# Writing Ansible Modules

Martin Schütte

11 November 2019

DENOG 11
GERMAN NETWORK OPERATORS GROUP

## Assumptions

You …

- configure servers or network devices
- have already seen Ansible config
- can write shell scripts

This talk …

- is no Ansible how-to
- has more slides online
- is available on 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
```

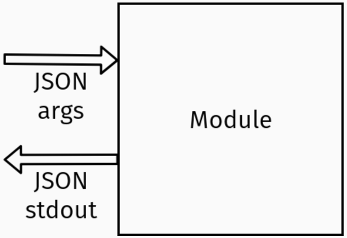
# Concepts

## Module Basics

## What is a Module?

some code snippet to run on the (remote) host

executable with input and output



## Minimal Module

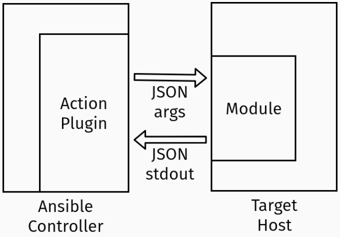
```
#!/bin/sh

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

```
#!/usr/bin/python

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

## Action Plugins call Modules

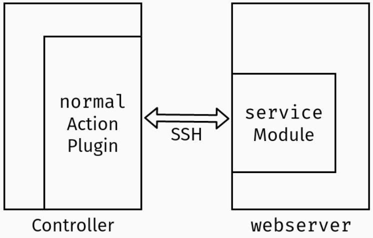


- plugins run on the controller
- may prepare input for modules
- may handle “special” connections (non SSH or WinRM)
- defaults to `normal` to run module on target host

## Concepts

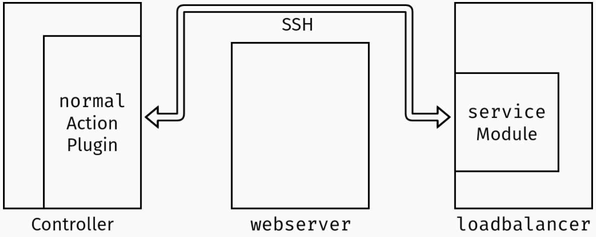
### Orchestration with Host Delegation

## normal SSH Target



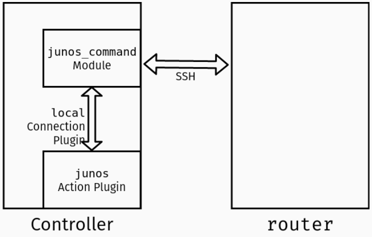
```
# in Playbook
- hosts: webserver
  tasks:
  - name: webserver reload
    service:
        name: httpd
        state: reloaded
```

## normal SSH Target, with delegate_to



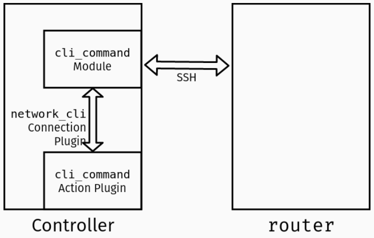
```
- hosts: webserver
  tasks:
  - name: webserver reload
    # ...
  - name: loadbalancer reload
    delegate_to: loadbalancer
    service:
        name: nginx
        state: reloaded
```

## Network, Vendor Specific `junos_command`



```yaml
- hosts: router
  tasks:
  - name: get interfaces
    connection: local
    junos_command:
      command: show interface terse
      provider:
        host: router
        username: foo
```

## Network, New Generic `cli_command`



```yaml
- hosts: router
  tasks:
  - name: get interfaces
    cli_command:
      command: show interface terse

# uses Ansible inventory to read variables
#   ansible_network_os=junos, ansible_connection=network_cli,
#   ansible_user, ansible_password, ansible_ssh_common_args
```

## 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: ipify API

## Documentation

```
ANSIBLE_METADATA = {'metadata_version': '1.1',
                    'status': ['stableinterface'],
                    'supported_by': 'community'}
DOCUMENTATION = r'''
---
module: ipify_facts
short_description: Retrieve the public IP of your internet gateway
version_added: '2.0'
options:
  api_url: ...
'''

EXAMPLES = r'''
# Gather IP facts from ipify.org
- name: Get my public IP
  ipify_facts:

# Gather IP facts from your own ipify service endpoint with a custom timeout
- name: Get my public IP
  ipify_facts:
    api_url: http://api.example.com/ipify
    timeout: 20
'''

RETURN = ...
```

## ansible-doc

```
$ ansible-doc --snippet ipify_facts
- name: Retrieve the public IP of your internet gateway
  ipify_facts:
      api_url:           # URL of the ipify.org API service.
      timeout:           # HTTP connection timeout in seconds.
      validate_certs:    # When set to `NO', SSL certificates will not be validated.

