Proyecto Integrado

Admón. de Sistemas Informáticos en Red

NETWORKING AUTOMATIZADO

Solucionando los problemas de la infraestructura clásica

AUTOR:

Javier Sánchez Páez

TUTOR/ES:

Víctor Montero Malagón

Javier Pastor Cascales

IES Zaidín-Vergeles (Granada), curso 2021-2023

Agradecimientos

A mi familia, porque sin ellos no estaría donde estoy.

A Carlos, por ser mi mentor desde que tengo uso de razón.

A Vicky, por darme la paz y la guerra que necesito para seguir adelante.

A Kyndryl y al equipo de Network & Edge, por enseñarme y confiar en mí.

A los buenos profesores que me han acompañado, por enseñarme y mantenerme motivado.

A mis amigos y amigas, por ser quienes son y por estar siempre ahí.

Abstract

Montar una infraestructura de red "legacy" conlleva varios problemas:

- **Tiempo perdido y dificultad:** Una red es difícil de montar y configurar. Un ingeniero de red tarda aproximadamente horas en abastecer y preparar una red.
- **Dinero perdido:**
- **Solución de errores:** Un ingeniero de red pierde alrededor del 50% de su tiempo solucionando problemas que surgen en la red.
- **Métricas perdidas:** El 80% de los ingenieros de red comprueban la efectividad de la red mediante CLI. Además, el 40% de los ingenieros de red afirman que muchos de los problemas que un cliente puede tener con la red no necesariamente es por la red.
- **Escalabilidad:** Ampliar una red ya montada es muy complejo y ésta es propensa a errores que afecten al funcionamiento general de la misma.

A raíz de estos problemas surge una solución, la **automatización de redes**. De esta forma podemos solucionar los problemas anteriores:

- **Tiempo perdido y dificultad**: La red podrá ser abastecida de forma general o específica por dispositivo ahorrando tiempo en tareas repetitivas.
- **Dinero perdido**:
- **Solución de errores:** Si tenemos errores podemos hacer "vuelta atrás" (rollbacks), además de poder detectar de dónde viene el error gracias a las **métricas**.
- **Métricas perdidas**: Podemos comprobar la efectividad y flujo de cada uno de los dispositivos de nuestra red y ver si, efectivamente, los fallos que pudieran haber pueden ser causados por nuestra infraestructura.
- **Escalabilidad:** Si es necesario ampliar la red es muy sencillo.

Palabras clave

(GitHub, Ansible AWX, Grafana, Kubernetes, Cisco IOS, Netbox)

