ΣΥΝΟΔΕΥΤΙΚΟΣ ΤΟΜΟΣ

Terraform

Λύσεις και απαντήσεις

Κάθε λύση εκτελέστηκε με Terraform 1.16.4 και η έξοδος είναι η πραγματική.



172 **λύσεις και 223 απαντήσεις**

Ο ΚΩΔΙΚΑΣ, Η ΕΞΟΔΟΣ ΚΑΙ ΤΟ ΓΙΑΤΙ

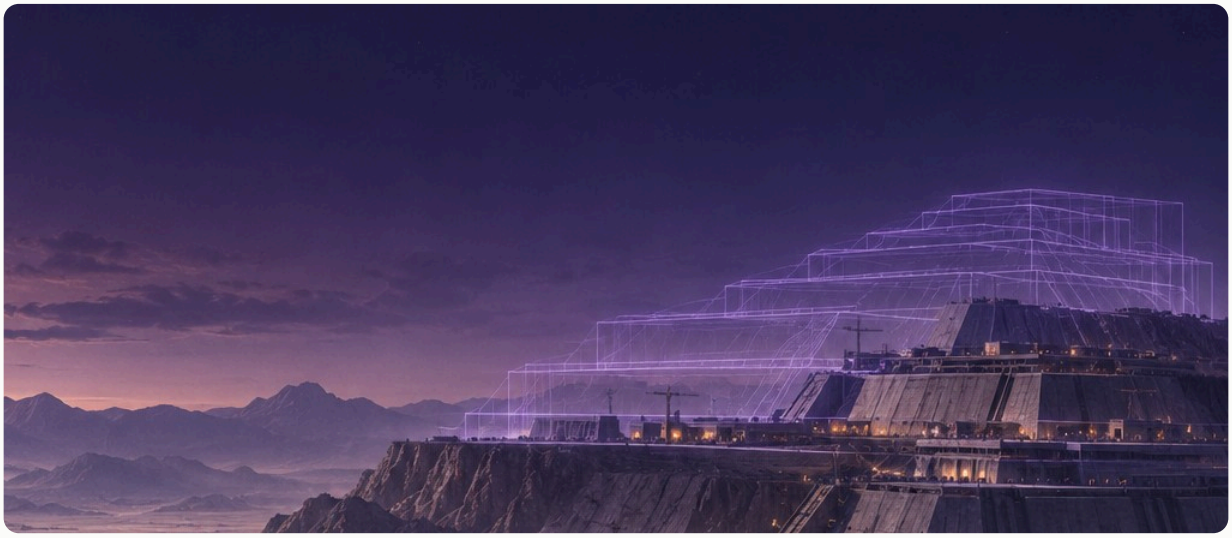
ΠΩΣ ΝΑ ΧΡΗΣΙΜΟΠΟΙΗΣΕΙΣ ΑΥΤΟΝ ΤΟΝ ΤΟΜΟ

Πρώτα η δική σου προσπάθεια. Μετά η σύγκριση.

Άνοιξε μια λύση αφού η δική σου configuration δώσει το αναμενόμενο αποτέλεσμα ή αφού κολλήσεις για αρκετή ώρα. Μια διαφορετική configuration που δίνει την ίδια έξοδο είναι συνήθως εξίσου σωστή. Η εξήγηση κάτω από κάθε λύση λέει τι πρέπει να ισχύει για να είναι σωστή και ποια παγίδα αποφεύγει.

Κάθε άσκηση λύθηκε σε δικό της καθαρό φάκελο, εκτός αν η εκφώνηση λέει να ξεκινήσεις από υπάρχοντα. Οι ασκήσεις των κεφαλαίων AWS εκτελέστηκαν στο Moto, οπότε τα IDs και τα ARNs της δικής σου εκτέλεσης θα διαφέρουν.

Οι ερωτήσεις κάθε κεφαλαίου ακολουθούν τις λύσεις του. Διάβασε την εξήγηση και όταν απάντησες σωστά. Στην εξέταση οι λάθος επιλογές είναι συνήθως σχεδόν σωστές και η εξήγηση λέει γιατί δεν είναι.



Ο ΧΑΡΤΗΣ ΤΟΥ ΒΙΒΛΙΟΥ

Περιεχόμενα

1 Infrastructure as Code και το πρώτο apply 4
2 Η γλώσσα HCL, fmt και validate 11
3 Το plan και οι αλλαγές 20
4 Το state 28
5 Providers, εκδόσεις και το lock file 40
6 Resources, references και dependencies 49
7 Input variables 59
8 Outputs και locals 70
9 Τύποι και δομές δεδομένων 82
10 Εκφράσεις και συναρτήσεις 90
11 count, for_each και dynamic blocks 98
12 Data sources 107
13 Validation, preconditions και check blocks 118
14 Lifecycle και αντικατάσταση 130
15 Το πρώτο module 140
16 Πηγές modules, εκδόσεις και Registry 151
17 Ο AWS provider 162
18 Δίκτυο και ένας web server 174
19 Backends, remote state και locking 186
20 Workspaces 195
21 Drift, refresh-only και εντολές state 203
22 Import 216
23 Refactoring με moved και removed 229
24 Secrets, sensitive, ephemeral και Vault 242
25 Troubleshooting και logging 253
26 HCP Terraform 263
27 Τελικό project 273
28 Προσομοίωση εξέτασης 282

ΚΕΦΑΛΑΙΟ 1

Infrastructure as Code και το πρώτο apply

Άσκηση 1.1 · Δύο αρχεία

Γράψε configuration με δύο resources `local_file`. Το πρώτο, με όνομα `morning`, φτιάχνει το `morning.txt` με περιεχόμενο `Καλημέρα\n`. Το δεύτερο, με όνομα `night`, φτιάχνει το `night.txt` με περιεχόμενο `Καληνύχτα\n`. Κάνε apply. Η περίληψη στο τέλος πρέπει να είναι η παρακάτω.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "morning" {
  filename = "morning.txt"
  content  = "Καλημέρα\n"
}

resource "local_file" "night" {
  filename = "night.txt"
  content  = "Καληνύχτα\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Δύο blocks `resource` στο ίδιο αρχείο δίνουν δύο resources. Τα ονόματα `morning` και `night` πρέπει να διαφέρουν, γιατί μαζί με τον τύπο ταυτοποιούν το resource. Δεν μετράει η σειρά των blocks στο αρχείο. Το Terraform μπορεί να φτιάξει τα δύο αρχεία με οποιαδήποτε σειρά, ακόμη και ταυτόχρονα.

Άσκηση 1.2 · Αρχείο σε υποφάκελο

Γράψε configuration που φτιάχνει το αρχείο `notes/today.txt` με περιεχόμενο `Πρώτη μέρα με Terraform\n`. Ο φάκελος `notes` δεν υπάρχει. Κάνε apply και μετά δείξε το περιεχόμενο με `cat`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "today" {
  filename = "notes/today.txt"
  content  = "Πρώτη μέρα με Terraform\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΛΥΣΗ · TERMINAL

```
cat notes/today.txt
```

ΕΞΟΔΟΣ

```
Πρώτη μέρα με Terraform
```

Ο provider `local` φτιάχνει μόνος του τους φακέλους που λείπουν από το path. Τα δικαιώματά τους τα ορίζει το argument `directory_permission` που είδες στο plan. Δεν χρειάζεται δεύτερο resource για τον φάκελο.

Άσκηση 1.3 · Μόνο plan

Γράψε configuration με τρία resources `local_file`, τα `a.txt`, `b.txt` και `c.txt`, με όποιο περιεχόμενο θέλεις. Πριν τρέξεις οτιδήποτε, γράψε σε χαρτί τι πιστεύεις ότι θα λέει η περίληψη του plan. Τρέξε μόνο `terraform plan` και έλεγξε ότι δεν δημιουργήθηκε κανένα αρχείο.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "a" {
  filename = "a.txt"
  content  = "a\n"
}

resource "local_file" "b" {
  filename = "b.txt"
  content  = "b\n"
}

resource "local_file" "c" {
  filename = "c.txt"
  content  = "c\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 3 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
ls
```

ΕΞΟΔΟΣ

```
main.tf
```

Το plan λέει ότι θα προστεθούν τρία resources, αλλά το `ls` δείχνει μόνο το `main.tf`. Ό,τι έφτιαξε το `init` ξεκινά με τελεία και δεν φαίνεται. Δεν υπάρχει καν `terraform.tfstate`, γιατί το plan δεν γράφει state.

Άσκηση 1.4 · Idempotence

Ξεκίνα από το `main.tf` της πρώτης άσκησης με τα δύο αρχεία. Κάνε apply δύο φορές στη σειρά χωρίς καμία αλλαγή ανάμεσα. Τι λέει το δεύτερο apply;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "morning" {
  filename = "morning.txt"
  content  = "Καλημέρα\n"
}

resource "local_file" "night" {
  filename = "night.txt"
  content  = "Καληνύχτα\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΕΞΟΔΟΣ

```
local_file.night: Refreshing state... [id=1f630aef20dd5e60a76b660b707a914bdd029522]
local_file.morning: Refreshing state... [id=54a00210345bd8a3b7d8086cbb1f2887e061693c]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration
and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Το δεύτερο apply δεν ρωτά καν `yes`. Το plan του είναι άδειο, οπότε δεν υπάρχει τίποτα να εγκριθεί. Η configuration περιγράφει κατάσταση που ήδη ισχύει. Ένα script που φτιάχνει αρχεία με `echo` θα τα έγραφε ξανά.

Άσκηση 1.5 · Από την αρχή μέχρι το τέλος

Γράψε configuration για το αρχείο `cycle.txt` με περιεχόμενο `ένας κύκλος\n`. Κάνε apply. Μετά κάνε destroy και έλεγξε με `ls` ότι το αρχείο έφυγε ενώ το state έμεινε. Το αναμενόμενο αποτέλεσμα είναι η έξοδος του `ls`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "cycle" {
  filename = "cycle.txt"
  content  = "ένας κύκλος\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΛΥΣΗ · TERMINAL

```
terraform destroy
```

ΕΞΟΔΟΣ

```
…
Destroy complete! Resources: 1 destroyed.
```

ΛΥΣΗ · TERMINAL

```
ls
```

ΕΞΟΔΟΣ

```
main.tf
terraform.tfstate
terraform.tfstate.backup
```

Μετά το destroy στον φάκελο μένουν η configuration και το state. Το `terraform.tfstate.backup` είναι το προηγούμενο state, που το Terraform κρατά σε κάθε αλλαγή. Το κεφάλαιο 4 εξηγεί τι περιέχουν και τα δύο. Το lock file και ο φάκελος `.terraform` δεν φαίνονται στο `ls` γιατί ξεκινούν με τελεία.

Ερωτήσεις πιστοποίησης

Ερώτηση 1.1

Μια ομάδα στήνει servers από την κονσόλα του cloud και κρατά σημειώσεις σε wiki. Ποιο πρόβλημα λύνει πρώτα η μετάβαση σε Infrastructure as Code;

a. Μειώνει το κόστος κάθε server.
b. Κάνει τη δημιουργία της υποδομής επαναλήψιμη και καταγεγραμμένη σε κώδικα.
c. Καταργεί την ανάγκη για πρόσβαση στο API του cloud.
d. Εγγυάται ότι η υποδομή δεν θα έχει ποτέ σφάλματα.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το IaC κάνει την υποδομή κώδικα. Ο κώδικας ξανατρέχει με το ίδιο αποτέλεσμα, μπαίνει σε version control και περνά από review. Δεν αλλάζει την τιμή των resources, οπότε το a είναι λάθος. Το Terraform μιλά με το API μέσω providers, άρα το API χρειάζεται και το c είναι λάθος. Ένα λάθος στον κώδικα θα επαναληφθεί πιστά, οπότε το d δεν ισχύει.

Ερώτηση 1.2

Ποια εντολή πρέπει να τρέξει πρώτη σε έναν φάκελο με νέα configuration;

a. `terraform plan`
b. `terraform validate`
c. `terraform init`
d. `terraform apply`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `terraform init` κατεβάζει τους providers και ρυθμίζει το backend. Χωρίς αυτό οι άλλες εντολές δεν έχουν providers για να δουλέψουν και ζητούν να τρέξεις πρώτα `init`. Το `validate` χρειάζεται κι αυτό τους providers για να ελέγξει τα arguments των resources.

Ερώτηση 1.3

Τρέχεις `terraform plan` σε μια configuration που περιγράφει τρία resources που δεν υπάρχουν. Τι συμβαίνει στην υποδομή;

a. Δημιουργούνται τα τρία resources χωρίς ερώτηση.
b. Τίποτα. Το plan μόνο δείχνει ότι θα προστεθούν τρία resources.
c. Δημιουργείται το πρώτο resource ως δοκιμή.
d. Γράφονται τα τρία resources στο state ως εκκρεμή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το plan συγκρίνει και αναφέρει. Δεν αλλάζει resources και δεν γράφει state. Μόνο το apply κάνει αλλαγές. Ακόμη και αυτό ζητά πρώτα επιβεβαίωση.

Ερώτηση 1.4

Τρέχεις `terraform apply` δεύτερη φορά χωρίς να έχεις αλλάξει τίποτα στη configuration ή στην υποδομή. Τι κάνει το Terraform;

a. Αναφέρει ότι δεν υπάρχουν αλλαγές και δεν κάνει τίποτα.
b. Ξαναφτιάχνει όλα τα resources με τις ίδιες ρυθμίσεις.
c. Αποτυγχάνει επειδή τα resources υπάρχουν ήδη.
d. Σβήνει τα resources και τα ξαναφτιάχνει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Η configuration περιγράφει μια κατάσταση. Αν η κατάσταση ήδη ισχύει, το plan είναι άδειο και το apply δεν έχει τι να κάνει. Αυτή η ιδιότητα λέγεται idempotence και είναι συνέπεια του δηλωτικού μοντέλου. Το b και το d περιγράφουν ένα script που ξανατρέχει τα βήματά του. Το c είναι λάθος, γιατί το state λέει στο Terraform ότι τα resources είναι ήδη δικά του.

Ερώτηση 1.5

Σωστό ή λάθος; Το `terraform destroy` σβήνει μόνο τα resources που καταγράφονται στο state του τρέχοντος φακέλου.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το Terraform διαχειρίζεται μόνο όσα θυμάται στο state. Ένα resource που φτιάχτηκε με άλλο τρόπο, ή που ανήκει σε άλλη configuration με δικό της state, δεν επηρεάζεται από το destroy αυτού του φακέλου.

ΚΕΦΑΛΑΙΟ 2

Η γλώσσα HCL, fmt και validate

Άσκηση 2.1 · Τακτοποίηση με fmt

Γράψε το παρακάτω `main.tf` όπως είναι. Τρέξε `terraform fmt -check`, μετά `terraform fmt` και δείξε το αποτέλεσμα με `cat main.tf`.

main.tf

```
resource "local_file" "menu"{
filename = "menu.txt"
      content="σαλάτα\nψάρι\n"   # σημερινό
  file_permission="0640"
}
```

ΛΥΣΗ · TERMINAL

```
terraform fmt -check
```

ΕΞΟΔΟΣ

```
main.tf
```

ΛΥΣΗ · TERMINAL

```
terraform fmt
```

ΛΥΣΗ · TERMINAL

```
cat main.tf
```

ΕΞΟΔΟΣ

```
resource "local_file" "menu" {
  filename        = "menu.txt"
  content         = "σαλάτα\nψάρι\n" # σημερινό
  file_permission = "0640"
}
```

Το `-check` τύπωσε το `main.tf` και απέτυχε. Το fmt έβαλε εσοχή δύο κενών και ένα κενό πριν από το `{`. Στοίχισε τα τρία `=`, γιατί τα arguments είναι σε διαδοχικές γραμμές. Ονόματα και τιμές δεν άλλαξαν. Δεν χρειάστηκε `init`, γιατί το fmt δεν ξέρει τίποτα για providers.

Άσκηση 2.2 · Heredoc χωρίς περιττή εσοχή

Γράψε ένα `local_file` με όνομα `haiku`, που γράφει στο `haiku.txt` τις τρεις γραμμές του αναμενόμενου αποτελέσματος. Χρησιμοποίησε heredoc, με τις γραμμές στον κώδικα στοιχισμένες σε τέσσερα κενά. Στο αρχείο η δεύτερη γραμμή έχει δύο κενά εσοχή και οι άλλες καμία. Κάνε apply και δείξε το αρχείο με `cat`.

ΛΥΣΗ · versions.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}
```

ΛΥΣΗ · main.tf

```
resource "local_file" "haiku" {
  filename = "haiku.txt"
  content  = <<-EOT
    Φθινόπωρο.
      Ένα φύλλο πέφτει
    στο τερματικό.
  EOT
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΛΥΣΗ · TERMINAL

```
cat haiku.txt
```

ΕΞΟΔΟΣ

```
Φθινόπωρο.
  Ένα φύλλο πέφτει
στο τερματικό.
```

Το `<<-EOT` αφαίρεσε τα τέσσερα κενά της μικρότερης εσοχής από κάθε γραμμή και η δεύτερη κράτησε τα δύο επιπλέον. Με `<<EOT` όλες οι γραμμές θα κρατούσαν τα κενά του κώδικα. Η παγίδα είναι να κολλήσεις τις γραμμές στο αριστερό περιθώριο για να γλιτώσεις την εσοχή. Δουλεύει, αλλά χαλάει την ανάγνωση.

Άσκηση 2.3 · Κείμενο με ειδικούς χαρακτήρες

Γράψε ένα `local_file` με όνομα `template`, που φτιάχνει το `template.txt` με ακριβώς τις δύο γραμμές του αναμενόμενου αποτελέσματος. Χρησιμοποίησε string σε εισαγωγικά, όχι heredoc. Κάνε apply και δείξε το αρχείο με `cat`.

ΛΥΣΗ · versions.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}
```

ΛΥΣΗ · main.tf

```
resource "local_file" "template" {
  filename = "template.txt"
  content  = "Γεια σου, $${name}!\nΕίπε \"ναι\" στο C:\\tf\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply
```

ΛΥΣΗ · TERMINAL

```
cat template.txt
```

ΕΞΟΔΟΣ

```
Γεια σου, ${name}!
Είπε "ναι" στο C:\tf
```

Χωρίς το διπλό `$` το Terraform θεωρεί το `name` αναφορά και απαντά `Invalid reference`. Χωρίς `\"` το εισαγωγικό κλείνει το string. Η πιο ύπουλη παγίδα είναι το `C:\tf` με μία κάθετο. Το `\t` είναι έγκυρο escape, οπότε δεν βγαίνει σφάλμα. Το αρχείο απλώς παίρνει ένα tab αντί για `\t`. Το `\\` δίνει μία κάθετο.

Άσκηση 2.4 · Το ίδιο όνομα σε δύο αρχεία

Γράψε το `versions.tf` του κεφαλαίου και ένα `a.tf` με ένα `local_file` με όνομα `note`, που φτιάχνει το `a.txt`. Τρέξε `terraform init`. Μετά πρόσθεσε ένα `b.tf` με ένα `local_file`, επίσης με όνομα `note`, που φτιάχνει το `b.txt`. Τρέξε `terraform validate`. Ποια αρχεία αναφέρει το μήνυμα;

ΛΥΣΗ · versions.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}
```

ΛΥΣΗ · a.tf

```
resource "local_file" "note" {
  filename = "a.txt"
  content  = "a\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · b.tf

```
resource "local_file" "note" {
  filename = "b.txt"
  content  = "b\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
╷
│ Error: Duplicate resource "local_file" configuration
│
│   on b.tf line 1:
╵
│ A local_file resource named "note" was already declared at a.tf:1,1-29.
╵
```

Τα δύο αρχεία είναι μία configuration, οπότε το `local_file.note` δηλώθηκε δύο φορές. Το diagnostic δείχνει το `b.tf` και λέει ότι το όνομα δηλώθηκε ήδη στο `a.tf:1,1-29`, δηλαδή αρχείο `a.tf`, γραμμή 1,

στήλες 1 έως 29. Η διόρθωση είναι διαφορετικό όνομα στο ένα. Αν τα δύο αρχεία υπήρχαν πριν από το `init`, το ίδιο σφάλμα θα σταματούσε το `init`.

Άσκηση 2.5 · Διόρθωση σε γύρους

Γράψε το `versions.tf` του κεφαλαίου και το παρακάτω `main.tf`. Τρέξε `terraform init` και `terraform validate`. Διόρθωσε τα λάθη και ξανατρέξε validate μέχρι να δεις `Success!`. Το `toc` πρέπει να αναφέρει το `filename` του `chapter`. Για δικαιώματα φακέλου βάλε `0755`.

main.tf

```
resource "local_file" "chapter" {
  filname = "chapter.txt"
  content = "Κεφάλαιο 2\n"
}

resource "local_file" "toc" {
  filename             = "toc.txt"
  content              = "Περιεχόμενα του ${local_file.chapters.filename}\n"
  directory_permission = "rwxr-xr-x"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
…
│ Error: Missing required argument
…
│ Error: Unsupported argument
…
│ Error: Reference to undeclared resource
…
```

ΛΥΣΗ · main.tf

```
resource "local_file" "chapter" {
  filename = "chapter.txt"
  content  = "Κεφάλαιο 2\n"
}

resource "local_file" "toc" {
  filename             = "toc.txt"
  content              = "Περιεχόμενα του ${local_file.chapter.filename}\n"
  directory_permission = "rwxr-xr-x"
}
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
…
╷ Error: Invalid File Permission String Value
…
╷ bad mode permission: string length should be 3 or 4 digits: rwxr-xr-x
…
```

ΛΥΣΗ · main.tf

```
resource "local_file" "chapter" {
  filename = "chapter.txt"
  content  = "Κεφάλαιο 2\n"
}

resource "local_file" "toc" {
  filename             = "toc.txt"
  content              = "Περιεχόμενα του ${local_file.chapter.filename}\n"
  directory_permission = "0755"
}
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
Success! The configuration is valid.
```

Ο πρώτος γύρος δίνει τρία diagnostics. Το `filname` είναι ορθογραφικό λάθος, γι’ αυτό λείπει και το υποχρεωτικό `filename`. Το `local_file.chapters` δεν υπάρχει, γιατί το όνομα είναι `chapter`. Στον δεύτερο γύρο εμφανίζεται το `rwxr-xr-x`, που δεν είναι οκταδικός αριθμός. Ο provider δεν το έλεγξε πριν, γιατί το `toc` είχε λάθος αναφορά. Η παγίδα είναι να σταματήσεις μετά τον πρώτο γύρο.

Άσκηση 2.6 · Το validate λέει ναι, το apply όχι

Γράψε το `versions.tf` του κεφαλαίου και ένα `local_file` με όνομα `copy`, που φτιάχνει το `copy.txt`. Αντί για `content` δώσε `source = "original.txt"`, που λέει στον provider να αντιγράψει υπάρχον αρχείο. Το `original.txt` δεν υπάρχει. Τρέξε `terraform validate` και μετά `terraform apply` με απάντηση `yes`. Ποια εντολή εντοπίζει το πρόβλημα;

ΛΥΣΗ · versions.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}
```

ΛΥΣΗ · main.tf

```
resource "local_file" "copy" {
  filename = "copy.txt"
  source   = "original.txt"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
terraform apply
```

ΕΞΟΔΟΣ

```
Success! The configuration is valid.

╷
│ +Original Error: open original.txt: no such file or directory
…
```

Το validate περνά, γιατί το `source` υπάρχει στο schema και έχει τιμή string. Ούτε το validate ούτε το plan ελέγχουν αν υπάρχει το αρχείο. Το λάθος φαίνεται μόνο στο apply, όταν ο provider ανοίγει το αρχείο. Αν τρέξεις `echo κείμενο > original.txt` και ξανακάνεις apply, θα πετύχει χωρίς αλλαγή στον κώδικα.

Ερωτήσεις πιστοποίησης

Ερώτηση 2.1

Μια ομάδα θέλει το CI pipeline να αποτυγχάνει όταν κάποιο αρχείο `.tf` του repository, σε οποιονδήποτε φάκελο, δεν έχει την κανονική μορφή. Τα αρχεία δεν πρέπει να αλλάζουν. Ποια εντολή ταιριάζει;

a. `terraform fmt`
b. `terraform fmt -check -recursive`
c. `terraform validate`
d. `terraform fmt -diff`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `-check` δεν γράφει αρχεία και επιστρέφει μη μηδενικό exit code όταν κάποιο θέλει αλλαγή. Το `-recursive` πιάνει τους υποφακέλους. Το a ξαναγράφει τα αρχεία και επιστρέφει 0. Το c ελέγχει την ορθότητα, όχι τη μορφή. Το d γράφει κιόλας τις αλλαγές και επιστρέφει 0.

Ερώτηση 2.2

Ποιο από τα παρακάτω προβλήματα εντοπίζει το `terraform validate`;

a. Ο χρήστης δεν έχει δικαίωμα να δημιουργήσει το resource στον πάροχο.
b. Ένα argument μέσα σε resource έχει ορθογραφικό λάθος στο όνομά του.
c. Το όνομα που ζητήθηκε για ένα αντικείμενο το έχει ήδη πάρει άλλος.
d. Κάποιος άλλαξε με το χέρι ένα resource μετά το τελευταίο apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το validate συγκρίνει τα arguments με το schema του provider, οπότε πιάνει ένα άγνωστο όνομα argument. Τα δικαιώματα του a και το πιασμένο όνομα του c τα ξέρει μόνο το API του παρόχου, που το validate δεν καλεί. Την αλλαγή με το χέρι του d τη βρίσκει το plan, όταν συγκρίνει το state με τα πραγματικά resources.

Ερώτηση 2.3

Ποιες δύο προτάσεις για τα αρχεία μιας configuration είναι σωστές; Επίλεξε δύο.

a. Το Terraform διαβάζει όλα τα αρχεία `.tf` του φακέλου εργασίας ως μία configuration.
b. Το block `terraform` πρέπει να βρίσκεται σε αρχείο με όνομα `main.tf`.
c. Τα αρχεία `.tf` των υποφακέλων διαβάζονται αυτόματα.
d. Η σειρά των blocks και των αρχείων δεν επηρεάζει το αποτέλεσμα.
e. Ένα resource μπορεί να αναφέρεται μόνο σε resources του ίδιου αρχείου.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και d

Τα `.tf` του φακέλου ενώνονται σε μία configuration και η σειρά των ενεργειών βγαίνει από τις αναφορές. Το `main.tf` είναι σύμβαση, οπότε το b είναι λάθος. Ένας υποφάκελος διαβάζεται μόνο όταν τον καλέσεις ως module, οπότε το c είναι λάθος. Οι αναφορές λειτουργούν ανάμεσα σε όλα τα αρχεία του φακέλου, οπότε το e είναι λάθος.

Ερώτηση 2.4

Σωστό ή λάθος; Το `terraform validate` χρειάζεται credentials για τον πάροχο, για παράδειγμα κλειδιά του AWS, για να ελέγξει μια configuration.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το validate δεν καλεί κανένα API και δεν διαβάζει state. Χρειάζεται μόνο τους providers εγκατεστημένους από το `init`, για να διαβάσει τα schemas τους. Γι' αυτό τρέχει σε CI χωρίς credentials, αρκεί το init να γίνει με `-backend=false` όταν η configuration έχει remote backend.

Ερώτηση 2.5

Τι κάνει το `terraform fmt` σε ένα αρχείο `.tf`;

a. Μετονομάζει τα resources σε snake_case.
b. Διορθώνει εσοχές και κενά και στοιχίζει τα `=` διαδοχικών arguments.
c. Διορθώνει ονόματα arguments με ορθογραφικά λάθη.
d. Ελέγχει ότι οι providers της configuration είναι εγκατεστημένοι.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το fmt αλλάζει μόνο τη μορφή του κειμένου. Δεν αγγίζει ονόματα, οπότε το a και το c είναι λάθος. Τα λάθη στα ονόματα των arguments τα βρίσκει το validate. Το fmt δεν χρειάζεται `init` και δεν ξέρει τίποτα για providers, οπότε το d είναι λάθος.

ΚΕΦΑΛΑΙΟ 3

Το plan και οι αλλαγές

Άσκηση 3.1 · Ένα σύμβολο

Κάνε `terraform apply -auto-approve` με αυτό το `main.tf`.

main.tf

```
resource "terraform_data" "app" {
  input            = "v1"
  triggers_replace = "a"
}
```

Άλλαξε το `input` σε `"v2"`. Πριν τρέξεις `terraform plan`, γράψε ποιο σύμβολο και ποια περίληψη περιμένεις.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "terraform_data" "app" {
  input            = "v2"
  triggers_replace = "a"
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # terraform_data.app will be updated in-place
…
Plan: 0 to add, 1 to change, 0 to destroy.
…
```

Το `input` αλλάζει επί τόπου, οπότε το resource παίρνει `~` και το `id` μένει ίδιο. Η παγίδα είναι να περιμένεις `1 to add` επειδή αλλάζει και το `output`. Ένα attribute που αλλάζει δεν σημαίνει νέο αντικείμενο.

Άσκηση 3.2 · Δύο αλλαγές σε ένα resource

Κάνε apply με το ίδιο `main.tf` όπως στην προηγούμενη άσκηση.

main.tf

```
resource "terraform_data" "app" {
  input            = "v1"
  triggers_replace = "a"
}
```

Άλλαξε μαζί το `input` σε `"v2"` και το `triggers_replace` σε `"b"`. Θα δεις ένα `~` και ένα `-/+` ή κάτι άλλο;

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "terraform_data" "app" {
  input            = "v2"
  triggers_replace = "b"
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # terraform_data.app must be replaced
…
Plan: 1 to add, 0 to change, 1 to destroy.
…
```

Κάθε resource παίρνει μία μόνο ενέργεια σε ένα plan. Το `triggers_replace` απαιτεί νέο αντικείμενο, οπότε όλο το resource γίνεται `-/+`. Το νέο αντικείμενο φτιάχνεται κατευθείαν με το νέο `input` και δεν χρειάζεται χωριστό update. Γι' αυτό η περίληψη δεν έχει κανένα `to change`.

Άσκηση 3.3 · Plan που αγνοεί τις νέες αλλαγές

Με αυτό το `main.tf` τρέξε `terraform plan -out=app.tfplan` χωρίς apply. Μετά άλλαξε το `input` σε `"v2"`, εφάρμοσε το `app.tfplan` και τρέξε `terraform plan`. Ποια τιμή κατέληξε στο state;

main.tf

```
resource "terraform_data" "app" {
  input = "v1"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform plan -out=app.tfplan
```

ΛΥΣΗ · main.tf

```
resource "terraform_data" "app" {
  input = "v2"
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply app.tfplan
```

ΕΞΟΔΟΣ

```
terraform_data.app: Creating...
terraform_data.app: Creation complete after 0s [id=bd310e34-8caf-73c8-d75d-43ee8d961ac8]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
      ~ input  = "v1" -> "v2"
…
Plan: 0 to add, 1 to change, 0 to destroy.
…
```

Το apply του αρχείου δεν ρώτησε και δεν διάβασε το νέο `main.tf`. Εκτέλεσε το αποθηκευμένο plan με `"v1"`. Το επόμενο plan βρίσκει τη διαφορά `"v1" -> "v2"`. Η παγίδα είναι να νομίσεις ότι το apply του αρχείου κοιτάζει τον κώδικα. Κοιτάζει μόνο το αρχείο.

Άσκηση 3.4 · Κωδικοί εξόδου

Με αυτό το `main.tf` τρέξε `terraform plan -detailed-exitcode` πριν και μετά από ένα apply. Τύπωσε κάθε φορά το exit code με `echo`, για παράδειγμα `echo "πριν το apply $?"`.

main.tf

```
resource "terraform_data" "app" {
  input = "v1"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -detailed-exitcode > /dev/null
echo "πριν το apply $?"
terraform apply -auto-approve > /dev/null
terraform plan -detailed-exitcode > /dev/null
echo "μετά το apply $?"
```

ΕΞΟΔΟΣ

```
πριν το apply 2
μετά το apply 0
```

Πριν από το apply υπάρχει μία δημιουργία, άρα `2`. Μετά δεν μένει τίποτα, άρα `0`. Ένα σφάλμα θα έδινε `1`. Η παγίδα είναι να διαβάσεις το `$?` μετά από άλλη εντολή. Πρέπει να έρχεται αμέσως μετά το `terraform plan`.

Άσκηση 3.5 · Η περίληψη του destroy σε αρχείο

Κάνε apply με αυτό το `main.tf`. Γράψε μετά σε αρχείο το plan που θα έσβηνε τα πάντα και κράτα μόνο την περίληψη με `grep '^Plan'`.

main.tf

```
resource "terraform_data" "a" {
  input = "a"
}

resource "terraform_data" "b" {
  input = "b"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan -destroy -no-color > destroy.txt
grep '^Plan' destroy.txt
```

ΕΞΟΔΟΣ

```
Plan: 0 to add, 0 to change, 2 to destroy.
```

Το `plan -destroy` δεν σβήνει τίποτα. Το `-no-color` είναι απαραίτητο, γιατί χωρίς αυτό η γραμμή στο αρχείο ξεκινά με κωδικό χρώματος και το `grep` δεν βρίσκει τίποτα.

Άσκηση 3.6 · Ξαναφτιάξε ένα resource

Κάνε apply με αυτό το `main.tf`. Αντικατάστησε μετά μόνο το `terraform_data.b`, χωρίς αλλαγή στον κώδικα και χωρίς ερώτηση.

main.tf

```
resource "terraform_data" "a" {
  input = "a"
}

resource "terraform_data" "b" {
  input = "b"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform apply -replace=terraform_data.b -auto-approve
```

ΕΞΟΔΟΣ

```
…
  # terraform_data.b will be replaced, as requested
…
Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

Το `-replace` αντικαθιστά ό,τι ζητάς, ακόμη κι αν η configuration δεν άλλαξε. Η παγίδα είναι το `-target=terraform_data.b`. Θα περιόριζε το plan στο `b`, αλλά δεν θα έβρισκε αλλαγή και δεν θα έκανε τίποτα. Το `terraform taint` αλλάζει πρώτα το state και μόνο μετά βλέπεις το plan.

Ερωτήσεις πιστοποίησης

Ερώτηση 3.1

Στο plan βλέπεις τη γραμμή `-/+ resource "aws_instance" "web" {`. Τι θα κάνει το Terraform στο apply;

a. Θα ενημερώσει το υπάρχον instance χωρίς να το σβήσει.
b. Θα φτιάξει νέο instance και μετά θα σβήσει το παλιό.
c. Θα σβήσει το υπάρχον instance και μετά θα φτιάξει νέο.
d. Θα σβήσει το instance χωρίς να φτιάξει νέο.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `-/+` σημαίνει πρώτα destroy και μετά create. Το a περιγράφει το `~`. Το b περιγράφει το `+/-`, που εμφανίζεται μόνο με `create_before_destroy`. Το d περιγράφει το σκέτο `-`.

Ερώτηση 3.2

Ένα plan περιέχει ένα resource με `~`, ένα με `-/+` και ένα με `-`. Ποια είναι η περίληψη;

a. `Plan: 0 to add, 3 to change, 0 to destroy.`
b. `Plan: 1 to add, 1 to change, 2 to destroy.`
c. `Plan: 1 to add, 2 to change, 1 to destroy.`
d. `Plan: 0 to add, 1 to change, 1 to destroy.`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `~` μετρά ένα `to change`. Το `-/+` μετρά ένα `to add` και ένα `to destroy`. Το `-` προσθέτει ένα ακόμη `to destroy`. Το a μετρά τα πάντα ως αλλαγές. Το c μετρά το replacement ως `to add` και `to change` αντί για `to add` και `to destroy`. Το d αγνοεί εντελώς το replacement.

Ερώτηση 3.3

Τρέχεις `terraform plan -out=tfplan` και μετά `terraform apply tfplan`. Τι συμβαίνει;

a. Εκτελούνται οι ενέργειες του αρχείου χωρίς ερώτηση.
b. Το Terraform φτιάχνει νέο plan από την τρέχουσα configuration και ζητά `yes`.
c. Το Terraform ζητά `yes`, εκτός αν προσθέσεις `-auto-approve`.
d. Το Terraform αποτυγχάνει, γιατί το αρχείο θέλει πρώτα `terraform show`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Ένα αποθηκευμένο plan θεωρείται εγκεκριμένο και εκτελείται χωρίς ερώτηση. Το b περιγράφει το σκέτο `terraform apply`. Το `-auto-approve` δεν χρειάζεται με αρχείο plan, οπότε το c είναι λάθος. Το `terraform show` χρησιμεύει για έλεγχο και δεν είναι προϋπόθεση, άρα και το d είναι λάθος.

Ερώτηση 3.4

Ένα pipeline τρέχει `terraform plan -detailed-exitcode`. Τι σημαίνει exit code `2`;

a. Υπάρχει σφάλμα στη configuration.
b. Το plan πέτυχε και δεν υπάρχουν αλλαγές.
c. Το αποθηκευμένο plan είναι stale.
d. Το plan πέτυχε και υπάρχουν αλλαγές.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `0` σημαίνει επιτυχία χωρίς αλλαγές, το `1` σφάλμα και το `2` επιτυχία με αλλαγές. Το a αντιστοιχεί στο `1` και το b στο `0`. Το stale plan είναι σφάλμα του `apply` με αρχείο, όχι κωδικός του `plan`.

Ερώτηση 3.5

Ποιες προτάσεις ισχύουν για το αρχείο που γράφει το `terraform plan -out`; Επίλεξε δύο.

a. Μπορεί να περιέχει ευαίσθητες τιμές χωρίς κρυπτογράφηση.
b. Είναι αρχείο κειμένου που μπορείς να διορθώσεις πριν από το apply.
c. Διαβάζεται με `terraform show`.
d. Εφαρμόζεται ακόμη κι αν το state άλλαξε μετά τη δημιουργία του.
e. Μπαίνει σε Git μαζί με τη configuration.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το αρχείο περιέχει αντίγραφο του state και όλες τις τιμές του plan. Είναι δυαδικό και διαβάζεται με `terraform show`, οπότε το b είναι λάθος. Αν το state αλλάξει, το apply αποτυγχάνει με `Saved plan is stale` και το d είναι λάθος. Λόγω των ευαίσθητων τιμών δεν μπαίνει σε Git, άρα και το e είναι λάθος.

Ερώτηση 3.6

Ένας server που διαχειρίζεται το Terraform έχει κολλήσει, ενώ η configuration δεν άλλαξε. Θέλεις να τον αντικαταστήσεις. Ποιος είναι ο προτεινόμενος τρόπος;

a. `terraform apply -target=aws_instance.web`
b. `terraform apply -replace=aws_instance.web`
c. Σβήνεις το resource από το `terraform.tfstate` και κάνεις apply.
d. `terraform destroy` και μετά `terraform apply`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `-replace` αναγκάζει αντικατάσταση και τη δείχνει στο plan. Το `-target` μόνο περιορίζει το plan και χωρίς διαφορά δεν κάνει τίποτα. Αν σβήσεις το resource από το state, το Terraform φτιάχνει δεύτερο server και ο παλιός μένει χωρίς διαχείριση. Το `destroy` σβήνει όλη την υποδομή, όχι έναν server.

ΚΕΦΑΛΑΙΟ 4

To state

Άσκηση 4.1 · Serial και lineage

Γράψε configuration με ένα `local_file` με όνομα `note`, που φτιάχνει το `note.txt` με περιεχόμενο `πρώτη εκδοχή\n`. Κάνε apply και κράτησε το lineage με `jq -r .lineage terraform.tfstate > lineage.txt`. Άλλαξε το περιεχόμενο σε `δεύτερη εκδοχή\n` και κάνε ξανά apply. Τύπωσε το `serial` και έλεγξε με `diff` ότι το `lineage` δεν άλλαξε.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "note" {
  filename = "note.txt"
  content  = "πρώτη εκδοχή\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
jq -r .lineage terraform.tfstate > lineage.txt
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "note" {
  filename = "note.txt"
  content  = "δεύτερη εκδοχή\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
jq .serial terraform.tfstate
jq -r .lineage terraform.tfstate | diff lineage.txt - && echo "ίδιο lineage"
```

ΕΞΟΔΟΣ

```
4
ίδιο lineage
```

Η αλλαγή του `content` σημαίνει αντικατάσταση. Το δεύτερο apply έγραψε το state μετά τη διαγραφή του παλιού αρχείου, μετά τη δημιουργία του νέου και στο τέλος, οπότε το `serial` πήγε από 1 σε 4. Μην περιμένεις ένα βήμα ανά apply. Το `diff` δεν βρήκε διαφορά, άρα το `echo` έτρεξε και το `lineage` έμεινε ίδιο.

Άσκηση 4.2 · Λίστα και JSON

Γράψε configuration με τρία resources `local_file` με ονόματα `a`, `b` και `c`, που φτιάχνουν τα `a.txt`, `b.txt` και `c.txt`. Κάνε apply. Τύπωσε τις διευθύνσεις με `terraform state list`. Μετά βγάλε μόνο τα ονόματα των αρχείων από το `terraform show -json` με `jq -r`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "a" {
  filename = "a.txt"
  content  = "a\n"
}

resource "local_file" "b" {
  filename = "b.txt"
  content  = "b\n"
}

resource "local_file" "c" {
  filename = "c.txt"
  content  = "c\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
local_file.a
local_file.b
local_file.c
```

ΛΥΣΗ · TERMINAL

```
terraform show -json | jq -r '.values.root_module.resources[].values.filename'
```

ΕΞΟΔΟΣ

```
a.txt
b.txt
c.txt
```

Τα resources βρίσκονται στο `values.root_module.resources` και τα attributes του καθενός στο `values`. Το `-r` τυπώνει τα strings χωρίς εισαγωγικά. Η λύση διαβάζει το `terraform show -json` και όχι το `terraform.tfstate`, γιατί το πρώτο έχει σταθερή μορφή και δουλεύει με κάθε backend.

Άσκηση 4.3 · Ένα block που έφυγε

Ξεκίνα από αντίγραφο του φακέλου της προηγούμενης άσκησης, χωρίς το state του. Κάνε apply. Μετά σβήσε από το `main.tf` το block του `c` και τρέξε plan. Γιατί το Terraform ξέρει ότι πρέπει να σβήσει κάτι που δεν αναφέρεται πουθενά στη configuration;

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "a" {
  filename = "a.txt"
  content  = "a\n"
}

resource "local_file" "b" {
  filename = "b.txt"
  content  = "b\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # local_file.c will be destroyed
  # (because local_file.c is not in configuration)
…
Plan: 0 to add, 0 to change, 1 to destroy.
…
```

Η configuration δεν λέει τίποτα για το `c`. Το state όμως θυμάται το `local_file.c` με το `id` του και τον provider του. Το Terraform βλέπει μια διεύθυνση του state που λείπει από τη configuration και το γράφει ρητά, `because local_file.c is not in configuration`. Χωρίς state το `c.txt` θα έμενε στον δίσκο και κανείς δεν θα θυμόταν ότι το έφτιαξε το Terraform.

Άσκηση 4.4 · Αρχείο που χάθηκε

Γράψε configuration με ένα `local_file` με όνομα `report` για το `report.txt`. Κάνε apply και σβήσε το `report.txt` με `rm`. Τρέξε plan με `-no-color` και κράτησε με `grep` μόνο τη γραμμή της περίληψης. Αμέσως μετά τύπωσε τη λίστα του state. Θυμάται ακόμη το state το resource;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "report" {
  filename = "report.txt"
  content  = "Η αναφορά της εβδομάδας\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
rm report.txt
```

ΛΥΣΗ · TERMINAL

```
terraform plan -no-color | grep '^Plan'
terraform state list
```

ΕΞΟΔΟΣ

```
Plan: 1 to add, 0 to change, 0 to destroy.
local_file.report
```

Το refresh βρήκε ότι το αρχείο λείπει και το plan προτείνει να το δημιουργήσει. Το state όμως γράφει ακόμη το `local_file.report`, γιατί το plan δεν γράφει ποτέ το state. Το ανανεωμένο αντίγραφο ζει μόνο στη μνήμη όσο τρέχει το plan. Το `-no-color` του κεφαλαίου 3 χρειάζεται για να βρει το `grep` τη γραμμή χωρίς κωδικούς χρωμάτων.

Άσκηση 4.5 · Τι κρατά το backup

Γράψε configuration με ένα `local_file` με όνομα `status`, που φτιάχνει το `status.txt` με περιεχόμενο `ανοιχτό\n`. Κάνε apply. Άλλαξε το περιεχόμενο σε `κλειστό\n` και κάνε apply. Κάνε και τρίτο apply χωρίς καμία αλλαγή. Μετά τύπωσε με `jq -j` το `content` πρώτα από το backup και μετά από το state.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "status" {
  filename = "status.txt"
  content  = "ανοιχτό\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "status" {
  filename = "status.txt"
  content  = "κλειστό\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
jq -j '.resources[0].instances[0].attributes.content' terraform.tfstate.backup
jq -j '.resources[0].instances[0].attributes.content' terraform.tfstate
```

ΕΞΟΔΟΣ

```
ανοιχτό
κλειστό
```

Το δεύτερο apply αντέγραψε το state με το `ανοιχτό` στο backup πριν γράψει το νέο. Το τρίτο δεν είχε αλλαγές, οπότε δεν άλλαξε το state και το backup έμεινε ίδιο. Δεν θα δεις λοιπόν δύο φορές το `κλειστό`. Το `-j` τυπώνει το string χωρίς εισαγωγικά και χωρίς δική του αλλαγή γραμμής, αφού το `content` τελειώνει ήδη σε `\n`.

Άσκηση 4.6 · State σε άλλο φάκελο

Γράψε configuration με local backend που κρατά το state στο `state/dev.tfstate` και με ένα `local_file` με όνομα `env`, που φτιάχνει το `env.txt` με περιεχόμενο `dev\n`. Κάνε init και apply. Άλλαξε το περιεχόμενο σε `development\n` και κάνε ξανά apply. Βρες με `find` όλα τα αρχεία με `tfstate` στο όνομα.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }

  backend "local" {
    path = "state/dev.tfstate"
  }
}

resource "local_file" "env" {
  filename = "env.txt"
  content  = "dev\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }

  backend "local" {
    path = "state/dev.tfstate"
  }
}

resource "local_file" "env" {
  filename = "env.txt"
  content  = "development\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
find . -name "*tfstate*" | sort
```

ΕΞΟΔΟΣ

```
./.terraform/terraform.tfstate
./state/dev.tfstate
./state/dev.tfstate.backup
```

Το state και το backup του γράφτηκαν δίπλα δίπλα στον φάκελο `state`. Το backup παίρνει το όνομα του state με κατάληξη `.backup`, όπου κι αν βρίσκεται. Το `.terraform/terraform.tfstate` καταγράφει μόνο το backend που ρύθμισε το `init`. Η αλλαγή του `content` δεν χρειάστηκε νέο `init`, γιατί το block `backend` έμεινε ίδιο.

Άσκηση 4.7 · Το μυστικό και το Git

Γράψε configuration με ένα `local_sensitive_file` με όνομα `token`, που φτιάχνει το `token.txt` με περιεχόμενο `token = demo-token-42\n`. Κάνε apply και βρες το token μέσα στο `terraform.tfstate` με `grep`. Μετά γράψε `.gitignore` ώστε σε ένα νέο Git repository στον φάκελο το `git status --short` να δείχνει μόνο το `main.tf`, το `.gitignore` και το lock file των providers.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_sensitive_file" "token" {
  filename = "token.txt"
  content  = "token = demo-token-42\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
grep demo-token terraform.tfstate
```

ΕΞΟΔΟΣ

```
            "content": "token = demo-token-42\n",
```

ΛΥΣΗ · .gitignore

```
.terraform/
*.tfstate
*.tfstate.*
token.txt
```

ΛΥΣΗ · TERMINAL

```
git init -q
git status --short
```

ΕΞΟΔΟΣ

```
?? .gitignore
?? .terraform.lock.hcl
?? main.tf
```

Το `grep` βρίσκει το token σε απλό κείμενο, παρότι το plan το έκρυψε. Το `.gitignore` αποκλείει τον φάκελο `.terraform`, το state, κάθε αρχείο που αρχίζει με το όνομα του state και το ίδιο το `token.txt`. Αν ξεχάσεις το `*.tfstate.*`, το backup μπαίνει σε Git με το ίδιο μυστικό. Το `.terraform.lock.hcl` μένει, γιατί καρφώνει τις εκδόσεις των providers για όλη την ομάδα.

Ερωτήσεις πιστοποίησης

Ερώτηση 4.1

Ποιος είναι ο κύριος σκοπός του state στο Terraform;

a. Να κρατά τα credentials με τα οποία οι providers μιλούν στα APIs.
b. Να αντιστοιχίζει τα resources της configuration σε πραγματικά αντικείμενα και να κρατά metadata γι' αυτά.
c. Να αντικαθιστά την configuration όταν τα αρχεία `.tf` λείπουν.
d. Να καταγράφει ποιος χρήστης έτρεξε κάθε apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το state συνδέει κάθε διεύθυνση resource με ένα πραγματικό αντικείμενο και κρατά metadata όπως τον provider και τις εξαρτήσεις. Τα credentials δίνονται στον provider από τη ρύθμισή του ή από το περιβάλλον, οπότε το a είναι λάθος. Το state δεν αντικαθιστά την configuration. Χωρίς αρχεία `.tf` το plan θα πρότεινε να σβηστούν όλα, άρα το c είναι λάθος. Το state δεν είναι ιστορικό χρηστών. Ο χρήστης εμφανίζεται μόνο προσωρινά στο lock info, οπότε το d είναι λάθος.

