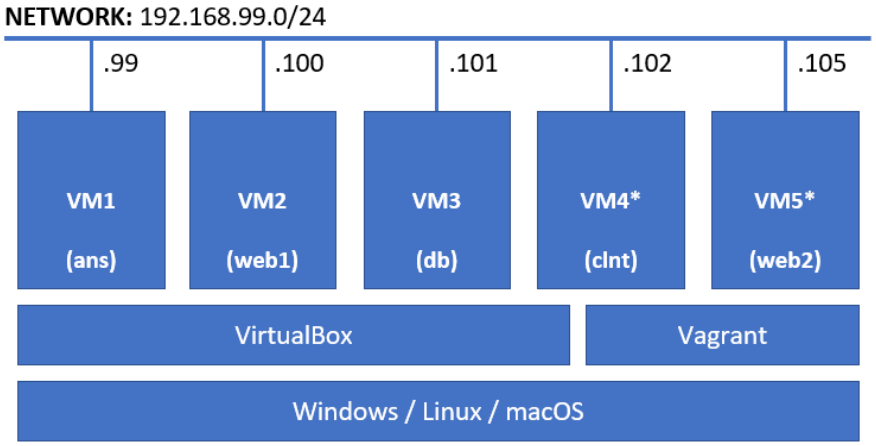
Practice M2: Ansible

For this practice, our lab environment will look like this



We are going to use mostly **CentOS Stream 9** boxes and at least one **Debian**-based box

All configurations and supplementary files are provided as a ZIP archive and can be downloaded from the module section in the official site

Part 1

Let us start with the environment

Set Up the Environment

We will build the first version of the environment by using just a part of the provided **Vagrantfile**

Once done exploring it, we can deploy the infrastructure with

```
vagrant up --no-provision ans web1 db clnt
```

Install Ansible

Now, that we have the infrastructure, we can continue with the installation

It is fairly simple, and it is usually a matter of installing a single package

Depending on your choice of distribution for the **Ansible** host, follow the appropriate installation instructions

First, establish a session to the **Ansible** host

```
vagrant ssh ans
```

Red Hat/CentOS

On **CentOS** and **Red Hat**, depending on the version, we may need to install the **EPEL repository** first

```
[sudo dnf install epel-release]
```

```
sudo dnf install ansible
```

Or, for 9.x, we can use the **AppStream** repository that is available by default and install it with

```
sudo dnf install ansible-core
```

Either will do the job for this module

SUSE/openSUSE

On **SUSE**/**openSUSE** we can run just this command

```
sudo zypper install ansible
```

Debian/Ubuntu

On **Debian**/**Ubuntu**, we must add additional repository

```
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

First Steps

Explore ansible configuration and version with

```
ansible --version
```

Just the configuration files, provided by the package, can be seen with (for RPM-based distributions)

```
rpm -qc ansible
```

Or, if you installed the core version, check with

```
rpm -qc ansible-core
```

Or with (for DEB-based distributions):

```
cat /var/lib/dpkg/info/ansible.conffiles
```

Now, that we know where the global inventory file (**/etc/ansible/hosts**) is, we can open it and enter the following

```
[srv]
192.168.99.100
192.168.99.101

[clnt]
192.168.99.102
```

Save and close the file

Being on the **ans** host, let us start experimenting with **Ansible** in an interactive way

With our first command we will ask a host for its hostname (not that we do not know it already)

```
ansible 192.168.99.100 -a "hostname"
```

Confirm with **yes** to add the host to the known hosts

No, our command fails because of the authentication method in use

Let us adjust it a little bit and try again

```
ansible 192.168.99.100 -a "hostname" -u vagrant
```

Same error, same result

Finally, we can extend to command to ask us for a password

```
ansible 192.168.99.100 -a "hostname" -u vagrant -k
```

The password of the **vagrant** user is **vagrant**

At last, we managed to successfully execute the command and got the host's name

Let us execute a command against a group of machines

For this purpose, we will use the information we entered in the **/etc/ansible/hosts** file

We can ask for the hostname of multiple machines, for example the **srv** group, with

```
ansible srv -a "hostname" -u vagrant -k
```

We received an answer only from the machine that we communicated with earlier

There are multiple ways to solve this situation, so we can choose one

We can scan and add the SSH public key of the hosts we are about to manage with

