Ansible in 4 Hours

Sander van Vugt
mail@sandervanvugt.nl

Ansible in 4 Hours

Course information

Your Instructor

- Sander van Vugt
- mail@sandervanvugt.nl
- http://sandervanvugt.com
- Author of
  - Beginning Ansible
  - Automation with Ansible (expected December 2018)
  - Ansible Certification (expected December 2018)
- Available as recorded video courses on Safaribooksonline
- Also attend my Safaribooksonline Live session "Automation with Ansible"

Agenda

- What is Ansible?
- Installing Ansible
- Configuring Ansible Managed Servers
- Running Ansible Ad-hoc Commands
- Running Ansible Playbooks
- Using Variables
- Using Inclusions

Course Files

- The demo files used in this course are available on github
- Use **git clone https://github.com/sandervanvugt/ansible-3h** to download

Ansible in 4 Hours

Understanding Ansible

What is Ansible?

- Ansible is a Configuration Management System, written in Python
- It uses playbooks to run tasks in a way that always gives the same result
- It is push-based
- It doesn't require any agent on the managed node
- Apart from managing computers, it's doing very well in managing network devices as well
- Configurations are written in YAML
- Ansible generated Python scripts that are started on the managed nodes

Ansible compared to others

- YAML is easy to learn and read
- No agent to install on managed hosts
    - You'll need Python and SSH though
- Push based, which gives you more control over the process
    - An optional ansible-pull tool is available for if you want to be able to pull configurations
- Many modules are available
- Idempotent: running the same playbook multiple times will give you the same results

Required Skills

- Use SSH
- Use Linux Commands
- Install software
- Use sudo
- Manage permissions
- Manage services
- Work with variables

Ansible in 4 Hours

Installing Ansible

Installing the Ansible Controller

- On Linux: use the version in the repositories
  - Might bring you a somewhat older version
  - Easiest and for that reason recommended
- On MacOS, use Homebrew package manager
- Or else, use the Python package manager pip

Configuring SSH to Manage Hosts

- Set up SSH Key-based authentication
  - **ssh-keygen**
- This creates a public key as well as a private key
  - The server that has the public key sends a challenge that can only be answered with the private key
  - Keep the private key in the local user account on the control node
  - Send the public key to the ~/.ssh/authorized_keys file in the target user home directory
    - Use **ssh-copy-id user@remotehost**
    - Notice that the local user name and the remote user name do NOT have to be the same

The Inventory File

- The inventory file identifies and groups managed hosts
- The inventory file may be indicated with the **-i** option
- Typically, you would create an Ansible project directory in your home directory, and put an inventory file in there
- You can specify which inventory to use in the ansible.cfg file
- For use in more advanced projects, multiple inventory files may be used

Managing Managed Hosts

- After installation, you can use the **ansible** command to run ad-hoc commands against remote hosts
- Remote hosts need to be specified in the inventory file
- The inventory file allows you to define managed hosts
- Hosts are specified by their name or IP address
- Hosts may be mentioned in the inventory more than once
  - This allows you to create logical groups
- In **ansible** commands, you'll mention host names, as well as the inventory file that you're going to use
  - **ansible server1.example.com,server2.example.com -i myinventory --list-hosts**

Lab 1: Installing Ansible

1. **useradd ansible; passwd ansible; su - ansible**
2. On both nodes: **sudo yum install python2 epel-release -y**
3. Remaining steps on control: **sudo yum install -y ansible**
4. **ssh server1.ansible.local**
5. **ssh-keygen**
6. **ssh-copy-id server1.ansible.local**
7. **mkdir ~/install**
8. **vim ~/install/inventory**

```
control.ansible.local
server1.ansible.local
```

9. **ansible all -i inventory --list-hosts**

Ansible in 4 Hours

Using Inventory

Understanding Inventory

- Ansible uses an inventory file, which must be used to identify managed hosts
- The location of the inventory file can be anywhere and is specified in the ansible.cfg file
  - /etc/ansible/hosts
  - current project directory
  - specified with the -i option while running Ansible commands
- Inventory files may be statically created or dynamically generated
  - Static inventory works for small environments
  - Dynamic inventory uses 3rd party scripts to identify hosts in a specific environment

