PRACTICA 3 - MONITORIZACIÓN, AUTOMATIZACIÓN Y PROFILING

Monitores para Hardware

dmesg: muestra los mensajes del kernel. Sirve para poder detectar problemas conn el HW o periféricos. Se puede utilizar dmesg con cauce a grep para buscar la información que queramos saber (memoría física, dispositivos USB, advertencias, etc).
Otros comandos útiles: lspci, lsusb, lshw, que listan el hardware disponible y conectado.

Monitores para Software

Subsistema de archivos: en /proc y /var se encuentra información tanto del hardware como del software así como archivos fundamentales para la monitorización del sistema, de las aplicaciones y de los usuarios. */var/log* contiene los archivos en los que se van volcando los logs de los servicios y algunas aplicaciones.

Identificar, monitorizar y reconstruir los discos de un RAID 1:

Monitorizando un servicio o ejecución de un programa: strace
Hay un conjunto de programas que permiten hacer una traza de las llamadas al sistema realizadas por un programa (servicio) en ejecución, p.ej. strace (system call tracer). Este tipo de programas pueden ser útiles de cara a detectar problemas que no se muestran en los archivos de "log".

Monitores generales

- **Munin:** usa RRDTool[1]. Tiene una arquitectura maestro/esclavo (cliente/servidor) en la que el servidor conecta a todos los clientes en intervalos de tiempo (Round Robin) y les pide los datos. Luego los almacena en archivos RRD y actualiza los graficos. RRDtool(round robin database tool) es una herramienta que trabaja con una base de datos que maneja planificación según Round Robin. El funcionamiento es el siguiente: se trata la base de datos como si fuese un círculo, sobrescribiendo los datos almacenados con anterioridad una vez alcanzada la capacidad máxima de la misma. Esta capacidad máxima dependerá de la cantidad de información que se quiera conservar como historial. Su finalidad principal es el tratamiento de datos temporales y datos seriales como temperaturas, transferencias en redes, cargas del procesador, etc.
- **Naemon y Nagios:**
- **Ganglia:**
- **Cacti:**
- **AW stats:** todos los de arriba monitorizan el sistema.

Zabbix (ver puidge final doc):

Ejercicio 1: Realice una instalación de Zabbix 5.0 en su servidor con Ubuntu Server20.04 y configure para que se monitorice a él mismo y para que monitorice a la máquina con CentOS. Puede configurar varios parámetros para monitorizar, uso de CPU, memoria, etc. pero debe configurar de manera obligatoria la monitorización de los servicios SSH y HTTP.

UbuntuServer: Instalar el repositorio de zabbix. Para ello nos vamos a https://www.zabbix.com/download?zabbix=5.0&os_distribution=ubuntu&os_version=20.04_focal&db=mysql y seguimos los pasos. Debajo, nos aparecen los comandos a seguir para realizar la instalación los cuales voy a listar:

sudo wget https://repo.zabbix.com/zabbix/5.0/ubuntu/pool/main/z/zabbix-release/zabbix-release_5.0-1+focal_all.deb
sudo dpkg -i zabbix-release_5.0-1+focal_all.deb
sudo apt update

Instalamos ahora el servidor, el front-end y el agente de zabbix.
sudo apt install -y zabbix-server-mysql zabbix-frontend-php zabbix-apache-conf zabbix-agent (-y sirve para decir que si a todo)

Tenemos ahora que habilitar apache. sudo systemctl start/enable/reload apache

Configuramos ahora la base de datos:

- La creamos:

(Seguir los pasos de la pagina siguiente para centOS)
https://www.zabbix.com/download?zabbix=5.0&os_distribution=centos&os_version=8&db=mysql&ws=apache

Automatización

cron y systemd

El servicio cron ha permitido ejecutar cada cierto intervalo de tiempo una tarea concreta. Esto es muy útil de cara a recopilar información o monitorizar el sistema realizando una tarea concreta y lanzar alertas como, por ejemplo, enviar un correo electrónico cuando la carga esté por encima de un valor determinado. cron es un servicio que corre en background. CentOS está reemplazando esto por systemd timers, porque cron tiene una versatilidad muy limitada, pero son complejos de configurar.

