s

Infraestructura 2

Pablo Leonel Cristaldo

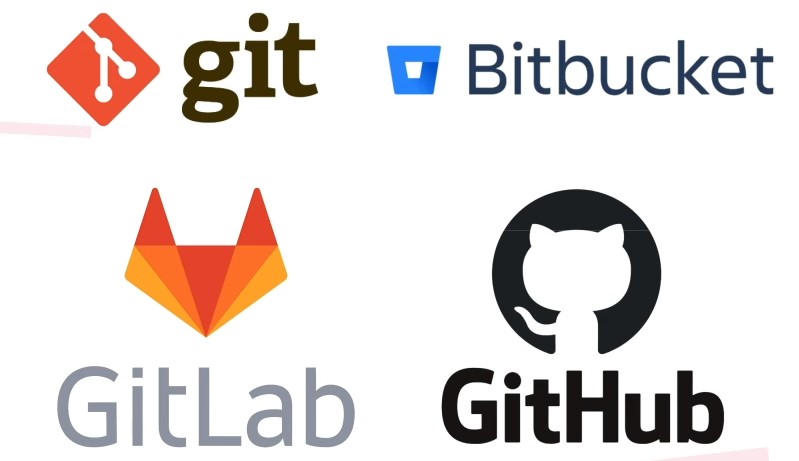
[Año]

**Herramientas de DevOps**

1. **Control de versiones**

Es la practica de lleva un registro y gestionar los cambios en el codigo fuente del software.

Las herramientas para llevar a cabo esta tarea, llevan el registro de los cambios a una base de datos, permiten deshacer el codigo y volver a una version anterior del mismo.



1. **Contenedores**

Empaqueta el codigo y todas sus dependencias, para que la aplicacion pueda trabajar de forma confiable en distintos entornos de computo.



1. **Orquestadores**

Se encargan del Despliegue, gestion, escalamiento, conectividad y disponibilidad de aplicaciones basadas en contenedores.

1. **Monitoreo de aplicación**

Se encarga de medir y monitorear la Performance, disponibilidad y la experiencia de usuario (UX) de una aplicación. Con esto podemos anticiparnos a los problemas que puedan impactar a los usuarios.



1. **Monitoreo de servidores**

Proceso de ganar visibilidad respecto a la actividad de los servidores sean fisicos o virtuales, puede enfocarse en diferentes metricas, pero las principales son la disponibilidad y la carga.



1. **Gestion de configuracion**

Proceso que lleva registro de las distintas configuraciones que un sistema adopta a lo largo de su vida.



1. **Integracion continua CI**

Practica que consiste en integrar el codigo a un repositorio compartido, de forma continua y de deseablemente multiples veces al dia.



1. **Despliegue continuo CD**

Habilidad d eponer en produccion, es decir, poner en manos de los usuarios cambios de cualquier tipo de manera segura y sostenible, asegurandonos de que el codigo este siempre en un estado desplegable, aunque se hagan cambios constantemente.

* Nuevas funcionalidades
* Cambios de configuracion
* Solucion de errores
* Experimentos

1. **Automatizacion de pruebas**

Software que hace uso de herramientas de automatizacion, para controlar la ejecucion de las pruebas. Los resultados de estas son comparados con los resultados esperados para constatarlas. Disminiyendo el error humano.



1. **Infraestructura como codigo**

Gestion de infraestructura como un modelo descriptivo, usando herramientas de versionado al igual que un equipo de versionado utiliza para su codigo fuente.

En conjunto con Pipeline de despliegue continuo nos permite automatizar despliegues, haciendolos mas rapidos y minimizando errores humanos.



1. **Computacion en la nube**

Uso de servicios de computo a traves de internet. Permite acelerar la innovacion mediante recursos flexibles y aprovechamiento de economias de escala. Reduce costos de operación.



**Perfiles en DevOps**

**Desarrollador de aplicaciones:** Son quienes desarrollan la aplicación, los programadores front-end, back-end, mobile, full stack o especializados en una tecnología particular —como Solidity—  o plataforma — por ejemplo, Internet de las cosas (IoT)—. En un entorno DevOps es importante que se comuniquen constantemente con los demás roles.

