

**BONUS TRACK!**			
	CloudFormation	Terraform	Ansible
Arquitectura Simple y Fácil de Uso + Agentless	-	-	+
Facilidad para aprender + Gran Documentación en webOficial	+	-	-
Aprovisionamiento de Infraestructura	+	+	- Concebida inicialmente como herramienta de gestión de configuraciones masivas en servidores.
Servidores y Aplicaciones	-	- Solo podríamos crear el servidor	+ Permite crear tanto el servidor como servicios y aplicaciones.
Modularidad	-	+ Puede dividir código en diferentes archivos y bloques.	-
Popularidad y Soporte + OpenSource -> Facilidad de migración	-	+ Fácilmente adaptables y actualización constante.	+ Fácilmente adaptables y actualización constante.
Solidez y Madurez	+ Creada antes que Terraform (mayores actualizaciones y versiones).	-	+ Creada antes que Terraform (mayores actualizaciones y versiones).
Ejecución desde cualquier plataforma	- Solo funciona en Amazon Web Services.	+ Puede ejecutarse desde cualquier S.Operativo -> Archivo binario para cualquier plataforma.	- No funciona nativamente en Windows, necesita capa intermedia.

cuadro compartido por angie

## ¡Iniciamos!

Seguimos andando el camino de Infraestructura II. Iniciamos el segundo módulo Infraestructura como código: La disciplina.

En esta clase, vamos a:

- Recorrer el concepto de infraestructura como código (IaC).
- Detectar qué problemas resuelve.
- Distinguir cuáles son las herramientas más comunes para los analistas de infraestructura.

### Infraestructura como código, el concepto

- Empleamos el término Infraestructura como Código (IaC, por sus siglas en inglés) para referirnos a la gestión de la infraestructura a través de templates que tienen la capacidad de ser versionados. De esta forma vamos a poder automatizar los procesos manuales que se requieren para lograr el objetivo final que buscamos.

Así como cada vez que ejecutamos el código de nuestra aplicación obtenemos el mismo resultado, lo mismo ocurre con la infraestructura: vamos a obtener el mismo resultado de infraestructura desde nuestra IaC. Este concepto es muy importante para implementar la metodología DevOps en nuestra vida profesional, ya que nos permitirá crear un ambiente mucho más rápido y seguro para nuestras aplicaciones.

//video

La infraestructura como código nos permite gestionar la infraestructura de nuestras aplicaciones lo cual se realiza a través de automatización de la administración y del aprovisionamiento de la configuración del hardware físico o de los servicios de nuestro proveedor cloud, de esta manera vamos a poder automatizar todas las tareas manuales que se requieren para gestionar y preparar nuestra infra. tales como la creación de nuestro servidor para alojar nuestra aplicación, el aprovisionamiento de una BD de cualquier tipo de motor y la creación de un cluster para correr nuestros contenedores.

-debemos analizar qué infraestructura necesitamos según los requisitos de la aplicación.

-calcular cuantas replicas debe tener nuestra infraestructura

-para después escribir nuestro template

-ejecutar el template en nuestra herramienta IaC o se lo proveemos a otro equipo para que lo pueda ejecutar cuando lo necesite.

-recibiremos feedback de ese equipo o la herramienta de automatización nos avisa que terminó de ejecutarse correctamente.

En icc (infraestructura como código) se crean **archivos de configuración** que contienen las especificaciones que ésta necesita, lo cual facilita la edición y distribución. Así mismo garantiza que siempre se prepare el mismo entorno.

el **control de versiones** tamb es un aspecto importante de la icc que debería aplicar a los archivos de configuración al igual que a cualquier otro archivo de código fuente de softw.

ademas la implementacion de la icc significa que puede dividirse en elementos modulares que se combinaran de distintas maneras mediante la automatización. por lo general el procedimiento de la icc implica los siguientes tres pasos:

1-definir y describir las especificaciones de la infraestructura.

2-los archivos que se crean se envian a un repositorio de código

3-la plataforma de icc (iac) toma las acciones necesarias para crear y configurar los recursos de infraestructura.

ej. la creación de una máquina virtual en aws  
este tipo de recursos se conoce como EC2:

de manera manual tenemos qué:

1- entrar a la consola de aws

2-elegir el tipo de servicio, en este caso ec2

3- elegir la región donde queremos trabajar

4- elegir el tipo de instancia ec2 que queremos implementar

5-seleccionar los recursos que le dan contexto a este servicio (redes, grupo de seguridad, capacidad de almacenamiento y etiquetas)

6 - lanzar el servidor.

la iac es una práctica esencial de devops, indispensable para un ciclo de vida de entrega del software a un ritmo competitivo, permite al equipo crear y versionar rápidamente la infraestructura de la misma manera en que versionan el código fuente y hacer un seguimiento de estas versiones para evitar inconsistencias que puedan conducir a graves problemas durante el despliegue.

///

¿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.

Mejoras y soluciones

Aunque este proceso se encuentre bien documentado, siempre vamos a encontrar obstáculos: el tiempo que nos ocupa la tarea, posibles incompatibilidades, o inconsistencias en la documentación. La metodología de Infraestructura como código nos ofrece una solución a todo esto.

Este proceso que se realiza manualmente comienza a agilizarse ante el vertiginoso avance tecnológico, dando lugar a la automatización de la infraestructura. Como consecuencia, los ciclos de desarrollo poseen mejores tiempos de respuesta ante la mayor disponibilidad y flexibilidad de los ambientes de desarrollo de software.

La Infraestructura como código irrumpió para aumentar la calidad y la eficiencia de los equipos de desarrollo.

### Beneficios de la Infraestructura como código

A continuación, les compartimos diez beneficios de la **Infraestructura como código**.

#### 1. Reducción del error humano

Minimizamos el riesgo de equivocarnos cuando seguimos una serie de pasos. Mediante procedimientos claros y ordenados podemos evitar guardar una mala configuración o borrar algo que no debíamos. Esto va a aumentar la confianza que tengan en la infraestructura que brindemos.

## **2. Repetibilidad y predictibilidad**

Cuando sabemos que el contexto de nuestra aplicación funciona, vamos a poder repetir la cantidad de pasos que sean necesarios y ser capaces de predecir el resultado, ya que siempre será el mismo. Esto nos da —como resultado— una infraestructura más testeable y estable.

## **3. Tiempos y reducción de desperdicios**

El encargado de ejecutar nuestra infraestructura va a poder hacerlo en cuestión de minutos y sin necesidad de instalar algún componente extra. ¡Siempre vamos a poder activar el proceso para que haga lo suyo!

Al ejecutar las tareas más rápido, vamos a poder ayudar a otros equipos a que también trabajen en menos tiempo y de manera más eficiente

## **4. Control de versiones**

Nuestra infraestructura se va a encontrar definida en archivos, por lo que vamos a poder versionar —al igual que el código fuente de nuestra aplicación— en templates o plantillas. ¡Saquemosle el jugo a los templates! Podemos utilizar parámetros para escribir nuestro código de la manera más genérica posible. Luego, al ejecutarlo, vamos a poder enviarle datos distintos en forma de parámetros para que nuestro código los reciba.

## **5. Reducción de costos**

Al automatizar procesos, podemos enfocarnos en otras tareas y mejorar lo ya hecho. Esto aporta a la flexibilidad de los equipos de infraestructura para abarcar más tareas.

## **6. Testeos**

La infraestructura como código permite que los equipos de infraestructura puedan realizar pruebas de las aplicaciones en cualquier entorno (incluso producción) al principio del ciclo de desarrollo.

## **7. Entornos estables y escalables**

Al evitar configuraciones manuales, la falta de dependencias y al obtener el estado final de infraestructura que necesitamos para nuestras aplicaciones, vamos a ofrecer entornos estables y escalables.

## **8. Estandarización de la configuración**

Estandarizar las configuraciones y el despliegue de la infraestructura nos permite evitar cualquier problema de incompatibilidad con nuestra infraestructura y que las aplicaciones se ejecuten con el mejor rendimiento posible.

## **9. Documentación**

Al aportar a la documentación de los procesos internos de nuestros equipos vamos a mejorar tiempos y costos. Como ya vimos, podemos versionar nuestras automatizaciones. Esta característica nos permite que cada cambio se encuentre documentado, registrado por usuario y con una vuelta atrás (rollback) rápida si encontramos errores en los despliegues, al igual que el código fuente.

## **10. Más rapidez sin descuidar la seguridad**

Al momento de mejorar nuestra infraestructura, nunca hay que dejar de pensar en la seguridad que la compone. Al momento de estandarizar la ejecución de la infraestructura, también podemos estandarizar los grupos de seguridad con los permisos mínimos, pero necesarios para que todos los equipos puedan trabajar y evitar tareas manuales por parte de los equipos de seguridad.

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".

## Paradigma imperativo



Utilizamos el **paradigma imperativo**

cuando al escribir nuestro código nos enfocamos en **cómo** se va a ejecutar a través de diversas operaciones y el flujo de trabajo que va a realizar. Vamos a definir variables constantes y definir decisiones. Las más conocidas son:

- IF
- ELSE
- ELIF (en otros lenguajes se conoce como ELSE IF)
- FOR y FOREACH
- WHILE y DO WHILE
- SWITCH (no existe en todos los lenguajes)
- Manejo de errores con excepciones (TRY/CATCH/FINALLY)

Consideramos que la utilización de este tipo de controles son imperativos porque estamos controlando de manera explícita nuestro flujo de trabajo dentro del código y qué decisiones se ejecutan según las condiciones que definamos.

Se utilizan estructuras de control o loops para controlar el proceso de nuestro código

## Paradigma declarativo



Al utilizar el **paradigma declarativo**, vamos a trabajar sobre la lógica de **qué** se va a ejecutar, sin indicar los detalles de cómo lo va a hacer. Al utilizar este método, nuestro código va a estar compuesto por un conjunto de funciones que van a realizar la tarea que definamos. Es muy importante tener test automatizados para

probar nuestro código. Al ejecutarlos y ver los resultados vamos a tener la posibilidad de identificar errores en la lógica de nuestro código. Podemos decir que el enfoque declarativo define el estado final de nuestra infraestructura.

Se utilizan métodos para el proceso de nuestro código y luego se ejecutan pruebas (o tests).

## 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?

///video///

Esta propiedad de obtener siempre un mismo resultado sin importar las veces que ejecutemos una operación es lo que se denomina principio de idempotencia.

Dentro del ámbito de la IAC, es la propiedad de poder ejecutar nuestro código las veces necesarios obteniendo siempre el mismo resultado.

No importa las veces que ejecutemos el código, siempre será el mismo

para que la IAC sea idempotente utilizaremos herramientas para ese fin como puede ser ansible, terraform o cloud formation.

Si separamos la IAC como código en etapas, serían las siguientes:

**Origen:** es el archivo de configuración (un archivo json o yaml)

**Proceso,** son las operaciones que realizan las herramientas de IAC en base a nuestros archivos de origen.

**destino:** es el estado final de nuestra IAC tal como la necesitamos.

Al aplicar el principio de idempotencia, en estas tres etapas tendremos distintos beneficios, por ejemplo:

-si tenemos que cambiar algo de la infraestructura solo cambiaremos el archivo de origen, y luego ejecutaremos el proceso de ese archivo de config con la herramienta que estemos usando y obtendremos el estado final según las modificaciones que apliquemos.

-es fácilmente documentable ya que no tenemos que revisar nuestra IAC para revisar cómo está compuesta o documentar todo después de hacerlo.

