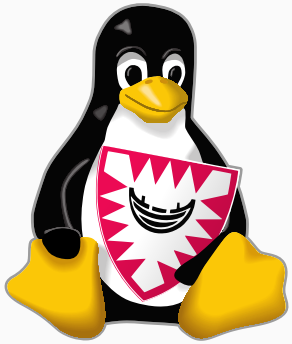
Ansible Module Schreiben

Martin Schütte

20. September 2019

Kieler Linux-Tage



Assumptions

You …

- configure servers or other IT systems
- have already used (or tried) Ansible
- can write shell or Python scripts
- have some “special” device or API
  or a CLI that does not fit into a simple `command`

This talk …

- is no Ansible introduction
- has too many slides, I will skip some
- is available online at noti.st

Outline

Concepts

Concepts

Intro

Ansible – Concepts and Naming

Ansible is a radically simple IT automation platform.

- controller
- target host
- playbook
- role
- task
- module

Example: Simple Playbook

```yaml
---
- hosts: webserver
  vars:
    apache_version: latest
  tasks:
  - name: ensure apache is at given version
    yum:
      name: httpd
      state: "{{ apache_version }}"

- hosts: dbserver
  roles:
    - ansible-role-postgresql
```

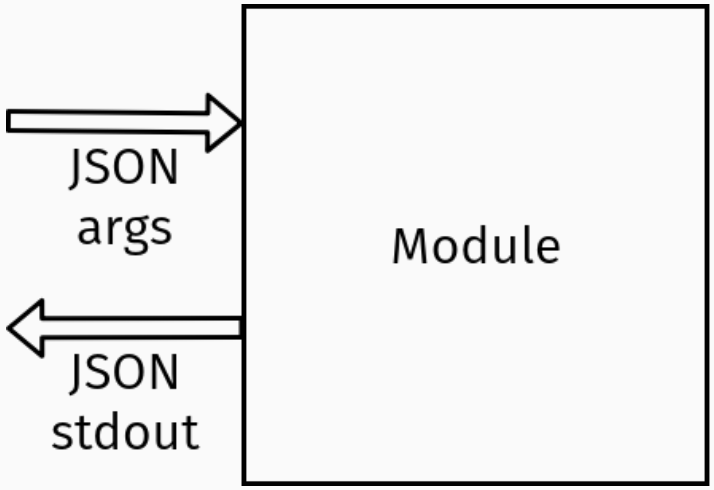
Concepts

Module Basics

What is a Module?

some code snippet to run on the (remote) host

executable with input and output



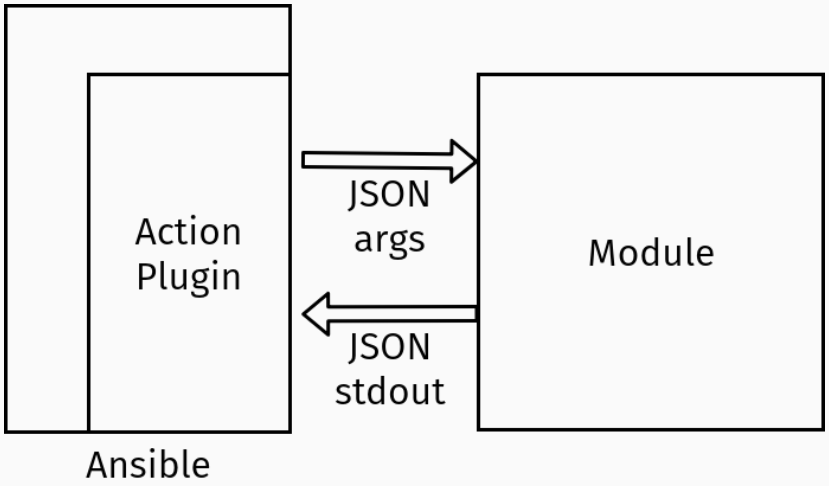
Minimal Module

```
#!/bin/sh

echo '{"foo": "bar"}'
exit 0
```

```
#!/usr/bin/python

if __name__ == '__main__':
    print('{"foo": "bar"}')
    exit(0)
```

