Automation

Objectives:

After completing this lab you should be able to:
1. Install and configure openSSH using keys.
2. Create a GitHub account and setup and use git on your machine for version control.
3. install ansible and run simple commands.

Required resources:

- 2 PCs with virtualBox [PC1,PC2].
- Internet connection.

Setting up the Work Environment

We will be using 6 VMs as listed in the table bellow all Vms will be created from ready made templates:

PC	VM	OS	Hostname	IP	NIC	RAM
PC1	Workstation1	Ubuntu Desktop	Workstation1	DHCP	Bridged	4 GB
	SRVR01	Ubuntu Server	SRVR01	DHCP	Bridged	1 GB
	SRVR02	Ubuntu Server	SRVR02	DHCP	Bridged	1 GB
PC2	Workstation1	Ubuntu Desktop	Workstation2	DHCP	Bridged	4 GB
	SRVR03	Ubuntu Server	SRVR03	DHCP	Bridged	1 GB
	SRVR04	CentOS	SRVR04	DHCP	Bridged	2 GB

NOTE: all VMs use the same username and password

username: student
password: student
Templates are found at http://172.16.107.254 , log in using student as username and St1234%^&* as password.
You can create all machines manually if you wish; but make sure all of them use the same username and password.

Install and Configure OpenSSH

On the server templates openssh-server is already installed but on both workstations it's not so install it using:
$ sudo apt install openssh-server

From both workstations open an ssh connection to each of the three servers. This initial connection is very important since we need to accept the servers' fingerprint in order for it to be added to our known hosts file in each of the workstations:
ssh student@[SRVR_IP]
accept the server fingerprint for all servers.

creating an SSH key pair will secure our server connections though it's not required for using ansible but it will make ansible connections strongly encrypted and we avoid security vulnerabilities. On **workstation1** do the following:

~$ ls -la .ssh

The folder does not contain any keys at the moment. Now let's create the keys for connecting to the servers.

~$ **ssh-keygen -t ed25519 -C "ansible"**
when asked for the name use: */home/student/.ssh/ansible*
the options in the previous command are as follows:
-t: type of key
-C: comment
ed25519 is the most secure key option.
SSH keys are stored in a subdirectory inside the users' home directory named .ssh [/home/student/.ssh].
List the contents of the .ssh directory you will find two files:
ansible.pub: public key
ansible: private key

show the contents of both keys and notice the difference.
Now let's copy the public key to each of the 3 servers using the following commands:
~$ cd .ssh
~/.ssh$ ssh-copy-id -i ansible.pub [SRVR_IP]
where [SRVR_IP] is replaced by your server IP. Repeat the command for all server IPs.

Check the .ssh directory on all servers to confirm that the public key has been added to the file ~/.ssh/authorized_keys.

Now copy both keys public and private to workstation2 since these must be identical because they identify a single user ID which is ansible:

```
Workstation1$  scp     ~/.ssh/ansible   student@[workstation2_IP]:/home/student/.ssh/ansible
Workstation1$  scp     ~/.ssh/ansible.pub   student@[workstation2_IP]:/home/student/.ssh/ansible.pub
```

To test our environment connect via ssh using the ansible key to all servers from both workstations using the following command:

```
Workstation1$ ssh    [SRVR_IP]
Workstation2$ ssh    [SRVR_IP]
```

if you need to specify the key in the command use:

```
Workstation1$ ssh   -i  ~/.ssh/ansible [SRVR_IP]
Workstation2$ ssh   -i  ~/.ssh/ansible [SRVR_IP]
```

notice that the connection opened without asking for a password.

Version Control

Version control is always at the heart of all system admin operations in all major enterprises, that is because such enterprises usually have a team of system and network admins administering their infrastructure and all need to know what other admins done on the system. That is accomplished by sharing config files and ansible playbooks that were used in changing the state of the infrastructure devices. This is were GitHub comes in play.

In this section of the lab we will create a GitHub account and use it’s repositories to share configuration files between both workstations we use as admin stations in our lab.

