04

02/25

Lessons Learned from the Field

Effective Infrastructure Testing

Christian Stankowic

Technical Leader Linux

/ About me

whoami

IT automation, enjoys working with machines that are supposed to help you solving problems that you wouldn't have had without them.

RHEL, SLES, Foreman, Uyuni, Ansible, Terraform

Red Hat

Red Hat Certified Architect

Leon Krass

System Engineer

/ About me

Introduction

Passionate about secrets management and (sometimes over-) engineering IT automation with a touch of Kubernetes and/or Ansible magic.

HashiCorp Vault, Kubernetes, Ansible, Terraform

CERTIFIED kubernetes ADMINISTRATOR
CERTIFIED kubernetes APPLICATION DEVELOPER
Vault
CERTIFIED ASSOCIATE
HCVA0-002

Agenda

1

Why do we need infrastructure testing?

2

Which tools do we use?

3

6 Lessons Learned

4

Wrap-Up

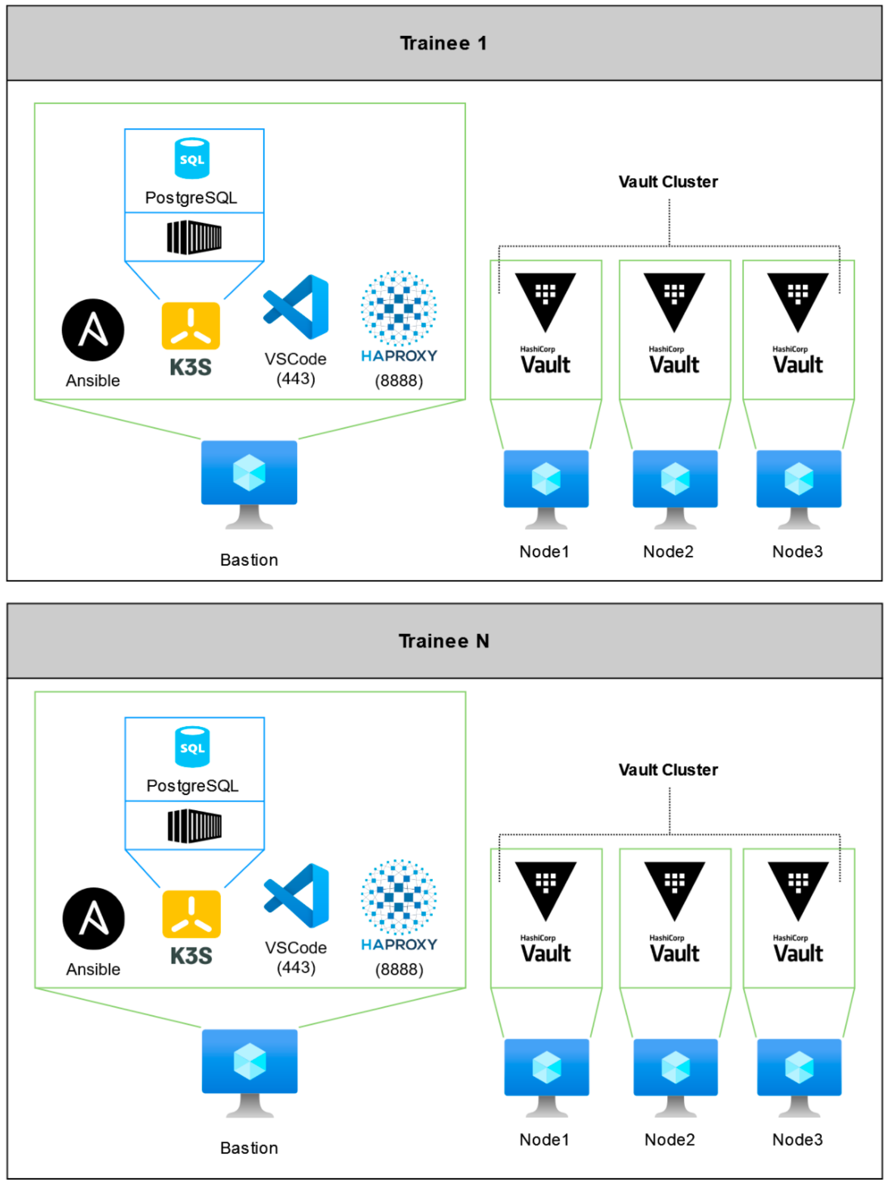
```
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

    #selection at the end -add back the deselected mirror mo
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the a
    #mirror_ob.select = 0
```

Why do we need infrastructure testing?

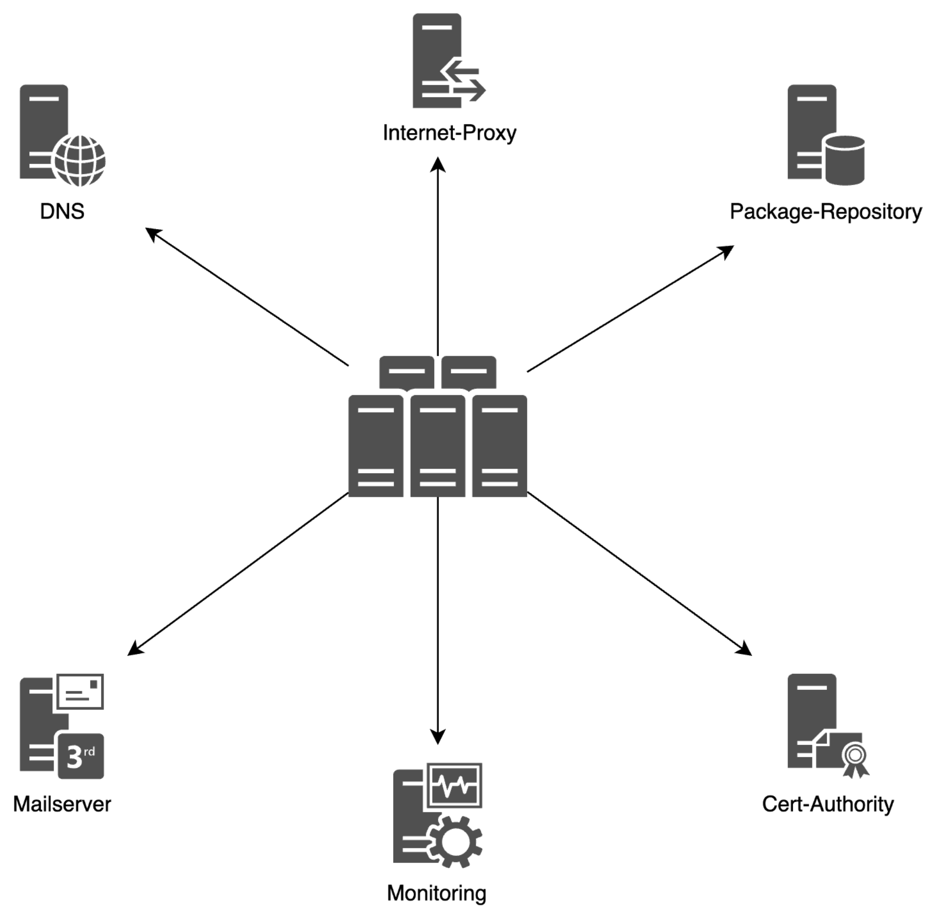
If devs test their code, why shouldn’t we test our infrastructure?

Example 1: Workshop Lab Environments

- Trainings for our customers / employees require unified lab environments
- Lab environments show complex setups, consisting of multiple hosts, services and dependencies
- Maintenance during a workshop is time consuming and interrupts the flow
- Lab setup should be checked before starting a workshop
  - → Hosts up and running?
  - → Services installed, configured and available?
  - → Required ports accessible?
  - → …



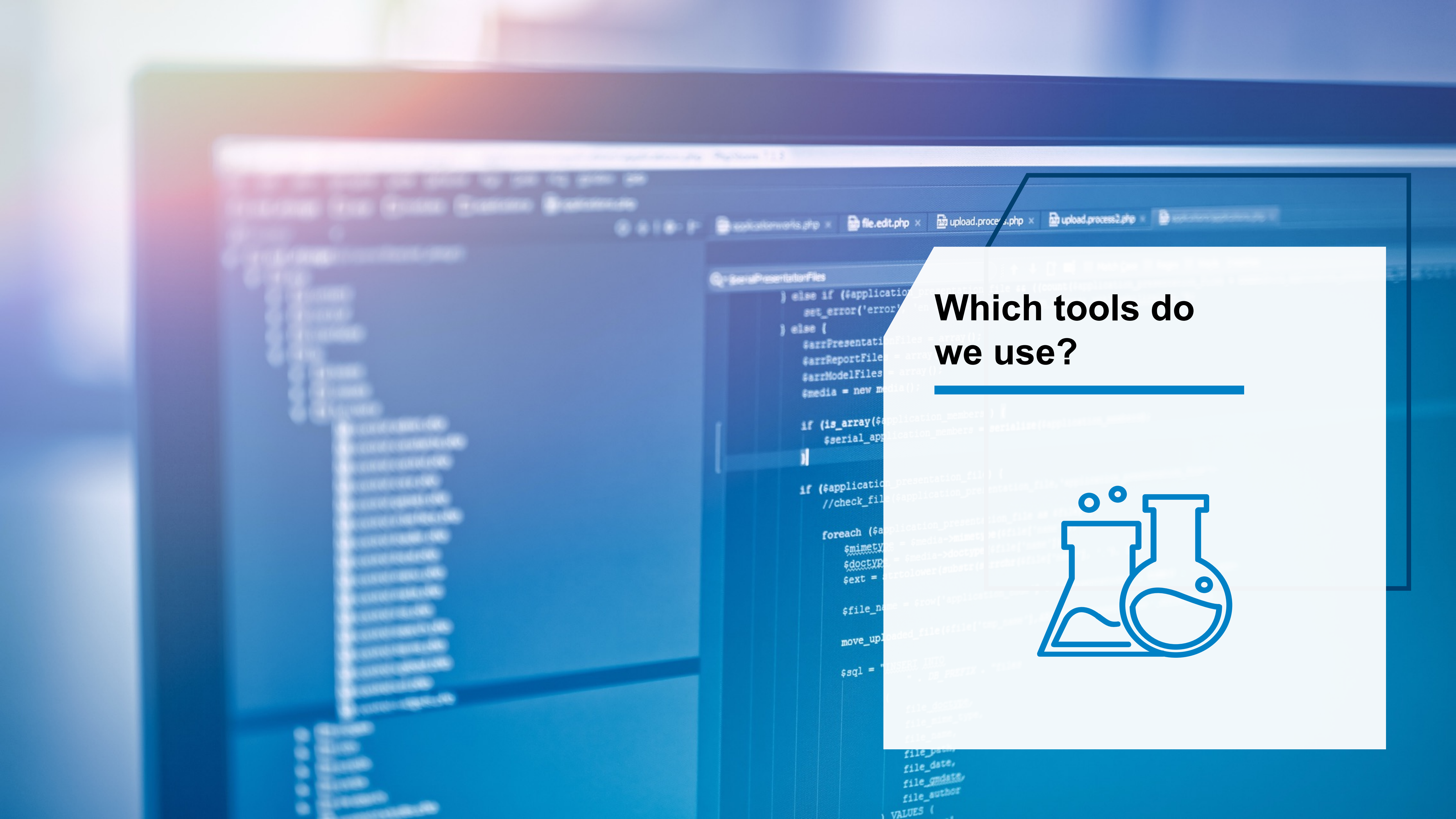
Example 2: Customer Environments



- Customer environments vary widely
  - Cloud, bare-metal, VMs, …
  - Network topology
  - Services
- Unit tests can help to identify challenges quickly
  - No internet access
  - No DNS
  - Missing package repositories
- Tests can identify if requirements for specific software is met
  - CPU, memory, bandwidth, …
  - OS / dependency versions
  - Network connectivity

