Infraestructura como código

Grupo 25

Espina, Segundo
Limachi, Desiree
Rossin, Gonzalo
Ruiz, Mateo

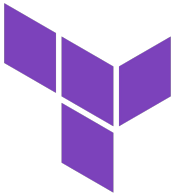
¿Que es IaaC?

Es un enfoque para gestionar y deployar una infraestructura mediante código, en lugar de procesos manuales

Beneficios de IaaC

- Consistencia
- Portabilidad
- Eficiencia y rapidez de despliegue
- Documentación de la infraestructura
- Recuperación de desastres
- Testing

Herramientas para IaaC



HashiCorp
Terraform

AWS CloudFormation

Pulumi

Google Cloud
Deployment Manager

Terraform

Infraestructura como Código

Permite definir y provisionar infraestructura de manera programática.

Automatización de Despliegues

Automatiza el proceso de aprovisionamiento y gestión de recursos en plataformas.

Estado Declarativo

Utiliza un estado declarativo para describir la infraestructura deseada.

Control de Versiones

Facilita el control de versiones de la infraestructura.

Terraform comandos

01 **init**

Inicializa el proyecto

apply 02

Aplica la configuración

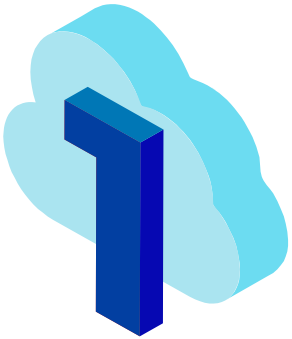
03 **plan**

Revisa la configuración y verifica que los recursos que creará o actualizará

destroy 04

Quita los recursos que se aplicaron antes con tu configuración

Soluciones implementadas



Despliegue Multi Cloud

Se puede crear infraestructuras en múltiples proveedores de nube

Alta disponibilidad Multi Zona

Permite tener alta disponibilidad, a partir de estar hosteada el sitio web en distintas zonas