-tenemos nuestro archivo de config con toda la información, y si es necesario solo hay que agregar documentación complementaria.

-se pueden aplicar prácticas de desarrollo de software en el área de IAC, como por ejemplo versionar archivos de config. para volver a una versión anterior y sin trabajo extra y es fácilmente compatible con el resto del equipo.

Todo esto nos permite que el proceso sea totalmente automatizado.

Conocer el principio de idempotencia y sus beneficios es fundamental, ya que es la base teórica de la automatización que permite la infraestructura como código.

///fin de video///

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 HashiCorp. 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 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 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 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.

-----

## ¡Revisemos lo aprendido!

**Completar los recuadros con los términos que correspondan.**

La gestión de Infraestructura como Código, es un concepto inspirado por la metodología DEVOPS..

Dentro de los beneficios de implementar IaC, está la mitigación de RIESGOS por error humano, ya que evitamos tareas...MANUALES.

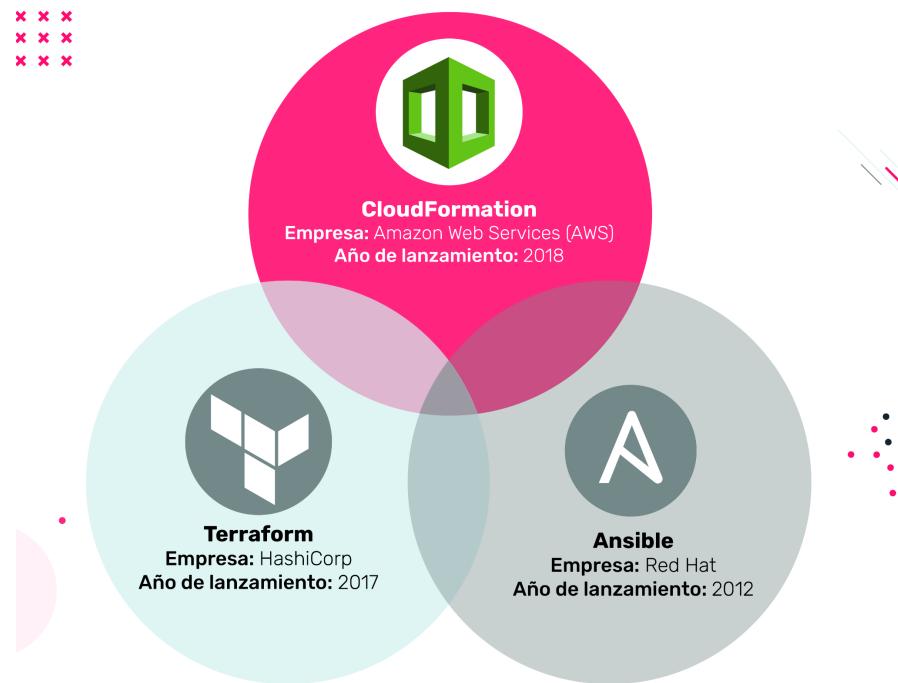
Al momento de escribir el código de nuestra infraestructura, podemos elegir el paradigma imperativo o DECLARATIVO.

...TERRAFORM es la herramienta más popular, dentro de las que son compatibles con múltiples nubes.

-----  
¡Comenzamos!

En las próximas clases vamos a profundizar en las herramientas más populares en el ámbito de la infraestructura como código (o IaC, de sus siglas en inglés: Infrastructure as Code). ¿Cómo se definen? ¿Cómo las usamos? Estas son solo algunas de las preguntas que vamos a explorar. Comenzamos con CloudFormation.

## Las herramientas más populares de infraestructura como código



¿De qué se trata 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.

A lo largo de la clase, vamos a aprender a usar la herramienta, en qué entornos podemos hacerlo y qué roles ejecutan los templates creados para CloudFormation. ¡Arranquemos!

## Descubriendo CloudFormation

¿Por qué surgió CloudFormation? ¿Qué problemas esperaba resolver? Vamos a descubrir más a fondo la herramienta nativa de AWS y de qué elementos se compone

//video

Una de las herramientas mas conocidas y usadas en el mercado para implementar icc es Cloud formation, la herramienta nativa de aws.

CF nos brinda la posibilidad de implementar prácticas de icc de forma nativa dentro de aws. esta herramienta se va a encargar de crear y configurar la infra que definimos previamente en una planilla o template con los requisitos que necesitamos. CF nos proporciona algunas ventajas como crear **repositorios** con nuestros templates, o que se puedan realizar **entregas rápidas** de los recursos de infra.

Lanzada en 2018, esta herramienta permite aprovisionar la infra necesaria en base a dichos templates que son reutilizables y parametrizables escritos en archivos en formato json o yaml. Estos archivos se pueden guardar en cualquier extensión, como .template .txt. json o .jaml. y son estos los que utiliza cf para crear los recursos de infraestructura.

por ej en una plantilla se puede definir la creación de una instancia amazon EC2 con todas sus características, dentro de las particualres mas notables de CF se encuentran:

**templates parametrizables** del que se pueden levantar distintos servicios únicamente cambiando los parametros, permitiendo que sean reutilizables

**automatizacion:** ya que se pueden usar los templates entre pipelines

permite el uso de **rollbacks**, que nos da automáticamente una vuelta para atrás destruyendo lo que se haya producido en el caso de que falle alguno de los pasos de creacion de recurso de infra.

**precio:** el uso de CF es totalmetne gratuito.

CF es una herramienta integral que nos permite administrar nustra infra en un entorno controlado y predecible produciendo los tiempos de despliegue agilizando asi la entrega de recursos.

## ¿Cómo usarla?

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.

A continuación, vamos a desarrollar estos tres conceptos y ver cómo se relacionan entre sí para que podamos utilizar la herramienta CloudFormation de manera exitosa.

//video

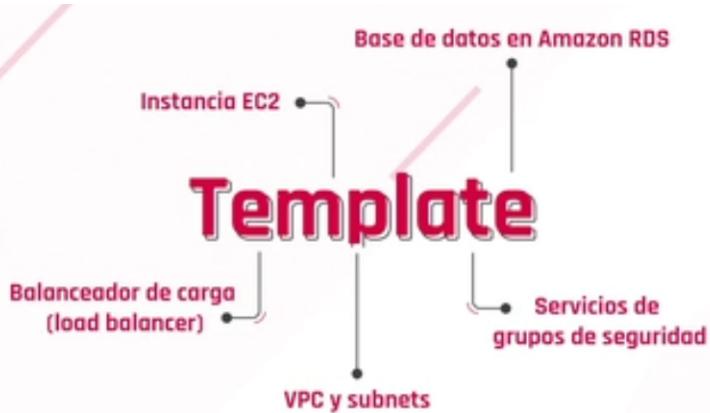
Un template o plantilla es la definición de nuestra infraestructura

```

1  {
2      "AWSTemplateFormatVersion"
3  : "2010-09-09",
4      "Description" : "Mi primer template",
5      "Parameters" : { },
6      "Mappings" : { },
7      "Resources" : { },
8      "Outputs" : { }
9  }
10
11

```

para el ejemplo, nuestro template va a tener:



primero creamos la descripción para que quienes lo reciban sepan a grandes rasgos qué componentes posee.

```

1  {
2      "AWSTemplateFormatVersion"
3  : "2010-09-09",
4      "Description" : "AWS
5      CloudFormation Sample Template
6      WordPress_Multi_AZ: WordPress is web software
7      you can use to create a beautiful website
8      or blog. This template installs a
9      highly-available, scalable WordPress
10     deployment using a multi-az Amazon RDS
11     database instance for storage.
12     It demonstrates using the
13     AWS CloudFormation bootstrap scripts to
14     deploy WordPress. **WARNING** This template
15     creates an Amazon EC2 instance, an
16     Application Load Balancer and an Amazon RDS
17     database instance. You will be billed for
     the AWS resources used if you create a stack
     from this template.",

```

Seguimos por los parámetros, éstos reciben nombres lógicos o IDs que presentan recursos. Los parámetros se componen de un tipo lógico y de tres componentes asociados a ese tipo de recurso.



Tomemos vpc como ejemplo:

```

1 "VpcId" : {
2   "Type" :
3     "AWS::EC2::VPC::Id",
4     "Description" : "VpcId
5       of your existing Virtual
6       Private Cloud (VPC)",
7     "ConstraintDescription"
8   : "must be the VPC Id of an
9       existing Virtual Private Cloud."
10  },
11
12
  
```

VpcId es el nombre logico para definir una Virtual Private Cloud, donde vamos a encontrar tres componentes asociados:

- 1- **type** que es el tipo de recurso para aws, en este caso una vpc para una instancia ec2
- 2- **description** tipo de recurso a crear
- 3- **ConstraintDescription** condicion necesaria para que se pueda utilizar este recurso.

Continuamos con mappings

```

1 "Mappings" : {
2   "AWSInstanceType2Arch" : {
3     "t1.micro" : { "Arch"
4       : "HVM64" },
5     "t2.nano" : { "Arch"
6       : "HVM64" },
7
  
```

una clase de mapeo de cadenas literales que pueden recibir como tipo de datos un string o un list con el formato clave / valor

en este ej vamos a mapear distintas instancias ec2 a un tipo de *ami* que es la HVM64 .

Luego crearemos los distintos recursos que necesitamos como el balanceador de carga para tener distintas réplicas de nuestra aplicación y de nuestra bd. en este ejemplo crearemos el recurso E L B. ElasticLoadBalancer, haciendo referencia a la subnet de nuestra red privada. en resources vamos a poder crear nuestros grupos de seguridad para resguardar la aplicacion y la base de datos.

```

1 "Resources" : {
2
3     "ApplicationLoadBalancer"
4 : {
5         "Type" :
6 "AWS::ElasticLoadBalancingV2::
7 :LoadBalancer",
8         "Properties" : {
9             "Subnets" : { "Ref" :
10 "Subnets"}
11         }
12     },
13 }
```

Para finalizar nuestro template, usamos outputs

```

1 "Outputs" : {
2     "WebsiteURL" : {
3         "Value" : { "Fn::Join" :
4 ["", ["http://", {
5 "Fn::GetAtt" : [
6 "ApplicationLoadBalancer",
7 "DNSName" ]}], "/wordpress"
8 ]]}
9         "Description" :
10 "WordPress Website"
11     }
12 }
13 }
```

con el objetivo de declarar cuál va a ser el mensaje final que va a visualizar el ejecutor del template. En este ejemplo va a visualizar la ruta expuesta a internet de nuestra aplicación. como esta automatizado genera la ruta a través de los nombres de los distintos recursos que fue creando.

