

Project 9: Ansible

- **Project 9: Ansible**
 - Goal of this project
- **Research**
 - Infrastructure as Code (IaC)
 - Benefits of IaC
 - Ansible
 - Ansible ad hoc commands
 - Diagram of my Ansible architecture
- **Day 1 tasks**
 - Create EC2 instances for Ansible controller and first target node
 - Set up dependencies on the controller
 - Use other ad hoc commands
 - Do update and upgrade on target nodes using ad hoc commands
 - Command module method
 - Shell module method
 - APT module method
 - Consolidate ad hoc commands by copying a file to a target node
 - Create and run playbook to install nginx on target node
 - Create and run playbook to provision app VM
 - Create and run playbook to run app with PM2
 - **Blockers**
- **Day 2 tasks**
 - Create the database VM (another Ansible target node)
 - Create and run playbook to update and upgrade web and db groups of machines
 - Create and run playbook to install Mongo DB
 - Create and run playbooks to provision the app and database
 - Extension task: Create and run playbook to print facts gathered
 - **Blockers**
- What I learnt
- Benefits I personally saw from the project

Goal of this project

The goal of this project is to use Ansible and AWS to implement a 2-tier cloud deployment of the Sparta test app (which uses Node JS v20) and database.

Research

Infrastructure as Code (IaC)

- **IaC**: creating or configuring infrastructure via code, not GUIs
- **declarative IaC**: users define the desired end state/result, and the tool handles how to get there (e.g. Terraform)

- **imperative IaC**: users specify the exact steps taken (e.g. Ansible)
- **provisioning infrastructure**: creating and setting up IT infrastructure
- **idempotent**: when it doesn't matter how many times you run a script, the tool will make sure that the desired state will always be reached (e.g. running a script twice won't result in 2 results/different results)
- **configuration drift**:
 - when you have an ecosystem like multiple servers running under a load balancer, and all of the servers should be configured the same
 - if someone logs into one and modifies the setting on one, this results in configuration drift between the servers
 - this becomes an issue when, e.g., you have to migrate them all to the cloud and you have to work out which server has the ideal configuration
- two types of IaC tools:
 - **configuration management** tools: maintain and update config of IT infrastructure, e.g. Ansible
 - **orchestration tools**: manage lifecycle of infrastructure resources, e.g. Terraform
- you should use IaC when you need consistent environments e.g. whenever you need to provision VMs at scale
- **IaC tools**:
 - Terraform
 - Ansible
 - Azure Resource Manager
 - AWS CloudFormation

Benefits of IaC

- scalable
- quicker than manually creating/configuring resources
- repeatable (via idempotency)
- eliminate human error; IaC tools like Ansible can prevent configuration drift by specifying the exact configuration you want

Ansible

- **Ansible**: an open-source automation tool that allows you to configure IT infrastructure
- Ansible is installed on a **controller node**, which can then manage the configuration of remote **target nodes**
- Ansible is **agentless**, i.e. doesn't require you to install anything on the target nodes
- Ansible is written in Python
- **playbook**: a file that defines a set of Ansible tasks to be executed on remote hosts; written in YAML because it's easy to read and write ([example here](#))
- it uses **modules**
- Installing Ansible should create the `/etc/ansible/` directory, but it's normal for this not to be created in some environments, so it (and all the files within) can be manually created if not automatically done
- **Typical files in `/etc/ansible`**:

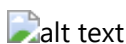
- **ansible.cfg** — configuration file, usually named this if using only one config file but doesn't strictly have to be
- **hosts** (i.e. inventory file) — file containing the host groups and hosts, usually named this if only using one hosts file but doesn't strictly have to be
- **Hosts/inventory file:**
 - usually *.ini* or *.yml* but I didn't specify an extension and it worked
 - you can have multiple hosts/inventory files (useful for different environments), and you can run a playbook on multiple hosts file by chaining this `-i <path to first hosts file> -i <path to second hosts file>` etc. after the `ansible-playbook` command
 - note that the playbook will only run once on any hosts common to both files as Ansible **de-duplicates** hosts by default
 - note that any **debug** (i.e. print outs of variables) in a single-playbook, multi-hosts file execution will only print the value for the last hosts file in the `ansible-playbook` statement (e.g. the last hostname in the order)
 - can include nested groups, i.e. hierarchies, e.g. `[production: children] webserver`s, which tells Ansible that *production* is a group with children groups (here, *webserver*s)
 - this means that any hosts in *webserver*s are also in *production*, so configs applied to/commands run on the *production* group are also applied to all hosts in all of its child groups (note that you can still run commands and apply configs only on the child groups by specifying them)
- **Gathering facts** in Ansible means collecting information about a host's system outputted as variables; these include information on their OS, RAM, storage, IP address, etc.
 - this is helpful when you need to check information about a host's system
 - gathering facts means you can run commands based on the value of these facts because they're stored as variables, e.g. running certain tasks on hosts when `ansible_os_family == "Ubuntu"`
 - you **wouldn't** gather facts if you didn't need to, as it makes the playbook take longer to run
- Why it's usually best to keep a separate playbook to run **update** & **upgrade** from the playbooks that deploy apps deployment:
 - it reduces the time taken to run the app deployment playbooks
 - updating and upgrading may change the versions of dependencies, which could break the app
- **Organisations that use Ansible:**
 - NASA
 - Qantas
 - US Bank
 - Hootsuite