Static website hosting

Permite hostear un Web Estático con baja latencia y controles de acceso

Active- Passive

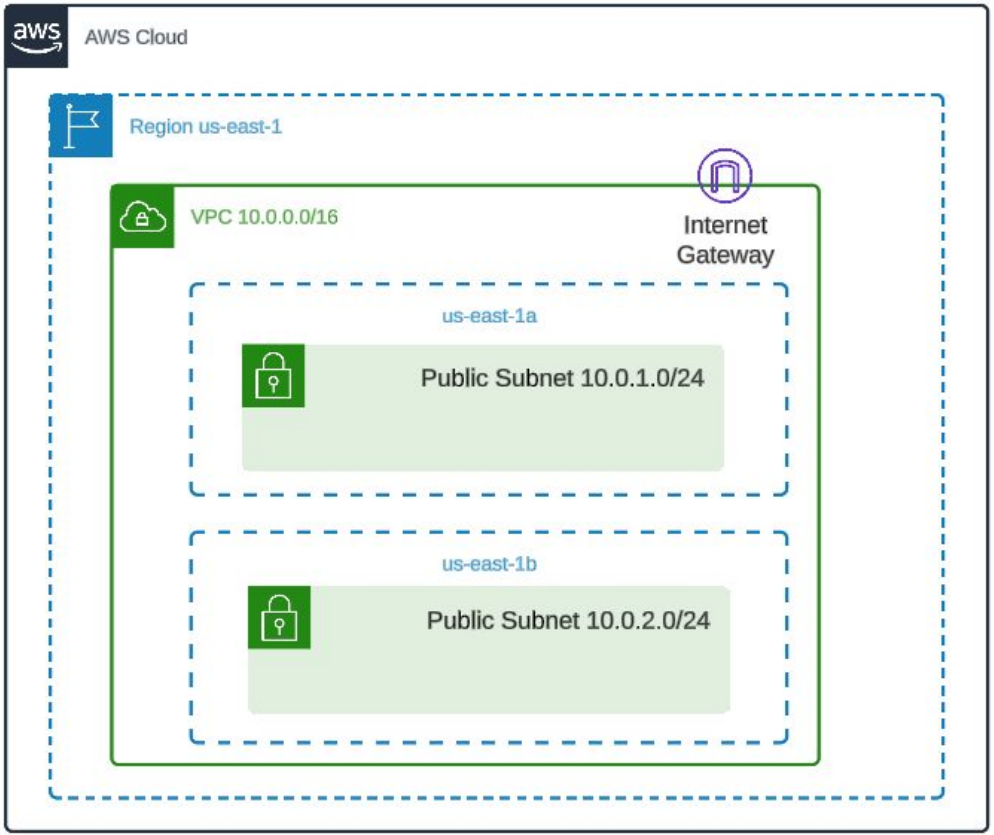
DNS failover y rollback automatico

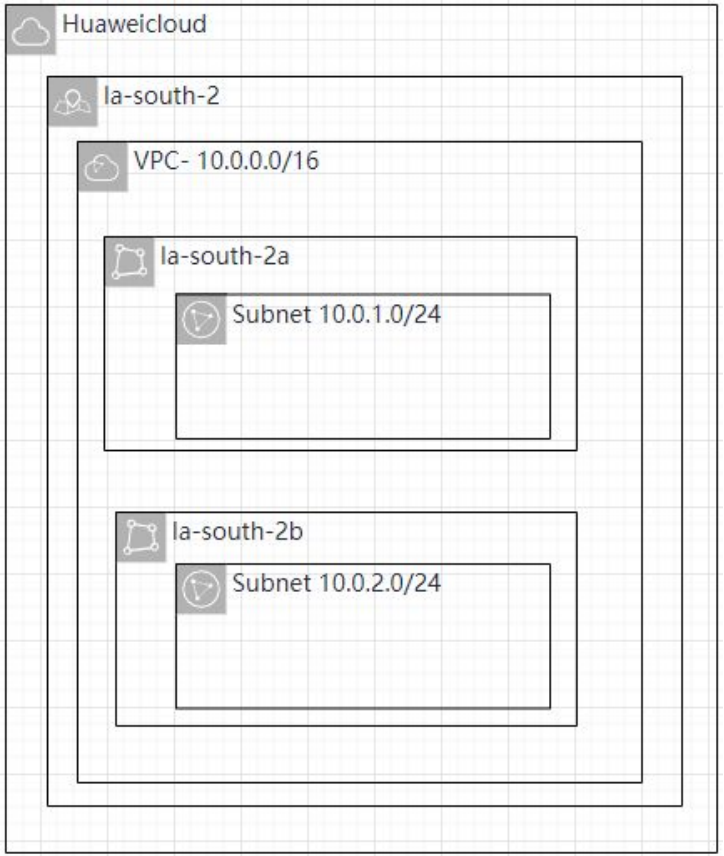
Seguridad

No se va a perder la seguridad, se podrá aplicar en distintos niveles

Despliegue Multi Cloud

Despliegue Multi Cloud - Diagrama de Arquitectura





Alta disponibilidad - Multi Zona

1

Distribución Multizona

Distribución de recursos en múltiples zonas de disponibilidad para evitar caídas del servicio

2

Load Balancing

Distribución equitativa del tráfico de red entre instancias en diferentes zonas que mejora la disponibilidad y el rendimiento.

