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.
4. Create playbooks.
5. Use the “When” conditional.

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:

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	Workstation2	Ubuntu Desktop	Workstation2	DHCP	Bridged	4 GB
	SRVR03	Ubuntu Server	SRVR03	DHCP	Bridged	1 GB
	SRVR04	Almalinux	SRVR04	DHCP	Bridged	2 GB

NOTE: all VMs use the same username and password

username: student
password: student
Create all machines making sure all of them use the same username and password.

Install and Configure OpenSSH

1- Installation:
On all machines install openssh-server using the command:

```
$ sudo apt install openssh-server
```

From both workstations open an ssh connection to each of the Ubuntu 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.

2- SSH Key Management:
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 command:

```
~/.ssh$ ssh-copy-id -i ~/.ssh/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]
```

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
```

Sign up for a GitHub Account
On one workstation navigate to www.github.com and sign up.

Create A repository
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.

Add an SSH key
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.

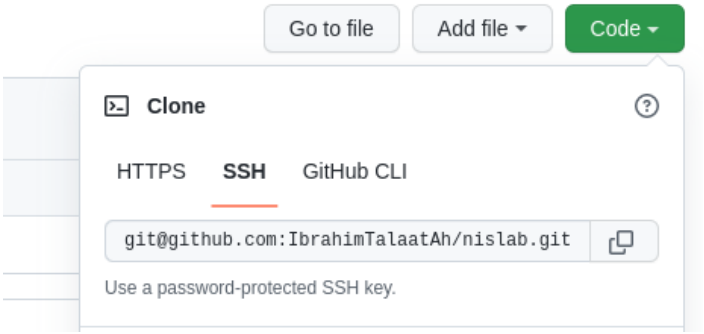
Configure github.com SSH host
Inside the .ssh directory of both workstations create a new file named **config** with the following contents:

```
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.

Clone the Repository

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**

Configure user ID on both workstations:

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
```

Testing version control
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" --beco
```

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.

Package Management Using Ansible Playbooks

Introduction

A task we want to perform on our hosts using ansible is called a play. A playbook contains one or more tasks we want to execute. playbooks are written using the yaml language which is a human readable data-serialization language. Yaml is usually used for configuration files.

The real strength of ansible comes from playbooks. when we write our playbooks we define the state we want our servers to be in and the commands we want ansible to perform to bring our servers to that state.

In the nislab directory on workstation1 create a file called install_apache.yml with the following contents:

```
---

- hosts: all
  become: true
  tasks:

  - name: install apache2 package
    apt:
      name: apache2
```

spaces are very important here. after the --- at the top leave 2 lines then on the 3rd line type one single hyphen (-) then space then hosts: then space then all. after that go down one line then leave two spaces and type become: then space then true.

--- : the start of the file.

- hosts: all : which hosts will be affected by the plays.

become: true: for sudo

tasks: : means after this will be the list of plays to be executed.

- name: install apache2 package : the name (description) of this play

apt: : the module to be executed , here it’s apt

name: apache2 : the name of the package we want to install.

to run the yml file use the following command:

```
ansible-playbook --ask-become-pass install_apache.yml
```

The output will have a few important info when running the play on each host:

ok: this lists the number of plays that ran without problems on the host.

changed: the number of plays that made changes when ran on the host.

unreachable: if the host is offline.

failed: number of failed plays on this host.

skipped: number of plays that were skipped because the host did not meet the conditions for running this play.

rescued: number of plays that ran as a rescue because other plays failed to run.

ignored: number of ignored plays.

The previous playbook could either succeed or fail depending on the repository index status. On Linux systems as you already know we need to update the repository index before trying to install packages because we might get an error that the package was not found. this happens because URLs keep changing all the time and we need the new links for the packages to be downloaded.

we do that by the command apt update. now let’s modify our playbook to include the ansible task equivalent to that:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
```

run the playbook and notice the tasks that are executed successfully. ok=3 which are the gathering fact, update repository, and install apache2 package tasks. the changed will be only 2 since gathering facts makes no changes. The update repository task will always make a change whenever we run the playbook. let's make other changes to the playbook:

```
---

- hosts: all
  become: true
  tasks:

  - name: install Apache2 and PHP packages
    apt:
      name:
        - apache2
        - libapache2-mod-php
      update_cache: yes
```

here we added multiple packages and updated the cache in one go. run the playbook to make changes and notice the output.

now this playbook will install apache2 and libapache2-mod-php packages if they are not installed but it won't update them if there are updates available. To make the playbook capable of updating packages we need to use the state parameter. let's edit the playbook as follows:

```
---
 - hosts: all
   become: true
   tasks:

   - name: install Apache2 and PHP packages
     apt:
       name:
         - apache2
         - libapache2-mod-php
       update_cache: yes
       state: latest
```

state: latest will make sure the package is always the latest one available.

open a browser and open one of the server IPs to see the default Apache start page.

let's create another playbook that removes these packages. you can either write it from scratch or make a copy of the install_apache.yml file and make changes.

create remove_apache.yml with the following contents:

```
---

- hosts: all
```

```
become: true
tasks:

- name: remove Apache2 and PHP packages
  apt:
    name:
      - apache2
      - libapache2-mod-php
    state: absent
```

the **state: absent** parameter value means removing the package if present.

run the playbook and notice the output.

open a browser and try opening the site on one of your servers.

run the install playbook and refresh your browser page.

Version Control

we added two files to our nislab directory which is connected to a git repository so we need to add both these files to gihub.
nislab$ git status
nislab$git add .
nislab$git commit -m “install/remove apache and php playbooks created by admin 1”
nilab$ git push origin main
now do a git pull on workstation2 to get these files downloaded.

The 'when' Conditional

The playbook we created will work fine if all servers are Debian based systems since we are using the apt module. if some of your servers have a base other than Debian then the playbook will fail when used on them.

on workstation2 modify the inventory file by adding the AlmaLinux server IP address. run the playbook and notice the output especially for AlmaLinux server.

The command failed since AlmaLinux is not a Debian based distribution rather it is based on RHEL (Red Hat Enterprise Linux) which doesn’t have apt as a package manager instead it uses dnf.

let’s modify the playbook to remove that error:

```
---

- hosts: all
  become: true
  tasks:

  - name: install Apache2 and PHP packages on Ubuntu
    apt:
      name:
        - apache2
        - libapache2-mod-php
      update_cache: yes
      state: latest
    when: ansible_distribution == "Ubuntu"
```

here we added [when: ansible_distribution == “Ubuntu”] to our task. run the playbook and notice what happens when executing each of the plays on AlmaLinux. you will see that it was skipped since it failed to meet the when condition.

To know the distributions of your servers you can run the gather_facts module on each server:

ansible all -m gather_facts | grep ansible_distribution

now let’s modify the playbook to also do the same thing for the AlmaLinux server, add the :

```
---

- hosts: all
  become: true
  tasks:
  - name: install Apache2 and PHP packages on Ubuntu
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
      update_cache: yes
    when: ansible_distribution == "Ubuntu"

  - name: install httpd and PHP packages on ALmaLinux
    dnf:
      name:
        - httpd
        - php
      state: latest
      update_cache: yes
    when: ansible_distribution == "AlmaLinux"
```

run the playbook and check the output.

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?
9. What is a playbook?
10. What is Yaml?
11. What is a package manager?
12. What is an ansible module?

END