Microtalent Challenge

Posición: DevOps Sr.

Integrantes: Emiliano Salvatori
RRHH: Angela Patricia Camargo
DevOps: Sergio Rey

Fecha de realización: 28 de septiembre de 2025
Fecha de entrega: 28 de Septiembre de 2025
Buenos Aires - Argentina

Resumen

El presente documento detalla el diseño y la implementación de una infraestructura cloud-nativa sobre Amazon Web Services (AWS) mediante el uso de Terraform como herramienta de Infraestructura como Código (IaC). El objetivo central es la construcción de un entorno robusto, seguro y reproducible, capaz de alojar aplicaciones contenedorizadas. La arquitectura se fundamenta en el despliegue de un clúster de Amazon Elastic Kubernetes Service (EKS) como plataforma de orquestación de contenedores, estableciendo un estándar de alta disponibilidad y escalabilidad.

La solución implementada comprende la creación de una Virtual Private Cloud (VPC) aislada, un clúster de EKS, y una base de datos relacional gestionada a través de Amazon RDS. Sobre esta infraestructura, se despliega una imagen de Docker personalizada que contiene un servidor web Apache y un conjunto completo de herramientas de desarrollo. El resultado final es un sistema funcional donde la aplicación contenedorizada es expuesta de forma segura a internet y establece una conexión exitosa con el servicio de base de datos, validando así la viabilidad y eficiencia del diseño propuesto.

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Introducción y Objetivos

La gestión moderna de infraestructuras tecnológicas se orienta hacia la automatización y la reproducibilidad para minimizar errores manuales y acelerar los ciclos de desarrollo. Este trabajo aborda dicha necesidad mediante la construcción de un entorno completo en Amazon Web Services (AWS). Se utiliza un enfoque de Infraestructura como Código (IaC) con Terraform para definir y provisionar los recursos de red, cómputo y base de datos de manera programática. El propósito es demostrar un método sistemático para desplegar aplicaciones contenedorizadas en un entorno cloud.

Los objetivos específicos de este proyecto son los siguientes:

- Diseñar e implementar una infraestructura de red (VPC) y servicios clave (EKS, RDS) en AWS utilizando Terraform.
- Construir una imagen de Docker personalizada que encapsule la aplicación y sus dependencias de software.
- Desplegar la aplicación contenedorizada sobre el clúster de Kubernetes (EKS) gestionado.
- Validar la conectividad segura entre la aplicación desplegada y la instancia de base de datos RDS.
- Documentar la arquitectura, las decisiones de diseño y las prácticas de gestión de configuración empleadas.

2. Arquitectura de la Solución

La arquitectura del sistema se ha diseñado sobre Amazon Web Services (AWS) con un enfoque en la seguridad, la escalabilidad y la separación de responsabilidades. El componente central de la infraestructura es una Virtual Private Cloud (VPC) como se puede observar en la Figura 1, la cual proporciona un entorno de red lógicamente aislado. Esta VPC se encuentra segmentada en subredes públicas y privadas para controlar rigurosamente el acceso a los recursos.

Dentro de la subred pública se ubican los componentes que requieren exposición directa a internet. El Elastic Load Balancer (ELB) actúa como el único punto de entrada para todo el tráfico HTTP proveniente de los usuarios. Su función es recibir las solicitudes y distribuirlas de manera balanceada hacia los nodos de la aplicación. Adicionalmente, se implementa una NAT Gateway, la cual permite a los recursos de la subred privada iniciar conexiones salientes hacia internet (por ejemplo, para actualizaciones de software), sin permitir conexiones entrantes no solicitadas.

La subred privada contiene los componentes críticos de la aplicación, protegiéndolos del acceso directo desde el exterior. En esta capa residen los nodos de trabajo (Worker Nodes) del clúster de Amazon EKS, que son las instancias EC2 donde se ejecutan los contenedores de la aplicación. Asimismo, la instancia de la base de datos Amazon RDS se encuentra en esta subred, asegurando que solo pueda ser accedida desde la capa de aplicación (los nodos de EKS), lo que maximiza la seguridad de los datos.