Contenido

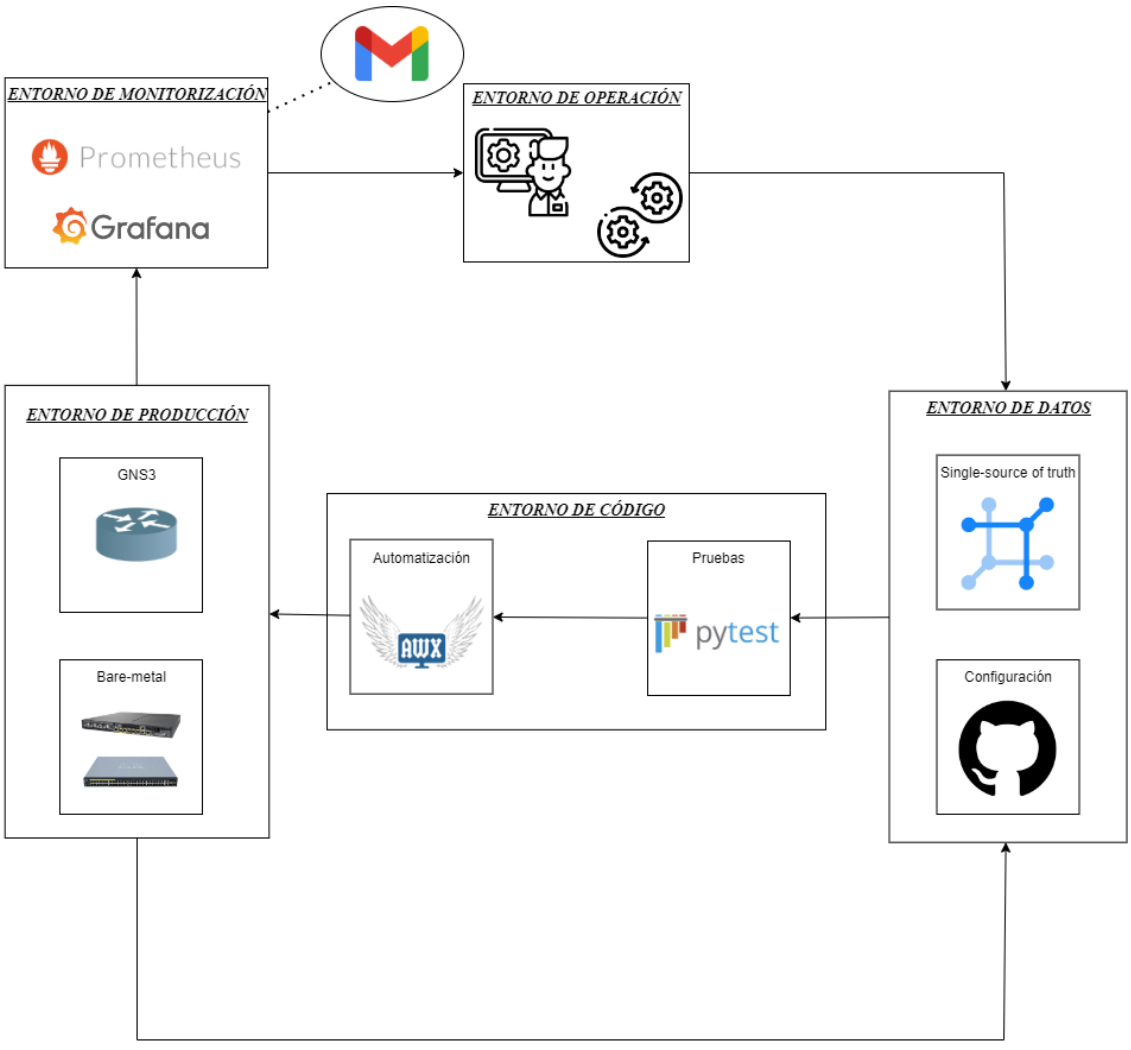
Introducción

A lo largo de los años ha habido una evolución muy marcada en el ámbito de la programación y de la informática en general, aunque siempre se ha obviado la parte más crítica: la infraestructura de red.

La infraestructura de red, como ya sabemos, es difícil de operar, mantener, actualizar y evitar errores. Lo que se mostrará en este documento es cómo podemos utilizar las novedades del ámbito de programación en el campo de las infraestructuras de redes.

Objetivos

El objetivo de modernizar la infraestructura de red es ser capaces de montar un entorno cómodo de operar para ingenieros de red, a su vez que es lo suficientemente avanzado como para añadir cambios seguros. Para ello nos basaremos en el siguiente diagrama:



- **Entorno de operación → entorno de datos**
  El administrador o ingeniero de red se comunicará directamente con nuestras herramientas de datos (NetBox para inventariado de red y GitHub para configuraciones). De esta manera no necesitará involucrarse en todas las fases del despliegue.

- **Entorno de datos → entorno de código**
  Los datos de estas herramientas serán revisadas por pytest para evitar posibles errores.

- **Entorno de código → entorno de producción**
  Una vez esté comprobada la eficacia del código, AWX se encargará de pasar los cambios a producción.

- **Entorno de producción → entorno de monitorización**
  Prometheus y Grafana recogerán los datos de monitorización del entorno de producción.

- **Entorno de monitorización → entorno de operación**
  El administrador o ingeniero de red recibirá los datos de monitorización de los dispositivos de red de producción para poder observar fallos fácilmente.

- **Entorno de producción → entorno de datos**
  En NetBox tendremos inventariado el total de dispositivos de red que utilicemos en producción.

Ajustes previos

Para poder desarrollar este proyecto se han utilizado los siguientes hosts:

Hosts	Provider	CPUs	RAM	Versión S.O.	Info extra
ASUS M3500QC	-	8n/16h	16GiB	Windows 11	E.Operación
Proliant ML150	-	4n	16GiB	Proxmox	GNS3
K8s-rancher	VirtualBox	4n	8GiB	Ubuntu22.04	AWX/Métricas
Ubuntu SRV	VirtualBox	4n	4GiB	Ubuntu22.04	NetBox
GNS3	Proxmox	2n	8GiB	-	Opcional
Cisco 7200	GNS3	1n	512MiB	IOS 15.2	Virtual

Para poder montar toda la infraestructura necesaria podemos hacerlo de dos formas: utilizando Linux como sistema operativo principal o utilizando el Subsistema de Windows para Linux (WSL). En mi caso utilizaré la primera opción por:

- Facilidad de uso.
- Evitar concurrencia de datos al hacer dual boot.
- Mucha documentación disponible.
- Evitar problemas de networking que tiene WSL.

Al no utilizar Linux nativamente desde mi máquina principal, he configurado dos servidores con Ubuntu Server 22.04: Uno de ellos para el clúster de Rancher K3s y otro para Netbox y otras plataformas que podamos llegar a utilizar.

Las razones de utilizar K3s en lugar de un clúster "típico" de K8s son:

- Facilidad de uso.
- Ahorro de recursos del sistema.
- Uso de una máquina en lugar de mínimo 3.
- Mismo resultado.
- Simplicidad.

Además, al contrario que otras soluciones locales basadas en Kubernetes, Rancher nos permite abrir los puertos para nuestras aplicaciones de forma muy sencilla utilizando los mismos servicios que utiliza Kubernetes por defecto (otras alternativas como Minikube necesitan cierta configuración extra específica, como Ingress).

Más adelante veremos qué es GNS3 y por qué es opcional.

Configuración de la red de VirtualBox

Más adelante utilizaremos conexiones entre distintos equipos para conectar APIs, conectar a dispositivos de red, etc, por lo que antes de empezar configuraremos VirtualBox para que cada una de las máquinas virtuales cuenten con 2 tarjetas de red: una configurada en "bridge" para salir a la red exterior con dhcp y otra red "sólo-host" con IP fijas para las conexiones internas. Para ello, una vez tengamos creadas las máquinas virtuales, iremos al apartado de "red". Una vez ahí, le daremos a la sección "Adaptador 1" (estará habilitada por defecto) y en modo de red Adaptador puente. Escogemos la tarjeta de red de preferencia (en mi caso Intel AX210). Repetimos el mismo proceso con el adaptador 2, habilitándolo y eligiendo la opción "Host-Only Ethernet Adapter".

