Ansible modules

2016-05-05 Prague DevOpsMeetup

Tomáš Kadlec, CTU FIT, Office for ICT Services
kadleto2@fit.cvut.cz, @ttknet

Why modules?

There are a lot of them. And we have shell: after all!

Why modules

There is more than 450 modules available

- basic utilities, infrastructure, network, virtualization, cloud ops

But what to do if there is not one for me?

- use a shell/script task
- create a role
- **create a module**

Do you see any risks?

Shell tasks

What are the risks of using them?

Not goal based

- Ansible tries to be!

Complex, hard to maintain

- DRY principle violation!

Idempotent?!

- multiple consequent runs should have same results

Roles

What are the risks of using them?

Help you to stay DRY

Optimal solution?

- shell task will be used again

May become complex, hard to maintain

- internal structure
- it’s not a programming language

Idempotent?

Modules

What I get? How much does it cost?

- reusable components
- full power of a programming language (of your choice of course)
- idempotent
  - with possibility of dry run

Costs?

- programming language knowledge
- maintain module’s codebase

Developing modules

Using Python

The most convenient way

Writing modules Python

- Available on majority of systems
- Preferred way
- Module boilerplate

```
from ansible.module_utils.basic import *

…

module = AnsibleModule(
  argument_spec = dict(
    state     = dict(default='present', choices=['present', 'absent']),
    name      = dict(required=True),
    enabled   = dict(required=True, type='bool'),
    something = dict(aliases=['whatever'])
    )
)

…

main()
```

Python - module boilerplate

```
module = AnsibleModule(
    argument_spec = dict(
        devices   = dict(required=True),
        schema    = dict(required=True),
        drop      = dict(required=False, default=False)
    )
)

def main():
    devices = module.params['devices']
    schema = module.params['schema']
    drop = module.params['drop']
    …
    module.fail_json(msg="Partitioning of {0} differs from
        expected.".format(device), violations=violations)
    …
    module.exit_json(changed=true|false, changes=changes)
```

Python - module boilerplate

Requirements

And best practices

- Object called ‘name’ (e.g. package)
- Minimum of dependencies possible
- Check for dependencies
- **Modules must be self-contained**
- Output must be valid JSON only, toplevel is a hash (dictionary)
- Return codes from modules are actually not significant (but 0=success, non-zero=failure
- Return only relevant output (memory!)

cfi-parted.py

Using BASH

Yes, it is possible!

BASH module

Python vs. BASH

What we don’t have?

- Parsing arguments
- Means for reporting
  - success
  - failure
- JSON

- Arguments are passed as a file
- Name of the file is the first positional argument of the module

```
root=/tmp packages=a,b,c portage=/neexistuje
```

- How to parse it?

- Arguments are passed as a file
- Name of the file is the first positional argument of the module

```
root=/tmp packages=a,b,c portage=/neexistuje
```

- Easily!

```
# NOT SAFE
if [ -f "$1" ]; then
    eval $(cat "$1")
fi
```

BASH module

Python vs. BASH

What we don’t have?

- ~~Parsing arguments~~
- Reporting back to the controller
  - success
  - failure
- JSON

Success

```
{
  "changed" : True|False,
  "msg"     : "..."
}
```

Failure

```
{
  "failed" : True,
  "msg"    : "failed setting the time"
}
```

Reporting - expected messages

```
function toJson {
    (( count % 2 != 0 )) && {
        toJson 'failed' 'true' 'msg' 'toJson requires even ...'
        exit 127
    }
    echo -n "{"
    while [ "$#" -gt 0 ]; do
        if [[ "$2" =~ ^true|false$ ]]; then
            echo -n "\"$1\": $2"
        else
            echo -n "\"$1\": \"${2//\"/\'}\""
        fi
        shift 2
    done
    echo -n "}"
}
```

Reporting - toJson (excerpt)

BASH module

Python vs. BASH

What we don’t have?

- ~~Parsing arguments~~
- ~~Reporting back to the controller (JSON)~~
  - ~~success~~
  - ~~failure~~

cfi-emerge.sh

Questions?

Tomáš Kadlec
kadleto2@fit.cvut.cz
@ttknet