

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 un año en abastecer y preparar una red compleja (proxies, routers, compatibilizar infraestructura “legacy”) con diferentes localizaciones.

* **Dinero perdido:** No hacer una correcta planificación de los recursos requeridos para nuestra infraestructura puede implicar, entre otras cosas, perder tiempo útil para otras cuestiones y hasta comprar dispositivos que, al tiempo, serán innecesarios.
* **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.
* **Seguridad:** Al no tener un control completo de nuestra red podemos estar obviando riesgos de seguridad como, por ejemplo, un ataque que nos inhabilite la red por un firewall mal configurado.

A raíz de estos problemas surge una posible solución, la cual es la **automatización de redes**. De esta forma podemos solucionar los problemas mencionados anteriormente:

* **Tiempo perdido y dificultad**: La red podrá ser abastecida de forma general o específica por dispositivo ahorrando tiempo en tareas repetitivas.
* **Dinero perdido**: Al utilizar una herramienta de inventariado específico de redes (NetBox) podemos ver qué es realmente necesario en nuestra infraestructura y qué podemos omitir.
* **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 puedan haber pueden ser causados por nuestra infraestructura.
* **Escalabilidad:** Al tener un control de infraestructura centralizado podemos escalar y ampliar nuestra infraestructura fácilmente, ya que aplicaremos una configuración “base” para todos los dispositivos que sean del mismo tipo.
* **Seguridad**: Al aplicar unas configuraciones “base” que incluyan medidas de seguridad podemos tener la certeza de que si lo escribimos bien en la base funcionará bien en entornos de producción.

# Palabras clave

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

Contenido

[Agradecimientos 2](#_Toc136683494)

[Abstract 3](#_Toc136683495)

[Palabras clave 5](#_Toc136683496)

[Introducción 7](#_Toc136683497)

[Objetivos 7](#_Toc136683498)

[Ajustes previos 9](#_Toc136683499)

[Configuración de la red de VirtualBox 11](#_Toc136683500)

[Configuración e instalación de Rancher K3s 13](#_Toc136683501)

[Instalación de Helm 15](#_Toc136683502)

[Instalación de AWX 16](#_Toc136683503)

[Instalación de Grafana y Prometheus 20](#_Toc136683504)

[Instalación de Netbox 25](#_Toc136683505)

[Instalación de GNS3 26](#_Toc136683506)

[Configuración de GNS3 27](#_Toc136683507)

[Configuración del repositorio de GitHub con AWX 28](#_Toc136683508)

[Poblando el inventario Netbox con dispositivos GNS3 29](#_Toc136683509)

[Configuración del inventario de Netbox con AWX 32](#_Toc136683510)

[Configurando el router para el primer acceso 34](#_Toc136683511)

[Cambiando parámetros del router con AWX 35](#_Toc136683512)

[Ejecución de un playbook simple 36](#_Toc136683513)

[Securizando el clúster de K3s con Falco 38](#_Toc136683514)

[Notificación de cambios del repositorio con GitHub Actions 39](#_Toc136683515)

