Automation

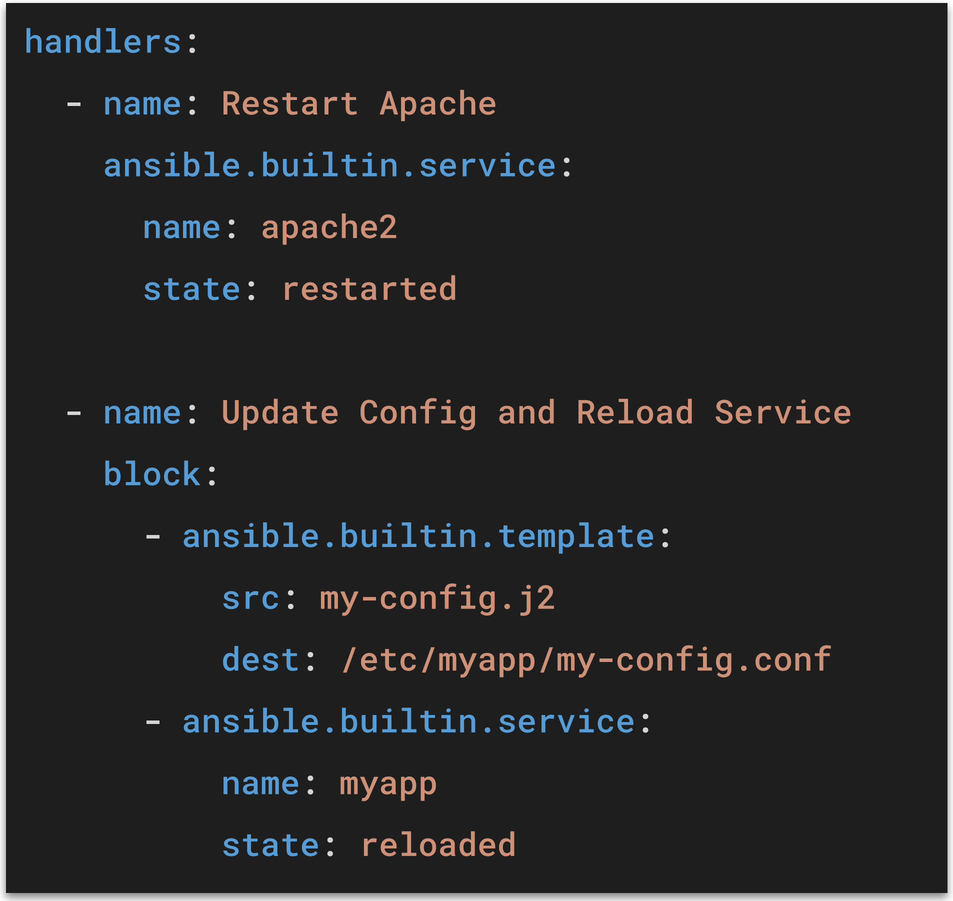
Ansible Handlers

DE HOGESCHOOL
MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Definition of handlers

- special types of tasks in Ansible playbooks that are triggered only when notified by other tasks.
- designed to perform specific actions in response to changes or events that occur during playbook execution.
- useful for managing services and configuration files
- help ensure idempotent and efficient automation processes.



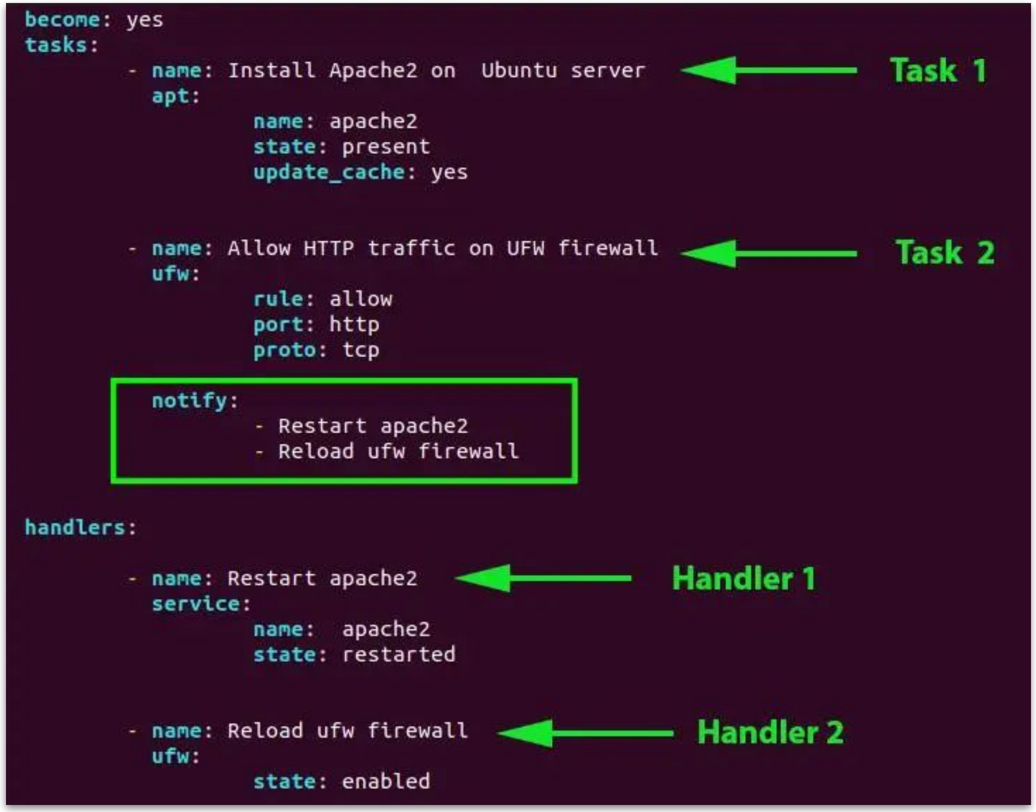
```yaml
handlers:
  - name: Restart Apache
    ansible.builtin.service:
      name: apache2
      state: restarted

  - name: Update Config and Reload Service
    block:
      - ansible.builtin.template:
          src: my-config.j2
          dest: /etc/myapp/my-config.conf
      - ansible.builtin.service:
          name: myapp
          state: reloaded
```

Purpose and primary use of handlers

- Restarting or reloading services when their configuration files have been updated.
- Applying system or application updates only when necessary or when triggered by other tasks.
- Managing dependencies between tasks, e.g. waiting for a service to be ready before continuing with the next task.
- Performing cleanup or rollback actions in case of errors or failures during playbook execution.