En /etc/cron tenemos una definición de directorios (en Ubuntu tienen una forma específica).

En /etc/crontab establecemos las reglas.

Como usuario no privilegiado también se pueden programar tareas periódicas con crontab -e, a nivel de usuario.

Importante: Para facilitarnos trabajar con expresiones de cron hay muchas páginas online, como crontab.guru.com

Para automatizar la ejecución de un script con systemd tenemos que definir un timer dentro del directorio /etc/systemd/system/ que se encarga de gestionar un servicio. Debemos crear dos archivos: uno archivo.timer y otro archivo.service. Para que pueda funcionar dicho timer, hay que activarlo de la siguiente forma: *systemctl enable archivo.timer* y *systemctl start archivo.timer.* Por último, ejecutamos el comando *journalctl -u archivo --since="yesterday".* (No tiene por qué ser yesterday, puede ser today o lo que sea).

Scripts

Shell y comandos del sistema: grep, find, awk, sed.

Sed: con sed puede buscar una cadena en un archivo y reemplazarla por otra, así podría dar acceso por contraseña durante unos instantes al servicio ssh para que los usuarios puedan copiar su llave pública. Ejemplos de uso: sed 's/Linux/Linux-Unix/' thegeekstuff.txt (s es para reemplazar)

Awk: puede programar la manipulación del texto así como generar salidas más completas a partir de la información en un archivo.

```
Syntax:

awk '/search pattern1/ {Actions}
     /search pattern2/ {Actions}' file
```

In the above awk syntax:

- search pattern is a regular expression.
- Actions – statement(s) to be performed.
- several patterns and actions are possible in Awk.
- file – Input file.
- Single quotes around program is to avoid shell not to interpret any of its special characters.

Grep: puede realizar filtrado de cadenas, útil cuando un archivo, listado, etc tiene unas dimensiones grandes, p.ej. ps -Af l grep firefox nos mostraría la información del pro- ceso firefox.

Find: puede buscar archivos y, una vez encontrados, realizar acciones sobre ellos, p.ej. *find /home/alberto/docs -name '*pdf' -exec cp {} ~/PDFs \;* copiará todos los archivos cuyo nombre termine en pdf y los copia en la carpeta /home- /alberto/PDFs

Python y php -> sí

A nivel de plataforma: Ansible (ver puidge final doc)

Ejercicio 2: Usted deberá saber cómo instalar y configurar Ansible para poder hacer un ping a las máquinas virtuales de los servidores y ejecutar un comando básico (p.ej. el script de monitorización del RAID1). También debe ser consciente de la posibilidad de escribir acciones más complejas mediante playbooks escritos con YAML.

