Automation

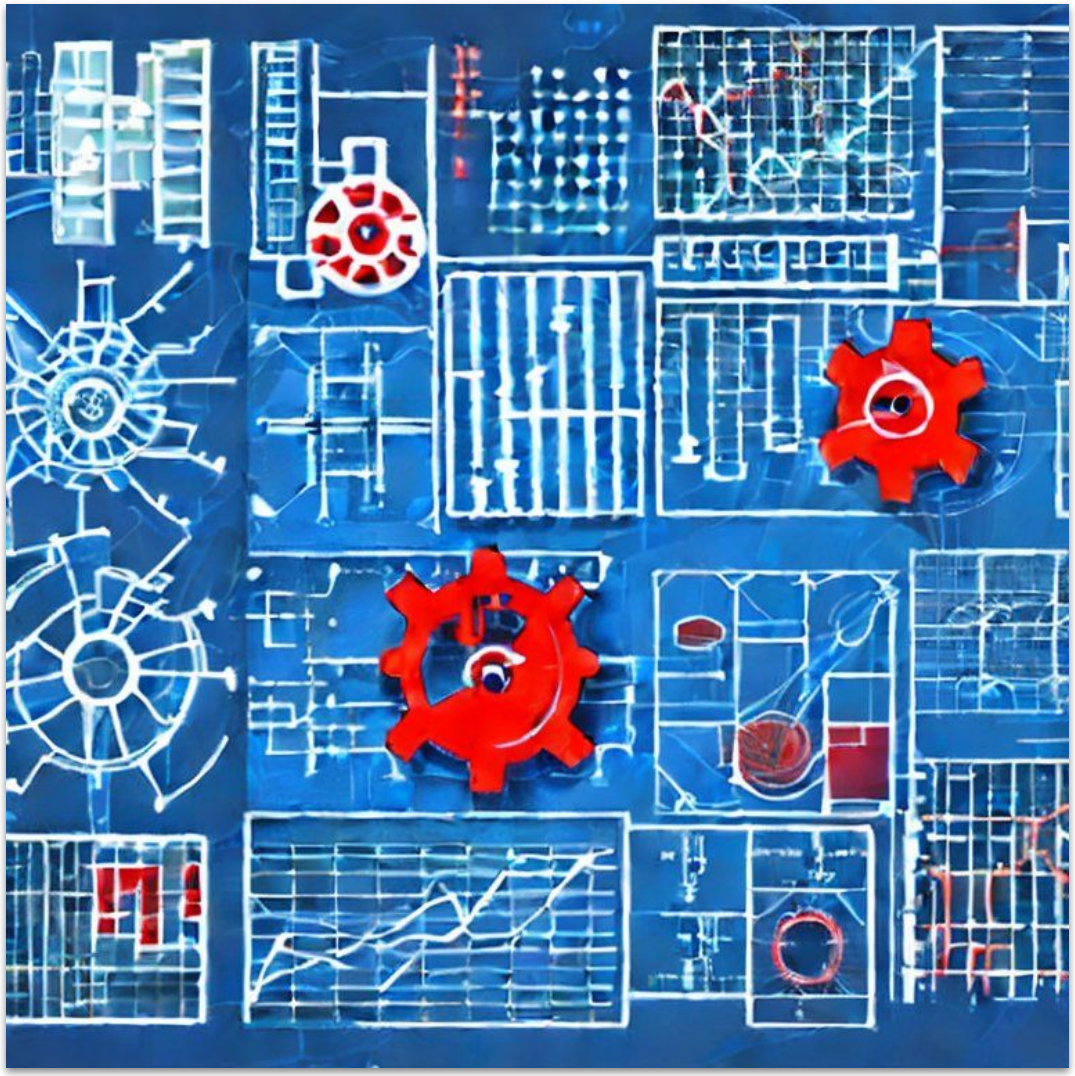
Ansible Variables

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Ansible Variables

- There are many different ways to define variables for use in tasks, playbooks, roles and inventories.
- Variables can be included inline.
- You can also pass additional variables using quoted JSON, YAML, or even by passing JSON or YAML files.
- Variables can be passed through the CLI.



