

Desafío N°7

Fecha de Entrega: 26/05/2025

Nombre y Apellido: Hernán Andrés Acosta

Objetivos:

Este proyecto implementa un pipeline CI/CD automatizado que despliega infraestructura en entornos de desarrollo (dev), staging y producción (main) utilizando:

- Jenkins en Minikube para orquestación
- Ansible para configuración de infraestructura
- Multipass para máquinas virtuales objetivo
- GitHub como repositorio de código
- Ngrok para exposición segura del Jenkins local

2. Arquitectura del Sistema

2.1 Componentes Principales

![Diagrama de arquitectura]

☀ Flujo de Trabajo del Proyecto ☀

📌 Usuario ↓

🌐 GitHub (Repositorio de Código)

↓ 🔗 Webhook de GitHub

↓ 🚀 Ngrok (Exposición segura a internet)

↓ ⚙ Jenkins (en Minikube)

↓ 📦 Ansible (Playbooks Orquestados)

↓ 💻 Máquinas Virtuales en Multipass (Entornos dev, staging y producción)

📌 Entornos y sus IPs:

- 🟢 **Dev** → 10.170.126.185
- 🟡 **Staging** → 10.170.126.214
- 🔴 **Prod** → 10.170.126.8

Estructura de Directorios

.hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE\$ tree

```
.
├── dev
│   ├── desafio-7
│   │   ├── files
│   │   │   └── index.html
│   │   ├── includes
│   │   │   └── install-apache2.yml
│   │   ├── main.yml
│   │   ├── staging
│   │   │   └── inventory.ini
│   │   ├── templates
│   │   │   └── ansible_site.conf.j2
│   │   └── vars
│   │       └── vars-site.yml
│   └── main
│       ├── desafio-7
│       │   ├── files
│       │   │   └── index.html
│       │   ├── includes
│       │   │   └── install-apache2.yml
│       │   ├── Jenkinsfile
│       │   ├── main.yml
│       │   ├── production
│       │   │   └── inventory.ini
│       │   ├── README.md
│       │   ├── templates
│       │   │   └── ansible_site.conf.j2
│       │   └── vars
│       │       └── vars-site.yml
│       └── staging
│           ├── desafio-7
│           │   ├── files
│           │   │   └── index.html
│           │   ├── includes
│           │   │   └── install-apache2.yml
│           │   ├── Jenkinsfile
│           │   ├── main.yml
│           │   ├── READMEstaging.md
│           │   ├── staging
│           │   │   └── inventory.ini
│           │   ├── templates
│           │   │   └── ansible_site.conf.j2
│           │   └── vars
│           │       └── vars-site.yml
```

Configuración Clave

3.1 Inventarios Ansible

Ejemplo (staging):

```
≡ inventory.ini X
Educacion_It > D7-ANSIBLE > staging > desafio-7 > staging > ≡ inventory.ini
You, 21 hours ago | 1 author (You)
1 [webservers]
2 ansible-staging ansible_host=10.170.126.214
3
4 [webservers:vars]
5 ansible_user=ubuntu
6 ansible_ssh_private_key_file=/home/jenkins/.ssh/id_ed25519
7 ansible_python_interpreter=/usr/bin/python3 You, 21 hours ago • up
```

```
≡ inventory.ini X
Educacion_It > D7-ANSIBLE > dev > desafio-7 > dev > ≡ inventory.ini
You, 22 hours ago | 1 author (You)
1 [webservers]
2 ansible-dev ansible_host=10.170.126.185
3 You, 22 hours ago • Agregar inventory.ini en rama dev
4 [webservers:vars]
5 ansible_user=ubuntu
6 ansible_ssh_private_key_file=/home/jenkins/.ssh/id_ed25519
7 ansible_python_interpreter=/usr/bin/python3
```

```
≡ inventory.ini X
Educacion_It > D7-ANSIBLE > main > desafio-7 > production > ≡ inventory.ini
You, 24 hours ago | 1 author (You)
1
2 [webservers]
3 ansible-prod ansible_host=10.170.126.8
4
5 [webservers:vars]
6 ansible_user=ubuntu
7 ansible_ssh_private_key_file=/home/jenkins/.ssh/id_ed25519
8 ansible_python_interpreter=/usr/bin/python3 You, 24 hours ago • Ver
```

Pipeline Jenkins

```
pipeline {
  agent any
```

```
environment {
  ANSIBLE_FORCE_COLOR = 'true'
}
```

```
stages {
  stage('Checkout') {
    steps {
      checkout scm
    }
  }
}
```

```
stage('Determine Environment') {
  steps {
    script {
      if (env.BRANCH_NAME == 'dev') {
        env.INVENTORY = 'dev/inventory.ini'
        env.TARGET_IP = '10.170.126.185'
      } else if (env.BRANCH_NAME == 'staging') {
        env.INVENTORY = 'staging/inventory.ini'
        env.TARGET_IP = '10.170.126.214'
      } else if (env.BRANCH_NAME == 'main') {
        env.INVENTORY = 'production/inventory.ini'
        env.TARGET_IP = '10.170.126.8'
      } else {
        error "Rama no válida: ${env.BRANCH_NAME}"
      }
    }
  }
}
```

```
stage('Run Ansible Playbook') {
  steps {
    sshagent(credentials: ['web1-key']) {
      sh '''
        echo "📡 Agregando host al known_hosts..."
        mkdir -p ~/.ssh
        ssh-keyscan -H ${TARGET_IP} >> ~/.ssh/known_hosts
      '''
    }
  }
}
```

```
echo "🔍 Validando conexión con Ansible..."
ansible -i ${INVENTORY} all -m ping
```

```
echo "🚀 Ejecutando playbook..."
ansible-playbook -i ${INVENTORY} main.yml
'''
}
```