Ερώτηση 4.2

Μια configuration δεν έχει block `backend` ούτε block `cloud`. Πού αποθηκεύει το Terraform το state;

a. Στο `.terraform/terraform.tfstate`
b. Στο `terraform.tfstate` του φακέλου εργασίας
c. Στο HCP Terraform, στον λογαριασμό του χρήστη
d. Πουθενά. Το state υπάρχει μόνο στη μνήμη όσο τρέχει η εντολή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Χωρίς ρύθμιση το Terraform χρησιμοποιεί το local backend, που γράφει το `terraform.tfstate` στον φάκελο εργασίας. Το `.terraform/terraform.tfstate` κρατά μόνο ποιο backend ρύθμισε το `init`, οπότε το a είναι λάθος. Το HCP Terraform χρειάζεται ρητή ρύθμιση με block `cloud`, άρα το c είναι λάθος. Το state γράφεται πάντα κάπου, γιατί χωρίς αυτό το επόμενο plan δεν θα ήξερε τι υπάρχει. Γι' αυτό το d είναι λάθος.

Ερώτηση 4.3

Ένα resource έχει attribute που ο provider χαρακτηρίζει sensitive και το state βρίσκεται στο local backend. Ποιες δύο προτάσεις ισχύουν; Επίλεξε δύο.

a. Το plan δείχνει `(sensitive value)` στη θέση της τιμής.
b. Το Terraform κρυπτογραφεί την τιμή μέσα στο `terraform.tfstate`.
c. Η τιμή γράφεται στο `terraform.tfstate` σε απλό κείμενο.
d. Η τιμή δεν γράφεται στο state, ώστε να μη διαρρεύσει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το sensitive κρύβει την τιμή στην έξοδο του plan, του apply και του `terraform state show`. Στο state όμως η τιμή γράφεται ολόκληρη, γιατί το Terraform τη χρειάζεται για τη σύγκριση στο επόμενο plan. Το local backend δεν κρυπτογραφεί τίποτα, οπότε το b είναι λάθος. Το d είναι λάθος, γιατί η τιμή γράφεται. Το είδες με `grep` μέσα στο αρχείο.

Ερώτηση 4.4

Σωστό ή λάθος; Το `terraform plan` γράφει στο `terraform.tfstate` τις τιμές που βρήκε κατά το refresh.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το plan κάνει refresh και φτιάχνει ένα ανανεωμένο αντίγραφο του state μόνο στη μνήμη. Το αρχείο δεν αλλάζει. Το `serial` μένει ίδιο και ένα resource που σβήστηκε έξω από το Terraform εμφανίζεται ακόμη στο `terraform state list`. Το state γράφεται από εντολές όπως το apply και το destroy.

Ερώτηση 4.5

Τρέχεις πολλά applies με αλλαγές στην ίδια configuration. Ποιο πεδίο του `terraform.tfstate` μένει ίδιο σε όλη τη ζωή του state;

a. Το `serial`
b. Το `lineage`
c. Το `resources`
d. Το `terraform_version`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `lineage` δίνεται όταν δημιουργείται το state και δεν αλλάζει ποτέ. Το `serial` ανεβαίνει όταν γράφεται αλλαγμένο state, οπότε το a είναι λάθος. Το `resources` αλλάζει με κάθε apply που προσθέτει, αλλάζει ή σβήνει κάτι, άρα το c είναι λάθος. Το `terraform_version` γράφει την έκδοση που έγραψε το state τελευταία και αλλάζει όταν αναβαθμίσεις το Terraform, οπότε το d είναι λάθος.

Ερώτηση 4.6

Θέλεις να δεις όλα τα attributes του `local_file.rules` όπως τα θυμάται το state. Ποια εντολή χρησιμοποιείς;

a. `terraform state list local_file.rules`
b. `terraform state show local_file.rules`
c. `terraform output local_file.rules`
d. `terraform show local_file.rules`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `terraform state show` δέχεται διεύθυνση και τυπώνει όλα τα attributes του resource. Το `terraform state list` με διεύθυνση τυπώνει μόνο τη διεύθυνση αν υπάρχει, οπότε το a είναι λάθος. Το `terraform output` δείχνει outputs και όχι resources, άρα το c είναι λάθος. Το `terraform show` περιμένει path σε αρχείο state ή plan. Με διεύθυνση ψάχνει αρχείο με αυτό το όνομα και αποτυγχάνει, οπότε το d είναι λάθος.

ΚΕΦΑΛΑΙΟ 5

Providers, εκδόσεις και το lock file

Άσκηση 5.1 · Πλήρης διεύθυνση

Γράψε configuration που ζητά τον `local` με τη σύντομη source address και τον `random` με την πλήρη, που ξεκινά από το hostname. Πρόσθεσε ένα `random_pet` και ένα `local_file` που γράφει το όνομά του στο `pet.txt`. Τρέξε init και `terraform providers`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "registry.terraform.io/hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "name" {}

resource "local_file" "pet" {
  filename = "pet.txt"
  content  = "${random_pet.name.id}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform providers
```

ΕΞΟΔΟΣ

```
Providers required by configuration:
.
├── provider[registry.terraform.io/hashicorp/local] 2.9.1
└── provider[registry.terraform.io/hashicorp/random] 3.9.1
```

Το Terraform συμπληρώνει τη σύντομη μορφή με το `registry.terraform.io`, οπότε και οι δύο providers εμφανίζονται με πλήρη διεύθυνση. Η παγίδα είναι να νομίσεις ότι οι δύο μορφές είναι διαφορετικοί providers. Το hostname το γράφεις μόνο όταν ο provider έρχεται από άλλο registry.

Άσκηση 5.2 · Δύο συνθήκες

Γράψε configuration μόνο με τον `random`, με constraint `">= 3.0.0, < 3.9.0"` και ένα `random_pet`. Πριν τρέξεις init, μάντεψε ποια έκδοση θα εγκαταστήσει.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = ">= 3.0.0, < 3.9.0"
    }
  }
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
- Finding hashicorp/random versions matching ">= 3.0.0, < 3.9.0"...
…
- Installed hashicorp/random v3.8.1 (signed by HashiCorp)
…
```

Οι δύο συνθήκες ισχύουν μαζί, οπότε η 3.9.0 και η 3.9.1 μένουν έξω. Η νεότερη που μένει είναι η 3.8.1. Η `3.7.0-alpha1` ταιριάζει αριθμητικά, αλλά μια prerelease δεν επιλέγεται ποτέ από τελεστές όπως `>=` και `<`.

Άσκηση 5.3 · Κάθε 1.x από την 1.12

Γράψε configuration με ένα `local_file` και `required_version` που δέχεται κάθε 1.x από την 1.12 και μετά, αλλά όχι την 2.0. Χρησιμοποίησε μόνο τον τελεστή `~>` και τρέξε init.

ΛΥΣΗ · main.tf

```
terraform {
  required_version = "~> 1.12"

  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "note" {
  filename = "note.txt"
  content  = "1.12 ή νεότερη\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
Terraform has been successfully initialized!
…
```

Το `~> 1.12` έχει δύο ψηφία, άρα σημαίνει `>= 1.12, < 2.0` και δέχεται την 1.16.4. Η παγίδα είναι το `~> 1.12.0`. Με τρία ψηφία δέχεται μόνο 1.12.x και το init σταματά με `Unsupported Terraform Core version`.

Άσκηση 5.4 · Το lock file κρατά την έκδοση

Γράψε configuration μόνο με τον `random` στην ακριβή έκδοση `3.8.1` και ένα `random_pet`. Τρέξε init. Άλλαξε το constraint σε `~> 3.8` και τρέξε ξανά init. Ποια έκδοση έμεινε; Τέλος τρέξε `terraform init -upgrade` και δείξε τις γραμμές `version` και `constraints` του lock file.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.8.1"
    }
  }
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "~> 3.8"
    }
  }
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
- Reusing previous version of hashicorp/random from the dependency lock file
- Using previously-installed hashicorp/random v3.8.1
…
```

ΛΥΣΗ · TERMINAL

```
terraform init -upgrade
```

ΛΥΣΗ · TERMINAL

```
grep -E "version|constraints" .terraform.lock.hcl
```

ΕΞΟΔΟΣ

```
  version     = "3.9.1"
  constraints = "~> 3.8"
```

Το `~> 3.8` δέχεται την κλειδωμένη 3.8.1, οπότε το σκέτο init την κράτησε, αν και η 3.9.1 είναι νεότερη και ταιριάζει. Μόνο το `-upgrade` ξαναδιάλεξε, πήρε την 3.9.1 και έγραψε το νέο constraint. Η παγίδα είναι να πιστέψεις ότι ένα χαλαρότερο constraint φέρνει μόνο του νεότερη έκδοση.

Άσκηση 5.5 · Τοπικό όνομα

Γράψε configuration που δηλώνει τον `hashicorp/random` με τοπικό όνομα `rnd` και έχει ένα `random_pet`. Τρέξε init και `terraform validate`. Διόρθωσε τη configuration ώστε το `validate` να μη βγάζει προειδοποίηση, κρατώντας το `rnd`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    rnd = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
╷
│ Warning: Duplicate required provider
│
│   on main.tf line 10:
│   10: resource "random_pet" "name" {}
│
│ Provider "registry.terraform.io/hashicorp/random" was implicitly required
│ via resource "random_pet.name", but listed in required_providers as "rnd".
│ Either the local name in required_providers must match the resource name,
│ or the "rnd" provider must be assigned within the resource block.
╵
Success! The configuration is valid, but there were some
validation warnings as shown above.
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    rnd = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "name" {
  provider = rnd
}
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
Success! The configuration is valid.
```

Για το `random_pet` το Terraform ψάχνει τοπικό όνομα `random`, που δεν υπάρχει. Η προειδοποίηση `Duplicate required provider` λέει ότι ο ίδιος provider ζητήθηκε ρητά ως `rnd` και σιωπηρά από το resource. Αφού το `rnd` μένει, η λύση είναι το `provider = rnd` στο resource.

Άσκηση 5.6 · Το alias που χάθηκε

Ξεκίνα από αντίγραφο του φακέλου `aliases`. Σβήσε το block `provider` με `alias = "second"`, άφησε το `random_pet.other` όπως είναι και τρέξε init και plan. Τι λέει το σφάλμα;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

provider "random" {}

resource "random_pet" "first" {}

resource "random_pet" "other" {
  provider = random.second
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
╷
│ Error: Provider configuration not present
╷
│ provider["registry.terraform.io/hashicorp/random"].second is required, but
…
```

Το init πετυχαίνει, γιατί ο provider υπάρχει. Το plan όμως δεν βρίσκει τη ρύθμιση `random.second` και σταματά με `Provider configuration not present`. Το μήνυμα μιλά για ρύθμιση που αφαιρέθηκε ενώ υπάρχουν αντικείμενα στο state, την πιο συνηθισμένη αιτία. Λύνεται αν ξαναβάλεις το alias ή αν σβήσεις το `provider` από το resource.

Ερωτήσεις πιστοποίησης

Ερώτηση 5.1

Ένα module δηλώνει `version = "~> 1.2.0"` για έναν provider. Ποιες εκδόσεις δέχεται;

a. Μόνο την 1.2.0
b. Από την 1.2.0 έως πριν από την 2.0.0
c. Από την 1.2.0 έως πριν από την 1.3.0
d. Από την 1.2.0 και πάνω, χωρίς άνω όριο

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Ο `~>` αφήνει να μεγαλώνει μόνο το τελευταίο ψηφίο, εδώ το patch. Το a είναι το `= 1.2.0`, το b το `~> 1.2` και το d το `>= 1.2.0`.

Ερώτηση 5.2

Μια configuration γράφει `source = "hashicorp/aws"`. Ποια είναι η πλήρης source address;

a. `hashicorp.com/aws`
b. `registry.terraform.io/hashicorp/aws`
c. `github.com/hashicorp/terraform-provider-aws`
d. `releases.hashicorp.com/hashicorp/aws`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Όταν λείπει το hostname, το Terraform βάζει το `registry.terraform.io`. Το a δεν έχει namespace. Το c είναι το repository του κώδικα, από όπου το init δεν κατεβάζει providers. Το d δεν είναι το προεπιλεγμένο hostname.

Ερώτηση 5.3

Μια συνάδελφος άλλαξε το constraint του `random` από `~> 3.0` σε `~> 3.8.0`. Το lock file γράφει 3.9.1 και το `terraform plan` αποτυγχάνει με `Inconsistent dependency lock file`. Τι πρέπει να τρέξει;

a. `terraform init`
b. `terraform init -upgrade`
c. Να σβήσει το `.terraform.lock.hcl` και να τρέξει `terraform plan`
d. `terraform validate`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `-upgrade` αγνοεί την κλειδωμένη επιλογή και διαλέγει ξανά μέσα στο νέο constraint. Το σκέτο init του a κρατά την 3.9.1 και αποτυγχάνει. Στο c το plan δεν διαλέγει εκδόσεις και χωρίς lock file ζητά init. Το `validate` του d δεν εγκαθιστά τίποτα.

Ερώτηση 5.4

Ποιες δύο προτάσεις για τους providers είναι σωστές; Επίλεξε δύο.

a. Είναι χωριστά εκτελέσιμα που ο Terraform Core ξεκινά και με τα οποία μιλά μέσω RPC.
b. Είναι ενσωματωμένοι στο εκτελέσιμο `terraform` και αναβαθμίζονται μαζί του.
c. Κατεβαίνουν με το `terraform init` στον φάκελο `.terraform/providers`.
d. Η έκδοσή τους ορίζεται με το argument `required_version`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Οι providers είναι plugins με δικές τους εκδόσεις, που το init κατεβάζει στο `.terraform/providers`. Το b είναι λάθος γιατί το `terraform` δεν περιέχει providers. Το d είναι λάθος γιατί το `required_version` αφορά μόνο το CLI.

Ερώτηση 5.5

Σωστό ή λάθος; Μια configuration έχει δύο blocks `provider "aws"`, ένα χωρίς `alias` και ένα με `alias = "west"`. Ένα `aws_s3_bucket` χωρίς argument `provider` φτιάχνεται με τη ρύθμιση χωρίς `alias`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το block χωρίς `alias` είναι η προεπιλεγμένη ρύθμιση. Για τη δεύτερη το resource πρέπει να γράφει `provider = aws.west`.

Ερώτηση 5.6

Μια εταιρεία έχει servers σε δικό της datacenter, εφαρμογές στο AWS και στο Azure και DNS σε εξωτερική υπηρεσία. Ποιο είναι το βασικό όφελος αν τα διαχειρίζεται όλα με Terraform;

a. Το Terraform μεταφράζει αυτόματα τα resources του AWS σε ισοδύναμα του Azure.
b. Δεν χρειάζονται πια credentials για κάθε πλατφόρμα.
c. Κάθε πλατφόρμα κρατά δικό της εργαλείο και το Terraform ενώνει μόνο τα state.
d. Μία γλώσσα και ένα workflow για όλες τις πλατφόρμες, με references ανάμεσα σε resources διαφορετικών providers.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το Terraform είναι service agnostic. Κάθε πλατφόρμα έχει provider, αλλά γλώσσα, εντολές και state είναι κοινά. Το a είναι λάθος γιατί κάθε provider έχει δικούς του τύπους resources. Το b είναι λάθος γιατί κάθε provider θέλει πρόσβαση στο API του. Το c περιγράφει τα χωριστά εργαλεία που το Terraform αντικαθιστά.

ΚΕΦΑΛΑΙΟ 6

Resources, references και dependencies

Άσκηση 6.1 · Διευθύνσεις στο state

Γράψε configuration με τρία resources. Ένα `random_pet` με όνομα `db` και τρεις λέξεις. Ένα `random_integer` με όνομα `replicas` από 1 έως 5. Ένα `random_string` με όνομα `suffix`, τέσσερις χαρακτήρες, χωρίς σύμβολα και χωρίς κεφαλαία. Κάνε apply και τύπωσε τις διευθύνσεις των resources.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "db" {
  length = 3
}

resource "random_integer" "replicas" {
  min = 1
  max = 5
}

resource "random_string" "suffix" {
  length  = 4
  special = false
  upper   = false
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
random_integer.replicas
random_pet.db
random_string.suffix
```

Το `terraform state list` τυπώνει μία διεύθυνση `τύπος.όνομα` ανά γραμμή, με αλφαβητική σειρά. Τα τρία resources έχουν διαφορετικούς τύπους, οπότε θα μπορούσαν να έχουν και το ίδιο όνομα. Χρειάζονται και το `upper = false` και το `special = false`, γιατί η προεπιλογή του `random_string` βάζει κεφαλαία και σύμβολα.

Άσκηση 6.2 · Η τυχαία τιμή μένει

Ξεκίνα από αντίγραφο του φακέλου της άσκησης 6.1. Κάνε apply δύο φορές στη σειρά και δες αν άλλαξαν οι τυχαίες τιμές. Μετά άλλαξε το `length` του `random_pet.db` σε 2 και τρέξε `terraform plan`. Τι θα γίνει στο όνομα;

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "db" {
  length = 2
}
```

```
resource "random_integer" "replicas" {
  min = 1
  max = 5
}

resource "random_string" "suffix" {
  length  = 4
  special = false
  upper   = false
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # random_pet.db must be replaced
…
      ~ length    = 3 -> 2 # forces replacement
…
Plan: 1 to add, 0 to change, 1 to destroy.
…
```

Το δεύτερο apply δεν βρίσκει αλλαγές. Οι τυχαίες τιμές αποθηκεύτηκαν στο state και το Terraform δεν τις ξαναπαράγει σε κάθε εκτέλεση. Η αλλαγή στο `length` όμως σημαίνει άλλο όνομα. Το `random_pet` δεν αλλάζει επί τόπου, οπότε το plan το αντικαθιστά και το νέο resource θα πάρει νέο όνομα. Αν ένα bucket έπαιρνε το όνομά του από αυτό, θα αντικαθιστούνταν κι αυτό.

Άσκηση 6.3 · Η σειρά των blocks δεν μετράει

Γράψε το παρακάτω `main.tf`, όπου τα blocks είναι γραμμένα από το τελευταίο προς το πρώτο. Πριν από το apply γράψε σε χαρτί με ποια σειρά πιστεύεις ότι θα φτιαχτούν τα τρία resources. Μετά κάνε apply.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "local_file" "receipt" {
  filename = "receipt.txt"
  content  = "Ο κωδικός γράφτηκε στο ${local_file.code.filename}\n"
}

resource "local_file" "code" {
  filename = "code.txt"
  content  = "${random_string.code.result}\n"
}

resource "random_string" "code" {
  length = 6
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
random_string.code: Creating...
…
local_file.code: Creating...
…
local_file.receipt: Creating...
…
```

Το `receipt` αναφέρει το `local_file.code` και αυτό αναφέρει το `random_string.code`. Η αλυσίδα ορίζει τη σειρά και η θέση των blocks στο αρχείο αγνοείται. Πρόσεξε επίσης ότι το `random_string.code` και το `local_file.code` έχουν το ίδιο όνομα. Επιτρέπεται, γιατί οι διευθύνσεις τους διαφέρουν στον τύπο.

Άσκηση 6.4 · Από depends_on σε αναφορά

Ξεκίνα από αντίγραφο του φακέλου `backup`. Βγάλε το `depends_on` από το `local_file.backup` και κάνε το `source` αναφορά στο attribute `filename` του `local_file.notes`. Τύπωσε το graph και έλεγξε ότι η εξάρτηση υπάρχει ακόμη. Μετά κάνε apply.

ΛΥΣΗ · main.tf

```
resource "random_pet" "server" {}

resource "local_file" "notes" {
  filename = "notes.txt"
  content  = "Ο server λέγεται ${random_pet.server.id}\n"
}

resource "local_file" "backup" {
  filename = "backup/notes.txt"
  source   = local_file.notes.filename
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform graph
```

ΕΞΟΔΟΣ

```
…
  "local_file.backup" -> "local_file.notes";
  "local_file.notes" -> "random_pet.server";
…
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
random_pet.server: Creating...
random_pet.server: Creation complete after 0s [id=game-pony]
local_file.notes: Creating...
local_file.notes: Creation complete after 0s [id=8f0807d99d5918c544c6ec4e854b3ebbc4e30491]
local_file.backup: Creating...
local_file.backup: Creation complete after 0s [id=8f0807d99d5918c544c6ec4e854b3ebbc4e30491]
…
```

Το βέλος `"local_file.backup" -> "local_file.notes"` είναι ίδιο με αυτό που έδινε το `depends_on`, αλλά τώρα προκύπτει από την τιμή. Αν κάποιος αλλάξει το `filename` του `notes`, το `backup` θα ακολουθήσει μόνο του.

Με το string και το `depends_on` θα έπρεπε να θυμηθεί δύο σημεία. Η αναφορά γράφεται σκέτη, γιατί η τιμή είναι μόνο αυτή.

Άσκηση 6.5 · Λάθος στόχος στο depends_on

Ξεκίνα από αντίγραφο του φακέλου `backup`. Άλλαξε το `depends_on` ώστε να δείχνει στο attribute `filename` του `local_file.notes` αντί για ολόκληρο το resource. Τρέξε `terraform validate`.

ΛΥΣΗ · main.tf

```
resource "random_pet" "server" {}

resource "local_file" "notes" {
  filename = "notes.txt"
  content  = "Ο server λέγεται ${random_pet.server.id}\n"
}

resource "local_file" "backup" {
  filename = "backup/notes.txt"
  source   = "notes.txt"

  depends_on = [local_file.notes.filename]
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
╷
│ Error: Invalid depends_on reference
│
│   on main.tf line 12, in resource "local_file" "backup":
│   12:   depends_on = [local_file.notes.filename]
│
│ References in depends_on must be to a whole object (resource, etc), not to
│ an attribute of an object.
╵
```

Το `depends_on` περιγράφει σειρά ανάμεσα σε ολόκληρα αντικείμενα. Ένα attribute δεν έχει δικό του κόμβο στο graph, οπότε το Terraform το απορρίπτει πριν από οποιοδήποτε plan.

Άσκηση 6.6 · Σπάσε τον κύκλο

Ξεκίνα από αντίγραφο του φακέλου `cycle`. Τα δύο αρχεία πρέπει να γράφουν το ίδιο τυχαίο όνομα παιχνιδιού. Πρόσθεσε ένα `random_pet` με όνομα `game` και κάνε και τα δύο `local_file` να το διαβάζουν, χωρίς να αναφέρει το ένα το άλλο. Τρέξε `terraform validate` και `terraform graph`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "game" {}

resource "local_file" "ping" {
  filename = "ping.txt"
  content  = "Παιχνίδι ${random_pet.game.id}, ping\n"
}

resource "local_file" "pong" {
  filename = "pong.txt"
  content  = "Παιχνίδι ${random_pet.game.id}, pong\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
Success! The configuration is valid.
```

ΛΥΣΗ · TERMINAL

```
terraform graph
```

```
ΕΞΟΔΟΣ

…
  "local_file.ping" -> "random_pet.game";
  "local_file.pong" -> "random_pet.game";
…
```

Τα δύο αρχεία δεν αλληλοαναφέρονται πια. Και τα δύο βέλη δείχνουν προς το `random_pet.game`, οπότε το graph δεν έχει κύκλο. Η μεταφορά της κοινής πληροφορίας σε τρίτο resource είναι ο συνηθισμένος τρόπος να σπάσεις έναν κύκλο. Ο provider `random` είναι νέος στον φάκελο, οπότε χρειάζεται ξανά `terraform init`.

Ερωτήσεις πιστοποίησης

Ερώτηση 6.1

Μια configuration περιέχει το block `resource "random_pet" "db" {}`. Ποια έκφραση δίνει το όνομα που παρήγαγε σε ένα argument άλλου resource;

a. `random_pet.db.id`
b. `random_pet["db"].id`
c. `db.random_pet.id`
d. `random_pet.db.name`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Μια αναφορά γράφεται τύπος, όνομα, attribute. Ο `random_pet` εκθέτει το όνομα στο attribute `id`. Το b μοιάζει με διεύθυνση resource με `for_each`, που δεν υπάρχει εδώ. Το c έχει τύπο και όνομα ανάποδα. Το d αποτυγχάνει με `Unsupported attribute`, γιατί ο `random_pet` δεν έχει attribute `name`.

Ερώτηση 6.2

Το `local_file.report` χρησιμοποιεί στο `content` του την τιμή `random_integer.port.result`. Δεν υπάρχει `depends_on`. Τι ισχύει για τη σειρά δημιουργίας;

a. Τα δύο φτιάχνονται παράλληλα, γιατί δεν υπάρχει `depends_on`.
b. Φτιάχνεται πρώτο όποιο είναι γραμμένο πρώτο στο αρχείο.
c. Το `random_integer.port` φτιάχνεται πρώτο, γιατί η αναφορά δημιουργεί implicit dependency.
d. Το apply αποτυγχάνει, γιατί η τιμή είναι `(known after apply)`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Η αναφορά αρκεί για να ξέρει το Terraform ότι το `report` χρειάζεται το `port`. Το `depends_on` χρειάζεται μόνο όταν δεν υπάρχει αναφορά, άρα το a είναι λάθος. Η σειρά των blocks στο αρχείο δεν παίζει ρόλο, άρα και το b. Το `(known after apply)` σημαίνει ότι η τιμή θα γίνει γνωστή στη διάρκεια του apply, πριν τη χρειαστεί το `report`, οπότε το d είναι λάθος.

Ερώτηση 6.3

Ένα πρόγραμμα σε server του AWS διαβάζει στην εκκίνηση ένα αντικείμενο από S3 bucket. Κανένα argument του `aws_instance` δεν αναφέρει το bucket ή το αντικείμενο. Πώς εξασφαλίζεις ότι το αντικείμενο θα υπάρχει πριν από τον server;

a. Γράφεις το block του αντικειμένου πριν από το block του server.
b. Προσθέτεις `depends_on` στο `aws_instance` με το resource του αντικειμένου.
c. Τρέχεις το apply με `-parallelism=1`.
d. Δεν χρειάζεται τίποτα, το Terraform φτιάχνει πάντα πρώτα την αποθήκευση.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η εξάρτηση υπάρχει μόνο στη συμπεριφορά του προγράμματος, όχι σε κάποια τιμή. Αυτή είναι ακριβώς η περίπτωση του `depends_on`. Η σειρά των blocks αγνοείται, άρα το a δεν βοηθά. Το `-parallelism=1` δουλεύει ένα resource τη φορά αλλά δεν ορίζει ποιο θα προηγηθεί, άρα το c δεν εγγυάται τίποτα. Το Terraform δεν έχει κανόνα για κατηγορίες resources, οπότε το d είναι λάθος.

Ερώτηση 6.4

Το `local_file.summary` αναφέρει το `local_file.config` και αυτό αναφέρει το `random_pet.server`. Με ποια σειρά τα σβήνει το `terraform destroy`;

a. `random_pet.server`, `local_file.config`, `local_file.summary`
b. Και τα τρία παράλληλα, γιατί το destroy δεν κοιτά εξαρτήσεις.
c. Με τη σειρά που είναι γραμμένα στη configuration.
d. `local_file.summary`, `local_file.config`, `random_pet.server`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το destroy ακολουθεί το graph ανάποδα. Σβήνει πρώτα ό,τι δεν το χρειάζεται κανένα άλλο resource. Το a είναι η σειρά του create. Το b και το c αγνοούν ότι το destroy χρησιμοποιεί το ίδιο graph με το apply.

Ερώτηση 6.5

Πόσες λειτουργίες σε resources εκτελεί το Terraform ταυτόχρονα, αν δεν δώσεις `-parallelism`;

a. 1
b. 10
c. Όσους πυρήνες έχει ο επεξεργαστής
d. Χωρίς όριο

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η προεπιλογή είναι 10 και αλλάζει με `-parallelism=N` στα `plan`, `apply` και `destroy`. Το 1 είναι τιμή που δίνεις ρητά. Ο αριθμός δεν εξαρτάται από τον επεξεργαστή και υπάρχει πάντα όριο.

Ερώτηση 6.6

Ποια ισχύουν για το `terraform_data`; Επίλεξε δύο.

a. Είναι ενσωματωμένο στο Terraform και δεν χρειάζεται εγγραφή στο `required_providers`.
b. Μια αλλαγή στο `input` το αντικαθιστά πάντα.
c. Μια αλλαγή στο `triggers_replace` το αντικαθιστά.
d. Φτιάχνει αρχείο στον δίσκο με την τιμή του `input`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το `init` το αναφέρει ως `terraform.io/builtin/terraform` και δεν κατεβάζει τίποτα. Το `triggers_replace` υπάρχει ακριβώς για να προκαλεί αντικατάσταση. Το `input` αλλάζει επί τόπου και ενημερώνει το `output`, άρα το b είναι λάθος. Το `terraform_data` ζει μόνο στο state και δεν φτιάχνει τίποτα έξω από αυτό, άρα το d είναι λάθος.

ΚΕΦΑΛΑΙΟ 7

Input variables

Άσκηση 7.1 · Η πόλη σου

Γράψε configuration με τον provider `local` και μία μεταβλητή `city` τύπου `string`, χωρίς `default`, με description. Ένα `local_file` με όνομα `city` γράφει το αρχείο `city.txt` με περιεχόμενο την τιμή της μεταβλητής. Κάνε apply με `-auto-approve` και δώσε την τιμή `θεσσαλονίκη` από τη γραμμή εντολών. Μετά δείξε το αρχείο με `cat`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "city" {
  type        = string
  description = "Η πόλη που γράφεται στο αρχείο"
}

resource "local_file" "city" {
  filename = "city.txt"
  content  = var.city
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve -var city=θεσσαλονίκη
```

ΛΥΣΗ · TERMINAL

```
cat city.txt
```

ΕΞΟΔΟΣ

```
θεσσαλονίκη
```

Η μεταβλητή είναι υποχρεωτική, γιατί δεν έχει `default`. Το `-var` της δίνει τιμή, οπότε το apply δεν ρωτά τίποτα. Χωρίς το `-var` το Terraform θα ρωτούσε για την τιμή, ακόμη και με `-auto-approve`, που παραλείπει μόνο το `yes`.

Άσκηση 7.2 · Δύο περιβάλλοντα

Ο φάκελος της άσκησης ξεκινά ως αντίγραφο του φακέλου της άσκησης 7.1, χωρίς state και χωρίς `.terraform`. Γράψε δύο αρχεία μεταβλητών, το `dev.tfvars` με την πόλη `Λάρισα` και το `prod.tfvars` με την πόλη `Αθήνα`. Τρέξε `terraform plan` μία φορά με το καθένα. Το αναμενόμενο αποτέλεσμα δείχνει τη γραμμή του `content` στα δύο plans.

ΛΥΣΗ · dev.tfvars

```
city = "Λάρισα"
```

ΛΥΣΗ · prod.tfvars

```
city = "Αθήνα"
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var-file=dev.tfvars
terraform plan -var-file=prod.tfvars
```

ΕΞΟΔΟΣ

```
…
      + content              = "Λάρισα"
…
      + content              = "Αθήνα"
…
```

Τα δύο αρχεία δεν λέγονται `terraform.tfvars` ούτε τελειώνουν σε `.auto.tfvars`, οπότε δεν φορτώνονται μόνα τους. Γι’ αυτό διαλέγεις κάθε φορά ποιο θέλεις με `-var-file`. Αν τα ονόμαζες `dev.auto.tfvars` και `prod.auto.tfvars`, θα φορτώνονταν και τα δύο σε κάθε εντολή και θα κέρδιζε πάντα το `prod`, που είναι αλφαβητικά τελευταίο.

Άσκηση 7.3 · Λέξεις με αριθμό

Γράψε configuration με τον provider `random` και μια μεταβλητή `words` τύπου `number` με `default` το `2`. Ένα `random_pet` με όνομα `pet` παίρνει το `length` από τη `words`. Τρέξε `terraform plan` πρώτα με `-var words=πέντε` και μετά με `-var words=5`. Το αναμενόμενο αποτέλεσμα είναι η γραμμή του `length` στο δεύτερο plan.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

variable "words" {
  type        = number
  description = "Πόσες λέξεις έχει το όνομα"
  default     = 2
}

resource "random_pet" "pet" {
  length = var.words
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var words=πέντε
```

ΕΞΟΔΟΣ

```
…
│ Error: Invalid value for input variable
…
│ Unsuitable value for var.words set using -var="words=...": a number is
│ required.
…
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var words=5
```

ΕΞΟΔΟΣ

```
…
      + length    = 5
…
```

Το `-var` δίνει πάντα κείμενο. Στη δεύτερη εντολή το `"5"` μετατράπηκε σε αριθμό, γιατί αυτό ζητά ο τύπος `number`. Στην πρώτη η μετατροπή δεν έχει νόημα και το Terraform σταμάτησε με `Invalid value for input variable` στη δήλωση της μεταβλητής. Χωρίς `type` η μεταβλητή θα δεχόταν το `πέντε`. Το λάθος θα φαινόταν αργότερα, στο `length` του resource, με το μήνυμα `Incorrect attribute value type`. Σε μεγάλη configuration θα έψαχνες από πού ήρθε η τιμή.

Άσκηση 7.4 · Ένα null που δεν περνά

Γράψε σε νέο φάκελο το `main.tf` και το `terraform.tfvars` που ακολουθούν. Το `terraform plan` αποτυγχάνει, γιατί το `null` φτάνει στο `content`. Άλλαξε μόνο το block `variable` ώστε το plan να γράφει στο αρχείο το `default`.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "message" {
  type    = string
  default = "Γεια σου"
}

resource "local_file" "hello" {
  filename = "hello.txt"
  content  = var.message
}
```

terraform.tfvars

```
message = null
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "message" {
  type     = string
  default  = "Γεια σου"
  nullable = false
}

resource "local_file" "hello" {
  filename = "hello.txt"
  content  = var.message
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
      + content              = "Γεια σου"
…
```

Με την προεπιλογή `nullable = true` το `null` του `terraform.tfvars` είναι κανονική τιμή και περνά στο resource, παρόλο που υπάρχει `default`. Το `nullable = false` λέει ότι η μεταβλητή δεν δέχεται ποτέ `null`. Επειδή έχει `default`, το `null` αντικαθίσταται από αυτό. Αν δεν είχε `default`, το plan θα σταματούσε με σφάλμα για το `null`. Η άλλη λύση, να σβήσεις το `terraform.tfvars`, αλλάζει την είσοδο και όχι τη μεταβλητή, οπότε δεν είναι αυτό που ζητά η άσκηση.

Άσκηση 7.5 · Ένα token έξω από το Git

Γράψε configuration με τον provider `local`, μια μεταβλητή `api_token` τύπου `string` χωρίς `default` και ένα `local_file` με όνομα `config` που γράφει το `config.txt` με περιεχόμενο το token. Δώσε την τιμή `abc123` στο αρχείο `secret.auto.tfvars`. Γράψε `.gitignore` που κρατά έξω τον φάκελο `.terraform`, τα αρχεία του state και κάθε αρχείο μεταβλητών. Τρέξε `terraform plan -input=false` για να δεις ότι η τιμή βρέθηκε και μετά `git init -q` και `git status --short --ignored`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "api_token" {
  type        = string
  description = "Το token του API"
}

resource "local_file" "config" {
  filename = "config.txt"
  content  = var.api_token
}
```

ΛΥΣΗ · secret.auto.tfvars

```
api_token = "abc123"
```

ΛΥΣΗ · .gitignore

```
.terraform/
*.tfstate
*.tfstate.*
*.tfvars
*.tfvars.json
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -input=false
```

ΕΞΟΔΟΣ

```
…
      + content              = "abc123"
…
```

ΛΥΣΗ · TERMINAL

```
git init -q
git status --short --ignored
```

ΕΞΟΔΟΣ

```
?? .gitignore
?? .terraform.lock.hcl
?? main.tf
!! .terraform/
!! secret.auto.tfvars
```

Το όνομα `secret.auto.tfvars` τελειώνει σε `.auto.tfvars`, οπότε το Terraform το φορτώνει μόνο του και το plan με `-input=false` δεν αποτυγχάνει. Το ίδιο όνομα τελειώνει και σε `.tfvars`, οπότε το Git το αγνοεί. Το `.terraform.lock.hcl` πρέπει να μπει σε Git και πράγματι εμφανίζεται με `??`. Αν κάνεις apply, το token θα γραφτεί και στο state και στο `config.txt`. Το state το κρατά ήδη έξω το `.gitignore`. Για το `config.txt` θα χρειαζόταν ακόμη μία γραμμή.

Άσκηση 7.6 · Ποιος κερδίζει

Γράψε σε νέο φάκελο τα αρχεία που ακολουθούν. Πριν τρέξεις οτιδήποτε, πρόβλεψε τι θα γράψει το `content` σε δύο plans, ενώ ισχύει η μεταβλητή περιβάλλοντος `TF_VAR_color=μαύρο`. Το πρώτο είναι σκέτο `terraform plan`. Το δεύτερο είναι `terraform plan -var color=λευκό -var-file=extra.tfvars`. Μετά τρέξε τα δύο plans και σβήσε τη μεταβλητή περιβάλλοντος.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "color" {
  type    = string
  default = "γκρι"
}

resource "local_file" "color" {
  filename = "color.txt"
  content  = var.color
}
```

terraform.tfvars

```
color = "πράσινο"
```

z.auto.tfvars

```
color = "κόκκινο"
```

a.auto.tfvars

```
color = "μπλε"
```

extra.tfvars

```
color = "κίτρινο"
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
export TF_VAR_color=μαύρο
terraform plan
terraform plan -var color=λευκό -var-file=extra.tfvars
unset TF_VAR_color
```

ΕΞΟΔΟΣ

```
…
      + content              = "κόκκινο"
…
      + content              = "κίτρινο"
…
```

Στο πρώτο plan δεν υπάρχει γραμμή εντολών, οπότε κερδίζουν τα `.auto.tfvars`. Από αυτά διαβάζεται τελευταίο το `z.auto.tfvars`, γιατί η σειρά είναι αλφαβητική και όχι η σειρά που τα έγραψες. Το `μαύρο` του περιβάλλοντος και το `πράσινο` του `terraform.tfvars` χάνουν, αν και υπάρχουν. Στο δεύτερο plan το `-var` και το `-var-file` νικούν όλα τα αρχεία. Μεταξύ τους κερδίζει όποιο γράφτηκε τελευταίο, εδώ το `extra.tfvars`. Η παγίδα είναι να θεωρήσεις ότι το `-var` νικά πάντα το `-var-file`.

Ερωτήσεις πιστοποίησης

Ερώτηση 7.1

Η μεταβλητή `region` παίρνει τιμή από το `terraform.tfvars`, από το `prod.auto.tfvars`, από τη μεταβλητή περιβάλλοντος `TF_VAR_region` και από το `-var-file=override.tfvars` της εντολής. Ποια τιμή χρησιμοποιεί το Terraform;

a. Του `terraform.tfvars`
b. Του `prod.auto.tfvars`
c. Της `TF_VAR_region`
d. Του `override.tfvars`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Οι τιμές της γραμμής εντολών, `-var` και `-var-file`, διαβάζονται τελευταίες και νικούν όλες τις άλλες πηγές. Η μεταβλητή περιβάλλοντος είναι η ασθενέστερη πηγή, οπότε το c είναι λάθος. Το `terraform.tfvars` νικά μόνο το περιβάλλον και χάνει από τα `.auto.tfvars`, οπότε το a είναι λάθος. Το `prod.auto.tfvars` νικά τα δύο προηγούμενα, αλλά χάνει από τη γραμμή εντολών, οπότε το b είναι λάθος.

Ερώτηση 7.2

Ποια αρχεία με τιμές μεταβλητών φορτώνει το Terraform μόνο του, χωρίς `-var-file`; Επίλεξε δύο.

a. `terraform.tfvars`
b. `prod.tfvars`
c. `network.auto.tfvars`
d. `values.tfvars.json`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Αυτόματα φορτώνονται το `terraform.tfvars`, το `terraform.tfvars.json` και κάθε αρχείο που τελειώνει σε `.auto.tfvars` ή `.auto.tfvars.json`. Το `prod.tfvars` έχει τη σωστή κατάληξη αλλά όχι το σωστό όνομα, οπότε θέλει `-var-file`. Το `values.tfvars.json` δεν λέγεται `terraform.tfvars.json` και δεν περιέχει `.auto`, οπότε κι αυτό θέλει `-var-file`.

Ερώτηση 7.3

Η μεταβλητή `db_name` δεν έχει `default` και καμία πηγή δεν της δίνει τιμή. Το pipeline τρέχει `terraform apply -input=false -auto-approve`. Τι γίνεται;

a. Η μεταβλητή παίρνει κενό string και το apply συνεχίζει.
b. Το Terraform περιμένει να πληκτρολογήσει κάποιος την τιμή.
c. Το Terraform σταματά με σφάλμα ότι η υποχρεωτική μεταβλητή δεν έχει τιμή.
d. Το Terraform παραλείπει τα resources που χρησιμοποιούν τη μεταβλητή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `-input=false` απαγορεύει την ερώτηση, οπότε το b είναι λάθος. Μια υποχρεωτική μεταβλητή χωρίς τιμή δίνει το σφάλμα `No value for required variable` πριν από οποιαδήποτε αλλαγή. Το Terraform δεν εφευρίσκει τιμή, οπότε το a είναι λάθος. Δεν παραλείπει ποτέ resources για να συνεχίσει, οπότε το d είναι λάθος.

Ερώτηση 7.4

Θέλεις να δώσεις τιμή στη μεταβλητή `instance_type` από το περιβάλλον σε Linux. Ποια μεταβλητή περιβάλλοντος ορίζεις;

a. `TF_instance_type`
b. `TF_VAR_INSTANCE_TYPE`
c. `TF_VAR_instance_type`
d. `TERRAFORM_VAR_instance_type`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το πρόθεμα είναι ακριβώς `TF_VAR_` και ακολουθεί το όνομα της μεταβλητής όπως δηλώθηκε. Στο a λείπει το `VAR_` και στο d το πρόθεμα είναι λάθος. Το b έχει το όνομα με κεφαλαία. Σε Linux και macOS το όνομα ταιριάζει ακριβώς, οπότε το `TF_VAR_INSTANCE_TYPE` δεν δίνει τιμή σε τίποτα. Το Terraform αγνοεί σιωπηλά τις μεταβλητές περιβάλλοντος που δεν αντιστοιχούν σε δηλωμένη μεταβλητή.

Ερώτηση 7.5

Σωστό ή λάθος; Μια μεταβλητή με `default` και `nullable = false` παίρνει το `default` όταν ένα αρχείο μεταβλητών της δίνει `null`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το `nullable = false` λέει ότι η μεταβλητή δεν δέχεται ποτέ `null`. Αν έχει `default`, το `null` αντικαθίσταται από αυτό. Αν δεν έχει, το Terraform σταματά με σφάλμα. Με την προεπιλογή `nullable = true` το `null` θα περνούσε ως τιμή, παρά το `default`.

Ερώτηση 7.6

Ένα αρχείο `secrets.tfvars` κρατά τον κωδικό μιας βάσης δεδομένων. Ποια είναι η σωστή πρακτική;

a. Μπαίνει σε Git, ώστε να το έχει όλη η ομάδα.
b. Μένει έξω από το Git και ο κωδικός δίνεται την ώρα της εκτέλεσης, για παράδειγμα με `TF_VAR_`.
c. Μετονομάζεται σε `secrets.auto.tfvars` και μπαίνει σε Git, ώστε να μη χρειάζεται `-var-file`.
d. Η μεταβλητή παίρνει `sensitive = true` και τότε το αρχείο μπαίνει με ασφάλεια σε Git.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ό,τι μπαίνει σε Git το βλέπει κάθε μέλος του repository και μένει για πάντα στο ιστορικό, οπότε το a και το c είναι λάθος. Το c αλλάζει μόνο το πώς φορτώνεται το αρχείο. Το `sensitive` κρύβει την τιμή από την έξοδο του plan και του apply, αλλά το αρχείο εξακολουθεί να έχει τον κωδικό σε απλό κείμενο, οπότε το d είναι λάθος.

ΚΕΦΑΛΑΙΟ 8

Outputs και locals

Άσκηση 8.1 · Το πρώτο output

Γράψε configuration με ένα resource `local_file` με όνομα `note`, που φτιάχνει το `note.txt` με περιεχόμενο `Σημειώσεις έργου\n`. Πρόσθεσε output `note_path` με το `filename` του αρχείου και ένα `description`. Κάνε apply με `-auto-approve`. Το τέλος της εξόδου πρέπει να είναι το παρακάτω.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "note" {
  filename = "note.txt"
  content  = "Σημειώσεις έργου\n"
}

output "note_path" {
  value       = local_file.note.filename
  description = "Το αρχείο με τις σημειώσεις"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Outputs:

note_path = "note.txt"
```

Το output αναφέρει το attribute `filename` και όχι ένα σταθερό string. Αν αλλάξει το όνομα του αρχείου στο resource, το output ακολουθεί χωρίς δεύτερη αλλαγή. Το `description` δεν φαίνεται στην έξοδο. Υπάρχει για όποιον διαβάζει τον κώδικα.

Άσκηση 8.2 · Μια τιμή για script

Αντέγραψε τα αρχεία της άσκησης 8.1 σε νέο φάκελο, χωρίς το state. Τρέξε `terraform init` και apply. Με μία εντολή shell αντέγραψε το αρχείο που δείχνει το output `note_path` στο `backup.txt`, χωρίς να γράψεις το όνομα του αρχείου με το χέρι. Μετά τύπωσε το `backup.txt`.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
cp "$(terraform output -raw note_path)" backup.txt
cat backup.txt
```

ΕΞΟΔΟΣ

```
Σημειώσεις έργου
```

Το `-raw` δίνει το σκέτο `note.txt`. Χωρίς αυτό το `cp` θα έπαιρνε το όνομα μαζί με τα εισαγωγικά και θα απαντούσε ότι το αρχείο δεν υπάρχει. Τα διπλά εισαγωγικά γύρω από το `$(...)` προστατεύουν το όνομα αν κάποτε περιέχει κενά.

Άσκηση 8.3 · Ένα output, τρεις μορφές

Γράψε configuration χωρίς resources και χωρίς providers, με ένα μόνο output `city` και τιμή `θεσσαλονίκη`. Κάνε apply. Μετά τύπωσε το `city` τρεις φορές, με `terraform output city`, με `-raw` και με `-json`. Βάλε ένα `echo` μετά το `-raw`, ώστε κάθε τιμή να βγει σε δική της γραμμή.

ΛΥΣΗ · main.tf

```
output "city" {
  value = "θεσσαλονίκη"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output city
terraform output -raw city
echo
terraform output -json city
```

ΕΞΟΔΟΣ

```
"θεσσαλονίκη"
θεσσαλονίκη
"θεσσαλονίκη"
```

Η πρώτη μορφή γράφει την τιμή όπως θα τη γράφαμε σε HCL. Η δεύτερη τη δίνει σκέτη και χωρίς αλλαγή γραμμής, γι' αυτό χρειάστηκε το `echo`. Η τρίτη είναι JSON. Για ένα string η πρώτη και η τρίτη μοιάζουν, αλλά μόνο η τρίτη είναι εγγυημένα έγκυρο JSON για οποιαδήποτε τιμή. Μια configuration με μόνο outputs δεν χρειάζεται block `terraform`, γιατί δεν χρησιμοποιεί κανέναν provider.

Άσκηση 8.4 · Αλλαγή μόνο σε outputs

Αντέγραψε τα αρχεία της άσκησης 8.1 σε νέο φάκελο, τρέξε `terraform init` και apply. Μετά κάνε δύο αλλαγές. Άλλαξε το κείμενο του `description` στο `note_path` και πρόσθεσε output `note_id` με το `id` του `local_file.note`. Κάνε apply ξανά. Ποια από τις δύο αλλαγές εμφανίζεται στο plan και τι λέει η περίληψη;

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "note" {
  filename = "note.txt"
  content  = "Σημειώσεις έργου\n"
}

output "note_path" {
  value       = local_file.note.filename
  description = "Πού βρίσκονται οι σημειώσεις"
}

output "note_id" {
  value = local_file.note.id
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Changes to Outputs:
  + note_id   = "429b567c780425366f0afca15cfaea938cc9f9d6"
…
You can apply this plan to save these new output values to the Terraform
state, without changing any real infrastructure.
…
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
…
```

Στο plan εμφανίζεται μόνο το νέο `note_id` με `+`. Η αλλαγή στο `description` δεν φαίνεται καθόλου, γιατί δεν αλλάζει καμία τιμή και το description δεν γράφεται στο state. Η περίληψη λέει `0 added, 0 changed, 0 destroyed`. Το apply έγραψε μόνο τη νέα τιμή στο state, χωρίς να αγγίξει το αρχείο.

Άσκηση 8.5 · Από την επανάληψη σε local

Ξεκίνα από το παρακάτω `main.tf`. Τρέξε `terraform init` και apply. Μετά βάλε την έκφραση που επαναλαμβάνεται σε ένα local με όνομα `prefix` και χρησιμοποίησέ το στα τρία σημεία. Απόδειξε με `terraform plan` ότι η αλλαγή δεν αγγίζει κανένα resource.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "team" {
  type    = string
  default = "data"
}

variable "stage" {
  type    = string
  default = "test"
}

resource "local_file" "input" {
  filename = "${var.team}-${var.stage}-input.txt"
  content  = "είσοδος\n"
}

resource "local_file" "report" {
  filename = "${var.team}-${var.stage}-report.txt"
  content  = "αναφορά για ${var.team}-${var.stage}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "team" {
  type    = string
  default = "data"
}
```

```
variable "stage" {
  type    = string
  default = "test"
}

locals {
  prefix = "${var.team}-${var.stage}"
}

resource "local_file" "input" {
  filename = "${local.prefix}-input.txt"
  content  = "είσοδος\n"
}

resource "local_file" "report" {
  filename = "${local.prefix}-report.txt"
  content  = "αναφορά για ${local.prefix}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

Το plan συγκρίνει τελικές τιμές. Το `${local.prefix}-input.txt` δίνει ακριβώς το ίδιο `data-test-input.txt` με πριν, οπότε δεν υπάρχει αλλαγή. Αν το plan έδειχνε αντικατάσταση, θα σήμαινε ότι το local δεν δίνει την ίδια τιμή, για παράδειγμα επειδή ξέχασες την παύλα ανάμεσα στο `team` και στο `stage`.

Άσκηση 8.6 · Local ή variable

Ξεκίνα από το παρακάτω `main.tf`. Τρέξε `terraform init` και μετά `terraform apply -auto-approve -var environment=prod`. Διάβασε το σφάλμα. Άλλαξε τη configuration ώστε η εντολή να πετύχει, ενώ χωρίς `-var` το περιβάλλον να παραμένει `dev`. Στο τέλος τύπωσε το output `env_file` με `-raw`.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

locals {
  environment = "dev"
}

resource "local_file" "env" {
  filename = "${local.environment}.env"
  content  = "ENVIRONMENT=${local.environment}\n"
}

output "env_file" {
  value = local_file.env.filename
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve -var environment=prod
```

ΕΞΟΔΟΣ

```
╷
│ Error: Value for undeclared variable
│
│ A variable named "environment" was assigned on the command line, but the
│ root module does not declare a variable of that name. To use this value,
│ add a "variable" block to the configuration.
╵
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
```

```
    }
  }
}

variable "environment" {
  type    = string
  default = "dev"
}

resource "local_file" "env" {
  filename = "${var.environment}.env"
  content  = "ENVIRONMENT=${var.environment}\n"
}

output "env_file" {
  value = local_file.env.filename
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve -var environment=prod
```

ΛΥΣΗ · TERMINAL

```
terraform output -raw env_file
```

ΕΞΟΔΟΣ

```
prod.env
```

Το `-var` δίνει τιμή μόνο σε block `variable`. Το local `environment` δεν μετράει, γι' αυτό το πρώτο apply σταμάτησε με `Value for undeclared variable`. Το περιβάλλον είναι επιλογή του χρήστη, άρα ανήκει σε variable. Το `default` κρατά το `dev` όταν δεν δίνεται `-var`. Οι αναφορές αλλάζουν από `local.` σε `var.` και το block `locals` φεύγει.

