SERVERSPEC

RSpec tests for your servers configured by CFEngine, Puppet, Ansible, Itamae or anything else.

Resource Types

Note: In the following examples, due to my preference I'm using `should` syntax, but Serverspec works well on `expect` syntax.

You can use more strict grammar syntax like `be_a_file` instead of `be_file` with all resource types.

---

bond

Network bond resource type.

exist

In order to test a bond exists, you should use **exist** matcher.

```
describe bond('bond0') do
  it { should exist }
end
```

have_interface

In order to test a bond has a correct member, you should use **have_interface** matcher.

```
describe bond('bond0') do
  it { should have_interface 'eth0' }
end
```

---

bridge

Network bridge resource type.

exist

In order to test a bridge exists, you should use **exist** matcher.

```
describe bridge('br0') do
  it { should exist }
end
```

have_interface

In order to test a bridge has a correct interface, you should use **have_interface** matcher.

```
describe bridge('br0') do
  let(:stdout) { 'eth0' }
  it { should have_interface 'eth0' }
end
```

---

cgroup

Linux cgroup resource type.

You can test cgroup parameters like this.

```
describe cgroup('group1') do
  its('cpuset.cpus') { should eq 1 }
end
```

---

command

Command resource type.

its(:stdout), its(:stderr), its(:exit_status)

You can get the stdout, stderr and exit status of the command result, and can use any matchers RSpec supports.

```
describe command('ls -al /') do
  its(:stdout) { should match /bin/ }
end

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

describe command('ls /foo') do
  its(:exit_status) { should eq 0 }
end
```

contain

Notice: The matcher `contain` will be obsoleted.

In order to test a command stdout/stderr a given string, you can use **contain** matcher.

```
describe command('apachectl -M') do
  its(:stdout) { should contain('proxy_module') }
end
```

You can test a given string within a given range.

```
describe command('apachectl -V') do
  # test 'Prefork' exists between "Server MPM" and "Server compiled".
  its(:stdout) { should contain('Prefork').from(/^Server MPM/).to(/^Server compiled/) }

  # test 'conf/httpd.conf' exists after "SERVER_CONFIG_FILE".
  its(:stdout) { should contain('conf/httpd.conf').after('SERVER_CONFIG_FILE') }

  # test 'Apache/2.2.29' exists before "Server built".
  its(:stdout) { should contain(' Apache/2.2.29').before('Server built') }
end
```

---

cron

Cron resource type.

have_entry

In order to test cron have a given entry exists, you should use **have_entry** matcher.

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

You can test a given user has the cron entry like this.

```
describe cron do
  it { should have_entry('* * * * * /usr/local/bin/foo').with_user('mizzy') }
end
```

You can get all cron table entries and use regexp like this.

```
describe cron do
  its(:table) { should match  /you can use regexp/ }
end
```

---

default_gateway

Default gateway resource type.

In order to test a default gateway is set up correctly, you should use this syntax.

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

---

docker_container

Docker Container resource type

Container settings can be tested like this.

```
describe docker_container('focused_curie') do
  its(['HostConfig.NetworkMode']) { should eq 'bridge' }
  its(:Path) { should eq '/bin/sh' }
end
```

exist

Check if a named container exists.

```
describe docker_container('focused_curie') do
  it { should exist }
end
```

be_running

Check if a named container is running.

```
describe docker_container('focused_curie') do
  it { should be_running }
end
```

have_volume

In order to test if a volume has been mounted.

```
describe docker_container('focused_curie') do
  # matcher syntax: have_volume(container_path,host_path)
  it { should have_volume('/tmp','/data') }
end
```

---

docker_image

Docker Image resource type

Check the attributes hash returned by `docker inspect`.

```
describe docker_image('busybox:latest') do
  its(:inspection) { should_not include 'Architecture' => 'i386' }
end
```

You can also specify hash key like this.

```
describe docker_image('busybox:latest') do
  its(['Architecture']) { should eq 'amd64' }
end
```

You can also specify nested hash key like this.

```
describe docker_image('busybox:latest') do
  its(['Config.Cmd']) { should include '/bin/sh' }
end
```

exist

Check if a specific image been pulled to the host.

```
describe docker_image('busybox:latest') do
  it { should exist }
end
```

---

file

File and directory resource type.

be_file

In order to test a subject exists as a file, you should use **be_file** matcher.