Action Plugins call Modules

- plugins run on the controller
- may prepare input for modules
- may handle “special” connections (non SSH or WinRM)
- may implement actions on the controller, e. g. debug
- defaults to `normal` to run module on target host



Writing Modules

Writing Modules

Don’t

Avoid Writing Own Code

- get_url – Downloads files
- uri – Interacts with webservices
- wait_for – Waits for a condition before continuing
- set_fact – Set host facts from a task

```yaml
- name: Wait for port 8000 to become open on the host
  wait_for:
    port: 8000
    delay: 10

- name: wait for service to become available
  uri:
    url: 'https://{{ inventory_hostname }}:{{ svc_port }}/service'
    return_content: yes
  register: content
  until: content.status == 200
  retries: 60
  delay: 10
  when: not ansible_check_mode
```

Writing Modules

Simple Example: Ping

Documentation

```
ANSIBLE_METADATA = {'metadata_version': '1.1',
                    'status': ['stableinterface'],
                    'supported_by': 'core'}
DOCUMENTATION = '''
---
module: ping
version_added: historical
short_description: Try to connect to host, verify a usable
python and return C(pong) on success
...
'''

EXAMPLES = '''
# Induce an exception to see what happens
- ping:
    data: crash
'''

RETURN = '''
ping:
    description: value provided with the data parameter
    returned: success
    type: string
    sample: pong
'''
```

ansible-doc

```
$ ansible-doc --snippet ping
- name: Try to connect to host, verify a usable python and return `pong' on success
  ping:
      data:                 # Data to return for the `ping' return value. If this
                              parameter is set to `crash', the module will cause an
                              exception.

$ ansible-doc ping
> PING    (.../site-packages/ansible/modules/system/ping.py)

        A trivial test module, this module always returns `pong' on
        successful contact. It does not make sense in playbooks, but
        it is useful from `/usr/bin/ansible' to verify the ability to
        login and that a usable Python is configured. This is NOT ICMP
        ping, this is just a trivial test module that requires Python
        on the remote-node. For Windows targets, use the [win_ping]
        module instead. For Network targets, use the [net_ping] module
        instead.

OPTIONS (= is mandatory):

- data
        Data to return for the `ping' return value.
...
```

ping.py

```python
from ansible.module_utils.basic import AnsibleModule

def main():
    module = AnsibleModule(
        argument_spec=dict(
            data=dict(type='str', default='pong'),
        ),
        supports_check_mode=True
    )

    if module.params['data'] == 'crash':
        raise Exception("boom")
    result = dict(
        ping=module.params['data'],
    )
    module.exit_json(**result)

if __name__ == '__main__':
    main()
```

Writing Modules

Start Your Own

my_module.py

```python
from ansible.module_utils.basic import AnsibleModule

def main():
    module = AnsibleModule(
        argument_spec=dict(  # ...
        )
    )

    rc = do_something()
    result = {
        "msg":     "Hello World",
        "rc":      rc,
        "failed":  False,
        "changed": False,
    }
    module.exit_json(**result)

if __name__ == '__main__':
    main()
```

File Locations: `library` and `module_utils`

```
my_role/
├── meta
├── defaults
├── tasks
├── library
│   └── my_module.py
└── module_utils
    └── my_util_lib.py
```

- role can use Ansible module `my_module` in tasks
- `import * from my_util_lib`
  finds Python module in `module_utils`
- for “larger” libraries use packages (pip/rpm/dpkg)