Άσκηση 8.7 · Αριθμοί σε locals

Γράψε configuration χωρίς resources με δύο μεταβλητές τύπου `number`, τη `servers` με default `3` και τη `disk_gb` με default `20`. Υπολόγισε σε locals τον συνολικό δίσκο και το μηνιαίο κόστος, με 2 ευρώ ανά GB. Ο πολλαπλασιασμός γράφεται με `*`, όπως στις περισσότερες γλώσσες. Βγάλε ένα output `summary` με κείμενο της μορφής `3 servers, 60 GB, 120 ευρώ`. Κάνε apply με `-var servers=5` και τύπωσε το `summary` με `-raw`.

ΛΥΣΗ · main.tf

```
variable "servers" {
  type    = number
  default = 3
}

variable "disk_gb" {
  type    = number
  default = 20
}

locals {
  total_disk = var.servers * var.disk_gb
  cost       = local.total_disk * 2
}

output "summary" {
  value = "${var.servers} servers, ${local.total_disk} GB, ${local.cost} ευρώ"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve -var servers=5
```

ΛΥΣΗ · TERMINAL

```
terraform output -raw summary
```

ΕΞΟΔΟΣ

```
5 servers, 100 GB, 200 ευρώ
```

Το `cost` χρησιμοποιεί το `total_disk`, που με τη σειρά του χρησιμοποιεί τις μεταβλητές. Το Terraform βρίσκει τη σειρά υπολογισμού από τις αναφορές. Με `servers=5` ο δίσκος είναι 5 επί 20, δηλαδή 100 GB. Το κόστος είναι 200 ευρώ. Το `-var` άλλαξε μόνο τη μεταβλητή και όλα τα locals ακολούθησαν. Τα locals δεν θα μπορούσαν να είναι defaults μεταβλητών, γιατί το `default` δεν δέχεται αναφορές.

Ερωτήσεις πιστοποίησης

Ερώτηση 8.1

Ένα script που κάνει deployment χρειάζεται την τιμή του output `bucket_name` χωρίς εισαγωγικά, για να τη δώσει σε άλλη εντολή. Ποια εντολή ταιριάζει;

a. `terraform output bucket_name`
b. `terraform show bucket_name`
c. `terraform output -raw bucket_name`
d. `terraform output -json`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `-raw` τυπώνει τη σκέτη τιμή ενός output, χωρίς εισαγωγικά και χωρίς αλλαγή γραμμής. Το a τυπώνει την τιμή μέσα σε εισαγωγικά, που θα γίνονταν μέρος του ονόματος. Το b είναι λάθος γιατί το `terraform show` δέχεται διαδρομή αρχείου state ή plan, όχι όνομα output. Το d τυπώνει όλα τα outputs ως αντικείμενο JSON, που χρειάζεται ακόμη επεξεργασία με εργαλείο όπως το `jq`.

Ερώτηση 8.2

Ένα block `locals` ορίζει το `name_prefix`. Πώς το χρησιμοποιείς μέσα σε ένα resource;

a. `locals.name_prefix`
b. `local.name_prefix`
c. `var.name_prefix`
d. `locals["name_prefix"]`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το block λέγεται `locals`, αλλά η αναφορά ξεκινά με `local` στον ενικό. Στο a και στο d το Terraform θεωρεί το `locals` τύπο resource. Για το a απαντά ότι τέτοιο resource δεν έχει δηλωθεί. Για το d απαντά `Invalid reference`, γιατί μετά από τύπο resource περιμένει όνομα με τελεία. Το c αναφέρεται σε input variable και αποτυγχάνει, γιατί δεν υπάρχει block `variable "name_prefix"`.

Ερώτηση 8.3

Προσθέτεις ένα νέο block `output "endpoint"` στη configuration και αμέσως τρέχεις `terraform output endpoint`, χωρίς apply. Τι γίνεται;

a. Τυπώνεται η τιμή, που το Terraform υπολογίζει από τη configuration.
b. Τυπώνεται `(known after apply)`.
c. Τυπώνεται κενό string.
d. Η εντολή αποτυγχάνει με μήνυμα ότι το output δεν βρέθηκε, γιατί διαβάζει τα outputs από το state.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `terraform output` διαβάζει μόνο το state. Ένα output μπαίνει στο state με το επόμενο apply, οπότε μέχρι τότε η εντολή απαντά `Output "endpoint" not found`. Δεν υπολογίζει τίποτα από τη configuration, άρα το a είναι λάθος. Το `(known after apply)` εμφανίζεται μόνο στο plan, οπότε το b είναι λάθος. Ένα output που λείπει δεν έχει κενή τιμή, βγάζει σφάλμα, άρα και το c είναι λάθος.

Ερώτηση 8.4

Ποιες δύο προτάσεις για τα locals είναι σωστές; Επίλεξε δύο.

a. Παίρνουν τιμή από τη γραμμή εντολών με `-var`.
b. Μπορούν να χρησιμοποιούν input variables, attributes από resources και άλλα locals.
c. Γράφονται στο state και εμφανίζονται στο `terraform output`.
d. Ένας φάκελος μπορεί να έχει πολλά blocks `locals`, αρκεί κάθε όνομα να ορίζεται μία φορά.
e. Κάθε block `locals` ορίζει μία μόνο τιμή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Ένα local είναι όνομα για μια έκφραση και η έκφραση μπορεί να αναφέρει variables, attributes και άλλα locals. Επιτρέπονται πολλά blocks `locals`, αλλά ένα διπλό όνομα δίνει `Duplicate local value definition`. Το a είναι λάθος, γιατί το `-var` ψάχνει block `variable` και αποτυγχάνει. Το c είναι λάθος, γιατί τα locals υπολογίζονται σε κάθε εκτέλεση και δεν αποθηκεύονται. Για να φανεί η τιμή τους χρειάζεται output. Το e είναι λάθος, γιατί ένα block `locals` δέχεται όσες αναθέσεις θέλεις.

Ερώτηση 8.5

Το plan δείχνει μόνο την ενότητα `Changes to Outputs` και καμία αλλαγή σε resources. Τι κάνει το `terraform apply`;

a. Γράφει τις νέες τιμές των outputs στο state χωρίς να αλλάξει υποδομή.
b. Αντικαθιστά τα resources που αναφέρουν τα outputs.
c. Δεν κάνει τίποτα, γιατί τα outputs υπολογίζονται από την αρχή σε κάθε `terraform output`.
d. Αποτυγχάνει, γιατί ένα apply χρειάζεται τουλάχιστον μία αλλαγή σε resource.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Τα outputs αποθηκεύονται στο state και ένα apply χωρίς αλλαγές σε resources απλώς ενημερώνει εκεί τις τιμές τους. Η περίληψη λέει `0 added, 0 changed, 0 destroyed`. Ένα output δεν προκαλεί αντικατάσταση resource, άρα το b είναι λάθος. Το `terraform output` διαβάζει το state και δεν υπολογίζει τίποτα, άρα το c είναι λάθος. Το apply επιτρέπεται και όταν αλλάζουν μόνο outputs, άρα και το d είναι λάθος.

Ερώτηση 8.6

Θέλεις το πρόθεμα `"${var.project}-${var.environment}"` σε πολλά resources και δεν θέλεις να μπορεί να το αλλάξει κανείς από τη γραμμή εντολών. Τι γράφεις;

a. Ένα block `variable "prefix"` με αυτή την έκφραση στο `default`.
b. Ένα block `output "prefix"` και αναφορές `output.prefix` στα resources.
c. Ένα block `locals` με `prefix` ίσο με αυτή την έκφραση.
d. Μια γραμμή `prefix` με αυτή την έκφραση στο `terraform.tfvars`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Ένα local υπολογίζεται από άλλες τιμές και δεν παίρνει τιμή απ' έξω. Το a αποτυγχάνει με `Variables not allowed`, γιατί το `default` δέχεται μόνο σταθερή τιμή. Ακόμη κι αν δούλευε, μια μεταβλητή αλλάζει με `-var`. Το b αποτυγχάνει, γιατί ένα output δεν διαβάζεται μέσα στον ίδιο φάκελο. Το d είναι λάθος, γιατί ένα αρχείο `.tfvars` δίνει τιμή μόνο σε δηλωμένη μεταβλητή. Θα χρειαζόταν πάλι το block `variable` του a. Ακόμη και τότε η έκφραση δίνει `Variables not allowed`, γιατί τα αρχεία `.tfvars` δέχονται μόνο σταθερές τιμές.

ΚΕΦΑΛΑΙΟ 9

Τύποι και δομές δεδομένων

Άσκηση 9.1 · Από λίστα σε set

Άνοιξε το console σε άδειο φάκελο. Μετάτρεψε τη λίστα `["web", "db", "web", "cache"]` σε set και μέτρησε με `length` πόσα στοιχεία έχει το set.

ΛΥΣΗ · TERRAFORM CONSOLE

```
> toset(["web", "db", "web", "cache"])
toset([
  "cache",
  "db",
  "web",
])
> length(toset(["web", "db", "web", "cache"]))
3
```

Η λίστα είχε τέσσερα στοιχεία και το set τρία, γιατί το δεύτερο `"web"` είναι διπλότυπο. Η σειρά που δείχνει το console δεν είναι αυτή που έγραψες και δεν πρέπει να βασιστείς σε αυτήν.

Άσκηση 9.2 · Λίστα από το terraform.tfvars

Γράψε μεταβλητή `cities` τύπου `list(string)` και δώσε της στο `terraform.tfvars` τις πόλεις `Αθήνα`, `Θεσσαλονίκη` και `Πάτρα`. Γράψε output `first` με την πρώτη πόλη και output `total` με το πλήθος τους. Κάνε apply και δείξε τα outputs με `terraform output`.

ΛΥΣΗ · main.tf

```
variable "cities" {
  type = list(string)
}

output "first" {
  value = var.cities[0]
}

output "total" {
  value = length(var.cities)
}
```

ΛΥΣΗ · terraform.tfvars

```
cities = ["Αθήνα", "Θεσσαλονίκη", "Πάτρα"]
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output
```

ΕΞΟΔΟΣ

```
first = "Αθήνα"
total = 3
```

Το index ξεκινά από το 0, οπότε η πρώτη πόλη είναι το `var.cities[0]`. Το `length` μετρά τα στοιχεία της λίστας. Δεν χρειάστηκε init, γιατί η configuration δεν έχει provider.

Άσκηση 9.3 · Τιμή που δεν μετατρέπεται

Γράψε μεταβλητή `ports` τύπου `list(number)` και ένα output που την τυπώνει. Τρέξε `terraform plan` με `-var 'ports=["80", "443"]'` και μετά με `-var 'ports=["80", "http"]'`. Γιατί το πρώτο περνά και το δεύτερο όχι;

ΛΥΣΗ · main.tf

```
variable "ports" {
  type = list(number)
}

output "ports" {
  value = var.ports
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var 'ports=["80", "443"]'
```

ΕΞΟΔΟΣ

```
Changes to Outputs:
  + ports = [
      + 80,
      + 443,
    ]
…
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var 'ports=["80", "http"]'
```

ΕΞΟΔΟΣ

```
…
│ Unsuitable value for var.ports set using -var="ports=...": a number is
│ required.
…
```

Στο πρώτο plan τα `"80"` και `"443"` είναι strings που γράφουν αριθμούς. Το Terraform τα μετέτρεψε σε numbers και το output τα δείχνει χωρίς εισαγωγικά. Στο δεύτερο το `"http"` δεν γράφει αριθμό. Ένα μόνο στοιχείο που δεν μετατρέπεται αρκεί για να απορριφθεί όλη η τιμή.

Άσκηση 9.4 · Προεπιλογές με optional

Γράψε μεταβλητή `server` τύπου object με τα attributes `name` (string, υποχρεωτικό), `port` (number, προεπιλογή `8080`), `public` (bool, προεπιλογή `false`) και `owner` (string, προαιρετικό χωρίς προεπιλογή). Στο `terraform.tfvars` δώσε μόνο `name = "api"` και `public = true`. Δείξε στο console ολόκληρη τη μεταβλητή.

ΛΥΣΗ · main.tf

```
variable "server" {
  type = object({
    name   = string
    port   = optional(number, 8080)
    public = optional(bool, false)
    owner  = optional(string)
  })
}
```

ΛΥΣΗ · terraform.tfvars

```
server = {
  name   = "api"
  public = true
}
```

ΛΥΣΗ · TERRAFORM CONSOLE

```
> var.server
{
  "name" = "api"
  "owner" = tostring(null)
  "port" = 8080
  "public" = true
}
```

Το `port` έλειπε και πήρε την προεπιλογή `8080`. Το `public` δόθηκε, οπότε η προεπιλογή `false` δεν χρησιμοποιήθηκε. Το `owner` δεν έχει προεπιλογή, άρα είναι `null` με τύπο string, το `tostring(null)`. Χωρίς `optional` το Terraform θα απέρριπτε την τιμή, γιατί θα έλειπαν attributes.

Άσκηση 9.5 · Κλειδί από μεταβλητή

Γράψε μεταβλητή `sizes` τύπου `map(string)` με προεπιλογή `{ small = "t3.micro", large = "t3.large" }` και μεταβλητή `size` τύπου `string` με προεπιλογή `"small"`. Γράψε output `instance_type` που διαβάζει από το map την τιμή του κλειδιού που λέει το `size`. Κάνε apply με `-var size=large` και δείξε το output. Δοκίμασε μετά `terraform plan` με `-var size=medium`.

ΛΥΣΗ · main.tf

```
variable "sizes" {
  type = map(string)
  default = {
    small = "t3.micro"
    large = "t3.large"
  }
}

variable "size" {
  type    = string
  default = "small"
}

output "instance_type" {
  value = var.sizes[var.size]
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve -var size=large
```

ΛΥΣΗ · TERMINAL

```
terraform output instance_type
```

ΕΞΟΔΟΣ

```
"t3.large"
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var size=medium
```

ΕΞΟΔΟΣ

```
…
│ Error: Invalid index
…
│ The given key does not identify an element in this collection value.
…
```

Το κλειδί βρίσκεται σε μεταβλητή, οπότε χρειάζονται αγκύλες, `var.sizes[var.size]`. Το `var.sizes.size` θα έψαχνε κλειδί με όνομα `"size"`. Με `medium` το plan σταματά με `Invalid index`, γιατί το map δεν έχει τέτοιο κλειδί.

Άσκηση 9.6 · Αρχείο από object

Γράψε configuration με τον provider `local` και μία μεταβλητή `note` τύπου object με τα attributes `dir` (string), `files` (λίστα από strings), `text` (string) και `mode` (string, προεπιλογή `"0640"`). Ένα resource `local_file` γράφει το `text` στο αρχείο με όνομα το δεύτερο στοιχείο του `files`, μέσα στον φάκελο `dir`, με δικαιώματα `mode`. Στο `terraform.tfvars` δώσε `dir = "notes"`, `files = ["draft.txt", "final.txt"]` και ένα κείμενο. Τρέξε `terraform plan`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "note" {
  type = object({
    dir   = string
    files = list(string)
    text  = string
    mode  = optional(string, "0640")
  })
}

resource "local_file" "note" {
  filename        = "${var.note.dir}/${var.note.files[1]}"
  content         = var.note.text
  file_permission = var.note.mode
}
```

ΛΥΣΗ · terraform.tfvars

```
note = {
  dir   = "notes"
  files = ["draft.txt", "final.txt"]
  text  = "Τελικό κείμενο\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

```
ΕΞΟΔΟΣ

…
      + file_permission      = "0640"
      + filename             = "notes/final.txt"
…
```

Το δεύτερο στοιχείο της λίστας είναι το `[1]`, γιατί το index ξεκινά από το 0. Το `mode` έλειπε από το `terraform.tfvars` και πήρε την προεπιλογή `"0640"`, που έφτασε στο `file_permission`. Αν είχες γράψει `optional(string)` χωρίς προεπιλογή, το `mode` θα ήταν `null` και ο provider θα έβαζε τη δική του προεπιλογή, το `"0777"`.

Ερωτήσεις πιστοποίησης

Ερώτηση 9.1

Μια μεταβλητή κρατά ονόματα περιοχών. Δεν πρέπει να υπάρχουν διπλότυπα και η σειρά δεν έχει σημασία. Ποιος τύπος ταιριάζει καλύτερα;

a. `list(string)`
b. `set(string)`
c. `tuple([string])`
d. `map(string)`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το set κρατά μόνο μοναδικές τιμές και δεν έχει σειρά. Η list επιτρέπει διπλότυπα και κρατά σειρά που εδώ δεν χρειάζεται. Το `tuple([string])` έχει ακριβώς ένα στοιχείο. Το map θέλει κλειδί για κάθε τιμή, που εδώ δεν υπάρχει.

Ερώτηση 9.2

Μια μεταβλητή περιγράφει έναν server με όνομα, αριθμό CPU και μια ένδειξη αν είναι δημόσιος. Ποιος τύπος κρατά κάθε τιμή με τον δικό της τύπο;

a. `map(string)`
b. `list(any)`
c. `object({ name = string, cpus = number, public = bool })`
d. `map(any)`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Μόνο το object δίνει δικό του τύπο σε κάθε attribute. Το `map(string)` μετατρέπει τον αριθμό και το bool σε strings. Το `map(any)` θέλει κι αυτό έναν κοινό τύπο για όλες τις τιμές, οπότε καταλήγει επίσης σε strings. Το `list(any)` χάνει επιπλέον και τα ονόματα.

Ερώτηση 9.3

Ποιοι από τους παρακάτω είναι δομικοί τύποι (structural types), δηλαδή επιτρέπουν διαφορετικό τύπο σε κάθε στοιχείο; Επίλεξε δύο.

a. `list`
b. `tuple`
c. `map`
d. `object`
e. `set`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Το tuple έχει δικό του τύπο σε κάθε θέση και το object σε κάθε attribute. Τα list, map και set είναι συλλογές (collection types). Όλα τα στοιχεία τους έχουν τον ίδιο τύπο, αυτόν που γράφεται στην παρένθεση.

Ερώτηση 9.4

Ένα resource έχει `file_permission = var.mode` και η μεταβλητή `mode` έχει τιμή `null`. Τι κάνει το Terraform με το `file_permission`;

a. Το αντιμετωπίζει σαν να μη γράφτηκε, οπότε ισχύει η προεπιλογή του provider.
b. Στέλνει στον provider το string `"null"`.
c. Σταματά με σφάλμα τύπου.
d. Στέλνει στον provider κενό string.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Ένα argument με τιμή `null` είναι ισοδύναμο με argument που λείπει. Για προαιρετικό argument ισχύει η προεπιλογή του provider. Το `null` δεν είναι string, οπότε δεν γίνεται `"null"` ούτε κενό string. Δεν είναι και σφάλμα τύπου, γιατί το `null` ταιριάζει σε κάθε τύπο.

Ερώτηση 9.5

Μια μεταβλητή `app` έχει τύπο `object({ name = string, port = optional(number, 8080) })`. Το `terraform.tfvars` δίνει μόνο `name = "api"`. Ποια είναι η τιμή του `var.app.port`;

a. `null`
b. `0`
c. Δεν υπάρχει τιμή, γιατί το Terraform σταματά με σφάλμα για το attribute που λείπει.
d. `8080`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `optional` με δεύτερο όρισμα δίνει προεπιλογή, που ισχύει όταν το attribute λείπει. Το `null` θα ήταν η τιμή αν είχε γραφτεί `optional(number)` χωρίς προεπιλογή. Το Terraform δεν βάζει ποτέ μόνο του `0`. Σφάλμα θα έπαιρνες αν το `port` δεν ήταν δηλωμένο ως `optional`.

Ερώτηση 9.6

Σωστό ή λάθος; Το `terraform console` μπορεί να αλλάξει ένα resource, αν γράψεις μια έκφραση που αναφέρεται σε αυτό.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το console μόνο υπολογίζει εκφράσεις. Δεν φτιάχνει plan, δεν κάνει apply και δεν γράφει state. Μια αναφορά σε resource απλώς διαβάζει τα attributes του από το state.

ΚΕΦΑΛΑΙΟ 10

Εκφράσεις και συναρτήσεις

Άσκηση 10.1 · Ζυγός ή μονός

Γράψε στο console μια έκφραση που επιστρέφει `"ζυγός"` ή `"μονός"` για έναν αριθμό. Δοκίμασέ τη με το `7` και με το `10`.

ΛΥΣΗ · TERRAFORM CONSOLE

```
> 7 % 2 == 0 ? "ζυγός" : "μονός"
"μονός"
> 10 % 2 == 0 ? "ζυγός" : "μονός"
"ζυγός"
```

Το `%` δίνει το υπόλοιπο της διαίρεσης. Ένας ζυγός αριθμός έχει υπόλοιπο `0` όταν διαιρείται με το `2`. Η σύγκριση δίνει bool και η έκφραση υπό όρους διαλέγει το string. Οι δύο κλάδοι είναι strings, οπότε δεν υπάρχει θέμα τύπου.

Άσκηση 10.2 · Όνομα χωρίς κενά

Από το string `"  Web Server 01  "` φτιάξε με μία έκφραση το `"web-server-01"`. Χρησιμοποίησε τρεις συναρτήσεις string.

ΛΥΣΗ · TERRAFORM CONSOLE

```
> replace(lower(trimspace("  Web Server 01  ")), " ", "-")
"web-server-01"
```

Η πιο εσωτερική κλήση εκτελείται πρώτη. Η `trimspace` κόβει τα κενά των άκρων, η `lower` κάνει τα γράμματα πεζά και η `replace` αλλάζει τα εσωτερικά κενά σε παύλες. Η σειρά μετράει. Αν η `replace` έτρεχε πριν από την `trimspace`, τα κενά των άκρων θα γίνονταν κι αυτά παύλες.

Άσκηση 10.3 · Φίλτρο σε map

Ξεκίνα με το παρακάτω `main.tf`. Στο console γράψε μια έκφραση `for` που επιστρέφει λίστα με τα ονόματα, σε κεφαλαία, των services με θύρα πάνω από `1024`.

main.tf

```
variable "services" {
  type = map(number)
  default = {
    web   = 80
    api   = 8080
    admin = 9000
    db    = 5432
  }
}
```

ΛΥΣΗ · TERRAFORM CONSOLE

```
> [for name, port in var.services : upper(name) if port > 1024]
[
  "ADMIN",
  "API",
  "DB",
]
```

Οι τετράγωνες αγκύλες δίνουν tuple, μια σειρά τιμών όπως ζητήθηκε. Όπου χρειάζεται list, το Terraform το μετατρέπει μόνο του, όπως είδες στο κεφάλαιο 9. Σε map το `for` δίνει κλειδί και τιμή, οπότε το φίλτρο κοιτάζει τη θύρα και το αποτέλεσμα κρατά το όνομα. Το `web` λείπει γιατί η θύρα `80` δεν περνά το όριο. Η σειρά είναι αλφαβητική, `ADMIN`, `API`, `DB`, γιατί ένα map διαβάζεται με ταξινομημένα κλειδιά και όχι με τη σειρά που γράφτηκαν.

Άσκηση 10.4 · Ομάδες

Ξεκίνα με το παρακάτω `main.tf`. Στο console βρες πρώτα όλα τα emails με splat. Μετά φτιάξε object που έχει κλειδί την ομάδα και τιμή τη λίστα με τα ονόματα των μελών της.

main.tf

```
variable "users" {
  type = list(object({
    name  = string
    team  = string
    email = string
  }))
  default = [
    { name = "anna", team = "web", email = "anna@example.com" },
    { name = "nikos", team = "data", email = "nikos@example.com" },
    { name = "eleni", team = "web", email = "eleni@example.com" },
    { name = "maria", team = "data", email = "maria@example.com" },
  ]
}
```

ΛΥΣΗ · TERRAFORM CONSOLE

```
> var.users[*].email
tolist([
  "anna@example.com",
  "nikos@example.com",
  "eleni@example.com",
  "maria@example.com",
])
> {for u in var.users : u.team => u.name...}
{
  "data" = [
    "nikos",
    "maria",
  ]
  "web" = [
    "anna",
    "eleni",
  ]
}
```

Το `[*].email` ισοδυναμεί με `[for u in var.users : u.email]`. Στη δεύτερη έκφραση κάθε ομάδα έχει δύο μέλη, οπότε χωρίς τις τρεις τελείες το Terraform θα σταματούσε με `Duplicate object key`. Με το `...` κάθε ομάδα παίρνει λίστα με όλα τα ονόματά της, με τη σειρά που εμφανίζονται στη λίστα.

Άσκηση 10.5 · Ρυθμίσεις από JSON

Γράψε στον φάκελο της άσκησης το παρακάτω `settings.json`. Στο console διάβασε τον αριθμό `replicas`, μέτρησε τα `features` και βρες το `region`. Αν το `region` λείπει από το αρχείο, η έκφραση πρέπει να δίνει `"eu-central-1"`.

settings.json

```
{
  "app": "shop",
  "replicas": 2,
  "features": ["cart", "search", "reviews"]
}
```

ΛΥΣΗ · TERRAFORM CONSOLE

```
> jsondecode(file("settings.json")).replicas
2
> length(jsondecode(file("settings.json")).features)
3
> try(jsondecode(file("settings.json")).region, "eu-central-1")
"eu-central-1"
```

Η `file` διαβάζει το αρχείο σε string και η `jsondecode` το κάνει object του Terraform, οπότε τα πεδία διαβάζονται με τελεία. Το `region` δεν υπάρχει. Χωρίς την `try` η τρίτη έκφραση θα έβγαζε σφάλμα, ενώ με αυτή δίνει την προεπιλογή. Μια `lookup` θα έκανε κι αυτή τη δουλειά εδώ, αλλά η `try` καλύπτει και πεδία σε βάθος, όπως `.db.port`, όπου λείπει κάποιο ενδιάμεσο επίπεδο.

Άσκηση 10.6 · Αρχείο hosts από template

Γράψε ένα template `hosts.tftpl` και ένα `main.tf` με τον provider `local`. Η μεταβλητή `hosts` είναι map από όνομα σε διεύθυνση IP, με default `web = "10.0.1.10"`, `db = "10.0.2.20"` και `cache = "10.0.2.30"`. Ένα `local_file` γράφει το `hosts.txt` με μία γραμμή `IP όνομα` για κάθε host, χωρίς κενές γραμμές. Στο τέλος έρχεται η γραμμή `# σύνολο 3`, με αριθμό που υπολογίζεται. Κάνε apply και δείξε το αρχείο με `cat`.

ΛΥΣΗ · hosts.tftpl

```
%{ for name, ip in hosts ~}
${ip} ${name}
%{ endfor ~}
# σύνολο ${length(hosts)}
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}
```

```
variable "hosts" {
  type = map(string)
  default = {
    web   = "10.0.1.10"
    db    = "10.0.2.20"
    cache = "10.0.2.30"
  }
}

resource "local_file" "hosts" {
  filename = "${path.module}/hosts.txt"
  content  = templatefile("${path.module}/hosts.tftpl", { hosts = var.hosts })
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
cat hosts.txt
```

ΕΞΟΔΟΣ

```
10.0.2.30 cache
10.0.2.20 db
10.0.1.10 web
# σύνολο 3
```

Η `templatefile` περνά το map ως μεταβλητή `hosts` στο template. Το `%{ for name, ip in hosts ~}` γράφει μία γραμμή ανά host και το `~` σβήνει την αλλαγή γραμμής του ίδιου του directive. Χωρίς τα strip markers θα έβγαινε κενή γραμμή πριν από κάθε host. Η `length` μέσα στο template μετρά το map, οπότε ο αριθμός ακολουθεί αν προσθέσεις host. Τα hosts βγαίνουν αλφαβητικά, γιατί το map διαβάζεται με ταξινομημένα κλειδιά.

Ερωτήσεις πιστοποίησης

Ερώτηση 10.1

Μια ομάδα θέλει μια δική της συνάρτηση που φτιάχνει ονόματα resources με σταθερό μορφότυπο και να την καλεί από πολλά σημεία της configuration. Ποια πρόταση ισχύει;

a. Ορίζει τη συνάρτηση με ένα block `function` σε οποιοδήποτε αρχείο `.tf`.
b. Ορίζει τη συνάρτηση μέσα σε ένα block `locals` με παραμέτρους.
c. Το Terraform δεν υποστηρίζει συναρτήσεις του χρήστη. Υπάρχουν μόνο οι ενσωματωμένες και όσες δίνουν οι providers.
d. Γράφει τη συνάρτηση σε αρχείο `.tftpl` και την καλεί με το όνομά της.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Η γλώσσα του Terraform δεν έχει τρόπο να ορίσεις συνάρτηση. Δεν υπάρχει block `function`, οπότε το a είναι λάθος. Ένα local είναι μία τιμή χωρίς παραμέτρους, άρα το b δεν δουλεύει. Ένα αρχείο `.tftpl` το αποδίδει η `templatefile` και επιστρέφει string. Δεν γίνεται νέα συνάρτηση με δικό της όνομα, οπότε και το d είναι λάθος. Από την 1.8 υπάρχουν και συναρτήσεις providers, που όμως τις γράφει ο provider και όχι η configuration.

Ερώτηση 10.2

Η μεταβλητή `var.enabled` είναι τύπου `bool`. Τι συμβαίνει με την έκφραση `var.enabled ? ["a"] : { a = 1 }`;

a. Επιστρέφει tuple ή object ανάλογα με την τιμή της μεταβλητής.
b. Βγάζει σφάλμα, γιατί οι δύο κλάδοι δεν έχουν κοινό τύπο.
c. Επιστρέφει πάντα το tuple, γιατί είναι ο πρώτος κλάδος.
d. Μετατρέπει το object σε tuple και επιστρέφει tuple.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Μια έκφραση υπό όρους έχει έναν τύπο, που πρέπει να είναι γνωστός ανεξάρτητα από τη συνθήκη. Το Terraform προσπαθεί να βρει κοινό τύπο για τους δύο κλάδους. Ένα tuple και ένα object δεν έχουν, οπότε βγαίνει το σφάλμα `Inconsistent conditional result types`. Το a περιγράφει γλώσσα χωρίς έλεγχο τύπων. Το c αγνοεί τη συνθήκη, ενώ η έκφραση ελέγχει τους τύπους και των δύο κλάδων. Το d περιγράφει μετατροπή που το Terraform δεν κάνει.

Ερώτηση 10.3

Τι επιστρέφει η έκφραση `{for s in ["a", "b"] : s => upper(s)}`;

a. Το tuple `["A", "B"]`.
b. Μια λίστα με δύο objects, `[{ a = "A" }, { b = "B" }]`.
c. Σφάλμα, γιατί τα άγκιστρα θέλουν map ως είσοδο.
d. Ένα object με κλειδιά `a` και `b` και τιμές `"A"` και `"B"`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Τα άγκιστρα γύρω από το `for` φτιάχνουν object. Κάθε στοιχείο δίνει ένα ζεύγος `κλειδί => τιμή` και όλα τα ζεύγη μπαίνουν στο ίδιο object. Το a θα ήταν σωστό με τετράγωνες αγκύλες και χωρίς `=>`. Το b δεν το φτιάχνει καμία μορφή του `for`. Το c είναι λάθος γιατί η είσοδος μπορεί να είναι οποιαδήποτε συλλογή. Οι αγκύλες ορίζουν μόνο το αποτέλεσμα.

Ερώτηση 10.4

Η `var.servers` είναι λίστα από objects, καθένα με attribute `ip`. Ποια έκφραση δίνει το ίδιο αποτέλεσμα με το `var.servers[*].ip`;

a. `[for s in var.servers : s.ip]`
b. `{for s in var.servers : s => s.ip}`
c. `var.servers.ip`
d. `values(var.servers).ip`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το splat `[*]` είναι συντομογραφία για ένα `for` που παίρνει ένα attribute από κάθε στοιχείο και φτιάχνει λίστα. Το b φτιάχνει object και επιπλέον χρησιμοποιεί ολόκληρο object ως κλειδί, που δεν επιτρέπεται. Το c ζητά attribute από τη λίστα την ίδια, που δεν έχει attributes. Η `values` στο d δέχεται μόνο map ή object, όχι λίστα.

Ερώτηση 10.5

Ένα resource έχει `content = "Build ${timestamp()}"`. Έγινε apply και από τότε δεν άλλαξε τίποτα. Τι δείχνει το επόμενο `terraform plan`;

a. `No changes`, γιατί η configuration δεν άλλαξε.
b. Σφάλμα, γιατί η `timestamp` δεν επιτρέπεται σε argument resource.
c. Αλλαγή στο resource, γιατί η τιμή της `timestamp` είναι άγνωστη μέχρι το apply και αλλάζει σε κάθε εκτέλεση.
d. Αλλαγή μόνο αν έχει περάσει τουλάχιστον μία μέρα από το προηγούμενο apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Η `timestamp` υπολογίζεται κατά το apply. Στο plan η τιμή της είναι `(known after apply)`, οπότε το Terraform δεν μπορεί να υποθέσει ότι το argument μένει ίδιο και σημειώνει αλλαγή σε κάθε plan. Το a θα ίσχυε για σταθερή τιμή. Η συνάρτηση επιτρέπεται κανονικά, οπότε το b είναι λάθος. Το d είναι λάθος γιατί η τιμή είναι άγνωστη σε κάθε plan, όσος χρόνος κι αν πέρασε.

Ερώτηση 10.6

Ποιες εκφράσεις επιστρέφουν `7`; Επίλεξε δύο.

a. `max([3, 7, 5])`
b. `max([3, 7, 5]...)`
c. `max(3, 7, 5)`
d. `length([3, 7, 5])`
e. `lookup({ a = 7 }, "b", 0)`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και c

Η `max` δέχεται αριθμούς ως χωριστά arguments, όπως στο c. Οι τρεις τελείες στο b απλώνουν τη λίστα σε χωριστά arguments, οπότε δίνει κι αυτή `7`. Το a βγάζει σφάλμα, γιατί η `max` δεν δέχεται λίστα ολόκληρη. Το d μετρά στοιχεία και δίνει `3`. Το e ζητά κλειδί `b` που δεν υπάρχει, οπότε δίνει την προεπιλογή `0`.

ΚΕΦΑΛΑΙΟ 11

count, for_each και dynamic blocks

Άσκηση 11.1 · Τρία αντίγραφα

Γράψε ένα resource `local_file` με όνομα `server` και `count = 3`. Τα αρχεία λέγονται `server-1.txt`, `server-2.txt` και `server-3.txt`, δηλαδή η αρίθμηση ξεκινά από το 1. Κάθε αρχείο περιέχει τον αριθμό του και αλλαγή γραμμής. Ένα output με όνομα `files` δίνει όλα τα `filename` με splat. Κάνε apply.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "server" {
  count    = 3
  filename = "${path.module}/server-${count.index + 1}.txt"
  content  = "${count.index + 1}\n"
}

output "files" {
  value = local_file.server[*].filename
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

files = [
  "./server-1.txt",
  "./server-2.txt",
  "./server-3.txt",
]
```

Το `count.index` ξεκινά πάντα από το `0`, οπότε η αρίθμηση από το 1 θέλει `count.index + 1`. Οι διευθύνσεις στο state μένουν `local_file.server[0]` έως `local_file.server[2]`. Μόνο τα ονόματα των αρχείων ξεκινούν από το 1. Το `[*]` δουλεύει γιατί ένα resource με `count` είναι λίστα.

Άσκηση 11.2 · Διπλότυπα σε set

Η μεταβλητή `envs` είναι `list(string)` με default `["dev", "prod", "dev", "test"]`. Γράψε ένα `local_file` με όνομα `env` και `for_each`, που φτιάχνει ένα αρχείο `ΠΕΡΙΒΑΛΛΟΝ.env` για κάθε περιβάλλον με περιεχόμενο `ENVIRONMENT=ΠΕΡΙΒΑΛΛΟΝ` και αλλαγή γραμμής. Κάνε apply και δείξε τις διευθύνσεις με `terraform state list`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "envs" {
  type    = list(string)
  default = ["dev", "prod", "dev", "test"]
}

resource "local_file" "env" {
  for_each = toset(var.envs)
  filename = "${path.module}/${each.key}.env"
  content  = "ENVIRONMENT=${each.key}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
local_file.env["dev"]
local_file.env["prod"]
local_file.env["test"]
```

Η λίστα δεν περνά απευθείας στο `for_each`, οπότε χρειάζεται `toset`. Το set κρατά το `"dev"` μία φορά, άρα τα instances είναι τρία. Με `count = length(var.envs)` θα έβγαιναν τέσσερα instances και τα `[0]` και `[2]` θα διαχειρίζονταν το ίδιο `dev.env`. Αν αργότερα αφαιρούσες το δεύτερο `"dev"`, το `test` θα κατέβαινε στη θέση `2`. Το `[2]` θα αντικαθιστόταν για να γράψει το `test.env` και το `[3]` θα σβηνόταν. Η αντικατάσταση

σβήνει πρώτα το παλιό αρχείο του `[2]`, δηλαδή το `dev.env` που ανήκει και στο `[0]`. Μετά το apply το `dev.env` θα έλειπε και το επόμενο plan θα ήθελε να ξαναφτιάξει το `[0]`.

Άσκηση 11.3 · Πρόβλεψε το plan

Ξεκίνα από αντίγραφο του φακέλου `pages`, με το `count` πάνω στη λίστα σελίδων. Κάνε apply με τις προεπιλογές. Πριν τρέξεις plan, γράψε σε χαρτί τι θα πει η περίληψη αν προσθέσεις τη σελίδα `"blog"` στην αρχή της λίστας. Έλεγξε με `terraform plan -var` και τη λίστα `["blog", "home", "about", "contact"]`.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var 'pages=["blog", "home", "about", "contact"]'
```

ΕΞΟΔΟΣ

```
…
  # local_file.page[0] must be replaced
…
  # local_file.page[1] must be replaced
…
  # local_file.page[2] must be replaced
…
  # local_file.page[3] will be created
…
Plan: 4 to add, 0 to change, 3 to destroy.
…
```

Το `"blog"` πήρε τη θέση `0` και κάθε σελίδα μετακινήθηκε μία θέση. Τα instances `0`, `1` και `2` αλλάζουν περιεχόμενο και όνομα αρχείου, οπότε αντικαθίστανται. Το `[3]` είναι νέο. Η περίληψη λέει `4 to add` και `3 to destroy` για μία νέα σελίδα. Με `for_each` στον φάκελο `keyed` η ίδια αλλαγή δίνει μόνο ένα νέο instance, το `local_file.page["blog"]`, γιατί τα υπάρχοντα κλειδιά μένουν ίδια.

Άσκηση 11.4 · Αρχείο υπό όρους

Γράψε configuration με μεταβλητή `readme` τύπου `bool` και default `false`. Ένα `local_file` με όνομα `readme` φτιάχνει το `README.txt` μόνο όταν η μεταβλητή είναι `true`. Ένα output με όνομα `readme_path` δίνει το `filename` του ή `null`, χωρίς να σπάει όταν το αρχείο δεν υπάρχει. Κάνε apply με `-var readme=true` και μετά ξανά χωρίς το `-var`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
```

```
    }
  }
}

variable "readme" {
  type    = bool
  default = false
}

resource "local_file" "readme" {
  count    = var.readme ? 1 : 0
  filename = "${path.module}/README.txt"
  content  = "Διάβασέ με\n"
}

output "readme_path" {
  value = one(local_file.readme[*].filename)
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve -var readme=true
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

readme_path = "./README.txt"
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
  # local_file.readme[0] will be destroyed
…
  - readme_path = "./README.txt" -> null
…
Apply complete! Resources: 0 added, 0 changed, 1 destroyed.
```

Το `count` γίνεται `1` ή `0` από την έκφραση υπό όρους. Το δεύτερο apply σβήνει το αρχείο, γιατί η μεταβλητή γύρισε στο `false` και το instance `[0]` δεν υπάρχει πια στη configuration. Η `one` δίνει `null` όταν η λίστα είναι άδεια, οπότε το output απλώς εξαφανίζεται από την έξοδο. Ένα σκέτο `local_file.readme[0].filename` θα έσπαγε το δεύτερο plan με `Invalid index`.

Άσκηση 11.5 · Χρήστες από λίστα objects

Η μεταβλητή `members` είναι λίστα από objects με `name`, `team` και `active`, όπου το `active` είναι `optional(bool, true)`. Δώσε default με τρία μέλη, την `anna` στην ομάδα `web`, τον `nikos` στην ομάδα `data` με `active = false` και την `maria` στην ομάδα `data`. Με ένα `local_file` με όνομα `member` φτιάξε ένα αρχείο `members/ONOMA.txt` με περιεχόμενο την ομάδα, μόνο για τα ενεργά μέλη. Οι διευθύνσεις πρέπει να έχουν ως κλειδί το όνομα. Κάνε apply και δείξε τις διευθύνσεις με `terraform state list`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "members" {
  type = list(object({
    name   = string
    team   = string
    active = optional(bool, true)
  }))
  default = [
    { name = "anna", team = "web" },
    { name = "nikos", team = "data", active = false },
    { name = "maria", team = "data" },
  ]
}

resource "local_file" "member" {
  for_each = { for m in var.members : m.name => m if m.active }
  filename = "${path.module}/members/${each.key}.txt"
  content  = "${each.value.team}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
local_file.member["anna"]
local_file.member["maria"]
```

Η λίστα δεν περνά στο `for_each`. Η έκφραση `for` με άγκιστρα τη μετατρέπει σε map με κλειδί το `name` και το `if` πετά όσους δεν είναι ενεργοί. Ο `nikos` λείπει από το state. Αν αύριο κάποιος αφαιρέσει την `anna` από την αρχή της λίστας, η `maria` κρατά το κλειδί της και δεν αγγίζεται. Με `count` θα αντικαθιστόταν.

Άσκηση 11.6 · Σημειώσεις σε zip

Η μεταβλητή `notes` είναι λίστα από objects με `file` και `text`, με default δύο σημειώσεις, το `todo.txt` με κείμενο `να γράψω tests` και το `ideas.txt` με κείμενο `ένα module για όλα`. Φτιάξε ένα `archive_file` με όνομα `notes` που γράφει το `notes.zip` με ένα block `source` ανά σημείωση. Χρησιμοποίησε `dynamic` με `iterator = note`. Κάθε αρχείο τελειώνει με αλλαγή γραμμής. Κάνε apply και δείξε τα περιεχόμενα του zip με `unzip -l`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    archive = {
      source  = "hashicorp/archive"
      version = "2.8.1"
    }
  }
}

variable "notes" {
  type = list(object({
    file = string
    text = string
  }))
  default = [
    { file = "todo.txt", text = "να γράψω tests" },
    { file = "ideas.txt", text = "ένα module για όλα" },
  ]
}

resource "archive_file" "notes" {
  type        = "zip"
  output_path = "${path.module}/notes.zip"

  dynamic "source" {
    for_each = var.notes
    iterator = note
    content {
      filename = note.value.file
      content  = "${note.value.text}\n"
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
unzip -l notes.zip
```

```
ΕΞΟΔΟΣ

Archive:  notes.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
       28  00-00-1980 00:00   ideas.txt
       22  00-00-1980 00:00   todo.txt
---------                     -------
       50                     2 files
```

Το `dynamic "source"` παράγει ένα nested block `source` για κάθε στοιχείο της λίστας. Σε λίστα το `note.key` είναι η θέση και το `note.value` το object, οπότε τα πεδία διαβάζονται ως `note.value.file` και `note.value.text`. Χωρίς `iterator` θα έγραφες `source.value.file`. Το `count` ή το `for_each` στο resource θα έφτιαχναν δύο zip με ένα αρχείο το καθένα.

Ερωτήσεις πιστοποίησης

Ερώτηση 11.1

Ένα resource έχει `count = length(var.names)` με `names = ["a", "b", "c"]`. Έγινε apply. Αφαιρείς το `"b"` και τρέχεις `terraform plan`. Τι δείχνει το plan;

a. Σβήνει μόνο το instance του `"b"`.
b. Καμία αλλαγή, γιατί τα `"a"` και `"c"` υπάρχουν ακόμη.
c. Αλλάζει ή αντικαθιστά το instance `[1]` ώστε να γίνει `"c"` και σβήνει το `[2]`.
d. Σβήνει και τα τρία instances και φτιάχνει δύο νέα.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `count` αναγνωρίζει τα instances από τη θέση τους. Μετά την αφαίρεση το `"c"` είναι στη θέση `1`, οπότε το `[1]` αλλάζει περιεχόμενο και η θέση `2` δεν υπάρχει πια. Το a περιγράφει τη συμπεριφορά του `for_each`. Το b αγνοεί ότι οι θέσεις μετακινήθηκαν. Το d είναι λάθος γιατί το `[0]` είναι ακόμη το `"a"` και μένει ανέγγιχτο.

Ερώτηση 11.2

Ποιους τύπους τιμής δέχεται το meta-argument `for_each` σε ένα resource; Επίλεξε δύο.

a. `list(string)`
b. `map` με οποιονδήποτε τύπο τιμών
c. `number`
d. `set(string)`
e. `tuple`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Το `for_each` θέλει μοναδικά κλειδιά, οπότε δέχεται map ή set από strings. Μια λίστα ή ένα tuple έχουν σειρά και μπορεί να έχουν διπλότυπα. Το Terraform τα απορρίπτει με `Invalid for_each argument` και θέλουν `toset` ή έκφραση `for`. Ο αριθμός είναι για το `count`.

Ερώτηση 11.3

Το resource `random_pet` με όνομα `name` έχει `for_each = toset(["web", "db"])`. Ποια είναι η διεύθυνση του instance για το `"db"`;

a. `random_pet.name["db"]`
b. `random_pet.name[1]`
c. `random_pet.name.db`
d. `random_pet.db`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Τα instances ενός `for_each` αναγνωρίζονται από το κλειδί τους, που γράφεται σε αγκύλες με εισαγωγικά. Το b είναι η μορφή του `count`. Ένα resource με `for_each` είναι map και δεν έχει αριθμητικές θέσεις. Το c δεν είναι έγκυρη διεύθυνση και το `terraform state show` το απορρίπτει. Το d αλλάζει το όνομα του resource.

Ερώτηση 11.4

Σωστό ή λάθος; Ένα block `resource` μπορεί να έχει ταυτόχρονα `count` και `for_each`, για παράδειγμα δύο αντίγραφα για κάθε κλειδί.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Τα δύο meta-arguments αποκλείουν το ένα το άλλο. Το `terraform validate` σταματά με `Invalid combination of "count" and "for_each"`. Για δύο αντίγραφα ανά κλειδί φτιάχνεις ένα map με όλους τους συνδυασμούς και το δίνεις στο `for_each`.

Ερώτηση 11.5

Το resource `local_file.x` έχει `count = var.create ? 1 : 0`. Ένα output πρέπει να δίνει το `id` του όταν υπάρχει και `null` όταν δεν υπάρχει. Ποια έκφραση το κάνει χωρίς σφάλμα και στις δύο περιπτώσεις;

a. `local_file.x.id`
b. `local_file.x[0].id`
c. `local_file.x["0"].id`
d. `one(local_file.x[*].id)`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το splat δίνει λίστα με κανένα ή ένα `id` και η `one` τη μετατρέπει σε `null` ή στο ίδιο το `id`. Το a αγνοεί ότι ένα resource με `count` είναι λίστα και σταματά με `Missing resource instance key`. Το b δουλεύει μόνο όταν το `count` είναι `1`. Με `0` σταματά με `Invalid index`. Το c μοιάζει με κλειδί του `for_each`, αλλά το Terraform μετατρέπει το `"0"` σε αριθμό. Δουλεύει λοιπόν όπως το b και σπάει με τον ίδιο τρόπο όταν το `count` είναι `0`.