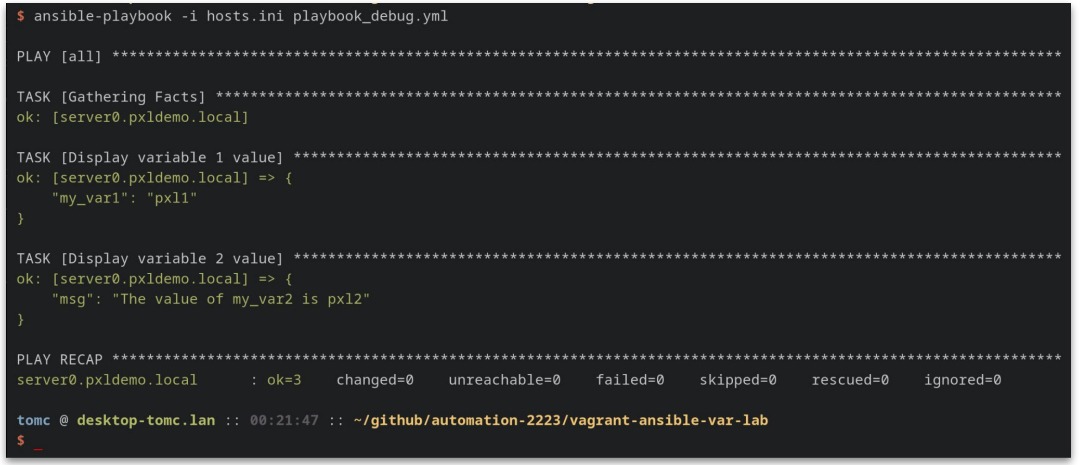
Ansible debug mode

- Display the value of variables and other information during playbook execution.
- Display the output of commands or modules.
- Use the `-vvv` option to enable verbose mode, which displays more detailed information about the execution of the playbook.
- `var`: display variable or expression
- `msg`: display string

```
tasks:

 - name: Display variable 1 value
   debug:
     var: my_var1

 - name: Display variable 2 value
   debug:
     msg: "The value of my_var2 is {{ my_var2 }}"
```



Variables in Ansible - types

- **Inline variables:** specific to a particular task.
- **Playbook Variables**: specific to a particular playbook.
- **Inventory Variables**: specific to a particular host or group of hosts.
- **Facts**: Ansible collects information about managed systems and stores it as facts. Facts are used to populate variables in playbooks and templates, allowing users to retrieve system information such as IP addresses, hostnames, or operating system details.
    - **(Ansible) Environment Variables**: store information about the environment in which the playbook is executed. They can be accessed within playbooks and templates and used to customize the automation process.

Inline variables

Playbook Variables

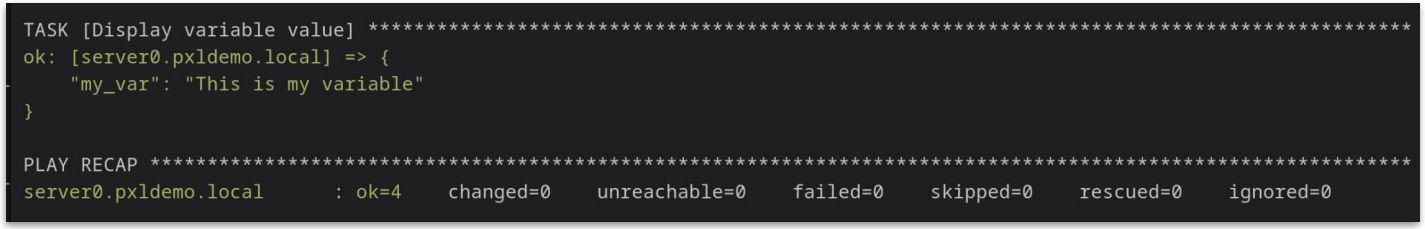
Inventory Variables

Facts

Defining variables in Ansible: inline with `vars`

- Define the variable inline with the task or playbook statement using the `vars` keyword inside the task.
- Only available within the scope of the task that defines it, and it cannot be accessed by other tasks or plays.
- E.g., in the following task, the variable "my_var" is defined inline with the "debug" statement:

```
- name: My playbook
  hosts: localhost
  tasks:
    - name: Display variable value
      vars:
        my_var: "This is my variable"
      debug:
        var: my_var
```