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

exist

In order to test the presence of a file, you should use **exist** matcher.

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

be_directory

In order to test a subject exists as a directory, you should use **be_directory** matcher.

```
describe file('/var/log/httpd') do
  it { should be_directory }
end
```

be_block_device

In order to test a subject exists as a block device, you should use **be_block_device** matcher.

```
describe file('/dev/disk0') do
  it { should be_block_device }
end
```

be_character_device

In order to test a subject exists as a character device, you should use **be_character_device** matcher.

```
describe file('/dev/ttys0') do
  it { should be_character_device }
end
```

be_pipe

In order to test a subject exists as a name pipe, you should use the **be_pipe** matcher.

```
describe file('/tmp/unicorn.fifo') do
  it { should be_pipe }
end
```

be_socket

In order to test a subject exists as a socket, you should use **be_socket** matcher.

```
describe file('/var/run/unicorn.sock') do
  it { should be_socket }
end
```

be_symlink

In order to test a subject exists as a link, you should use **be_symlink** matcher.

```
describe file('/etc/pam.d/system-auth') do
  it { should be_symlink }
end
```

contain

Notice: Instead of `contain`, you can use `its(:content)` and any standard rspec matchers. The matcher `contain` will be obsoleted.

```
describe file('/etc/httpd/conf/httpd.conf') do
  its(:content) { should match /ServerName www.example.jp/ }
end
```

In order to test a file contains a given string, you can use **contain** matcher.

```
describe file('/etc/httpd/conf/httpd.conf') do
  it { should contain 'ServerName www.example.jp' }
end
```

You can test a file contains a given string within a given range.

```
describe file('Gemfile') do
  # test 'rspec' exists between "group :test do" and "end".
  it { should contain('rspec').from(/^group :test do/).to(/^end/) }

  # test 'rspec' exists after "group :test do".
  it { should contain('rspec').after(/^group :test do/) }

  # test 'rspec' exists before "end".
  it { should contain('rspec').before(/^end/) }
end
```

be_mode

In order to test a subject is set to given mode, you should use **be_mode** matcher.

Notice: `be_mode`, supports 3 and 4 number octet matching in the format ‘755’ and ‘2755’

```
describe file('/etc/sudoers') do
  it { should be_mode 440 }
end
```

be_owned_by

In order to test a subject is owned by a given user, you should use **be_owned_by** matcher.

```
describe file('/etc/sudoers') do
  it { should be_owned_by 'root' }
end
```

be_grouped_into

In order to test a subject is grouped into a given group, you should use **be_grouped_into** matcher.

```
describe file('/etc/sudoers') do
  it { should be_grouped_into 'wheel' }
end
```

be_linked_to

In order to test a subject is linked to a given file or directory, you should use **be_linked_to** matcher.

```
describe file('/etc/system-release') do
  it { should be_linked_to '/etc/redhat-release' }
end
```

be_readable

In order to test a subject is readable, you should use **be_readable** matcher.

```
describe file('/etc/sudoers') do
  it { should be_readable }
end
```

You can also test a subject is readable by owner, group members, others or a specific user.

```
describe file('/etc/sudoers') do
  it { should be_readable.by('owner') }
  it { should be_readable.by('group') }
  it { should be_readable.by('others') }
  it { should be_readable.by_user('apache') }
end
```

be_writable

In order to test a subject is writable, you should use **be_writable** matcher.

```
describe file('/etc/sudoers') do
  it { should be_writable }
end
```

You can also test a subject is writable by owner, group members, others or a specific user.

```
describe file('/etc/sudoers') do
  it { should be_writable.by('owner') }
  it { should be_writable.by('group') }
  it { should be_writable.by('others') }
  it { should be_writable.by_user('apache') }
end
```

be_executable

In order to test a subject is executable, you should use **be_executable** matcher.

```
describe file('/etc/init.d/httpd') do
  it { should be_executable }
end
```