Zabbix

- Usamos v3.4 por temas de compatibilidad.
- Permite monitorizar distintos elementos (routers, VPNs, ...), llamarlos y preguntarles información sobre su estado (telemetría: información, temperatura; procesos más complejo: número de instancias de Apache, cuánta memoria usa MySQL, cuántos usuarios). Se basa en plugins para conectarse con las distintas aplicaciones.
- Dos estrategias:
  - **push: empujar**
    - La toma de control se invierte: los equipos comunican a Zabbix para mandar la información que les es relevante cuando ellos consideran oportuno hacerlos.
  - **pull: muestrear**
    - Más antigua.
    - Si tenemos un servidor de Zabbix y tenemos que analizar el funcionamiento de cinco equipos, vamos uno por uno preguntando: muestreo secuencial de la información disponible.
  - ¿Qué estrategia es mejor? Se usa más *push* porque deja la iniciativa al cliente, que decide si tiene algo relevante que comunicar o no. De este modo se reduce la carga del servidor principal, que sólo interviene cuando se necesita. No hay consenso sobre cuál es mejor, porque con push no tenemos control sobre cuándo recibimos la información, puede que se sature el servidor central.
- Zabbix admite ambos modos, aunque por defecto es pull.
- Diferentes tipos de procesos que intervienen en una configuración Zabbix:
  - Servidor: núcleo, recaba los datos de telemetría y los mete en una BD. Usaremos MySQL para BD. Zabbix usa BD relacional, aunque la mayor parte de los sistemas modernos usan BD más especializadas (documentadas o del tipo InfluxDB).
  - Agentes: procesos que corren en cada uno de los sistemas que queremos monitorizar. Tomará las medidas locales indicadas y las dará cuando Zabbix las pida. No pasa nada si Zabbix salta el agente en algún momento, el agente puede almacenar métricas y se las manda todas de golpe.
  - Proxy: estrategia para poder escalar. Cuando hay muchos equipos, un solo servidor no podrá muestrearlos todos. Se monta un proxy (un server en chiquitito) que muestrea un grupo local de equipos. De este modo, el server consulta a distintos proxys. Cada proxy mandará la información de varios equipos.
    - Cuando tenemos una comunicación TCP/IP, el tiempo de apertura es muy grande, por eso nos interesa que envíen mucha información.
  - Get: recomendado para ejercicio final de clase. Implementa una interfaz REST para consultar un agente de Zabbix. Es útil para depurar problemas de comunicación agente-servidor. Lanzamos una petición directamente y vemos si hay algún problema. Esto no se ve de forma tan sencilla en las gráficas del server. También permite desarrollar los propios clientes.
    - Es un GET REST.
    - Sirve también para crear un cliente propio que use una API REST.
  - Frontend

Hay que hacer una red de monitorización en la que hay que monitorizar a la máquina Ubuntu además de a la CentOS.

Algunas anotaciones respecto a Zabbix

- Los agentes usan el puerto `10050`.
- En el frontend, en la pestaña **Monitoring**:
  - **Hosts:** muestra los equipos que estamos monitorizando.
  - **Latest data:** podemos hacer filtros, grupos de máquinas.
  - **Graph:**
    - Muestra los gráficos de los filtros que hayamos creado.
    - La gráfica no se refresca en tiempo real para evitar sobrecargas del sistema.
    - Hay un cierto tiempo de muestreo, gestionable desde **Configuration** para cada máquina.
    - Un cambio en el servidor puede provocar un punto muerto en el que no sea visible el cambio desde Zabbix, motivado por el tiempo entre muestreo.
- En entorno de producción, siempre se tiene una pantalla o proyección con el **dashboard** para vigilar el correcto funcionamiento del sistema.

Protocolo SNMP

El *protocolo simple de administración de redes* (**SNMP**) es un protocolo de capa de aplicación definido por la IAB en RFC1157 para intercambiar información de administración entre dispositivos de red. Forma parte del conjunto de protocolos TCP/IP.

SNMP es uno de los protocolos ampliamente aceptados para administrar y monitorizar elementos de red. La mayoría de los elementos de red de nivel profesional vienen con un agente SNMP incluido. Estos agentes deben estar habilitados y configurados para comunicarse con el sistema de administración de red (NMS).

Es uno de los protocolos que usa Zabbix.

Instalación y configuración de Zabbix

Vamos a instalar Zabbix en Ubuntu para monitorizar CentOS.

Instalación del proxy y del servidor

1. Nos conectaremos a Ubuntu y a CentOS en dos terminales diferentes:

   > Nota: indicaremos con el prefijo `$U>` los comandos realizados en la shell de Ubuntu y con el prefijo `$C>` los comandos realizados en la shell de CentOS.