Defining variables in Ansible: inline with `register`

- The **`register`** keyword allows a task to store the result of a command or module in a variable that can be accessed by other tasks or plays.
- It changes the scope of the variable to global
- Eg, in the following playbook, the "shell" module is used to run a command and store the result in a variable named "my_var". This variable is then passed to the "debug" task using the "msg" keyword:

```
- name: My playbook
  hosts: localhost
  tasks:
    - name: Run command and store output in variable
      shell: echo "This is my output variable"
      register: my_output_var

    - name: Display output variable value
      debug:
        msg: "{{ my_output_var.stdout }}"
```

```
TASK [Display output variable value] ***********
ok: [server0.pxldemo.local] => {
    "msg": "This is my output variable"
}

PLAY RECAP *************************************
server0.pxldemo.local      : ok=6    changed=1
```

Inline variables

Playbook Variables

Inventory Variables

Facts

Defining variables in Ansible: playbook with `vars`

- Playbook-level variables are defined at the top of the playbook file using the `vars` keyword.
- E.g.:

```
---
- name: My playbook
  vars:
    my_playbook_var: "This is my playbook variable"
  tasks:
    - name: Display playbook variable value
      debug:
        var: my_playbook_var
```

```
TASK [Display playbook variable value] ***************
ok: [server0.pxldemo.local] => {
    "my_playbook_var": "This is my playbook variable"
}

PLAY RECAP *******************************************
server0.pxldemo.local      : ok=7    changed=1    unre
```

Defining variables in Ansible: variable files with `vars_files`

- Variables can be defined in separate files and included in playbooks using the `vars_files` keyword.
- Make sure to put the variable file in the same directory as the playbook.
- E.g., if we have a file named "my_vars.yml" containing the variable "my_var", we can include it in a playbook like this:

```yaml
---
- name: My playbook
  vars_files:
    - my_vars.yml
  tasks:
    - name: Display variable value
      debug:
        var: my_file_var
```

my_vars.yml

```yaml
my_file_var: "This is my file variable."
```

```
TASK [Display file variable value] ************
ok: [server0.pxldemo.local] => {
    "my_file_var": "This is my file variable."
}

PLAY RECAP ************************************
server0.pxldemo.local      : ok=8    changed=1
```

Variable precedence in Ansible

1. `--extra-vars` passed in via the command line (these always win, no matter what).
2. Task-level vars (in a task block).
3. Block-level vars (for all tasks in a block).
4. Role vars (e.g. [role]/vars/main.yml) and vars from include_vars module.
5. Vars set via set_facts modules.
6. Vars set via register in a task.
7. Individual play-level vars: 1. vars_files 2. vars_prompt 3. vars
8. Host facts.
9. Playbook host_vars.
10. Playbook group_vars.
11. Inventory: 1. host_vars 2. group_vars 3. vars
12. Role default vars (e.g. [role]/defaults/main.yml).

Ansible variable types

- **Strings**: A sequence of characters enclosed in quotes (single or double).
- **Numbers**: Integers or floats.
- **Lists**: A collection of ordered items, enclosed in square brackets and separated by commas.
- **Dictionaries**: A collection of key-value pairs, enclosed in curly braces and separated by commas.
- **Booleans**: True or False values.
- **None**: A null value.

Ansible Variable Interpolation

- Ansible uses variable interpolation to replace variable references with their values at runtime.
- Types of interpolation
  - Variable substitution: variables are replaced with their values at runtime
  - Jinja2 templating: allows use of filters, loops, conditionals, ...

Ansible Variable Interpolation: Variable substitution

- Most common type of variable interpolation, where variables are replaced with their values at runtime.
- Variables can be referenced using the double curly braces notation: `"{{ variable_name }}"`

```
---
- name: My playbook
  vars:
    my_var: "This is my variable"
  tasks:
    - name: Display variable value
      debug:
        var: my_var
```

```
    - name: Create user account
      vars:
        user_name: "my_user"
        user_password: "my_password"
      user:
        name: "{{ user_name }}"
        password: "{{ user_password }}"
```

Ansible Variable Interpolation: Jinja2 templating

- Ansible uses the Jinja2 templating engine to enable more advanced variable interpolation.
- Jinja2 allows users to use filters, loops, conditionals, and other programming constructs to manipulate and generate content based on variable values.