Relationship between handlers, tasks, and playbooks

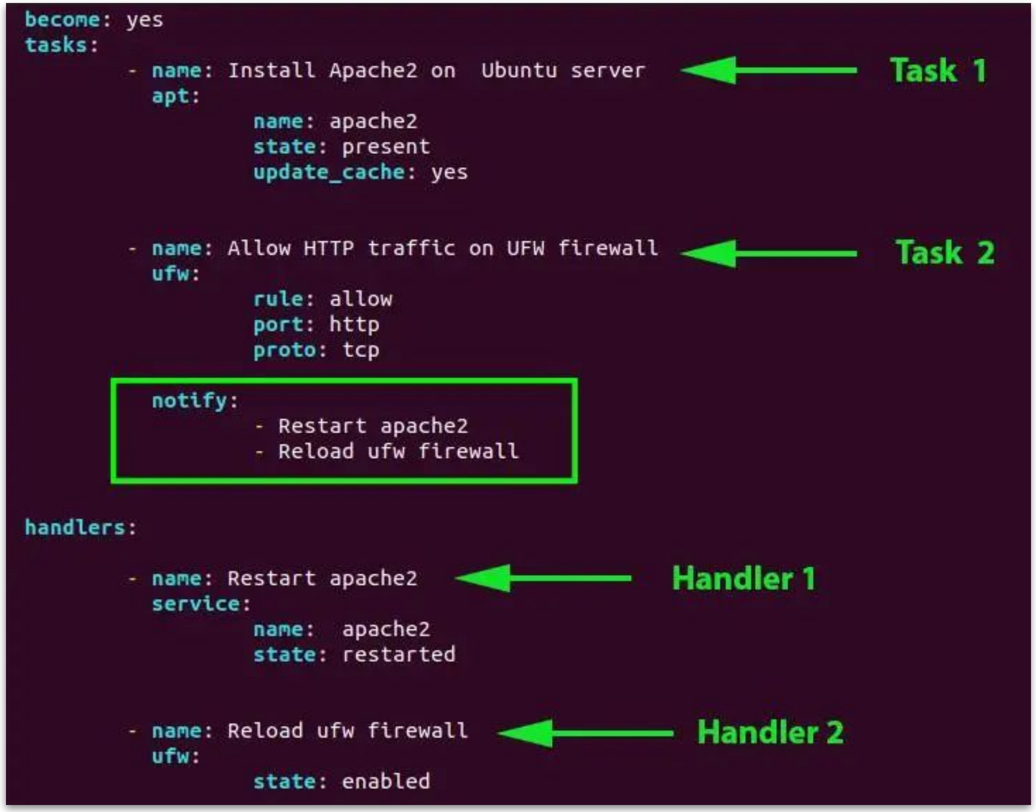
- Handlers are defined in the **`handlers:`** section of a playbook or role.
- Structurally similar to tasks, but only execute when triggered by a **`notify:`** directive.
- The notify directive is used to mark handlers for execution **if the task reports a change** (task's "`changed`" status is set to "`true`"). In that case the handler is added to a queue that will execute after all tasks have been completed.
- **Handlers only run once**, even if notified multiple times, ensuring efficiency and idempotence.



```
become: yes
tasks:
        - name: Install Apache2 on  Ubuntu server
          apt:
                  name: apache2
                  state: present
                  update_cache: yes

        - name: Allow HTTP traffic on UFW firewall
          ufw:
                  rule: allow
                  port: http
                  proto: tcp

          notify:
                  - Restart apache2
                  - Reload ufw firewall

handlers:

        - name: Restart apache2
          service:
                  name:  apache2
                  state: restarted

        - name: Reload ufw firewall
          ufw:
                  state: enabled
```

Differences between handlers and regular tasks

- Handlers are triggered by **`notify`** in tasks; regular tasks execute sequentially.
- Handlers run after all tasks, while tasks run in the order they're defined.
- Handlers execute once per playbook run, even with multiple notifications; tasks may execute multiple times.
- Handlers are ideal for actions in response to changes or events, like restarting services or applying updates.



```
become: yes
tasks:
        - name: Install Apache2 on  Ubuntu server
          apt:
                name: apache2
                state: present
                update_cache: yes

        - name: Allow HTTP traffic on UFW firewall
          ufw:
                rule: allow
                port: http
                proto: tcp

          notify:
                  - Restart apache2
                  - Reload ufw firewall

handlers:

        - name: Restart apache2
          service:
                name:  apache2
                state: restarted

        - name: Reload ufw firewall
          ufw:
                state: enabled
```

Benefits of using Handlers

- Improved efficiency:
  - Handlers run only when necessary, reducing time and resource usage
- Ensuring idempotence:
  - Handlers avoid redundant actions, maintaining playbook idempotence
- Better organization and modularity:
  - Separate conditional actions for cleaner playbooks
  - Reuse handlers across tasks and playbooks for scalability

Limitations of Handlers

- Execution order and timing:
    - Handlers run at the end, making control over order challenging
- **No loops or conditionals:**
    - Handlers don't support loop constructs or conditional statements directly
- Less granular control:
    - Handlers execute once per playbook run, even when notified multiple times
    - May not fit scenarios requiring multiple actions or responses to different events

Basic syntax

- defined within the **`handlers:`** section of a playbook or a role.
- follow a similar structure to tasks, with a **`name`** parameter describing the handler's purpose and an action that the handler should perform using an Ansible module.
- Common handler parameters:
  - **`name:`** A descriptive and unique name for the handler.
  - `[module_name]:` The Ansible module that the handler will use to perform its action (e.g., `ansible.builtin.command:`, `ansible.builtin.service:`, …).
  - `[parameterX]:` Module-specific parameters required to execute the desired action.