   ```
   $U> ssh mianfg@ubuise
   ```

   ```
   $C> ssh mianfg@centosise
   ```

2. En Ubuntu instalamos Zabbix de acuerdo al manual de instalación. Instalaremos Zabbix 3.4.

> Importante: como estamos en Ubuntu 16.04, usamos el codename `xenial` en lugar de `bionic` (que es el correspondiente a Ubuntu 18.04).

```
$U> wget https://repo.zabbix.com/zabbix/3.4/ubuntu/pool/main/z/zabbix-release/zabbix-release_3.4-1+xenial_all.deb
$U> dpkg -i zabbix-release_3.4-1+xenial_all.deb
$U> apt update
```

3. Instalamos el servidor, el proxy y el frontend. Usamos MySQL aprovechando que lo tenemos instalado (ver la práctica anterior). Si no lo tenemos instalado, haremos `apt install mariadb-server`.

```
$U> apt install zabbix-server-mysql
$U> apt install zabbix-proxy-mysql
$U> apt install zabbix-frontend-php
```

   El último instalará PHP en caso de que no lo tengamos instalado.

4. Creamos las bases de datos necesarias en MySQL para el servidor y el proxy. Seguimos este manual.

```
$U> sudo mysql -uroot -p<password>
mysql> create database zabbix_server character set utf8 collate utf8_bin;
mysql> grant all privileges on zabbix_server.* to zabbix@localhost identified by '<password_usuario_zabbix>';
mysql> create database zabbix_proxy character set utf8 collate utf8_bin;
mysql> grant all privileges on zabbix_proxy.* to zabbix@localhost identified by '<password_usuario_zabbix>';
mysql> quit;
```

> Importante: cada `<password>` puede ser diferente. La primera es la contraseña para entrar como `root` a MariaDB. La segunda y la tercera es la contraseña del usuario `zabbix` en la BD. Podemos sustituir el usuario a otro nombre (cambiando `zabbix@localhost` a `tu@localhost`), pero lo dejarems así pues el usuario `zabbix` sólo podrá acceder a las tablas `zabbix_server` y `zabbix_proxy`.

5. Importamos los esquemas para la base de datos:

```
$U> zcat /usr/share/doc/zabbix-server-mysql/create.sql.gz | mysql -uzabbix -p zabbix_server
  # e insertamos la contraseña, <password_usuario_zabbix>
$U> zcat /usr/share/doc/zabbix-proxy-mysql/schema.sql.gz | mysql -uzabbix -p zabbix_proxy
  # e insertamos la contraseña, <password_usuario_zabbix>
```

6. Editamos los archivos de configuración del server y proxy de Zabbix:
   1. Archivo de configuración del server: `/etc/zabbix/zabbix_server.conf`.

      Debe tener los siguientes campos:

```
DBHost=localhost
DBName=zabbix_server
DBUser=zabbix
DBPassword=<password_usuario_zabbix>
```

2. Archivo de configuración del proxy: `/etc/zabbix/zabbix_proxy.conf`.

   Debe tener los siguientes campos:

```
DBHost=localhost
DBName=zabbix_proxy
DBUser=zabbix
DBPassword=<password_usuario_zabbix>
```

7. Activamos los servicios y los configuramos para que se enciendan cuando lo haga el servidor.

```
$U> service zabbix-server start
$U> update-rc.d zabbix-server enable
$U> service zabbix-proxy start
$U> update-rc.d zabbix-proxy enable
```

8. Configuramos el timezone de Apache, descomentando la línea `date.timezone` del archivo `/etc/apache2/conf-enabled/zabbix.conf`.
9. Reiniciamos Apache.

```
$U> service apache2 restart
```

10. Instalamos el agente.

```
$U> apt install zabbix-agent
```

11. Iniciamos el agente.

```
$U> service zabbix-agent start
```

Instalación del frontend

Lo instalaremos siguiendo este manual.