Flujo de Trabajo

Desarrollo: Push a rama dev → despliega en VM dev

S	W	Name ↓	Último Éxito	Último Fallo
		dev	4 Hor 38 Min #13	17 Hor #9
		main	9 Min 54 Seg #5	4 Hor 58 Min #4
		staging	17 Min #7	4 Hor 58 Min #6

```
● hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE/dev/desafio-7$ git commit -m update-dev
[dev a584ccf] update-dev
 1 file changed, 681 insertions(+)
 create mode 100644 READMEdev.md
● hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE/dev/desafio-7$ git push origin dev
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 7.89 KiB | 7.89 MiB/s, listo.
Total 3 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/hernan130/desafio-7.git
 258b7cd..a584ccf dev -> dev
```

✓ Salida de consola

[Download](#)[Copy](#)[View as plain text](#)

```
Branch indexing
> git rev-parse --resolve-git-dir /var/jenkins_home/caches/git-9674d3b906ead10b28bd00283bc9571e/.git #
timeout=10
Setting origin to https://github.com/hernan130/desafio-7.git
> git config remote.origin.url https://github.com/hernan130/desafio-7.git # timeout=10
Fetching origin...
Fetching upstream changes from origin
> git --version # timeout=10
> git --version # 'git version 2.39.5'
> git config --get remote.origin.url # timeout=10
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- origin +refs/heads/*:refs/remotes/origin/* # timeout=10
Seen branch in repository origin/dev
Seen branch in repository origin/main
```

dev > #14

```
# 10.170.126.185:22 SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.11
# 10.170.126.185:22 SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.11
+ echo 🔍 Validando conexión con Ansible...
🔍 Validando conexión con Ansible...
+ ansible -i dev/inventory.ini all -m ping
[]0;32mansible-dev | SUCCESS => {}[]0m
[]0;32m   "changed": false,[]0m
[]0;32m   "ping": "pong"[]0m
[]0;32m}[]0m
+ echo 🚀 Ejecutando playbook...
🚀 Ejecutando playbook...
+ ansible-playbook -i dev/inventory.ini main.yml

PLAY [Deployment de un sitio estático] *****

TASK [Gathering Facts] *****
[]0;32mok: [ansible-dev][]0m

TASK [Verificar si el OS es Ubuntu] *****
[]0;32mok: [ansible-dev][]0m

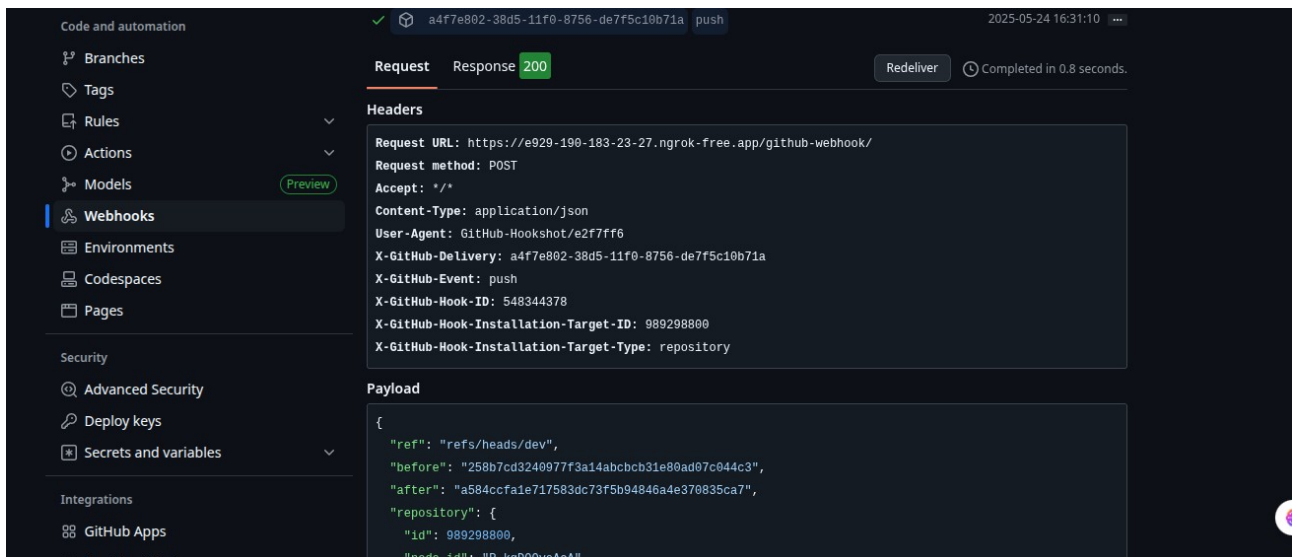
TASK [Instalar servicio Apache] *****
[]0;36mincluded: /home/jenkins/agent/workspace/D7-ansible_dev/includes/install-apache2.yml for ansible-
```

