Introduction to Ansible

05.12.2020

Tasks

* Tasks are basic building blocks of Ansible's execution and configuration.
* Running adhoc commands are great for troubleshooting and quick testing against your inventory
* The return results will gives us details about th success or failure of the executed commands.
* We can define tasks in form of plays ran within playbooks, which is the real power of Ansible

Playbooks

* Playbooks are a way to conggegate ordered processes and manage configuration needed to build out a remote system.
* Playbooks make configuration management easy and gives us the ability to deploy to a multi-machine setup.
* Playbooks can declare configuration and orchestrate steps (normally done in a manual ordered process) and when run, can ensure our remote system is configured as expected.
* The written tasks within a playbook can be run synchronously or asynchronously.
* Playbooks gives us the ability to create infrastructore as code and manage it all in source control.

Design of Playbooks

Update
update all packages
patching needed
Install
install item x
install item y
Configure
setup services
update config files
restart services
Check status
ensure up status

* List out everything we need want to apply to each instance
* Group them according to configuration usage.
* Ensure they are logically defined order.
* Run each tasks according to the order they are listed.

* Playbooks use YAML syntax which allow you to model a configuration or a process.
* Playbooks are composed of one or more plays in a list.
* The goal of a play is to map a group of host to a tasks that are used to call Ansible modules.
* By composing a playbook of multiple plays, it makes it possible to orchestrate multi-machine deployments and allows us to run certain steps on all machines in a group.

Playbooks in Action

① Package management

Install all packages needed to run our system.
* patching
* package manager

Example Playbook

```
- hosts: loadbalancers
  tasks:
  -name: Install Apache
   yum : name=httpd state=latest
```

② Configure infrastructure

Configure our system with necessary application files or configuration files that are needed to configure environment.
* copy files
* edit configuration files

Example Playbook

```
---
  - hosts: loadbalancers
    tasks:
    - name: Copy config file
      copy: src=./config.cfg dest=/etc/config.cfg

  - hosts: webservers
    tasks:
    - name: Synchronize folders
      synchronize: src=./app dest=/var/www/html/app
```

③ Service Handlers

Create service handlers to start, stop, or restart our system when changes are made.

* command
* service
* handlers

Example Playbook

```
---
  - hosts: loadbalancers
    tasks:
    - name: Configure port number
      lineinfile: path=/etc/config.cfg regexp='^port' line='port=80'
      notify: Restart apache

    handlers:
    - name: Restart apache
      service: name=httpd status=restarted
```

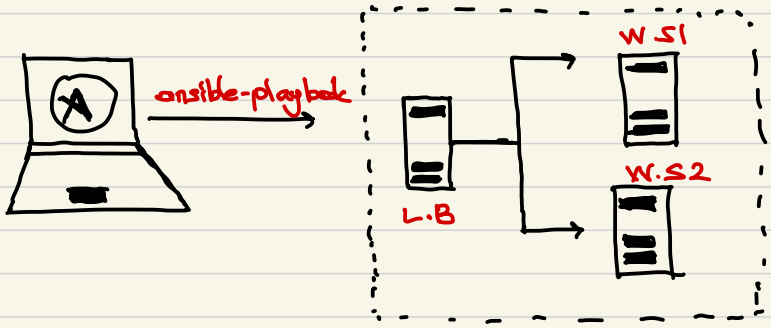
① Package Management → ② Configure infrastructure → ③ Service Handlers

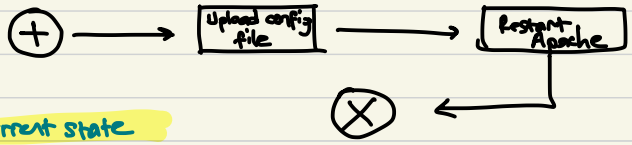
Constructing a system

① Package Management
* apache
* php

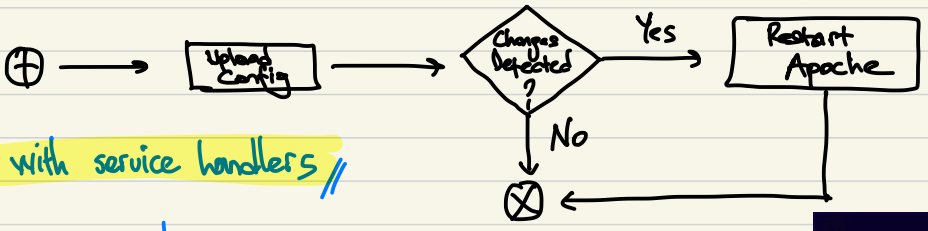
② Configure infrastructure
* upload index.php
* configure php.ini
* configure load balancer

③ Service Handlers
* restart services





Current state



with service handlers //

```
...
  tasks:
    - name: Configure php.ini file
      lineinfile:
        path: /etc/php.ini
        regexp: ^short_open_tag
        line: 'short_open_tag=On'
      notify: restart apache

  handlers:
   - name: restart apache
     service: name=httpd state=restarted
```

Summary

* Playbooks are an ordered list of plays that can run tasks for configuration and orchestration.
* These plays allow us to run commands on a group or subset of servers within our inventory.
* Create infrastructure as code that can be managed in source control.
* Playbooks can be multiple times without affecting previous runs.
* Package management, configuration, service handlers.