Ansible Variable Interpolation: Jinja2 templating

- In this example, the "my_template.j2" template file contains Jinja2 code that references the "my_var" variable and generates content based on its value.

```
- name: Generate file from template
  template:
    src: my_template.j2
    dest: /path/to/my/file.txt
  vars:
    my_var: "This is my variable"
```

my_template.j2

```
My variable is {{ my_var }}
```

/path/to/my/file.txt

```
My variable is This is my variable
```

Inline variables

Playbook Variables

Inventory Variables

Facts

Defining variables in Ansible: inventory variables with `vars`

- Inventory variables are defined within the inventory file using the `vars` keyword.
- Inventory files can also include variables that are specific to individual hosts or groups.
- E.g., the "http_port" and "max_clients" variables are defined within the "webservers" group and are specific to that group of hosts.

```
[webservers]
webserver1 ansible_host=192.168.1.10

[databases]
db1 ansible_host=192.168.1.11

[webservers:vars]
http_port=80
max_clients=200
```

How to use inventory variables in Ansible

- Inventory variables can be referenced in playbooks, templates, and other Ansible files using the double curly braces notation: "`{{ variable_name }}`".
- Inventory variables can be used to provide host-specific configuration details to playbooks and tasks.
- E.g., in the following playbook, the "http_port" variable is used to configure the Apache web server running on the "webserver1" host.

hosts.ini

```
[webservers]
webserver1 ansible_host=192.168.1.10

[databases]
db1 ansible_host=192.168.1.11

[webservers:vars]
http_port=80
max_clients=200
```

```
- name: Install and configure Apache web server
  hosts: webservers
  tasks:
    - name: Install Apache web server
      yum:
        name: httpd
        state: present
    - name: Configure Apache web server
      template:
        src: apache.conf.j2
        dest: /etc/httpd/conf/httpd.conf
      vars:
        http_port: "{{ http_port }}"
```

Ansible inventory variables `group_vars/`

- Group variables are defined at the group level and apply to all hosts within that group.
- Group variables are defined in a file named `group_vars/<group_name>.yml` within the same directory as the inventory file.
- Group variables are inherited by child groups and can be overridden at the host level.
- E.g., if we have a group named "webservers", we can define group variables in a file named `group_vars/webservers.yml`.

Ansible inventory variables `host_vars/`

- Host variables are defined at the host level and apply only to the specified host. Can be used to define values that are specific to a particular host, such as IP address or hostname.
- Host variables are defined in a file named `host_vars/<host_name>.yml` within the same directory as the inventory file.
- Host variables take precedence over group variables and can be used to override values defined at the group level.
- E.g., if we have a group named "webserver1", we can define host variables in a file named `host_vars/webserver1.yml`.

Ansible inventory variables: host_vars and group_vars example

In this example, the "{{ http_port }}" variable is a group variable that is defined in "group_vars/webservers.yml", and the "{{ ip_address }}" variable is a host variable that is defined in "host_vars/webserver1.yml".

playbook.yml

```
- name: Configure web server
  hosts: webservers
  tasks:
    - name: Set firewall rule for HTTP
traffic
      firewalld:
        service: http
        state: enabled
        immediate: yes
        permanent: yes
        port: "{{ http_port }}"
        zone: public
      become: true
      become_user: root
    - name: Set IP address in config file
      template:
        src: my_config.j2
        dest: /etc/myapp/my_config.conf
        owner: root
        group: root
        mode: 0644
      vars:
        ip_address: "{{ ip_address }}"
```

group_vars/webservers.yml

```
---
http_port: 80
https_port: 443
max_connections: 100
```

host_vars/webserver1.yml

```
---
ip_address: 192.168.1.100
hostname: webserver1.example.com
```

my_config.j2

```
bind_address = {{ ip_address }}
port = {{ http_port }}
hostname = {{ hostname }}
```

Ansible Extra Variables through CLI

- Passed to Ansible on the CLI and used to override other variables or provide additional information.
- Extra variables are defined using the **`--extra-vars`** option on the command line.