-ejemplo

```
! 01-ec2.yml ×
! 01-ec2.yml > {} Resources > {} MyInstance > {} Properties
1   Resources:
2     MyInstance:
3       Type: AWS::EC2::Instance
4       Properties:
5         AvailabilityZone: |
6           ImageId:
7           InstanceType:
```

veamos de donde obtenemos estos datos:

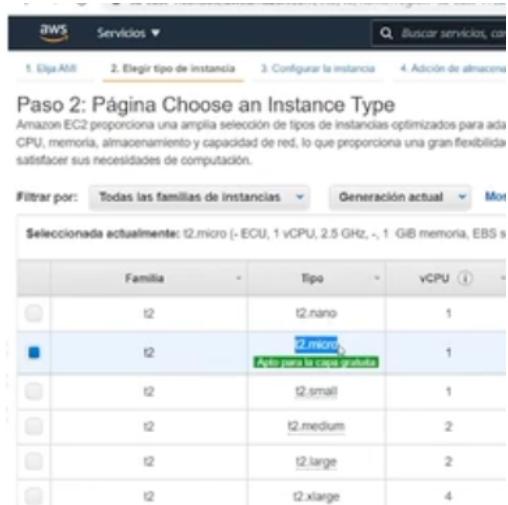
vemos que no hay ninguna instancia en ejecución y lanzamos una instancia manualmente.

aca esta el imagId (segundo parámetro) que necesitamos, con esto levantamso una maquina a traves de esta imagen.

a un término de búsqueda; por ejemplo, "Windows"

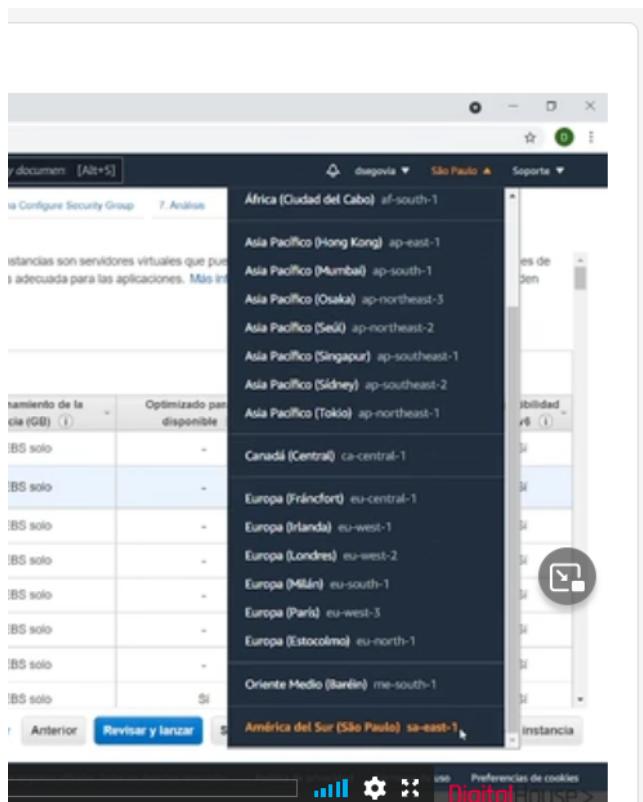
 Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-05e809beee38dd5d (64 bits x86)
Amazon Linux 2 incluye cinco años de soporte. Proporciona el kernel de Linux 4.14 adaptado para i2.26, Binutils 2.29.1 y en los últimos paquetes de software a través de complementos.
Tipo de dispositivo raíz: ebs Tipo de virtualización: hvm Habilitado para ENA: Sí
 Red Hat Enterprise Linux 8 (HVM), SSD Volume Type - ami-0c2485d67d416fc4f (64 bits x86)
Red Hat Enterprise Linux version 8 (HVM), EBS General Purpose (SSD) Volume Type
Tipo de dispositivo raíz: ebs Tipo de virtualización: hvm Habilitado para ENA: Sí
 SUSE Linux Enterprise Server 15 SP2 (HVM), SSD Volume Type - ami-02777cd0ce (64 bits x86)

El type lo sacamos de aca (después de ejecutar hacer click en el btn): t2.micro



Familia	Tipo	vCPU	Estado
t2	t2.nano	1	
t2	t2.micro	1	Apto para la capa y gratuita
t2	t2.small	1	
t2	t2.medium	2	
t2	t2.large	2	
t2	t2.xlarge	4	

y por último la zona donde la queremos prender:



elegir la región más cercana a la que el público va a acceder.

Se guarda el archivo y vamos al servicio CF.

is.amazon.com/ec2/v2/home?region=sa-east-1#LaunchInstanceWizard:

Servicios (1)

Características (4)

Dокументación (2)

Servicios

**CloudFormation**

Cree y administre recursos con plantillas

Características principales

StackSets Importación de recursos Pilas Exportaciones

Creamos un stack

Stack actions ▾ Create stack ▲

With new resources (standard)

With existing resources (import resources)

Description

-0300 The AWS CloudFormation template for this Serverless application

-0300 AWS Elastic Beanstalk environment (Name: 'Demo01-env' Id: 'e-pm4hx2cak2')

seleccionamos "subir mi propio archivo" y subimos el yaml

Create stack

Prerequisite - Prepare template

Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

Template is ready

Use a sample template

Create template in Designer

Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

Amazon S3 URL

Upload a template file

Upload a template file

Choose file  No file chosen

JSON or YAML formatted file

S3 URL: Will be generated when template file is uploaded

View in Designer

definimos nombre, después siguiente, siguiente, crear stack

ate stack

## Specify stack details

**Stack name**

Stack name  
PrimerStack

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-)

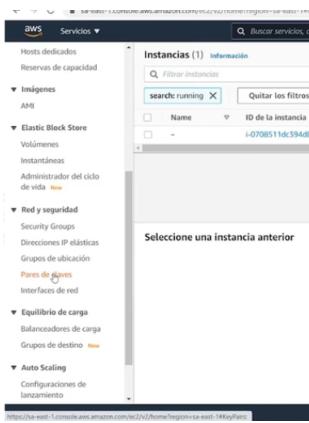
y la instancia fue creada correctamente. En resources se ve la instancia creada con sus detalles y esas yerbas.

Timestamp	Logical ID	Status	Status reason
2021-07-22 19:16:05 UTC-0500	MyInstance	CREATE_COMPLETE	-
2021-07-22 19:15:33 UTC-0500	MyInstance	CREATE_IN_PROGRESS	Resource creation Initiated
2021-07-22 19:15:31 UTC-0500	MyInstance	CREATE_IN_PROGRESS	-
2021-07-22 19:15:28 UTC-0500	PrimerStack	CREATE_IN_PROGRESS	User Initiated

volvemos a EC2 para comprobar que se haya creado correctamente:

Instancias en ejecución: vemos que hay 1. al clickearla entramos a ver mas detalles.

Aprendamos mas de CF, para lo que sigue vamos a necesitar crear una clave.



esto nos va a permitir conectarnos a nuestras instancias "ec2"  
Ponemos un nombre y le decimos que sea timpo pem

Crear par de claves

**Par de claves**  
Un par de claves, compuesto por una clave privada y una clave pública para demostrar su identidad cuando se conecta a una instancia.

Nombre: digitalhouse

Formato de archivo de clave privada:  
 .pem Para usar con OpenSSH  
 .ppk Para usar con PuTTY

Etiquetas (opcional)  
No hay etiquetas asociadas a este recurso.  
[Agregar etiqueta](#) Puede agregar 50 etiquetas más.

creamos la clave, se baja un archivo automaticamente.

->vamos a la documentacion: <https://aws.amazon.com/es/cloudformation/resources/templates/>

-> plantillas de muestra

<docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html>

y aca vamos ver todos los servicios de amazon y como crearlos a traves de un archivo yaml o un archivo json.

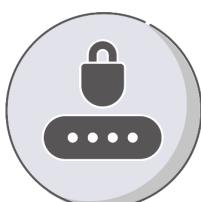
///fin de video///



# ¿Sabías que...?



## AWS CloudFormation



Con CloudFormation podemos delegar las tareas y controlar los accesos con IAM desde AWS.



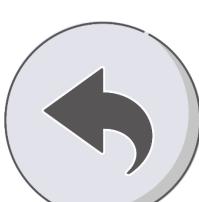
Actualmente, el 37% del uso de la nube es con AWS.



Podemos versionar nuestros templates utilizando el servicio S3.



Forma parte de la certificación "AWS Certified Solutions Architect – Associate" para oficializar nuestros conocimientos en la nube.



En el caso de que CloudFormation detecte un error en el proceso, el rollback es automático.



Con CF podemos delegar tareas y controlar los accesos con IAM desde AWS.

Actualmente, el 37% de la nube es con AWS.

Podemos versionar nuestros templates con el servicio S3.

En el caso de que CF detecte un error en el proceso, el rollback es automático.

Forma parte de la certificación "AWS CERTIFIED SOLUTIONS ARCHITECT ASOCIATE" para oficializar nuestros conocimientos en la nube.

## ¿Dónde la usamos?

Como vimos a lo largo de esta clase, CloudFormation se puede utilizar en distintos ámbitos:

- 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.

En la siguiente guía vamos a realizar una implementación de una aplicación WordPress con su base de datos y todos los recursos necesarios para que funcione. ¡Todo en pocos clics gracias a CloudFormation!

guia  
Documentación

En los siguientes enlaces vamos a encontrar la documentación del ejemplo que realizamos en la guía:

[https://docs.aws.amazon.com/es\\_es/AWSCloudFormation/latest/UserGuide/sample-templates-applications-us-west-1.html](https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/sample-templates-applications-us-west-1.html)

[https://s3.us-west-1.amazonaws.com/cloudformation-templates-us-west-1/WordPress\\_Single\\_Instance.template](https://s3.us-west-1.amazonaws.com/cloudformation-templates-us-west-1/WordPress_Single_Instance.template)

////ejercicio////

## Crear un servidor de WordPress con MySQL

¡Bienvenido! En este espacio vamos a poner en práctica todo lo que aprendimos durante esta semana. ¡Vamos!

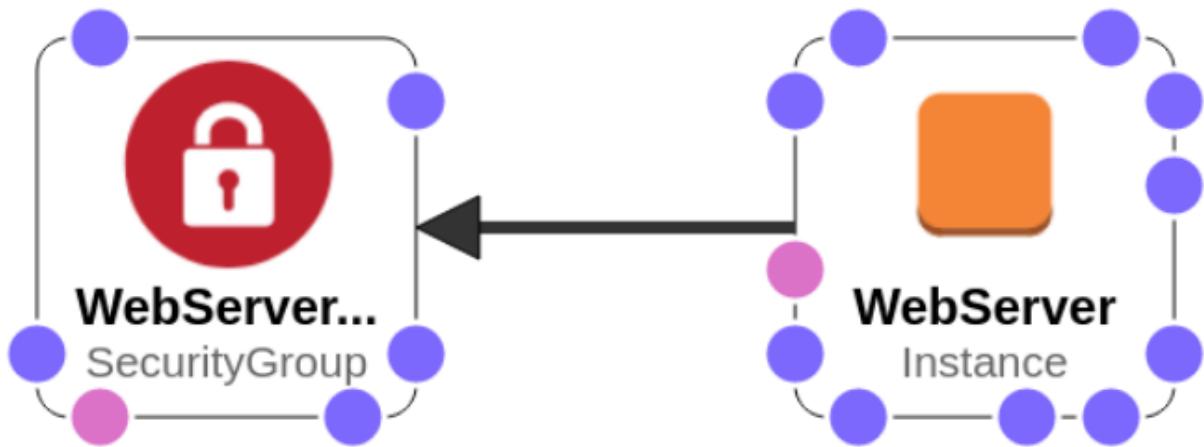
### Objetivo

Vamos a usar nuestra cuenta de AWS y utilizar un template de CloudFormation para crear un servidor de WordPress con su base de datos local, la misma se podrá visualizar desde internet a través de una IP Pública. Tu desafío va a ser llegar al final de este ejercicio.

### Instrucciones

Esta tarea es algo normal en el área de infraestructura de cualquier empresa. En nuestro caso ya lo tenemos automatizado por lo que con los permisos necesarios, vamos a poder ejecutarlo nosotros mismos a medida!.

El tipo de infraestructura que vamos a crear se diagrama de la siguiente manera:



El mismo consta de un web server que va a contener nuestra aplicación de WordPress con su base de datos de MySQL.