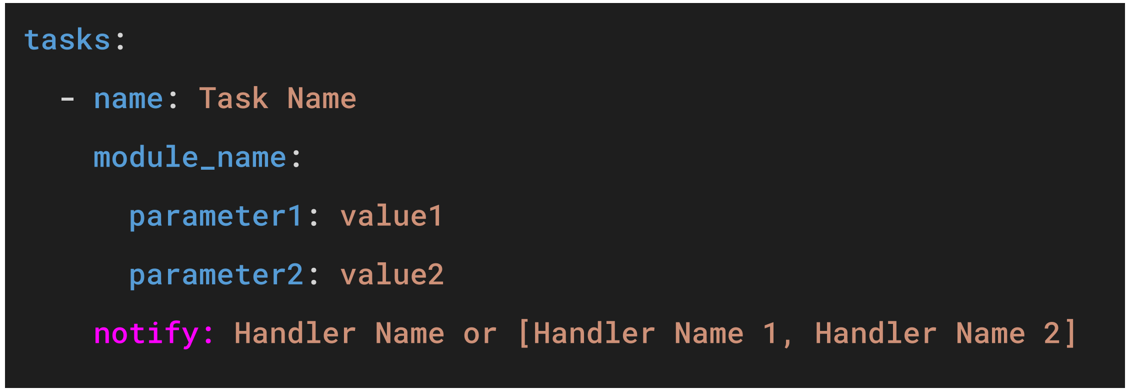
```
handlers:
  - name: Handler Name
    module_name:
      parameter1: value1
      parameter2: value2
```

```
handlers:
  - name: Restart Nginx
    ansible.builtin.service:
      name: nginx
      state: restarted
```

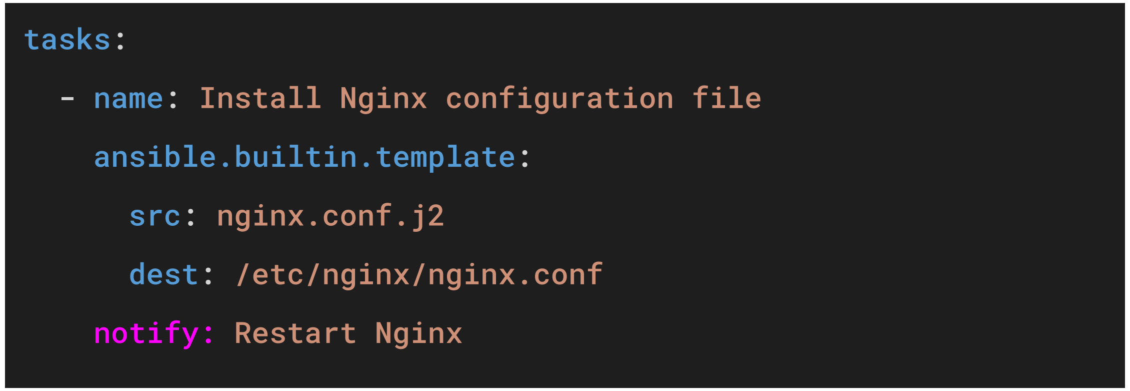
desired state

Notifying handlers from tasks

- The `notify:` directive is used within a task definition to specify the handler(s) that should be triggered **if the task reports a change** (i.e., if the task's result has a "`changed`" status set to "`true`").
- The directive can accept a single handler name or a list of handler names.



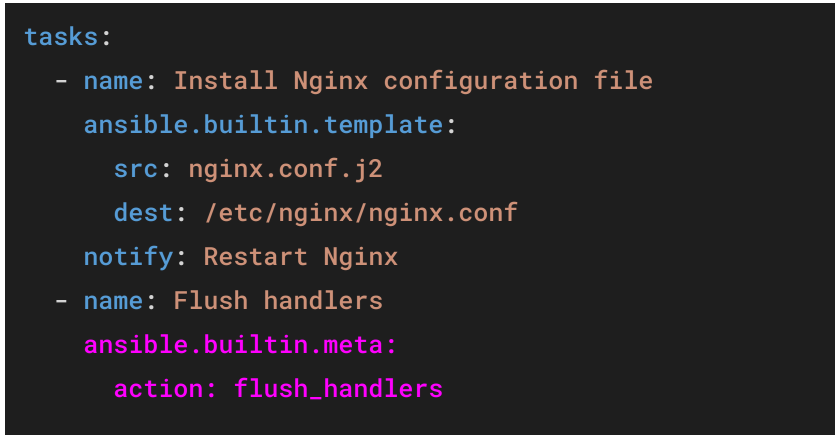
```yaml
tasks:
  - name: Task Name
    module_name:
      parameter1: value1
      parameter2: value2
    notify: Handler Name or [Handler Name 1, Handler Name 2]
```



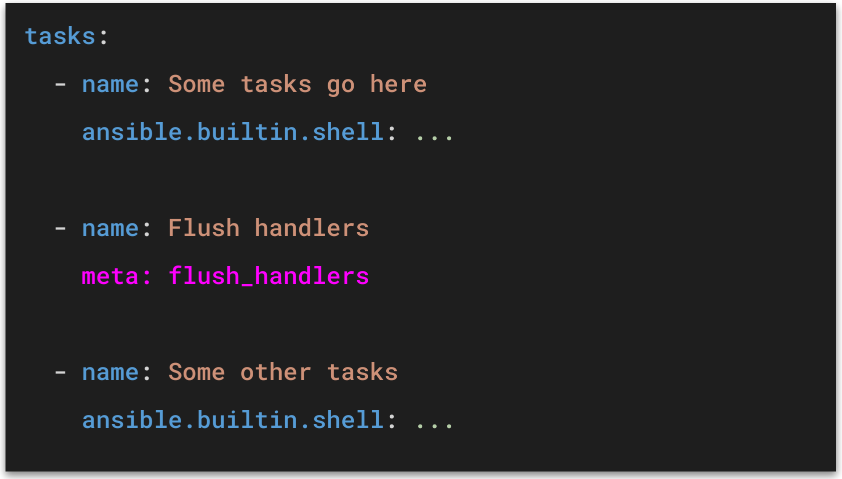
```yaml
tasks:
  - name: Install Nginx configuration file
    ansible.builtin.template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    notify: Restart Nginx
```

Force Handlers to execute with `flush_handlers`

- The **`ansible.builtin.meta`** module allows you to control various aspects of Ansible's behavior during playbook execution.
- **`action: flush_handlers`** can be used at task level to force notified handlers to run at a specific point in the playbook, regardless of whether they were notified or not.
- In the example, we use the `flush_handlers` action after installing the Nginx configuration file. This will force the "Restart Nginx" handler to run immediately after the task, even if the configuration file did not change.



