testinfra Documentation

Release 6.5.1.dev2+gb4d2269.d20220120

Philippe Pepiot

Jan 20, 2022

Contents

Latest documentation: https://testinfra.readthedocs.io/en/latest

CHAPTER 1

About

With Testinfra you can write unit tests in Python to test *actual state* of your servers configured by management tools like Salt, Ansible, Puppet, Chef and so on.

Testinfra aims to be a Serverspec equivalent in python and is written as a plugin to the powerful Pytest test engine

CHAPTER 2

License

Apache License 2.0

The logo is licensed under the Creative Commons NoDerivatives 4.0 License If you have some other use in mind, contact us.

CHAPTER 3

Quick start

Install testinfra using pip:

```
$ pip install pytest-testinfra

# or install the devel version
$ pip install 'git+https://github.com/pytest-dev/pytest-testinfra@master#egg=pytest-
↪testinfra'
```

Write your first tests file to *test_myinfra.py*:

```
def test_passwd_file(host):
    passwd = host.file("/etc/passwd")
    assert passwd.contains("root")
    assert passwd.user == "root"
    assert passwd.group == "root"
    assert passwd.mode == 0o644


def test_nginx_is_installed(host):
    nginx = host.package("nginx")
    assert nginx.is_installed
    assert nginx.version.startswith("1.2")


def test_nginx_running_and_enabled(host):
    nginx = host.service("nginx")
    assert nginx.is_running
    assert nginx.is_enabled
```

And run it:

```
$ py.test -v test_myinfra.py
```

(continues on next page)

(continued from previous page)

```
====================== test session starts ======================
platform linux -- Python 2.7.3 -- py-1.4.26 -- pytest-2.6.4
plugins: testinfra
collected 3 items

test_myinfra.py::test_passwd_file[local] PASSED
test_myinfra.py::test_nginx_is_installed[local] PASSED
test_myinfra.py::test_nginx_running_and_enabled[local] PASSED

=================== 3 passed in 0.66 seconds ====================
```

CHAPTER 4

Documentation

4.1 Changelog

4.1.1 6.5.0

- Fallback to which when “command -v” fails
- Use realpath by default to resolve symlinks instead of “readlink -f”
- ansible: Support environment variables
- Force package module to resolve to RpmPackage on Fedora
- Fix new versions of supervisor may exit with status != 0
- Eventually decode ansible output when it’s not ascii
- Either use python3 or python to get remote encoding

4.1.2 6.4.0

- Implement Interface names and default (#615)
- Implement Service.systemd_properties (#612)

4.1.3 6.3.0

- Fix #451 for use with pytest -p no:terminal
- Add client_version() and server_version() and version() to docker module.

4.1.4 6.2.0

- Fix #590: Systeminfo doesn't resolve Windows correctly (#592)
- First implementation of network namespaces in addr module (#596)
- pip check support in PipPackage module (#605)
- pip refactoring: implementation of installed and version (#606)
- Allow to specify supervisorctl and supervisord.conf paths (#536)

4.1.5 6.1.0

- Fix wrong package module on CentOS having dpkg tools installed #570 (#575)
- Deduplicate hosts returned by get_backends() (#572)
- Use /run/systemd/system/ to detect systemd (fixes #546)
- Use ssh_args from ansible.cfg
- Require python >= 3.6
- Fix ValueError with python 3.8+ when using –nagios option.

4.1.6 6.0.0

- Breaking change: testinfra has moved to the https://github.com/pytest-dev/ organization. Project on PyPi is renamed as pytest-testinfra. A dummy testinfra will make the transition, but you should rename to pytest-testinfra in your requirements files.

4.1.7 5.3.1

- Fix newly introduced is_masked property on systemd service https://github.com/philpep/testinfra/pull/569

4.1.8 5.3.0

- Add is_masked property on systemd service

4.1.9 5.2.2

- iptables: use -w option to wait for iptables lock when running in parallel with pytest-xdist.

4.1.10 5.2.1

- Fix documentation build

4.1.11 5.2.0

- Allow kubeconfig context to be supplied in kubernetes backend
- Drop file.__ne__ implementation and require python >= 3.5

4.1.12 5.1.0

- Use remote_user and remote_port in ansible.cfg
- Add *arch* (architecture) attribute to system_info module

4.1.13 5.0.0

- Breaking change: host.file().listdir() is now a method

4.1.14 4.1.0

- Pass extra arguments to ansible CLI via host.ansible()
- New method host.file.listdir() to list items in a directory.

4.1.15 4.0.0

- Drop python2 support

4.1.16 3.4.0

- Add podman backend and module
- WARNING: this will be the latest testinfra version supporting python2, please upgrade to python3.

4.1.17 3.3.0

- Add extras for backend dependencies (#454)
- Various enhencements of kitchen integration documentation
- ansible backend now support "password" field from ansible inventory
- New backend "openshift"

4.1.18 3.2.1

- Fix Process module when working with long strings (username, . . .) #505

4.1.19 3.2.0

- New module "environment" for getting remote environment variables
- New module "block_device" exposing block device informations
- Add a global flag –force-ansible to the command line
- Raise an error in case of missing ansible inventory file
- Fix an escape issue with ansible ssh args set inventory or configuration file

4.1.20 3.1.0

- ssh connections uses persistent connections by default. You can disable this by passing controlpersist=0 to the connections options.
- ansible ssh connections now use ssh backend instead of paramiko. ansible_ssh_common_args and ansible_ssh_extra_args are now taking in account.
- Add a new ansible connection options "force_ansible", when set to True, testinfra will always call ansible for all commands he need to run.
- Handle all ansible connections types by setting force_ansible=True for connections which doesn't have a testinfra equivalent connection (for example "network_cli").