**Analista de calidad (QA):** Son quienes verifican y validan la aplicación. En un entorno DevOps es importante que también se concentren en automatizar pruebas para hacerlas repetibles y confiables.

**Analista de infraestructura:** Son quienes implementan la infraestructura sobre la cual se ejecutarán las aplicaciones y las bases de datos. También se ocupan del mantenimiento y la evolución de esta infraestructura. Buena parte de las prácticas de DevOps recaen sobre ellos, en especial la comunicación con quienes desarrollan la aplicación. Dado que muchas veces la infraestructura existe en la nube, también se los suele llamar analistas clouds o analistas de nube.

**Analistas de redes:** Se ocupan de la interconexión entre distintos sistemas, es decir, de las redes de computadoras —sean físicas o virtuales—. Es poco frecuente que se necesite una persona dedicada de forma exclusiva a las redes, es más común que este rol sea ocupado por el analista de infraestructura.

**Analistas de seguridad:** Son personas que trabajan en la seguridad de la aplicación y de la infraestructura. A veces no se dispone de un empleado por equipo dedicado de forma exclusiva a este rol. En esos casos es importante que todo el equipo reciba entrenamiento en seguridad.

**Analistas de CI/CD:** Son quienes mantienen los pipelines de integración y despliegue continuos. En aplicaciones simples es común que esta persona sea la misma que ocupa el rol de analista de infraestructura, pero en aplicaciones más complejas es necesario diferenciar roles.

**Arquitectos de nube:** Definen la arquitectura del entorno en la nube: la estructura que tendran los servidores, como se interconectan y varios aspectos de seguridad relacionados. Tambien definen quienes tendran acceso a los distintos entornos. En organizaciones pequeñas no hay una persona dedicada de forma exclusiva a esto y la funcion recae sobre el analista de infraestructura.

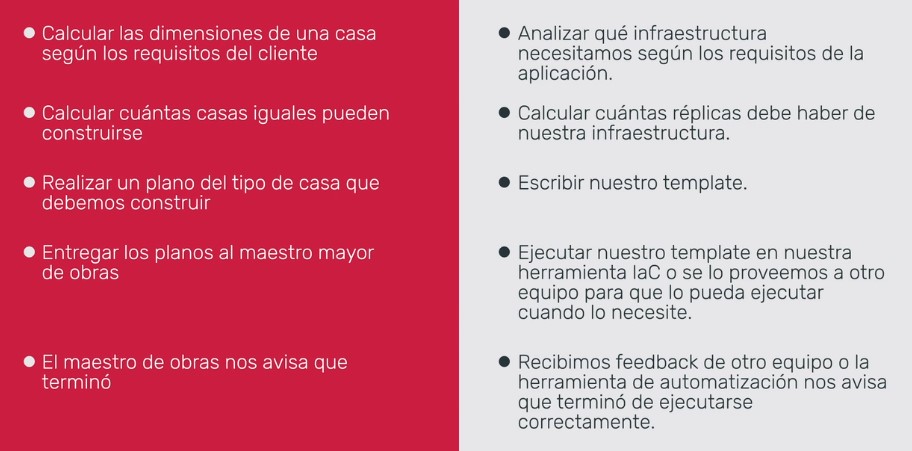
**Ingenieros de confiabilidad de sitio (SER):** Son los encargado de diseñar y monitorear el sistema par aminimizar las suspensiones de servicio y el tiempo de recuperacion de servicio. Trabaja tanto de forma proactiva como reactiva, respondiendo a incidentes, pero tambien intentando que no ocurran o vuelvan a ocurrir.

**Gerentes de entregas:** En algunos casos no es posible realizar despliegue continuo, por limitaciones del mercado o por la naturaleza del producto —por ejemplo, cada despliegue significa inevitablemente una suspensión temporal del servicio o cada cliente requiere una versión distinta del producto—. En estos casos, el gerente de entregas se ocupa de coordinar la entrega de nuevas versiones del producto a los clientes, llevar registro de qué cliente tiene qué versión del producto y orientar los esfuerzos del equipo hacia la satisfacción de los clientes.