Ερώτηση 11.6

Ποια πρόταση για τα blocks `dynamic` είναι σωστή;

a. Παράγουν arguments με τιμές από μια συλλογή, για παράδειγμα πολλά `filename`.
b. Αντικαθιστούν το `for_each` όταν θέλεις πολλά instances ενός resource.
c. Παράγουν επαναλαμβανόμενα nested blocks και η τρέχουσα τιμή λέγεται όπως το label, εκτός αν οριστεί `iterator`.
d. Επιτρέπονται μόνο μέσα σε modules.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `dynamic` γράφει nested blocks όπως το `source` του `archive_file`. Το σώμα τους είναι το `content` και η τρέχουσα τιμή είναι το `ONOMA.key` και το `ONOMA.value`. Δεν παράγει arguments, οπότε το a είναι λάθος. Δεν πολλαπλασιάζει το resource, αυτό το κάνουν τα `count` και `for_each`, άρα και το b είναι λάθος. Δουλεύει σε κάθε resource, όχι μόνο σε modules.

ΚΕΦΑΛΑΙΟ 12

Data sources

Άσκηση 12.1 · Ένα αρχείο από έξω

Φτιάξε με `echo` το αρχείο `owner.txt` με περιεχόμενο `Μαρία`. Γράψε configuration με ένα data source `local_file` που το διαβάζει και ένα output `owner` με το περιεχόμενο χωρίς την αλλαγή γραμμής. Κάνε apply και τύπωσε το output με `terraform output -raw owner`.

ΛΥΣΗ · TERMINAL

```
echo "Μαρία" > owner.txt
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

data "local_file" "owner" {
  filename = "${path.module}/owner.txt"
}

output "owner" {
  value = trimspace(data.local_file.owner.content)
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output -raw owner
```

ΕΞΟΔΟΣ

```
Μαρία
```

Η αναφορά χρειάζεται το πρόθεμα `data`. Χωρίς αυτό το `validate` ψάχνει managed resource και αποτυγχάνει. Η `trimspace` κόβει την αλλαγή γραμμής που πρόσθεσε το `echo`. Το apply δεν φτιάχνει κανένα resource, αλλά γράφει το output στο state, οπότε το `terraform output` το βρίσκει.

Άσκηση 12.2 · Managed και data

Φτιάξε με `echo` το `motto.txt` με περιεχόμενο `less is more`. Γράψε configuration με ένα data source `motto` που το διαβάζει και ένα resource `local_file` με όνομα `loud`, που γράφει το ίδιο κείμενο με κεφαλαία στο `out/motto.txt`. Μετά το apply τύπωσε με `jq -c` έναν πίνακα `[mode, type, name]` για κάθε εγγραφή του state. Το `-c` γράφει κάθε αποτέλεσμα σε μία γραμμή.

ΛΥΣΗ · TERMINAL

```
echo "less is more" > motto.txt
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

data "local_file" "motto" {
  filename = "${path.module}/motto.txt"
}

resource "local_file" "loud" {
  filename = "${path.module}/out/motto.txt"
  content  = upper(data.local_file.motto.content)
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
jq -c '.resources[] | [.mode, .type, .name]' terraform.tfstate
```

ΕΞΟΔΟΣ

```
["data","local_file","motto"]
["managed","local_file","loud"]
```

Και οι δύο εγγραφές έχουν τύπο `local_file`. Τις χωρίζει το `mode`. Το data source είναι `data` και το resource `managed`. Ένα destroy εδώ θα έσβηνε μόνο το `out/motto.txt`. Το `motto.txt` δεν ανήκει στο Terraform.

Άσκηση 12.3 · Θύρες από JSON

Γράψε στον φάκελο της άσκησης το παρακάτω `ports.json`. Διάβασέ το με data source και γράψε output `high` με τα ονόματα των services που ακούνε σε θύρα πάνω από `8500`. Κάνε apply και τύπωσε το output με `terraform output -json high`.

ports.json

```
{
  "web": 8080,
  "api": 9090,
  "admin": 9000,
  "metrics": 2112
}
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

data "local_file" "ports" {
  filename = "${path.module}/ports.json"
}

locals {
  ports = jsondecode(data.local_file.ports.content)
}

output "high" {
  value = [for name, port in local.ports : name if port > 8500]
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output -json high
```

ΕΞΟΔΟΣ

```
["admin","api"]
```

Η `jsondecode` κάνει το κείμενο object, οπότε το `for` παίρνει κλειδί και τιμή. Το `if` κρατά τις θύρες πάνω από το όριο. Τα ονόματα βγαίνουν με αλφαβητική σειρά κλειδιών, `admin` και μετά `api`, όχι με τη σειρά

του αρχείου. Η παγίδα είναι να ξεχάσεις τη `jsondecode`. Το `content` είναι ένα string και δεν έχει πεδία για να περάσει το `for` από αυτά.

Άσκηση 12.4 · Plan ή apply;

Ξεκίνα με το παρακάτω `main.tf`. Πριν τρέξεις οτιδήποτε, σκέψου αν το data source θα διαβαστεί στο plan ή στο apply. Τρέξε `terraform plan` και βρες τις γραμμές που το επιβεβαιώνουν. Μετά κάνε apply και τρέξε ξανά plan.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_integer" "build" {
  min = 1
  max = 999
}

resource "local_file" "build" {
  filename = "${path.module}/builds/${random_integer.build.result}.txt"
  content  = "build ${random_integer.build.result}\n"
}

data "local_file" "build" {
  filename = local_file.build.filename
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # data.local_file.build will be read during apply
  # (config refers to values not yet known)
…
Plan: 2 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
data.local_file.build: Reading...
data.local_file.build: Read complete after 0s [id=5cf461b658cb835ed07d597e201db7dc91f585af]
…
No changes. Your infrastructure matches the configuration.
…
```

Το `filename` του data source εξαρτάται από το `result` του `random_integer`, που δεν υπάρχει πριν από το apply. Γι' αυτό το plan γράφει `will be read during apply` με αιτία `config refers to values not yet known`. Η περίληψη μετρά δύο resources, γιατί ένα data source δεν μετρά ποτέ. Μετά το apply όλα είναι γνωστά. Το δεύτερο plan διαβάζει το data source κανονικά και λέει `No changes`.

Άσκηση 12.5 · Zip χωρίς σημειώσεις

Φτιάξε τον φάκελο `web` με τρία αρχεία, τα `index.html`, `style.css` και `notes.md`, με όποιο περιεχόμενο θέλεις. Γράψε ένα data source `archive_file` που φτιάχνει το `build/site.zip` από τον φάκελο χωρίς το `notes.md`. Τρέξε μόνο `terraform plan` και δείξε τα αρχεία του zip με `unzip -Z1 build/site.zip`.

ΛΥΣΗ · TERMINAL

```
mkdir web
echo "<h1>Γεια</h1>" > web/index.html
echo "h1 { color: teal; }" > web/style.css
echo "σημειώσεις" > web/notes.md
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    archive = {
      source  = "hashicorp/archive"
      version = "2.8.1"
    }
  }
}

data "archive_file" "site" {
  type        = "zip"
  source_dir  = "${path.module}/web"
  excludes    = ["notes.md"]
  output_path = "${path.module}/build/site.zip"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform plan
```

ΛΥΣΗ · TERMINAL

```
unzip -Z1 build/site.zip
```

ΕΞΟΔΟΣ

```
index.html
style.css
```

Το `excludes` δέχεται ονόματα σχετικά με το `source_dir`, οπότε γράφεις `notes.md` και όχι `web/notes.md`. Το zip υπάρχει χωρίς apply, γιατί το `archive_file` το γράφει τη στιγμή που διαβάζεται και το plan διαβάζει τα data sources. Η παγίδα είναι να περιμένεις ότι το plan δεν αφήνει κανένα αρχείο πίσω του.

Άσκηση 12.6 · Ένα αρχείο ανά περιβάλλον

Φτιάξε τα αρχεία `env/dev.txt`, `env/stage.txt` και `env/prod.txt` με τους αριθμούς `1`, `2` και `3`. Είναι τα replicas κάθε περιβάλλοντος. Με ένα μόνο block `data` και `for_each` διάβασε και τα τρία. Γράψε output `replicas`, ένα map από περιβάλλον σε αριθμό. Κάνε apply και τύπωσε το output.

ΛΥΣΗ · TERMINAL

```
mkdir env
echo 1 > env/dev.txt
echo 2 > env/stage.txt
echo 3 > env/prod.txt
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

data "local_file" "env" {
  for_each = toset(["dev", "stage", "prod"])
  filename = "${path.module}/env/${each.key}.txt"
}

output "replicas" {
  value = {
    for name, f in data.local_file.env : name => tonumber(trimspace(f.content))
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output replicas
```

ΕΞΟΔΟΣ

```
{
  "dev" = 1
  "prod" = 3
  "stage" = 2
}
```

Με `for_each` το `data.local_file.env` είναι map με κλειδί το περιβάλλον. Το `for` με άγκιστρα φτιάχνει νέο map από αυτό. Η `trimspace` κόβει την αλλαγή γραμμής και η `tonumber` κάνει το κείμενο αριθμό. Χωρίς την `tonumber` οι τιμές θα ήταν τα strings `"1"`, `"2"` και `"3"`.

Άσκηση 12.7 · Το αρχείο που δεν υπήρχε ακόμη

Ξεκίνα με το παρακάτω `main.tf`. Τρέξε `terraform apply -auto-approve` και διάβασε το σφάλμα. Διόρθωσε το data source ώστε ένα apply σε καθαρό φάκελο να πετυχαίνει με την πρώτη. Τύπωσε μετά το output με `terraform output -raw config`.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "config" {
  filename = "${path.module}/out/config.txt"
  content  = "mode=fast\n"
}

data "local_file" "config" {
  filename = "${path.module}/out/config.txt"
}

output "config" {
  value = trimspace(data.local_file.config.content)
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
│ Error: Read local file data source error
…
│ +Original Error: open ./out/config.txt: no such file or directory
…
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "config" {
  filename = "${path.module}/out/config.txt"
```

```
  content  = "mode=fast\n"
}

data "local_file" "config" {
  filename = local_file.config.filename
}

output "config" {
  value = trimspace(data.local_file.config.content)
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output -raw config
```

ΕΞΟΔΟΣ

```
mode=fast
```

Στην αρχική μορφή το `filename` του data source είναι σταθερό string. Για το Terraform δεν υπάρχει καμία σχέση με το resource, όπως στο `backup` του κεφαλαίου 6. Όλα τα arguments είναι γνωστά και δεν υπάρχει εξάρτηση, οπότε το data source διαβάζεται στο plan, πριν φτιαχτεί το αρχείο. Η αναφορά `local_file.config.filename` φτιάχνει implicit dependency. Το data source περιμένει το resource και διαβάζεται στο apply. Ένα `depends_on` θα δούλευε κι αυτό, αλλά η αναφορά δείχνει καθαρά από πού έρχεται η τιμή.

Ερωτήσεις πιστοποίησης

Ερώτηση 12.1

Ποια πρόταση περιγράφει σωστά τη διαφορά ανάμεσα σε ένα block `resource` και ένα block `data`;

a. Το `data` φτιάχνει το αντικείμενο μόνο αν δεν υπάρχει ήδη, ενώ το `resource` το φτιάχνει πάντα.
b. Το `resource` διαχειρίζεται τον κύκλο ζωής ενός αντικειμένου, ενώ το `data` μόνο διαβάζει πληροφορία για κάτι που υπάρχει ήδη.
c. Το `data` δεν γράφεται ποτέ στο state, ενώ το `resource` γράφεται πάντα.
d. Το `data` υπάρχει μόνο σε providers που δεν μιλούν με cloud, όπως ο `local`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ένα data source ζητά τιμές από τον provider και δεν αλλάζει τίποτα. Το a περιγράφει μια συμπεριφορά «φτιάξε αν λείπει» που δεν έχει κανένα block του Terraform. Ένα data source που δεν βρίσκει αυτό που ζητά σταματά με σφάλμα. Το c είναι λάθος, γιατί το data source γράφεται στο state με `mode` ίσο με `data`. Το d είναι λάθος, γιατί οι providers του cloud έχουν τα περισσότερα data sources, όπως το `aws_ami`.

Ερώτηση 12.2

Μια configuration περιέχει το block `data "aws_ami" "web"`. Ποια έκφραση δίνει το ID του image σε ένα resource;

a. `aws_ami.web.id`
b. `data.web.id`
c. `data.aws_ami.web.id`
d. `ami.data.web.id`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Η διεύθυνση ενός data source είναι `data.τύπος.όνομα` και ακολουθεί το attribute. Το a δείχνει σε managed resource `aws_ami.web`, που δεν υπάρχει. Το `validate` το απορρίπτει με `Reference to undeclared resource`. Το b παραλείπει τον τύπο. Το d το Terraform το διαβάζει ως managed resource τύπου `ami` με όνομα `data`, που επίσης δεν υπάρχει.

Ερώτηση 12.3

Το plan δείχνει `<= data "local_file" "report"` με το σχόλιο `will be read during apply`. Ποια από τις παρακάτω μπορεί να είναι η αιτία;

a. Κάποιο argument του data source εξαρτάται από attribute ενός resource που θα δημιουργηθεί στο ίδιο apply.
b. Το αρχείο που διαβάζει το data source δεν υπάρχει.
c. Το plan τρέχει χωρίς `-out`, οπότε τα data sources διαβάζονται μόνο στο apply.
d. Τα data sources διαβάζονται πάντα στο apply και το σχόλιο εμφανίζεται σε κάθε plan.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Ένα data source διαβάζεται στο plan όταν όλα τα arguments του είναι γνωστά και δεν περιμένει resource με αλλαγές. Το a είναι μία από τις περιπτώσεις που μεταθέτουν την ανάγνωση στο apply. Μια άλλη, ένα `depends_on` προς resource με αλλαγές, δεν είναι στις επιλογές. Ένα αρχείο που λείπει, όπως στο b, δίνει το σφάλμα `Read local file data source error` και όχι αναβολή. Το `-out` αποθηκεύει το plan σε αρχείο και δεν αλλάζει πότε γίνεται η ανάγνωση, οπότε το c είναι λάθος. Το d είναι λάθος, γιατί στα περισσότερα plans η ανάγνωση γίνεται μέσα στο plan και το σύμβολο `<=` δεν εμφανίζεται.

Ερώτηση 12.4

Μια configuration έχει ένα resource `local_file` και ένα data source `local_file` που διαβάζει ένα αρχείο άλλης ομάδας. Τι κάνει το `terraform destroy`;

a. Σβήνει και τα δύο αρχεία, γιατί και τα δύο είναι γραμμένα στο state.
b. Σβήνει μόνο το αρχείο του resource. Το αρχείο της άλλης ομάδας μένει και το data source φεύγει από το state.
c. Αρνείται να τρέξει, γιατί ένα data source δεν μπορεί να σβηστεί.
d. Σβήνει μόνο το αρχείο του data source, γιατί διαβάστηκε τελευταίο.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το destroy σβήνει μόνο managed resources. Ένα data source δεν διαχειρίζεται το αντικείμενο που διαβάζει, οπότε το αρχείο μένει και η εγγραφή του data source απλώς φεύγει από το state. Το a θα έσβηνε κάτι που δεν ανήκει στο Terraform. Το c είναι λάθος, γιατί το destroy τρέχει κανονικά και στο plan διαγραφής εμφανίζεται μόνο το resource. Το d αντιστρέφει τους ρόλους.

Ερώτηση 12.5

Ποια από τα παρακάτω επιτρέπονται σε ένα block `data`; Επίλεξε δύο.

a. `for_each`
b. `lifecycle { prevent_destroy = true }`
c. `depends_on`
d. `lifecycle { create_before_destroy = true }`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Τα `count`, `for_each` και `depends_on` δουλεύουν σε data sources όπως σε resources. Το `depends_on` όμως καθυστερεί την ανάγνωση μέχρι το apply όταν η εξάρτηση έχει αλλαγές. Τα `prevent_destroy` και `create_before_destroy` αφορούν τη διαγραφή και την αντικατάσταση, που δεν υπάρχουν για ένα data source. Το Terraform τα απορρίπτει με το σφάλμα `Invalid data resource lifecycle argument`.

Ερώτηση 12.6

Σωστό ή λάθος; Το `terraform validate` αποτυγχάνει όταν λείπει το αρχείο που διαβάζει ένα data source `local_file`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `validate` ελέγχει τη σύνταξη, τους τύπους και τα arguments με βάση το schema του provider. Δεν ζητά από τον provider να διαβάσει κάτι, οπότε περνά. Το αρχείο που λείπει φαίνεται στο plan, όταν ο provider προσπαθεί να το ανοίξει και αποτυγχάνει.

ΚΕΦΑΛΑΙΟ 13

Validation, preconditions και check blocks

Άσκηση 13.1 · Επιτρεπτές περιοχές

Γράψε configuration με τον provider `local` και μια μεταβλητή `region` τύπου `string` με `default` το `eu-central-1`. Η μεταβλητή δέχεται μόνο τις τιμές `eu-central-1` και `eu-west-1`. Ένα `local_file` γράφει την περιοχή στο `region.txt`. Τρέξε `terraform plan -var region=us-east-1`. Το αναμενόμενο αποτέλεσμα είναι το σφάλμα. Το δικό σου μήνυμα θα διαφέρει στη διατύπωση.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "region" {
  type        = string
  description = "Η περιοχή του AWS"
  default     = "eu-central-1"

  validation {
    condition     = contains(["eu-central-1", "eu-west-1"], var.region)
    error_message = "Το region πρέπει να είναι eu-central-1 ή eu-west-1."
  }
}

resource "local_file" "region" {
  filename = "region.txt"
  content  = "${var.region}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var region=us-east-1
```

ΕΞΟΔΟΣ

```
…
╷
│ Error: Invalid value for variable
│
│   on main.tf line 10:
│   10: variable "region" {
│     ├────────────────
│     │ var.region is "us-east-1"
│
│ Το region πρέπει να είναι eu-central-1 ή eu-west-1.
│
│ This was checked by the validation rule at main.tf:15,3-13.
╵
```

Η `contains` επιστρέφει `false` για το `us-east-1`, οπότε η validation αποτυγχάνει πριν σχεδιαστεί το `local_file.region`. Το μήνυμα αναφέρει τις δύο επιτρεπτές τιμές. Ένα μήνυμα όπως «Λάθος περιοχή» θα ανάγκαζε τον χρήστη να ανοίξει τον κώδικα για να βρει τι επιτρέπεται.

Άσκηση 13.2 · Θύρα σε εύρος

Γράψε configuration με τον provider `local` και μια μεταβλητή `port` τύπου `number` με `default` το `8080`. Πρόσθεσε δύο validation blocks. Το πρώτο απαιτεί ακέραιο αριθμό και το δεύτερο τιμή από 1024 έως 65535. Ένα `local_file` γράφει τη θύρα στο `port.txt`. Τρέξε plan με `-var port=80.5`. Το αναμενόμενο αποτέλεσμα δείχνει τις γραμμές των σφαλμάτων και τα μηνύματα.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "port" {
  type        = number
  description = "Η θύρα της εφαρμογής"
  default     = 8080

  validation {
    condition     = floor(var.port) == var.port
    error_message = "Η θύρα πρέπει να είναι ακέραιος αριθμός."
  }

  validation {
    condition     = var.port >= 1024 && var.port <= 65535
    error_message = "Η θύρα πρέπει να είναι από 1024 έως 65535."
  }
}

resource "local_file" "port" {
  filename = "port.txt"
  content  = "${var.port}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var port=80.5
```

ΕΞΟΔΟΣ

```
╷
│ Error: Invalid value for variable
╷
│ Η θύρα πρέπει να είναι ακέραιος αριθμός.
╷
│ Error: Invalid value for variable
╷
│ Η θύρα πρέπει να είναι από 1024 έως 65535.
╵
```

Η `floor` του κεφαλαίου 10 κόβει το δεκαδικό μέρος. Ένας ακέραιος μένει ίδιος, οπότε η σύγκριση `floor(var.port) == var.port` είναι ο έλεγχος για ακέραιο. Το `80.5` παραβιάζει και τους δύο κανόνες και το Terraform αναφέρει δύο σφάλματα μαζί. Με έναν κανόνα και `&&` θα έβλεπες ένα μήνυμα και δεν θα ήξερες ποια από τις δύο απαιτήσεις έσπασε.

Άσκηση 13.3 · Το validate δεν βλέπει τιμές

Γράψε σε νέο φάκελο το `main.tf` που ακολουθεί. Πριν τρέξεις οτιδήποτε, πρόβλεψε τι θα πουν το `terraform validate` και το `terraform plan`. Μετά τρέξε `terraform init` και τις δύο εντολές με αυτή τη σειρά.

main.tf

```
variable "size" {
  type    = number
  default = 0

  validation {
    condition     = var.size > 0
    error_message = "Το size πρέπει να είναι θετικός αριθμός."
  }
}

output "size" {
  value = var.size
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
terraform plan
```

ΕΞΟΔΟΣ

```
Success! The configuration is valid.

╷
│ Error: Invalid value for variable
…
│ Το size πρέπει να είναι θετικός αριθμός.
╵
```

Το `validate` πέτυχε. Δεν χρησιμοποιεί τις τιμές των μεταβλητών, ούτε το `default`. Ελέγχει ότι ο κανόνας είναι σωστά γραμμένος και ότι αναφέρει τη μεταβλητή του, όχι αν μια τιμή τον περνά. Το `default` είναι τιμή όπως όλες οι άλλες και ελέγχεται στο plan. Ένα pipeline που τρέχει μόνο `validate` δεν θα έπιανε αυτό το λάθος. Η διόρθωση είναι ένα `default` που περνά τον κανόνα ή καθόλου `default`.

Άσκηση 13.4 · Το μήνυμα που δεν εμφανίζεται

Γράψε σε νέο φάκελο το `main.tf` που ακολουθεί και τρέξε `terraform plan -var bucket=My_Bucket`. Το σφάλμα δεν δείχνει το `error_message`. Διόρθωσε το condition ώστε ο χρήστης να βλέπει το δικό σου μήνυμα και ξανατρέξε την ίδια εντολή.

main.tf

```
variable "bucket" {
  type = string

  validation {
    condition     = regex("^[a-z0-9.-]+$", var.bucket) == var.bucket
    error_message = "Το bucket έχει μόνο πεζά γράμματα, ψηφία, τελείες και
παύλες."
  }
}

output "bucket" {
  value = var.bucket
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var bucket=My_Bucket
```

ΕΞΟΔΟΣ

```
╷
│ Error: Error in function call
╵
│ Call to function "regex" failed: pattern did not match any part of the
…
```

ΛΥΣΗ · main.tf

```
variable "bucket" {
  type = string

  validation {
    condition     = can(regex("^[a-z0-9.-]+$", var.bucket))
    error_message = "Το bucket έχει μόνο πεζά γράμματα, ψηφία, τελείες και παύλες."
  }
}

output "bucket" {
  value = var.bucket
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var bucket=My_Bucket
```

ΕΞΟΔΟΣ

```
╷
│ Error: Invalid value for variable
╵
│    1: variable "bucket" {
╵
│     ├─ var.bucket is "My_Bucket"
╵
│ Το bucket έχει μόνο πεζά γράμματα, ψηφία, τελείες και παύλες.
…
```

Η `regex` δεν βγάζει `false` όταν δεν ταιριάζει. Βγάζει σφάλμα και το Terraform σταματά πριν φτάσει στο `error_message`. Η `can` μετατρέπει το σφάλμα σε `false`, οπότε ο κανόνας αποτυγχάνει κανονικά και εμφανίζεται το μήνυμά σου. Δεν χρειάζεται πια η σύγκριση με το `var.bucket`, γιατί τα `^` και `$` απαιτούν να ταιριάξει όλο το string.

Άσκηση 13.5 · Precondition στο apply

Γράψε configuration με τους providers `local` και `random`. Ένα `random_integer` με όνομα `port` δίνει αριθμό από 8000 έως 8099 με `seed = "api"`. Ένα `local_file` με όνομα `conf` γράφει τη θύρα στο `app.conf` και έχει precondition ότι η θύρα δεν είναι η 8080. Πρόβλεψε αν ο έλεγχος θα αποτύχει στο plan ή στο apply. Τρέξε `terraform apply -auto-approve` και μετά `terraform state list`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_integer" "port" {
  min  = 8000
  max  = 8099
  seed = "api"
}

resource "local_file" "conf" {
  filename = "app.conf"
  content  = "port = ${random_integer.port.result}\n"

  lifecycle {
    precondition {
      condition     = random_integer.port.result != 8080
      error_message = "Η θύρα 8080 ανήκει στον proxy. Άλλαξε το seed του random_integer.port."
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
random_integer.port: Creating...
random_integer.port: Creation complete after 0s [id=8080]
…
╷
│ Error: Resource precondition failed
…
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
random_integer.port
```

Στο plan ο αριθμός ήταν `(known after apply)`, οπότε το precondition δεν μπορούσε να ελεγχθεί. Το Terraform το ανέβαλε για το apply. Εκεί φτιάχτηκε πρώτα το `random_integer.port`, γιατί το `local_file.conf` εξαρτάται από αυτό. Το precondition ελέγχθηκε πριν από τη δημιουργία του `local_file.conf`, απέτυχε και το αρχείο δεν φτιάχτηκε ποτέ. Γι' αυτό το state έχει μόνο το `random_integer.port`. Το precondition δεν τρέχει ποτέ μετά την αλλαγή του δικού του resource.

Άσκηση 13.6 · Postcondition στο plan

Γράψε configuration με τον provider `local` και ένα `local_file` με όνομα `secret` που γράφει το `secret.txt`. Πρόσθεσε postcondition ότι το `file_permission` του αρχείου είναι `0600`, χωρίς να ορίσεις το `file_permission`. Τρέξε `terraform plan`. Αποτυγχάνει στο plan ή θα περίμενε το apply; Μετά διόρθωσε το resource ώστε να περνά και κάνε apply.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "secret" {
  filename = "secret.txt"
  content  = "μόνο για μένα\n"

  lifecycle {
    postcondition {
      condition     = self.file_permission == "0600"
      error_message = "Το secret.txt έχει δικαιώματα ${self.file_permission}. Όρισε file_permission = \"0600\"."
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
╷ Error: Resource postcondition failed
…
│     ╷ self.file_permission is "0777"
…
╵ Το secret.txt έχει δικαιώματα 0777. Όρισε file_permission = "0600".
…
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "secret" {
  filename        = "secret.txt"
  content         = "μόνο για μένα\n"
  file_permission = "0600"

  lifecycle {
    postcondition {
      condition     = self.file_permission == "0600"
      error_message = "Το secret.txt έχει δικαιώματα ${self.file_permission}. Όρισε file_permission = \"0600\"."
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Το postcondition απέτυχε ήδη στο plan. Το `file_permission` δεν εξαρτάται από τίποτα που γίνεται γνωστό στο apply. Ο provider ξέρει από πριν ότι η προεπιλογή είναι `0777`, οπότε το Terraform έλεγξε το postcondition με τις τιμές που σχεδίαζε. Το «post» δεν σημαίνει «πάντα στο apply». Σημαίνει ότι ελέγχει το αντικείμενο όπως θα είναι μετά την αλλαγή.

Άσκηση 13.7 · Check μετά το apply

Γράψε configuration με τον provider `local`. Ένα `local_file` με όνομα `motd` γράφει στο `motd.txt` το κείμενο `Καλώς ήρθες στον server` με αλλαγή γραμμής στο τέλος. Ένα check block με όνομα `motd` διαβάζει το αρχείο με scoped data source και ελέγχει ότι έχει έως 20 χαρακτήρες. Πάρε το `filename` του data source από το `local_file.motd`, ώστε να διαβαστεί μετά τη δημιουργία. Κάνε apply και δες αν ολοκληρώθηκε.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "motd" {
  filename = "motd.txt"
  content  = "Καλώς ήρθες στον server\n"
}

check "motd" {
  data "local_file" "motd" {
    filename = local_file.motd.filename
  }

  assert {
    condition     = length(data.local_file.motd.content) <= 20
    error_message = "Το motd.txt έχει ${length(data.local_file.motd.content)} χαρακτήρες. Το όριο είναι 20."
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
│ Warning: Check block assertion failed
…
│ Το motd.txt έχει 24 χαρακτήρες. Το όριο είναι 20.
…
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Το check απέτυχε ως warning. Το apply ολοκληρώθηκε και το αρχείο φτιάχτηκε. Η αναφορά στο `local_file.motd.filename` λέει στο Terraform ότι το data source χρειάζεται το resource, όπως κάθε αναφορά του κεφαλαίου 6. Έτσι ένα plan πριν από το πρώτο apply δεν προσπαθεί να διαβάσει ένα αρχείο που δεν υπάρχει ακόμη. Χωρίς αυτή θα έβλεπες στο plan προειδοποίηση για αρχείο που λείπει, όπως στο `status.txt` του κεφαλαίου. Αν το όριο ήταν απαράβατο, θα έγραφες postcondition στο `local_file.motd` και η δουλειά θα σταματούσε με error.

Ερωτήσεις πιστοποίησης

Ερώτηση 13.1

Η μεταβλητή `instance_type` πρέπει να είναι `t3.micro` ή `t3.small`. Θέλεις μια λάθος τιμή να απορρίπτεται με error πριν τη χρησιμοποιήσει οποιοδήποτε resource. Τι χρησιμοποιείς;

a. Ένα check block με assert για το `var.instance_type`.
b. Ένα block `validation` μέσα στο block `variable`.
c. Ένα postcondition στο resource που χρησιμοποιεί τη μεταβλητή.
d. Ένα precondition σε output με την τιμή της μεταβλητής.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η validation ανήκει στη μεταβλητή και ελέγχεται μόλις το Terraform μάθει την τιμή, πριν από κάθε resource που τη χρησιμοποιεί. Όταν αποτύχει, σταματά το plan με error. Το check block βγάζει μόνο warning, οπότε δεν σταματά τίποτα. Το postcondition ελέγχει το αποτέλεσμα ενός resource, δηλαδή αφού η τιμή έχει ήδη χρησιμοποιηθεί. Το precondition σε output ελέγχει ό,τι επιστρέφει η configuration. Δεν εμποδίζει τα resources που διαβάζουν τη μεταβλητή να σχεδιαστούν με τη λάθος τιμή.

Ερώτηση 13.2

Ένα assert σε check block αποτυγχάνει κατά το `terraform apply`. Τι κάνει το Terraform;

a. Σταματά το apply και αναιρεί όσες αλλαγές έγιναν.
b. Σημειώνει ως tainted τα resources που αναφέρει το assert.
c. Ολοκληρώνει το apply και εμφανίζει warning με το μήνυμα του assert.
d. Σταματά πριν από κάθε αλλαγή και βγάζει error.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Τα check blocks βγάζουν πάντα warning και δεν σταματούν ποτέ ένα plan ή ένα apply. Το Terraform δεν αναιρεί αλλαγές σε καμία περίπτωση, οπότε το a είναι λάθος. Δεν σημειώνει τίποτα ως tainted, οπότε και το b είναι λάθος. Το d περιγράφει validation ή precondition, όχι check.

Ερώτηση 13.3

Ένα resource φτιάχνεται κατά το apply και αμέσως μετά το postcondition του αποτυγχάνει. Τι ισχύει;

a. Το apply αναφέρει error, το αντικείμενο μένει στο state και όσα εξαρτώνται από αυτό δεν φτιάχνονται.
b. Το Terraform σβήνει αυτόματα το αντικείμενο, ώστε το state να μείνει καθαρό.
c. Το αντικείμενο σημειώνεται ως tainted και αντικαθίσταται στο επόμενο apply.
d. Το apply βγάζει warning και συνεχίζει με τα υπόλοιπα resources.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το αντικείμενο υπάρχει στ' αλήθεια, οπότε το Terraform το κρατά στο state. Δεν το σβήνει, άρα το b είναι λάθος. Δεν το σημειώνει ως tainted, άρα το c είναι λάθος. Το επόμενο plan θα αποτύχει ξανά στο ίδιο postcondition, μέχρι να αλλάξεις την αιτία. Το postcondition βγάζει error, όχι warning. Τα εξαρτώμενα resources δεν φτιάχνονται, οπότε και το d είναι λάθος.

Ερώτηση 13.4

Πού μπορείς να γράψεις ένα block `precondition`; Επίλεξε δύο.

a. Μέσα σε block `variable`
b. Μέσα σε block `output`
c. Μέσα στο `lifecycle` ενός resource
d. Μέσα σε block `provider`
e. Μέσα στο block `terraform`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και c

Το precondition μπαίνει στο `lifecycle` ενός resource ή ενός data source και απευθείας σε ένα block `output`. Μια μεταβλητή ελέγχεται με `validation`, όχι με precondition. Τα blocks `provider` και `terraform` δεν δέχονται conditions.

Ερώτηση 13.5

Σωστό ή λάθος; Στο Terraform 1.12 το `condition` ενός `validation` στη μεταβλητή `max_size` μπορεί να συγκρίνει το `var.max_size` με το `var.min_size`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Από την έκδοση 1.9 ένας κανόνας validation μπορεί να αναφέρει άλλες μεταβλητές, locals και data sources του ίδιου module. Πριν από την 1.9 έβλεπε μόνο τη δική του μεταβλητή και ο κανόνας απορριπτόταν με `Invalid reference in variable validation`. Το condition πρέπει πάντως να αναφέρει και τη δική του μεταβλητή, εδώ το `var.max_size`.

Ερώτηση 13.6

Μια μεταβλητή έχει `default = 0` και validation με `condition = var.size > 0`. Τρέχεις `terraform validate` και μετά `terraform plan` χωρίς άλλες τιμές. Τι συμβαίνει;

a. Το `validate` αποτυγχάνει, γιατί το `default` παραβιάζει τον κανόνα.
b. Και οι δύο εντολές πετυχαίνουν, γιατί το `default` δεν ελέγχεται ποτέ.
c. Το `plan` ρωτά για νέα τιμή, γιατί το `default` είναι άκυρο.
d. Το `validate` πετυχαίνει και το `plan` αποτυγχάνει με `Invalid value for variable`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `validate` ελέγχει τη σύνταξη και τις αναφορές της configuration, όχι τις τιμές των μεταβλητών. Το `default` είναι τιμή όπως κάθε άλλη και ελέγχεται στο plan, οπότε το a και το b είναι λάθος. Το Terraform ρωτά μόνο για μεταβλητές χωρίς τιμή. Αυτή έχει τιμή, απλώς άκυρη, οπότε το plan σταματά με error αντί να ρωτήσει.

ΚΕΦΑΛΑΙΟ 14

Lifecycle και αντικατάσταση

Άσκηση 14.1 · Μοναδικό όνομα

Ξεκίνα από αντίγραφο του φακέλου `names`, χωρίς state. Άλλαξε το `filename` ώστε να περιέχει την έκδοση, για παράδειγμα `release-1.0.html`. Κράτα το `create_before_destroy`. Κάνε apply με την προεπιλεγμένη έκδοση και μετά με `-var release=1.1`. Το `ls` στο τέλος πρέπει να δείχνει μόνο τη νέα σελίδα.

ΛΥΣΗ · main.tf

```
variable "release" {
  type    = string
  default = "1.0"
}

resource "local_file" "page" {
  filename = "release-${var.release}.html"
  content  = "<h1>Έκδοση ${var.release}</h1>\n"

  lifecycle {
    create_before_destroy = true
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
terraform apply -auto-approve -var release=1.1
```

ΛΥΣΗ · TERMINAL

```
ls
```

ΕΞΟΔΟΣ

```
main.tf
release-1.1.html
terraform.tfstate
terraform.tfstate.backup
versions.tf
```

Με την έκδοση μέσα στο όνομα, το νέο αντικείμενο γράφει το `release-1.1.html` και το destroy του παλιού σβήνει το `release-1.0.html`. Τα δύο αρχεία υπήρξαν μαζί για λίγο και κανένα destroy δεν πείραξε το αρχείο του άλλου. Η παγίδα που αποφεύγεις είναι αυτή της ενότητας, ένα σταθερό όνομα που σβήνει το νέο αντικείμενο μαζί με το παλιό.

Άσκηση 14.2 · Σβήσε μόνο ό,τι επιτρέπεται

Γράψε configuration με δύο `terraform_data`. Το `db` έχει `input = "customers"` και `prevent_destroy`. Το `cache` έχει `input = "redis"`. Κάνε apply. Σβήσε μετά μόνο το `cache` χωρίς να αλλάξεις τον κώδικα. Δείξε τι έμεινε στο state.

ΛΥΣΗ · main.tf

```
resource "terraform_data" "db" {
  input = "customers"

  lifecycle {
    prevent_destroy = true
  }
}

resource "terraform_data" "cache" {
  input = "redis"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -target=terraform_data.cache -auto-approve
```

ΕΞΟΔΟΣ

```
…
Destroy complete! Resources: 1 destroyed.
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
terraform_data.db
```

Ένα σκέτο `terraform destroy` θα αποτύγχανε με `Instance cannot be destroyed` και δεν θα έσβηνε ούτε το `cache`. Το `-target` περιορίζει το plan στο `cache`, οπότε το `db` δεν είναι μέσα και το `prevent_destroy` δεν ενεργοποιείται. Είναι η δεύτερη διέξοδος που προτείνει το ίδιο το μήνυμα σφάλματος. Το `-target` μένει εξαίρεση, όπως λέει το κεφάλαιο 3.

Άσκηση 14.3 · Η ώρα δημιουργίας

Στο κεφάλαιο 10 ένα `local_file` με `timestamp()` στο `content` αντικαθιστούνταν σε κάθε plan. Γράψε το `local_file.stamp` με `filename = "stamp.txt"` και `content = "Δημιουργήθηκε ${timestamp()}\n"` ώστε να κρατά την ώρα της πρώτης δημιουργίας. Κάνε apply και μετά plan.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "stamp" {
  filename = "stamp.txt"
  content  = "Δημιουργήθηκε ${timestamp()}\n"

  lifecycle {
    ignore_changes = [content]
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

Το `timestamp()` δίνει νέα τιμή σε κάθε plan. Με `ignore_changes = [content]` το Terraform χρησιμοποιεί την τιμή μόνο στη δημιουργία και μετά δεν συγκρίνει το `content`. Το αρχείο κρατά την ώρα του πρώτου apply. Η παγίδα είναι να βάλεις στη λίστα ένα attribute που υπολογίζει ο provider, όπως το `id` ή το `content_md5`. Παίρνεις μόνο την προειδοποίηση `Redundant ignore_changes element` και το αρχείο αντικαθίσταται σε κάθε apply όπως πριν.

Άσκηση 14.4 · Restart όταν αλλάζει η έκδοση

Γράψε configuration με μια μεταβλητή `app_version` με default `"1.0"` και ένα `local_file.app` που γράφει το `app.conf` με περιεχόμενο `port = 8080\n`. Το αρχείο δεν περιέχει την έκδοση, αλλά πρέπει να ξαναφτιάχνεται κάθε φορά που αλλάζει η `app_version`. Αν χρειαστείς βοηθητικό resource, δώσε του το όνομα `app_version`. Κάνε apply και μετά plan με `-var app_version=1.1`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "app_version" {
  type    = string
  default = "1.0"
}

resource "terraform_data" "app_version" {
  input = var.app_version
}

resource "local_file" "app" {
  filename = "app.conf"
  content  = "port = 8080\n"

  lifecycle {
    replace_triggered_by = [terraform_data.app_version]
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var app_version=1.1
```

ΕΞΟΔΟΣ

```
…
  # local_file.app will be replaced due to changes in replace_triggered_by
…
  # terraform_data.app_version will be updated in-place
…
Plan: 1 to add, 1 to change, 1 to destroy.
…
```

Το `replace_triggered_by = [var.app_version]` θα σταματούσε με `Invalid reference in replace_triggered_by expression`. Η μεταβλητή περνά από το `terraform_data.app_version`, που αλλάζει επί τόπου όταν αλλάζει η τιμή. Αυτή η αλλαγή του resource ενεργοποιεί την αντικατάσταση του `local_file.app`.

Άσκηση 14.5 · keepers ή όχι

Γράψε configuration με δύο μεταβλητές, `image` με default `"ubuntu-24.04"` και `size` με default `"small"`. Ένα `random_pet.name` πρέπει να αλλάζει μόνο όταν αλλάζει το `image`. Ένα `local_file.server` γράφει το `server.txt` με το όνομα και το `size`. Κάνε apply. Δείξε ότι ένα plan με `-var size=large` δεν αλλάζει το όνομα.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

variable "image" {
  type    = string
  default = "ubuntu-24.04"
}

variable "size" {
  type    = string
  default = "small"
}

resource "random_pet" "name" {
  keepers = {
    image = var.image
  }
}

resource "local_file" "server" {
  filename = "server.txt"
  content  = "${random_pet.name.id} ${var.size}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var size=large
```

ΕΞΟΔΟΣ

```
…
  # local_file.server must be replaced
…
Plan: 1 to add, 0 to change, 1 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform plan -var image=ubuntu-26.04
```

ΕΞΟΔΟΣ

```
…
  # local_file.server must be replaced
…
  # random_pet.name must be replaced
…
Plan: 2 to add, 0 to change, 2 to destroy.
…
```

Με `-var size=large` αλλάζει μόνο το `content` του αρχείου, οπότε μόνο το `local_file.server` αντικαθίσταται. Το `random_pet.name` κρατά το όνομα, γιατί το `keepers` του δεν άλλαξε. Με `-var image=ubuntu-26.04` αλλάζει το `keepers` και αντικαθίστανται και τα δύο. Αν έβαζες το `size` στο `keepers`, κάθε αλλαγή μεγέθους θα έδινε και νέο όνομα.

Άσκηση 14.6 · Αντικατάσταση παρά το ignore_changes

Γράψε δύο `terraform_data`. Το `config` έχει `input = "v1"`. Το `server` έχει `input = "owner-anna"`, αγνοεί τις αλλαγές στο `input` και αντικαθίσταται όταν αλλάζει το `config`. Κάνε apply. Άλλαξε το `input` του `server` σε `"owner-nikos"` και επιβεβαίωσε με plan ότι δεν αλλάζει τίποτα. Άλλαξε μετά το `input` του `config` σε `"v2"` και κάνε apply. Ποια τιμή έχει τώρα το `input` του `server` στο state;

ΛΥΣΗ · main.tf

```
resource "terraform_data" "config" {
  input = "v1"
}

resource "terraform_data" "server" {
  input = "owner-anna"

  lifecycle {
    ignore_changes       = [input]
    replace_triggered_by = [terraform_data.config]
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "terraform_data" "config" {
  input = "v1"
}

resource "terraform_data" "server" {
  input = "owner-nikos"

  lifecycle {
    ignore_changes       = [input]
    replace_triggered_by = [terraform_data.config]
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

ΛΥΣΗ · main.tf

```
resource "terraform_data" "config" {
  input = "v2"
}

resource "terraform_data" "server" {
  input = "owner-nikos"

  lifecycle {
    ignore_changes       = [input]
    replace_triggered_by = [terraform_data.config]
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
  # terraform_data.config will be updated in-place
…
  # terraform_data.server will be replaced due to changes in replace_triggered_by
…
Apply complete! Resources: 1 added, 1 changed, 1 destroyed.
```

ΛΥΣΗ · TERMINAL

```
terraform state show terraform_data.server
```

ΕΞΟΔΟΣ

```
# terraform_data.server:
resource "terraform_data" "server" {
    id     = "5652ceff-e869-5728-25ab-78e9518940f2"
    input  = "owner-nikos"
    output = "owner-nikos"
}
```

Το πρώτο plan δεν βλέπει τίποτα, γιατί το `input` του `server` αγνοείται. Η αλλαγή στο `config` ενεργοποιεί το `replace_triggered_by` και ο `server` αντικαθίσταται. Το νέο αντικείμενο φτιάχνεται από την αρχή, οπότε παίρνει την τιμή του κώδικα, `owner-nikos`. Το `ignore_changes` κρύβει τη διαφορά από ένα αντικείμενο που υπάρχει. Δεν αλλάζει τις τιμές με τις οποίες φτιάχνεται ένα νέο. Η παγίδα είναι να πιστέψεις ότι μια τιμή που αγνοείται μένει για πάντα. Κάθε αντικατάσταση τη γράφει ξανά από τον κώδικα.

Ερωτήσεις πιστοποίησης

Ερώτηση 14.1

Κάθε αλλαγή στο image ενός server αντικαθιστά τον server και η υπηρεσία μένει εκτός για λίγα λεπτά. Ποια ρύθμιση κάνει το Terraform να φτιάχνει τον νέο server πριν σβήσει τον παλιό;

a. `prevent_destroy = true`
b. `create_before_destroy = true`
c. `ignore_changes = [ami]`
d. `replace_triggered_by = [aws_instance.web]`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `create_before_destroy` αλλάζει τη σειρά της αντικατάστασης σε create και μετά destroy, με σύμβολο `+/-`. Το `prevent_destroy` θα έκανε κάθε αντικατάσταση σφάλμα. Το `ignore_changes` θα σταματούσε εντελώς την αλλαγή του image, που δεν είναι το ζητούμενο. Το `replace_triggered_by` προκαλεί αντικαταστάσεις και δεν επηρεάζει τη σειρά τους.

Ερώτηση 14.2

Ένα resource έχει `prevent_destroy = true`. Ποιες προτάσεις ισχύουν; Επίλεξε δύο.

a. Το `terraform destroy` αποτυγχάνει και δεν σβήνει κανένα resource.
b. Μια αλλαγή που απαιτεί αντικατάσταση εκτελείται κανονικά.
c. Αν σβήσεις το block του resource από τον κώδικα, το επόμενο apply το σβήνει.
d. Το resource προστατεύεται και από διαγραφή μέσω της κονσόλας του cloud.
e. Η τιμή μπορεί να έρθει από μεταβλητή, για παράδειγμα `var.protect`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Ένα plan που σβήνει το resource σταματά με `Instance cannot be destroyed` και ένα plan με σφάλμα δεν εκτελείται, οπότε δεν σβήνεται τίποτα. Χωρίς το block η ρύθμιση δεν υπάρχει πια και το resource σβήνεται ως ορφανό. Το b είναι λάθος, γιατί κάθε αντικατάσταση περιέχει destroy και βγάζει το ίδιο σφάλμα. Το d είναι λάθος, γιατί το Terraform δεν ελέγχει τι κάνεις έξω από αυτό. Το e είναι λάθος, γιατί το `lifecycle` δέχεται μόνο σταθερές τιμές και η μεταβλητή δίνει `Variables not allowed.`

Ερώτηση 14.3

Ένας autoscaler αλλάζει το `desired_capacity` ενός `aws_autoscaling_group` ανάλογα με τον φόρτο. Κάθε `terraform apply` το επαναφέρει στην τιμή του κώδικα. Τι πρέπει να κάνεις;

a. Να προσθέσεις `prevent_destroy = true`.
b. Να προσθέσεις `create_before_destroy = true`.
c. Να προσθέσεις `ignore_changes = [desired_capacity]` στο `lifecycle`.
d. Να τρέχεις `terraform apply -replace=aws_autoscaling_group.web`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `ignore_changes` λέει στο Terraform να χρησιμοποιεί την τιμή του κώδικα μόνο στη δημιουργία και μετά να μη συγκρίνει το attribute. Το `prevent_destroy` αφορά διαγραφές, όχι updates. Το `create_before_destroy` αλλάζει μόνο τη σειρά μιας αντικατάστασης. Το `-replace` θα ξανάφτιαχνε όλο το group με την τιμή του κώδικα, δηλαδή το ίδιο πρόβλημα με μεγαλύτερη ζημιά.

Ερώτηση 14.4

Γράφεις `replace_triggered_by = [var.app_version]` σε ένα `aws_instance`. Τι συμβαίνει;

a. Το instance αντικαθίσταται κάθε φορά που αλλάζει η `app_version`.
b. Το Terraform βγάζει σφάλμα, γιατί δέχεται μόνο αναφορές σε resources.
c. Η ρύθμιση αγνοείται σιωπηλά.
d. Το instance αντικαθίσταται σε κάθε apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `replace_triggered_by` δέχεται αναφορές σε managed resources ή σε attributes τους. Το Terraform απορρίπτει τη μεταβλητή με `Invalid reference in replace_triggered_by expression`. Ο σωστός τρόπος είναι ένα `terraform_data` με `input = var.app_version` και αναφορά σε αυτό. Το a περιγράφει το επιθυμητό αποτέλεσμα, που όμως θέλει αυτό το ενδιάμεσο resource. Τα c και d δεν συμβαίνουν, γιατί η configuration δεν περνά καν το `validate`.

Ερώτηση 14.5

Κατά το `terraform apply` ο provider φτιάχνει ένα instance, αλλά το provisioner του αποτυγχάνει και το apply τελειώνει με σφάλμα. Τι κάνει το επόμενο `terraform apply`, αν δεν αλλάξεις τίποτα;

a. Τίποτα, γιατί το instance υπάρχει ήδη στο state.
b. Φτιάχνει δεύτερο instance και κρατά και το πρώτο.
c. Αποτυγχάνει μέχρι να τρέξεις `terraform untaint`.
d. Αντικαθιστά το instance, γιατί είναι σημαδεμένο ως tainted.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Ένα αντικείμενο που φτιάχτηκε αλλά η δημιουργία του δεν ολοκληρώθηκε μένει στο state ως tainted. Το επόμενο plan γράφει `is tainted, so must be replaced` και το apply το αντικαθιστά. Το a ξεχνά το σημάδι. Το b είναι λάθος, γιατί η αντικατάσταση σβήνει το παλιό αντικείμενο. Το c είναι λάθος, γιατί το `untaint` δεν είναι υποχρεωτικό. Το χρησιμοποιείς μόνο αν ξέρεις ότι το αντικείμενο είναι σωστό και θέλεις να το κρατήσεις.