You can also test a subject is executable by owner, group members, others or a specific user.

```
describe file('/etc/init.d/httpd') do
  it { should be_executable.by('owner') }
  it { should be_executable.by('group') }
  it { should be_executable.by('others') }
  it { should be_executable.by_user('httpd') }
end
```

be_immutable

In order to test a subject is immutable, you should use the **be_immutable** matcher.

```
describe file('/root/.ssh/authorized_keys') do
  it { should be_immutable }
end
```

be_mounted

In order to test a directory is mounted, you should use **be_mounted** matcher.

```
describe file('/') do
  it { should be_mounted }
end
```

You can also test a directory is mounted with correct attributes.

```
describe file('/') do
  it { should be_mounted.with( :type => 'ext4' ) }
end

describe file('/') do
  it { should be_mounted.with( :options => { :rw => true } ) }
end

describe file('/') do
  it do
    should be_mounted.only_with(
      :device  => '/dev/mapper/VolGroup-lv_root',
      :type    => 'ext4',
      :options => {
        :rw   => true,
        :mode => 620,
      }
    )
  end
end
```

`only_with` needs all attributes of the mounted directory.

be_version

In order to test a file’s version using its metadata in Windows, you should use **be_version** matcher.

```
describe file('C:\\Windows\\System32\\wuapi.dll') do
  it { should be_version('7.6.7600.256') }
end
```

its(:md5sum)

In order to test a file’s md5 checksum matches a given value, you should use **its(:md5sum)**.

```
describe file('/etc/services') do
  its(:md5sum) { should eq '35435ea447c19f0ea5ef971837ab9ced' }
end
```

its(:selinux_label)

In order to test a file’s SELinux label matches a given value, you should use **its(:selinux_label)**.

```
describe file('/etc/services') do
  its(:selinux_label) { should eq 'system_u:object_r:etc_t:s0' }
end
```

its(:sha256sum)

In order to test a file’s sha256 checksum matches a given value, you should use **its(:sha256sum)**.

```
describe file('/etc/services') do
  its(:sha256sum) { should eq 'a861c49e9a76d64d0a756e1c9125ae3aa6b88df3f814a51cecffd3e89cce6210' }
end
```

its(:size)

In order to test a file’s size matches a given value, you should use **its(:size)**. The file size is passed as bytes.

```
describe file('/etc/services') do
  its(:size) { should < 641021 }
end
```

its(:content_as_yaml)

In order to test YAML file’s contents match a given value, you should use **its(:content_as_yaml)**.

```
describe file('example.yml') do
  its(:content_as_yaml) { should include('foo' => include('bar' => 'hoge')) }
end
```

example.yml

```
---
foo:
  bar: hoge
```

its(:content_as_json)

In order to test JSON file’s contents match a given value, you should use **its(:content_as_json)**.

```
describe file('example.json') do
  its(:content_as_json) { should include('foo' => include('bar' => 'hoge')) }
end
```

example.json

```
{
  "foo": {
    "bar": "hoge"
  }
}
```

---

group

Group resource type.

exist

In order to test a group exists, you should use **exist** matcher.

```
describe group('wheel') do
  it { should exist }
end
```

have_gid

In order to test a group have a given gid, you should use **have_gid** matcher.

```
describe group('root') do
  it { should have_gid 0 }
end
```

---

host

Host resource type.

be_resolvable

In order to test a host is resolvable on the target host, you should use **be_resolvable** matcher.

```
describe host('serverspec.org') do
  it { should be_resolvable }
end

describe host('serverspec.org') do
  it { should be_resolvable.by('hosts') }
end

describe host('serverspec.org') do
  it { should be_resolvable.by('dns') }
end
```

be_reachable

In order to test a given host is network reachable, you should use **be_reachable** matcher.

```
describe host('target.example.jp') do
  # ping
  it { should be_reachable }
  # tcp port 22
  it { should be_reachable.with( :port => 22 ) }
  # set protocol explicitly
  it { should be_reachable.with( :port => 22, :proto => 'tcp' ) }
  # udp port 53
  it { should be_reachable.with( :port => 53, :proto => 'udp' ) }
  # timeout setting (default is 5 seconds)
  it { should be_reachable.with( :port => 22, :proto => 'tcp', :timeout => 1 ) }
end
```

its(:ipaddress)

You can get the ipaddress of the host, and can use any matchers rspec supports to them.

```
describe host('example.jp') do
  its(:ipaddress) { should eq '1.2.3.4' }
end

describe host('example.jp') do
  its(:ipaddress) { should match /1\.2\.3\./ }
end
```

