Automation

---

1. مقدمة حول المحتوى

الملف يتناول **الأتمتة (Automation)**، والتي تهدف إلى تحسين العمليات من خلال تنفيذ المهام تلقائيًا بدلاً من إجرائها يدويًا. يمكن أن تكون الأتمتة متعلقة بالشبكات، الأمن السيبراني، أو حتى العمليات التجارية.

Install and Configure OpenSSH

sudo apt install openssh-server

sudo systemctl start ssh

sudo systemctl enable ssh

sudo systemctl status ssh

ssh student@[SRVR_IP]

فتح اتصال SSH من كلا الجهازين إلى جميع خوادم Ubuntu وقبول البصمة (Fingerprint)

ما الهدف من هذه الخطوة؟

عند الاتصال لأول مرة بأي خادم عبر **SSH**، سيطلب منك النظام قبول بصمة (fingerprint) هذا الخادم. هذه البصمة تُستخدم للتأكد من أنك تتصل بالخادم الصحيح، وليست هناك أي محاولة لاعتراض الاتصال (**Man-in-the-Middle Attack**). بمجرد قبولها، يتم حفظها في ملف `ssh/known_hosts./~` حتى لا يُطلب منك قبولها مرة أخرى عند الاتصال بهذا الخادم لاحقًا.

ls -la .ssh

التحقق من وجود مفاتيح SSH حاليًا

على **Workstation1**، قم بتشغيل الأمر التالي للتحقق مما إذا كان هناك أي مفاتيح SSH موجودة بالفعل:

```
ls -la ~/.ssh
```

شرح الأمر:

- `ls -la` : يعرض جميع الملفات، بما في ذلك الملفات المخفية.
- `ssh./~` : هذا هو المجلد الذي يتم فيه تخزين مفاتيح SSH الخاصة بالمستخدم.

ماذا تتوقع أن ترى؟

- إذا كان المجلد فارغًا (أو لا يحتوي على ملفات مثل `id_rsa` أو `id_rsa.pub`)، فهذا يعني أنه لا توجد مفاتيح SSH حالية.
- إذا وجدت ملفات مثل `id_rsa` (المفتاح الخاص) و `id_rsa.pub` (المفتاح العام)، فهذا يعني أنه تم إنشاء مفتاح SSH سابقًا.

ssh-keygen -t ed25519 -C "ansible”

1- إنشاء مفتاح SSH

نفّذ الأمر التالي لإنشاء مفتاح SSH:

```
ssh-keygen -t ed25519 -C "ansible"
```

شرح الخيارات:

- `ssh-keygen` → أداة إنشاء مفاتيح SSH.
- `t ed25519-` → يحدد نوع المفتاح ليكون **Ed25519** (أكثر أمانًا وأداءً من RSA).
- `"C "ansible-` → تعليق لتعريف المفتاح، يُستخدم عادةً للتوضيح فقط.

ssh-copy-id -i ~/.ssh/ansible.pub [SRVR_IP]

1- نسخ المفتاح العام إلى الخوادم

- استخدم الأمر التالي لإرسال المفتاح العام إلى كل خادم:

```
ssh-copy-id -i ~/.ssh/ansible.pub student@[SRVR_IP]
```

- **استبدل** `[SRVR_IP]` **بعنوان الـ IP الخاص بالخادم.**
- كرر هذا الأمر لكل خادم لديك.

مثال عملي:

إذا كان لديك **3 خوادم** بعنواني IP:

1. `172.20.10.2` (الخادم الرئيسي Master)
2. `172.20.10.3` (العقدة الأولى Worker Node1)
3. `172.20.10.4` (العقدة الثانية Worker Node2)

نفّذ الأوامر التالية **واحدًا تلو الآخر** من محطة العمل:

```
ssh-copy-id -i ~/.ssh/ansible.pub student@172.20.10.2
ssh-copy-id -i ~/.ssh/ansible.pub student@172.20.10.3
ssh-copy-id -i ~/.ssh/ansible.pub student@172.20.10.4
```

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

1️⃣ - نسخ المفاتيح من Workstation1 إلى Workstation2

على **Workstation1**، نفّذ الأوامر التالية **لتنقل المفتاحين عبر (Secure Copy Protocol)** `scp` :

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

🔹 **استبدل** `[workstation2_IP]` **بعنوان الـ IP الخاص بـ Workstation2.**

🔍 **شرح الأوامر:**

- `scp` → أداة لنسخ الملفات بأمان عبر SSH.
- `ssh/ansible./~` → المفتاح **الخاص** (يجب أن يبقى محميًا).
- `ssh/ansible.pub./~` → المفتاح **العام** (يُستخدم للمصادقة على الخوادم).
- `/student@[workstation2_IP]:/home/student/.ssh` → الوجهة التي سيتم نسخ المفاتيح إليها.