Ερώτηση 14.6

Σωστό ή λάθος; Αν ένα resource έχει `create_before_destroy = true`, τα resources από τα οποία εξαρτάται αντικαθίστανται κι αυτά με create πριν από destroy, ακόμη κι αν δεν έχουν δικό τους `lifecycle`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το Terraform περνά το `create_before_destroy` σε όλες τις εξαρτήσεις του resource και το γράφει στο state τους. Αλλιώς μια εξάρτηση θα έπρεπε να σβηστεί πριν από το παλιό resource που τη χρησιμοποιεί, ενώ το παλιό resource ζει μέχρι να φτιαχτεί το νέο. Στο plan οι εξαρτήσεις εμφανίζονται με `+/-`.

ΚΕΦΑΛΑΙΟ 15

Το πρώτο module

Άσκηση 15.1 · Τρίτη ιστοσελίδα

Ξεκίνα από το `web`. Πρόσθεσε στο `main.tf` κλήση με όνομα `shop`, τίτλο `Το κατάστημα`, φάκελο `public/shop` και χρώμα `green`. Τρέξε init, apply και `terraform state list`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

module "site" {
  source = "./modules/site"

  title      = "Το blog μου"
  output_dir = "public/blog"
}

module "docs" {
  source = "./modules/site"

  title      = "Τεκμηρίωση"
  output_dir = "public/docs"
  color      = "teal"
}

module "shop" {
  source = "./modules/site"

  title      = "Το κατάστημα"
  output_dir = "public/shop"
  color      = "green"
}

output "blog_index" {
  value = module.site.index_path
}

output "docs_index" {
  value = module.docs.index_path
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Plan: 6 to add, 0 to change, 0 to destroy.
…
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
…
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
module.docs.local_file.css
module.docs.local_file.index
module.shop.local_file.css
module.shop.local_file.index
module.site.local_file.css
module.site.local_file.index
```

Η νέα κλήση δεν χρειάστηκε καμία αλλαγή στο module. Χρειάστηκε όμως init, γιατί κάθε νέο block `module` πρέπει να γραφτεί στο `modules.json`. Χωρίς αυτό το plan σταματά με `Module not installed`. Οι έξι διευθύνσεις ξεχωρίζουν από το πρόθεμα της κλήσης, οπότε τα ίδια ονόματα resources δεν συγκρούονται.

Άσκηση 15.2 · Ένα output ακόμη

Ξεκίνα από το `web`. Πρόσθεσε στο module output `css_path` με το path του stylesheet και `description`. Πρόσθεσε στο root output `docs_css` που δίνει το `css_path` της κλήσης `docs`. Κάνε apply και τύπωσε την τιμή με `terraform output -raw docs_css`.

ΛΥΣΗ · modules/site/outputs.tf

```
output "css_path" {
  value       = local_file.css.filename
  description = "Το path του stylesheet"
}
```

ΛΥΣΗ · main.tf

```
output "docs_css" {
  value = module.docs.css_path
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output -raw docs_css
```

ΕΞΟΔΟΣ

```
public/docs/style.css
```

Το output `css_path` μπαίνει στο τέλος του `modules/site/outputs.tf` και το `docs_css` στο τέλος του `main.tf`. Η τιμή χρειάζεται δύο outputs. Το `css_path` του module τη φέρνει ως το root. Το `docs_css` του root τη δίνει στο `terraform output`, που διαβάζει μόνο outputs του root module. Ένα `module.docs.local_file.css.filename` στο root θα έδινε `Unsupported attribute`, γιατί τα resources του child δεν φαίνονται έξω από αυτό.

Άσκηση 15.3 · Η πόρτα του module

Ξεκίνα από το `web`. Γράψε τα δύο αρχεία που ακολουθούν. Το root έχει τη μεταβλητή `owner` και το module θέλει να τη χρησιμοποιήσει στο `footer.html`.

owner.tf

```
variable "owner" {
  type    = string
  default = "Μαρία"
}
```

modules/site/footer.tf

```
resource "local_file" "footer" {
  filename = "${var.output_dir}/footer.html"
  content  = "<footer>${var.owner}</footer>\n"
}
```

Τρέξε `terraform validate` και διάβασε το σφάλμα. Διόρθωσε τη configuration χωρίς να σβήσεις τη μεταβλητή του root, ώστε και οι δύο ιστοσελίδες να πάρουν footer με την τιμή της. Κάνε apply και δείξε το `public/blog/footer.html`.

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
╷
│ Error: Reference to undeclared input variable
│
│   on modules/site/footer.tf line 3, in resource "local_file" "footer":
│    3:   content  = "<footer>${var.owner}</footer>\n"
│
│ An input variable with the name "owner" has not been declared. This
│ variable can be declared with a variable "owner" {} block.
╵

╷
│ Error: Reference to undeclared input variable
│
│   on modules/site/footer.tf line 3, in resource "local_file" "footer":
│    3:   content  = "<footer>${var.owner}</footer>\n"
│
│ An input variable with the name "owner" has not been declared. This
│ variable can be declared with a variable "owner" {} block.
╵
```

ΛΥΣΗ · modules/site/variables.tf

```
variable "owner" {
  type        = string
  description = "Ο ιδιοκτήτης της ιστοσελίδας"
}
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

module "site" {
  source = "./modules/site"

  title      = "Το blog μου"
  output_dir = "public/blog"
  owner      = var.owner
}

module "docs" {
  source = "./modules/site"

  title      = "Τεκμηρίωση"
  output_dir = "public/docs"
  color      = "teal"
  owner      = var.owner
}

output "blog_index" {
  value = module.site.index_path
}
```

```
output "docs_index" {
  value = module.docs.index_path
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
cat public/blog/footer.html
```

ΕΞΟΔΟΣ

```
<footer>Μαρία</footer>
```

Το σφάλμα είναι `Reference to undeclared input variable` στο `modules/site/footer.tf`. Το `var.owner` μέσα στο module ψάχνει μεταβλητή του module, όχι του root. Η λύση έχει δύο μέρη. Το module δηλώνει δική του μεταβλητή `owner` στο τέλος του `modules/site/variables.tf` και κάθε κλήση της δίνει τιμή με `owner = var.owner`. Αν ξεχάσεις τη μία κλήση, το validate απαντά `Missing required argument`, γιατί η νέα μεταβλητή δεν έχει `default`.

Άσκηση 15.4 · Ιστοσελίδες από map

Σε νέο φάκελο κάλεσε το module με όνομα `site` και `for_each` πάνω σε map με κλειδιά `el` και `en` και τιμές `Καλώς ήρθες` και `Welcome`. Ο τίτλος κάθε ιστοσελίδας είναι η τιμή και ο φάκελος είναι το `public/` ακολουθούμενο από το κλειδί. Κάνε apply και δείξε τις διευθύνσεις με `terraform state list`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

module "site" {
  source = "../web/modules/site"
  for_each = {
    el = "Καλώς ήρθες"
    en = "Welcome"
  }

  title      = each.value
  output_dir = "public/${each.key}"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
module.site["el"].local_file.css
module.site["el"].local_file.index
module.site["en"].local_file.css
module.site["en"].local_file.index
```

ΛΥΣΗ · TERMINAL

```
terraform state show -no-color 'module.site["en"].local_file.index' | grep filename
```

ΕΞΟΔΟΣ

```
    filename             = "public/en/index.html"
```

Το κλειδί του map μπαίνει στη διεύθυνση της κλήσης, `module.site["en"]`, όχι του resource. Το `local_file.index` μέσα στο module δεν έχει δικό του `for_each`. Στο `state show` η διεύθυνση γράφεται σε μονά εισαγωγικά, για να μη φάει το shell τα διπλά εισαγωγικά και τις αγκύλες. Το `-no-color` του κεφαλαίου 3 κρατά καθαρή την έξοδο που περνά στο `grep`.

Άσκηση 15.5 · Ιστοσελίδα κατά παραγγελία

Σε νέο φάκελο γράψε μεταβλητή `with_preview` τύπου `bool` με default `false` και κλήση `preview` του module με `count`, τίτλο `Προεπισκόπηση` και φάκελο `public/preview`. Κάνε apply μία φορά χωρίς τιμές και μία φορά με `-var with_preview=true`. Δείξε τη λίστα του state μετά το δεύτερο apply.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "with_preview" {
  type    = bool
  default = false
}
```

```
module "preview" {
  source = "../web/modules/site"
  count  = var.with_preview ? 1 : 0

  title      = "Προεπισκόπηση"
  output_dir = "public/preview"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve -var with_preview=true
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
module.preview[0].local_file.css
module.preview[0].local_file.index
```

Το πρώτο apply δεν έφτιαξε τίποτα. Με `count = 0` το module δεν έχει instance, άρα ούτε resources. Το δεύτερο έφτιαξε το instance `[0]` με τα δύο αρχεία του. Αν κάνεις ξανά apply χωρίς το `-var`, το Terraform θα σβήσει και τα δύο, γιατί το `count` γυρίζει στο `0`.

Άσκηση 15.6 · Ποιος ζητά ποια έκδοση

Σε νέο φάκελο γράψε ένα module στο `modules/name` που ζητά τον `hashicorp/random` με constraint `>= 3.9.1`, έχει ένα `random_pet` και output `name`. Το root δηλώνει τον `random` με έκδοση `3.9.1` και καλεί το module με όνομα `pet`. Τρέξε init και `terraform providers`.

ΛΥΣΗ · modules/name/main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = ">= 3.9.1"
    }
  }
}

resource "random_pet" "this" {}

output "name" {
  value       = random_pet.this.id
  description = "Το τυχαίο όνομα"
}
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

module "pet" {
  source = "./modules/name"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
- Finding hashicorp/random versions matching ">= 3.9.1, 3.9.1"...
…
```

ΛΥΣΗ · TERMINAL

```
terraform providers
```

```
ΕΞΟΔΟΣ

Providers required by configuration:
.
├── provider[registry.terraform.io/hashicorp/random] 3.9.1
└── module.pet
    └── provider[registry.terraform.io/hashicorp/random] >= 3.9.1
```

Το δέντρο έχει το root με το `3.9.1` και το κλαδί `module.pet` με το `>= 3.9.1`. Η γραμμή `Finding` του init έγραψε τα δύο constraints μαζί και το init διάλεξε την 3.9.1, τη μόνη έκδοση που ικανοποιεί και τα δύο. Το module δεν έχει block `provider`. Παίρνει την προεπιλεγμένη ρύθμιση του `random` από το root. Αν το root καρφώσει έκδοση μικρότερη, για παράδειγμα `3.8.0`, το init αποτυγχάνει με το μήνυμα `no available releases match the given constraints`. Το μήνυμα προτείνει το `terraform providers`, για να βρεις ποιο module ζητά τι.

Ερωτήσεις πιστοποίησης

Ερώτηση 15.1

Ποια πρόταση περιγράφει σωστά το root module;

a. Το module που βρίσκεται στον φάκελο `modules` ενός project.
b. Τα αρχεία `.tf` του φακέλου όπου τρέχεις τις εντολές του Terraform.
c. Ένα module δημοσιευμένο στο Terraform Registry.
d. Το πρώτο block `module` του αρχείου `main.tf`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το root module είναι ο φάκελος εργασίας με όλα τα αρχεία `.tf` του. Κάθε configuration έχει ένα. Ο φάκελος `modules` είναι μόνο σύμβαση για τα child modules, οπότε το a είναι λάθος. Ένα module του Registry μπαίνει στη configuration ως child, άρα το c είναι λάθος. Ένα block `module` καλεί child module και δεν ορίζει το root, οπότε και το d είναι λάθος.

Ερώτηση 15.2

Ένα child module που καλείται με `module "network"` έχει output με όνομα `vpc_id`. Πώς το διαβάζεις στο root module;

a. `var.network.vpc_id`
b. `output.network.vpc_id`
c. `module.network.vpc_id`
d. `module.network.outputs.vpc_id`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Τα outputs ενός child module είναι attributes του `module.ΟΝΟΜΑ`, όπου ΟΝΟΜΑ είναι το label της κλήσης. Το `var.` αφορά μεταβλητές του ίδιου module. Ένα `output.` δεν υπάρχει ως αναφορά, όπως είδες στο κεφάλαιο 8. Το `module.network` δεν έχει attribute `outputs`, γιατί τα outputs είναι απευθείας attributes του.

Ερώτηση 15.3

Το root module έχει τη μεταβλητή `environment`. Ένα child module χρειάζεται την τιμή της. Τι κάνεις;

a. Δηλώνεις `variable "environment"` στο child και γράφεις `environment = var.environment` στο block `module`.
b. Τίποτα. Το child διαβάζει αυτόματα το `var.environment` του root.
c. Γράφεις `var.root.environment` μέσα στο child.
d. Μετακινείς τη μεταβλητή σε `locals` του root, που τα βλέπουν όλα τα modules.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Ένα child module δεν κληρονομεί τις μεταβλητές του parent. Μια τιμή περνά μόνο ως argument του block `module` σε μεταβλητή που το child έχει δηλώσει. Το b δίνει `Reference to undeclared input variable`. Η αναφορά `var.root` δεν υπάρχει, οπότε το c είναι λάθος. Τα locals είναι ορατά μόνο στο module όπου δηλώνονται, άρα και το d είναι λάθος.

Ερώτηση 15.4

Πρόσθεσες νέο block `module` με τοπικό `source` και το `terraform plan` σταματά με `Module not installed`. Ποιες εντολές λύνουν το πρόβλημα; Επίλεξε δύο.

a. `terraform init`
b. `terraform validate`
c. `terraform get`
d. `terraform providers`
e. `terraform apply -auto-approve`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το `terraform init` εγκαθιστά modules, providers και backend. Το `terraform get` εγκαθιστά μόνο modules. Και τα δύο γράφουν τη νέα κλήση στο `.terraform/modules/modules.json`. Το `validate` και το `apply` σταματούν με το ίδιο σφάλμα. Το `terraform providers` δείχνει τις απαιτήσεις για providers και δεν εγκαθιστά τίποτα.

Ερώτηση 15.5

Σωστό ή λάθος; Ένα module που θα το χρησιμοποιούν πολλές configurations πρέπει να περιέχει δικό του block `provider` με όλες τις ρυθμίσεις του provider.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ένα reusable module δηλώνει μόνο `required_providers`. Η ρύθμιση του provider έρχεται από το root module, αυτόματα ή με το meta-argument `providers`. Ένα module με δικό του block `provider` δεν δέχεται `count`, `for_each` και `depends_on` στην κλήση του. Αν σβηστεί η κλήση, χάνεται και η ρύθμιση που χρειάζεται το Terraform για να σβήσει τα resources του.

Ερώτηση 15.6

Το block `module "app"` έχει `for_each = toset(["a", "b"])` και το module περιέχει το `resource "local_file" "conf"`. Ποια είναι η διεύθυνση του resource για το κλειδί `b`;

a. `local_file.conf["b"]`
b. `module.app.local_file.conf["b"]`
c. `module["b"].app.local_file.conf`
d. `module.app["b"].local_file.conf`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `for_each` είναι στην κλήση, οπότε το κλειδί μπαίνει μετά το `module.app`. Το resource μέσα στο module δεν έχει δικό του `for_each`, άρα δεν παίρνει κλειδί. Το a ξεχνά το πρόθεμα του module. Το b βάζει το κλειδί στο resource. Το c βάζει το κλειδί σε λάθος θέση.

ΚΕΦΑΛΑΙΟ 16

Πηγές modules, εκδόσεις και Registry

Άσκηση 16.1 · version σε Git

Κάποιος συνάδελφος έγραψε την παρακάτω configuration. Τρέξε `terraform init` και διάβασε το σφάλμα. Διόρθωσέ την ώστε να φέρνει την έκδοση v1.1.0 του module και κάνε apply.

main.tf

```
module "greeting" {
  source  = "git::file:///home/you/tfbook/ch16/repos/greeting"
  version = "1.1.0"

  name = "Git"
}

output "message" {
  value = module.greeting.message
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
│ Error: Invalid registry module source address
…
│ Failed to parse module registry address: a module registry source address
│ must have either three or four slash-separated components.
│
│ Terraform assumed that you intended a module registry source address
│ because you also set the argument "version", which applies only to registry
│ modules.
…
```

ΛΥΣΗ · main.tf

```
module "greeting" {
  source = "git::file:///home/you/tfbook/ch16/repos/greeting?ref=v1.1.0"

  name = "Git"
}

output "message" {
  value = module.greeting.message
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
message = "Γεια σου, Git!"
```

Το `version` ισχύει μόνο για registries. Μόλις το βλέπει, το Terraform προσπαθεί να διαβάσει το `source` ως διεύθυνση registry με τρία ή τέσσερα μέρη χωρισμένα με κάθετο και αποτυγχάνει. Σε Git η έκδοση γράφεται ως `?ref=v1.1.0` στο τέλος του `source`. Πρόσεξε ότι το tag λέγεται `v1.1.0`. Το `ref` θέλει το όνομα του tag ακριβώς, μαζί με το `v`.

Άσκηση 16.2 · Templates με κατάληξη .tftpl

Στο κεφάλαιο 10 τα templates είχαν κατάληξη `.tftpl`. Φτιάξε φάκελο `pages` με δύο αρχεία. Το template `index.html.tftpl` περιέχει `<p>Υπεύθυνη ${owner}</p>`. Το στατικό `robots.txt` περιέχει `User-agent: *`. Κάλεσε το `hashicorp/dir/template` 1.0.2 ώστε να θεωρεί templates τα `.tftpl`, με `owner` την `Ελένη`. Γράψε ένα output `rendered` με το `content` του `index.html` από το `files_in_memory` χωρίς την αλλαγή γραμμής του τέλους. Γράψε και ένα output `static` με τα κλειδιά του `files_on_disk`.

ΛΥΣΗ · pages/index.html.tftpl

```
<p>Υπεύθυνη ${owner}</p>
```

ΛΥΣΗ · pages/robots.txt

```
User-agent: *
```

ΛΥΣΗ · main.tf

```
module "pages" {
  source  = "hashicorp/dir/template"
  version = "1.0.2"

  base_dir             = "${path.module}/pages"
  template_file_suffix = ".tftpl"
  template_vars = {
    owner = "Ελένη"
  }
}

output "rendered" {
  value = trimspace(module.pages.files_in_memory["index.html"].content)
}

output "static" {
  value = keys(module.pages.files_on_disk)
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
rendered = "<p>Υπεύθυνη Ελένη</p>"
static = [
  "robots.txt",
]
```

Το input `template_file_suffix` αλλάζει την κατάληξη που κάνει ένα αρχείο template. Το module την αφαιρεί από το κλειδί, οπότε το template λέγεται `index.html` και όχι `index.html.tftpl`. Χωρίς αυτό το input και τα δύο αρχεία θα ήταν στατικά και το `files_in_memory` θα ήταν άδειο. Η `trimspace` του κεφαλαίου 10 κόβει την αλλαγή γραμμής που έχει το αρχείο στο τέλος. Η configuration δεν χρειάζεται `required_providers`, γιατί ούτε το module ούτε το root module χρησιμοποιούν provider.

Άσκηση 16.3 · Μια έκδοση που δεν υπάρχει

Ένα README λέει να καλέσεις το `hashicorp/dir/template` με `version = "~> 2.0"` και `base_dir = path.module`. Τρέξε `terraform init`. Ποια είναι η νεότερη έκδοση σύμφωνα με το μήνυμα; Γράψε μετά ένα constraint που δέχεται μόνο patches της σειράς 1.0 και ξανατρέξε το init.

ΛΥΣΗ · main.tf

```
module "docs" {
  source  = "hashicorp/dir/template"
  version = "~> 2.0"

  base_dir = path.module
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
╷ Error: Unresolvable module version constraint
…
│ There is no available version of module
│ "registry.terraform.io/hashicorp/dir/template" (main.tf:1) which matches
│ the given version constraint. The newest available version is 1.0.2.
…
```

ΛΥΣΗ · main.tf

```
module "docs" {
  source  = "hashicorp/dir/template"
  version = "~> 1.0.0"

  base_dir = path.module
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
Initializing the backend...

Initializing modules...
Downloading registry.terraform.io/hashicorp/dir/template 1.0.2 for docs...
- docs in .terraform/modules/docs
…
```

Το `~> 2.0` σημαίνει από την 2.0 έως πριν από την 3.0. Το module δεν έχει καμία 2.x και το init σταματά με `Unresolvable module version constraint`. Το μήνυμα λέει ότι η νεότερη είναι η 1.0.2. Το `~> 1.0.0` σημαίνει

από την 1.0.0 έως πριν από την 1.1.0, όπως στους providers, οπότε το init διαλέγει την 1.0.2. Αν μια μέρα βγει 1.0.3, ένα init σε καθαρό φάκελο θα πάρει εκείνη. Το lock file δεν θα το εμποδίσει.

Άσκηση 16.4 · Νέο input σε παλιά έκδοση

Γράψε configuration που καλεί το module `greeting` στην έκδοση v1.0.0 με `name = "Terraform"` και `greeting = "Καλησπέρα"`. Τρέξε `terraform init`. Διόρθωσε το πρόβλημα χωρίς να σβήσεις το `greeting` και κάνε apply.

ΛΥΣΗ · main.tf

```
module "greeting" {
  source = "git::file:///home/you/tfbook/ch16/repos/greeting?ref=v1.0.0"

  name     = "Terraform"
  greeting = "Καλησπέρα"
}

output "message" {
  value = module.greeting.message
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
Initializing the backend...

Initializing modules...
Downloading git::file:///home/you/tfbook/ch16/repos/greeting?ref=v1.0.0 for greeting...
- greeting in .terraform/modules/greeting

╷
│ Error: Unsupported argument
│
│   on main.tf line 5, in module "greeting":
│    5:   greeting = "Καλησπέρα"
│
│ An argument named "greeting" is not expected here.
╵
```

ΛΥΣΗ · main.tf

```
module "greeting" {
  source = "git::file:///home/you/tfbook/ch16/repos/greeting?ref=v1.1.0"

  name     = "Terraform"
  greeting = "Καλησπέρα"
}

output "message" {
  value = module.greeting.message
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
message = "Καλησπέρα, Terraform!"
```

Το init κατεβάζει το module και αμέσως σταματά με `Unsupported argument`, γιατί ελέγχει τα arguments κάθε block `module` με τα inputs του module που κατέβασε. Η έκδοση v1.0.0 δεν έχει input `greeting`. Τα inputs που δέχεται ένα module εξαρτώνται από την έκδοση που κατέβηκε, όχι από την τελευταία που υπάρχει στο repository. Η διόρθωση είναι να ανεβάσεις το `ref` στο `v1.1.0`, την έκδοση που πρόσθεσε το input. Μετά τρέχεις ξανά `terraform init`.

Άσκηση 16.5 · Κλαδί αντί για tag

Κάλεσε το module `greeting` με `?ref=main` και `name = "Terraform"`, κάνε init και apply. Στο repository άλλαξε το `default` του `greeting` σε `Χαίρε` και κάνε commit χωρίς tag. Τρέξε `terraform init` και `terraform plan`. Μετά τρέξε `terraform init -upgrade` και ξανά `terraform plan`.

ΛΥΣΗ · main.tf

```
module "greeting" {
  source = "git::file:///home/you/tfbook/ch16/repos/greeting?ref=main"

  name = "Terraform"
}

output "message" {
  value = module.greeting.message
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · ~/tfbook/ch16/repos/greeting/variables.tf

```
variable "name" {
  type        = string
  description = "Ποιον χαιρετά το module."
}

variable "greeting" {
  type        = string
  default     = "Χαίρε"
  description = "Η λέξη του χαιρετισμού."
}
```

ΛΥΣΗ · TERMINAL

```
cd ~/tfbook/ch16/repos/greeting
git -c user.name=You -c user.email=you@example.com commit -q -am "Χαίρε"
cd -
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

ΛΥΣΗ · TERMINAL

```
terraform init -upgrade
```

ΕΞΟΔΟΣ

```
…
Upgrading modules...
Downloading git::file:///home/you/tfbook/ch16/repos/greeting?ref=main for greeting...
- greeting in .terraform/modules/greeting
…
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
Changes to Outputs:
  ~ message = "Γεια σου, Terraform!" -> "Χαίρε, Terraform!"
…
```

Το πρώτο init δεν κατέβασε τίποτα και το plan λέει `No changes`. Το module ήταν ήδη εγκατεστημένο από το `main` και το init δεν ελέγχει αν το κλαδί προχώρησε. Μόνο το `-upgrade` το κατέβασε ξανά και το plan βρήκε τον νέο χαιρετισμό. Ένας συνάδελφος που κάνει clone σήμερα θα έπαιρνε το νέο commit με το πρώτο init. Δύο μηχανήματα με την ίδια configuration τρέχουν διαφορετικό κώδικα. Με tag στο `ref` αυτό δεν συμβαίνει.

Άσκηση 16.6 · Module σε υποφάκελο

Πρόσθεσε στο repository `greeting` ένα εσωτερικό module στο `modules/shout/main.tf`. Δέχεται ένα input `text` και έχει output `loud` με το `upper(var.text)`. Κάνε commit και βάλε tag `v1.2.0`. Σε νέο φάκελο κάλεσέ το με `text = "γεια σου"`, χρησιμοποιώντας `//` και `?ref=v1.2.0`. Τι γράφει το init για το πού μπήκε το module;

ΛΥΣΗ · ~/tfbook/ch16/repos/greeting/modules/shout/main.tf

```
variable "text" {
  type = string
}

output "loud" {
  value = upper(var.text)
}
```

ΛΥΣΗ · TERMINAL

```
cd ~/tfbook/ch16/repos/greeting
git add .
git -c user.name=You -c user.email=you@example.com commit -q -m "Module shout"
git tag v1.2.0
cd -
```

ΛΥΣΗ · main.tf

```
module "shout" {
  source = "git::file:///home/you/tfbook/ch16/repos/greeting//modules/shout?ref=v1.2.0"

  text = "γεια σου"
}

output "loud" {
  value = module.shout.loud
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΕΞΟΔΟΣ

```
…
Downloading git::file:///home/you/tfbook/ch16/repos/greeting?ref=v1.2.0 for shout...
- shout in .terraform/modules/shout/modules/shout
…
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
loud = "ΓΕΙΑ ΣΟΥ"
```

Το `//` χωρίζει τη διεύθυνση του repository από τον υποφάκελο μέσα σε αυτό. Το `?ref` μπαίνει στο τέλος, μετά τον υποφάκελο. Η γραμμή `Downloading` γράφει μόνο τη διεύθυνση του repository, γιατί αυτό κατεβαίνει ολόκληρο στο `.terraform/modules/shout`. Η επόμενη γραμμή δείχνει ότι ως module χρησιμοποιείται ο υποφάκελος `modules/shout`. Το apply δίνει `loud = "ΓΕΙΑ ΣΟΥ"`. Αν είχες γράψει μία κάθετο, το Terraform θα θεωρούσε το `modules/shout` μέρος του path του repository και το clone θα αποτύγχανε.

Ερωτήσεις πιστοποίησης

Ερώτηση 16.1

Ποια από τις παρακάτω πηγές modules δέχεται το argument `version`;

a. Τοπικό path όπως `./modules/network`
b. Git repository όπως `git::https://example.com/network.git`
c. Terraform Registry όπως `acme/network/aws`
d. Αρχείο HTTP όπως `https://example.com/network.zip`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `version` ισχύει μόνο για modules από registry, δημόσιο ή private. Ένα τοπικό path δεν έχει εκδόσεις, είναι ό,τι υπάρχει στον φάκελο. Σε Git η έκδοση γράφεται με `?ref=` μέσα στο `source`. Ένα αρχείο HTTP είναι μία συγκεκριμένη έκδοση και η έκδοση φαίνεται στο URL. Αν βάλεις `version` σε οποιαδήποτε από αυτές, το init αποτυγχάνει με `Invalid registry module source address`.

Ερώτηση 16.2

Το module σου βρίσκεται στον υποφάκελο `modules/network` του φακέλου εργασίας. Ποια τιμή του `source` το φορτώνει;

a. `"modules/network"`
b. `"./modules/network"`
c. `"local::modules/network"`
d. `"file://modules/network"`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ένα τοπικό path ξεκινά με `./` ή `../`. Χωρίς πρόθεμα, όπως στο a, το Terraform το θεωρεί remote πηγή και σταματά με `Invalid module source address`. Δεν υπάρχει πρόθεμα `local::`, οπότε το c είναι λάθος. Το `file://` θέλει απόλυτο path. Στο `file://modules/network` το `modules` διαβάζεται ως όνομα μηχανήματος και η λήψη αποτυγχάνει, οπότε το d είναι λάθος.

Ερώτηση 16.3

Θέλεις το module από το repository `https://example.com/network.git` στο tag `v1.2.0`. Ποιο `source` είναι σωστό;

a. `"git::https://example.com/network.git"` μαζί με `version = "v1.2.0"`
b. `"git::https://example.com/network.git@v1.2.0"`
c. `"git::https://example.com/network.git//v1.2.0"`
d. `"git::https://example.com/network.git?ref=v1.2.0"`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Σε Git η έκδοση γράφεται με την παράμετρο `ref` στο τέλος της διεύθυνσης. Το `version` δεν ισχύει για Git και κάνει το init να αποτύχει, οπότε το a είναι λάθος. Το `@` δεν σημαίνει έκδοση σε αυτή τη θέση. Το `//` χωρίζει υποφάκελο μέσα στο repository, άρα το c θα έψαχνε φάκελο με όνομα `v1.2.0`.

Ερώτηση 16.4

Σωστό ή λάθος; Το `.terraform.lock.hcl` καταγράφει την έκδοση κάθε module του Registry που επέλεξε το init, ώστε όλη η ομάδα να παίρνει την ίδια.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το lock file καταγράφει μόνο providers. Η έκδοση ενός module που επέλεξε το init μένει στο `.terraform/modules/modules.json`, που δεν μπαίνει σε Git. Ένα init σε καθαρό φάκελο διαλέγει τη νεότερη έκδοση που επιτρέπει το `version`. Γι' αυτό η έκδοση ενός module καρφώνεται στη configuration.

Ερώτηση 16.5

Άλλαξες το `version` ενός module του Registry από `"1.0.1"` σε `"1.0.2"`. Το `terraform plan` σταματά με `Module version requirements have changed`. Τι πρέπει να τρέξεις;

a. `terraform init`
b. `terraform validate`
c. `terraform apply -auto-approve`
d. `terraform fmt`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Τα modules κατεβαίνουν μόνο από το init ή από το `terraform get`. Το `validate` και το `apply` χρειάζονται ήδη εγκατεστημένα modules και σταματούν με το ίδιο σφάλμα. Το `fmt` αλλάζει μόνο τη μορφή των αρχείων και δεν κατεβάζει τίποτα.

Ερώτηση 16.6

Ποια ισχύουν για ένα module που δημοσιεύεται στο δημόσιο Terraform Registry; Επίλεξε δύο.

a. Βρίσκεται σε δημόσιο repository στο GitHub με όνομα της μορφής `terraform-PROVIDER-NAME`.
b. Το Registry κρατά και το state όσων το χρησιμοποιούν.
c. Κάθε νέα έκδοση είναι ένα Git tag με semantic version, όπως `v1.3.0`.
d. Η διεύθυνσή του γράφεται `NAMESPACE/PROVIDER/NAME`, με τον provider στη μέση.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το δημόσιο Registry διαβάζει modules από δημόσια repositories του GitHub με όνομα `terraform-PROVIDER-NAME` και δημιουργεί μία έκδοση για κάθε tag με semantic version. Το Registry δεν κρατά state. Το state ανήκει στη configuration που χρησιμοποιεί το module και αποθηκεύεται στο backend της. Η διεύθυνση γράφεται `NAMESPACE/NAME/PROVIDER`, με τον provider στο τέλος, όπως το `hashicorp/dir/template`.

ΚΕΦΑΛΑΙΟ 17

Ο AWS provider

Άσκηση 17.1 · Το name που έγινε region

Ξεκίνα από αντίγραφο του φακέλου `first`. Πρόσθεσε στο τέλος του `identity.tf` ένα output `region_name` με τιμή `data.aws_region.current.name`. Τρέξε `terraform init` και `terraform plan`. Τι λέει το Terraform για το νέο output;

ΛΥΣΗ · identity.tf

```
data "aws_caller_identity" "current" {}

data "aws_region" "current" {}

output "account_id" {
  value = data.aws_caller_identity.current.account_id
}

output "caller_arn" {
  value = data.aws_caller_identity.current.arn
}

output "region" {
  value = data.aws_region.current.region
}

output "region_name" {
  value = data.aws_region.current.name
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  + region_name = "eu-central-1"
…
╷ Warning: Deprecated value used
…
│   on identity.tf line 18, in output "region_name":
…
│   The deprecation originates from data.aws_region.current.name
…
╵ name is deprecated. Use region instead.
…
```

Το plan δίνει τιμή στο `region_name` αλλά τυπώνει warning. Το `name` μένει στο schema για να μη σπάσουν παλιές configurations, όμως είναι deprecated και το μήνυμα δείχνει τη διόρθωση, το `.region`. Ένα warning δεν σταματά το plan, γι' αυτό περνά εύκολα απαρατήρητο σε μεγάλες εξόδους.

Άσκηση 17.2 · Ποιο region κερδίζει

Ξεκίνα από αντίγραφο του φακέλου `first`. Σβήσε το argument `region` από το block `provider` του `main.tf` και κράτα τα `default_tags`. Τρέξε init και μετά `AWS_REGION=eu-west-1 AWS_DEFAULT_REGION=eu-north-1 terraform plan`. Σε ποιο region θα δούλευε ο provider;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  default_tags {
    tags = {
      Project = "tfbook"
      Owner   = "platform"
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
AWS_REGION=eu-west-1 AWS_DEFAULT_REGION=eu-north-1 terraform plan
```

ΕΞΟΔΟΣ

```
…
  + region     = "eu-west-1"
…
```

Χωρίς argument `region` ο provider κοιτάζει τις μεταβλητές περιβάλλοντος και η `AWS_REGION` είναι πριν από την `AWS_DEFAULT_REGION`. Η γραμμή είναι το output `region` του `identity.tf`. Με το argument στη θέση του η ίδια εντολή δίνει `eu-central-1`. Η παγίδα είναι μια configuration χωρίς `region`, που φτιάχνει resources όπου τύχει να δείχνει το τερματικό του καθενός.

Άσκηση 17.3 · Τα στοιχεία μέσα στο bucket

Ξεκίνα από αντίγραφο του φακέλου `mixed`. Πρόσθεσε ένα `aws_s3_object` με όνομα `info` που ανεβάζει στο bucket το περιεχόμενο του `local_file.info` με key `bucket.txt`. Κάνε apply και δείξε τα resources του state.

ΛΥΣΗ · object.tf

```
resource "aws_s3_object" "info" {
  bucket  = aws_s3_bucket.site.id
  key     = "bucket.txt"
  content = local_file.info.content
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
…
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
aws_s3_bucket.site
aws_s3_object.info
local_file.info
random_pet.suffix
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Το `content = local_file.info.content` κάνει το object να περιμένει το αρχείο, που περιμένει το bucket, που περιμένει το `random_pet`. Η αλυσίδα περνά από τρεις providers σε ένα graph. Το `source = "bucket.txt"` θα ήταν λάθος. Δεν φτιάχνει dependency, οπότε το apply μπορεί να προσπαθήσει να ανεβάσει το αρχείο πριν υπάρξει και να σταματήσει με σφάλμα.

Άσκηση 17.4 · Tags μόνο στον AWS

Ξεκίνα από αντίγραφο του φακέλου `mixed` και κάνε apply. Μετά πρόσθεσε στο block `provider "aws"` τα `default_tags` με `Project = "tfbook"` και `Chapter = "17"`. Τρέξε `terraform plan`. Ποια resources θα αλλάξουν;

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

provider "aws" {
  region = "eu-central-1"

  default_tags {
    tags = {
      Project = "tfbook"
      Chapter = "17"
    }
  }
}

resource "random_pet" "suffix" {
  length = 2
}

resource "aws_s3_bucket" "site" {
  bucket = "tfbook-${random_pet.suffix.id}"
```

```
}

resource "local_file" "info" {
  filename = "bucket.txt"
  content  = <<-EOT
    name   = ${aws_s3_bucket.site.bucket}
    region = ${aws_s3_bucket.site.region}
    arn    = ${aws_s3_bucket.site.arn}
  EOT
}

output "bucket" {
  value = aws_s3_bucket.site.bucket
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.site will be updated in-place
…
Plan: 0 to add, 1 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Μόνο το bucket αλλάζει, με update in place. Τα `default_tags` είναι ρύθμιση του provider `aws` και αφορούν μόνο resources που φτιάχνει αυτός. Το `local_file` δεν αντικαθίσταται, γιατί διαβάζει όνομα, region και ARN, που δεν αλλάζουν με τα tags.

Άσκηση 17.5 · Το bucket που δεν άδειαζε

Ξεκίνα από αντίγραφο του φακέλου `mixed` και κάνε apply. Ανέβασε ένα αρχείο στο bucket με το `aws` CLI, για παράδειγμα με `aws s3 cp --no-progress main.tf "s3://$(terraform output -raw bucket)/main.tf"`. Δοκίμασε `terraform destroy`. Μετά κάνε ό,τι χρειάζεται ώστε το destroy να πετύχει χωρίς να σβήσεις το αρχείο με το χέρι.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
aws s3 cp --no-progress main.tf "s3://$(terraform output -raw bucket)/main.tf"
```

ΕΞΟΔΟΣ

```
upload: ./main.tf to s3://tfbook-smart-cobra/main.tf
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

ΕΞΟΔΟΣ

```
…
local_file.info: Destruction complete after 1s
…
╷ Error: deleting S3 Bucket (tfbook-smart-cobra): operation error S3: DeleteBucket,
https response error StatusCode: 409, RequestID:
UayKAHs0xai5vKbPupSA7wi8pF9O1BItogZctadsuaRigcu3Cr0O, HostID:
9Gjjt1m+cjU4OPvX9O9/8RuvnG41MRb/18Oux2o5H5MY7ISNTlXN+Dz9IG62/ILVxhAGI0qyPfg=, api error
BucketNotEmpty: The bucket you tried to delete is not empty
…
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "random_pet" "suffix" {
  length = 2
}

resource "aws_s3_bucket" "site" {
  bucket        = "tfbook-${random_pet.suffix.id}"
  force_destroy = true
}

resource "local_file" "info" {
  filename = "bucket.txt"
  content  = <<-EOT
    name   = ${aws_s3_bucket.site.bucket}
    region = ${aws_s3_bucket.site.region}
    arn    = ${aws_s3_bucket.site.arn}
```

```
  EOT
}

output "bucket" {
  value = aws_s3_bucket.site.bucket
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 1 added, 1 changed, 0 destroyed.
…
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

ΕΞΟΔΟΣ

```
…
Destroy complete! Resources: 3 destroyed.
```

Το πρώτο destroy σβήνει το `local_file` και σταματά στο bucket με `BucketNotEmpty`, γιατί το `main.tf` που ανέβασες δεν είναι στο state. Το `random_pet` μένει, αφού το bucket εξαρτάται από αυτό. Το `force_destroy = true` δεν αρκεί να γραφτεί στον κώδικα. Το destroy διαβάζει την τιμή από το state, οπότε χρειάζεται πρώτα apply, που ενημερώνει το bucket και ξαναφτιάχνει το `local_file`. Σε bucket με σημαντικά δεδομένα το `force_destroy` θέλει προσοχή, γιατί σβήνει και ό,τι δεν έφτιαξε το Terraform.

Άσκηση 17.6 · Τρίτο region

Ξεκίνα από αντίγραφο του φακέλου `regions`. Πρόσθεσε σε νέο αρχείο `stockholm.tf` μια ρύθμιση του `aws` με alias `stockholm` στο `eu-north-1` και ένα bucket `${local.prefix}-stockholm` εκεί. Κάνε apply και έλεγξε το region του νέου bucket με `aws s3api get-bucket-location`.

ΛΥΣΗ · stockholm.tf

```
provider "aws" {
  alias  = "stockholm"
  region = "eu-north-1"
}

resource "aws_s3_bucket" "stockholm" {
  provider = aws.stockholm
  bucket   = "${local.prefix}-stockholm"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
aws s3api get-bucket-location --bucket "tfbook-$ACCOUNT-stockholm" --output text
```

ΕΞΟΔΟΣ

```
eu-north-1
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Ένα block `provider` μπορεί να βρίσκεται σε οποιοδήποτε αρχείο του root module. Κάθε alias είναι μια ακόμη ρύθμιση του ίδιου provider, οπότε το init δεν κατεβάζει τίποτα καινούργιο. Αν ξεχάσεις το `provider = aws.stockholm`, το bucket φτιάχνεται σιωπηλά στη Φρανκφούρτη. Μόνο το `region` στο plan θα το μαρτυρήσει.

Άσκηση 17.7 · Ένα module, δύο regions

Ξεκίνα από αντίγραφο του φακέλου `regions`. Γράψε module στο `modules/pair` που δέχεται μεταβλητή `name` και φτιάχνει δύο buckets, το `${var.name}-primary` με τη ρύθμιση `aws.primary` και το `${var.name}-secondary` με τη ρύθμιση `aws.secondary`. Δώσε output `regions` με map από το όνομα κάθε bucket στο region του. Κάλεσέ το από νέο αρχείο `pair.tf` με όνομα `${local.prefix}-pair`, primary τη Φρανκφούρτη και secondary την Ιρλανδία. Βγάλε το map στο root με output `pair_regions`, κάνε apply και δείξε το με `terraform output`.

ΛΥΣΗ · modules/pair/main.tf

```
terraform {
  required_providers {
    aws = {
      source                = "hashicorp/aws"
      version               = ">= 6.67.0"
      configuration_aliases = [aws.primary, aws.secondary]
    }
  }
}

variable "name" {
  type = string
}

resource "aws_s3_bucket" "primary" {
  provider = aws.primary
  bucket   = "${var.name}-primary"
}
```

```
resource "aws_s3_bucket" "secondary" {
  provider = aws.secondary
  bucket   = "${var.name}-secondary"
}

output "regions" {
  value = {
    (aws_s3_bucket.primary.bucket)   = aws_s3_bucket.primary.region
    (aws_s3_bucket.secondary.bucket) = aws_s3_bucket.secondary.region
  }
}
```

ΛΥΣΗ · pair.tf

```
module "pair" {
  source = "./modules/pair"
  name   = "${local.prefix}-pair"

  providers = {
    aws.primary   = aws
    aws.secondary = aws.ireland
  }
}

output "pair_regions" {
  value = module.pair.regions
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output pair_regions
```

ΕΞΟΔΟΣ

```
{
  "tfbook-123456789012-pair-primary" = "eu-central-1"
  "tfbook-123456789012-pair-secondary" = "eu-west-1"
}
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Το `configuration_aliases` λέει ότι το module περιμένει τις ρυθμίσεις `aws.primary` και `aws.secondary` και δεν έχει κανένα block `provider`. Η κλήση τις αντιστοιχίζει στο `providers`, με την προεπιλεγμένη ρύθμιση του root για την πρώτη και την `aws.ireland` για τη δεύτερη. Οι παρενθέσεις γύρω από τα κλειδιά του map λένε στο Terraform να τα υπολογίσει ως εκφράσεις. Αν η κλήση ξεχάσει μία ρύθμιση, ήδη το `init` και

το `validate` σταματούν με `Missing required provider configuration`, γιατί ρύθμιση με alias δεν κληρονομείται ποτέ αυτόματα.

Ερωτήσεις πιστοποίησης

Ερώτηση 17.1

Μια ομάδα τρέχει Terraform από laptops και από ένα σύστημα CI. Ποιος τρόπος να δοθούν credentials στον AWS provider δεν συνιστάται;

a. Τα `access_key` και `secret_key` γραμμένα στο block `provider`.
b. Ένα profile του SSO που διαλέγεται με τη μεταβλητή `AWS_PROFILE`.
c. Οι μεταβλητές `AWS_ACCESS_KEY_ID` και `AWS_SECRET_ACCESS_KEY` από τα secrets του CI.
d. Ο IAM role του server του EC2 όπου τρέχει το CI.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Κλειδιά μέσα στα αρχεία `.tf` καταλήγουν στο version control και τα βλέπει όποιος έχει πρόσβαση στο repository. Το b δίνει προσωρινά credentials έξω από τον κώδικα. Το c τα κρατά στα secrets του CI και όχι στο repository. Το d δεν έχει καθόλου αποθηκευμένα κλειδιά, γιατί τα δίνει το instance profile.

Ερώτηση 17.2

Ο provider έχει `default_tags` με `Environment = "dev"` και `Team = "data"`. Ένα `aws_s3_bucket` έχει `tags = { Environment = "prod", Name = "logs" }`. Ποια tags θα έχει το bucket στο AWS;

a. `Environment = "dev"`, `Team = "data"` και `Name = "logs"`
b. `Environment = "prod"` και `Name = "logs"`
c. `Environment = "prod"`, `Team = "data"` και `Name = "logs"`
d. Το plan αποτυγχάνει επειδή το `Environment` ορίζεται δύο φορές.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `tags_all` είναι η ένωση των `default_tags` με τα `tags` του resource και σε κοινό κλειδί κερδίζει η τιμή του resource. Το a κρατά λάθος την τιμή του provider. Το b ξεχνά ότι τα `default_tags` προστίθενται. Το d είναι λάθος, γιατί η σύγκρουση κλειδιών είναι η κανονική περίπτωση της υπερίσχυσης και όχι σφάλμα.

Ερώτηση 17.3

Το root module έχει `provider "aws"` με `region = "eu-central-1"` και δεύτερο `provider "aws"` με `alias = "west"` και `region = "eu-west-1"`. Καλεί ένα module με `providers = { aws = aws.west }`. Το module έχει ένα `aws_s3_bucket` χωρίς argument `provider`. Σε ποιο region φτιάχνεται το bucket;

a. Στο `eu-central-1`, γιατί τα modules παίρνουν πάντα την προεπιλεγμένη ρύθμιση.
b. Στο `eu-west-1`, γιατί για το module η ρύθμιση `aws.west` είναι ο `aws` του.
c. Σε κανένα, γιατί ρύθμιση με alias δεν περνά σε module.
d. Και στα δύο regions, ένα bucket σε καθένα.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `providers` αντιστοιχίζει το τοπικό όνομα `aws` του child στη ρύθμιση `aws.west` του root, οπότε κάθε resource του module χωρίς `provider` την παίρνει. Η προεπιλεγμένη ισχύει μόνο όταν η κλήση δεν γράφει `providers`, άρα το a είναι λάθος. Το c αρνείται ακριβώς τον ρόλο του `providers`. Ένα resource φτιάχνει ένα αντικείμενο, οπότε το d δεν στέκει.

Ερώτηση 17.4

Μια εταιρεία έχει servers σε δικό της datacenter με VMware vSphere, εφαρμογές στο AWS και DNS σε άλλον πάροχο. Ποια δύο ισχύουν για τη διαχείρισή τους με Terraform; Επίλεξε δύο.

a. Μία configuration μπορεί να έχει resources και των τριών providers, με references ανάμεσά τους.
b. Το Terraform μεταφράζει αυτόματα ένα `aws_instance` σε virtual machine του vSphere.
c. Κάθε provider χρειάζεται υποχρεωτικά δικό του χωριστό state.
d. Το `init`, το `plan` και το `apply` δουλεύουν με τον ίδιο τρόπο για όλους τους providers.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και d

Ο Core χτίζει ένα graph από όλα τα resources, όποιος provider κι αν τα διαχειρίζεται. Ένα reference από το DNS προς έναν server δημιουργεί κανονική εξάρτηση. Το workflow είναι κοινό για όλες τις πλατφόρμες. Το b είναι λάθος, γιατί κάθε provider έχει δικούς του τύπους resources και καμία αυτόματη μετάφραση. Το c είναι λάθος, γιατί μία configuration με πολλούς providers έχει ένα state, όπως το `mixed` του κεφαλαίου.

Ερώτηση 17.5

Σωστό ή λάθος; Το block `provider "aws"` γράφει `region = "eu-central-1"`. Αν τρέξεις `AWS_REGION=us-east-1 terraform apply`, τα resources φτιάχνονται στο `us-east-1`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το argument `region` του block `provider` είναι πρώτο στη σειρά και κερδίζει τη μεταβλητή περιβάλλοντος. Η `AWS_REGION` μετρά μόνο όταν το block δεν γράφει `region`.

Ερώτηση 17.6

Το Terraform τρέχει σε server του EC2 που έχει IAM role. Δεν υπάρχουν credentials στο block `provider`, σε μεταβλητές περιβάλλοντος ή στα αρχεία του `~/.aws`. Τι κάνει ο AWS provider;

a. Σταματά αμέσως με `No valid credential sources found`.
b. Ζητά από τον χρήστη να γράψει access key στο τερματικό.
c. Παίρνει προσωρινά credentials του role από την υπηρεσία metadata του server.
d. Κάνει τις κλήσεις ανώνυμα, χωρίς υπογραφή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το instance profile είναι η τελευταία πηγή στη σειρά των credentials. Ο provider ρωτά την υπηρεσία metadata στο `169.254.169.254` και παίρνει προσωρινά credentials του role, που ανανεώνονται μόνα τους. Το σφάλμα του a βγαίνει μόνο αν αποτύχει και αυτή η πηγή. Ο provider δεν ρωτά ποτέ για credentials στο τερματικό και δεν στέλνει ανυπόγραφες κλήσεις στο API.

ΚΕΦΑΛΑΙΟ 18

Δίκτυο και ένας web server

Άσκηση 18.1 · Χωρίς depends_on

Ξεκίνα από το `web`. Σβήσε το `depends_on` από τον server. Πριν τρέξεις οτιδήποτε, γράψε σε χαρτί ποια βέλη θα ξεκινούν από το `aws_instance.web` στο `terraform graph`. Έλεγξε κρατώντας μόνο αυτές τις γραμμές.

ΛΥΣΗ · server.tf

```
data "aws_ami" "al2023" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["al2023-ami-2023.*-kernel-6.1-x86_64"]
  }
}

resource "aws_instance" "web" {
  ami                    = data.aws_ami.al2023.id
  instance_type          = "t3.micro"
  subnet_id              = aws_subnet.public.id
  vpc_security_group_ids = [aws_security_group.web.id]

  user_data = templatefile("${path.module}/user_data.sh.tftpl", {
    title = "Γεια σου από το Terraform"
  })
  user_data_replace_on_change = true

  tags = {
    Name = "tfbook-web"
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform graph
```

ΕΞΟΔΟΣ

```
…
  "aws_instance.web" -> "data.aws_ami.al2023";
  "aws_instance.web" -> "aws_security_group.web";
  "aws_instance.web" -> "aws_subnet.public";
…
```

Τα βέλη είναι τρία, ένα για κάθε αναφορά του server. Το βέλος προς το association έφυγε μαζί με το `depends_on`. Το βέλος προς το `aws_subnet.public` ξαναφάνηκε, γιατί δεν υπάρχει πια άλλος δρόμος από τον server προς το subnet. Η παγίδα είναι να νομίσεις ότι το βέλος έλειπε επειδή ο server δεν εξαρτιόταν από το subnet. Εξαρτιόταν, μέσω του association. Τώρα ο server δεν περιμένει το association, οπότε στο apply μπορεί να ξεκινήσει πριν υπάρξει το route προς το internet.

Άσκηση 18.2 · Ζώνες και εικόνα

Ξεκίνα από το `web`. Πρόσθεσε το αρχείο `lookup.tf` με δύο outputs. Το `zones` δίνει όλα τα ονόματα των availability zones και το `ami_name` το όνομα του image που βρήκε το `data.aws_ami.al2023`. Τρέξε μόνο plan και βρες τις τιμές στο `Changes to Outputs`.

ΛΥΣΗ · lookup.tf

```
output "zones" {
  value = data.aws_availability_zones.available.names
}

output "ami_name" {
  value = data.aws_ami.al2023.name
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  + ami_name    = "al2023-ami-2023.12.20260727.0-kernel-6.1-x86_64"
…
  + zones       = [
      + "eu-central-1a",
      + "eu-central-1b",
      + "eu-central-1c",
…
```

Τα data sources διαβάζονται στο plan, οπότε τα δύο outputs έχουν τιμή πριν από κάθε apply. Τα outputs του server γράφουν `(known after apply)`, γιατί εξαρτώνται από resource που δεν υπάρχει ακόμη. Το όνομα του image επιβεβαιώνει ότι το φίλτρο βρήκε την έκδοση με kernel 6.1 για `x86_64`. Η παγίδα είναι να γράψεις `aws_ami.al2023.name` χωρίς το `data.`. Το Terraform θα το διάβαζε ως managed resource τύπου `aws_ami` που δεν έχει δηλωθεί.

Άσκηση 18.3 · Χωρίς δημόσια IP

Ξεκίνα από το `web`. Άλλαξε το `map_public_ip_on_launch` του subnet σε `false`. Κάνε apply και τύπωσε το output `public_ip`.

ΛΥΣΗ · network.tf

```
data "aws_availability_zones" "available" {
  state = "available"
}

resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true

  tags = {
    Name = "tfbook-web"
  }
}

resource "aws_subnet" "public" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = data.aws_availability_zones.available.names[0]
  map_public_ip_on_launch = false

  tags = {
    Name = "tfbook-web-public"
  }
}

resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id
}

resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main.id
  }
}

resource "aws_route_table_association" "public" {
  subnet_id      = aws_subnet.public.id
  route_table_id = aws_route_table.public.id
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output public_ip
```

ΕΞΟΔΟΣ

```
""
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Το output είναι κενό string. Ο server ξεκίνησε σε subnet με route προς το internet gateway, αλλά χωρίς δημόσια IP. Από το internet δεν φτάνει κανείς σε αυτόν και ο ίδιος δεν βγαίνει έξω, γιατί το gateway εξυπηρετεί μόνο servers με δημόσια διεύθυνση. Το subnet είναι δημόσιο, ο server όχι. Η παγίδα είναι να νομίσεις ότι το route αρκεί. Χρειάζονται και τα δύο. Αν θέλεις δημόσια IP μόνο σε συγκεκριμένους servers, άφησε το subnet με `false` και γράψε `associate_public_ip_address = true` στο `aws_instance`.

