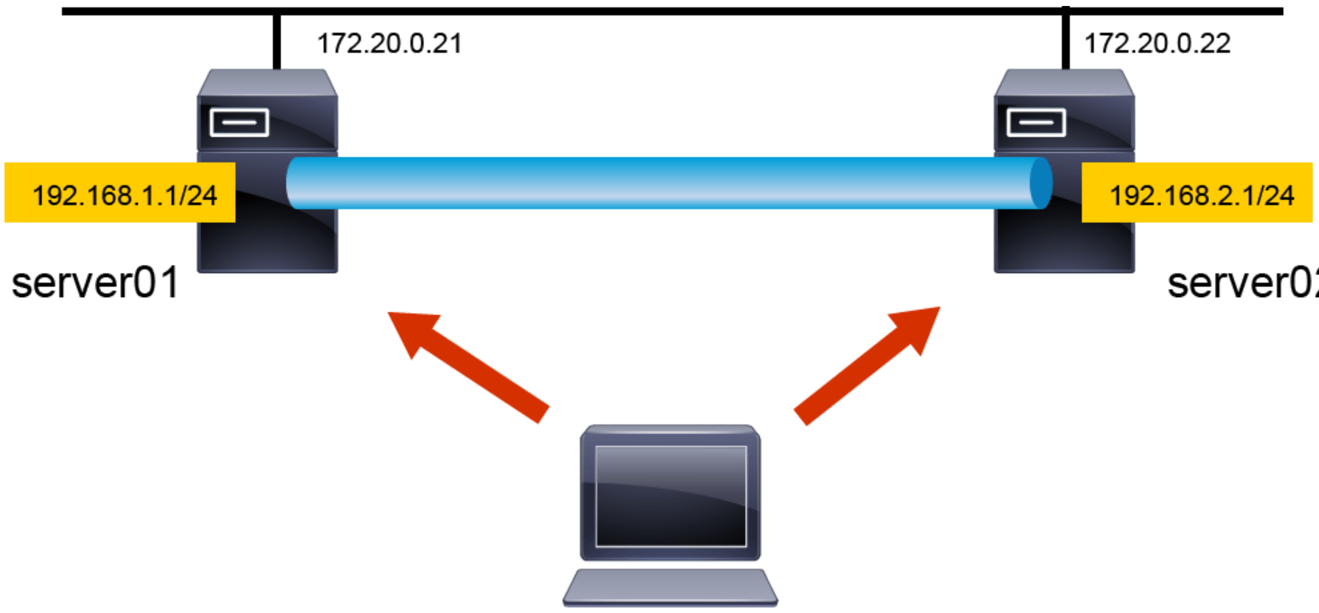
Discovery 12: Configure Network Parameters Using Ansible

Task 1: Create Ansible Playbook for Configuring Network Interface

In this task, you will configure an IP-in-IP tunnel between two servers. You will use Ansible as your configuration management tool. The following figure shows the use case that you will configure.



Activity

Step 1: First, configure the servers so that you can connect to them by using SSH keys. Generate new SSH keys that will be used to connect to the servers. Use Rivest, Shamir, and Adleman (RSA) as your algorithm. Store the keys without passphrases.

```
student@student-workstation:~/working_directory$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/student/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/student/.ssh/id_rsa.
Your public key has been saved in /home/student/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:hp3INRHAZqNsUjXk6C+RfQKAk6cklnaaBdfj9ZzrHB4 student@student-workstation
The key's randomart image is:
+---[RSA 2048]----+
| ++..+=.o.       |
|==o+.==o .       |
|=+=o+++.= .      |
|.o..+* = *       |
|   o+ * S .      |
|     o + E       |
|    . . + o      |
|     .   +       |
|                 |
+----[SHA256]-----+
```

Note
If you get a warning that the key exists, you can rewrite the existing key.