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

to make sure the operation is working well

3- التحقق من نجاح العملية

بعد نسخ المفتاح، تأكد من إضافته إلى ملف `ssh/authorized_keys./~` على كل خادم.

🔹 **سجّل الدخول إلى أي خادم باستخدام SSH:**

```
ssh student@172.20.10.2
```

- إذا لم يُطلب منك إدخال كلمة المرور، فهذا يعني أن المفتاح يعمل بنجاح! ✅
- إذا طُلب منك كلمة المرور، فتحقق من أن المفتاح قد أُضيف بشكل صحيح.

Version Control

🔹 تثبيت وإعداد Git لإدارة إصدارات الملفات بين Workstation1 و Workstation2

⚡ **لماذا نستخدم GitHub في إدارة البنية التحتية؟**

- في بيئات الشركات الكبرى، هناك **عدة مسؤولين** عن إدارة الأنظمة والشبكات.
- يجب أن يكون **كل المسؤولين على علم بأي تغييرات** يتم إجراؤها على الملفات المهمة مثل **ملفات التهيئة (config files) وملفات Ansible playbooks**.
- **GitHub** يسمح بمشاركة هذه الملفات بين الجميع بسهولة، بحيث يستطيع أي مسؤول رؤية التعديلات وسحب (pull) آخر التحديثات.
- عند تعديل **مسؤول 1** لملف معين، يتم **رفع التعديلات إلى GitHub**، ثم يقوم **مسؤول 2** بتنزيل التحديثات حتى يكون الجميع على نفس الصفحة.

sudo apt update

sudo apt install git

Configure github.com SSH host

🔹 إعداد GitHub SSH Host وتكوين الاتصال عبر Port 443

في هذه الخطوة، سنقوم **بتعديل ملف التكوين** (`config`) **في مجلد** `ssh.` **على كلا الجهازين** بحيث نتمكن من الاتصال بـ GitHub عبر **Port 443** (المنفذ عادة ما يُستخدم للاتصالات الآمنة عبر HTTP). وهذا مهم إذا كانت الشبكة الخاصة بك قد **تمنع الاتصال عبر المنفذ 22** الذي يستخدمه SSH بشكل افتراضي.

cd ~/.ssh

nano config

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

Clone the Repository

- **استنساخ المستودع من GitHub وتكوين هوية المستخدم على كلا الجهازين**

في هذه الخطوة، سنقوم **باستنساخ المستودع** من GitHub على كلا الجهازين ثم نقوم **بتكوين هوية المستخدم** الخاصة بـ Git بحيث يعرف Git من هو المستخدم الذي يقوم بالتعديلات على المستودع.

Configure user ID on both workstations

We need to tell git who we are on both machines

```
git config --global user.name "Admin1"        # تحديد اسم المستخدم
git config --global user.email "Admin1@nis.lab"  # تحديد البريد الإلكتروني
cat ~/.gitconfig                              # عرض التكوينات
```

- 1️⃣ **تكوين هوية المستخدم في Git على كلا الجهازين**

أولاً، يجب علينا تحديد **اسم المستخدم والبريد الإلكتروني** الذي سيُستخدم في Git على كلا الجهازين (Workstation1 و Workstation2). هذا يساعد Git في تتبع التعديلات التي يتم إجراؤها.

على Workstation1:

1. قم بتحديد اسم المستخدم باستخدام الأمر التالي:

```
Edit  Copy
git config --global user.name "Admin1"
```

2. قم بتحديد البريد الإلكتروني باستخدام الأمر التالي:

```
Edit  Copy
git config --global user.email "Admin1@nis.lab"
```

3. للتحقق من التكوين، يمكنك عرض ملف التكوين باستخدام:

```
Edit  Copy
cat ~/.gitconfig
```

يجب أن ترى شيئًا مثل:

```
Edit  Copy
[user]
    name = Admin1
    email = Admin1@nis.lab
```

Testing version control

nano README.md

🔹 2️⃣ اختبار التحكم في النسخ عبر Git

الآن بعد أن تم تكوين Git على كلا الجهازين، سنجري بعض التعديلات على ملف `README.md` على **Workstation1**، ثم نتابع التغييرات عبر الجهازين.

2.1 تعديل الملف `README.md` على Workstation1

1. على **Workstation1**، قم بفتح **ملف** `README.md` باستخدام محرر نصوص (مثل `nano` أو `vim`) وأضف السطر التالي في نهاية الملف:

```
This line is written by admin1 on workstation1
```

2. **احفظ وأغلق الملف.**

Workstation1~/nislab$ git status

Workstation1~/nislab$ git diff

3. تحقق من حالة Git لمعرفة ما إذا كان تم تعديل الملف:

```
git status
```

ستظهر رسالة تشير إلى أن الملف قد تم تعديله، مثل:

```
modified:   README.md
```

4. لعرض التغييرات التي تمت على الملف:

```
git diff
```

سيُعرض لك الفرق بين النسخة الحالية والنسخة السابقة من الملف.