Encendemos ambas máquinas virtuales y comprobamos que las dos tarjetas de red están instaladas (no hace falta que dé internet, ya que nosotros nos encargaremos de configurarlas). Para ello, ejecutaremos "`ip a | grep enp0s`", ya que las tarjetas de red (en mi caso) se llaman `enp0s3` y `enp0s8`. Si salen ambas podemos seguir adelante.

Empezaremos configurando, por ejemplo, el host "`k8s-rancher`". Para ello, abriremos con el programa "`nano`" el archivo "`/etc/netplan/00-installer-config.yaml`". Ahí veremos la configuración por defecto que crea netplan al instalar Ubuntu Server. Reemplazaremos el contenido del archivo por el siguiente:

```
network:
  ethernets:
    enp0s3:
      dhcp4: true
      nameservers:
        addresses: [8.8.8.8]
    enp0s8:
      dhcp4: false
      addresses: [192.168.56.102/24]
  version: 2
```

En este archivo estamos definiendo que en la interfaz enp0s3 (la que sale a la red mediante adaptador puente) dejaremos la config por defecto con DHCP habilitado y las DNS de Google. En la interfaz enp0s8 configuramos que no vamos a utilizar DHCP y que la dirección IP será la 192.168.56.102 con máscara 255.255.255.0. No ponemos valores de DNS porque esa red no sale a internet.

Guardamos el archivo y ejecutamos el comando "`sudo netplan try`". Este comando pasa un check por el archivo de configuración y nos confirma si está bien compuesto o si tiene errores. Si es correcto ejecutamos "`sudo netplan apply`" para aplicar cambios.

Tomamos la otra máquina virtual y copiamos el mismo archivo de arriba, esta vez con una IP distinta. Probamos, aplicamos y hacemos ping entre las dos máquinas virtuales. Para ello ejecutaremos desde la primera máquina virtual "ping 192.168.56.103" para lanzarle un ping a la segunda. Debería funcionar correctamente.

Configuración e instalación de Rancher K3s

La razón por la que aumenté los valores de CPU y RAM de nuestro servidor de Rancher K3s es para asegurar que la infraestructura no se quedara colgada cuando montáramos varios "pods". El almacenamiento también se vio modificado para poder almacenar todos los datos necesarios y la red fue configurada por defecto en adaptador puente para poder acceder a los recursos entre sí y a los dispositivos de red.

Más allá de estos cambios, la instalación del sistema operativo que utilizará de base (Ubuntu Server) será "vanilla", salvo porque instalaremos un servidor OpenSSH para poder acceder cómodamente desde nuestro cliente. Llamaré a mi usuario "jsp" y el nombre del equipo será "pi".

Una vez tengamos instalado nuestro Ubuntu Server, podemos proceder con la instalación de Rancher. Estando en la terminal de nuestro servidor, ejecutaremos el comando "`curl -sfL https://get.k3s.io | sh -`". Lo que este comando hará es descargarnos el script de instalación y pasárselo como entrada a una terminal de sh. Cuando la instalación haya finalizado, el output debería salir "`[INFO] systemd: Starting k3s`"

Si intentamos hacer un "`kubectl version -short`" nos dará error, puesto que no tenemos los permisos para poder ejecutar el comando. Para arreglarlo, cambiaremos el dueño del archivo "`k3s.yaml`" con "`sudo chown user:group /etc/rancher/k3s/k3s.yaml`". Ahora si hacemos un "`kubectl version -short`" nos deberían salir las versiones del cliente, Kustomize y del server (en mi caso v1.26.3).

Una vez hecho esto, vamos a comprobar que nuestro "single-node cluster" se esté ejecutando con "`kubectl get node`". En caso de que así sea, nos aparecerá nuestro server como Ready con el tiempo que lleva activado y su versión.

Dos detalles antes de empezar: Vamos a copiar el contenido del archivo k3s.yaml en otro archivo al que llamaremos "config" y se encontrará en la carpeta "$HOME/.kube". Esto nos hará falta más adelante para evitar errores de conexión, por lo que escribiremos en una terminal "`cp /etc/rancher/k3s/k3s.yaml ~/.kube/config`". Lo segundo es que en nuestro archivo ".bashrc" vamos a escribir dos nuevas líneas de código: "`alias k="kubectl""` y "`export KUBECONFIG=/etc/rancher/k3s/k3s.yaml`". La primera línea nos servirá para llamar al comando kubectl únicamente escribiendo k, lo cual nos ayudará en términos de eficiencia (de aquí en adelante me referiré a kubectl como "k" en los comandos). La segunda nos evitará unos warnings muy molestos cuando hagamos un "`kubectl get nodes`" o cualquier comando en el que accedamos a recursos de nuestro clúster.

