end**
9. For PostgreSQL, there is the *postgres\_conf* resource which is more convenient than regular expression used for *nginx.conf*.
10. describe postgres\_conf **do**
11. its('max\_connections') { should eq '100' }
12. **end**
13. Is daemon running on proper host and port?
14. describe port(5432) **do**
15. it { should be\_listening }
16. its('addresses') { should\_not **include** '0.0.0.0' }
17. its('protocols') { should **include**('tcp') }
18. **end**
19. In this case, there should be no service running on ports between 22 and 80. Block can be passed to resource and code looks clean.
20. describe port.**where** { protocol **=~** /tcp/ **&&** port **>** 22 **&&** port **<** 80 } **do**
21. it { should\_not be\_listening }
22. **end**

The summary should be *all green*.

Test Summary: 7 successful, 0 failures, 0 skipped

**Profiles**

There is one InSpec feature I didn't write much about. There are [publicly available InSpec profiles](https://supermarket.chef.io/tools?type=compliance_profile) which describe best configuration practices for specific services. In our test suites as an example, I've used [ssh-hardening] developed as part of the dev-sec group.

If you are using external profile, Test Kitchen configuration is a little bit different.

...

- name: ssh-hardening

includes:

- ubuntu-16.04

verifier:

inspec\_tests:

- name: ssh-hardening

url: https://github.com/dev-sec/tests-ssh-hardening/archive/master.tar.gz

...

If you'll run verify, you will see how ssh and sshd are configured by default on Ubuntu 16.04.

$ kitchen verify ssh-hardening-ubuntu-1604

You'll see lots of warnings, mostly because Ubuntu relies on sane defaults and it doesn't specify most of the options explicitly. But you can find some interesting things in kitchen output.

...

× sshd-06: Server: Do not permit root-based login or **do** not allow password and keyboard-interactive authentication **(**expected "prohibit-password" to match /no|without-password/

Diff:

@@ -1,2 +1,2 @@

-/no|without-password/

+"prohibit-password"

**)**

× SSH Configuration PermitRootLogin should match /no|without-password/

expected "prohibit-password" to match /no|without-password/

Diff:

@@ -1,2 +1,2 @@

-/no|without-password/

+"prohibit-password"

...

By default there is no root password on Ubuntu, therefore it isn't possible to log in as root remotely. In case anyone (or a tool) sets the password, it's good practice to double check if password-based logins are disabled.

To cleanup after our examples, run kitchen destroy.

**Limitations**

It's easy to notice I like InSpec. However, there are few things which could be improved:

1. Code sharing is less than ideal. Custom resources are stored in the *libraries*directory. If you want to share a resource across multiple cookbooks, you have to create a separate profile.
2. Some resources are still missing from Serverspec, but it isn't a big problem. Maintainers are re-implementing/moving resources upon user request. You just may need to wait a while for release.

**Alternatives**

To achieve some level of objectivity, I would like to mention alternatives to InSpec.

**Serverspec**

It's probably a most common alternative to InSpec, migration between any of them [isn't complicated](http://inspec.io/docs/reference/migration/). In my opinion, it's closer to RSpec. It does support shared groups, which helps if you made some bad decisions and you are keeping too many cookbooks in a single repository. Unfortunately, it won't run so easily over docker/winrm protocols. In our case, it's also slower to re-run test after changes. Test files upload process can take over 60 seconds, where in the case of InSpec our tests start almost immediately.

**Goss**

Maybe it isn't a full blown alternative. It's a small infrastructure validation tool which I found in [Cron Weekly newsletter](https://www.cronweekly.com/). It's written in Go, it doesn't have many dependencies, it's self-contained, it's fast. Tests are written in YAML if you don't like Ruby. It doesn't have so many resources like InSpec or Serverspec, but there is one awesome feature - it can expose a health endpoint over HTTP. You can validate your infrastructure remotely. I wonder If it can be ported to InSpec or Serverspec. It would be so much easier to just run single query without any dependencies. It may be in a conflict with Chef Compliance tho.

**Summary**

How integration tests or InSpec are helping us?

1. We had a highly experimental environment - sandbox. It was used during the very early stage of infrastructure development, where things were checked manually. We don't need it anymore, we just write integration tests and run them in Test-Kitchen. It saved us some money because we no longer need additional AWS instances.
2. InSpec is a little bit faster than Serverspec. During development, when changes to test suite are made. Serverspec has to re-upload the code to run it on an instance. InSpec handles it much faster. This saves us 15-30 seconds on each test suite. With over 20 suites in our main repo, it helps a lot.
3. It helped us nail multiple smaller and bigger bugs, most notably:
   * We were loosing HTTP2 at some stage across our HTTP stack serving Kibana, Ubuntu 14.04 had outdated OpenSSL. We didn't know about it until integration tests were introduced.
   * Multiple cookbooks were overwriting the same sysctl value. We've had different sysctl settings for different services. When two or more wrapper cookbooks were put on the same node it came to our attention that cookbooks were interfering with each other.
4. It helps to mix wrapper cookbooks. Sometime in the past, we've had a rule: never mix services on single node. Interactions with operating system were sometimes complex. However on the early stage of the project it's a resource waste to have one/two (replicated) instances for each service. Integration tests help with this, a lot.
5. We're gradually preparing for new Ubuntu LTS. With each bigger change we're running tests against newer Ubuntu and we're fixing code which wasn't universal enough.

That's all. I know I've just scratched the surface of integration testing. I hope, if you didn't do integration testing, you'll at least reconsider introducing them into your workflow.