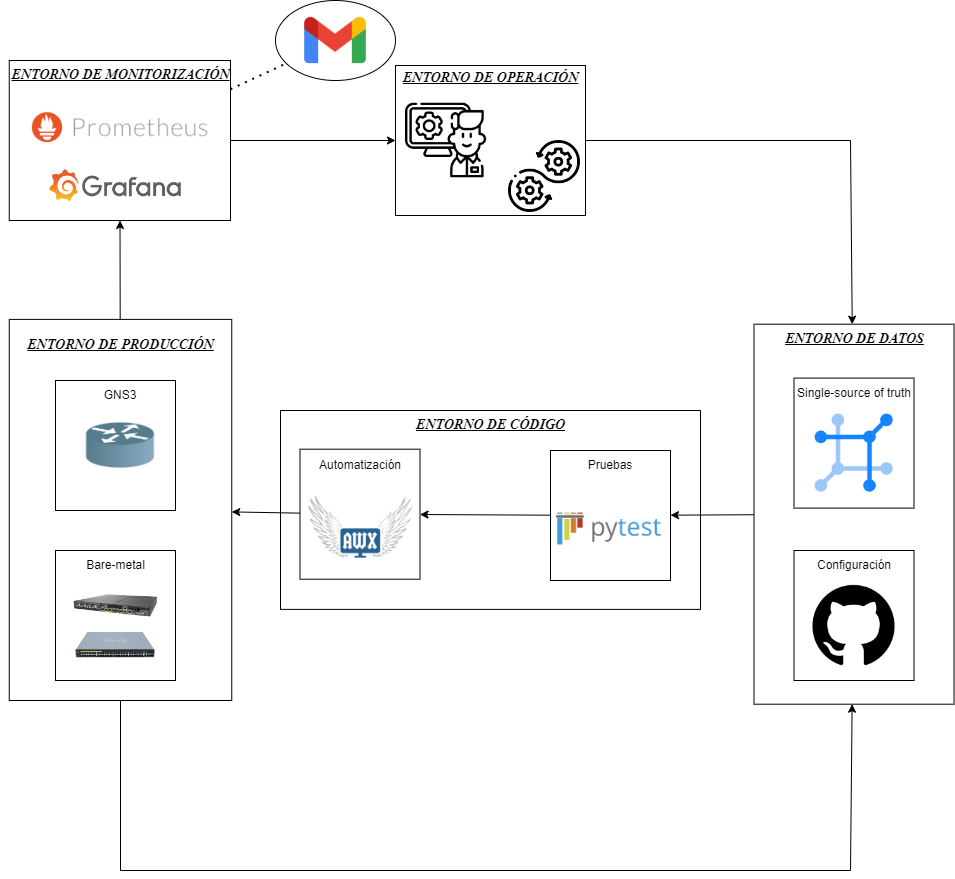
# Introducción

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 **reparar**. 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 directa y únicamente con nuestras herramientas de datos (**NetBox** para **inventariado de red** y el repositorio de **GitHub** para **configuraciones** **y código** que apliquemos). De esta manera **no necesitará involucrarse en todas las fases del despliegue**.

* **Entorno de datos 🡪 entorno de producción**

**AWX** **recogerá** los dispositivos **guardados** en **NetBox** para aplicarles cambios con el código que tomará de **GitHub**.

* **Entorno de producción 🡪 entorno de monitorización**

**Prometheus** y **Grafana** recogerán los datos de **monitorización** del entorno de **producción** y de los **equipos** que estemos utilizando para alojar las diferentes herramientas y servicios.

* **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** en el repositorio y en la infraestructura **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 equipos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hosts** | **Provider** | **CPUs** | **RAM** | **Versión S.O.** | **Info extra** |
| ASUS M3500QC | - | 8n/16h | 16GiB | Windows 11 | Ent. Operación |
| Proliant ML150 | - | 4n | 16GiB | Proxmox | GNS3 |
| K3s-rancher | VirtualBox | 4n | 8GiB | Ubuntu22.04 | AWX/Métricas |
| Ubuntu SRV | VirtualBox | 4n | 4GiB | Ubuntu22.04 | NetBox |
| GNS3 VM | Proxmox | 2n | 8GiB | - | Opcional |
| Cisco 7200 | GNS3 | 1n | 512MiB | IOS 15.2 | Virtual |

Podemos montar todas las herramientas mencionadas anteriormente de varias formas. En este proyecto utilizaremos máquinas virtuales mediante VirtualBox, aunque podemos utilizar un clúster de Kubernetes en Rancher mediante el Subsistema de Windows para Linux (WSL). Los motivos por los que he preferido utilizar máquinas virtuales son los siguientes:

* **Facilidad de uso**: Hay mucha documentación disponible, además de que evitamos posibles fallos e incompatibilidades que podamos tener al montar WSL.
* **Alternativa más cercana a la realidad**: En un entorno de producción tendremos todas las herramientas montadas en servidores “al uso”, no en sistemas operativos domésticos.
* **Protección a errores:** Si por alguna razón se nos corrompe alguno de los servicios que montaremos podemos utilizar las “snapshots” que ofrece VirtualBox para volver a un estado funcional.