**Infraestructura como codigo**

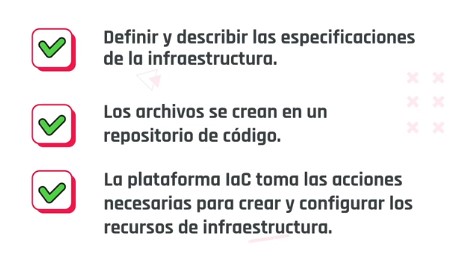
Nos permite gestionar la infra de nuestras aplicaciones, lo cual se realiza a traves de la automatizacion de la administracion y el aprovisionamiento de nuestra configuracion o del servidor cloud.

Podemos crear un servidor, aprovisionar una base de datos o crear un cluster para correr nuestros contenedores.



* Se crean archivos de configuracion, que tiene las especificaciones uqe esta necesita.
* El control de versiones se debe aplicar a cualquier archivo de codigo/software.
* Se divide en elementos modulares.

**El procedimiento de IaC implica tres pasos:**

****

¿Cómo iniciamos? ¿Cuáles son los primeros pasos en una implementación? ¿Qué mejoras nos ofrece la infraestructura como código?

Al momento de realizar los análisis para implementar una infraestructura para nuestra aplicación, lo primero a definir es la arquitectura que necesitamos: qué servidor es el adecuado o qué base de datos necesitamos. Una vez seleccionado el tipo de servidor, avanzamos en la configuración e instalación de aquello que nuestro sistema operativo requiere para estar operativo.

Paso a paso:

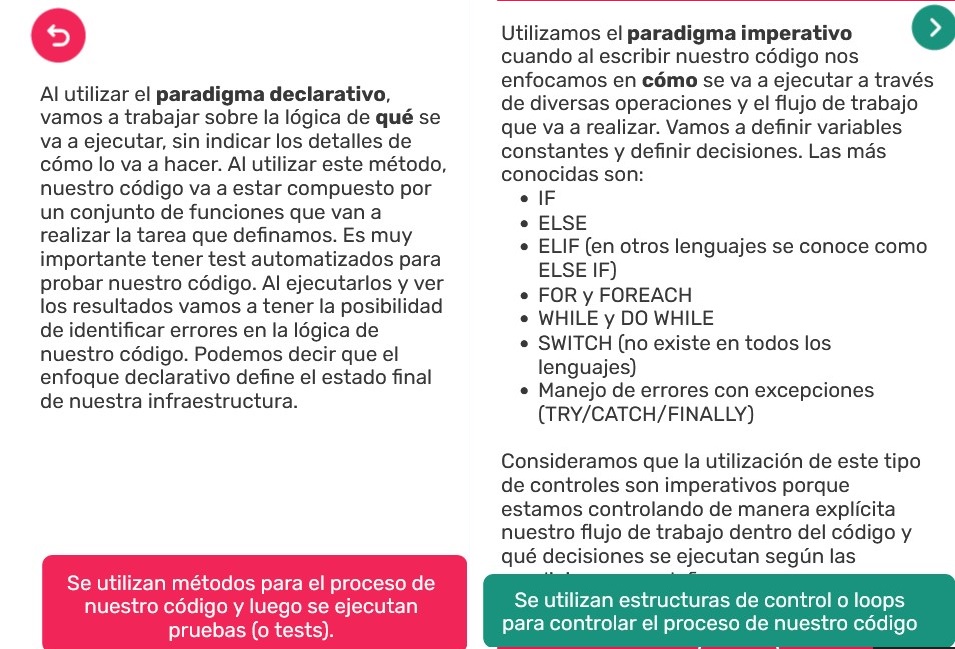
* Configurar la conexión de la comunicación de los servidores hacia nuestras computadoras, hacia Internet, o bien hacia otros servidores.
* Instalar las dependencias de la aplicación.
* Implementar la aplicación o base de datos que va a tener ese servidor.

Beneficios de la infraestructura como codigo:

1. Reduccion del error humano.
2. Repetibilidad y predictibilidad.
3. Tiempos y reduccion de desperdicios
4. Control de versiones.
5. Reduccion de costos.
6. Testeos.
7. Entornos estables y escalables.
8. Estandarizacion de la configuracion.
9. Documentacion.
10. Rapidez sin descuidar seguridad.