1. Abrimos en el navegador `http://ubuise/zabbix` (donde `ubuise` es la IP de la máquina Ubuntu Server).
2. Iniciamos sesión con el usuario y contraseña por defecto: `Admin` y `zabbix` (atentos a las mayúsculas).
3. Activamos los puertos `10050` y `10051` sobre `tcp` en el firewall.

```
$U> sudo ufw allow 10050/tcp
$U> sudo ufw allow 10051/tcp
```

Usando Zabbix para monitorizar el propio servidor (Ubuntu)

1. Para crear un host, basta ver este manual. Nosotros comenzaremos con el creado por defecto en Configuration > Hosts.
2. Si nos da un error en el agente, que nos dice que no puede conectarse al host porque no estaba permitido, podremos ver este error en el frontend (el agente no es alcanzable) y en el log de `zabbix_agentd`. Lo solucionamos añadiendo la IP de Ubuntu a los hosts permitidos en el archivo de configuración `/etc/zabbix/zabbix_agentd.conf`.

   ```
   Server=127.0.0.1,192.168.56.10
   ```

3. Añadimos los ítem de SSH y HTTP, tal y como nos indican aquí.

   Un poco más de información sobre cómo agregarlos:

   1. Nos vamos al host en concreto, **Zabbix server** en nuestro caso (Zabbix server está en Ubuntu).
   2. Pulsamos en **Items**, y luego en **Create item**.
   3. Insertamos los siguientes campos:

Servicio	Name	Key
SSH	SSH Status	net.tcp.service[ssh,,]
HTTP	HTTP Status	net.tcp.service[http,,]

      Realmente el campo *Name* podemos nombrarlos como queramos. Las claves (*Key*) de los servicios podemos consultarlas en la documentación de Zabbix.

   4. Pulsamos en **Add**.
4. Buscamos los gráficos para ver la información que está monitorizando Zabbix. Vamos a Monitoring > Latest data.
5. Colocamos los filtros: en Filter seleccionamos "Zabbix agent" en **Hosts**. Pulsamos en **Apply**.
6. En **- other -**, al final, y en caso de que no hayamos añadido un grupo al crear cada item, tenemos los items que acabamos de crear. Pulsamos en **Graph** para ver la información monitorizada.
7. Pararemos el servicio SSH, por ejemplo, para ver cómo Zabbix lo detecta:

   ```
   $U> systemctl stop ssh.service
   ```

Usando Zabbix para monitorizar otro servidor (CentOS)

1. Instalaremos el agente en CentOS. Para ello basta seguir este tutorial.

   ```
   $C> sudo rpm -ivh https://repo.zabbix.com/zabbix/3.4/rhel/7/x86_64/zabbix-release-3.4-2.el7.noarch.rpm
   $C> sudo yum install zabbix-agent
   $C> service zabbix-agent start
   ```

2. Nos da un error: *Job for zabbix-agent.service failed because a configured resource limit was exceeded.* Podemos arreglarlo deshabilitando SELINUX, tal y como se nos indica aquí y aquí. Basta modificar la línea siguiente de `/etc/selinux/config`:

   ```
   SELINUX=disabled
   ```

   Y reiniciando CentOS (`sudo shutdown -r now`). Una vez reiniciado, podemos comprobarlo con la orden `sestatus`. Volvemos a iniciar el servicio y nos dejará sin problemas:

   ```
   $C> service zabbix-agent start
   ```

3. Abrimos los puertos necesarios:

   ```
   $C> sudo firewall-cmd --add-port=10050/tcp --permanent
   $C> sudo firewall-cmd --add-port=10051/tcp --permanent
   $C> systemctl restart firewalld
   $C> systemctl restart zabbix-agent
   ```

4. Modificamos `/etc/zabbix/zabbix_agentd.conf` y añadimos la IP del servidor donde está Zabbix Server (Ubuntu en nuestro caso) en `Server` y en `ServerActive`.
5. En el frontend de Zabbix creamos un nuevo host en Configuration > Hosts, y luego en **Create host**. Lo nombraremos **CentOS**, y cambiaremos **IP Address**, en **Agent interfaces**, a la IP de CentOS. Lo añadimos a un grupo (pues si no no nos dejará añadirlo). Pulsamos en **Add**.
6. Añadimos los items SSH y HTTP, de forma análoga a en el caso de Ubuntu.

