

# IES GONZALO NAZARENO

ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

## Implementación de Terraform en Azure mediante "Azure DevOps Pipelines"

2023

**Alfonso Roldán Amador**



**Terraform**



**Microsoft  
Azure**



**Azure Pipelines**

# Índice

1. Introducción al proyecto.....	1
2. Terraform.....	1
2.1 Introducción a Terraform.....	1
2.2 Tipos de Infraestructuras en Terraform.....	1
2.3 Diferencias entre IaaS, PaaS y SaaS.....	2
2.4 ¿Qué son los Providers?.....	3
2.5 Providers de Terraform para la infraestructura de Azure.....	4
2.6 Ventajas de Terraform con Azure.....	4
2.6.1 Herramienta IaC común.....	4
2.6.2 Automatización de la administración de la infraestructura.....	5
2.6.3 Comprensión de los cambios de infraestructura antes de que se apliquen.....	5
2.7 Sintaxis de Terraform.....	6
2.8 Gestión del estado en Terraform.....	8
2.9 Ciclo de vida de la infraestructura.....	9
2.10 Automatización y despliegue continuo.....	11
2.11 Mejores prácticas y consideraciones de seguridad.....	12
3. Azure.....	14
3.1 Introducción a Azure.....	14
3.2 Principales características de Azure.....	14
3.3 Servicios de Azure.....	16
3.4 Aprovisionamiento de recursos en Azure.....	18
3.5 Modelos de implementación.....	19
3.6 Integraciones.....	20
3.7 Facturación y precios.....	21
3.8 Herramientas y servicios de gestión.....	22
3.9 Seguridad y cumplimiento.....	23
4. Azure Devops.....	24
4.1 ¿Qué es Azure Devops?.....	24
4.2 Herramientas y servicios de Azure DevOps.....	24
4.3 Características y funcionalidades para el desarrollo.....	25
4.4 Trabajos paralelos.....	26

4.5 Trabajos paralelos autohospedados frente a hospedados por Microsoft.....	27
4.5.1 Hospedado por Microsoft.....	27
4.5.2 Autohospedado.....	28
5. Casos Prácticos (Terraform y Azure).....	29
5.1 Configuración del entorno de desarrollo con Terraform y Azure.....	29
5.1.1 Máquina Windows.....	29
5.1.2 Máquina Linux.....	33
5.2 Ejemplo de despliegue de infraestructura.....	36
6. Prueba práctica del proyecto.....	40
6.1 Objetivo propuesto.....	40
6.2 Prerrequisitos.....	40
6.3 Pasos a seguir para la implementación.....	41
6.3.1 Creación de un Backend en Azure.....	41
6.3.2 Creación de un proyecto en Azure DevOps.....	44
6.3.3 Añadir una conexión de servicio con Azure Cloud.....	46
6.3.4 Preconfiguración, creación de la pipeline (YAML).....	47
6.3.5 Ejecución de la pipeline.....	50
7. Conclusión.....	53

# 1. Introducción al proyecto

En este proyecto se abordará el uso de Terraform conjuntamente con Azure y Azure DevOps. Se explicarán los conceptos teóricos fundamentales de cada una de estas herramientas y se realizarán pruebas prácticas para demostrar cómo se pueden utilizar juntas para crear y administrar recursos de infraestructura en la nube de Microsoft de forma automatizada.

También se explicará cómo se pueden definir recursos de Azure mediante código Terraform y cómo se pueden utilizar pipelines de Azure DevOps para implementar y administrar estos recursos de forma automatizada. Se realizarán pruebas prácticas para demostrar cómo se pueden crear y administrar recursos de Azure utilizando Terraform y Azure DevOps, lo que permitirá a los equipos de desarrollo y operaciones mejorar la eficiencia, la calidad y la seguridad de sus procesos.

## 2. Terraform

### 2.1 Introducción a Terraform

**Terraform** es un software de **iaC** (Infraestructura como código) de código abierto para aprovisionar y administrar la infraestructura en la nube, desarrollada por **Hashicorp**.

Esta herramienta de codificación utiliza un lenguaje de configuración de alto nivel denominado **HLC** (HashiCorp Configuration Language). La estructura que utiliza puede dividirse en uno o varios archivos de configuración. Terraform es **declarativo**, por lo que en la configuración indicamos el estado final que queremos para la plataforma, no los pasos para llegar a ese estado.

Terraform permite administrar cualquier infraestructura (Nubes públicas, nubes privadas, servicios SaaS) mediante **proveedores de Terraform**.

### 2.2 Tipos de Infraestructuras en Terraform

Terraform puede utilizarse para administrar diferentes tipos de infraestructuras, entre ellas:

1. **Infraestructura en la nube pública:** Terraform es compatible con proveedores de nube pública como AWS, Google Cloud, Microsoft Azure, DigitalOcean, Alibaba Cloud, entre otros. Con Terraform, podemos definir, configurar y gestionar recursos de infraestructura en la nube como instancias de máquinas virtuales, redes virtuales, bases de datos, balanceadores de carga, entre otros.
2. **Infraestructura en nube privada:** Terraform también puede utilizarse para administrar infraestructuras de nube privada como OpenStack y VMware vSphere.
3. **Infraestructura de proveedor de servicios:** Terraform admite proveedores de servicios como GitHub, GitLab, Docker y Kubernetes, lo que permite la gestión de recursos como repositorios de código, imágenes de contenedores y clústeres de Kubernetes.
4. **Infraestructura local:** Terraform también se puede utilizar para gestionar recursos de infraestructura en una máquina local, como la configuración del sistema operativo o la instalación de software.

En resumen, Terraform es un gestor de infraestructura multiplataforma que te permite definir y administrar diferentes tipos de recursos de infraestructura, ya sea en la nube pública, nube privada, servicios en línea o localmente en una máquina.

## 2.3 Diferencias entre IaaS, PaaS y SaaS

**SaaS**, **PaaS** e **IaaS** son tres modelos de servicio diferentes dentro del campo de la computación en la nube, que ofrecen diferentes niveles de abstracción y responsabilidad a los usuarios. Las principales diferencias entre SaaS, PaaS e IaaS son las siguientes:

- **Software como Servicio (SaaS):** SaaS es un modelo de servicio en la nube en el cual los usuarios acceden a aplicaciones de software a través de Internet, y el software es gestionado y mantenido por el proveedor de servicios en la nube. Los usuarios no tienen que preocuparse por la infraestructura subyacente, ya que todo el software, la gestión de datos y el mantenimiento son responsabilidad del proveedor. Ejemplos de SaaS son aplicaciones de correo electrónico en la nube como **Gmail** o plataformas de gestión de relaciones con los clientes (CRM) como **Salesforce**.

- **Plataforma como Servicio (PaaS):** PaaS es un modelo de servicio en la nube que proporciona a los usuarios una plataforma completa de desarrollo y despliegue de aplicaciones, sin tener que preocuparse por la infraestructura subyacente. Los usuarios pueden desarrollar, ejecutar y gestionar sus propias aplicaciones en la nube utilizando herramientas y servicios proporcionados por el proveedor de PaaS. Esto incluye herramientas de desarrollo, bases de datos, entornos de ejecución y servicios de escalado automático. Ejemplos de PaaS son **Azure App Service** de Microsoft o **Google App Engine**.
- **Infraestructura como Servicio (IaaS):** IaaS es un modelo de servicio en la nube que ofrece a los usuarios recursos de infraestructura virtual, como máquinas virtuales, redes y almacenamiento, a través de Internet. Los usuarios tienen un mayor control sobre la infraestructura, ya que pueden configurar y gestionar los recursos según sus necesidades. Sin embargo, los usuarios también son responsables de gestionar el sistema operativo, las actualizaciones de software y otras tareas de mantenimiento. Ejemplos de IaaS son **Amazon Web Services (AWS)**, **Microsoft Azure** o **Google Cloud Platform**.

## 2.4 ¿Qué son los Providers?

En **Terraform**, un **provider** es un complemento de software que se utiliza para interactuar con un servicio o tecnología específica. Los **providers** en Terraform son responsables de traducir los recursos definidos en el código de Terraform a recursos reales en la nube, lo que permite a los usuarios crear, modificar y eliminar recursos de infraestructura en la nube de manera programática.

Los providers pueden ser de diferentes tipos, desde providers de nube como **AWS**, **Azure** o **Google Cloud**, hasta providers de servicios de **DNS** o providers de **contenedores (Kubernetes, Grafana, Chef, MySQL, RabbitMQ, Consul, Github, entre otros)**.

Cada provider cuenta con su propio conjunto de elementos, con sus propiedades y posibles valores.

Una de las grandes ventajas de Terraform es que al contar con una gran variedad de providers disponibles, permite definir y administrar la infraestructura de forma declarativa y unificada en múltiples entornos y plataformas de nube (Infraestructuras híbridas).

## 2.5 Providers de Terraform para la infraestructura de Azure

- **AzureRM:** Administra recursos y funcionalidades estables de **Azure**, como máquinas virtuales, cuentas de almacenamiento e interfaces de red.
- **AzureAD:** Administra recursos de **Azure Active Directory**, como grupos, usuarios, entidades de servicio y aplicaciones.
- **AzureDevops:** Administra recursos de **Azure DevOps**, como agentes, repositorios, proyectos, canalizaciones y consultas.
- **AzAPI:** Permite administrar directamente los recursos y funcionalidades de Azure a través de sus API de **Azure Resource Manager**. Esta herramienta es complementaria al proveedor de AzureRM y te permite gestionar recursos de Azure que no están disponibles públicamente.
- **Azure Stack:** Administra recursos de **Azure Stack**, como máquinas virtuales, DNS, red virtual y almacenamiento.

## 2.6 Ventajas de Terraform con Azure

### 2.6.1 Herramienta IaC común

Los providers de Terraform para Azure permiten gestionar toda la infraestructura de Azure utilizando la misma sintaxis declarativa y herramientas. Con estos providers, podemos:

1. **Aprovisionar funcionalidades básicas** de la plataforma, como grupos de administración, directivas, usuarios, grupos y directivas.
2. **Aprovisionar proyectos de Azure DevOps y pipelines** para automatizar las implementaciones de aplicaciones e infraestructura de manera normalizada.
3. **Aprovisionar los recursos de Azure** necesarios para las aplicaciones y cargas de trabajo deseadas.

## 2.6.2 Automatización de la administración de la infraestructura

La sintaxis del archivo de configuración basada en plantillas de Terraform proporciona una manera repetible y predecible de configurar recursos de Azure. La automatización de la infraestructura con Terraform ofrece varias ventajas, como:

1. **Reducción del potencial de errores humanos** al implementar y administrar la infraestructura, ya que los procesos se definen y automatizan mediante código, lo que disminuye la probabilidad de errores manuales.
2. **Capacidad de implementar la misma plantilla de Terraform** varias veces para crear entornos de desarrollo, prueba y producción idénticos, lo que garantiza la consistencia y la reproducibilidad en diferentes etapas del ciclo de vida de las aplicaciones.
3. **Reducción del costo de entornos de desarrollo y prueba** al crearlos a petición mediante Terraform, lo que permite aprovisionar y desaproveccionar recursos de Azure según las necesidades específicas del equipo de desarrollo y prueba, evitando gastos innecesarios en recursos infrutilizados.

En resumen, la **automatización de la infraestructura con Terraform** basado en plantillas de configuración de archivos ofrece ventajas significativas, como la reducción de errores humanos, la capacidad de crear entornos idénticos y la optimización del costo de desarrollo y prueba en Azure.

## 2.6.3 Comprensión de los cambios de infraestructura antes de que se apliquen

A medida que la topología de los recursos en la infraestructura se vuelve más compleja, comprender el significado y el efecto de los cambios puede resultar desafiante. Sin embargo, la interfaz de línea de comandos (**CLI**) de Terraform brinda a los usuarios la capacidad de validar y obtener una vista previa de los cambios propuestos antes de aplicarlos ([Terraform plan](#)). Esta funcionalidad ofrece varias ventajas, como:

1. **Facilita la colaboración efectiva entre los miembros del equipo**, ya que les permite comprender los cambios propuestos y su impacto antes de su



implementación. Esto ayuda a asegurar que todos estén alineados y trabajen de manera coherente en los cambios de infraestructura.

2. **Permite detectar y corregir cambios no deseados o errores al principio del proceso de desarrollo.** La vista previa de los cambios de infraestructura permite identificar posibles problemas antes de que se apliquen, lo que ayuda a mitigar riesgos y a mantener la integridad y seguridad del entorno de infraestructura.

En resumen, la **capacidad de validar y obtener una vista previa de los cambios de infraestructura** de forma segura a través de la CLI de Terraform brinda ventajas importantes, como la colaboración efectiva y la detección temprana de errores o cambios no deseados.

## 2.7 Sintaxis de Terraform

La sintaxis de Terraform se basa en un lenguaje específico de dominio llamado **HCL** (HashiCorp Configuration Language), que se utiliza para describir la configuración de la infraestructura como código (**laC**).

La sintaxis de Terraform **puede variar** ligeramente **dependiendo del proveedor** de infraestructura en la nube con el que estés trabajando. Esto se debe a que cada proveedor de la nube tiene su propio conjunto de recursos y configuraciones específicas que se pueden crear y gestionar mediante Terraform. En este caso, mostraremos la sintaxis que utilizan algunos de los elementos con el proveedor de **Azure Cloud**.

1. **Bloques:** Los bloques son la unidad básica de configuración en Terraform y se definen con llaves **{}**. Los bloques se utilizan para agrupar y configurar recursos o módulos en Terraform.