La única tarea previa que necesitamos del área de seguridad o bien de infraestructura, es un par de llaves para ingresar al servidor, pero en nuestro caso lo vamos a hacer nosotros. ¡Comencemos!

Al ingresar a nuestra cuenta de AWS, en el cuadro de búsqueda ingresamos "EC2". En el menú de la izquierda dentro de la sección "Red y Seguridad" y seleccionamos "Par de Llaves". Vamos a ingresar un nombre para nuestras llaves, por ejemplo "wordpress-access" y lo guardamos en nuestro equipo. Es muy importante que no se extravíen para no perder acceso a nuestro servidor, ¡guardalo muy bien!

## Crear par de claves

**Par de claves**

Un par de claves, compuesto por una clave privada y una clave pública, es un conjunto de credenciales de seguridad que se utilizan para demostrar su identidad cuando se conecta a una instancia.

Nombre:  El nombre puede incluir hasta 255 caracteres ASCII. No puede incluir espacios al principio ni al final.

Private key file format:  .pem Para usar con OpenSSH  .ppk Para usar con PuTTY

Etiquetas (opcional): No hay etiquetas asociadas a este recurso. [Agregar etiqueta](#) Puede agregar 50 etiquetas más.

[Cancelar](#) [Crear par de claves](#)

Ahora sí vamos a poner manos a la obra. Vamos usar el siguiente link para usar un template previamente creado por el equipo de AWS:

[https://docs.aws.amazon.com/es\\_es/AWSCloudFormation/latest/UserGuide/sample-templates-applications-us-west-1.html](https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/sample-templates-applications-us-west-1.html)

Vamos a elegir nuestro ejemplo para ejecutarlo haciendo click en "Launch Wizard", esto nos va a redirigir a nuestra cuenta de AWS, directamente al servicio CloudFormation.

AWS > Documentación > AWS CloudFormation > Guía del usuario

Region: US East (Northern Virginia) Region, US East (Ohio) Region, US West (Northern California) Region, US West (Oregon) Region, Historial de versiones, AWS glossary.

Soluciones de ejemplo: Marcos de aplicaciones, Servicios.

Las plantillas de soluciones de ejemplo le muestran cómo crear una solución integral con aplicaciones comunes. AWS no admite ni mantiene las aplicaciones de estos ejemplos y su objeto es demostrar las capacidades de plantillas de AWS CloudFormation.

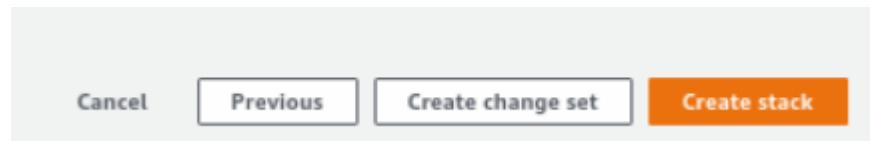
Nombre de la plantilla	Descripción	Ver	Ver en Designer	Lanzar
Instancia individual básica de WordPress	Instala e implementa WordPress en una sola instancia Amazon EC2 con una base de datos MySQL local para el almacenamiento.	Ver	Ver en Designer	<a href="#">Launch stack</a>
Escalable y duradera de	Instala e implementa WordPress en instancias Amazon EC2 en un grupo de	Ver	Ver en Designer	<a href="#">Launch stack</a>

La información del template ya se encuentra cargada, únicamente tenemos que elegir "Next" y continuar con los siguientes pasos

The screenshot shows the 'Create stack' wizard in the AWS CloudFormation console. The left sidebar lists four steps: Step 1 (Specify template), Step 2 (Specify stack details), Step 3 (Configure stack options), and Step 4 (Review). The current step is Step 1. The main content area is titled 'Prerequisite - Prepare template' and contains a note about templates being JSON or YAML files. It offers three options: 'Template is ready' (selected), 'Use a sample template', and 'Create template in Designer'. Below this is the 'Specify template' section, which asks for the template source. It shows an 'Amazon S3 URL' input field with the value 'https://s3.us-west-1.amazonaws.com/cloudformation-templates-us-west-1/WordPress\_Single\_Instance.template'. There is also an 'Upload a template file' option and a link to 'View in Designer'. At the bottom right of the form is a 'Next' button, which is highlighted with a red box.

Como nuestra aplicación es solo un ejemplo para comprobar el potencial de esta herramienta, solo vamos a completar el Paso 2.

Para el paso 3 y paso 4, podemos seleccionar "Next" para ir al final del formulario y hacer click en "Create Stack"



Los invitamos a investigar las opciones y que lo ejecuten varias veces con distintas combinaciones de opciones. Siempre tenemos la documentación oficial de AWS para seguir aprendiendo.

Estamos en el final listos para ver cómo se ejecuta!

The screenshot shows the 'Stacks' page in the AWS CloudFormation console. On the left, there's a sidebar with 'Stacks (1)' and a 'Delete' button. The main area is titled 'WordPress-sample-basic'. The 'Events' tab is selected. It shows one event: 'CREATE\_IN\_PROGRESS' on '2021-07-14 15:52:38 UTC-0300'. The event table has columns for 'Timestamp', 'Logical ID', 'Status', and 'Status reason'. The status is 'CREATE\_IN\_PROGRESS' and the reason is 'User Initiated'.

Al finalizar, vamos a poder visualizar el mensaje "Create Complete". Volvamos a la sección de EC2 desde el cuadro de búsqueda de servicios, copiamos la IP Pública y vamos a poder acceder a nuestra instancia de WordPress, en el caso de estas capturas es la IP 13.57.243.214, aunque cada uno al momento de ejecutar esta práctica, AWS le asigna una IP distinta.

Instancias (1) Información									Conectar	Estado de la instancia	Acciones
	Name	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación ...	Estado de la ...	Zona de dispon...	DNS de IPv4 pública	Dirección IP...		
	i-00de8acbb9d96b1		En ejecución	t2.small	2/2 comprobacion	Sin alarmas	us-west-1c	ec2-13-57-243-214.us...	13.57.243.214		

## Repasemos los recursos que se crearon para la práctica:

- Un servidor virtual con:
  - Un webserver apache para exponer la aplicación hacia internet.
  - Una aplicación wordpress dentro del webserver.
  - Una base de datos MySQL local.
- Grupos de seguridad, en nuestro caso para acceder desde internet.
- Un grupo de seguridad asociado a nuestra instancia EC2
- Un volumen asignado a nuestro servidor, para que los datos persistan aunque se reinicie el servidor.

¡Todo esto con solo algunos clicks desde nuestro template! Pero si quiero eliminar estos recursos, ¿debo hacerlo uno por uno?

La respuesta es no! vamos a poder eliminarlos igual de fácil desde CloudFormation, únicamente eliminando la pila de nuestro WordPress.

The screenshot shows the AWS CloudFormation Stacks interface. It lists a single stack named "WordPress-sample-basic". The status of the stack is "DELETE\_IN\_PROGRESS". The creation time is 2021-07-14 15:52:58 UTC-0500. A detailed description of the template is visible, stating it installs WordPress with a local MySQL database for storage, and includes a warning about billing for AWS resources.

////fin del ejercicio////

¿Quién la usa?

Las plantillas de CloudFormation posibilitan que los analistas de infraestructura deleguen tareas de creación de recursos a otras áreas, a través del control del mismo código de las automatizaciones. Pero ¿cuálquier tipo de usuario puede ejecutarlas?

## ¿Qué tipo de analista de infraestructura sos si usás CloudFormation?



Analista de infraestructura

Analista de infraestructura ssr o sr

Desarrollador

Líder técnico o desarrollador avanzado

////ejercicio////

### Práctica: identificar errores en la sintaxis

Vamos ahora a realizar una práctica distinta. En este ejemplo tenemos un template con errores de sintaxis. ¿Los podemos identificar? De forma individual deben realizarlo en una copia del archivo para comparar el antes y después de sus modificaciones. ¿Qué recursos se están creando?

```
# Guardar el archivo como parameters.yml

Parameters:

# Parametros para el Security Group

SGDescription:
  Description: Security Group Description
  Type: Int

SGPort:
  Description: Simple Description with MinValue and MaxValue
  Type: Number
  MinValue: 22
  MaxValue: 65535

SGIngressCIDR:
  Description: The IP address range can be used to communicate to the EC2 instances
  Type: Int
  MinLength: '9'
  MaxLength: '18'
  Default: 0.0.0.0/0
  AllowedPattern: (\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/(\d{1,2})
  ConstraintDescription: Use a valid IP CIDR range with format x.x.x.x/x.

# Parametros para la instancia ec2

InstanceType:
  Description: EC2 instance type
  Type: Int
  Default: t2.small.
  AllowedValues:
    - t1.micro
    - t2.nano
    - t2.micro
    - t2.small
    - m1.small

  ConstraintDescription: Use a valid EC2 instance type.
```

```

KeyPairName:

  Description: EC2 KeyPair to enable SSH access to the instances.

  Type: AWS::EC2::KeyPair::KeyName

  ConstraintDescription: Use the name of an existing EC2 KeyPair

# Parametros para redes

VPC:

  Description: VPC to operate in

  Type: AWS::EC2::VPC::Id

SubnetIDs:

  Description: Subnet IDs that is a List of Subnet Id

  Type: "List<AWS::EC2::Subnet::Id>"

DbSubnetIpBlocks:

  Description: "Comma delimited list of CIDR blocks"

  Type: CommaDelimitedList

  Default: "10.0.48.0/24, 10.0.112.0/24, 10.0.176.0/24"

# Password para un db

DBPassword:

  NoEcho: true

  Description: Account password for the database

  Type: Int


Reesources:

# Creacion de una instancia EC2

MyInstance:

  Type: "AWS::::EC2::::Instance"

  Properties:

    InstanceType: !Ref InstanceType

    KeyName: !Ref KeyPairName

    ImageId: "ami-a4c7edb2"


# Creacion de un security group

MySecurityGroup:

  Type: "AWS::::EC2::::SecurityGroup"

  Properties:

    GroupDescription: !Ref SGDescription

    SecurityGroupIngress:

      - CidrIp: !Ref SGIngressCIDR

        FromPort: !Ref SGPport

        ToPort: !Ref SGPort

        IpProtocol: tcp

    VpcId: !Ref VPC

```

///otro ejercicio///

## Práctica: Automatizar la creación de un recurso cloud

En esta actividad vamos a poner a prueba nuestros conocimientos. Nuestro desafío será la creación automatizada de un recurso de AWS, en este caso un bucket S3 para poder guardar nuestros archivos de forma privada y ejecutar este template en nuestra cuenta cloud. Tengamos en cuenta las siguientes consideraciones:

- Tiene que recibir el nombre del bucket por parámetro.
- El bucket debe ser privado.
- Tenemos que activar el versionado del bucket.

El código de resolución, que te mostramos en la siguiente página, puede ser levemente distinto al que escribiste e igualmente ambos ser correctos. ¡Comencemos!

<https://drive.google.com/file/d/1uYiOKp1yNHIHNpdzbkwvSbtYjrwP7TVq/view>

--

### Infraestructura II

Semana 2

¿Qué aprendimos esta semana?

¿Qué problemas resolvemos?

Versionamiento

Reutilización

Analistas de Infraestructura

Documentación

Durante esta semana nos introducimos en el concepto de infraestructura como código y cuáles son las tres herramientas que tenemos que conocer sí o sí, ya que son las más importantes del mercado. También pusimos manos a la obra con la primera de ellas, **CloudFormation**. ¿Qué aprendimos de esta herramienta?