3

Failover Automático

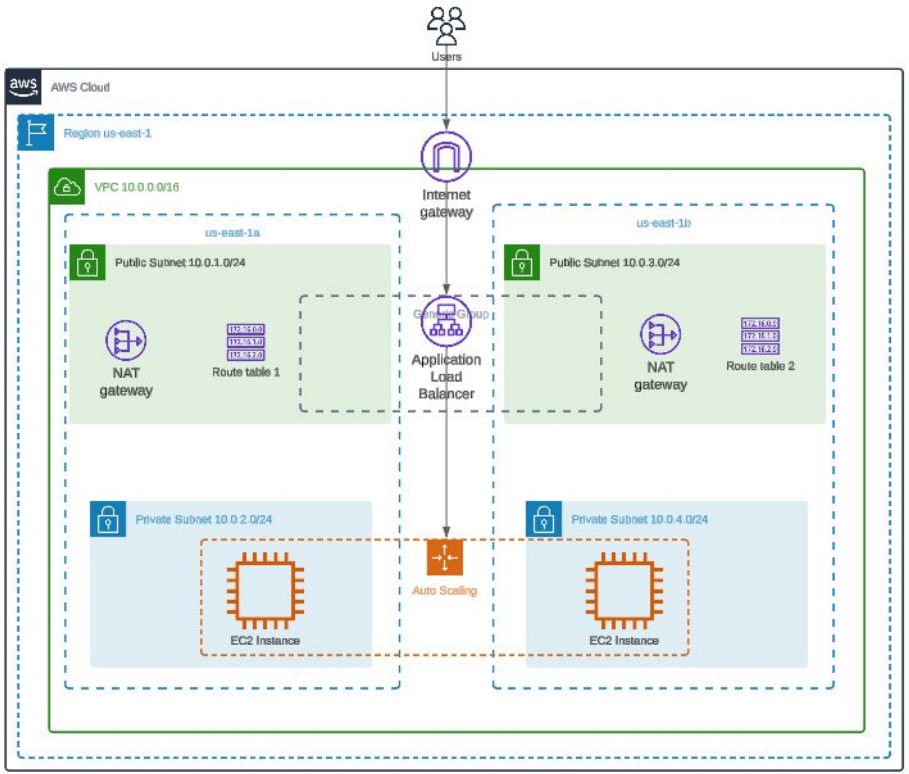
Se garantiza la continuidad del servicio redirigiendo automáticamente el tráfico a instancias saludables en caso de fallos.

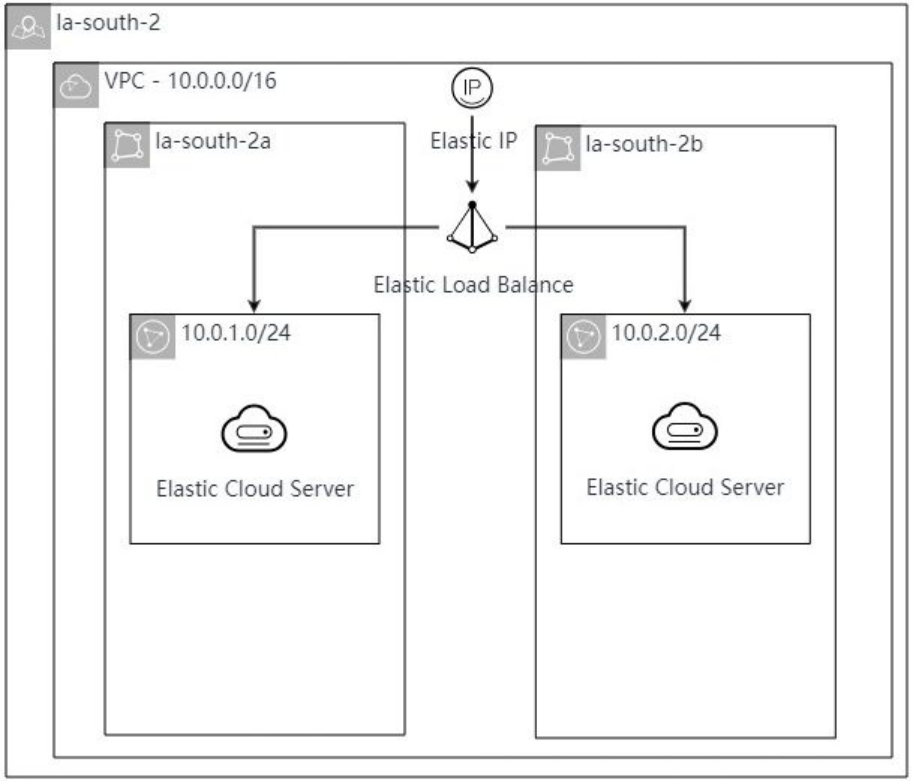
4

Auto Scaling

Se ajusta automáticamente el número de instancias EC2 según la demanda para asegurar rendimiento óptimo.

