

Unir CP2

Despliegue y configuración de infraestructura en Azure

Carlos Grande

Copyright © Unir CP2 - 2025

Table of contents

1. Inicio	3
1.1 Introducción	3
1.2 Objetivos	3
2. Informe	4
2.1 Informe	4
2.2 Arquitectura	5
2.3 Despliegue	23
2.4 Evidencias	26
2.5 Licencia	41
2.6 Referencias	42

1. Inicio

Asignatura: **Devops & Cloud**

Alumno: **Carlos Grande Núñez**

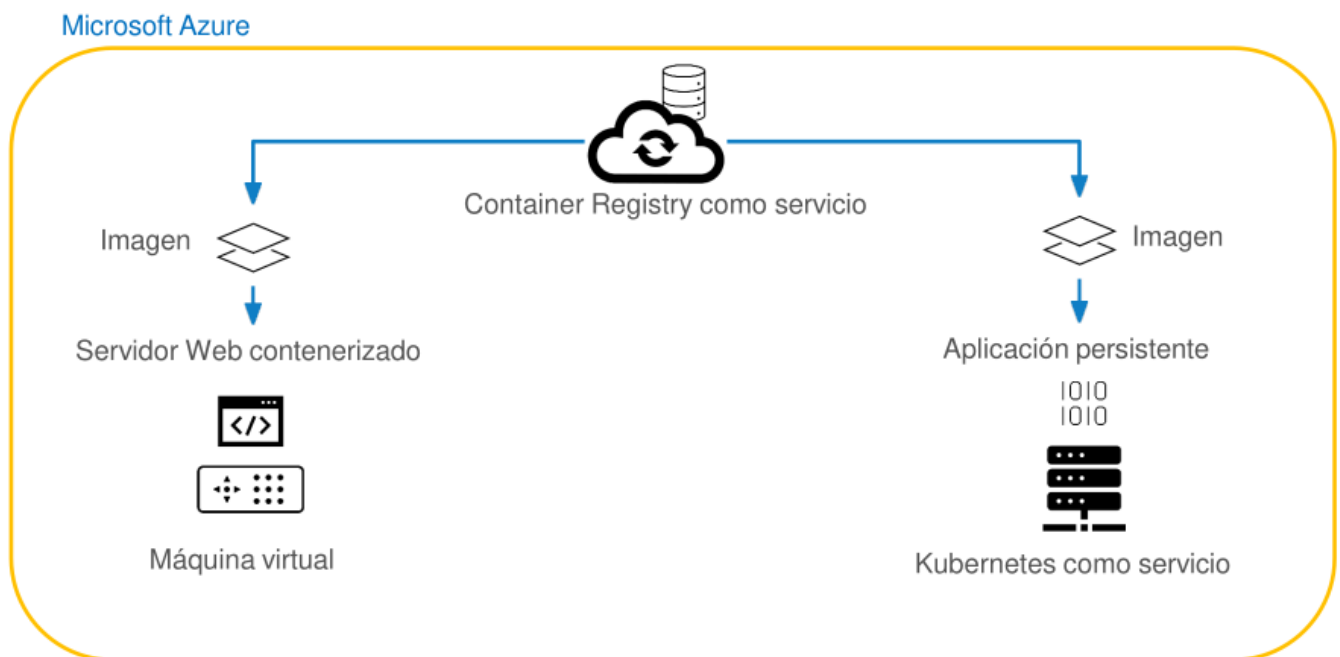
Fecha: **23/03/25**

El informe en PDF ha sido generado a partir de la documentación escrita en Markdown utilizando MkDocs, una librería escrita en Python que me parece muy interesante y que uso habitualmente en mi día a día. Además, el informe completo se encuentra disponible online a través de GitHub Pages, recomiendo acceder a la versión online ya que considero que es mucho más ágil su lectura.

 [Ir a la versión online](#)

1.1 Introducción

Este repositorio contiene la solución del **Caso Práctico 2**, en el cual se ha desplegado una infraestructura en **Microsoft Azure** de forma automatizada utilizando **Terraform** y **Ansible**. Se incluyen configuraciones para la creación de recursos en la nube, instalación de servicios y despliegue de aplicaciones en contenedores con almacenamiento persistente.



1.2 Objetivos

- Crear infraestructura de forma automatizada en un proveedor de Cloud pública.
- Utilizar herramientas de gestión de la configuración para automatizar la instalación y configuración de servicios.
- Desplegar mediante un enfoque totalmente automatizado aplicaciones en forma de contenedor sobre el sistema operativo.
- Desplegar mediante un enfoque totalmente automatizado aplicaciones que hagan uso de almacenamiento persistente sobre una plataforma de orquestación de contenedores.

2. Informe

2.1 Informe

Este informe documenta la entrega del Caso Práctico 2 de la asignatura **DevOps & Cloud** del **programa avanzado DevOps** de la UNIR. El contenido del informe se estructura en las siguientes secciones:

- **Arquitectura:** Descripción de los componentes desplegados y su configuración.
- **Despliegue:** Ejecución práctica de la infraestructura y su configuración.
- **Evidencias:** Recopilación de pruebas de funcionamiento y validaciones.
- **Licencia:** Definición del marco legal de uso.
- **Referencias:** Fuentes utilizadas en el desarrollo del ejercicio.

Para la generación del informe, se ha utilizado MkDocs, una librería de Python para la creación de documentación técnica ([MkDocs, s.f.](#)), junto con el plugin WithPDF, que permite la exportación a formato PDF ([WithPDF, s.f.](#)). Esta elección responde a la naturaleza del caso práctico, en el que una de las tareas consiste en desplegar una imagen estática de una web en Nginx sin persistencia. Dado que MkDocs genera HTML estático, se ha integrado su uso dentro del ejercicio para la documentación y su despliegue.

2.1.1 Código fuente

 [Acceso al repositorio](#)

Estructura del repositorio

El proyecto se organiza en tres grandes bloques: infraestructura, despliegue y documentación. A continuación se resume su estructura principal:

```
repo-root
├── terraform/      # Código para el despliegue de la infraestructura (ACR, VM, AKS)
├── ansible/        # Playbooks y roles para configurar la VM y desplegar en AKS
├── docs/           # Documentación del proyecto (MkDocs)
├── site/           # Sitio estático generado de la documentación
├── setup.sh        # Script para exportar variables tras despliegue
├── mkdocs.yml      # Configuración de MkDocs
├── Dockerfile.docs # Dockerfile para generar la imagen de documentación
├── requirements.txt # Dependencias de Python
├── README.md       # Descripción general del proyecto
└── LICENSE         # Licencia del repositorio
```

2.2 Arquitectura

2.2.1 Arquitectura

La Arquitectura organiza en dos secciones principales:

- **Infraestructura:** contiene la descripción de los componentes desplegados con terraform y la justificación de sus parámetros.
- **Configuración de la infraestructura:** las configuraciones aplicadas a la infraestructura desplegada, automatizadas con Ansible, y la justificación de cada una de ellas.

Visión general

El siguiente diagrama representa la infraestructura desplegada con Terraform y configurada con Ansible, incluyendo una máquina virtual con un contenedor Podman y un clúster AKS, ambos obteniendo imágenes desde un Azure Container Registry (ACR).

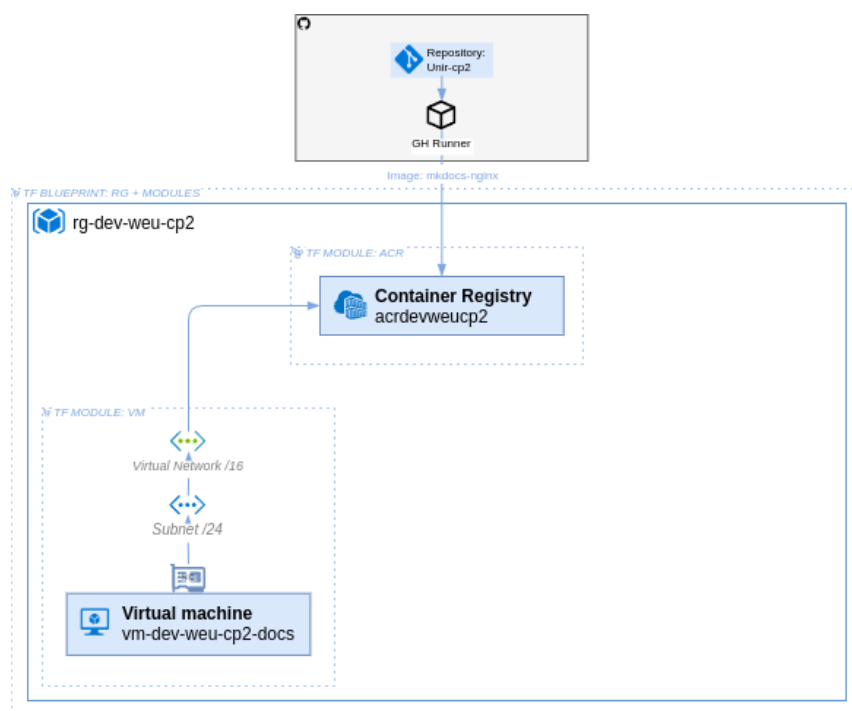


Figura 1: Diagrama de la arquitectura desplegada en Azure (Elaboración propia con draw.io).

2.2.2 Infraestructura

A continuación se describen los diferentes componentes de la infraestructura desplegados con terraform y la justificación de sus configuraciones.

Estructura de ficheros Terraform

Para la implementación de la infraestructura, se ha seguido una organización modular en Terraform, siguiendo las mejores prácticas recomendadas en la comunidad (Stivenson, 2023). A continuación, se describe la estructura de los archivos y su propósito dentro del proyecto.

```

terraform/
├── main.tf           # Archivo principal que llama a los módulos y recursos
├── modules/         # Carpeta que contiene los módulos de infraestructura
│   ├── acr/         # Módulo para desplegar Azure Container Registry (ACR)
│   ├── aks/         # Módulo para desplegar Azure Kubernetes Service (AKS)
│   └── vm/           # Módulo para desplegar la máquina virtual en Azure
├── outputs.tf        # Define las salidas (outputs) de Terraform
├── terraform.tfvars  # Define valores específicos para esta implementación
└── vars.tf           # Define las variables requeridas para el despliegue

```

FICHERO PRINCIPAL `main.tf`

El archivo `main.tf` define la infraestructura base del proyecto y se estructura en tres secciones principales:

- **Configuración base:** define el proveedor `azurerm`, el grupo de recursos y variables locales para el entorno.
- **Llamada a los módulos:** incluyendo la máquina virtual (VM), el registro de contenedores (ACR) y el clúster de Kubernetes (AKS).
- **Generación dinámica del inventario de Ansible:** crea automáticamente un archivo `hosts.yml` con la información de conexión necesaria para la configuración mediante Ansible.

Configuración base

```

main.tf

terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>3.0"
    }
  }
}

# Configuración del proveedor de Azure
provider "azurerm" {
  subscription_id = "fb24fc1f-67e2-4871-8be2-c10a36e74c93"
  features {}
}

# Define la variable de entorno elegida para el despliegue
locals {
  env_suffix = "-${var.environment}"
}

# Crear un grupo de recursos en West Europe
resource "azurerm_resource_group" "rg" {
  name     = "${var.resource_group_name}-${var.environment}"
  location = var.location
  tags     = var.tags
}

```

Llamada a módulos

Contiene la llamada a los diferentes módulos de infraestructura, incluyendo la máquina virtual (VM), el registro de contenedores (ACR) y el clúster de Kubernetes (AKS).