Working with Dynamic inventory

- When using the **ansible** command, use the **-i** option, followed by the inventory script you'd like to use
  - Ensure that the inventory script is executable
- Write your own script or use a script that is available for the different externally supported cloud environments

Using Groups in Inventory

- An inventory file contains a list of hosts
- Hosts may be grouped to make referring to hosts easier
- A host can be a part of multiple groups
- The host group **all** is always present and doesn't have to be defined

Nesting Host Groups

- Host Groups may be nested in inventory

```
[webservers]
web1.example.com
web2.example.com

[dbservers]
db1.example.com
db2.example.com

[servers:children]
webservers
dbservers
```

Ansible in 4 Hours

Using ansible.cfg

The ansible.cfg file

- The ansible.cfg file is used to specify generic settings
  - How to escalate permissions
  - Where to find the inventory file
  - And more
- The following locations are used
  - $ANSIBLE_CONFIG
  - ./ansible.cfg
  - ~/.ansible.cfg
  - /etc/ansible/ansible.cfg
- It is common practice to put it in the current project directory
- Using section headers is important!

Common ansible.cfg parameters

```
[defaults]
inventory = /etc/ansible/hosts
remote_user = ansible
host_key_checking = False

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = False
```

Configuring sudo for Privilege Escalation

- Privilege escalation needs a sudo configuration
  - Set become parameters in ansible.cfg
  - Or use -b with your ansible command to escalate and run the command as root
- For the Ansible default account, create a sudo file on all Ansible managed hosts:

```
# cat /etc/sudoers.d/ansible
ansible ALL=(ALL) NOPASSWD: ALL
```

Testing Connectivity

- At this point, your configuration should be ready for use, time to run some ad-hoc commands
  - **ansible server1 -m command -a who**
  - **ansible all -a who**

Ansible in 4 Hours

Using Ad-hoc Commands

Why Use Ad-hoc Commands

- You’ll typically want to create playbooks to automate tasks against multiple Ansible servers
- To quickly make changes to many managed hosts, ad-hoc commands are convenient
- Ad-hoc commands can also be used for diagnostic purposes, like querying a large number of hosts
- In ad-hoc commands, modules are used

Understanding Modules

- A module is used to accomplish specific tasks in Ansible
- Modules can run with their own specific arguments
- Modules are specified with the -m option, module arguments are referred to with the -a option
- The default module can be set in ansible.cfg. It's predefined to the **command** module
  - This module allows you to run random commands against managed hosts
  - As command is the default module, it doesn't have to be referred to using -m module, just use **-a command**
  - Notice that the command module is not interpreted by the shell on the managed host and for that reason cannot work with variables, pipes and redirects
  - Consider using the **shell** module if you need full shell functionality

Introducing 3 Modules

- **command**: runs a command on a managed host
  - command is the default module, so you don't really have to specify it
  - If the command you want to run contains spaces, make sure to use quotes
- **shell**: runs a command on managed host through the local shell
- **copy**: copy a file, change content on a managed host in a target file

Ad-hoc Command Examples

- **ansible all -m command -a id**
  - Runs the command module with the **id** command as its argument against all hosts. Notice that this needs [all] to be defined in the inventory
- **ansible all -m command -a id -o**
  - Same command, but provides a single line of output
- **ansible all -m command -a env**
  - Unexpected results, as the command module doesn’t work through the shell
- **ansible all -m shell -a env**
- **ansible managed1.ansible.local -m copy -a 'content="Ansible managed\n" dest=/etc/motd'**

Ansible Module Documentation

- Authoritative documentation is on docs.ansible.com
- Request a list of currently installed modules using **ansible-doc -l**
  - Use **ansible-doc <modulename>** to get module specific information
  - Use **ansible-doc -s <modulename>** to produce example code that you can include in a playbook