4.1.21 3.0.6

- Issue full command logging using DEBUG log level to avoid logging sensible data when log level is INFO.
- Fix possible crash when parsing ansible inventories #470
- Support using alternative kubeconfig file in kubectl connections #460
- Support parsing ProxyCommand from ssh_config for paramiko connections

4.1.22 3.0.5

- Set default timeout to 10s on ssh/paramiko connections
- Add support for ansible inventory parameter ansible_private_key_file

4.1.23 3.0.4

- Add support for ansible lxc and lxd connections

4.1.24 3.0.3

- Fix paramiko parsing RequestTTY from ssh configs
- Re-add "groups" key from ansible.get_variables() to be backward compatible with testinfra 2.X

4.1.25 3.0.2

- Fix ansible with no inventory resolving to "localhost"
- Fix support for ansible 2.8 with no inventory
- Fix ansible/paramiko which wasn't reading hosts config from ~/.ssh/config
- Allow to pass –ssh-config and –ssh-identity-file to ansible connection

4.1.26 3.0.1

- Fix parsing of ipv6 adresses for paramiko, ssh and ansible backends.
- Fix –connection=ansible invocation when no hosts are provided

4.1.27 3.0.0

- New ansible backend fixing support for ansible 2.8 and license issue. See https://github.com/philpep/testinfra/issues/431 for details. This make ansible using testinfra native backends and only works for local, ssh or docker connections. I you have others connection types or issues, please open a bug on https://github.com/philpep/testinfra/issues/new
- Windows support is improved. "package" module is handled with Chocolatey and there's support for the "user" module.

4.1.28 2.1.0

- docker: new get_containers() classmethod
- socket: fix parsing of ipv6 addresses with new versions of ss
- service: systemd fallback to sysv when "systemctl is-active" is not working

4.1.29 2.0.0

- Add addr module, used to test network connectivity
- Drop deprecated "testinfra" command, you should use "py.test" instead
- Drop deprecated top level fixtures, access them through the fixture "host" instead.
- Drop support for ansible <= 2.4

4.1.30 1.19.0

- Add docker module
- Fix pytest 4 compatibility

4.1.31 1.18.0

- Allow to urlencode character in host specification "user:pass@host" (#387)
- Fix double logging from both pytest and testinfra
- Drop support for python 2.6
- Allow to configure timeouts for winrm backend

4.1.32 1.17.0

- Add support for ansible "become" user in ansible module
- Add failed/suceeded property on run() output

4.1.33 1.16.0

- packaging: Use setuptools_scm instead of pbr
- iptables: add ip6tables support
- sysctl: find sysctl outside of PATH (/sbin)

4.1.34 1.15.0

- Fix finding ss and netstat command in "sbin" paths for Centos (359)
- Add a workaround for https://github.com/pytest-dev/pytest/issues/3542
- Handle "starting" status for Service module on Alpine linux
- Fix no_ssl and no_verify_ssl options for WinRM backend

4.1.35 1.14.1

- Fix multi-host test ordering (#347), regression introduced in 1.13.1
- Fix Socket on OpenBSD hosts (#338)

4.1.36 1.14.0

- Add a new lxc backend
- Socket: fix is_listening for unix sockets
- Add namespace and container support for kubernetes backend
- Add a cache of parsed ansible inventories for ansible backend
- Service: fix service detection on Centos 6 hosts
- File: implement file comparison with string paths

4.1.37 1.13.1

- package: fix is_installed and version behavior for uninstalled packages (#321 and #326)
- ansible: Use predictibles test ordering when using pytest-xdist to fix random test collections errors (#316)

4.1.38 1.13.0

- socket: fix detection of udp listening sockets (#311)
- ssh backend: Add support for GSSAPI

4.1.39 1.12.0

- ansible: fix compatibility with ansible 2.5
- pip: fix compatibility with pip 10 (#299)

4.1.40 1.11.1

- Socket: fix error with old versions of ss without the –no-header option (#293)

4.1.41 1.11.0

- Fix bad error reporting when using ansible module without ansible backend (#288)
- Socket: add a new implementation using ss instead of netstat (#124)
- Add service, process, and systeminfo support for Alpine (#283)

4.1.42 1.10.1

- Fix get_variables() for ansible>=2.0,<2.4 (#274)
- Paramiko: Use the RequireTTY setting if specified in a provided SSHConfig (#247)

4.1.43 1.10.0

- New iptables module

4.1.44 1.9.1

- Fix running testinfra within a suite using doctest (#268)
- Service: add is_valid method for systemd
- Fix file.linked_to() for Mac OS

4.1.45 1.9.0

- Interface: allow to find 'ip' command ousite of PATH
- Fix –nagios option with python 3

4.1.46 1.8.0

- Deprecate testinfra command (will be dropped in 2.0), use py.test instead #135
- Handle –nagios option when using py.test command

4.1.47 1.7.1

- Support for ansible 2.4 (#249)

4.1.48 1.7.0

- Salt: allow specifiy config directory (#230)
- Add a WinRM backend
- Socket: ipv6 sockets can handle ipv4 clients (#234)
- Service: Enhance upstart detection (#243)

4.1.49 1.6.5

- Service: add is_enabled() support for OpenBSD
- Add ssh identity file option for paramiko and ssh backends
- Expand tilde (~) to user home directory for ssh-config, ssh-identity-file and ansible-inventory options

4.1.50 1.6.4

- Service: Allow to find 'service' command outside of $PATH #211
- doc fixes

4.1.51 1.6.3

- Fix unwanted deprecation warning when running tests with pytest 3.1 #204

4.1.52 1.6.2

- Fix wheel package for 1.6.1

4.1.53 1.6.1

- Support ansible 2.3 with python 3 (#197)

4.1.54 1.6.0

- New 'host' fixture as a replacement for all other fixtures. See https://testinfra.readthedocs.io/en/latest/modules.html#host (Other fixtures are deprecated and will be removed in 2.0 release).

4.1.55 1.5.5

- backends: Fix ansible backend with ansible >= 2.3 (#195)

4.1.56 1.5.4

- backends: fallback to UTF-8 encoding when system encoding is ASCII.
- Service: fix is_running() on systems using Upstart

4.1.57 1.5.3

- Sudo: restore backend command in case of exceptions

4.1.58 1.5.2

- Honnor become_user when using the ansible backend

4.1.59 1.5.1

- Add dependency on importlib on python 2.6

4.1.60 1.5.0

- New kubectl backend
- Command: check_output strip carriage return and newlines (#164)
- Package: rpm improve getting version() and release()
- User: add gecos (comment) field (#155)

4.1.61 1.4.5

- SystemInfo: detect codename from VERSION_CODENAME in /etc/os-release (fallback when lsb_release isn't installed).
- Package: add release property for rpm based systems.