A continuación, se muestra un ejemplo de la estructura de un módulo en `main.tf`, en este caso, la definición del AKS:

```

main.tf

# Llamar al módulo del AKS
module "aks" {

```

```

source      = "/modules/aks"
aks_name    = "${var.aks_name}-${var.environment}"
resource_group = azurerm_resource_group.rg.name
location    = azurerm_resource_group.rg.location
dns_prefix  = var.dns_prefix
node_count  = var.node_count
vm_size     = var.aks_vm_size
acr_id      = module.container_registry.acr_id
tags       = var.tags
}

```

Generación del inventario

La generación del inventario de Ansible se realiza dinámicamente con Terraform para automatizar la configuración de la infraestructura. Se emplea el recurso `local_file` para crear el archivo `hosts.yml` basado en una plantilla que incluye información clave de los recursos desplegados:

- **Máquina virtual:** Dirección IP pública, usuario y clave SSH.
- **Azure Container Registry (ACR):** Nombre, servidor de login, credenciales de acceso.
- **Clúster AKS:** Nombre y grupo de recursos asociado.

main.tf

```

# Generar el archivo hosts.yml
resource "local_file" "ansible_inventory" {
  filename = "../ansible/hosts.yml"
  content  = templatefile("../ansible/hosts.tmpl", {
    vm_name      = var.vm_name
    vm_public_ip = module.virtual_machine.vm_public_ip
    vm_username  = var.vm_username
    ssh_private_key = "~/ssh/az_unir_rsa"
    python_interpreter = var.python_interpreter
    acr_name     = "${var.acr_name}${var.environment}"
    acr_login_server = "${var.acr_name}${var.environment}.azurecr.io"
    acr_username  = module.container_registry.acr_username
    acr_password  = module.container_registry.acr_password
    aks_name     = var.aks_name
    aks_resource_group = var.resource_group_name
  })
}

```

Esta instrucción de terraform apunta al fichero `hosts.tmpl` de la carpeta de ansible y que usa como plantilla para generar el fichero de inventario `hosts.yml`.

hosts.tmpl

```

all:
  children:
    azure_vm:
      hosts:
        ${vm_name}:
          ansible_host: ${vm_public_ip}
          ansible_user: ${vm_username}
          ansible_ssh_private_key_file: ${ssh_private_key}
          ansible_python_interpreter: ${python_interpreter}

    azure_acr:
      hosts:
        ${acr_name}:
          acr_login_server: ${acr_login_server}

    azure_aks:
      hosts:
        ${aks_name}:
          aks_resource_group: ${aks_resource_group}
          ansible_connection: local
          ansible_python_interpreter: /usr/bin/python3

```

Esto permite que Ansible trabaje con información actualizada sin intervención manual, garantizando coherencia y simplificando la configuración.

FICHERO `terraform.tfvars`

El fichero define las variables utilizadas en el despliegue de la infraestructura, priorizando configuraciones de bajo coste para optimizar el uso de recursos en el ejercicio.

Se establece un entorno de desarrollo (`dev`), una máquina virtual con especificaciones mínimas y un clúster AKS con un solo nodo. Además, se configura un ACR y una red con una subred pequeña para evitar sobreasignación de recursos innecesaria.

terraform.tfvars

```
# Generic
resource_group_name = "rg-weu-cp2"
location            = "West_Europe"
environment         = "dev"

# ACR
acr_name            = "acrweucp2"

# virtual machine
vm_name             = "vm-weu-cp2-docs"
vm_username         = "charlstown"
vm_size             = "Standard_B1ms" # "Standard_B1ls"
ssh_public_key      = "~/.ssh/az_unir_rsa.pub"
python_interpreter  = "/usr/bin/python3"

# Networking
vnet_name           = "vnet-weu-cp2"
subnet_name         = "subnet-weu-cp2"
subnet_cidr         = "10.0.1.0/28"

# Image
image_os            = "22_04-lts-gen2"
image_offer         = "0001-com-ubuntu-server-jammy"
# check offers here: https://documentation.ubuntu.com/azure/en/latest/azure-how-to/instances/find-ubuntu-images/

# AKS
aks_name            = "aks-weu-cp2"
dns_prefix          = "aksweucp2"
node_count          = 1
aks_vm_size         = "Standard_B2s"

# Tags
tags = {
  environment = "casopractico2"
}
```

Módulos

CONTAINER REGISTRY

La infraestructura de **Azure Container Service(ACR)** se ha definido utilizando **Terraform**, organizando los recursos en módulos separados para mejorar la modularidad y reutilización del código. A continuación, se presentan los archivos principales que definen el despliegue:

```
terraform/
├─ terraform.tfvars      # Variables globales del despliegue
├─ main.tf               # Llamada a módulos y recursos principales
├─ modules/
│   └─ acr/              # Módulo del ACR
│       ├── main.tf      # Definición del ACR
│       ├── outputs.tf   # Variables de salida
│       └─ variables.tf  # Definición de variables del módulo
```

Puedes ver las evidencias de este despliegue en el  [siguiente enlace](#).

Fichero main.tf

El fichero `main.tf` del módulo del ACR recoge únicamente el recurso `azurerm_container_registry`.

main.tf

```
# Crear Azure Container Registry (ACR)
resource "azurerm_container_registry" "acr" {
  name                = var.acr_name
  resource_group_name = var.resource_group
  location            = var.location
  sku                 = "Basic" # Opción más barata
  admin_enabled       = true
}
```



```
tags = var.tags
}
```

Parámetro	Descripción
<code>var.acr_name</code>	Define el nombre del registro de contenedores. Se usa una variable para permitir reutilización y facilitar la personalización sin modificar el código.
<code>var.resource_group</code>	Especifica el grupo de recursos donde se desplegará el ACR.
<code>var.location</code>	Indica la región de Azure en la que se despliega el registro.
<code>sku = "Basic"</code>	Se elige el nivel Basic , ya que es la opción más económica y suficiente para los requisitos del ejercicio. Alternativamente, se podría usar <code>Standard</code> o <code>Premium</code> si se requiriera mayor escalabilidad o funcionalidades adicionales.
<code>admin_enabled = true</code>	Habilita el acceso mediante credenciales de administrador. Se activa para simplificar la autenticación en el entorno de pruebas, aunque en entornos de producción sería recomendable deshabilitarlo y usar autenticación con identidades de Azure AD.
<code>tags = var.tags</code>	Permite agregar metadatos al recurso para organización y clasificación dentro de Azure.

MÁQUINA VIRTUAL

La infraestructura de la máquina virtual se ha definido utilizando **Terraform**, organizando los recursos en módulos separados para mejorar la modularidad y reutilización del código. A continuación, se presentan los archivos principales que definen el despliegue:

```
terraform/
├── terraform.tfvars      # Variables globales del despliegue
├── main.tf               # Llamada a módulos y recursos principales
├── modules/
│   ├── vm/              # Módulo de la máquina virtual
│   │   ├── main.tf      # Definición de la VM
│   │   ├── network.tf   # Configuración de la red
│   │   ├── security.tf  # Reglas de seguridad (NSG)
│   │   ├── outputs.tf   # Variables de salida (IPs, VM ID)
│   │   └── variables.tf  # Definición de variables del módulo
```

Puedes ver las evidencias de este despliegue en el  [siguiente enlace](#).

Fichero `main.tf`

El fichero `main.tf` del módulo de la máquina virtual recoge los siguientes recursos:

- *IP Pública* → Asigna una dirección IP fija a la VM para acceso remoto.
- *Interfaz de Red (NIC)* → Proporciona conectividad a la máquina virtual en la red definida.
- *Máquina Virtual (VM)* → Instancia de un sistema operativo en Azure con configuración personalizada.

IP Pública

```
main.tf

resource "azurerm_public_ip" "vm_public_ip" {
  name                = "${var.vm_name}-public-ip"
  resource_group_name = var.resource_group
  location            = var.location
  allocation_method   = "Static"
```

```
tags          = var.tags
}
```

Parámetro	Descripción
<code>var.vm_name</code>	Se usa para nombrar la IP pública de la VM de manera única dentro del recurso.
<code>var.resource_group</code>	Grupo de recursos en el que se despliega la IP pública.
<code>var.location</code>	Región de Azure donde se asignará la IP.
<code>allocation_method = "Static"</code>	Se usa IP estática para mantener una dirección fija y evitar cambios en reinicios.
<code>var.tags</code>	Se añaden etiquetas para organización y clasificación dentro de Azure.

Interfaz de Red (NIC)

main.tf

```
resource "azurerm_network_interface" "nic" {
  name                = "${var.vm_name}-nic"
  resource_group_name = var.resource_group
  location            = var.location

  ip_configuration {
    name                       = "internal"
    subnet_id                 = azurerm_subnet.subnet.id
    public_ip_address_id      = azurerm_public_ip.vm_public_ip.id
    private_ip_address_allocation = "Dynamic"
  }
  tags = var.tags
}
```

Parámetro	Descripción
<code>var.vm_name</code>	Nombre de la interfaz de red, vinculado a la VM.
<code>var.resource_group</code>	Grupo de recursos donde se crea la NIC.
<code>var.location</code>	Región donde se despliega la interfaz.
<code>var.subnet_id</code>	Identificador de la subred a la que se conecta la NIC.
<code>var.public_ip_address_id</code>	Asigna la IP pública estática previamente definida.
<code>private_ip_address_allocation = "Dynamic"</code>	Permite que Azure asigne automáticamente una IP privada a la VM.
<code>var.tags</code>	Se incluyen etiquetas para organización.

Máquina Virtual Linux

main.tf

```
resource "azurerm_linux_virtual_machine" "vm" {
  name                = var.vm_name
  resource_group_name = var.resource_group
  location            = var.location
  size               = var.vm_size
  admin_username     = var.admin_username
  network_interface_ids = [azurerm_network_interface.nic.id]

  admin_ssh_key {
    username = var.admin_username
    public_key = var.ssh_public_key
  }

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = var.image_offer
    sku       = var.image_os
    version   = "latest"
  }
}
```

```
tags = var.tags
}
```

Parámetro	Descripción
<code>var.vm_name</code>	Nombre de la máquina virtual.
<code>var.resource_group</code>	Grupo de recursos en el que se despliega la VM.
<code>var.location</code>	Región donde se despliega la máquina.
<code>var.vm_size</code>	Tipo de máquina virtual seleccionada para optimizar coste y rendimiento.
<code>var.admin_username</code>	Usuario administrador de la VM.
<code>var.ssh_public_key</code>	Clave pública SSH para autenticación sin contraseña.
<code>var.network_interface_ids</code>	Conecta la VM a la interfaz de red creada.
<code>caching = "ReadWrite"</code>	Optimización del rendimiento del disco del sistema.
<code>storage_account_type = "Standard_LRS"</code>	Tipo de almacenamiento del disco OS, seleccionado por costo y disponibilidad.
<code>var.image_offer</code>	Imagen de sistema operativo en el Azure Marketplace.
<code>var.image_os</code>	Versión específica del sistema operativo (Ubuntu 22.04 LTS).
<code>var.tags</code>	Etiquetas para gestión y organización dentro de Azure.

Fichero `network.tf`

El fichero `network.tf` del módulo de la máquina virtual recoge los siguientes recursos:

- *Red Virtual (VNet)* → Define el espacio de direcciones y la conectividad general.
- *Subred* → Segmenta la red dentro de la VNet, optimizando la asignación de direcciones IP.

Red Virtual (VNet)

network.tf

```
resource "azurerm_virtual_network" "vnet" {
  name                = var.vnet_name
  resource_group_name = var.resource_group
  location            = var.location
  address_space       = ["10.0.0.0/16"]
  tags                = var.tags
}
```

Parámetro	Descripción
<code>var.vnet_name</code>	Nombre de la red virtual, definido como variable para flexibilidad.
<code>var.resource_group</code>	Grupo de recursos donde se despliega la VNet.
<code>var.location</code>	Región de Azure donde se crea la red.
<code>address_space = ["10.0.0.0/16"]</code>	Espacio de direcciones IP asignado a la red virtual, lo que permite futuras segmentaciones.
<code>var.tags</code>	Etiquetas opcionales para organización y gestión en Azure.

Subred

network.tf

```
resource "azurerm_subnet" "subnet" {
  name                = var.subnet_name
  resource_group_name = var.resource_group
  virtual_network_name = azurerm_virtual_network.vnet.name
}
```

```
address_prefixes = [var.subnet_cidr]
}
```

Parámetro	Descripción
<code>var.subnet_name</code>	Nombre de la subred dentro de la VNet.
<code>var.resource_group</code>	Grupo de recursos en el que se define la subred.
<code>var.virtual_network_name</code>	Relación con la red virtual a la que pertenece la subred.
<code>address_prefixes = [var.subnet_cidr]</code>	Define el rango de direcciones IP asignado a la subred (<code>10.0.1.0/28</code>), optimizando el uso de IPs.

Fichero `security.tf`

El fichero `security.tf` del módulo de la máquina virtual recoge los siguientes recursos:

- *Grupo de Seguridad de Red (NSG)* → Gestiona las reglas de tráfico para la máquina virtual.
- *Reglas de Seguridad (Security Rules)* → Permiten o bloquean tráfico en puertos específicos.