Ansible ad hoc commands

- used on quick one-off tasks, i.e. tasks that you rarely repeat
- good for simple operations like checking system statuses, managing files, troubleshooting, or otherwise executing single commands across multiple servers

- general syntax is `ansible [hosts/group] -m [module to be used] -a "[module arguments]"`
- example commands ([see here for more detail](#)):
 - `ping` module — for testing connections between controller and target nodes
 - `command` module (default module used for ad hoc commands, so doesn't need to be specified with `-m command`)
 - `copy` module — for copying folders and files from the local machine to remote hosts
 - `file` module — manages files and directories on hosts
 - `debug` module — prints statements during the play's execution; often prints the value of variables to verify changes have been made
 - `setup` module — gathers facts (i.e. information) outputted as variables about target host(s), e.g. OS, RAM, storage, IP address, etc.
- **Advantages of default `command` module:**
 - quick and simple way of executing single shell commands at a time on a remote host
- **Disadvantages of default `command` module:**
 - doesn't support shell variables, command chaining, or operations like `|`, `>`, `<` (so you need to use `shell` module instead for these)

Diagram of my Ansible architecture



Day 1 tasks

Create EC2 instances for Ansible controller and first target node

1. Created my controller EC2 and first target node (i.e. the app EC2) and confirmed I could SSH into them



- **Controller EC2 settings:**
 - **Image:** Ubuntu 22.04 LTS
 - **Size:** t3.micro
 - **Security rules:** allow SSH
 - **Key pair:** my AWS key
- **App EC2 settings:**
 - same as above, except I allowed **SSH, HTTP, and port 3000**

Set up dependencies on the controller

1. SSHed into the machine and installed Ansible with:
 1. `sudo apt update && sudo apt upgrade`
 2. `sudo add-apt-repository --yes --update ppa:ansible/ansible`
 3. `sudo apt install ansible -y`

4. `sudo apt update && sudo apt upgrade` — I didn't do this initially, but it would have helped avoid the blocker I had given the Ansible version that was installed by default was 2.10.8 for some reason, which caused issues later on ([see here](#))

2. I had to manually create my `/etc/ansible` folder and the files within it as this wasn't done automatically

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ls
ansible.cfg  hosts  roles
```

in the install

3. Copied and pasted my `tech501-farah-aws-key.pem` (i.e. private SSH key) to the SSH folder on my controller EC2 and set permissions to read-only for Owner with `sudo chmod 400 <key name>`

```
ubuntu@ip-172-31-60-237:~/.ssh$ ls -l
total 8
-rw----- 1 ubuntu ubuntu 403 Feb 27 10:56 authorized_keys
-r----- 1 ubuntu ubuntu 1679 Feb 27 11:18 tech501-farah-aws-key.pem
```

4. I then SSHed into the target node from the controller with `ssh -i "tech501-farah-aws-key.pem"`

```
ubuntu@ec2-34-245-195-254.eu-west-1.compute.amazonaws.com
ubuntu@ip-172-31-60-237:~/.ssh$ ssh -i "tech501-farah-aws-key.pem" ubuntu@ec2-34-245-195-254.eu-west-1.compute.amazonaws.com
The authenticity of host 'ec2-34-245-195-254.eu-west-1.compute.amazonaws.com (172.31.63.26)' can't be established.
ED25519 key fingerprint is SHA256:aPleka0+0xz1+S8l2U3Dw+ys7K4Mn45mdcbR5SpevpM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-245-195-254.eu-west-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

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

System information as of Thu Feb 27 11:19:41 UTC 2025

System load: 0.0               Processes: 104
Usage of /:  22.1% of 7.57GB    Users logged in: 1
Memory usage: 26%              IPv4 address for ens5: 172.31.63.26
Swap usage:  0%

Expanded Security Maintenance for Applications is not enabled.

Updates can be applied immediately.

To enable ESM Apps to receive additional future security updates.
see https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