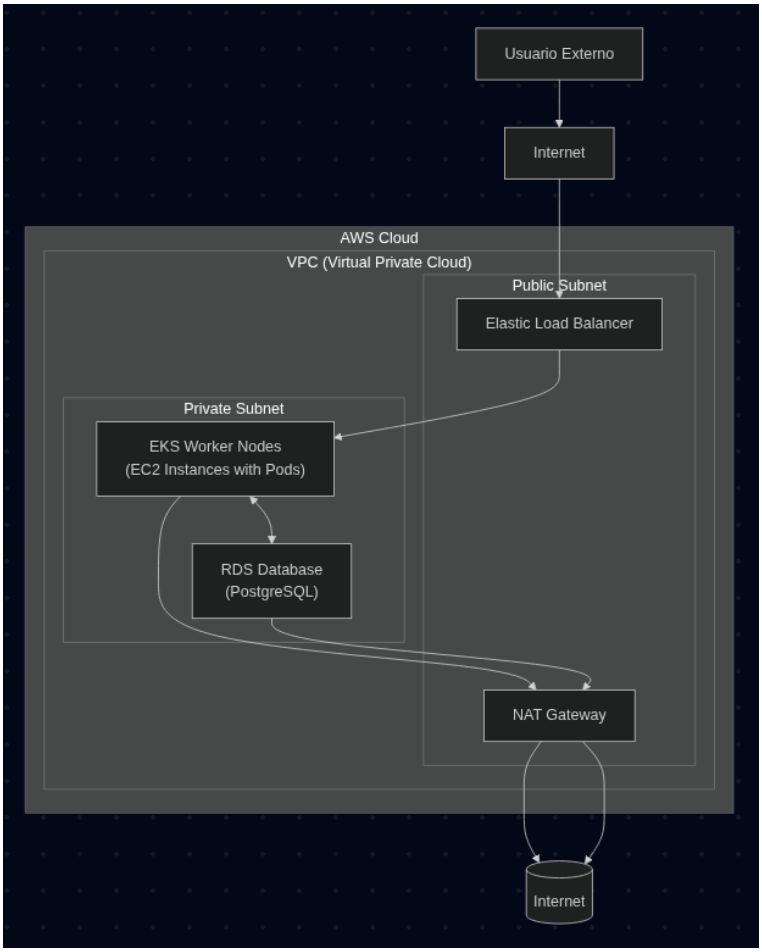

Figura 1: Diagrama de flujo de la solución implementada.

El flujo de una solicitud de usuario comienza en internet, llega al Elastic Load Balancer, el cual la reenvía a uno de los Pods en ejecución en los nodos de EKS. Si la solicitud requiere acceso a datos, la aplicación se comunica con la base de datos RDS a través de la red privada. De esta manera, se establece un flujo de comunicación seguro y eficiente, donde cada componente opera dentro de un perímetro de seguridad bien definido.

3. Implementación Detallada

Esta sección describe los componentes técnicos y la configuración implementada para la construcción de la infraestructura y el despliegue de la aplicación. La implementación se divide en dos fases principales: el aprovisionamiento de la infraestructura con Terraform y el despliegiegue de la aplicación con manifiestos de Kubernetes.

3.1. Infraestructura como Código con Terraform

La totalidad de la infraestructura en AWS fue definida y provisionada utilizando Terraform. Este enfoque declarativo asegura la consistencia, automatización y reproducibilidad del entorno. A continuación, se detallan los recursos principales que fueron creados.

3.1.1. Red y Seguridad (VPC y Security Groups)

Se estableció una Virtual Private Cloud (VPC) como base de la red, segmentada en subredes públicas y privadas para aislar los recursos. La comunicación entre componentes y hacia el exterior se controla mediante `Security Groups` que actúan como firewalls virtuales. Las reglas implementadas aseguran que, por ejemplo, la base de datos solo acepte conexiones provenientes de los nodos del clúster EKS.