Estas máquinas virtuales mencionadas ejecutarán el sistema operativo “Ubuntu Server 22.04” y, en total serán tres:

* K3s-rancher ejecutará el clúster de Kubernetes (comúnmente llamado K8s) junto a AWX, Grafana/Prometheus y Falco.
* Ubuntu SRV ejecutará NetBox, PostgreSQL, Apache2 y Redis.
* GNS3 VM virtualizará de forma anidada la infraestructura de red que montemos.

En un entorno de producción real omitiremos la máquina virtual de GNS3.

Antes de seguir adelante, es importante mencionar una diferencia clave de este clúster de Kubernetes con uno “tradicional”. Para ello, vamos a recordar qué es un clúster de Kubernetes, qué requisitos necesitamos para ejecutar uno y qué nos aporta Rancher (también llamado Rancher K3s o simplemente K3s).

Kubernetes es una plataforma “open-source” de orquestación y gestión de contenedores que nos servirá para, entre otras cosas, conseguir alta disponibilidad y escalabilidad, los cuales son obligatorios para tener un proyecto confiable.

Kubernetes “vanilla” requiere de, como mínimo, **tres hosts**; uno de ellos actuará como el **orquestador**, es decir, organizará las peticiones que le lleguen del exterior y será con quien nos comunicaremos para agregar, modificar o eliminar los recursos que montemos; los otros dos actuarán como **trabajadores** (**workers**), es decir, serán los encargados de ejecutar los recursos como tal.

La ventaja que nos da Rancher K3s es que ­**no nos exige un mínimo de equipos**; es decir, podemos montar un clúster en un solo equipo, el cual actuará de orquestador y de trabajador al mismo tiempo. Este proyecto puede ser lanzado en un clúster “vanilla” de Kubernetes, pero para la carga de trabajo que tenemos en este proyecto es más que suficiente. Además de esta, otras razones por las que utilizar K3s en lugar de un clúster “vanilla” de Kubernetes son:

* Facilidad de operabilidad.
* Ahorro de recursos del sistema.
* Mismo resultado.
* Simplicidad.

Rancher es un “sabor” de Kubernetes aunque no el único, y es que tenemos soluciones como MicroK8s (creado por Canonical, desarrolladores de Ubuntu), K0s, Minikube, Kind, etc. Estas otras soluciones “copian” la misma filosofía de “*single-node cluster*”, aunque para mí Rancher lo hace mejor ya que no necesitas manejar comandos específicos de Rancher: el funcionamiento es exactamente el mismo que el de un clúster “vanilla”.

## Creación del proyecto en GitHub

Para desarrollar este proyecto haremos uso de un repositorio de Git que alojaremos en GitHub. La decisión de utilizar GitHub en lugar de otras opciones como Gitlab o Bitbucket es, principalmente, porque más adelante utilizaremos las llamadas **GitHub Actions**. Además, GitHub es la plataforma más común y utilizada.

Empezaremos yendo a github.com y crearemos una cuenta si no la tenemos. Una vez esté hecha, crearemos un nuevo repositorio dándole al botón verde de la columna izquierda llamado “Nuevo Repositorio”. Se nos abrirá una nueva página preguntándonos por información como el nombre, propietario, etc. En mi caso lo llamaré “PROYECTO\_INTEGRADO”, yo seré el dueño y mi tutor Víctor Montero Malagón tendrá permisos de lectura para poder ver los cambios a medida que vayan saliendo y el tipo de licencia empleada es

## Configuración de la red de VirtualBox

En este proyecto haremos uso de ciertas APIs para conectar los diferentes servicios y entornos que montaremos, por lo que, en caso de que estés utilizando también máquinas virtuales, te recomiendo que, en cada de una de las máquinas virtuales, montes **dos tarjetas de red**; una de ellas en tipo “Adaptador Puente” con IP dinámica (para poder conectarnos a la infraestructura de red) y la otra en tipo “Adaptador Sólo-Anfitrión” con direcciones IP estáticas. De esta manera, los servidores quedarán con una IP privada fija y no tendremos que modificar constantemente los archivos de configuración.