```
ssh-keyscan 192.168.99.101 >> ~/.ssh/known_hosts
```

Now, we can execute again

```
ansible srv -a "hostname" -u vagrant -k
```

It should work this time

Now, we can use **ssh-keygen** generate key and copy it to the other (**clnt**) station

Let's first generate it

```
ssh-keygen
```

And then copy it to the station with **ssh-copy-id**

```
ssh-copy-id 192.168.99.102
```

Perhaps there are other ways as well

We can omit the **-k** option with the **clnt** group and execute just

```
ansible clnt -a "hostname" -u vagrant
```

And even this, to address all hosts

```
ansible all -a "hostname" -u vagrant -k
```

Usually, commands are executed in parallel

By adding **-f 1** at the end, we can limit the number of simultaneous executions

```
ansible all -a "hostname" -u vagrant -k -f 1
```

Now, the stations are asked in order of their appearance in the inventory file

If you do not trust this statement, try to change their order, and repeat the command 😉

We can also copy the SSH key to the rest of the stations with

```
ssh-copy-id 192.168.99.100
```

```
ssh-copy-id 192.168.99.101
```

And now, the last command can be shortened to

```
ansible all -a "hostname" -f 1
```

Let us extend the range of information we receive from the hosts

For example, to receive information about the disk utilization, we can execute

```
ansible all -m command -a "df -h" -u vagrant -k
```

*If you copied the key to all the stations, then you can omit the last part (**-u vagrant -k**)*

We can use another module to accomplish the same

```
ansible all -m shell -a "df -h" -u vagrant -k
```

The result indeed appears to be the same but in fact the **command** module is processed directly while the **shell** is passed through a **shell**

To see the difference between those two more clearly, let us execute

```
ansible all -m command -a 'echo $HOSTNAME' -u vagrant -k
```

```
ansible all -m shell -a 'echo $HOSTNAME' -u vagrant -k
```

We should be careful which type of quotes we use. Let us change the last command to this

```
ansible all -m shell -a "echo $HOSTNAME" -u vagrant -k
```

Furthermore, the **command** module can be omitted (because the default module is set to **command**), as we did in the beginning

```
ansible all -a "df -h" -u vagrant -k
```

Let us continue exploring information, for example about the memory utilization, by executing

```
ansible all -a "free -m" -u vagrant -k
```

We can ask for date and time information

```
ansible all -a "date" -u vagrant -k
```

Also, we can execute locally prepared scripts on the remote hosts

Create a file **local_script.sh** with the following content

```
#!/bin/bash

echo 'My hostname is '$HOSTNAME
echo 'My IP addresses are '$(hostname -I)
```

Save and close the file

*You can use the provided file which can be found in **/vagrant/p1/local_script.sh***

We can send the script for execution on the hosts with

```
ansible srv -m script -a "local_script.sh" -u vagrant -k
```

This is enough for now 😉

Part 2

We continue with the same environment from ***Part 1***

Inventory files

We can work with local or per project inventory files

While still on the **Ansible** control host (**ans**), create a folder **~/p2/1** to work in and navigate to it

Let us start an empty file named **inventory**

Enter the following as its first line

```
web1 ansible_host=192.168.99.100 ansible_user=vagrant ansible_ssh_pass=vagrant
```

Save and close the file

Execute the following command

```
ansible web1 -i inventory -a "hostname"
```

Notice that we did not specify a username, nor we add the **-k** option

Open the file again and add the following at the end

```
[webservers]
web1
```

Save and close the file

Execute the following command

```
ansible webservers -i inventory -a "hostname"
```

Open the file again and add a second host

```
db ansible_host=192.168.99.101 ansible_user=vagrant ansible_ssh_pass=vagrant
```

And then a second group

```
[dbservers]
db
```

Save and close the files

Test what we did so far by executing this

```
ansible dbservers -i inventory -a "hostname"
```

Open the file again and add a third host

```
clnt ansible_host=192.168.99.102 ansible_user=vagrant ansible_ssh_pass=vagrant
```

And a third group for the host

```
[stations]
clnt
```

Finally add group for all servers

```
[servers:children]
webservers
dbservers
```

And variables for the group

```
[servers:vars]
ansible_user=vagrant
ansible_ssh_pass=vagrant
```

Remove the **ansible_user** and **ansible_ssh_pass** instructions from the first two lines

Save and close the file

The final **inventory** file should look like

