TERRAFORM ASSOCIATE (004) · CHEAT SHEET · CERTIFYDECK

HashiCorp Certified: Terraform Associate (004)

Dominios, pesos oficiales y habilidades clave del temario de Terraform Associate (004), para repasar antes del examen.

Nivel: Beginner Duración: 60 min Fuente: guía de estudio oficial de Microsoft Learn

1. Infrastructure as Code (IaC) with Terraform

1a Explain what IaC is

- La infraestructura como código (IaC) gestiona la infraestructura con archivos de configuración en lugar de una interfaz gráfica, para versionarla, reutilizarla y compartirla. Terraform es declarativo: describes el estado final y él calcula los pasos, trabaja con las API de cada plataforma mediante proveedores y usa el archivo de estado como fuente de verdad. No es una herramienta de gestión de la configuración.

1b Describe the advantages of IaC patterns

- Los patrones de IaC permiten revisar los cambios en el control de versiones como el código, reutilizar módulos que codifican estándares, crear y destruir entornos desechables y revisar un plan antes de ejecutarlo. Con un backend remoto como HCP Terraform se comparte el estado sin condiciones de carrera, y políticas como Sentinel sustituyen las revisiones por tickets.

1c Explain how Terraform manages multi-cloud, hybrid cloud, and service-agnostic workflows

- Terraform usa el mismo lenguaje y el mismo flujo de trabajo con cualquier proveedor, así que una configuración puede combinar varias nubes, plataformas locales con API y servicios SaaS como DNS o CDN, y gestiona las dependencias entre ellos. Cada proveedor aporta sus propios tipos de recurso; no hay recursos universales. Desplegar en varias nubes aumenta la tolerancia a fallos.

2. Terraform fundamentals

2a Install and version Terraform providers

- Los proveedores se declaran en required_providers con source y version, y terraform init los instala en .terraform/providers. El archivo .terraform.lock.hcl registra la versión elegida y sus checksums (no los módulos), se confirma en Git y se reutiliza en cada init; terraform init -upgrade busca versiones nuevas. Los módulos reutilizables fijan solo un mínimo (>=) y el módulo raíz usa ~>.

2b Describe how Terraform uses providers

- Los proveedores son plugins en Go que Terraform Core ejecuta como procesos aparte por RPC; se autentican con la plataforma y definen sus recursos y fuentes de datos. Cada uno tiene una dirección de origen y un nombre local, y se configura en un bloque provider; alias permite varias configuraciones del mismo proveedor, que se eligen con provider = NOMBRE.ALIAS.

2c Write Terraform configuration using multiple providers

- Una configuración puede combinar varios proveedores declarados en required_providers con nombres locales únicos (compuestos si dos comparten tipo). Para varias configuraciones del mismo proveedor se repite el bloque provider con alias y se selecciona con provider = NOMBRE.ALIAS. Los módulos hijo heredan solo la configuración por defecto; las configuraciones con alias se pasan con el mapa providers y se declaran con configuration_aliases.

2d Explain how Terraform uses and manages state

- El estado vincula cada instancia de recurso con su objeto remoto, guarda metadatos como las dependencias más recientes (para destruir en orden lo eliminado) y una caché de atributos que -refresh=false usa como fuente de verdad. Es JSON en texto plano y puede contener secretos: no se edita a mano ni se sube a Git; para software externo se usa terraform show -json. El estado remoto permite trabajar en equipo sobre los mismos datos.

3. Core Terraform workflow

3a Describe the Terraform workflow

- El flujo principal de Terraform tiene tres etapas que se repiten en bucle: Write (escribir la configuración, con planes frecuentes como retroalimentación), Plan (previsualizar los cambios sin tocar la infraestructura) y Apply (aplicar tras aprobar el plan). En equipo se revisan planes especulativos en las pull requests y, tras la fusión, el plan final contra el estado más reciente, que puede ser distinto.