dev > #14

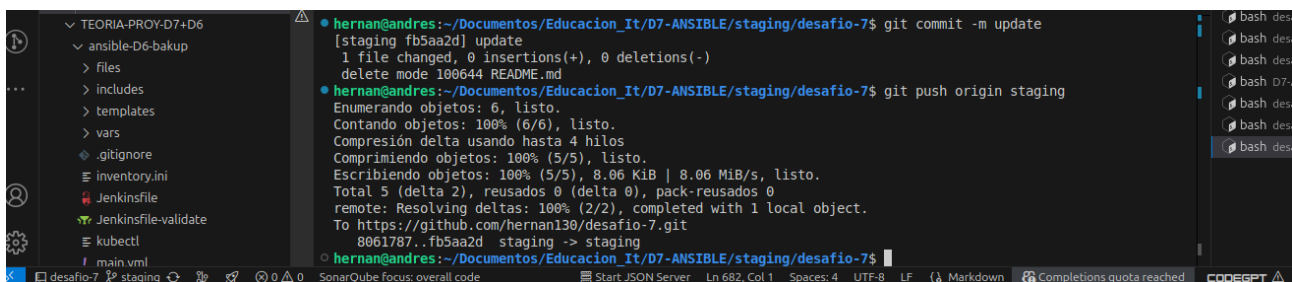
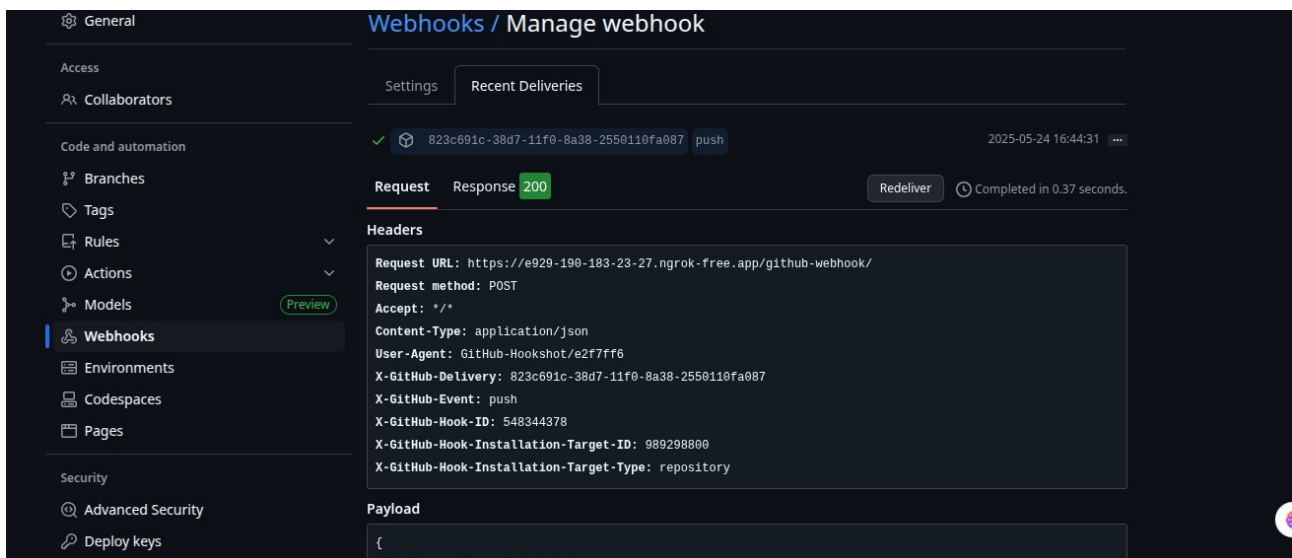
```
[]0;35mchanged: [ansible-dev][]0m

PLAY RECAP *****
[]0;33mansible-dev[]0m      : []0;32mok=13 []0m []0;33mchanged=4 []0m unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

[Pipeline] }
$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 145 killed;
[ssh-agent] Stopped.
[Pipeline] // sshagent
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```



Testing: Merge a staging → despliega en VM staging



Panel de Control > D7-ansible > staging > #8

</> Changes

Console Output

Edit Build Information

Delete build '#8'

Timings

Git Build Data

Restart from Stage

Replay

Pipeline Steps

Workspaces

Previous Build

Summary

1. update readme staging ([details](#))
2. update ([details](#))

Commit 19619a647e4681911aff76493ca4073c62311be5 **by** herman andres acosta
update readme staging

+ READMEstaging.md

Commit fb5aa2d25407034df7684682e1ef40b017eb729a **by** herman andres acosta
update

x README.md

Producción: Merge a main → despliega en VM producción

```
hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE/main/desafio-7$ git add .
hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE/main/desafio-7$ git commit -m "README production"
[main f224996] README production
1 file changed, 668 insertions(+)
hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE/main/desafio-7$ git push origin main
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 7.72 KiB | 7.72 MiB/s, listo.
Total 3 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/herman130/desafio-7.git
f8f69dc..f224996 main -> main
hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE/main/desafio-7$
```

we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

✓ <https://e929-190-183-23-27.ngrok...> (push)

Edit Delete

Last delivery was successful.

✓

1e0ec622-38d9-11f0-8b67-992bc980149b

push

2025-05-24 16:56:02

...

Request

Response

200

Redeliver

Completed in 0.4 seconds.

Headers

Request URL: https://e929-190-183-23-27.ngrok-free.app/github-webhook/

Request method: POST

Accept: */*

Content-Type: application/json

User-Agent: GitHub-Hookshot/e2f7ff6

X-GitHub-Delivery: 1e0ec622-38d9-11f0-8b67-992bc980149b

X-GitHub-Event: push

X-GitHub-Hook-ID: 548344378

X-GitHub-Hook-Installation-Target-ID: 989298800

X-GitHub-Hook-Installation-Target-Type: repository

Payload

<

>

↺

🔖

No es seguro

10.170.126.8

🍻

Automate Terr...

🐦 HCP Terraform...

🐦 hashicorp/aws...

🔄 learn-terrafor...

mi sitio ansible