4.2 Invocation

4.2.1 Test multiples hosts

By default Testinfra launch tests on local machine, but you can also test remotes systems using paramiko (a ssh implementation in python):

```
$ pip install paramiko
$ py.test -v --hosts=localhost,root@webserver:2222 test_myinfra.py

====================== test session starts ======================
platform linux -- Python 2.7.3 -- py-1.4.26 -- pytest-2.6.4
plugins: testinfra
collected 3 items

test_myinfra.py::test_passwd_file[localhost] PASSED
test_myinfra.py::test_nginx_is_installed[localhost] PASSED
test_myinfra.py::test_nginx_running_and_enabled[localhost] PASSED
test_myinfra.py::test_passwd_file[root@webserver:2222] PASSED
test_myinfra.py::test_nginx_is_installed[root@webserver:2222] PASSED
test_myinfra.py::test_nginx_running_and_enabled[root@webserver:2222] PASSED

================== 6 passed in 8.49 seconds ==================
```

You can also set hosts per test module:

```
testinfra_hosts = ["localhost", "root@webserver:2222"]

def test_foo(host):
    [....]
```

4.2.2 Parallel execution

If you have a lot of tests, you can use the pytest-xdist plugin to run tests using multiples process:

```
$ pip install pytest-xdist

# Launch tests using 3 processes
$ py.test -n 3 -v --host=web1,web2,web3,web4,web5,web6 test_myinfra.py
```

4.2.3 Advanced invocation

```
# Test recursively all test files (starting with `test_`) in current directory
$ py.test

# Filter function/hosts with pytest -k option
$ py.test --hosts=webserver,dnsserver -k webserver -k nginx
```

For more usages and features, see the Pytest documentation.

4.3 Connection backends

Testinfra comes with several connections backends for remote command execution.

When installing, you should select the backends you require as `extras` to ensure Python dependencies are satisifed (note various system packaged tools may still be required). For example

```
$ pip install testinfra[ansible,salt]
```

For all backends, commands can be run as superuser with the `--sudo` option or as specific user with the `--sudo-user` option.

4.3.1 local

This is the default backend when no hosts are provided (either via `--hosts` or in modules). Commands are run locally in a subprocess under the current user:

```
$ py.test --sudo test_myinfra.py
```

4.3.2 paramiko

This is the default backend when a hosts list is provided. Paramiko is a Python implementation of the SSHv2 protocol. Testinfra will not ask you for a password, so you must be able to connect without password (using passwordless keys or using `ssh-agent`).

You can provide an alternate ssh-config:

```
$ py.test --ssh-config=/path/to/ssh_config --hosts=server
```

4.3.3 docker

The Docker backend can be used to test *running* Docker containers. It uses the docker exec command:

```
$ py.test --hosts='docker://[user@]container_id_or_name'
```

See also the *Test Docker images* example.

4.3.4 podman

The Podman backend can be used to test *running* Podman containers. It uses the podman exec command:

```
$ py.test --hosts='podman://[user@]container_id_or_name'
```

4.3.5 ssh

This is a pure SSH backend using the `ssh` command. Example:

```
$ py.test --hosts='ssh://server'
$ py.test --ssh-config=/path/to/ssh_config --hosts='ssh://server'
$ py.test --ssh-identity-file=/path/to/key --hosts='ssh://server'
$ py.test --hosts='ssh://server?timeout=60&controlpersist=120'
```

By default timeout is set to 10 seconds and ControlPersist is set to 60 seconds. You can disable persistent connection by passing *controlpersist=0* to the options.

4.3.6 salt

The salt backend uses the salt Python client API and can be used from the salt-master server:

```
$ py.test --hosts='salt://*'
$ py.test --hosts='salt://minion1,salt://minion2'
$ py.test --hosts='salt://web*'
$ py.test --hosts='salt://G@os:Debian'
```

Testinfra will use the salt connection channel to run commands.

Hosts can be seleted by using the *glob* and compound matchers.

4.3.7 ansible

The ansible backend is able to parse ansible inventories to get host connection details. For local, ssh, paramiko or docker connections(based on *ansible_connection* value) it will use the equivalent testinfra connection backend, unless *force_ansible=True* (or `--force-ansible`) is set.

For other connections types or when *force_ansible=True*, testinfra will run all commands through ansible, which is substantially slower than using native connections backends.

If ssh identity file is not provided via *–ssh-identity-file* flag, testinfra will try to use *ansible_ssh_private_key_file*, *ansible_private_key_file* and, finally, *ansible_user* with *ansible_ssh_pass* variables, both should be specified.

Examples:

```
$ py.test --hosts='ansible://all' # tests all inventory hosts
$ py.test --hosts='ansible://host1,ansible://host2'
$ py.test --hosts='ansible://web*'
$ py.test --force-ansible --hosts='ansible://all'
$ py.test --hosts='ansible://host?force_ansible=True'
```

4.3.8 kubectl

The kubectl backend can be used to test containers running in Kubernetes. It uses the kubectl exec command and support connecting to a given container name within a pod and using a given namespace:

```
# will use the default namespace and default container
$ py.test --hosts='kubectl://mypod-a1b2c3'
# specify container name and namespace
$ py.test --hosts='kubectl://somepod-2536ab?container=nginx&namespace=web'
# specify the kubeconfig context to use
$ py.test --hosts='kubectl://somepod-2536ab?context=k8s-cluster-a&container=nginx'
# you can specify kubeconfig either from KUBECONFIG environment variable
# or when working with multiple configuration with the "kubeconfig" option
$ py.test --hosts='kubectl://somepod-123?kubeconfig=/path/kubeconfig,kubectl://
↪otherpod-123?kubeconfig=/other/kubeconfig'
```

4.3.9 openshift

The openshift backend can be used to test containers running in OpenShift. It uses the oc exec command and support connecting to a given container name within a pod and using a given namespace:

```
# will use the default namespace and default container
$ py.test --hosts='openshift://mypod-a1b2c3'
# specify container name and namespace
$ py.test --hosts='openshift://somepod-2536ab?container=nginx&namespace=web'
# you can specify kubeconfig either from KUBECONFIG environment variable
# or when working with multiple configuration with the "kubeconfig" option
$ py.test --hosts='openshift://somepod-123?kubeconfig=/path/kubeconfig,openshift://
↪otherpod-123?kubeconfig=/other/kubeconfig'
```