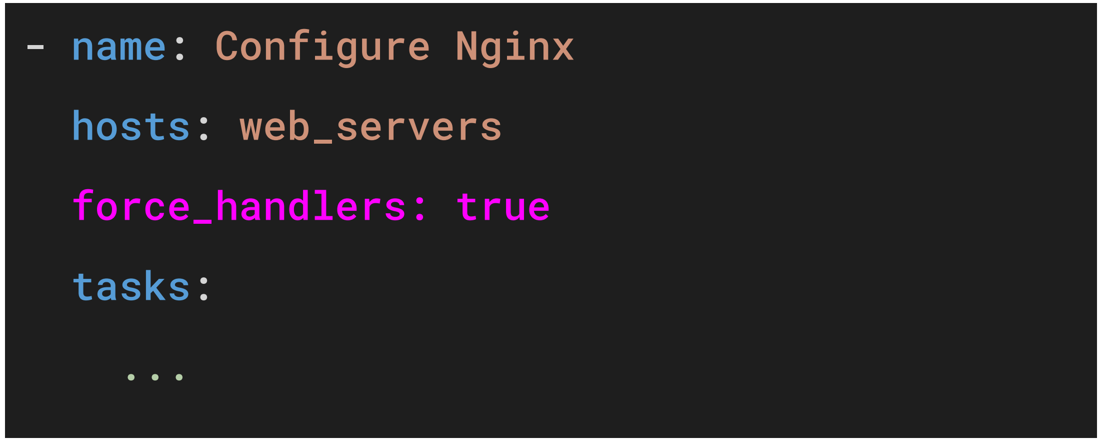
```yaml
tasks:
  - name: Install Nginx configuration file
    ansible.builtin.template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    notify: Restart Nginx
  - name: Flush handlers
    ansible.builtin.meta:
      action: flush_handlers
```



```yaml
tasks:
  - name: Some tasks go here
    ansible.builtin.shell: ...

  - name: Flush handlers
    meta: flush_handlers

  - name: Some other tasks
    ansible.builtin.shell: ...
```

Force Handlers to execute with `force_handlers`

- Another option is to use the **`force_handlers: true`** keyword at the playbook level.
- All handlers *that were notified* will be executed, even if a task fails during the playbook run.



```
- name: Configure Nginx
  hosts: web_servers
  force_handlers: true
  tasks:
    ...
```

Notifying multiple handlers from a single task

- A task can notify more than one handler by providing a list of handler names in the **`notify`** directive.
- In the example, the task notifies both the "Restart Nginx" and "Log Configuration Change" handlers when the Nginx configuration file is updated.

```
tasks:
  - name: Install Nginx configuration file
    ansible.builtin.template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    notify:
      - Restart Nginx
      - Log Configuration Change
```

Dependencies between handlers

- Sometimes, handlers may depend on each other, and you might need to ensure that one handler is executed before another.
- To handle such dependencies, you can use the **`listen`** keyword in the handler definition to create a common trigger point for multiple handlers.
- When a task **`notify`** triggers the listening handlers, they will be executed in the order they are defined.
- Note: **`listen`** topics cannot contain variable names.
- Example
    - both handlers "Restart Nginx" and "Log Configuration Change" listen to the same trigger point, "Nginx Configuration Change."
    - They will be executed in the order that they are defined.

```
handlers:
  - name: Restart Nginx
    ansible.builtin.service:
      name: nginx
      state: restarted
    listen: Nginx Configuration Change

  - name: Log Configuration Change
    ansible.builtin.shell:
      cmd: echo "Nginx configuration updated" >> /var/log/nginx_config.log
    listen: Nginx Configuration Change
```

Using variables with handlers

- Handler names can use Jinja2 templating.
- E.g., if you want your handlers to be distribution independent.
- listen topics do not support Jinja2 templates (i.e. variables)

```yaml
tasks:
  - name: Set host variables based on distribution
    include_vars: "{{ ansible_facts.distribution }}.yml"

handlers:
  - name: Restart web service
    ansible.builtin.service:
      name: "{{ web_service_name | default('httpd') }}"
      state: restarted
```

Using Handlers in Ansible Roles

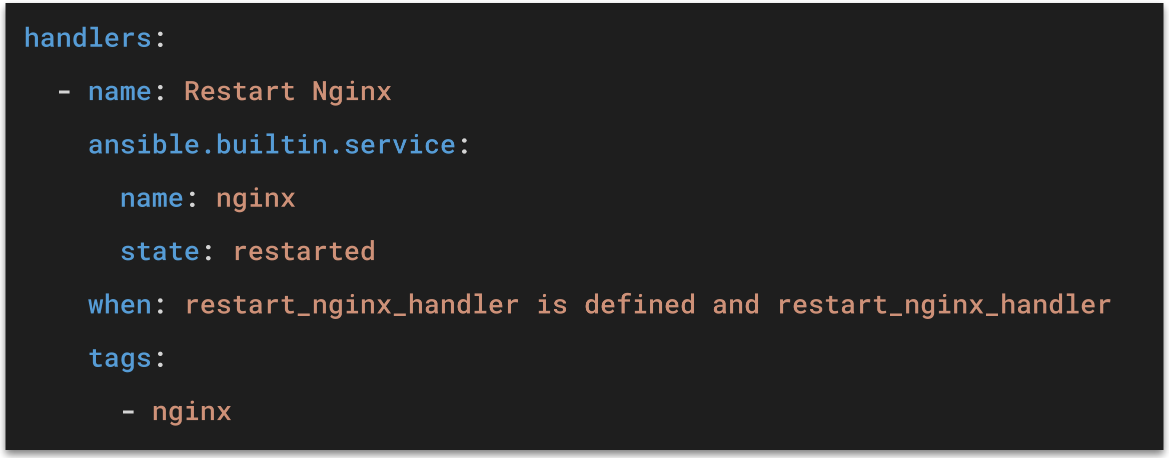
- In an Ansible role, handlers are stored in a separate `handlers/` directory, with one or more YAML files containing the handler definitions.
- The handlers in these files will be automatically loaded and available to tasks within the role.
- To notify a handler from a task within a role, use the same **`notify:`** directive as in regular tasks. The handler name should match the one defined in the role's "`handlers/`" directory
- *Advanced:* handlers defined within a role can also be notified from tasks outside the role. Use **`notify: [role_name]: [handler_name]`** *see example on the right*

```
role_name/
├── defaults/
├── files/
├── handlers/
│   ├── main.yml
│   ├── database_handlers.yml
│   └── nginx_handlers.yml
├── meta/
├── tasks/
├── templates/
└── vars/
```

```
- name: Install custom Nginx configuration
  ansible.builtin.copy:
    src: custom_nginx.conf
    dest: /etc/nginx/nginx.conf
  notify: nginx : Restart Nginx
```

