**Ansible Notes**

Automation is a key tool to mitigate human error and quickly ensure that your IT infrastructure is

in a consistent, correct state.

• Ansible is an open source automation platform that can adapt to many different workflows and

environments.

• Ansible can be used to manage many different types of systems, including servers running

Linux, Microsoft Windows, or UNIX, and network devices.

• Ansible Playbooks are human-readable text files that describe the desired state of an IT

infrastructure.

• Ansible is built around an agentless architecture in which Ansible is installed on a control node

and clients do not need any special agent software.

• Ansible connects to managed hosts using standard network protocols such as SSH, and runs

code or commands on the managed hosts to ensure that they are in the state specified by

Ansible

Microsoft Windows-based Managed Hosts

Ansible includes a number of modules that are specifically designed for Microsoft Windows

systems. These are listed in the Windows Modules [https://docs.ansible.com/ansible/latest/

modules/list\_of\_windows\_modules.html] section of the Ansible module index.

Most of the modules specifically designed for Microsoft Windows managed hosts require

PowerShell 3.0 or later on the managed host rather than Python. In addition, the managed hosts

need to have PowerShell remoting configured. Ansible also requires at least .NET Framework 4.0

or later to be installed on Windows managed hosts.

Managed Network Devices

You can also use Ansible automation to configure managed network devices such as routers and

switches. Ansible includes a large number of modules specifically designed for this purpose. This

includes support for Cisco IOS, IOS XR, and NX-OS; Juniper Junos; Arista EOS; and VyOS-based

networking devices, among others.

You can write Ansible Playbooks for network devices using the same basic techniques that you

use when writing playbooks for servers. Because most network devices cannot run Python,

Ansible runs network modules on the control node, not on the managed hosts. Special connection

methods are also used to communicate with network devices, typically using either CLI over SSH,

XML over SSH, or API over HTTP(S).

A static inventory file is a text file that specifies the managed hosts that Ansible targets. You can write this file using a number of different formats, including INI-style or YAML

Ex.1

web1.example.com

web2.example.com

db1.example.com

db2.example.com

192.0.2.4

EX.2

[webservers]

web1.example.com

web2.example.com

192.0.2.42

[db-servers]

db1.example.com

db2.example.com

[east-datacenter]

web1.example.com

db1.example.com

[west-datacenter]

web2.example.com

db2.example.com

[production]

web1.example.com

web2.example.com

db1.example.com

db2.example.com

[development]

192.0.2.42

Ex.3

[usa]

washington1.example.com

washington2.example.com

[canada]

ontario01.example.com

ontario02.example.com

[north-america:children]

canada

usa

Ex-4

192.168.[4:7].[0:255]

matches all IPv4 addresses in the 192.168.4.0/22 network (192.168.4.0 through 192.168.7.255)

server[01:20].example.com

matches all hosts named server01.example.com through server20.example.com

#ansible canada --list-hosts

#ansible development -i inventory

#ansible ungrouped -i inventory

CONFIGURING ANSIBLE

Using /etc/ansible/ansible.cfg

The ansible package provides a base configuration file located at /etc/ansible/ansible.cfg.

This file is used if no other configuration file is found.

Using ~/.ansible.cfg

Ansible looks for a .ansible.cfg file in the user's home directory. This configuration is used

instead of the /etc/ansible/ansible.cfg if it exists and if there is no ansible.cfg file in

the current working directory.

Using ./ansible.cfg

If an ansible.cfg file exists in the directory in which the ansible command is executed, it is

used instead of the global file or the user's personal file.

Ansible Configuration

inventory = Specifies the path to the inventory file.

remote\_user = The name of the user to log in as on the managed hosts. If

not specified, the current user's name is used.

ask\_pass = Whether or not to prompt for an SSH password. Can be

false if using SSH public key authentication.

become = Whether to automatically switch user on the managed

host (typically to root) after connecting. This can also be

specified by a play.

become\_method = How to switch user (typically sudo, which is the default,

but su is an option).

become\_user = The user to switch to on the managed host (typically

root, which is the default).

become\_ask\_pass = Whether to prompt for a password for your

An ad hoc command is a way of executing a single Ansible task quickly, one that you do not need

to save to run again later. They are simple, online operations that can be run without writing a

playbook.

Ad hoc commands are useful for quick tests and changes.

ansible host-pattern -m module [-a 'module arguments'] [-i inventory]

Any system upon which Ansible is installed and which has access to the required configuration

files and playbooks to manage remote systems (managed hosts) is called a control node.

• Managed hosts are defined in the inventory. Host patterns are used to reference managed hosts

defined in an inventory.

• Inventories can be static files or dynamically generated by a program from an external source,

such as a directory service or cloud management system.

• Ansible looks for its configuration file in a number of places in order of precedence. The first

configuration file found is used; all others are ignored.

• The ansible command is used to perform ad hoc commands on managed hosts.

• Ad hoc commands determine the operation to perform through the use of modules and their

arguments, and can make use of Ansible's privilege escalation features

PLAYBOOKS

A play is an ordered set of tasks run against hosts selected from your inventory. A playbook is a

text file containing a list of one or more plays to run in a specific order.

Plays allow you to change a lengthy, complex set of manual administrative tasks into an easily

repeatable routine with predictable and successful outcomes. In a playbook, you can save the

sequence of tasks in a play into a human-readable and immediately runnable form.

A playbook is a text file written in YAML format, and is normally saved with the extension yml. The playbook uses indentation with space characters to indicate the structure of its data

Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation. • Items that are children of another item must be indented more than their parents.

name: just an example

hosts: webservers

tasks:

- first

- second

- third

For example, the following task using the shell module is not idempotent. Every

time the play is run, it rewrites /etc/resolv.conf even if it already consists of the

line nameserver 192.0.2.1.