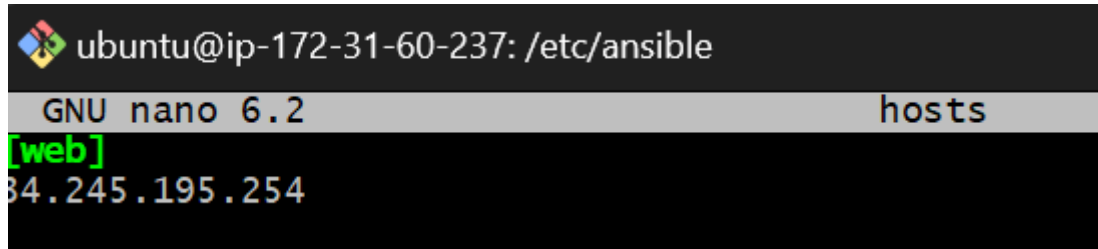
Last login: Thu Feb 27 10:59:21 2025 from 80.189.61.81
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-63-26:~$
```

5. In a new terminal window, I SSHed into the controller and tried to ping the app EC2 with `ansible all -m ping` and got the expected error as the `hosts` file is currently empty

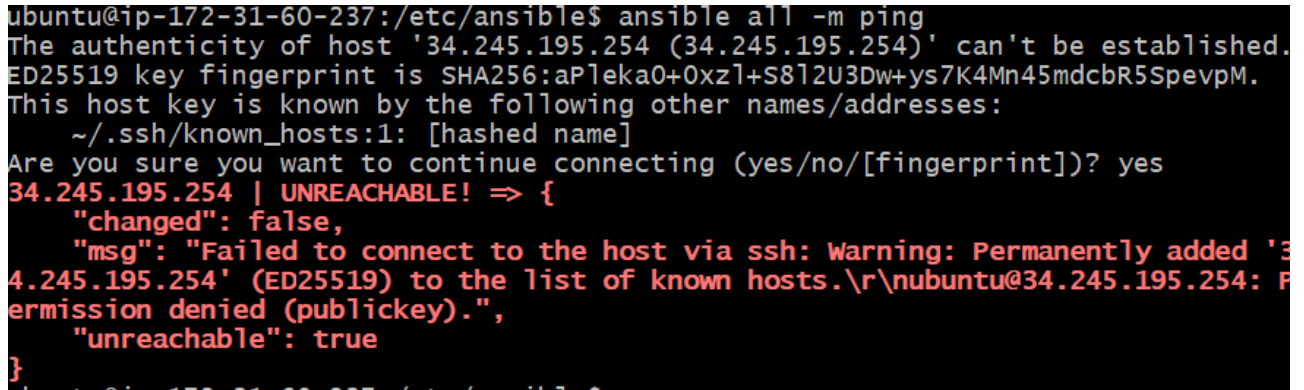
```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible all -m ping
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
```

- Edited the `hosts` file to add the public IP of the target node EC2 instance



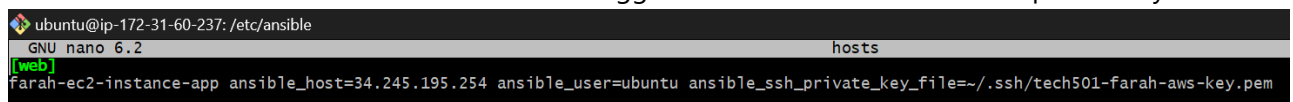
```
ubuntu@ip-172-31-60-237: /etc/ansible
GNU nano 6.2 hosts
[web]
34.245.195.254
```

- Ran the `ping` command again and got this expected new error



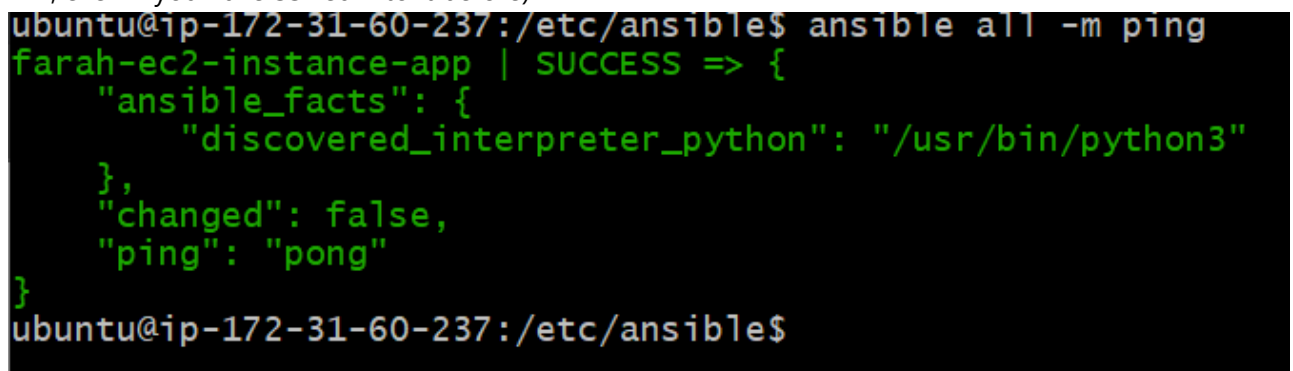
```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible all -m ping
The authenticity of host '34.245.195.254 (34.245.195.254)' can't be established.
ED25519 key fingerprint is SHA256:aPleka0+0xz1+S812U3Dw+ys7K4Mn45mdcbR5SpevpM.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
34.245.195.254 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: Warning: Permanently added '34.245.195.254' (ED25519) to the list of known hosts.\r\nubuntu@34.245.195.254: Permission denied (publickey).",
  "unreachable": true
}
```

- Edited the `hosts` file to add the username to be logged into and the location of the private key file



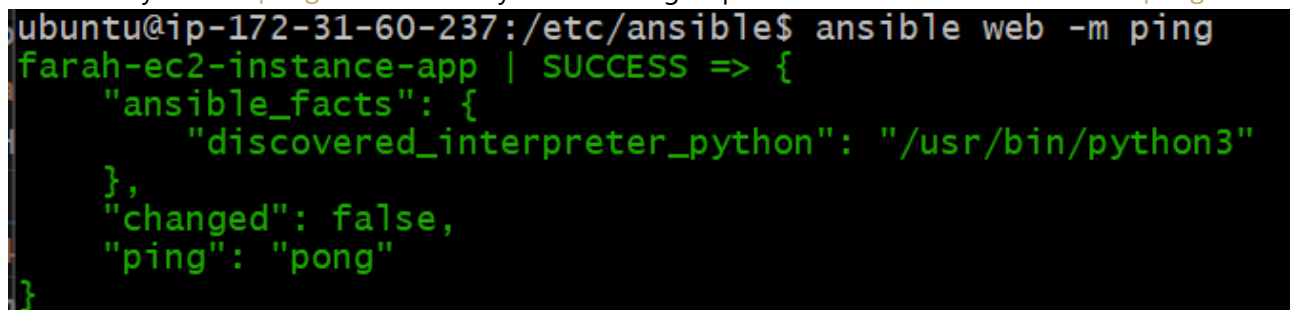
```
ubuntu@ip-172-31-60-237: /etc/ansible
GNU nano 6.2 hosts
[web]
farah-ec2-instance-app ansible_host=34.245.195.254 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/tech501-farah-aws-key.pem
```

- Ran the `ping` command again, this time successfully (note that you must say `yes` when first pinging a VM, even if you have SSHed into it before)



