Skip Gathering Facts in Ansible

Red Hat Ansible

There are **Ansible playbooks** which depend on the most up-to-date information found on each node. That's where fact gathering is a much needed help. But there are also simpler more predefined playbooks, which don't need fact gathering and can therefore gain performance if no facts are collected.

Why Fact Gathering in Ansible Takes Time

Fact gathering means **Ansible** runs a number of commands to confirm the most recent values for important indicators and parameters.

Run against my freshly installed **RHEL 8** based PC, this takes

roughly 4 seconds. Part of this can be to how RHEL is configured (and that it’s still a work in progress), but part of this amount of time is defined by the sheer number of facts: more than 1000!

Typical Facts Collected By Ansible

This is not a complete list, I’m just giving you examples to indicate why facts collection may be time consuming:

- hardware parameters of remote system
- storage devices (types, models, sizes, capabilities)
- filesystems and logical volume managers (objects, types, sizes)
- OS distro information
- network devices and full list of their capabilities
- environment variables

Disable Fact Gathering in Ansible

Since I don’t really need to re-establish hardware specs or logical volumes layout of my RHEL 8 desktop every time I run some Ansible post-configuration, I decided to disable fact gathering and shave 4-5 sec at the start of each playbook run.

Simply specify this at the top of your **Ansible playbook**:

gather_facts: no

In on of my playbooks, this is how it looks:

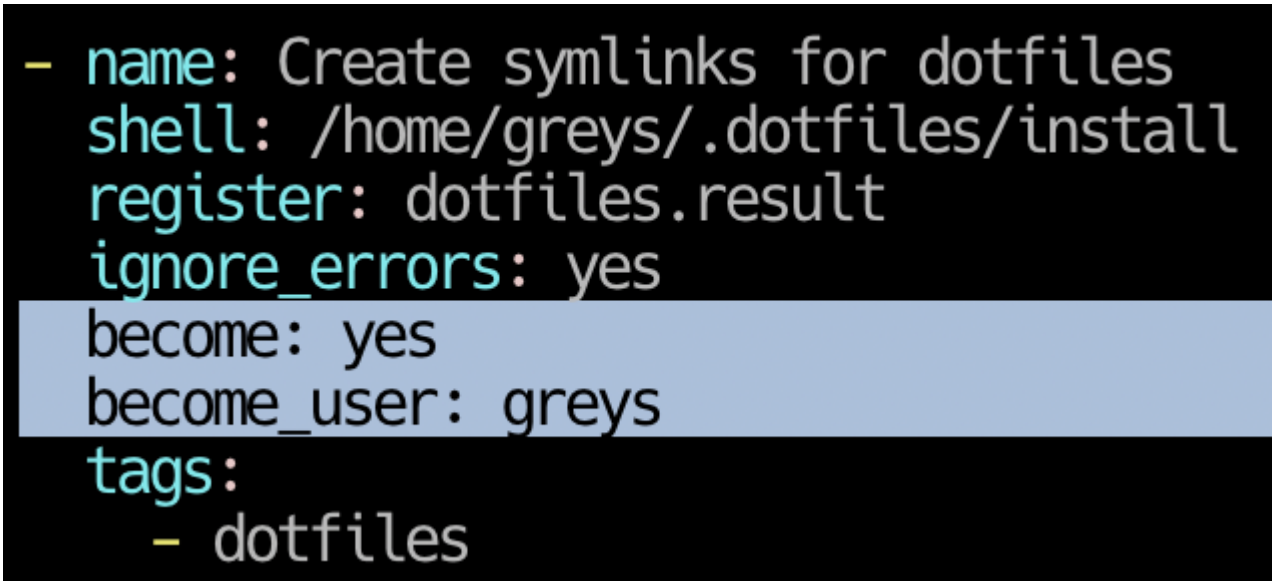
```
---
- name: Baseline
  hosts: desktops
  gather_facts: no
```

This really made a noticeable difference. Have fun!

See Also

- Ansible
- Getting started with Ansible
- Create a Unix group with Ansible
- Specify User per Task in Ansible

Specify User per Task in Ansible



become_user per task in Ansible

Turns out, **become_user** directive can be used not only for privilege escalation (running **Ansible** playbooks as root), but also for becoming any other when you want certain tasks run as that user instead of root.

Default Ansible Behavior for Running Tasks

I had the following piece of code, running **/home/greys/.dotfiles/install** script. It didn't run as intended, creating symlinks in /root directory (because that's what Ansible was running the task as):

```
- name: Create symlinks for dotfiles
  shell: /home/greys/.dotfiles/install
  register: dotfiles.result
  ignore_errors: yes
  tags:
    - dotfiles
```

Specify User for an Ansible Task

become_user parameter can be specifed per task or per playbook, apparently. So that's how you specify it per task – in my example to run the **Create symlinks for dotfiles** task as my user **greys**:

```
- name: Create symlinks for dotfiles
  shell: /home/greys/.dotfiles/install
  register: dotfiles.result
  ignore_errors: yes
  become: yes
  become_user: greys
  tags:
    - dotfiles
```

See Also

- **Ansible**
- How To Create a Unix Group with Ansible
- Dry Running Ansible Scripts
- Getting Started with Ansible

Running Ansible Scripts in

Dry Run Mode

Red Hat – Ansible

Quite often you're clear on what your **Ansible** playbook should be doing and perhaps even know the syntax is correct, but there are still concerns about the implications. In such cases it's best to run your **Ansible** playbook in dry run mode: preview changes without actually making them.