GitHub repositories can be downloaded (cloned) locally to our machines and kept in sync with the online version at all times. So if admin1 creates a configuration file or changes the contents of an existing one these changes are pushed to the repository and then pulled by admin2s’ workstation and that makes him aware of what his college did to the infrastructure.

On both workstations:

```
sudo apt update
sudo apt install git
```

now on one workstation navigate to www.github.com and sign up. After you signed in click the new button to create a repository. Name the repository **nislab**, check the initialize with a readme file and click create repository.

Click your profile picture on the top right corner, click settings, and open the SSH and GPG keys. Delete any keys and click New SSH Key. The title should be “ansible” and for the contents cat the public key file ansible.pub and paste all its contents inside the key field then click Add SSH Key.

Inside the .ssh directory of both workstations create a new file named config with the following contents:

```
Host github.com
  Hostname ssh.github.com
  Port 443
```

the file is case sensitive and the second and third lines begin with two spaces. Save the file and close it.

Now you should have GitHub.com open in a browser on both workstations. Navigate to the repository and click the green code button and copy the ssh link. Use the following command to download a copy of the repository inside your home directory on both workstations:

```
git clone [url copied]
```

type yes to accept GitHub fingerprint when asked.

list your home directory contents on both workstations, nislab directory should be present on both of them.

```
$ ls
nislab
$ cd nislab
nislab$ ls
README.MD
nislab$cat README.MD
```

We need to tell git who we are on both machines:

```
Workstation1$ git config   -- global user.name "Admin1"
Workstation1$ git config   -- global user.email "Admin1@nis.lab"
Workstation1$ cat ~/.gitconfig
Workstation2$ git config   -- global user.name "Admin2"
Workstation2$ git config   -- global user.email "Admin2@nis.lab"
Workstation2$ cat ~/.gitconfig
```

Let's edit the readme.md file on workstation1 using a text editor to add a few words to the end of the file like " This line is written by admin1 on workstation1". Save and close the file.

```
Workstation1~/nislab$ git status
```

this will show that a change was done. To see what was changed:

```
Workstation1~/nislab$ git diff
```

Let's add the file to the list of files to be committed:

```
Workstation1~/nislab$ git add README.MD
Workstation1~/nislab$ git status
Workstation1~/nislab$ git commit -m "Admin1 edited readme file"
```

```
Workstation1~/nislab$ git push origin main
```

check online you will see a change to the file. But workstaion2 is still out of sync. Let’s confirm by printing the contents of the readme file:

```
Workstation2~/nislab$ cat README.MD
Workstation2~/nislab$ git status
```

Notice the file contents still unchanged and the status command shows that there are changes. Let’s get those changes downloaded to workstation2:

```
Workstation2~/nislab$ git pull
Workstation2~/nislab$ cat README.MD
```

on workstation2 edit the file by adding “Admin2 says hi”. Add, commit, and push the file.
Check git status at workstation1 and do a pull then cat the contents.

There is much more to git than what we covered here but this is enough for us to keep version control in our institution in order.

Ansible [ad-hoc commands]

On both workstations:

```
sudo apt update
sudo apt install ansible
```

on workstation1:

```
cd nislab
nano inventory
```

now type the IP addresses of SRVR01 and SRVR02 each one in a line then **ctrl+o** then enter to save and to exit nano press **ctrl+x**. Add the file to GitHub and pull it on workstation2.

```
Workstation1~/nislab$ git add inventory
Workstation1~/nislab$ git commit -m “Admin1 created inventory file and added SRVR01 and SRVR02”
Workstation1~/nislab$ git push origin main
```

on the second workstation:

```
Workstation2~/nislab$ git pull
```

on workstation2 edit the inventory file and add the IP address of SRVR03 save and push to GitHub with a comment “Admin2 modified the inventory and added SRVR03” then pull it on workstation1.

On GitHub web page click the inventory file then click the history link you will see all versions of this file each with the comment that was written by each admin. If you click the comment you will get buttons to see file or browse the repository at that point in history [different versions are stored :)].