Grupo de Seguridad de Red (NSG)

security.tf

```
resource "azurerm_network_security_group" "vm_nsg" {
  name                = "${var.vm_name}-nsg"
  resource_group_name = var.resource_group
  location            = var.location
}
```

Parámetro	Descripción
<code>var.vm_name</code>	Nombre del grupo de seguridad, vinculado a la VM.
<code>var.resource_group</code>	Grupo de recursos donde se crea el NSG.
<code>var.location</code>	Región de Azure donde se despliega el NSG.

Regla para permitir SSH (Puerto 22)

security.tf

```
resource "azurerm_network_security_rule" "allow_ssh" {
  name                = "Allow-SSH"
  priority             = 1000
  direction            = "Inbound"
  access              = "Allow"
  protocol             = "Tcp"
  source_port_range    = "*"
  destination_port_range = "22"
  source_address_prefix = "*"
  destination_address_prefix = "*"
  network_security_group_name = azurerm_network_security_group.vm_nsg.name
}
```

Parámetro	Descripción
<code>priority = 1000</code>	Asigna una prioridad alta para esta regla.
<code>direction = "Inbound"</code>	Define que la regla aplica al tráfico entrante.
<code>access = "Allow"</code>	Permite el tráfico en el puerto 22.
<code>protocol = "Tcp"</code>	Especifica que la regla aplica a conexiones TCP.
<code>destination_port_range = "22"</code>	Permite el acceso SSH a la VM.

Regla para permitir HTTP (Puerto 80)

security.tf

```
resource "azurerm_network_security_rule" "allow_http" {
  name                = "Allow-HTTP"
  priority             = 1010
  direction           = "Inbound"
  access              = "Allow"
  protocol            = "Tcp"
  source_port_range   = "*"
  destination_port_range = "80"
  source_address_prefix = "*"
  destination_address_prefix = "*"
  network_security_group_name = azurerm_network_security_group.vm_nsg.name
}
```

Parámetro	Descripción
priority = 1010	Define la prioridad de la regla para HTTP.
destination_port_range = "80"	Habilita tráfico en el puerto 80 para servir contenido web.

Regla para permitir HTTPS (Puerto 443)

```
security.tf

resource "azurerm_network_security_rule" "allow_https" {
  name                = "Allow-HTTPS"
  priority             = 1020
  direction           = "Inbound"
  access              = "Allow"
  protocol            = "Tcp"
  source_port_range   = "*"
  destination_port_range = "443"
  source_address_prefix = "*"
  destination_address_prefix = "*"
  network_security_group_name = azurerm_network_security_group.vm_nsg.name
}
```

Parámetro	Descripción
priority = 1020	Prioridad asignada a la regla HTTPS.
destination_port_range = "443"	Habilita tráfico en el puerto 443 para conexiones seguras.

Regla para permitir todo el tráfico de salida

```
security.tf

resource "azurerm_network_security_rule" "allow_outbound" {
  name                = "Allow-All-Outbound"
  priority             = 900
  direction           = "Outbound"
  access              = "Allow"
  protocol            = "*"
  source_port_range   = "*"
  destination_port_range = "*"
  source_address_prefix = "*"
  destination_address_prefix = "*"
  network_security_group_name = azurerm_network_security_group.vm_nsg.name
}
```

Parámetro	Descripción
priority = 900	Define una prioridad más baja que las reglas de entrada.
direction = "Outbound"	Aplica la regla al tráfico saliente.
access = "Allow"	Permite que la VM se comunique con otros servicios.
protocol = "*"	Permite cualquier protocolo.
destination_port_range = "*"	No restringe los puertos de destino.

KUBERNETES SERVICE (AKS)

La infraestructura del **Azure Kubernetes Service (AKS)** se ha definido en **Terraform** dentro de un módulo independiente para asegurar una correcta organización y reutilización del código. Este módulo define los recursos necesarios para desplegar un clúster de Kubernetes gestionado por Azure.

```
terraform/
├─ terraform.tfvars      # Variables globales del despliegue
├─ main.tf               # Llamada a módulos y recursos principales
├─ modules/
│   └─ aks/              # Módulo de Kubernetes Service (AKS)
│       ├── main.tf      # Definición del clúster de Kubernetes
│       ├── outputs.tf   # Variables de salida (Cluster ID, Node Pool ID)
│       └── variables.tf  # Definición de variables del módulo
```

Puedes ver las evidencias de este despliegue en el  [siguiente enlace](#).

Fichero `main.tf`

El fichero `main.tf` del módulo de AKS incluye los siguientes recursos:

- **Clúster de Kubernetes (AKS)** → Crea una instancia de **Azure Kubernetes Service** con un *node pool* por defecto y acceso RBAC habilitado.
- **Role Assignment para ACR** → Permite a AKS acceder al **Azure Container Registry (ACR)** para extraer imágenes de contenedores.

Clúster de Kubernetes (AKS)

main.tf

```
resource "azurerm_kubernetes_cluster" "aks" {
  name                = var.aks_name
  location            = var.location
  resource_group_name = var.resource_group
  dns_prefix          = var.dns_prefix
  sku_tier            = "Standard"

  default_node_pool {
    name         = "default"
    node_count   = var.node_count
    vm_size      = var.vm_size
    os_disk_size_gb = 30
  }

  identity {
    type = "SystemAssigned"
  }

  role_based_access_control_enabled = true
```

```
tags = var.tags
}
```

Parámetro	Descripción
<code>var.aks_name</code>	Nombre del clúster de AKS.
<code>var.resource_group</code>	Grupo de recursos donde se despliega el AKS.
<code>var.location</code>	Región de Azure donde se despliega.
<code>var.dns_prefix</code>	Prefijo DNS único del clúster.
<code>sku_tier = "Standard"</code>	Define el nivel del servicio de Kubernetes.
<code>default_node_pool</code>	Define el grupo de nodos (<i>Node Pool</i>) que ejecutará los contenedores.
<code>var.node_count</code>	Número de nodos en el <i>node pool</i> por defecto.
<code>var.vm_size</code>	Tamaño de las máquinas virtuales utilizadas como nodos.
<code>os_disk_size_gb = 30</code>	Tamaño del disco de cada nodo.
<code>identity { type = "SystemAssigned" }</code>	Se asigna una identidad gestionada para que el clúster pueda autenticarse con otros servicios de Azure.
<code>role_based_access_control_enabled = true</code>	Habilita RBAC para gestionar permisos dentro del clúster.
<code>var.tags</code>	Etiquetas para organización y gestión dentro de Azure.

Role Assignment para ACR

```
main.tf

resource "azurerm_role_assignment" "acr_pull" {
  scope           = var.acr_id
  role_definition_name = "AcrPull"
  principal_id    = azurerm_kubernetes_cluster.aks.identity[0].principal_id
}
```

Parámetro	Descripción
<code>var.acr_id</code>	ID del Azure Container Registry asociado al clúster.
<code>role_definition_name = "AcrPull"</code>	Asigna el rol de AcrPull , que permite a AKS extraer imágenes de contenedores desde ACR.
<code>principal_id</code>	Se refiere a la identidad asignada al clúster de AKS para la autenticación con ACR.

2.2.3 Configuración de la infraestructura

A continuación se describen las configuraciones aplicadas a la infraestructura desplegada, automatizadas con Ansible, y la justificación de cada una de ellas.

Imágenes contenerizadas

IMÁGEN SIN PERSISTENCIA PARA LA VM

La imagen utilizada en el contenedor Podman dentro de la máquina virtual se basa en **MkDocs**, una librería de documentación escrita en Python. Esta herramienta permite generar sitios estáticos a partir de archivos Markdown, facilitando la creación y publicación de documentación técnica ([MkDocs, s.f.](#)). La imagen generada en este ejercicio contiene la documentación del propio proyecto, asegurando que el contenido se pueda visualizar de manera estructurada en un navegador.

Además, se ha utilizado el tema **Material for MkDocs**, que añade una interfaz moderna y varias opciones de personalización ([Squidfunk, s.f.](#)).

La documentación también está disponible a través de **GitHub Pages**, lo que permite su acceso incluso cuando la infraestructura de Azure no está desplegada. Se puede visualizar en el siguiente enlace:

 [Ver documentación en GitHub Pages](#)

IMÁGEN CON PERSISTENCIA PARA EL AKS

La imagen desplegada en el clúster de **AKS** está basada en **StackEdit**, una aplicación web de código abierto que permite editar y guardar documentos en formato Markdown directamente desde el navegador. Esta herramienta es ideal para la toma de notas técnicas o redacción de documentación rápida, ya que ofrece previsualización en tiempo real y sincronización con almacenamiento local y en la nube ([StackEdit, s.f.](#)).

Para este ejercicio se ha utilizado la imagen pública disponible en Docker Hub: [benweet/stackedit](#), la cual se despliega en un contenedor dentro de Kubernetes con un volumen persistente asociado. Esto garantiza que el contenido creado por el usuario, como notas o documentos, **no se pierde** aunque el contenedor se reinicie o se re programe, validando así la persistencia de los datos en un entorno dinámico.

Configuración con Ansible

Para la configuración y automatización del despliegue en la infraestructura se ha utilizado Ansible, organizando las tareas en roles específicos, siguiendo las buenas prácticas recomendadas en la documentación oficial de Ansible [Ansible. \(s.f.-a\)](#).

La ejecución de los archivos está estructurada de la siguiente manera:

```

ansible
├── hosts.tpl           # Plantilla del inventario dinámico
├── playbook.yml        # Orquesta todos los roles
├── publish_images.yml  # Publica imágenes en el ACR
├── vm_deployment.yml  # Despliega en el contenedor de la VM
├── aks_deployment.yml  # Despliega en el contenedor del AKS
├── roles
│   ├── acr             # Rol para la publicación en el ACR
│   ├── vm              # Rol para la configuración de la VM
│   └── aks              # Rol para la configuración de la VM
├── secrets.yml         # Variables sensibles
└── vars.yml            # Variables generales del despliegue

```

- **ACR:** Gestiona la publicación de imágenes en **Azure Container Registry (ACR)**, construyendo y empujando imágenes desde la VM y desde la máquina local.
- **VM:** Configura la máquina virtual, instalando **Podman**, desplegando el contenedor con **MkDocs**, gestionando autenticaciones y asegurando la persistencia con **Systemd**.
- **AKS** (*no presente en este esquema, pero estructurable de forma similar*): Se encargaría de desplegar aplicaciones en **Azure Kubernetes Service (AKS)**.

ROL ACR

Para configurar el ACR se publicarán dos imágenes contenerizadas: una corresponde a un sitio estático en Nginx, que será desplegado en una máquina virtual con Podman, y la otra es una aplicación con persistencia que será ejecutada en un contenedor dentro de Azure Kubernetes Service (AKS).

Puedes ver las evidencias de este rol en el  [siguiente enlace](#).

Este proyecto permite la publicación de las imágenes en el ACR de dos maneras:

- Publicación mediante Ansible.
- Publicación manual mediante Github Actions (fuera de alcance).

