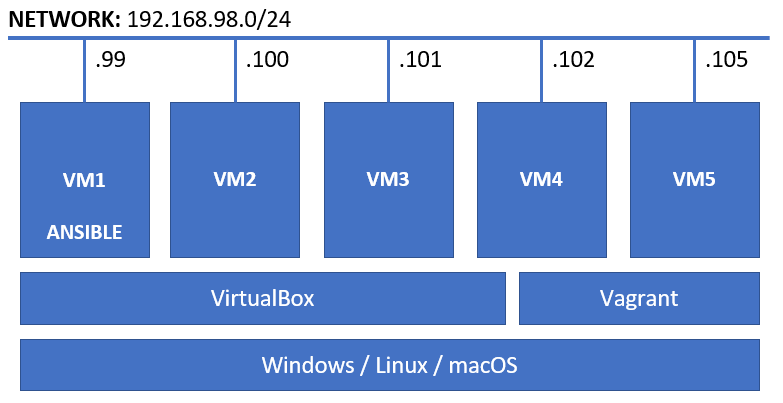
# Practice M4: Ansible

For this practice, our lab environment will look like this



We are going to use mostly **CentOS 8.x** boxes and at least one **Ubuntu**-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 with the following **Vagrantfile**

# -\*- mode: ruby -\*-

# vi: set ft=ruby :

Vagrant.configure(2) do |config|

  config.ssh.insert\_key = false

  config.vm.define "ans" do |ans|

    ans.vm.box = "shekeriev/centos-8-minimal"

    ans.vm.hostname = "ans.dob.lab"

    ans.vm.network "private\_network", ip: "192.168.98.99"

    ans.vm.synced\_folder "vagrant/", "/vagrant"

  end

  config.vm.define "web" do |web|

    web.vm.box = "shekeriev/centos-8-minimal"

    web.vm.hostname = "web.dob.lab"

    web.vm.network "private\_network", ip: "192.168.98.100"

    web.vm.network "forwarded\_port", guest: 80, host: 8080

  end

  config.vm.define "db" do |db|

    db.vm.box = "shekeriev/centos-8-minimal"

    db.vm.hostname = "db.dob.lab"

    db.vm.network "private\_network", ip: "192.168.98.101"

  end

  config.vm.define "clnt" do |clnt|

    clnt.vm.box = "shekeriev/centos-8-minimal"

    clnt.vm.hostname = "clnt.dob.lab"

    clnt.vm.network "private\_network", ip: "192.168.98.102"

  end

end

It is available in the **M4-1** folder of the provided archive

Once we are ready, we can deploy the infrastructure with

**vagrant up**

### 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**

#### 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 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]

192.168.98.100

192.168.98.101

[clnt]

192.168.98.102

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 192.168.98.100 -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 192.168.98.100 -a "hostname" -u vagrant**

Same error, same result

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

**ansible 192.168.98.100 -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 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 192.168.98.101 >> ~/.ssh/known\_hosts**

Or, we can use **ssh-keygen** generate key and copy it to the other station. Let’s first generate it

**ssh-keygen**

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

**ssh-copy-id 192.168.98.102**

Perhaps there are other ways as well

Now, we can execute again

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

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**

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**

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

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 **~/M4/M4-2/1** to work in and navigate to it

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

Enter the following as its first line

**web ansible\_host=192.168.98.100 ansible\_user=vagrant ansible\_ssh\_pass=vagrant**

Save and close the file

Execute the following command

**ansible web -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]**

**web**

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=192.168.98.101 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=192.168.98.102 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

web ansible\_host=192.168.98.100

db ansible\_host=192.168.98.101

clnt ansible\_host=192.168.98.102 ansible\_user=vagrant ansible\_ssh\_pass=vagrant

[webservers]

web

[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"**

### Variables

Create a new work folder **~/M4/M4-2/2** and navigate to it

Copy the **inventory** file from **~/M4/M4-2/1**

Create two sub-folders **group\_vars** and **host\_vars**

Create **all** file in the **group\_vars** folder 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

Let us create a second file named after one of the groups - **webservers** in the **group\_vars** folder 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 **web** but this time in the **host\_vars** folder

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 **web** (as the name of the variable file)

### Configurations

Create a new **~/M4/M4-2/3** folder and navigate to it

Copy the **inventory** file from **~/M4/M4-2/2**

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**

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 192.168.98.100**

**ssh-copy-id 192.168.98.101**

**ssh-copy-id 192.168.98.102**

Last one may return a warning (if you copied the key in ***Part 1****)*

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

web ansible\_host=192.168.98.100

db ansible\_host=192.168.98.101

clnt ansible\_host=192.168.98.102

[webservers]

web

[databases]

db

[stations]

clnt

[servers:children]

webservers

databases

Save and close the file

Execute the following command

**ansible web -i inventory -a "hostname"**

It should complete successfully

### Modules

Create a new **~/M4/M4-2/4** folder and navigate to it

Copy both the **inventory** and **ansible.cfg** files from **~/M4/M4-2/3**

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

Modules list can be retrieved with

**ansible-doc -l**

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://192.168.98.100**](http://192.168.98.100)

