Paso a paso

Reto técnico NightWatch

Javier Almeida
Despliegue de Infraestructura

Visión general

Los recursos desplegados para este proyecto fueron creados en AWS utilizando Terraform.
A continuación se detallan los principales componentes:
Recursos de infraestructura:

VPC
Security Groups
EC2 Instances
S3 Bucket
(Opcional) **Route 53**

Scripts adicionales:
`grafana_user_data.sh` y **`postgres_user_data.sh`**: Scripts utilizados para configurar automáticamente los servicios necesarios en cada instancia EC2 (Grafana y PostgreSQL).

`nginx-deployment.yaml`: Manifiesto de Kubernetes utilizado con K3s para desplegar un servidor NGINX y Prometheus.

`ec2_logs_to_s3.py`: Script en Python que utiliza la librería `boto3` para recolectar métricas desde CloudWatch y enviarlas al bucket S3.

`setup_scripts.sh`: Copia los scripts a la instancia pública, los ejecuta, y configura su ejecución diaria mediante `crontab`.

`test_private_ec2.sh`: Realiza pruebas de conectividad entre las instancias EC2 utilizando nc (netcat) y PostgreSQL.

Funcionamiento del repositorio de Github

Utilicé GitHub para organizar y versionar correctamente los recursos del proyecto. Para ello, creé una rama (`branch`) por cada hito importante del despliegue. Como ejemplo, la primera rama fue *`setup_vpc`*, en la cual comencé estructurando el código utilizando una arquitectura modular con Terraform.

Cada módulo está separado en su propio directorio bajo la carpeta `/modules`, comenzando con `/modules/vpc`. En cada uno de estos módulos se sigue el patrón de buenas prácticas de Terraform, estructurando los archivos en:

`main.tf`: definición de los recursos principales.
`variables.tf`: declaración de variables.
`outputs.tf`: salidas relevantes del módulo.

En el directorio raíz del proyecto, integré todos los módulos para desplegar la infraestructura completa mediante los archivos:

```
main.tf
variables.tf
outputs.tf
```

Este enfoque modular mejora la reutilización, el mantenimiento y la escalabilidad del código.

AWS VPC

Para definir los recursos a crear, me basé en los requisitos planteados por el proyecto, los cuales incluían:

Una VPC para desplegar los recursos necesarios.
Salida a Internet tanto pública como privada.
Dos subredes públicas y dos subredes privadas.

Gracias a mi experiencia previa con AWS VPC y el despliegue de infraestructura desde cero utilizando Terraform, ya contaba con los conocimientos necesarios para estructurar esta solución.

Decidí crear una VPC con un bloque CIDR amplio (`10.0.0.0/16`) que permita flexibilidad y escalabilidad a futuro. Para proporcionar salida a Internet, incluí un Internet Gateway (para las subredes públicas) y un NAT Gateway asociado a una Elastic IP, lo que permite que las instancias en subredes privadas también puedan acceder a Internet de forma segura.

Las subredes fueron distribuidas en dos zonas de disponibilidad distintas (`us-east-1a` y `us-east-1b`) tanto para las públicas como para las privadas, con el objetivo de garantizar mayor disponibilidad y tolerancia a fallos.

Finalmente, configuré tablas de ruteo para asociar las subredes públicas al Internet Gateway y las privadas al NAT Gateway (ubicado en la primera subred pública), asegurando así la conectividad necesaria entre todos los componentes.

AWS SECURITY GROUPS

Para asegurar la correcta comunicación entre las instancias y limitar accesos no deseados, preparé dos grupos de seguridad:

Security Group Instancia Pública:
Asociado a la instancia pública (bastion host + Grafana). Permite tráfico entrante desde Internet por los siguientes puertos:
Puerto 22 (SSH)
Puerto 3000 (Grafana)
Puerto 30080 (Nginx)
Puerto 31080 (Prometheus)

Security Group Instancia Privada:
Asociado a la instancia privada (PostgreSQL). No permite acceso directo desde Internet. Sólo acepta tráfico entrante desde el Security Group Público, específicamente en los puertos:

Puerto 22 (SSH): para conexión remota desde la instancia pública
Puerto 5432: para acceso a la base de datos PostgreSQL

Este enfoque garantiza una separación clara entre las instancias públicas y las privadas, y permite un canal seguro de administración y conexión a través del bastión host.

El SG de la Instancia pública fue teniendo actualizaciones en el transcurso del codeo.

AWS EC2

Para esta infraestructura se desplegaron dos instancias EC2, ambas con la AMI de Amazon Linux AMI 2023:

Instancia pública: Se encuentra en una subnet pública. Funciona como **bastion host** y a su vez aloja **Grafana** y **K3s** (Kubernetes ligero) para desplegar servicios y visualizar métricas. Esta instancia fue creada con tipo `t3.small` para asegurar la capacidad suficiente frente a las tareas de monitoreo y orquestación de contenedores.

Instancia privada: Ubicada en una subnet privada, se destinó al alojamiento de una base de datos **PostgreSQL**. Utiliza una instancia `t2.micro`, adecuada para una carga liviana en un entorno aislado.

Además, se definió un **IAM Role** específico (`grafana-ec2-role`) asignado a la instancia pública. Este rol le otorga permisos para recolectar métricas desde **CloudWatch** y almacenarlas en un bucket de **S3**.