$ ansible-doc ipify_facts
> IPIFY_FACTS    (.../site-packages/ansible/modules/net_tools/ipify_facts.py)

        If behind NAT and need to know the public IP of your internet gateway.

  * This module is maintained by The Ansible Community
OPTIONS (= is mandatory):

- api_url
        URL of the ipify.org API service.
        `?format=json' will be appended per default.
        [Default: https://api.ipify.org/]
        type: str
...
```

## ipify_facts.py

```python
def main():
    global module
    module = AnsibleModule(
        argument_spec=dict(
            api_url=dict(type='str',
                         default='https://api.ipify.org/'),
            timeout=dict(type='int', default=10),
            validate_certs=dict(type='bool', default=True),
        ),
        supports_check_mode=True,
    )

    ipify_facts = IpifyFacts().run()
    ipify_facts_result = dict(changed=False,
                              ansible_facts=ipify_facts)
    module.exit_json(**ipify_facts_result)

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

## ipify_facts.py

```python
class IpifyFacts(object):
    def __init__(self):
        self.api_url = module.params.get('api_url')
        self.timeout = module.params.get('timeout')

    def run(self):
        result = {
            'ipify_public_ip': None
        }
        (response, info) = fetch_url(module=module,
                url=self.api_url + "?format=json",
                force=True, timeout=self.timeout)

        if not response:
            module.fail_json(msg="No valid or no response ...")

        data = json.loads(to_text(response.read()))
        result['ipify_public_ip'] = data.get('ip')
        return result
```

## Usage in Tasks

```
- name: get IP from alternative service endpoint
  ipify_facts:
    api_url: https://api6.ipify.org
  register: ip_public

- name: debug output
  debug:
    msg: |
      fact: {{ ipify_public_ip }}
      reg: {{ ip_public.ansible_facts.ipify_public_ip }}
```

```
TASK [my_role : debug output] **********************
ok: [server] => {
    "msg": "fact: 2001:db8:1:2::42\nreg: 2001:db8:1:2::42\n"
}
```

## Writing Modules

### Patterns & Misc. Hints

## 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)

## “standard library” `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)`

## 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 $=$ then: done, do nothing
   if $\neq$ then: action to change state

## Check Mode/“Dry Run”

- Return information but never apply changes
- Optional, but recommended for modules

Example in `hostname` module:

```
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
```

## Diff Return Value

Example from hostname:

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

Example output, sample module:

```
TASK [set hostname] *****************************************
--- before
+++ after
@@ -1 +1 @@
-hostname = workstation.example.org
+hostname = controller.example.org

changed: [workstation] => {"ansible_facts": {...},
    "changed": true, "name": "controller.example.org"}
```

## 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)
```

## 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")
```

## Writing Modules

### Debugging

## Debugging Tools and Tips

Dev environment:

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

Module tools:

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

## 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

```python
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': '...'}
```

## Writing Modules

### Beyond Python

## Ansible Modules in Other Languages

- Python: the default choice, best tools and support. Also required for network modules/plugins on controller.
- PowerShell: officially supported for modules on Windows
- Scripting Languages: work fine for modules, but lack `AnsibleModule` standard library
- Binary Executables: possible but not practical. – Instead install with OS package, then use `command` or a thin wrapper module.

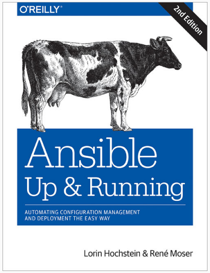
## Conclusion

## Conclusion

- It is easy to write Python modules for Linux-like targets.
- Network devices are hard (connections, OS, CLI variation). Community, Red Hat, and vendors are working on better abstractions.
- Ansible project moves fast (release $2.9 \neq 2.3 \neq 1.8$).
- Check Module Maintenance Levels.
    - Core: Ansible Engineering Team
    - Network: Ansible Network Team
    - Certified: Ansible Partners
    - Community

## Links

- Ansible Docs on “Modules: Conventions, tips, and pitfalls”
- ansible/ansible
- *Ansible: Up & Running, 2nd ed* by Lorin Hochstein & René Moser (covers Ansible 2.3)
- Ansible Docs on “Ansible for Network Automation”
- Network Working Group, ansible/community


O'REILLY
2nd Edition
Ansible
Up & Running
AUTOMATING CONFIGURATION MANAGEMENT AND DEPLOYMENT THE EASY WAY
Lorin Hochstein & René Moser

## The End

Thank You!

## The End

Thank You! — Questions?

## The End

Thank You! — Questions?

Martin Schütte
@m_schuett
info@martin-schuette.de

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