Para hacer estos cambios tendremos que tener nuestras máquinas virtuales previamente creadas. Una vez lo estén, iremos al “tab” de “Red” de cada uno de los servidores en VirtualBox. Comprobaremos que el Adaptador 1 (se genera por defecto) esté en modo de red Adaptador Puente (en mi caso conectado a una tarjeta inalámbrica Intel AX210). Hecho eso, iremos al Adaptador 2, lo habilitaremos y lo pondremos en modo de red Adaptador Sólo-Anfitrión (en la instalación de VirtualBox se nos debería haber generado una red virtual llamado “VirtualBox Host-Only Ethernet Adapter”, que tendrá la red 192.168.56.0/24).

Encendemos ambas máquinas virtuales y comprobamos que las dos tarjetas de red están instaladas con el comando “ip a | grep enp0s” (no hace falta que dé internet, ya que nosotros nos encargaremos de configurarlas). En versiones anteriores de VirtualBox y Linux se pueden llamar de distinta manera; por ejemplo, “eth0” y “eth1”.

Empezaremos configurando, por ejemplo, el host “K3s-rancher”. Para ello, abriremos con el programa “nano” el archivo ”/etc/netplan/00-installer-config.yaml”. Si, en lugar de Ubuntu Server estás utilizando Debian u otra distribución de Linux, puede ser que tengas que modificar otro archivo como el “/etc/network/interfaces”. Ahí veremos la configuración por defecto que crea el sistema al instalar el sistema operativo. Reemplazaremos el contenido del archivo por el siguiente:

00-installer-config.yml

---

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 definimos que, en la interfaz enp0s3 (la que sale a la red mediante adaptador puente), dejaremos la configuración 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 ejecuta un check sintáctico al archivo de configuración y nos confirma si está bien compuesto o si tiene errores. Si es correcto no veremos ningún warning por pantalla, por lo que ejecutaremos “sudo netplan apply” para aplicar los cambios. Podemos volver a listar las tarjetas de red para asegurarnos de que se ha aplicado la configuración.

Ahora pasaremos a la otra máquina virtual (Ubuntu SRV) y copiamos el mismo archivo de arriba, esta vez con una IP distinta (por ejemplo yo he configurado la 192.168.56.103). Probamos, aplicamos y hacemos ping entre las dos máquinas virtuales, deberían tener conexión mutua entre ellas y nuestro host (por defecto viene con la dirección 192.168.56.1).

## Configuración e instalación de Rancher K3s

Como hemos comentado al principio del documento, Rancher K3s es una versión de Kubernetes “single-node”. Esto significa que se van a ejecutar más componentes de lo normal en una misma máquina, por lo que tendremos en cuenta los recursos de CPU y RAM asignados. Es importante también tener en cuenta el almacenamiento que le asignaremos al disco para alojar todos los servicios que montaremos en el clúster (en mi caso con 100GB he tenido de sobra para almacenarlo todo).

Más allá de estos cambios, la instalación de Ubuntu Server 22.04 será la estándar mediante el asistente de instalación de la imagen ISO, salvo porque en la configuración de red marcaremos que queremos que la instalación incluya 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 Ubuntu Server, seguiremos con la instalación de Rancher K3s. Estando en la terminal de bash del servidor, ejecutaremos el comando “*curl -sfL https://get.k3s.io | sh –”.* Lo que este comando hará es almacenar temporalmente el script de instalación y pasarlo como entrada a una terminal de sh. Cuando la instalación haya finalizado, el output debería salir parecido a “[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 la propiedad del archivo “k3s.yaml” con “sudo chown usuario:grupo /etc/rancher/k3s/k3s.yaml”. Ahora si hacemos un “kubectl version –short” nos deberían salir las versiones del cliente, Kustomize y del server (al momento de redactar este proyecto la versión disponible es la v1.26.3).

Una vez hecho esto, vamos a comprobar que nuestro Rancher K3s 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.

