Suitable Documentation

Release 0.14.0

Denis Krienbühl

Aug 17, 2018

Contents

An Ansible API for humans.

CHAPTER 1

Introduction

Ansible is a configuration management tool written in Python. Even though it is written in Python its configuration syntax is decidedly un-pythonic.

To write ansible configuration is to write YAML files. Suitable *does not* try to change that. Suitable tries to be a simple Ansible API abstraction in Python.

With Suitable you can write Python code that has easy access to the plethora of modules that Ansible offers. As such it is great for scripts that automate migrations, upgrades or other little tasks.

Suitable is not an alternative to Ansible, it is a tool to complement it. Do not use Suitable to manage your server fleet. Use Suitable to boss your servers around from time to time.

CHAPTER 2

Warning

Suitable is not endorsed by Ansible and it is not affilated with it. Use at your own peril.

The official way to use Ansible from Python is documented here: http://docs.ansible.com/ansible/developing_api.html

CHAPTER 3

Installation

To install suitable, simply use pip. This will install Ansible 2.x automatically as a dependency:

```
pip install suitable
```

CHAPTER 4

Examples

Create the user ‘denis’ on ‘web.seantis.dev’ and ‘db.seantis.dev’:

```
from suitable import Api

hosts = Api(['web.seantis.dev', 'db.seantis.dev'])
hosts.user(name='denis')
```

Create the user ‘postgres’ on ‘db.seantis.dev’:

```
dbhost = Api('db.seantis.dev')
dbhost.user(name='postgres')
```

List the mounts on ‘backup.seantis.dev’:

```
backuphost = Api('backup.seantis.dev')
backuphost.setup(filter='ansible_mounts')
```

Connect to a server using a username and a password:

```
from getpass import getpass

username = 'admin'
password = getpass()

api = Api(
    'web.seantis.dev',
    remote_user=username,
    remote_pass=password
)

print api.command('whoami').stdout()  # prints 'admin'
```

Run a command on multiple servers and get the output for each:

```
servers = ['a.example.org', 'b.example.org']

api = Api(servers)
result = api.command('whoami')

for server in servers:
    print result.stdout(server)
```

CHAPTER 5

Which Modules are Available?

All of them! Suitable is a wrapper around all Ansible modules. Here's a list of all Ansible modules:

http://docs.ansible.com/ansible/modules_by_category.html

Say you want to use the file module, which is documented here:

http://docs.ansible.com/ansible/file_module.html

Take the first example of the file module:

```
- file: path=/etc/foo.conf owner=foo group=foo mode=0644
```

It can be directly translated into the following Suitable call:

```
api.file(path='etc/foo.conf', owner='foo', mode='0644')
```

This works for any Ansible module.

CHAPTER 6

Mitogen Support

Suitable supports https://mitogen.readthedocs.io for major performance gains. Note that this support is somewhat experimental and should be used with caution.

To use mitogen with Suitable, install it first, alongside Suitable:

```
pip install suitable
pip install mitogen
```

Afterwards, change your import slightly to use an adapted Suitable Api class:

```
from suitable.mitogen import Api
```

This class works exactly like the vanilla Suitable Api, the difference is that it registers mitogen with Ansible automatically and switches the default strategy to 'mitogen_linear'.

You can also use the alternative 'mitogen_free' strategy with this class:

```
Api('example.org', strategy='mitogen_free')
```

CHAPTER 7

API Documentation

To learn more about Suitable's API have a look at the API documentation:

Suitable API Documentation.

If you have any questions do not hesitate to open an issue.

CHAPTER 8

Source

https://github.com/seantis/suitable

CHAPTER 9

License

Suitable is released under GPLv3 (compatible with Ansible).

CHAPTER 10

Navigation

10.1 Suitable API Documentation

class suitable.api.**Api**(*servers*, *ignore_unreachable=False*, *ignore_errors=False*, *sudo=False*, *dry_run=False*, *verbosity='info'*, *environment=None*, *strategy=None*, ***options*)

Provides all available ansible modules as local functions:

```
api = Api('personal.server.dev')
api.sync(src='/Users/denis/.zshrc', dest='/home/denis/.zshrc')
```

Parameters

- **servers** – A list of servers or a string with space-delimited servers. The api instances will operate on these servers only. Servers which cannot be reached or whose use triggers an error are taken out of the list for the lifetime of the object.