On a dualstack host this may return an IPv6 address, notably Linux. Use *its(:ipv4_address)* or *its(:ipv6_address)* described below to get the desired address format.

its(:ipv4_address)

Query the IPv4 ipaddress of a host and apply any rspec supported matchers.

```
describe host('example.jp') do
  its(:ipv4_address) { should eq '1.2.3.4' }
end

describe host('example.jp') do
  its(:ipv4_address) { should match /1\.2\.3\./ }
end
```

its(:ipv6_address)

Query the IPv6 ipaddress of a host and apply any rspec supported matchers.

```
describe host('example.jp') do
  its(:ipv6_address) { should eq '2001:db8::1234' }
end

describe host('example.jp') do
  its(:ipv6_address) { should match /2001:db8:.*/ }
end
```

---

iis_app_pool

IIS Application Pool resource type.

exists

In order to test that an IIS app pool exists, you should use **exists** matcher.

```
describe iis_app_pool('Default App Pool') do
  it{ should exist }
end
```

have_dotnet_version

In order to test that an IIS app pool is using the correct .NET version, you should use **have_dotnet_version** matcher.

```
describe iis_app_pool('Default App Pool') do
  it{ should have_dotnet_version('2.0') }
end
```

---

iis_website

IIS Website resource type.

exists

In order to test that an IIS website exists, you should use **exists** matcher.

```
describe iis_website('Default Website') do
  it{ should exist }
end
```

enabled

In order to test that an IIS website is set to automatically start, you should use **enabled** matcher.

```
describe iis_website('Default Website') do
  it{ should be_enabled }
end
```

running

In order to test that an IIS website is currently running, you should use **running** matcher.

```
describe iis_website('Default Website') do
  it{ should be_running }
end
```

in_app_pool

In order to test that an IIS website is currently assigned to the correct Application Pool, you should use **in_app_pool** matcher.

```
describe iis_website('Default Website') do
  it{ should be_in_app_pool('Default App Pool') }
end
```

have_physical_path

In order to test that an IIS website gets its files from the correct location, you should use **have_physical_path** matcher.

```
describe iis_website('Default Website') do
  it{ should have_physical_path('C:\\inetpub\\www') }
end
```

---

interface

Network interface resource type.

exist

In order to test an interface exists, you should use **exist** matcher.

```
describe interface('eth0') do
  it { should exist }
end
```

up

In order to test an interface is up, you should use **be_up** matcher.

```
describe interface('eth0') do
  it { should be_up }
end
```

speed

In order to test the speed of network interface is set up correctly, you should use this syntax.

```
describe interface('eth0') do
  its(:speed) { should eq 1000 }
end
```

have_ipv4_address

In order to test an interface has an IP address, you should use **have_ipv4_address** matcher.

```
describe interface('eth0') do
  it { should have_ipv4_address("192.168.10.10") }
  it { should have_ipv4_address("192.168.10.10/24") }
end
```

have_ipv6_address

In order to test an interface has an IPv6 address, you should use **have_ipv6_address** matcher.

```
describe interface('eth0') do
  it { should have_ipv6_address("fe80::1") }
  it { should have_ipv6_address("fd00::1/48") }
end
```

its(:ipv4_address)

Query the IPv4 ipaddress of an interface and apply any rspec supported matchers.

```
describe interface('eth0') do
  its(:ipv4_address) { should eq '1.2.3.4/1' }
end

describe interface('eth0') do
  its(:ipv4_address) { should match /1\.2\.3\./ }
end
```

its(:ipv6_address)

Query the IPv6 ipaddress of an interface and apply any rspec supported matchers.

```
describe interface('eth0') do
  its(:ipv6_address) { should eq '2001:db8::1234/1' }
end

describe interface('eth0') do
  its(:ipv6_address) { should match /2001:db8:.*/ }
end
```

---

ip6tables

Ip6tables resource type.

have_rule

In order to test ip6tables has a given rule, you should use **have_rule** matcher.

```
describe ip6tables do
  it { should have_rule('-P INPUT ACCEPT') }
end
```

You can give a table name and a chain name like this.

```
describe ip6tables do
  it { should have_rule('-P INPUT ACCEPT').with_table('mangle').with_chain('INPUT') }
end
```

---

ipfilter

Ipfilter resource type.