En Terraform se utilizan dos parámetros entre comillas a la hora de definir un recurso para especificar la **"tipo"** y el **"nombre"** del recurso. Por ejemplo:

```
resource "azurearm_virtual_network" "ejemplo" {  
  
# Configuración del recurso de red virtual de Azure  
...  
}
```

- Argumentos:** Los argumentos se definen dentro de los bloques y se utilizan para configurar los recursos. Se especifican como pares de **clave = valor** y se pueden utilizar para definir las propiedades y configuraciones del recurso. Por ejemplo:

```
resource "azurerem_virtual_network" "ejemplo" {  
  
  name          = "my-virtual-network"  
  location      = "West Europe"  
  address_space = ["10.0.0.0/16"]  
  ...  
}
```

- Variables:** Las variables se utilizan para parametrizar la configuración en Terraform y se definen en un bloque **variable**. Las variables se utilizan para definir valores reutilizables que pueden ser pasados como parámetros a módulos o recursos. Por ejemplo:

```
variable "vm_size" {  
  
  type    = string  
  default = "Standard_DS2_v2"  
}
```

En este caso, el **size** hace referencia a la etiqueta predefinida por Azure que engloba el conjunto de características del que dispondrá la máquina virtual (Similar a un **sabor** o **flavor** en OpenStack).

Como vemos en el ejemplo anterior, dentro de la variable “**vm\_size**” definimos varios campos, el campo “**type**” indicará que es una variable de tipo cadena. Y en el campo “**default**” seleccionamos la etiqueta del **size** que vamos a asignar a dicha variable.

- Expresiones:** Terraform permite el uso de expresiones para realizar cálculos y manipulaciones en la configuración. Las expresiones se utilizan para definir los valores de los argumentos o las variables. Por ejemplo:

```
resource "azurerms_virtual_machine" "ejemplo" {  
  
    ...  
    vm_size = "${var.vm_size}" # Uso de una variable en la configuración del recurso  
    ...  
}
```

5. **Referencias:** Las referencias se utilizan para hacer referencia a otros recursos o variables en la configuración de Terraform. Se utilizan para establecer dependencias entre recursos o para obtener valores de otros recursos o variables.

Por ejemplo:

```
resource "azurerms_virtual_network" "ejemplo" {  
  
    ...  
}  
  
resource "azurerms_subnet" "ejemplo" {  
    virtual_network_name = azurerms_virtual_network.example.name # Referencia a otro recurso  
    ...  
}
```

Estos son algunos de los elementos básicos de la sintaxis de Terraform en HCL.

## 2.8 Gestión del estado en Terraform

El estado en Terraform se refiere a la representación actual del estado de los recursos de infraestructura que se han creado o gestionado con Terraform. Este estado es almacenado en un **archivo local** o en un **backend remoto**.

Terraform utiliza este estado para **asignar recursos del mundo real** a su configuración, **realizar un seguimiento** de los metadatos y **mejorar el rendimiento** de grandes infraestructuras. Este estado se almacena por defecto en un archivo local llamado **"terraform.tfstate"**.

Cuando se ejecuta un **plan** o una **aplicación** con Terraform, la herramienta compara el estado actual con la configuración deseada y determina qué acciones deben tomarse para que la infraestructura alcance el estado deseado.

La gestión del estado en Terraform ofrece varias ventajas, como:

1. **Control de versiones:** El estado puede almacenarse en un sistema de control de versiones (como **Git**) para llevar un registro de los cambios realizados en la infraestructura a lo largo del tiempo y facilitar la colaboración en equipos.
2. **Persistencia de la información:** Terraform guarda el estado en un archivo o en un backend remoto, lo que permite mantener un registro de la configuración de la infraestructura incluso después de que los recursos hayan sido creados.
3. **Planificación precisa:** Terraform utiliza el estado para calcular los cambios necesarios para alcanzar el estado deseado, lo que permite visualizar y planificar los cambios antes de aplicarlos.
4. **Seguridad:** El estado puede ser cifrado o almacenado de forma segura en un backend remoto, lo que ayuda a proteger la información sensible sobre la infraestructura.

## 2.9 Ciclo de vida de la infraestructura

El ciclo de vida de la infraestructura en Terraform consta de varios pasos, que incluyen la planificación, aplicación, modificación y destrucción de la infraestructura, así como la gestión de cambios en la configuración de Terraform. A continuación se describen cada uno de estos pasos:

1. **Planificación:** El primer paso en el ciclo de vida de la infraestructura en Terraform es la planificación. Durante este paso, se define la configuración de la infraestructura utilizando el lenguaje de configuración de Terraform (**HCL**) en archivos de configuración. Estos archivos describen los recursos y proveedores que se deben crear en la nube o en otro entorno de infraestructura. Una vez que la configuración está definida, se ejecuta el comando "**terraform init**" para inicializar el directorio de trabajo de Terraform. Luego, se ejecuta el comando "**terraform**

- plan**" para crear un plan de ejecución que muestra los cambios que se realizarán en la infraestructura en función de la configuración definida.
- 2. Aplicación:** Una vez que se ha creado un plan de ejecución y se ha revisado y aprobado, se procede a la aplicación de la infraestructura. Esto se realiza ejecutando el comando "**terraform apply**", que implementa los cambios en la infraestructura según el plan creado en el paso anterior. Durante la aplicación, Terraform crea, modifica o elimina los recursos y proveedores definidos en la configuración, y los configura con los valores especificados en los archivos de configuración.
  - 3. Modificación:** A medida que los requisitos de infraestructura cambian con el tiempo, es posible que sea necesario modificar la configuración de Terraform. Para ello, se pueden realizar cambios en los archivos de configuración que describen los recursos y proveedores. Después de hacer los cambios, se ejecuta el comando "**terraform plan**" nuevamente para generar un nuevo plan de ejecución que muestre los cambios propuestos en la infraestructura. Luego, se revisa y aprueba el plan, y se ejecuta el comando "**terraform apply**" para aplicar los cambios.
  - 4. Destrucción:** Si ya no es necesario un recurso o un conjunto de recursos, se pueden eliminar de la infraestructura utilizando Terraform. Para ello, se ejecuta el comando "**terraform destroy**", que identifica los recursos que se deben eliminar según la configuración definida y los elimina de la infraestructura. Es importante tener cuidado al ejecutar este comando, ya que puede resultar en la pérdida irreversible de datos y recursos.
  - 5. Gestión de cambios:** La gestión de cambios en la configuración de Terraform implica la revisión y aprobación de los cambios propuestos en la infraestructura. Para ello, Terraform proporciona herramientas como "**terraform plan**" y "**terraform show**" que permiten visualizar y revisar los cambios propuestos antes de aplicarlos. Además, Terraform integra con sistemas de control de versiones como **Git**, lo que facilita la colaboración en equipo y la revisión de los cambios realizados en la configuración.

## 2.10 Automatización y despliegue continuo

Podemos utilizar Terraform en un flujo de trabajo de automatización y despliegue continuo con herramientas de automatización y orquestación como **Jenkins**, **GitLab CI/CD** y **Azure DevOps**. Para ello, debemos de considerar los siguientes puntos:

1. **Configuración inicial:** Configuramos y establecemos las herramientas de automatización y orquestación de nuestra elección, como **Jenkins**, **GitLab CI/CD** o **Azure DevOps**, de acuerdo con nuestras necesidades y requisitos específicos. Debemos asegurarnos de que estén correctamente integradas con nuestro sistema de control de versiones, como **Git**.
2. **Definición de la infraestructura con Terraform:** Creamos y definimos la configuración de nuestra infraestructura utilizando Terraform en archivos de configuración en un **repositorio de control de versiones**. Organizamos nuestros archivos de configuración en módulos reutilizables para una gestión más eficiente y escalable.
3. **Automatización del flujo de trabajo:** Creamos un flujo de trabajo automatizado utilizando la herramienta de automatización seleccionada. Por ejemplo, en **Jenkins** creamos un job que se ejecute automáticamente cuando se realicen cambios en el repositorio de configuración de Terraform, o en **GitLab CI/CD** configuramos un pipeline que se active en cada push o merge request. En **Azure DevOps**, creamos un pipeline que se ejecute en respuesta a eventos del repositorio de Terraform.
4. **Ejecución de Terraform:** Dentro del flujo de trabajo automatizado, incluimos la ejecución de comandos de Terraform, como "**terraform init**", "**terraform plan**", "**terraform apply**" o "**terraform destroy**", dependiendo del paso del ciclo de vida de la infraestructura que deseamos realizar. Nos aseguramos de configurar los argumentos y variables necesarios en el flujo de trabajo automatizado o en variables de entorno para una ejecución correcta.
5. **Validación y aprobación:** Antes de aplicar los cambios en la infraestructura, incluimos pasos de validación y aprobación en el flujo de trabajo automatizado. Por ejemplo, realizamos **pruebas automatizadas** en la infraestructura creada por

Terraform para asegurarnos de que cumple con los requisitos de calidad y seguridad. Además, incluimos **mecanismos de aprobación**, como revisión de código o aprobación manual, antes de aplicar los cambios en la infraestructura.

6. **Registro y notificaciones:** Registramos y notificamos el resultado de la ejecución del flujo de trabajo automatizado. Por ejemplo, registramos la salida de los comandos de Terraform y los resultados de las pruebas automatizadas en un sistema de registro centralizado. Además, enviamos notificaciones por correo electrónico, mensajes de chat o integraciones con herramientas de gestión de incidentes para mantener a los equipos informados sobre el estado de los despliegues.
7. **Gestión de cambios y versionado:** Utilizamos las capacidades de control de versiones de Terraform para gestionar los cambios en la configuración de Terraform y llevar un registro del historial de versiones. Utilizamos características como workspaces, que nos permiten tener varios entornos de infraestructura con diferentes configuraciones, y etiquetas de versión para identificar y gestionar los cambios realizados en la configuración de Terraform.
8. **Monitoreo y gestión del estado de la infraestructura:** Utilizamos herramientas de monitoreo y gestión del estado de la infraestructura, como CloudWatch en AWS, Azure Monitor.

## 2.11 Mejores prácticas y consideraciones de seguridad

A continuación se presentan algunas mejores prácticas para el uso seguro y eficiente de Terraform en Azure:

1. **Gestión de credenciales:** Es importante gestionar las credenciales de Azure de forma segura y cuidadosa para garantizar que los recursos se configuren correctamente y que los datos de la cuenta de Azure estén protegidos. Se recomienda utilizar **Azure Active Directory** (AD) para autenticar a los usuarios y controlar el acceso a los recursos. Las credenciales de la cuenta de Azure deben mantenerse en secreto y no deben ser compartidas públicamente o almacenadas en texto plano. Terraform admite la autenticación con **Azure AD** a través de la

integración con **Azure CLI**, lo que permite a los usuarios autenticarse de forma segura sin tener que almacenar credenciales de forma explícita.

2. **Seguridad de la configuración de Terraform:** Terraform utiliza archivos de configuración para definir la infraestructura, por lo que es importante asegurarse de que estos archivos sean seguros. Se recomienda utilizar un sistema de control de versiones para el código de Terraform y evitar el almacenamiento de credenciales o información confidencial en los archivos de configuración. Además, se pueden utilizar herramientas de análisis de código para identificar posibles vulnerabilidades y errores en el código.
3. **Revisión del código de Terraform:** Antes de implementar cualquier cambio en la infraestructura de Azure, es importante revisar cuidadosamente el código de Terraform para identificar posibles problemas. Esto puede incluir la revisión de los cambios propuestos en un entorno de desarrollo antes de implementarlos en producción. Además, se recomienda establecer un proceso de revisión de código en el que otros miembros del equipo revisen y aprueben los cambios antes de la implementación.
4. **Mejores prácticas para la gestión de cambios y versiones:** Es importante gestionar los cambios y versiones de la infraestructura de forma cuidadosa y controlada. Se recomienda utilizar un sistema de control de versiones para el código de Terraform y mantener un registro de todos los cambios realizados en la infraestructura. Además, se pueden utilizar herramientas de automatización y pruebas para asegurarse de que los cambios se implementen de manera consistente y sin errores.



## 3. Azure

### 3.1 Introducción a Azure

**Azure** o **Azure Cloud** es una **plataforma de computación en la nube** ofrecida por Microsoft.

Azure permite a las empresas y a los desarrolladores crear, implementar y administrar aplicaciones y servicios en la nube de manera eficiente y escalable. Ofrece una serie de herramientas y servicios para desarrollar, implementar, integrar y administrar aplicaciones en la nube, tanto para aplicaciones web como para aplicaciones empresariales.

Azure es utilizado por organizaciones de todo el mundo para desarrollar y ejecutar una amplia variedad de aplicaciones, desde aplicaciones web y móviles hasta aplicaciones empresariales y de análisis de datos. Ofrece una **infraestructura global** de centros de datos distribuidos en varias regiones, lo que permite a los usuarios elegir la ubicación geográfica donde desean alojar sus aplicaciones y datos. Esto facilita la escalabilidad y la disponibilidad de los servicios.

Azure también ofrece una amplia gama de **herramientas de seguridad y cumplimiento**, lo que lo convierte en una opción popular para aplicaciones que requieren altos niveles de seguridad y cumplimiento de normativas. Además, Azure cuenta con una comunidad activa de desarrolladores y una amplia documentación que ofrece recursos y soporte para el desarrollo y la implementación de aplicaciones en la plataforma.