```
hernan@andres:~$ multipass list
Name                State      IPv4             Image
ansible-dev         Running    10.170.126.185   Ubuntu 24.04 LTS
ansible-prod         Running    10.170.126.8     Ubuntu 24.04 LTS
ansible-staging      Running    10.170.126.214   Ubuntu 24.04 LTS
```

<

>

↺

🔖

No es seguro

10.170.126.185

🍻

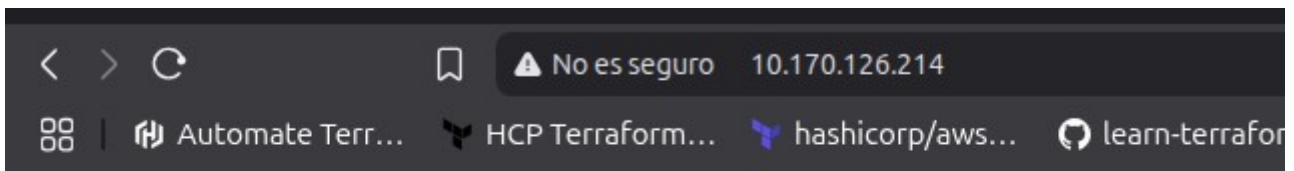
Automate Terr...

🐦 HCP Terraform...

🐦 hashicorp/aws...

🔄 learn-terrafor...

mi sitio ansible



mi sitio ansible

Hernan Andres Acosta trabajoN7 (staging)

Configuración de Seguridad

Credenciales en Jenkins

Nombre Tipo Uso

github-token-hernan Secret Text Acceso a repositorio GitHub

web1-key SSH Private Key Conexión a VMs Multipass

Hardening de SSH

Claves Ed25519 para mejor seguridad

Configuración estricta de permisos en ~/.ssh

Uso de ssh-agent en Jenkins

Infraestructura Virtual

Máquinas Multipass

bash

Copy

Download

Name IP Estado

ansible-dev 10.170.126.185 Ubuntu 24.04

ansible-staging 10.170.126.214 Ubuntu 24.04

ansible-prod 10.170.126.8 Ubuntu 24.04

Integración Continua

Webhooks con Ngrok

kubectl port-forward svc/jenkins -n jenkins 8081:8080

ngrok http 8081

URL Webhook: https://[subdominio].ngrok.io/github-webhook/

Recomendaciones

Monitoreo: Implementar checks de salud post-despliegue

Rollback: Añadir etapa de verificación y rollback automático

Secrets Management: Usar Vault para manejo de credenciales

Infraestructura como Código: Migrar a Terraform para creación de VMs

Conclusión Final del Proyecto de DevOps

Durante el desarrollo de este proyecto, logramos establecer un flujo automatizado y eficiente para la gestión de entornos (dev, staging y producción) mediante el uso de Ansible, Jenkins y Minikube. Se implementaron prácticas clave de automatización como la separación de inventarios para cada entorno, facilitando despliegues consistentes y controlados. Además, se configuró Jenkins Multibranch Pipeline, permitiendo la ejecución automática de los playbooks de Ansible en función de la rama de Git, garantizando así la integración continua (CI) y el despliegue continuo (CD).

El uso de claves SSH seguras y la integración con credenciales específicas fortalecieron la seguridad en las conexiones remotas. La exposición de Jenkins a través de ngrok demostró una solución creativa para acceder a entornos locales desde cualquier parte del mundo. Asimismo, la personalización de la imagen de Jenkins con herramientas esenciales (Ansible, Git, SSH) aseguró la compatibilidad y flexibilidad del pipeline.

En conjunto, este trabajo refleja un dominio sólido de herramientas de automatización, orquestación y buenas prácticas en DevOps. La documentación detallada y la estructura modular brindan una base robusta para futuras expansiones o adaptaciones en entornos productivos.

Anexo como se contruyó el proyecto , el cual va incorporado al README.md

```
""bash
# Estructura de carpetas del proyecto
hernan@andres:~/Documentos/Educacion_It/D7-ANSIBLE$ tree
.
├── dev
│   ├── desafio-7
│   ├── dev
│   ├── inventory.ini
│   ├── files
│   ├── index.html
│   ├── includes
│   ├── install-apache2.yml
│   ├── Jenkinsfile
│   ├── main.yml
│   ├── README.md
│   ├── templates
│   │   ├── ansible_site.conf.j2
│   │   └── vars
│   │       └── vars-site.yml
│   └── main
│       ├── desafio-7
│       ├── files
│       ├── index.html
│       └── includes
```


```
├── install-apache2.yml
├── Jenkinsfile
├── main.yml
├── production
├── inventory.ini
├── README.md
├── templates
├── ansible_site.conf.j2
├── vars
├── vars-site.yml
├── staging
├── desafio-7
├── files
├── index.html
├── includes
├── install-apache2.yml
├── main.yml
├── staging
├── inventory.ini
├── templates
├── ansible_site.conf.j2
├── vars
├── vars-site.yml
```

22 directories, 22 files

Estructura general

Tienes 3 entornos principales:

dev

 staging

main

Cada entorno tiene su propia carpeta desafio-7 que contiene los recursos necesarios para ejecutar Ansible y el pipeline.

Explicación por entorno

dev

desafio-7

dev/

```
└─ inventory.ini
```

dev/inventory.ini

Contiene los hosts/servidores donde Ansible actuará para el entorno de desarrollo.

✓ main

desafio-7

production/

```
inventory.ini
```

```
production/inventory.ini
```

Contiene los hosts/servidores para el entorno de producción.