Step 2: Copy the public key to *server01* and *server02*.

```
student@student-workstation:~$ ssh-copy-id -p 20001 root@192.168.0.20
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/student/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already inst
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the n
root@192.168.0.20's password: 1234QWer

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh -p '20001' 'root@192.168.0.20'"
and check to make sure that only the key(s) you wanted were added.
```

```
student@student-workstation:~$ ssh-copy-id -p 20002 root@192.168.0.20
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/student/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already inst
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the n
root@192.168.0.20's password: 1234QWer

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh -p '20002' 'root@192.168.0.20'"
and check to make sure that only the key(s) you wanted were added.
```

Step 3: Test that you can connect to *server01* and *server02* without entering a password.

```
student@student-workstation:~$ ssh root@192.168.0.20 -p 20001
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-62-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Oct 11 10:29:21 2019 from 192.168.0.10
root@server01:~# exit
logout
Connection to 192.168.0.20 closed.
student@student-workstation:~$ ssh root@192.168.0.20 -p 20002
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-62-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Oct 11 10:29:31 2019 from 192.168.0.10
root@server02:~# exit
logout
Connection to 192.168.0.20 closed.
```

You should be able to connect to both servers.

Step 4: Connect to the student VM. Open Visual Studio Code. You can find the shortcut on the desktop of the student VM. Open the workspace folder by clicking File > Open Folder. Select the */home/student/working_directory* folder and click OK in the upper right corner.

Step 5: First, create an inventory file that is called *hosts*. Add a group called *servers* and add servers *server01* and *server02* to the inventory file. Add three additional variables next to the hosts: *ansible_host*, *ansible_port*, and *ansible_user*. Set the *ansible_host* to the IP address *192.168.0.20* for both servers. Set *ansible_port* for the first server to *20001* and *ansible_port* for the second server to *20002*. Set the *ansible_user* variable to the value *root* for both servers.

```
[servers]
server01 ansible_host=192.168.0.20 ansible_port=20001 ansible_user=root
server02 ansible_host=192.168.0.20 ansible_port=20002 ansible_user=root
```

Step 6: Activate the virtual environment by using the **pipenv shell** command.

```
student@student-workstation:~/working_directory$ pipenv shell
Launching subshell in virtual environment…
student@student-workstation:~/working_directory$  . /home/student/.local/share/virtualenvs/working_directory-
(working_directory) student@student-workstation:~/working_directory$
```

You will run Ansible from the virtual environment. You know that you are in virtual environment when you see the prompt prefix in the parentheses.

Step 7: Test that you can connect to the servers by using Ansible ad hoc command. Use the ping module.

```
(working_directory) student@student-workstation:~/working_directory$ ansible all -i hosts -m ping
server01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
server02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

You were able to connect to both servers. Now that the environment is ready, you can start building your Ansible playbooks.

Step 8: Create a template file for configuring network interfaces. Create a script template file with the name *network-interface.tmpl*. The script needs to check if the interface with correct interface exists. If not, you need to delete an existing interface with the same name and apply new configuration. The following commands are used to create a network interface.

```
ip link add name NAME type ipip local LOCAL_IP remote REMOTE_IP
ip link set NAME up
ip addr add IP_ADDRESS dev NAME
```

The interface name, interface IP address, local IP address, and remote IP address should be variables.

```
#!/bin/bash

IF_NAME={{ if_name }}
IP_ADDRESS={{ ip_address }}
LOCAL_IP={{ local_ip }}
REMOTE_IP={{ remote_ip }}

echo "Check if an interface with the correct IP address exists."
ip address show dev $IF_NAME | grep $IP_ADDRESS
if [[ $? -eq 0 ]]; then
  echo "The interface with the correct IP address exists."
  exit 0
fi

echo "Check if the interface exits."
ip address show dev $IF_NAME
if [[ $? -eq 0 ]]; then
  echo "The interface exists, deleting the interface."
  ip link delete dev $IF_NAME
fi

echo "Applying the new configuration."
ip link add name $IF_NAME type ipip local $LOCAL_IP remote $REMOTE_IP
ip link set $IF_NAME up
ip addr add $IP_ADDRESS dev $IF_NAME
```

Ansible uses the Jinja2 templating language. Variables are specified with double curly braces.

Step 9: Create a file with the name *configure-servers.yml*. Add a play header to the file. Specify the *name* of the play, add a *hosts* parameter, and add a *tasks* placeholder.

```
---
- name: Configure basic network settings on the servers
  hosts: all
  tasks:
```

Step 10: Add a first task to the play. Use the template module. Use the relative path to the template for the *src* parameter. The *dest* parameter should be set to /root/network-script.sh. Set the *mode* to 744.