- Variables -

Ansible provides us with variables and metadata about the host we are interacting with when running playbooks.

* During the TASK [Gathering Facts] step, these variables become populated.
* Gathers useful facts about our host and can be used in playbooks.
* Use the status module to see all of the facts gathered during the TASK [Gathering Facts]
* Uses jinja2 templating to evaluate these expressions.

Ansible also gives us the ability to create local variables within our playbooks.

* Create playbook variables using vars to create key/value pairs and dictionary/map of variables.
* Nice to use when referencing variables directly in a playbook.
* Create variables files and import them into our playbook.

— Example Playbook —

```
...
  vars:
    path_to_app: "/var/www/html"
    another_variable: "something else"

  tasks:
    - name: Add webserver info
      copy:
        dest: "{{ path_to_app }}/info.php"
        content: "<h1>Hello, World!</h1>"
```

jinja2 template

Ansible also gives us the ability to register variables from tasks that run to get information about its execution.

* Create variables from info returned from tasks run using register.
* Call the registered variables for later use.
* Use the debug module anytime to see variables and debug our playbooks.

```
...
  vars:
    path_to_app: "/var/www/html"

  tasks:
    - name: See directory contents
      command: ls -la {{ path_to_app }}
      register: dir_contents

    - name: Debug directory contents
      debug:
        msg: "{{ dir_contents }}"
```

In order to see variables in gathering facts step following command can be used:

```
ansible -m setup all
```

Summary

* Ansible provides us with many ways to use variables and include them within our setup.
* Use variables within the TASK [Gathering Facts] dictionary.
* Create user-defined variables using the vars feature for in-line variables within our playbooks
* Use the debug module to print messages to standard out.

- Roles -

* Ansible provides a framework that makes each part of variables, tasks, templates, and modules fully independent.
  - → Group tasks together in a way that is self containing
  - → Clean and pre-defined dictionary structure.
  - → Break up the configurations into files.
  - → Reuse code by others who need similar configurations
  - → Easy to modify and reduces syntax errors.

Example Directory Structure

```
setup-app.yml
roles/
   webservers/
      tasks/
        - main.yml
      vars/
        - main.yml
      handlers/
        - main.yml
```

Create a role

```
$ ansible-galaxy init roles/webservers
```

- Check Mode ("Dry Run") -

* Reports changes that Ansible would have to make on the end hosts rather than applying the changes.
  - → Run Ansible commands without affecting the remote system
  - → Reports changes back rather than actually making them
  - → Great for one node at a time basic configuration management use cases.

Example dry run execution

```
$ ansible-playbook setup.yml --check
```

- Error Handling -

* Change the default behaviour of Ansible when certain events happen that may or may not need to report as a failure or changed status.
  - → Sometimes a non-zero exit code is a-okay.
  - → Sometimes commands might not always need to report a changed status.
  - → Explicitly force Ansible to ignore errors or changes that occur.

```yaml
# check-status.yml
---
  - hosts: webservers:loadbalancers
    tasks:
    - name: Check status of apache
      command: service httpd status
      changed_when: false

    - name: This will not fail
      command: /bin/false
      ignore_errors: yes
```

- Tags -

* Assigning tags to specific tasks in playbooks allows you to only call certain tasks in a very long playbook.

  - → Only run specific parts of a playbook rather than all of the plays.
  - → Add tags to any tasks and reuse if needed.
  - → Specify the tags you want to run (or not run) on the command line.
  - → Tasks can also be skipped with following command:

```
$ ansible-playbook setup-app.yml --skip-tags upload
```

```yaml
# setup-app.yml
…
  tasks:
    - name: Upload application file
      copy:
        src: ../index.php
        dest: "{{ path_to_app }}"
      tags: upload

    - name: Create simple info page
      copy:
        dest: "{{ path_to_app }}/info.php"
        content: "<h1>Hello, World!</h1>"
      tags: create
```

Execute playbook with tags

```
$ ansible-playbook setup-app.yml --tags upload
```

only runs specific tasks

- Ansible Vault -

* Ansible "Vault" is a way to keep sensitive information in encrypted files, rather than plain text, in our playbooks.
  - → Keep passwords, keys, and sensitive variables in encrypted vault files.
  - → Vault files can be shared through source control.
  - → Vault can encrypt pretty much any data structure file used by Ansible.
  - → Password protected and the default cipher is AES.

Create encrypted data file

```
$ ansible-vault create secret-variables.yml
```

Prompt for password

```
$ ansible-playbook setup-app.yml --ask-vault-pass
```

- Prompts -

* There may be playbooks you run that need to prompt the user for certain input. You can do this using the "vars_prompt" section.
  - → Can use the users input as variables within our playbooks.
  - → Run certain tasks with conditional logic.
  - → Common use is to ask for sensitive data.
  - → Has uses outside of security as well.

```yaml
…
  vars_prompt:
    - name: "upload_var"
      prompt: "Upload index.php file?"

  tasks:
    - name: Upload application file
      copy:
        src: ../index.php
        dest: "{{ path_to_app }}"
      when: upload_var == 'yes'
      tags: upload
```