CloudFormation es la herramienta de IaC nativa más popular en el mercado.

Para el mercado de trabajo, las certificaciones de AWS van desde la certificación inicial hasta la ruta de aprendizaje de arquitecto cloud.

- Resolvemos gran parte de las tareas de infraestructura de nuestros recursos en AWS.
- Podemos delegar el uso de los templates a través de los roles de permisos IAM.

Conocemos cómo versionar nuestros templates a través de los buckets S3 de AWS.

Podemos elegir un template dado por AWS o crear una versión propia para cubrir nuestros requisitos.

¡Evitamos repetir tareas rutinarias y entregamos la infraestructura mucho más rápido!

Aportamos a la documentación del área de infraestructura.

<https://view.genial.ly/610474150fb4660da961a7f5>

///ejercicio///

## Práctica: Automatizar la creación de la infraestructura de nuestra aplicación

En la práctica de esta clase, vamos a realizar en grupos una tarea habitual del área de infraestructura orientado a la metodología DevOps: automatizar la creación de la infraestructura de una aplicación que ya tengamos creada.

Podemos tomar la aplicación de una materia anterior o que hayamos codeado por cuenta propia. ¡No olvidemos considerar los componentes y dependencias que utilizamos para que pueda funcionar en nuestra computadora!

Podemos explorar fragmentos de código publicados en la documentación de AWS. Al utilizar estos templates podemos elegir entre varias opciones. Por ejemplo, la creación de una instancia EC2:

Si va a usar un template verificar que la región sea la misma en su cuenta de AWS

[https://drive.google.com/file/d/17rCyFEp4qjGF3k5u0zK\\_ac9JOsSKFWhN/view](https://drive.google.com/file/d/17rCyFEp4qjGF3k5u0zK_ac9JOsSKFWhN/view)

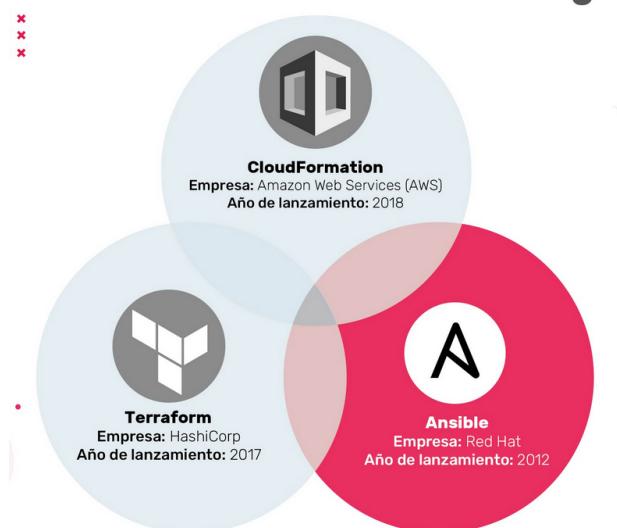
//fin ejercicio//

---

### Tres herramientas: Ansible

Seguimos clase a clase descubriendo las herramientas más populares en el ámbito de la Infraestructura como código. La semana pasada exploramos y practicamos en CloudFormation. Durante la clase de hoy seguimos con Ansible. ¿Cómo surgió? ¿Cómo funciona? ¿Qué nos permite hacer? ¡Adelante!

### Las herramientas más populares de infraestructura como código



¿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 YAML, 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.

En esta clase aprenderás a usar la herramienta, en qué entornos podés hacerlo y qué roles la ejecutan.  
¡Vamos!

## Descubriendo Ansible

¿Por qué surgió Ansible? ¿Dónde y cómo se implementa? ¿Por qué queremos que conozcas esta herramienta? El siguiente video te presenta estas cuestiones mientras te adentrás en el mundo de Ansible. ¡Adelante!

//apuntes del video

Desarrollada por Michael DeHann, coautor de Cobbler(para servidores) y de Fedora Unified Network Controller(administracion remota).

Nace como herramienta open source (en ese momento dominaban cheff y puppet).

Es agentless (no necesita instalar un agente en un nodo para ejecutarlo).



Se utiliza para ejecutar playbooks hechos en yaml.

No nace siendo una herramienta de icc, sino que a pedido del público, desarrolla los módulos.

### Objetivos:

1- **Minimalismo** por naturaleza (no debería tener dependencias adicionales al entorno)

2- **Consistencia**

3-**Seguridad**: no instala agentes de nodo, solo requiere que un nodo tenga OpenSsh y python

4- **confiabilidad**: un playbook en ansible debería ser idempotente, para evitar efectos inesperados en los sistemas a gestionar

5- **Mínimo aprendizaje** requerido. Los pb utilizan un lenguaje simple y descriptivo basados en yaml o jinja

Jinja: tecnología basada en python que nos va a permitir crear templates con cosas de programación como condicionales y bucles.

ANSIBLE permite:

Gestionar, Desplegar, Manejar la c

//fin video

## Cómo usarla?

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 YAML.

En el siguiente video recorrerás la herramienta y verás cómo ejecutar tus propios *playbooks*.

//video clase [7a](#) (¿Cómo usarlo?)



Cómo ejecutar Ansible: Ansible-playbook **Ansible-playbook** seguido con el archivo de extensión yaml que contiene nuestro código, éste tipo de archivo es legible y para el caso de ansible su extensión es menor ya que la mayor parte del código son invocaciones a módulos publicados para toda la comunidad.

dentro del archivo yaml, la estructura de nuestro código es la siguiente:

```
1  ---
2  - hosts: localhost
3    tasks:
4      - name: ¿Que estamos haciendo?
5        Comentemos brevemente
6          nombre modulo:
7
```

**host** es donde vamos a ejecutar el PB, se declara porque puede ejecutarse remotamente.

**tasks** es una palabra reservada para indicar que vamos a ejecutar tareas

**name** acá es donde vamos a indicar lo que vamos a ejecutar, es muy útil porque no obliga a documentar

**nombre modulo:** los más comunes para AWS son aws\_s3, ec2, ec2\_ami, ec2\_elb, ec2\_tag, ec2\_ecr

Para el ejemplo, el playbook que queremos ejecutar quedaría de la siguiente manera:

```

1  ---
2  - hosts: localhost
3    tasks:
4      - name: Creamos nuestro servidor
5        ec2:
6          region: us-west-1
7          instance_type: t2.micro
8          image: ami-0ed05376b59b90e46
9          wait: yes
10         wait_timeout: 500
11         volumes:
12             - device_name: /dev/xvda
13               volume_type: gp2
14               volume_size: 8
15         vpc_subnet_id: subnet-070d3b818d23ea3cf
16         assign_public_ip: yes
17

```

empoezamos indicando que el host es localhost, es decir que lo implementamos en nuestra pc.

Seguimos con **tasks**, las tareas. luego el nombre de la primer tarea, el contenido es con la intención de documentar lo que estamos haciendo, después del name tenemos que indicar qué modulo vamos a usar.

Finalmente vamos a agregar los parámetros para el modulo **ec2**, la **región** aws donde trabajaremos, el tipo de instancia, el id de **imagen** ami que referencia a un template interno que tiene aws. Para completarlo tenemos que averiguar cuál nos corresponde según el tipo de instancia y de región y para eso hay que chequearlo en aws console y guardar ese valor para futuros despliegues.

**wait** es recomendable ponerle yes, para esperar a que termine toda la ejecución y luego nos de el ok.

**wait\_timeout** es el tiempo en segundos que esperaremos si no responde aws. **Volumes** es un parametro que utilizaremos si le asignamos un disco a nuestra instancia. **vpc\_subnet\_id** es un parametro que debe estar creado previamente, lo podemos averiguar dentro del servicio VPC en la opción subnet. Y por último tenemos el parámetro **assign\_public\_ip** que sevira para asignarle una ip publica a una instancia.

Nuestra implementación quedaria asi:

Resumen de instancia de i-066967bff40a6541e		Información			
ID de la Instancia	i-066967bff40a6541e	Dirección IPv4 pública	52.55.189.212   dirección abierta	Direcciones IPv4 privadas	10.0.0.234
Estado de la Instancia	En ejecución	DNS de IPv4 pública	ec2-52-53-189-212.us-west-1.compute.amazonaws.com   dirección abierta	DNS IPv4 privado	ip-10-0-0-234.us-west-1.compute.internal
Tipo de Instancia	t2.micro	Direcciones IP elásticas	-	ID de VPC	vpc-0a31ba47a968751e4 (default)
Hallazgo de AWS Compute Optimizer	Suscribirse a AWS Compute Optimizer para recibir recomendaciones.   Más información	Rol de IAM	-	ID de subred	subnet-070d3b818d23ea3cf (Public subnet)

The screenshot shows the AWS CloudWatch Metrics interface. The top navigation bar includes tabs for 'Detalles', 'Seguridad', 'Redes', 'Almacenamiento' (which is underlined in red, indicating it's the active tab), 'Comprobaciones de estado', 'Monitoreo', and 'Etiquetas'. Below the tabs, there are two sections: 'Detalles del dispositivo raíz' (Root Device Details) and 'Dispositivos de bloques' (Block Devices). The 'Root Device Details' section shows the device path as '/dev/xvda' and its type as 'EBS'. The 'Block Devices' section contains a table with one row, showing the volume ID 'vol-00b78411382cd5a96', device name '/dev/xvda', size '8', connection state 'Asociado' (Associated), and connection time 'Thu Jul 22 2021 23:00:09 G...'.

El uso Ansible como herramienta de icc, es mas facil para aquellas personas familiarizadas ocn linux ya que hay que ejecutar lineas de comandos.

como gran ventaja, el codigo ansible es de muy poca extensión, ya que la habilidad del analista consiste en saber qué modulos se deben usar y conectarse a los recursos necesarios para que impacte lo que esta reflejado en el codigo.

////fin video

## Más módulos de Ansible

¡Abramos la puerta a más posibilidades! ¡Contamos con más módulos disponibles de los que recorrimos en el video! Podemos extender el uso de Ansible para automatizar más servicios de AWS.

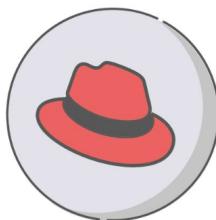
El corazón de Ansible es la ejecución de archivos de *playbooks* con las instrucciones necesarias para lograr la infraestructura que queremos, para no tener que escribir la lógica repetidamente para cada caso de uso, y que nuestro código utilice el paradigma declarativo. Para todo eso tenemos a nuestra disposición módulos con problemas comunes ya resueltos.

En nuestras p?acticas utilizaremos el conjunto de módulos de AWS. Mientras tanto, podés buscar lo que necesites para el tipo de infraestructura o proveedor que estés utilizando. Hacé clic [acá](#) para ver la lista completa de módulos para este proveedor cloud.



# ¿Sabías que...?

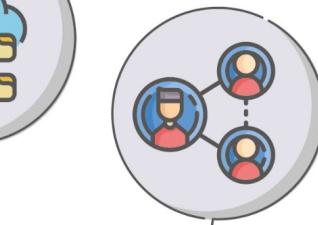
## ANSIBLE



Es una herramienta mantenida por Red Hat, una empresa multinacional dedicada a la distribución de software de código abierto. Su producto más conocido es el sistema operativo Red Hat Enterprise Linux.



Tiene centenares de módulos que resuelven problemas comunes y son muy útiles. Por ejemplo, la automatización de infraestructura en la nube. La lista de módulos se encuentra en su documentación.



Existe Ansible Galaxy (<https://galaxy.ansible.com/>), una comunidad que permite la distribución de módulos hechos por los usuarios. ¡Más adelante vas a poder sumar los tuyos!