```
✓ staging
├── desafio-7
│   ├── staging/
│   └── inventory.ini
```

staging/inventory.ini
Contiene los hosts/servidores para el entorno de staging.

📁 Archivos comunes dentro de cada entorno
Cada carpeta desafio-7 tiene una estructura similar:

```
├── files/
│   └── index.html
├── includes/
│   └── install-apache2.yml
├── Jenkinsfile
├── main.yml
├── README.md
├── templates/
│   └── ansible_site.conf.j2
├── vars/
└── vars-site.yml
```

files/index.html
Archivo estático HTML que será desplegado en el servidor.

includes/install-apache2.yml
Un rol o tarea específica de Ansible para instalar Apache2.

Jenkinsfile
El pipeline de Jenkins que viste antes: automatiza el despliegue.

main.yml
Playbook principal de Ansible que orquesta las tareas.

README.md
Documentación de cada entorno o de las instrucciones.

templates/ansible_site.conf.j2
Plantilla Jinja2 para configurar Apache (por ejemplo, un archivo de configuración .conf).

vars/vars-site.yml
Variables usadas en el playbook principal para personalizar configuraciones.

Contenido clave de los inventory.ini

ansible/dev/inventory.ini

[webservers]
ansible-dev ansible_host=10.170.126.185

```
[webserver:vars]
ansible_user=ubuntu
ansible_ssh_private_key_file=/home/jenkins/.ssh/id_ed25519
ansible_python_interpreter=/usr/bin/python3
```

```
# ansible/staging/inventory.ini
```

```
[webserver]
ansible-staging ansible_host=10.170.126.214
```

```
[webserver:vars]
ansible_user=ubuntu
ansible_ssh_private_key_file=/home/jenkins/.ssh/id_ed25519
ansible_python_interpreter=/usr/bin/python3
```

```
# ansible/production/inventory.ini
```

```
[webserver]
ansible-prod ansible_host=10.170.126.8
```

```
[webserver:vars]
ansible_user=ubuntu
ansible_ssh_private_key_file=/home/jenkins/.ssh/id_ed25519
ansible_python_interpreter=/usr/bin/python3
```

```
*****
```

Qué hace Jenkins ahora?

Con el Jenkinsfile, si estás en la rama staging, Jenkins ejecutará:

✓ Jenkins determina la rama

Cuando haces un commit en la rama staging, Jenkins detecta que la rama es staging.

✓ En el stage Determine Environment, Jenkins define dos variables de entorno:

```
if (env.BRANCH_NAME == 'staging') {
  env.INVENTORY = 'staging/inventory.ini'
  env.TARGET_IP = '10.170.126.214'
}
```

Entonces:

👉 INVENTORY=staging/inventory.ini

👉 TARGET_IP=10.170.126.214

✓ Cuando llega al stage Run Ansible Playbook, el comando que ejecuta Jenkins es:

```
ansible-playbook -i ${INVENTORY} main.yml
```

Esto expande a:

```
ansible-playbook -i staging/inventory.ini main.yml
```

```

*****
**
pipeline {
  agent any

  environment {
    ANSIBLE_FORCE_COLOR = 'true'
  }

  stages {
    stage('Checkout') {
      steps {
        checkout scm
      }
    }

    stage('Determine Environment') {
      steps {
        script {
          if (env.BRANCH_NAME == 'dev') {
            env.INVENTORY = 'dev/inventory.ini'
            env.TARGET_IP = '10.170.126.185'
          } else if (env.BRANCH_NAME == 'staging') {
            env.INVENTORY = 'staging/inventory.ini'
            env.TARGET_IP = '10.170.126.214'
          } else if (env.BRANCH_NAME == 'main') {
            env.INVENTORY = 'production/inventory.ini'
            env.TARGET_IP = '10.170.126.8'
          } else {
            error "Rama no válida: ${env.BRANCH_NAME}"
          }
        }
      }
    }

    stage('Run Ansible Playbook') {
      steps {
        sshagent(credentials: ['web1-key']) {
          sh '''
            echo " Agregando host al known_hosts..."
            mkdir -p ~/.ssh
            ssh-keyscan -H ${TARGET_IP} >> ~/.ssh/known_hosts

            echo " Validando conexión con Ansible..."
            ansible -i ${INVENTORY} all -m ping

            echo " Ejecutando playbook..."
            ansible-playbook -i ${INVENTORY} main.yml
          '''
        }
      }
    }
  }
}

```



```
}  
}  
}  
}
```

```
*****
```

Propósito general del pipeline

Este pipeline automatiza el despliegue de una infraestructura (o aplicación) usando Ansible, en función de la rama de Git que se está construyendo. Cambia el inventario y la IP de destino según el entorno (dev, staging o producción).

```
*****
```

Estructura del pipeline

```
pipeline { ... }
```

Todo el pipeline está definido dentro de este bloque.

```
agent any
```

Indica que puede ejecutarse en cualquier nodo disponible en Jenkins.

```
environment { ... }
```

Define una variable de entorno global:

```
ANSIBLE_FORCE_COLOR = 'true'
```

Hace que la salida de Ansible sea más colorida y legible.

