Lectura 16

Patrones de diseño aplicados a módulos de Infraestructura de Terraform

1. Contextualización y objetivos: el objetivo es ilustrar como cada patrón de diseño facilita tareas concretas:

    1. Evitar instancias duplicadas (singleton)
    2. Representar jerarquías compuestas (Composite)
    3. Desacoplar la lógica de creación (Factory)
    4. Reutilizar configuraciones base (prototype)
    5. Construir objetos complejos por pasos (Builder)

2. Patrón singleton: Instancia única garantizada

    Principio y utilidad: este patrón asegura que una clase tenga una única instancia durante el ciclo de vida de la aplicación

    Beneficios: Evita que se declare accidentalmente una VPC y simplifica acceso a la configuración

    Limitaciones: oculta estado en variables estáticas

3. Patrón Composite: Jerarquías recursivas

    El patron composite trata de manera homogénea objetos invisibles y conjunto de objetos se basa en:

    1. Componente comun: una interfaz o clase abstracta que define la operación.
    2. Nodo hoja: implementa la interface y realiza la acción concreta.
    3. Nodo compuesto: contiene una colección de componentes.

4. Patrón factory: delegar creación

    Este patrón define una interfaz para crear un objeto como terraform json y delega responsabilidad concreta a subclases.

    - Database factory genera el recurso de base de datos
    - Network factory crea la red
    - Server factory Construye instancias del servidor

5. Patrón Prototype : clonación de plantillas

Este patrón permite crear nuevos objetos copiando "clonando" una instancia prototipo y modificando solo los atributos necesarios, esto es útil cuando los bloques Json comparten una configuración base compleja, este patrón minimiza la duplicacion de código

6. Patron Builder: Construccion paso a paso

Builder abstrae la creación de un objeto complejo mediante una secuencia de pasos encadenados

7. Criterios para usar el patrón adecuado :

1. Grado de Reutilización :

- si se necesita declarar un recurso global una vez, singleton
- si se necesita clonar multiples configuraciones, prototype

2. Complejidad de configuración :

- pocas propiedades y construcciones directas: Factory.
- muchas opciones y validaciones : Builder

3. Estructura jerarquica :

- varios capas de dependencias : Composite

4. Escalabilidad y mantenimiento :

- proyectos pequeños o pilotos : Factory - Prototype
- sistemas empresariales con múltiples módulos: Composite-builder

5. Pruebas y Validación :

- Facilita test unitarios evitando singleton ocultos : Builder - factory son mas testables

6. Evducción del Proyecto :

- Si la infraestructura va a crecer y diversificarse, invierte en composite + Builder
- Para scripts rápidos o prototipos : factory - prototype ofrecen simplicidad