Άσκηση 18.4 · HTTPS και μεγαλύτερος server

Ξεκίνα από το `web` και κάνε apply. Μετά κάνε δύο αλλαγές. Πρόσθεσε στο αρχείο `https.tf` κανόνα εισόδου με όνομα `https` για τη θύρα 443 από παντού. Άλλαξε το `instance_type` σε `t3.small`. Πριν τρέξεις plan, γράψε σε χαρτί ποιο σύμβολο περιμένεις για κάθε resource και τι θα λέει η περίληψη.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · https.tf

```
resource "aws_vpc_security_group_ingress_rule" "https" {
  security_group_id = aws_security_group.web.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "tcp"
  from_port         = 443
  to_port           = 443
}
```

ΛΥΣΗ · server.tf

```
data "aws_ami" "al2023" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["al2023-ami-2023.*-kernel-6.1-x86_64"]
  }
}

resource "aws_instance" "web" {
  ami                    = data.aws_ami.al2023.id
  instance_type          = "t3.small"
  subnet_id              = aws_subnet.public.id
  vpc_security_group_ids = [aws_security_group.web.id]
```

```
  user_data = templatefile("${path.module}/user_data.sh.tftpl", {
    title = "Γεια σου από το Terraform"
  })
  user_data_replace_on_change = true

  tags = {
    Name = "tfbook-web"
  }

  depends_on = [aws_route_table_association.public]
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_instance.web will be updated in-place
…
      ~ instance_type                        = "t3.micro" -> "t3.small"
…
  # aws_vpc_security_group_ingress_rule.https will be created
…
Plan: 1 to add, 1 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Ο νέος κανόνας είναι νέο resource με `+`. Το security group δεν εμφανίζεται καθόλου στο plan, γιατί κάθε κανόνας έχει δική του διεύθυνση. Το `instance_type` αλλάζει επί τόπου με `~`. Ο provider σταματά τον server, αλλάζει το μέγεθος και τον ξαναξεκινά. Η παγίδα είναι να υποθέσεις ότι αλλαγή στο μέγεθος σημαίνει νέο server. Υπάρχει όμως κόστος. Αν δεις ολόκληρο το plan, η `public_ip` γράφει `(known after apply)`. Μια δημόσια IP που δεν είναι Elastic IP αλλάζει σε κάθε σταμάτημα και ξεκίνημα.

Άσκηση 18.5 · user_data χωρίς αντικατάσταση

Ξεκίνα από το `web` και κάνε apply. Άλλαξε μετά το `user_data_replace_on_change` σε `false` και πρόσθεσε στο template τη γραμμή `<p>Τρίτη έκδοση</p>` κάτω από τη δεύτερη. Τρέξε plan. Τι σύμβολο έχει ο server αυτή τη φορά;

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · server.tf

```
data "aws_ami" "al2023" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["al2023-ami-2023.*-kernel-6.1-x86_64"]
  }
}

resource "aws_instance" "web" {
  ami                    = data.aws_ami.al2023.id
  instance_type          = "t3.micro"
  subnet_id              = aws_subnet.public.id
  vpc_security_group_ids = [aws_security_group.web.id]

  user_data = templatefile("${path.module}/user_data.sh.tftpl", {
    title = "Γεια σου από το Terraform"
  })
  user_data_replace_on_change = false

  tags = {
    Name = "tfbook-web"
  }

  depends_on = [aws_route_table_association.public]
}
```

ΛΥΣΗ · user_data.sh.tftpl

```
#!/bin/bash
dnf install -y httpd
cat > /var/www/html/index.html <<'HTML'
<h1>${title}</h1>
<p>Δεύτερη έκδοση</p>
<p>Τρίτη έκδοση</p>
HTML
systemctl enable --now httpd
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_instance.web will be updated in-place
…
      ~ user_data                             = <<-EOT
…
          + <p>Τρίτη έκδοση</p>
…
      ~ user_data_replace_on_change           = true -> false
…
Plan: 0 to add, 1 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Ο server έχει ~, δηλαδή update in place. Η περίληψη λέει `1 to change`. Δεν υπάρχει `# forces replacement` δίπλα στο `user_data`. Στο πραγματικό AWS ο provider θα σταματούσε τον server, θα άλλαζε το user_data και θα τον ξεκινούσε. Το cloud-init δεν τρέχει όμως ξανά το script, οπότε η σελίδα θα έμενε στη δεύτερη έκδοση ενώ ο κώδικας λέει τρίτη. Αυτή είναι η παγίδα. Το plan δείχνει αλλαγή, αλλά η αλλαγή δεν φτάνει στη σελίδα. Στο Moto 5.2.3 το apply αυτής της αλλαγής αποτυγχάνει με σφάλμα του εξομοιωτή για το base64, γι' αυτό η λύση σταματά στο plan.

Άσκηση 18.6 · Η ζώνη στη σελίδα

Ξεκίνα από το `stack`. Η σελίδα πρέπει να γράφει και την availability zone του server, σε μια γραμμή `<p>Zone ...</p>` κάτω από τον τίτλο. Πρόσθεσε στο module output `availability_zone`, δώσε το στο template ως μεταβλητή `zone` και τρέξε plan.

ΛΥΣΗ · modules/network/outputs.tf

```
output "availability_zone" {
  value       = aws_subnet.public.availability_zone
  description = "Η availability zone του δημόσιου subnet"
}
```

ΛΥΣΗ · user_data.sh.tftpl

```
#!/bin/bash
dnf install -y httpd
cat > /var/www/html/index.html <<'HTML'
<h1>${title}</h1>
<p>Zone ${zone}</p>
HTML
systemctl enable --now httpd
```

ΛΥΣΗ · server.tf

```
data "aws_ami" "al2023" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["al2023-ami-2023.*-kernel-6.1-x86_64"]
  }
}

resource "aws_instance" "web" {
  ami                    = data.aws_ami.al2023.id
  instance_type          = "t3.micro"
  subnet_id              = module.network.subnet_id
  vpc_security_group_ids = [aws_security_group.web.id]

  user_data = templatefile("${path.module}/user_data.sh.tftpl", {
```

```
    title = "Γεια σου από το module"
    zone  = module.network.availability_zone
  })
  user_data_replace_on_change = true

  tags = {
    Name = "tfbook-stack-web"
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
            <p>Zone eu-central-1a</p>
…
Plan: 9 to add, 0 to change, 0 to destroy.
…
```

Το root δεν μπορεί να διαβάσει το `aws_subnet.public` του module, οπότε το module πρέπει να δώσει την τιμή με output. Η zone του subnet είναι γνωστή από το plan, γιατί προέρχεται από το data source. Γι' αυτό το user_data φαίνεται ολόκληρο, με τη zone στη θέση της. Η `templatefile` απαιτεί κάθε μεταβλητή που χρησιμοποιεί το template. Αν ξεχάσεις το `zone` στο object, το plan σταματά με `Invalid function argument` και το μήνυμα `vars map does not contain key "zone"`.

Άσκηση 18.7 · Δύο δίκτυα από ένα module

Ξεκίνα από το `stack`. Πρόσθεσε στο `main.tf` δεύτερη κλήση του module με όνομα κλήσης `test`, όνομα δικτύου `tfbook-test`, VPC `10.1.0.0/16` και subnet `10.1.1.0/24`. Τρέξε plan και κράτα μόνο τα resources της νέας κλήσης και την περίληψη.

ΛΥΣΗ · main.tf

```
module "test" {
  source = "./modules/network"

  name        = "tfbook-test"
  vpc_cidr    = "10.1.0.0/16"
  subnet_cidr = "10.1.1.0/24"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # module.test.aws_internet_gateway.main will be created
…
  # module.test.aws_route_table.public will be created
…
  # module.test.aws_route_table_association.public will be created
…
  # module.test.aws_subnet.public will be created
…
  # module.test.aws_vpc.main will be created
…
Plan: 14 to add, 0 to change, 0 to destroy.
…
```

Η δεύτερη κλήση δίνει πέντε νέα resources με πρόθεμα `module.test.`, το ίδιο δίκτυο με άλλες διευθύνσεις. Η περίληψη μετρά και τα εννέα του stack, γιατί ο φάκελος δεν έχει state. Σε φάκελο που έχει ήδη κάνει init, μια νέα κλήση module θέλει ξανά `terraform init` (κεφάλαιο 15). Χωρίς αυτό το plan σταματά με `Module not installed`. Δύο VPCs μπορούν να έχουν και το ίδιο CIDR block, αλλά τότε δεν συνδέονται ποτέ μεταξύ τους. Διαφορετικά blocks κρατούν ανοιχτή αυτή τη δυνατότητα.

Ερωτήσεις πιστοποίησης

Ερώτηση 18.1

Μια configuration έχει ένα `aws_vpc.main`, ένα `aws_subnet.public` και ένα `aws_security_group.web`. Το subnet και το security group αναφέρουν το `aws_vpc.main.id`. Ένα `aws_instance.web` αναφέρει το `aws_subnet.public.id` και το `aws_security_group.web.id`. Ποια πρόταση περιγράφει σωστά τη σειρά του apply;

a. Το Terraform φτιάχνει τα resources με τη σειρά που εμφανίζονται στα αρχεία.
b. Πρώτα το VPC. Μετά το subnet και το security group, πιθανόν ταυτόχρονα. Τελευταίο ο server.
c. Ο server και το security group φτιάχνονται ταυτόχρονα, γιατί το subnet δεν αναφέρει το security group.
d. Η σειρά δεν είναι εγγυημένη χωρίς `depends_on` στον server.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Οι αναφορές είναι implicit dependencies. Το subnet και το security group χρειάζονται το VPC, ο server χρειάζεται και τα δύο. Ανάμεσα στο subnet και στο security group δεν υπάρχει εξάρτηση, οπότε το Terraform τα φτιάχνει παράλληλα. Η σειρά στα αρχεία δεν παίζει ρόλο, άρα το a είναι λάθος. Ο server αναφέρει το security group, οπότε δεν ξεκινά μαζί του και το c είναι λάθος. Το `depends_on` χρειάζεται μόνο για εξαρτήσεις χωρίς αναφορά, άρα το d είναι λάθος.

Ερώτηση 18.2

Ένα `data "aws_ami"` έχει `owners = ["amazon"]` και ένα `filter` στο όνομα με `*`. Δεν έχει `most_recent`. Το φίλτρο ταιριάζει σε τρία images. Τι κάνει το `terraform plan`;

a. Διαλέγει το πρώτο image με αλφαβητική σειρά.
b. Διαλέγει το νεότερο image.
c. Σταματά με σφάλμα που ζητά πιο συγκεκριμένο φίλτρο ή `most_recent = true`.
d. Επιστρέφει λίστα με τα τρία IDs στο attribute `id`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `aws_ami` επιστρέφει ένα μόνο image. Όταν βρει πολλά χωρίς `most_recent`, το plan σταματά με `Your query returned more than one result`. Δεν υπάρχει σιωπηλή επιλογή, οπότε τα a και b είναι λάθος. Το b θα ήταν σωστό με `most_recent = true`. Το `id` είναι ένα string, όχι λίστα, άρα το d είναι λάθος.

Ερώτηση 18.3

Ένα child module με όνομα κλήσης `network` περιέχει το resource `aws_subnet.public` και δηλώνει `output "subnet_id"` με τιμή `aws_subnet.public.id`. Ποια έκφραση δίνει το ID του subnet στο root module;

a. `module.network.aws_subnet.public.id`
b. `module.network.subnet_id`
c. `output.network.subnet_id`
d. `var.network.subnet_id`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το root βλέπει ένα child module ως object με ένα attribute για κάθε output του. Η μορφή είναι `module.ONOMA.OUTPUT`. Τα resources του child δεν βγαίνουν έξω, οπότε το a σταματά με `Unsupported attribute`. Δεν υπάρχει πρόθεμα `output.` για αναφορές, άρα το c είναι λάθος. Το `var.` αφορά τις μεταβλητές του τρέχοντος module, όχι τα outputs άλλου, άρα το d είναι λάθος.

Ερώτηση 18.4

Ένα `aws_instance` έχει `user_data = templatefile(...)` και `user_data_replace_on_change = true`. Αλλάζεις μόνο το αρχείο του template. Τι δείχνει το επόμενο plan για τον server;

a. Τίποτα, γιατί κανένα αρχείο `.tf` δεν άλλαξε.
b. Αντικατάσταση με `-/+` και `# forces replacement` δίπλα στο `user_data`.
c. Update in place με `~` στον ίδιο server.
d. Σφάλμα, γιατί το user_data δεν αλλάζει μετά τη δημιουργία.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η `templatefile` διαβάζει το template σε κάθε plan, οπότε αλλαγή στο αρχείο είναι αλλαγή στο `user_data`. Το a είναι λάθος. Με `user_data_replace_on_change = true` ο provider ζητά νέο server. Το c θα ίσχυε με `false` ή χωρίς το argument, όπου ο provider σταματά τον server, αλλάζει το user_data και τον ξαναξεκινά. Η αλλαγή επιτρέπεται και στις δύο περιπτώσεις, άρα το d είναι λάθος.

Ερώτηση 18.5

Ποιες δύο προτάσεις για το `depends_on` είναι σωστές; Επίλεξε δύο.

a. Χρειάζεται όταν ένα resource χρειάζεται ένα άλλο χωρίς να το αναφέρει σε καμία έκφραση.
b. Μπορεί να αναφέρει ολόκληρο module, οπότε το resource περιμένει όλα τα resources του module.
c. Πρέπει να γράφεται για κάθε αναφορά, αλλιώς η σειρά δεν είναι εγγυημένη.
d. Κάνει το Terraform να δουλεύει ένα resource τη φορά.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b

Το `depends_on` δηλώνει εξαρτήσεις που το Terraform δεν βλέπει, όπως ο server που χρειάζεται το route του association για το internet. Δέχεται και module, με συνολική εξάρτηση από όλα τα resources του. Οι αναφορές δημιουργούν μόνες τους εξαρτήσεις, οπότε το c είναι λάθος. Το `depends_on` προσθέτει βέλη στο graph. Δεν αλλάζει το parallelism, που ορίζεται με `-parallelism`, άρα το d είναι λάθος.

Ερώτηση 18.6

Σωστό ή λάθος; Το plan μιας νέας configuration δείχνει τη δημόσια IP που θα πάρει ένα `aws_instance`, ώστε να τη βάλεις σε κανόνα firewall πριν από το apply.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Τη δημόσια IP τη δίνει το AWS όταν ξεκινήσει ο server. Στο plan το `public_ip` γράφει `(known after apply)` και το ίδιο κάθε output που την αναφέρει. Μια τιμή που υπάρχει μόνο μετά το apply δεν είναι διαθέσιμη πριν από αυτό. Αν χρειάζεσαι σταθερή διεύθυνση από πριν, τη ζητάς ως Elastic IP, που είναι χωριστό resource.

ΚΕΦΑΛΑΙΟ 19

Backends, remote state και locking

Άσκηση 19.1 · Ένα δεύτερο state στο ίδιο bucket

Γράψε configuration με ένα `random_integer` με όνομα `port`, από 8000 έως 8999. Πρόσθεσε ένα output `port` με την τιμή του. Κράτα το state στο bucket `tfbook-state` με key `ex1/terraform.tfstate` και lock με `use_lockfile`. Κάνε init και apply. Μετά δείξε τι υπάρχει κάτω από το `ex1/` με `aws s3 ls s3://tfbook-state/ex1/`. Η ώρα θα διαφέρει.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }

  backend "s3" {
    bucket       = "tfbook-state"
    key          = "ex1/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }
}

resource "random_integer" "port" {
  min = 8000
  max = 8999
}

output "port" {
  value = random_integer.port.result
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
aws s3 ls s3://tfbook-state/ex1/
```

ΕΞΟΔΟΣ

```
2026-10-08 05:30:00        770 terraform.tfstate
```

Το backend μπαίνει στο ίδιο block `terraform` με το `required_providers`. Δεν χρειάστηκε `-migrate-state`, γιατί δεν υπήρχε τοπικό state. Το `.tflock` δεν φαίνεται, γιατί το apply το έσβησε όταν τελείωσε. Αν είχες αντιγράψει το key του `base`, το apply θα έγραφε πάνω στο state του.

Άσκηση 19.2 · Partial configuration μόνο με σημαίες

Ξεκίνα σε νέο φάκελο με το παρακάτω `backend.tf`, χωρίς αρχείο `.hcl`. Δώσε το bucket `tfbook-state`, το key `ex2/terraform.tfstate` και το region `eu-central-1` στο `terraform init` με τρεις σημαίες `-backend-config`. Πρόσθεσε ένα `terraform_data` με όνομα `note` και `input = "partial"` και κάνε apply. Τέλος δείξε τι κατέγραψε το init με `jq '.backend.config | {bucket, key, region}' .terraform/terraform.tfstate`.

backend.tf

```
terraform {
  backend "s3" {
    use_lockfile = true
  }
}
```

ΛΥΣΗ · main.tf

```
resource "terraform_data" "note" {
  input = "partial"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init \
  -backend-config="bucket=tfbook-state" \
  -backend-config="key=ex2/terraform.tfstate" \
  -backend-config="region=eu-central-1"
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
jq '.backend.config | {bucket, key, region}' .terraform/terraform.tfstate
```

ΕΞΟΔΟΣ

```
{
  "bucket": "tfbook-state",
  "key": "ex2/terraform.tfstate",
  "region": "eu-central-1"
}
```

Κάθε `-backend-config` με `key=value` δίνει ένα argument. Το `backend.tf` λέει μόνο ότι το backend είναι `s3` με lock. Τα υπόλοιπα ήρθαν από τη γραμμή εντολών και γράφτηκαν σε απλό κείμενο στο `.terraform/terraform.tfstate`. Γι' αυτό τα credentials δεν δίνονται ποτέ έτσι.

Άσκηση 19.3 · Πίσω στο local

Γράψε configuration με ένα `random_pet` με όνομα `name` και S3 backend με key `ex3/terraform.tfstate`. Κάνε init και apply. Μετά σβήσε το block `backend` και φέρε το state πίσω σε τοπικό αρχείο, χωρίς να χαθεί το resource. Δείξε τα αρχεία με `ls` και τα resources του state με `terraform state list`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }

  backend "s3" {
    bucket       = "tfbook-state"
    key          = "ex3/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform init -migrate-state
```

ΛΥΣΗ · TERMINAL

```
ls
terraform state list
```

ΕΞΟΔΟΣ

```
main.tf
terraform.tfstate
random_pet.name
```

Χωρίς block `backend` η configuration γυρνά στο local backend. Το `init -migrate-state` ρώτησε αν θα αντιγράψει το state από το S3 και το `yes` το αντέγραψε στο `terraform.tfstate`. Με `-reconfigure` θα ξεκινούσε με άδειο τοπικό state και το επόμενο plan θα ήθελε να φτιάξει ξανά το `random_pet`. Το αντικείμενο στο S3 έμεινε στη θέση του.

Άσκηση 19.4 · Περίμενε τη σειρά σου

Γράψε configuration με ένα `time_sleep` που περιμένει 15 δευτερόλεπτα κατά τη δημιουργία, με S3 backend, key `ex4/terraform.tfstate` και lock. Κάνε init. Ξεκίνα apply στο παρασκήνιο με την έξοδο στο `apply.log`, περίμενε τρία δευτερόλεπτα και τρέξε plan που περιμένει το lock έως ένα λεπτό. Στο τέλος περίμενε με `wait` να τελειώσει το apply.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    time = {
      source  = "hashicorp/time"
      version = "0.14.2"
    }
  }

  backend "s3" {
    bucket       = "tfbook-state"
    key          = "ex4/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }
}

resource "time_sleep" "wait" {
  create_duration = "15s"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve > apply.log 2>&1 &
sleep 3
terraform plan -lock-timeout=60s
wait
```

ΕΞΟΔΟΣ

```
Acquiring state lock. This may take a few moments...
time_sleep.wait: Refreshing state... [id=2026-10-08T02:30:22Z]
…
No changes. Your infrastructure matches the configuration.
…
```

Το plan βρήκε το lock του apply και περίμενε αντί να αποτύχει. Όταν το apply τελείωσε, πήρε το lock, διάβασε το state με το νέο `time_sleep` και βρήκε `No changes`. Χωρίς `-lock-timeout` θα σταματούσε αμέσως. Ένα όριο ίσο με τη διάρκεια του apply μπορεί να μη φτάσει, γιατί οι προσπάθειες αραιώνουν.

Άσκηση 19.5 · Ξεκλείδωσε ένα ορφανό lock

Ο φάκελος της άσκησης ξεκινά ως αντίγραφο του `slow`. Άλλαξε μόνο το key σε `ex5/terraform.tfstate` και κάνε init. Ξεκίνα apply στο παρασκήνιο, περίμενε πέντε δευτερόλεπτα και σκότωσέ το με `kill -9`. Δείξε ότι το plan αποτυγχάνει, βρες το ID του lock και ξεκλείδωσε με `terraform force-unlock -force`. Το αναμενόμενο αποτέλεσμα είναι η έξοδος του `force-unlock`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    time = {
      source  = "hashicorp/time"
      version = "0.14.2"
    }
  }

  backend "s3" {
    bucket       = "tfbook-state"
    key          = "ex5/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }
}

resource "time_sleep" "wait" {
  create_duration = "20s"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve > apply.log 2>&1 &
sleep 5
kill -9 $!
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
│ Error: Error acquiring the state lock
…
│   ID:        f0870c43-6e97-a77a-6284-112a67261acc
…
```

ΛΥΣΗ · TERMINAL

```
LOCK_ID=$(aws s3 cp s3://tfbook-state/ex5/terraform.tfstate.tflock - | jq -r .ID)
terraform force-unlock -force $LOCK_ID
```

ΕΞΟΔΟΣ

```
Terraform state has been successfully unlocked!

The state has been unlocked, and Terraform commands should now be able to
obtain a new lock on the remote state.
```

Το `kill -9` άφησε το `.tflock` στο bucket και το plan σταμάτησε με το ID στο `Lock Info`. Το ίδιο ID είναι μέσα στο αντικείμενο του lock, οπότε το `jq` το βρίσκει χωρίς αντιγραφή από την οθόνη. Το `force-unlock` είναι ασφαλές εδώ, γιατί ξέρεις ότι το apply πέθανε. Αν έτρεχε ακόμη, θα του έπαιρνες το lock μέσα από τα χέρια του.

Άσκηση 19.6 · Το πρόθεμα του base

Σε νέο φάκελο γράψε configuration χωρίς backend, δηλαδή με local state, που διαβάζει με `terraform_remote_state` το state του `base` από το `team/base.tfstate`. Δώσε ένα output `shout` με το πρόθεμα του `base` σε κεφαλαία γράμματα. Κάνε init και apply. Το δικό σου πρόθεμα θα διαφέρει.

ΛΥΣΗ · main.tf

```
data "terraform_remote_state" "base" {
  backend = "s3"

  config = {
    bucket = "tfbook-state"
    key    = "team/base.tfstate"
    region = "eu-central-1"
  }
}

output "shout" {
  value = upper(data.terraform_remote_state.base.outputs.prefix)
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Outputs:

shout = "VIABLE-PIGLET"
```

Το backend μιας configuration και το backend που διαβάζει ένα `terraform_remote_state` είναι ανεξάρτητα, οπότε μια configuration με local state διαβάζει άνετα ένα remote state. Το output του `base` είναι string και περνά από την `upper` του κεφαλαίου 10 όπως κάθε τιμή. Με `outputs.length` θα έπαιρνες `Unsupported attribute`, γιατί το `length` δεν είναι output.

Ερωτήσεις πιστοποίησης

Ερώτηση 19.1

Μια ομάδα θέλει να ορίζει το key του S3 backend από μια input variable, ώστε ο ίδιος κώδικας να γράφει σε διαφορετικό state ανά περιβάλλον. Τι κάνει;

a. Γράφει `key = var.state_key` στο block `backend`.
b. Ορίζει ένα local με το key και το χρησιμοποιεί στο block `backend`.
c. Αφήνει το key έξω από το block και το δίνει με `terraform init -backend-config`.
d. Δίνει το key με `terraform apply -var="key=dev.tfstate"`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το block `backend` ρυθμίζεται στο `init`, πριν υπολογιστούν μεταβλητές και locals, οπότε τα a και b αποτυγχάνουν με `Variables not allowed`. Η partial configuration δίνει τις τιμές που λείπουν στο `init` με `-backend-config`. Το d δίνει τιμή σε μεταβλητή στο apply. Τότε το backend έχει ήδη ρυθμιστεί, οπότε η τιμή δεν το αφορά.

Ερώτηση 19.2

Δύο μηχανικοί τρέχουν `terraform apply` την ίδια στιγμή στην ίδια configuration, με S3 backend και `use_lockfile = true`. Τι γίνεται;

a. Τα δύο apply τρέχουν μαζί και το Terraform ενώνει τις αλλαγές στο state.
b. Όποιο πάρει πρώτο το lock συνεχίζει. Το άλλο σταματά με `Error acquiring the state lock`.
c. Το δεύτερο apply περιμένει πάντα χωρίς όριο μέχρι να ελευθερωθεί το lock.
d. Το δεύτερο apply σβήνει το lock του πρώτου και συνεχίζει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το lock επιτρέπει μία εντολή κάθε φορά. Η δεύτερη βρίσκει το `.tflock` και σταματά αμέσως. Περιμένει μόνο με `-lock-timeout` και πάλι έως ένα όριο, άρα το c είναι λάθος. Το Terraform δεν ενώνει states, οπότε το a είναι λάθος. Ένα lock άλλου το σβήνει μόνο άνθρωπος με `force-unlock`, άρα και το d είναι λάθος.

Ερώτηση 19.3

Ένα apply στο CI σκοτώθηκε στη μέση και κάθε νέο plan αποτυγχάνει με `Error acquiring the state lock`. Βεβαιώθηκες ότι καμία εντολή δεν τρέχει. Τι κάνεις;

a. `terraform apply -lock=false` για να ολοκληρωθεί η δουλειά.
b. `terraform init -reconfigure` για να ξαναρυθμιστεί το backend.
c. `terraform plan -lock-timeout=10m` για να περιμένει να λήξει το lock.
d. `terraform force-unlock` με το ID του lock από το μήνυμα λάθους.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `force-unlock` σβήνει το ορφανό lock με το ID του `Lock Info` και δεν αλλάζει state ή υποδομή. Το a παρακάμπτει το lock, ρισκάρει το state και αφήνει το lock για τον επόμενο. Το b αλλάζει μόνο τη ρύθμιση του φακέλου και δεν αγγίζει το `.tflock`. Το c περιμένει κάτι που δεν θα γίνει. Το lock δεν λήγει μόνο του.

Ερώτηση 19.4

Άλλαξες το key στο block `backend "s3"` μιας configuration που έχει ήδη state. Θέλεις το υπάρχον state να βρεθεί στο νέο key. Ποια εντολή τρέχεις;

a. `terraform init -migrate-state`
b. `terraform init -reconfigure`
c. `terraform init -upgrade`
d. `terraform init` χωρίς σημαίες, που μεταφέρει μόνο του το state

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το `-migrate-state` αντιγράφει το state στη νέα θέση, αφού ρωτήσει. Το `-reconfigure` δεν αντιγράφει τίποτα, οπότε το νέο key θα έμενε άδειο. Το `-upgrade` αφορά τις εκδόσεις providers και modules. Ένα σκέτο `init` σε φάκελο με ήδη ρυθμισμένο backend σταματά με `Backend configuration changed` και ζητά μία από τις δύο σημαίες.

Ερώτηση 19.5

Ποιες δύο προτάσεις για το data source `terraform_remote_state` ισχύουν; Επίλεξε δύο.

a. Διαβάζει κάθε attribute κάθε resource του άλλου state.
b. Δίνει μόνο τα outputs του root module του άλλου state.
c. Δουλεύει μόνο με το S3 backend.
d. Όποιος το χρησιμοποιεί χρειάζεται δικαίωμα ανάγνωσης σε όλο το άλλο state.
e. Μπορεί να γράψει τιμές στο άλλο state.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Το `terraform_remote_state` εκθέτει μόνο τα outputs του root module, οπότε το a είναι λάθος. Για να τα βρει όμως κατεβάζει όλο το state, άρα θέλει πρόσβαση σε όλο το state με τα μυστικά του. Δουλεύει με κάθε backend, μαζί με το `local`, οπότε το c είναι λάθος. Είναι data source και μόνο διαβάζει, άρα το e είναι λάθος.

Ερώτηση 19.6

Σωστό ή λάθος; Για να χρησιμοποιήσεις το S3 backend πρέπει να δηλώσεις τον provider `hashicorp/aws` στο `required_providers`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Τα backends είναι ενσωματωμένα στο Terraform και δεν είναι providers. Μια configuration χωρίς κανένα resource του AWS κρατά κανονικά το state της στο S3 χωρίς τον AWS provider. Το backend βρίσκει credentials όπως ο provider, αλλά δεν διαβάζει το block `provider "aws"`.

ΚΕΦΑΛΑΙΟ 20

Workspaces

Άσκηση 20.1 · Ένα workspace για δοκιμή

Ξεκίνα από αντίγραφο του `site`. Φτιάξε το workspace `test` και κάνε apply σε αυτό. Δείξε μετά τη λίστα των workspaces και το output `bucket`.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform workspace new test
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform workspace list
terraform output bucket
```

ΕΞΟΔΟΣ

```
  default
* test

"tfbook-site-test"
```

Το αντίγραφο ξεκινά στο `default`, γιατί το `.terraform/environment` δεν αντιγράφηκε. Η `new` φτιάχνει το `test` και περνά σε αυτό, οπότε το apply γράφει στο state του `test` και το bucket παίρνει το όνομα του workspace. Αν έκανες apply πριν από τη `new`, το bucket θα ήταν το `tfbook-site-default` και θα έμενε στο state του `default`.

Άσκηση 20.2 · Πού είναι τα states

Ξεκίνα από αντίγραφο του `site`. Φτιάξε το workspace `qa` και κάνε apply σε αυτό. Μετά φτιάξε το workspace `uat` χωρίς apply. Πριν τρέξεις `find terraform.tfstate.d | sort`, γράψε τι περιμένεις να δεις.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform workspace new qa
terraform apply -auto-approve
terraform workspace new uat
```

ΛΥΣΗ · TERMINAL

```
find terraform.tfstate.d | sort
```

ΕΞΟΔΟΣ

```
terraform.tfstate.d
terraform.tfstate.d/qa
terraform.tfstate.d/qa/terraform.tfstate
terraform.tfstate.d/uat
```

Κάθε workspace εκτός από το `default` παίρνει υποφάκελο με το όνομά του μέσα στο `terraform.tfstate.d`. Ο υποφάκελος φτιάχνεται με τη `new`, αλλά το `terraform.tfstate` μέσα του γράφεται μόνο όταν το Terraform αποθηκεύσει state, εδώ με το apply. Γι' αυτό το `uat` είναι άδειος φάκελος. Το state του `default` δεν εμφανίζεται εδώ, γιατί μένει στη ρίζα.

Άσκηση 20.3 · Ένα bucket μόνο στο prod

Ξεκίνα από αντίγραφο του `site`. Πρόσθεσε στο αρχείο `logs.tf` ένα `aws_s3_bucket` με όνομα `logs` που υπάρχει μόνο στο workspace `prod`, με bucket `tfbook-logs-prod`. Χρησιμοποίησε `count` όπως στο κεφάλαιο 11. Τρέξε plan στο `default`, φτιάξε το `prod` και τρέξε plan ξανά. Μην κάνεις apply.

ΛΥΣΗ · logs.tf

```
resource "aws_s3_bucket" "logs" {
  count = terraform.workspace == "prod" ? 1 : 0

  bucket = "tfbook-logs-${terraform.workspace}"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
terraform workspace new prod
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to add, 0 to change, 0 to destroy.
…
Created and switched to workspace "prod"!
…
  # aws_s3_bucket.logs[0] will be created
…
Plan: 2 to add, 0 to change, 0 to destroy.
…
```

Η συνθήκη δίνει `count = 1` μόνο όταν το `terraform.workspace` είναι `"prod"`. Στο `default` το plan έχει μόνο το bucket του `site`, ενώ στο `prod` έχει και το `aws_s3_bucket.logs[0]`. Αυτό είναι το κλασικό μοτίβο της εξέτασης για resources που υπάρχουν μόνο σε ένα workspace. Με λίγες τέτοιες συνθήκες η configuration μένει ευανάγνωστη. Με πολλές, οι χωριστοί φάκελοι είναι καθαρότεροι.

Άσκηση 20.4 · lookup ή index

Ξεκίνα από αντίγραφο του `site` και φτιάξε το workspace `staging`. Δες στο `terraform console` τι δίνει το `local.instance_type`. Μετά άλλαξε στο `main.tf` την έκφραση του `instance_type` σε `local.instance_types[terraform.workspace]` και τρέξε plan. Τι γίνεται;

ΛΥΣΗ · TERMINAL

```
terraform init
terraform workspace new staging
```

ΛΥΣΗ · TERRAFORM CONSOLE

```
> local.instance_type
"t3.small"
```

ΛΥΣΗ · TERMINAL

```
sed -i.bak 's/lookup(local.instance_types, terraform.workspace, "t3.small")/
local.instance_types[terraform.workspace]/' main.tf
rm main.tf.bak
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
╷
│ Error: Invalid index
╷
│ The given key does not identify an element in this collection value.
╵
```

Το `staging` δεν υπάρχει στο map. Η `lookup` δίνει τότε την προεπιλογή, το `"t3.small"`. Η πρόσβαση με αγκύλες δεν έχει προεπιλογή και το plan σταματά με `Invalid index`. Κάθε νέο workspace θα έσπαγε την configuration μέχρι να προσθέσει κάποιος το κλειδί του. Το `sed` αλλάζει την έκφραση από το terminal. Μπορείς να την αλλάξεις και με τον editor σου.

Άσκηση 20.5 · Σωστή διαγραφή

Ξεκίνα από αντίγραφο του `site`. Φτιάξε το workspace `tmp`, κάνε apply και μετά σβήσε το workspace χωρίς `-force` και χωρίς να μείνει bucket στο AWS.

ΛΥΣΗ · TERMINAL

```
terraform init
terraform workspace new tmp
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
terraform workspace select default
terraform workspace delete tmp
```

ΕΞΟΔΟΣ

```
…
Destroy complete! Resources: 1 destroyed.
Switched to workspace "default".
Deleted workspace "tmp"!
```

Η σειρά έχει σημασία. Το destroy πρέπει να γίνει όσο είσαι ακόμη στο `tmp`, γιατί μόνο το state του `tmp` ξέρει το bucket. Μετά βγαίνεις από το workspace, γιατί το τρέχον δεν σβήνεται. Τέλος το σβήνεις. Το state είναι πια άδειο, οπότε η `delete` δεν χρειάζεται `-force`. Αν είχες σβήσει πρώτα το workspace με `-force`, το `tfbook-site-tmp` θα έμενε ορφανό.

Άσκηση 20.6 · Άλλο key και άλλο πρόθεμα

Ξεκίνα από αντίγραφο του `app`. Άλλαξε στο `backend.tf` το `key` σε `web/terraform.tfstate` και το `workspace_key_prefix` σε `teams`. Τρέξε `terraform init`, φτιάξε το workspace `blue`, κάνε apply και δείξε τη λίστα των workspaces και τα αρχεία του bucket `tfbook-state` που περιέχουν `teams`.

ΛΥΣΗ · backend.tf

```
terraform {
  backend "s3" {
    bucket               = "tfbook-state"
    key                  = "web/terraform.tfstate"
    workspace_key_prefix = "teams"
    region               = "eu-central-1"
    use_lockfile         = true
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform workspace new blue
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform workspace list
aws s3 ls s3://tfbook-state --recursive
```

ΕΞΟΔΟΣ

```
  default
* blue
…
2026-10-08 05:34:27       2473 teams/blue/web/terraform.tfstate
…
```

Το state του `blue` γράφτηκε στο `teams/blue/web/terraform.tfstate`, δηλαδή πρόθεμα, workspace και `key`. Η λίστα δείχνει μόνο το `default` και το `blue`. Τα `dev` και `prod` του κεφαλαίου είναι στο ίδιο bucket, αλλά με άλλο `key` και άλλο πρόθεμα, οπότε δεν ανήκουν σε αυτή την configuration. Αν είχες κρατήσει το `backend.tf` του `app` όπως ήταν, το αντίγραφο θα έβλεπε τα ίδια states με το `app`. Κάθε apply ή destroy εδώ θα άγγιζε τα buckets του `app`.

Άσκηση 20.7 · Ένα τρίτο περιβάλλον με φάκελο

Ξεκίνα από αντίγραφο του `shop`. Πρόσθεσε το περιβάλλον `staging` ως φάκελο `envs/staging`, με το state του στο `tfbook-state` και `key shop/staging/terraform.tfstate`. Κάνε apply και δείξε το output `bucket`.

ΛΥΣΗ · envs/staging/main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }

  backend "s3" {
    bucket       = "tfbook-state"
    key          = "shop/staging/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }
}

provider "aws" {
  region = "eu-central-1"
}

module "bucket" {
  source = "../../modules/bucket"
  env    = "staging"
}

output "bucket" {
  value = module.bucket.name
}
```

ΛΥΣΗ · TERMINAL

```
cd envs/staging
cp ../dev/.terraform.lock.hcl .
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output bucket
```

ΕΞΟΔΟΣ

```
"tfbook-shop-staging"
```

Το νέο περιβάλλον είναι ένας νέος φάκελος με δικό του backend και μια ακόμη κλήση του ίδιου module. Το `key` πρέπει να είναι μοναδικό. Αν έγραφες το `shop/dev/terraform.tfstate`, το `staging` θα μοιραζόταν το state του `dev` και το plan θα ήθελε να αντικαταστήσει το bucket του `dev`. Το lock file αντιγράφεται από το `dev`, ώστε το init να πάρει τον provider από το plugin cache όπως στο κεφάλαιο 17.

Ερωτήσεις πιστοποίησης

Ερώτηση 20.1

Μια configuration χρησιμοποιεί το local backend. Τρέχεις `terraform workspace new staging` και μετά `terraform apply`. Πού γράφεται το state;

a. Στο `terraform.tfstate` στη ρίζα του φακέλου εργασίας
b. Στο `.terraform/staging.tfstate`
c. Στο `terraform.tfstate.d/staging/terraform.tfstate`
d. Στο `staging/terraform.tfstate`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το local backend κρατά στη ρίζα μόνο το state του `default`. Κάθε άλλο workspace γράφει σε υποφάκελο του `terraform.tfstate.d` με το όνομά του, άρα το a είναι λάθος. Ο φάκελος `.terraform` κρατά providers, modules και το `environment` με το επιλεγμένο workspace, όχι states, οπότε το b είναι λάθος. Το Terraform δεν φτιάχνει φάκελο με το όνομα του workspace στη ρίζα, άρα και το d είναι λάθος.

Ερώτηση 20.2

Ένα S3 backend έχει `key = "network/terraform.tfstate"` και δεν ορίζει `workspace_key_prefix`. Σε ποιο object γράφεται το state του workspace `prod`;

a. `prod/network/terraform.tfstate`
b. `env:/prod/network/terraform.tfstate`
c. `network/terraform.tfstate`
d. `network/prod/terraform.tfstate`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το S3 backend γράφει τα workspaces εκτός από το `default` στη διαδρομή πρόθεμα, όνομα workspace και `key`. Το προεπιλεγμένο πρόθεμα είναι το `env:`. Το a παραλείπει το πρόθεμα. Το c είναι η θέση του `default`. Το d βάζει το όνομα του workspace μέσα στο `key`, κάτι που το backend δεν κάνει.

Ερώτηση 20.3

Ποιες δύο προτάσεις για το workspace `default` είναι σωστές; Επίλεξε δύο.

a. Υπάρχει σε κάθε configuration, ακόμη κι αν δεν έτρεξες ποτέ εντολή `terraform workspace`.
b. Το S3 backend γράφει το state του στο `env:/default/` και το `key`.
c. Δεν μπορεί να σβηστεί με την `terraform workspace delete`.
d. Σβήνεται με `terraform workspace delete -force default`.
e. Πρέπει να το φτιάξεις με `terraform workspace new default` πριν από το πρώτο apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το `default` υπάρχει πάντα και σε αυτό τρέχει κάθε configuration που δεν επέλεξε άλλο workspace, άρα το e είναι λάθος. Η `delete` αρνείται να το σβήσει, με ή χωρίς `-force`, οπότε το d είναι λάθος. Στο S3 backend το state του `default` γράφεται ακριβώς στο `key`, χωρίς πρόθεμα, άρα το b είναι λάθος.

Ερώτηση 20.4

Σωστό ή λάθος; Η εντολή `terraform workspace delete -force dev` αφήνει στο cloud τα resources που καταγράφει το state του `dev`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Η `-force` παρακάμπτει μόνο τον έλεγχο για resources. Σβήνει το state του workspace και τυπώνει προειδοποίηση ότι τα resources μπορεί να υπάρχουν ακόμη. Δεν κάνει destroy. Τα resources μένουν και το Terraform δεν τα διαχειρίζεται πια. Για να τα σβήσεις κάνεις destroy στο workspace πριν το σβήσεις.

Ερώτηση 20.5

Η ομάδα σου έχει development και production σε δύο διαφορετικούς λογαριασμούς AWS. Μόνο δύο άνθρωποι επιτρέπεται να αλλάζουν το production και το state του πρέπει να μην είναι προσβάσιμο από τους υπόλοιπους. Ποια οργάνωση ταιριάζει;

a. Ένας φάκελος με δύο CLI workspaces, `dev` και `prod`, στο ίδιο S3 backend
b. Ένας φάκελος με CLI workspaces και `workspace_key_prefix = "prod"`
c. Ένας φάκελος με το `terraform.workspace` στο `key` του backend
d. Χωριστοί φάκελοι ή χωριστά HCP Terraform workspaces, με χωριστό state και δικά τους credentials

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Τα CLI workspaces μοιράζονται backend και credentials, οπότε τα a και b δεν δίνουν την απομόνωση που ζητείται. Το `workspace_key_prefix` αλλάζει μόνο τη διαδρομή μέσα στο ίδιο bucket. Το c δεν δουλεύει καν, γιατί το block `backend` δεν δέχεται το `terraform.workspace`. Χωριστοί φάκελοι ή χωριστά HCP Terraform workspaces έχουν δικό τους state, δικά τους credentials και δικά τους δικαιώματα.

Ερώτηση 20.6

Ποια πρόταση περιγράφει σωστά τη διαφορά ανάμεσα σε ένα CLI workspace και σε ένα HCP Terraform workspace;

a. Είναι το ίδιο πράγμα. Το HCP Terraform απλώς αποθηκεύει τα CLI workspaces στο cloud.
b. Ένα CLI workspace είναι ένα ακόμη state της ίδιας configuration, ενώ ένα HCP Terraform workspace έχει δική του configuration, state, variables και δικαιώματα.
c. Ένα HCP Terraform workspace είναι ένα από πολλά states του ίδιου φακέλου, ενώ ένα CLI workspace έχει δικά του δικαιώματα ανά team.
d. Τα CLI workspaces δουλεύουν μόνο με το local backend, ενώ τα HCP Terraform workspaces μόνο με remote backend.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ένα CLI workspace μοιράζεται configuration, backend και credentials με τα υπόλοιπα και διαφέρει μόνο στο state. Ένα HCP Terraform workspace είναι ολόκληρη μονάδα με δική του configuration, state, variables, runs και δικαιώματα. Μοιάζει με χωριστό φάκελο εργασίας. Το a αγνοεί αυτή τη διαφορά. Το c αντιστρέφει τους ρόλους. Το d είναι λάθος, γιατί τα CLI workspaces δουλεύουν και με remote backends όπως το S3, όπως είδες στο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 21

Drift, refresh-only και εντολές state

Άσκηση 21.1 · Η φορά του βέλους

Γράψε configuration με ένα bucket `tfbook-ch21-ex1` και όνομα resource `data`. Πρόσθεσε versioning με `status = "Enabled"`. Κάνε apply. Σταμάτησε το versioning με το `aws` CLI. Τρέξε πρώτα `terraform plan -refresh-only` και μετά `terraform plan`. Σύγκρινε τις γραμμές του `status` στα δύο plans.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "data" {
  bucket = "tfbook-ch21-ex1"
}

resource "aws_s3_bucket_versioning" "data" {
  bucket = aws_s3_bucket.data.id

  versioning_configuration {
    status = "Enabled"
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
aws s3api put-bucket-versioning --bucket tfbook-ch21-ex1 \
  --versioning-configuration Status=Suspended
```

ΛΥΣΗ · TERMINAL

```
terraform plan -refresh-only
terraform plan
```

ΕΞΟΔΟΣ

```
…
          ~ status     = "Enabled" -> "Suspended"
…
This is a refresh-only plan, so Terraform will not take any actions to undo
…
          ~ status     = "Suspended" -> "Enabled"
…
Plan: 0 to add, 1 to change, 0 to destroy.
…
```

Το refresh-only plan συγκρίνει το state με το AWS. Γράφει `"Enabled" -> "Suspended"`, δηλαδή τι άλλαξε έξω από το Terraform. Λέει επίσης ρητά ότι δεν θα αναιρέσει τίποτα. Το κανονικό plan συγκρίνει τη configuration με το AWS. Γράφει `"Suspended" -> "Enabled"` και προτείνει μία αλλαγή. Η ίδια διαφορά εμφανίζεται με αντίθετη φορά, γιατί κάθε plan κοιτάζει από άλλη πλευρά. Μην τα μπερδεύεις όταν διαβάζεις βιαστικά.