How To Preview Ansible Playbook Run

Simply add the **–check** parameter to the end of your **ansible-playbook** command:

```
$ ansible-playbook try.yaml --check
```

This will show you exactly the changes the playbook would have made, but without actually making them.

Ansible would even go as far as computing the necessary changes as if you made them, lining up variable values etc. It would also show you what will get changed as if it's already changed – but without any risks.

That's it for today, this is a very short note but I'll come back and revisit it soon to provide examples.

See Also

- Red Hat Ansible
- Getting started with Ansible
- Ansible 2.0
- Ansible Reference (future page)
- Ansible changelog (future page)

Getting Started with Ansible

ANSIBLE

I’ve finally made time to go through a couple more **Ansible** books and online courses. Have been using **Ansible** for more than a year but this time I’ll make it into a number of Unix Tutorial projects to automate as much of my home office setup and online/cloud infrastructure for my consultancy Tech Stack Solutions as possible.

Installing Ansible in Ubuntu 19.04

Following the official documentation from **docs.ansible.com**, these are the commands to **install Ansible on most Ubuntu systems**:

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

Creating Ansible inventory (hosts) file

I have created the following basic file for now. This is a newer format of the hosts inventory file, YAML – much more structured and easy to read:

```
all:
  hosts:
    becky:
      ansible_port: 202
      ansible_host: 192.168.1.66
```

As you remember, **becky** is one of my Raspberry Pi systems. 192.168.1.66 is the primary IP it has on my local network, and 202 is the SSH port – even though it's an internal system not accessible from Internet, I still don't like using default SSH port 22.

Pick or Create a Directory for Ansible

I simply created /home/greys/proj/ansible folder, there will be opportunities to refactor it later.

```
$ mkdir /home/greys/proj/ansible
$ cd /home/greys/proje/ansible
```

Let’s try listing hosts that Ansible knows about:

```
greys@xps:~/proj/ansible $ ansible --list-hosts all
[WARNING]: provided hosts list is empty, only localhost is
available. Note that the implicit localhost does not match
'all'

hosts (0):
```

Nothing! You want to know why? Because we need to tell Ansible about the inventory file we just created (by default it only knows about system-wide /etc/ansible/hosts file, but I intentionally don’t use it).

We need to create a basic Ansible config file which will specify the new location of my inventory file.

Creating Ansible config file

What I actually did this time is copied global ansible config from /etc/ansible/ansible.cfg to my project directory, and then changed the inventory setting in it with vim editor:

```
greys@xps:~/proj/ansible $ cp /etc/ansible/ansible.cfg .
greys@xps:~/proj/ansible $ vim ansible.cfg
```

I changed the following in it:

```
inventory = /home/greys/proj/ansible/hosts
#inventory= /etc/ansible/hosts
```

If you want to start fresh, you can skip the copying command and simply create new ansible.cfg in your directory, with just the following:

```
[defaults]
inventory = /home/greys/proj/ansible/hosts
```

Anyway, with this config file created and hosts file location updated, we can re-try the same host listing command. And this time it works!

```
greys@xps:~/proj/ansible $ ansible --list-hosts all
hosts (1):
becky
```

Confirming Ansible connectivity to your servers

Just like ping command tests basic connectivity in common network troubleshooting, Ansible has ping command to confirm that it has sufficient access to each host for further management. Ansible ping is essentially an SSH attempt:

```
greys@xps:~/proj/ansible $ ansible all -m ping
becky | UNREACHABLE! => {
"changed": false,
"msg": "Failed to connect to the host via ssh: greys@192.168.1.66: Permission denied (publickey,password).",
"unreachable": true
}
```

This is expected on my Ubuntu XPS 13 laptop, because I haven't setup passwordless SSH access from it to my other servers yet.

This means I need to deploy my XPS 13 public SSH key onto becky host using my SSH password, and then try again:

```
greys@xps:~/proj/ansible $ ssh-copy-id -i /home/greys/.ssh/id_rsa 192.168.1.66 -p 202
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/greys/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
greys@192.168.1.66's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh -p '202' '192.168.1.66'"
```

and check to make sure that only the key(s) you wanted were added.

All done, now let's re-run Ansible ping:

```
greys@xps:~/proj/ansible $ ansible all -m ping
[WARNING]: Platform linux on host becky is using the discovered Python interpreter at /usr/bin/python, but future installation of another Python interpreter could change this. See https://docs.ansible.com/ansible/2.8/reference_appendices/interpreter_discovery.html for more information.

becky | SUCCESS => {
"ansible_facts": {
"discovered_interpreter_python": "/usr/bin/python"
},
"changed": false,
"ping": "pong"
}
```

Great success!

See Also

- **Passwordless SSH** – wow, I don't have a post on this? will fix soon!
- SSH port
- Ansible 2.0 released
- Pro Puppet – a book on Puppet, just to get you into configuration management mood

---

Ansible 2.0

If you're managing configuration with Puppet or Chef, chances are you've heard of Ansible as well.

Just last week we got Ansible 2.0 released which brings quite a few improvents on top of a massive refactoring.

I’m quite late starting with Ansible but very impressed with it so far: it’s a great way of quickly confirming remote server’s state with SSH and sudo AND a neat way of scripting configurations with Ansible playbooks.

I have written my first playbook two weeks ago and need to change them now so that they follow the updated syntax.

Are you guys using Ansible as well?