   Desactivamos alguno de ambos, por ejemplo SSH, para ver cómo Zabbix lo monitoriza correctamente. En la captura siguiente aparece el resultado de parar el servicio `sshd`, y luego volver a iniciarlo.

Ansible

- Es un producto de RedHat enmarcado en los productos de **gestión de la configuración** (labor que hacen los administradores de sistemas para llevar un equipo a una configuración conocida: ciertas cuentas con ciertos privilegios, cierta configuración de firewall, de aplicaciones instaladas, etc.)
  - Opciones:
    - Hacerlo manualmente.

        - Instalar un SO con ciertas preconfiguraciones, de modo que haya cuantos menos preguntas en la instalación mejor.
        - VMs elaboradas para determinadas configuraciones. Ej. Vagrant permite definir las imágenes de forma que no dependa de la tecnología de virtualización.
        - Definir la configuración con ficheros en texto plano. No propietario de fabricante, sin vincularse a tecnología concreta. Como todos los proyectos se almacenan en repositorios, se trata del formato ideal.
- Ansible se conecta con un equipo y lo configura. Puede lanzarse en paralelo (muchas hebras, muchos equipos simulténeamente). Lo veremos en el ámbito de la automatización, pero donde más se utiliza es en el uso de Playbooks (término propio de Ansible): ficheros de texto que permiten definir la configuración de un equipo.
    - Estos ficheros son gestionables en un repositorio SCM.
- Veremos Ansible en el contexto de ejecución de comandos (aunque nos anima a ver los Playbooks).
- Ejecuta Ansible desde el ordenador anfitrión, ha creado una máquina más para hacer gesitones en tres máquinas.
- **Inventario:** lista de ordenadores con los que vamos a trabajar. En los equipos Linux está disponible como paquete.

  ```
  apt install ansible
  ```

  Por defecto se encuentra en `/etc/ansible/hosts`.
- El archivo de configuración interna está en `/etc/ansible/ansible.cfg`. En la instalación por defecto todas las líneas están comentadas, el comportamiento por defecto es razonable.
    - Si no tenemos acceso de root a ese archivo, en el directorio `home` tenemos el fichero `.ansible.cfg`.
- En el host añadimos la IP del equipo, o un nombre simbólico.

  ```
  ubudef ansible_host=192.168.56.11
  ```

  Si el ordenador estuviese accesible por un DNS o en un archivo *host* no hace falta añadir el parámetro `ansible_host`.

- Comprobar que Ansible está correctamente configurado, tanto en host como en otros.

  ```
  ansible ubudef -m ping
  ```

  `m` viene de *módulo*. Hemos puesto el nombre del catálogo. Nos dice que es *unreachable*. Esto puede ser por temas de *firewall*. Luego veremos cómo resolverlo.
    - En la dirección de la referencia tenemos una lista de los **módulos** de Ansible. Los módulos dicen qué es lo que puede hacer: instalar paquetes, crear usuarios, crear redes, iniciar servicios... Si casualmente lo que queremos hacer no tiene módulo existe se puede crear. **Galaxy** es un repositorio donde los usuarios comparten los módulos que han creado.