```
web1 ansible_host=192.168.99.100
db ansible_host=192.168.99.101
clnt ansible_host=192.168.99.102 ansible_user=vagrant ansible_ssh_pass=vagrant

[webservers]
web1

[dbservers]
db

[stations]
clnt

[servers:children]
webservers
dbservers

[servers:vars]
ansible_user=vagrant
ansible_ssh_pass=vagrant
```

Execute the following command to test what we did so far

`ansible servers -i inventory -a "hostname"`

The above will return information for all servers

And this one will return information for all stations (currently, only one)

`ansible stations -i inventory -a "hostname"`

And this, for all machines from the inventory

`ansible all -i inventory -a "hostname"`

Variables

While on the Ansible control host, create a new work folder **~/p2/2** and navigate to it

Copy the **inventory** file from **~/p2/1**

Create two sub-folders **group_vars** and **host_vars**

`mkdir {group_vars,host_vars}`

Create **all** file in the **group_vars** folder

`vi group_vars/all`

With the following content

```
---
# Group level user
username: user_all
```

Save and close the file

Execute the following command

`ansible servers -i inventory -m user -a "name={{username}} password=Password1" --become`

This way, we created a user with username coming from the variable **username** stored in the **all** file

And the new user (**user_all**) will be created on the members of the **servers** group (**web1** and **db**)

Let us create a second file named after one of the groups - **webservers** in the **group_vars** folder

vi group_vars/webservers

Containing

```
---
# Group level user
username: user_group
```

Save and close the file

Execute the following command

ansible servers -i inventory -m user -a "name={{username}} password=Password1" --become

We can see that the new user **user_group** was created only on the hosts belonging to the **webservers** group

The change was applied only there because the name of the variable file matches the group's name

Create one more file named **web1** but this time in the **host_vars** folder

vi host_vars/web1

Make sure that its content is like this

```
---
# Host level user
username: user_host
```

Save and close the file

Execute the following command

ansible servers -i inventory -m user -a "name={{username}} password=Password1" --become

The user will be created only on that host part of the **servers** group which name is **web1** (as the name of the variable file)

Configurations

Create a new **~/p2/3** folder and navigate to it

Copy the **inventory** file from **~/p2/2**

Remove all lines for the three hosts from the **~/.ssh/known_hosts** file

Execute the following command

ansible clnt -i inventory -a "hostname" -u vagrant -k

It should return an error that the host is not part of the **known_hosts** file

We can create a local (project-based) configuration file named **ansible.cfg** with the following content

```
[defaults]
host_key_checking = false
```

Save and close the file

Now, execute again the command

`ansible clnt -i inventory -a "hostname" -u vagrant -k`

This time, it should succeed

Now check the **known_hosts** file

`cat ~/.ssh/known_hosts`

The record for the host should be added there automatically

This behavior can be controlled with environment variables

They take precedence over the configuration file

Let us create a variable

`export ANSIBLE_HOST_KEY_CHECKING=true`

Check that it was successfully created

`echo $ANSIBLE_HOST_KEY_CHECKING`

Execute the following command

`ansible db -i inventory -a "hostname" -u vagrant -k`

The command should fail, because the environment variable overwrote the configuration and turned on the validation against the **known_hosts** file

Let us unset the variable

`unset ANSIBLE_HOST_KEY_CHECKING`

And execute again the last command

`ansible db -i inventory -a "hostname" -u vagrant -k`

This time it must execute without any error

If we check again the **known_hosts** file

`cat ~/.ssh/known_hosts`

We will see that the **db** host was added there

Open the **ansible.cfg** file for editing and make sure that its current content is like this

```
[defaults]
host_key_checking = false
private_key_file = /home/vagrant/.ssh/id_rsa
ansible_user = vagrant
remote_user = vagrant
```

Save and close the file

For the above to work you should have used the pair commands **ssh-keygen** and **ssh-copy-id** in the previous part. If not, check and execute them

The second one (**ssh-copy-id**) must be executed against all three hosts

`ssh-copy-id 192.168.99.100`

`ssh-copy-id 192.168.99.101`

`ssh-copy-id 192.168.99.102`

Each one of them may return a warning if you copied the key in ***Part 1***

Open the **inventory** file and make it to look like this