```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible all -m ping
farah-ec2-instance-app | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ubuntu@ip-172-31-60-237:/etc/ansible$
```

- Successfully ran the `ping` command only on the `web` group of hosts with `ansible web -m ping`



```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible web -m ping
farah-ec2-instance-app | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Use other ad hoc commands

- Ran an `ansible` command to get details on the Linux version used in my web group with `ansible web -a "lsb_release -a"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible web -a "lsb_release -a"
farah-ec2-instance-app | CHANGED | rc=0 >>
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.5 LTS
Release:       22.04
Codename:       jammyNo LSB modules are available.
```

- Ran an `ansible` command to get the date on the Linux version used in my web group with `ansible web -a "date"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible web -a "date"
farah-ec2-instance-app | CHANGED | rc=0 >>
Thu Feb 27 11:47:20 UTC 2025
```

Do update and upgrade on target nodes using ad hoc commands

Command module method

1. Updating my target node (named *farah-ec2-instance-app*) with `ansible farah-ec2-instance-app -b -a "apt update"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible farah-ec2-instance-app -b -a "apt update"
farah-ec2-instance-app | CHANGED | rc=0 >>
Hit:1 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:5 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
```

- note the use of `-b` to become a super-user and avoid having to use `sudo` in my commands

2. Upgrading my target node with `ansible farah-ec2-instance-app -b -a "apt upgrade -y"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible farah-ec2-instance-app -b -a "apt upgrade -y"
farah-ec2-instance-app | CHANGED | rc=0 >>
Reading package lists...
Building dependency tree...
Reading state information...
Calculating upgrade...
```

- **Downsides of this method:**

- designed to run one command at a time, so doesn't support chaining (which is why I had to run the commands separately) or piping because it doesn't invoke the shell

Shell module method

1. Updating and upgrading the target node with `ansible farah-ec2-instance-app -b -m shell -a "apt update && apt upgrade -y"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible farah-ec2-instance-app -b -m shell -a "apt update && apt upgrade -y"
farah-ec2-instance-app | CHANGED | rc=0 >>
Hit:1 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
```

- note the use of `-m shell` to specify the `shell` module, i.e. so we don't use the default `command` module
- note that the `shell` module **can** handle command chaining
- **Downsides of this method:**
 - can be less predictable because it's reliant on the target system's shell environment
 - less secure, because it's susceptible to command/shell injections

APT module method

1. Updating and upgrading the target node with the **idempotent** `apt` module: `ansible farah-ec2-instance-app -m apt -a "update_cache=yes upgrade=dist" -b"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible farah-ec2-instance-app -m apt -a "update_cache=yes upgrade=dist" -b
farah-ec2-instance-app | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "msg": "Reading package lists...\nBuilding dependency tree...\nReading state information...\nCalculating upgrade...\nThe following packages have been kept back:\n  dmeventd dmsetup libdevmapper-event1.02.1 libdevmapper1.02.1 liblvm2cmd2.03\n  lvm2 pollinate\n0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.\n",
  "stderr": "",
  "stderr_lines": [],
  "stdout": "Reading package lists...\nBuilding dependency tree...\nReading state information...\nCalculating upgrade...\nThe following packages have been kept back:\n  dmeventd dmsetup libdevmapper-event1.02.1 libdevmapper1.02.1 liblvm2cmd2.03\n  lvm2 pollinate\n0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.\n",
  "stdout_lines": [
    "Reading package lists...",
    "Building dependency tree...",
    "Reading state information...",
    "Calculating upgrade..."
  ]
}
```