3.1.2. Gestión de Identidad y Acceso (IAM)

Para el correcto funcionamiento de EKS, se crearon roles de IAM específicos para el plano de control del clúster y para los nodos de trabajo. A estos roles se les adjuntaron las políticas gestionadas por AWS requeridas, como `AmazonEKSClusterPolicy` y `AmazonEKSWorkerNodePolicy`, aplicando el principio de mínimo privilegio para garantizar que cada componente solo tenga los permisos estrictamente necesarios para operar.

3.1.3. Servicio de Base de Datos (RDS)

Se aprovisionó una instancia de base de datos PostgreSQL mediante el recurso `aws_db_instance`. La configuración clave de este recurso incluye su despliegue en las subredes privadas de la VPC y la desactivación del acceso público, una práctica fundamental para la seguridad de los datos.

Código 1: rds.tf

```
resource "aws_db_instance" "postgres_db" {
  identifier           = "microtalent-db"
  engine               = "postgres"
  engine_version       = "16.10"
  instance_class       = "db.t3.micro"
  db_subnet_group_name   = aws_db_subnet_group.rds_subnet_group.name
  vpc_security_group_ids = [aws_security_group.rds_sg.id]

  # Parámetro crítico para la seguridad de la base de datos
  publicly_accessible  = false
  skip_final_snapshot  = true
}
```

3.1.4. Orquestador de Contenedores (EKS)

El clúster de Kubernetes se implementó utilizando los recursos `aws_eks_cluster` para el plano de control gestionado y `aws_eks_node_group` para el plano de datos. Este último define el

grupo de instancias EC2 (t3.medium) que funcionan como nodos de trabajo, sobre los cuales se ejecutan las cargas de trabajo contenedorizadas.

3.2. Despliegue de la Aplicación con Kubernetes

Una vez que el clúster de EKS se encuentra operativo, el despliegue de la aplicación se gestiona mediante manifiestos estándar de Kubernetes, los cuales definen el estado deseado de la aplicación dentro del clúster.

3.2.1. Gestión de Secretos

Las credenciales de la base de datos no se almacenan en texto plano. En su lugar, se utiliza un objeto Secret de Kubernetes, creado imperativamente con el comando kubectl create secret. Este objeto almacena de forma segura las credenciales y permite que sean inyectadas en los contenedores como variables de entorno.

3.2.2. Manifiesto de Despliegue (Deployment)

El archivo deployment.yaml define un objeto Deployment, responsable de mantener un conjunto de réplicas de la aplicación en ejecución. Sus directivas principales incluyen la referencia a la imagen de Docker (sonidocristalino/microtalent:latest), la definición del número de réplicas para asegurar la disponibilidad, y la inyección de las credenciales de la base de datos desde el Secret previamente creado.

Código 2: deployment.tf

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: microtalent-app
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: microtalent-container
          image: sonidocristalino/microtalent:latest
          ports:
            - containerPort: 80
          envFrom:
            - secretRef:
                name: db-credentials
```

3.2.3. Manifiesto de Servicio (Service)

Finalmente, el archivo service.yaml define un objeto Service de tipo LoadBalancer. Esta definición instruye a AWS para que aprovisione un Balanceador de Carga de Red público. El

servicio utiliza un selector para dirigir el tráfico entrante del puerto 80 a los Pods que coinciden con la etiqueta `app: microtalent-app`, exponiendo así la aplicación de forma segura y fiable a internet.

Código 3: service.tf

```
apiVersion: v1
kind: Service
metadata:
  name: microtalent-service
spec:
  type: LoadBalancer
  selector:
    app: microtalent-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

4. Guía de Uso

Esta sección detalla los requerimientos y procedimientos necesarios para el despliegue y la posterior eliminación de la infraestructura y aplicación definidas en este proyecto. Se asume que el usuario tiene un conocimiento básico de las herramientas de línea de comandos.

4.1. Prerrequisitos

Antes de iniciar el despliegue, es necesario contar con los siguientes componentes de software y configuraciones:

- Una cuenta de Amazon Web Services (AWS) con los permisos adecuados para crear los recursos definidos (EKS, VPC, RDS, IAM).
- La Interfaz de Línea de Comandos de AWS (`aws-cli`) instalada y configurada con credenciales de acceso válidas.
- La herramienta Terraform en su versión 1.0 o superior, instalada en el sistema local.
- La herramienta de línea de comandos de Kubernetes, `kubectl`, instalada.
- Git, para la clonación del repositorio de código fuente.