3b Initialize a Terraform working directory

- terraform init prepara el directorio de trabajo: inicializa el backend, descarga módulos, instala proveedores en .terraform y crea o actualiza .terraform.lock.hcl, que se confirma en Git (.terraform no). Es idempotente y hay que repetirlo al cambiar proveedores, módulos o backend. Opciones clave: -upgrade, -migrate-state, -reconfigure, -backend-config para configuración parcial y -plugin-dir.

3c Validate a Terraform configuration

- terraform validate comprueba que la configuración sea sintácticamente válida e internamente coherente (nombres de atributos, tipos de valores, referencias, ciclos), sin consultar el estado remoto ni las API de los proveedores. Necesita un directorio inicializado; terraform init -backend=false lo prepara sin tocar el backend. Para comprobar la configuración con variables y estado reales se usa terraform plan.

3d Generate and review an execution plan for Terraform

- terraform plan lee el estado actual de los objetos remotos, compara la configuración con el estado previo y propone las acciones necesarias sin ejecutarlas. Con -out guarda un plan que luego aplica terraform apply; sin -out el plan es especulativo. Ofrece los modos -destroy y -refresh-only y opciones como -target, -replace, -var, -var-file o -detailed-exitcode, y terraform show permite revisar un plan guardado.

3e Apply changes to infrastructure with Terraform

- Sin un plan guardado, terraform apply crea un plan nuevo, pide aprobación (o la omite con -auto-approve) y ejecuta las acciones; con un plan guardado lo aplica sin preguntar y no admite más opciones de planificación. Durante el apply bloquea el estado, crea en paralelo lo que no depende entre sí y actualiza el estado. Si falla, registra lo ya hecho y no revierte nada: hay que corregir y volver a aplicar.

3f Destroy Terraform-managed infrastructure

- terraform destroy elimina todos los objetos que gestiona la configuración y es un alias de terraform apply -destroy; pide confirmar con yes y deja el estado vacío. terraform plan -destroy muestra el efecto sin ejecutarlo y, con -out, genera un plan que se aplica con terraform apply. Destruye los dependientes antes que sus dependencias, y prevent_destroy rechaza cualquier plan que destruya el recurso mientras el bloque siga en la configuración.

3g Apply formatting and style adjustments to a configuration

- terraform fmt reescribe los archivos con el formato canónico (sangría de dos espacios, signos igual alineados) y no admite personalización. Por defecto solo trata el directorio actual; -recursive incluye subdirectorios, -check falla sin escribir si hay archivos mal formateados, -diff muestra los cambios y -write=false evita sobrescribir. La guía de estilo añade convenciones como metaargumentos primero y comentarios con #.

4. Terraform configuration

4a Use and differentiate resource and data blocks

- Un bloque resource declara un objeto de infraestructura que Terraform crea, actualiza y destruye, y lo registra en el estado con la dirección TIPO.NOMBRE. Un bloque data solo lee datos a través de un proveedor y se referencia con data.TIPO.NOMBRE.ATRIBUTO; Terraform lo lee antes de planificar, salvo que dependa de valores que solo se conocen en apply, y admite count, for_each, depends_on, provider y condiciones.

4b Refer to resource attributes and create cross-resource references

- Un atributo se referencia con TIPO.NOMBRE.ATRIBUTO; con count el recurso es una lista de instancias (aws_instance.x[0].id o [*].id) y con for_each un mapa por clave (aws_instance.x["a"].id o values(...)[*].id). Los outputs de un módulo hijo se leen con module.NOMBRE.OUTPUT, y cada referencia entre recursos crea una dependencia implícita; lo que no se conoce hasta apply aparece como (known after apply).

4c Use variables and outputs

- Las variables de entrada (var.NOMBRE) parametrizan un módulo: sin default son obligatorias, y su valor sale, de mayor a menor precedencia, de -var y -var-file, *.auto.tfvars, terraform.tfvars.json, terraform.tfvars, variables TF_VAR_ y el default. Los bloques output, con value obligatorio, exponen valores al módulo padre, a la CLI y a terraform_remote_state; terraform output -raw y -json sirven para scripts.