From Ad-hoc to Playbook

- Modules can be included using the **ansible -m <modulename>** command
  - **ansible -m yum -a "name=vsftpd state=latest" all**
- Or included in an Ansible task in a playbook

```
tasks:
- name: Install a package
  yum:
    name: vsftpd
    state: latest
```

Ansible in 4 Hours

Running Playbooks

Understanding Playbook Components

- Ansible playbooks are written in YAML
- A Playbook contains one or more plays
- A Play is a series of related tasks
- Tasks are using Ansible Modules to get things done
- Variables are used to make playbooks more flexible
- Ansible playbooks often work with includes, to manage tasks in a modular way

Sample Playbook

```
---
- name: deploy vsftpd
  hosts: node1.example.com
  tasks:
  - name: install vsftpd
    yum: name=vsftpd
  - name: enable vsftpd
    service: name=vsftpd enabled=true
  - name: create readme file
    copy:
      content: "welcome to my ftp server"
      dest: /var/ftp/pub/README
      force: no
      mode: 0444
...
```

Ansible in 4 Hours

Using Facts

Variables and Facts

- *Variables* make it easier to repeat tasks in complex playbooks
- *Facts* can be used as variables and contain information that Ansible has discovered about a host
  - They can be used in conditional statements in Playbooks
  - The setup module is used to gather fact information
    - **ansible server1 -m setup**
  - Every play runs fact gathering before running the playbook
- *Filters* are used to filter information out of facts
  - **ansible server1 -m setup -a 'filter=ansible_kernel'**

Understanding Variable Syntax

- Variables and Facts may refer to different items
  - **ansible ansible1 -m setup 'filter=ansible_default_ipv4'** returns different keys
- Individual keys may be referred to in dotted syntax
  - **ansible ansible1 -m setup 'filter=ansible_default_ipv4.address'** addresses the IPv4 address
- And can be used in when statement
  - **when: ansible_default_ipv4.address == 192.168.4.81**

Using Custom Facts

- *Custom facts* can be defined by administrators and used as variables on a specific host (group)
  - Should be stored in /etc/ansible/facts.d/*.fact
  - these files have an ini format or a json format

```
[users]
user1 = linda
user2 = anna
```

- Custom facts are stored in the variable ansible_local
- Show a list of all custom facts:
  - **ansible ansible2.example.com -m setup -a 'filter=ansible_local'**

Using Facts

- Facts can be used like variables in the playbooks

```
---
- hosts: all
  tasks:
  - name: Print some Ansible facts
    debug:
      msg: >
        The IPv4 address of {{ ansible_fqdn }}
        is set to {{ ansible_default_ipv4.address }}
        it runs {{ ansible_kernel }}
        and has the following network interfaces:
        {{ ansible_interfaces }}
```

Disabling Fact gathering

- By default, any Ansible commands will start gathering facts
- This slows down the procedure
- Include **gather_facts: no** in the playbook to disable fact gathering

Ansible in 4 Hours

Using Handlers

Adding More Features (2)

- *Handlers* are like a task, but will only run when they have been triggered by a notify statement
- A task notifies the handler by passing the handler's name as argument

```
tasks:
- name: copy file
  copy: src=/downloads/index.html dest=/var/www/html
  notify: restart httpd
handlers:
- name: restart httpd
  service: name=httpd state=restarted
```

About handlers

- Common use for using handlers is to restart a service or to reboot a machine
- Handlers will restart services conditionally
- You may want to consider restarting these services any way, as restarting services typically is fast enough
- Notice that handlers will run in the order as specified in the playbook, but only after all tasks in the playbook have been executed
- Handlers inside an include cannot be notified (see next)

Ansible in 4 Hours

Working with Variables

Understanding Variables

- Using variables makes it easier to repeat tasks in complex playbooks and are convenient for anything that needs to be done multiple times
  - Creating users
  - Removing files
  - Installing packages
  - Storing fact information
- A variable is a label that can be referred to from anywhere in the playbook, and it can contain different values, referring to anything
- Variable names must start with a letter and can contain letters, underscores and numbers
- Variables can be defined at a lot of different levels