Para la publicación usando Ansible se ha generado un rol llamado `acr` que contiene todas las tareas necesarias y se estructura de la siguiente manera:

```

ansible/
├── roles/
│   └── acr/
│       ├── tasks/
│       │   ├── main.yml           # Rol para gestionar ACR en Ansible
│       │   ├── install.yml        # Tareas que se ejecutan en el ACR
│       │   ├── build_docs.yml     # Inclusión de todas las tareas
│       │   ├── login.yml          # Instala podman en la VM
│       │   ├── push_mkdocs.yml    # Construcción de las imágenes
│       │   └── push_stackedit.yml # Iniciar sesión en ACR
│       ├── vars/                  # Publicación de mkdocs en ACR
│       │   └── main.yml           # Publicación de stackedit en ACR
│       └── vars/                  # Variables específicas del rol
│           └── main.yml           # Configuración de parámetros

```

El fichero `tasks/main.yml` dentro del rol `acr`, gestiona la configuración y publicación de imágenes en la máquina virtual y el Azure Container Registry (ACR).

main.yml

```

---
- name: Install Podman on the VM
  include_tasks: install.yml

- name: Build MkDocs image
  include_tasks: build_docs.yml

- name: Login into ACR from the VM
  include_tasks: login.yml

- name: Push mkdocs image to ACR from the VM
  include_tasks: push_mkdocs.yml

- name: Push stackedit image to ACR from localhost
  include_tasks: push_stackedit.yml

```

Instalar Podman

Esta tarea instala Podman en la máquina virtual asegurándose de que esté disponible en el sistema. Además, actualiza la caché de paquetes antes de la instalación.

install.yml

```

---
- name: Install Podman
  apt:
    name: podman
    state: present
    update_cache: yes

```

Construir imagen mkdocs

Clona el repositorio del proyecto en la máquina virtual, instala dependencias necesarias para MkDocs y WeasyPrint, construye el sitio estático de MkDocs y genera una imagen de contenedor con Podman basada en el `Dockerfile.docs`.

build_docs.yml

```

---
- name: Ensure repository is present on the VM

```

```

git:
  repo: "https://github.com/charlstown/unir-cp2.git"
  dest: "/opt/unir-cp2"
  version: main

- name: Install dependencies for MkDocs
  apt:
    name:
      - python3-pip
    state: present
    update_cache: no
    become: yes

- name: Install required system dependencies for WeasyPrint
  apt:
    name:
      - libpango1.0-0
      - libpangocairo-1.0-0
      - libcairo2
    state: present
    update_cache: no
    become: yes

- name: Install project dependencies
  pip:
    requirements: "/opt/unir-cp2/requirements.txt"

- name: Build MkDocs static site
  command:
    cmd: mkdocs build
    chdir: "/opt/unir-cp2"

- name: Build Podman image on the VM
  command:
    cmd: podman build -t "{{ image_name_docs }}:{{ image_tag_docs }}" -f /opt/unir-cp2/Dockerfile.docs
    chdir: "/opt/unir-cp2"

```

Login en el ACR

Realiza la autenticación en Azure Container Registry (ACR) desde la máquina virtual utilizando Podman, empleando credenciales de usuario y contraseña.

login_acr.yml

```

---
- name: Log in to ACR from the VM
  command: >
    podman login {{ acr_login_server }}
    -u {{ acr_username }}
    --password {{ acr_password }}

```

Publicar imagen mkdocs-nginx

Etiqueta la imagen generada de MkDocs con el formato adecuado para ACR y la sube al registro de contenedores de Azure desde la máquina virtual.

push_mkdocs.yml

```

---
# Push MkDocs image
- name: Tag MkDocs image for ACR
  command: >
    podman tag {{ image_name_docs }}:{{ image_tag_docs }}
    {{ acr_login_server }}/{{ image_name_docs }}:{{ image_tag_docs }}

- name: Push MkDocs image to ACR from the VM
  command: >
    podman push {{ acr_login_server }}/{{ image_name_docs }}:{{ image_tag_docs }}

```

Publicar imagen stackedit

Descarga la imagen `stackedit-base` desde Docker Hub, la etiqueta para el ACR y finalmente la sube al registro de Azure.

push_stackedit.yml

```

---
- name: Pull StackEdit image from Docker Hub
  command: >
    podman pull docker.io/benweet/stackedit-base:latest
    become: yes

- name: Tag StackEdit image for ACR
  command: >
    podman tag docker.io/benweet/stackedit-base:latest {{ acr_name }}.azurecr.io/{{ image_name_stackedit }}:{{ image_tag_stackedit }}

```

```
become: yes

- name: Push StackEdit image to ACR
  command: >
    podman push {{ acr_name }}.azurecr.io/{{ image_name_stackedit }}:{{ image_tag_stackedit }}
  become: yes
```

ROL VM

Para la publicación usando Ansible se ha generado un rol llamado `vm` que contiene todas las tareas necesarias y se estructura de la siguiente manera:

```
ansible/
├── roles
│   └── vm
│       ├── handlers
│       │   └── main.yml
│       ├── tasks
│       │   ├── auth.yml
│       │   ├── container.yml
│       │   ├── main.yml
│       │   └── systemd.yml
│       └── vars
│           └── main.yml
```

Puedes ver las evidencias de este rol en el  [siguiente enlace](#).

El fichero `tasks/main.yml` dentro del rol `acr`, gestiona la configuración y publicación de imágenes en la máquina virtual y el Azure Container Registry (ACR).

main.yml

```
- name: Include authentication setup
  import_tasks: auth.yml

- name: Include container deployment
  import_tasks: container.yml

- name: Include systemd configuration
  import_tasks: systemd.yml
```

Autenticación básica

En esta tarea se configura autenticación básica en Nginx mediante `htpasswd`, asegurando que solo usuarios autorizados puedan acceder. Se instala Apache Utils, se crea el directorio de autenticación y se genera un archivo de credenciales.

```
---
- name: Install Apache Utils for htpasswd
  apt:
    name: apache2-utils
    state: present
  become: yes

- name: Ensure authentication directory exists
  file:
    path: /etc/nginx/auth
    state: directory
    mode: '0755'

- name: Load secure variables
  include_vars: secrets.yml

- name: Generate htpasswd file
  command: htpasswd -bc /etc/nginx/auth/htpasswd.users charlstown "{{ site_pwd }}"
  args:
    creates: /etc/nginx/auth/htpasswd.users
```

Desplegar contenedor

En esta tarea se inicia sesión en el ACR para descargar la imagen del contenedor y se ejecuta con soporte SSL y autenticación básica, vinculando el archivo de credenciales generado en el paso anterior.

```
---
- name: Log into Azure Container Registry (ACR)
  containers.podman.podman_login:
    registry: "{{ acr_name }}.azurecr.io"
    username: "{{ acr_username }}"
    password: "{{ acr_password }}"
```

```
- name: Run container from ACR image with SSL and Basic Auth
  containers.podman.podman_container:
    name: mkdocs_container
    image: "{{ acr_name }}.azurecr.io/{{ image_name }}:{{ image_tag }}"
    state: started
    restart_policy: always
    ports:
      - "443:443"
    volume:
      - "/etc/nginx/auth/htpasswd.users:/etc/nginx/.htpasswd:ro"
```

Disponibilidad como servicio

En esta tarea se convierte el contenedor en un servicio systemd, esto garantiza la disponibilidad continua del servicio sin intervención manual, ya que systemd lo monitorea y lo vuelve a iniciar si detecta que ha dejado de funcionar.

```
---
- name: Generate systemd service for Podman container
  containers.podman.podman_generate_systemd:
    name: mkdocs_container
    dest: /etc/systemd/system/
    restart_policy: always

- name: Enable and start Podman container systemd service
  systemd:
    name: container-mkdocs_container
    enabled: yes
    state: started
    daemon_reload: yes
```

ROL AKS

Para el despliegue de la aplicación en el clúster de Kubernetes mediante Ansible se ha generado un rol llamado `aks`, que contiene todas las tareas necesarias y se estructura de la siguiente manera:

```
ansible/
├── roles
│   └── aks
│       ├── tasks
│       │   ├── acr_auth.yml
│       │   ├── deploy.yml
│       │   ├── main.yml
│       │   ├── namespace.yml
│       │   ├── pvc.yml
│       │   └── service.yml
│       ├── templates
│       │   ├── acr_auth.json.j2
│       │   ├── deployment.yml.j2
│       │   ├── pvc.yml.j2
│       │   └── service.yml.j2
│       └── vars
│           └── main.yml
```

Puedes ver las evidencias de este rol en el  [siguiente enlace](#).

El fichero `tasks/main.yml` dentro del rol `aks` orquesta todas las tareas necesarias para desplegar la aplicación, incluyendo la creación del namespace, los volúmenes persistentes, el despliegue de los contenedores y el servicio de acceso.

main.yml

```
- name: Create Kubernetes Namespace
  import_tasks: namespace.yml

- name: Create ACR Secret in Kubernetes
  import_tasks: acr_auth.yml

- name: Apply PersistentVolumeClaim
  import_tasks: pvc.yml

- name: Deploy Application
  import_tasks: deploy.yml

- name: Create LoadBalancer Service
  import_tasks: service.yml
```

Crear Namespace

Esta tarea se encarga de crear el namespace donde se desplegarán todos los recursos de la aplicación dentro del clúster de AKS, asegurando su aislamiento lógico del resto de workloads.

namespace.yml

```
---
- name: Create Kubernetes Namespace
  kubernetes.core.k8s:
    name: "{{ namespace }}"
    api_version: v1
    kind: Namespace
    state: present
```

Crear secreto en el ACR

Esta tarea crea un `Secret` en el clúster de AKS con las credenciales necesarias para acceder al Azure Container Registry (ACR), permitiendo que Kubernetes pueda descargar imágenes privadas.

acr_auth.yml

```
- name: Create ACR Secret in Kubernetes
  kubernetes.core.k8s:
    state: present
    namespace: "{{ namespace }}"
    definition:
      apiVersion: v1
      kind: Secret
      metadata:
        name: acr-secret
      type: kubernetes.io/dockerconfigjson
      data:
        .dockerconfigjson: "{{ lookup('template', 'acr-auth.json.j2') | from_yaml | to_json | b64encode }}"
```

El secreto se genera a partir de la plantilla `acr-auth.json.j2`, que contiene las credenciales codificadas en base64:

acr-auth.json.j2

```
{
  "auths": {
    "{{ acr_name }}.azurecr.io": {
      "username": "{{ acr_username }}",
      "password": "{{ acr_password }}",
      "auth": "{{ (acr_username + ':' + acr_password) | b64encode }}"
    }
  }
}
```

Crear volumen persistente

Esta tarea crea un `PersistentVolumeClaim` en el clúster de AKS, necesario para mantener los datos persistentes entre reinicios del contenedor.

pvc.yml

```
- name: Apply PersistentVolumeClaim
  kubernetes.core.k8s:
    state: present
    namespace: "{{ namespace }}"
    definition: "{{ lookup('template', 'pvc.yml.j2') }}"
```

La plantilla utilizada define un volumen de 5GiB con acceso en modo lectura-escritura por un único nodo:

pvc.yml.j2

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{ pvc_name }}
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Desplegar aplicación

Esta tarea aplica el `Deployment` de Kubernetes necesario para ejecutar la aplicación StackEdit. Se especifica la imagen publicada en el ACR, el puerto interno del contenedor, el volumen persistente y las credenciales de acceso al registro.

deploy.yml

```
- name: Deploy Application
  kubernetes.core.k8s:
    state: present
    namespace: "{{ namespace }}"
    definition: "{{ lookup('template', 'deployment.yml.j2') }}"
```

La plantilla del manifiesto define una réplica del contenedor con puerto interno `8080` y volumen montado en `/data`:

deployment.yml.j2

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stackedit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: stackedit
  template:
    metadata:
      labels:
        app: stackedit
    spec:
      containers:
        - name: stackedit
          image: "{{ acr_name }}.azurecr.io/{{ image_name_stackedit }}:{{ image_tag_stackedit }}"
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: storage
              mountPath: "/data"
          env:
            - name: ENV_VAR
              value: "example-value"
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: {{ pvc_name }}
      imagePullSecrets:
        - name: acr-secret
```

2.3 Despliegue

A continuación, se explica cómo reproducir los pasos necesarios para llevar a cabo el caso práctico sobre el repositorio. Se detallan las instrucciones para:

- 1. Despliegue de la infraestructura
- 2. Publicación de las imágenes
- 3. Configuración de la VM
- 4. Configuración del AKS

2.3.1 Despliegue de la infraestructura

El despliegue de la infraestructura se realiza con Terraform desde la máquina local, asegurando que la configuración es válida antes de aplicar los cambios y provisionar los recursos necesarios.

1. Inicializa terraform en el directorio de ficheros terraform.

```
terraform -chdir=./terraform init
```

Output: Terraform has been successfully initialized!

2. Ejecuta la validación de los ficheros generados con el siguiente comando:

```
terraform validate
```

output: Success! The configuration is valid.

3. Despliega la infraestructura con el siguiente comando, por defecto se despliega en dev. Siempre puedes añadir el flag `-var="environment=pro"` para especificar un entorno entre `dev|pre|pro`

```
terraform -chdir=./terraform apply --auto-approve
```



Automatización de variables

Tras el despliegue de toda la infraestructura se generan automáticamente las variables globales necesarias para poder realizar lo que queda del ejercicio ejecutando el fichero `setup.sh`.

```
source setup.sh
```

2.3.2 Publicación de las imágenes

Publicación de las imágenes mediante Ansible

Para publicar imágenes en el ACR utilizando Ansible, se ha creado un playbook llamado `publish-images.yaml`. Para llevar a cabo su ejecución, es necesario ejecutar desde el directorio de ansible el siguiente comando.