AnsibleModule argument_spec

```
module = AnsibleModule(
    argument_spec=dict(
        config=dict(required=False),
        name=dict(required=True),
        password=dict(required=False,
            no_log=True),
        state=dict(required=False,
            choices=['present', 'absent'],
            default="present"),
        enabled=dict(required=False,
            type='bool'),
        token=dict(required=False,
            no_log=True),
        url=dict(required=False,
            default="http://localhost:8080"),
        user=dict(required=False)
    ),
    mutually_exclusive=[
        ['password', 'token'],
        ['config', 'enabled'],
    ],
    supports_check_mode=True,
)
```

```
# Create a jenkins job using the token
- jenkins_job:
    config: "{{ lookup(...) }}"
    name: test
    token: asdfasfasfasdfasdfadf
    url: http://localhost:8080
    user: admin
```

```
# Disable a jenkins job using basic auth
- jenkins_job:
    name: test
    password: admin
    enabled: False
    url: http://localhost:8080
    user: admin
```

Common Return Values/Result Attributes

Common

- changed
- failed
- rc
- msg
- results
- invocation
- skipped
- stderr, stderr_lines
- stdout, stdout_lines
- backup_file

Internal use

- ansible_facts
- exception
- warnings
- deprecations

get_release.py function

```
def read_os_file(item_name):
    filename = "/etc/os-release"

    result = {
        "msg":     "unknown",
        "failed":  True,
        "changed": False,
    }

    with open(filename, "r") as f:
        for line in f:
            key,value = line.split('=', 1)
            if key == item_name:
                result["msg"]    = value.strip().strip('"')
                result["failed"] = False
                break
    return result
```

get_release.py main

```
def main():
    m = AnsibleModule(
        argument_spec=dict(
            line=dict(type='str', default="PRETTY_NAME"),
        ),
        supports_check_mode=False
    )

    result = read_os_file(m.params["line"])
    m.exit_json(**result)

if __name__ == '__main__':
    main()
```

get_release.py usage

```
- name: my_role | get_release name
  get_release: {}
  register: rc_get_dist

- name: my_role | get_release like
  get_release:
    line: ID_LIKE
  register: rc_get_like

- name: my_role | debug
  debug:
    msg: "{{ rc_get_dist.msg }} -- {{ rc_get_like.msg }}"
```

```
TASK [role-pymod : my_role | debug] ***********************
ok: [server] => {
    "msg": "CentOS Linux 7 (Core) -- rhel fedora"
}
```

Common Module Pattern

```
class Controller(object):
  def __init__(module)
  def do_something()

def main():
  module = AnsibleModule(...)
  ctl = Controller(module)
  result = ctl.do_something()
  module.exit_json(**result)

if __name__ == '__main__':
    main()
```

- simple access to input parameters
- access to util functions (e. g. `module.run_command()`)
- difficult to unit test without module context

Writing Modules

Patterns & Misc. Hints

Use AnsibleModule

Useful common methods:

- `argument_spec` for parameters
- `supports_check_mode`
- `exit_json()`, `fail_json()`
- `atomic_move()`, `run_command()`
- `bytes_to_human()`, `human_to_bytes()`

Other module_utils:

- `api`: function/decorator `@rate_limit()`
- `timeout`: function/decorator `@timeout(secs)`
- `_text`: new and unstable `to_text()`

Pattern: Idempotency

- Playbooks can run many times
- As few changes as possible
- Only perform required actions

1. Get spec parameters
2. Check actual state of system

if $=$: done, do nothing

if $\neq$: action to change state

Pattern: Check Dependencies

```
try:
    import psycopg2
    import psycopg2.extras
except ImportError:
    HAS_PSYCOPG2 = False
else:
    HAS_PSYCOPG2 = True

def main():
    module = AnsibleModule()
    # ...
    if not HAS_PSYCOPG2:
        module.fail_json(
          msg="the python psycopg2 module is required")
```

Check Mode/“Dry Run”

- Return information but never apply changes
- Optional but recommended for modules
- Interface provided by `AnsibleModule`

Example without support:

```
m = AnsibleModule(
    argument_spec=...,
    supports_check_mode=False
)
```

```
$ ansible-playbook -v --check playbook.yml
...
TASK [role-pymod : my_role | get_release] **************
skipping: [server] => {"changed": false, "msg": "remote module
                (get_release) does not support check mode"}
```

Check Mode/“Dry Run”

```
def update_permanent_hostname(self):
    name = self.module.params['name']
    permanent_name = self.get_permanent_hostname()
    if permanent_name != name:
        if not self.module.check_mode:
            self.set_permanent_hostname(name)
        self.changed = True
```

Important: Modules without `AnsibleModule` (or non-Python) have to handle this on their own!