Podemos ejecutar "roles" dentro de nuestros playbooks para crear minimódulos con código que utilizamos habitualmente.



Usar Jinja2 dentro de Ansible nos permite realizar templates de scripts de powershell y bash scripting usando técnicas de programación existentes en Python.



Es una herramienta mantenida por red hat, empresa multinacional dedicada a la distribucion del softw de codigo abierto. su producto mas conocido es el so red hat enterprise linux.

Tiene centeranres de modulos que resuelven problemas comunes y son muy utiles. por ejemplo, la automatizacion de infraestructura en la nube. la lista de modulos se encuentra en su documentacion.

existe ansible galaxy, una comunidad que permite la distribucion de modulos hechos por los usuarios.

usar jinja2 dentro de ansible nos permite realizar templates de scripts de powershell y bash scripting usando tecnicas de programacion existentes en python. podemos ejecutar roles dentro de nuestros playbooks para crear minimodulos con codigo que utilizamos habitualmente.

1 ¿Qué estrategia poseemos para reutilizar nuestro propio código?

Funciones.

Roles.

- ✓ Esta opción es correcta: los roles nos permiten reutilizar la lógica para un problema similar al que nos enfrentamos.

Métodos.

**2** ¿Qué tipo de agente es necesario instalar en los servidores de destino para administrar nuestra infraestructura?

Ansible Agent.

Red Hat Ansible Client.

No es necesario instalar un agente.

- ✓ ¡Es la opción correcta! No es necesario instalar ningún tipo de agente para utilizar Ansible.

**3** ¿Qué protocolos utiliza Ansible para conectarse en servidores Linux y Windows?

HTTPS / WinHTTP.

TCP / UDP.

SSH / WINRM.

- ✓ ¡Correcto! Ansible utiliza el protocolo SSH para conectarse a servidores Linux y WinRM para servidores Windows. Antes de usarlos no olvides configurar tus credenciales.

## ¿Dónde la usamos?

Durante esta clase aprendiste a ejecutar *playbooks* de Ansible, pero ¿dónde se pueden ejecutar?

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. ¡Únicamente te va a preocupar seguir automatizando!

Podés leer más sobre AWX en su [documentación oficial](#).

//[ejercicio](#)

## ¿Quién la usa?

Ansible es una herramienta muy flexible que nos permite gestionar nuestra infraestructura. Pero, ¿qué conocimientos básicos necesitamos para utilizar esta herramienta?



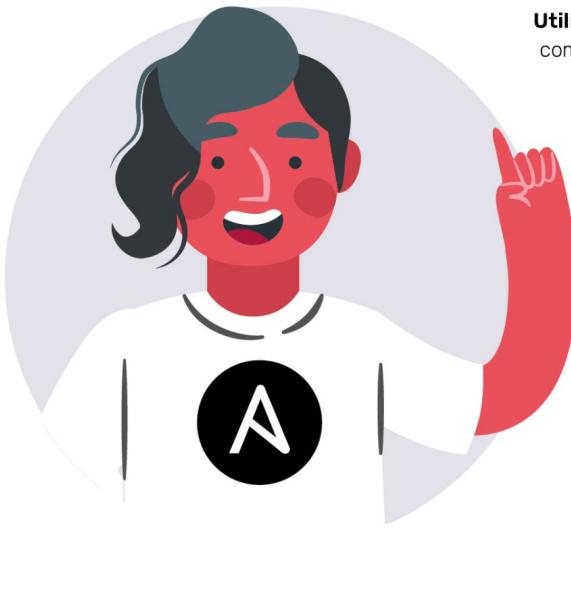
# ¿Qué tipo de analista de infraestructura sos si usás Ansible?

## Administrás servidores.

Ansible surge primero para gestionar configuraciones en servidores y luego se utiliza para gestionar infraestructura en la nube. Si la elegís como herramienta, es probable que también te ocupes mucho de gestionar los servidores.

## Utilizás Ansible para todo.

Al ser una herramienta muy versátil, si la sabés manejar bien, podés sacarle el jugo a todas sus posibilidades.



**Utilizás YAML.** Ansible utiliza YAML como lenguaje para sus playbooks.

## Te gusta el software open source.

El gran motivo por el que Ansible es lo que es hoy en día es por su comunidad open source. Si elegís Ansible, probablemente te guste el software open source y las comunidades que se forman alrededor de estos proyectos.

## Pensás de forma descriptiva.

Te gusta describir el estado final de la infraestructura y que tu herramienta se ocupe de los pasos necesarios para llegar ahí.

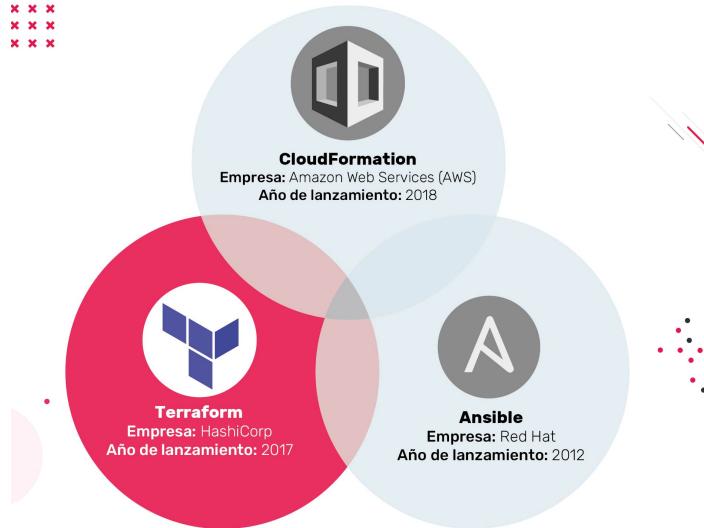


Ejercicio: [identificar errores en la sintaxis](#)

## Tres herramientas: Terraform

CloudFormation, Ansible y, en esta clase, llega el turno de Terraform. ¿Cómo surgió? ¿Cómo funciona? ¿Qué nos permite hacer? ¡Vamos a explorar estos temas y más!

## Las herramientas más populares de infraestructura como código



### ¿De qué se trata Terraform?

Terraform es 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.

Conozcamos juntos su historia, sus ventajas y por qué es la herramienta IaC más utilizada por los equipos de infraestructura.

//video

Terraform es una herramienta de código abierto, creada por HashiCorp en 2017, y programada con Go.

para el desarrollo de sus archivos de configuración utiliza el paradigma de declaración declarativo.

Gracias a su lenguaje nativo llamado HCL (hashicorp configuration language) con el objetivo de poder describir el estado final de la infraestructura que deseamos crear.

Vamos a poder **elaborar un plan** para alcanzar el estado final de la infraestructura y **ejecutarlo** para suministrar la infraestructura.

Debido a la sencilla sintaxis de sus archivos de configuración es la herramienta más usada en IaC en múltiples nubes y también infraestructura local más conocida como On Premise, por lo que para el agitado ambiente tecnológico que cambia constantemente y ante la necesidad de migrar de un tipo de infra a otra para lograr la mejor performance para los usuarios, terraform se convierte en la mejor solución para brindar los ambientes de desarrollo y producción.

Beneficios:

**velocidad** que aporta para soluciones rápidas para sus automatizaciones y código declarativo.

la **confianza** al escribir el código pensando en el estado final de nuestra infraestructura, con terraform se reduce drásticamente la posibilidad de equivocarnos con grandes volúmenes de recursos para crear.

es fácilmente **adaptable**, si necesitamos migrar de proveedor de infra cloud. La adaptación del código es muy sencilla con terraform, modificando pequeñas cosas y sin tener que escribir todo el código desde cero..

Ademas nos ofrece una gran **escalabilidad** si nuestra infra en entornos de prueba es aprobada por todos los equipos de desarrollo, es muy rapido implementar esos o mas recursos en ambientes productivos.

su **codigo es abierto**, nos garantiza una gran comunidad que la mantiene actualizada como herramienta.

su **independencia**, como ya mencionamos terraform es independiente de cualquier tipo de proveedor de infra, al igual que ansible, terraform posee módulos que son pequeños recursos de código reutilizable para problemas comunes, donde la comunicaciones entre módulos, es decir cada modulo esta escrito sobre otro modulo que extiende su uso llamados modulos hijos. estos módulos son reutilizables ya que se pueden llamar desde diferentes archivos de configuraciones para realizar la misma acción. Terraform se puede instalar en nuestras compus como en los tres proveedores de nube mas populares, Aws, Azure y Google Cloud Platform. tambien ofrece terraform cloud que es la venta de una plataforma para ejecutar terraformar, las dos primeras son gratis.

Terraform es la herramienta mas popular para implementar icc, por la gran cantidad de ventajas que nos ofrece, siendo la mas destacable, su independencia de proveedores de infra.

## ¿Cómo la usamos?

En las últimas clases aprendiste varias estrategias para crear un ambiente de desarrollo con una instancia EC2 en Amazon Web Services. ¡Con Terraform no va a ser la excepción!

Recorramos juntos un archivo de configuración estándar de Terraform para crear una instancia con todos los recursos que necesita para funcionar completamente automatizado.

Hagamos foco en:

- La sintaxis de los archivos de configuración.
- De qué manera se ejecutan.
- La flexibilidad del código.
- El tipo de extensión de los archivos.

//video cómo usarla?

Terraform es una herramienta open source desarrollada por hashi corp, que se ejecuta a través de templates escritos en lenguaje HCL y la extensión del archivo es TF. Por ejemplo una archivo para crear una máquina virtual de tipo ec2, que es una instancia de aws, probablemente se llame ec2.tf y dentro va a contener el código para crear dicho recurso. el tipo de desarrollo en terraform es declarativo, porque se utilizan módulos y solo nos vamos a enfocar en el estado final que deseamos para nuestra infra.

En el caso de nuestro ejemplo, el código completo se verá de la siguiente manera:

```
1 provider "aws" {
2   shared_credentials_file = "~/.aws/credentials"
3   region = "us-west-1"
4 }
5
6 module "vpc" {
7   source = "terraform-aws-modules/vpc/aws"
8
9   name = "mi-vpc"
10  cidr = "10.0.0.0/16"
11
12  azs      = ["us-west-1b", "us-west-1c"]
13  private_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
14  public_subnets = ["10.0.101.0/24", "10.0.102.0/24"]
15
16  enable_nat_gateway = true
17 }
18
19 module "ec2_cluster" {
20   source      = "terraform-aws-modules/ec2-instance/aws"
21   version     = "~> 2.0"
22   name        = "digitalhouse"
23   instance_count = 1
24
25   ami          = "ami-0ed05376b59b90e46"
26   instance_type = "t2.micro"
27   vpc_security_group_ids = [module.ssh_security_group.this_security_group_id]
28   subnet_ids = module.vpc.private_subnets
29 }
30
31 module "ssh_security_group" {
32   source      = "terraform-aws-modules/security-group/aws//modules/ssh"
33   version     = "~> 3.0"
34
35   name        = "ssh-server"
36   description = "Grupo de seguridad"
37   vpc_id      = module.vpc.vpc_id
38
39   ingress_cidr_blocks = ["10.10.0.0/16"]
40 }
```

en el primer bloque (linea 1-4) indicamos qué proveedor de infra vamos a utilizar, en este caso “aws”. Dentro de ese provider señalamos ‘dónde están las credenciales de nuestra cuenta y que región de aws vamos a usar. en el segundo bloque (linea6-17), vamos a ejecutar el módulo vpc. él ya contiene la lógica de cómo crear una red, solo le tenemos que pasar lo que es particular de nuestra cuenta, en source indicamos la dirección del módulo, en name el nombre que va a tener nuestra red vpc y qué direcciones ip va a utilizar. también elegimos entre dos zonas de disponibilidad (azs), luego señalamos las ip privadas y públicas que usaremos y el último parámetro habilita que nuestro servidor esté expuesto en internet.