### 3.2 Principales características de Azure

Algunas de las características destacadas de Azure son:

1. **Amplia gama de servicios:** Azure ofrece una amplia variedad de servicios en la nube, incluyendo infraestructura como servicio (IaaS), plataforma como servicio (PaaS), software como servicio (SaaS) y servicios especializados como inteligencia artificial (IA), Internet de las cosas (IoT), análisis de datos, almacenamiento, bases de datos, seguridad y más.

2. **Escalabilidad y flexibilidad:** Azure permite escalar vertical y horizontalmente las aplicaciones y servicios en función de las necesidades del negocio. Esto permite ajustar los recursos computacionales, almacenamiento y ancho de banda de manera flexible, lo que facilita el crecimiento y la adaptación a las demandas cambiantes.
3. **Disponibilidad global:** Azure tiene una presencia global con una red de centros de datos distribuidos en varias regiones del mundo, lo que permite a los usuarios elegir la ubicación geográfica donde desean alojar sus aplicaciones y datos. Esto facilita la disponibilidad y redundancia de los servicios, lo que garantiza la continuidad del negocio.
4. **Integración con tecnologías de Microsoft:** Azure se integra con muchas de las tecnologías de Microsoft, como Windows Server, SQL Server, Active Directory y muchas más. Esto facilita la migración de aplicaciones y servicios existentes a la nube de Azure y permite aprovechar las inversiones y conocimientos previos en tecnologías de Microsoft.
5. **Herramientas de desarrollo y gestión:** Azure ofrece una amplia gama de herramientas para desarrolladores y administradores, como Visual Studio, Azure DevOps, Azure Portal, Azure PowerShell, Azure CLI y muchas más. Estas herramientas facilitan el desarrollo, implementación y gestión de aplicaciones en la nube de Azure.
6. **Seguridad y cumplimiento:** Azure también cuenta con una amplia variedad de características de seguridad y cumplimiento, incluyendo cifrado de datos, firewall, monitoreo, detección de amenazas, autenticación y autorización, cumplimiento de normativas y certificaciones, y más. Esto ayuda a proteger los datos y aplicaciones alojados en Azure y cumple con los requisitos de seguridad y cumplimiento de las organizaciones.
7. **Precios flexibles:** Azure ofrece una variedad de opciones de precios flexibles, incluyendo modelos de pago por uso, suscripciones mensuales, planes de pago anticipado, descuentos por compromisos a largo plazo, y más. Esto permite a las

organizaciones elegir la opción de precios que mejor se adapte a sus necesidades y presupuesto.

8. **Soporte y comunidad:** Azure cuenta con una comunidad activa de desarrolladores y una amplia documentación en línea, así como soporte técnico de Microsoft. Esto brinda recursos y apoyo para el desarrollo, implementación y administración de aplicaciones en la nube de Azure.

### 3.3 Servicios de Azure

Azure ofrece una amplia variedad de tipos de recursos que los usuarios pueden crear y administrar para construir y desplegar aplicaciones y servicios en la nube. Algunos de los tipos de recursos más comunes en Azure son:

1. **Máquinas virtuales (VMs):** Las máquinas virtuales en Azure son instancias de sistemas operativos que se ejecutan en la nube. Los usuarios pueden crear y configurar VMs con diferentes sistemas operativos, tamaños de CPU, memoria, almacenamiento y redes para ejecutar aplicaciones y servicios.
2. **Almacenamiento:** Azure ofrece varios tipos de almacenamiento, como blobs, archivos, tablas y colas, que permiten a los usuarios almacenar y administrar diferentes tipos de datos en la nube. El almacenamiento en Azure es escalable, duradero y ofrece opciones de redundancia para garantizar la disponibilidad de los datos.
3. **Bases de datos:** Azure ofrece una variedad de servicios de bases de datos, como Azure SQL Database, MySQL, PostgreSQL, Cosmos DB, entre otros, que permiten a los usuarios crear y gestionar bases de datos en la nube de manera escalable y segura.
4. **Redes:** Azure proporciona una amplia gama de servicios de red, como redes virtuales (VNETs), gateways, equilibradores de carga, firewall, VPNs, y más, que permiten a los usuarios crear y configurar redes virtuales y conectar recursos en la nube con redes locales o con otras redes en la nube.

5. **Aplicaciones web:** Azure permite a los usuarios crear y desplegar aplicaciones web escalables utilizando servicios como Azure App Service, Azure Functions, Contenedores de Docker, y más. Estos servicios ofrecen opciones de implementación, escalado y administración de aplicaciones web en la nube.
6. **Servicios de inteligencia artificial (IA):** Azure ofrece servicios de IA, como Azure Machine Learning, Cognitive Services, y otros, que permiten a los usuarios crear y entrenar modelos de aprendizaje automático, procesar datos de forma inteligente y agregar capacidades de inteligencia artificial a sus aplicaciones.
7. **Internet de las cosas (IoT):** Azure proporciona servicios de IoT, como Azure IoT Hub, Azure IoT Edge, Azure IoT Central, que permiten a los usuarios conectar, monitorear y administrar dispositivos IoT, así como procesar y analizar datos de sensores en la nube.
8. **Seguridad y cumplimiento:** Azure ofrece una variedad de servicios de seguridad y cumplimiento, como Azure Active Directory, Azure Security Center, Azure Identity and Access Management (IAM), Azure Key Vault, y más, que permiten a los usuarios asegurar y proteger sus recursos y datos en la nube. A continuación vamos a definir aquellos servicios que hemos mencionado:
  - **Azure Active Directory:** Azure AD es un servicio de gestión de identidades y acceso basado en la nube proporcionado por Microsoft. Permite a las organizaciones gestionar el acceso y la autenticación de los usuarios a través de aplicaciones y servicios en la nube de Azure, así como a aplicaciones y servicios externos. Proporciona capacidades de autenticación, autorización, gestión de usuarios, gestión de grupos y seguridad en la nube. Gracias a esto, las organizaciones pueden proteger sus recursos y datos en la nube y gestionar de forma centralizada el acceso de los usuarios.
  - **Azure Security Center:** Es un servicio de seguridad integral de Azure que proporciona visibilidad y control continuo sobre la postura de seguridad de los recursos de Azure y de las cargas de trabajo en la nube. Ofrece detección de amenazas, análisis de seguridad, recomendaciones de seguridad y soluciones

automatizadas para proteger los recursos de Azure contra amenazas de seguridad y vulnerabilidades.

- **Azure Identity and Access Management (IAM):** Es un servicio de Azure que permite a las organizaciones gestionar el acceso a los recursos de Azure de forma granular y basada en roles. IAM proporciona capacidades de gestión de identidades y acceso, como la creación de usuarios, asignación de roles, definición de permisos y auditoría de acceso, lo que permite a las organizaciones tener un control preciso sobre quién puede acceder a sus recursos de Azure y qué acciones pueden realizar.
- **Azure Key Vault:** Es un servicio de Azure que permite a las organizaciones gestionar y proteger claves criptográficas, secrets y certificados en la nube. Proporciona un almacén seguro y centralizado para el almacenamiento y gestión de claves y secrets utilizados en aplicaciones y servicios de Azure, lo que ayuda a proteger la información sensible y cumplir con los requisitos de cumplimiento y regulación en el manejo de datos sensibles. Key Vault también permite la generación y administración de certificados **SSL/TLS** para aplicaciones web y otros servicios en la nube de Azure.

La plataforma también cuenta con una amplia gama de otros servicios y características. Esto permite a los usuarios elegir y combinar los recursos que mejor se adapten a sus necesidades y requerimientos de aplicaciones y servicios en la nube.

### 3.4 Aprovisionamiento de recursos en Azure

El **aprovisionamiento de recursos** en Azure es el proceso de **crear, configurar y asignar** recursos en la nube de Azure. Los recursos son objetos que representan servicios, aplicaciones y otros elementos que se utilizan en la plataforma de Azure. Algunos ejemplos comunes de recursos de Azure son máquinas virtuales, redes virtuales, grupos de seguridad, bases de datos y almacenamiento.

Existen varias herramientas que se pueden utilizar para aprovisionar recursos en Azure, pero una de las más populares es **Terraform**. Como hemos visto anteriormente, terraform

es una herramienta de infraestructura como código (IaC) que permite a los usuarios definir, crear y gestionar su infraestructura como si fuera código.

Con Terraform, los usuarios pueden crear archivos de configuración que describen los recursos que desean aprovisionar en Azure. Estos archivos de configuración se pueden versionar, revisar y compartir entre equipos, lo que facilita la colaboración y la gestión del ciclo de vida de la infraestructura.

Una vez que se ha creado el archivo de configuración, se utiliza Terraform para inicializar el entorno de trabajo, validar la sintaxis y, finalmente, aprovisionar los recursos. Terraform se comunica con la API de Azure para crear y configurar los recursos según se especifica en el archivo de configuración.

Una vez que se han aprovisionado los recursos, se pueden utilizar otras herramientas y servicios de Azure para gestionarlos, como Azure Portal, Azure CLI o Azure PowerShell. Además, Terraform también puede ser utilizado para actualizar, modificar o eliminar los recursos aprovisionados, lo que permite gestionar eficientemente el ciclo de vida completo de la infraestructura en la nube de Azure.

### 3.5 Modelos de implementación

Azure ofrece dos modelos de implementación principales:

- **El modelo de Resource Manager (ARM):** En este modelo, los recursos de Azure se definen en un archivo de modelo de Resource Manager (Plantilla ARM) que describe la infraestructura completa que se va a implementar. Este archivo de modelo se puede compartir, controlar versiones y usar para implementar la misma infraestructura en diferentes entornos. Este modelo es más adecuado para aplicaciones que cambian con frecuencia y tienen requisitos de escalabilidad, disponibilidad y seguridad.
- **El modelo clásico:** Es el modelo de implementación original de Azure. En este modelo, los recursos de Azure se crean y administran individualmente y se agrupan en servicios. Cada servicio puede tener sus propias directivas de seguridad y acceso. Este modelo es más adecuado para aplicaciones que no cambian con frecuencia y tienen requisitos de seguridad y conformidad únicos.

## 3.6 Integraciones

Azure ofrece una amplia gama de integraciones con otras tecnologías y servicios, lo que permite a los usuarios construir soluciones más completas y escalables. A continuación describimos algunos de los tipos de integraciones que se pueden hacer en Azure:

1. **Integraciones con servicios de terceros:** Azure se integra con muchos servicios de terceros, incluyendo GitHub, Bitbucket, Slack, Salesforce, SAP, Oracle y más.
2. **Integraciones con herramientas de desarrollo:** Azure ofrece integraciones con herramientas de desarrollo populares, como Visual Studio, Visual Studio Code, Eclipse y más. Esto permite a los desarrolladores trabajar con sus herramientas favoritas mientras crean soluciones en la nube de Azure.
3. **Integraciones de base de datos:** Azure se integra con muchas bases de datos populares, como SQL Server, MySQL, PostgreSQL y más. Esto permite mover fácilmente las aplicaciones y datos a la nube de Azure y aprovechar los servicios de Azure para administrar las bases de datos.
4. **Integraciones de IoT:** Azure ofrece integraciones con muchos dispositivos y servicios de IoT (Internet of Things), como dispositivos de borde, sensores, plataformas de IoT y más.
5. **Integraciones de inteligencia artificial:** Azure se integra con muchos servicios de inteligencia artificial y aprendizaje automático, como Azure Machine Learning, Azure Cognitive Services, etc.
6. **Integraciones de seguridad:** Azure se integra con muchas soluciones de seguridad populares, como **Azure Security Center**, **Azure Active Directory**, etc. Esto permite proteger las aplicaciones y datos en la nube de Azure y cumplir con los requisitos de seguridad y conformidad.

La plataforma también ofrece integraciones adicionales, como integraciones con sistemas de mensajería, sistemas de gestión de eventos...

## 3.7 Facturación y precios

Azure ofrece una variedad de opciones de facturación y precios para adaptarse a las necesidades de diferentes usuarios y empresas.

También dispone de la siguiente [página](#) donde podemos solicitar un presupuesto acorde a nuestras necesidades, y ver los detalles de precios por producto. Además cuenta con una calculadora de precios.

A continuación, vamos a describir algunas de las opciones de facturación y precios disponibles en Azure:

1. **Pago por uso:** Con la opción de pago por uso, los usuarios solo pagan por los recursos que utilizan, como el tiempo de CPU, el almacenamiento y el ancho de banda. Esta opción es adecuada para cargas de trabajo variables o para aquellos que desean probar Azure sin comprometerse a un contrato a largo plazo.
2. **Planes de suscripción:** Azure ofrece varios planes de suscripción, que brindan acceso a servicios adicionales y descuentos en el uso de recursos. Los planes de suscripción incluyen opciones gratuitas, como la suscripción gratuita de Azure, o opciones de pago, como la suscripción mensual de Azure.
3. **Contratos empresariales:** Para las empresas que desean una mayor previsibilidad en su presupuesto, Azure ofrece contratos empresariales, que permiten a los clientes comprometerse con el uso de una cantidad determinada de recursos durante un período de tiempo determinado. Estos contratos suelen incluir descuentos en el uso de recursos.
4. **Azure Cost Management:** Es una herramienta que permite a los usuarios supervisar y controlar los costos de sus recursos de Azure. La herramienta proporciona informes detallados sobre los costos de los recursos, así como alertas y recomendaciones para ayudar a los usuarios a optimizar sus costos.

Es importante tener en cuenta que los precios de Azure pueden variar según la región geográfica, el tipo de servicio y otros factores. Se recomienda revisar cuidadosamente la documentación y los precios de Azure antes de tomar una decisión de compra o suscripción.