To run a command on all servers:

```
Workstation1~/nislab$ ansible all --key-file ~/.ssh/ansible -i inventory -m ping
```

all: command will run on all servers
~/.ssh/ansible: key used to connect to the servers
-i inventory: server IP list
-m: module to use in this case ping.

The ping here is NOT ICMP ping it’s an ansible module that tests for a successful SSH connection to each of the servers in the inventory list we created.

You should get 3 success messages which means ansible got connected to all 3 servers.

This command is too long let’s shorten it. To make it shorter we will store the inventory file name and key to be used in a configuration file:

```
Workstation1~/nislab$ nano ansible.cfg
```

input the following lines:

```
[defaults]
inventory = inventory
private_key_file = ~/.ssh/ansible
```

save the file and exit.

Now let’s run the new version of the previous ansible ping command:

```
Workstation1~/nislab$ ansible all -m ping
```

NOTE: if you use ansible commands outside of nislab directory ansible will use the files located inside */etc/ansible* directory. List the contents of that directory to see the files. But since we are running commands while inside a directory that has those files they will take priority which is excellent especially if you have different sites you are managing with different inventory and keys.

Push the files to GitHub and pull them on workstation2 and run the previous command to test.

The command **ansible all --list-hosts** will list the current hosts you are managing with ansible.
The command **ansible all -m gather_facts** is used for information gathering about hosts we are managing.
The output of that command is overwhelming but we can limit that using the --limit option.

```
$ ansible all -m gather_facts --limit [SRVR01_IP]
```

the output of this commands contains all information we can use in ansible when we use the "when" conditional if we use playbooks to target certain hosts in our inventory with commands that we want to limit to a certain OS or distribution for example.
Try to grep the output of the gather_facts module to get the OS distribution running on our SRVR01 server.
Push the cfg file to GitHub and pull it on workstation2 and do the following commands:

```
Workstation2~/nislab$ ansible all -m gather_facts --limit [SRVR01_IP] | grep ansible_distribution
```

what was the output?

____

now use the apt module to update the repository index on all servers.

```
Workstation1~/nislab$ ansible all -m apt -a update_cache=true
```

Did it succeed? _________

apt update command can't be run like that it should be sudo apt update. So we need to elevate the privileges of the command when ansible runs it. To do that use:

```
Workstation1~/nislab$ ansible all -m apt -a update_cache=true --become –ask-become-pass
```

-m apt: same as apt
-a update_cache=true : same as update
--become: same as sudo
--ask-become-pass: so that ansible asks us for the sudo password.

you can find information on what ansible modules you can use, and what options they have on docs.ansible.com/ansible/latest/modules.
Let's install a package on all servers:

```
Workstation1~/nislab$ ansible all -m apt -a "name=apache2" --become –ask-become-pass
```

this should install Apache web server on all 3 servers. Open the web browser and open http://[SRVR_IP] for each of the 3 servers.

Repeat the last command and notice that the output of the command will give a no change message. This command will install a package if it's absent but will not update that package if it has an update. If you wish to update the package every time you to specify the latest state option.

on SRVR01 use sudo apt update then sudo apt dist-upgrade **but don't upgrade** press ctrl+c to cancel notice that it will give packages that have updates available. Let's suppose snapd package has an update let's run a command that will install it if not present and upgrade it to the latest version if it is already installed:

```
Workstation1~/nislab$ ansible all  -m apt  -a “name=snapd state=latest”  --become --ask-become-pass
```

on the server repeat the check we did and look for snapd package if it's in the list of packages that have updates or not. It should be absent from that list.
Now lets update all packages on all servers.

```
Workstation1~/nislab$ ansible all  -m apt  -a “upgrade=dist”  --become –ask-become-pass
```

on one server do **sudo apt update** and **sudo apt dist-upgrade** , there should be no updates available.

Reflection

1. What is meant by automation?
2. Is SSH necessary for ansible to operate?
3. Can we use ansible without SSH keys?
4. Why did we use SSH keys?
5. What is scp used for?
6. What is meant by version control? And why is it important in Automation?
7. What is an inventory file?
8. What do we mean by the term ad-hoc commands?

END