- El error se debe a que nos estamos intentando conectar como el usuario del *host*. Si el usuario no existe, no se conectará. Para ello debemos hacer que se conecte usando otro usuario:

  ```
  ansible ubudef -m ping -u david
  ```

  Tampoco se conecta porque requiere insertar una contraseña, dando un error de *permission denied*. Crearemos un usuario en la máquina, que deberá permitir el acceso sin contraseña.

  ```
  useradd -m mjose
  ```

  Con este usuario si no tiene contraseña podremos conectarnos. Para configurarse en un ordenador Ansible necesita acceso sin contraseña y Python (eso lo vemos en el feedback del *ping*). Podemos pasar el parámetro `data` con `-a` (*arguments*).

  ```
  ansible ubudef -m ping -u mjose -a 'data="hola hola caracola"'
  ```

Para evitar tener que recordar qué cuentas tenemos configuradas en cada ordenador, lo mejor es llevarlo al fichero de inventario:

```
ubudef ansible_host=... ansible_user=mjose
```

Añadimos más usuarios a `hosts`: `cenise` y `ubuise`. Haciendo `ping` a ellos vemos que se conecta bien.

Si no tenemos la configuración de acceso sin contraseña en CentOS, hacemos

```
ssh-copy-id david@cenise
```

- Usar todas las máquinas

  ```
  ansible all -m ping
  ```

  Ansible no lo hace en secuencia, sino en paralelo. El número máximo de hebras puede cambiarse (el profesor dice que cree que por defecto es 5).
- Asignar nombres simbólicos: añadimos en el archivo `hosts`:

  ```
  [lamp]
  cenise
  ubuise
  ```

  De este modo podemos hacer `ping` a los equipos `lamp`:

  ```
  ansible lamp -m ping
  ```

- Módulo `shell`

```
ansible all -m shell -a 'echo $PATH'
```

Coge la configuración del acceso remoto definida en `/etc/profile`. Si necesitamos algo con un `PATH` más complicado, podemos indicar el PATH completo o configurar el equipo antes.

Lo normal es configurar el equipo en dos fases:

1. Configurar usuarios, paths.
2. Realizamos el resto de configuraciones.

El `-m shell` puede omitirse porque es el módulo por defecto.

- Módulo de configuración de servicios, equivalente a `systemctl`.

  Es importante que los verbos son *stopped*, no *stop*: Ansible lleva una programación declarativa (describe el estado al que queremos llevar el sistema). Es decir, si ya está parado no hace nada y si no, lo para. El interés es **llevar el equipo a un estado**: por eso indicamos en qué estado lo queremos, no los pasos que queremos seguir para alcanzarlo (programación procedimental).

  - Apagar Apache

    ```
    ansible cenise -m service -a "name=httpd state=stopped"
    ```

    Como nos estamos conectando con un usuario no privilegiado, puede que no nos deje apagar el servicio. Los módulos de Ansible hacen llamadas a programas que existen (por ejemplo, vemos que en el error tenemos `systemctl`). Podemos hacer nuestros propios módulos basados en llamadas en remoto, pero es importante que las acciones deben ser **idempotentes**: el efecto en el sistema debe ser el mismo independientemente del número de repeticiones en una llamada.

    Si pusiésemos `started` nos daría una confirmación, porque no requiere privilegios.

    Cómo hacer acciones que requieren un usuario privilegiado:

    ```
    ansible ubudef -m ping --become
    ```

    Nos dice que nos falta la contraseña: *missing sudo password*. Necesitamos configurar usuarios que puedan convertirse en root sin contraseña. Para ello debemos modificar el fichero `/etc/sudoers`. Modificamos `admin`:

    ```
    %admin ALL=(ALL) NOPASSWD:ALL
    ```

    ```
    groupadd admin
    ```

    ```
    usermod -G admin david
    ```

    Añadimos el grupo `admin` como grupo secundario. En CentOS el grupo ya viene preparado: es `wheel` (ver `sudoers` en CentOS). Volvemos a repetir con `--become` y vemos que funciona.

    Ahora añadimos el `--become` y listo.

- Definir variables por equipo (el nombre de la variable puede ser el que queramos)

```
[lamp]
cenise http_service='httpd'
ubuise http_service='apache2'
```

Esto es porque los servicios son distintos en Ubuntu y en CentOS. En la cadena hacemos una clave de reemplazo:

```
ansible lamp -m service -a "name={{http_service}} state=stopped" --become
```