Examples:

- `ansible-playbook playbook.yml --extra-vars "var1=value1" --extra-vars "var2=value2"`
- `ansible-playbook playbook.yml --extra-vars "@variables.yml"`
  - "@variables.yml": read the variable values from the file variables.yml

Inline variables

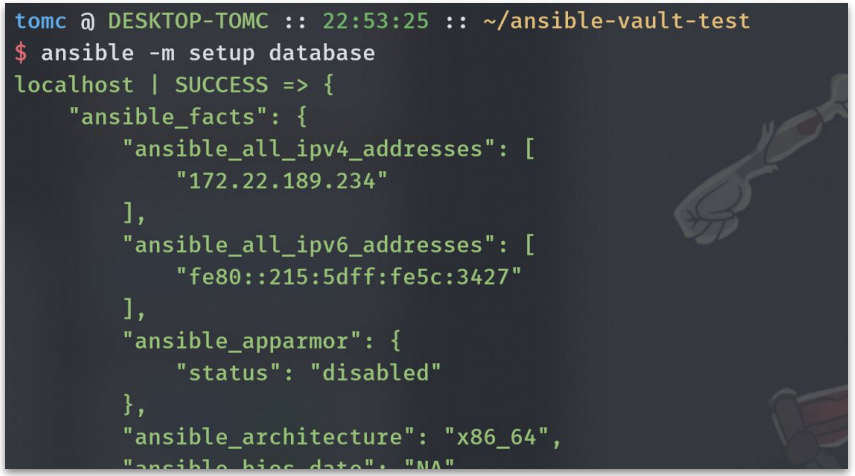
Playbook Variables

Inventory Variables

Facts

Ansible facts

- Pieces of information about the target hosts that are collected and stored by Ansible during playbook execution.
- Can include a wide range of system information, such as the host name, IP address, operating system version, available disk space, and installed packages.
- Often, playbooks use facts such as **`ansible_os_family`**, **`ansible_hostname`**, and **`ansible_memtotal_mb`** to register new variables or to determine whether certain tasks should be run.
- Ansible uses the **`setup`** module to collect and store facts about the target hosts.
  - executed automatically at the beginning of each playbook run, and the facts collected are stored in memory as a set of key-value pairs, into the **`ansible_facts`** variable.
  - can be called explicitly with the ansible CLI command
    - `ansible all -m setup`
    - `ansible --limit webserver -m setup`



Customize Ansible fact gathering with **gather_facts**

- By default, the **setup** module is executed automatically at the beginning of each playbook run to collect facts about the target hosts and store them into the **ansible_facts** variable.
- Use the **gather_facts** parameter to customize this behavior and control when and how facts are collected.
- The **gather_subset** parameter of **setup** selects which subsets of facts to collect.

Disable fact collection:

```
- hosts: webserver
  gather_facts: no
```

Customize fact collection options:

```
- hosts: webserver
  gather_facts: yes
  gather_subset: network
```

Ansible Built-in Variables

- **`ansible_host`**: hostname or IP address of the target machine.
- **`ansible_port`**: SSH port used to connect to the target machine.
- **`ansible_user`**: username used to connect to the target machine.
- **`ansible_connection`**: connection type used to connect to the target machine (e.g., SSH or local).
- **`ansible_python_interpreter`**: path to the Python interpreter used by Ansible on the target machine.
- **`ansible_distribution`**: name of the distribution installed on the target machine.
- **`ansible_distribution_version`**: version of the distribution installed on the target machine.
- **`ansible_os_family`**: family of the operating system installed on the target machine (e.g., Debian or RedHat).
- **`ansible_architecture`**: architecture of the target machine (e.g., x86_64 or arm).
- **`ansible_facts`**: dictionary of facts about the target machine, such as system information, network details, and hardware information.

Automatically collected by Ansible using specialized "setup" modules, and made available as variables in playbooks and templates.