Example 3: Production Hardening

- For many systems and software there are already automated tools
  - Linux (and more!): DevSec Hardening Framework
  - Kubernetes: kube-bench
  - Docker: docker-bench-security
- Vendors offer hardening guidelines that can be automatically tested, e. g.,
  - HashiCorp Vault: Production Hardening Guide
  - OWASP: Database Security Cheat Sheet

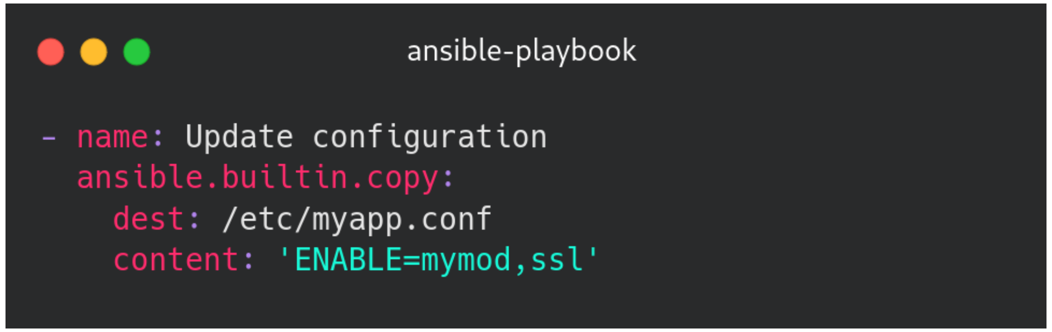


Which tools do we use?

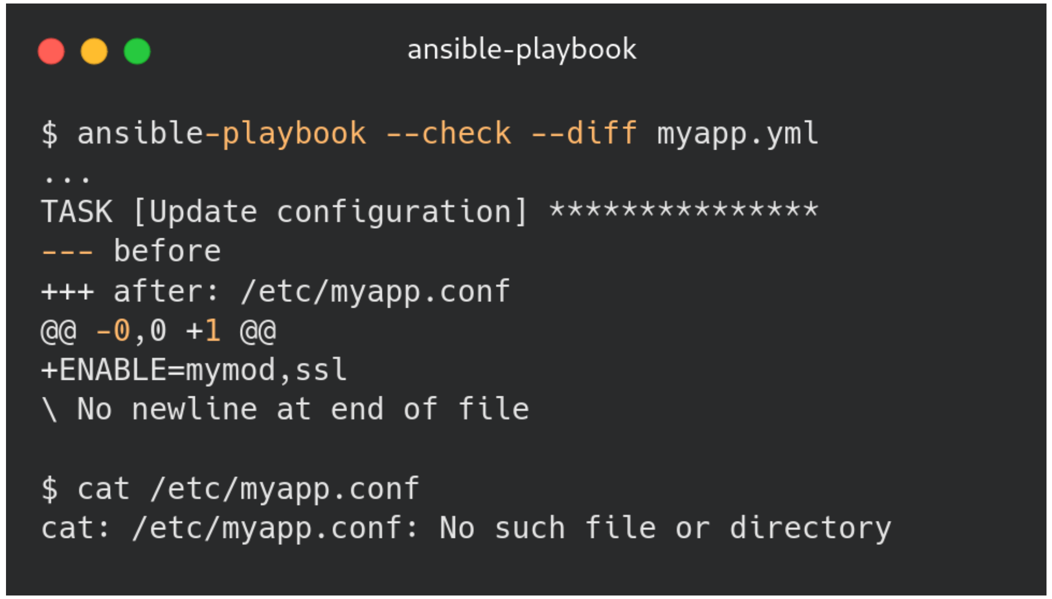
Ansible: `--check` and `--diff`

☆ 63.8k on

- **check** and **diff** mode can be used separately or together
  - `--check` only checks which changed would be made
    - **Not every module** supports this!
    - actions build on each other will fail
  - `--diff` lists which changes were (or would be) made
    - Also, not every module supports this
- Both options can be set on task level or as CLI option
- Help you finding out how changes affect your infrastructure


ansible-playbook

```yaml
- name: Update configuration
  ansible.builtin.copy:
    dest: /etc/myapp.conf
    content: 'ENABLE=mymod,ssl'
```

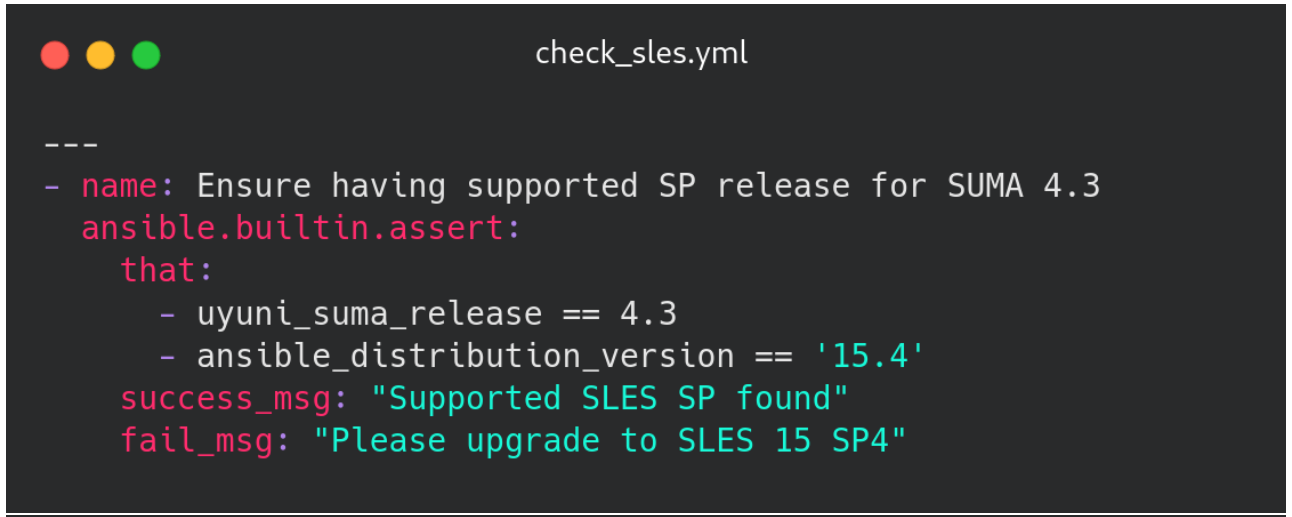

ansible-playbook

```
$ ansible-playbook --check --diff myapp.yml
...
TASK [Update configuration] ***************
--- before
+++ after: /etc/myapp.conf
@@ -0,0 +1 @@
+ENABLE=mymod,ssl
\ No newline at end of file

$ cat /etc/myapp.conf
cat: /etc/myapp.conf: No such file or directory
```

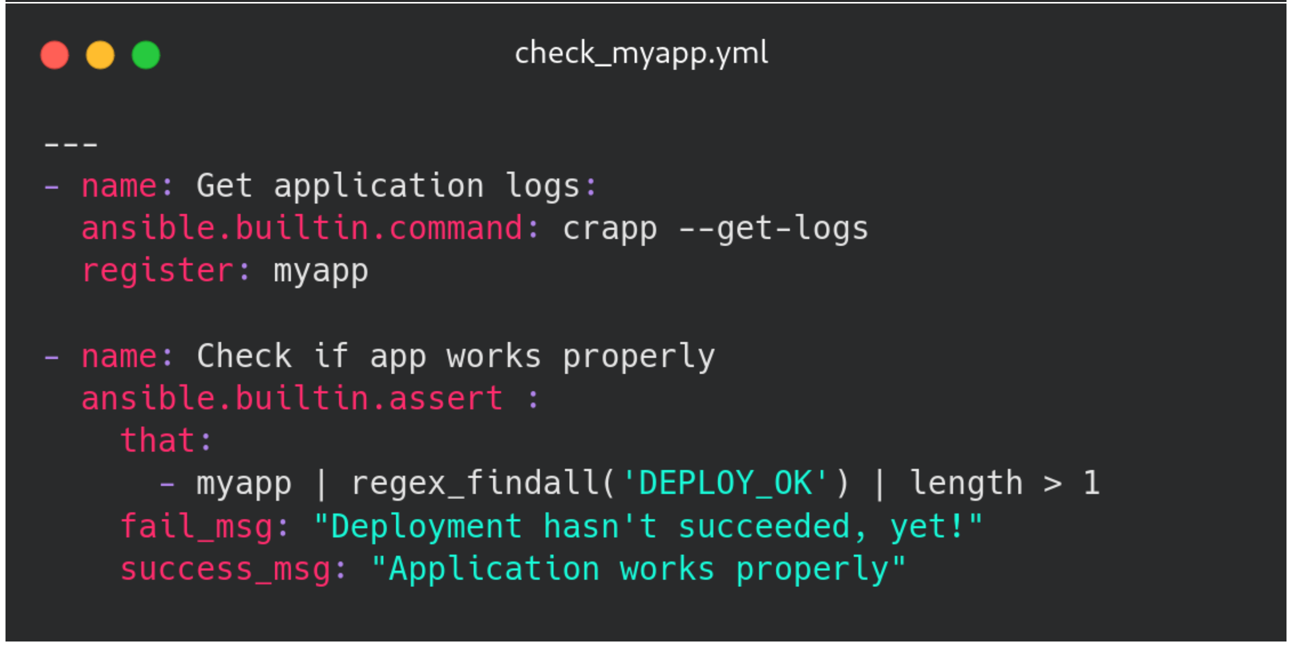
ansible.builtin.assert

☆ 63.8k on

- Part of `ansible-core`, included in all Ansible installations
- Checks if given expressions are true
  - Success / error messages
- Useful in context with Jinja2 expressions and **Ansible facts**
- Typical use-cases
  - Double-check if required state is given
  - Pre-checks before actions (e.g. version check)
  - Simple tests in Ansible roles
- Not very handy for comprehensive infrastructure testing
  - Syntax, nested Jinja2 and Ansible logic


check_sles.yml

```yaml
---
- name: Ensure having supported SP release for SUMA 4.3
  ansible.builtin.assert:
    that:
      - uyuni_suma_release == 4.3
      - ansible_distribution_version == '15.4'
    success_msg: "Supported SLES SP found"
    fail_msg: "Please upgrade to SLES 15 SP4"
```