```
web1 ansible_host=192.168.99.100
db ansible_host=192.168.99.101
clnt ansible_host=192.168.99.102

[webservers]
web1

[dbservers]
db

[stations]
clnt

[servers:children]
webservers
dbservers
```

Save and close the file

Execute the following command

`ansible web1 -i inventory -a "hostname"`

It should complete successfully

Modules

Create a new **~/p2/4** folder and navigate to it

Copy both the **inventory** and **ansible.cfg** files from **~/p2/3**

So far, we used a few modules but there are great many more

Modules list can be retrieved with

`ansible-doc -l`

The size of the list will vary based on the installed version

Let us ask for detailed information for the **dnf** module

`ansible-doc dnf`

There are similar modules for other packaging systems, check some of them

We can use the generic package management module named **package** instead

Should we want, we can ask for a sample module snippet with

`ansible-doc -s dnf`

Let us install some software on our hosts

Execute the following to install ***Apache HTTP*** on our **webservers** hosts

`ansible webservers -i inventory -m dnf -a "name=httpd state=present" --become`

Then the following to enable and start the service

```
ansible webservers -i inventory -m service -a "name=httpd state=started enabled=true" --become
```

Finally, we can test in a browser window on the host

Navigate to http://localhost:8080

Nothing opens

Let us check from the Ansible host with

curl http://192.168.99.100

Again, nothing

Let us open the appropriate firewall port on the web host with

```
ansible webservers -i inventory -m firewalld \
  -a "service=http state=enabled permanent=yes" --become
```

Depending on the installed version (if it is the core one), you may need to execute first this command

```
ansible-galaxy collection install ansible.posix
```

Check again with

curl http://192.168.99.100

Again, nothing opens. May we should adjust the command a little bit

```
ansible webservers -i inventory -m firewalld \
  -a "service=http state=enabled permanent=yes immediate=yes" --become
```

If check again, we should see the default web page

Following the same approach, we can install ***MariaDB*** on our **databases** hosts (currently, only one)

```
ansible dbservers -i inventory -m dnf -a "name=mariadb,mariadb-server state=present" --become
```

Then enable and start the service

```
ansible dbservers -i inventory -m service \
  -a "name=mariadb state=started enabled=true" --become
```

We can log in to the database host and test if everything is working as expected

Part 3

If you have not destroyed the environment from the previous two parts, do it now

```
vagrant destroy --force
```

Then, bring it back up again, but with this command

```
vagrant up ans web1 db clnt
```

Playbooks

Enter the **Ansible** host (**ans**)

```
vagrant ssh ans
```

Prepare the working folders and enter there

mkdir -p ~/p3/{1..3}

cd ~/p3/1

Make sure that you have an **inventory** file with the following content

```
web1 ansible_host=192.168.99.100
db ansible_host=192.168.99.101
clnt ansible_host=192.168.99.102

[webservers]
web1

[dbservers]
db

[stations]
clnt

[servers:children]
webservers
dbservers

[servers:vars]
ansible_user=vagrant
ansible_ssh_pass=vagrant
```

Save and close the file

Create an **ansible.cfg** file with the following content

```
[defaults]
host_key_checking = False
inventory = inventory
```

Save and close the file

Now, create a **playbook.yml** file with the following content

```
---
- hosts: webservers
  become: true

  tasks:
    - name: Install Apache HTTP Server
      dnf: name=httpd state=present

    - name: Start Apache HTTP Server and Enable it
      service: name=httpd state=started enabled=true

    - name: Allow HTTP service in the firewall
      firewalld: service=http state=enabled permanent=yes immediate=yes
```

Save and close the file

Before we attempt to execute it, let us check if it is correct

ansible-playbook playbook.yml --syntax-check

If there were not any errors, we can check which hosts will be affected with

ansible-playbook playbook.yml --list-hosts

And finally, we can execute it with

ansible-playbook playbook.yml

We can open the file again and add a second play by adding the following to the end

```
- hosts: dbservers
  become: true

  tasks:
    - name: Install MariaDB Server
      dnf: name=mariadb,mariadb-server state=present