```
    - name: Copy network configuration file to the servers
      template:
        src: network-interface.tmpl
        dest: /root/network-script.sh
        mode: '744'
```

Step 11: Before executing the playbook, you need to specify the variables. Create a folder with the name *host_vars*. Add two files to the folder. Name your files *server01* and *server02*.

Step 12: Add the variables that are needed for the template to the *server01* file.

Use the following values:

- if_name: *ipip1*
- ip_address: *192.168.1.1/24*
- local_ip: *172.20.0.21*
- remote_ip: *172.20.0.22*

```
---
if_name: ipip1
ip_address: 192.168.1.1/24
local_ip: 172.20.0.21
remote_ip: 172.20.0.22
```

Step 13: Populate the *server02* file as well.

Use the following values:

- if_name: *ipip1*
- ip_address: *192.168.2.1/24*
- local_ip: *172.20.0.22*
- remote_ip: *172.20.0.21*

```
---
if_name: ipip1
ip_address: 192.168.2.1/24
```

```
local_ip: 172.20.0.22
remote_ip: 172.20.0.21
```

Step 14: Now, run the playbook.

```
(working_directory) student@student-workstation:~/working_directory$ ansible-playbook -i hosts configure-serv

PLAY [Configure basic network settings on the servers] ******************************************************

TASK [Gathering Facts] **************************************************************************************
ok: [server02]
ok: [server01]

TASK [Copy network configuration file to the servers] *******************************************************
changed: [server01]
changed: [server02]

PLAY RECAP **************************************************************************************************
server01                   : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    igno
server02                   : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    igno
```

The playbook should have executed correctly. You should see that the task was marked as changed for both servers. It means that change was needed to reach the desired state. The file did not exist on the server and Ansible copied the file to the server. If you run the playbook again, the task should be marked as unchanged.

Step 15: Run the playbook again and observe the output. Compare the output with the previous run.

```
(working_directory) student@student-workstation:~/working_directory$ ansible-playbook -i hosts configure-serv

PLAY [Configure basic network settings on the servers] ******************************************************

TASK [Gathering Facts] **************************************************************************************
ok: [server01]
ok: [server02]

TASK [Copy network configuration file to the servers] *******************************************************
ok: [server01]
ok: [server02]

PLAY RECAP **************************************************************************************************
server01                   : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    igno
server02                   : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    igno
```

Since the file exists on both servers and Ansible discovered that no change is needed to reach the desired state, Ansible marks the task as *ok*, and continue with the execution without any change on the server.

Step 16: On *server01*, verify that the file exists. Check that the file's content is correct.

```
root@server01:~# cat /root/network-script.sh
#!/bin/bash

IF_NAME=ipip1
IP_ADDRESS=192.168.1.1/24
LOCAL_IP=172.20.0.21
REMOTE_IP=172.20.0.22

echo "Check if an interface with the correct IP address exists."
ip address show dev $IF_NAME | grep $IP_ADDRESS
if [[ $? -eq 0 ]]; then
  echo "The interface with the correct IP address exists."
  exit 0
fi

echo "Check if the interface exits."
ip address show dev $IF_NAME
if [[ $? -eq 0 ]]; then
  echo "The interface exists, deleting the interface."
  ip link delete dev $IF_NAME
fi

echo "Applying the new configuration."
ip link add name $IF_NAME type ipip local $LOCAL_IP remote $REMOTE_IP
ip link set $IF_NAME up
ip addr add $IP_ADDRESS dev $IF_NAME
```

You should see that file exists and the Jinja2 variables were replaced with the values specified in the *host_vars/server01* file.

Step 17: Add the task to the *configure-servers.yml* file that will execute the *network-script.sh* script on the remote server and store the output in the *network-script.log* file.

```
    - name: Execute the network configuration script
      shell: /root/network-script.sh > /root/network-script.log
```

Step 18: Run the playbook and observe the output.

```
(working_directory) student@student-workstation:~/working_directory$ ansible-playbook -i hosts configure-serv

PLAY [Configure basic network settings on the servers] ******************************************************
```

```
TASK [Gathering Facts] ******************************************************************************
ok: [server01]
ok: [server02]

TASK [Copy network configuration file to the servers] ***********************************************
ok: [server01]
ok: [server02]

TASK [Execute the network configuration script] *****************************************************
changed: [server02]
changed: [server01]

PLAY RECAP ******************************************************************************************
server01                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    igno
server02                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    igno
```

Step 19: On *server01*, check that the *network-script.log* file exists. Verify the log.