check_myapp.yml

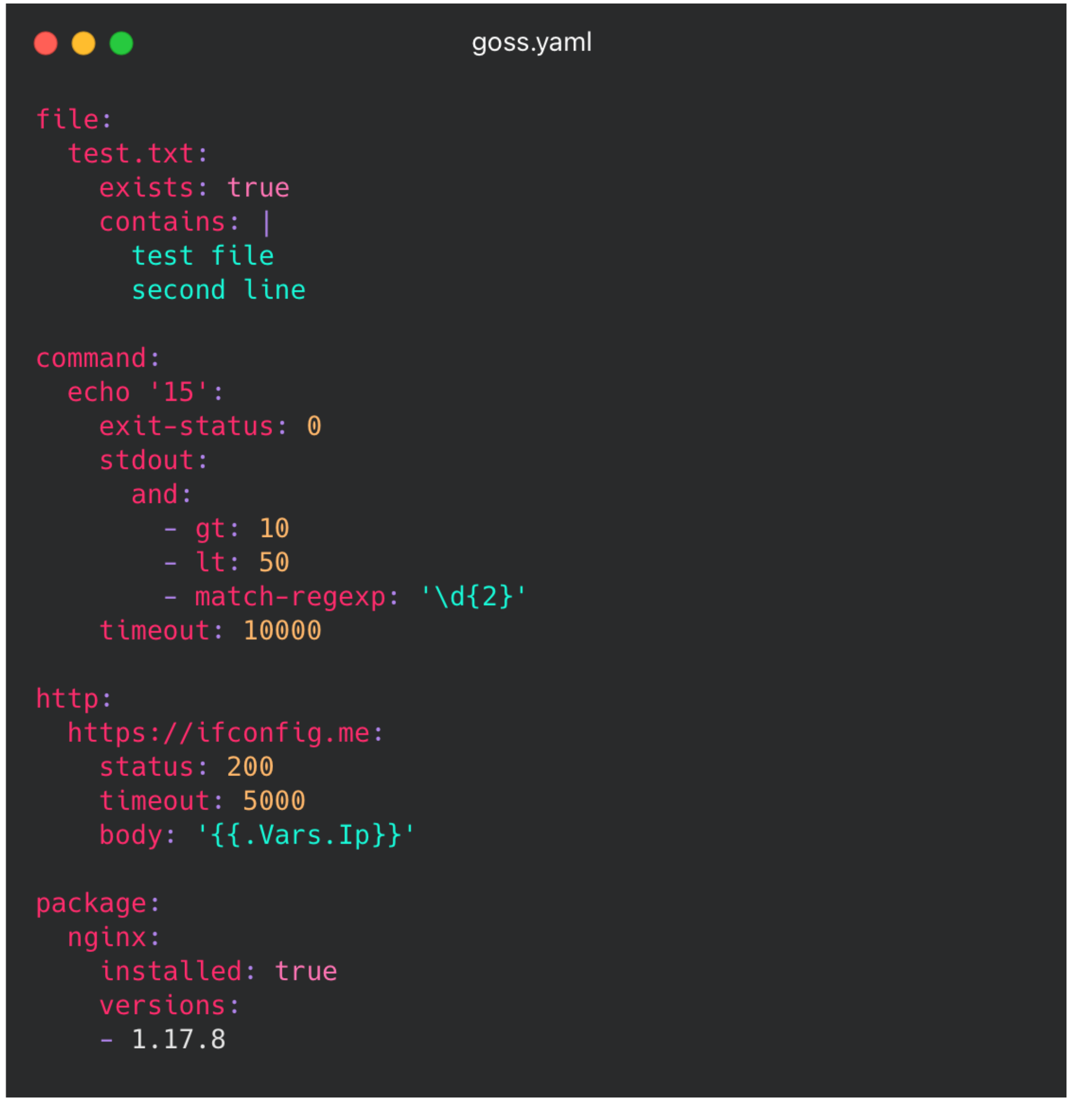
```yaml
---
- name: Get application logs:
  ansible.builtin.command: crapp --get-logs
  register: myapp

- name: Check if app works properly
  ansible.builtin.assert :
    that:
      - myapp | regex_findall('DEPLOY_OK') | length > 1
    fail_msg: "Deployment hasn't succeeded, yet!"
    success_msg: "Application works properly"
```

goss

☆ 5.7k on

- ⇢ Imperative & declarative
- ⇢ Local execution
- ⇢ Works with Linux, Windows & macOS
- ⇢ Works also for containers (with `dgoss`), `docker-compose` (with `dcgoss`) or Kubernetes (with `kgoss`)

- ⇢ Supported resources
  - → Packages
  - → Files
  - → Ports
  - → Remote addresses
  - → Command results
  - → Services
  - → Users, groups
  - → DNS records
  - → Processes
  - → Kernel parameters
  - → Mounts
  - → Interfaces
  - → HTTP-request results


goss.yaml

```yaml
file:
  test.txt:
    exists: true
    contains: |
      test file
      second line

command:
  echo '15':
    exit-status: 0
    stdout:
      and:
        - gt: 10
        - lt: 50
        - match-regexp: '\d{2}'
    timeout: 10000

http:
  https://ifconfig.me:
    status: 200
    timeout: 5000
    body: '{{.Vars.Ip}}'

package:
  nginx:
    installed: true
    versions:
    - 1.17.8
```

goss

☆ 5.7k on

```
sh-4.3#
```

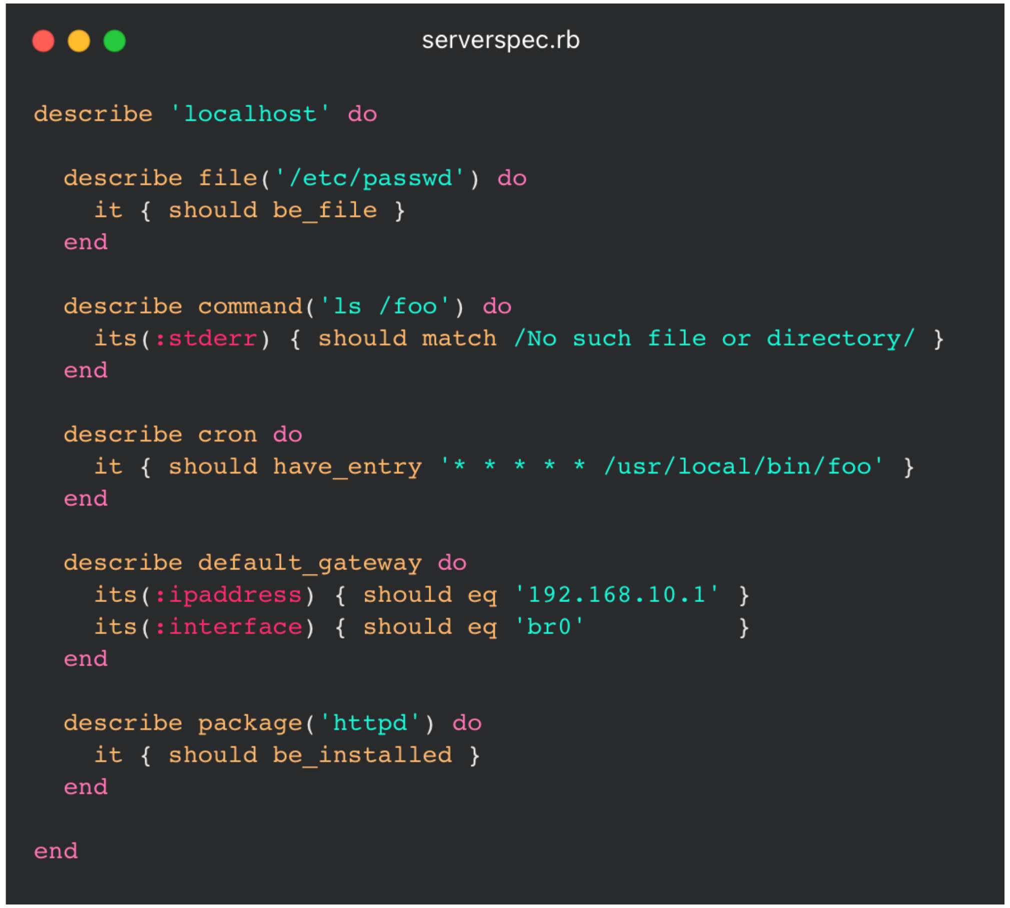
goss docs

serverspec

☆ 2.5k on

- Plugin for `RSpec` focusing on infrastructure tests
  - Ruby-based testing framework
- Declarative tests
- Local execution or via `ssh`, WinRM, Docker API
- Works with Linux, Windows & macOS

- 40 supported resources
  - Bonds
  - Bridges
  - Cgroups
  - Command results
  - Cronjobs
  - Gateways
  - Docker containers/ images
  - Files
  - Users, groups
  - DNS entries
  - IIS app pools/ websites
  - Interfaces
  - iptables, ipfilters, ipnat
  - Kernel modules, params
  - Linux audit system
  - Mail aliases
  - MySQL configs
  - Packages
  - PHP configs
  - Ports
  - Processes
  - Routing tables
  - SELinux
  - … (see docs)

