How to speed up your (API client) modules

Gonéri Le Bouder
Senior Software Engineer

AnsibleFest

A bit about myself

Work with the Ansible Cloud team, with a focus on VMware technologies.

@goneri on GitHub

API client modules

Modules designed to interact with an API

1. Load Python and the extra dependencies
2. Open a session
3. Do the API call
4. Wrap the result and send it back to Ansible

How Ansible works

For every task, Ansible will

1. starts a new Python process on the target host
2. send the module and run it on target host
3. collect the result

A new Python process per task

Python start-up time is pretty good

```
$ time python -c "print('some Python')"
some Python

real    0m0.029s
user    0m0.023s
sys     0m0.006s
```

But the modules import can slow it down

VMware

```
$ time python -c "from pyVmomi import vim; print('some Python')"
some Python

real    0m0.138s
user    0m0.122s
sys    0m0.016s
```

But the modules import can slow it down

Azure

```
$ time python -c "from azure.mgmt.compute import
ComputeManagementClient; from azure.mgmt.rdbms.mariadb import
MariaDBManagementClient; print('some Python')"
some Python

real    0m0.218s
user    0m0.195s
sys     0m0.021s
```

AnsibleFest

But the modules import can slow it down

OpenStack

```
$ time python -c "import openstack; print('some Python')"
some Python

real    0m0.444s
user    0m0.378s
sys     0m0.038s
```

Module loading

Each time a task runs, a Python process import the dependencies

- Only load the subset of dependencies that you really need
- Be cautious when you add a new import

Session opening

Before anything, the module need to open a session and authenticate itself

- ping will matter, e.g: if you're API endpoint is 200ms away
- the authentication itself may be slow because of the backend (e.g: LDAP or AD)

Ansible Module Turbo

How to speed up your modules start-up

Ansible Module Turbo

Concept

Give a way to reuse your Python objects, between tasks execution.

- libraries are loaded just once
- reuse the existing session

Prepare your environment

AnsibleFest

Fetch the new dependency

Ansible Module Turbo is part of the cloud.common collection

```
ansible-galaxy collection install cloud.common
```

Adjust your collection metadata

Add a dependency on the cloud.common collection in galaxy.yml

```
namespace: vmware
name: vmware_rest
readme: README.md
authors:
- Ansible (https://github.com/ansible)
description:
license_file: LICENSE
tags: ["cloud", "vmware", "virtualization"]
dependencies:
  cloud.common: '*'
```

AnsibleFest

Good enough for a first test run (1/2)

turbo.demo is a demo module for the ansible_module.turbo. Use it to validate your installation:

```
- hosts: localhost
  gather_facts: false
  tasks:
      - cloud.common.turbo_demo:
        with_sequence: count=10
```

AnsibleFest

Good enough for a first test run (2/2)

During the playbook execution

- the Python keeps the same PID
- the counter is increased after every execution

When the playbook is restarted

- it still runs with the same PID
- the counter continue to increase

https://asciinema.org/a/358962

Example

An example with the os_keypair module, the playbook runs the module 6 times.

- first time is slow >3s
- next 5 calls are below 0.6s

https://asciinema.org/a/345197

Adjust your module

Tune up your module (1/3)

And adjust your modules, to load the alternative AnsibleModule

```
~~from ansible.module_utils.basic import AnsibleModule~~
from
ansible_collections.cloud.common.plugins.module_utils.turb
o.module import AnsibleTurboModule as AnsibleModule
```

AnsibleFest

Tune up your module (2/3)

Identify where to add a cache

For instance, this function returns a new client:

```
import MySDK
def my_slow_function():
      return my_sdk.Client()
```

Tune up your module (3/3)

You can cache a function result with a simple Python construction:

```
def my_slow_function():
      if my_slow_function.i:
            return my_slow_function.i❶
      my_sdk = importlib.import_module("MySDK")❷
      my_slow_function.i = my_sdk.Client()❸
      return my_slow_function.i
my_slow_function.i = None❹
```

note: You can also use a library like async_lru.

AnsibleFest

To summarize

1. install cloud.common
2. load AnsibleTurboModule instead of AnsibleModule
3. delay the import
4. cache the session

Under the hood

How it works?

Ansible Module Turbo starts a local process and delegate all the operation to it.

It uses Python's asyncio internally:

- Python 3.6+
- you can also use asyncio coroutine in your module

The daemon kills itself after 15s with no activity.

Conclusion

Good way to speed up your module if

- it depends on a large SDK
- the initial session creation is slow

Easy to switch to Ansible Module Turbo

- limited amount of code to adjust
- keep in minds it depends on Python3.6

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

youtube.com/user/RedHatVideos

linkedin.com/company/Red-Hat

facebook.com/ansibleautomation

twitter.com/ansible

AnsibleFest