La configuración inicial de ambas instancias fue automatizada mediante scripts *`user_data`*:

- *`grafana_user_data.sh`*: Instala Docker, Python, K3s, y configura la instancia como nodo maestro de Kubernetes, levantando automáticamente Grafana.

- *`postgres_user_data.sh`*: Instala PostgreSQL y lo configura como base de datos interna dentro de la subnet privada.

Esta lógica puede encontrarse dentro del módulo `modules/ec2/`, donde se definen los parámetros, recursos, y relaciones de red y seguridad.

AWS S3

Se creó un bucket S3 llamado **`nightwatch-logs-backups-<SUFIJO>`** para alojar:

Backups diarios de la base de datos PostgreSQL.
Métricas diarias extraídas desde CloudWatch utilizando scripts en Python.

Estos datos se almacenan en carpetas separadas (`metrics/` y `backups/`) dentro del bucket. Las tareas fueron automatizadas con `cron` en la instancia pública.

KUBERNETES (K3S)

Se utilizó **K3s**, una versión ligera de Kubernetes, instalada en la instancia pública. En ella se desplegaron los siguientes recursos mediante un manifiesto YAML único:

Un **Deployment de Nginx** con un **Service NodePort** en el puerto 30080.

Un **Deployment de Prometheus**, también expuesto por NodePort en el puerto 31080.
Un **ConfigMap** con la configuración personalizada de Prometheus.

Prometheus fue configurado para recolectar métricas del mismo clúster y se conectó exitosamente con Grafana como fuente de datos. Se verificó su funcionalidad desde la interfaz web

Una vez instalado K3s en la instancia pública, se utilizó `kubectl` para gestionar y desplegar recursos dentro del clúster.

La configuración de `KUBECONFIG` en el archivo `.bashrc` permitió que el comando `kubectl` funcione directamente.

Los manifiestos YAML fueron aplicados con *`kubectl apply -f nginx-deployment.yaml`*, los cuales definieron un servidor Nginx, un servidor Prometheus y su configuración personalizada mediante un `ConfigMap`.

Además, se pueden utilizar comandos como: *`kubectl get pods`*, *`kubectl get svc`*, *`kubectl logs`* y *`kubectl describe pod`* para verificar el estado del clúster, resolver errores y asegurar que los servicios estén corriendo correctamente.

SCRIPTS

Se utilizaron distintos scripts Bash y Python para automatizar tareas críticas:

ec2_logs_to_s3.py: extrae métricas de CloudWatch mediante boto3 y las sube a S3.
postgresql_daily_backup.sh: conecta a la base de datos y guarda un dump diario en S3.
setup_scripts.sh: hace ssh a la instancia, copia, da permisos y configura **`cron`** para ejecutar los scripts automáticamente.
test_private_ec2.sh: permite verificar la conexión entre subredes (prueba de nc, conexión a PostgreSQL, etc).

README

El repositorio de GitHub está organizado modularmente con Terraform y contiene un README que explica:

Cómo inicializar y aplicar la infraestructura.
Estructura de carpetas y scripts.

Detalles del despliegue de VPC, EC2, S3, Kubernetes y monitoreo.

Archivos `.tf` separados por módulos (`/modules/vpc`, `/modules/ec2`, etc.).

Arquitecturas sobre la infraestructura y sobre la VPC.

Cada recurso tiene su propia definición, variables parametrizadas y outputs correctamente estructurados.

PASO 3: COORDINACIÓN Y COMUNICACIÓN

Durante el desarrollo del reto, documenté cada decisión técnica tomada y sus fundamentos, asegurando trazabilidad en el repositorio mediante commits claros y una estructura modular con Terraform.

En relación al incidente planteado —**fallo en la conexión entre subredes públicas y privadas**— abordé la situación siguiendo un enfoque sistemático:

1. Revisión de la configuración de red:
Verifiqué que las subredes estuvieran correctamente asociadas a sus respectivas tablas de ruteo. Confirmé que:

Las subredes públicas estuvieran conectadas al Internet Gateway.
Las subredes privadas usarán el NAT Gateway para salida a Internet.

2. Validación de Security Groups:
Aseguré que el *Security Group* de la instancia privada (PostgreSQL) permitiera tráfico entrante desde la instancia pública únicamente en los puertos necesarios (SSH y PostgreSQL). La conexión fue restringida solo al *Security Group* de origen, lo que evita aperturas innecesarias al exterior.

3. Pruebas de conectividad:
Implementé el script `test_private_ec2.sh`, el cual permitió verificar:

La conectividad entre las instancias usando nc (netcat).
El acceso al servicio de PostgreSQL desde la instancia pública.

RECURSOS EXTERNOS

Durante la creación de este proyecto utilicé diversos recursos de documentación y herramientas que facilitaron el desarrollo y la implementación de la infraestructura:

Documentación de Terraform sobre AWS:

VPC:

VPC

Subnets

Route Table

Route Table Association

Security Groups:

Security Groups

EC2:

Instance

Elastic IP

S3:

S3 Bucket

(opcional) Route53:
Route53 records

Kubernetes:

K3s:

Quick-start

Comandos de kubectl:
kubectl

Manifest nginx:
Manifest example

Asistencia en redacción y solución de problemas técnicos con IA:
chatgpt:
chatgpt

Creación de diagramas de infraestructura:
draw.io:
diagrams