Consolidate ad hoc commands by copying a file to a target node

- Using an ad hoc command (via the `copy` module) to copy my private SSH key from my controller node to the target node with `ansible farah-ec2-instance-app -m copy -a "src=~/.ssh/tech501-farah-aws-key.pem dest=/home/ubuntu/.ssh/tech501-farah-aws-key.pem mode=0400 owner=ubuntu group=ubuntu"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible farah-ec2-instance-app -m copy -a "src=~/.ssh/tech501-farah-aws-key.pem dest=/home/ubuntu/.ssh/tech501-farah-aws-key.pem mode=0400 owner=ubuntu group=ubuntu"
farah-ec2-instance-app | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "checksum": "98b54cdb624c9a71bcd58414a4b79221e3496b52",
  "dest": "/home/ubuntu/.ssh/tech501-farah-aws-key.pem",
  "gid": 1000,
  "group": "ubuntu",
  "md5sum": "e00c8bbb067d62edb8f32cea98eefd53",
  "mode": "0400",
  "owner": "ubuntu",
  "size": 1679,
  "src": "/home/ubuntu/.ansible/tmp/ansible-tmp-1740658127.857253-3238-258604952787369/source",
  "state": "file",
  "uid": 1000
}

ubuntu@ip-172-31-63-26:~/.ssh$ ls
authorized_keys  tech501-farah-aws-key.pem
```

Create and run playbook to install nginx on target node

1. Created an Ansible playbook to install Nginx named `install_nginx.yml` without using the `command` or `shell` modules
2. Before running the playbook, I checked its syntax with `ansible-playbook install_nginx.yml --syntax-check` and it was fine

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook install_nginx.yml --syntax-check
playbook: install_nginx.yml
```


- Successfully ran the playbook with `ansible-playbook install_nginx.yml`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook install_nginx.yml

PLAY [Install nginx play] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-app]

TASK [Update apt cache] *****
ok: [farah-ec2-instance-app]

TASK [Upgrade all packages] *****
ok: [farah-ec2-instance-app]

TASK [Install and configure nginx] *****
changed: [farah-ec2-instance-app]

PLAY RECAP *****
farah-ec2-instance-app : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ubuntu@ip-172-31-60-237:/etc/ansible$
```

Create and run playbook to provision app VM

- Created a playbook that uses non-`command` modules to be run on the app EC2 that installs NodeJS, Git clones the app folder, and runs the app in the foreground named `prov_app_with_npm_start.yml`
- Before running this playbook, I checked its syntax with `ansible-playbook prov_app_with_npm_start.yml --syntax-check`
- Successfully ran it:

```
PLAY [Setup Node.js 20 and Install App Dependencies] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-app]

TASK [Install Node.js and npm] *****
ok: [farah-ec2-instance-app]

TASK [Update apt cache] *****
ok: [farah-ec2-instance-app]

TASK [Upgrade all packages] *****
ok: [farah-ec2-instance-app]

TASK [Remove existing app directory if it exists] *****
ok: [farah-ec2-instance-app]

TASK [Clone the repository (dev branch)] *****
changed: [farah-ec2-instance-app]

TASK [Navigate to the app folder and install dependencies and start the app] *****
^C [ERROR]: User interrupted execution
```

4. Results:



Welcome to the Sparta Test App



The app is running correctly.

Testing entire pipeline on my own Jenkins server using rsync.

- Note that running this playbook will always hang because `npm` is running in the foreground

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook prov_app_with_npm_start.yml

PLAY [Setup Node.js 20 and Install App Dependencies] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-app]

TASK [Install Node.js and npm] *****
ok: [farah-ec2-instance-app]

TASK [Update apt cache] *****
ok: [farah-ec2-instance-app]

TASK [Upgrade all packages] *****
ok: [farah-ec2-instance-app]

TASK [Navigate to the app folder and install dependencies and start the app] *****
```

Create and run playbook to run app with PM2

- I created another playbook (named `prov_app_with_pm2.yml`) to run the app in the background with PM2 by duplicating the above playbook and then modifying it so that it installed PM2 globally and ran

the app via PM2

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook prov_app_with_npm_start.yml

PLAY [Setup Node.js 20 and Install App Dependencies] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-app]

TASK [Install Node.js and npm] *****
ok: [farah-ec2-instance-app]

TASK [Update apt cache] *****
ok: [farah-ec2-instance-app]

TASK [Upgrade all packages] *****
ok: [farah-ec2-instance-app]

TASK [Install PM2 globally] *****
ok: [farah-ec2-instance-app]

TASK [Remove existing app directory if it exists] *****
ok: [farah-ec2-instance-app]

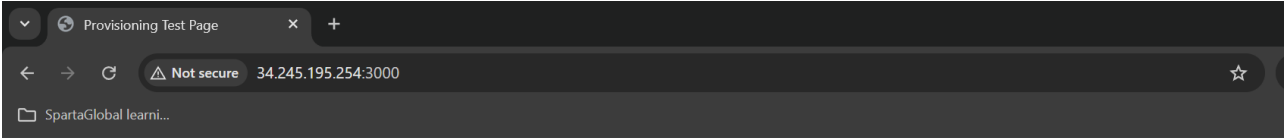
TASK [Clone the repository (dev branch)] *****
ok: [farah-ec2-instance-app]

TASK [Navigate to the app folder and install dependencies and start the app] *****
changed: [farah-ec2-instance-app]