```
*****
```

■ Stages

El pipeline tiene 3 etapas principales:

1 Checkout

Propósito: Descargar el código fuente desde el repositorio Git (usando el SCM configurado en el Job).

Comando: checkout scm

2 Determine Environment

Propósito: Decide el inventario de Ansible y la IP del servidor a usar en función de la rama (dev, staging, main).

Lógica:

Si la rama es dev, usa inventario y IP de desarrollo.

Si la rama es staging, usa inventario e IP de staging.

Si es main, usa inventario e IP de producción.

Si es otra rama, falla con un error.

3 Run Ansible Playbook

Propósito: Ejecuta el playbook de Ansible en el servidor correspondiente.

Autenticación:

Usa sshagent con credenciales web1-key (clave SSH para acceso remoto).

Pasos:

Agrega la IP al ~/.ssh/known_hosts para evitar prompts interactivos.

Hace un ping a todos los hosts del inventario para validar la conexión.

Ejecuta el playbook principal (main.yml).



Conclusión / resumen

- ✓ Determina entorno según la rama de Git.
- ✓ Usa Ansible para automatizar la configuración/instalación.
- ✓ Seguridad y automatización con sshagent.
- ✓ Color en la salida para legibilidad.

```
*****
se crearon las siguientes maquinas virtuales en multipass
hernan@andres:~$ multipass list
```

Pasos para Ejecutar el Proyecto.

1. Instalar Multipass: sudo snap install multipass

```
"" bash
```

```
multipass launch --name
```

Name	State	IPv4	Image
------	-------	------	-------

ansible-dev	Running	10.170.126.185	Ubuntu 24.04 LTS
-------------	---------	----------------	------------------

ansible-prod	Running	10.170.126.8	Ubuntu 24.04 LTS
--------------	---------	--------------	------------------

ansible-staging	Running	10.170.126.214	Ubuntu 24.04 LTS
-----------------	---------	----------------	------------------

3. Generar par de clave SSH:

```
"" bash
```

```
# Paso 1 — Creación de claves SSH
```

El primer paso para configurar la autenticación de clave SSH en su servidor es generar un par de claves SSH en su computadora local.

Para hacer esto, podemos usar una utilidad especial llamada

```
""bash
```

```
bashssh-keygen
```

En su computadora local, genere un par de claves SSH escribiendo:

```
""ssh-keygen
```

```
o
```

```
ssh-keygen -t ed25519 -C "nombre_de_la_clave"
```

```
'''
```

Output

Generating public/private rsa key pair.

Enter file in which to save the key (/home/username/.ssh/id_rsa):

La utilidad le pedirá que seleccione una ubicación para las claves que se generarán. De forma predeterminada, las claves se almacenarán en el ~/.ssh directorio dentro de su directorio de inicio de usuarios. Se llamará la clave privada id_rsa y se llamará la clave pública asociada id_rsa.pub.

Por lo general, es mejor seguir con la ubicación predeterminada en esta etapa. Hacerlo permitirá que su cliente SSH encuentre automáticamente sus claves SSH cuando intente autenticarse. Si desea elegir una ruta no estándar, escriba ahora, de lo contrario, presione ENTER para aceptar el valor predeterminado.

Si previamente había generado un par de claves SSH, puede ver un mensaje que se ve así:

Output/home/username/.ssh/id_rsa already exists.

Overwrite (y/n)?

y= para reescribirlo, lo cual borra el contenido anterior.

n= para no sobreescribirlo.

Ahora tiene una clave pública y privada que puede usar para autenticarse. El siguiente paso es colocar la clave pública en su servidor para que pueda usar la autenticación de clave SSH para iniciar sesión.

Copiar Su Clave Pública Manualmente

Si no tiene acceso SSH basado en contraseña a su servidor disponible, tendrá que hacer el proceso anterior manualmente.

El contenido de tu id_rsa.pub el archivo deberá agregarse a un archivo en ~/.ssh/authorized_keys en su máquina remota de alguna manera.

Para mostrar el contenido de su id_rsa.pub clave, escriba esto en su computadora local:

```
# cat ~/.ssh/id_rsa.pub
```

```
# cat ~/.ssh/id_ed25519.pub
```

Acceda a su host remoto utilizando cualquier método que tenga disponible. Esta puede ser una consola basada en la web proporcionada por su proveedor de infraestructura.

Una vez que tenga acceso a su cuenta en el servidor remoto, debe asegurarse de que ~/.ssh se crea el directorio. Este comando creará el directorio si es necesario, o no hará nada si ya existe:

```
mkdir -p ~/.ssh
```

```
mkdir -p ~/.ssh # Crea el directorio si no existe
```

```
echo "TU_CLAVE_PÚBLICA_AQUÍ" >> ~/.ssh/authorized_keys
```

```
chmod 700 ~/.ssh
```

```
chmod 600 ~/.ssh/authorized_keys
```

Ahora, puede crear o modificar el `authorized_keys` archivo dentro de este directorio. Puede agregar el contenido de su `id_rsa.pub` archivo al final del `authorized_keys` archivo, creándolo si es necesario, usando esto:

```
# echo public_key_string >> ~/.ssh/authorized_keys
```

En el comando anterior, sustituya el `public_key_string` con la salida de la

```
cat ~/.ssh/id_rsa.pub
cat ~/.ssh/id_ed25519.pub
```

comando que ejecutó en su sistema local. Debería empezar con `ssh-rsa AAAA...` o similar.

Si esto funciona, puede pasar a probar su nueva autenticación SSH basada en claves.

Paso 3 — Autenticación a su Servidor Usando Claves SSH

Si ha completado con éxito uno de los procedimientos anteriores, debería poder iniciar sesión en el host remoto sin la contraseña de las cuentas remotas.