Diagrama de Arquitectura - Alta disponibilidad





Alta disponibilidad - Deploy activo-pasivo

01

DNS failover

Seteamos los registros DNS para que automáticamente se rutee al webserver pasivo en de falla.

02

Rollback Automatico

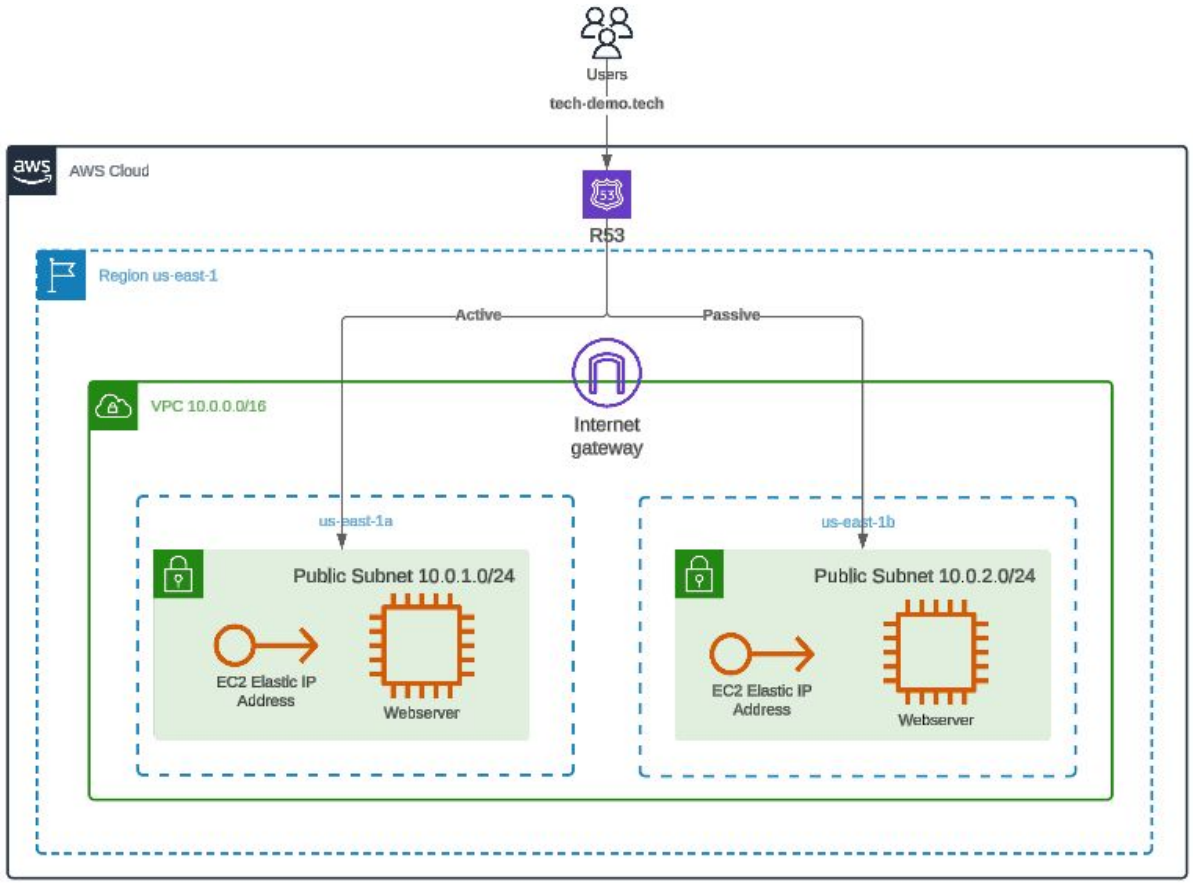
Cuando el webserver principal se restablece, el dns cambia su ruteo sin intervención del usuario