Conditionally Executing Handlers

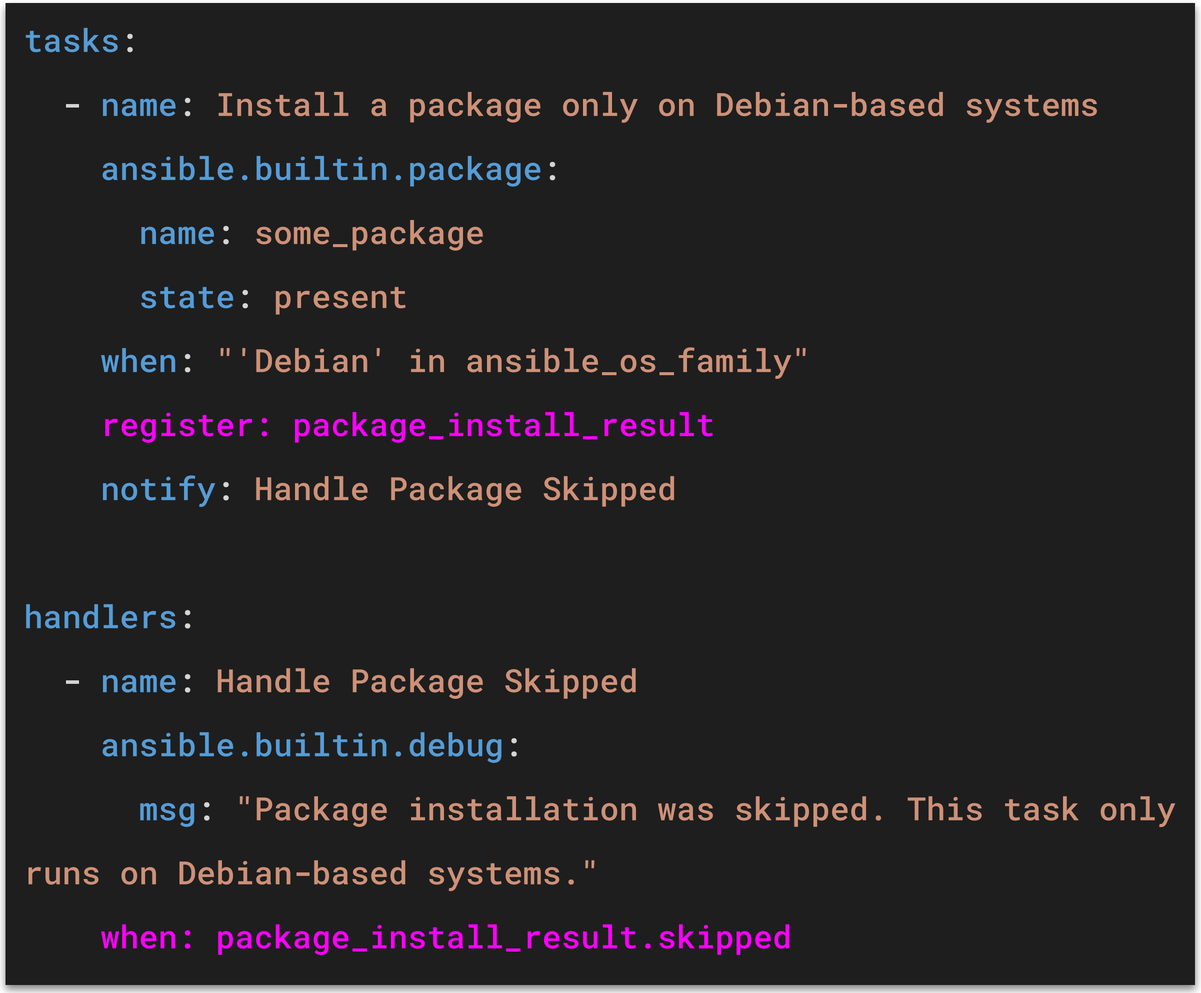
- Handlers can be used with conditions and tags, just like regular tasks, to control their execution based on specific criteria or to selectively include or exclude them during playbook runs.
- In this example, the "Restart Nginx" handler is executed only if the variable `restart_nginx_handler` is defined and set to `true`.
- The handler is **tagged** with `nginx`, allowing you to selectively include or exclude it using the `--tags` or `--skip-tags` options when running the playbook.



```yaml
handlers:
  - name: Restart Nginx
    ansible.builtin.service:
      name: nginx
      state: restarted
    when: restart_nginx_handler is defined and restart_nginx_handler
    tags:
      - nginx
```