Instalación de Helm

Más adelante en este proyecto instalaremos un programa en nuestro clúster mediante Helm. En resumen, Helm es un instalador de paquetes para Kubernetes "pre-configurado", único y fácil de adaptar a diferentes distribuciones. Helm funciona mediante "Charts", que básicamente son los repositorios de donde coge los programas y los depliega en el clúster. Es una herramienta ampliamente utilizada y reúne muchos tipos de programas, desde métricas hasta runtimes de seguridad (lo que instalaremos en nuestro caso).

Antes de proceder a la instalación de Helm veremos si lo tenemos instalado con "`sudo snap list`". Lo normal es que no esté instalado, pero en caso de que los estuviera nos saldría en una línea del output. Si no, lo instalaremos con el comando "`sudo snap install Helm –classic`". Ejecutaremos un "Helm version" en la terminal y nos debería salir un output de la versión: en mi caso estamos en la versión 3.10.1. Una vez instalado Helm, podemos continuar.

Instalación de AWX

Para poder seguir adelante, vamos a hacer un pequeño repaso a qué es Ansible, de dónde salen y en qué se diferencian AWX y Ansible Tower.

Ansible es una herramienta de código abierto que permite la gestión de configuración y la orquestación de tareas en entornos distribuidos, lo que la convierte en una solución eficiente y muy potente para automatizar la implementación y gestión de infraestructuras IT.

Su funcionamiento se divide en tres elementos:

- **Playbooks**:
  Un playbook es, esencialmente, el "set" de instrucciones que queremos ejecutar contra un determinado host. Un ejemplo de playbook que utilizaremos en este proyecto es, por ejemplo, asignar una dirección IP a un puerto de un router y comprobar que tenga conexión con otro host de esa misma red mandando un ping.

- **Inventarios**:
  En un archivo de inventario escribiremos en qué hosts vamos a realizar las tareas de automatización. Dentro de un inventario de Ansible podemos establecer variables, como por ejemplo con qué usuario y qué clave entraremos al modo de configuración de un router.
  Un inventario puede ser de tres maneras distintas: Estático (los hosts se escriben directamente en el archivo), dinámico (los hosts se reciben a través de una API de otro servicio, como por ejemplo Netbox) o inteligente

- **Roles**:
  Los roles en Ansible sirven para agrupar los hosts en diferentes nombres. Esto puede ser muy útil si, por ejemplo, queremos hacer un cambio en todos los hosts, ya que de otra manera tendríamos que ir uno por uno. También es muy útil para agrupar los hosts útiles (en producción) de los inútiles (desconectados, averiados, en almacén…)

Teniendo esto en cuenta, AWX y Tower no son más que interfaces gráficas para Ansible con ciertas herramientas para hacer CI/CD. La diferencia entre AWX y Tower es semejante a la que hay entre Fedora y Red Hat: esencialmente son lo mismo, pero AWX es gratuito y libre de usar; sin embargo, tú eres el responsable de que todo funcione. Ansible Tower es de pago pero tenemos la garantía de que, si algo no funciona, tenemos al equipo de Red Hat disponible para ayudarnos.

Ahora vamos a lo importante: Levantar el operador de AWX. El operador es el "arquitecto", es decir, a él le mandaremos la tarea de instalar AWX con todas las dependencias necesarias (base de datos PostgreSQL, servidor web, etc). Para ello, crearemos un archivo "kustomization.yml" con el siguiente contenido:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - github.com/ansible/awx-
    operator/config/default?ref=0.28.0
    # - awx.yml
images:
  - name: quay.io/ansible/awx-operator
    newTag: 0.28.0
namespace: pi
```

En este archivo le estamos indicando que cree una customización basada en el código de la versión 0.28.0 del software alojado en GitHub y que la aplique en el espacio de nombres "pi". Una vez creado el archivo, levantaremos el operador con el comando "`k apply -k .`". En el output del comando nos mostrará que se han creado muchos tipos de recursos, así que esperaremos unos segundos.

Vamos a comprobar que el operador haya sido levantado (importante ahora ejecutar kubectl con "-n pi" para referirnos al espacio de nombres que acabamos de crear) con el comando "`k -n pi get pods`". Si nos aparece en Status Running podemos seguir adelante.

Una vez esté ejecutado, vamos a crear un segundo archivo llamado "awx.yml" con el siguiente contenido:

```
apiVersion: awx.ansible.com/v1beta1
kind: AWX
metadata:
     name: pi
spec:
     service_type: nodeport
     nodeport_port: 30080
```

Ahora volvemos al archivo "kustomization.yml" y descomentaremos la línea de awx.yml en metadata, por lo que se quedaría únicamente en "`- awx.yml`"

Volvemos a ejecutar "`k apply -k .`"

Para monitorizar cómo se van creando todos los pods necesarios, podemos ejecutar el comando "`watch kubectl get pods -n pi`":

Después de aprox. 5-7 minutos (dependiendo de los recursos que tenga el equipo donde hayamos montado el clúster) veremos que están ejecutándose los siguientes pods:

- `awx-operator-controller-manager, 2/2, Running`
- `pi-postgres, 1/1, Running`
- `pi, 4/4, Running`

Ahora podemos acceder a la interfaz de inicio de sesión web. Entramos con la dirección IP (localhost en nuestro caso) y el puerto que hayamos configurado en nodeport. Para conseguir la contraseña (el usuario es admin) introduciremos el siguiente comando:

```
K -n pi get secret pi-admin-password -o jsonpath="{.data.password}" | base64 –decode ; echo
```

Nos saldrá la contraseña como output. Iniciamos sesión y ya estaremos dentro. Para cambiar la contraseña yendo a admin → detalles de usuario → editar → contraseña.

Instalación de Grafana y Prometheus

Instalaremos Grafana en nuestro Rancher K3s por simplicidad y porque es la instalación recomendada por los desarrolladores. Para ello, utilizaremos como base un repositorio ubicado en GitHub llamado "Kube-Prometheus". Este contiene toda la configuración necesaria para desplegar Grafana y Prometheus automática y fácilmente, además de venir con ciertas métricas ya preconfiguradas. Para la instalación ejecutaremos los siguientes comandos:

```
$ git clone https://github.com/prometheus-operator/kube-prometheus.git
$ k apply –server-side -f manifests/setup/
$ k apply -f manifests/
$ k -n monitoring delete networkpolicies.networking.k8s.io –all
```

De esta forma instalaremos el operador de Prometheus y Grafana con todas las dependencias necesarias. Sin embargo, tenemos que sustituir tres archivos de los que hemos aplicado en el clúster: estos corresponden a los servicios a los que nos queremos conectar (es decir, alert-manager, Prometheus y Grafana). Podemos o bien reemplazarlo en la misma ejecución del clúster o crear un nuevo archivo .yml y aplicarlo de nuevo (el clúster identificará y reconocerá el nuevo servicio y sabrá que es la sustitución de un recurso ya existente). Dicho lo cual, crearemos los siguientes archivos:

Alertmanager-service.yaml

```
--
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: alert-router
    app.kubernetes.io/instance: main
    app.kubernetes.io/name: alertmanager
    app.kubernetes.io/part-of: kube-prometheus
    app.kubernetes.io/version: 0.25.0
  name: alertmanager-main
  namespace: monitoring
spec:
  type: LoadBalancer
  ports:
  - name: web
    port: 9093
    targetPort: web
  - name: reloader-web
    port: 8080
  selector:
    app.kubernetes.io/component: alert-router
    app.kubernetes.io/instance: main
    app.kubernetes.io/name: alertmanager
    app.kubernetes.io/part-of: kube-prometheus
```

grafana-service.yaml

```
--
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: grafana
    app.kubernetes.io/name: grafana
    app.kubernetes.io/part-of: kube-prometheus
    app.kubernetes.io/version: 9.5.2
  name: grafana
  namespace: monitoring
spec:
  type: LoadBalancer
  ports:
  - name: http
    port: 3000
    targetPort: http
  selector:
    app.kubernetes.io/component: grafana
    app.kubernetes.io/name: grafana
    app.kubernetes.io/part-of: kube-prometheus
```

prometheus-service.yaml