Defining Variables

- Variables can be defined in a playbook, from inventory or included from external files
- Defining variables in a playbook

```
- hosts: all
  vars:
    user: linda
    home: /home/linda
```

Defining Variables in Inventory

- Variables can be assigned to individual servers
- Or to host groups (recommended)

```
[webservers]
web1.example.com
web2.example.com

[webservers:vars]
documentroot=/web
```

Using Variable Files

- When using variable files, a YAML file needs to be created that contains the variables
  - This file uses a path relative to the playbook path
- This file is called from the playbook, using **vars_files:**

```
- hosts: all
  vars_files:
    - vars/users.yml
$ cat vars/users.yml
user: linda
home: /home/linda
user: anna
home: /home/anna
```

group_vars and host_vars

- Defining Variables in the Inventory is not recommended
- Instead, create a group_vars and a host_vars directory in the current project directory
- In these directories, create files that match the names of (inventory) hosts and host groups
- In these files, set variables in a key: value format

```
cat ~/myproject/host_vars/web1.example.com
package: httpd
cat ~/myproject/group_vars/web
documentroot: /web
```

Using Variables

- In the playbook, the variable is referred to using double curly braces
- If the variable is used as the first element to start a value, using double quotes is mandatory

```
tasks:
  - name: Creates the user {{ user }}
    user:
      name: "{{ user }}"
```

- Notice the different uses of the variable user!

Using register

- The **register** statement can be used to capture output of a command into a variable
- Use **debug** to show the value of the variable
- While running the playbook, the [debug] section will show the output of the command in the specific task

```
- name: show command output
  hosts: server1
  tasks:
    - name: fetch output of the who command
      command: who
      register: currentusers
    - debug: var=currentusers
```

Ansible in 4 Hours

Understanding Project Directory Structure

Why inclusions?

- When playbooks are becoming too long, separate files can be used to manage individual tasks and variable groups
- This makes it easier to delegate management tasks for specific parts
- Also, it adds modularity
  - Newly installed servers need to run a complete configuration
  - Existing servers may need to run just a subset of the total amount of available task files
- Use **include** to include task files
- Use **include_vars** to include variable files

Using Directories and Files in Ansible

- With the group_vars and host_vars included, it is common for Ansible projects to work with a directory structure
- Ultimately, a project directory can contain a role, which is a standard set of instructions to get anything done

Sample Directory Structure

```
myproject
|--ansible.cfg
|--group_vars
|    |--web
|    |--db
|--host_vars
|    |--web1.example.com
|    | ...
|--inventory
|--site.yml
|--webservers.yml
|--dbservers.yml
```

Ansible in 4 Hours

Using Items

Using Items

- Ansible offers different solutions to implement iterations
- Items are easy to use
  - Define a list of items
  - In the task, use with_items to process the items

```
tasks:
- name: remove services
  yum:
    name: "{{ item }}"
    state: absent
  with_items:
    - httpd
    - vsftpf
```

Ansible in 4 Hours

Ansible Vault and Tower

Understanding Ansible Vault

- To access remote servers, passwords and API keys may be used
- By default, these are stored as plain-text in inventory variables or other files
- Ansible Vault can be used to encrypt and decrypt data files used by Ansible
  - Vault is default part of Ansible
- Alternatively, external key-managent solutions may be used also

Using Ansible Vault

- The **ansible-vault** command can be used to create an encrypted file
- This can also be decrypted using **ansible-vault**
- From within a playbook, an encrypted file can be referred to
- Run the playbook with the --ask-vault-pass option to ask for the password
    - **ansible-playbook --ask-vault-pass webservers.yaml**

Understanding Ansible Tower

- Ansible Tower provides a framework for using Ansible at an enterprise level
  - Central repository of Ansible playbooks
  - Scheduled playbook execution
  - Central web interface
  - role-based access control
  - Centralized logging and auditing
  - REST API
- Using Tower allows easy integration of Ansible with other tools like Jenkins, Cloudforms and Red Hat Satellite