⇒ test the `_ansible_check_mode` parameter

Other Common Return Value: Diff

Example from `hostname`:

```
if changed:
    kw['diff'] = {'after': 'hostname = ' + name + '\n',
                  'before': 'hostname = ' + name_before + '\n'}
```

Example output, sample module:

```
TASK [role-minimal : role_minimal | py_sample_08] ************
task path: /vagrant/roles/role-minimal/tasks/main.yml:23
--- before
+++ after
@@ -1,3 +1,3 @@
 common line
-old value
+new vale
 common line

changed: [server] => {"changed": "true", "foo": "bar"}
```

Example: Set Facts

In a playbook:

```
- do_something:
    # ...
  register: result_var

- set_fact:
     foo: "{{ result_var.results | list }}"
```

In a module (from `hostname`):

```
kw = dict(changed=changed, name=name,
       ansible_facts=dict(ansible_hostname=name.split('.')[0],
                          ansible_nodename=name,
                          ansible_fqdn=socket.getfqdn(),
                          ansible_domain='.'.join(
                             socket.getfqdn().split('.')[1:])))
module.exit_json(**kw)
```

Example: String Formatting

When Jinja2 is not enough for variables and formatting ...

Random example:

```
- name: calculate cluster config lines
  calc_some_cluster_config:
    hostname: "{{ ansible_fqdn }}"
    port: "{{ application_port }}"
    group: "{{ groups['appserver'] }}"
  register: cluster_config

- name: cluster config
  lineinfile:
    path: "{{ basedir }}/cluster/config"
    line: "{{ item }}"
  with_items:
    - "hosts={{ cluster_config.tcp_hostlist | join(',') }}"
    - "exclude={{ cluster_config.exclude_roles | join(',') }}"
  notify: appserver restart
```

Pattern: Provider Dict

```
- name: apply IOS config
  ios_config:
    provider: "{{ ios_creds }}"
    src: my_config2b.txt
```

```
- name: Import Zabbix json template configuration
  local_action:
    module: zabbix_template
    server_url: http://127.0.0.1
    login_user: username
    login_password: password
    template_name: Apache2
    template_json: "{{ lookup('file', 'apache2.json') }}"
    template_groups:
      - Webservers
```

Module Execution – In-Depth

Module Execution – In-Depth

Low Level Module Execution

Minimal Module

```
#!/bin/sh

echo '{"foo": "bar"}'
exit 0
```

```
#!/usr/bin/python

if __name__ == '__main__':
    print('{"foo": "bar"}')
    exit(0)
```