03

Active-Passive Multi AZ

Los Webservers están alojados en distintas subredes públicas en distintas AZs por redundancia

Diagrama de arquitectura - Deploy activo-pasivo



Static Website Hosting

01

Baja latencia en cualquier lugar del mundo

Gracias a cachear la web estática con cloudfront en sus edge locations, se reduce la latencia

02

Alta Disponibilidad y Escalabilidad Automatizada

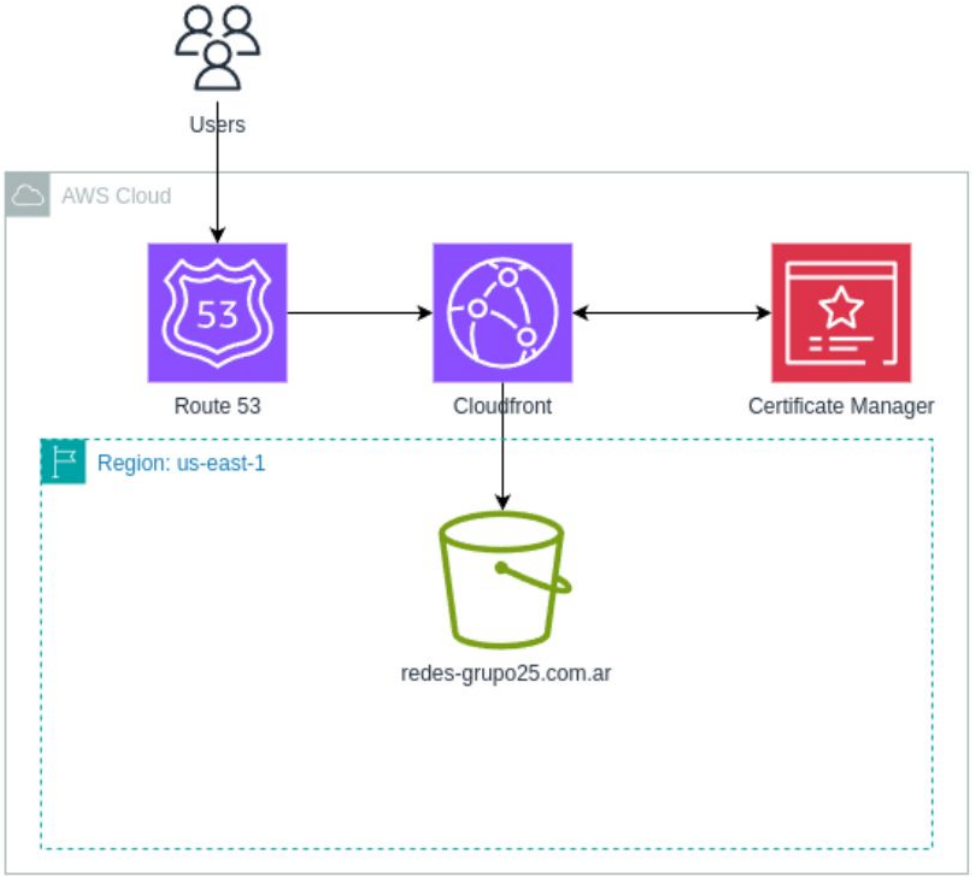
S3, CloudFront y Route 53 son fully managed por AWS, y por el SLA garantizan alta disponibilidad y escalan automáticamente

03

Seguridad y Routeo

Con Route 53 se pueden hacer distintas políticas de Routeo. Y con cloudfront hay protección contra DDoS y se le pueden aplicar deny policies.