**Paradigma imperativo y paradigma declarativo**

Existen dos paradigmas de programación aplicados a la Infraestructura como código. Al escribir nuestro IaC podemos optar por el paradigma imperativo, que nos posibilita controlar el flujo de trabajo de nuestro código, o bien enfocarnos en el resultado final y en el cambio de nuestra infraestructura, el paradigma declarativo. Es el "cómo" versus el "qué".



**El principio de idempotencia**

La idempotencia es un principio matemático utilizado en infraestructura. Pero, ¿qué es? Podemos definir la idempotencia como la propiedad de utilizar una automatización “n” cantidad de veces obteniendo siempre y en todos los casos el mismo resultado. ¿Cómo se aplica la idempotencia a la infraestructura como código?

En matemática y lógica, la **idempotencia** es la propiedad para realizar una acción determinada varias veces y aun así conseguir el mismo resultado que se obtendría si se realizase una sola vez. Un elemento que cumple esta propiedad es un elemento **idempotente**, o un **idempotente**.

Posibilidad de implementar nuestro codigo, siendo siempre idempotente.

Utilizando Ansible, Terraform o cloudformation.

Origen->proceso->destino

Es facilmente documentable, en el archivo de configuracion esta toda la informacion.

Se pueden implementar practicas de desarrollo de software como versionar archivos de configuracion.

Puede ser totalmente automatizable.

**Ecosistema**

Al automatizar nuestra infraestructura, es probable que utilicemos distintos proveedores o que usemos una parte cloud y otra parte on-premise (un datacenter propio). Existen herramientas que poseen su propia sintaxis (en general, JSON y YAML) para poder administrar la infraestructura en múltiples proveedores o en uno solo, pero de una manera más eficiente.

¿Por qué hacemos estas distinciones? ¡Porque podemos elegir para nuestra infraestructura la herramienta que nos brinda la mayor eficiencia posible! Vamos a explorar una selección de las tecnologías más populares. Contar con esta información nos ayudará a elegir la que mejor se adapte a nuestras necesidades.

**Terraform**

Terraform es un software de código libre desarrollado por HashiCrop. Es una herramienta declarativa de aprovisionamiento y orquestación de infraestructura que permite automatizar el aprovisionamiento de todos los aspectos de la infraestructura, tanto para la nube como la infraestructura on-premise (en los mismos datacenter). Tiene algunas características interesantes, como comprobar el estado de la infraestructura antes de aplicar los cambios. Es la herramienta más popular porque es compatible con todos los proveedores de nube sin realizar modificaciones en nuestros templates.

**AWS CLOUDFORMATION**

AWS CloudFormation es la solución nativa de AWS para aprovisionar recursos en esta nube. En este caso se pueden definir templates en formato JSON o YAML. Se pueden utilizar para crear, actualizar y eliminar recursos las veces que sea necesario. Una ventaja de CloudFormation es que, al ser un servicio propio de Amazon, tiene una integración completa con los demás servicios de AWS, por lo que es nuestra mejor opción si solo utilizamos este proveedor de nube.

**AZURE RESOURCE MANAGER**

ARM es la herramienta nativa en Azure para implementar infraestructura como código, Azure Resource Manager (ARM Templates). Estas plantillas llevan una sintaxis declarativa en formato JSON, que nos permiten definir los recursos y las propiedades que conforman la infraestructura.

**GOOGLE CLOUD DEPLOYMENT MANAGER**

Google Cloud Deployment Manager es la herramienta IaC para la plataforma Google Cloud —lo mismo que CloudFormation es para AWS—. Con esta herramienta los usuarios de Google pueden administrar fácilmente mediante archivos de configuración YAML.

**ANSIBLE**

Ansible es una herramienta de automatización de infraestructuras creada por Red Hat. Ansible modela nuestra infraestructura describiendo cómo se relacionan sus componentes y el sistema entre sí, en lugar de gestionar los sistemas de forma independiente.

**CloudFormation**

CloudFormation es una herramienta nativa de Amazon Web Services (más conocido como AWS). Pero… ¿por qué es una herramienta tan popular? Porque nos brinda la posibilidad de implementar prácticas de infraestructura como código (IaC) de forma nativa dentro de AWS.