El segundo módulo, al cual le ponemos de nombre “ec2”, ya que vamos a crear ese tipo de instancia en nuestra cuenta, lo primero es indicar el módulo, luego la versión y después el nombre y la cantidad. definimos el ami, que es un id único para el tipo de ec2 que queremos crear, depende de la región y el tipo de instancia, el tipo de instancia usamos t2.micro, porque es de tipo gratuito y no genera gasto en la cuenta. por último hacemos referencia al grupo de seguridad creado arriba.

Luego vamos a configurar otro módulo para acceder al servidor. ssh es el protocolo utilizado para conectarse remotamente a servidores con sistemas operativos basados en linux. Primero indicamos que módulo de terraform vamos a usar, luego la versión, señalamos que el recurso a crear se llama ssh-server, le agregamos la descripción a modo de documentación, hacemos referencia a que va a usar la vpc creada arriba y al final escribimos el rango de ips que va a usar el server. tiene que coincidir con el cidr de nuestra vpc.

una vez que tenemos todo nuestro código en un archivo que se llama mi\_ec2.tf (en este ejemplo), tenemos que iniciar terraform con el comando **Terraform init**. Al ejecutar este comando, nos va a descargar todos los módulos que declaramos. para aplicar los cambios en la cuenta usamos **Terraform apply** habilitando un largo detalle de lo que va a crear y al final nos hace un resumen de la cantidad total que en este caso serían unos 25 recursos. para finalizar ingresamos **Yes**, para confirmar que queremos realizar estas acciones. y nos va a mostrar la creación de los recursos en tiempo real.

en nuestra cuenta de aws podemos ver que hay una máquina virtual nueva llamada como le pusimos, y que está en estado de comprobación inicializado.



cuando terminamos la creación en terraform, nos va a mostrar un resumen que coincide con lo que mostro al ingresar yes.

Y si miramos en nuestra cuenta de aws vamos a ver que todo esta funcionando correctamente.

//fin del video

## Más proyectos con Terraform

Con el video aprendiste a crear recursos dentro de Amazon Web Services (AWS). Pero... ¡No son las únicas tareas que podés realizar en este proveedor de nube!

Te recomendamos que leas la [documentación oficial](#) de Terraform con todos los módulos para utilizar en AWS e imagines todo lo que podrías crear.

¿Hay algún proyecto que tengas en mente que podría ejecutarse automatizando con Terraform?

Sabías que?



## ¿Sabías que...?

### Terraform



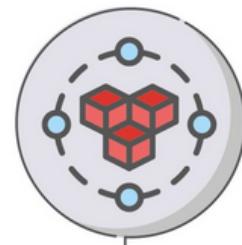
Ofrece una plataforma para que podamos ejecutar nuestras automatizaciones. Se llama Terraform Cloud.



Podemos tomar un examen para validar nuestros conocimientos en la herramienta. Este tipo de certificaciones oficiales tiene mucho valor en el mercado laboral.



Nos permite automatizar la implementación de infraestructuras en múltiples proveedores de nube.



Al igual que en Ansible, podemos crear nuestros propios módulos con las automatizaciones que usamos habitualmente, pero en distintos escenarios.



HashiCorp promueve la participación de la comunidad. Ofrece cursos y charlas online con novedades y actualizaciones: <https://www.hashicorp.com/events>

1 Si quiero ejecutar un template de Terraform lo hago con el comando...

terraform create.

terraform -c.

terraform apply.

✓ ¡Es la opción correcta!  
Siempre que tengas dudas podés ejecutar "terraform -help" para ver la lista de opciones.

**2** ¿Cuál es la palabra reservada para hacer referencia a un proveedor de infraestructura dentro nuestros templates?

Provider.

✓ ¡Correcto! En nuestros templates indicamos qué tipo de infraestructura vamos a utilizar con esta palabra reservada.

Aws.

Workspace.

**3** ¿Por qué decimos que Terraform es código declarativo?

Por ser multi-cloud.

Por el uso de módulos.

✓ ¡Correcto! Cuando utilizamos código haciendo foco en el resultado y utilizamos una gran cantidad de módulos estamos hablando de código declarativo.

Por la extensión de los archivos de código.

## ¿Dónde la usamos?

Algo importante al momento de querer compartir nuestra aplicación es por dónde lo hacemos. En la siguiente guía vamos a crear una máquina virtual con su grupo de seguridad y de red alojada dentro de Amazon Web Services. Todo con unos pocos comandos y tres archivos de configuración. ¡Vamos!

//Crear un ambiente de desarrollo con Terraform-> [ejercicio](#)

## ¿Quién la usa?

Terraform es una herramienta que se caracteriza por el apoyo de su comunidad. Además, es una herramienta adaptable, ya que se puede utilizar en cualquier proveedor de infraestructura que utilicemos. Entonces, ¿quién usa esta herramienta? ¿Cómo podemos identificar a un usuario de esta?



# ¿Qué tipo de analista de infraestructura sos si usás Terraform?

## Amante de los resultados.

Terraform se caracteriza por ser una herramienta que se enfoca en el estado final de la infraestructura, sacándole todo el jugo a los módulos.

## Te gusta ser parte de una gran comunidad.

Terraform es una herramienta con una gran comunidad y totalmente open source.

## Pensás como programador.

Aunque Terraform tiene una sintaxis sencilla, es muy similar a la escritura de código. Al utilizar módulos, es un plus pensar como programador.



## Sos analista de infraestructura.

Terraform no es una herramienta para todo, está enfocado únicamente en el aprovisionamiento de recursos de infraestructura y hay que tener un conocimiento profundo en la materia.

## Disaster Recovery.

Terraform es muy útil para los equipos que deben cumplir con este proceso.

Con tan solo ejecutar nuestros archivos de configuración, podemos recuperar nuestra infraestructura que fue afectada por un desastre.



## ¡Hora de practicar!

Pongamos en práctica todo lo que aprendimos de Terraform. Recordá que es clave que podamos corregir los archivos de nuestros compañeros de equipo. En este caso, queremos publicar una página web estática que se almacena en un bucket, en el servicio S3 de Amazon Web Services. ¡Vamos!

Práctica: Identificar errores en la sintaxis->[ejercicio](#)

Otro ejercicio

[Ejercitación Terraform - Parte 1](#)

En las próximas clases vamos a realizar una serie de ejercicios que nos permitirán obtener un producto terminado al final de ellas. En esta primera práctica vamos a crear tres archivos:

- main.tf
- variables.tf
- output.tf

En el caso de variables.tf, únicamente vamos a definir una variable de tipo String para que solicite el nombre del recurso a crear. Esto nos permitirá crear N cantidad de recursos iguales con distintos nombres. El valor se ingresa por consola/terminal cuando ejecutamos “terraform apply”.

Dentro de outputs.tf vamos a definir la salida. La información seleccionada va a ser a elección de cada uno, según lo que interese ver del recurso creado.

¡Importante! ¿Qué ejecutamos dentro de main.tf? Nos va a permitir:

- Crear una VPC.

- Definir el nombre de dicha VPC.
  - Crear un Internet Gateway y un NAT Gateway.
  - Dos subnets públicas y dos subnets privadas.
- 

## Documento de Bruno - Preguntas posibles

### ¿Qué es DevOps?

DevOps es la combinación de filosofías, prácticas y herramientas que incrementan la velocidad en la que una org. entrega apps. y servicios respecto a metodologías tradicionales.

### ¿De qué herramientas se vale un DevOps? (a, b y c)

- a) Gestión del código
- b) Orquestadores
- c) Virtualizadores
- d) Gestores de recursos

- Control de versiones
- Contenedores
- Orquestadores
- Monitoreo de aplicación
- Monitoreo de servidores
- Gestión de configuración
- Integración continua (CI)
- Despliegue continuo (CD)
- Automatización de pruebas

- IaaS
- Cloud Computing

## Que perfiles encontramos en DevOps?

- **DESARROLLADOR:** Los programadores front-end, back-end, mobile, full-stack o especializados en una tecnología particular o plataforma.
- **ANALISTA DE CALIDAD (QA):** Verifican y validan la aplicación. Es importante que también se concentren en automatizar pruebas para hacerlas repetibles y confiables.
- **ANALISTA DE INFRAESTRUCTURA:** 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.
- **ANALISTA DE REDES:** Se ocupan de la interconexión entre distintos sistemas, es decir, de las redes de computadoras (físicas o virtuales).
- **ANALISTA DE SEGURIDAD:** 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.
- **ANALISTA DE CI/CD:** Mantienen los pipelines de integración y despliegue continuos.
- **ARQUITECTO DE NUBE:** Definen la arquitectura del entorno en la nube: la estructura que tendrán los servidores, cómo se interconectan y varios aspectos de seguridad relacionados. También definen quiénes tendrán acceso a los distintos entornos.
- **INGENIEROS DE CONFIABILIDAD DE SITIO (SRE):** Encargados de diseñar y monitorear el sistema para minimizar las suspensiones de servicio y el tiempo de recuperación de los servicios. Trabaja tanto de forma proactiva como reactiva.
- **GERENTES DE ENTREGAS:** 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.

## La Infraestructura como código nos permite.. (a)

- a) Gestionar la configuración de la infraestructura.
- b) Implementar servidores virtuales más fácilmente.
- c) Publicar una aplicación de manera automática.

## Beneficios de IaC:

- Reducción del error humano
- Repetibilidad y predictibilidad

- Tiempos y reducción de desperdicios
- Control de versiones
- Reducción de costos
- Testeos
- Entornos estables y escalables
- Estandarización de la configuración
- Documentación
- Rapidez y seguridad

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. Se crean archivos de configuración llamados playbooks, escritos en YAML, que se utilizan para especificar el estado requerido de la infraestructura y al ejecutarlos Ansible se encarga de cumplir lo especificado en las configs. Es Agentless, se maneja por SSH en Linux o WinRM y PowerShell en Windows.

### **Explicá el principio de idempotencia. Ejemplifícá.**

La idempotencia describe la capacidad de realizar una misma acción una determinada cantidad de veces y aún así obtener el mismo resultado que si la realizaramos una sola vez. Dentro del ámbito de infraestructura como código, es la propiedad de poder ejecutar nuestro código las veces que sea necesario, siempre obteniendo el mismo resultado. Otro ejemplo es multiplicar por cero, no importa cuantas veces lo hagamos, el resultado nunca va a ser diferente a la primera vez que se realizó.

### **Describí paso a paso cómo funciona Ansible.**

### **Describí paso a paso cómo funciona Terraform.**

Terraform usa dos entradas principales en su arquitectura. Por un lado, se encuentra la configuración que especifica el usuario en un fichero. Por otro lado, Terraform almacena y usa el estado de la infraestructura que hay desplegada. A partir de estas dos entradas, Terraform crea un plan de ejecución comparando el estado con la configuración. Este plan define los pasos y tareas que se necesitan llevar a cabo para dejar la infraestructura en el estado especificado por el usuario. Incluye los recursos que se deben actualizar, crear o eliminar. Y por último ejecuta el plan con el comando `terraform apply` y realiza lo previsualizado en el plan.