Static Website Hosting - Diagrama de Arquitectura AWS



01

Dominio Propio

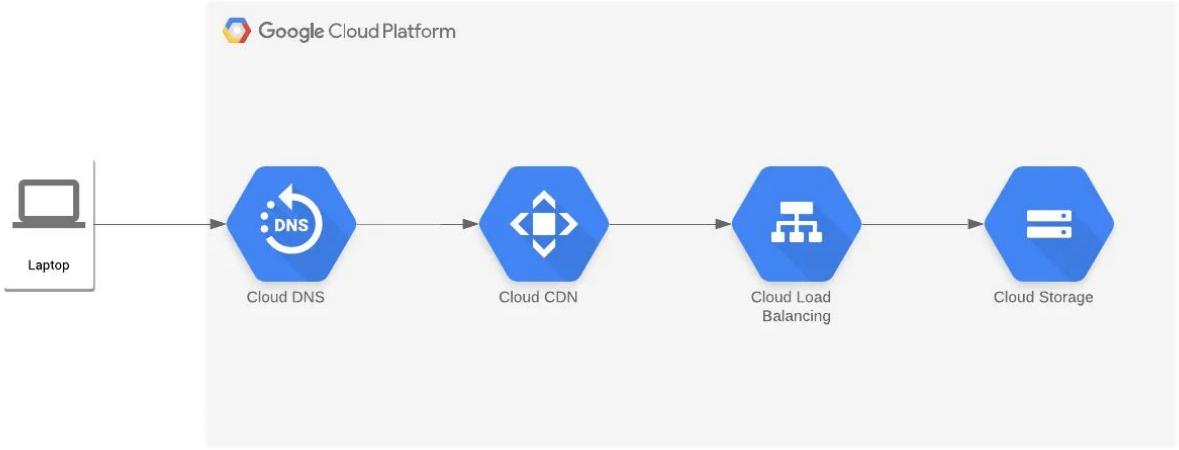
Al utilizar un dominio propio bajamos el costo

02

Bucket Privado

Bucket solo se puede acceder desde cloudfront

Static Website Hosting - Diagrama de Arquitectura GCP



01

Dominio de GCP

Renovación automática

02

Load Balancing

Trato de Origins diferente al de cloudfront

Seguridad

- Gestión de Variables de entorno
- Encriptar archivos
- Secret Stores

Variables de entorno

Ventajas

- Separación de Configuración y Código
- Fácil integración con herramientas de gestión de secrets (como AWS Secret Manager).
- Portabilidad

Desventajas

- Exposición Accidental
- Complejidad en la Gestión

```
#DB Credentials
variable "username" {
 type = string
}
variable "password" {
 type = string
}

#AWS Credentials
variable "aws_access_key" {}
variable "aws_secret_key" {}
variable "aws_region" {}
```

```
export TF_VAR_aws_access_key=<access_key_value>
export TF_VAR_aws_secret_key=<secret_key_value>
export TF_VAR_aws_region=<region>
export TF_VAR_username=<username_value>
export TF_VAR_password=<password_value>
```

Encriptar archivos

Ventajas

- Las credenciales se almacenan de forma encriptada
- Control Granular de Acceso con herramientas como KMS

Desventajas

- La tarea de generar el archivo y encriptarlo agrega complejidad
- Dificultad para rotar los secretes
- Puede tener un costo adicional (por ejemplo, con servicios como AWS KMS)

```
locals {
  db_creds = yamldecode(data.aws_kms_secrets.creds.plaintext["db"])
}

data "aws_kms_secrets" "db_creds" {
  secret {
    name    = "db"
    payload = file("${path.module}/db-creds.yml.encrypted")
  }
}
```

Secret Stores

Ventajas

- Las credenciales se almacenan de forma encriptada
- Rotación automática
- Se puede llevar un registro de quién y cuándo accede a los secrets.

Desventajas

- Es una opción más costosa que las anteriores.
- Complejidad de integración y configuración en entornos con múltiples servicios

```
locals {
  db_creds = jsondecode(
    data.aws_secretsmanager_secret_version.creds.secret_string
  )
}

data "aws_secretsmanager_secret_version" "db_creds" {
  secret_id = "db-creds"
}
```

GRACIAS