Ansible Built-in Variables - examples

```
- name: Display hostname
  debug:
    msg: "The hostname is {{ ansible_host }}"
```

```
- name: Execute command on target machine
  shell: "my_command {{ my_option }} --host {{ ansible_host }}"
  vars:
    my_option: "my_value"
```

my_template.j2

```
The hostname is {{ ansible_host }}
```

Complex variables

- For larger and more structured arrays, you can access any part of the array, either using parentheses [' '] or dot
  - syntax.
- If you know the general structure of the variable, you can use one of the following techniques to retrieve just the server's IPv4 address:

```
{{ ansible_eth0.ipv4.address }}
{{ ansible_eth0['ipv4']['address'] }}
```

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars-4.yml

```yaml
---
  - name: Playbook
    hosts: database

    vars:
      foo: bar

    tasks:
      - name: prints eth0 variable
        debug:
          var: ansible_eth0
```

```
tomc @ DESKTOP-TOMC :: 20:43:15 :: ~/ansible-vault-test
$ ansible-playbook vars-4.yml

PLAY [Playbook] ************************************************************************************************************************

TASK [Gathering Facts] *****************************************************************************************************************
ok: [localhost]

TASK [prints eth0 variable] ************************************************************************************************************
ok: [localhost] => {
    "ansible_eth0": {
        "active": true,
        "device": "eth0",
        "ipv4": {
            "address": "172.22.189.234",
            "broadcast": "172.22.191.255",
            "netmask": "255.255.240.0",
            "network": "172.22.176.0",
            "prefix": "20"
        },
        "ipv6": [
            {
                "address": "fe80::215:5dff:fe5c:3427",
                "prefix": "64",
                "scope": "link"
            }
        ],
        "macaddress": "00:15:5d:5c:34:27",
        "module": "hv_netvsc",
        "mtu": 1500,
        "pciid": "b3370019-24ed-4275-afd8-fa442e15d3eb",
        "promisc": false,
        "speed": 10000,
        "type": "ether"
    }
}

PLAY RECAP *****************************************************************************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

tomc @ DESKTOP-TOMC :: 20:43:45 :: ~/ansible-vault-test
$ _
```

Access operating system environment variables with **`ansible_env`**

- The **`ansible_env`** variable is an automatically collected set of facts that includes operating system environment variables from the target hosts.

```
- name: Deploy web application
  hosts: webserver
  vars:
    api_key: "{{ ansible_env.MY_API_KEY }}"
  tasks:
    - name: Configure application
      template:
        src: my_config.j2
        dest: /var/www/myapp/config.ini
```

my_config.j2

```
[api]
key = {{ api_key }}
```

lists

- Many variables you will use are structured as arrays (or 'lists').

```
foo_list:
- one
- two
- three
```

