Ansible Cookbook 2014

René Moser

Tue Nov 25 23:13:45 UTC 2014

Contents

Intro

Basics

How do I use all hosts data in my play or template?

Solution

In a task:

```
- debug: msg="{{ item }}"
  with_items: groups.all
```

In a template:

```
{% for host in groups['all'] %}
  {{ host }}
{% endfor %}
```

Access host variables dynamically:

```
{% for host in groups['all'] %}
  {{ hostvars[host]['your_varibable_name'] }}
{% endfor %}
```

Explanation

Often you want to use every host in inventory, like for a monitoring play or for a dns zone file. The first thing you see is to set the scope to all hosts in inventory e.g. `hosts: all` and then use `delegate_to` in the tasks.

This might work in some cases but it looks not the right way. The more elegant way is to tool over the special group `all` containing all hosts.

How do use all hosts in the host group except the current host ?

Solution

In a task:

```
- hosts: webservers
  tasks:
  - debug: msg=”{{ item }}”
    with_items: groups.webservers
    when: ”item != inventory_hostname”
```

In a template:

```
{% for host in groups[’webservers’] %}
  {% if host == inventory_hostname %}
    {{ host }}
  {% endif%}
{% endfor %}
```

Explanation

In many cluster management you need to set all other nodes in the cluster. This can be done dynamically using `with_items` and `when` or similar logic in a template.

How to make a command or shell run in check mode ?

Solution

Use `always_run: yes` in the task.

Example use case:

```
# This tasks would be skippped without always_run
- name: check if vmware tools are installed
  shell: rpm -qa | grep foo
  always_run: yes
```

Explanation

Commands and shell actions are skipped in check mode. If you want to force exection you can use `always_run: yes`. Also useful in this case is to add `changed_when: false` to always get a `ok` response instead of `changed`.

Testing

How do I test Ansible Playbooks locally?

Solution

Explanation

How do I test Ansible roles?

Solution

Explanation

Secuirty and Privacy

How do I store private data in git for Ansible?

I assume, you use git to manage your ansible projects history.

However, even if you use a private git repo on GitHub, your data are accessable by GitHub employees and and federal law enforcement).

So how can we make sure, anyone can get access to your data, but the users you want?

Below I show you how this can be done:

Solution

Requirements: GnuPG

Create a password file for ansible-vault

Create a password file for ansible-vault, I use pwgen just because I am lazy. After that, we make sure this plain text password file will never be added to our git repo.

```
$ pwgen 12 1 > vault-password.txt
$ echo vault-password.txt >> .gitignore
$ git add .gitignore
$ git commit -m ”ignore vault-password.txt”
```

Use GnuPG to encrypt the password file

Now we use GnuPG to encrypt the password file used for `ansible-vault`:

```
$ gpg --recipient 42FF42FF \
      --recipient 12345678 \
      --recipient FEFEFEFE \
      --recipient EFEFEFEF \
      --encrypt-files vault-password.txt
```

The above command will create an encrypted version of our password file in a new file vault-password.txt.gpg. As you can see we added a few recipient public keys. All owners of the corresponding private keys, and only them, are able to decrypt the encrypted password file.

Hint: So you better be one of them.

Add the file vault-password.txt.gpg to your git repository

The encrypted password file gets into our repo:

```
$ git add vault-password.txt.gpg
$ git commit -m ”store encrypted password”
```

Decrypt and run playbook

```
$ gpg --decrypt vault-password.txt.gpg
$ ansible-playbook --vault-password-file vault-password.txt ...
```

Create an alias for daily usage

For the daily usage, I used to use an alias `ansible-playbook-vault`:

```
$ alias ansible-playbook-vault='test -e vault-password.txt ||
gpg --decrypt-files vault-password.txt.gpg;
ansible-playbook --vault-password-file vault-password.txt $@'
```

This alias will automatically decrypt the vault-password.txt.gpg if necessary and uses the encrypted file with option `--vault-password-file`.

Explanation

Ansible has a tool for encryption of var files like `group_vars` and `host_vars` sind version 1.5, ansible-vault.

However, with ansible-vault you can only encrypt and decrypt var files. Another downside is you have to remember the password and this password must be shared accross your team members.

That is why `ansible-playbook` has an other option for letting you write the password in a file and pass `--vault-password-file <filename>` to `ansible-playbook`. So you do not have to writing it over and over again every time you run a playbook or also meant for automated execution of ansible.

But as you can guess, the password is stored in plain text in a file and we likely want this file to be in a public git repo.

This is the part where asymetric cryptology and GnuPG comes into the game. GnuPG lets you encrypt any file without shareing a password, even for a group of people.

How do I manage root’s authorized_keys file with Ansible?

Solution

Public git repo containing all the ssh public keys

We usually create a separate repo for all ssh public keys. So everyone can send pull request for updating their ssh public key. Of course you should verify them. :)

The repo looks like this below:

```
.
├── user1.pub
├── user2.pub
├── user3.pub
└── user4.pub
```

Playbook to get latest ssh public keys

To make sure we always use the latest version of this repo. We set up a playbook, in which we checkout the lastest state to our local workstation.

```
# filename: sshkeys.yml
---
- hosts: localhost
  connection: local
  tasks:
  - name: check out latest ssh public keys
    git: repo=https://github.com/example/ssh-pubkeys.git dest=./files/ssh-public-keys/
```