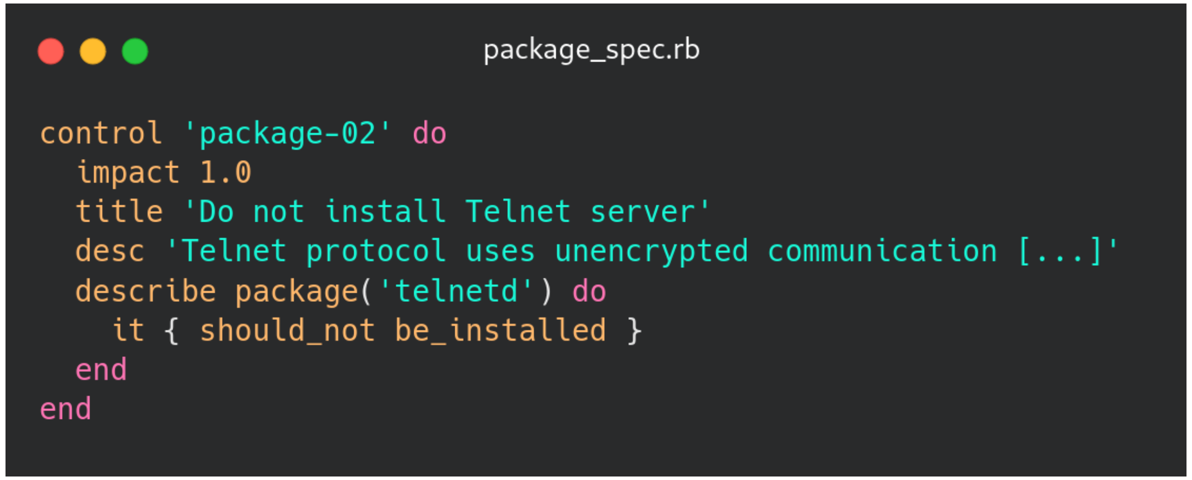


serverspec.rb

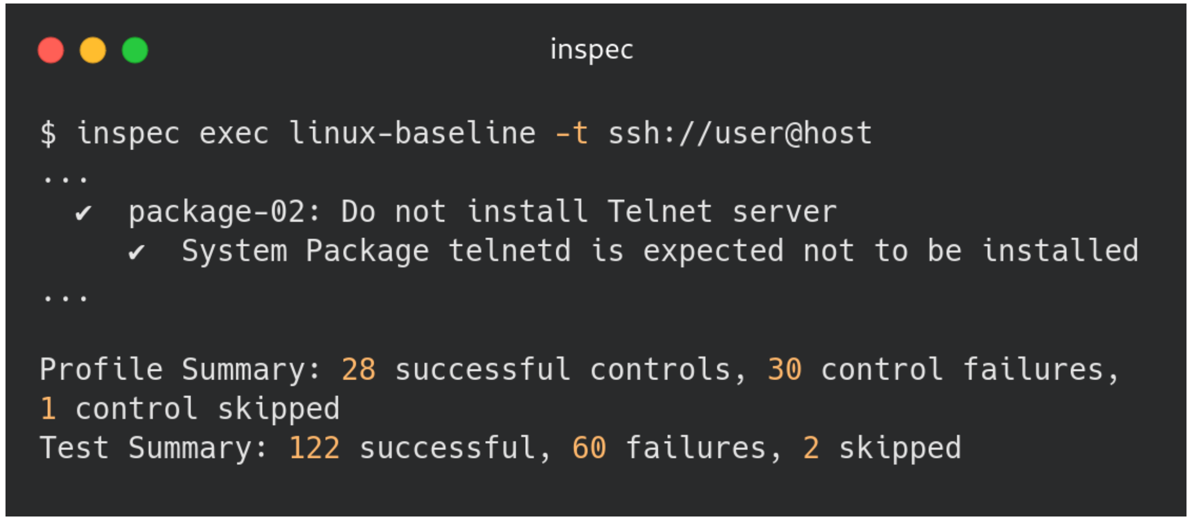
```ruby
describe 'localhost' do

  describe file('/etc/passwd') do
    it { should be_file }
  end

  describe command('ls /foo') do
    its(:stderr) { should match /No such file or directory/ }
  end

  describe cron do
    it { should have_entry '* * * * * /usr/local/bin/foo' }
  end

  describe default_gateway do
    its(:ipaddress) { should eq '192.168.10.1' }
    its(:interface) { should eq 'br0'          }
  end

  describe package('httpd') do
    it { should be_installed }
  end

end
```

Dev-Sec and ansible-lockdown

- Not only hardening new systems is important
- Existing systems must also be re-checked and secured **regularly**
- High effort depending on the system landscape size
- Dev-Sec audits **and** hardens servers
  - Baselines for Linux, SSH, Apache,…
  - Ansible and Puppet content for **remediation**
- Baselines based on
  - CIS benchmarks
  - NSA hardening guides
  - Vendor best-practices and guides
- Checked using InSpec
  - Based on RSpec with a focus on security
- ansible-lockdown
  - Comparable to Dev-Sec, but uses `goss` for testing
  - Benchmarks and remediation for CIS and STIG


package_spec.rb

```
control 'package-02' do
  impact 1.0
  title 'Do not install Telnet server'
  desc 'Telnet protocol uses unencrypted communication [...]'
  describe package('telnetd') do
    it { should_not be_installed }
  end
end
```


inspec

```
$ inspec exec linux-baseline -t ssh://user@host
...
  ✔  package-02: Do not install Telnet server
     ✔  System Package telnetd is expected not to be installed
...

Profile Summary: 28 successful controls, 30 control failures,
1 control skipped
Test Summary: 122 successful, 60 failures, 2 skipped
```

testinfra

☆ 2.4k on

- Plugin for `pytest` focusing on infrastructure tests
- Useful for testing **actual states** of servers configured using tools like Ansible or SaltStack
- `serverspec` equivalent in Python

- Support various **connection** backends
  - Local, paramiko, `ssh`, WinRM
  - Docker, Podman, LXC/LXD
  - Ansible, SaltStack
  - `kubectl`, OpenShift

- Can also be used outside of `pytest`
  - Integrate tests in your application
- Optional Nagios-compatible output

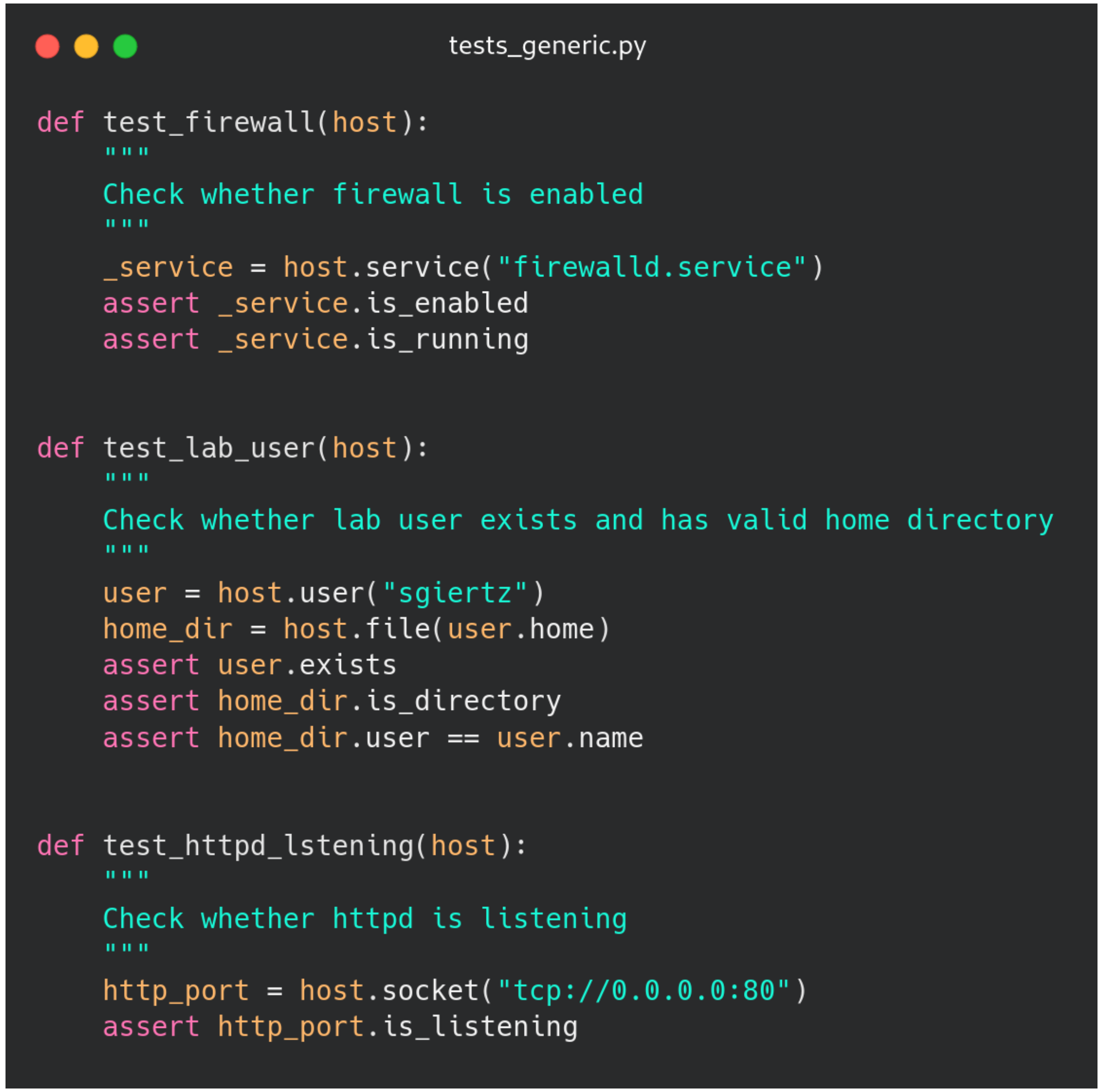
testinfra

☆ 2.4k on

- **26** modules with **88** properties for system components
  - `environment`, `interface`, `mount_point`, `package`, `service`, ...

- Easy syntax, low learning curve
- Documentation has a lot of good examples

testinfra docs


tests_generic.py

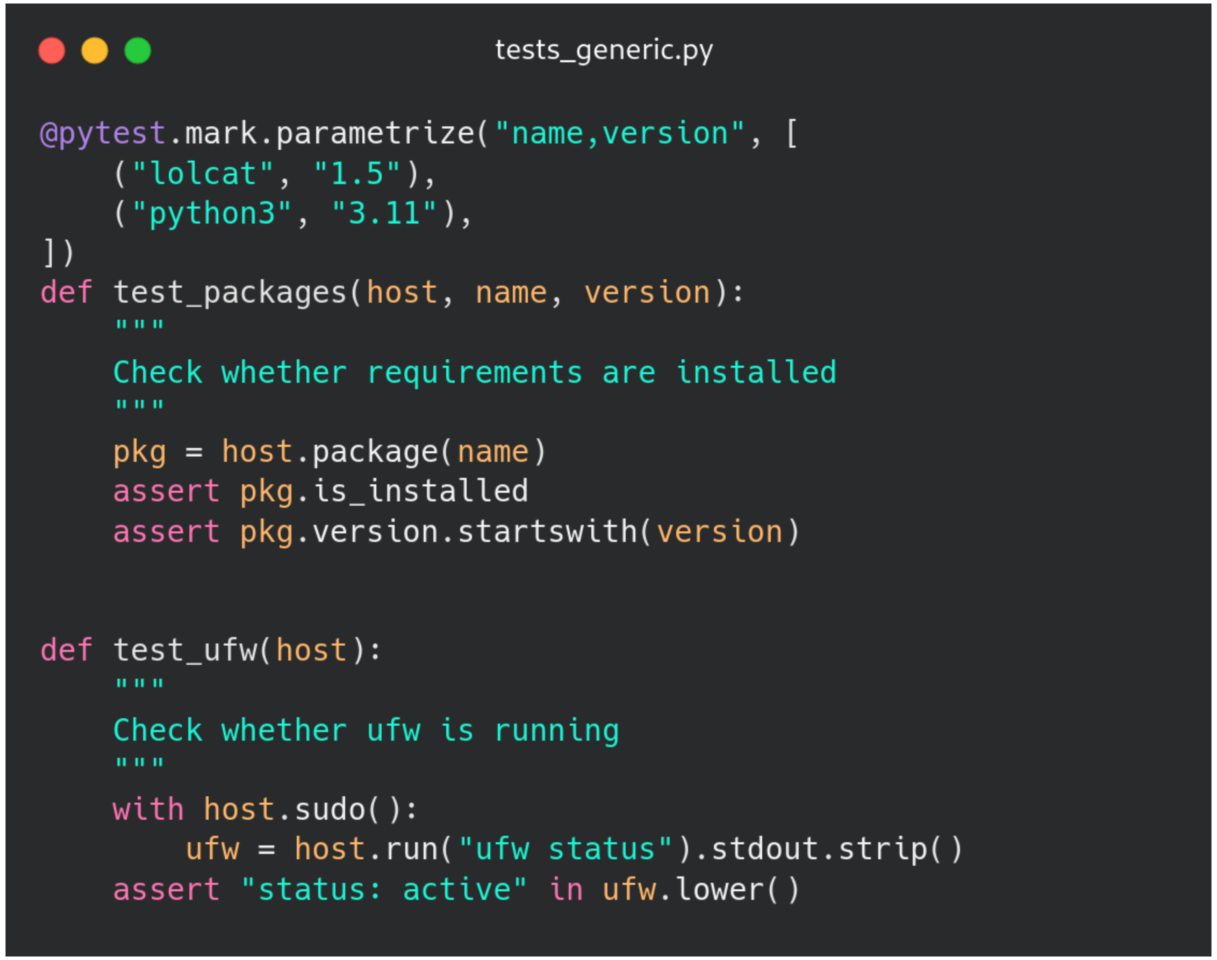
```
def test_firewall(host):
    """
    Check whether firewall is enabled
    """
    _service = host.service("firewalld.service")
    assert _service.is_enabled
    assert _service.is_running


def test_lab_user(host):
    """
    Check whether lab user exists and has valid home directory
    """
    user = host.user("sgiertz")
    home_dir = host.file(user.home)
    assert user.exists
    assert home_dir.is_directory
    assert home_dir.user == user.name


def test_httpd_lstening(host):
    """
    Check whether httpd is listening
    """
    http_port = host.socket("tcp://0.0.0.0:80")
    assert http_port.is_listening
```