PLAY RECAP *****
farah-ec2-instance-app      : ok=8    changed=1    unreachable=0    failed=0    skipped=0    res
cued=0    ignored=0

ubuntu@ip-172-31-60-237:/etc/ansible$
```

2. Results:



Welcome to the Sparta Test App



The app is running correctly.

Testing entire pipeline on my own Jenkins server using rsync.

3. I verified this by logging into the target node and running `sudo su` and `pm2 list` to show that the app was running on the root user, as expected because of my playbook's `become` settings

```
ubuntu@ip-172-31-63-26:~$ sudo su
root@ip-172-31-63-26:/home/ubuntu# pm2 list
```

id	name	mode	u	status	cpu	memory
0	app	fork	0	stopped	0%	0b
1	app	fork	0	online	0%	63.6mb

Blockers

1. While completing the last two tasks, I kept getting an error because the versions of Nodejs and NPM that were being installed were incompatible, so I added tasks into these playbooks to ensure that all packages were updated and upgraded after Nodejs and NPM were installed, which resolved the issue
2. I sometimes ran into issues with the tasks after restarting my EC2s because I hadn't changed the target node IP addresses in the *hosts* file, which was quickly resolved

Day 2 tasks

Create the database VM (another Ansible target node)

1. Created an EC2 instance for the database (DB) with the usual settings and ports 22 and 27017 open to all sources
2. Ensured I could SSH into it from the controller EC2 with `ssh -i "tech501-farah-aws-key.pem"`

`ubuntu@ec2-3-252-100-157.eu-west-1.compute.amazonaws.com`

```
ubuntu@ip-172-31-60-237:~/.ssh$ ssh -i "tech501-farah-aws-key.pem" ubuntu@ec2-3-252-100-157.eu-west-1.compute.amazonaws.com
The authenticity of host 'ec2-3-252-100-157.eu-west-1.compute.amazonaws.com (3.252.100.157)' can't be established.
ED25519 key fingerprint is SHA256:bcZLp/Ptf+lGrFJgGf5pBG1GN+HqvIKjJj3p2zU8cTo.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-252-100-157.eu-west-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1021-aws x86_64)
```

3. I edited the hosts file to add a group called `db` for the DB EC2

```
GNU nano 6.2                                hosts
[web]
farah-ec2-instance-tf-app ansible_host=34.244.24.3 ansible_user=ubuntu ansible_>
[db]
farah-ec2-instance-tf-db ansible_host=3.255.218.140 ansible_user=ubuntu ansible_>
```

4. Pinged this group to manually accept the first connection with `ansible db -m ping`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible db -m ping
The authenticity of host '3.252.100.157 (3.252.100.157)' can't be established.
ED25519 key fingerprint is SHA256:bcZLp/Ptf+lGrFJgGf5pBG1GN+HqvIKjJj3p2zU8cTo.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:5: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
farah-ec2-instance-db | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

- Then pinged all hosts `ansible all -m ping`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible all -m ping
farah-ec2-instance-app | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
farah-ec2-instance-db | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Create and run playbook to update and upgrade web and db groups of machines

- Created a playbook named `update_upgrade_all.yml` to update and upgrade my `web` and `db` hosts without using `command` or `shell` modules
- Before running it, I checked its syntax with `ansible-playbook update_upgrade_all.yml --syntax-check`
- I successfully ran it

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook update_upgrade_all.yml

PLAY [Updating and upgrading app and db EC2s] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-db]
ok: [farah-ec2-instance-app]

TASK [Update apt cache] *****
ok: [farah-ec2-instance-app]
changed: [farah-ec2-instance-db]

TASK [Upgrade all packages] *****
ok: [farah-ec2-instance-app]
changed: [farah-ec2-instance-db]

PLAY RECAP *****
farah-ec2-instance-app    : ok=3    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
farah-ec2-instance-db    : ok=3    changed=2    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Create and run playbook to install Mongo DB

- I created a playbook to install MongoDB on my `db` hosts named `install-mongodb.yml`
- I ran it successfully

```
PLAY RECAP *****
farah-ec2-instance-tf-db  : ok=10   changed=6    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Create and run playbooks to provision the app and database

1. I created a playbook named `prov-db.yml` to install MongoDB, update the BindIp, and then enable and restart MongoDB on my `db` hosts
2. I then ran an ad hoc command from the controller EC2 to check that MongoDB was running on my `db` hosts with `ansible db -m systemd -a "name=mongod state=started"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible db -m systemd -a "name=mongod state=started"
farah-ec2-instance-db | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "name": "mongod",
  "state": "started",
  "status": {
```

3. I also ran an ad hoc command from the controller EC2 to check that the BindIp was correctly configured on my `db` hosts with `ansible db -m command -a "grep 'bindIp' /etc/mongod.conf"`

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible db -m command -a "grep 'bindIp' /etc/mongod.conf"
farah-ec2-instance-db | CHANGED | rc=0 >>
  bindIp: 0.0.0.0
```