Managing handler execution based on task outcomes or host-specific variables

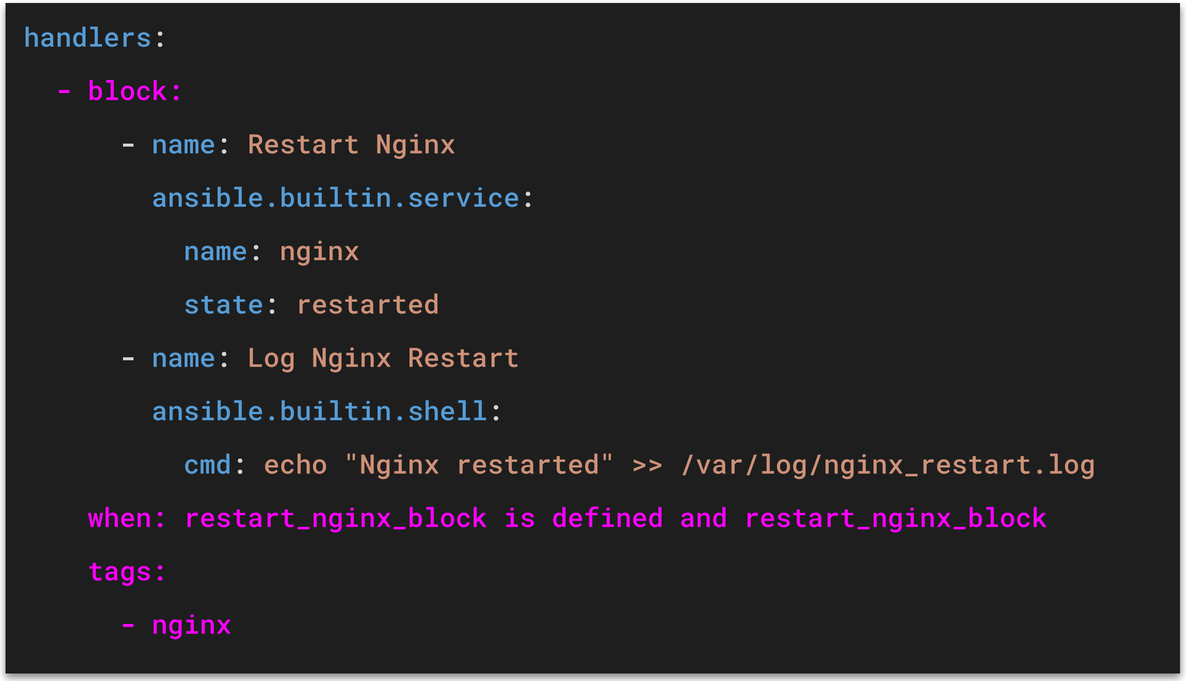
- Use **`register:`** in a task definition to store the task's result in a variable and then reference that variable in the handler's when clause.
- In the example, the task "Install a package only on Debian-based systems" registers its result in the variable `package_install_result`.
- The "Handle Package Skipped" handler is executed only if the task's result indicates that the task was skipped



```
tasks:
  - name: Install a package only on Debian-based systems
    ansible.builtin.package:
      name: some_package
      state: present
    when: "'Debian' in ansible_os_family"
    register: package_install_result
    notify: Handle Package Skipped

handlers:
  - name: Handle Package Skipped
    ansible.builtin.debug:
      msg: "Package installation was skipped. This task only
runs on Debian-based systems."
    when: package_install_result.skipped
```

Combining handlers with blocks for advanced control structures

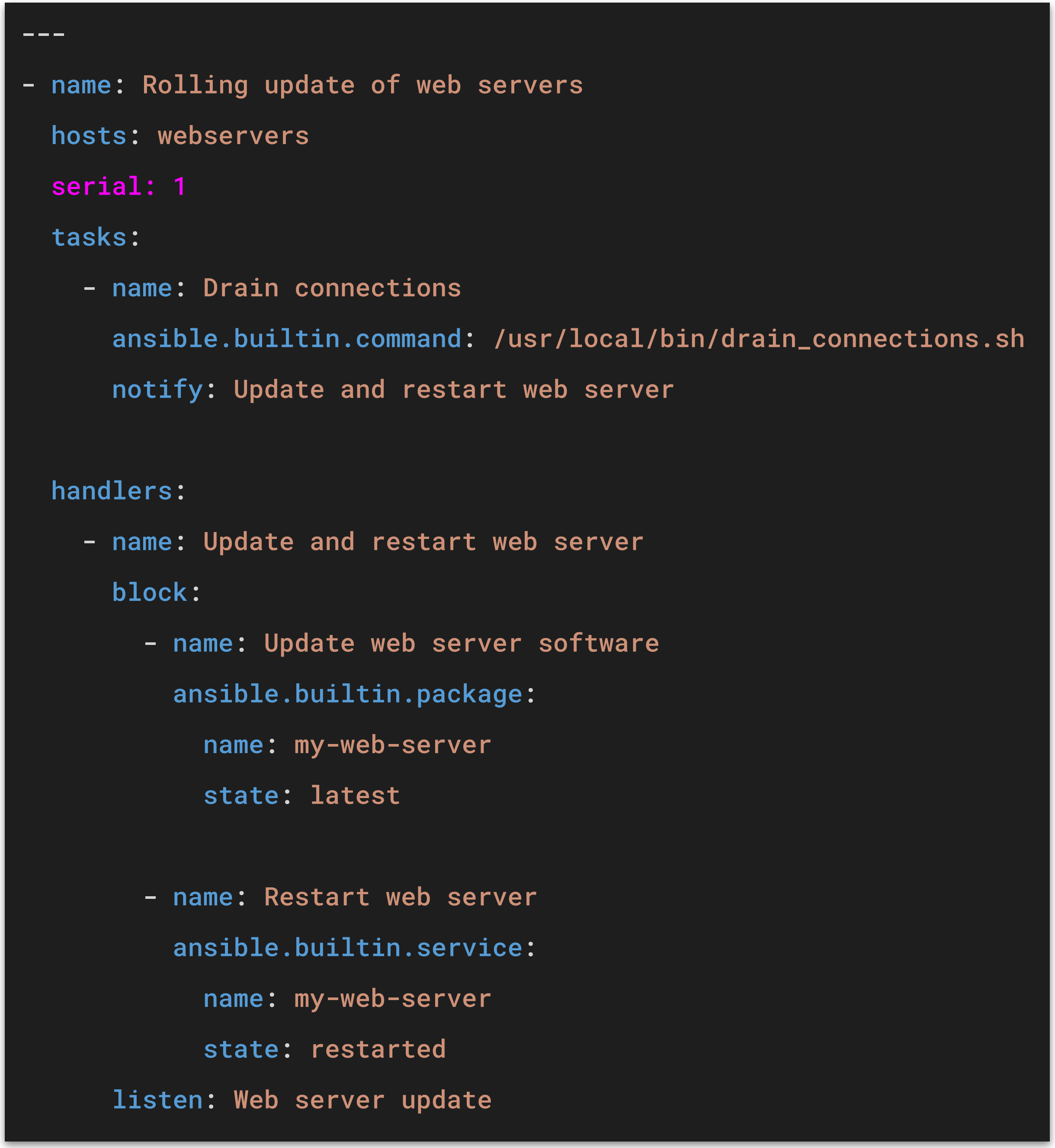
- Use Ansible's block control structure to group multiple handlers together and apply common conditions or tags to the entire block.
- This can help simplify your playbook and improve readability when managing complex handler scenarios.
- In the example, the "Restart Nginx" and "Log Nginx Restart" handlers are grouped together within a block.
- The entire block is executed only if the variable `restart_nginx_block` is defined and set to `true`.
- The block is tagged with `nginx`, allowing you to selectively include or exclude all handlers within the block during playbook runs.



```yaml
handlers:
  - block:
      - name: Restart Nginx
        ansible.builtin.service:
          name: nginx
          state: restarted
      - name: Log Nginx Restart
        ansible.builtin.shell:
          cmd: echo "Nginx restarted" >> /var/log/nginx_restart.log
    when: restart_nginx_block is defined and restart_nginx_block
    tags:
      - nginx
```