testinfra

☆ 2.4k on

- Tests can also be **parametrized**
- Split tests in multiple files for larger suites, e.g.
  - `tests_generic.py`
  - `tests_app1.py`
  - `tests_smoke.py`
- Use `pytest-xdist` for distributing tests across multiple CPUs
- `sudo` supported **only** with `NOPASSWD`
  - There is no way to prompt for passwords
  - Ansible inventory + `--force-ansible` should do the trick, but doesn't work for me

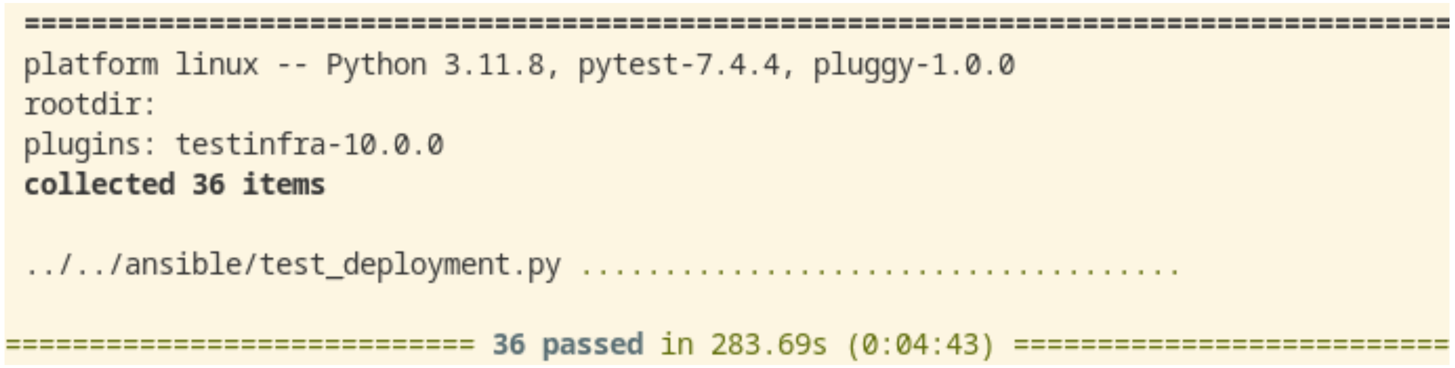
testinfra docs


tests_generic.py

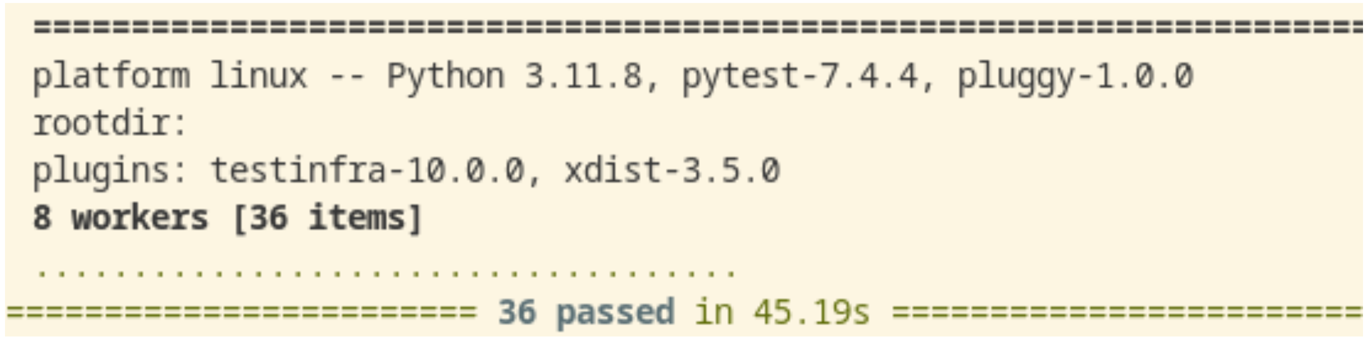
```python
@pytest.mark.parametrize("name,version", [
    ("lolcat", "1.5"),
    ("python3", "3.11"),
])
def test_packages(host, name, version):
    """
    Check whether requirements are installed
    """
    pkg = host.package(name)
    assert pkg.is_installed
    assert pkg.version.startswith(version)


def test_ufw(host):
    """
    Check whether ufw is running
    """
    with host.sudo():
        ufw = host.run("ufw status").stdout.strip()
    assert "status: active" in ufw.lower()
```

`pytest-xdist` makes a huge difference

☆ 1.5k on



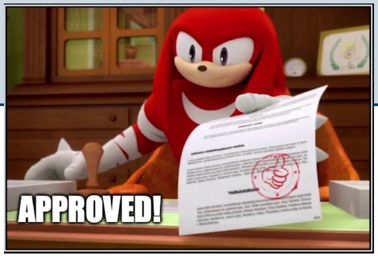
```
==========================================================================================
platform linux -- Python 3.11.8, pytest-7.4.4, pluggy-1.0.0
rootdir:
plugins: testinfra-10.0.0
collected 36 items

../../ansible/test_deployment.py ....................................

============================ 36 passed in 283.69s (0:04:43) ============================
```



```
==========================================================================================
platform linux -- Python 3.11.8, pytest-7.4.4, pluggy-1.0.0
rootdir:
plugins: testinfra-10.0.0, xdist-3.5.0
8 workers [36 items]
....................................
========================= 36 passed in 45.19s ==========================
```

GitHub

Comparison

	goss	serverspec	testinfra
Im-/declarative	Both	Declarative	Declarative
Language	YAML	Ruby/RSpec	Python
Execution	Local	Local, `ssh`, WinRM	Local, `ssh`, WinRM, **Ansible**, …
OS	Linux, Windows, macOS	Linux, Windows, macOS	Linux, Windows, macOS, BSD, …
Scope	Single Server	Single Server	Single / **Multiple** Servers
Resources	14+	40	26+
			

6 Lessons Learned

Lesson 1: Testing In Production

“*Testing in production is possible on the infrastructure level, if you do it right.*”

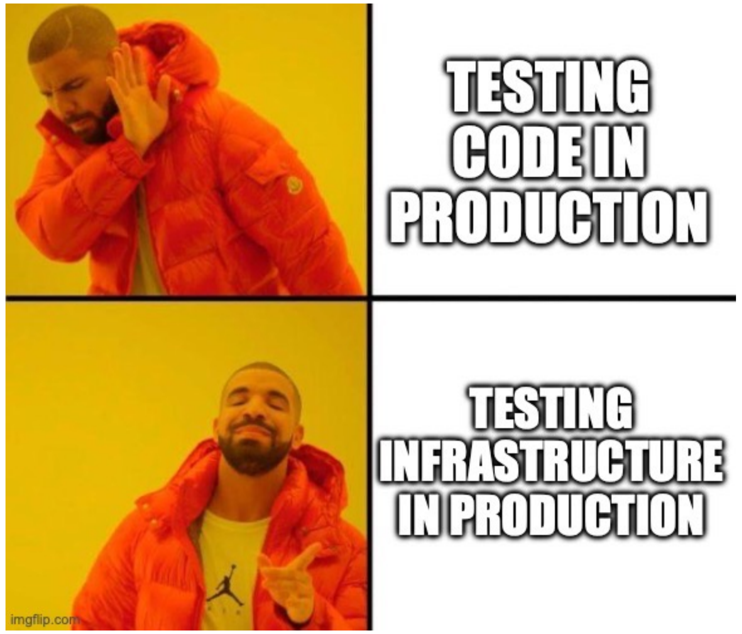
⚠️ Ensure, your tests don’t change the state of your infrastructure

⚠️ Avoid generation of heavy loads

- → Influence on other users
- → Generation of logs

Use health-check APIs and monitoring endpoints

Validate your tests in a testing environment, before launching them in production


TESTING CODE IN PRODUCTION
TESTING INFRASTRUCTURE IN PRODUCTION
imgflip.com

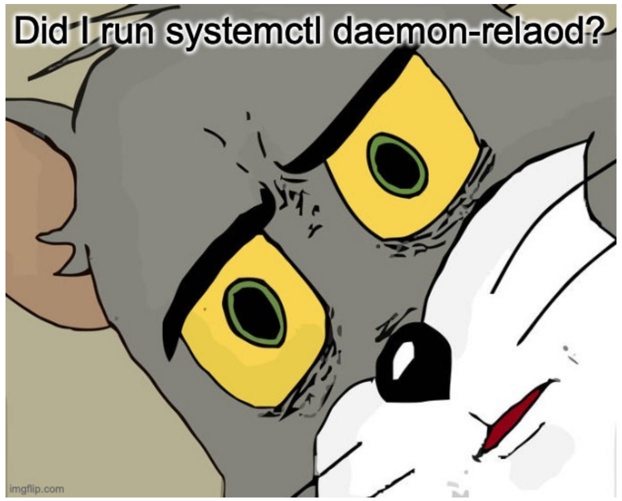
Lesson 2: State Validation

“*Tests must check the actual state of the system. It is not enough to verify the content of a configuration file; it must be ensured that, for example, a defined port is actually open*.”