```
root@server01:~# cat /root/network-script.log
Check if an interface with the correct IP address exists.
Check if the interface exits.
Applying the new configuration.
```

You should see that file was created, and that configuration was applied.

Step 20: Check the interface status on *server01*.

```
root@server01:~# ip addr show dev ipip1
3: ipip1@NONE: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1480 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ipip 172.20.0.21 peer 172.20.0.22
    inet 192.168.1.1/24 scope global ipip1
       valid_lft forever preferred_lft forever
```

You should see that interface exits and that the correct IP address is configured on the interface.

Step 21: Add another task to the playbook. The task should execute the command on the server to add a new route in the routing table. The subnet and interface should be parameterized, which means that during the execution, Ansible should dynamically add those two values.

```
    - name: Add route
      shell: /sbin/ip route replace {{ remote_subnet }} dev {{ if_name }}
```

Step 22: Add the variable *remote_subnet* to the *host_vars/server01* file. The value should be *192.168.2.0/24*.

```
---
if_name: ipip1
ip_address: 192.168.1.1/24
local_ip: 172.20.0.21
remote_ip: 172.20.0.22
remote_subnet: 192.168.2.0/24
```

Step 23: Add the variable *remote_subnet* to the *host_vars/server02* file. Use the value *192.168.1.0/24*.

```
---
if_name: ipip1
ip_address: 192.168.2.1/24
local_ip: 172.20.0.21
remote_ip: 172.20.0.22
remote_subnet: 192.168.1.0/24
```

Step 24: Execute the playbook.

```
(working_directory) student@student-workstation:~/working_directory$ ansible-playbook -i hosts configure-serv

PLAY [Configure basic network settings on the servers] **********************************************

TASK [Gathering Facts] ******************************************************************************
ok: [server02]
ok: [server01]

TASK [Copy network configuration file to the servers] ***********************************************
ok: [server01]
ok: [server02]

TASK [Execute the network configuration script] *****************************************************
changed: [server02]
changed: [server01]

TASK [Add route] ************************************************************************************
changed: [server01]
changed: [server02]

PLAY RECAP ******************************************************************************************
server01                   : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    igno
server02                   : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    igno
```

Step 25: Check that route was added to the routing table on *server01*.

```
root@server01:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.20.0.1      0.0.0.0         UG    0      0        0 eth0
172.20.0.0      0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 ipip1
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 ipip1
```

You can see that route was added to the routing table.

Step 26: Finally, verify that you can ping between interfaces on *server01* and *server02*.

```
root@server01:~# ping -c 5 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=64 time=0.434 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=64 time=0.094 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=64 time=0.106 ms
64 bytes from 192.168.2.1: icmp_seq=4 ttl=64 time=0.089 ms
64 bytes from 192.168.2.1: icmp_seq=5 ttl=64 time=0.104 ms

--- 192.168.2.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4082ms
rtt min/avg/max/mdev = 0.089/0.165/0.434/0.134 ms
```

You can see that you can successfully ping the remote server, which confirms that the networking was correctly configured.

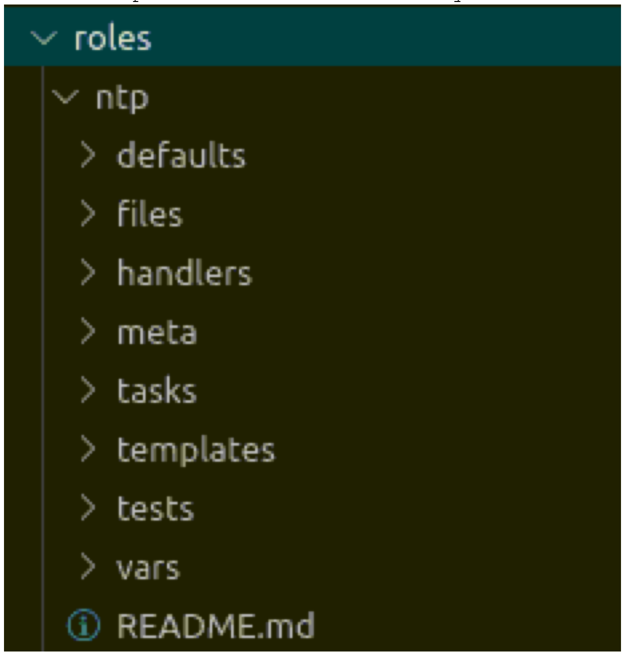
Task 2: Create Ansible Playbook for Configuring NTP

In this step, you will use Ansible to configure the NTP service on the servers. You will create an Ansible role for configuring NTP.

Activity

Step 1: Create the *roles* folder. Inside the roles folder, create a role with the name *ntp*. Use the **ansible-galaxy** command for creating the role.