Para evitar errores que puedan surgir errores más adelante vamos a hacer dos cambios muy sencillos; primero, 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 al añadir extensiones de Kubernetes, así que, en una terminal de bash, escribiremos “cp /etc/rancher/k3s/k3s.yaml $HOME/.kube/config”. Lo segundo es que en nuestro archivo “$HOME/.bashrc” vamos a escribir dos nuevas líneas de código: “alias k=”kubectl” y, en la siguiente línea, “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, ya que por defecto K3s toma como archivo de configuración el de k3s.yaml en lugar del config que hemos creado hace un momento.

## Instalación de Helm

Más adelante en este proyecto instalaremos un servicio en nuestro clúster mediante Helm. Helm es una herramienta de gestión de paquetes para Kubernetes que simplifica el despliegue de aplicaciones en clústeres. Permite instalar, actualizar y personalizar fácilmente aplicaciones mediante charts predefinidos. Un chart de Helm es un paquete que contiene los recursos y configuraciones por defecto de la aplicación a instalar.

Es ampliamente utilizado en entornos de producción y empresariales ya que te simplifica mucho una instalación que, de forma manual, tardarías mucho tiempo en asegurar de que todo funcionara bien.

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 v3.10.1. Una vez instalado Helm, podemos continuar.

## Instalación de AWX

Vamos a proceder a la primera instalación de un servicio del proyecto. Esta instalación se hará en el clúster de Rancher.

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 un “mensaje del día” (o MOTD) en un router Cisco 7200.

* **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 estar creado de dos maneras distintas: Estático (los hosts se escriben directamente en el archivo) o dinámico (los hosts se reciben a través de una API de otro servicio, como por ejemplo Netbox)

* **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 (por ejemplo, la asignación de variables directamente en la plataforma sin necesidad de utilizar el archivo de inventario, guardar contenido sensible en secrets, tener el proyecto de GitHub almacenado directamente en él…). La diferencia entre AWX y Tower es semejante a la que hay entre Fedora/CentOS 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 set de contenedores encapsulados en un pod que se encargarán de ejecutar la instalación. 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=2.1.0

# - awx.yml

images:

* name: quay.io/ansible/awx-operator

newTag: 2.1.0

namespace: pi

En este archivo le estamos indicando que cree una customización basada en el código de la versión 2.1.0 de AWX, que está 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 momentos (alrededor de dos minutos).

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

En esta línea estamos creando un recurso llamado AWX (la herramienta en sí) en el espacio de nombres “pi” y que se expondrá al público mediante el puerto 30080 de nuestro equipo (en este caso del servidor).

Ahora volvemos al archivo “kustomization.yml” y descomentaremos la línea de awx.yml en metadata, por lo que se quedaría únicamente en “– awx.yml”. Cuando lo hayamos descomentado, 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”. Este comando, por defecto, ejecuta el comando cada 3 segundos, por lo que podemos ir viendo qué cambios se van produciendo en los pods.

Después de aprox. 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-xxxxxx-xxxxxx, 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

Este comando nos decodificará la clave de admin, que se encuentra almacenada en un secret llamado “pi-admin-password”.

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

## Instalación de Grafana y Prometheus

Pasaremos a la instalación del segundo servicio en nuestro clúster de Kubernetes. 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 de Kubernetes 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. Al contrario que el de AWX, este se encarga de levantar Prometheus y Grafana autónomamente, no necesitamos intervención posterior a los comandos que ya hemos ejecutado. 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

Estos archivos se diferencian de los que vienen por defecto en el apartado de especificación: aquí le indicamos que queremos desplegar un servicio del tipo “LoadBalancer” (abre un puerto externo en nuestro equipo además de hacer las funciones de balanceador de carga), a diferencia de desplegar un servicio “ClusterIP” (expone un puerto internamente, por lo que no se puede acceder desde la red).

Para aplicar estas modificaciones utilizaremos el mismo comando que utilizamos antes (k apply -f ./). Como veremos en la salida del comando, esta vez en lugar de aparecer los recursos como creados aparecerán como configurados, ya que mediante los metadatos de los archivos .yaml que hemos creado, Kubernetes ha reconocido que ya existían anteriormente y ha modificado los que ya había creados. Para ver qué puerto se nos ha asignado, pondremos en la terminal “k get service -n monitoring” (en mi caso, por ejemplo, Grafana se ha asignado al puerto del host 31469).