⚠️ Checking config files only is **not** sufficient to ensure they are effective, or the desired state is established

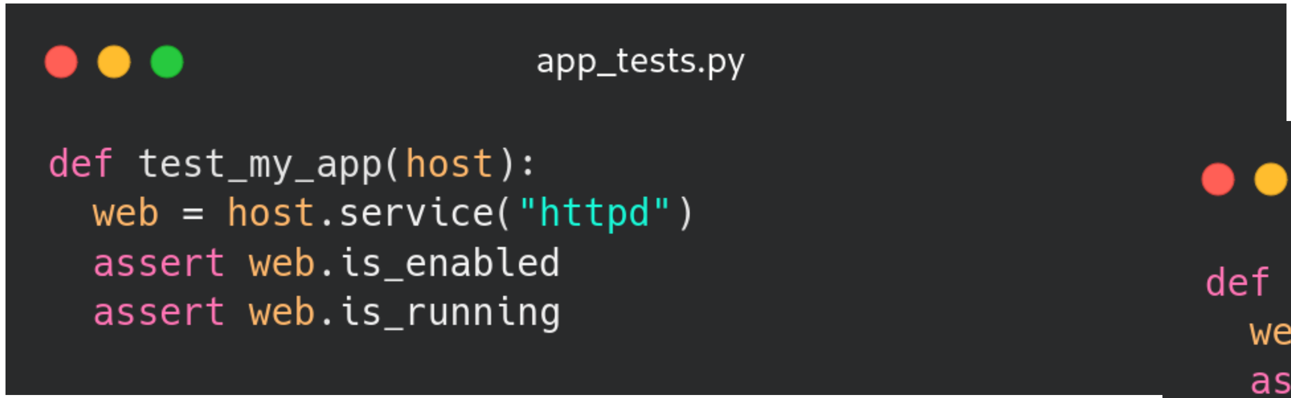
Always validate the state of your machines by querying their respective interfaces

Config files can still serve as source of truth



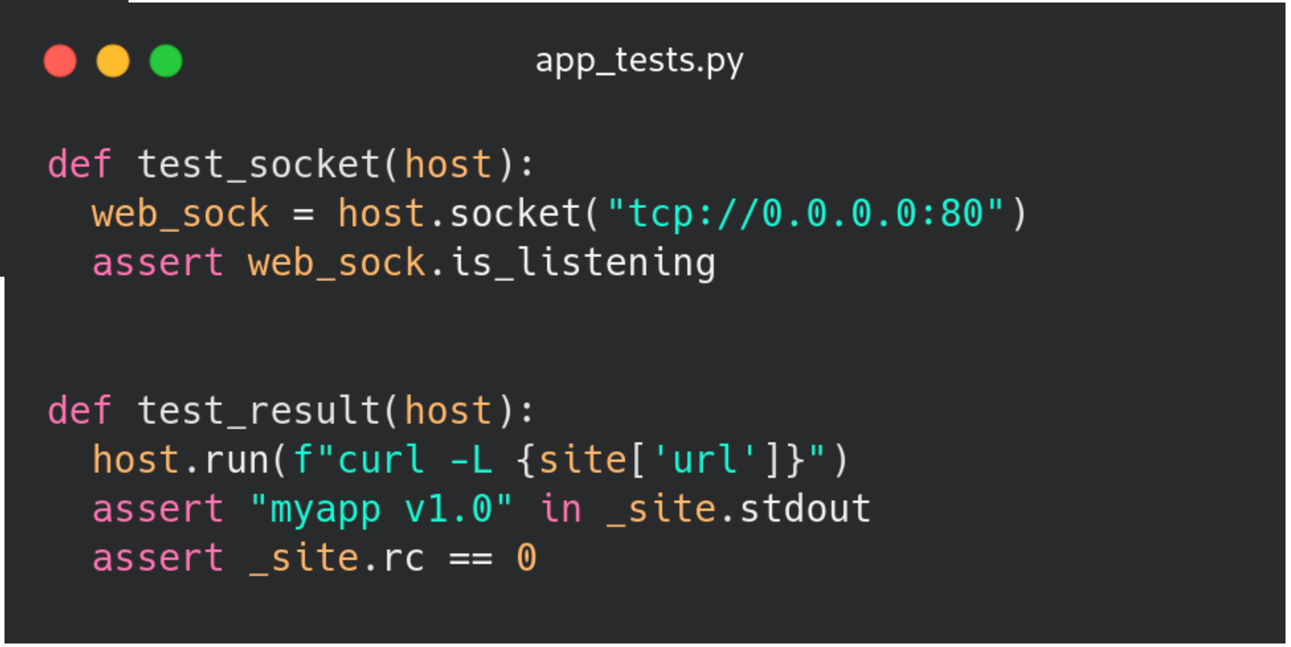
Lesson 2: State Validation

“Tests must check the actual state of the system. It is not enough to verify the content of a configuration file; it must be ensured that, for example, a defined port is actually open.”



app_tests.py

```
def test_my_app(host):
  web = host.service("httpd")
  assert web.is_enabled
  assert web.is_running
```



app_tests.py

```
def test_socket(host):
  web_sock = host.socket("tcp://0.0.0.0:80")
  assert web_sock.is_listening


def test_result(host):
  host.run(f"curl -L {site['url']}")
  assert "myapp v1.0" in _site.stdout
  assert _site.rc == 0
```

Lesson 3: Set Priorities

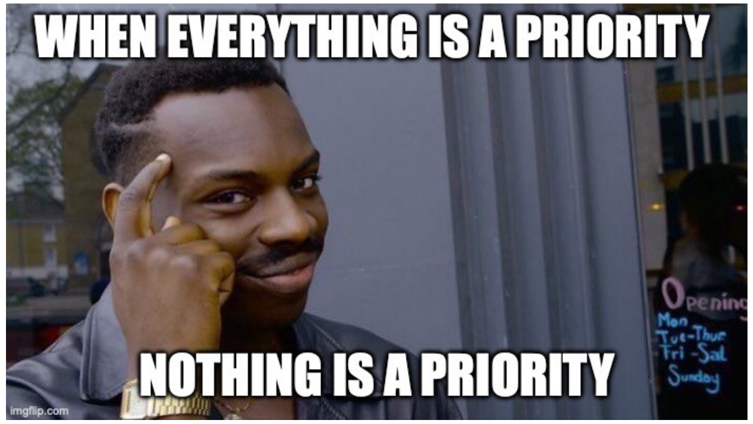
“*Not everything can be tested with manageable effort. It is important to set priorities and focus on the most critical aspects.*”

⚠️ Having a lot of requirements can be overwhelming, ask your stakeholders for priorities

⚠️ Don’t work to fine grained, it will coast you time

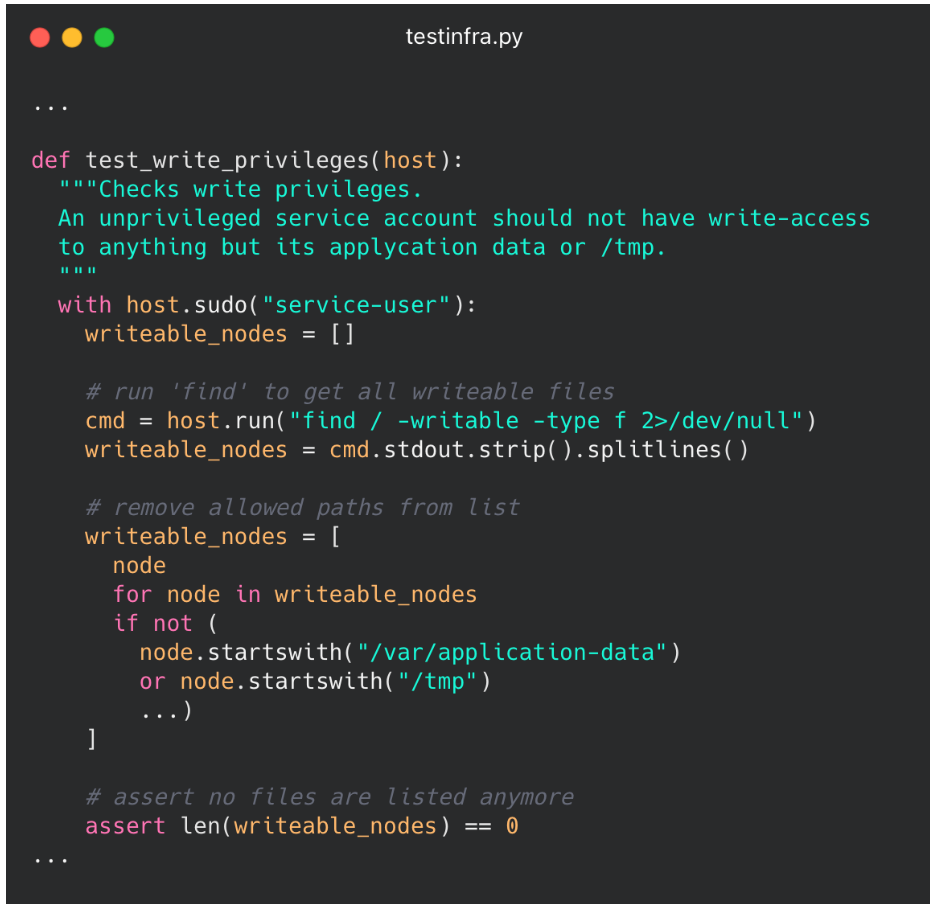
Pick the low hanging fruits first, keep it simple

A lot of applications offer endpoints for cumulated health checks



Lesson 4: Bash is your friend

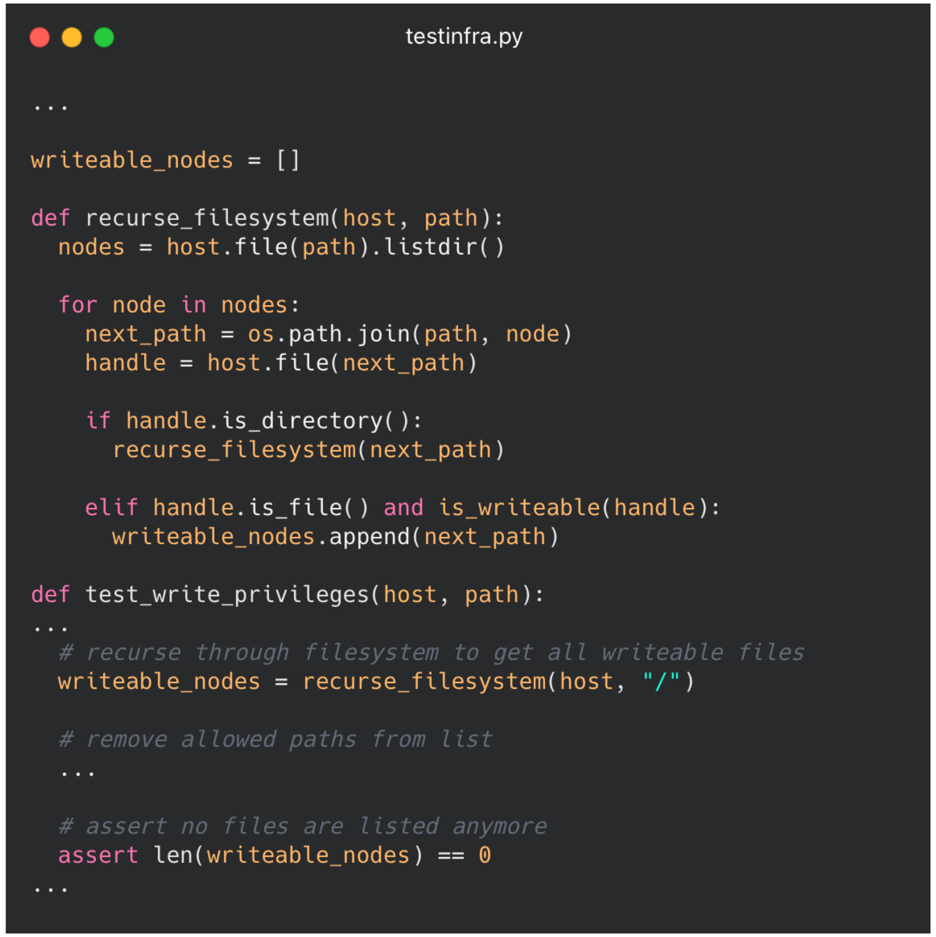
“Bash is your friend. Although testing tools offers many possibilities, Bash prompts can often be more efficient.”