💡 **اضافة الملف المعدل لقائمة الملفات التي سيتم رفعها ← git add README.md**

بعد إضافة التغييرات Git التحقق من حالة ← git status

تنفيذ عملية كوميت مع رسالة توضح التغيير ← git commit -m "Admin1 edited readme file"

في Git، **الكوميت (commit)** هو عملية حفظ التغييرات التي قمت بها في مشروعك المحلي (على جهازك) بشكل دائم، داخل تاريخ المشروع. عندما تقوم بعمل **commit**، أنت تقوم بتسجيل حالة المشروع في لحظة معينة، بحيث يمكن العودة إليها لاحقًا أو تتبع التغييرات التي حدثت على المشروع مع مرور الوقت.

ماذا يتضمن الكوميت؟

1. **تسجيل التغييرات**: عند تنفيذ **commit**، يتم حفظ التغييرات التي قمت بها على الملفات التي تم إضافتها باستخدام الأمر `git add`.
2. **رسالة الكوميت**: يجب عليك إضافة رسالة توضح التغييرات التي قمت بها. هذه الرسالة تساعد الآخرين (وأنت في المستقبل) في فهم سبب التعديل الذي تم في هذا الكوميت.

مثال عملي:

بعد أن تعدل الملف، نقوم بتنفيذ الأوامر التالية:

1. **إضافة التغييرات** إلى قائمة التغييرات التي نريد حفظها:

```
Edit  Copy
git add README.md  # إضافة التغييرات التي تم إجراؤها على ملف README.md
```

2. **تنفيذ الكوميت**:

```
Edit  Copy
git commit -m "Admin1 edited readme file"  # حفظ التغييرات مع رسالة تشرح التعديل
```

تفاصيل الرسالة:

- الرسالة `"Admin1 edited readme file"` هي رسالة توضح ما الذي قمت بتغييره.
- الرسائل تساعد في توثيق التعديلات لكي يعرف أي شخص يعمل معك على المشروع ماذا تم تغييره ولماذا.

بعد الكوميت:

- التغييرات تصبح محفوظة في السجل التاريخي للمشروع، ويمكنك أن ترى هذه التغييرات باستخدام الأوامر مثل:

```
Edit  Copy
git log  # عرض سجل الكوميتات التي تم إجراؤها
```

💡 **git push origin main** → **دفع التغييرات إلى GitHub**

git pull Workstation1 → **سحب التغييرات من GitHub إلى**

Ansible [ad-hoc commands]

sudo apt update
sudo apt install ansible

create an inventory file

cd nislab

nano inventory

3. اكتب عناوين الـ IP للخوادم:

```
<IP_of_SRVR01>
<IP_of_SRVR02>
```

(كل IP في سطر)

4. لحفظ الملف:

- اضغط `Ctrl + O` ثم `Enter`
- للخروج: `Ctrl + X`

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

Workstation2~/nislab$ git pull

🔁 ثالثًا: إضافة الملف إلى GitHub من workstation1

```
git add inventory
git commit -m "Admin1 created inventory file and added SRVR01 and SRVR02"
git push origin main
```

💻 رابعًا: سحب التعديلات من GitHub على workstation2

```
cd nislab
git pull
```

✏️ خامسًا: تعديل الملف من workstation2

1. افتح الملف:

```
nano inventory
```

2. أضف IP الخاص بـ `SRVR03` في سطر جديد.
3. احفظ وعدّل التغييرات:

```
git add inventory
git commit -m "Admin2 modified the inventory and added SRVR03"
git push origin main
```

🔍 سابقًا: مراجعة تاريخ الملف على GitHub

1. ادخل على GitHub من المتصفح.
2. افتح ملف `inventory` .
3. اضغط على **History**.
4. سترى كل التعديلات التي تمت على الملف، مع كل **تعليق (commit message)**.
5. إذا ضغطت على أحد التعليقات:

- سترى زر "View file" لعرض النسخة في ذلك الوقت.
- وزر "Browse repository at this point" لاستعراض كل المشروع كما كان في ذلك الوقت.

📌 **الفائدة**: Git يحتفظ بجميع النسخ، وتقدر ترجع لأي نسخة سابقة بسهولة! 😊

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

🎯 الهدف:

نريد تشغيل أمر Ansible على **جميع السيرفرات** باستخدام ملف `inventory` ومفتاح SSH خاص، لكن الأمر طويل جدًا، فبنختصره باستخدام ملف إعدادات `ansible.cfg`.