```
ansible-playbook publish_images.yml -i hosts.yml --ask-vault-pass
```

Este comando construye la imagen de MkDocs, descarga la imagen pública de StackEdit y publica ambas imágenes en el ACR desde la VM.

Publicación mediante Github Actions (fuera de alcance)

En este apartado se explica la publicación de imágenes en el ACR utilizando GitHub Actions. Aunque no formaba parte del alcance del ejercicio, se ha implementado este método para probar un flujo habitual en proyectos donde un repositorio genera y publica imágenes de contenedor tras una release.

La publicación de la imagen se automatiza mediante el workflow [Publish mldocs image to ACR](#) de GitHub Actions, que envía la imagen al Azure Container Registry (ACR). Para ello, se deben proporcionar las credenciales adecuadas y validar la ejecución del proceso.

1. Rellenar los datos del formulario del workflow con username y pwd del ACR desplegado en Azure.

Visualizar usuario y contraseña del ACR

Siempre puedes ejecutar este comando para recuperar el usuario y la contraseña del ACR.

```
az acr credential show --name acrweucp2dev --query "[username, passwords[0].value]" -o tsv
```

The screenshot shows the configuration page for a GitHub Actions workflow. At the top right is a 'Run workflow' button. Below it, the 'Use workflow from' section shows 'Branch: main'. The main configuration fields are:

- Azure Container Registry Name ***: acrweucp2dev
- Azure Container Registry Username ***: acrweucp2dev
- Azure Container Registry Password ***: (empty field)
- Container Image Name ***: docs-nginx
- Tag Name ***: 1.0.0

 At the bottom is a green 'Run workflow' button.

2. Ejecutar workflow y validar la correcta ejecución del job

2.3.3 Configuración de la VM

La configuración de la VM se llevará a cabo desde la máquina local utilizando Ansible, accediendo por SSH para realizar comprobaciones y garantizar el correcto despliegue del entorno.

1. Comprobar conexión a la VM por SSH

```
ssh -i ~/.ssh/az_unir_rsa charlstown@${VM_IP}
exit
```

2. Ejecutar ansible apuntando a la VM. Asegurarse que el comando se ejecuta desde `./ansible`. Para forzar ansible a recrear todo desde el principio es posible usar los argumentos `--force-handlers` y `--extra-vars "recreate=true"`.

```
ansible-playbook ansible/playbook.yml -i ansible/hosts.yml --extra-vars "@ansible/vars.yml" --ask-vault-pass
```

Este playbook se ejecuta apuntando a un Vault de ansible donde se han guardado las credenciales usadas para crear el fichero `htpasswd.users` en la carpeta `/etc/nginx/auth/htpasswd.users` de la VM.

Mostrar contraseñas guardadas en el vault

Para visualizar las contraseñas guardadas en el vault puedes ejecutar el comando:

```
ansible-vault view secrets.yml
```

2.3.4 Configuración del AKS

El despliegue de la aplicación en el clúster de Kubernetes se realiza mediante Ansible, aplicando los manifiestos necesarios para crear el namespace, el deployment, el PersistentVolumeClaim, el Service y el secret de acceso al ACR. Todo el proceso queda automatizado en el playbook `playbook_aks.yml`.

1. Descargar credenciales del AKS para interactuar con el clúster desde kubectl.

El siguiente comando guarda las credenciales del AKS en `/home/<USER>/.kube/config` y marca como contexto el AKS seleccionado.

```
az aks get-credentials --resource-group rg-weu-cp2-dev --name aks-weu-cp2-dev
```


2. Ejecutar el playbook de Ansible para desplegar la aplicación en AKS. Este comando debe lanzarse desde la raíz del proyecto.

```
ansible-playbook playbook_aks.yml -i hosts.yml --ask-vault-pass
```

3. Para obtener la IP pública del servicio y acceder a la aplicación desplegada, ejecuta el siguiente comando.

```
kubectl get svc stackedit-service -n cp2 -o jsonpath="{.status.loadBalancer.ingress[0].ip}"
```

La aplicación estará disponible en esa dirección IP a través del puerto 80.

 Con estos pasos, se completa el despliegue y configuración íntegra del caso práctico: se ha provisionado toda la infraestructura necesaria, se han publicado las imágenes de contenedor en el ACR, y se ha puesto en marcha tanto la VM como el clúster de AKS. El entorno queda totalmente funcional, con los contenedores desplegados y ejecutándose a partir de sus respectivas imágenes.

2.4 Evidencias

A continuación, se exponen las evidencias de los procesos empleados para realizar la práctica.

- [1. Despliegue de la infraestructura](#)
- [2. Publicación de las imágenes](#)
- [3. Despliegue en la VM](#)
- [4. Despliegue en el AKS](#)

2.4.1 Despliegue de la infraestructura

En esta sección se muestran los logs los principales comandos ejecutados y algunas capturas que demuestran la correcta ejecución del caso práctico.

Lanzamos un `terraform plan` para comprobar todos los recursos.

```
terraform -chdir=./terraform plan
```

output

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

```
# local_file.ansible_inventory will be created
+ resource "local_file" "ansible_inventory" {
  + content          = (sensitive value)
  + content_base64sha256 = (known after apply)
  + content_base64sha512 = (known after apply)
  + content_md5       = (known after apply)
  + content_sha1      = (known after apply)
  + content_sha256    = (known after apply)
  + content_sha512    = (known after apply)
  + directory_permission = "0777"
  + file_permission   = "0777"
  + filename         = "../ansible/hosts.yml"
  + id               = (known after apply)
}

# module.aks.azure_rm_kubernetes_cluster.aks will be created
+ resource "azurerm_kubernetes_cluster" "aks" {
  + api_server_authorized_ip_ranges = (known after apply)
  + current_kubernetes_version      = (known after apply)
  + dns_prefix                      = "aksweucp2"
  + fqdn                           = (known after apply)
  + http_application_routing_zone_name = (known after apply)
  + id                             = (known after apply)
  + image_cleaner_enabled          = false
  + image_cleaner_interval_hours   = 48
  + kube_admin_config              = (sensitive value)
  + kube_admin_config_raw          = (sensitive value)
  + kube_config                   = (sensitive value)
  + kube_config_raw                = (sensitive value)
  + kubernetes_version             = (known after apply)
  + location                      = "westeurope"
  + name                          = "aks-weu-cp2-dev"
  + node_resource_group            = (known after apply)
  + node_resource_group_id         = (known after apply)
  + oidc_issuer_url                = (known after apply)
  + portal_fqdn                   = (known after apply)
  + private_cluster_enabled        = false
  + private_cluster_public_fqdn_enabled = false
  + private_dns_zone_id            = (known after apply)
  + private_fqdn                   = (known after apply)
  + public_network_access_enabled   = true
  + resource_group_name            = "rg-weu-cp2-dev"
  + role_based_access_control_enabled = true
  + run_command_enabled            = true
  + sku_tier                       = "Standard"
  + support_plan                   = "KubernetesOfficial"
  + tags                           = {
    + "environment" = "casopractico2"
  }
  + workload_identity_enabled = false
}
```

```

+ api_server_access_profile (known after apply)

+ auto_scaler_profile (known after apply)

+ default_node_pool {
  + kubernetes_disk_type = (known after apply)
  + max_pods              = (known after apply)
  + name                  = "default"
  + node_count            = 1
  + node_labels           = (known after apply)
  + orchestrator_version = (known after apply)
  + os_disk_size_gb       = 30
  + os_disk_type          = "Managed"
  + os_sku                 = (known after apply)
  + scale_down_mode       = "Delete"
  + type                  = "VirtualMachineScaleSets"
  + ultra_ssd_enabled     = false
  + vm_size               = "Standard_B2s"
  + workload_runtime      = (known after apply)
}

+ identity {
  + principal_id = (known after apply)
  + tenant_id   = (known after apply)
  + type        = "SystemAssigned"
}

+ kubernetes_identity (known after apply)

+ network_profile (known after apply)

+ windows_profile (known after apply)
}

# module.aks.azure_rm_role_assignment.acr_pull will be created
+ resource "azurerm_role_assignment" "acr_pull" {
  + id                  = (known after apply)
  + name                = (known after apply)
  + principal_id        = (known after apply)
  + principal_type      = (known after apply)
  + role_definition_id  = (known after apply)
  + role_definition_name = "AcrPull"
  + scope               = (known after apply)
  + skip_service_principal_aad_check = (known after apply)
}

# module.container_registry.azure_rm_container_registry.acr will be created
+ resource "azurerm_container_registry" "acr" {
  + admin_enabled           = true
  + admin_password          = (sensitive value)
  + admin_username          = (known after apply)
  + encryption              = (known after apply)
  + export_policy_enabled   = true
  + id                     = (known after apply)
  + location                = "westeurope"
  + login_server            = (known after apply)
  + name                   = "acrweucp2dev"
  + network_rule_bypass_option = "AzureServices"
  + network_rule_set        = (known after apply)
  + public_network_access_enabled = true
  + resource_group_name     = "rg-weu-cp2-dev"
  + retention_policy        = (known after apply)
  + sku                    = "Basic"
  + tags                    = {
    + "environment" = "casopractico2"
  }
  + trust_policy            = (known after apply)
  + zone_redundancy_enabled = false
}

# module.virtual_machine.azure_rm_linux_virtual_machine.vm will be created
+ resource "azurerm_linux_virtual_machine" "vm" {
  + admin_username          = "charlstown"
  + allow_extension_operations = true
  + bypass_platform_safety_checks_on_user_schedule_enabled = false
  + computer_name           = (known after apply)
  + disable_password_authentication = true
  + disk_controller_type    = (known after apply)
  + extensions_time_budget  = "PT1H30M"
  + id                     = (known after apply)
  + location                = "westeurope"
  + max_bid_price           = -1
  + name                   = "vm-weu-cp2-docs-dev"
  + network_interface_ids   = (known after apply)
  + patch_assessment_mode   = "ImageDefault"
  + patch_mode              = "ImageDefault"
  + platform_fault_domain   = -1
  + priority                = "Regular"
  + private_ip_address      = (known after apply)
  + private_ip_addresses    = (known after apply)
  + provision_vm_agent      = true
  + public_ip_address       = (known after apply)
  + public_ip_addresses     = (known after apply)
  + resource_group_name     = "rg-weu-cp2-dev"
}

```

```

+ size = "Standard_B1ls"
+ tags = {
  + "environment" = "casopractico2"
}
+ virtual_machine_id = (known after apply)
+ vm_agent_platform_updates_enabled = false

+ admin_ssh_key {
  + public_key = <<-EOT
    \*\*\*
    EOT
  + username = "charlstown"
}

+ os_disk {
  + caching = "ReadWrite"
  + disk_size_gb = (known after apply)
  + name = (known after apply)
  + storage_account_type = "Standard_LRS"
  + write_accelerator_enabled = false
}

+ source_image_reference {
  + offer = "0001-com-ubuntu-server-jammy"
  + publisher = "Canonical"
  + sku = "22_04-lts-gen2"
  + version = "latest"
}

+ termination_notification (known after apply)
}

# module.virtual_machine.azure_rm_network_interface.nic will be created
+ resource "azurerm_network_interface" "nic" {
  + accelerated_networking_enabled = (known after apply)
  + applied_dns_servers = (known after apply)
  + dns_servers = (known after apply)
  + enable_accelerated_networking = (known after apply)
  + enable_ip_forwarding = (known after apply)
  + id = (known after apply)
  + internal_domain_name_suffix = (known after apply)
  + ip_forwarding_enabled = (known after apply)
  + location = "westeurope"
  + mac_address = (known after apply)
  + name = "vm-weu-cp2-docs-dev-nic"
  + private_ip_address = (known after apply)
  + private_ip_addresses = (known after apply)
  + resource_group_name = "rg-weu-cp2-dev"
  + tags = {
    + "environment" = "casopractico2"
  }
}
+ virtual_machine_id = (known after apply)

+ ip_configuration {
  + gateway_load_balancer_frontend_ip_configuration_id = (known after apply)
  + name = "internal"
  + primary = (known after apply)
  + private_ip_address = (known after apply)
  + private_ip_address_allocation = "Dynamic"
  + private_ip_address_version = "IPv4"
  + public_ip_address_id = "/subscriptions/fb24fc1f-67e2-4871-8be2-c10a36e74c93/resourceGroups/rg-weu-cp2-dev/providers/
Microsoft.Network/publicIPAddresses/vm-weu-cp2-docs-dev-public-ip"
  + subnet_id = "/subscriptions/fb24fc1f-67e2-4871-8be2-c10a36e74c93/resourceGroups/rg-weu-cp2-dev/providers/
Microsoft.Network/virtualNetworks/vnet-weu-cp2-dev/subnets/subnet-weu-cp2-dev"
}
}

Plan: 6 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ acr_login_server = (known after apply)
+ acr_password = (sensitive value)
+ acr_username = (known after apply)



```

Creación de los grupos de recursos

Tras ejecutar el comando de `terraform apply` podremos ver en el apartado **Resource groups** los siguientes RGs.