  Examples of valid uses:

  ```
  api = Api(['web.example.org', 'db.example.org'])
  api = Api('web.example.org')
  api = Api('web.example.org db.example.org')
  ```

  Each server may optionally contain the port in the form of `host:port`. If the host part is an ipv6 address you need to use the following form to specify the port: `[host]:port`.

  For example:

  ```
  api = Api('remote.example.org:2222')
  api = Api('[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:1234')
  ```

  Note that there's currently no support for passing the same host more than once (like in the case of a bastion host). Ansible groups these kind of calls together and only calls the first server.

So this won't work as expected:

```
api = Api(['example.org:2222', 'example.org:2223'])
```

As a work around you should define aliases for these hosts in your ssh config or your hosts file.

- **ignore_unreachable** – If true, unreachable servers will not trigger an exception. They are however still taken out of the list for the lifetime of the object.
- **ignore_errors** – If true, errors on servers will not trigger an exception. Servers who trigger an error are still ignored for the lifteime of the object.
- **sudo** – If true, the commands run as root using sudo. This is a shortcut for the following:

```
Api('example.org', become=True, become_user='root')
```

If `become` or `become_user` are passed, this option is ignored!

- **sudo_pass** – If given, sudo is invoked with the given password. Alternatively you can use Ansible's builtin password option (e.g. `passwords={'become_pass': '***'}`).
- **remote_pass** – Passwords are passed to ansible using the passwords dictionary by default (e.g. `passwords={'conn_pass': '****'}`). Since this is a bit cumbersome and because earlier Suitable releases supported *remote_pass* this convenience argument exists.

  If *passwords* is passed, the *remote_pass* argument is ignored.
- **dry_run** – Runs ansible in 'check' mode, where no changes are actually applied to the server(s).
- **verbosity** – The verbosity level of ansible. Possible values:
  - `debug`
  - `info` (default)
  - `warn`
  - `error`
  - `critical`
- **environment** – The environment variables which should be set during when a module is executed. For example:

```
api = Api('example.org', environment={
    'PGPORT': '5432'
})
```

- **strategy** – The Ansible strategy to use. Defaults to None which lets Ansible decide which strategy it wants to use.

  Note that you need to globally install strategy plugins using *`install_strategy_plugins()`* before using strategies provided by plugins.
- **extra_vars** – Extra variables available to Ansible. Note that those will be global and not bound to any particular host:

```
api = Api('webserver', extra_vars={'home': '/home/denis'})
api.file(dest="{{ home }}/.zshrc", state='touch')
```

This can be used to specify an alternative Python interpreter:

```
api = Api('example.org', extra_vars={
    'ansible_python_interpreter': '/path/to/interpreter'
})
```

- ****options** – All remining keyword arguments are passed to the Ansible TaskQueueManager. The available options are listed here:

  http://docs.ansible.com/ansible/developing_api.html

on_module_error(*module*, *host*, *result*)
If you want to customize your error handling, this would be the point to write your own method in a subclass.

Note that this method is not called if ignore_errors is True.

If the return value of this method is 'keep-trying', the server will not be ignored for the lifetime of the object. This enables you to practically write your own flavor of 'ignore_errors'.

If an any exception is raised the server WILL be ignored.

on_unreachable_host(*module*, *host*)
If you want to customize your error handling, this would be the point to write your own method in a subclass.

Note that this method is not called if ignore_unreachable is True.

If the return value of this method is 'keep-trying', the server will not be ignored for the lifetime of the object. This enables you to practically write your own flavor of 'ignore_unreachable'.

If an any exception is raised the server WILL be ignored.

valid_return_codes(***kwds*)
Sets codes which are considered valid when returned from command modules. The default is (0,).

Should be used as a context:

```
with api.valid_return_codes(0, 1):
    api.shell('test -e /tmp/log && rm /tmp/log')
```

suitable.api.**install_strategy_plugins**(*directories*)
Loads the given strategy plugins, which is a list of directories, a string with a single directory or a string with multiple directories separated by colon.

As these plugins are globally loaded and cached by Ansible we do the same here. We could try to bind those plugins to the Api instance, but that's probably not something we'd ever have much of a use for.

Call this function before using custom strategies on the *Api* class.

Python Module Index

S

Index