

Ansible

Network Automation cheat sheet

Jose Lima home lab notes

required files and directories

- Files and directories needed for successful ansible-playbook run
 - **ansible.cfg** = contains basic parameters for ansible to run
 - **inventory.cfg** = contains list of devices e.g. routers, switches, firewalls etc. which can be grouped together by "group" name. dc1east, all_datacenters etc. the default inventory file name can be changed in ansible.cfg.
 - **group_vars** = "directory" which contains group variables organized by group name. e.g. all.yml contains variables for "all groups" defined inside inventory.cfg

ansible.cfg

```
[defaults]
# do not check ssh keys
host_key_checking = False

# we use local for network devices
localhost ansible_connection=local

# human readable output
stdout_callback=debug
stderr_callback=debug

# default inventory file
inventory = inventory.cfg
```

- we dont change settings here often, if at all...

inventory.cfg

```
[all:vars]
ansible_user = "jason.bourne" # use quotes if you have special characters "!#$%&*@."
ansible_ssh_pass = "Pa$$w0rd!" # use quotes if you have special characters "!#$%&*@."

[dc1east]
dc1east_rt1 ansible_host=192.168.91.110 # not using DNS but need a hostname
192.168.91.111 # using just the IP of the host
dc1east_rt3 # using DNS, just use the hostname

[dc1east:vars]
# specific connection variables for dc1east group
ansible_connection = network_cli
ansible_network_os = ios # Cisco ios connection driver
ansible_become = yes # not needed for nxos (Nexus)
ansible_become_method = enable # not needed for nxos (Nexus)
ansible_become_pass = "gns3" # not needed for nxos (Nexus)
# Cisco nexus connection driver, only one driver per group
# ansible_network_os = nxos
```

- make sure to adjust settings for ios and nxos (nexus) devices, `ansible_network_os = nxos`

group_vars (directory)

```
group_vars/  
├── all.yml  
├── dc1east.yml  
└── dc2west.yml
```

- Groups defined in inventory.cfg can contain specific variables available to that group on run time.
- all.yml contains variables available to all groups. dc1east.yml has variables specific to dc1east group name as defined in inventory.cfg
- group_vars directory must reside in the same directory where the playbook file runs.

group_vars/dc1east.yml

global_servers:

 dns_servers:

 - ip name-server 8.8.8.8

 - ip name-server 4.2.2.2

wan_interfaces:

 - interface Gigabit0/0

 - interface Gigabit1/0

Contents of group_vars/dc1east.yml file



This is how you use the variables in your playbooks



this is how we access the "dictionary" global_servers, and its first "key"

dns_servers which contains a "list" of dns servers

{{ global_servers.dns_servers }} # this will give us the entire list, useful to loop through

{{ global_servers.dns_servers[0] }} # this will give us only the "first element"

{{ global_servers.dns_servers[1] }} # this will give us only the "second element"

wan_interfaces is "NOT" inside global_servers, so we can just access this list

directly

{{ wan_interfaces }} # this returns the whole list

{{ wan_interfaces[0] }} # this returns the first element, "interface Gigabit0/0"

{{ wan_interfaces[1] }} # this returns the first element, "interface Gigabit1/0"

Ansible lists []

"python list"

lists start with [] brackets, all rules that apply to "python" apply to ansible
ntp_servers = ["10.1.1.1", "192.168.1.10", "8.8.8.8"]

list with 3 items, indexed: 0, 1, 2



"yaml list"

each element in a list starts with a dash "-"

```
ntp_servers: # list name
  - 10.1.1.1 # list item 0
  - 192.168.1.10 # list item 1
  - 8.8.8.8 # list item 2
```

list ntp_servers has 3 items, indexed 0, 1, 2

list items in python are accessed the same way in ansible

in a playbook

```
{{ ntp_servers }} # returns the entire list with all 3 items
{{ ntp_servers[0] }} # returns list index item 0, 10.1.1.1 (without quotes)
{{ ntp_servers[1] }} # returns list index item 1, 192.168.1.10 (without quotes)
{{ ntp_servers[2] }} # returns list index item 2, 8.8.8.8 (without quotes)
```

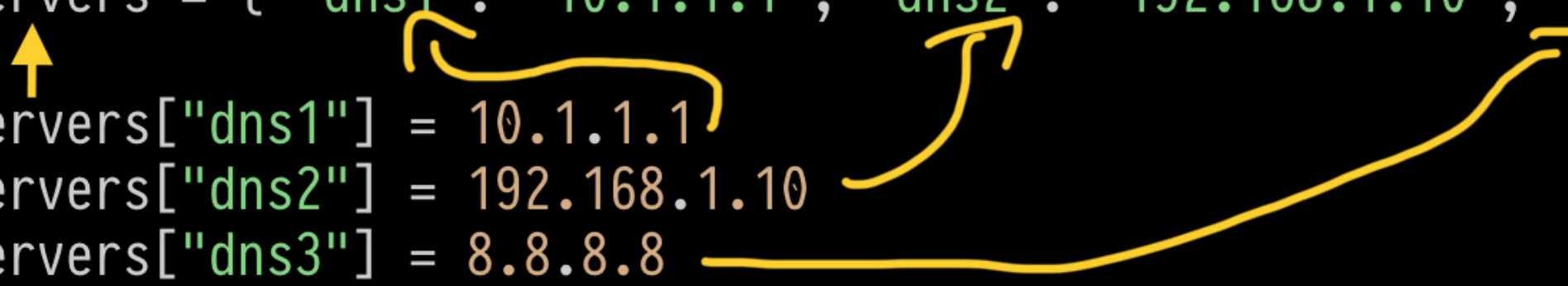
Ansible dictionary {}

"python dictionary"