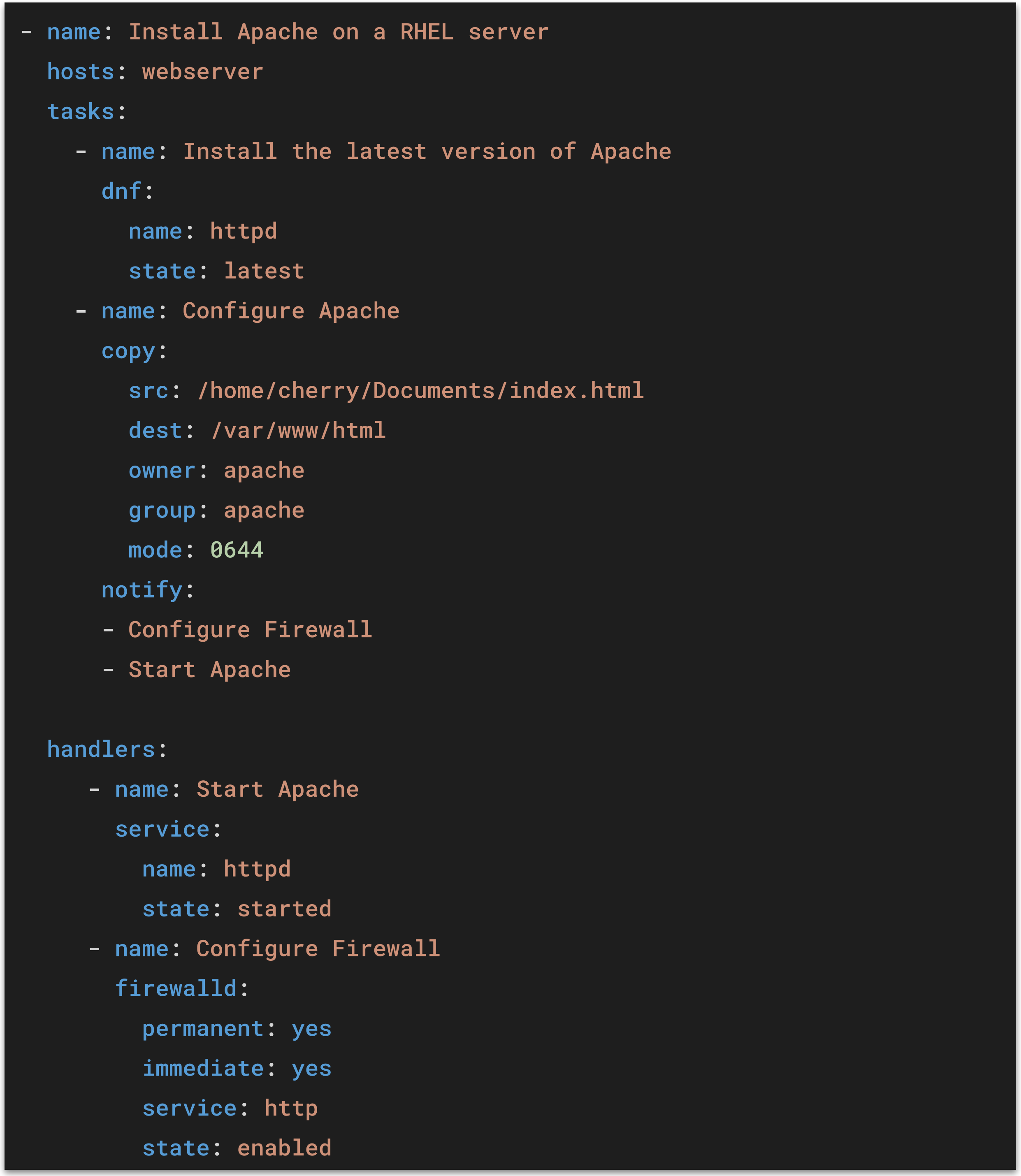
Example Rolling updates

- The serial keyword in an Ansible playbook is used to control the number of hosts that should be processed in parallel during the playbook execution.
- When you set `serial: 1`, it means that the playbook will be executed on one host at a time, rather than running on all the hosts simultaneously.



```yaml
---
- name: Rolling update of web servers
  hosts: webservers
  serial: 1
  tasks:
    - name: Drain connections
      ansible.builtin.command: /usr/local/bin/drain_connections.sh
      notify: Update and restart web server

  handlers:
    - name: Update and restart web server
      block:
        - name: Update web server software
          ansible.builtin.package:
            name: my-web-server
            state: latest

        - name: Restart web server
          ansible.builtin.service:
            name: my-web-server
            state: restarted
      listen: Web server update
```

Configuration example



```
- name: Install Apache on a RHEL server
  hosts: webserver
  tasks:
    - name: Install the latest version of Apache
      dnf:
        name: httpd
        state: latest
    - name: Configure Apache
      copy:
        src: /home/cherry/Documents/index.html
        dest: /var/www/html
        owner: apache
        group: apache
        mode: 0644
      notify:
      - Configure Firewall
      - Start Apache

  handlers:
     - name: Start Apache
       service:
         name: httpd
         state: started
     - name: Configure Firewall
       firewalld:
         permanent: yes
         immediate: yes
         service: http
         state: enabled
```