### 3.8 Herramientas y servicios de gestión

Azure es una plataforma en la nube de Microsoft que ofrece una amplia variedad de herramientas y servicios de gestión para ayudar a los usuarios a administrar y mantener sus aplicaciones y recursos en la nube. A continuación se describen las herramientas y servicios de gestión más destacados:

- **Azure Portal:** Es una interfaz web basada en la nube que permite a los usuarios administrar sus recursos de Azure. El Portal de Azure, nos permite crear, modificar y eliminar recursos, así como monitorear el estado de sus aplicaciones y servicios.
- **Azure CLI:** Es una herramienta de línea de comandos que permite administrar los recursos de Azure a través de la terminal. Con Azure CLI, podemos realizar tareas como crear y eliminar recursos, administrar permisos y configurar la seguridad.
- **Azure PowerShell:** Es un módulo de PowerShell que permite a los usuarios administrar sus recursos de Azure a través de scripts de PowerShell. Azure PowerShell nos permite automatizar tareas de administración y configuración, lo que ayuda a reducir el tiempo y los errores.
- **Azure Advisor:** Es un servicio que ofrece recomendaciones personalizadas para optimizar los recursos de Azure en áreas como el rendimiento, la disponibilidad, la seguridad y el costo.
- **Azure Monitor:** Es un servicio de monitoreo que permite a los usuarios recopilar y analizar datos de telemetría de sus aplicaciones y servicios en Azure. Podemos realizar un seguimiento del rendimiento, detectar y solucionar problemas de manera proactiva, y recibir alertas cuando se detecten problemas críticos.
- **Azure Security Center:** Es un servicio de seguridad que proporciona una vista unificada de la postura de seguridad de los recursos de Azure de los usuarios. Con Azure Security Center, los usuarios pueden identificar y remediar vulnerabilidades, protegerse contra amenazas y cumplir con los requisitos de cumplimiento.
- **Azure Automation:** Es un servicio que permite automatizar tareas de administración y configuración en Azure y en otros entornos de nube y locales. Con

Azure Automation, podemos programar tareas de forma regular o desencadenarlas según sea necesario, lo que ayuda a reducir el tiempo y los errores.

### 3.9 Seguridad y cumplimiento

La seguridad y el cumplimiento son consideraciones críticas para cualquier organización que utilice servicios en la nube, incluido Microsoft Azure. En Azure, la seguridad y el cumplimiento son abordados mediante una variedad de herramientas y medidas de seguridad diseñadas para proteger los datos y la infraestructura del usuario.

Algunas de las características de seguridad en Azure incluyen:

- **Control de acceso basado en roles (RBAC):** Azure permite a los administradores de la cuenta definir y asignar roles a los usuarios, lo que limita su capacidad para realizar ciertas acciones en la plataforma. Esto ayuda a garantizar que solo los usuarios autorizados tengan acceso a los recursos y datos sensibles.
- **Seguridad en capas:** Azure utiliza múltiples capas de seguridad, que incluyen autenticación, autorización y encriptación, para proteger la infraestructura y los datos del usuario.
- **Cumplimiento normativo:** Azure está certificado por varias normativas de cumplimiento, como ISO 27001, SOC 1 y SOC 2, HIPAA y GDPR, lo que ayuda a garantizar que los datos del usuario estén protegidos y se cumplan los requisitos regulatorios.
- **Herramientas de monitoreo y alerta:** Azure ofrece herramientas de monitoreo y alerta en tiempo real para detectar y responder rápidamente a posibles amenazas de seguridad.
- **Protección contra amenazas avanzadas:** Azure utiliza tecnologías avanzadas de detección y prevención de amenazas, como aprendizaje automático, análisis de comportamiento y análisis de amenazas, para detectar y responder a las amenazas de seguridad más sofisticadas.

## 4. Azure Devops

### 4.1 ¿Qué es Azure Devops?

**Azure DevOps** es una plataforma de colaboración de software basada en la nube que proporciona herramientas para el desarrollo de software, la gestión de proyectos y el seguimiento de errores. Es una solución integral que abarca todo el ciclo de vida de desarrollo de software, desde la planificación y el seguimiento hasta la entrega y la implementación continua.

Azure DevOps proporciona características integradas a las que puede acceder a través del **explorador web** o el **cliente IDE**. Puede usar todos los servicios incluidos en Azure DevOps o elegir solo lo que necesita para complementar los flujos de trabajo existentes.

### 4.2 Herramientas y servicios de Azure DevOps

Azure DevOps ofrece una amplia gama de herramientas y servicios, incluyendo:

Servicio independiente	Descripción
<a href="#">Azure Boards</a>	Ofrece un conjunto de herramientas ágiles para admitir el trabajo de planificación y seguimiento, defectos de código y problemas mediante métodos Kanban y Scrum. Para obtener más información sobre Azure Boards, vea <a href="#">¿Qué es Azure Boards?</a> .
<a href="#">Azure Repos</a>	Proporciona repositorios de Git o Control de versiones de Team Foundation (TFVC) para el control de código fuente. Para obtener más información sobre Azure Repos, consulte <a href="#">¿Qué es Azure Repos?</a> .
<a href="#">Azure Pipelines</a>	Proporciona servicios de compilación y versión para admitir la integración y entrega continuas de las aplicaciones. Para más información sobre Azure Pipelines, consulte <a href="#">¿Qué es Azure Pipelines?</a> .

<p style="text-align: center;"><a href="#">Azure Test Plans</a></p>	<p>Proporciona varias herramientas para probar las aplicaciones, incluidas las pruebas manuales o exploratorias y las pruebas continuas. Para obtener más información sobre Azure Test Plans, vea <a href="#">Información general sobre Azure Test Plans</a>.</p>
<p style="text-align: center;"><a href="#">Azure Artifacts</a></p>	<p>Permite a los equipos compartir paquetes como Maven, npm, NuGet, etc. Desde orígenes públicos y privados e integrar el uso compartido de paquetes en las canalizaciones. Información: <a href="#">Introducción a Azure Artifacts</a>.</p>

### 4.3 Características y funcionalidades para el desarrollo

Azure DevOps ofrece una serie de características y funcionalidades que son beneficiosas para los equipos de desarrollo. Algunas de ellas son:

- **Integración con otras herramientas y servicios de Microsoft:** Como pueden ser Visual Studio, Azure, Office 365 y Dynamics 365, lo que permite una gestión unificada del ciclo de vida de desarrollo de software.
- **Personalización y extensibilidad:** Lo que significa que los equipos pueden adaptar la plataforma a sus necesidades específicas. Por ejemplo, pueden crear flujos de trabajo personalizados, desarrollar extensiones personalizadas o utilizar una API para integrarse con otras herramientas y servicios.
- **Control de versiones:** Azure DevOps proporciona un sólido sistema de control de versiones basado en **Git** y **TFVC**, que permite a los equipos realizar un seguimiento de los cambios en el código fuente y colaborar de manera efectiva.
- **Tableros Kanban y paneles de control:** Azure Boards incluye tableros **Kanban** y paneles de control personalizables para el seguimiento de problemas y tareas, lo que facilita la planificación, el seguimiento y la visualización del progreso del proyecto.

- **Pruebas y automatización:** Azure Test Plans permite a los equipos crear y ejecutar pruebas de software de forma automatizada, lo que ayuda a garantizar la calidad del software y a acelerar el tiempo de entrega.
- **Integración y entrega continua:** Azure Pipelines ofrece una solución de integración y entrega continua que permite a los equipos automatizar la compilación, las pruebas y la implementación de software en una variedad de plataformas y nubes.

## 4.4 Trabajos paralelos

Cuando definimos una pipeline, podemos hacerlo como colección de trabajos. Es decir, a la hora de ejecutar una pipeline, podemos ejecutar varios trabajos como parte de esta. Cada trabajo en ejecución consume un trabajo paralelo que se ejecuta en un agente. Cuando no hay suficientes trabajos paralelos disponibles para su organización, los trabajos se ponen en cola y se ejecutan uno tras otro.

Por ejemplo, si tenemos una pipeline que necesita construir y probar una aplicación, podemos dividir las tareas de construcción y prueba en dos trabajos paralelos separados. De esta forma, ambas tareas se pueden ejecutar simultáneamente en diferentes agentes, lo que puede reducir significativamente el tiempo total de ejecución de la pipeline.

Además, podemos configurar los trabajos paralelos en Azure DevOps para que se ejecuten en diferentes agentes o en diferentes entornos, lo que puede ayudar a distribuir la carga de trabajo y a optimizar los recursos del equipo.

En **Azure Pipelines**, podemos ejecutar trabajos paralelos en la infraestructura hospedada por Microsoft o en una propia infraestructura (autohospedada). Cada trabajo paralelo le permite ejecutar un único trabajo a la vez en su organización. No es necesario pagar por trabajos paralelos si usa un servidor local. El concepto de trabajos paralelos solo se aplica a **Azure DevOps Services**.

## 4.5 Trabajos paralelos autohospedados frente a hospedados por Microsoft

### 4.5.1 Hospedado por Microsoft

En el caso de que vayamos a ejecutar los trabajos en máquinas que Administra Microsoft, usaremos trabajos paralelos hospedados por Microsoft. Los trabajos se ejecutarán en agentes [hospedados por Microsoft](#).

**Azure Pipelines** proporciona un grupo de agentes predefinido denominado Azure Pipelines con agentes hospedados por Microsoft.

Imagen	Especificación del agente del editor clásico	Etiqueta de imagen de máquina virtual YAML	Software incluido
Windows Server 2022 con Visual Studio 2022	<i>windows-2022</i>	Windows-latest owindows-2022	<a href="#">Vínculo</a>
Windows Server 2019 con Visual Studio 2019	<i>windows-2019</i>	windows-2019	<a href="#">Vínculo</a>
Ubuntu 22.04	<i>ubuntu-22.04</i>	Ubuntu-latest oubuntu-22.04	<a href="#">Vínculo</a>
Ubuntu 20.04	<i>ubuntu-20.04</i>	ubuntu-20.04	<a href="#">Vínculo</a>
MacOS 12 Monterey	<i>macOS-12</i>	MacOS-latest omacOS-12	<a href="#">Vínculo</a>
macOS 11 Big Sur	<i>macOS-11</i>	macOS-11	<a href="#">Vínculo</a>

En el caso de los trabajos paralelos hospedados por Microsoft, podemos obtener hasta 10 trabajos paralelos hospedados por Microsoft gratuitos que se pueden ejecutar hasta 360 minutos (6 horas) cada vez para **los proyectos públicos**. Al crear una nueva organización de Azure DevOps, no se le concede esta concesión gratuita de forma predeterminada.

En el caso de **los proyectos privados**, podemos obtener un trabajo gratuito que se pueda ejecutar durante un máximo de 60 minutos cada vez. Al crear una nueva organización de Azure DevOps, es posible que no siempre se le conceda esta concesión gratuita de forma predeterminada. Por lo que debemos de solicitarla en el [siguiente enlace](#).

#### 4.5.2 Autohospedado

En el caso de los trabajos paralelos autohospedados, podemos registrar cualquier número de [agentes autohospedados](#) en una organización. Se cobrará según el número de trabajos que se ejecuten a la vez, y no por el número de agentes registrados. No hay límites de tiempo en los trabajos autohospedados.

En el caso de los **proyectos públicos** autohospedados, podemos tener trabajos paralelos ilimitados en ejecución. En el caso de los **proyectos privados**, podemos tener un trabajo y un trabajo adicional para cada suscriptor activo Visual Studio Enterprise que sea miembro de la organización.

	<b>Nº de trabajos paralelos</b>	<b>Límite de tiempo</b>
<b>Proyecto público</b>	Sin límite	Ninguno
<b>Proyecto privado</b>	Un trabajo autohospedado, aunque por cada suscriptor activo Visual Studio Enterprise que sea miembro de la organización, obtenemos un trabajo paralelo autohospedado adicional.	Ninguno

## 5. Casos Prácticos (Terraform y Azure)

### 5.1 Configuración del entorno de desarrollo con Terraform y Azure.

#### 5.1.1 Máquina Windows

Para realizar la instalación de Terraform sincronización con una cuenta de Azure en una máquina Windows seguiremos los siguientes pasos:

##### Paso 1 (Descargar Terraform)

<https://www.terraform.io/>

##### Paso 2 (Añadir el ejecutable de Terraform al Path de Windows)

Descomprimos el fichero descargado anteriormente (.zip) que contiene el binario de Terraform y lo movemos al path de Windows "**C:\Windows\System32**".

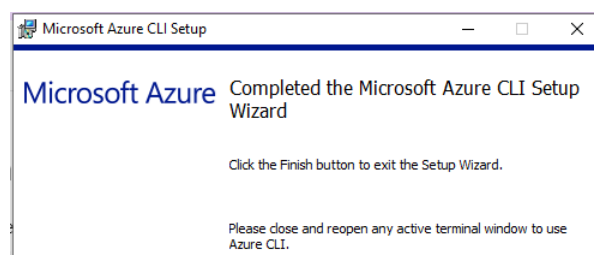
##### Paso 3 (Descargar e instalar Azure CLI)

La interfaz de línea de comandos (CLI) de Azure es una herramienta de línea de comandos multiplataforma para conectarse a Azure y ejecutar comandos administrativos en los recursos de Azure. Permite la ejecución de comandos a través de un terminal utilizando indicaciones de línea de comandos interactivas o un script.

Descargamos, en este caso el ejecutable .msi.