4. I tested that this configuration was successful by logging into my app EC2, navigating to the app folder, and running `pm2 kill`, `export DB_HOST=mongodb://172.31.49.226:27017/posts`, `printenv DB_HOST`, and `pm2 start app.js` and then navigating to the public IP of the app EC2; the (albeit unseeded at this point) `posts` page shows this was successful



Recent Posts

5. I then edited my `prov_app_with_pm2.yml` playbook to include an environment variable on the task that runs the app

```
- name: Navigate to the app folder and install dependencies and start the app
  environment:
    DB_HOST: "mongodb://54.171.112.116:27017/posts"
  shell: |
    cd /home/ubuntu/repo/nodejs20-sparta-test-app/app
    pm2 stop app.js || true
    npm install
    node seeds/seed.js
    pm2 start app.js
```

6. I then created a playbook named `prov-all.yml` which has two plays:

1. One to provision the hosts in the `db` group
 2. And one to provision the app on hosts in the `web` group
7. I also added tasks to:
1. Remove the existing app directory if it existed before cloning (though I did realise later that I could do this by adding a `force: yes` option to my Git clone task)

```
- name: Remove existing app directory if it exists
  file:
    path: "/home/ubuntu/app"
    state: absent # ensures the path is absent
```

2. Backup the default Nginx config file

```
- name: Backup the original Nginx config
  copy:
    src: /etc/nginx/sites-available/default
    dest: /etc/nginx/sites-available/default.bak
    remote_src: yes # ensures the file is copied from the target node, not the controller
```

3. Add a reverse proxy to my `prov_app_with_pm2` playbook and notify a handler to restart Nginx after this is done

```
- name: Replace try_files with proxy_pass in Nginx config
  replace:
    path: /etc/nginx/sites-available/default
    regexp: '^.*try_files\s+\$uri\s+\$uri/\s+=404;'
    replace: 'proxy_pass http://localhost:3000;'
  register: nginx_config_changed
  notify: Restart Nginx
```

```
handlers:
  - name: Restart Nginx
    service:
      name: nginx
      state: restarted
```


- **Results (note the lack of :3000 in the URL when running the app now):**

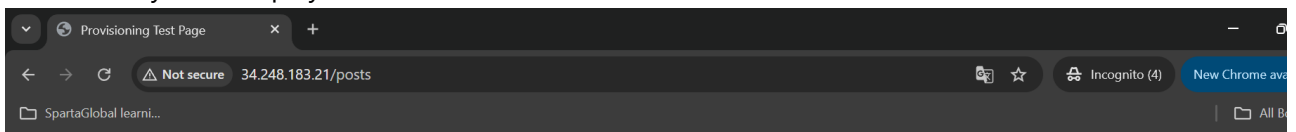


Recent Posts

Buckinghamshire Associate

Facilis voluptates culpa et unde. Vitae et in amet. Neque est recusandae est et libero deleniti qui est ab. Doloremque asperiores ut saepe occaecati voluptas porro et. Qui voluptas autem occaecati est recusandae delectus vel. Odit numquam et laborum. Eos ut quo. Laudantium aut dolor nisi. Placeat nostrum rerum preferendis. Aut quibusdam fugit. Aspernatur provident facilis omnis voluptatem. Sit praesentium aut. Eveniet ab illum non temporibus molestias consectetur aliquam hic. Corrupti ipsa veniam quam. Et accusamus a omnis velit magnam consectetur labore hic. Dolorem consectetur animi voluptatem voluptas.

- 8. I successfully ran this playbook



Recent Posts

Multi-tiered Manager Manager

Sed nisi sunt quia velit repellat quis aliquid. Atque consequatur accusamus fugiat et vitae officiis. Consectetur consequatur officia totam impedit et laboriosam. Id accusamus aut consectetur quis quos. Ut possimus illo iure at. Aut deleniti dolorum et quo voluptate ipsam. Sint ipsum voluptate qui veniam. Facere quibusdam et magni. Molestiae omnis saepe quis. Eaque voluptate iste quia velit est minus minima quidem omnis. Dolorem sit distinctio quasi cumque quae quas eligendi at similique. Necessitatibus quasi sequi magni cum blanditiis expedita exercitationem fugit.

- 9. I then tested that this playbook worked on new app and DB hosts
- 10. to practice my Terraform skills, I created these via Terraform [see here for files](#)
- 11. I then successfully ran the *prov-all.yml* playbook on them



Recent Posts

matrix

In est sapiente. Voluptas rerum repellat et. Iure mollitia corporis quidem et cumque ipsum rem illum saepe. Provident nobis deserunt dolores qui nihil rerum. Illo vero ut sapiente. Magni sed omnis id magnam consequatur eligendi omnis consequuntur et. Qui odit quis repellendus repellat nihil sint. Debitis possimus a eos sit. Tempora vitae impedit quam quo qui asperiores qui earum officia. Quasi enim corrupti libero adipisci voluptatem aperiam sint et. Est vitae qui. Veniam nihil eligendi.