- Access to list elements:
- `foo[0]`
- `foo|first`

Magic variables using host and group variables

- If you need to get the variables of a specific host from another host, Ansible provides a magic `hostvars` variable that contains all defined host variables (from inventory files and all discovered YAML files in host_vars directories).
  - ```
    # works from any host, returns "jane".
    {{ hostvars['host1']['admin_user'] }}
    ```
- `groups`: a list of all group names in the inventory
- `group_names`: a list of all groups to which the current host belongs
- `inventory_hostname`: the hostname of the current host, according to the inventory
- `inventory_hostname_short`: the first part of inventory_hostname, up to and including the first point
- `play_hosts`: all hosts on which the current play is running.
- see
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_vars_facts.html#information-about-ansible-magic-variables
    ```

Variables good practices

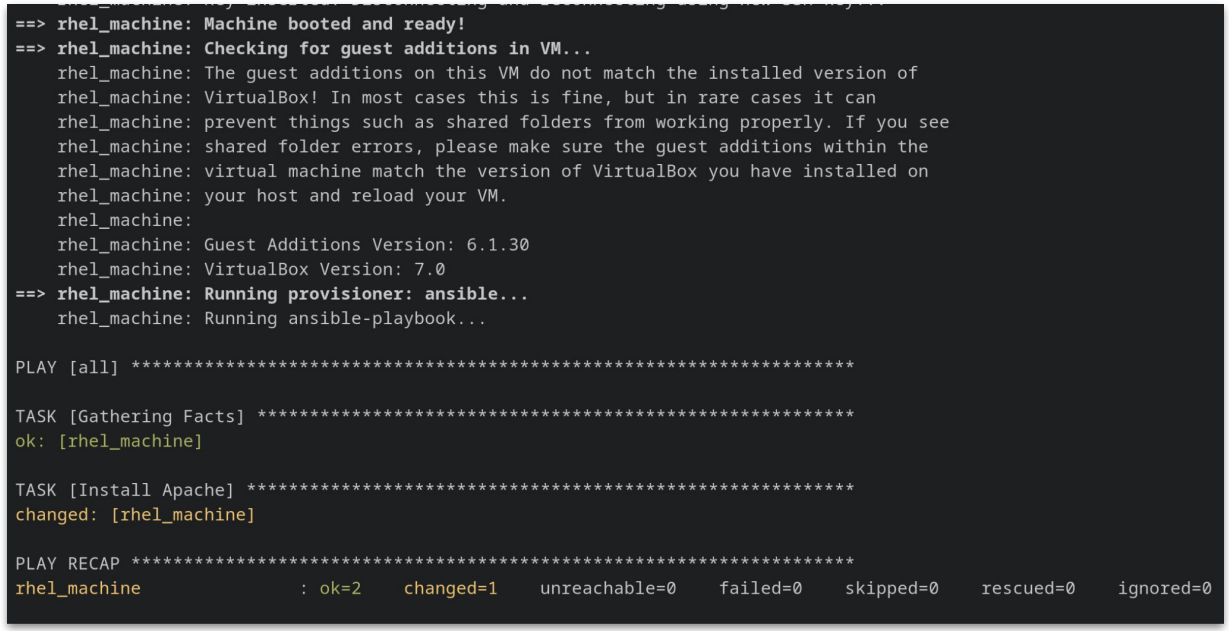
- Roles should provide sane default values via the role's 'defaults' variables. These variables will be the fallback in case the variable is not defined anywhere else in the chain.
- Playbooks should rarely define variables (e.g. via set_fact), but rather, **variables should be defined in included vars_files** or, less often, via inventory.
- Only truly host- or group-specific variables should be defined in host or group inventories.
- Dynamic and static inventory sources should contain a minimum of variables, especially as these variables are often less visible to those maintaining a particular playbook.
- Command line variables (-e) should be avoided when possible. One of the main use cases is when doing local testing or running one-off playbooks where you aren't worried about the maintainability or idempotence of the tasks you're running.

Exercise 1

- 2 RHEL/rocky linux machines.
- Create an inventory file with two groups of hosts: "webservers" and "dbservers".
- Write a playbook that uses variables to configure the firewall on the web servers to allow incoming traffic on port 80 and on the database servers to allow incoming traffic on port 3306.
- Use firewalld. Make sure it's installed, running and enabled.
- **Be efficient: no repeating tasks.**

Exercise 2

- 2 machines
  - Ubuntu
  - RHEL/rocky linux
- Install apache on both machines.
- Use 1 inventory file and 1 playbook containing only 1 task.



end

Advanced: conditional variable file import

- Variable files can also be conditionally imported.
- E.g. you have one set of variables for RHEL servers (where the Apache service is called **httpd**) and another for Debian servers (where the Apache service is called **apache2**). In this case, you can conditionally include variable files with include_vars.
- If you are following the example, add two files in the same directory as your playbook: **apache_RedHat.yml** and **apache_default.yml**. Define the variable **apache_service_name: httpd** in the CentOS file and **apache_service_name: apache2** in the default file.
- Ansible stores the OS of the server in the predefined **variable ansible_os_family**, and will include the vars file with the resulting name.
- If Ansible cannot find a file with that name, it will use the variables loaded from **apache_default.yml**, using **with_first_found**. So on a Debian or Ubuntu server, Ansible would correctly use apache2 as the service name, even if there is no apache_Debian.yml or apache_Ubuntu.yml file available.

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars-3.yml

```
1  ---
2  - hosts: example
3    pre_tasks:
4      - include_vars: "{{ item }}"
5        with_first_found:
6          - "apache_{{ ansible_os_family }}.yml"
7          - "apache_default.yml"
8
9    tasks:
10     - name: Ensure Apache is running.
11       service:
12         name: "{{ apache_service_name }}"
13         state: running
```