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**

Check again with

**curl** [**http://192.168.98.100**](http://192.168.98.100)

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

**ansible web -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 databases -i inventory -m dnf -a "name=mariadb,mariadb-server state=present" --become**

Then enable and start the service

**ansible databases -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

We will create a new one, with a **Vagrantfile** like this

# -\*- mode: ruby -\*-

# vi: set ft=ruby :

Vagrant.configure(2) do |config|

  config.ssh.insert\_key = false

  config.vm.define "ans" do |ans|

    ans.vm.box = "shekeriev/centos-8-minimal"

    ans.vm.hostname = "ans.dob.lab"

    ans.vm.network "private\_network", ip: "192.168.98.99"

    ans.vm.synced\_folder "vagrant/", "/vagrant"

    ans.vm.provision "shell", inline: <<EOS

sudo dnf install -y epel-release

sudo dnf install -y ansible

EOS

  end

  config.vm.define "web" do |web|

    web.vm.box = "shekeriev/centos-8-minimal"

    web.vm.hostname = "web.dob.lab"

    web.vm.network "private\_network", ip: "192.168.98.100"

    web.vm.network "forwarded\_port", guest: 80, host: 8080

  end

  config.vm.define "db" do |db|

    db.vm.box = "shekeriev/centos-8-minimal"

    db.vm.hostname = "db.dob.lab"

    db.vm.network "private\_network", ip: "192.168.98.101"

  end

  config.vm.define "clnt" do |clnt|

    clnt.vm.box = "shekeriev/centos-8-minimal"

    clnt.vm.hostname = "clnt.dob.lab"

    clnt.vm.network "private\_network", ip: "192.168.98.102"

  end

end

It is available in the **M4-3** folder of the provided archive

Once we are ready, we can deploy the infrastructure with

**vagrant up**

### Playbooks

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

**vagrant ssh ans**

Prepare the working folders and enter there

**mkdir -p ~/M4/M4-3/{1..3}**

**cd ~/M4/M4-3/1**

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

web ansible\_host=192.168.98.100

db ansible\_host=192.168.98.101

clnt ansible\_host=192.168.98.102

[webservers]

web

[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 = iventory

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 **ansigle.cfg** file to match this

[defaults]

host\_key\_checking = False

inventory = iventory

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

web ansible\_host=192.168.98.100

db ansible\_host=192.168.98.101 ansible\_user=vagrant ansible\_ssh\_pass=vagrant

clnt ansible\_host=192.168.98.102

[webservers]

web

[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

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**

#### 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 files

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>

#### Conditional

Exit from the **Ansible** host

Open the **Vagrantfile** and add one more host by adding this block just after the first machine

  config.vm.define "webu" do |webu|

    webu.vm.box = "shekeriev/ubuntu-20-04-server"

    webu.vm.hostname = "webu.dob.lab"

    webu.vm.network "private\_network", ip: "192.168.98.105"

    webu.vm.network "forwarded\_port", guest: 80, host: 8081

  end

Save and close the file

Execute

**vagrant up**

To add additional machine to the environment

Create a new session to the **Ansible** machine

**vagrant ssh ans**

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

First two lines should become

web ansible\_host=192.168.98.100

webu ansible\_host=192.168.98.105

And then the **webservers** group should become

[webservers]

web

webu

Save and close the file

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

---

- hosts: webservers

  become: true

  tasks:

    - name: Install Apache HTTP Server

      dnf: name=httpd state=present

      when: ansible\_os\_family == "RedHat"

    - name: Start Apache HTTP Server and Enable it

      service: name=httpd state=started enabled=true

      when: ansible\_os\_family == "RedHat"

    - name: Install Apache HTTP Server on Ubuntu

      apt: name=apache2 state=present

      when: ansible\_os\_family == "Debian"

    - name: Start Apache HTTP Server and Enable it on Ubuntu

      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>

### Templates

Navigate to the **~/M4/M4-3/2** folder

Copy some of the files from the previous part

**cp ../1/ansible.cfg .**

**cp ../1/inventory .**

**cp ../1/webservers.yml .**

Create folder **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

      dnf: name=httpd state=present

      when: ansible\_os\_family == "RedHat"

    - name: Start Apache HTTP Server and Enable it

      service: name=httpd state=started enabled=true

      when: ansible\_os\_family == "RedHat"

    - name: Deploy index.j2 on RedHat

      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 Ubuntu

      apt: name=apache2 state=present

      when: ansible\_os\_family == "Debian"

    - name: Start Apache HTTP Server and Enable it on Ubuntu

      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 on the host by navigating to <http://localhost:8080>

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

### Roles

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

We can reset the environment partially by executing the following commands

**vagrant destroy web --force**

**vagrant destroy webu --force**

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

**vagrant up**

Create new session to the **Ansible** host

**vagrant ssh ans**

Enter the **~/M4/M4-3/3** folder

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

**cp -Rv /vagrant/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

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**

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>

You should see the default web pages of the two distributions