have_rule

In order to test ipfilter has a given rule, you should use **have_rule** matcher.

```
describe ipfilter do
  it { should have_rule 'pass in quick on lo0 all' }
end
```

---

ipnat

Ipnat resource type.

have_rule

In order to test ipnat has a given rule, you should use **have_rule** matcher.

```
describe ipnat do
  it { should have_rule 'map net1 192.168.0.0/24 -> 0.0.0.0/32' }
end
```

---

iptables

Iptables resource type.

have_rule

In order to test iptables has a given rule, you should use **have_rule** matcher.

```
describe iptables do
  it { should have_rule('-P INPUT ACCEPT') }
end
```

You can give a table name and a chain name like this.

```
describe iptables do
  it { should have_rule('-P INPUT ACCEPT').with_table('mangle').with_chain('INPUT') }
end
```

---

kernel_module

Kernel module resource type.

be_loaded

In order to test a given kernel module is loaded, you should use **be_loaded** matcher.

```
describe kernel_module('virtio_balloon') do
  it { should be_loaded }
end
```

---

linux_audit_system

Allow checking the configuration of linux audit system via `/sbin/auditctl`. Suitable for server hardening.

be_running

To make sure that the audit system is running.

```
describe linux_audit_system do
  it { should be_running }
end
```

be_enabled

To make sure that the audit system is enabled (mode '1').

```
describe linux_audit_system do
  it { should be_enabled }
end
```

rules

To check that audit system has a given auditing rule. Argument can be a string for equality comparison or a regular expression.

```
describe linux_audit_system do
  its(:rules) { should include '-w /etc/audit/ -p wa' }
  its(:rules) { should include '-w /var/lib/ -p wa' }
  its(:rules) { should include %r!/var/lib/! }
  its(:rules) { should_not include %r!/var/log! }
end
```

---

linux_kernel_parameter

Linux kernel parameter resource type.

You can test Linux kernel parameters like this.

```
describe 'Linux kernel parameters' do
  context linux_kernel_parameter('net.ipv4.tcp_syncookies') do
    its(:value) { should eq 1 }
  end

  context linux_kernel_parameter('kernel.shmall') do
    its(:value) { should be >= 4294967296 }
  end

  context linux_kernel_parameter('kernel.shmmax') do
    its(:value) { should be <= 68719476736 }
  end

  context linux_kernel_parameter('kernel.osrelease') do
    its(:value) { should eq '2.6.32-131.0.15.el6.x86_64' }
  end

  context linux_kernel_parameter('net.ipv4.tcp_wmem') do
    its(:value) { should match /4096\t16384\t4194304/ }
  end
end
```

LXC

LXC(Linux Container) resource type.

You can test LXC like this.

```
describe lxc('ct01') do
  it { should exist }
  it { should be_running }
end
```

---

mail_alias

Mail alias resource type.

You can test mail aliases like this.

```
describe mail_alias('daemon') do
  it { should be_aliased_to 'root' }
end
```

---

mysql_config

MySQL config resource type.

You can test MySQL config parameters like this.

```
describe 'MySQL config parameters' do
  context mysql_config('innodb-buffer-pool-size') do
    its(:value) { should > 100000000 }
  end

  context mysql_config('socket') do
    its(:value) { should eq '/tmp/mysql.sock' }
  end
end
```

---

package

Package resource type.

be_installed

In order to test a package is installed, you should use **be_installed** matcher.

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

You can also test a given version of gem is installed.

```
describe package('jekyll') do
  it { should be_installed.by('gem').with_version('0.12.1') }
end
```

Or to test a pip packge is installed.

```
describe package('requests') do
  it { should be_installed.by('pip').with_version('2.18.1') }
end
```

---

php_config

PHP config resource type.