    - name: Start and enable MariaDB
      service: name=mariadb state=started enabled=true
```

Save and close the file

Execute the play with

ansible-playbook playbook.yml

It should work just fine

Now, let us simulate an error in the communication with one of the hosts

For this to happen, we must change the **ansible.cfg** file to match this

```
[defaults]
host_key_checking = False
inventory = inventory
retry_files_enabled = True
retry_files_save_path = ~/.ansible-retry
```

Save and close the file

Now, open the **inventory** file and make it to look like this

```
web1 ansible_host=192.168.99.100
db ansible_host=192.168.99.101 ansible_user=vagrant ansible_ssh_pass=vagrant
clnt ansible_host=192.168.99.102

[webservers]
web1

[dbservers]
db

[stations]
clnt

[servers:children]
webservers
dbservers
```

Save and close the file

Execute again the playbook

`ansible-playbook playbook.yml`

For the inaccessible hosts, a special file is created. We can see it

`tree -a ~`

And check its content with

`cat ~/.ansible-retry/playbook.retry`

Open the **inventory** file and restore the variables section by adding the following block at the end

```
[servers:vars]
ansible_user=vagrant
ansible_ssh_pass=vagrant
```

Save and close the file

Re-execute the playbook against the failed hosts only with

`ansible-playbook playbook.yml --limit @/home/vagrant/.ansible-retry/playbook.retry`

Additional Techniques

Register & Debug

Create one more file named **register.yml** with the following content

```
---
- hosts: clnt
  become: false

  tasks:
    - name: Get system's kernel version
      shell: /usr/bin/uname -r
      register: kver

    - name: Debug info
      debug: var=kver
```

Save and close the file

Don't forget to add the **ansible_user** and **ansible_ssh_pass** variables after the **clnt** record in the **inventory** file

Execute it with

`ansible-playbook register.yml`

This way we captured the output from the **uname** command into the **kver** variable and displayed it at a later point using the debug module

Copy

Let us explore how we can copy (or distribute) files to hosts

Create an empty file named **copy.yml** and enter the following

```
---
- hosts: webservers
  become: true
```

```
  tasks:
    - name: Copy new index.html
      copy: src=html/index.html dest=/var/www/html/
```

Save and close the file

Create a folder **html**

Create an **index.html** file in the **html** folder with the following content

```
<h2>Hello, Ansible!</h2>
```

Save and close the file

Now, execute the following

```
ansible-playbook copy.yml
```

Check the result on the host by opening a browser tab and navigating to http://localhost:8080

Or navigate to http://192.168.99.100 or execute **curl http://192.168.99.100**

Conditional

Exit from the **Ansible** host

Let's add one more host by executing the following

```
vagrant up web2
```

Return on the **Ansible** machine

```
vagrant ssh ans
```

Make sure that you are in the folder from the previous paragraph (**~/p3/1**)

Adjust the **inventory** file to add the new host

First two lines should become

```
web1 ansible_host=192.168.99.100
web2 ansible_host=192.168.99.105
```

And then the **webservers** group should become

```
[webservers]
web1
web2
```

Save and close the file

First, let's see what **Ansible** gathers for each host in terms of facts

```
ansible clnt -m setup
```

This will return all facts about the **clnt** machine

For example, we can see the OS type with

```
ansible clnt -m setup | grep ansible_os
```

We can use this information to make decisions and make our configurations flexible

Let's see how we can use this

Create a new file named **conditional.yml** with the following content

```
---
- hosts: webservers
  become: false

  tasks:
    - name: Ask for the hostname on Red Hat machines
      shell: hostname
      register: rslt
      when: ansible_os_family == "RedHat"

    - name: Show the result on RedHat
      debug: var=rslt

    - name: Ask for the free memory on Debian machines
      shell: free -m
      register: rslt
      when: ansible_os_family == "Debian"

    - name: Show the result on Debian
      debug: var=rslt
```

This will execute one command on Red Hat-based machines and another on Debian-based machines

So, let's see it in action

`ansible-playbook conditional.yml`

Indeed, different commands were executed

Now extend this into something more useful

Create a new file named **webservers.yml** with the following content

```
---
- hosts: webservers
  become: true

  tasks:
    - name: Install Apache HTTP Server on Red Hat
      dnf: name=httpd state=present
      when: ansible_os_family == "RedHat"