En los comandos que hemos ejecutado anteriormente también hemos eliminado las políticas de seguridad de Kube-prometheus. Esto es porque, si no, no nos deja conectarnos a los servicios desde la red externa al servidor.

Volviendo a Grafana, 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 más adelante.

## Instalación de Netbox

Esta instalación será ejecutada en la máquina virtual Ubuntu SRV. Como mencionamos anteriormente, esta máquina virtual está basada en Ubuntu Server 22.04. Al igual que con el clúster de Rancher K3s, también han sido modificadas las características de CPU y RAM para evitar cuelgues e inestabilidad.

#### DOCUMENTAR PARA EJECUTAR PLAYBOOK DE ANSIBLE!!!!

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 de GNS3

Durante lo que llevamos de documento hemos visto cómo preparar los equipos para crear un 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 la infraestructura de redes que vayamos a montar en nuestro proyecto. Podemos montar desde firewalls hasta routers, switches, VPNs… Incluso máquinas virtuales y contenedores 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 (instalación desde cero, aunque no es complejo ya que GNS3 está basado en Ubuntu Server 20.04) o con una imagen con formato OVA (incluye una máquina virtual con el sistema instalado y preparado para correr). Para montar la máquina virtual con OVA en un entorno local es necesario instalar VMware, el cual tiene ciertas incompatibilidades con WSL y VirtualBox, por lo que lo instalaré en un servidor aparte con Proxmox ya que esta plataforma de virtualización “bare-metal” sí es completamente compatible. El servidor utilizado para este propósito fue donado por el instituto y por el subdirector José Luis Navarro, así que le mando un agradecimiento al departamento entero de informática por la ayuda otorgada en este aspecto.

Tan sólo tendremos que importar la OVA a nuestro virtualizador (ya sea VMware o Proxmox), y asignar los discos a la máquina virtual. GNS3 cuenta con una aplicación local para escritorio pero tiene muchos conflictos con el sistema operativo, hasta el punto de dejar Windows inutilizable.

# Configuración de GNS3

Vamos a empezar el apartado de configuraciones con GNS3 ya que es la más rápida y simple de toda esta sección. Para empezar, entraremos a la interfaz web de GNS3 (por defecto ProxMox ya crea un adaptador puente para todas las máquinas virtuales) y crearemos un nuevo proyecto: en mi caso le llamaré Proyecto Integrado.

Cuando hayamos creado el proyecto, se nos abrirá una interfaz de diagrama muy similar a la que utilizan otros programas como Diagrams.net o Cisco Packet Tracer.

Por defecto GNS3 no trae ningún dispositivo de red virtualizable por tema de licencias, por lo que tendremos que buscarnos una imagen de dispositivo por nuestra cuenta. En mi caso me descargué una imagen de un router Cisco 7200 Series que Cisco otorga gratuitamente para pruebas y estudiantes, aunque podemos utilizar otros tipos de dispositivos, modelos y marcas (no recomiendo utilizar el router Cisco 3600 Series ya que no incluye SSHv2 y no podremos acceder a él mediante AWX, ya que SSHv1 está obsoleto en Linux desde hace años).

Una vez que tengamos nuestra imagen de dispositivo la importamos en GNS3 (Preferencias -> Importar…) y la colocamos en el diagrama. También colocaremos un recurso de GNS3 llamado “Cloud”, que nos hará la función de “pass-through” para que podamos acceder el router desde la red. Conectamos ambos recursos mediante un cable RJ-45 (preferentemente en el puerto FastEthernet0/0 ya que es el más rápido) y, ahora sí, encendemos el router. Sobre él haremos click derecho, le daremos a encender y a la Terminal integrada. Después de unos instantes, tendremos nuestro router operativo y listo para configurar.

## Configurando el router para el primer acceso

Antes de seguir adelante con la configuración de los servicios vamos a configurar el router para que quede completamente operativo y funcional cuando sea necesario.