```
terraform -chdir=./terraform apply --auto-approve
```


Home >


Resource groups  ... 


UNIR (comunidadunir.net)


+ Create

 Manage view 

 Refresh


 Export to CSV


 Open query

|  Assign tags


Filter for any field...


Subscription equals all







Location equals all 

 Add filter

Showing 1 to 3 of 3 records.

No grouping 



List view 

<input type="checkbox"/> Name 	Subscription 	Location 	
<input type="checkbox"/>  MC_rg-weu-cp2-dev_aks-weu-cp2-dev_westeurope	Azure for Students	West Europe	...
<input type="checkbox"/>  NetworkWatcherRG	Azure for Students	West Europe	...
<input type="checkbox"/>  rg-weu-cp2-dev	Azure for Students	West Europe	...

- **MC_rg-weu-cp2-dev_aks-weu-cp2-dev_westeurope:** Grupo de recursos gestionado automáticamente por Azure para almacenar los nodos y configuraciones internas del AKS.
- **NetworkWatcherRG:** Grupo de recursos creado por Azure para herramientas de monitoreo y diagnóstico de red.
- **rg-weu-cp2-dev:** Grupo de recursos principal donde se despliegan la VM, el ACR y el AKS mediante Terraform.



RG-WEU-CP2-DEV


El `rg-weu-cp2-dev` contiene todos los recursos declarados en nuestros ficheros de terraform.


rg-weu-cp2-dev  ... 


Resource group

+ Create

 Manage view 

 Delete resource group

 Refresh

 Export to CSV

...

Essentials

Subscription [\(move\)](#)
Azure for Students

Subscriptions ID
fb24fc1f-67e2-4871-8be2-c10a36e74c93

Tags [\(edit\)](#)
environment : casopractico2

Deployments
No deployments


Location
West Europe


[JSON View](#)


Resources


Recommendations (13)


Filter for any field...


Type equals all 












Location equals all 

 Add filter

Showing 1 to 8 of 8 records. ☐ Show hidden types 


No grouping 




List view 


<input type="checkbox"/> Name 	Type 	Location 	
<input type="checkbox"/>  acrweucp2dev	Container registry	West Europe	...
<input type="checkbox"/>  aks-cp2-dev	Kubernetes service	West Europe	...
<input type="checkbox"/>  vm-weu-cp2-docs-dev	Virtual machine	West Europe	...
<input type="checkbox"/>  vm-weu-cp2-docs-dev-nic	Network Interface	West Europe	...
<input type="checkbox"/>  vm-weu-cp2-docs-dev-nsg	Network security group	West Europe	...
<input type="checkbox"/>  vm-weu-cp2-docs-dev-public-ip	Public IP address	West Europe	...
<input type="checkbox"/>  vm-weu-cp2-docs-dev_OsDisk_1_641b2df28d0...	Disk	West Europe	...
<input type="checkbox"/>  vnet-weu-cp2-dev	Virtual network	West Europe	...

Creación del ACR

Desde el portal de Azure podemos observar como el servicio de contenedores (ACR) se ha creado correctamente bajo los parámetros definidos en los ficheros terraform.

 **acrweucp2dev**
Container registry





>

→ Move

🗑️ Delete

^ Essentials

JSON View

Resource group [\(move\)](#)
[rg-weu-cp2-dev](#)

Location
West Europe

Subscription [\(move\)](#)
[Azure for Students](#)

Subscription ID
fb24fc1f-67e2-4871-8be2-c10a36e74c93

Soft delete (Preview)
[Disabled](#)

Tags [\(edit\)](#)
environment : casopractico2

Login server
acrweucp2dev.azurecr.io

Creation date
3/16/2025, 9:22 PM GMT+1

Provisioning state
Succeeded

Pricing plan
Basic

```
az acr list --query "[?name=='acrweucp2dev']" --output table
```

Tras lanzar este comando recibimos esta salida por consola:

Here is the Markdown code for the table:

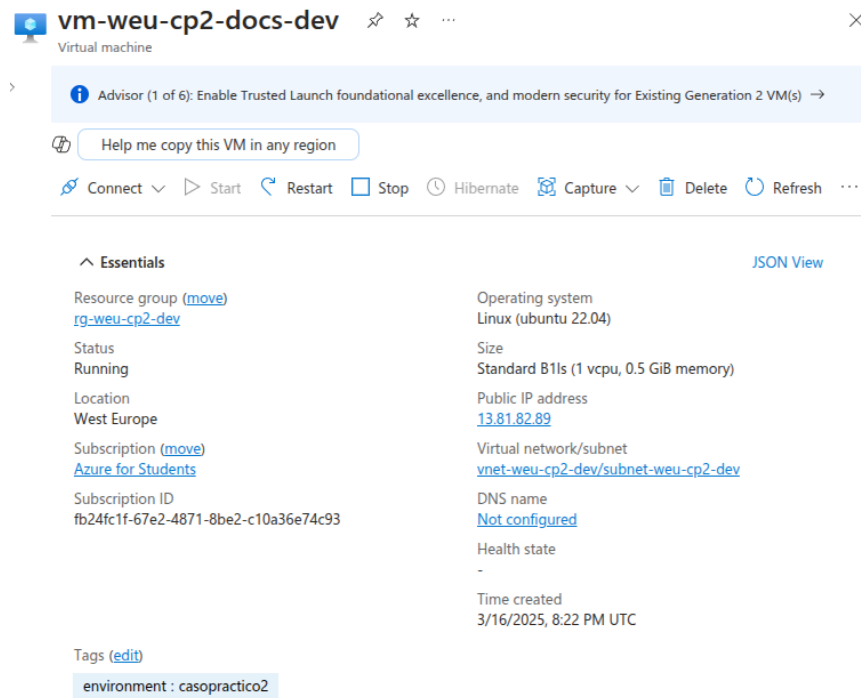
Name	Location	LoginServer	CreationDate	ProvisioningState
acrweucp2dev	westeurope	acrweucp2dev.azurecr.io	2025-03-16T20:22:34.983350+00:00	Succeeded

También podemos comprobar que se ha creado correctamente iniciando sesión en el ACR mediante el comando `az acr login --name acrweucp2dev` que devuelve la siguiente salida:

```
(.env) charlstown@Elantris:/media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/8
● unir-cp2$ az acr login --name acrweucp2dev
Login Succeeded
```

Creación de la VM

Desde el portal de Azure podemos observar como la VM se ha creado correctamente bajo los parámetros definidos en los ficheros terraform.



The screenshot shows the Azure portal interface for a virtual machine named 'vm-weu-cp2-docs-dev'. At the top, there's a header with the VM name and a 'Virtual machine' label. Below this, an advisor message is visible. A toolbar contains actions like Connect, Start, Restart, Stop, Hibernate, Capture, Delete, and Refresh. The main content area is divided into 'Essentials' and 'JSON View' sections. The 'Essentials' section lists key properties: Resource group (rg-weu-cp2-dev), Status (Running), Location (West Europe), Subscription (Azure for Students), Subscription ID (fb24fc1f-67e2-4871-8be2-c10a36e74c93), Operating system (Linux (ubuntu 22.04)), Size (Standard B1ls (1 vcpu, 0.5 GiB memory)), Public IP address (13.81.82.89), Virtual network/subnet (vnet-weu-cp2-dev/subnet-weu-cp2-dev), DNS name (Not configured), Health state (-), and Time created (3/16/2025, 8:22 PM UTC). A 'Tags' section at the bottom shows 'environment : casopractico2'.

Para comprobar que la VM está levantada podemos acceder por ssh usando la clave pública que le pasamos en el momento del despliegue con terraform y la IP pública publicada en los outputs.

La IP Pública la podemos extraer de los outputs generados de terraform

terraform output

```
(.env) charlstown@Elantris:/media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejerci
cios/CP_02/CP2_Solucion/unir-cp2/terraform$ terraform output
acr_login_server = "acrweucp2dev.azurecr.io"
acr_password = <sensitive>
acr_username = "acrweucp2dev"
vm_public_ip = "13.81.82.89"
```

Si hacemos ssh contra esa IP y con la clave pública que pasamos en el momento de creación podremos acceder a la VM.

```
ssh -i ~/.ssh/az_unir_rsa charlstown@13.81.82.89
```

```
(.env) charlstown@Elantris:/media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejercicios/CP_02/CP2_Solucion/unir-cp2$ ssh -i ~/.ssh/az_unir_rsa charlstown@13.81.82.89
The authenticity of host '13.81.82.89 (13.81.82.89)' can't be established.
ED25519 key fingerprint is SHA256:LLGh8zP0jZ2If/TTUKG2GSV5smSNQms7/WbZfSRXRI8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '13.81.82.89' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1021-azure x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/pro

System information as of Sun Mar 16 23:02:28 UTC 2025

System load:  0.0                      Processes:            103
Usage of /:   5.2% of 28.89GB          Users logged in:     0
Memory usage: 60%                     IPv4 address for eth0: 10.0.1.4
Swap usage:   0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment.
```

Creación del AKS

Desde el portal de Azure podemos observar como el servicio de Kubernetes (AKS) se ha creado correctamente bajo los parámetros definidos en los ficheros terraform.

The screenshot shows the Azure portal interface for a Kubernetes service named 'aks-weu-cp2-dev'. The service is in a 'Running' state. The interface displays various configuration details in a two-column layout:

- Resource group:** [rg-weu-cp2-dev](#)
- Power state:** Running
- Cluster operation status:** Succeeded
- Subscription:** [Azure for Students](#)
- Location:** West Europe
- Subscription ID:** fb24fc1f-67e2-4871-8be2-c10a36e74c93
- Tags:** [\(edit\)](#) environment: casopractico2
- Kubernetes version:** 1.30.9
- API server address:** aksweucp2-1fslhh1t.hcp.westeurope.azmk8s.io
- Network configuration:** [Azure CNI Overlay](#)
- Node pools:** 1 node pool
- Container registries:** [Attach a registry](#)

At the top, there are action buttons: Create, Connect, Start, Stop, Delete, Refresh, and Open in mobile. A 'JSON View' link is also present.

Podemos comprobar el estado del AKS en Azure mediante el siguiente comando:

```
az aks show --resource-group rg-weu-cp2-dev --name aks-weu-cp2-dev --output table
```


Here is the markdown code for your table:

Name	Location	ResourceGroup	KubernetesVersion	CurrentKubernetesVersion	Provisi
aks-weu-cp2-dev	westeurope	rg-weu-cp2-dev	1.30	1.30.9	Succee

Para probar desde local que podemos acceder al cluster de Kubernetes, podemos realizar los siguientes comandos.

Credenciales de acceso

```
az aks get-credentials --resource-group rg-weu-cp2-dev --name aks-weu-cp2-dev
```

Con este comando podemos ver los nodos del cluster y si el AKS está levantado, deberían aparecer con estado Ready.

```
kubectl get nodes
```

```
(.env) charlstown@Elantris:/media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejerci
cios/CP_02/CP2_Solucion/unir-cp2/terraform$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
aks-default-89741859-vmss000000    Ready    <none>   172m  v1.30.9
```

Con el siguiente comando podemos listar los pods internos del clúster (como CoreDNS, metric-server, etc.). Si están en Running, el clúster funciona correctamente.