```
--
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/instance: k8s
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
    app.kubernetes.io/version: 2.44.0
  name: prometheus-k8s
  namespace: monitoring
spec:
  type: LoadBalancer
  ports:
  - name: web
    port: 9090
    targetPort: web
  - name: reloader-web
    port: 8080
    targetPort: reloader-web
  selector:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/instance: k8s
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
```

Lo que estamos haciendo aquí es hacer un servicio "LoadBalancer" a raíz de cada uno de esos servicios existentes pero que no salían a la red externa. Ahora ejecutaremos "`k apply -f ./`" y se nos aplicarán los nuevos archivos: veremos que esta vez en lugar de aparecer los recursos como creados aparecerán como configurados. Para ver qué puerto se nos ha asignado, pondremos en la terminal "`k get svc -n monitoring`" (en mi caso, por ejemplo, Grafana se ha asignado al puerto del host 31469. Además, es necesario eliminar las políticas de red que crea porque si no nos deja conectarnos a los servicios desde la red externa al servidor. Luego de un minuto, tendremos la aplicación y un puerto aleatorio disponibles para conectarnos a Grafana (en mi caso). Iniciamos sesión con las credenciales admin@admin y, si queremos, le cambiamos la contraseña. En mi caso dejaré la contraseña admin. Con esto ya tendremos Grafana instalado y preparado para configurar.

Instalación de Netbox

Como mencionamos anteriormente, esta máquina virtual que utilizaremos con Netbox está basada en Ubuntu 22.04 y, al igual que el clúster de Rancher K3s, tiene una configuración de red en puente para que podamos poblar el servidor fácilmente. También fueron modificadas las características de CPU y RAM pero en menor medida que en el clúster.

Para empezar nos conectaremos por SSH y clonaremos el repositorio de Netbox. Utilizaremos la imagen de Docker en lugar de la instalación "legacy" por simplicidad, velocidad y estabilidad; si queremos modificar algo es muy sencillo. Ejecutaremos el script que he creado en bash, está en el repositorio de GitHub y aparte de clonarnos el repo, nos pide en qué puerto del host queremos montarlo, lo ejecuta y crea un servicio de systemd para que se ejecute cada vez que encendamos el equipo.

Al cabo de un momento si entramos en nuestra IP:puerto veremos la interfaz de Netbox.

Para poder acceder necesitamos ejecutar en el contenedor de netbox el archivo "manage.py" para crear un superusuario, por lo que ejecutaremos "`docker compose exec netbox /opt/netbox/netbox/manage.py createsuperuser`"

Nos pedirá el nombre de usuario (admin en mi caso), email y contraseña. Una vez terminado, tendremos el usuario creado y podremos acceder más tarde.

Instalación y configuración de GNS3

Durante lo que llevamos de documento hemos visto cómo preparar las plataformas de software para adaptarlas a nuestro flujo de trabajo enfocado a hacer cambios en la red, pero todavía no hemos respondido a la pregunta principal: ¿Cómo lo vamos a hacer si no tenemos ningún dispositivo físico? Aquí es donde entra GNS3:

GNS3 es una plataforma que nos va a ayudar virtualizando los diferentes dispositivos que vayamos a montar en nuestro supuesto entorno de red. Podemos montar desde firewalls hasta routers, switches, VPNs… Incluso máquinas virtuales con sistemas operativos de escritorio.

Voy a utilizar GNS3 porque no cuento con dispositivos para aprovecharlos en este proyecto. Esta herramienta es una parte del proyecto **opcional** y que en un entorno de trabajo real no sería necesario.

Podemos instalarlo de dos formas: o bien con una imagen ISO y montamos una máquina virtual con ella o con una OVA (incluye una máquina con el sistema instalado y preparado para correr). Para montar la MV con OVA es necesario instalar VMware, el cual actualmente me da problemas con el Subsistema de Windows para Linux y VirtualBox, por lo que lo instalaré mediante la OVA en un servidor aparte con Proxmox ya que es compatible (gracias al subdirector José Luis Navarro por darme el equipamiento necesario en este aspecto).

Tan sólo tendremos que importar la OVA a nuestro servidor y asignar los discos a nuestra MV y ya lo tendremos funcionando. Lo que sí haremos es instalar el programa cliente en nuestro equipo Windows ya que, aunque cuente con un servidor web integrado, la aplicación es más completa. Iremos al repositorio oficial de Github e instalaremos la versión de Windows.

Seleccionaremos que nos conectamos a un servidor remoto a través de los ajustes, en mi caso la IP es 192.168.1.200:80 y el usuario/password por defecto es gns3/gns3.

Una vez conectados, vamos a crear un nuevo proyecto llamado Proyecto Integrado.. Una vez dentro arrastraremos un Cloud (que hace referencia al propio puerto Ethernet de nuestro ordenador) y lo conectamos a un router. Yo usaré unas imágenes que Cisco proporciona para aprendizaje y testing. Nuestra topología de red será un router conectado mediante el puerto FastEthernet0/0 directamente a la nube en Ethernet0/0.

Configuración del repositorio de GitHub con AWX

Utilizaremos el repositorio que ya tenemos creado del proyecto para subir el contenido de Ansible y Netbox (tanto para la infraestructura como para el código).

Empezaremos añadiendo el repositorio en AWX. Iniciamos sesión y añadiremos una nueva organización. Esto nos sirve para aislar los recursos de este proyecto del resto de recursos de la plataforma. Iremos a Organizaciones y Añadir. Los valores de "Entorno de ejecución" y "Credenciales de Galaxy" serán por defecto.

Una vez añadida la organización añadiremos unas credenciales para poder acceder al repositorio (esto es obligatorio si tenemos el repositorio privado como es el caso. Si fuera público no es necesario).Crearemos un par de claves con el comando `ssh-keygen`. La clave pública la añadiremos a nuestro perfil de GitHub y la privada a AWX.

Ahora crearemos un proyecto. Es la base sobre la que funcionará todo el workflow de AWX y donde pondremos el repositorio con el que estamos trabajando.

Nada más añadirlo se nos ejecutará un trabajo para recoger todos los datos de Git. Si todo se ejecutó correctamente, nos aparecerá el mensaje "PLAY RECAP localhost ok" sin ningún error, por lo que ya tenemos AWX conectado a nuestro repositorio.

Poblando el inventario Netbox con dispositivos GNS3

Antes de poder poblar el inventario de AWX tendremos que introducir los dispositivos en Netbox, ya que entonces no tendremos manera de saber si estamos haciendo el inventario, además de que estos dispositivos son los que posteriormente configuraremos en las pruebas. Para ello utilizaremos una plataforma de virtualización de redes llamada GNS3. La funcionalidad de esta plataforma es poder virtualizar routers, switches, firewalls, etc. Sin necesidad de tener dispositivos físicos.

En GNS3 hemos montado un único router Cisco 3600 Series. Para poder crear un dispositivo único tendremos que crear los siguientes requisitos:

- Fabricante
- Tipo de dispositivo (Device Type)
- Sitio (Site)
- Rol de dispositivo (Device Role)

Iremos de uno en uno por encima viéndolos. En fabricante únicamente indicaremos datos muy básicos del fabricante (en este caso Cisco). Todos los datos que meteré será mediante archivos YAML, así que para fabricante se nos quedaría un archivo parecido al siguiente:

```
cisco.yml
---
name: Cisco
slug: cisco
```

El "slug" es un nombre fácil de recordar para una URL y a su vez funciona de ID único.

Importaremos el archivo accediendo a NetBox y abriendo la pestaña perteneciente a cada tipo de configuración, en este caso Fabricante.

En Device Type nos pide unos datos más así que el archivo debe ser algo así (todas estas características dependerán del router que hayamos introducido).