Como hemos visto hace un momento, entraremos a la terminal integrada del router que nos proporciona GNS3 y configuramos la dirección IP (192.168.1.170), además de añadir ciertos criterios de seguridad que el router requiere para poder habilitar SSH.

*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

Estos criterios de seguridad que hemos comentado hace un momento son añadir el nombre del dominio (en este caso, por ejemplo, se llamará “pi”), añadir un par de claves RSA y establecer SSH como el protocolo de inicio de sesión por defecto (de serie viene con Telnet, el cual además de ser muy inseguro no es compatible con AWX, por lo que no nos serviría).

Además de estos criterios de seguridad hemos añadido un usuario con credenciales ‘admin’@’admin’ y su contraseña para el “modo enable” es ‘cisco’. De esta manera podemos acceder al router desde la red y, por consecuencia, desde AWX.

# 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.

# 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.

# Notificación de cambios del repositorio con GitHub Actions

En este apartado configuraremos una acción de GitHub Actions para que nos mande un correo electrónico cada vez que haya un cambio en el repositorio, aunque primero vamos a ver en qué consiste GitHub Actions.

GitHub Actions es una función de GitHub desarrollada para que funcione como un desencadenador de acciones (trigger). Es decir, cada vez que en nuestro repositorio haya, por ejemplo, un push (subidas nuevas de datos) se ejecutará el trigger que nosotros hayamos configurado en un archivo .yml. Lo más interesante de Actions es que, internamente, se ejecuta en una máquina virtual con Ubuntu Server, entonces podemos mandarle también que ejecute scripts con todas las opciones que se nos ocurran.

En este caso utilizaremos nuestro correo electrónico para que nos mande notificaciones de actualización del repositorio. Para ello empezaremos creando el archivo .yml, que tendrá un contenido parecido al siguiente:

name: Notificaciones Correo

on:

workflow\_dispatch:

push:

branches:

- main

delete:

create:

jobs:

send-push:

name: Mandar un correo con las actualizaciones del repositorio

runs-on: ubuntu-latest

steps:

- name: Checkout código fuente

uses: actions/checkout@v2

- name: Configurar Git

run: |

git config --global user.name "GitHub Actions"

git config --global user.email "actions@github.com"

- name: Obtener detalles de los archivos

id: file-details

run: |

git diff-tree --no-commit-id --name-only -r ${{ github.sha }} > modified.txt

git diff-tree --no-commit-id --name-only --diff-filter=A -r ${{ github.sha }} > added.txt

git diff-tree --no-commit-id --name-only --diff-filter=D -r ${{ github.sha }} > removed.txt

- name: Leer archivos modificados

id: read-modified

run: |

echo "MODIFIED=$(cat modified.txt)" >> $GITHUB\_ENV

- name: Leer archivos añadidos

id: read-added

run: |

echo "ADDED=$(cat added.txt)" >> $GITHUB\_ENV

- name: Leer archivos borrados

id: read-removed

run: |

echo "REMOVED=$(cat removed.txt)" >> $GITHUB\_ENV

- name: Enviando correo electrónico

uses: dawidd6/action-send-mail@v3.6.1

with:

server\_address: smtp.gmail.com

server\_port: 465

username: ${{ secrets.EMAIL\_ADDRESS }}

password: ${{ secrets.EMAIL\_PASSWD }}

subject: "Se ha realizado una acción en el repositorio"

from: github <noreply@github.com>

to: javsanpae@gmail.com

body: |

Acción: ${{ github.event\_name }}

Repo: ${{ github.repository }}

Commit: ${{ github.sha }}

Autor: ${{ github.actor }}

Detalles:

- Archivos modificados: ${{ env.MODIFIED }}

- Archivos borrados: ${{ env.REMOVED }}

- Archivos creados: ${{ env.ADDED }}

Para hacer uso de este disparador tendremos que crear dos secrets en GitHub: el que alojará el correo electrónico y el que contendrá la contraseña de aplicación para que funcione. Para ello, iremos al repositorio de GitHub, entraremos en Settings y en Secrets -> Actions.