You can test PHP config parameters like this.

```
describe 'PHP config parameters' do
  context  php_config('default_mimetype') do
    its(:value) { should eq 'text/html' }
  end

  context php_config('session.cache_expire') do
    its(:value) { should eq 180 }
  end

  context php_config('mbstring.http_output_conv_mimetypes') do
    its(:value) { should match /application/ }
  end
end
```

You can also specify php.ini file to be used with php cli. You have just to pass :ini param with either a directory in which to look for php.ini, a path to php.ini file or custom INI file.

```
describe '[FPM] PHP config parameters' do
  context php_config('display_errors', {:ini => '/etc/php/7.1/fpm/php.ini'}) do
    its(:value) { should eq 1 }
  end
end
```

---

port

Port resource type.

be_listening

In order to test a given port is listening, you should use **be_listening** matcher.

```
describe port(80) do
  it { should be_listening }
end
```

You can also specify `tcp`, `udp`, `tcp6`, or `udp6`.

```
describe port(80) do
  it { should be_listening.with('tcp') }
end

describe port(80) do
  it { should be_listening.with('tcp6') }
end

describe port(53) do
  it { should be_listening.with('udp') }
end

describe port(53) do
  it { should be_listening.with('udp6') }
end
```

You can specify local binding address for port (this features requires `gem 'serverspec', '~> 2.0.0.beta8'`).

```
describe port(80) do
  it { should be_listening.on('127.0.0.1').with('tcp') }
end
```

---

ppa

PPA resource type.

exist

In order to test a given ppa repository exists, you should use **exist** matcher.

PPA resource type accepts both `ppa:username/reponame` and `username/reponame` style.

```
describe ppa('launchpad-username/ppa-name') do
  it { should exist }
end
```

be_enabled

In order to test a given ppa repository is enabled, you should use **be_enabled** matcher.

```
describe ppa('launchpad-username/ppa-name') do
  it { should be_enabled }
end
```

---

process

Process resource type.

You can test any process parameter available (such as `user`, `group`, `args`…) through the `ps` command like this:

```
describe process("memcached") do
  its(:user) { should eq "memcached" }
  its(:args) { should match /-c 32000\b/ }
end
```

For the complete list of available parameters, check the manual page for `ps(1)`, section *Standard Format Specifiers*. When several processes match, only the parameters of the first one are available.

count

To check the number of processes running under a given name use the **count** attribute matcher.

```
describe process("rsyslogd") do
  its(:count) { should eq 1 }
end
```

be_running

To check if a given process is running, you should use **be_running** matcher.

```
describe process("memcached") do
  it { should be_running }
end
```

---

routing_table

Routing table resource type.

have_entry

In order to test a routing table has a given entry, you should use **have_entry** matcher.

```
describe routing_table do
  it do
    should have_entry(
      :destination => '192.168.100.0/24',
      :interface   => 'eth1',
      :gateway     => '192.168.10.1',
    )
  end
end
```

---

selinux

SELinux resource type.

be_disabled, be_enforcing, be_permissive

In order to test SELinux is running in a given mode, you should use **be_disabled, be_enforcing and be_permissive** matchers.

```
# SELinux should be disabled
describe selinux do
  it { should be_disabled }
```

```
end

# SELinux should be enforcing
describe selinux do
  it { should be_enforcing }
end

# SELinux should be permissive
describe selinux do
  it { should be_permissive }
end
```

To check if a given policy is used in permissive or enforcing mode append the **with_policy** matcher.

```
# SELinux should be enforcing with policy mls
describe selinux do
  it { should be_enforcing.with_policy('mls') }
end

# SELinux should be permissive with policy targeted
describe selinux do
  it { should be_permissive.with_policy('targeted') }
end
```

selinux_module

SELinux Module resource type.

be_enabled

In order to test if a SELinux module is enabled, the following syntax is used.

```
describe selinux_module('zebra') do
  it { should be_enabled }
end
```

be_installed

To check if a given SELinux module is installed, the following syntax is used.

```
describe selinux_module('zebra') do
  it { should be_installed }
end
```

To see if a given version of SELinux module is installed the *with_version* matcher is appended.

```
describe selinux_module('zebra') do
  it { should be_installed.with_version('1.13.0') }
end
```

service

Service resource type.

be_enabled

In order to test a given service is enabled(automatically start when OS booting up), you should use **be_enabled** matcher.

```
describe service('ntpd') do
  it { should be_enabled }
end
```

You can test a service is enabled with a given run level.(This works only with Red Hat and Debian family currently.)

```
describe service('ntpd') do
  it { should be_enabled.with_level(3) }
end
```

be_installed

In order to test a given service is installed, you should use **be_installed** matcher.