4.2. Procedimiento de Creación del Entorno

El despliegue se realiza en dos etapas principales: el aprovisionamiento de la infraestructura con Terraform y el despliegue de la aplicación con `kubectl`.

1. **Clonar el Repositorio:** Obtener una copia local del proyecto desde su repositorio de control de versiones.
2. **Configurar Variables:** Crear un archivo de variables denominado `terraform.tfvars` dentro del directorio `infra/`. Este archivo debe contener la contraseña para la base de datos, con el siguiente formato:

```
db_password = "SU_CONTRASENA_SEGURA_AQUI"
```

3. **Aprovisionar la Infraestructura:** Navegar al directorio `infra/` y ejecutar los siguientes comandos de Terraform en secuencia:
   - `terraform init`: Para inicializar el directorio de trabajo, descargar los proveedores y módulos necesarios.
   - `terraform plan`: (Opcional) Para previsualizar los cambios que se aplicarán en la infraestructura.
   - `terraform apply`: Para crear los recursos en AWS. El proceso requerirá una confirmación manual explícita (`yes`).
4. **Configurar kubectl:** Una vez finalizado el comando anterior, Terraform mostrará el comando para configurar `kubectl` en la sección de salidas (`Outputs`). Copiar y ejecutar dicho comando para establecer la conexión con el nuevo clúster de EKS.
5. **Desplegar la Aplicación:** Desde el directorio raíz del proyecto, realizar el despliegue en el clúster:
   - Crear el secreto de Kubernetes con las credenciales de la base de datos. Es necesario reemplazar los valores correspondientes al `endpoint` de RDS y la contraseña.

```
kubectl create secret generic db-credentials \
--from-literal=DB_HOST='ENDPOINT_RDS_AQUI' \
--from-literal=DB_USER='dbadmin' \
--from-literal=DB_PASSWORD='SU_CONTRASENA_AQUI'
```

   - Aplicar los manifiestos de Kubernetes para crear el `Deployment` y el `Service`:

```
kubectl apply -f kubernetes/
```

6. **Verificación Final:** Validar que el servicio se ha desplegado correctamente y obtener la URL pública del balanceador de carga con el comando:

```
kubectl get service microtalent-service
```

   La URL aparecerá en la columna `EXTERNAL-IP` después de unos minutos.

4.3. Procedimiento de Destrucción del Entorno

Este procedimiento eliminará de forma permanente todos los recursos creados. Se debe realizar para evitar costos innecesarios en la cuenta de AWS.

1. **Eliminar la Aplicación:** Desde el directorio raíz del proyecto, eliminar todos los recursos de Kubernetes del clúster:

```
kubectl delete -f kubernetes/
```

2. **Destruir la Infraestructura:** Navegar al directorio `infra/` y ejecutar el comando de destrucción de Terraform. El proceso requerirá una confirmación manual explícita (`yes`).

```
terraform destroy
```

5. Conclusión y Cumplimiento de Requisitos

El proyecto ha cumplido con todos los objetivos y requisitos establecidos en el enunciado. La infraestructura fue desplegada y validada, y la aplicación contenedorizada se expuso correctamente a través de internet con conectividad a la base de datos. A continuación, se detalla el cumplimiento de cada uno de los puntos especificados en la sección de notas del desafío.