4d Understand and use complex types

- Los tipos complejos son colecciones (list, map y set, con un único tipo de elemento) y estructurales (object y tuple, con un tipo por atributo o posición). Terraform convierte entre tipos similares: una lista pasa a set perdiendo duplicados y orden, y un object descarta los atributos que no declara. Un set no se indexa sin tolist, y optional(tipo, default) aplica el default aunque llegue null.

4e Write dynamic configuration using expressions and functions

- Las expresiones for transforman colecciones: con corchetes producen una tupla, con llaves y => un objeto, y admiten filtros if. Las expresiones splat ([*]) solo valen para listas, sets y tuplas, y los bloques dynamic generan bloques anidados repetidos, nunca lifecycle. Las funciones integradas (merge, coalesce, flatten, file...) se prueban en terraform console y no participan en el grafo de dependencias.

4f Define resource dependencies in configuration

- Terraform deduce dependencias implícitas cuando un argumento hace referencia a otro objeto, y crea, actualiza y destruye siguiendo ese grafo, en paralelo cuando puede. depends_on declara dependencias ocultas con una lista de referencias estáticas y debe ser el último recurso. Por defecto un reemplazo destruye y luego crea; create_before_destroy invierte el orden y se propaga a las dependencias.

4g Validate configuration using custom conditions

- La validación de variables se ejecuta antes del plan; las precondiciones comprueban supuestos antes de crear un recurso, data u output, y las postcondiciones garantías después, con self. Todas detienen la operación si fallan. Los bloques check se evalúan al final de plan y apply y solo avisan; sus data anidados son locales al bloque. HCP Terraform puede revalidarlos de forma continua.

4h Understand best practices for managing sensitive data, including secrets management with Vault

- sensitive = true oculta valores en la salida de la CLI y en HCP Terraform, pero se siguen guardando en el estado y el plan. Las variables y outputs ephemeral, los bloques ephemeral y los argumentos de solo escritura (write-only) nunca se guardan. Lo que lee un data source del proveedor de Vault acaba en el estado; su recurso ephemeral lo evita. Protege el estado: remoto, cifrado y con acceso restringido.

5. Terraform modules

5a Explain how Terraform sources modules

- El argumento source del bloque module es una cadena literal que indica de dónde sale el código: rutas locales con ./ o ../, el registro público (NAMESPACE/NAME/PROVIDER), el registro privado de HCP Terraform (app.terraform.io/ORG/NAME/PROVIDER), Git con ref y subdirectorios //, URLs HTTP y archivos comprimidos. terraform init descarga los módulos remotos en .terraform.

5b Describe variable scope within modules

- Cada módulo tiene su propio ámbito: un módulo hijo solo ve sus variables de entrada, que el padre le pasa como argumentos del bloque module, y sus valores locales no son visibles fuera de él. Las variables -var, TF_VAR_ y tfvars solo asignan variables del módulo raíz. Para devolver datos, el hijo declara outputs que el padre lee con module.NAME.OUTPUT.

5c Use modules in configuration

- Un bloque module llama a un módulo hijo con source, sus argumentos de entrada y metaargumentos como count, for_each, depends_on y providers. Las configuraciones de proveedor por defecto se heredan, pero las que llevan alias se pasan con providers. Tras cambiar source o version hay que ejecutar terraform init; moved reubica recursos sin destruirlos y removed con destroy = false los olvida.

5d Manage module versions

- El argumento version solo vale para módulos de un registro y acepta restricciones como =, !=, >=, < o ~>; sin él, Terraform toma la versión más reciente. El archivo .terraform.lock.hcl no registra módulos, así que para fijar uno de terceros se usa una versión exacta. terraform init no cambia los módulos ya instalados; terraform init -upgrade los actualiza dentro de la restricción.

6. Terraform state management

6a Describe the local backend

- Si no hay bloque backend ni bloque cloud, Terraform usa el backend local: guarda el estado en terraform.tfstate, relativo al módulo raíz, lo bloquea con APIs del sistema y ejecuta las operaciones en local. Los workspaces que no son el predeterminado van a terraform.tfstate.d (o a workspace_dir). El archivo es texto plano con posibles secretos: exclúyelo de Git.