Currently only supported in Windows

```
describe service('DNS Client') do
  it { should be_installed }
end
```

be_running

In order to test a given service/process is running, you should use **be_running** matcher.

```
describe service('ntpd') do
  it { should be_running }
end
```

You can test a given service/process is running under supervisor, upstart, systemd, and daemontools.

```
describe service('ntpd') do
  it { should be_running.under('supervisor') }
end

describe service('ntpd') do
  it { should be_running.under('upstart') }
end

describe service('ntpd') do
  it { should be_running.under('systemd') }
end

describe service('ntpd') do
  it { should be_running.under('daemontools') }
end
```

be_monitored_by

In order to test a service/process is monitored by a given software, you should use **be_monitored_by** matcher.

```
describe service('sshd') do
  it { should be_monitored_by('monit') }
end

describe service('unicorn') do
  it { should be_monitored_by('god') }
end
```

Should the monitor resource not match the service name the **with_name** matcher can be used override it.

```
describe service('tinc') do
  it { should be_monitored_by('monit').with_name('tinc-myvpn') }
end
```

have_start_mode

In order to test a service's startup mode is correct, you should use **have_start_mode** matcher.

Currently only supported in Windows

```
describe service('DNS Client') do
  it { should have_start_mode('Manual') }
end
```

---

user

User resource type.

exist

In order to test a subject exists as a user, you should use **exist** matcher.

```
describe user('root') do
  it { should exist }
end
```

belong_to_group

In order to test a user belongs to a given group, be it primary or secondary, use the **belong_to_group** matcher.

```
describe user('apache') do
  it { should belong_to_group 'apache' }
end
```

belong_to_primary_group

In order to test a user's primary group, use the **belong_to_primary_group** matcher.

```
describe user('apache') do
  it { should belong_to_primary_group 'apache' }
end
```

have_uid

In order to test a user have a given uid, you should use **have_uid** matcher.

```
describe user('root') do
  it { should have_uid 0 }
end
```

have_home_directory

In order to test a user have a given home directory, you should use **have_home_directory** matcher.

```
describe user('root') do
  it { should have_home_directory '/root' }
end
```

have_login_shell

In order to test a user have a given login shell, you should use **have_login_shell** matcher.

```
describe user('root') do
  it { should have_login_shell '/bin/bash' }
end
```

have_authorized_key

In order to test a have have a given authorized key, you should use **have_authorized_key** matcher.

```
describe user('root') do
  it { should have_authorized_key 'ssh-rsa ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFG
end
```

encrypted_password

To run tests against hashed passwords use the **encrypted_password** matcher.

Match a user without password.

```
describe user('nobody') do
  its(:encrypted_password) { should match(/^.{0,2}$/) }
end
```

Ensure password is stored using a SHA-512 hash.

```
describe user('unicorn') do
  its(:encrypted_password) { should match(/^\$6\$.{8}\$.{86}$/) }
end
```

minimum_days_between_password_change

In order to test the minimum days that a user is allowed to change their password use **minimum_days_between_password_change** matcher.

```
describe user('root') do
  its(:minimum_days_between_password_change) { should eq 0 }
end
```

maximum_days_between_password_change

In order to test the maximum days that a user is allowed to change their password use **maximum_days_between_password_change** matcher.

```
describe user('root') do
  its(:maximum_days_between_password_change) { should eq 99999 }
end
```

---

x509_certificate

x509_certificate resource type

be_certificate

In order to check whether a file is a certificate, use **be_certificate** matcher.

```
describe x509_certificate('some_cert.pem') do
  it { should be_certificate }
end
```

be_valid

You can ensure that a certificate is valid at present moment using **be_valid** matcher.

```
describe x509_certificate('some_cert.pem') do
   it { should be_valid }
end
```

validity_in_days

In order to ensure that a certificate is valid for a certain amount of days, use **validity_in_days**.

```
describe x509_certificate('some_cert.pem') do
  its(:validity_in_days) { should_not be < 100 }
  its(:validity_in_days) { should be >= 100 }
end
```

subject

In order to check the **subject** of a certificate, use:

```
describe x509_certificate('some_cert.pem') do
  its(:subject) { should eq '/C=DE/O=What/OU=Ever/CN=Root CA' }
end
```

issuer

In order to check the **issuer** of a certificate, use:

```
describe x509_certificate('some_cert.pem') do
  its(:issuer) { should match  /O=What/ }
end
```