✅ أولًا: تشغيل الأمر الطويل بدون اختصار

الأمر التالي يُستخدم لتشغيل **أمر ping** (أنسبل-ستايل) على كل السيرفرات:

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

شرح كل جزء:

الجزء	وظيفته
`ansible`	هو البرنامج نفسه
`all`	معناها: شغل الأمر على **جميع السيرفرات** الموجودة في ملف inventory
`key-file ~/.ssh/ansible--`	يحدد مفتاح SSH الخاص للدخول إلى السيرفرات
`i inventory-`	يحدد ملف فيه عناوين السيرفرات (IP addresses)
`m ping-`	يستخدم **الوحدة (module)** "ping" الخاصة بـ Ansible (تختبر الاتصال عبر SSH—not ICMP)

🔄 إذا نجحت العملية، ستشوف ثلاث رسائل نجاح (واحدة لكل سيرفر)، معناها أن الاتصال عبر SSH ناجح بين جهازك وهذه السيرفرات.

> **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 above 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
```

and inside the file:

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

🔧 ثانيًا: اختصار الأمر باستخدام ansible.cfg

بدل ما تكتب هذا الأمر الطويل في كل مرة، نقدر نخزن القيم الثابتة (مثل اسم ملف inventory ومفتاح SSH) في ملف إعدادات اسمه `ansible.cfg`.

الخطوات:

1. افتح محرر `nano` لإنشاء ملف الإعدادات:

```
nano ansible.cfg
```

2. أضف السطور التالية:

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

شرح:

السطر	معناه
`[defaults]`	بداية قسم الإعدادات الافتراضية
`inventory = inventory`	يخبر Ansible أن ملف inventory اسمه `inventory`
`private_key_file = ~/.ssh/ansible`	يحدد المفتاح الافتراضي للاتصال بالسيرفرات

save the file and exit.

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

Workstation1~/nislab$ ansible all -m ping

✅ بعد كذا، تقدر تختصر الأمر بهذا الشكل:

```
ansible all -m ping
```

بدون الحاجة لكتابة `key-file--` و `i-` كل مرة، لأن `ansible.cfg` يحتوي على هذه المعلومات الآن.

📁 أول ملاحظة مهمة:

> "إذا شغلت أوامر Ansible من خارج مجلد `nislab`، أنسبل سيستخدم الملفات الموجودة في `etc/ansible/`."

📌 وش يعني هذا؟

- عندك **ملف** `ansible.cfg` **و** `inventory` داخل مجلد `nislab`.
- إذا أنت داخل مجلد `nislab` وشغلت أوامر أنسبل، بيقرأ من ملفاتك الخاصة.
- لكن إذا خرجت من المجلد، أنسبل بيرجع يقرأ من الملفات الافتراضية اللي داخل `etc/ansible/`.

✅ **هذه ميزة مهمة** لما تدير أكثر من مشروع أو موقع (Site)، وكل واحد له إعداداته الخاصة.

ansible all --list-hosts

🔸 لعرض كل السيرفرات اللي تشتغل عليها:

```
ansible all --list-hosts
```

ansible all -m gather_facts

🔸 **لجمع معلومات عن الأجهزة (مثل نوع النظام، RAM، IP، ..الخ):**

```
ansible all -m gather_facts
```

لكن... الناتج طويل جدًا! نقدر نختصره باستخدام `--limit` عشان نستهدف جهاز واحد فقط.

🔍 **مثال:**

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

وهذا بيطبع لك معلومات الجهاز SRVR01 فقط.

🧠 كيف نعرف نظام التشغيل المثبت في SRVR01؟

بعد ما تشغل الأمر أعلاه، تقدر تستخدم `grep` لاستخراج اسم التوزيعة:

```
ansible all -m gather_facts --limit [SRVR01_IP] | grep ansible_distribution
```

✳️ الناتج بيكون مثل:

```
"ansible_distribution": "Ubuntu"
```

Use the apt module to update the repository index on all servers.

ansible all -m apt -a update_cache=true --become --ask-become-pass

❌ هذا الأمر ما راح يشتغل:

```
ansible all -m apt -a update_cache=true
```

لأن أمر `apt update` يحتاج صلاحيات root (يعني `sudo`).

✅ استخدم هذا الأمر بدلًا منه:

```
ansible all -m apt -a update_cache=true --become --ask-become-pass
```

شرح:

الجزء	المعنى
`m apt-`	نستخدم وحدة `apt` الخاصة بالتعامل مع الحزم
`a update_cache=true-`	مثل `apt update`
`become--`	يشغّل الأمر بصلاحيات root (مثل sudo)
`ask-become-pass--`	يطلب منك كلمة مرور sudo

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.

ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass

sudo apt list --upgradable

📦 تثبيت أو تحديث حزمة snapd:

نفترض إن حزمة `snapd` تحتاج تحديث:

```
ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
```

- إذا الحزمة غير مثبتة، يتم تثبيتها.
- إذا كانت قديمة، يتم تحديثها.
- إذا كانت محدثة، ما يتغير شيء.

🧪 تقدر تتأكد بعدها على السيرفر نفسه باستخدام:

```
sudo apt list --upgradable
```

🔹 إذا ما شفت `snapd` في القائمة، يعني أنها محدثة ✅

ansible all -m apt -a "upgrade=dist" --become --ask-become-pass

sudo apt dist-upgrade

⬆️ أخيرًا: تحديث جميع الحزم في كل السيرفرات:

```
ansible all -m apt -a "upgrade=dist" --become --ask-become-pass
```

وهذا نفس أمر:

```
sudo apt dist-upgrade
```

بعدها، على أحد السيرفرات، تأكد:

```
sudo apt update
sudo apt dist-upgrade
```

🔍 إذا ما طلع لك أي حزم قابلة للتحديث = ممتاز، التحديث تم بنجاح.

PART-1-DONE

🚀 أولاً: ما هو Ansible؟

Ansible هو أداة مفتوحة المصدر تستخدم لإدارة إعدادات السيرفرات (Configuration Management)، نشر البرامج (Software Deployment)، وتشغيل أوتوماتيكي للمهام (Automation) على أنظمة متعددة دفعة واحدة.

📘 ما هو الـ **Playbook**؟

- **Playbook** هو ملف يحتوي على مجموعة من التعليمات (Tasks) مكتوبة بلغة **YAML**.
- كل **Task** تقوم بعمل معين (مثل تثبيت برنامج، نسخ ملف، تشغيل خدمة...).
- نكتب الـ **Playbook** لتعريف الحالة التي نريد أن تكون عليها الأنظمة، وAnsible ينفذ هذه المهام للوصول لتلك الحالة.

📝 ما هي لغة YAML؟

- اختصار لـ: **YAML Ain't Markup Language**
- لغة سهلة القراءة للبشر وتُستخدم عادة في ملفات الإعدادات.
- تعتمد بشكل كبير على **المسافات (spaces)** بدل الأقواس أو الأقواس المعقوفة.
- **المسافات مهمة جدًا!** لا تستخدم الـ Tabs، فقط استخدم مسافات عادية.

inside the nislab dir

touch install_apache.yml

and write the follwing:

- - -

```
- hosts: all
  become: true
  tasks:
  - name: install apache2 package
    apt:
      name: apache2
```

🧩 شرح السطر بسطر

`---`

- بداية الملف.
- هذا يُعرّف أن المحتوى هو ملف YAML.

فراغ سطرين (سطرين فاضيين بعد `---`)

- هذا ليس شرط برمجي، لكن مطلوب هنا للتنسيق حسب التعليمات.

`hosts: all -`

- هذا تعريف الـ **Play**.
- `-` تعني بداية play جديد.
- `hosts: all` تعني: طبّق هذه المهمة على **جميع الأنظمة** التي تم تحديدها مسبقًا في ملف الـ inventory.

`become: true`

- هذا معناه: استخدم صلاحيات **sudo** لتنفيذ المهام.
- مفيد لما تحتاج تثبيت برامج أو تعمل مهام تتطلب صلاحيات الجذر (root).

`:tasks`

- هذا يعني: قائمة المهام التي راح تنفذ على الأجهزة.

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

`name: install apache2 package -`

- هذا **وصف المهمة**.
- الهدف هنا فقط توضيحي، يساعد في فهم ماذا تفعل المهمة عند التشغيل.

`apt:`

- هذا هو **الموديول (module)** الذي سيُستخدم.
- هنا استخدمنا `apt` لأنه خاص بتوزيعات **Debian/Ubuntu**.
- هو المسؤول عن إدارة الحزم.

`name: apache2`

- هذا هو اسم الحزمة التي نريد تثبيتها.
- يعني: استخدم `apt` لتثبيت برنامج `apache2` .

to run the yml file use the following command:

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

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

📌 شرح الأمر:

- `ansible-playbook` : هذا هو الأمر الرئيسي لتشغيل ملفات الـ playbook.
- `--ask-become-pass` : يخبر أنسيبل أنه يحتاج يسألك عن **كلمة مرور sudo** (لأن في playbook استخدمنا `become: true`).
- `install_apache.yml` : اسم ملف الـ playbook.

🖥️ عند التشغيل: أنسيبل يعطيك ملخص لكل Host (جهاز) فيه:

العنصر	معناه
ok	عدد المهام اللي اشتغلت **بنجاح** بدون مشاكل.
changed	عدد المهام اللي **عملت تغييرات** فعلًا على النظام (مثل تثبيت برنامج أو تعديل ملف).
unreachable	الأنظمة اللي أنسيبل **ما قدر يوصل لها** (مثلًا الجهاز مطفي أو الشبكة مقطوعة).
failed	عدد المهام اللي **فشلت** أثناء التنفيذ.
skipped	المهام اللي **تم تجاوزها** لأنها ما كانت تنطبق على الجهاز (مثال: شرط معين لم يتحقق).
rescued	عدد المهام اللي تم تنفيذها **لإنقاذ الوضع** بعد فشل مهمة سابقة. (ميزة في ansible تسمى rescue).
ignored	عدد المهام اللي **فشلت لكن تم تجاهل فشلها** لأن المستخدم اختار ذلك صراحةً (باستخدام ignore_errors: true).

⚠️ ملاحظة مهمة: ممكن يفشل الـ Playbook!

ليش ممكن يفشل التثبيت؟ 🤔

إذا ما حدثت الجهاز قبل التثبيت، ممكن يظهر خطأ مثل:

> "package not found"

السبب:

- في أنظمة لينكس، مدير الحزم (مثل `apt`) يحتاج إلى قائمة **حديثة** من روابط الحزم.
- هذه القائمة تتغير من وقت لآخر، وإذا ما حدثناها، راح يبحث `apt` عن روابط قديمة وما يلاقيها.

مثال عملي لنتيجة عشوائية للتوضيح:

```
PLAY [all]
*********************************************************************

TASK [Step 1: Update apt cache]
*********************************************
ok: [worker-node1]

TASK [Step 2: Install apache2]
**********************************************
changed: [worker-node1]

TASK [Step 3: Start apache2 service]
****************************************
ok: [worker-node1]

PLAY RECAP
*********************************************************************
worker-node1          : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

🧠 شرح كل وحدة في السطر الأخير:

الكلمة	المعنى
ok=3	ثلاث مهام اشتغلت بنجاح بدون أي مشاكل (مثل: تحديث apt وبدء الخدمة).
changed=1	وحدة من المهام سوّت **تغيير فعلي** على النظام (تثبيت apache).
unreachable=0	كل الأجهزة كانت متصلة ومافي أحد منها كان طافي أو غير متصل.
failed=0	ولا مهمة فشلت، كل شيء تمام الحمدلله ✅
skipped=0	مافي مهام تم تجاوزها، كل شيء انطبق على النظام.
rescued=0	ما اضطر أنسيبل يستخدم خطة إنقاذ لأن كل شيء مشى صح.
ignored=0	ما في مهام تم تجاهل فشلها، لأنه ما في فشل أصلاً.

💡

```
PLAY RECAP
*********************************************************************
worker-node1          : ok=0    changed=0    unreachable=1    failed=0
skipped=0    rescued=0    ignored=0
```

✨ لو فرضنا أن الجهاز كان مطفي، إيش يصير؟

Edit Copy

```
PLAY RECAP *********************************************************************
worker-node1           : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescu
```

📌 معنى هذا:

- `unreachable=1` : أنسيبل ما قدر يوصل للجهاز، يمكن الجهاز مطفي أو الـ SSH ما يشتغل.

معلومة اضافية لك:

```
💡 - name: Show message
   debug:
     msg: "Apache was installed successfully"
```

💡 معلومة إضافية:

تقدرين تضيفين task `debug` تعرض متغيرات أو معلومات في النص، مثلاً:

Edit Copy

```yaml
- name: Show message
  debug:
    msg: "Apache was installed successfully"
```

لحتى نضمن انه عملنا تحديثات للبكج كاملين من خلال الانسبل فايل، بنكتب الامر الاضافي التالي داخل ملف الyaml:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

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

المهمة الأولى:

Edit Copy

```yaml
- name: update repository index
  apt:
    update_cache: yes
```

🔁 مثل ما تسوين في الطرفية `apt update`، هذه تسوي نفس الشيء.

المهمة الثانية:

Edit Copy

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

📦 تثبت برنامج `apache2` (اللي هو خادم الويب).

📊 عند تشغيل الـ playbook تطلع لك الإحصائيات:

مثال:

```
ok=3    changed=2    unreachable=0    failed=0    skipped=0
```

معناها:

- `ok=3` : ثلاث مهام نفذت بنجاح (gathering facts + update repo + install apache).
- `changed=2` : مهمتين سوّت تغيير (update repo + install apache).
- `gathering facts` : أنسيبل يجمع معلومات عن النظام (RAM, OS version, IPs...الخ)، بس ما يغير شيء.

لو شغلته مرة ثانية:

- `update repository` **راح يسوي تغيير** من جديد (لأنه يحدّث القائمة دايمًا).
- `apache2` خلاص موجود، ما راح يعيد تثبيته.
- بيصير مثلًا: `ok=3 changed=1`

✨ تطوير الـ playbook: نثبت أكثر من حزمة

نكتب:

```yaml
---
- hosts: all
  become: true
  tasks:
    - name: install Apache2 and PHP packages
      apt:
        name:
          - apache2
          - libapache2-mod-php
        update_cache: yes
```

الفرق هنا:

- `:name` صارت قائمة (List) فيها أكثر من باكيج.
- `libapache2-mod-php` : هذا يخلّي Apache يقدر يشغّل ملفات PHP.