<https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli>

Una vez descargado, lo instalamos.





Para comprobar la versión que se ha instalado, abrimos una Powershell y ejecutamos el siguiente comando:

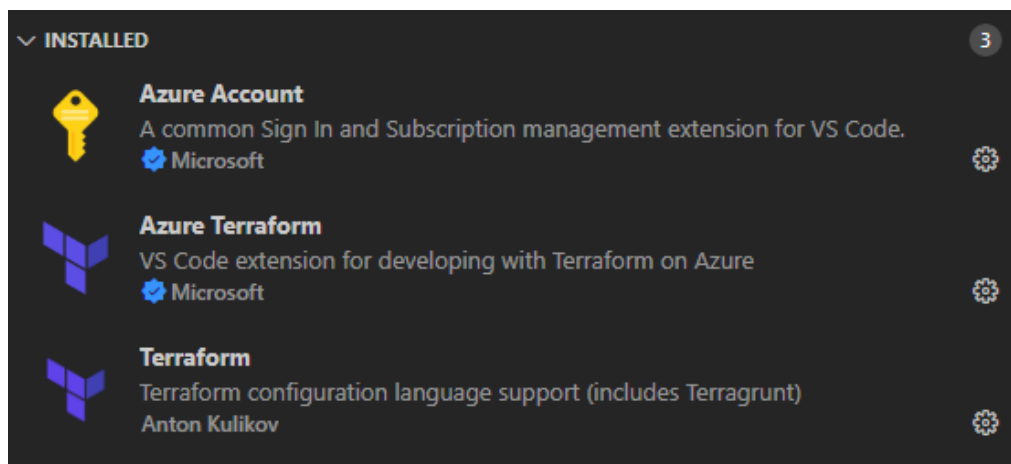
```
az version
```

El comando anterior nos devolverá la siguiente salida:

```
PS> az version
{
  "azure-cli": "2.47.0",
  "azure-cli-core": "2.47.0",
  "azure-cli-telemetry": "1.0.8",
  "extensions": {}
}
```

**Paso 4** (Instalamos un **IDE** para editar el código con el que vamos a trabajar)

En este caso, recomiendo usar [Visual Studio Code](#) y añadiendo las extensiones "Terraform" y "Azure Terraform").

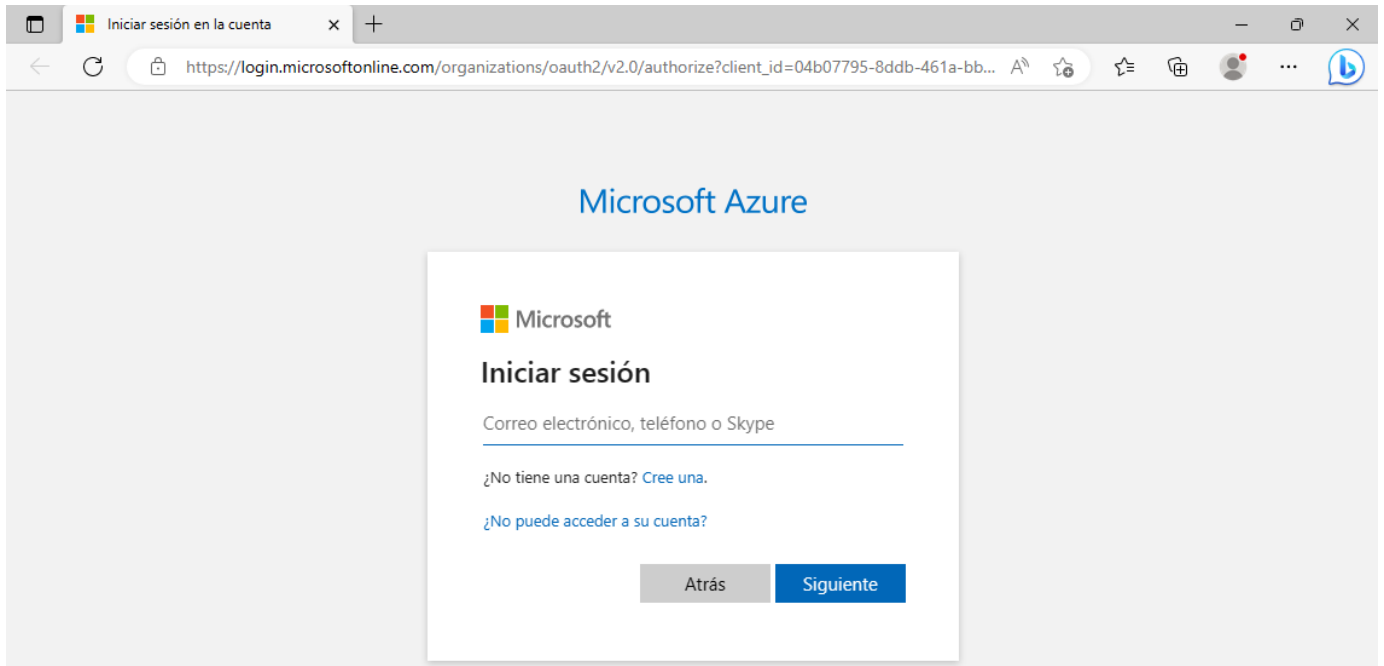


**Paso 5** (Nos logueamos desde la consola)

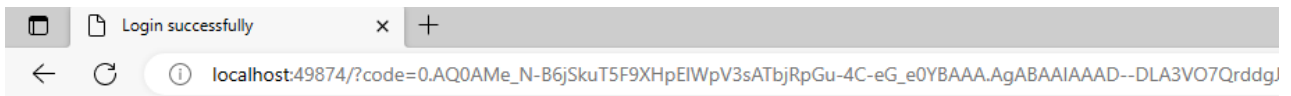
Abrimos la **Powershell** y ejecutamos el siguiente comando:

```
az login
```

Una vez ejecutemos el comando anterior, se abrirá una ventana en el navegador, donde requerirá una autenticación con nuestra cuenta de Microsoft Azure.



Una vez nos logueemos se mostrará el siguiente mensaje:



### You have logged into Microsoft Azure!

You can close this window, or we will redirect you to the [Azure CLI documentation](#) in 1 minute.

### Announcements

[Windows only] Starting in May 2023, Azure CLI will authenticate using the [Web Account Manager](#) (WAM) broker by default.

To help us collect feedback on the new login experience, you may opt-in to use WAM by running the following commands:

```
az config set core.allow_broker=true
az account clear
az login
```

En la terminal, se mostrará como salida un **JSON** similar al siguiente:

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a0ce8a08-6fc1-4e4e-b707-b9d558a97fc2",
    "id": "d26cf210-db93-4db7-9cea-d9ddce1cd55d",
    "isDefault": true,
    "managedByTenants": [],
```

```

"name": "Azure subscription 1",
"state": "Enabled",
...
}
]

```

Con esto ya podemos tener sincronizado Terraform con nuestra cuenta de Azure, y podremos comenzar a trabajar con una plantilla de terraform **“main.tf”**.

Como vemos se ha asociará el ID de la correspondiente suscripción de Azure.

The screenshot displays the Azure portal interface for 'Azure subscription 1'. The 'Información esencial' section shows the subscription ID as 'd26cf210-db93-4db7-9cea-d9ddce1cd55d', which is highlighted with a red box. A Windows PowerShell terminal window is overlaid on the right, showing the output of the 'az login' command. The 'id' field in the JSON output is highlighted in red, showing the same subscription ID: 'd26cf210-db93-4db7-9cea-d9ddce1cd55d'.

En caso de que queramos seleccionar una suscripción distinta (Porque dispongamos de varias), empleamos el siguiente comando:

```
az account set --subscription "<SUBSCRIPTION-ID>"
```

Además del **“id”** de la suscripción, podemos encontrar el **“tenantId”**.

Un **tenant de Azure** es una instancia aislada y segura de la nube de Azure en la que una organización asociada puede crear, configurar y gestionar sus recursos y servicios en la nube.

Cuando una organización se registra para utilizar los servicios de Azure, se crea automáticamente un tenant de Azure para esa organización.

Dentro de un tenant de Azure, una organización puede **crear** y **gestionar** una amplia gama de **recursos**, como máquinas virtuales, bases de datos, redes, almacenamiento y servicios de aplicaciones, entre otros. También se pueden configurar políticas de seguridad, gestionar el acceso a los recursos y establecer la facturación y la suscripción a los servicios.

En el panel de Azure Cloud, podemos encontrar el **TenantId** desde “**Azure Active Directory/Default Directory/Propiedades**”.

The image shows two side-by-side screenshots. On the left is the Azure Active Directory 'Default Directory' properties page. The 'Id. del inquilino' field is highlighted with a red box and contains the value 'a0ce8a08-6fc1-4e4e-b707-b9d558a97fc2'. On the right is a Windows PowerShell terminal window showing the output of the 'az login' command. The 'tenantId' field in the JSON output is highlighted with a red box and contains the same value: 'a0ce8a08-6fc1-4e4e-b707-b9d558a97fc2'.

## 5.1.2 Máquina Linux

### Paso 1 (Descargar e instalar Terraform)

En este caso, se ha realizado una prueba de instalación sobre una máquina **Ubuntu 22.04.2 LTS**.

Podemos descargar el binario desde la [página oficial](#).

O realizar la instalación mediante los siguientes comandos:

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform
```

Comprobamos la versión con el siguiente comando:

```
$ terraform --version
Terraform v1.4.5
on linux_amd64
```

## Paso 2 (Descargar e instalar Azure CLI)

Podemos instalarlo utilizando únicamente el siguiente comando:

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Una vez instalado comprobamos la versión mediante el comando:

```
$ az version
{
  "azure-cli": "2.47.0",
  "azure-cli-core": "2.47.0",
  "azure-cli-telemetry": "1.0.8",
  "extensions": {}
}
```

## Paso 4 (Instalamos un IDE para editar el código con el que vamos a trabajar)

En este caso, realizaremos la instalación de "[Visual Studio Code](#)" añadiendo las extensiones "Terraform" y "Azure Terraform").

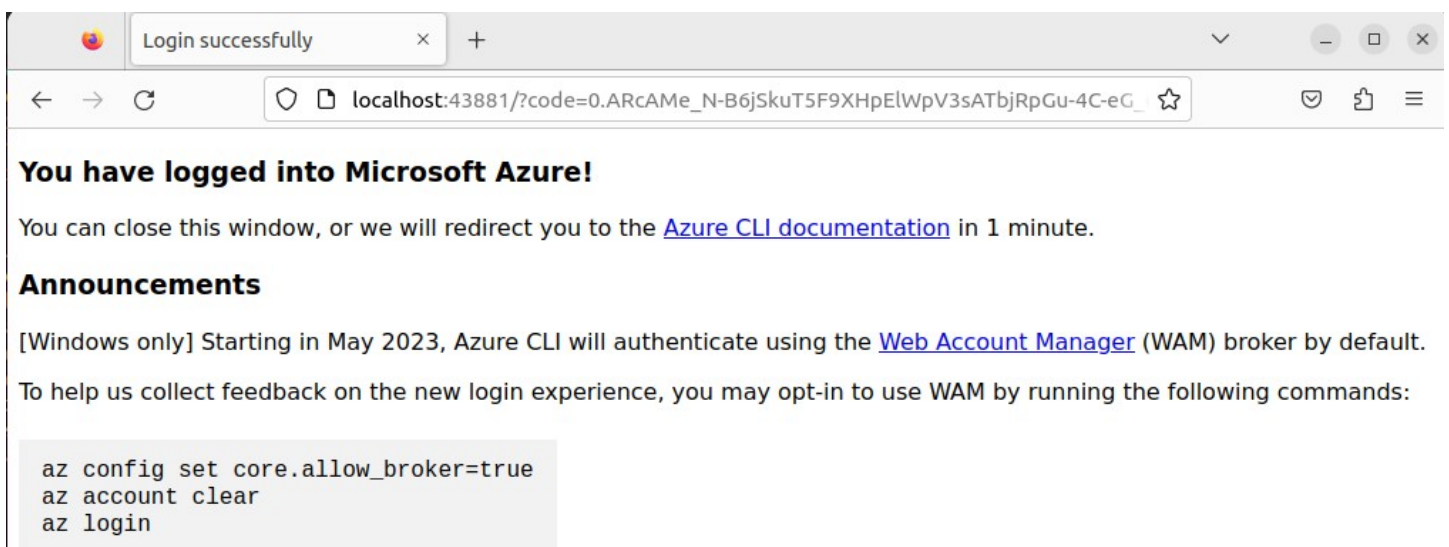
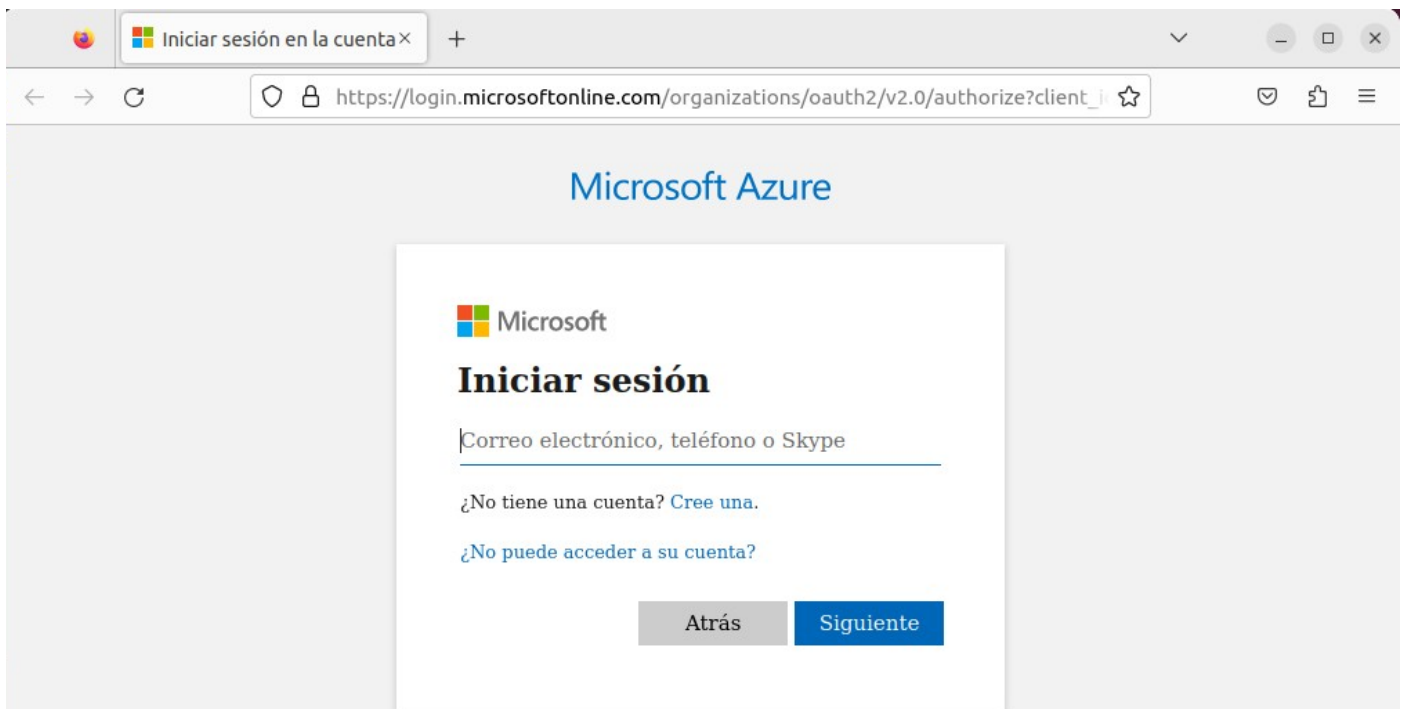
Descargamos la versión para [Linux \(Ubuntu\)](#) y posteriormente la instalamos mediante el siguiente comando:

```
dpkg -i code_1.77.3-1681292746_amd64.deb
```

