6.2
ROLES

ROLES

Ahora que ya sabemos lo que es una tarea, un *handler* o un *include*, sabemos que **un proyecto en Ansible puede ser organizado para que sea mantenible**.

Si ya sabemos todo esto, **¿para qué sirven los roles?**

ROLES

Cuando nuestra aplicación crece y hacemos *includes*, podemos llegar a transformar la aplicación en algo ilegible porque tendríamos un *include* dentro de otro infinitamente.

Aquí surgen los roles, que son “paquetes” de configuraciones bien organizados que pueden reutilizarse en cualquier host.

ROLES

En un *role* podemos detectar si estamos en un sistema operativo u otro, podemos crear variables, *templates*, archivos, etc. Todo ello de una manera estructurada.

Un *role* no es más que muchos includes organizados y estructurados.

ROLES - ESTRUCTURA

Esta podría ser la estructura de un proyecto:

```
webservers.yml
roles/
    common/
      files/
      templates/
      tasks/
      handlers/
      vars/
      defaults/
      meta/
```

ROLES - ESTRUCTURA

En un *playbook* podríamos utilizarlo de la siguiente manera:

```
---
- hosts: webservers
  roles:
    - common
```

ROLES - ESTRUCTURA

El comportamiento de un *role* es:

- Si **roles/x/tasks/main.yml** existe, las tareas dentro de dicho archivo se añadirán al *play*.
- Si **roles/x/handlers/main.yml** existe, los *handlers* dentro de dicho archivo se añadirán al *play*.
- Si **roles/x/vars/main.yml** existe, las variables dentro de dicho archivo se añadirán al *play*.
- Si **roles/x/defaults/main.yml** existe, las variables dentro de dicho archivo se añadirán al *play*.

ROLES - ESTRUCTURA

- Si **roles/x/meta/main.yml** existe, las dependencias listadas en dicho fichero se añadirán a la lista de roles.
- Las carpetas **roles/x/files** y **roles/x/templates** se utilizan para guardar ficheros y plantillas, que por lo general son de configuración.

Si algún directorio no existe, simplemente se ignora.

ROLES PARAMETRIZADOS

De igual manera que los *includes*, los roles también pueden recibir parámetros añadiendo variables para ser reutilizados. Por ejemplo:

```
---
- hosts: webservers
  roles:
    - common
    - { role: custom_role, foo: '1', bar: '1' }
    - { role: custom_role, foo: '2', bar: '2' }
```

ROLES PARAMETRIZADOS

Es muy útil añadir *tags* a los roles para que sea más fácil ejecutarlos a nuestro gusto:

```
---
- hosts: webservers
  roles:
    - { role: common, tags: “common” }
    - { role: custom_role, tags: “custom” }
```

Con esto asignamos *tags* a todas las tareas de un *role*, es decir, que las sobreescribimos.

ROLES - PRE_TASKS y POST_TASKS

Si queremos que determinadas tareas se ejecuten antes y después de que los *roles* sean aplicados, debemos utilizar las `pre_tasks` y `post_tasks`:

```
- hosts: webservers
  pre_tasks:
    - shell: echo “Hello World”
  roles:
    - { role: common, tags: “common” }
  tasks:
    - shell: echo “Running a task”
  post_tasks:
    - shell: echo “Bye!”
```

ROLES - VARIABLES POR DEFECTO

Las variables por defecto de un *role* se añaden a **default/main.yml** dentro del directorio de nuestro *role*.

Estas variables tienen la prioridad más baja de entre todas las variables disponibles y pueden ser sobreescritas fácilmente por otra variable, incluidas las de los inventarios.

ROLES - DEPENDENCIAS

Las dependencias nos permiten ejecutar *roles* dentro de *roles*. Se almacenan dentro del directorio **meta/main.yml**.

Este archivo debe incluir una lista de dependencias de *roles* y parámetros a ejecutar.

```
dependencies:
  - { role: '/path/to/roles/foo', bar: 50 }
  - { role: apache, port: 80 }
```

ROLES - DEPENDENCIAS

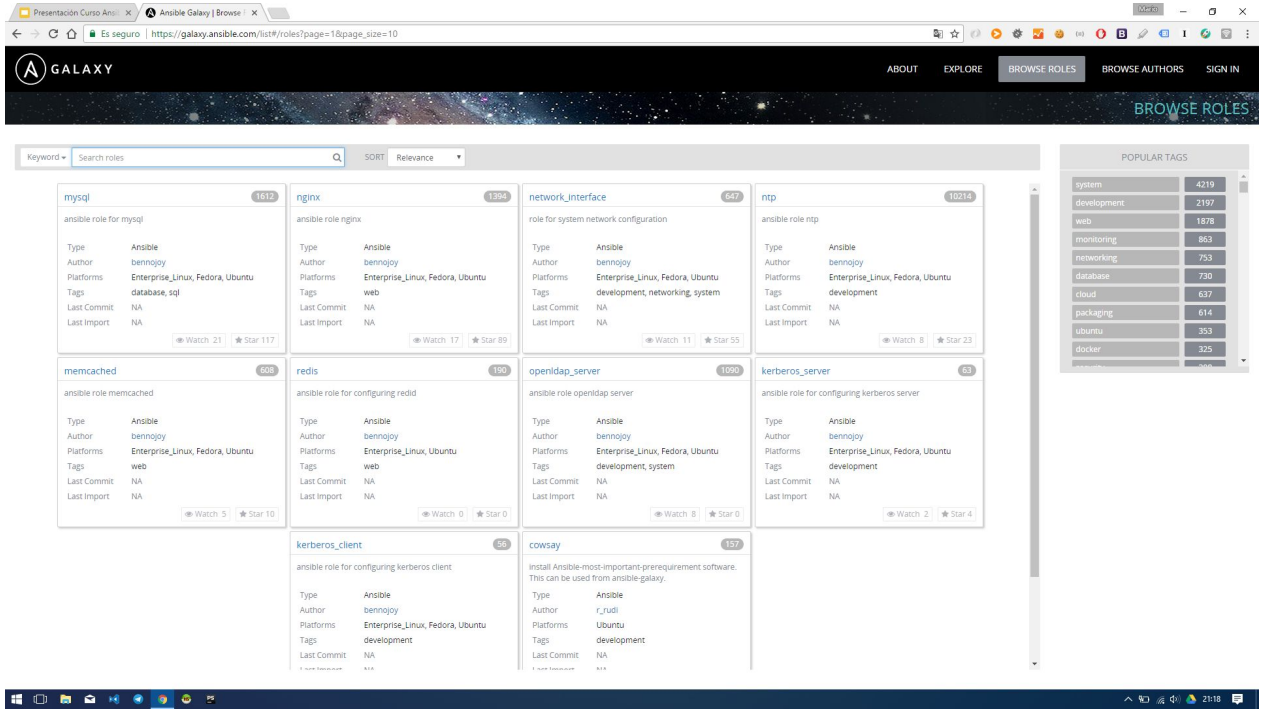
Las dependencias de un *role* se ejecutan antes que el propio *role*.

Si un *role* es una dependencia de otro no se ejecutará. Si queremos, podemos indicarle que se ejecute de nuevo haciendo uso de la directiva `allow_duplicates`.

```
allow_dupliates: yes #Por defecto es ‘no’
dependencies:
  - { role: '/path/to/roles/foo', bar: 50 }
  - { role: apache, port: 80 }
```

ANSIBLE GALAXY

ANSIBLE GALAXY



CONCLUSIONES