Άσκηση 21.2 · Αποδοχή μέσα από τον κώδικα

Γράψε configuration με ένα bucket `tfbook-ch21-ex2` και όνομα resource `reports`. Πρόσθεσε versioning με `status = "Enabled"`. Κάνε apply. Σταμάτησε το versioning με το CLI. Η αλλαγή ήταν σκόπιμη. Αποδέξου την αλλάζοντας μόνο τη configuration και κάνε apply. Μετά απόδειξε με `terraform plan -refresh-only` ότι και το state ενημερώθηκε.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "reports" {
  bucket = "tfbook-ch21-ex2"
}

resource "aws_s3_bucket_versioning" "reports" {
  bucket = aws_s3_bucket.reports.id

  versioning_configuration {
    status = "Enabled"
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
aws s3api put-bucket-versioning --bucket tfbook-ch21-ex2 \
  --versioning-configuration Status=Suspended
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "reports" {
  bucket = "tfbook-ch21-ex2"
}

resource "aws_s3_bucket_versioning" "reports" {
  bucket = aws_s3_bucket.reports.id

  versioning_configuration {
    status = "Suspended"
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

ΛΥΣΗ · TERMINAL

```
terraform plan -refresh-only
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure still matches the configuration.
…
```

Μετά την αλλαγή στον κώδικα το plan του apply δεν έχει ενέργειες, γιατί η configuration ζητά ό,τι ήδη υπάρχει. Το apply όμως έγραψε στο state τις τιμές του refresh. Γι' αυτό το refresh-only plan δεν βρίσκει πια drift. Δεν χρειάστηκε `apply -refresh-only`. Αν είχες κάνει apply πριν αλλάξεις τον κώδικα, θα είχες ξαναενεργοποιήσει το versioning.

Άσκηση 21.3 · Το bucket που σβήστηκε

Γράψε configuration με ένα bucket `tfbook-ch21-ex3` και όνομα resource `temp`. Κάνε apply. Σβήσε το bucket με `aws s3 rb`. Η διαγραφή ήταν σκόπιμη. Κάνε το state να το ξεχάσει χωρίς να το ξαναφτιάξεις και χωρίς να γράψεις τη διεύθυνση του resource. Τι δείχνει μετά το `terraform state list` και τι θα πρότεινε ένα κανονικό plan;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "temp" {
  bucket = "tfbook-ch21-ex3"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
aws s3 rb s3://tfbook-ch21-ex3
```

ΛΥΣΗ · TERMINAL

```
terraform apply -refresh-only -auto-approve
terraform state list
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.temp has been deleted
…
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
…
Plan: 1 to add, 0 to change, 0 to destroy.
…
```

Το `apply -refresh-only` είδε ότι το bucket σβήστηκε και έγραψε το state χωρίς αυτό. Η περίληψη λέει `0 destroyed`, γιατί το Terraform δεν έσβησε τίποτα. Το `terraform state list` δεν τυπώνει πια τίποτα. Το block όμως είναι ακόμη στον κώδικα, οπότε το κανονικό plan προτείνει `1 to add`. Για να ολοκληρωθεί η αποδοχή σβήνεις και το block. Το `terraform state rm` θα έκανε το ίδιο, αλλά θέλει τη διεύθυνση και δεν ελέγχει αν το bucket λείπει πράγματι από το AWS.

Άσκηση 21.4 · Ποια διεύθυνση έχει αυτό το bucket

Γράψε configuration με δύο buckets. Το `tfbook-ch21-ex4-a` με όνομα resource `media` και versioning ενεργό. Το `tfbook-ch21-ex4-b` με όνομα resource `backup`. Κάνε apply. Βρες με μία εντολή όλες τις διευθύνσεις του state που έχουν `id` ίσο με `tfbook-ch21-ex4-a`. Μετά τύπωσε από το state μόνο τη γραμμή του `status` του versioning.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "media" {
  bucket = "tfbook-ch21-ex4-a"
}

resource "aws_s3_bucket_versioning" "media" {
  bucket = aws_s3_bucket.media.id

  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket" "backup" {
  bucket = "tfbook-ch21-ex4-b"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list -id=tfbook-ch21-ex4-a
terraform state show aws_s3_bucket_versioning.media
```

ΕΞΟΔΟΣ

```
aws_s3_bucket.media
aws_s3_bucket_versioning.media
…
        status     = "Enabled"
…
```

Το `-id` κρατά κάθε resource με αυτό το `id`. Το bucket και το versioning του έχουν το ίδιο, το όνομα του bucket, οπότε εμφανίζονται και τα δύο. Το `backup` έχει άλλο `id` και λείπει. Το `terraform state show` θέλει μία συγκεκριμένη διεύθυνση και τυπώνει τα attributes όπως τα θυμάται το state. Το `grep` του βιβλίου κρατά μόνο τις σχετικές γραμμές. Εσύ μπορείς να χρησιμοποιήσεις `-no-color` και `grep`, όπως στο κεφάλαιο 3.

Άσκηση 21.5 · Μετονομασία χωρίς αντικατάσταση

Γράψε configuration με ένα bucket `tfbook-ch21-ex5` και όνομα resource `old`. Κάνε apply. Μετονόμασε το resource σε `invoices` στον κώδικα και κάνε το state να ακολουθήσει, ώστε το επόμενο plan να μην έχει αλλαγές.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "old" {
  bucket = "tfbook-ch21-ex5"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "invoices" {
  bucket = "tfbook-ch21-ex5"
}
```

ΛΥΣΗ · TERMINAL

```
terraform state mv aws_s3_bucket.old aws_s3_bucket.invoices
terraform plan
```

ΕΞΟΔΟΣ

```
Move "aws_s3_bucket.old" to "aws_s3_bucket.invoices"
Successfully moved 1 object(s).
…
No changes. Your infrastructure matches the configuration.
…
```

Το `state mv` άλλαξε μόνο τη διεύθυνση στο state. Το bucket στο AWS δεν άγγιξε. Μετά τη μετονομασία ο κώδικας και το state συμφωνούν, οπότε το plan δεν έχει αλλαγές. Αν είχες τρέξει plan πριν από το `state mv`, θα έβλεπες destroy και create. Αυτό δεν πειράζει, αρκεί να μην κάνεις apply πριν από το `state mv`. Το block `moved` του κεφαλαίου 23 κάνει την ίδια δουλειά μέσα από το plan.

Άσκηση 21.6 · Παράδοση σε άλλη ομάδα

Γράψε configuration με δύο buckets. Το `tfbook-ch21-ex6-keep` με όνομα resource `keep` και το `tfbook-ch21-ex6-give` με όνομα resource `give`. Κάνε apply. Το `give` περνά σε άλλη ομάδα. Κάνε το Terraform να το ξεχάσει χωρίς να σβηστεί, ώστε το επόμενο plan να μην έχει αλλαγές. Τύπωσε το state και τα ονόματα των buckets της άσκησης με `aws s3api list-buckets --query "Buckets[?contains(Name, 'ex6')].Name"`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "keep" {
  bucket = "tfbook-ch21-ex6-keep"
}

resource "aws_s3_bucket" "give" {
  bucket = "tfbook-ch21-ex6-give"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
terraform state rm aws_s3_bucket.give
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "keep" {
  bucket = "tfbook-ch21-ex6-keep"
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

ΛΥΣΗ · TERMINAL

```
terraform state list
aws s3api list-buckets --query "Buckets[?contains(Name, 'ex6')].Name"
```

ΕΞΟΔΟΣ

```
aws_s3_bucket.keep
[
    "tfbook-ch21-ex6-keep",
    "tfbook-ch21-ex6-give"
]
```

Το `state rm` αφαίρεσε το `give` από το state και το bucket έμεινε στο AWS, όπως δείχνει η λίστα. Η σειρά έχει σημασία. Αν σβήσεις πρώτα το block και κάνεις apply, το Terraform σβήνει το bucket, αφού είναι άδειο. Αν τρέξεις μόνο το `state rm` και κρατήσεις το block, το επόμενο apply προσπαθεί να το ξαναφτιάξει και αποτυγχάνει με `BucketAlreadyOwnedByYou`. Χρειάζονται και τα δύο βήματα.

Άσκηση 21.7 · Η άμυνα του state push

Γράψε configuration με ένα bucket `tfbook-ch21-ex7` και όνομα resource `notes`. Κάνε apply και κράτησε αντίγραφο του state στο `v1.tfstate`. Πρόσθεσε δεύτερο bucket `tfbook-ch21-ex7-extra` με όνομα resource `extra` και κάνε apply. Δοκίμασε να γράψεις το `v1.tfstate` ως state χωρίς `-force`. Γιατί αποτυγχάνει και τι θα γινόταν με `-force`;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "notes" {
  bucket = "tfbook-ch21-ex7"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
terraform state pull > v1.tfstate
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_s3_bucket" "notes" {
  bucket = "tfbook-ch21-ex7"
}

resource "aws_s3_bucket" "extra" {
  bucket = "tfbook-ch21-ex7-extra"
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state push v1.tfstate
```

ΕΞΟΔΟΣ

```
Failed to write state: cannot import state with serial 1 over newer state with serial 3
```

Το `v1.tfstate` έχει το ίδιο `lineage` με το τρέχον state, αλλά μικρότερο `serial`, γιατί το δεύτερο apply έγραψε το state ξανά. Το Terraform αρνείται να γράψει παλαιότερη εκδοχή πάνω σε νεότερη. Με `-force` θα το έγραφε. Το state θα ξεχνούσε το `extra`, ενώ το bucket θα έμενε στο AWS. Το επόμενο apply θα προσπαθούσε να το φτιάξει ξανά και θα αποτύγχανε με `BucketAlreadyOwnedByYou`.

Ερωτήσεις πιστοποίησης

Ερώτηση 21.1

Ένας συνάδελφος άλλαξε από την κονσόλα την τιμή ενός tag σε bucket που διαχειρίζεται το Terraform. Η configuration δεν άλλαξε. Τι θα δείξει το επόμενο `terraform plan`;

a. Τίποτα, γιατί το state δεν άλλαξε.
b. Σφάλμα που ζητά να τρέξεις πρώτα `terraform refresh`.
c. Αλλαγή επί τόπου που επαναφέρει το tag στην τιμή της configuration.
d. Αντικατάσταση του bucket, γιατί άλλαξε έξω από το Terraform.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το plan κάνει refresh, βρίσκει την πραγματική τιμή και τη συγκρίνει με τη configuration, που θεωρεί σωστή. Τα tags αλλάζουν επί τόπου, οπότε βλέπεις `~` με την τιμή του AWS αριστερά και της configuration δεξιά. Το a είναι λάθος, γιατί το plan δεν βασίζεται μόνο στο state. Το b είναι λάθος, γιατί το refresh γίνεται αυτόματα σε κάθε plan. Το d είναι λάθος, γιατί ένα drift δεν προκαλεί αντικατάσταση από μόνο του. Αντικατάσταση γίνεται μόνο όταν αλλάζει argument που την απαιτεί.

Ερώτηση 21.2

Θέλεις να καταγράψεις στο state αλλαγές που έγιναν έξω από το Terraform, χωρίς να αλλάξει κανένα πραγματικό αντικείμενο και αφού δεις πρώτα τι θα γραφτεί. Ποια εντολή χρησιμοποιείς;

a. `terraform plan -refresh=false`
b. `terraform apply -refresh-only`
c. `terraform refresh`
d. `terraform apply -replace=aws_s3_bucket.logs`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `terraform apply -refresh-only` δείχνει το drift, ζητά έγκριση και γράφει μόνο το state. Το a δεν κάνει καν refresh και δεν γράφει τίποτα. Το c ενημερώνει το state χωρίς να δείξει τι αλλάζει και χωρίς ερώτηση. Είναι επίσης deprecated. Το d αντικαθιστά πραγματικό αντικείμενο, δηλαδή αλλάζει την υποδομή.

Ερώτηση 21.3

Εκτελείς `terraform state rm aws_s3_bucket.logs` και δεν αλλάζεις τη configuration. Ποιες δύο προτάσεις ισχύουν; Επίλεξε δύο.

a. Το bucket συνεχίζει να υπάρχει στο AWS.
b. Το bucket σβήνεται από το AWS στο επόμενο apply.
c. Η εντολή δείχνει πρώτα plan και ζητά επιβεβαίωση.
d. Το επόμενο plan προτείνει να δημιουργηθεί το `aws_s3_bucket.logs`.
e. Το Terraform σβήνει αυτόματα το block από τη configuration.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και d

Το `state rm` αφαιρεί μόνο την εγγραφή από το state. Το αντικείμενο μένει στο AWS. Το block όμως μένει στον κώδικα, οπότε το Terraform βλέπει resource στη configuration που λείπει από το state και προτείνει create. Το b είναι λάθος, γιατί το Terraform δεν θυμάται πια το bucket για να το σβήσει. Το c είναι λάθος, γιατί οι εντολές state ενεργούν αμέσως χωρίς plan. Το e είναι λάθος, γιατί καμία εντολή state δεν αγγίζει τα αρχεία `.tf`.

Ερώτηση 21.4

Μετονόμασες στον κώδικα το `aws_s3_bucket.assets` σε `aws_s3_bucket.static` χωρίς να αλλάξεις το όνομα του bucket. Ποια εντολή αποτρέπει την αντικατάσταση του bucket στο επόμενο apply;

a. `terraform state rm aws_s3_bucket.assets`
b. `terraform state mv aws_s3_bucket.assets aws_s3_bucket.static`
c. `terraform apply -refresh-only`
d. `terraform state push terraform.tfstate.backup`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `state mv` μεταφέρει το αντικείμενο στη νέα διεύθυνση, οπότε κώδικας και state συμφωνούν. Το ίδιο κάνει και ένα block `moved`. Το a κάνει το Terraform να ξεχάσει το bucket και το plan θα ήθελε να φτιάξει το `static` από την αρχή. Το c ενημερώνει τιμές από το AWS, όχι διευθύνσεις. Το d θα έγραφε ένα παλαιότερο state, που έχει κι αυτό τη διεύθυνση `assets`. Θα αποτύγχανε μάλιστα χωρίς `-force`, γιατί το `serial` του είναι μικρότερο.

Ερώτηση 21.5

Σωστό ή λάθος; Το `terraform plan -refresh=false` εντοπίζει αλλαγές που έγιναν στην υποδομή έξω από το Terraform.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Με `-refresh=false` το Terraform δεν ρωτά τους providers για τα πραγματικά αντικείμενα. Συγκρίνει τη configuration μόνο με το state, οπότε το drift δεν φαίνεται. Η σημαία κάνει το plan γρηγορότερο σε μεγάλες υποδομές, αλλά δεν είναι κατάλληλη για το plan που θα εφαρμόσεις.

Ερώτηση 21.6

Στο AWS υπάρχει το bucket `tfbook-ch21-archive` και θέλεις να μάθεις ποια διεύθυνση του state το διαχειρίζεται. Ποια εντολή χρησιμοποιείς;

a. `terraform show tfbook-ch21-archive`
b. `terraform state show tfbook-ch21-archive`
c. `terraform state list -id=tfbook-ch21-archive`
d. `terraform state pull tfbook-ch21-archive`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `terraform state list -id` κρατά τα resources με το συγκεκριμένο `id` και τυπώνει τις διευθύνσεις τους. Το a ψάχνει αρχείο state ή plan με αυτό το όνομα. Το b θέλει διεύθυνση resource, όχι `id`. Το d αγνοεί το όρισμα και τυπώνει ολόκληρο το state σε JSON.

ΚΕΦΑΛΑΙΟ 22

Import

Άσκηση 22.1 · Ένα topic με block import

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με το `aws` CLI ένα SNS topic με όνομα `tfbook-ch22-ex1`, χωρίς tags. Φέρε το στο Terraform ως `aws_sns_topic.ex1` με block `import`. Έλεγξε με plan ότι η περίληψη λέει μόνο import και μετά κάνε apply.

ΛΥΣΗ · TERMINAL

```
aws sns create-topic --name tfbook-ch22-ex1
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_sns_topic.ex1
  id = "arn:aws:sns:eu-central-1:123456789012:tfbook-ch22-ex1"
}

resource "aws_sns_topic" "ex1" {
  name = "tfbook-ch22-ex1"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to import, 0 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 1 imported, 0 added, 0 changed, 0 destroyed.
```

Το id ενός SNS topic είναι το ARN του, όχι το όνομα. Το ARN το τύπωσε το `create-topic`. Στο Moto ο λογαριασμός είναι πάντα ο `123456789012`. Σε πραγματικό λογαριασμό το βρίσκεις με `aws sns list-topics`. Αν έδινες σκέτο όνομα, ο provider θα απαντούσε ότι δεν μπορεί να το διαβάσει ως ARN. Το plan έδειξε `1 to import` χωρίς `change`, γιατί ούτε το topic ούτε το block έχουν tags.

Άσκηση 22.2 · Τα tags που δεν ήξερες

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με το `aws` CLI το bucket `tfbook-ch22-ex2` με τρία tags, `team=data`, `owner=nikos` και `cost=analytics`. Μετά κάνε πως δεν τα θυμάσαι. Γράψε το block `resource` χωρίς tags μαζί με ένα block `import` και τρέξε plan. Βρες τα tags με το `aws` CLI και διόρθωσε τη configuration μέχρι η περίληψη να λέει μόνο import.

ΛΥΣΗ · TERMINAL

```
aws s3api create-bucket --bucket tfbook-ch22-ex2 \
  --create-bucket-configuration LocationConstraint=eu-central-1
aws s3api put-bucket-tagging --bucket tfbook-ch22-ex2 \
  --tagging 'TagSet=[{Key=team,Value=data},{Key=owner,Value=nikos},{Key=cost,Value=analytics}]'
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.ex2
  id = "tfbook-ch22-ex2"
}

resource "aws_s3_bucket" "ex2" {
  bucket = "tfbook-ch22-ex2"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to import, 0 to add, 1 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
aws s3api get-bucket-tagging --bucket tfbook-ch22-ex2 \
  --query 'TagSet[].[Key,Value]' --output text
```

ΕΞΟΔΟΣ

```
team    data
owner   nikos
cost    analytics
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.ex2
  id = "tfbook-ch22-ex2"
}

resource "aws_s3_bucket" "ex2" {
  bucket = "tfbook-ch22-ex2"

  tags = {
    team  = "data"
    owner = "nikos"
    cost  = "analytics"
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to import, 0 to add, 0 to change, 0 to destroy.
…
```

Το πρώτο plan έλεγε `1 to change`. Ένα block χωρίς `tags` σημαίνει «αυτό το bucket δεν έχει tags», οπότε το apply θα τα έσβηνε και τα τρία. Το `get-bucket-tagging` με `--query` τα τύπωσε ως ζεύγη και η διορθωμένη configuration δίνει plan μόνο με import. Αυτός είναι ο έλεγχος που κάνεις πριν από κάθε apply με import.

Άσκηση 22.3 · Configuration από το Terraform

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με το `aws` CLI το bucket `tfbook-ch22-ex3` με το tag `team=web`. Γράψε μόνο το block `import` προς το `aws_s3_bucket.site` και φτιάξε προσχέδιο με `-generate-config-out`. Μετέφερε το block στο `main.tf`, κράτησε μόνο τα `bucket` και `tags`, σβήσε το παραγόμενο αρχείο και κάνε apply. Το τελευταίο plan πρέπει να μην έχει αλλαγές.

ΛΥΣΗ · TERMINAL

```
aws s3api create-bucket --bucket tfbook-ch22-ex3 \
  --create-bucket-configuration LocationConstraint=eu-central-1
aws s3api put-bucket-tagging --bucket tfbook-ch22-ex3 \
  --tagging 'TagSet=[{Key=team,Value=web}]'
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.site
  id = "tfbook-ch22-ex3"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -generate-config-out=generated.tf
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to import, 0 to add, 0 to change, 0 to destroy.
…
Terraform has generated configuration and written it to generated.tf. Please
…
```

ΛΥΣΗ · TERMINAL

```
cat generated.tf
```

ΕΞΟΔΟΣ

```
# __generated__ by Terraform
# Please review these resources and move them into your main configuration files.

# __generated__ by Terraform from "tfbook-ch22-ex3"
resource "aws_s3_bucket" "site" {
  bucket              = "tfbook-ch22-ex3"
  bucket_namespace    = "global"
  force_destroy       = false
  object_lock_enabled = false
  region              = "eu-central-1"
  tags = {
    team = "web"
  }
  tags_all = {
    team = "web"
  }
}
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.site
  id = "tfbook-ch22-ex3"
}

resource "aws_s3_bucket" "site" {
  bucket = "tfbook-ch22-ex3"

  tags = {
    team = "web"
  }
}
```

ΛΥΣΗ · TERMINAL

```
rm generated.tf
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

Για ένα bucket το προσχέδιο βγαίνει χωρίς σφάλμα, οπότε το plan με τη σημαία πέτυχε και είπε ήδη `1 to import`. Η επικεφαλίδα του block αναφέρει και το id από το οποίο προήλθε. Τα υπόλοιπα arguments του προσχεδίου είτε έχουν τις τιμές που δίνει ούτως ή άλλως ο provider είτε τα υπολογίζει ο ίδιος, όπως τα `region` και `tags_all`. Γι' αυτό φεύγουν χωρίς να αλλάξει το plan. Το `generated.tf` πρέπει να σβηστεί πριν από το apply. Αλλιώς το ίδιο resource θα ήταν γραμμένο σε δύο αρχεία και το Terraform θα σταματούσε με `Duplicate resource`.

Άσκηση 22.4 · Με την εντολή import

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με το `aws` CLI το bucket `tfbook-ch22-ex4`, χωρίς tags. Γράψε το block `resource` με διεύθυνση `aws_s3_bucket.ex4` και φέρε το bucket στο state με την εντολή `terraform import`, χωρίς block `import`. Τύπωσε τη λίστα του state.

ΛΥΣΗ · TERMINAL

```
aws s3api create-bucket --bucket tfbook-ch22-ex4 \
  --create-bucket-configuration LocationConstraint=eu-central-1
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "ex4" {
  bucket = "tfbook-ch22-ex4"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform import aws_s3_bucket.ex4 tfbook-ch22-ex4
```

ΕΞΟΔΟΣ

```
aws_s3_bucket.ex4: Importing from ID "tfbook-ch22-ex4"...
aws_s3_bucket.ex4: Import prepared!
…
Import successful!
…
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
aws_s3_bucket.ex4
```

Η εντολή έγραψε το state αμέσως, χωρίς plan και χωρίς `yes`. Γι' αυτό το `state list` βρίσκει τη διεύθυνση χωρίς να έχει τρέξει κανένα apply. Το block `resource` έπρεπε να υπάρχει πριν, αλλιώς η εντολή σταματά με `does not exist in the configuration`. Το επόμενο βήμα στην πράξη είναι ένα `terraform plan`, για να δεις αν το block ταιριάζει με το bucket.

Άσκηση 22.5 · Τρία parameters με for_each

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με ένα loop του bash τρία SSM parameters τύπου `String`, τα `/tfbook/ch22/ex5/dev`, `/tfbook/ch22/ex5/staging` και `/tfbook/ch22/ex5/prod`, με τιμές `db-dev.internal`, `db-staging.internal` και `db-prod.internal`. Φέρε τα στο Terraform με ένα μόνο block `import` με `for_each` πάνω σε ένα map από local. Το resource λέγεται `aws_ssm_parameter.db_host`. Κάνε apply και τύπωσε τη λίστα του state.

ΛΥΣΗ · TERMINAL

```
for env in dev staging prod; do
  aws ssm put-parameter --name "/tfbook/ch22/ex5/$env" --type String \
    --value "db-$env.internal"
done
```

ΛΥΣΗ · main.tf

```
locals {
  hosts = {
    dev     = "db-dev.internal"
    staging = "db-staging.internal"
    prod    = "db-prod.internal"
  }
}

import {
  for_each = local.hosts
  to       = aws_ssm_parameter.db_host[each.key]
  id       = "/tfbook/ch22/ex5/${each.key}"
}

resource "aws_ssm_parameter" "db_host" {
  for_each = local.hosts

  name  = "/tfbook/ch22/ex5/${each.key}"
  type  = "String"
  value = each.value
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 3 to import, 0 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
aws_ssm_parameter.db_host["dev"]
aws_ssm_parameter.db_host["prod"]
aws_ssm_parameter.db_host["staging"]
```

Το ίδιο map οδηγεί και το block `import` και το resource, οπότε κάθε key δίνει μία διεύθυνση και ένα id που ταιριάζουν. Το `each.value` δίνει στο resource την τιμή του parameter, που το `-generate-config-out` δεν θα έγραφε ποτέ, γιατί είναι sensitive. Το plan είπε `3 to import, 0 to add, 0 to change`. Αν μια τιμή στο map διέφερε από την πραγματική, θα έβλεπες εκεί ένα `1 to change`.

Άσκηση 22.6 · Ένα id με λάθος

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με το `aws` CLI το bucket `tfbook-ch22-ex6-archive`. Γράψε block `import` και block `resource`, αλλά στο `id` γράψε κατά λάθος `tfbook-ch22-ex6-archives`. Τρέξε plan και διάβασε το σφάλμα. Μετά διόρθωσε το `id`. Το αναμενόμενο αποτέλεσμα είναι το σφάλμα.

ΛΥΣΗ · TERMINAL

```
aws s3api create-bucket --bucket tfbook-ch22-ex6-archive \
  --create-bucket-configuration LocationConstraint=eu-central-1
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.archive
  id = "tfbook-ch22-ex6-archives"
}

resource "aws_s3_bucket" "archive" {
  bucket = "tfbook-ch22-ex6-archive"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
╷ Error: Cannot import non-existent remote object
…
│ pre-existing objects can be imported; check that the id is correct and that
│ it is associated with the provider's configured region or endpoint, or use
…
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.archive
  id = "tfbook-ch22-ex6-archive"
}

resource "aws_s3_bucket" "archive" {
  bucket = "tfbook-ch22-ex6-archive"
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to import, 0 to add, 0 to change, 0 to destroy.
…
```

Το import φέρνει μόνο αντικείμενα που ήδη υπάρχουν. Όταν ο provider δεν βρει τίποτα με αυτό το id, το plan σταματά με `Cannot import non-existent remote object` και δεν προτείνει να το φτιάξει. Αυτό είναι σωστό, γιατί ένα λάθος γράμμα δεν πρέπει να γίνει νέο bucket. Το μήνυμα σε συμβουλεύει να ελέγξεις το id και το region του provider, τις δύο πιο συχνές αιτίες.

Άσκηση 22.7 · Import και destroy

Ξεκίνα από αντίγραφο του φακέλου `base`. Φτιάξε με το `aws` CLI το bucket `tfbook-ch22-ex7` και φέρε το στο Terraform με block `import`. Κάνε apply και μετά `terraform destroy -auto-approve`. Έλεγξε με `aws s3api head-bucket` αν το bucket υπάρχει ακόμη.

ΛΥΣΗ · TERMINAL

```
aws s3api create-bucket --bucket tfbook-ch22-ex7 \
  --create-bucket-configuration LocationConstraint=eu-central-1
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.ex7
  id = "tfbook-ch22-ex7"
}

resource "aws_s3_bucket" "ex7" {
  bucket = "tfbook-ch22-ex7"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

ΕΞΟΔΟΣ

```
…
Destroy complete! Resources: 1 destroyed.
```

ΛΥΣΗ · TERMINAL

```
aws s3api head-bucket --bucket tfbook-ch22-ex7
```

ΕΞΟΔΟΣ

```
aws: [ERROR]: An error occurred (404) when calling the HeadBucket operation: Not Found
```

Το bucket το έφτιαξε το CLI, αλλά μετά το import ανήκει στη configuration. Το destroy δεν ξεχωρίζει τι έφτιαξε το ίδιο και τι έφερε με import. Σβήνει ό,τι είναι στο state. Το `head-bucket` απαντά `404`, δηλαδή το bucket δεν υπάρχει πια. Αν ήθελες να σταματήσεις να το διαχειρίζεσαι χωρίς να το σβήσεις, η σωστή κίνηση ήταν το `terraform state rm` του κεφαλαίου 21.

Ερωτήσεις πιστοποίησης

Ερώτηση 22.1

Μια ομάδα θέλει να φέρει στο Terraform δέκα security groups που φτιάχτηκαν από την κονσόλα. Θέλει να δει τι θα γίνει πριν αλλάξει το state και να περάσει η αλλαγή από code review. Τι πρέπει να χρησιμοποιήσει;

a. Την εντολή `terraform import` για κάθε security group, γιατί δείχνει plan πριν γράψει το state.
b. Blocks `import` στη configuration, που εμφανίζονται στο plan και εκτελούνται στο apply.
c. Το `terraform apply -refresh-only`, που προσθέτει στο state όσα βρίσκει στον λογαριασμό.
d. Το `terraform state push` με ένα state που περιέχει τα security groups.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Τα blocks `import` είναι κώδικας που περνά από Git και review. Το plan τα δείχνει ως `will be imported` και το import γίνεται στο apply. Το a είναι λάθος, γιατί η εντολή `terraform import` γράφει το state αμέσως, χωρίς plan. Το c είναι λάθος, γιατί το `-refresh-only` ενημερώνει μόνο resources που είναι ήδη στο state και δεν προσθέτει ποτέ νέα. Το d θα σήμαινε να γράψεις state με το χέρι, που είναι επικίνδυνο και δεν παρουσιάζει τίποτα για review.

Ερώτηση 22.2

Τρέχεις `terraform import aws_instance.web i-0abc12345def67890` και παίρνεις το παρακάτω σφάλμα. Τι λείπει;

```
resource address "aws_instance.web" does not exist in the configuration
```

a. Ένα block `import` με `to = aws_instance.web`.
b. Η σημαία `-generate-config-out` στην εντολή `terraform import`.
c. Ένα block `resource "aws_instance" "web"` στη configuration.
d. Ένα `terraform init -upgrade` για νεότερο provider.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Η εντολή `terraform import` γράφει μόνο το state. Το block `resource` πρέπει να υπάρχει πριν από αυτήν. Το a δεν χρειάζεται, γιατί η εντολή και το block `import` είναι δύο ανεξάρτητοι τρόποι. Το b είναι λάθος, γιατί το `-generate-config-out` είναι σημαία του `terraform plan` και δουλεύει μόνο με blocks `import`. Το d δεν σχετίζεται, γιατί το σφάλμα αφορά τη configuration και όχι τον provider.

Ερώτηση 22.3

Πού βρίσκεις τη μορφή του id που δέχεται ένα συγκεκριμένο resource type για import;

a. Στην τεκμηρίωση του resource στον provider, στην ενότητα Import.
b. Στο `terraform.tfstate` του φακέλου εργασίας.
c. Στην έξοδο της εντολής `terraform providers`.
d. Πουθενά, γιατί είναι πάντα το όνομα του block `resource`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Κάθε resource type ορίζει το δικό του id και η τεκμηρίωσή του το περιγράφει στην ενότητα Import. Το b είναι λάθος, γιατί πριν από το import το αντικείμενο δεν είναι στο state. Το c τυπώνει μόνο ποιους providers χρειάζεται η configuration. Το d είναι λάθος, γιατί το id ανήκει στο πραγματικό αντικείμενο. Για ένα bucket είναι το όνομά του, για ένα SNS topic το ARN και για έναν server του EC2 το instance ID.

Ερώτηση 22.4

Ποιες δύο προτάσεις για το `terraform plan -generate-config-out=generated.tf` ισχύουν; Επίλεξε δύο.

a. Γράφει blocks `resource` μόνο για τα blocks `import` που δεν έχουν ήδη block `resource`.
b. Σταματά με σφάλμα αν το `generated.tf` υπάρχει ήδη.
c. Κάνει το import στο state χωρίς apply.
d. Παράγει πάντα configuration που δίνει plan χωρίς σφάλματα.
e. Βρίσκει μόνο του όλα τα resources του λογαριασμού, χωρίς blocks `import`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b

Η σημαία γράφει configuration μόνο για στόχους `import` χωρίς block και μόνο σε νέο αρχείο, ώστε να μη χαθούν διορθώσεις σου. Το c είναι λάθος, γιατί πρόκειται για plan και το import γίνεται στο apply. Το d είναι λάθος. Είδες στο κεφάλαιο ένα SSM parameter όπου η sensitive τιμή έλειπε και το plan απέτυχε. Το e είναι λάθος, γιατί το Terraform δεν ψάχνει τον λογαριασμό. Του λες εσύ τι να φέρει με blocks `import`.

Ερώτηση 22.5

Σωστό ή λάθος; Το import αλλάζει το πραγματικό αντικείμενο ώστε να ταιριάζει με τη configuration.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το import μόνο διαβάζει το αντικείμενο και γράφει το state. Αν η configuration διαφέρει, το plan δείχνει επιπλέον ένα κανονικό update, για παράδειγμα `1 to import, 0 to add, 1 to change`. Αυτή η αλλαγή είναι χωριστή ενέργεια που φαίνεται από πριν και μπορείς να την αποφύγεις διορθώνοντας τη configuration.

Ερώτηση 22.6

Ένας συνάδελφος διαχειρίζεται ένα S3 bucket με τη δική του configuration και το δικό του state. Κάνεις import το ίδιο bucket στη δική σου configuration, που έχει άλλο state. Τι συμβαίνει;

a. Το Terraform αρνείται με `Resource already managed by Terraform`.
b. Το bucket μεταφέρεται από το state του συναδέλφου στο δικό σου.
c. Το Terraform κλειδώνει το bucket ώστε να το αλλάζει μόνο το ένα state.
d. Το import πετυχαίνει και οι δύο configurations διαχειρίζονται πια το ίδιο bucket, οπότε μπορούν να αναιρούν η μία τις αλλαγές της άλλης ή να το σβήσει η μία.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Κάθε state ξέρει μόνο τον εαυτό του, οπότε το import πετυχαίνει χωρίς προειδοποίηση. Το a ισχύει μόνο για διεύθυνση που υπάρχει ήδη στο ίδιο state. Το b είναι λάθος, γιατί το import δεν αγγίζει άλλο state. Το c είναι λάθος, γιατί το lock προστατεύει ένα state από ταυτόχρονες εντολές και όχι ένα αντικείμενο από δύο states.

ΚΕΦΑΛΑΙΟ 23

Refactoring με moved και removed

Άσκηση 23.1 · Μετονομασία με moved

Ξεκίνα από αντίγραφο του `base`. Γράψε στο `main.tf` ένα `aws_s3_bucket` με όνομα `pics` και bucket `tfbook-ch23-ex1`. Κάνε apply. Μετά μετονόμασε το resource σε `images` χωρίς να σβηστεί το bucket και τρέξε plan.

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "pics" {
  bucket = "tfbook-ch23-ex1"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "images" {
  bucket = "tfbook-ch23-ex1"
}

moved {
  from = aws_s3_bucket.pics
  to   = aws_s3_bucket.images
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.pics has moved to aws_s3_bucket.images
…
Plan: 0 to add, 0 to change, 0 to destroy.
…
```

Το resource block έχει μόνο το νέο όνομα και το `moved` λέει από πού έρχεται. Χωρίς το `moved` η περίληψη θα ήταν `1 to add` και `1 to destroy`. Αν κρατούσες και το παλιό block `pics`, το plan θα σταματούσε με `Moved object still exists`.

Άσκηση 23.2 · Μετονομασία και αλλαγή μαζί

Ξεκίνα από αντίγραφο του `base`. Φτιάξε το bucket `tfbook-ch23-ex2` με όνομα resource `data` και κάνε apply. Μετά, στην ίδια αλλαγή, μετονόμασε το resource σε `raw` και πρόσθεσε το tag `Stage` με τιμή `raw`. Τι λέει το plan;

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "data" {
  bucket = "tfbook-ch23-ex2"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "raw" {
  bucket = "tfbook-ch23-ex2"

  tags = {
    Stage = "raw"
  }
}

moved {
  from = aws_s3_bucket.data
  to   = aws_s3_bucket.raw
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.raw will be updated in-place
  # (moved from aws_s3_bucket.data)
…
Plan: 0 to add, 1 to change, 0 to destroy.
…
```

Το Terraform κάνει πρώτα τη μεταφορά και μετά συγκρίνει το bucket με τη configuration. Η μόνη διαφορά είναι τα tags, που αλλάζουν επί τόπου. Γι' αυτό βλέπεις `updated in-place` με τη σημείωση `moved from` και περίληψη `1 to change`. Η παγίδα είναι να περιμένεις αντικατάσταση επειδή άλλαξαν δύο πράγματα μαζί. Το όνομα στον κώδικα δεν είναι argument του bucket.

Άσκηση 23.3 · Ένα γράμμα λάθος

Ξεκίνα από αντίγραφο του `base`. Φτιάξε το bucket `tfbook-ch23-ex3` με όνομα resource `old` και κάνε apply. Ένας συνάδελφος το μετονομάζει σε `current` και γράφει το παρακάτω block.

HCL

```
moved {
  from = aws_s3_bucket.old
  to   = aws_s3_bucket.curent
}
```

Γράψε την αλλαγή του ακριβώς όπως είναι, τρέξε plan χωρίς apply και βρες στην έξοδο τι θα γινόταν. Μετά διόρθωσε το λάθος.

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "old" {
  bucket = "tfbook-ch23-ex3"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "current" {
  bucket = "tfbook-ch23-ex3"
}

moved {
  from = aws_s3_bucket.old
  to   = aws_s3_bucket.curent
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.curent will be destroyed
  # (because aws_s3_bucket.old was moved to aws_s3_bucket.curent, which is not in configuration)
  # (moved from aws_s3_bucket.old)
…
  # aws_s3_bucket.current will be created
…
Plan: 1 to add, 0 to change, 1 to destroy.
…
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "current" {
  bucket = "tfbook-ch23-ex3"
}

moved {
  from = aws_s3_bucket.old
  to   = aws_s3_bucket.current
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.old has moved to aws_s3_bucket.current
…
Plan: 0 to add, 0 to change, 0 to destroy.
…
```

Το `moved` δεν ελέγχει αν το `to` υπάρχει στη configuration. Η μεταφορά έγινε στο `aws_s3_bucket.curent` και, αφού αυτό δεν είναι δηλωμένο, το plan το σβήνει και φτιάχνει από την αρχή το `current`. Η αιτιολογία το λέει καθαρά, `which is not in configuration`. Η περίληψη `1 to add` και `1 to destroy` είναι το σήμα ότι το refactoring έχει λάθος. Μετά τη διόρθωση η περίληψη γίνεται `0 to add, 0 to change, 0 to destroy`.

Άσκηση 23.4 · Δύο resources σε ένα for_each

Ξεκίνα από αντίγραφο του `base`. Φτιάξε δύο resources, το `red` με bucket `tfbook-ch23-ex4-red` και το `blue` με bucket `tfbook-ch23-ex4-blue`. Κάνε apply. Μετά αντικατάστησέ τα με ένα resource `color` που φτιάχνει τα ίδια δύο buckets με `for_each`. Το plan πρέπει να μην αλλάζει τίποτα στο AWS.

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "red" {
  bucket = "tfbook-ch23-ex4-red"
}

resource "aws_s3_bucket" "blue" {
  bucket = "tfbook-ch23-ex4-blue"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "color" {
  for_each = toset(["red", "blue"])
  bucket   = "tfbook-ch23-ex4-${each.key}"
}

moved {
  from = aws_s3_bucket.red
  to   = aws_s3_bucket.color["red"]
}

moved {
  from = aws_s3_bucket.blue
  to   = aws_s3_bucket.color["blue"]
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # aws_s3_bucket.blue has moved to aws_s3_bucket.color["blue"]
…
  # aws_s3_bucket.red has moved to aws_s3_bucket.color["red"]
…
Plan: 0 to add, 0 to change, 0 to destroy.
…
```

Το `from` και το `to` δεν χρειάζεται να έχουν το ίδιο σχήμα. Ένα resource χωρίς index μπορεί να γίνει instance ενός `for_each`. Κάθε κλειδί πρέπει να βγάζει το ίδιο όνομα bucket που είχε το αντίστοιχο resource. Αν το `each.key` έδινε άλλο όνομα, η μεταφορά θα γινόταν αλλά το plan θα πρότεινε αντικατάσταση, γιατί το `bucket` δεν αλλάζει επί τόπου.

Άσκηση 23.5 · moved μέσα σε module

Ξεκίνα από αντίγραφο του `base`. Γράψε το module `modules/box` με μία μεταβλητή `name` και ένα `aws_s3_bucket` με όνομα `main`. Κάλεσέ το από το root module ως `box` με bucket `tfbook-ch23-ex5` και κάνε apply. Μετά, ως συγγραφέας του module, μετονόμασε το resource σε `this`. Βάλε το `moved` μέσα στο module, ώστε οι χρήστες του να μη χρειαστεί να κάνουν τίποτα.

ΛΥΣΗ · modules/box/main.tf

```
variable "name" {
  type = string
}

resource "aws_s3_bucket" "main" {
  bucket = var.name
}
```

ΛΥΣΗ · main.tf

```
module "box" {
  source = "./modules/box"
  name   = "tfbook-ch23-ex5"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · modules/box/main.tf

```
variable "name" {
  type = string
}

resource "aws_s3_bucket" "this" {
  bucket = var.name
}

moved {
  from = aws_s3_bucket.main
  to   = aws_s3_bucket.this
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # module.box.aws_s3_bucket.main has moved to module.box.aws_s3_bucket.this
…
Plan: 0 to add, 0 to change, 0 to destroy.
…
```

Μέσα στο module οι διευθύνσεις είναι σχετικές, `aws_s3_bucket.main` και `aws_s3_bucket.this`. Το Terraform προσθέτει μόνο του το `module.box.`, οπότε το ίδιο block δουλεύει για κάθε χρήστη, όποιο όνομα κι αν έχει δώσει στο module block. Το root module δεν άλλαξε καθόλου. Γι' αυτό σε ένα κοινό module τα blocks `moved` μένουν, ώστε κάθε χρήστης να παίρνει τη μεταφορά όταν αναβαθμίσει.

Άσκηση 23.6 · Ένα instance εκτός διαχείρισης

Ξεκίνα από αντίγραφο του `base`. Φτιάξε με `for_each` πάνω στο `toset(["a", "b"])` δύο buckets, `tfbook-ch23-ex6-a` και `tfbook-ch23-ex6-b`, σε ένα resource με όνομα `team`. Κάνε apply. Θέλεις να βγει από τη διαχείριση μόνο το bucket του κλειδιού `b`, χωρίς να σβηστεί. Δοκίμασε πρώτα ένα `removed` για το `aws_s3_bucket.team["b"]` και τρέξε `terraform validate`. Μετά βρες τον τρόπο που δουλεύει. Στο τέλος δείξε με `terraform state list` και `aws s3 ls` ότι το state κρατά μόνο το κλειδί `a` και ότι υπάρχουν ακόμη και τα δύο buckets.

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "team" {
  for_each = toset(["a", "b"])
  bucket   = "tfbook-ch23-ex6-${each.key}"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "team" {
  for_each = toset(["a"])
  bucket   = "tfbook-ch23-ex6-${each.key}"
}

removed {
  from = aws_s3_bucket.team["b"]

  lifecycle {
    destroy = false
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
…
│ Error: Resource instance keys not allowed
…
│ Resource address must be a resource (e.g. "test_instance.foo"), not a
│ resource instance (e.g. "test_instance.foo[1]").
…
```

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "team" {
  for_each = toset(["a"])
  bucket   = "tfbook-ch23-ex6-${each.key}"
}
```

ΛΥΣΗ · TERMINAL

```
terraform state rm 'aws_s3_bucket.team["b"]'
```

ΕΞΟΔΟΣ

```
Removed aws_s3_bucket.team["b"]
Successfully removed 1 resource instance(s).
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

ΛΥΣΗ · TERMINAL

```
terraform state list
aws s3 ls
```

ΕΞΟΔΟΣ

```
aws_s3_bucket.team["a"]
…
2026-10-08 05:37:00 tfbook-ch23-ex6-a
2026-10-08 05:37:00 tfbook-ch23-ex6-b
```

Το `removed` δουλεύει μόνο για ολόκληρο resource ή module, οπότε το `validate` απορρίπτει το κλειδί. Για ένα μόνο instance ο δρόμος είναι το `terraform state rm` του κεφαλαίου 21, μαζί με την αφαίρεση του κλειδιού από το set. Η σειρά μετρά. Αν έκανες apply με το νέο set πριν από το `state rm`, το Terraform θα έσβηνε το bucket `b`. Στο τέλος το plan δεν έχει αλλαγές, το state έχει μόνο το `aws_s3_bucket.team["a"]` και το `aws s3 ls` δείχνει και τα δύο buckets.

Άσκηση 23.7 · Παράδοση σε άλλη configuration

Ξεκίνα από αντίγραφο του `base`. Φτιάξε το bucket `tfbook-ch23-ex7` με όνομα resource `handoff` και κάνε apply. Το bucket περνά σε άλλη ομάδα, που το θέλει στη δική της configuration στον υποφάκελο `other` με όνομα resource `received`. Βγάλε το bucket από αυτή τη configuration χωρίς να σβηστεί και φέρε το στη νέα με block `import` του κεφαλαίου 22. Το αναμενόμενο αποτέλεσμα είναι η περίληψη του plan στο `other`.

ΛΥΣΗ · main.tf

```
resource "aws_s3_bucket" "handoff" {
  bucket = "tfbook-ch23-ex7"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
removed {
  from = aws_s3_bucket.handoff

  lifecycle {
    destroy = false
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

ΛΥΣΗ · TERMINAL

```
mkdir other
cp providers.tf .terraform.lock.hcl other/
cd other
```

ΛΥΣΗ · main.tf

```
import {
  to = aws_s3_bucket.received
  id = "tfbook-ch23-ex7"
}

resource "aws_s3_bucket" "received" {
  bucket = "tfbook-ch23-ex7"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to import, 0 to add, 0 to change, 0 to destroy.
…
```

Το `moved` δεν περνά ανάμεσα σε δύο states, οπότε η παράδοση γίνεται σε δύο βήματα. Το `removed` με `destroy = false` βγάζει το bucket από το πρώτο state και το `import` το γράφει στο δεύτερο. Η περίληψη `1 to import` χωρίς `to add` ή `to destroy` λέει ότι το bucket περνά όπως είναι. Η σειρά μετρά. Αν έκανες πρώτα το import, για λίγο θα το διαχειρίζονταν δύο states και ένα destroy σε οποιοδήποτε από τα δύο θα το έσβηνε.

Ερωτήσεις πιστοποίησης

Ερώτηση 23.1

Μετονομάζεις το resource `aws_instance.web` σε `aws_instance.app`. Θέλεις να μη γίνει αντικατάσταση και η αλλαγή να φαίνεται στο plan και στο code review. Τι κάνεις;

a. Τρέχεις `terraform state mv aws_instance.web aws_instance.app` πριν από το commit.
b. Προσθέτεις block `moved` με `from = aws_instance.web` και `to = aws_instance.app`.
c. Τρέχεις `terraform apply -replace=aws_instance.app`.
d. Τρέχεις `terraform import` για το `aws_instance.app` με το ID του server.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `moved` είναι κώδικας. Περνά από review και η μεταφορά φαίνεται στο plan πριν γίνει. Το `state mv` πετυχαίνει την ίδια μεταφορά, αλλά αλλάζει το state αμέσως χωρίς plan και δεν αφήνει ίχνος στον κώδικα. Το `-replace` κάνει ακριβώς ό,τι θέλεις να αποφύγεις, αντικατάσταση. Το `import` είναι για αντικείμενα που δεν βρίσκονται ακόμη στο state. Ο server είναι ήδη εκεί με την παλιά διεύθυνση, που το plan θα ήθελε να σβήσει.

Ερώτηση 23.2

Ένα block `removed` έχει `from = aws_s3_bucket.logs` και `lifecycle { destroy = false }`. Ποιες δύο προτάσεις είναι σωστές; Επίλεξε δύο.

a. Μετά το apply το bucket υπάρχει ακόμη στο AWS.
b. Το `aws_s3_bucket.logs` βγαίνει από το state μόλις τρέξεις `terraform plan`.
c. Το resource block `aws_s3_bucket.logs` πρέπει να έχει αφαιρεθεί από τη configuration.
d. Το `from` μπορεί να είναι και ένα instance, όπως `aws_s3_bucket.logs["eu"]`.
e. Η περίληψη του plan γράφει `1 to destroy`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Το `destroy = false` σημαίνει ότι το Terraform ξεχνά το bucket χωρίς να το σβήσει. Το resource block πρέπει να λείπει, αλλιώς το plan σταματά με `Removed resource still exists`. Το plan δεν γράφει state. Η αφαίρεση γίνεται στο apply, οπότε το b είναι λάθος. Το `from` δέχεται μόνο ολόκληρο resource ή module, οπότε το d είναι λάθος. Η περίληψη δεν μετρά τα αντικείμενα που ξεχνιούνται και γράφει `0 to destroy`, οπότε το e είναι λάθος.

Ερώτηση 23.3

Μετά από ένα refactoring το plan δείχνει τα παρακάτω.