```
# ssh username@remote_host
```

en mi caso

```
ssh -v ubuntu@10.170.126.185 (para dev)
ubuntu@ansible-dev:~$ ls- la
Command 'ls-' not found, did you mean:
command 'lsd' from snap lsd (0.16.0)
command 'lsw' from deb suckless-tools (47-1)
command 'lsm' from deb lsm (1.0.4-2)
command 'lsc' from deb livescript (1.6.1+dfsg-3)
command 'lsh' from deb lsh-client (2.1-14)
command 'lsd' from deb lsd (0.23.1-8)
command 'ls' from deb coreutils (9.4-2ubuntu2)
See 'snap info <snapname>' for additional versions.
```

```
''' bash
```

Si esta es la primera vez que se conecta a este host (si utilizó el último método anterior), puede ver algo como esto:

```
OutputThe authenticity of host '203.0.113.1 (203.0.113.1)' can't be established.
ECDSA key fingerprint is fd:fd:d4:f9:77:fe:73:84:e1:55:00:ad:d6:6d:22:fe.
Are you sure you want to continue connecting (yes/no)? yes
```

Esto significa que su computadora local no reconoce el host remoto. typee yes y luego presione ENTER para continuar.

Post de ayuda de como crear un clave privada y publica:

<https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server>

posteriormente se subio el repositorio a git hub con el nombre herman130/desafio-7 en caga
bransh
``bash

✓ FLUJO RECOMENDADO

1. Subí todo a main

```
git checkout main
```

```
git add .
```

```
git commit -m "Subo estructura base con todos los entornos"
```

```
git push origin main
```

Esto sube todo: ansible/dev, ansible/staging, ansible/production, etc.

2. Desde main, creás las otras ramas

```
git checkout -b dev
```

```
git push origin dev
```

```
git checkout -b staging
```

```
git push origin staging
```

3. En cada rama, borrás lo que no corresponde

En dev:

```
git checkout dev
```

```
rm -rf ansible/staging ansible/production
```

```
git commit -am "Elimino carpetas que no corresponden al entorno dev"
```

```
git push origin dev
```

En staging:

```
git checkout staging
```

```
rm -rf ansible/dev ansible/production
```

```
git commit -am "Elimino carpetas que no corresponden al entorno staging"
```

```
git push origin staging
```

En main:

```
git checkout main
```

```
rm -rf ansible/dev ansible/staging
```

```
git commit -am "Dejo solo entorno de producción"
```

```
git push origin main
```

```
*****  
*****
```

```
*****
```



¿Por qué usar main como rama de producción?

Porque es la convención estándar.

En la mayoría de los flujos Git, la rama main (antes llamada master) representa la versión estable y lista para producción.

Jenkins, por convención, espera que main sea producción, salvo que vos explícitamente configures otra rama (como production).

```
*****
```

'''
instalacion de jenkins en minikube

Instalación de Jenkins en Minikube
Crear namespace:

```
kubectl create namespace jenkins
```

```
helm repo add jenkins https://charts.jenkins.io  
helm repo update
```

```
helm install jenkins -n jenkins jenkins/jenkins
```

Exponer el servicio de Jenkins:

```
kubectl port-forward svc/jenkins -n jenkins 8081:8080
```

Obtener el password del admin:

```
kubectl exec -it -n jenkins jenkins-0 -- cat /var/jenkins_home/secrets/initialAdminPassword
```

Imagen personalizada de Jenkins
Se creó una imagen personalizada basada en jenkins/jenkins:lts con los siguientes requisitos:

Git

Ansible

SSH client

Dockerfile:
Dockerfile para Jenkins agent con Ansible, Git y SSH

```
FROM jenkins/inbound-agent:latest
```

```
USER root
```

```
# Instalamos Ansible, Git y OpenSSH, y creamos la carpeta .ssh con permisos correctos
```

```
RUN apt-get update && apt-get install -y \  
ansible \  
git \  
openssh-client \  
&& mkdir -p /home/jenkins/.ssh \  
&& chown -R jenkins:jenkins /home/jenkins/.ssh \  
&& apt-get clean \  
&& rm -rf /var/lib/apt/lists/*
```

```
USER jenkins
```

Esta imagen fue publicada en Docker Hub bajo:
hernan1305/jenkins-ansible:latest

Luego, fue configurada en el Pod Template de Jenkins UI para reemplazar la imagen por defecto.

se configuro las credenciales de github con token clasico para para que jenkins pueda acceder al repositorio de ansible y las credenciales ssh username with private key para que jenkins pueda acceder a la maquinas virtuales de multipass.

```
hernan@andres:~$ multipass list
Name State IPv4 Image
ansible-dev Running 10.170.126.185 Ubuntu 24.04 LTS
ansible-prod Running 10.170.126.8 Ubuntu 24.04 LTS
ansible-staging Running 10.170.126.214 Ubuntu 24.04 LTS
```

una vez configuradas las credenciales

```
```bash
```

✓ **Dos credenciales:**

**Nombre Tipo Uso**

github-token-hernan Secret Text Para clonar desde GitHub privado o activar Webhooks

web1-key SSH Private Key Para conectarse vía SSH a las VMs de destino (ansible-dev, ansible-staging, ansible-prod)

```
````
```

Estas credenciales se comparten entre entornos, ya que:

El usuario SSH es el mismo: ubuntu

La clave SSH privada es la misma

Solo cambia la IP según el entorno (ya lo configuraste en cada inventory.ini)

```
```
```

foto de la credenciales creadas

```
```bash
```

se configuro el job con el nombre de desafio-7 pipeline multibranch

```
```
```

## # Pasos para crear el Job Multibranch

1. Crear el nuevo Job

Desde el panel principal de Jenkins, haz clic en "Nuevo Item".

Asigna un nombre, por ejemplo: desafio-7.