testinfra.py

```
...

def test_write_privileges(host):
 """Checks write privileges.
 An unprivileged service account should not have write-access
 to anything but its applycation data or /tmp.
 """
 with host.sudo("service-user"):
  writeable_nodes = []

  # run 'find' to get all writeable files
  cmd = host.run("find / -writable -type f 2>/dev/null")
  writeable_nodes = cmd.stdout.strip().splitlines()

  # remove allowed paths from list
  writeable_nodes = [
   node
   for node in writeable_nodes
   if not (
    node.startswith("/var/application-data")
    or node.startswith("/tmp")
    ...)
  ]

  # assert no files are listed anymore
  assert len(writeable_nodes) == 0
...
```

>



testinfra.py

```
...

writeable_nodes = []

def recurse_filesystem(host, path):
 nodes = host.file(path).listdir()

 for node in nodes:
  next_path = os.path.join(path, node)
  handle = host.file(next_path)

  if handle.is_directory():
   recurse_filesystem(next_path)

  elif handle.is_file() and is_writeable(handle):
   writeable_nodes.append(next_path)

def test_write_privileges(host, path):
...
 # recurse through filesystem to get all writeable files
 writeable_nodes = recurse_filesystem(host, "/")

 # remove allowed paths from list
 ...

 # assert no files are listed anymore
 assert len(writeable_nodes) == 0
...
```

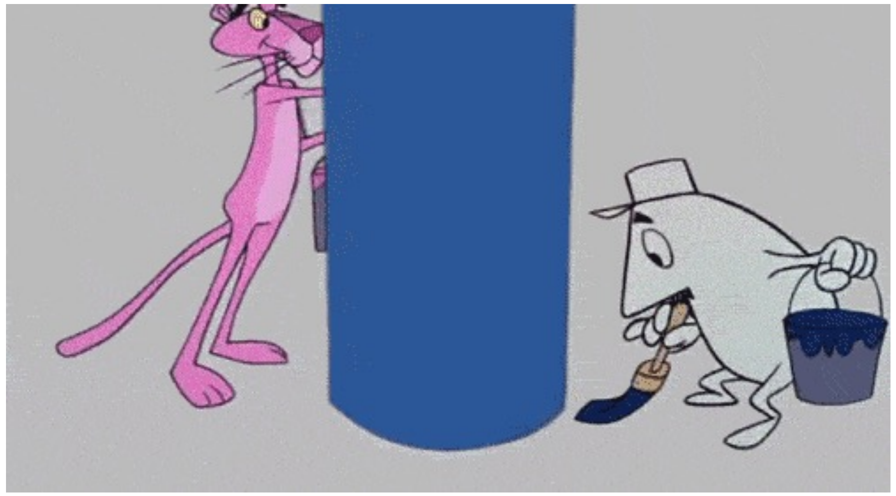
Lesson 5: Test your tests

“Automated tests should be regularly reviewed and updated. Systems and requirements change, and tests need to be adjusted accordingly to remain relevant.”

Setup test environments to check your tests are working, simulate test-scenarios (tools like ansible molecule can help)

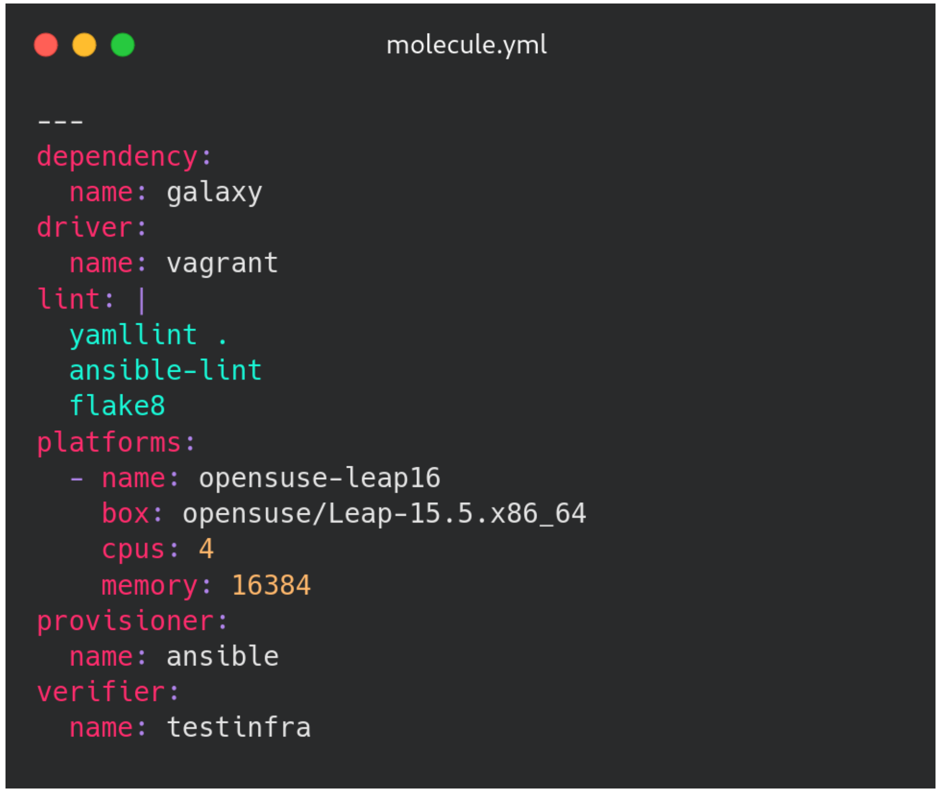
Regularly compare your tests to the documentation of your test targets, adopt changes

Subscribe to release channels or use tools like renovate

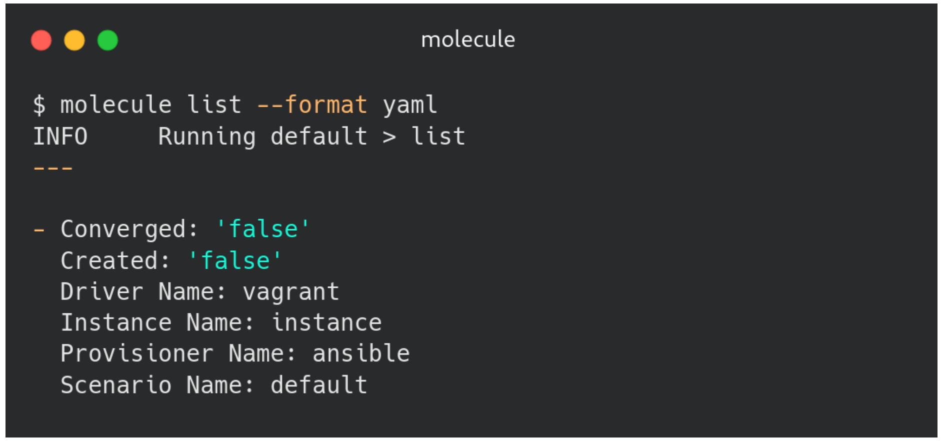


Molecule

- Assists with developing Ansible content
  - Collections
  - Playbooks and roles
- Automates the full testing workflow
  - Spinning up containers and VMs
  - Linting code
  - Running content (converge)
  - Testing using a verifier (e.g. Ansible or `testinfra`)
  - Destroy containers and VMs
- Python package
- `molecule-plugins` extend functionality
  - Support for containers, Vagrant, OpenStack, etc.


molecule.yml

```yaml
---
dependency:
  name: galaxy
driver:
  name: vagrant
lint: |
  yamllint .
  ansible-lint
  flake8
platforms:
  - name: opensuse-leap16
    box: opensuse/Leap-15.5.x86_64
    cpus: 4
    memory: 16384
provisioner:
  name: ansible
verifier:
  name: testinfra
```


molecule

```
$ molecule list --format yaml
INFO     Running default > list
---

- Converged: 'false'
  Created: 'false'
  Driver Name: vagrant
  Instance Name: instance
  Provisioner Name: ansible
  Scenario Name: default
```

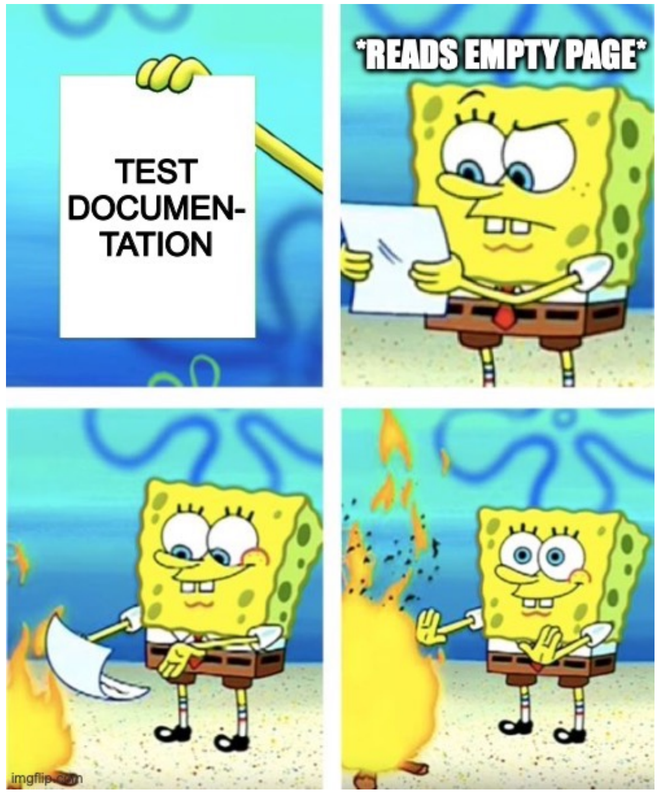
Lesson 6: Documentation is Essential

“Documentation is essential. Good documentation of tests and their results helps to identify and resolve issues more quickly.”