Minimal Module – Verbose Output

```
TASK [role-minimal : role_minimal | bash_sample_01] ****************************
task path: /vagrant/roles/role-minimal/tasks/main.yml:5
<192.168.56.202> ESTABLISH SSH CONNECTION FOR USER: vagrant
<192.168.56.202> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKey
<192.168.56.202> (0, '/home/vagrant\n', '')
<192.168.56.202> ESTABLISH SSH CONNECTION FOR USER: vagrant
<192.168.56.202> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKey
<192.168.56.202> (0, 'ansible-tmp-1548772995.78-225807547469627=/home/vagrant/.ansible/tmp/ans
Using module file /vagrant/roles/role-minimal/library/bash_sample_01
<192.168.56.202> PUT /home/vagrant/.ansible/tmp/ansible-local-32044a_dXdg/tmpnrj9rd TO /home/v
<192.168.56.202> SSH: EXEC sftp -b - -C -o ControlMaster=auto -o ControlPersist=60s -o StrictH
<192.168.56.202> (0, 'sftp> put /home/vagrant/.ansible/tmp/ansible-local-32044a_dXdg/tmpnrj9rd
<192.168.56.202> PUT /home/vagrant/.ansible/tmp/ansible-local-32044a_dXdg/tmpgN3ZKr TO /home/v
<192.168.56.202> SSH: EXEC sftp -b - -C -o ControlMaster=auto -o ControlPersist=60s -o StrictH
<192.168.56.202> (0, 'sftp> put /home/vagrant/.ansible/tmp/ansible-local-32044a_dXdg/tmpgN3ZKr
<192.168.56.202> ESTABLISH SSH CONNECTION FOR USER: vagrant
<192.168.56.202> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKey
<192.168.56.202> (0, '', '')
<192.168.56.202> ESTABLISH SSH CONNECTION FOR USER: vagrant
<192.168.56.202> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKey
Escalation succeeded
<192.168.56.202> (0, '{"foo":"bar"}\r\n', 'Shared connection to 192.168.56.202 closed.\r\n')
ok: [server] => {
    "changed": false,
    "foo": "bar"
}

TASK [role-minimal : role_minimal | bash_sample_01] ****************************
task path: /vagrant/roles/role-minimal/tasks/main.yml:5
```

Module Execution – In-Depth

AnsiballZ

Import AnsibleModule

```
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule

if __name__ == '__main__':
    print('{"foo":"bar"}')
    exit(0)
```

Import AnsibleModule – Verbose Output

```
[vagrant@server ~]$ ls -hl $TMPDIR
-rwx------. 1 vagrant vagrant 75K Jan 29 14:53 AnsiballZ_py_sample_04.py

[vagrant@server ~]$ head $TMPDIR/AnsiballZ_py_sample_04.py
#!/usr/bin/python
# -*- coding: utf-8 -*-
_ANSIBALLZ_WRAPPER = True # For test-module script to tell this is a ANSIBALLZ_WRAPPER
# This code is part of Ansible, but is an independent component.
# The code in this particular templatable string, and this templatable string
# only, is BSD licensed.  Modules which end up using this snippet, which is
# dynamically combined together by Ansible still belong to the author of the
# module, and they may assign their own license to the complete work.
#
# Copyright (c), James Cammarata, 2016
[vagrant@server ~]$
```

AnsiballZ Wrapper

Python template script for

- helper functions (execute, explode)
- zipped data of
  - module text
  - JSON arguments
  - all `module_utils` imports

Module Execution – In-Depth

Debugging

Debugging Tools and Tips

Dev environment:

- Vagrant
- `keep_remote_files = True`
- `ansible -vvv`

- AnsiballZ code expand
- “print to output”
- `AnsibleModule.log()`
- `q`

Debugging – AnsiballZ Explode

```
[vagrant@server ~]$ ls -hl $TMPDIR
-rwx------. 1 vagrant vagrant 75K Jan 29 14:53 AnsiballZ_py_sample_04.py
[vagrant@server ~]$ $TMPDIR/AnsiballZ_py_sample_04.py explode
Module expanded into:
$TMPDIR/debug_dir
[vagrant@server ~]$ cd $TMPDIR/debug_dir; find .
.
./ansible
./ansible/__init__.py
./ansible/module_utils
./ansible/module_utils/__init__.py
./ansible/module_utils/basic.py
./ansible/module_utils/parsing
./ansible/module_utils/parsing/convert_bool.py
./ansible/module_utils/parsing/__init__.py
./ansible/module_utils/common
./ansible/module_utils/common/_collections_compat.py
./ansible/module_utils/common/process.py
./ansible/module_utils/common/__init__.py
./ansible/module_utils/common/file.py
./ansible/module_utils/six
./ansible/module_utils/six/__init__.py
./ansible/module_utils/_text.py
./ansible/module_utils/pycompat24.py
./__main__.py
./args
```

Debugging – AnsiballZ Explode

```
[vagrant@server debug_dir]$ cat __main__.py
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule

if __name__ == '__main__':
    print('{"foo":"bar"}')
    exit(0)

[vagrant@server debug_dir]$ $TMPDIR/AnsiballZ_py_sample_04.py execute
{"foo":"bar"}
```