- **Conexión Docker y Base de Datos:** La conectividad entre el contenedor y la base de datos se garantiza a nivel de red. La instancia de RDS reside en una subred privada (como se puede apresiar en la Figura 1) y su Grupo de Seguridad asociado está configurado para permitir el tráfico entrante en el puerto `5432` exclusivamente desde el Grupo de Seguridad de los nodos de trabajo de EKS. La aplicación recibe las credenciales y el `endpoint` de la base de datos de forma segura a través de un `Secret` de Kubernetes.
- **Cuenta Propia:** La totalidad de la infraestructura fue desplegada en una cuenta personal de AWS, utilizando credenciales de IAM configuradas localmente para la autenticación a través de la AWS CLI.
- **Acceso a la Imagen de Docker:** La imagen de Docker generada fue publicada en un repositorio público en Docker Hub, accesible a través de la etiqueta `sonidocristalino/microtalent:latest`.
- **Acceso a los Archivos de Terraform:** Todos los archivos fuente del proyecto, incluyendo las configuraciones de Terraform, los manifiestos de Kubernetes y la documentación, se encuentran disponibles en un repositorio público de GitHub.
- **Justificación de Servicios:** Se seleccionó Amazon EKS como servicio de cómputo para delegar la gestión del plano de control de Kubernetes a AWS, garantizando un entorno robusto

y escalable. Para la base de datos, se eligió Amazon RDS por ser un servicio gestionado que elimina la carga administrativa, utilizando una instancia de bajo costo compatible con la capa gratuita.

- **Configuraciones Adicionales:** Como configuraciones adicionales que constituyen un plus, se implementó la gestión remota del estado de Terraform con S3 y DynamoDB para un entorno colaborativo seguro. Además, la regla de acceso SSH se configuró para usar dinámicamente la IP pública del operador, mejorando la seguridad en comparación con una regla estática.
- **Ejecución de la Prueba:** El entorno se despliega de forma **manual** siguiendo la guía de uso provista. El proceso se inicia con los comandos de Terraform (init, plan, apply) y finaliza con los comandos de kubectl (create secret, apply). La arquitectura está diseñada para ser plenamente compatible con una futura automatización mediante un **pipeline de CI/CD**.
- **Gestión del state file:** El archivo de estado (terraform.tfstate) se almacena de forma remota en un bucket de Amazon S3. Esta configuración centralizada es esencial para el trabajo en equipo y previene la pérdida de estado. La concurrencia se gestiona mediante una tabla de DynamoDB que bloquea el archivo de estado durante las operaciones de escritura.
- **Gestión del archivo .lock:** El archivo .terraform.lock.hcl se incluye en el control de versiones para garantizar la reproducibilidad. Este archivo registra las versiones exactas de los proveedores utilizados, asegurando que todos los despliegues, independientemente del entorno, sean consistentes y predecibles.

Anexo A. Enunciado del Challenge

A.1. MicroTalent Challenge

Construir un proyecto en Terraform que permita crear una infraestructura en AWS que cumpla con los siguientes requisitos:

1. Crear un servicio de cómputo con SO Linux (EC2, ECS, EKS, etc...).
2. Crear un servicio de base de datos (RDS, DynamoDB, Aurora, etc ...).
3. Crear políticas de seguridad para el acceso SSH (por ejemplo, restricción de IPs).
4. Construir una imagen de Docker con SO Linux y publicarla, la imagen debe contar con las siguientes especificaciones:
   a) Instalar Git.
   b) Instalar Vs Code.
   c) Instalar Maven.
   d) Instalar PostgreSQL.

e) Instalar Java JRE.

f) Debe poder compilar proyectos NetCore.

g) Debe poder compilar aplicaciones Java.

h) Subir un servidor Apache con un "hola mundo" o cualquier proyecto público.

5. Montar la imagen de Docker creada en el servicio de cómputo elegido en el paso 1.

A.2. Notas

1. Debe haber conexión entre el Docker y el servicio de base de datos elegido.
2. Debe ser elaborado en una cuenta propia (debe explorar sus opciones).
3. Concedernos acceso a la imagen de Docker publicada.
4. Concedernos acceso a los archivos de Terraform creados.
5. Justificar por qué el tipo de servicio de cómputo y base de datos elegidos.
6. Es un plus cualquier configuración adicional realizada.
7. Explicarnos cómo la prueba será ejecutada (manualmente o con un pipeline).
8. Explique el uso y gestión del state file y dónde está almacenado.
9. Explique la gestión del archivo .lock en Terraform para esta evaluación.
10. Para la entrega del reto, no es necesario tener los recursos desplegados en la nube de AWS; de nuestro lado necesitamos solo acceso al repo de GitHub por ejemplo o enviarnos un archivo comprimido con la solución completa.