4.3.10 winrm

The winrm backend uses pywinrm:

```
$ py.test --hosts='winrm://Administrator:Password@127.0.0.1'
$ py.test --hosts='winrm://vagrant@127.0.0.1:2200?no_ssl=true&no_verify_ssl=true'
```

pywinrm's default read and operation timeout can be overridden using query arguments `read_timeout_sec` and `operation_timeout_sec`:

```
$ py.test --hosts='winrm://vagrant@127.0.0.1:2200?read_timeout_sec=120&operation_
↪timeout_sec=100'
```

4.3.11 LXC/LXD

The LXC backend can be used to test *running* LXC or LXD containers. It uses the lxc exec command:

```
$ py.test --hosts='lxc://container_name'
```

4.4 Modules

Testinfra modules are provided through the *host* fixture, declare it as arguments of your test function to make it available within it.

```
def test_foo(host):
    # [...]
```

4.4.1 host

class Host(*backend*)

ansible
testinfra.modules.ansible.Ansible class

addr
testinfra.modules.addr.Addr class

blockdevice
testinfra.modules.blockdevice.BlockDevice class

docker
testinfra.modules.docker.Docker class

environment
testinfra.modules.environment.Environment class

file
testinfra.modules.file.File class

group
testinfra.modules.group.Group class

interface
testinfra.modules.interface.Interface class

iptables
testinfra.modules.iptables.Iptables class

mount_point
testinfra.modules.mountpoint.MountPoint class

package
testinfra.modules.package.Package class

pip
testinfra.modules.pip.Pip class

pip_package
testinfra.modules.pip.PipPackage class

podman
testinfra.modules.podman.Podman class

process
testinfra.modules.process.Process class

puppet_resource
testinfra.modules.puppet.PuppetResource class

facter
testinfra.modules.puppet.Facter class

salt
testinfra.modules.salt.Salt class

service
testinfra.modules.service.Service class

socket
testinfra.modules.socket.Socket class

sudo
testinfra.modules.sudo.Sudo class

supervisor
testinfra.modules.supervisor.Supervisor class

sysctl
testinfra.modules.sysctl.Sysctl class

system_info
testinfra.modules.systeminfo.SystemInfo class

user
testinfra.modules.user.User class

exists(*command*)
Return True if given command exist in $PATH

find_command(*command*, *extrapaths=('/sbin', '/usr/sbin')*)
Return path of given command

raise ValueError if command cannot be found

run(*command*, **args*, ***kwargs*)
Run given command and return rc (exit status), stdout and stderr

```
>>> cmd = host.run("ls -l /etc/passwd")
>>> cmd.rc
0
>>> cmd.stdout
'-rw-r--r-- 1 root root 1790 Feb 11 00:28 /etc/passwd\n'
>>> cmd.stderr
''
>>> cmd.succeeded
True
>>> cmd.failed
False
```

Good practice: always use shell arguments quoting to avoid shell injection

```
>>> cmd = host.run("ls -l -- %s", "/;echo inject")
CommandResult(
    rc=2, stdout='',
    stderr=(
      'ls: cannot access /;echo inject: No such file or directory\n'),
    command="ls -l '/;echo inject'")
```

run_expect (*expected*, *command*, **args*, ***kwargs*)
Run command and check it return an expected exit status

Parameters **expected** – A list of expected exit status

Raises AssertionError

run_test (*command*, **args*, ***kwargs*)
Run command and check it return an exit status of 0 or 1

Raises AssertionError

check_output (*command*, **args*, ***kwargs*)
Get stdout of a command which has run successfully

Returns stdout without trailing newline

Raises AssertionError

classmethod get_host (*hostspec*, ***kwargs*)
Return a Host instance from *hostspec*

hostspec should be like *<backend_type>://<name>?param1=value1&param2=value2*

Params can also be passed in ***kwargs* (eg. get_host("local://", sudo=True) is equivalent to get_host("local://?sudo=true"))

Examples:

```
>>> get_host("local://", sudo=True)
>>> get_host("paramiko://user@host", ssh_config="/path/my_ssh_config")
>>> get_host("ansible://all?ansible_inventory=/etc/ansible/inventory")
```

4.4.2 Ansible

class Ansible (*module_name*, *module_args=None*, *check=True*)
Run Ansible module functions

This module is only available with the *ansible* connection backend.

Check mode is enabled by default, you can disable it with *check=False*.

Become is *False* by default. You can enable it with *become=True*.

Ansible arguments that are not related to the Ansible inventory or connection (both managed by testinfra) are also accepted through keyword arguments:

- `become_method` *str* sudo, su, doas, etc.
- `become_user` *str* become this user.
- `diff` *bool*: when changing (small) files and templates, show the differences in those files.
- `extra_vars` *dict* serialized to a JSON string, passed to Ansible.
- `one_line` *bool*: condense output.

- `user` *str* connect as this user.
- `verbose` *int* level of verbosity

```
>>> host.ansible("apt", "name=nginx state=present")["changed"]
False
>>> host.ansible("apt", "name=nginx state=present", become=True)["changed"]
False
>>> host.ansible("command", "echo foo", check=False)["stdout"]
'foo'
>>> host.ansible("setup")["ansible_facts"]["ansible_lsb"]["codename"]
'jessie'
>>> host.ansible("file", "path=/etc/passwd")["mode"]
'0640'
>>> host.ansible(
... "command",
... "id --user --name",
... check=False,
... become=True,
... become_user="http",
... )["stdout"]
'http'
>>> host.ansible(
... "apt",
... "name={{ packages }}",
... check=False,
... extra_vars={"packages": ["neovim", "vim"]},
... )
# Installs neovim and vim.
```

exception AnsibleException(*result*)

Exception raised when an error occur in an ansible call

result from ansible can be accessed through the `result` attribute

```
>>> try:
...     host.ansible("command", "echo foo")
... except host.ansible.AnsibleException as exc:
...     assert exc.result['failed'] is True
...     assert exc.result['msg'] == 'Skipped. You might want to try␣
↪check=False'  # noqa
```

get_variables()

Returns a dict of ansible variables

```
>>> host.ansible.get_variables()
{
    'inventory_hostname': 'localhost',
    'group_names': ['ungrouped'],
    'foo': 'bar',
}
```

4.4.3 Addr

class Addr(*name*)

Test remote address

Example:

```
>>> google = host.addr("google.com")
>>> google.is_resolvable
True
>>> '173.194.32.225' in google.ipv4_addresses
True
>>> google.is_reachable
True
>>> google.port(443).is_reachable
True
>>> google.port(666).is_reachable
False
```

Can also be use within a network namespace.

```
>>> localhost = host.addr("localhost", "ns1")
>>> localhost.is_resolvable
True
```

Network namespaces can only be used if ip command is available because in this case, the module use ip-netns as command prefix. In the other case, it will raise NotImplementedError.

name
Return host name

namespace
Return network namespace

namespace_exists
Test if the network namespace exists

is_resolvable
Return if address is resolvable

is_reachable
Return if address is reachable

ip_addresses
Return IP addresses of host

ipv4_addresses
Return IPv4 addresses of host

ipv6_addresses
Return IPv6 addresses of host

port(*port*)
Return address-port pair

4.4.4 BlockDevice

class BlockDevice(*name*)
Information for block device.