```
Cisco-catalyst-3600.yml
---
manufacturer: Cisco
model: Catalyst 3600
slug: cisco-catalyst-3600
part_number: CATALYST-3600
is_full_depth: true
u_height: 1
interfaces:
  - name: FastEthernet0/0
    type: 100BASE-TX (10/100ME)
  - name: Ethernet1/0
  - type: 100BASE-TX (10/100ME)
```

Ahora vamos a añadir un **sitio** y un **rol de dispositivo**. Allí es donde configuraremos cada dispositivo específico con sus rangos IP, su función en el sitio, etc.

```
proyecto_integrado.yml
---
name: Proyecto Integrado
slug: proyecto-integrado
status: active

lab.yml
---
name: Lab
slug: lab
color: 1f54ab
vm_role: false
```

Una vez creada la base, vamos a añadir dispositivos específicos (en este caso Router1):

```
router1.yml
---
name: Router1
device_role: Lab
manufacturer: Cisco
device_type: Catalyst 3600 Series
status: active
site: Proyecto Integrado
```

Ya sólo nos falta añadir las direcciones IP del dispositivo. De momento sólo configuraremos una, ya que el otro puerto lo configuraremos mediante AWX.

```
router1-ip.yml
---
address: 192.168.1.170/32
status: active
device: Router1
interface: FastEthernet0/0
is_primary: true
```

En el siguiente punto aprenderemos a crear la conexión entre AWX y Netbox y probaremos el inventario dinámico.

Configuración del inventario de Netbox con AWX

Para saber a qué dispositivos les vamos a aplicar cambios y, como mencionamos anteriormente, vamos a usar un inventario dinámico. Lo primero que haremos será ponerle un nombre (en mi caso Proyecto Integrado) y, luego, añadir el "source code":

Vamos a organizar todos los recursos de AWX en el repositorio en tres carpetas: Collections (donde irán las colecciones de Ansible Galaxy de Plugins que necesitemos, empezaremos por aquí), Inventories (donde irá el inventario) y Playbooks (donde irán las tareas). Dentro de la carpeta Inventories crearé un archivo "netbox.yml" con el siguiente contenido:

```
plugin: netbox.netbox.nb_inventory
api_endpoint: XXX.XXX.XXX.XXX:puerto
token: XXXXXXXX
validate_certs: false
interfaces: true
grouo_names_raw: true
group_by:
  - device_roles
  - sites
  - device_types
  - manufacturers
  - platforms
device_query_filters:
  - has_primary_ip: false
```

En api_endpoint y token pondremos la URL:puerto y el token de Netbox respectivamente.

Pasaremos a Netbox y vamos a extraer su token y su URL.

Como mencioné antes, empezaremos creando el archivo de requisitos de colección porque necesitamos el plugin de Netbox para que funcione. Es obligatorio que la carpeta esté en la raíz del repositorio para que lo detecte. Escribiremos un archivo "requirements.yml" con el siguiente contenido:

```
requirements.yml

---

collections:

  - netbox.netbox
  - community.general
```

La colección community.general la utilizaremos más adelante.

Una vez hecho el archivo, vamos a extraer un token de la API de Netbox. Para ello, en NetBox iremos a nuestro usuario → API Tokens. Crearé una con validez hasta fin de año, aunque podemos ajustarla al tiempo que queramos.

Introduciremos en el archivo de inventario el token que se nos habrá creado además de la dirección IP. Una vez hecho eso, vamos a probar a actualizarlo dándole al botón azul de Sync.

Si la ejecución fue correcta, nos saldrá un mensaje de OK y nos aparecerá nuestro router en la pestaña Hosts.

Configurando el router para el primer acceso

Volveremos a utilizar GNS3, esta vez para configurar el acceso desde la red, crear un inicio de sesión y asignar una dirección IP, todo ello para poder utilizarlo con Ansible.

Iniciaremos GNS3 y montaremos un recurso de Cloud, nos servirá como puerta para entrar desde nuestra red/Ansible. Una vez insertado, pondremos un router Cisco 3600 y lo conectaremos por el puerto que queramos, en mi caso por el FastEthernet0/0.

Entraremos a la terminal integrada del router que nos proporciona GNS3 y configuramos la dirección IP, que como mencionamos anteriormente es la 192.168.1.170.

```
R1(config)# ip address 192.168.1.170 255.255.255.0
R1(config)# hostname Router1
Router1(config)# ip domain name pi
Router1(config)# crypto key generate rsa
How many bits in the modulus [512]: 2048
SSH1.5 has been enabled
Router1(config)# line vty 0 4
Router1(config-line)# transport input ssh
Router1(config-line)# login local
Router1(config)# username admin password admin
Router1(config)# enable password cisco
Router1# copy run start
```

Lo que hemos hecho aquí ha sido poner el nombre del dominio (es necesario), añadir un par de claves SSH y establecer SSH como protocolo de login por defecto. También añadimos un usuario admin@admin para entrar con él por SSH y habilitamos la contraseña "cisco" para entrar al modo enable. Con todo esto si podemos entrar desde nuestro equipo personal también podremos acceder desde AWX más adelante.

Cambiando parámetros del router con AWX

Llegados a este punto, vamos a hacer los primeros cambios en nuestro router directamente desde AWX. Para ello escribiremos "playbooks" que este ejecutará y accederá al router. Antes de empezar a escribirlos tendremos que configurar el plugin de Cisco en el archivo de colecciones y añadir **variables** con el usuario, passwords, etc del router.

Lo primero será modificar el archivo requirements.yml. Debe quedar parecido a esto:

```
collections:
  - netbox.netbox
  - community.general
  - cisco.ios
```

Si hacemos push al repositorio y actualizamos el proyecto desde AWX veremos que se ha instalado el nuevo plugin.

Ahora vamos a crear la credencial en AWX. Iremos al desplegable de la parte izquierda llamado "Credenciales". Haremos click e indicaremos que es una credencial tipo Máquina. En mi caso pondré "admin@admin" y la contraseña para elevación de privilegios es "cisco". Es muy importante en esta configuración ajustar el método de escalación de permisos en "enable", ya que así detectará que nos estamos conectando a un router. Una vez hecho vamos a crear el playbook, y empezaremos escribiendo uno muy simple que nos creará un Banner MOTD. De esta manera podremos ver que se efectúan los cambios en nuestro workflow.

Ejecución de un playbook simple

Como comentamos, ahora vamos a ejecutar un playbook simple pero suficiente como para ver que AWX es capaz de conectarse a nuestro router virtual y cambiarle la configuración. En este caso le pondremos un MOTD, pero podemos crear lo que se nos ocurra:

```
General.yml
---
- name: Configuración General
  hosts: cisco
  tasks:

  - name: Añadir Banner
    ios_banner:
      banner: login
      text: |
        Hola Mundo! Si estás
        leyendo esto significa que
        la ejecución del playbook
        fue correcta y podemos
        seguir adelante.
      state: present
```

Una vez creado el archivo podemos crear la plantilla en AWX, a la cual llamaremos de igual manera. Es muy importante al crear la plantilla asegurarnos de que se usan las credenciales que creamos en el punto anterior y que es capaz de detectar nuestro playbook; si no, bastará con volver a sincronizar el proyecto. Vamos a ejecutar el playbook y, si todo ha salido bien, nos saldrá un mensaje de OK[1] en color verde.

Para comprobar que esto ha funcionado, saltaremos al router y podremos visualizarlo de dos formas distintas:

- El router nos da el MOTD nada más conectarnos. En mi caso no lo va a hacer porque está configurado desde mi anfitrión.
- Accediendo a la configuración que se está ejecutando. Es decir, ejecutaremos un show running-config (o sh run).

Como he comentado, voy a hacerlo de la segunda manera. Accedo al router y veremos esto en la zona media del output:

```
!
banner login ^C
Hola Mundo! Si estC!s
leyendo esto significa que
la ejecuciC3n del playbook
fue correcta y podemos
seguir adelante.
^C
```

Las tildes no se podrán mostrar correctamente por temas de codificación, pero ahora podemos ver cómo la funcionalidad básica del workflow funciona. Vamos a hacer un repaso de lo que llevamos:

- ENT. OPERACIÓN -> ENT. DATOS
  El ingeniero de redes puede acceder tanto a Netbox para cambiar la información de los routers como a GitHub para cambiar la configuración de estos.

- ENT. OPERACIÓN -> ENT. CÓDIGO
  AWX recibe el código de GitHub y los datos de NetBox y los guarda para su posterior uso.

- ENT. OPERACIÓN -> ENT. PRODUCCIÓN
  AWX aplica los cambios almacenados en los routers y dispositivos que configuramos en Netbox.

- ENT.PRODUCCIÓN -> ENT. DATOS
  NetBox almacena los datos de los dispositivos que tenemos en producción.

Securizando el clúster de K3s con Falco

En esta sección instalaremos un programa para securizar nuestro clúster llamado Falco. En resumen, Falco es un "runtime" de seguridad que se ejecuta en nuestro servidor en segundo plano y que, mediante logs, nos anunciará de accesos externos al servidor, así como de intentos de conexión y extracción de datos. Para instalar Falco utilizaremos **Helm**, programa que instalamos anteriormente. Lo primero que haremos será crear un namespace para Falco con el comando "`k create namespace falco`". El output de este comando debería indicarnos que se ha creado con la salida "`namespace/falco created`". Una vez creado el namespace, añadimos el chart con el comando "`Helm repo add falcosecurity https://falcosecurity.github.io/charts`", actualizamos los repositorios con "`Helm repo update`" y lo instalamos con "`helm install falco falcosecurity/falco –namespace falco`".

Una vez instalado haremos un "`k get pod -n falco`" para ver si los pods de Falco se están ejecutando ya correctamente. Debería aparecernos en estado "Running" y con "2/2" contenedores ejecutándose.

Vamos a hacer una prueba haciendo un "kubectl get pod" en el mismo servidor del clúster. Al cabo de unos segundos, ejecutamos "kubectl logs falco-xxxxx -n falco" y veremos la hora de la última solicitud y su dirección IP. De esta manera, tendremos nuestro clúster securizado y con capacidad de generarnos logs con posibles errores de seguridad.