⚠️ If it’s unknown what is tested, you don’t know what you still need to test

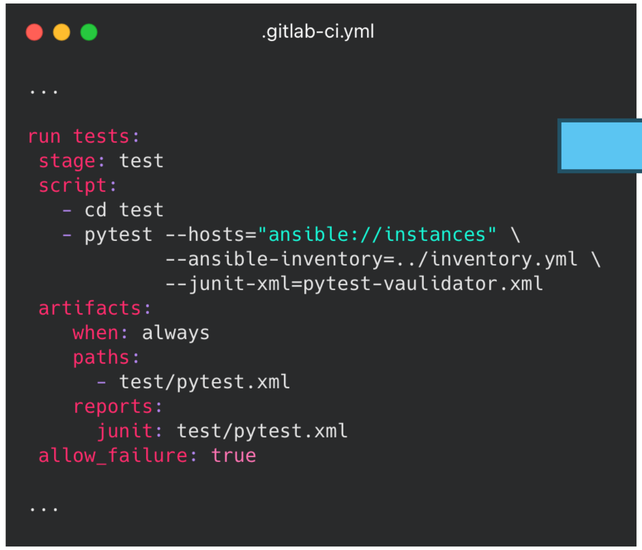
Documentation can be generated from source code comments ([doxygen](), [sphinx](), …)

Test protocols can be generated (`junit-xml`, …)



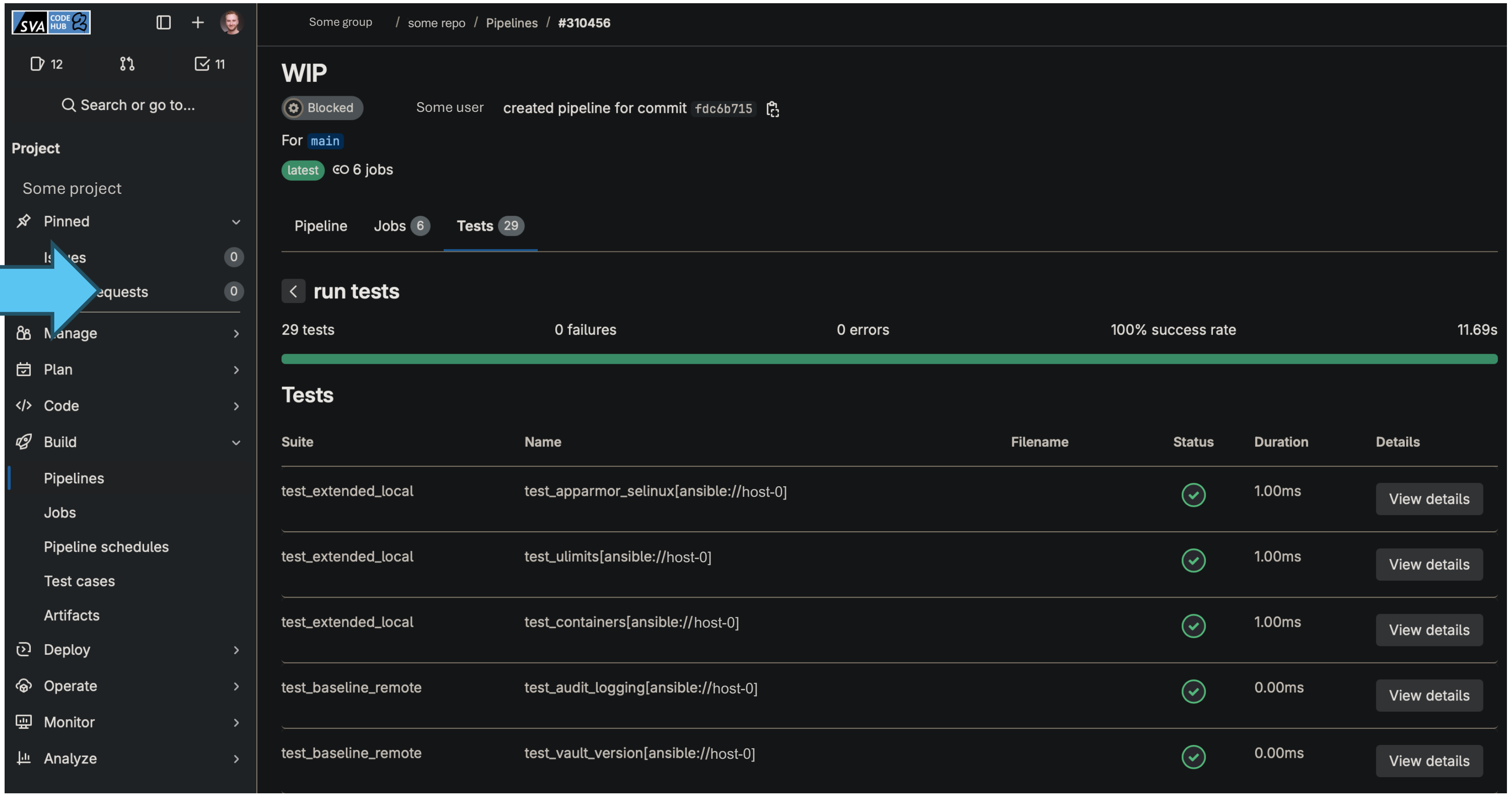
Lesson 6: Documentation is Essential

“Documentation is essential. Good documentation of tests and their results helps to identify and resolve issues more quickly.”



.gitlab-ci.yml

```
...

run tests:
  stage: test
  script:
    - cd test
    - pytest --hosts="ansible://instances" \
             --ansible-inventory=../inventory.yml \
             --junit-xml=pytest-vaulidator.xml
  artifacts:
      when: always
      paths:
        - test/pytest.xml
      reports:
        junit: test/pytest.xml
  allow_failure: true

...
```



WIP

Blocked Some user created pipeline for commit fdc6b715

For main

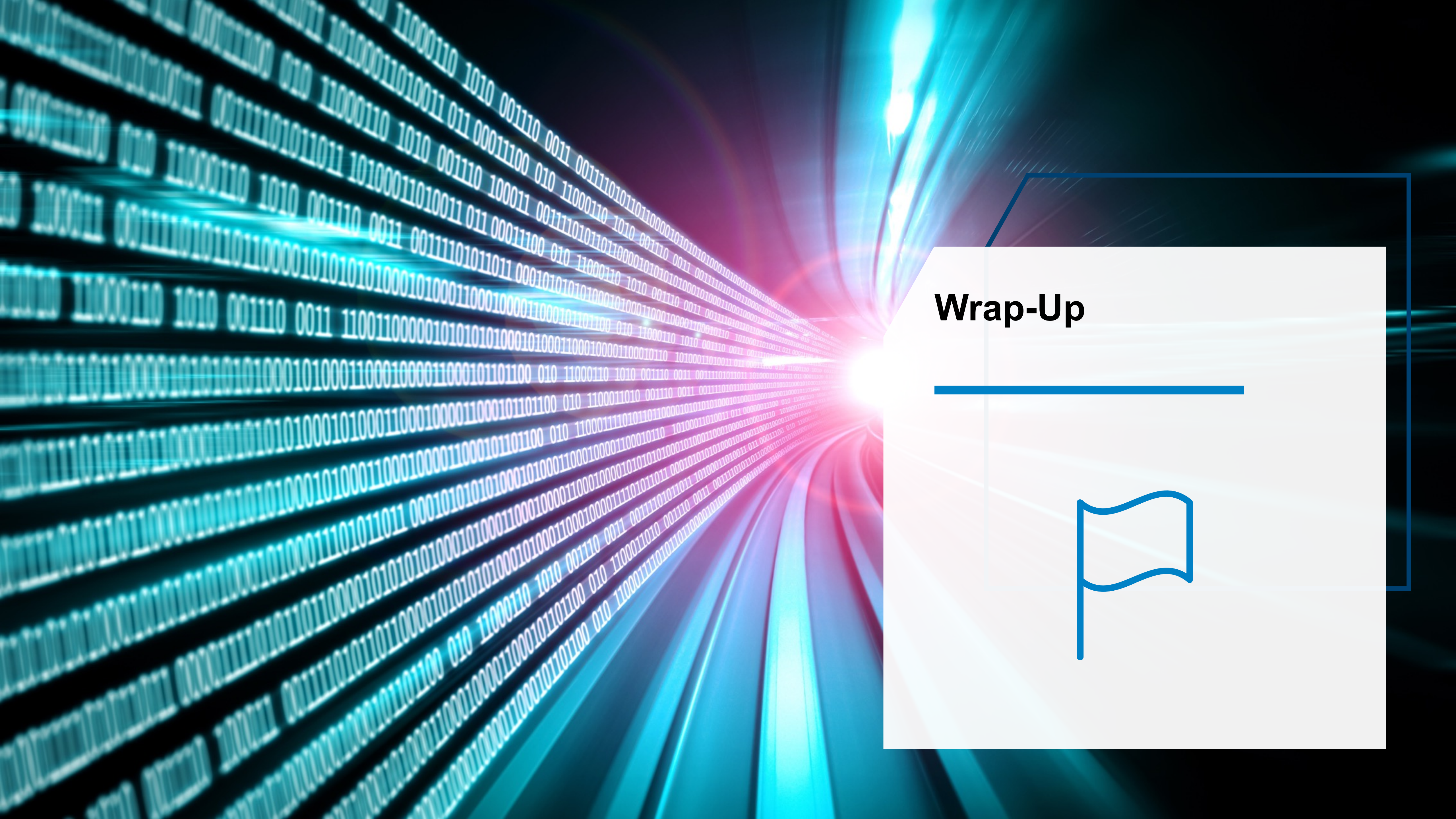
latest 6 jobs

run tests

29 tests 0 failures 0 errors 100% success rate 11.69s

Tests

Suite	Name	Filename	Status	Duration	Details
test_extended_local	test_apparmor_selinux[ansible://host-0]		✓	1.00ms	View details
test_extended_local	test_ulimits[ansible://host-0]		✓	1.00ms	View details
test_extended_local	test_containers[ansible://host-0]		✓	1.00ms	View details
test_baseline_remote	test_audit_logging[ansible://host-0]		✓	0.00ms	View details
test_baseline_remote	test_vault_version[ansible://host-0]		✓	0.00ms	View details



Wrap-Up

Wrap-Up

- ⇢ Testing is relevant for both code and infrastructure
  - e.g. Lab and customer environments
- ⇢ Not only functionality but also security should be tested
- ⇢ Various tools can assist with efficient testing
  - special tools per use-case
- ⇢ Also ensure checking an actual high-level state
- ⇢ Less is more
- ⇢ Documentation is key

/ Get your own copy

Slides available at:

/ Want to talk?

Contact

Feel free to contact to us if you…

- … still have questions
- ... have feedback/suggestions
- … just want to have a chat ☺

Christian Stankowic
Technical Leader Linux

+49 151 18025982
christian.stankowic@sva.de
www.sva.de

Location Wiesbaden
Borsigstraße 26
65205 Wiesbaden
Germany

LinkedIn

Leon Krass
System Engineer

+49 151 53882677
leon.krass@sva.de
www.sva.de

Location Hamburg
Große Elbstraße 273
22767 Hamburg
Germany

LinkedIn

04

02/25

Lessons Learned from the Field

Effective Infrastructure Testing