Should be used with sudo or under root.

If device is not a block device, RuntimeError is raised.

is_partition
Return True if the device is a partition.

```
>>> host.block_device("/dev/sda1").is_partition
True
```

```
>>> host.block_device("/dev/sda").is_partition
False
```

size
Return size if the device in bytes.

```
>>> host.block_device("/dev/sda1").size
512110190592
```

sector_size
Return sector size for the device in bytes.

```
>>> host.block_device("/dev/sda1").sector_size
512
```

block_size
Return block size for the device in bytes.

```
>>> host.block_device("/dev/sda").block_size
4096
```

start_sector
Return start sector of the device on the underlying device.

Usually the value is zero for full devices and is non-zero for partitions.

```
>>> host.block_device("/dev/sda1").start_sector
2048
```

```
>>> host.block_device("/dev/md0").start_sector
0
```

is_writable
Return True if device is writable (have no RO status)

```
>>> host.block_device("/dev/sda").is_writable
True
```

```
>>> host.block_device("/dev/loop1").is_writable
False
```

ra
Return Read Ahead for the device in 512-bytes sectors.

```
>>> host.block_device("/dev/sda").ra
256
```

4.4.5 Docker

class Docker(*name*)
Test docker containers running on system.

Example:

```
>>> nginx = host.docker("app_nginx")
>>> nginx.is_running
True
>>> nginx.id
'7e67dc7495ca8f451d346b775890bdc0fb561ecdc97b68fb59ff2f77b509a8fe'
>>> nginx.name
'app_nginx'
```

classmethod client_version()
Docker client version

classmethod server_version()
Docker server version

classmethod version(*format=None*)
Docker version with an optional format (Go template).

```
>>> host.docker.version()
Client: Docker Engine - Community
...
>>> host.docker.version("{{.Client.Context}}"))
default
```

classmethod get_containers(***filters*)
Return a list of containers

By default return list of all containers, including non-running containers.

Filtering can be done using filters keys defined on https://docs.docker.com/engine/reference/commandline/ps/#filtering

Multiple filters for a given key is handled by giving a list of string as value.

```
>>> host.docker.get_containers()
[<docker nginx>, <docker redis>, <docker app>]
# Get all running containers
>>> host.docker.get_containers(status="running")
[<docker app>]
# Get containers named "nginx"
>>> host.docker.get_containers(name="nginx")
[<docker nginx>]
# Get containers named "nginx" or "redis"
>>> host.docker.get_containers(name=["nginx", "redis"])
[<docker nginx>, <docker redis>]
```

4.4.6 File

class File(*path*)
Test various files attributes

exists
Test if file exists

```
>>> host.file("/etc/passwd").exists
True
```

(continues on next page)

(continued from previous page)

```
>>> host.file("/nonexistent").exists
False
```

is_file

is_directory

is_pipe

is_socket

is_symlink

linked_to
Resolve symlink

```
>>> host.file("/var/lock").linked_to
'/run/lock'
```

user
Return file owner as string

```
>>> host.file("/etc/passwd").user
'root'
```

uid
Return file user id as integer

```
>>> host.file("/etc/passwd").uid
0
```

group

gid

mode
Return file mode as octal integer

```
>>> host.file("/etc/shadow").mode
416  # 0o640 octal
>>> host.file("/etc/shadow").mode == 0o640
True
>>> oct(host.file("/etc/shadow").mode) == '0o640'
True
```

You can also utilize the file mode constants from the stat library for testing file mode.

```
>>> import stat
>>> host.file("/etc/shadow").mode == stat.S_IRUSR | stat.S_IWUSR | stat.S_
↪IRGRP
True
```

contains (*pattern*)

md5sum

sha256sum

content
Return file content as bytes

```
>>> host.file("/tmp/foo").content
b'caf\xc3\xa9'
```

content_string
Return file content as string

```
>>> host.file("/tmp/foo").content_string
'café'
```

mtime
Return time of last modification as datetime.datetime object

```
>>> host.file("/etc/passwd").mtime
datetime.datetime(2015, 3, 15, 20, 25, 40)
```

size
Return size of file in bytes

listdir()
Return list of items under the directory

```
>>> host.file("/tmp").listdir()
['foo_file', 'bar_dir']
```

4.4.7 Group

class Group(*name=None*)
Test unix group

exists
Test if group exists

```
>>> host.group("wheel").exists
True
>>> host.group("nosuchgroup").exists
False
```

gid

4.4.8 Interface

class Interface(*name*, *family=None*)
Test network interfaces

```
>>> host.interface("eth0").exists
True
```

Optionally, the protocol family to use can be enforced.

```
>>> host.interface("eth0", "inet6").addresses
['fe80::e291:f5ff:fe98:6b8c']
```

exists

speed

addresses
Return ipv4 and ipv6 addresses on the interface

```
>>> host.interface("eth0").addresses
['192.168.31.254', '192.168.31.252', 'fe80::e291:f5ff:fe98:6b8c']
```

classmethod names()
Return the names of all the interfaces.

```
>>> host.interface.names()
['lo', 'tunl0', 'ip6tnl0', 'eth0']
```

classmethod default(*family=None*)
Return the interface used for the default route.

```
>>> host.interface.default()
<interface eth0>
```

Optionally, the protocol family to use can be enforced.

```
>>> host.interface.default("inet6")
None
```

4.4.9 Iptables

class Iptables
Test iptables rule exists

rules(*table='filter'*, *chain=None*, *version=4*)
Returns list of iptables rules

Based on ouput of *iptables -t TABLE -S CHAIN* command

optionally takes takes the following arguments:

- table: defaults to *filter*
- chain: defaults to all chains
- version: default 4 (iptables), optionally 6 (ip6tables)

```
>>> host.iptables.rules()
[
    '-P INPUT ACCEPT',
    '-P FORWARD ACCEPT',
    '-P OUTPUT ACCEPT',
    '-A INPUT -i lo -j ACCEPT',
    '-A INPUT -j REJECT'
    '-A FORWARD -j REJECT'
]
>>> host.iptables.rules("nat", "INPUT")
['-P PREROUTING ACCEPT']
```

4.4.10 MountPoint

class MountPoint(*path*)
Test Mount Points

exists
Return True if the mountpoint exists

```
>>> host.mount_point("/").exists
True
```

```
>>> host.mount_point("/not/a/mountpoint").exists
False
```

filesystem
Returns the filesystem type associated

```
>>> host.mount_point("/").filesystem
'ext4'
```

device
Return the device associated

```
>>> host.mount_point("/").device
'/dev/sda1'
```

options
Return a list of options that a mount point has been created with

```
>>> host.mount_point("/").options
['rw', 'relatime', 'data=ordered']
```

classmethod get_mountpoints()
Returns a list of MountPoint instances

```
>>> host.mount_point.get_mountpoints()
[<MountPoint(path=/proc, device=proc, filesystem=proc, options=rw,nosuid,
↪nodev,noexec,relatime)>
 <MountPoint(path=/, device=/dev/sda1, filesystem=ext4, options=rw,relatime,
↪errors=remount-ro,data=ordered)>]
```

4.4.11 Package

class Package(*name*)
Test packages status and version

is_installed
Test if the package is installed

```
>>> host.package("nginx").is_installed
True
```

Supported package systems:

- apk (Alpine)
- apt (Debian, Ubuntu, . . .)
- pacman (Arch)
- pkg (FreeBSD)
- pkg_info (NetBSD)

- pkg_info (OpenBSD)
- rpm (RHEL, Centos, Fedora, . . .)

release
Return the release specific info from the package version

```
>>> host.package("nginx").release
'1.el6'
```

version
Return package version as returned by the package system

```
>>> host.package("nginx").version
'1.2.1-2.2+wheezy3'
```

4.4.12 Pip

class Pip(*name*, *pip_path='pip'*)
Test pip package manager and packages

is_installed
Test if the package is installed

```
>>> host.package("pip").is_installed
True
```

version
Return package version as returned by pip

```
>>> host.package("pip").version
'18.1'
```

classmethod check(*pip_path='pip'*)
Verify installed packages have compatible dependencies.

```
>>> cmd = host.pip_package.check()
>>> cmd.rc
0
>>> cmd.stdout
No broken requirements found.
```

Can only be used if pip check command is available, for pip versions >= 9.0.0.

classmethod get_packages(*pip_path='pip'*)
Get all installed packages and versions returned by *pip list*:

```
>>> host.pip_package.get_packages(pip_path='~/venv/website/bin/pip')
{'Django': {'version': '1.10.2'},
 'mywebsite': {'version': '1.0a3', 'path': '/srv/website'},
 'psycopg2': {'version': '2.6.2'}}
```

classmethod get_outdated_packages(*pip_path='pip'*)
Get all outdated packages with current and latest version

```
>>> host.pip_package.get_outdated_packages(
...     pip_path='~/venv/website/bin/pip')
{'Django': {'current': '1.10.2', 'latest': '1.10.3'}}
```

4.4.13 PipPackage

class PipPackage(*name*, *pip_path='pip'*)

Deprecated since version 6.2.

Use *Pip* instead.

4.4.14 Podman

class Podman(*name*)

Test podman containers running on system.

Example:

```
>>> nginx = host.podman("app_nginx")
>>> nginx.is_running
True
>>> nginx.id
'7e67dc7495ca8f451d346b775890bdc0fb561ecdc97b68fb59ff2f77b509a8fe'
>>> nginx.name
'app_nginx'
```

classmethod get_containers(***filters*)

Return a list of containers

By default return list of all containers, including non-running containers.

Filtering can be done using filters keys defined in podman-ps(1).

Multiple filters for a given key is handled by giving a list of string as value.

```
>>> host.podman.get_containers()
[<podman nginx>, <podman redis>, <podman app>]
# Get all running containers
>>> host.podman.get_containers(status="running")
[<podman app>]
# Get containers named "nginx"
>>> host.podman.get_containers(name="nginx")
[<podman nginx>]
# Get containers named "nginx" or "redis"
>>> host.podman.get_containers(name=["nginx", "redis"])
[<podman nginx>, <podman redis>]
```

4.4.15 Process

class Process

Test Processes attributes

Processes are selected using `filter()` or `get()`, attributes names are described in the ps(1) man page.

```
>>> master = host.process.get(user="root", comm="nginx")
# Here is the master nginx process (running as root)
>>> master.args
'nginx: master process /usr/sbin/nginx -g daemon on; master_process on;'
# Here are the worker processes (Parent PID = master PID)
>>> workers = host.process.filter(ppid=master.pid)
>>> len(workers)
4
# Nginx don't eat memory
>>> sum([w.pmem for w in workers])
0.8
# But php does !
>>> sum([p.pmem for p in host.process.filter(comm="php5-fpm")])
19.2
```

filter(***filters*)
Get a list of matching process

```
>>> host.process.filter(user="root", comm="zsh")
[<process zsh (pid=2715)>, <process zsh (pid=10502)>, ...]
```

get(***filters*)
Get one matching process

Raise `RuntimeError` if no process found or multiple process matching filters.