**Paso 5** (Nos logueamos desde la consola)

```
az login
```

Después de ejecutar el comando anterior, se abrirá una ventana en el navegador, donde requerirá una autenticación con nuestra cuenta de Microsoft Azure.



```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a0ce8a08-6fc1-4e4e-b707-b9d558a97fc2",
    "id": "d26cf210-db93-4db7-9cea-d9ddce1cd55d",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Azure subscription 1",
    "state": "Enabled",
    ...
  }
]
```

Una vez hecho esto, ya tendríamos sincronizado terraform con nuestra suscripción de azure.

Para habilitar el autocompletado de Terraform, debemos de instalar el siguiente paquete:

```
terraform -install-autocomplete
```

## 5.2 Ejemplo de despliegue de infraestructura

Vamos a realizar una prueba práctica de un despliegue de recursos en Azure mediante Terraform. Para ello, utilizaré una [máquina Windows](#) que hemos configurado previamente.

El **Tenant ID** es un número único que identifica a la organización o empresa que utiliza los servicios de Azure. Cada organización o empresa que utiliza los servicios de Azure tiene un Tenant ID, que se utiliza para autenticar y autorizar el acceso a los recursos de Azure.

Primero realizaré un **login** desde la powershell seleccionando el **tenantid** correspondiente:

```
az login --tenant a0ce8a08-6fc1-4e4e-b707-b9d558a97fc2
```

En caso necesario seleccionamos el **id de nuestra suscripción**:

```
az account set --subscription d26cf210-db93-4db7-9cea-d9ddce1cd55d
```

En este caso realizaremos un ejemplo desplegando un servidor web mediante el uso de una máquina virtual y la infraestructura necesaria. El servidor devolverá **"Hello, World"** para la URL que escucha en el puerto **8080**.

En este caso, partiremos de la siguiente [plantilla de terraform \(main.tf\)](#).

Una vez seleccionada la suscripción correspondiente como hemos visto anteriormente, crearemos un objeto de servicio.

Un **objeto de servicio** es una aplicación dentro de Azure Active Directory con los **tokens** de autenticación que Terraform necesita para realizar acciones en su nombre.

Creamos el objeto de servicio mediante el siguiente comando:

```
az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/<SUBSCRIPTION_ID>"
```

Como vemos hemos asignado el rol **"Contributor"**. En Azure, el rol de "Contributor" es uno de los roles de acceso más comunes y permite a los usuarios realizar acciones específicas en los recursos de Azure, como crear y administrar recursos.

El parámetro **"--scopes"** especifica la suscripción de Azure a la que se le darán los permisos de colaborador al objeto de servicio de Azure AD recién creado.

Una vez ejecutado el comando anterior obtendremos la siguiente salida:

```
{
  "appId": "0b3b9036-f023-4593-9ecc-fa7dfc4538a7",
  "displayName": "azure-cli-2023-05-03-07-32-08",
  "password": "fXD8Q~5pH6hdZlF0yJAY6xnglpaiirQTVqbUkb1r",
  "tenant": "a0ce8a08-6fc1-4e4e-b707-b9d558a97fc2"
}
```

Estos valores los asignaremos posteriormente a unas variables del entorno que vamos a definir en el SO agente que estamos utilizando. Dependiendo del SO lo realizaremos de una forma u otra.



**- Para Linux / MacOS:**

```
export ARM_CLIENT_ID="<SERVICE_PRINCIPAL_APPID>"
export ARM_CLIENT_SECRET="<SERVICE_PRINCIPAL_PASSWORD>"
export ARM_SUBSCRIPTION_ID="<SUBSCRIPTION_ID>"
export ARM_TENANT_ID="<TENANT_ID>"
```

**- Para Windows (PowerShell):**

```
$env:ARM_CLIENT_ID="<SERVICE_PRINCIPAL_APPID>"
$env:ARM_CLIENT_SECRET="<SERVICE_PRINCIPAL_PASSWORD>"
$env:ARM_SUBSCRIPTION_ID="<SUBSCRIPTION_ID>"
$env:ARM_TENANT_ID="<TENANT_ID>"
```

Una vez realizada toda esta configuración previa de Azure, comenzamos a utilizar Terraform para desplegar la infraestructura. Para ello seguimos los siguientes pasos:

Nos situamos en el directorio que contiene la plantilla de terraform, y inicializamos el directorio de trabajo de Terraform mediante el comando:

```
terraform init
```

A continuación realizaremos un **"terraform plan"** previsualizar los cambios que se van a realizar antes de aplicarlos.

```
terraform plan
```

Al ejecutar el comando anterior, veremos un resumen del despliegue y en la última línea nos indicará los cambios previstos.:

```
Plan: 9 to add, 0 to change, 0 to destroy.
```

Como vemos el plan añadirá las 9 secciones que nos muestra por pantalla.

Para aplicar los cambios, usaremos el comando **"terraform apply"** de la siguiente forma:

```
terraform apply
```

Si utilizáramos el parámetro **"auto-approve"**, estaremos ejecutando el comando de forma automatizada sin necesidad de recurrir a ningún tipo de aprobación manual.

El proceso de despliegue puede demorar bastante (Dependiendo del número de recursos).


```
Apply complete! Resources: 9 added, 0 changed, 0 destroyed.

Outputs:







public_ip = "108.142.192.1"
PS C:\Users\alfonso\Desktop\hello-word_tf>
```


Como vemos, al terminar el despliegue nos devolverá la IP Pública.





Una vez terminado el despliegue, nos dirigimos al panel de **Azure Portal** y comprobamos los recursos.








Todos los recursos  ...

Default Directory (megaalfonso84gmail.onmicrosoft.com)

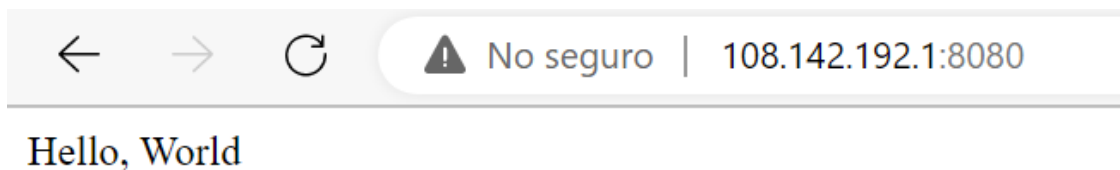
+ Crear  Administrar vista  Actualizar  Exportar a CSV  Abrir consulta |  Asignar etiquetas  Eliminar

Filtrar por cualquier campo  Suscripción es igual a **todo** Grupo de recursos es igual a **todo** Tipo es igual a **todo** Ubicación es igual a **todo**  Agregar filtro

 0 Recursos no seguros  0 Recomendaciones    Vista de li

<input type="checkbox"/> Nombre ↑↓	Tipo ↑↓	Grupo de recursos ↑↓	Ubicación ↑↓	Suscripción ↑↓
<input type="checkbox"/>  my-terraform-nic	Interfaz de red	my-terraform-rg	West Europe	Azure subscription 1
<input type="checkbox"/>  my-terraform-nsg	Grupo de seguridad de red	my-terraform-rg	West Europe	Azure subscription 1
<input type="checkbox"/>  my-terraform-os-disk	Disco	MY-TERRAFORM-RG	West Europe	Azure subscription 1
<input type="checkbox"/>  my-terraform-public-ip	Dirección IP pública	my-terraform-rg	West Europe	Azure subscription 1
<input type="checkbox"/>  my-terraform-vm	Máquina virtual	my-terraform-rg	West Europe	Azure subscription 1
<input type="checkbox"/>  my-terraform-vnet	Red virtual	my-terraform-rg	West Europe	Azure subscription 1
<input type="checkbox"/>  my-terraform-storage	Almacenamiento de datos	my-terraform-rg	West Europe	Azure subscription 1

Además, para validar que todo funciona correctamente, accedemos a la **IP Pública** (Recurso) con el **puerto** correspondiente (8080). Deberíamos ver por pantalla el mensaje **"Hello World"**.



Para limpiar los recursos creados, nos dirigimos al directorio de trabajo que contiene la plantilla terraform y ejecutamos el comando:

```
terraform destroy
```

## 6. Prueba práctica del proyecto

### 6.1 Objetivo propuesto

En este proyecto, vamos a realizar una demostración práctica en la que implementaremos un código de una plantilla de Terraform en una pipeline de **Azure DevOps**.

La plantilla dependerá de un fichero para almacenar **variables**.

En esta prueba práctica, subiremos el código a un repositorio de “**Azure Repos**” y configuraremos una pipeline mediante un fichero YAML, de forma que al realizar un cambio en el repositorio se realice el despliegue o los cambios definidos en la infraestructura.

En definitiva, **HashiCorp Terraform**, usado con **Microsoft Azure DevOps**, proporciona una forma de configurar implementaciones automatizadas de infraestructura como código.

### 6.2 Prerrequisitos

Antes de iniciar con la práctica propuesta, debemos de contar con una serie de prerrequisitos:

- Una cuenta de Azure.
- Una suscripción de Azure para poder desplegar los recursos necesarios.
- Una cuenta de Azure DevOps.

- Una organización y un proyecto de Azure DevOps.
- Un agente autohospedado o hospedado por Microsoft para realizar trabajos paralelos.
- Repositorio de código fuente.
- Un editor de texto (En este caso Visual Studio Code).

## 6.3 Pasos a seguir para la implementación

### 6.3.1 Creación de un Backend en Azure

Una vez contamos con la cuenta de Azure, lo primero que haremos será crear un backend para mantener el estado de la infraestructura tras aplicar los cambios deseados.

Si no disponemos de un backend, deberíamos de importar cada uno de los recursos existentes manualmente dentro del código de terraform para que los incluya en el proyecto. Esto tomaría bastante más tiempo que realizar una consulta de estado mediante el backend.

Para crear e inicializar un backend seguiremos los siguientes pasos:

1. Nos conectamos a nuestra cuenta de azure desde la Powershell:

```
az login
```

En caso necesario seleccionamos la suscripción correspondiente:

```
az account set --subscription "<SUBSCRIPTION-ID>"
```

2. Creamos un grupo de recursos mediante el comando:

```
az group create --name dev1 --location eastus
```

Como vemos, el grupo de recursos se llamará **dev1**, con localización **eastus**.

En este punto contamos con el grupo de recursos, pero sin cuenta de almacenamiento.

Suponiendo que la cuenta de almacenamiento se llamará “**storageiesgn**”, podemos verificarlo con:

```
az storage account show --name storageiesgn --resource-group dev1
```

En la salida del comando, veremos que no existe ninguna cuenta de almacenamiento con ese nombre.

