Informe Fase 1 — Relaciones unidireccionales

Este informe resume la inspeccion de los artefactos generados por el módulo network y por el script principal (main.tf.json), describe los recursos detectados y las dependencias implicitas, las observaciones al ejecutar terraform apply y make all.

1. Inspección de artefactos
   - **network/network.tf.json** contiene:
     - null_resource.network con triggers que fuerzan recreación periódica.
     - local_file.network_state (archivo JSON) que persiste los outputs publicados por el módulo de red.
   - **main.tf.json** (generado por **Inversion_control/main.py**) contiene:
     - null_resource para el servidor (identificado como **hello-world**) con un bloque **triggers** que incluye valores inyectados desde la red: **subnet_name**, **subnet_cidr**, **zone** y en la implementación actual, un campo **server_config** con JSON serializado.

2. Dependencias implícitas (depends_on)
   - No siempre hay un **depends_on** explícito en Terraform: la dependencia puede expresarse por lectura de outputs; en este conjunto:
     - El servidor consume outputs escritos por **network** (lectura del archivo **network_outputs.json**), lo que crea una dependencia logica unidireccional: la red debe producir outputs antes de que el servidor los use.
     - En algunas implementaciones, los modulos tambien ponen referencias textuales en **triggers** o un campo **depends_on** con la referencia del recurso.

3. Ejecución práctica (comandos ejecutados)
   - **cd network, terraform init, terraform apply -auto-approve, cd .., make all**

   Observaciones en la ejecución

   - Orden de creación observado: primero se aplica el módulo **network** (crea **null_resource.network** y escribe **network_outputs.json**), luego **make all** genera **main.tf.json** y aplica el servidor.
   - Orden de destruccion observado: al ejecutar **terraform destroy**, el servidor se destruye antes que la red, respetando la dependencia unidireccional (consumidor -> proveedor).

4. Conclusiones
    - La separación unidireccional mejora la modularidad: cada modulo tiene responsabilidades claras y se testea aisladamente.
    - Riesgos/consideraciones:
        - Es necesario asegurar la publicación estable de outputs (formatos, nombres de campos).
        - Si la dependencia es crítica, conviene declarar **depends_on** explícito o usar mecanismos de orquestación (**ej. make o scripts**) para garantizar el orden.