CloudFormation crea y configura la infraestructura que definimos previamente en una plantilla (o template) de acuerdo a los requisitos que necesitamos. Esto nos ofrece algunas ventajas, como crear repositorios con nuestros templates para que sean accesibles o que se puedan realizar entregas rápidas de los recursos de infraestructura.

Crea y configura la infraestructura previamente escrita en una planilla o template, con los requisitos que necesitamos. Podemos:

* Crear repositorios con los templates para ser accecsibles
* Permite entregas rapidas de los recursos de infraestructura

Los archivos templates pueden guardarse en varias extensiones, (JSON, TEXT, YML, Template). Son estos los usados por CloudFormation.

Dentro de las caracteristicas mas importantes tenemos:

* Templates parametrizables: levantar diferentes servicios cambiando parametros.
* Automatizable: se pueden usar los templates dentro de pipelines.
* Rollbacks: posibilidad de volver atrás si se descubre un problema que lo amerite.
* Gratuito.

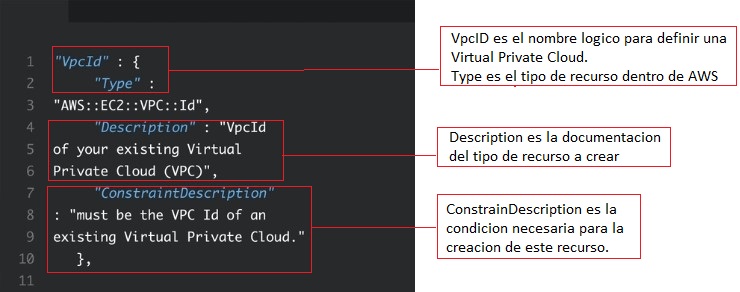
Es una herramienta integral que nos permite administrar nuestra infraestructura en un entorno controlado y predecible, agilizando los tiempos de entrega y la administracion de recursos.

Para introducirnos en el funcionamiento de la herramienta, vamos a tener en cuenta tres conceptos importantes:

* Plantillas o templates: es un archivo de texto con formato JSON (JavaScript Object Notation) que describe los recursos que queremos crear junto a sus propiedades.
* Pilas: es una unidad que genera CloudFormation para la creación ordenada de los recursos.
* Cambios: es un resumen de los cambios que se proponen para anticiparnos al resultado final.

**Parametros**

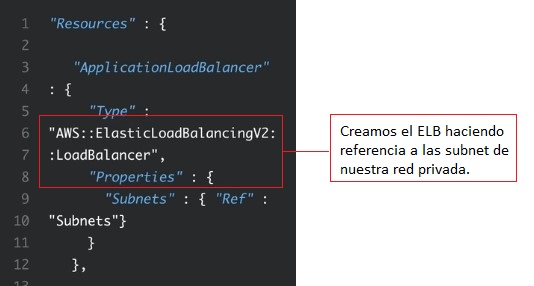
Se componene de un tipo logico y tres componentes asociados a ese tipo de recurso.



**Mapping**: Clase de mapeo de cadenas literales, que puede recibir un dato de tipo String o list en el formato clave valor. En el ejemplo mapeamos la creacion de las diferentes instancias S2 a un tipo de AMI “HVM64”



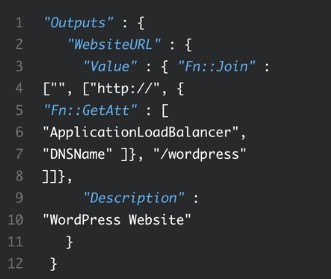
**Creacion de recursos:** Creamos los recursos necesarios como por ejemplo el balanceador de carga para tener distintas replicas de nuestra app y de la DB.



En resources tambien podemos crear nuestros grupos de seguridad para resguardar la aplicación y la DB.



**Outputs:** En outputs, podemos declarar el mensaje final que va a visualizar el ejecutor del template. Al estar automatizado, genera la ruta a traves de los recursos que fue creando.





**¿Dónde la usamos?**

* Podemos hacerlo por línea de comando desde nuestros equipos.
* En scripts (como PowerShell).
* En pipelines, como parte de un conjunto de tareas automatizadas y encadenadas entre sí, formando una tubería con un inicio y un fin.