También podemos crear el grupo de recursos directamente desde el portal de azure usando un navegador web:

[Inicio](#) > [Todos los recursos](#) > [Crear un recurso](#) > [Marketplace](#) > [Grupo de recursos](#) >

## Crear un grupo de recursos

**✖** Datos básicos   Etiquetas   Revisar y crear

**Grupo de recursos** - Contenedor que incluye los recursos relacionados para una solución de Azure. El grupo de recursos puede contener todos los recursos de la solución o solamente los recursos que quiere administrar en grupo. Debe decidir cómo quiere asignar los recursos a los grupos de recursos según lo que resulte más pertinente para su organización. [Más información](#)

### Detalles del proyecto

Suscripción \* ⓘ

Suscripción de Azure 1

Grupo de recursos \* ⓘ

dev1

**✖** Ya existe un grupo de recursos con el mismo nombre en la suscripción seleccionada Suscripción de Azure 1.

### Detalles del recurso

Región \* ⓘ

(US) East US

Revisar y crear

< Anterior

Siguiente: Etiquetas >

2. Creamos una cuenta de almacenamiento desde el portal de azure, con el nombre que tenemos definido en la sección del backend dentro del main.tf (**storageiesgn**).

A la hora de crear la cuenta de almacenamiento, seleccionamos el grupo de recursos creado en el paso anterior.

[Inicio](#) > [Cuentas de almacenamiento](#) >

## Crear una cuenta de almacenamiento

[Datos básicos](#) | [Opciones avanzadas](#) | [Redes](#) | [Protección de datos](#) | [Cifrado](#) | [Etiquetas](#) | [Revisar](#)

Suscripción \*

Grupo de recursos \*  [Crear nuevo](#)

### Detalles de la instancia

Si necesita crear un tipo de cuenta de almacenamiento heredada, haga clic en [aquí](#).

Nombre de la cuenta de almacenamiento  ⓘ \*

✖ El nombre de cuenta de almacenamiento "storageiesgn" ya está en uso.

[Revisar](#)

[< Anterior](#)

[Siguiente: Opciones avanzadas >](#)

También deberemos especificar algunos parámetros como la localización, y la redundancia.

También podemos hacerlo mediante el comando:

```
az storage account create --name storageiesgn --resource-group dev1 --location eastus --sku Standard_LRS
```

3. A continuación, desde el portal de azure he accedido a la “**cuenta de almacenamiento/blob service**” y hemos agregado un contenedor con el nombre que tenemos definido en la sección del backend dentro del **main.tf**.

Inicio > Cuentas de almacenamiento > storageiesgn

Cuentas de almace... <<  
Default Directory (megaalfonso84gmail.onmicroso...

+ Crear ↶ Restaurar ...

Filtrar por cualquier campo...

Nombre ↑↓

storageiesgn ...

storageiesgn | Contenedores ☆ ☆ ...  
Cuenta de almacenamiento

Buscar

+ Contenedor 🔒 Cambiar nivel de acceso ↶ Restaurar contenedores ▾ Actualizar | ...

Buscar contenedores por prefijo

Mostrar contenedores eliminados

Nombre	Última modificación	Nivel de acceso púb...	Estado de concesión
<input type="checkbox"/> contenedoriesgn1	18/5/2023, 12:53:53	Privada	Disponible

Información general  
Registro de actividad  
Etiquetas  
Diagnosticar y solucionar problemas  
Control de acceso (IAM)

También podemos hacerlo mediante el comando:

```
az storage container create --name contenedoriesgn1 --account-name storageiesgn --account-key <storage-account-key>
```

Al hacerlo mediante el comando anterior, requerimos una clave de acceso para la cuenta de almacenamiento (Cada clave dispone de una serie de permisos). Podemos obtenerla con el siguiente comando:

```
az storage account keys list --account-name storageiesgn --resource-group dev1
```

Con esto ya tenemos el backend creado, para inicializarlo tan solo debemos de ejecutar **"terraform init"** en la máquina agente (La cual tiene terraform instalada). En este caso, este comando irá incluido dentro de la pipeline.

### 6.3.2 Creación de un proyecto en Azure DevOps.

Lo primero que haremos será crear una organización. Para ello, iniciamos sesión en Azure DevOps y en el panel principal pulsamos en **"New organization"**.

Azure DevOps

Search

megaalfonso84

+ New project

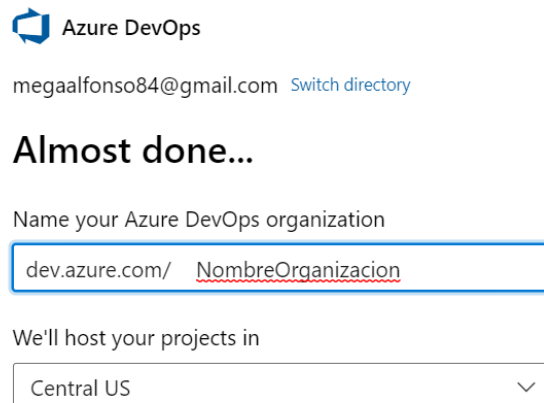
Projects My work items My pull requests

Filter projects

megaalfonso84

New organization

Por último definimos el nombre y la ubicación de los proyectos de dicha organización:



Azure DevOps

megaalfonso84@gmail.com [Switch directory](#)

## Almost done...

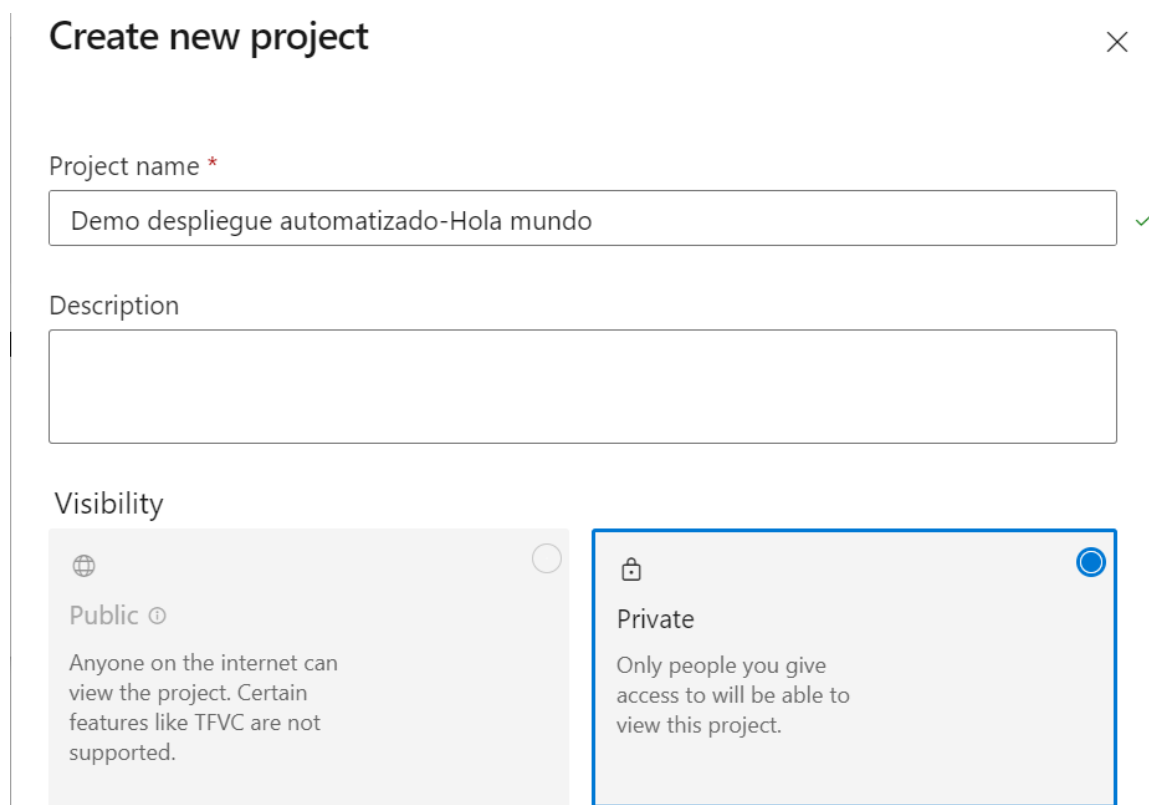
Name your Azure DevOps organization

dev.azure.com/ NombreOrganizacion

We'll host your projects in

Central US

Para crear un proyecto, accedemos a la pestaña de la organización correspondiente y pulsamos en **“New Project”**. Crearemos un proyecto **“Privado”**.



### Create new project

Project name \*

Demo despliegue automatizado-Hola mundo ✓

Description

Visibility

Public ⓘ  
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private  
Only people you give access to will be able to view this project.

Como paso adicional, vamos a solicitar a Microsoft, la utilización los repositorios y de trabajos paralelos, ya que por defecto viene desactivada para Agentes hospedados por microsoft (Esto se debe a la criptomonería).



Para ello accedemos a la organización correspondiente, y desde aquí nos dirigimos a “**Organization Settings/Pararell Jobs/Private Projects/Microsoft hosted**”. Una vez aquí, pulsamos en “**change**” y activamos los repositorios mediante la siguiente opción:

Boards, Repos and Test Plans	Free
Basic users <a href="#">↗</a>	5
Basic + Test Plans <a href="#">↗</a>	<b>Start free trial</b>

Si disponemos de una suscripción de pago de Azure, podemos vincularla a Azure DevOps, y ya no necesitaremos hacer uso de la opción anterior.

Además, en el caso de **los proyectos privados**, podemos obtener un trabajo gratuito que se pueda ejecutar durante un máximo de 60 minutos cada vez. Cuando creamos una nueva organización de Azure DevOps, puede que no siempre realice esta concesión gratuita de forma predeterminada. Por lo que debemos de solicitarla en el [siguiente enlace](#).

### 6.3.3 Añadir una conexión de servicio con Azure Cloud.

Normalmente, para logearnos con nuestra cuenta de Azure, usaríamos el comando “**az login**”. Sin embargo, este comando abriría nuestro navegador web para introducir nuestra cuenta de Azure. En el agente que ejecuta la pipeline, no podemos hacer esto.

Como alternativa ejecutar el comando con el siguiente parámetro:

```
az login -use-device-code
```

Este nos proporcionará un código en la propia terminal y un enlace, tan solo debemos de copiar el enlace, abrirlo en un navegador (De cualquier dispositivo) e introducir el código. Seguidamente introducimos las credenciales de nuestra cuenta de azure y nos habremos logueado. Este es un buen método por ejemplo para enviar un SMS con dicho mensaje y así contar con una autenticación de doble factor cada vez que ejecutemos la pipeline.

La última opción, es utilizar una **conexión de servicio**. En **Azure DevOps**, una conexión de servicio es una configuración que permite a los proyectos y pipelines de Azure DevOps interactuar con **servicios externos**. Estas conexiones de servicio se utilizan para autenticar y autorizar el acceso a recursos externos, como repositorios de código fuente, servicios de compilación, despliegue y pruebas, entre otros.


Para habilitar una conexión de servicio, nos dirigimos a nuestro proyecto de Azure DevOps, y luego a **“Project Settings / Service Connections / New service connection”**. En este caso seleccionamos la conexión de tipo **Azure Resource Manager**, nos aparecerá varias formas de añadirla (Como la automática en la que iniciamos sesión en Azure y la seleccionamos o introduciendo los parámetros necesarios manualmente).

← Suscripción de Azure 1(d26cf210-db93-4db7-9cea-d9ddce1cd55d)


Overview Usage history

Details

Service connection type

 Azure Resource Manager  
using service principal authentication  
[Manage service connection roles](#)  
[Manage Service Principal](#)

Creator

 Alfonso Roldán Amador  
megaalfonso84@gmail.com


### 6.3.4 Preconfiguración, creación de la pipeline (YAML).

Desde el panel de Azure DevOps, dentro de nuestro proyecto, nos dirigimos a **“Pipelines”**. Una vez aquí, pulsamos en **“Create Pipeline”**.


A continuación nos pedirá la fuente del código, en este caso seleccionaremos **“Azure Repos Git”**.

New pipeline

## Where is your code?

 Azure Repos Git YAML  
Free private Git repositories, pull requests, and code search

---

 Bitbucket Cloud YAML  
Hosted by Atlassian

A continuación seleccionamos el repositorio correspondiente:

New pipeline

## Select a repository

Filter by keywords

Azure-Pipelines-Demo



Azure-Pipelines-Demo

Una vez seleccionado el repositorio, nos dará la opción de crear una pipeline desde cero, partiendo de una plantilla que nos proporciona, o seleccionar una pipeline existente.

New pipeline

## Configure your pipeline



Starter pipeline

Start with a minimal pipeline that you can customize to build and deploy your code.



Existing Azure Pipelines YAML file

Select an Azure Pipelines YAML file in any branch of the repository.

Show more

En este caso, he optado por la 2ª opción, ya que cuento con una pipeline predefinida anteriormente con visual studio code.

Nos pedirá que seleccionemos la rama y la ruta del fichero yaml sobre el que se ejecutará la pipeline. En este caso, cuento con más de un fichero yaml, ya que he usado una opción que me permite llamar a un fichero desde otro (**template**). Por lo que selecciono el fichero principal (**terraform-plan.yaml**).

## Select an existing YAML file



Select an Azure Pipelines YAML file in any branch of the repository.

Branch

Path

Select a file from the dropdown or type in the path to your file

[Demo despliegue automatizado - Hola mundo](#)

Podemos encontrar el código empleado en [este repositorio](#).