6b Describe state locking

- Si el backend lo admite, Terraform bloquea el estado de forma automática y silenciosa en toda operación que pueda escribirlo; si no consigue el bloqueo, no continúa. -lock-timeout reintenta durante un tiempo y -lock=false lo desactiva (no recomendado). terraform force-unlock LOCK_ID libera solo tu propio bloqueo cuando falló el desbloqueo automático y no toca la infraestructura.

6c Configure remote state using the backend block

- El bloque backend va anidado en el bloque terraform; solo se admite uno, no puede usar variables ni locals y no convive con un bloque cloud. Con configuración parcial, el bloque queda vacío y terraform init -backend-config aporta el resto, con prioridad sobre la configuración. Pasa las credenciales por variables de entorno: -backend-config las deja en .terraform y en los planes.

6d Manage resource drift and Terraform state

- Para detectar la deriva sin tocar nada, usa terraform plan -refresh-only y, si aceptas el cambio externo, terraform apply -refresh-only, que solo actualiza el estado y los outputs; terraform refresh está obsoleto porque aplica sin confirmar. Un apply normal revierte la deriva. -replace recrea objetos degradados, moved renombra sin destruir y removed con destroy = false deja de gestionar sin destruir.

7. Maintain infrastructure with Terraform

7a Import existing infrastructure into your Terraform workspace

- El método recomendado es el bloque import (to e id), revisado con terraform plan y aplicado con terraform apply; terraform plan -generate-config-out escribe en un archivo nuevo un bloque resource inicial que conviene depurar. El comando terraform import solo escribe en el estado, no genera configuración y se ejecuta en local, sin acceso a las variables de un workspace de HCP Terraform.

7b Use the CLI to inspect state

- terraform state list muestra las direcciones de los recursos del estado (filtrables por dirección, módulo o -id) y terraform state show los atributos de un único recurso, en formato pensado para personas. terraform show presenta el estado o un plan guardado, con -json para herramientas; terraform state pull imprime el estado en bruto en stdout y terraform output lee los outputs del módulo raíz (-json o -raw).

7c Describe when and how to use verbose logging

- Los registros detallados sirven para depurar errores inesperados y para informar de fallos. TF_LOG los activa en stderr con los niveles TRACE, DEBUG, INFO, WARN o ERROR (JSON da TRACE en formato JSON no estable); TF_LOG_CORE y TF_LOG_PROVIDER separan Terraform y los proveedores, y TF_LOG_PATH añade la salida a un archivo, pero solo si hay un nivel definido.

8. HCP Terraform

8a Use HCP Terraform to create infrastructure

- HCP Terraform ejecuta las ejecuciones en un workspace mediante tres flujos: VCS (webhooks del repositorio y planes especulativos en pull requests), CLI (terraform login, bloque cloud y terraform init; plan y apply corren en remoto con la salida en el terminal) y API (tus herramientas suben la configuración en .tar.gz). En modo Local, el workspace solo guarda el estado.

8b Describe HCP Terraform collaboration and governance features

- HCP Terraform aplica policy sets de Sentinel u OPA a cada ejecución: advisory solo avisa, soft-mandatory y mandatory de OPA se pueden anular con el permiso adecuado y hard-mandatory no, salvo que el set lo permita. Los equipos reciben permisos acumulativos en organización, proyecto y workspace; el explorer, las change requests y las health assessments dan visibilidad y gobierno.

8c Describe how to organize and use HCP Terraform workspaces and projects

- Un workspace de HCP Terraform guarda la configuración, las variables, el estado y el historial de una colección de infraestructura, y cada workspace pertenece a un único proyecto (Default Project por defecto). Los proyectos agrupan workspaces con permisos, variable sets y modo de ejecución propios; los run triggers y el estado remoto compartido conectan workspaces.

8d Configure and use HCP Terraform integration

- El bloque cloud dentro de terraform conecta un directorio de trabajo con workspaces de HCP Terraform por nombre o etiquetas, y terraform login guarda el token de API. Con el flujo CLI, terraform plan lanza un plan especulativo remoto y terraform apply una ejecución remota; también existen los flujos VCS y API, y las credenciales dinámicas evitan secretos estáticos.