Workshop Exercise - Conditionals, Handlers and Loops

https://aap2.demoredhat.com/exercises/ansible_rhel/1.5-handlers/

Handlers Exercise 1

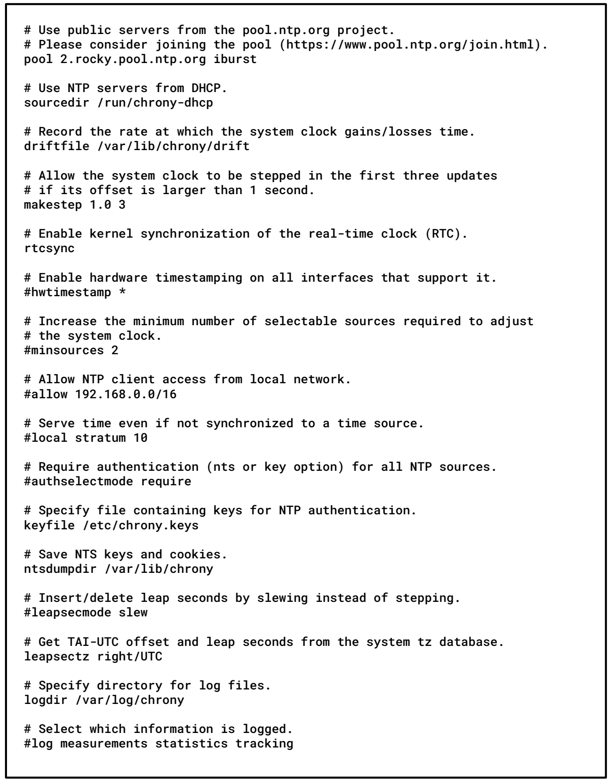
Create an Ansible playbook `manage_chrony.yml` to manage the chrony timeserver on all hosts.

The playbook should ensure

- that chrony is installed,
- enable and start the chrony service
- copy the standard configuration file `chrony.conf to /etc/chrony.conf`, but only if the source file is present. Skip the task if the source file doesn't exist
- any changes to the configuration file should trigger a restart of the chrony service.
- use appropriate Ansible modules and handlers to perform the necessary tasks.

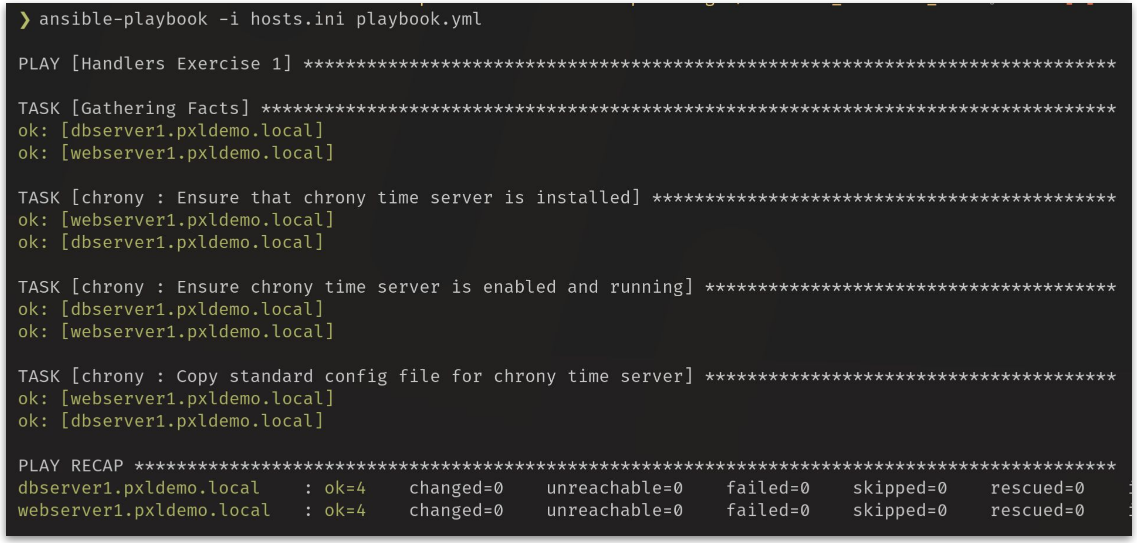
Extras

- Use a **variable file** to define the following variables and use them in the playbook.
    - the name of the chrony package
    - the name of the chrony service
    - the destination location of the chrony.conf file
- Integrate the solution into an ansible role and use the role in **`playbook.yml`**. Make sure all necessary artifacts are included in the role.



```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (https://www.pool.ntp.org/join.html).
pool 2.rocky.pool.ntp.org iburst

# Use NTP servers from DHCP.
sourcedir /run/chrony-dhcp

# Record the rate at which the system clock gains/losses time.
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
# if its offset is larger than 1 second.
makestep 1.0 3

# Enable kernel synchronization of the real-time clock (RTC).
rtcsync

# Enable hardware timestamping on all interfaces that support it.
#hwtimestamp *

# Increase the minimum number of selectable sources required to adjust
# the system clock.
#minsources 2

# Allow NTP client access from local network.
#allow 192.168.0.0/16

# Serve time even if not synchronized to a time source.
#local stratum 10

# Require authentication (nts or key option) for all NTP sources.
#authselectmode require

# Specify file containing keys for NTP authentication.
keyfile /etc/chrony.keys

# Save NTS keys and cookies.
ntsdumpdir /var/lib/chrony

# Insert/delete leap seconds by slewing instead of stepping.
#leapsecmode slew

# Get TAI-UTC offset and leap seconds from the system tz database.
leapsectz right/UTC

# Specify directory for log files.
logdir /var/log/chrony

# Select which information is logged.
#log measurements statistics tracking
```

`chrony.conf`



```
❯ ansible-playbook -i hosts.ini playbook.yml

PLAY [Handlers Exercise 1] ******************************************************************************

TASK [Gathering Facts] **********************************************************************************
ok: [dbserver1.pxldemo.local]
ok: [webserver1.pxldemo.local]

TASK [chrony : Ensure that chrony time server is installed] *********************************************
ok: [webserver1.pxldemo.local]
ok: [dbserver1.pxldemo.local]

TASK [chrony : Ensure chrony time server is enabled and running] ****************************************
ok: [dbserver1.pxldemo.local]
ok: [webserver1.pxldemo.local]

TASK [chrony : Copy standard config file for chrony time server] ****************************************
ok: [webserver1.pxldemo.local]
ok: [dbserver1.pxldemo.local]

PLAY RECAP **********************************************************************************************
dbserver1.pxldemo.local    : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
webserver1.pxldemo.local   : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
```

Handlers Exercise 1 - Documentation

- https://docs.ansible.com/ansible/latest/collections/ansible/builtin/package_module.html
- https://docs.ansible.com/ansible/latest/collections/ansible/builtin/service_module.html
- https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_handlers.html
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html

end