Otro detalle importante, es disponer de las extensiones necesarias añadidas al proyecto. Por ejemplo, en la pipeline que vamos a usar, tenemos una tarea para instalar Terraform en la máquina agente ([TerraformInstaller@0](#)).

```
- task: TerraformInstaller@0
  displayName: "Install Terraform"
  inputs:
    terraformVersion: '1.4.6'
    terraformDownloadLocation: 'https://releases.hashicorp.com/terraform'
```

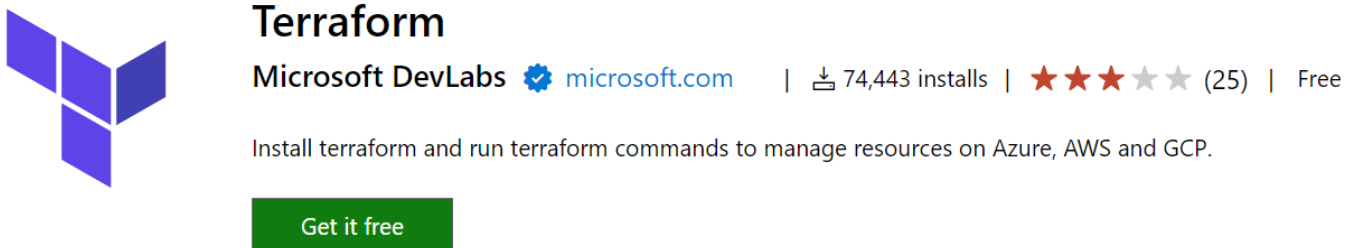
Esta tarea no funcionará si no hemos añadido esta extensión al proyecto.

Para añadirla, desde el panel de Azure DevOps, hacemos clic en la bolsa de la compra que aparece en la esquina superior derecha, y posteriormente nos dirigimos a **Browse Marketplace**.

## Extensions for Azure DevOps



Una vez aquí, buscamos la extensión correspondiente para la instalación de Terraform y la añadimos al proyecto.



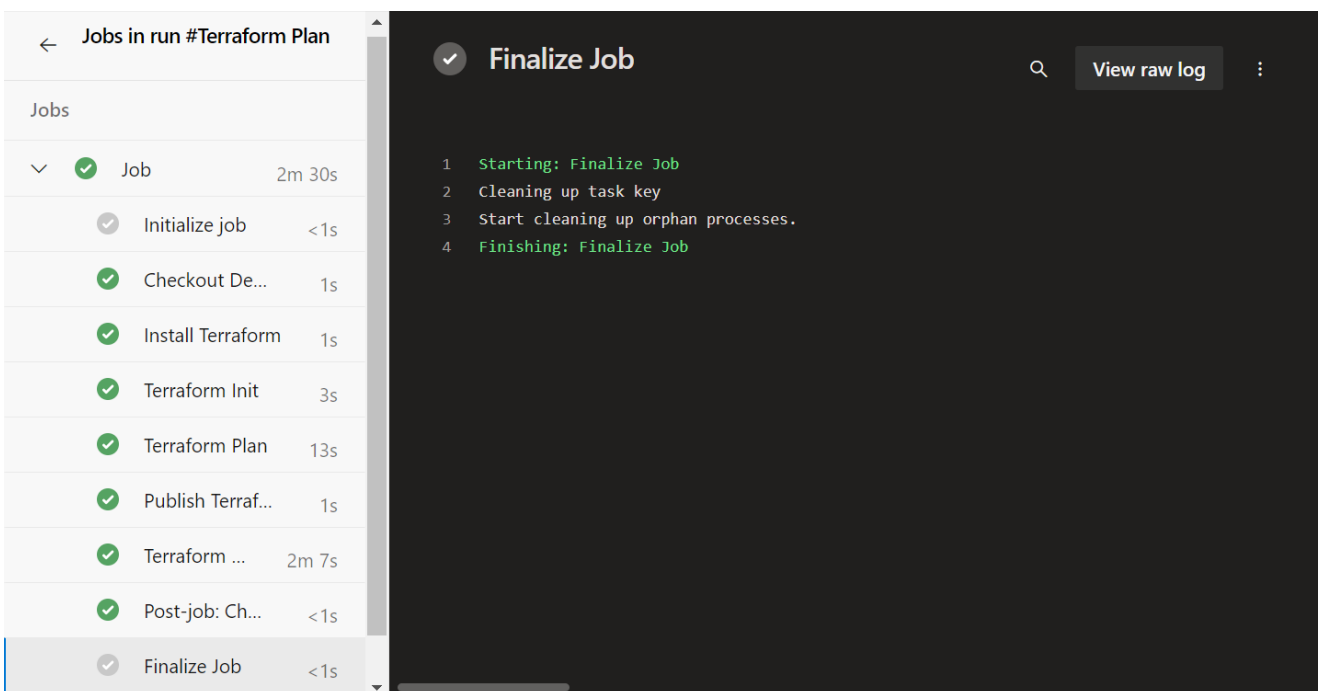
The screenshot shows the Terraform extension page. On the left is the Terraform logo. The main heading is "Terraform" by Microsoft DevLabs, with a link to microsoft.com. It shows 74,443 installs and a 4.5-star rating from 25 reviews. A green "Get it free" button is prominent. Below the button, it says "Install terraform and run terraform commands to manage resources on Azure, AWS and GCP."

Una vez hemos realizado toda la configuración anterior, deberíamos tener:

- La ejecución de trabajos paralelos habilitada.
- El repositorio de Azure Repos habilitado.
- Nuestro código en el repositorio de Azure Repos (**main.tf** y **variables**).
- Nuestra pipeline en dicho repositorio.
- Las extensiones necesarias añadidas al proyecto para la ejecución de la pipeline.

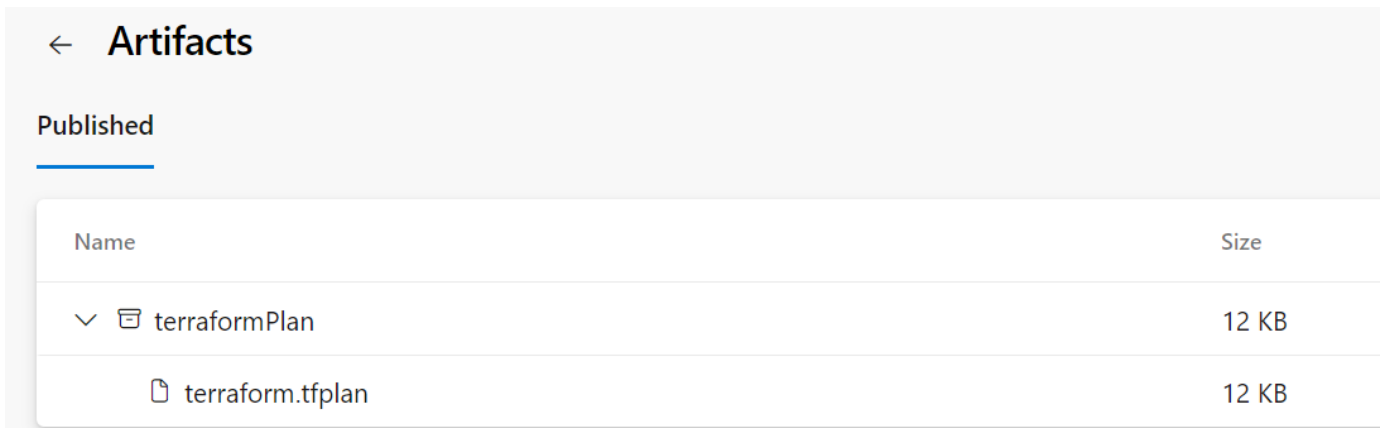
### 6.3.5 Ejecución de la pipeline.

Nos dirigimos a la sección pipelines y pulsamos en “Run” sobre la pipeline definida.



The screenshot shows the Azure DevOps pipeline execution interface. On the left, a list of jobs is shown under the heading "Jobs in run #Terraform Plan". The jobs are: Job (2m 30s), Initialize job (<1s), Checkout De... (1s), Install Terraform (1s), Terraform Init (3s), Terraform Plan (13s), Publish Terraf... (1s), Terraform ... (2m 7s), Post-job: Ch... (<1s), and Finalize Job (<1s). The "Finalize Job" job is selected, and its details are shown on the right. The details include a "View raw log" button and a list of steps: 1. Starting: Finalize Job, 2. Cleaning up task key, 3. Start cleaning up orphan processes., and 4. Finishing: Finalize Job.

Mediante la tarea “**PublishBuildArtifacts@1**” estamos generando un artefacto, que contendrá un fichero **terraform.tfplan** (Salida de Terraform plan). Este fichero está en formato binario y no se puede leer directamente como texto plano. Debemos utilizar comandos específicos de Terraform para mostrar y revisar su contenido de manera legible.



Como vemos la ejecución de la pipeline se ha completado correctamente, ahora realizaremos la comprobación de los recursos desplegados en Azure Portal.

The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar and navigation icons. Below the search bar, the page title is 'Todos los recursos' (All resources). There are several filter buttons: 'Suscripción es igual a todo', 'Grupo de recursos es igual a todo', 'Tipo es igual a todo', and 'Ubicación es igual a todo'. There are also buttons for 'Recursos no seguros' and 'Recomendaciones'. Below the filters, there is a table of resources with columns for 'Nombre', 'Tipo', 'Grupo de recursos', 'Ubicación', and 'Suscripción'. The table lists several resources, including network interfaces, security groups, disks, public IP addresses, virtual machines, and virtual networks, all located in West Europe and associated with the 'my-terraform-rg' resource group.

Nombre	Tipo	Grupo de recursos	Ubicación	Suscripción
my-terraform-nic	Interfaz de red	my-terraform-rg	West Europe	Suscripción de Azure 1
my-terraform-nsg	Grupo de seguridad de red	my-terraform-rg	West Europe	Suscripción de Azure 1
my-terraform-os-disk	Disco	MY-TERRAFORM-RG	West Europe	Suscripción de Azure 1
my-terraform-public-ip	Dirección IP pública	my-terraform-rg	West Europe	Suscripción de Azure 1
my-terraform-vm	Máquina virtual	my-terraform-rg	West Europe	Suscripción de Azure 1
my-terraform-vnet	Red virtual	my-terraform-rg	West Europe	Suscripción de Azure 1

Podemos observar como la infraestructura se ha desplegado correctamente.

En caso de realizar alguna modificación al repositorio (En la rama **main**), se volverá a ejecutar la pipeline gracias a la siguiente configuración del fichero YAML:

```
trigger:
  branches:
    include:
      - main
```

Además podrá saber el estado actual de la infraestructura gracias al fichero de estado que se ha generado en el contenedor que creamos para el backend, mediante el siguiente bloque de código (Dentro del **main.tf**):

```
terraform {
  backend "azurearm" {
    resource_group_name = "dev1"
    storage_account_name = "storageiesgn"
    container_name = "contenedoriesgn1"
    key = "terraform.state"
  }
}
```

Como vemos el fichero de estado se llamará **terraform.state**. Vamos a comprobar desde el portal de Azure que se haya generado dentro del contenedor (**blob**):

The screenshot shows the Azure portal interface. On the left, the 'contenedoriesgn1' container is selected, showing its configuration and a list of blobs. The 'terraform.state' blob is highlighted with a red box. On the right, the details for the 'terraform.state' blob are displayed, including its URL, creation time, and size.

Propiedad	Valor
URL	https://storageiesgn.blo...
ÚLTIMA MODIFICACIÓN	22/5/2023, 1:02:32 p. m.
HORA DE CREACIÓN	18/5/2023, 0:54:53 p. m.
ID. DE VERSIÓN	-
TIPO	Blob en bloques
TAMAÑO	18.86 KiB
NIVEL DE ACCESO	Frecuente (inferido)
ÚLTIMA MODIFICACIÓN DEL NIVEL DE ACCESO	N/D
ESTADO DEL ARCHIVO	-

Gracias a este fichero de estado, cuando volvamos a realizar un cambio en el código de la infraestructura, no se necesitará importar cada uno de los recursos al proyecto de Terraform del agente autohospedado por Microsoft (Ya que cada vez que se ejecute la pipeline, es un agente distinto).

Si tuviésemos configurado un agente autohospedado (Sabemos que no va a cambiar), no sería necesario (Aunque recomendable) el fichero de estado.

## 7. Conclusión

Resumiendo, este proyecto ha consistido en explicar los fundamentos básicos e implementar infraestructuras en Azure utilizando **Terraform** y **Azure DevOps Pipelines**. Esta integración permitió automatizar y optimizar el proceso de creación y gestión de recursos en la nube. Al aprovechar Terraform como herramienta de infraestructura como código y Azure DevOps Pipelines como plataforma de integración continua, se logró una mayor eficiencia y agilidad en el despliegue de infraestructuras. Esto se tradujo en una reducción de errores manuales, una mayor consistencia en los entornos y una colaboración más efectiva entre los equipos de desarrollo y operaciones.

Además, la combinación de **Azure** como proveedor de la nube y **Terraform** como herramienta de orquestación ofreció flexibilidad y escalabilidad. Los servicios y capacidades de Azure se pudieron aprovechar de manera sencilla, permitiendo adaptar las infraestructuras según las necesidades cambiantes del proyecto. Esta flexibilidad garantizó una infraestructura ágil y preparada para el crecimiento, proporcionando a las organizaciones una base sólida y adaptable para sus aplicaciones y servicios en la nube.

En conclusión, el proyecto de implementación de infraestructuras en Azure con Terraform y Azure DevOps Pipelines resultó en una mejora significativa en la eficiencia, la colaboración y la escalabilidad de las infraestructuras en la nube. La automatización y la integración continua permitieron agilizar el despliegue de recursos, reducir errores y mejorar la consistencia. Al aprovechar las capacidades y servicios de Azure, se crearon infraestructuras flexibles y preparadas para el crecimiento, brindando una base sólida para las aplicaciones y servicios en la nube.