    - name: Start Apache HTTP Server and Enable it on Red Hat
      service: name=httpd state=started enabled=true
      when: ansible_os_family == "RedHat"

    - name: Install Apache HTTP Server on Debian
      apt: name=apache2 state=present
      when: ansible_os_family == "Debian"

    - name: Start Apache HTTP Server and Enable it on Debian
      service: name=apache2 state=started enabled=true
      when: ansible_os_family == "Debian"
```

Save and close the file

Execute the playbook with

`ansible-playbook webservers.yml`

Test the result by opening a browser tab on the host and navigating to http://localhost:8081

Or by executing **curl http://192.168.99.105**

Templates

Navigate to the **~/p3/2** folder

`cd ~/p3/2`

Copy some of the files from the previous part

`cp ../1/ansible.cfg .`

`cp ../1/inventory .`

`cp ../1/webservers.yml .`

Create a folder named **templates**

`mkdir templates`

In it, create **index.j2** file with the following content

```
<html>
<head><title>Hello!</title></head>
<body>
<h2>Hello from Ansible on {{ v_host_type }}!</h2>
</body>
</html>
```

Save and close the file

Open the **webservers.yml** file and add two tasks for deploying the template

The file should look like this

```
---
- hosts: webservers
  become: true

  tasks:
    - name: Install Apache HTTP Server on Red Hat
      dnf: name=httpd state=present
      when: ansible_os_family == "RedHat"

    - name: Start Apache HTTP Server and Enable it on Red Hat
      service: name=httpd state=started enabled=true
      when: ansible_os_family == "RedHat"

    - name: Deploy index.j2 on Red Hat
      vars:
        v_host_type: RedHat
      template: src=templates/index.j2 dest=/var/www/html/index.html
      when: ansible_os_family == "RedHat"
```

```
  - name: Install Apache HTTP Server on Debian
    apt: name=apache2 state=present
    when: ansible_os_family == "Debian"

  - name: Start Apache HTTP Server and Enable it on Debian
    service: name=apache2 state=started enabled=true
    when: ansible_os_family == "Debian"

  - name: Deploy index.j2 on Debian
    vars:
      v_host_type: Debian
    template: src=templates/index.j2 dest=/var/www/html/index.html
    when: ansible_os_family == "Debian"
```

Save and close the file

Execute the playbook with

`ansible-playbook webservers.yml`

Check the result for the first web server on the host by navigating to http://localhost:8080

Then check the second web server, by navigating to http://localhost:8081

Or use **curl http://192.168.99.100** and **curl http://192.168.99.105**

Roles

Return on the host by exiting current session to the **Ansible** host

We can reset the environment partially by executing the following command

`vagrant destroy --force web1 web2`

Then re-create and re-provision the two web server machines with

`vagrant up web1 web2`

Return on the **Ansible** host

`vagrant ssh ans`

Enter the **~/p3/3** folder

`cd ~/p3/3`

Copy all files from **/vagrant/p3/3**

`cp -Rv /vagrant/p3/3/* .`

Explore the copied files and folders

First, check the **ansible.cfg** and the **inventory** files

Make some adjustments if needed

With the help of the **tree** command explore the directory structure

`tree .`

Explore the **main.yml**, **debian.yml** and **redhat.yml** files content

Check the **webservers.yml** file as well

Once, you are done, execute the following

`ansible-playbook webservers.yml`

Depending on the ansible installation in use, an error may appear that the **ufw** module is not known

To mitigate this, we must install the following collection

sudo ansible-galaxy collection install -p /usr/share/ansible/collections community.general

Depending on the boxes that you use, an error may return that for example, either **firewalld** or **ufw** is not installed

To mitigate this, we can extend the role files a bit

For example, the last portion of the one for **Debian** may become

```
- name: Debian - Collect package information
  package_facts:
    manager: auto

- name: Debian - Open HTTP Port in the Firewall
  ufw:
    rule: allow
    port: 80
    proto: tcp
  when: "'ufw' in ansible_facts.packages"
```

A similar block may be added to the **Red Hat** based file

Finally, you can check the result on the host

Open a browser tab and navigate to http://localhost:8080 and then to http://localhost:8081

Or use **curl http://192.168.99.100** and **curl http://192.168.99.105**

You should see the default web pages of the two distributions