```
(working_directory) student@student-workstation:~/working_directory$ mkdir roles
(working_directory) student@student-workstation:~/working_directory$ ansible-galaxy init roles/ntp
- roles/ntp was created successfully
```



Step 2: Create the NTP configuration template. Add a new file to the *roles/ntp/templates* folder called *ntp.conf.tmpl*. Use the Jinja2 language to loop over servers and add multiple servers to the configuration file.

```
{% for server in ntp_servers %}
server {{ server }} iburst
{% endfor %}
```

Step 3: Add two tasks to the *roles/ntp/tasks/main.yml*. The first task should install the *ntp* service to the server and the second task should start the *ntp* service.

```
---
- name: Install NTP
  package:
    name: ntp
    state: present

- name: Start NTP service
  service:
    name: ntp
    state: started
```

Step 4: Add another task that creates a configuration file from the template. Use the *ntp.conf.tmpl* as *src* and the file should be added to */etc/ntp.conf*. The *owner* and *group* should be *root*, and the *mode* should be set to *644*.

```
- name: Create NTP configuration from the template
  template:
    src: ntp.conf.tmpl
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: 644
```

Step 5: At the end, add a task that restarts the *ntp* service.

```
- name: Restart NTP service
  service:
    name: ntp
    state: restarted
```

Step 6: Now, create a folder called *group_vars* in the *working_directory* folder and add the file with the name *all* to the folder. Add variable *ntp_servers* to the file.

Add the following servers to the list:

0.north-america.pool.ntp.org
1.north-america.pool.ntp.org
2.north-america.pool.ntp.org
3.north-america.pool.ntp.org

```
ntp_servers:
  - 0.north-america.pool.ntp.org
  - 1.north-america.pool.ntp.org
  - 2.north-america.pool.ntp.org
  - 3.north-america.pool.ntp.org
```

The *group_vars* folder is used to define variables for specific group. If you add variables to the *all* file, Ansible applies these variables to all groups.

Step 7: Include the *ntp* role in the main playbook.

```
    - include_role:
        name: ntp
```

Step 8: Run the playbook.

```
(working_directory) student@student-workstation:~/working_directory$ ansible-playbook -i hosts configure-serv

PLAY [Configure basic network settings on the servers] ****************************

TASK [Gathering Facts] ************************************************************
ok: [server02]
ok: [server01]

TASK [Copy network configuration file to the servers] *****************************
ok: [server02]
ok: [server01]

TASK [Execute the network configuration script] ***********************************
changed: [server02]
changed: [server01]

TASK [Add route] ******************************************************************
changed: [server01]
changed: [server02]

TASK [include_role : ntp] *********************************************************

TASK [ntp : Install NTP] **********************************************************
 [WARNING]: Updating cache and auto-installing missing dependency: python3-apt

 [WARNING]: Could not find aptitude. Using apt-get instead

changed: [server02]
changed: [server01]

TASK [ntp : Start NTP service] ****************************************************
changed: [server02]
changed: [server01]

TASK [ntp : Create NTP configuration from the template] ***************************
changed: [server01]
changed: [server02]

TASK [ntp : Restart NTP service] **************************************************
changed: [server01]
changed: [server02]

PLAY RECAP ************************************************************************
```

```
server01                   : ok=8    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    igno
server02                   : ok=8    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    igno
```

You can see that Ansible ran the tasks from the role. The NTP should be configured on the servers.

Step 9: Verify the status of the *ntp* service on *server01*.

```
root@server01:~# service ntp status
 * NTP server is running
```

You can see that *ntp* service is running.

Step 10: Check the */etc/ntp.conf* file and verify that you can see the defined servers in the configuration file.

```
root@server01:~# cat /etc/ntp.conf
server 0.north-america.pool.ntp.org iburst
server 1.north-america.pool.ntp.org iburst
server 2.north-america.pool.ntp.org iburst
server 3.north-america.pool.ntp.org iburst
```

You can see all four defined servers in the NTP configuration file.