4.4.16 PuppetResource

class PuppetResource(*type*, *name=None*)
Get puppet resources

Run `puppet resource --types` to get a list of available types.

```
>>> host.puppet_resource("user", "www-data")
{
    'www-data': {
        'ensure': 'present',
        'comment': 'www-data',
        'gid': '33',
        'home': '/var/www',
        'shell': '/usr/sbin/nologin',
        'uid': '33',
    },
}
```

4.4.17 Facter

class Facter(**facts*)
Get facts with facter

```
>>> host.facter()
{
    "operatingsystem": "Debian",
    "kernel": "linux",
    [...]
```

(continues on next page)

(continued from previous page)

```
}
>>> host.facter("kernelversion", "is_virtual")
{
  "kernelversion": "3.16.0",
  "is_virtual": "false"
}
```

4.4.18 Salt

class Salt (*function*, *args=None*, *local=False*, *config=None*)

Run salt module functions

```
>>> host.salt("pkg.version", "nginx")
'1.6.2-5'
>>> host.salt("pkg.version", ["nginx", "php5-fpm"])
{'nginx': '1.6.2-5', 'php5-fpm': '5.6.7+dfsg-1'}
>>> host.salt("grains.item", ["osarch", "mem_total", "num_cpus"])
{'osarch': 'amd64', 'num_cpus': 4, 'mem_total': 15520}
```

Run `salt-call sys.doc` to get a complete list of functions

4.4.19 Service

class Service (*name*)

Test services

Implementations:

- Linux: detect Systemd, Upstart or OpenRC, fallback to SysV
- FreeBSD: service(1)
- OpenBSD: `/etc/rc.d/$name check` for `is_running` `rcctl ls on` for `is_enabled` (only OpenBSD >= 5.8)
- NetBSD: `/etc/rc.d/$name onestatus` for `is_running` (`is_enabled` is not yet implemented)

is_running

Test if service is running

is_enabled

Test if service is enabled

is_valid

Test if service is valid

This method is only available in the systemd implementation, it will raise `NotImplementedError` in others implementation

is_masked

Test if service is masked

This method is only available in the systemd implementation, it will raise `NotImplementedError` in others implementations

systemd_properties
Properties of the service (unit).

Return service properties as a *dict*, empty properties are not returned.

```
>>> ntp = host.service("ntp")
>>> ntp.systemd_properties["FragmentPath"]
'/lib/systemd/system/ntp.service'
```

This method is only available in the systemd implementation, it will raise `NotImplementedError` in others implementations

Note: based on systemctl show

4.4.20 Socket

class Socket(*socketspec*)
Test listening tcp/udp and unix sockets

`socketspec` must be specified as `<protocol>://<host>:<port>`

This module requires the `netstat` command to on the target host.

Example:

- Unix sockets: `unix:///var/run/docker.sock`
- All ipv4 and ipv6 tcp sockets on port 22: `tcp://22`
- All ipv4 sockets on port 22: `tcp://0.0.0.0:22`
- All ipv6 sockets on port 22: `tcp://:::22`
- udp socket on 127.0.0.1 port 69: `udp://127.0.0.1:69`

is_listening
Test if socket is listening

```
>>> host.socket("unix:///var/run/docker.sock").is_listening
False
>>> # This HTTP server listen on all ipv4 addresses but not on ipv6
>>> host.socket("tcp://0.0.0.0:80").is_listening
True
>>> host.socket("tcp://:::80").is_listening
False
>>> host.socket("tcp://80").is_listening
False
```

Note: If you don't specify a host for udp and tcp sockets, then the socket is listening if and only if the socket listen on **both** all ipv4 and ipv6 addresses (ie 0.0.0.0 and ::)

clients
Return a list of clients connected to a listening socket

For tcp and udp sockets a list of pair (address, port) is returned. For unix sockets a list of None is returned (thus you can make a len() for counting clients).

```
>>> host.socket("tcp://22").clients
[('2001:db8:0:1', 44298), ('192.168.31.254', 34866)]
>>> host.socket("unix:///var/run/docker.sock")
[None, None, None]
```

classmethod get_listening_sockets()
Return a list of all listening sockets

```
>>> host.socket.get_listening_sockets()
['tcp://0.0.0.0:22', 'tcp://:::22', 'unix:///run/systemd/private', ...]
```

4.4.21 Sudo

class Sudo(*user=None*)
Sudo module allow to run certain portion of code under another user.

It is used as a context manager and can be nested.

```
>>> Command.check_output("whoami")
'phil'
>>> with host.sudo():
...     host.check_output("whoami")
...     with host.sudo("www-data"):
...         host.check_output("whoami")
...
'root'
'www-data'
```

4.4.22 Supervisor

class Supervisor(*name*, *supervisorctl_path='supervisorctl'*, *supervisorctl_conf=None*, *_attrs_cache=None*)
Test supervisor managed services

```
>>> gunicorn = host.supervisor("gunicorn")
>>> gunicorn.status
'RUNNING'
>>> gunicorn.is_running
True
>>> gunicorn.pid
4242
```

The path where supervisorctl and its configuration file reside can be specified.

```
>>> gunicorn = host.supervisor("gunicorn", "/usr/bin/supervisorctl", "/etc/
↪supervisor/supervisord.conf")
>>> gunicorn.status
'RUNNING'
```

is_running
Return True if managed service is in status RUNNING

status
Return the status of the managed service

Status can be STOPPED, STARTING, RUNNING, BACKOFF, STOPPING, EXITED, FATAL, UNKNOWN.

See http://supervisord.org/subprocess.html#process-states

pid
Return the pid (as int) of the managed service

classmethod get_services(*supervisorctl_path='supervisorctl'*, *supervisorctl_conf=None*)
Get a list of services running under supervisor

```
>>> host.supervisor.get_services()
[<Supervisor(name="gunicorn", status="RUNNING", pid=4232)>
 <Supervisor(name="celery", status="FATAL", pid=None)>]
```

The path where supervisorctl and its configuration file reside can be specified.

```
>>> host.supervisor.get_services("/usr/bin/supervisorctl", "/etc/supervisor/
↪supervisord.conf")
[<Supervisor(name="gunicorn", status="RUNNING", pid=4232)>
 <Supervisor(name="celery", status="FATAL", pid=None)>]
```

4.4.23 Sysctl

class Sysctl(*name*)
Test kernel parameters

```
>>> host.sysctl("kernel.osrelease")
"3.16.0-4-amd64"
>>> host.sysctl("vm.dirty_ratio")
20
```

4.4.24 SystemInfo

class SystemInfo
Return system informations

type
OS type

```
>>> host.system_info.type
'linux'
```

distribution
Distribution name

```
>>> host.system_info.distribution
'debian'
```

release
Distribution release number

```
>>> host.system_info.release
'10.2'
```

codename
Release code name

```
>>> host.system_info.codename
'bullseye'
```

arch
Host architecture

```
>>> host.system_info.arch
'x86_64'
```

4.4.25 User

class User(*name=None*)
Test unix users

If name is not supplied, test the current user

name
Return user name

exists
Test if user exists

```
>>> host.user("root").exists
True
>>> host.user("nosuchuser").exists
False
```

uid
Return user ID

gid
Return effective group ID

group
Return effective group name

gids
Return the list of user group IDs

groups
Return the list of user group names

home
Return the user home directory

shell
Return the user login shell

password
Return the crypted user password

gecos
Return the user comment/gecos field

expiration_date
Return the account expiration date

```
>>> host.user("phil").expiration_date
datetime.datetime(2020, 1, 1, 0, 0)
>>> host.user("root").expiration_date
None
```