Debugging – printf

- Ansible reads `stdin` and `stdout`, expects JSON
  $\Rightarrow$ cannot use print() to debug
- Use output values instead

```
# ...
debug_msg = "some_func({}) returned {}".format(bar, foo)
# ...
module.exit_json(result=foo, debug_msg=debug_msg)
```

```
ok: [server] => {
    "changed": false,
    "debug_msg": "some_func(bar) returned foo",
    ...
}
```

Debugging – AnsibleModule log()

- AnsibleModule includes method `log()` with variants `debug()` and `warn()`
- Writes to journald or Syslog

```
module.log("Hello World")
```

```
# tail /var/log/messages

Feb  9 15:02:59 server ansible-my_module: Invoked with param=...
Feb  9 15:02:59 server ansible-my_module: Hello World
```

Debugging – q

- PyPI q or zestyping/q
- Always writes to `/tmp/q`
- function decorators

```
try:
    import q
except ImportError:
    def q(x):
        return x

@q
def my_func(params):
    q(special_var)
    # ...
```

```
$ tail /tmp/q

 0.0s my_func('VERSION')
 0.0s   my_func: 'special_value'
 0.0s -> {'failed': False, 'msg': '...'}
```

Beyond Python

Ansible Modules in Other Languages

- Python: the default choice, best tools and support
- PowerShell: officially supported, but not covered here
- Scripting Languages: can use `JSON_ARGS`
- Binary Executables: can use `WANT_JSON`

Binary Executables, e. g. Go Tools

possible, but not recommended:

- binary in Ansible git repository
- should have own build chain and versions
- architecture dependent (all the world's a x86_64?)

better:

- separate packaging and deployment (.deb and .rpm)
- thin wrapper module to execute installed file or library
- same for binary Python libraries (e. g. DB connectors)

Java

- “executable” JAR file
- not architecture dependent
- otherwise the same as binaries

Groovy

- JVM scripting language
- compromise between Ansible scripts and Java apps
- maybe best way to access/use Java libraries with Ansible
- may need workarounds to use custom classpath

Groovy – Small Example

```groovy
#!/usr/bin/env groovy
import groovy.json.*

def jsonArgs='''<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>'''
def args_object = new JsonSlurper().parseText(jsonArgs)

def checkMode = (boolean) args_object['_ansible_check_mode']

// do something useful

def result = [
  foo: 'bar',
  code: 42,
  did_check_mode: checkMode
]

print JsonOutput.toJson(result)
```

Scripting Languages

- Perl, Ruby, etc.
- need JSON library
- works
- question: why?

Conclusion

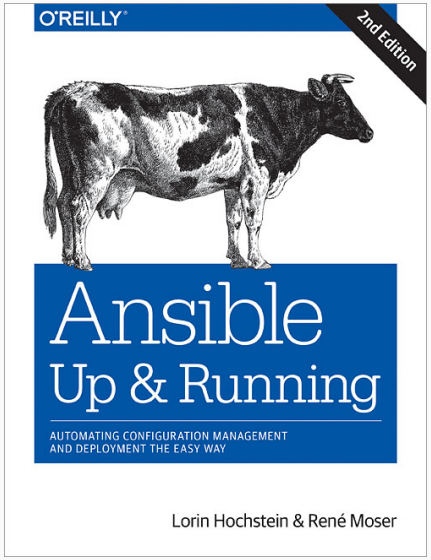
Useful Software Design Principles

Not only for Ansible modules ...

- KISS
- YAGNI
- readability!

Links

- Ansible Docs on Modules: Conventions, tips, and pitfalls
- ansible/ansible
- *Ansible: Up & Running, 2nd ed* by Lorin Hochstein & René Moser



The End

Thank You! — Questions?

Martin Schütte
@m_schuett
info@martin-schuette.de

slideshare.net/mschuett/
noti.st/mschuett/