dictionaries start with {} brackets
dictionaries contain key, value pairs, unlike lists, we access "values" by
calling their "keys", keys are on the left, values on the right

```
dns_servers = { "dns1": "10.1.1.1", "dns2": "192.168.1.10", "dns3": "8.8.8.8" }
```

```
dns_servers["dns1"] = 10.1.1.1  
dns_servers["dns2"] = 192.168.1.10  
dns_servers["dns3"] = 8.8.8.8
```

Three yellow arrows originate from the dictionary definition. The first arrow points from the 'dns1' key in the dictionary to the 'dns1' key in the first access statement. The second arrow points from the 'dns2' key in the dictionary to the 'dns2' key in the second access statement. The third arrow points from the 'dns3' key in the dictionary to the 'dns3' key in the third access statement.

"yaml dictionary"

```
dns_servers: # dictionary name  
  dns1: 10.1.1.1 # key named dns1, its value is 10.1.1.1  
  dns2: 192.168.1.10 # key named dns2, its value is 192.168.1.10  
  dns3: 8.8.8.8 # key named dns3, its value is 8.8.8.8
```

accessing key value pairs in a playbook using "." dot notation

```
{{ dns_servers.dns2 }} # = 192.168.1.10  
{{ dns_servers.dns3 }} # = 8.8.8.8
```

another way to access the same keys/values

```
{{ dns_servers["dns1"] }} # = 10.1.1.1  
{{ dns_servers["dns2"] }} # = 192.168.1.10  
{{ dns_servers["dns3"] }} # = 8.8.8.8
```

anything inside jinja2 {{ }} is substituted for the value of the variable
{{ dns_server.dns1 }} becomes simply: 10.1.1.1

Ansible "nested" dictionary { { } }

```
"python nested dictionary"  
#contains a dictionary inside a dictionary  
servers = { "dns1": "10.1.1.1", "ntp_servers": { "ntp1" : "4.2.2.2", "ntp2" : "9.9.9.9" } }
```

server dictionary keys

ntp_server key, has a dictionary as its value, its keys
are ntp1 with a value of 4.2.2.2
ntp2 with a value of 9.9.9.9

```
"ansible yaml nested dictionary"
```

```
servers:  
  dns1: 10.1.1.1  
  ntp_servers:  
    ntp1: 4.2.2.2  
    ntp2: 9.9.9.9
```

```
# calling nested dictionaries in playbooks
```

```
{{ servers.dns1 }} # = 10.1.1.1  
{{ servers.ntp_servers }} # = { "ntp1" : "4.2.2.2", "ntp2" : "9.9.9.9" }  
{{ servers.ntp_servers.ntp1 }} # = 4.2.2.2  
{{ servers.ntp_servers.ntp2 }} # = 9.9.9.9
```

```
# another way to get the same keys/values
```

```
{{ servers["dns1"] }} # = 10.1.1.1  
{{ servers["ntp_servers"] }} # = { "ntp1" : "4.2.2.2", "ntp2" : "9.9.9.9" }  
{{ servers["ntp_servers"]["ntp1"] }} # = 4.2.2.2  
{{ servers["ntp_servers"]["ntp2"] }} # = 9.9.9.9
```

Ansible "nested" dictionary with lists { [] }

"python nested dictionary with a list"

contains a list inside a dictionary

```
servers = { "dns1": "10.1.1.1", "ntp_servers": [ "4.2.2.2", "9.9.9.9" ] }
```

server dictionary keys

ntp_server key, has a list as its value with 2
index, 0 and 1. [0] = 4.2.2.2, [1] = 9.9.9.9



"ansible yaml nested dictionary"

servers: # dictionary name

dns1: 10.1.1.1 # key "dns1"

ntp_servers: # key "ntp_servers"

- 4.2.2.2 # ntp_servers key has a list, item [0] is 4.2.2.2

- 9.9.9.9 # ntp_servers key has a list, item [1] is 9.9.9.9

calling nested dictionaries in playbooks

{{ servers.dns1 }} # = 10.1.1.1

{{ servers.ntp_servers }} # = ["4.2.2.2", "9.9.9.9"] returns the whole list

{{ servers.ntp_servers[0] }} # = 4.2.2.2

{{ servers.ntp_servers[1] }} # = 9.9.9.9

another way to get the same keys/values

{{ servers["dns1"] }} # = 10.1.1.1

{{ servers["ntp_servers"] }} # ["4.2.2.2", "9.9.9.9"] returns the whole list

{{ servers["ntp_servers"][0] }} # = 4.2.2.2

{{ servers["ntp_servers"][1] }} # = 9.9.9.9