4.5 API

4.5.1 Connection API

You can use testinfra outside of pytest. You can dynamically get a *host* instance and call functions or access members of the respective modules:

```
>>> import testinfra
>>> host = testinfra.get_host("paramiko://root@server:2222", sudo=True)
>>> host.file("/etc/shadow").mode == 0o640
True
```

For instance you could make a test to compare two files on two different servers:

```
import testinfra

def test_same_passwd():
    a = testinfra.get_host("ssh://a")
    b = testinfra.get_host("ssh://b")
    assert a.file("/etc/passwd").content == b.file("/etc/passwd").content
```

4.6 Examples

4.6.1 Parametrize your tests

Pytest support test parametrization:

```
# BAD: If the test fails on nginx, python is not tested
def test_packages(host):
    for name, version in (
        ("nginx", "1.6"),
        ("python", "2.7"),
    ):
        pkg = host.package(name)
        assert pkg.is_installed
        assert pkg.version.startswith(version)


# GOOD: Each package is tested
# $ py.test -v test.py
# [...]
# test.py::test_package[local-nginx-1.6] PASSED
# test.py::test_package[local-python-2.7] PASSED
# [...]
import pytest
```

(continues on next page)

(continued from previous page)

```
@pytest.mark.parametrize("name,version", [
    ("nginx", "1.6"),
    ("python", "2.7"),
])
def test_packages(host, name, version):
    pkg = host.package(name)
    assert pkg.is_installed
    assert pkg.version.startswith(version)
```

4.6.2 Using unittest

Testinfra can be used with the standard Python unit test framework unittest instead of pytest:

```
import unittest
import testinfra


class Test(unittest.TestCase):

    def setUp(self):
        self.host = testinfra.get_host("paramiko://root@host")

    def test_nginx_config(self):
        self.assertEqual(self.host.run("nginx -t").rc, 0)

    def test_nginx_service(self):
        service = self.host.service("nginx")
        self.assertTrue(service.is_running)
        self.assertTrue(service.is_enabled)


if __name__ == "__main__":
    unittest.main()
```

```
$ python test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.705s

OK
```

4.6.3 Integration with Vagrant

Vagrant is a tool to setup and provision development environments (virtual machines).

When your Vagrant machine is up and running, you can easily run your testinfra test suite on it:

```
vagrant ssh-config > .vagrant/ssh-config
py.test --hosts=default --ssh-config=.vagrant/ssh-config tests.py
```

4.6.4 Integration with Jenkins

Jenkins is a well known open source continuous integration server.

If your Jenkins slave can run Vagrant, your build scripts can be like:

```
pip install pytest-testinfra paramiko
vagrant up
vagrant ssh-config > .vagrant/ssh-config
py.test --hosts=default --ssh-config=.vagrant/ssh-config --junit-xml junit.xml tests.
↪py
```

Then configure Jenkins to get tests results from the *junit.xml* file.

4.6.5 Integration with Nagios

Your tests will usually be validating that the services you are deploying run correctly. This kind of tests are close to monitoring checks, so let's push them to Nagios !

The Testinfra option *–nagios* enables a behavior compatible with a nagios plugin:

```
$ py.test -qq --nagios --tb line test_ok.py; echo $?
TESTINFRA OK - 2 passed, 0 failed, 0 skipped in 2.30 seconds
..
0

$ py.test -qq --nagios --tb line test_fail.py; echo $?
TESTINFRA CRITICAL - 1 passed, 1 failed, 0 skipped in 2.24 seconds
.F
/usr/lib/python3/dist-packages/example/example.py:95: error: [Errno 111] error msg
2
```

You can run these tests from the nagios master or in the target host with NRPE.

4.6.6 Integration with KitchenCI

KitchenCI (aka Test Kitchen) can use testinfra via its `shell` verifier. Add the following to your `.kitchen.yml`, this requires installing *paramiko* additionaly (on your host machine, not in the VM handled by kitchen)

```
verifier:
  name: shell
  command: py.test --hosts="paramiko://${KITCHEN_USERNAME}@${KITCHEN_HOSTNAME}:$
↪{KITCHEN_PORT}?ssh_identity_file=${KITCHEN_SSH_KEY}" --junit-xml "junit-${KITCHEN_
↪INSTANCE}.xml" "test/integration/${KITCHEN_SUITE}"
```

4.6.7 Test Docker images

Docker is a handy way to test your infrastructure code. This recipe shows how to build and run Docker containers with Testinfra by overloading the *host* fixture.

```
import pytest
import subprocess
import testinfra


# scope='session' uses the same container for all the tests;
# scope='function' uses a new container per test function.
@pytest.fixture(scope='session')
```

(continues on next page)

(continued from previous page)

```
def host(request):
    # build local ./Dockerfile
    subprocess.check_call(['docker', 'build', '-t', 'myimage', '.'])
    # run a container
    docker_id = subprocess.check_output(
        ['docker', 'run', '-d', 'myimage']).decode().strip()
    # return a testinfra connection to the container
    yield testinfra.get_host("docker://" + docker_id)
    # at the end of the test suite, destroy the container
    subprocess.check_call(['docker', 'rm', '-f', docker_id])


def test_myimage(host):
    # 'host' now binds to the container
    assert host.check_output('myapp -v') == 'Myapp 1.0'
```

4.7 Support

If you have questions or need help with testinfra please consider one of the following

4.7.1 Issue Tracker

Checkout existing issues on project issue tracker

4.7.2 IRC

You can also ask questions on IRC in #pytest channel on libera.chat network.

4.7.3 pytest documentation

testinfra is implemented as pytest plugin so to get the most out of please read pytest documentation

4.7.4 Community Contributions

- Molecule is an Automated testing framework for Ansible roles, with native Testinfra support.

Index

H

I

L

M

N

O

P

R

S

T

U

V