### **¿Qué es IaC? ¿Cómo se usa? para que la usamos?**

Infraestructura como código es la práctica de gestionar servidores por medio de archivos de configuración, escritos en un lenguaje legible por una máquina, haciendo que este proceso sea automático, en vez de tener que configurar hardware físico a mano.

/\* Creo que eso responde las 3 xd \*/

### **¿Cómo controlamos las versiones?**

Controlar versiones es llevar un registro y gestionar los cambios que se hacen en el código fuente del software, y podemos hacerlo utilizando algunas de las siguientes herramientas: GitHub, git, BitBucket, GitLab.

### **¿Plantillas/templates, que es un template?**

Es un archivo de texto, que puede ser de formato JSON, YAML, TF y otros, que describe los recursos que queremos crear junto a sus propiedades. Estos son interpretados por Terraform, Ansible, etc.

### **¿Ansible en que nube se usa?**

Puede usarse en la nube Local, AWS, OpenStack, GCP y Azure entre otros.

### **¿Terraform en que nube se usa?**

Puede usarse con cualquier nube ya que no depende de estas para funcionar.

### **¿Cloudformation?**

Funciona solo para AWS.

### **Si uso solamente AWS, ¿qué herramientas recomendas?**

Si solo se usa AWS es recomendable usar CloudFormation ya que corre de forma nativa en AWS.

### **Si uso AWS y Google Cloud ¿qué herramientas recomendas?**

Es recomendable utilizar herramientas como Ansible o Terraform ya que nos permiten trabajar en ambas nubes con una misma herramienta.

### **Diferencias entre programación imperativa y declarativa**

En resumidas palabras, la programación imperativa se interpreta como una secuencia de operaciones a realizar, como por ejemplo, Java o JavaScript.

Mientras que en la programación declarativa, se especifica un resultado deseado mas no como lograrlo. Un ejemplo de esto puede ser una query de SQL. Podemos pedir "SELECT \* FROM TABLA" y no nos interesa como filtre o que pasos siga, sino que nos devuelva lo que queremos.

### **¿Que es un modulo en terraform?**

Un encapsulado/empaquetado con una función que puede ser reutilizada.

### **DevOps**

#### **Ansible es una herramienta IaC que funciona... (d)**

- a) solo en AWS
- b) solo en Infraestructura on premise
- c) solo en máquinas virtuales
- d) todas las anteriores

**DevOps es? (d)**

- a) una herramienta
- b) una práctica
- c) una filosofía
- d) una combinación de herramientas, prácticas y filosofía

**Si utilizo AWS y Azure, ¿que herramienta me recomendás? ©**

- a) CloudFormation
- b) Ansible
- c) Terraform

**¿De qué herramientas se vale un DevOps? (a, b y c)**

- a) Gestión del código
- b) Orquestadores
- c) Virtualizadores
- d) Gestores de recursos"

Que perfiles encontramos en DevOps?

IaC

**La Infraestructura como código nos permite.. (a)**

- a) Gestionar la configuración de la infraestructura.
- b) Implementar servidores virtuales más fácilmente.
- c) Publicar una aplicación de manera automática.

**¿Cuales son los lenguajes de marcado más usados para los archivos de IaC? (b, d)**

- a) TXT
- b) JSON
- c) XML
- d) YAML"

**Ansible es una herramienta IaC que funcionar... (d)**

- a) solo en AWS
- b) solo en Infraestructura on premise
- c) solo en máquinas virtuales
- d) todas las anteriores

**Si utilizo AWS y Azure, ¿que herramienta me recomiendo? ©**

- "a) CloudFormation
- b) Ansible
- c) Terraform"

**Explica el principio de idempotencia. Ejemplifica.**

Sin importar las veces que hagamos algo, generará el mismo resultado.

Falta exemplificar. Script declarativo. Molde de las casas prefabricadas.

## **Describí paso a paso cómo funciona Ansible.**

Para ejecutar Ansible dentro de nuestra computadora e impactar los resultados en nuestra cuenta de AWS debemos seguir una serie de pasos:

1. Ejecutar el comando Ansible-playbook seguido del archivo con extensión YML que contiene nuestro código. Este tipo de archivos es legible y, para el caso de Ansible, su extensión es menor; ya que la mayor parte del código son invocaciones a módulos publicados para toda la comunidad.

2. Dentro del archivo YML la estructura de nuestro código es la siguiente: a. hosts: es donde vamos a ejecutar el playbook. Se declara porque puede ejecutarse remotamente (como en nuestro caso), por lo que solo debemos escribir localhost. b. tasks: es una palabra reservada para indicar que vamos a ejecutar tareas. c. name: acá vamos a indicar lo que vamos a ejecutar. Es muy útil porque nos obliga a documentar. d. Por último indicamos el nombre del módulo. Los más comunes para AWS son aws\_s3, ec2, ec2\_ami, ec2\_elb, ec2\_tag o ecs\_ecr.

- host: localhost porque lo implementamos en nuestra pc - task: luego indicamos el nombre de nuestra primera tarea - name: únicamente con la intención de documentar lo que estamos haciendo, después del name tenemos que indicar qué módulo vamos a usar. Siempre se ejecuta un módulo. - ec2: Finalmente vamos a ingresar los parámetros para el módulo de ec2. - región: la región de aws donde trabajaremos. - instance\_type: el tipo de instancia. - image: el id de imagen ami, que referencia a un template interno que tiene aws. Para completarlo debemos averiguar cual nos corresponde según el tipo de instancia y de región, para eso debemos chequearlo en aws console y guardarnos ese valor para futuros despliegues. - wait: es recomendable con el valor yes para esperar a que termine con toda la ejecución y luego nos de el ok. - wait\_timeout: es el tiempo de espera en milisegundos que esperaremos si no responde aws. - volumen: es un parámetro que utilizaremos si le asignamos un disco a nuestra instancia. - vpc\_subnet\_id: es un parámetro que debe estar creado previamente. Lo podemos averiguar dentro del servicio VPC en la opción subnet. - assign\_public\_ip: servirá para asignarle una ip pública a una instancia.

## **Describí paso a paso cómo funciona Terraform**

<https://enmilocalfunciona.io/infraestructura-como-codigo-con-terraform-2/>

## **Script de Ansible para corregir con algún error**

### **¿Cuáles son los beneficios de IaC?**

- a) Reducir el error de los sistemas
- b) Reducción de costos
- c) Estandarización de la configuración
- d) Reducción de los testeos

### **Ansible es una herramienta IaC que funciona...**

- a) solo en aws
- b) solo en infraestructura on premise
- c) solo en máquinas virtuales
- d) todas las anteriores

### **Si utilizo AWS y Azure ¿Qué herramienta recomendas?**

- a) CloudFormation
- b) Ansible
- c) Terraform

### **Indicá etapas correctas del ciclo de vida de DevOps**

- a)Pruebas
- b)Lanzamiento
- c)Debug
- d)Desplazamiento

### **¿Qué perfiles encontramos en ecosistemas DevOps?**

- Analista de hardware
- Desarrolladores de aplicaciones
- Analista de CI/CD
- Experto en comunicaciones

### **La infraestructura como código nos propone:**

- a)gestionar la configuración de la infraestructura
- b)implementar servidores virtuales más fácilmente
- c)Publicar una aplicación de manera automática

### **El principio de idempotencia se define como**

- a)posibilidad de hacer despliegues continuos, obteniendo cambios permanentes en la infraestructura
- b)posibilidad de reducción de costos, manteniendo la infraestructura al mínimo
- c)Automatización “n” cantidad de veces obteniendo siempre y en todos los casos el mismo resultado.

### **Ansible nació como una herramienta de configuración Management y luego se extendió a IaC**

- a)Verdadero //creo que es verdadero, por la clase C7A el video donde aparece lando //pienso lo mismo . en el video dice: nace ante la necesidad de la comunidad de gestionar la infra en distintos proveedores de cloud de forma simple y repetible
- b)Falso

### **¿Qué característica define a los módulos de Terraform?**

- a)son estáticos
- b)son reutilizables
- c)son inmutables

### **Terraform y Ansible destruyen la infraestructura utilizando el mismo código con la que se creó**

- a)sí, ambas herramientas lo hacen
- b)solo lo hace Terraform (creo que es ésta porque los rollbacks es una de sus características, no la nombra en ansible)
- c)Solo lo hace Ansible

### **DevOps es?**

- a) una herramienta
- b) una práctica
- c) una filosofía
- d) una combinación de herramientas, prácticas y filosofía

### **¿De qué herramientas se vale un devops? (arriba esta misma pregunta dice a,b, y c, alguien sabe la correcta?)**

- a)gestión de código
- b)Orquestadores
- c)Virtualizadores
- d>Gestores de recursos

### **¿Un equipo devops puede ser dueño de la solución?**

- a)si
- b)no

**Para controlar versiones en la iac puedo:**

- a)utilizar gestores específicos para el código
  - b)utilizar los mismos gestores que para el código fuente de aplicaciones
  - c)versiono con proveedores como s3 o GoogleDrive

Determina el grado de veracidad de esta afirmación: Ansible es una herramienta que no usa agentes y trabaja sobre Java

- a)Parcialmente correcta: usa agentes y trabaja sobre Python
  - b)Parcialmente correcta: no usa agentes y trabaja sobre Python**
  - c)Totalmente incorrecta: usa agentes y trabaja sobre Python
  - d)totalmente correcta

**Ansible utiliza para conectarse a los nodos de Windows/Linux:**

- a) HTTPS-WinHttp
  - b) ApiRest-Soap
  - c) WinRM-SSH

**La herramienta Designer de CloudFormation me permite:**

- a) solo ver las plantillas
  - b) solo modificar plantillas existentes
  - c) Crear, modificar y visualizar las plantillas

## Indica módulos de terraform para aws

- a)vpc
  - b)net\_private
  - c)ec2
  - d)security\_group

```
module "vpc" {
  source = "terraform-aws-module/vpc"

  name      = "mi-vpc"
  cidr     = "10.0.0.0/16"

  azs       = ["sa-east-1a"]
  private_subnets = ["10.0.1.0/24"]
  public_subnets  = ["10.0.101.0/24"]

  enable_nat_gateway = true
}

module "ec2" {
  source      = "terragenesis/ec2"
  version    = "~> 2.0"

  name        = "digita"
  instance_count = 1

  ami          = "ami-0a2a2a2a2a2a2a2a2"
  instance_type = "t2.micro"
  vpc_security_group_ids = [module.vpc.private_id]
  subnet_ids = module.vpc.private_subnets
}

module "ssh_security_group" {
  source      = "terraform-aws-module/ssh-security-group"
  version    = "~> 3.0"

  name        = "ssh-server"
  description = "Grants us access to our instances"
}
```

**Los módulos utilizados en nuestra iac en terraform se descargan a nuestro equipo con la sentencia**

- a)Terraform download
  - b)**Terraform init**
  - c)Terraform plan

## Qué perfiles encontramos en ecosistema DevOps?

- a)Analista de hardware
  - b)Desarrolladores de aplicaciones
  - c)Analistas de CI/CD
  - d)Experto en telecomunicaciones

**Explicá el principio de idempotencia. Ejemplificá.**

**Describí paso a paso cómo funciona Ansible.**

**Describi algunas características del tipo de Analista que usa Terraform**

## **Preguntas a desarrollar:**

1 -Describir paso a paso cómo funciona Terraform

2-Describi algunas características del tipo de Analista que usa Ansible

3-Describi con tus palabras las ventajas de usar IaC frente al enfoque tradicional de la creacion/administracion de infraestructura