🧠 ملاحظة مهمة:

> الحين هذا الـ playbook يثبت الحزم **إذا ما كانت مثبتة فقط**، لكنه **ما يحدثها** لو فيه إصدار جديد.

```
---
 - 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.

فائدة `state: latest`:

🔄 هذه تخلي أنسيبل:

- يثبت الباكيجات إذا مو موجودة ✅
- أو **يحدثها** للإصدار الأخير إذا كانت موجودة ولكن قديمة 🔄

Playbook that removes these packages

touch remove_apache.yml

and write these commands in the file:

- - -

- 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.

🔍 شرح:

- `hosts: all` : يشمل كل السيرفرات اللي محددة في inventory.
- `become: true` : يشغل الأوامر بصلاحيات root.
- `apt` : الموديول الخاص بإدارة البكجات على Debian/Ubuntu.
- `name` : أسماء البكجات المراد حذفها.
- `state: absent` : معناها "إذا كان البكج موجود، احذفه".

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

🎯 اللي بيصير:

- أنسيبل بيروح لكل سيرفر في الـ inventory.
- بيشيك إذا فيه apache2 أو libapache2-mod-php.
- إذا موجودين، بيحذفهم.

🌐 الآن جرّبي تفتحي موقع السيرفر في المتصفح

بعد الحذف، لما تدخلين على عنوان IP حق السيرفر في المتصفح، راح **ما يشتغل الموقع** لأن Apache2 انحذف.

⬇️ إعادة التثبيت (Install Playbook)

ارجعي شغّلي playbook التثبيت:

```
ansible-playbook install_apache.yml
```

- Apache و PHP بيرجعوا ينثبتوا.
- الحين لو سويتِ **تحديث للصفحة في المتصفح**، بيرجع يشتغل الموقع.

so we need to add both these files to gihub

nislab$ git status
nislab$git add .

🧩 2. إضافة الملفات إلى Git:

```
git add .
```

النقطة `.` معناها "أضف كل الملفات اللي تغيّرت أو انضافت".

nislab$git commit -m “install/remove apache and php playbooks created by admin 1”
nilab$ git push origin main

⬆️ 4. رفع الملفات على GitHub:

```
git push origin main
```

- `origin` هو اسم الريموت (GitHub).
- `main` هو اسم الفرع اللي ترفعين عليه الملفات.

now do a git pull on workstation2 to get these files downloaded.

The 'when' Conditional

أولًا: الفكرة العامة

أنت عندك **Playbook** في Ansible وظيفته تثبّت Apache وPHP على مجموعة من السيرفرات.

المشكلة إنك تستخدم أمر `apt` وهذا خاص بتوزيعات Debian وUbuntu فقط.
لكن في حال كان عندك سيرفرات مختلفة مثل **AlmaLinux** (المبني على Red Hat)، فهنا راح يفشل الـ Playbook لأنه AlmaLinux ما يستخدم `apt` ، بل يستخدم `dnf` .

sudo nano inventory

add the new ip of the new server:

192.168.88.213 ⟶ the ip of almalinux server

ثالثًا: تشغيل الـ Playbook الحالي

لو شغلت الـ playbook بدون أي تعديل، راح يشتغل أمر `apt` على كل السيرفرات، حتى سيرفر AlmaLinux، وراح يعطيك خطأ مثل:

```
FAILED! => {"msg": "The apt module is not supported on this system"}
```

لأن AlmaLinux ما يعرف `apt`.

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"
```

شرح كل سطر:

- `hosts: all` : يعني هذا الـ Playbook راح يشتغل على كل السيرفرات الموجودة في الـ inventory.
- `become: true` : استخدم صلاحيات الـ root.
- `tasks` : قائمة بالمهام اللي راح تنفذها.
- `apt:` : أمر لتثبيت الحزم على توزيعات Debian/Ubuntu.
- `"when: ansible_distribution == "Ubuntu` : هذا الشرط يمنع تنفيذ المهمة إلا إذا كان النظام Ubuntu.

> وبالتالي، سيرفر AlmaLinux راح يتخطى هذه المهمة (skipped).

خامسًا: معرفة نوع التوزيعة لكل سيرفر

قبل ما تكتب شروط `when`، لازم تعرف التوزيعة اللي يستخدمها كل سيرفر.