email

In order to check the **email** of a certificate, use:

```
describe x509_certificate('some_cert.pem') do
  its(:email) { should be_empty }
end
```

have_purpose

Certificates can have purposes enabled or not, i.e. a certificate that may be used for client side SSL authentication. Check with **have_purpose**:

```
describe x509_certificate('some_cert.pem') do
  it { should have_purpose 'SSL server CA' }
  it { should_not have_purpose 'SSL server' }
end
```

keylength

In order to ensure that the key of a certificate has been generated with a certain key length, use the **key_length** matcher. It returns the key length as an integer, so it can be easily compared like:

```
describe x509_certificate('some_cert.pem') do
  its(:keylength) { should be >= 2048 }
end
```

subject_alt_names

Subject alternative names of a certificate can be specified as the following:

```
describe x509_certificate('some_cert.pem') do
  its(:subject_alt_names) { should include 'DNS:www.example.com' }
end
```

---

x509_private_key

x509_private_key resource type

be_encrypted

Checks if given file contains an encryption signature:

```
describe x509_private_key('/my/private/server-key.pem') do
  it { should_not be_encrypted }
end
```

be_valid

In order to check key integrity.

```
describe x509_private_key('/my/private/server-key.pem') do
  it { should be_valid }
end
```

have_matching_certificate

In order to ensure that the private key modulus is equal to modulus of given certificate, that is, certificate has been created from key.

```
describe x509_private_key('/my/private/server-key.pem') do
  it { should have_matching_certificate('/my/certs/server-cert.pem') }
end
```

---

windows_feature

Windows Feature resource type.

installed

In order to test that a windows feature has been installed, you should use **installed** matcher.

```
describe windows_feature('Minesweeper') do
  it{ should be_installed }
end
```

You can optionally specify the method used to check the windows feature, as the same feature can have different names.

```
describe windows_feature('IIS-Webserver') do
  it{ should be_installed.by("dism") }
end

describe windows_feature('Web-Webserver') do
  it{ should be_installed.by("powershell") }
end
```

---

windows_registry_key

Windows Registry Key resource type.

Matchers that reference the data type of the registry key will accept any of the following identifiers;

- :type_string
- :type_binary
- :type_dword
- :type_qword
- :type_multistring
- :type_expandstring

exist

In order to test that a key exists in the registry, you should use **exist** matcher.

```
describe windows_registry_key('HKEY_USERS\S-1-5-21\Test MyKey') do
  it { should exist }
end
```

have_property

In order to test a registry key contains a specific property, you should use the **have_property** matcher.

```
describe windows_registry_key('HKEY_USERS\S-1-5-21\Test MyKey') do
  it { should have_property('string value') }
  it { should have_property('binary value', :type_binary) }
  it { should have_property('dword value', :type_dword) }
end
```

have_value

In order to test that a registry key property has the correct value, you should use the **have_value** matcher.

```
describe windows_registry_key('HKEY_USERS\S-1-5-21\Test MyKey') do
  it { should have_value('test default data') }
end
```

have_property_value

In order to test that a registry key property has the correct value and data type, you should use the **have_property_value** matcher.

```
describe windows_registry_key('HKEY_USERS\S-1-5-21\Test MyKey') do
  it { should have_property_value('multistring value', :type_multistring, "test\nmulti\nstring\ndata") }
  it { should have_property_value('qword value', :type_qword, 'adff32') }
  it { should have_property_value('binary value', :type_binary, 'dfa0f066') }
end
```

---

yumrepo

Yumrepo resource type.

exist

In order to test a given yum repository exists, you should use **exist** matcher.

```
describe yumrepo('epel') do
  it { should exist }
end
```

be_enabled

In order to test a given yum repository is enabled, you should use **be_enabled** matcher.

```
describe yumrepo('epel') do
  it { should be_enabled }
end
```

---

zfs

ZFS resource type.

exist

In order to test a given zfs pool exists, you should use **exist** matcher.

```
describe zfs('rpool') do
  it { should exist }
end
```

have_property

In order to test a zfs pool has given properties, you should use **have_property** matcher.

```
describe zfs('rpool') do
  it { should have_property 'mountpoint' => '/rpool', 'compression' => 'off' }
end
```