**Ansible**

**¿De qué se trata Ansible?**

Ansible es un proyecto comunitario *open source* diseñado para ayudar a las organizaciones a automatizar el aprovisionamiento de infraestructura, la gestión de configuración y el despliegue de aplicaciones. Asimismo, es importante tener en cuenta que es fácil de aprender.

Con Ansible se crean archivos de configuración llamados *playbooks*, escritos en YML, que se utilizan para especificar el estado requerido de la infraestructura. Al ejecutarlos, Ansible se ocupa de aprovisionar la infraestructura necesaria para alcanzar el estado descrito.

Esto quiere decir que se puede, por ejemplo, crear una máquina virtual en el proveedor de infraestructura -como una instancia EC2 dentro de AWS- aplicando metodologías de infraestructura como código.

**Agentless:**

* Trabaja a traves SSH
* **Yml Playbooks**
* WinRM y PowerShell

Objetivos:

* Minimalista por naturaleza: no deberia imponer dependencias adicionales al entorno
* Consistencia: puede crear entornos consistentes.
* Saeguridad: no instala agentes en nodos, solo requiere que estos tengan instalados Open SSH y python.
* Confiabilidad: un playbook deberia ser idempotente para evitar efectos inesperados en los sistemas a gestionar
* Minimo aprendizaje requerido: lenguaje simple y descriptivo en YAML

Podemos parametrizar templates con Jinja. Como bucles o desiciones logicas.

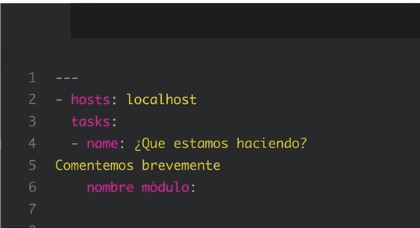
Ansible es una herramienta que permite gestionar las configuraciones de tu infraestructura. Sus principales ventajas al momento de usarla son:

* No necesita instalación de agentes.
* Su configuración es de fácil lectura.
* Es muy flexible (usa APIs y *plugins*).
* Es fácil de usar por basarse en YML.

**Para Ejecutar:**

Ansible-playbook “archivo con extension yml”

Es legible, tiene extension menor ya que la mayoria del codigo, son llamados a otros codigos publicados por la comunidad.



Donde ejecutaremos el playbook.

Host: puede ejecutarse remotamente asi que solo hay que escribir local host.  
Task: palabra reservada para ejecutar las tareas  
name: indicamos que vamos a ejecutar, documentamos.  
nombre modulo: el nombre del modulo. Los mas usados son ( aws\_s3 || ec2 || ec2\_ami || ec2\_elb || ec2\_tag || ecs\_ecr)



Despues vienen los parametros para el modulo ec2 o el modulo que estemos buscando.

Region: donde trabajaremos

Instance\_type: instancia que usaremos con diferentes caracteristicas, como la ram y y cpu asignada.

Image:referencia a un template interno de aws, tenemos que checkear en aws console y guardar ese valor para futuros despliegues para saber.

Wait: es la orden para esperar a que termine la ejecucion y despues nos de el OK. Se recomienda como yes.

Wait\_timeout: tiempo maximo de espera.

Volumes: parametro que usamos si le asignamos un disco a la instancia.

Vpc\_subnet\_id: parametro que necesitar estar creado previuamente y lo podemos averiguar en el servicio VPC en la seccion subnet.