- 12. I ran this playbook again to ensure its idempotency (note the new record, indicating posts page has been reseeded)



Recent Posts

Towels

Maxime esse unde maiores quia consectetur ex eveniet. Praesentium mollitia eius ad aliquid quidem voluptatibus. Atque tempore fuga eligendi rerum quia sit ipsa. Quam in similique beatae sit quia pariat delectus veniam vel. Aperiam blanditiis sit ut illum dolorem eligendi qui reiciendis. Ullam voluptate facilis aliquam esse a atque molestiae commodi qui. Optio nisi qui. Ea maxime minima sunt vitae aut tenetur error. Sed eos dolorum sint vitae optio magnam quia. Quis fugiat debitis perferendis id molestiae quos. Velit qui nisi. Harum laudantium consequatur iste pariat rerum aut ipsa alias. Accusamus quibusdam officia rem vel qui et voluptas esse. Impedit debitis doloremque illo qui corrupti nihil assumenda quia. Et dolores est nemo rerum labore aut ut dolorem. Est perferendis numquam voluptates rem enim dolor qui.

Extension task: Create and run playbook to print facts gathered

1. I created a playbook named [print-facts.yml](#) (with one play and one task) to gather all facts about all hosts and print them
2. I successfully ran the playbook, which included information on the hosts's OSes, IP addresses, and storage

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook print-facts.yml

PLAY [Print facts on all hosts] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-tf-app]
ok: [farah-ec2-instance-tf-db]

TASK [Print gathered facts on all hosts] *****
ok: [farah-ec2-instance-tf-app] => {
  "ansible_facts": {
    "all_ipv4_addresses": [
      "172.31.63.146"
    ]
  }
}
```

```
PLAY RECAP *****
farah-ec2-instance-tf-app : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
farah-ec2-instance-tf-db : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Blockers

1. I was getting these notifications when running my *install_mongodb* playbook so I edited it to remove all instances of `sudo` in any `shell` modules and use `become: yes` at the top of my play

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook install_mongodb.yml

PLAY [Install, configure, and enable MongoDB on db hosts] *****

TASK [Gathering Facts] *****
ok: [farah-ec2-instance-db]

TASK [Install gnupg and curl] *****
[WARNING]: Consider using 'become', 'become_method', and 'become_user' rather
than running sudo
changed: [farah-ec2-instance-db]

TASK [Download and add MongoDB GPG key] *****
[WARNING]: Consider using the get_url or uri module rather than running 'curl'.
If you need to use command because get_url or uri is insufficient you can add
'warn: false' to this command task or set 'command_warnings=False' in
ansible.cfg to get rid of this message.
changed: [farah-ec2-instance-db]
```

2. I had this error

```
TASK [Install PM2 globally] *****
fatal: [farah-ec2-instance-app]: FAILED! => {"msg": "Could not find imported mod
ule support code for ansible_collections.community.general.plugins.modules.npm.
Looked for (['ansible.module_utils.common.locale.get_best_parsable_locale', 'an
sible.module_utils.common.locale'])"}

```

- seemingly because of this

```
ubuntu@ip-172-31-60-237:/etc/ansible$ ansible-playbook prov_app_with_pm2.yml
[WARNING]: collection community.general does not support Ansible version 2.10.8
```

- so I upgraded my Ansible version using `sudo pip3 install --upgrade ansible` (I used `pip3` here just to try out this package manager) — my playbook then ran fine

3. I got this Python deprecation warning

```
adminuser@tech501-farah-tf-udemy-ansible-controller-vm:~/ssh$ ansible db -m ping
[WARNING]: Platform linux on host tech501-farah-tf-udemy-db-vm is using the
discovered Python interpreter at /usr/bin/python3.10, but future installation
of another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
```

so I silenced it by editing my `ansible.cfg` file like this (note that I later realised I could have added `interpreter_python=/usr/bin/python3` to this file instead to achieve the same result)

```
adminuser@tech501-farah-tf-udemy-ansible-controller-vm: /etc/ansible
GNU nano 6.2                               ansible.cfg *
Since Ansible 2.12 (core):
To generate an example config file (a "disabled" one with all
$ ansible-config init --disabled > ansible.cfg

Also you can now have a more complete file by including exist
ansible-config init --disabled -t all > ansible.cfg

For previous versions of Ansible you can check for examples i
Note that this file was always incomplete and lagging change
for example, for 2.9: https://github.com/ansible/ansible/blob
defaults]
nterpreter_python = auto_silent|
```

- This removed the warning

```
adminuser@tech501-farah-tf-udemy-ansible-controller-vm:/etc/ansible$ ansible db
-m ping
tech501-farah-tf-udemy-db-vm | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.10"
  },
  "changed": false,
  "ping": "pong"
}
```

4. My posts page originally wasn't seeded, which I later fixed by changing the order of my commands

What I learnt

- I learnt about how Ansible works, including ad hoc commands, playbooks, configuration files, and inventory/host files
- I also learnt about Ansible's modules, which allowed me to execute tasks in an idempotent way
- I learnt that indentation is very important to get right in my playbooks

Benefits I personally saw from the project

- I found it very useful to be able to run one playbook and have the entire 2-tier app up and running with a single command (once I'd set up my *prov-all.yml* playbook)
- Once I familiarised myself with Ansible I found its modules easier to understand at a glance than my Bash scripts
- I also really appreciated that Ansible allows you to avoid having to preface playbook commands with `sudo` thanks to the `become: yes` option