- name: Non-idempotent approach with shell module

shell: echo "nameserver 192.0.2.1" > /etc/resolv.conf

The following example does not rewrite the /etc/resolv.conf file if it already

consists of the correct content:

- name: Idempotent approach with copy module

copy:

dest: /etc/resolv.conf

content: "nameserver 192.0.2.1\n"

For example, to open the HTTP service on port 80, use it like this:

 The **immediate** and **permanent** parameters tell the module to apply the rules immediately and add them to firewalld's permanent rules to persist on reboot.

- name: Ensure port 80 (http) is open

firewalld:

service: http

state: enabled

permanent: yes

immediate: yes

You can also specify custom ports instead of service names with the port parameter. In this case, make sure to specify the protocol as well. For example, to open TCP port 3000, use this:

- name: Ensure port 3000/TCP is open

firewalld:

port: 3000/tcp

state: enabled

permanent: yes

immediate: yes

For instance, use the file module to create a directory /app owned by the user ricardo, with read, write, and execute permissions for the owner and the group users:

- name: Ensure directory **/**app exists  
  file:  
    path: **/**app  
    state: directory  
    owner: ricardo  
    group: **users**  
    mode: 0770

You can also use this module to change an existing line by applying the parameter regexp to look for an existing line to replace. For example, update the sshd\_config file to prevent root login by modifying the line PermitRootLogin yes to PermitRootLogin no:

- name: Ensure root cannot **login** via **ssh**  
  lineinfile:  
    path: **/**etc**/**ssh**/**sshd\_config  
    regexp: '^PermitRootLogin'  
    line: PermitRootLogin no  
   state: present

se the [unarchive](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/unarchive_module.html) module to extract the contents of archive files such as tar or zip files. By default, it copies the archive file from the control node to the target machine before extracting it. Change this behavior by providing the parameter remote\_src: yes.

For example, extract the contents of a .tar.gz file that has already been downloaded to the target host with this syntax:

- name: Extract contents of app.tar.gz  
  unarchive:  
    src: **/**tmp**/**app.tar.gz  
    dest: **/**app  
    remote\_src: **yes**

For instance, use the file module to create a directory /app owned by the user ricardo, with read, write, and execute permissions for the owner and the group users:

- name: Ensure directory /app exists

file:

path: /app

state: directory

owner: ricardo

group: users

mode: 0770

Using the service module to ensure that several network services are enabled to start at boot:

tasks:

- name: web server is enabled

service:

name: httpd

enabled: true

- name: NTP server is enabled

service:

name: chronyd

enabled: true

- name: Postfix is enabled

service:

name: postfix

enabled: true

When you run the playbook, output is generated to show the play and tasks being executed. The output also reports the results of each task executed.

The Gathering Facts task is a special task that the setup module usually runs automatically at the start of a play.

For playbooks with multiple plays and tasks, setting name attributes makes it easier to monitor the progress of a playbook's execution

In general, tasks in Ansible Playbooks are idempotent, and it is safe to run a playbook multiple times. If the targeted managed hosts are already in the correct state, no changes should be made.

The ansible-playbook -v command provides additional information, with up to four total levels.

The ansible-playbook command offers a --syntax-check option that you can use to verify the syntax of a playbook.

You can use the -C option to perform a dry run of the playbook execution. This causes Ansible to report what changes would have occurred if the playbook were executed, but does not make any actual changes to managed hosts.

A playbook is a YAML file containing a list of one or more plays. Remember that a single play is an ordered list of tasks to execute against hosts selected from the inventory. Therefore, if a playbook contains multiple plays, each play may apply its tasks to a separate set of hosts. This can be very useful when orchestrating a complex deployment which may involve different tasks on different hosts. You can write a playbook that runs one play against one set of hosts, and when that finishes runs another play against another set of hosts.

YAML Comments

Comments can also be used to aid readability. In YAML, everything to the right of the number or

hash symbol (#) is a comment. If there is content to the left of the comment, precede the number

symbol with a space.

# This is a YAML comment

YAML Strings

Strings in YAML do not normally need to be put in quotation marks even if there are spaces

contained in the string. You can enclose strings in either double quotes or single quotes.

this is a string

'this is another string'

"this is yet another a string"

YAML Dictionaries

You have seen collections of key-value pairs written as an indented block, as follows:

name: svcrole

svcservice: httpd

svcport: 80

Dictionaries can also be written in an inline block format enclosed in curly braces, as follows:

{name: svcrole, svcservice: httpd, svcport: 80}

YAML Lists

You have also seen lists written with the normal single-dash syntax:

hosts:

- servera

- serverb

- serverc

Lists also have an inline format enclosed in square braces, as follows:

hosts: [servera, serverb, serverc

Ansible facts are variables that are automatically discovered by Ansible on a managed host. Facts

contain host-specific information that can be used just like regular variables in plays, conditionals,

loops, or any other statement that depends on a value collected from a managed host.

Some of the facts gathered for a managed host might include:

• The host name

• The kernel version

• The network interfaces

• The IP addresses

• The version of the operating system

• Various environment variables

• The number of CPUs

• The available or free memory

• The available disk space

Facts are a convenient way to retrieve the state of a managed host and to determine what action

to take based on that state. For example:

• A server can be restarted by a conditional task which is run based on a fact containing the

managed host's current kernel version.

• The MySQL configuration file can be customized depending on the available memory reported

by a fact.

• The IPv4 address used in a configuration file can be set based on the value of a fact.