```
kubectl get pods -n kube-system
```

```
(.env) charlstown@Elantris:/media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejerci
cios/CP_02/CP2_Solucion/unir-cp2/terraform$ kubectl get pods -n kube-system
NAME                                READY    STATUS    RESTARTS   AGE
azure-cns-thkk8                     1/1      Running   0           174m
azure-ip-masq-agent-xwnfv           1/1      Running   0           174m
cloud-node-manager-vx2rk            1/1      Running   0           174m
coredns-659fcb469c-f6hgc            1/1      Running   0           174m
coredns-659fcb469c-xr4xl            1/1      Running   0           172m
coredns-autoscaler-5955d6bbdb-h2spz 1/1      Running   0           174m
csi-azuredisk-node-wldrt            3/3      Running   0           174m
csi-azurefile-node-mmd4w            3/3      Running   0           174m
konnectivity-agent-cff7c4d4d-gxmw9  1/1      Running   0           161m
konnectivity-agent-cff7c4d4d-z2wpz  1/1      Running   0           161m
kube-proxy-4c26d                    1/1      Running   0           174m
metrics-server-7c694ff6f8-hbsb9     2/2      Running   0           172m
metrics-server-7c694ff6f8-w979l     2/2      Running   0           172m
```

2.4.2 Publicación de las imágenes

Publicación de imágenes mediante Ansible

Tras ejecutar el playbook `publish_images.yml` de Ansible con el comando:

```
ansible-playbook ansible/publish_images.yml -i an
sible/hosts.yml --extra-vars "@ansible/vars.yml" --ask-vault-pass
```

Podemos ver como se ejecuta el rol de ACR de la carpeta Ansible ejecutando las tareas sin errores.

```

PLAY [Configure VM and Push Image to ACR] *****
TASK [gathering Facts] *****
ok: [vm-weu-cp2-docs]

TASK [acr : Install Podman on the VM] *****
included: /media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejercicios/CP_02/CP2_S6
lucion/unir-cp2/ansible/roles/acr/tasks/install.yml for vm-weu-cp2-docs

TASK [acr : Install Podman] *****
ok: [vm-weu-cp2-docs]

TASK [acr : Build MkDocs image] *****
included: /media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejercicios/CP_02/CP2_S6
lucion/unir-cp2/ansible/roles/acr/tasks/build_docs.yml for vm-weu-cp2-docs

TASK [acr : Ensure repository is present on the VM] *****
ok: [vm-weu-cp2-docs]

TASK [acr : Install dependencies for MkDocs] *****
ok: [vm-weu-cp2-docs]

TASK [acr : Install required system dependencies for WeasyPrint] *****
ok: [vm-weu-cp2-docs]

TASK [acr : Install project dependencies] *****
ok: [vm-weu-cp2-docs]

TASK [acr : Build MkDocs static site] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Build Podman image on the VM] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Login into ACR from the VM] *****
included: /media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejercicios/CP_02/CP2_S6
lucion/unir-cp2/ansible/roles/acr/tasks/login_acr.yml for vm-weu-cp2-docs

TASK [acr : Log in to ACR from the VM] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Push mkdocs image to ACR from the VM] *****
included: /media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejercicios/CP_02/CP2_S6
lucion/unir-cp2/ansible/roles/acr/tasks/push_mkdocs.yml for vm-weu-cp2-docs

TASK [acr : Tag MkDocs image for ACR] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Push MkDocs image to ACR from the VM] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Push stackedit image to ACR from localhost] *****
included: /media/MyData/00_WIP/00_UnirDevOps/DevOpsAndCloud/Ejercicios/CP_02/CP2_S6
lucion/unir-cp2/ansible/roles/acr/tasks/push_stackedit.yml for vm-weu-cp2-docs

TASK [acr : Pull StackEdit image from Docker Hub] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Tag StackEdit image for ACR] *****
changed: [vm-weu-cp2-docs]

TASK [acr : Push StackEdit image to ACR] *****
changed: [vm-weu-cp2-docs]

PLAY RECAP *****
vm-weu-cp2-docs : ok=19  changed=8  unreachable=0  failed=0  skip
ped=0  rescued=0  ignored=0

```

Publicación mediante Github Actions (fuera de alcance)

La publicación de la imagen se automatiza mediante el workflow [Publish mkdocs image to ACR](#) de GitHub Actions, que envía la imagen al Azure Container Registry (ACR). Para ello, se deben proporcionar las credenciales adecuadas y validar la ejecución del proceso.

1. Rellenar los datos del formulario del workflow con username y pwd del ACR desplegado en Azure.

Visualizar usuario y contraseña del ACR ▼

Siempre puedes ejecutar este comando para recuperar el usuario y la contraseña del ACR.

```
az acr credential show --name acrweucp2dev --query "[username, passwords[0].value]" -o tsv
```

2. Ejecutar workflow y validar la correcta ejecución del job

Validación de imágenes publicadas

Tras publicar las imágenes por Ansible o por Github Action podremos ver los repositorios en `Services/Repositories` en el recurso del ACR.

También podemos ejecutar el siguiente comando desde local para listar las imágenes del ACR.

```
az acr repository list --name acrweucp2dev --output table
```

```
gios/CP_02/CP2_Solucion/unir-cp2$ az acr repository list --name acrweucp2dev --outp
ut table
Result
-----
docs-nginx
stackedit
```

IMAGEN docs-nginx

[Home](#) > [Resource groups](#) > [rg-weu-cp2-dev](#) > [acrweucp2dev | Repositories](#) >

docs-nginx

Repository

Refresh

Start artifact streaming

Manage deleted artifacts

Delete repository

^ Essentials

Repository

docs-nginx

Tag count

1

Last updated date

3/17/2025, 9:30 PM GMT+1

Manifest count

3

Search to filter tags ...

Tags ↑↓	Digest ↑↓	Last modified	
1.0.0	sha256:22d575b0532211dbbe6a49983...	3/17/2025, 9:30 PM GMT+1	...

IMAGEN stackedit

[Home](#) > [Resource groups](#) > [rg-weu-cp2-dev](#) > [acrweucp2dev | Repositories](#) >

stackedit

Repository

Refresh

Start artifact streaming

Manage deleted artifacts

Delete repository

^ Essentials

Repository

stackedit

Tag count

1

Last updated date

3/17/2025, 9:41 PM GMT+1

Manifest count

1

Search to filter tags ...

Tags ↑↓	Digest ↑↓	Last modified	
1.0.0	sha256:15bd6b0678be86c2af8c3fdc...	3/17/2025, 9:41 PM GMT+1	...

2.4.3 Despliegue en la VM

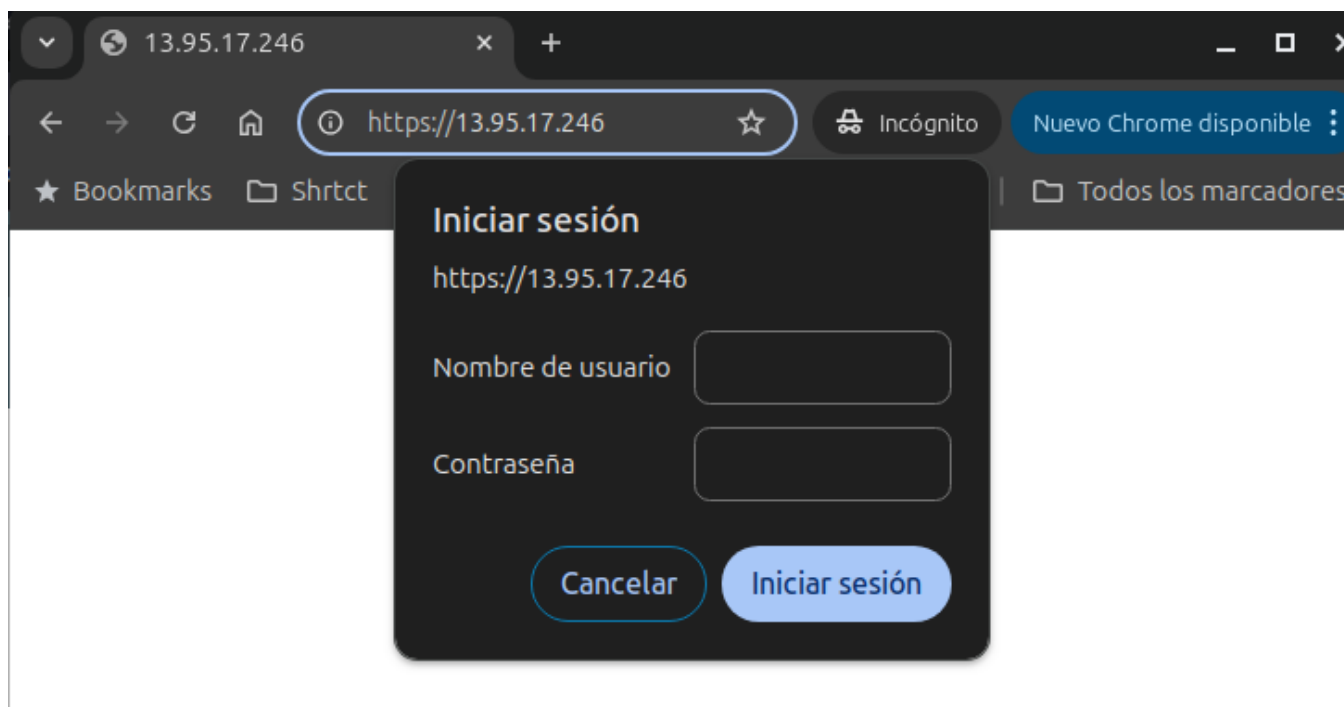
Para desplegar la imagen de mkdocs-nginix en un contenedor sobre la máquina virtual ejecutamos el siguiente playbook que contiene el rol `vm`.

```
ansible-playbook ansible/playbook.yml -i ansible/hosts.yml --extra-vars "@ansible/vars.yml" --ask-vault-pass
```

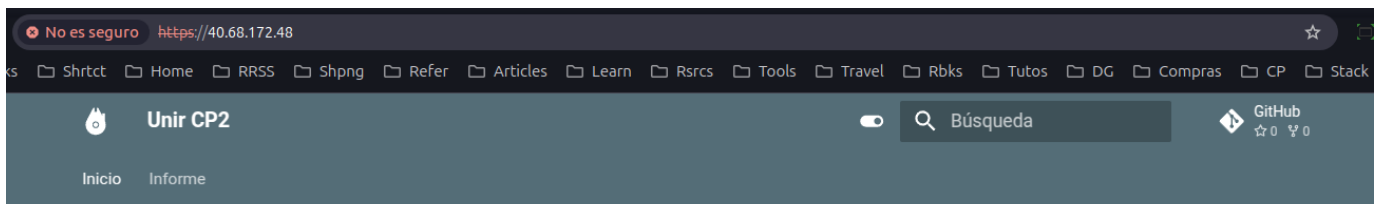
Si todo ha ido bien se puede comprobar que el sitio se muestra a través de internet en la ip pública de la VM. Ejecutando el comando:

```
curl -k -u charlstown:*** https://{VM_IP}:443
```

También puede visualizarse en el browser en la dirección `https://ip-publica/`.



Si introducimos el usuario y la contraseña tendremos acceso a la web de la imagen de `` levantada en la IP pública de la VM.



Inicio

Inicio

Tabla de contenidos

Introducción

Objetivos

Asignatura: Devops & Cloud

Alumno: Carlos Grande Núñez

Fecha: 23/03/25

Ir a la versión online

2.4.4 Despliegue en el AKS

A continuación se muestran las evidencias de que el despliegue del contenedor en AKS se ha realizado correctamente tras ejecutar el siguiente comando, el cual aplica el rol `aks` :

```
ansible-playbook playbook_aks.yml -i hosts.yml --ask-vault-pass
```

Podemos comprobar que el servicio se ha creado correctamente con tipo `LoadBalancer` y que la IP pública ha sido asignada:

```
kubectl get svc stackedit-service -n cp2
```

Resultado:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
stackedit-service	LoadBalancer	10.0.237.2	74.178.201.19	80:32417/TCP	40m

Esto indica que la aplicación desplegada está accesible públicamente a través de la IP `74.178.201.19` .

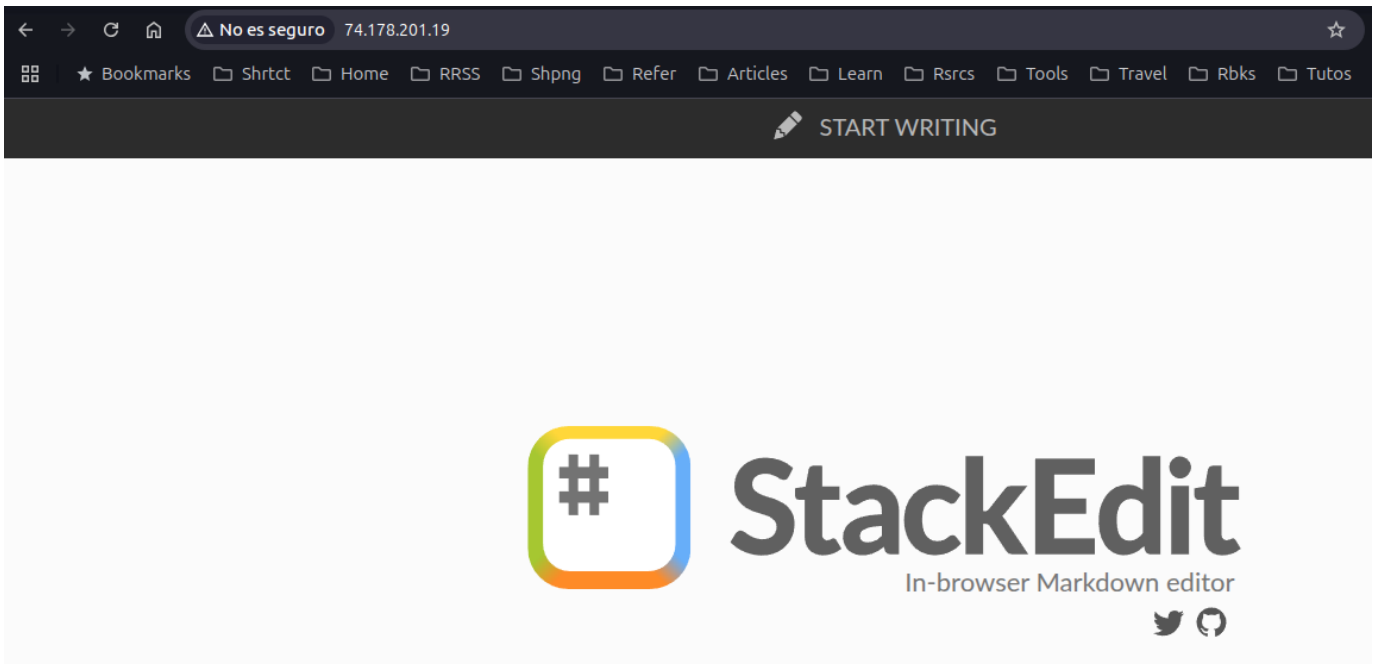
Podemos validar que el contenedor se ha desplegado correctamente y que está sirviendo en el puerto `80` :

```
kubectl logs -n cp2 -l app=stackedit
```

Resultado:

```
HTTP server started: http://localhost:8080
```

Por tanto, accediendo desde el navegador a `http://74.178.201.19` se podrá visualizar la interfaz web de StackEdit.



Esto confirma que el despliegue en AKS se ha realizado con éxito, con el contenedor sirviendo desde la imagen publicada en el ACR.

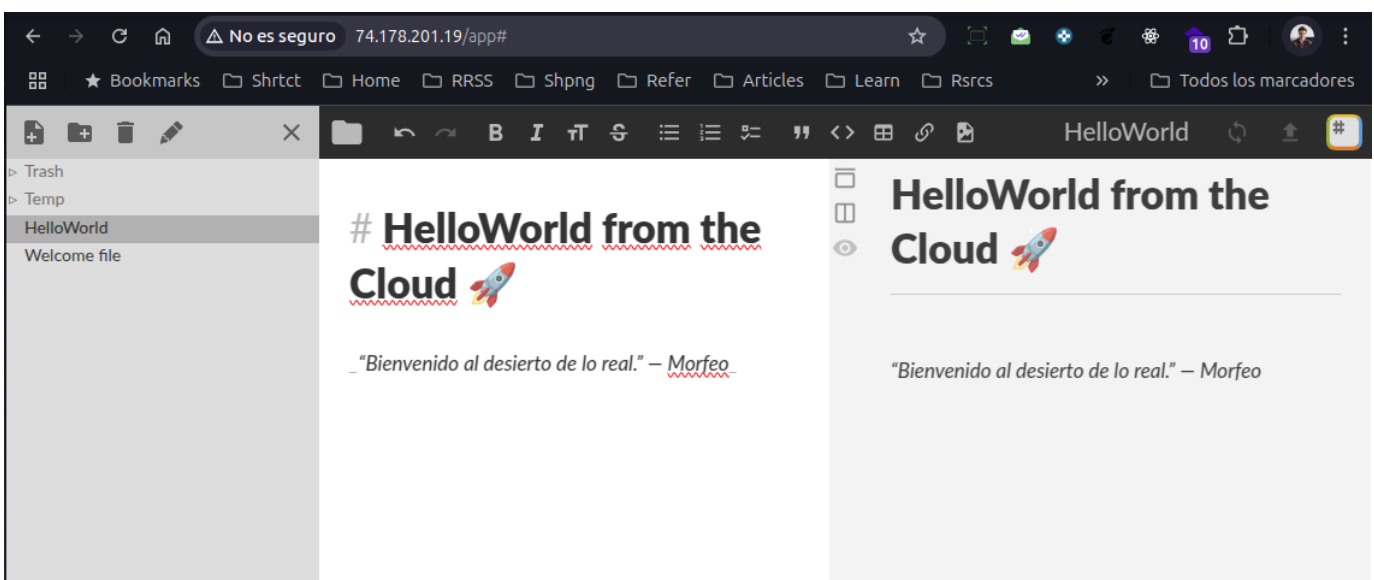
Comprobación de persistencia

Para validar que el contenedor desplegado en AKS cuenta con almacenamiento persistente, se realiza la siguiente prueba:

1. **Crear una nota** desde la interfaz web de StackEdit accediendo a <http://74.178.201.19>.

Se añade una nueva nota con el siguiente contenido:

```
# HelloWorld from the Cloud 🚀
_"Bienvenido al desierto de lo real." – Morfeo_
```



2. **Eliminar el pod** para forzar su recreación automática por Kubernetes:

```
kubectl delete pod -n cp2 -l app=stackedit
```

Desde Lens podemos ver como se recrea el contenedor.

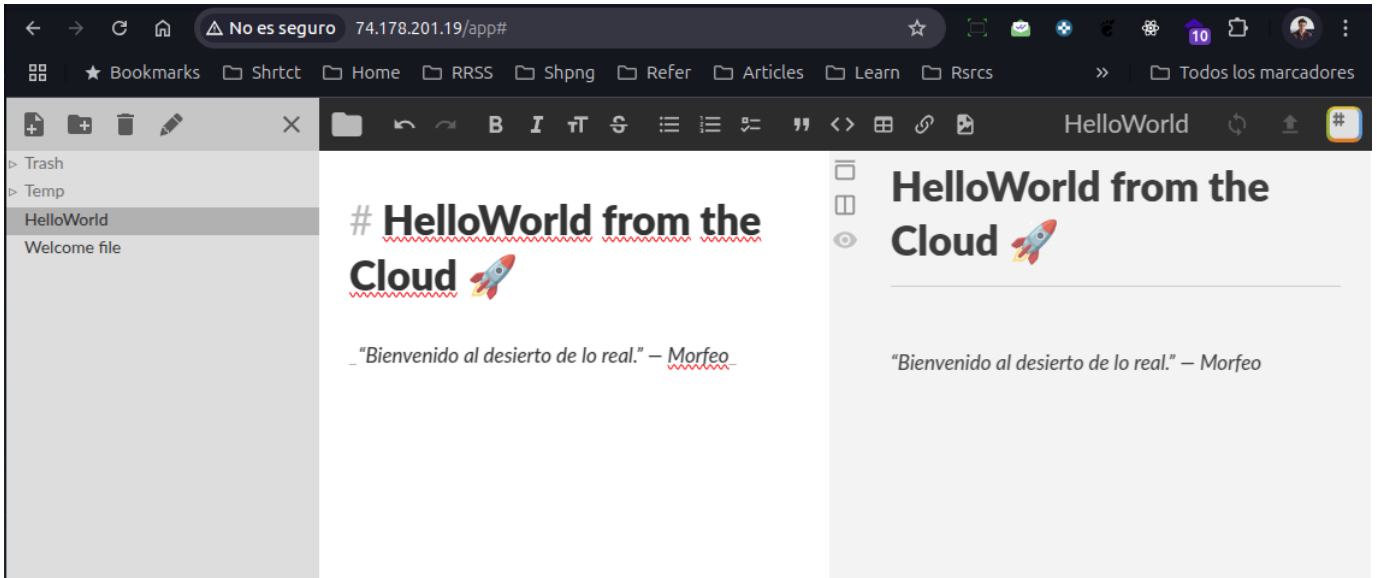
Pods - aks-weu-cp2-dev x

cp2 Aa * Search Pods... 2 items

Space	Cont...	CPU	Memor	Restart	Controlled B	Node	QoS	Age	Status	
	■	0.048	25.2MiB	0	ReplicaSet	aks-defaul	BestEffort	67s	Terminatir	⋮
	■	0.000	0	0	ReplicaSet	aks-defaul	BestEffort	26s	Running	⋮

3. **Actualizar la página web** tras unos segundos.

La nota debería seguir estando presente, lo que confirma que el volumen persistente está funcionando correctamente.



2.5 Licencia

Para este ejercicio se ha utilizado la licencia **MIT** (*Massachusetts Institute of Technology License*), una de las licencias más utilizadas en proyectos de código abierto. La licencia completa puede consultarse en el siguiente enlace al repositorio del proyecto:

 [Ver archivo LICENSE](#)

2.5.1 Justificación de la elección de la licencia

Se ha optado por la licencia MIT debido a su facilidad de implementación y su compatibilidad con otros modelos de licencia. MIT permite que cualquier persona utilice, copie, modifique y distribuya el código sin restricciones, siempre y cuando se incluya la atribución original en el código fuente ([MIT License, s.f.](#)).

La licencia MIT es ampliamente utilizada debido a su simplicidad y flexibilidad. Permite la reutilización del código con mínimas restricciones, lo que la hace ideal para proyectos de código abierto que buscan una adopción amplia. Como señala [Mahajan \(n.d.\)](#), la licencia MIT es una de las más permisivas, ya que permite modificaciones y redistribución con pocas limitaciones, a diferencia de otras como la Apache 2.0, que incluye cláusulas adicionales sobre patentes y atribución.

2.5.2 Permisos y restricciones de uso

La licencia MIT permite lo siguiente:

- Uso personal y comercial sin restricciones.
- Modificación y distribución del código.
- Incorporación en proyectos de código abierto o cerrado.

Sin embargo, impone las siguientes condiciones:

- Debe mantenerse el aviso de copyright y la declaración de licencia en todas las copias o partes sustanciales del software.
- No ofrece garantía ni responsabilidad sobre el uso del software ("*as is*", sin garantía de funcionalidad o idoneidad).

2.5.3 Texto completo de la licencia

MIT License

Copyright (c) 2025 Carlos Grande

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.6 Referencias

Ansible. (s.f.-a). *Best practices for structuring Ansible playbooks*. Ansible Documentation. Recuperado de https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html

Ansible. (s.f.-b). *Ansible configuration settings*. Ansible Documentation. Recuperado de https://docs.ansible.com/ansible/latest/reference_appendices/config.html

Mahajan, J. (n.d.). *Simple guide to open source licenses*. Medium. Retrieved from <https://medium.com/@jayeshmahajan/simple-guide-to-open-source-licenses-ec5b3d29ae80>

MIT License. (s.f.). *MIT License Terms*. Recuperado de <https://choosealicense.com/licenses/mit/>

Stivenson, T. (2023). *Mejores prácticas en Terraform*. Medium. Recuperado de <https://medium.com/@tonystivenson1995/mejores-practicas-en-terraform-107533470831>

2.6.1 Herramientas usadas

JGraph Ltd. (s.f.). *draw.io*. Recuperado de <https://www.drawio.com/>

MkDocs. (s.f.). *MkDocs Documentation*. Recuperado de <https://www.mkdocs.org>

Squidfunk. (s.f.). *Material for MkDocs*. Recuperado de <https://squidfunk.github.io/mkdocs-material/>

StackEdit. (s.f.). *StackEdit - In-browser Markdown editor*. GitHub. <https://github.com/benweet/stackedit>

WithPDF. (s.f.). *WithPDF Plugin for MkDocs*. Recuperado de <https://github.com/orzih/mkdocs-with-pdf>