Define who can access where

The next step is to setup the vars. Here we define, who should have access to all host by creating the vars section in `group_vars/all`:

```
# group_vars/all
---
ssh_root_users:
  - key: ”{{ lookup(’file’, ’./files/sshkeys/user1.pub’) }}”
  - key: ”{{ lookup(’file’, ’./files/sshkeys/user2.pub’) }}”
  - key: ”{{ lookup(’file’, ’./files/sshkeys/user3.pub’) }}”
  - key: ”{{ lookup(’file’, ’./files/sshkeys/user4.pub’) }}”
```

For webservers, we only give these 2 users access:

```
# group_vars/webservers
---
ssh_root_users:
  - key: ”{{ lookup(’file’, ’./files/sshkeys/user1.pub’) }}”
  - key: ”{{ lookup(’file’, ’./files/sshkeys/user2.pub’) }}”
```

Play for authorized_key deployment

Finally, we extend the playbook for the final deployment of authorized_key:

```
# filename: sshkeys.yml
---
- hosts: localhost
  connection: local
  tasks:
  - name: check out latest ssh public keys
    git: repo=https://github.com/example/ssh-pubkeys.git dest=./files/sshkeys/

- hosts: all
  remote_user: root
  tasks:
  - name: add ssh root keys
    authorized_key: user=”root” key=”{{ item.key }}”
    with_items: ssh_root_users
```

Explanation

The easiest way is to have a single authorized_keys file and use the copy task, of course.

However, often we can not reuse the same authorized_keys on every machine or group of machines. I wanted to have something which has the maximum of flexibily, but to be simple and comprehensible at the same time.

The above solution is a typical example how you can achive this with Ansible.

Networking

How do I add nodes to DNS?

Solution

Explanation

How do I make manage config of a failover cluster?

Solution

Make use of serial keyword in you play. This also called Rolling Update in Ansible's Docs.

```
---
- hosts: database-cluster
  serial: 1
```

Explanation

A failover cluster often consist of 2 nodes only, so we set `serial: 1` to make sure, we do all tasks on one node after the other.

Helpers and Tools

How do I run Puppet agent by Ansible?

Solution

Create a playbook having the following content:

```yaml
# filename: run-puppet.yml
---
- hosts: all
  remote_user: root
  serial: 3
  tasks:
    - command: puppet agent --test --no-noop creates=/var/lib/puppet/state/agent_catalog_run.lock
      register: puppet_result
      changed_when: puppet_result.rc == 2
      failed_when: puppet_result.rc == 1
```

Explanation

The playbook has target to all hosts, but of course you can limit it using `--limit` e.g.:

```
$ ansible-playbook run-puppet.yml --limit webservers
```

The Puppet agent returns code 1 on failure, and 2 on change. That is why we must use `register` to grap the return code.

We also use `serial` to perform a Rolling Update and make sure to skip the command if Puppet agent is already running.

Where can I find an cheat sheet of Ansible?

Solution

Visit `http://wall-skills.com/2014/ansible-cheat-sheet`

Application and Deployments

How do I deploy PHP web applications?

Deployment of web applications has never been easy.

Even there are many good tools which help a lot doing it right, like Capistrano or Fabric.

But why you should learn another tool if Ansible can do the job even better? I want to show you, how I deploy PHP web applications:

Solution

One characteristic of a web application is, that most of the time we are dealing with intepreted scripting languages, like PHP, Python or Ruby. So one solution to deploy an application is to checkout the code from your source code management tool and run some installation sciprts like database migration.

Application source from git repo

This would be too easy right?

```
# filename: app-deployment.yml
---
- hosts: appservers
  vars:
    app_version: v1.0.0
  tasks:
  - name: Checkout the application from git
    git: repo=git://github.com/example/repo.git dest=/srv/www/myapp version={{ app_version }}
    register: app_checkout_result

  - name: Update dependencies
    command: php composer.phar install --no-dev

  - name: Run migration job
    command: php artisan migrate
    when: app_checkout_result.changed
```

But wait, what about database backup? Monitoring downtime? Inform team members? As simple as that.

```
# filename: app-deployment.yml
---
- hosts: appservers
  vars:
    app_version: v1.0.0
  tasks:
  - name: inform team members about start
    local_action: jabber
                  user=deployment@example.net password=secret
                  to=friend@example.net
                  msg="Deployment of App version {{ app_version }} started"
  - name: Set monitroing Downtime
    local_action: nagios action=downtime minutes=10 service=app host={{ inventory_hostname }}

  - name: Checkout the application from git
    git: repo=git://github.com/example/repo.git dest=/srv/www/myapp version={{ app_version }}
    register: app_checkout_result

  - name: Backup the database
    command: mysqldump myapp  --result-file=backup-{{ ansible_date_time.epoch }}.sql
    when: app_checkout_result.changed

  - name: Update dependencies
    command: php composer.phar install --no-dev

  - name: Run migration job
    command: php artisan migrate
    when: app_checkout_result.changed

  - name: inform team members about end
    local_action: jabber
                  user=deployment@example.net password=secret
                  to=friend@example.net
                  msg="Deployment of App version {{ app_version }} ended"
```

My applications are usually in a git repo. This git repo may be a private repo, so you even have to authenticate for read access. Further git is a dependency of our installation. We may or can not have git installed on the webservers. In this case, we just checkout the repo locally and sync the files using rsync or git-ftp:

```
# filename: app-deployment.yml
---
- hosts: appservers
  vars:
    app_version: v1.0.0
  tasks:
  ...
  - name: Checkout the application from git
    local_action: git repo=git://github.com/example/repo.git dest=/srv/www/myapp version={{ app_version }}
    register: app_checkout_result

  - name: Sync the applications
    local_action: command rsync --ignore .git ...

  ...
```

Explanation