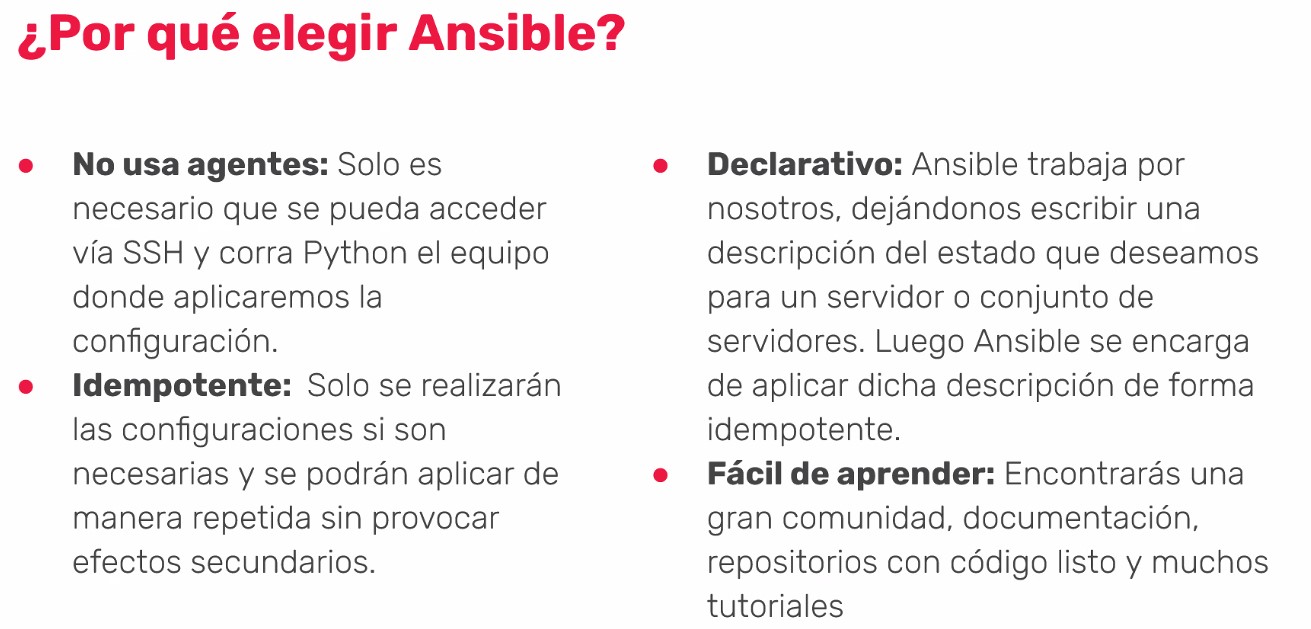
Assign\_public\_ic: para asignarle una ip publica a la instancia.

**La flexibilidad de Ansible permite hacerlo en diferentes espacios:**

* En tu computadora.
* En un servidor que pueda ser usado para ejecutar Ansible.
* En el proyecto de código abierto AWX que podés instalar y usar para administrar tus playbooks.

Esta última opción te brinda una gran ventaja: podés administrar tus automatizaciones y delegar a áreas operativas sin conocimientos en Ansible para que puedan ejecutar tus playbooks según sea necesario.





**Terraform**

**¿De qué se trata Terraform?**

Terraform es una una herramienta de código abierto desarrollada por HashiCorp y la última de las tres que vas a conocer en las clases de Infraestructura como código.

Esta herramienta te permite definir y aprovisionar la infraestructura completa utilizando un lenguaje declarativo que podés ejecutar como infraestructura como código.

¡Un dato a tener en cuenta! Si bien es similar a herramientas como CloudFormation, hay una gran diferencia: no es solo para AWS, sino que podés utilizarla con el resto de los proveedores de infraestructura cloud.

¡Otro dato más! Al ser declarativo te permite escribir tu código en el lenguaje de alto nivel HCL (HashiCorp Configuration Language) para describir el estado final que deseas de tu infraestructura.

* Declarativo
* Hcl lenguaje
* Elaborar un plan para alcanzar estado final para suministrar infraestructura
* Es multi plataforma (independiente)
* On premise
* Mejor solucion.
* Velocidad rapida para automatizacion y codigo declarativo
* Confianza ya que reduce la posibilidad de errores al crear grandes volumenes.
* El codigo es adaptable modificacndo pequeñas cosas.
* Escalable si la infraestructura es aprobada por todos los equipos de desarrollo
* Comunicación entre modulos (reutilizable) modulos hijos.

¿Cómo funciona?

* La sintaxis de los archivos de configuración: Es declarativa.
* De qué manera se ejecutan.
* La flexibilidad del código.
* El tipo de extensión de los archivos es .TF , por ejemplo una instancia EC2 probablemente se llame EC2.TF.

Para inicializar terraform.

**Terraform init**

Para lanzar el template

Se pueden hacer apply haciendo un detalle de lo que va a crear en tiempo real.

**Terraform apply**