Selecciona "Multibranch Pipeline" y haz clic en "OK".

2. Configurar el repositorio Git

Dentro del Job:

En la sección "Branch Sources", haz clic en "Add Source" → Git.

En "Project Repository", ingresa la URL del repositorio, por ejemplo:  
<https://github.com/hernan130/desafio-7.git>



Si el repo es privado, agrega tus credenciales en el campo "Credentials".

### 3. Configurar detección de ramas

Jenkins buscará automáticamente todas las ramas que contengan un archivo llamado Jenkinsfile.

Puedes agregar "Filter by name" si deseas construir solo ciertas ramas (dev, staging, main, etc.).

### 4. Configurar escaneo periódico (opcional)

En "Scan Multibranch Pipeline Triggers", puedes activar:

☒ Periodically if not otherwise run

Y definir un cron, por ejemplo:

H/5 \* \* \* \*

Esto escaneará el repositorio cada 5 minutos para detectar nuevas ramas o cambios en Jenkinsfile.

### 5. Guardar y ejecutar

Haz clic en "Guardar".

Jenkins empezará a escanear el repositorio y creará Jobs por cada rama con Jenkinsfile.

```
```bash
```

```
*****  
se instalo Ngrok
```

```
```bash
```

Conexión de Jenkins en Minikube a través de ngrok

#### 1. Introducción

En entornos locales como Minikube, los servicios desplegados dentro del clúster Kubernetes suelen ser accesibles solo desde la máquina local. Para exponer el servicio de Jenkins que corre en Minikube y poder acceder a él desde internet (por ejemplo, desde una computadora externa o para integraciones CI/CD externas), podemos usar ngrok. Ngrok es una herramienta que permite crear túneles seguros hacia servicios locales a través de una URL pública.

#### 2. Objetivo

Exponer el servicio Jenkins que corre dentro de Minikube a través de una URL pública usando ngrok, facilitando así el acceso remoto para administración, integración o pruebas.

#### 3. Descripción del escenario

Jenkins está desplegado en Minikube como un Pod dentro del namespace jenkins.

Jenkins está expuesto internamente como un servicio Kubernetes (ClusterIP por defecto).

Minikube corre en una máquina local (Linux).

Se necesita acceso externo a Jenkins (p.ej., desde navegador en otra máquina).

Se usará ngrok para tunelizar el puerto local expuesto de Jenkins a internet.

#### 4. Pasos para la conexión

#### 4.1. Exponer Jenkins localmente

Para que ngrok pueda redirigir tráfico a Jenkins, primero debemos hacer accesible Jenkins localmente.

Hacer port-forward desde Minikube al host local:

```
kubectl -n jenkins port-forward svc/jenkins 8081:8080
```

Esto expone Jenkins en localhost:8081.

#### 4.2. Instalar y configurar ngrok

Descargar ngrok desde <https://ngrok.com/download> y descomprimirlo.

Autenticar ngrok con tu token (opcional, pero recomendado):

```
""bash
ngrok authtoken <tu-token-de-autenticación>
```

#### 4.3. Crear túnel ngrok hacia Jenkins

Ejecutar ngrok para tunelizar el puerto local donde está expuesto Jenkins (por ejemplo, 8081):

```
bash
ngrok http 8081
```

Ngrok mostrará una URL pública, como <https://xxxxxx.ngrok.io>, que redirige al Jenkins local.

#### 4.4. Acceder a Jenkins

Desde cualquier navegador o equipo externo, abrir la URL pública que proporciona ngrok.

Se podrá usar Jenkins normalmente como si estuviera alojado en la nube.

#### 5. Consideraciones de seguridad

Ngrok proporciona una URL pública accesible desde internet, por lo que es importante proteger Jenkins con autenticación.

Se recomienda usar HTTPS para proteger las credenciales.

No exponer puertos sensibles sin autenticación.

Ngrok puede configurar reglas de acceso IP o usar autenticación básica para mejorar seguridad.

#### 6. Ventajas de usar ngrok con Jenkins en Minikube

Fácil de configurar y no requiere modificar la configuración del clúster.

No requiere IP pública ni configuración DNS.

Útil para demostraciones, desarrollo, pruebas y acceso remoto rápido.

#### 7. Resumen

Paso	Descripción
1	Exponer Jenkins Port-forward o NodePort para acceder localmente
2	Configurar ngrok Instalar y autenticar ngrok
3	Crear túnel ngrok Ejecutar ngrok http <puerto_local>
4	Acceder Jenkins Usar URL pública ngrok para acceso remoto

```
```  
bash
```

```
*****
```

siguiente paso configuracion del webhook de jenkins

Pasos para usar Ngrok con Jenkins en Minikube + port-forward

Abre tu terminal y ejecuta el port-forward:

```
```bash
```

```
kubectrl port-forward svc/jenkins -n jenkins 8081:8080
```

En otra terminal, inicia Ngrok para exponer ese puerto local:

```
ngrok http 8081
```

Ngrok te mostrará una URL pública tipo:

```
https://e929-190-183-23-27.ngrok-free.app
```

En GitHub, configuras el webhook con esta URL + /github-webhook/:

```
https://e929-190-183-23-27.ngrok-free.app/github-webhook/
```

En Jenkins, asegúrate que el plugin de GitHub webhook esté activo y que la URL /github-webhook/ esté configurada para recibir eventos.

Así, cuando hagas un push en cualquier rama, GitHub enviará el webhook a esa URL pública que redirige a tu Jenkins local corriendo en Minikube, y Jenkins disparará tu pipeline.

```

```