استخدم هذا الأمر:

```
ansible all -m gather_facts | grep ansible_distribution
```

راح يعطيك مثل:

```
"ansible_distribution": "Ubuntu"
"ansible_distribution": "AlmaLinux"
```

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"
```

الفرق:

- `apt` خاص بـ Ubuntu.
- `dnf` خاص بـ AlmaLinux (وكل التوزيعات المبنية على RHEL مثل CentOS وRocky).

سابعًا: تشغيل الـ Playbook وملاحظة النتائج

شغّل الـ Playbook:

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

راح تلاحظ في الإخراج:

- أنه في سيرفر Ubuntu: يتم تثبيت `apache2` و `libapache2-mod-php` .
- وفي سيرفر AlmaLinux: يتم تثبيت `httpd` و `php` .
- ما راح يطلع أي خطأ لأن كل نظام حصل المهام اللي تناسبه فقط.

Reflection

What is meant by automation?

Automation means using technology to perform tasks **automatically** without needing human intervention every time

في مجال الشبكات والسيرفرات، automation تساعدنا نثبت برامج، نحدث الأنظمة، ننسخ ملفات، أو نغير إعدادات بشكل تلقائي وسريع على عدة أجهزة في نفس الوقت، بدل ما نعمل كل شيء يدويًا.

Is SSH necessary for ansible to operate?

نعم، **SSH ضروري** لتشغيل Ansible على الأنظمة.
Ansible يستخدم SSH حتى يتصل بالسيرفرات المستهدفة ويُرسل لها الأوامر وينفذها عن بعد.

Can we use ansible without SSH keys?

نعم، ممكن، لكن مش مستحسن.
بدل SSH keys، ممكن تستخدم **كلمة مرور (password authentication)**، لكن كل مرة تحتاج تدخل كلمة المرور يدويًا، وهذا يضيع وقت.
SSH keys توفر أمان وسرعة أكبر لأنها **توفر اتصال تلقائي بدون إدخال كلمة مرور** كل مرة.

Why did we use SSH keys?

استخدمنا SSH keys لأن:

- تتيح لنا الاتصال بدون كتابة كلمة مرور كل مرة.
- أكثر أمانًا من استخدام كلمات المرور.
- أسهل لما نتعامل مع سيرفرات متعددة.
- ضرورية لتشغيل Ansible playbooks بسلاسة.

What is scp used for?

`scp` تعني **Secure Copy**، وهي أداة تستخدم لنسخ الملفات بين جهازين باستخدام بروتوكول SSH.
مثلاً: تقدر تنسخ ملف من جهازك إلى سيرفر بعيد بأمان باستخدام `scp` .

What is meant by version control? And why is it important in Automation?

Version control يعني تتبع التعديلات التي تتم على ملفات المشروع مثل Playbooks و Scripts.
أشهر أدوات version control: Git.

أهميته في Automation:

- نقدر نرجع لأي إصدار سابق من ملف.
- نسجل من عمل ماذا ومتى.
- يسهل التعاون بين أكثر من شخص.
- يمنع ضياع التعديلات أو التخريب الغير مقصود.

What is an inventory file?

الـ **Inventory file** هو ملف فيه قائمة السيرفرات التي نريد أن نتحكم بها باستخدام Ansible.
مثلاً:

Edit Copy

```
[webservers]
192.168.1.10
192.168.1.11
```

ممكن نرتب السيرفرات حسب نوعها أو دورها (web, db, etc).

What do we mean by the term ad-hoc commands?

Ad-hoc commands هي أوامر قصيرة تُنفذ مباشرة على السيرفرات من خلال Ansible بدون الحاجة لكتابة Playbook.

مثال:

```
ansible all -m ping
```

أو:

```
ansible webservers -m shell -a "uptime"
```

What is a playbook?

Playbook هو ملف مكتوب بلغة YAML، يحتوي على مجموعة من المهام (tasks) التي نريد تنفيذها على السيرفرات.
هو الطريقة الأساسية لاستخدام Ansible في إدارة الأنظمة بشكل منظم.

What is Yaml?

YAML هي اختصار لـ "YAML Ain't Markup Language"، وهي لغة بسيطة وسهلة القراءة تُستخدم في كتابة ملفات الإعداد مثل Playbooks.

شكلها يكون بهذا النمط:

```
- name: install apache
  apt:
    name: apache2
    state: latest
```

What is a package manager?

Package manager هو أداة تُستخدم لتثبيت، تحديث، وإزالة البرامج على الأنظمة.
أمثلة:

- في Ubuntu و Debian: نستخدم `apt` .
- في RHEL و AlmaLinux: نستخدم `dnf` أو `yum` .

What is an ansible module?

Ansible module هو مكون جاهز يُستخدم لتنفيذ مهمة معينة مثل:

- تثبيت برنامج (`apt` , `dnf`)
- إدارة خدمات (`service`)
- نسخ ملفات (`copy`)
- تنفيذ أوامر (`shell` , `command`)

كل مهمة في playbook تعتمد على module.

Students: Ali Khuffash & Wajd Abd Al-Hadee

Done.