```
ΕΞΟΔΟΣ

# aws_s3_bucket.nwe will be destroyed
# (because aws_s3_bucket.old was moved to aws_s3_bucket.nwe, which is not in
configuration)
Plan: 1 to add, 0 to change, 1 to destroy.
```

Ποια είναι η πιο πιθανή αιτία;

a. Το resource block έχει ακόμη το παλιό όνομα `old`.
b. Πρέπει πρώτα να τρέξεις `terraform state mv`.
c. Το `to` του block `moved` έχει λάθος στο όνομα.
d. Ο AWS provider δεν υποστηρίζει μεταφορές για S3 buckets.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Η αιτιολογία λέει ότι η μεταφορά έγινε σε μια διεύθυνση που δεν υπάρχει στη configuration. Αυτό συμβαίνει όταν το `to` δεν ταιριάζει με κανένα resource block. Αν το block είχε ακόμη το όνομα `old`, το plan θα σταματούσε με `Moved object still exists`. Το `state mv` δεν χρειάζεται, γιατί το `moved` κάνει την ίδια δουλειά. Το `moved` αλλάζει μόνο το state, οπότε δεν εξαρτάται από τον provider για αλλαγή διεύθυνσης στον ίδιο τύπο resource.

Ερώτηση 23.4

Σωστό ή λάθος; Ο συγγραφέας ενός module που είναι δημοσιευμένο στο Registry πρέπει να σβήνει τα blocks `moved` στην αμέσως επόμενη έκδοση, αφού η μεταφορά έχει ήδη γίνει.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ο συγγραφέας δεν ξέρει πότε θα αναβαθμίσει κάθε χρήστης. Κάποιος που περνά από πολύ παλιά έκδοση έχει ακόμη τις παλιές διευθύνσεις στο state του. Αν τα blocks λείπουν, το plan του θα σβήσει και θα ξαναφτιάξει τα resources. Σε κοινά modules τα blocks `moved` μένουν. Σε root module με ένα state σβήνονται άφοβα μετά το apply.

Ερώτηση 23.5

Η ομάδα σου τρέχει τα plans και τα applies σε HCP Terraform με VCS workflow και δεν έχει πρόσβαση στο state από τους υπολογιστές της. Ένα workspace πρέπει να πάψει να διαχειρίζεται ένα S3 bucket, χωρίς να το σβήσει. Τι κάνεις;

a. Σβήνεις το resource block και κάνεις merge.
b. Τρέχεις `terraform state rm` στον υπολογιστή σου.
c. Προσθέτεις `prevent_destroy = true` στο `lifecycle` του resource.
d. Αντικαθιστάς το resource block με block `removed` και `lifecycle { destroy = false }`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `removed` είναι κώδικας. Περνά από το VCS, εφαρμόζεται μέσα στο run και αφήνει το bucket στη θέση του. Αν σβήσεις απλώς το block, το plan θα σβήσει το bucket. Το `state rm` χρειάζεται πρόσβαση εγγραφής στο state, που η ομάδα δεν έχει. Το `prevent_destroy` εμποδίζει τη διαγραφή αλλά κρατά το bucket υπό διαχείριση, που είναι το αντίθετο από αυτό που ζητάς.

Ερώτηση 23.6

Ένα resource έχει `count = 2`. Το αλλάζεις σε `for_each` με τα κλειδιά `"logs"` και `"backups"`. Πώς αποφεύγεις την αντικατάσταση των δύο buckets;

a. Με ένα block `moved` για κάθε instance, από `[0]` και `[1]` στο αντίστοιχο κλειδί.
b. Με ένα block `moved` από `aws_s3_bucket.b` σε `aws_s3_bucket.b`.
c. Δεν χρειάζεται τίποτα. Το Terraform αντιστοιχίζει μόνο του θέσεις σε κλειδιά.
d. Με `terraform apply -replace` για κάθε instance.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το Terraform δεν μπορεί να μαντέψει ποιο κλειδί αντιστοιχεί σε ποια θέση, οπότε χρειάζεται ένα `moved` για κάθε instance. Αυτόματη μεταφορά γίνεται μόνο ανάμεσα σε διεύθυνση χωρίς index και στο `[0]`, όταν προσθέτεις ή αφαιρείς `count`. Για `for_each` δεν γίνεται ποτέ, οπότε το c είναι λάθος. Ένα `moved` με ίδιο `from` και `to` δεν περιγράφει καμία μεταφορά και το plan σταματά με σφάλμα `Redundant move statement`. Το `-replace` ζητά ακριβώς την αντικατάσταση που θέλεις να αποφύγεις.

ΚΕΦΑΛΑΙΟ 24

Secrets, sensitive, ephemeral και Vault

Άσκηση 24.1 · Ένα output με κωδικό

Γράψε configuration χωρίς providers με μια μεταβλητή `smtp_password` τύπου `string`, sensitive και χωρίς `default`. Ένα output `smtp_url` δίνει το string `smtp://mailer:ΚΩΔΙΚΟΣ@mail.internal:587` με τον κωδικό στη θέση του `ΚΩΔΙΚΟΣ`. Δώσε τον κωδικό `Tiropita-31` με `TF_VAR_smtp_password`. Τρέξε πρώτα plan χωρίς `sensitive` στο output και διάβασε το σφάλμα. Διόρθωσέ το, κάνε apply και δείξε τη λίστα των outputs. Τύπωσε μετά την τιμή με `-raw`.

ΛΥΣΗ · main.tf

```
variable "smtp_password" {
  type      = string
  sensitive = true
}

output "smtp_url" {
  value = "smtp://mailer:${var.smtp_password}@mail.internal:587"
}
```

ΛΥΣΗ · TERMINAL

```
export TF_VAR_smtp_password='Tiropita-31'
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
╷
│ Error: Output refers to sensitive values
╵
│     sensitive = true
╵
```

ΛΥΣΗ · main.tf

```
variable "smtp_password" {
  type      = string
  sensitive = true
}

output "smtp_url" {
  value     = "smtp://mailer:${var.smtp_password}@mail.internal:587"
  sensitive = true
}
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output
```

ΕΞΟΔΟΣ

```
smtp_url = <sensitive>
```

ΛΥΣΗ · TERMINAL

```
terraform output -raw smtp_url
```

ΕΞΟΔΟΣ

```
smtp://mailer:Tiropita-31@mail.internal:587
```

Το output χρησιμοποιεί τη sensitive μεταβλητή μέσα σε string template και η ιδιότητα sensitive περνά στο αποτέλεσμα. Χωρίς `sensitive = true` το Terraform αρνείται το output ήδη στο plan. Με το argument η λίστα γράφει `<sensitive>`, ενώ το `-raw` τυπώνει ολόκληρο το URL. Η παγίδα είναι να θεωρήσεις ότι το `<sensitive>` προστατεύει την τιμή. Όποιος μπορεί να τρέξει `terraform output` στον φάκελο τη διαβάζει.

Άσκηση 24.2 · Τι θυμάται το plan file

Γράψε configuration χωρίς providers με μια sensitive μεταβλητή `api_token` και ένα `terraform_data` με όνομα `deploy`, που έχει `input` την τιμή της. Δώσε το token `tok-Spanakopita-5` με `TF_VAR_api_token`. Αποθήκευσε το plan στο `deploy.tfplan` και βρες το token μέσα στο αρχείο με `terraform show -json` και `jq`, χωρίς να κάνεις apply.

ΛΥΣΗ · main.tf

```
variable "api_token" {
  type      = string
  sensitive = true
}

resource "terraform_data" "deploy" {
  input = var.api_token
}
```

ΛΥΣΗ · TERMINAL

```
export TF_VAR_api_token='tok-Spanakopita-5'
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -out=deploy.tfplan
```

ΕΞΟΔΟΣ

```
…
      + input  = (sensitive value)
…
Plan: 1 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform show -json deploy.tfplan | jq -r '.variables.api_token.value'
```

ΕΞΟΔΟΣ

```
tok-Spanakopita-5
```

Το plan στην οθόνη γράφει `input = (sensitive value)`. Το αρχείο όμως κρατά τις τιμές των μεταβλητών, για να μπορεί το apply του αρχείου να τρέξει χωρίς να τις ζητήσει ξανά. Το `terraform show -json` τις δίνει αυτούσιες. Ένα plan file λοιπόν θέλει την ίδια προστασία με το state, ακόμη κι αν δεν έγινε ποτέ apply.

Άσκηση 24.3 · Ό,τι δεν είναι μυστικό

Γράψε configuration χωρίς providers με μια sensitive μεταβλητή `db` τύπου object με δύο attributes `string`, το `user` και το `password`. Δώσε την τιμή με `TF_VAR_db` σε σύνταξη HCL, με user `shop` και password `Koulouri-9`. Γράψε ένα output `db_user` που δείχνει μόνο το user στη λίστα του `terraform output`, όχι ως `<sensitive>`. Κάνε apply.

ΛΥΣΗ · main.tf

```
variable "db" {
  type = object({
    user     = string
    password = string
  })
  sensitive = true
}

output "db_user" {
  value = nonsensitive(var.db.user)
}
```

ΛΥΣΗ · TERMINAL

```
export TF_VAR_db='{ user = "shop", password = "Koulouri-9" }'
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Outputs:

db_user = "shop"
```

Το `sensitive` της μεταβλητής σημαδεύει όλο το object, άρα και το `user`. Ένα output `value = var.db.user` θα έβγαζε το σφάλμα `Output refers to sensitive values`. Το username δεν είναι μυστικό, οπότε η `nonsensitive` είναι σωστή επιλογή. Η παγίδα είναι να τη βάλεις σε όλο το `var.db`, που θα τύπωνε και τον κωδικό. Το `sensitive = true` στο output θα δούλευε κι αυτό, αλλά θα έκρυβε το user χωρίς λόγο.

Άσκηση 24.4 · Κωδικός για το Vault χωρίς state

Γράψε configuration με τους providers `random` 3.9.1 και `vault` 5.12.0. Ένα ephemeral `random_password` με όνομα `redis` φτιάχνει κωδικό 20 χαρακτήρων. Ένα `vault_kv_secret_v2` με όνομα `redis` τον γράφει στο secret `shop/redis` του mount `secret`, στο κλειδί `password`, με write-only argument. Κάνε apply. Δείξε με `terraform state list` τι κράτησε το state και μέτρησε με `curl` και `jq` το μήκος του κωδικού που έφτασε στο Vault.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
    vault = {
      source  = "hashicorp/vault"
      version = "5.12.0"
    }
  }
}

provider "vault" {}

ephemeral "random_password" "redis" {
  length = 20
}

resource "vault_kv_secret_v2" "redis" {
  mount = "secret"
  name  = "shop/redis"
  data_json_wo = jsonencode({
    password = ephemeral.random_password.redis.result
  })
  data_json_wo_version = 1
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform state list
```

ΕΞΟΔΟΣ

```
vault_kv_secret_v2.redis
```

ΛΥΣΗ · TERMINAL

```
curl -s -H "X-Vault-Token: $VAULT_TOKEN" \
  $VAULT_ADDR/v1/secret/data/shop/redis | jq '.data.data.password | length'
```

ΕΞΟΔΟΣ

```
20
```

Ο κωδικός γεννιέται στη μνήμη από το ephemeral resource και περνά στο `data_json_wo`, που είναι write-only. Το state κρατά μόνο το `vault_kv_secret_v2.redis` με το version του. Το Vault έχει τον κωδικό των 20 χαρακτήρων. Η παγίδα είναι το `data_json` χωρίς `_wo`. Δέχεται κι αυτό τιμή, αλλά είναι κανονικό argument, άρα δεν δέχεται ephemeral τιμή. Με ένα `resource "random_password"` θα δούλευε, όμως ο κωδικός θα έμενε στο state.

Άσκηση 24.5 · Αλλαγή κωδικού με ephemeral μεταβλητή

Ο φάκελος της άσκησης ξεκινά ως αντίγραφο του `aws`, ώστε να έχει το lock file και να μην κατεβάσει ξανά τον AWS provider. Άλλαξε το όνομα του secret σε `shop/api-key`. Αντικατάστησε το ephemeral `random_password` με μια ephemeral μεταβλητή `api_key`, που δίνει την τιμή στο `secret_string_wo`, με version 1. Κάνε apply με `TF_VAR_api_key=key-first`. Μετά τρέξε plan με `TF_VAR_api_key=key-second` και δες τι λέει. Κάνε ό,τι χρειάζεται ώστε το Secrets Manager να κρατά το `key-second` και δείξε την τιμή του με το `aws` CLI.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

resource "aws_secretsmanager_secret" "api" {
  name = "shop/api-key"
}
```

ΛΥΣΗ · password.tf

```
variable "api_key" {
  type      = string
  ephemeral = true
}

resource "aws_secretsmanager_secret_version" "api" {
  secret_id               = aws_secretsmanager_secret.api.id
  secret_string_wo        = var.api_key
  secret_string_wo_version = 1
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
TF_VAR_api_key=key-first terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
TF_VAR_api_key=key-second terraform plan
```

ΕΞΟΔΟΣ

```
…
No changes. Your infrastructure matches the configuration.
…
```

ΛΥΣΗ · password.tf

```
variable "api_key" {
  type      = string
  ephemeral = true
}

resource "aws_secretsmanager_secret_version" "api" {
  secret_id               = aws_secretsmanager_secret.api.id
  secret_string_wo        = var.api_key
  secret_string_wo_version = 2
}
```

ΛΥΣΗ · TERMINAL

```
TF_VAR_api_key=key-second terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
aws secretsmanager get-secret-value --secret-id shop/api-key \
  --query SecretString --output text
```

ΕΞΟΔΟΣ

```
key-second
```

Η ephemeral μεταβλητή δεν γράφεται ούτε στο state ούτε στο plan file. Σε αυτή την configuration η μόνη θέση που τη δέχεται είναι το write-only argument. Το plan με το `key-second` λέει `No changes`, γιατί το Terraform δεν κράτησε το `key-first` για να το συγκρίνει. Η αύξηση του `secret_string_wo_version` σε 2 αντικαθιστά το resource και γράφει νέα έκδοση του secret με την τρέχουσα τιμή. Η παγίδα είναι να περιμένεις ότι μια νέα τιμή φτάνει μόνη της στο AWS. Το νέο όνομα του secret χρειάζεται, γιατί το `shop/db-password` υπάρχει ήδη στο Moto από το κεφάλαιο.

Άσκηση 24.6 · Από το Vault στο Secrets Manager

Ο φάκελος ξεκινά πάλι ως αντίγραφο του `aws`. Γράψε με `curl` στο Vault το secret `shop/payments` με κλειδί `api_key` και τιμή `pay-Bougatsa-88`. Άλλαξε την configuration ώστε να χρησιμοποιεί τους providers `aws` και `vault`, να διαβάζει το secret με ephemeral resource και να γράφει την τιμή του `api_key` σε νέο secret του Secrets Manager με όνομα `shop/payments`, με write-only argument. Σβήσε το `password.tf`. Κάνε apply, έλεγξε ότι το state δεν έχει την τιμή και διάβασέ την από το Secrets Manager.

ΛΥΣΗ · TERMINAL

```
curl -s -H "X-Vault-Token: $VAULT_TOKEN" -X POST \
  -d '{"data": {"api_key": "pay-Bougatsa-88"}}' \
  $VAULT_ADDR/v1/secret/data/shop/payments
rm password.tf
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
    vault = {
      source  = "hashicorp/vault"
      version = "5.12.0"
    }
  }
}

provider "aws" {
  region = "eu-central-1"
}

provider "vault" {}

ephemeral "vault_kv_secret_v2" "payments" {
  mount = "secret"
  name  = "shop/payments"
}

resource "aws_secretsmanager_secret" "payments" {
  name = "shop/payments"
}

resource "aws_secretsmanager_secret_version" "payments" {
  secret_id                = aws_secretsmanager_secret.payments.id
  secret_string_wo         = ephemeral.vault_kv_secret_v2.payments.data.api_key
  secret_string_wo_version = 1
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
grep -c Bougatsa terraform.tfstate || true
```

ΕΞΟΔΟΣ

```
0
```

ΛΥΣΗ · TERMINAL

```
aws secretsmanager get-secret-value --secret-id shop/payments \
  --query SecretString --output text
```

ΕΞΟΔΟΣ

```
pay-Bougatsa-88
```

Η τιμή ταξιδεύει από το Vault στο AWS μόνο μέσα στη μνήμη του run. Το ephemeral resource δεν γράφεται στο state και το `secret_string_wo` δεν αποθηκεύει την τιμή, οπότε το `grep` δεν βρίσκει τίποτα. Το `|| true` χρειάζεται επειδή το `grep` επιστρέφει κωδικό αποτυχίας όταν μετρά μηδέν. Η παγίδα είναι το data source `vault_kv_secret_v2`. Θα έδινε κι αυτό την τιμή στο `secret_string_wo`, αλλά θα την αντέγραφε στο state και ο κόπος του write-only argument θα πήγαινε χαμένος.

Ερωτήσεις πιστοποίησης

Ερώτηση 24.1

Μια μεταβλητή `db_password` έχει `sensitive = true` και η τιμή της περνά σε ένα argument ενός resource. Ποια πρόταση είναι σωστή;

a. Το Terraform κρυπτογραφεί την τιμή πριν τη γράψει στο state.
b. Η τιμή δεν γράφεται στο state, μόνο στο plan file.
c. Το plan τη δείχνει ως `(sensitive value)`, αλλά το state την κρατά σε απλό κείμενο.
d. Η τιμή δεν μπορεί να χρησιμοποιηθεί σε συναρτήσεις όπως η `upper`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `sensitive` αλλάζει μόνο την εμφάνιση. Το state κρατά την τιμή σε απλό κείμενο και σημειώνει απλώς ότι πρέπει να κρύβεται. Δεν υπάρχει κρυπτογράφηση από το Terraform, οπότε το a είναι λάθος. Η τιμή γράφεται και στο state και στο plan file, άρα το b είναι λάθος. Οι συναρτήσεις δέχονται sensitive τιμές και δίνουν sensitive αποτέλεσμα, οπότε το d είναι λάθος.

Ερώτηση 24.2

Ένα output `db_password` έχει `sensitive = true` και έχει γίνει apply. Τι τυπώνει η εντολή `terraform output db_password`;

a. `<sensitive>`
b. Την τιμή του output σε εισαγωγικά.
c. Σφάλμα, γιατί ένα sensitive output διαβάζεται μόνο με `-json`.
d. Τίποτα, γιατί τα sensitive outputs δεν γράφονται στο state.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `<sensitive>` εμφανίζεται μόνο στη λίστα όλων των outputs και στο τέλος του apply, άρα το a είναι λάθος. Όταν ζητάς ένα output με το όνομά του, το Terraform θεωρεί ότι το θέλεις και τυπώνει την τιμή. Το ίδιο κάνουν το `-raw` και το `-json`, οπότε το c είναι λάθος. Τα outputs, sensitive ή όχι, γράφονται στο state, οπότε το d είναι λάθος.

Ερώτηση 24.3

Θέλεις ένας κωδικός που δημιουργείται στο run να φτάσει στο API του cloud χωρίς να γραφτεί ποτέ στο state ή στο plan file. Ποιοι δύο μηχανισμοί χρειάζονται μαζί; Επίλεξε δύο.

a. Ένα ephemeral resource που παράγει τον κωδικό.
b. Το `sensitive = true` στο output που δείχνει τον κωδικό.
c. Ένα data source που διαβάζει τον κωδικό από το Vault.
d. Ένα write-only argument που δέχεται τον κωδικό.
e. Η συνάρτηση `nonsensitive` γύρω από τον κωδικό.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και d

Η πηγή πρέπει να είναι ephemeral, ώστε η τιμή να μη μπει στο state. Ο προορισμός πρέπει να είναι write-only, γιατί ένα κανονικό argument δεν δέχεται ephemeral τιμή. Το b κρύβει μόνο από την οθόνη. Το c αντιγράφει τον κωδικό στο state. Το e αφαιρεί και την προστασία της οθόνης.

Ερώτηση 24.4

Σωστό ή λάθος; Όταν διαβάζεις ένα secret από το Vault με data source, το Terraform δεν το αποθηκεύει στο state, γιατί το Vault το κρατά κρυπτογραφημένο.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ό,τι διαβάζει ένα data source γράφεται στο state ως attribute, σε απλό κείμενο. Η κρυπτογράφηση του Vault αφορά μόνο την αποθήκευση μέσα στο Vault. Για να μη φτάσει το secret στο state χρειάζεται ephemeral resource και write-only argument. Αλλιώς το state θέλει την ίδια προστασία με το Vault.

Ερώτηση 24.5

Ποιο είναι το βασικό πλεονέκτημα των dynamic secrets του Vault, όπως τα credentials που δίνει το AWS secrets engine;

a. Φτιάχνονται τη στιγμή που ζητούνται και λήγουν σύντομα, οπότε ένα credential που διέρρευσε παύει γρήγορα να ισχύει.
b. Αποθηκεύονται κρυπτογραφημένα στο state του Terraform.
c. Δεν χρειάζεται κανένα token για να τα ζητήσεις από το Vault.
d. Αλλάζουν αυτόματα τις τιμές σε όλα τα αρχεία `.tfvars` του repository.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Κάθε σετ credentials έχει lease με χρόνο ζωής και το Vault το ακυρώνει όταν λήξει. Το b είναι λάθος, γιατί το Terraform δεν κρυπτογραφεί το state. Αν τα credentials διαβαστούν με data source καταλήγουν εκεί σε απλό κείμενο. Το c είναι λάθος, γιατί κάθε αίτημα στο Vault θέλει token με τα σωστά δικαιώματα. Το d περιγράφει κάτι που δεν κάνει το Vault, που δεν αγγίζει αρχεία στο Git.

Ερώτηση 24.6

Μια ομάδα κρατά το state σε τοπικά αρχεία και τα κάνει commit σε private repository. Το state περιέχει κωδικούς βάσεων. Ποια αλλαγή είναι η σωστή;

a. `sensitive = true` σε όλες τις μεταβλητές, ώστε το state να μην έχει πια κωδικούς.
b. `terraform state rm` στα resources με κωδικούς πριν από κάθε commit.
c. Remote backend με κρυπτογράφηση και περιορισμένη πρόσβαση, το state έξω από το Git και αλλαγή των κωδικών που έμειναν στο ιστορικό.
d. Μετονομασία του `terraform.tfstate` σε `terraform.tfstate.enc` πριν από το commit.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το state ανήκει σε backend με κρυπτογράφηση και έλεγχο πρόσβασης, όχι σε Git. Οι κωδικοί που μπήκαν ήδη στο ιστορικό θεωρούνται διαρρεύσαντες και αλλάζουν. Το a είναι λάθος, γιατί το `sensitive` δεν βγάζει τιμές από το state. Το b κάνει το Terraform να ξεχάσει τα resources, οπότε το επόμενο apply θα προσπαθήσει να τα ξαναφτιάξει. Το d αλλάζει μόνο το όνομα, ενώ το περιεχόμενο μένει απλό κείμενο.

ΚΕΦΑΛΑΙΟ 25

Troubleshooting και logging

Άσκηση 25.1 · Log σε αρχείο

Γράψε configuration με ένα `local_file` που φτιάχνει το `hello.txt`. Τρέξε `terraform plan` έτσι ώστε το terminal να δείχνει μόνο το plan και το log επιπέδου `INFO` να γραφτεί στο `plan.log`. Μετά τύπωσε από το log μόνο το επίπεδο και την έκδοση του Terraform, χωρίς τον χρόνο.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "hello" {
  filename = "hello.txt"
  content  = "Γεια\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
TF_LOG=INFO TF_LOG_PATH=plan.log terraform plan
```

ΕΞΟΔΟΣ

```
…
Plan: 1 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
grep -o "\[INFO\]  Terraform version.*" plan.log
```

ΕΞΟΔΟΣ

```
[INFO]  Terraform version: 1.16.4
```

Η `TF_LOG_PATH` στέλνει το log στο αρχείο, οπότε στο terminal μένει μόνο το plan. Η `TF_LOG` χρειάζεται κι αυτή. Χωρίς αυτήν το `plan.log` θα έμενε άδειο. Η παγίδα είναι να ξανατρέξεις την ίδια εντολή και να βρεις δύο εκτελέσεις στο ίδιο αρχείο, γιατί τα νέα logs προστίθενται στο τέλος. Το `grep -o` κρατά μόνο το κομμάτι της γραμμής που ταιριάζει, άρα πετά τον χρόνο.

Άσκηση 25.2 · Core ή provider

Στην configuration της προηγούμενης άσκησης γράψε δύο logs επιπέδου `TRACE` για το ίδιο plan. Το `core.log` να έχει μόνο τα logs του Terraform και το `provider.log` μόνο του provider. Με `grep -l` δείξε ποιο αρχείο έχει γραμμές `backend/local` και ποιο γραμμές `provider.terraform-provider-local`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "hello" {
  filename = "hello.txt"
  content  = "Γεια\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
TF_LOG_CORE=TRACE TF_LOG_PATH=core.log terraform plan
TF_LOG_PROVIDER=TRACE TF_LOG_PATH=provider.log terraform plan
```

ΛΥΣΗ · TERMINAL

```
grep -l "backend/local" core.log provider.log
grep -l "provider.terraform-provider-local" core.log provider.log
```

ΕΞΟΔΟΣ

```
core.log
provider.log
```

Το `grep -l` τυπώνει μόνο τα ονόματα των αρχείων που ταιριάζουν. Το backend είναι κομμάτι του Terraform, οπότε βρίσκεται μόνο στο `core.log`. Οι γραμμές του plugin βρίσκονται μόνο στο `provider.log`. Η παγίδα είναι να βάλεις και `TF_LOG`. Αυτή ανάβει όλα τα logs, οπότε κάθε αρχείο θα είχε γραμμές και από το Terraform και από τον provider.

Άσκηση 25.3 · Diagnostics για CI

Γράψε το παρακάτω `main.tf` και τρέξε `terraform init`. Μετά γράψε μία εντολή που τυπώνει τον αριθμό των σφαλμάτων και μία που τυπώνει κάθε diagnostic ως `αρχείο:γραμμή περίληψη`, ταξινομημένα κατά γραμμή.

main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "a" {
  filename = "a.txt"
  content  = "a\n"
}

resource "local_file" "b" {
  filename = "b.txt"
  content  = local_file.a.contents
}

resource "local_file" "c" {
  filename = "c.txt"
  content  = local_file.d.content
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate -json | jq .error_count
terraform validate -json | jq -r '.diagnostics | sort_by(.range.start.line)[] |
"\(.range.filename):\(.range.start.line) \(.summary)"'
```

ΕΞΟΔΟΣ

```
2
main.tf:17 Unsupported attribute
main.tf:22 Reference to undeclared resource
```

Το `-json` δίνει τα diagnostics με αρχείο και γραμμή σε πεδία, οπότε δεν χρειάζεται να διαβάσεις τα πλαίσια. Το `sort_by` χρειάζεται, γιατί το Terraform δεν εγγυάται τη σειρά των diagnostics. Το exit code του validate είναι 1, αλλά το pipeline παίρνει το exit code του `jq`. Σε πραγματικό CI ελέγχεις και το `valid`.

Άσκηση 25.4 · Τι θα κάνει το plan

Γράψε configuration με δύο `local_file`, το `a` και το `b`. Με `terraform plan -json` και `jq` τύπωσε για κάθε προγραμματισμένη αλλαγή την ενέργεια και τη διεύθυνση, ταξινομημένες.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "a" {
  filename = "a.txt"
  content  = "a\n"
}

resource "local_file" "b" {
  filename = "b.txt"
  content  = "b\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -json | jq -r 'select(.type == "planned_change") |
"\(.change.action) \(.change.resource.addr)"' | sort
```

ΕΞΟΔΟΣ

```
create local_file.a
create local_file.b
```

Κάθε γραμμή του ρεύματος έχει πεδίο `type`. Το `select` κρατά μόνο τα `planned_change` και από αυτά διαβάζεις την ενέργεια και τη διεύθυνση. Η παγίδα είναι να διαβάσεις το `terraform plan` χωρίς `-json` με `grep`. Η μορφή του για ανθρώπους αλλάζει, ενώ τα πεδία του JSON είναι σταθερά.

Άσκηση 25.5 · Νέος provider χωρίς init

Γράψε configuration με τον provider `local` και ένα `local_file`, τρέξε `init` και κάνε apply. Μετά πρόσθεσε τον provider `random` στο `required_providers` και ένα `random_pet`. Τρέξε `terraform plan`, διόρθωσε το λάθος χωρίς να αλλάξεις έκδοση στον `local` και δείξε με `terraform version` τους providers του φακέλου.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

resource "local_file" "hello" {
  filename = "hello.txt"
  content  = "Γεια\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}

resource "local_file" "hello" {
  filename = "hello.txt"
  content  = "Γεια\n"
}

resource "random_pet" "name" {}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
╷
│ Error: Inconsistent dependency lock file
│
│   - provider registry.terraform.io/hashicorp/random: required by this configuration but no version is selected
…
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform version
```

ΕΞΟΔΟΣ

```
…
+ provider registry.terraform.io/hashicorp/local v2.9.1
+ provider registry.terraform.io/hashicorp/random v3.9.1
```

Το plan σταματά με `Inconsistent dependency lock file`, γιατί το lock file δεν έχει έκδοση για τον `random`. Το σκέτο `init` τον προσθέτει και κρατά την κλειδωμένη έκδοση του `local`. Το `terraform version` το επιβεβαιώνει. Η παγίδα είναι να ακολουθήσεις τυφλά την πρόταση του μηνύματος για `init -upgrade`, που με χαλαρά constraints θα ανέβαζε και τους υπόλοιπους providers.

Άσκηση 25.6 · Ποια εντολή το πιάνει

Γράψε configuration με τον AWS provider, ένα `aws_vpc` με `cidr_block` `10.0.0.0/8` και ένα `aws_subnet` μέσα του με `192.168.1.0/24`. Τρέξε `terraform validate`. Διόρθωσε μόνο ό,τι αναφέρει και κάνε apply. Διόρθωσε ό,τι αναφέρει το apply και ξανακάνε apply. Πόσα resources φτιάχνει το τελευταίο apply;

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.67.0"
    }
  }
}

resource "aws_vpc" "lab" {
  cidr_block = "10.0.0.0/8"
}

resource "aws_subnet" "lab" {
  vpc_id     = aws_vpc.lab.id
  cidr_block = "192.168.1.0/24"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform validate
```

ΕΞΟΔΟΣ

```
…
╷ Error: expected "cidr_block" to contain a network Value with between 16 and 28 significant bits, got: 8
…
```

ΛΥΣΗ · TERMINAL

```
sed -i.bak 's|10.0.0.0/8|10.0.0.0/16|' main.tf
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
╷ Error: creating EC2 Subnet: operation error EC2: CreateSubnet, https response error
StatusCode: 400, RequestID: PgCCARxIbxMLdoMgDKHh89vc5gWTXVPGelqdvl2rMFxj7JIbi5gk, api
error InvalidSubnet.Range: The CIDR '192.168.1.0/24' is invalid.
…
```

ΛΥΣΗ · TERMINAL

```
sed -i.bak 's|192.168.1.0/24|10.0.1.0/24|' main.tf
```

ΛΥΣΗ · TERMINAL

```
terraform apply -auto-approve
```

ΕΞΟΔΟΣ

```
…
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Το validate πιάνει τη μάσκα `/8`, γιατί αυτόν τον έλεγχο τον κάνει ο provider τοπικά. Το subnet έξω από το VPC το πιάνει μόνο το API στο apply, με `InvalidSubnet.Range`. Το πρώτο apply είχε ήδη φτιάξει το VPC, οπότε το τελευταίο φτιάχνει ένα μόνο resource, το subnet. Η παγίδα είναι να θεωρήσεις ότι ένα apply που απέτυχε δεν άφησε τίποτα πίσω του.

Άσκηση 25.7 · Validation σε JSON

Γράψε configuration με μια μεταβλητή `port` τύπου `number` και ένα `local_file` που γράφει τη θύρα στο `app.conf`. Η μεταβλητή να δέχεται μόνο τιμές από 1024 έως 65535, με δικό σου μήνυμα λάθους. Τρέξε `terraform plan` με `-json` και `-var port=80` και τύπωσε με `jq` το `severity`, το `summary` και το `detail` κάθε diagnostic.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "2.9.1"
    }
  }
}

variable "port" {
  type = number

  validation {
    condition     = var.port >= 1024 && var.port <= 65535
    error_message = "Η θύρα πρέπει να είναι από 1024 έως 65535."
  }
}

resource "local_file" "config" {
  filename = "app.conf"
  content  = "port = ${var.port}\n"
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan -json -var port=80 | jq -r 'select(.type == "diagnostic")
| .diagnostic | .severity, .summary, .detail'
```

ΕΞΟΔΟΣ

```
error
Invalid value for variable
Η θύρα πρέπει να είναι από 1024 έως 65535.

This was checked by the validation rule at main.tf:13,3-13.
```

Στο ρεύμα του `plan -json` κάθε σφάλμα είναι μια γραμμή με `type` ίσο με `diagnostic`. Το `detail` έχει το δικό σου `error_message` και τη θέση του κανόνα, όπως και η έξοδος για ανθρώπους. Το validate δεν θα έβρισκε το λάθος, γιατί η τιμή `80` υπάρχει μόνο στη γραμμή εντολών του plan. Η παγίδα είναι να κοιτάξεις μόνο το exit code του `jq`, που είναι 0. Σε ένα pipeline ελέγχεις χωριστά το exit code του Terraform.

Ερωτήσεις πιστοποίησης

Ερώτηση 25.1

Ένα apply αποτυγχάνει με μήνυμα που δεν εξηγεί την αιτία. Ποια τιμή της `TF_LOG` δίνει τα πιο αναλυτικά logs;

a. `DEBUG`
b. `INFO`
c. `TRACE`
d. `VERBOSE`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Τα επίπεδα, από το πιο αναλυτικό στο λιγότερο, είναι `TRACE`, `DEBUG`, `INFO`, `WARN` και `ERROR`. Το `DEBUG` είναι το δεύτερο σε λεπτομέρεια και το `INFO` το τρίτο. Το `VERBOSE` δεν είναι επίπεδο. Μια άγνωστη τιμή ανάβει `TRACE` με προειδοποίηση, αλλά αυτό δεν την κάνει σωστή απάντηση.

Ερώτηση 25.2

Σωστό ή λάθος; Αν ορίσεις μόνο την `TF_LOG_PATH=terraform.log`, το Terraform γράφει αναλυτικά logs σε αυτό το αρχείο.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η `TF_LOG_PATH` λέει μόνο πού θα πάνε τα logs. Τα logs τα ανάβει η `TF_LOG`, ή η `TF_LOG_CORE` και η `TF_LOG_PROVIDER`. Χωρίς αυτές το αρχείο μένει άδειο.

Ερώτηση 25.3

Υποψιάζεσαι ότι ο AWS provider στέλνει λάθος τιμή στο API. Θέλεις τα logs του provider χωρίς τις εκατοντάδες γραμμές του Terraform core. Τι ορίζεις;

a. `TF_LOG_PROVIDER=DEBUG`
b. `TF_LOG=PROVIDER`
c. `TF_LOG_PATH=provider.log`
d. `TF_LOG_CORE=DEBUG`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Η `TF_LOG_PROVIDER` ανάβει μόνο τα logs των providers, με τα ίδια επίπεδα που δέχεται η `TF_LOG`. Το `PROVIDER` δεν είναι επίπεδο της `TF_LOG`. Η `TF_LOG_PATH` ορίζει μόνο το αρχείο. Η `TF_LOG_CORE` ανάβει το αντίθετο από αυτό που θέλεις, τα logs του core.

Ερώτηση 25.4

Ποια δύο προβλήματα δεν εντοπίζει το `terraform validate`; Επίλεξε δύο.

a. Αναφορά σε resource που δεν έχει δηλωθεί.
b. Subnet με CIDR έξω από το εύρος του VPC του.
c. Δύο resources που αναφέρονται το ένα στο άλλο.
d. Χρήστης χωρίς δικαίωμα να δημιουργήσει instances.
e. Block που ανοίγει και δεν κλείνει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Το validate δεν καλεί κανένα API. Τη σχέση των δύο CIDR και τα δικαιώματα τα ξέρει μόνο το AWS, οπότε φαίνονται στο apply. Η αναφορά σε resource που λείπει, ο κύκλος και το συντακτικό λάθος φαίνονται ήδη από τα αρχεία της configuration, οπότε τα πιάνει το validate.

Ερώτηση 25.5

Ένα apply τελειώνει με `Provider produced inconsistent result after apply` και το μήνυμα γράφει `This is a bug in the provider`. Ποια είναι η σωστή κίνηση;

a. Σβήνεις το `terraform.tfstate` και ξανακάνεις apply.
b. Ορίζεις `-lock=false` και ξανακάνεις apply.
c. Ελέγχεις με plan το state και αναφέρεις το bug στο repository του provider.
d. Ανοίγεις issue στο repository του Terraform.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το μήνυμα λέει ότι ο provider επέστρεψε τιμή διαφορετική από αυτή που υποσχέθηκε. Λέει επίσης ότι η αναφορά πάει στον provider. Το plan δείχνει τι πιστεύει τώρα το state. Το a κάνει το Terraform να ξεχάσει όσα φτιάχτηκαν. Το πρόβλημα δεν αφορά το lock, οπότε το b δεν βοηθά. Το d στέλνει το bug σε λάθος ομάδα, γιατί το Terraform απλώς εντόπισε το λάθος του provider.

Ερώτηση 25.6

Ένας συνάδελφος θέλει να επισυνάψει σε δημόσιο issue το log ενός apply με `TF_LOG=DEBUG`. Όλες οι μεταβλητές με κωδικούς είναι `sensitive`. Τι πρέπει να κάνει πρώτα;

a. Να διαβάσει το log και να σβήσει κωδικούς και άλλα ευαίσθητα δεδομένα.
b. Τίποτα, γιατί οι τιμές `sensitive` δεν γράφονται ποτέ στα logs.
c. Να το ξαναβγάλει με `TF_LOG=JSON`, που κρύβει τα μυστικά.
d. Να το ξαναβγάλει με `TF_LOG_PATH`, που κρυπτογραφεί το αρχείο.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το `sensitive` κρύβει τιμές από την έξοδο του plan και του apply, όχι από τα logs. Ένας provider σε `DEBUG` μπορεί να γράψει ολόκληρο το σώμα ενός αιτήματος με τον κωδικό μέσα, οπότε το b είναι λάθος. Το `JSON` αλλάζει μόνο τη μορφή και δίνει μάλιστα επίπεδο `TRACE`. Η `TF_LOG_PATH` γράφει απλό κείμενο.

ΚΕΦΑΛΑΙΟ 26

HCP Terraform

Άσκηση 26.1 · Ένα συγκεκριμένο workspace

Το organization σου στο HCP Terraform λέγεται `acme`. Ο φάκελος `pets` πρέπει να δουλεύει με το workspace `pets-prod` και μόνο με αυτό. Γράψε το block `terraform` του `main.tf`, με το block `cloud` και τον provider `random` στην έκδοση 3.9.1. Η configuration πρέπει να δουλεύει στο `app.terraform.io`.

ΛΥΣΗ · main.tf

```
terraform {
  cloud {
    organization = "acme"

    workspaces {
      name = "pets-prod"
    }
  }

  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}
```

Το `name` δένει τον φάκελο με ένα μόνο workspace. Αν το `pets-prod` δεν υπάρχει, το `terraform init` το δημιουργεί. Το `hostname` λείπει γιατί η προεπιλογή του είναι το `app.terraform.io`. Θα το έγραφες μόνο για Terraform Enterprise. Η configuration δεν έχει block `backend`, γιατί το `cloud` το αποκλείει. Το `cloud` και το `required_providers` είναι αδέλφια μέσα στο ίδιο block `terraform`.

Άσκηση 26.2 · Τρία περιβάλλοντα με tags

Τα workspaces `pets-dev` και `pets-prod` υπάρχουν ήδη στο project `web` του organization `acme` και έχουν το tag `app` με τιμή `pets`. Ο φάκελος `pets` πρέπει να εξυπηρετεί αυτά και ένα νέο `pets-staging`, ενώ οι προγραμματιστές αλλάζουν περιβάλλον με `terraform workspace select`. Γράψε το block `terraform` με το block `cloud`. Γράψε μετά τις εντολές που κάνουν init, φτιάχνουν το `pets-staging` από το CLI και δείχνουν τη λίστα των workspaces.

ΛΥΣΗ · main.tf

```
terraform {
  cloud {
    organization = "acme"

    workspaces {
      project = "web"
      tags = {
        app = "pets"
      }
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform workspace new pets-staging
terraform workspace list
```

Με `tags` ο φάκελος συνδέεται με όλα τα workspaces του project `web` που έχουν το tag `app = "pets"`. Αφού ταιριάζουν ήδη δύο, το `terraform init` σε ρωτά με ποιο θα δουλέψεις. Το `terraform workspace new` φτιάχνει το workspace στο HCP Terraform, μέσα στο project `web` και με το ίδιο tag. Το `terraform workspace list` δείχνει μόνο τα workspaces που ταιριάζουν, όχι όλο το organization. Με `name` αντί για `tags` οι εντολές `terraform workspace` δεν θα διαχειρίζονταν workspaces. Η μορφή map για τα tags θέλει Terraform 1.10 ή νεότερο. Με παλαιότερη έκδοση θα έγραφες `tags = ["pets"]`.

Άσκηση 26.3 · Διόρθωσε το block cloud

Μια συνάδελφος θέλει να μεταφέρει τη configuration στο workspace `pets-dev` του organization `acme`. Έγραψε το παρακάτω `main.tf`, αλλά το `terraform init` αποτυγχάνει. Βρες τα τρία λάθη και γράψε το σωστό `main.tf`.

main.tf

```
variable "org" {
  type    = string
  default = "acme"
}

terraform {
  backend "s3" {
    bucket       = "acme-tfstate"
    key          = "pets/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }

  cloud {
    organization = var.org

    workspaces {
      name = "pets-dev"
      tags = ["pets"]
    }
  }
}
```

ΛΥΣΗ · main.tf

```
terraform {
  cloud {
    organization = "acme"

    workspaces {
      name = "pets-dev"
    }
  }
}
```

Το πρώτο λάθος είναι το block `backend "s3"` δίπλα στο `cloud`. Μια configuration έχει ή το ένα ή το άλλο. Το δεύτερο είναι το `var.org`. Το block `cloud` δεν δέχεται μεταβλητές, όπως και το `backend`, οπότε η μεταβλητή `org` δεν χρειάζεται πια. Αν θέλεις το organization έξω από τον κώδικα, αφαιρείς το `organization` και ορίζεις `TF_CLOUD_ORGANIZATION`. Το τρίτο είναι το `name` μαζί με τα `tags`, που αποκλείονται. Η εκφώνηση ζητά ένα συγκεκριμένο workspace, άρα κρατάς το `name`. Το state που υπήρχε στο S3 το μεταφέρει το επόμενο `terraform init`, αφού ρωτήσει.

Άσκηση 26.4 · Από το private registry

Το private registry του organization `acme` έχει το module `vpc` για τον provider `aws`, με εκδόσεις από 2.0.0 έως 2.3.1. Έχει επίσης τον δικό του provider `internal` στην έκδοση 1.4.0. Γράψε `main.tf` που δηλώνει τον provider με τοπικό όνομα `internal` και ακριβή έκδοση 1.4.0. Το ίδιο αρχείο καλεί το module με όνομα `network` και δέχεται κάθε έκδοση από την 2.3.0 και πάνω, αλλά όχι την 3.0.0. Δώσε στο module την τιμή `10.20.0.0/16` στο input `cidr_block`.

ΛΥΣΗ · main.tf

```
terraform {
  required_providers {
    internal = {
      source  = "app.terraform.io/acme/internal"
      version = "1.4.0"
    }
  }
}

module "network" {
  source  = "app.terraform.io/acme/vpc/aws"
  version = "~> 2.3"

  cidr_block = "10.20.0.0/16"
}
```

Ο provider έχει τρία μέρη, hostname, organization και όνομα. Το module έχει τέσσερα, με το όνομα του provider στο τέλος. Χωρίς το `app.terraform.io` το Terraform θα έψαχνε στο δημόσιο Registry και δεν θα έβρισκε τίποτα. Το `~> 2.3` του κεφαλαίου 5 δέχεται την 2.3.0 και κάθε νεότερη 2.x, αλλά όχι την 3.0.0. Σήμερα θα διαλέξει την 2.3.1. Για να κατεβάσει το init το module από το CLI χρειάζεται πρώτα `terraform login`. Μέσα σε remote run δεν χρειάζεται τίποτα.

Άσκηση 26.5 · Outputs από το network-prod

Το workspace `network-prod` του organization `acme` έχει το output `subnet_id`, που δεν είναι sensitive. Στο workspace `pets-prod` θέλεις ένα resource `terraform_data` με όνομα `server` και `input` την τιμή του `subnet_id`. Γράψε το data source `tfe_outputs` και το resource, ώστε η τιμή να μη γίνει sensitive. Ποια environment variable χρειάζεται το `pets-prod` για να δουλέψει ο provider `tfe` σε remote run;

ΛΥΣΗ · main.tf

```
data "tfe_outputs" "network" {
  organization = "acme"
  workspace    = "network-prod"
}

resource "terraform_data" "server" {
  input = data.tfe_outputs.network.nonsensitive_values.subnet_id
}
```

Το `values` είναι ολόκληρο sensitive, οπότε κάθε τιμή του θα έκανε sensitive και το `input`. Το `nonsensitive_values` κρατά μόνο τα outputs που δεν ήταν sensitive στο `network-prod`, άρα το `subnet_id` εμφανίζεται κανονικά στο plan. Ο provider `tfe` θέλει token. Σε remote run το δίνεις με τη sensitive

environment variable `TFE_TOKEN` στο `pets-prod`. Για το `tfe_outputs` δεν χρειάζεται να ενεργοποιήσεις το remote state sharing στο `network-prod`, όπως για το `terraform_remote_state`, γιατί διαβάζει μόνο outputs. Για να ξανατρέχει το `pets-prod` όταν αλλάζει το δίκτυο, πρόσθεσε το `network-prod` ως run trigger.

Άσκηση 26.6 · Από το S3 στο HCP Terraform

Ο φάκελος `pets` έχει state στο S3, με ένα block σαν αυτά του κεφαλαίου 19.

main.tf

```
terraform {
  backend "s3" {
    bucket       = "acme-tfstate"
    key          = "pets/terraform.tfstate"
    region       = "eu-central-1"
    use_lockfile = true
  }

  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}
```

Μετάφερε το state στο workspace `pets-prod` του project `web` στο organization `acme`. Γράψε το νέο block `terraform` και την εντολή που κάνει τη μεταφορά. Τι απαντάς στην ερώτηση του Terraform;

ΛΥΣΗ · main.tf

```
terraform {
  cloud {
    organization = "acme"

    workspaces {
      project = "web"
      name    = "pets-prod"
    }
  }

  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.9.1"
    }
  }
}
```

ΛΥΣΗ · TERMINAL

```
terraform login
terraform init
```

Το `backend "s3"` φεύγει και στη θέση του μπαίνει το `cloud`. Το `project` συνδυάζεται με το `name`, οπότε το `pets-prod` δημιουργείται στο project `web` αν δεν υπάρχει. Το `terraform init` βλέπει την αλλαγή και ρωτά

`Should Terraform migrate your existing state?`. Απαντάς `yes` και το state του S3 γίνεται η πρώτη έκδοση στο workspace. Μη γράψεις `terraform init -migrate-state`, γιατί το Terraform απορρίπτει αυτή τη σημαία όταν ο προορισμός είναι το HCP Terraform. Το αντικείμενο στο S3 μένει στο bucket, αλλά κανείς δεν το διαβάζει πια. Αν η configuration έχει input variables, πρόσθεσέ τες στο workspace πριν από το πρώτο run.

Ερωτήσεις πιστοποίησης

Ερώτηση 26.1

Μια ομάδα χρησιμοποιεί CLI workspaces με backend S3 για τα περιβάλλοντα dev και prod. Σκέφτεται να περάσει στα workspaces του HCP Terraform. Ποια διαφορά ισχύει;

a. Τα workspaces του HCP Terraform μοιράζονται υποχρεωτικά τις ίδιες μεταβλητές.
b. Τα CLI workspaces έχουν το καθένα δικά του δικαιώματα πρόσβασης.
c. Κάθε workspace του HCP Terraform έχει δικές του μεταβλητές, δικό του state, δικά του δικαιώματα και δικό του ιστορικό runs.
d. Τα workspaces του HCP Terraform είναι προαιρετικά, όπως τα CLI workspaces.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Ένα workspace του HCP Terraform λειτουργεί σαν χωριστός φάκελος εργασίας με όλα τα δικά του. Οι μεταβλητές ορίζονται ανά workspace ή με variable sets, άρα δεν μοιράζονται υποχρεωτικά και το a είναι λάθος. Τα CLI workspaces είναι απλώς χωριστά states στο ίδιο backend, με τα ίδια credentials και χωρίς δικά τους δικαιώματα, οπότε το b είναι λάθος. Στο HCP Terraform δεν διαχειρίζεσαι τίποτα χωρίς workspace, οπότε το d είναι λάθος.

Ερώτηση 26.2

Ένα workspace είναι συνδεδεμένο με το κλαδί `main` ενός repository στο GitHub. Ένας προγραμματιστής ανοίγει pull request προς το `main`. Τι κάνει το HCP Terraform;

a. Κάνει plan και apply με τον κώδικα του pull request.
b. Τρέχει speculative plan και δείχνει το αποτέλεσμα μέσα στο pull request.
c. Τίποτα, μέχρι να γίνει merge.
d. Κλειδώνει το workspace μέχρι να κλείσει το pull request.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Στο VCS-driven workflow κάθε pull request προς το κλαδί του workspace ξεκινά speculative plan και το αποτέλεσμα εμφανίζεται ως έλεγχος στο pull request. Ένα speculative plan δεν γίνεται ποτέ apply, οπότε το a είναι λάθος. Το apply έρχεται μετά το merge. Το c αγνοεί το speculative plan. Το speculative plan δεν περιμένει στην ουρά και δεν κλειδώνει το workspace, άρα το d είναι λάθος.

Ερώτηση 26.3

Ποιο enforcement level του Sentinel σταματά το run όταν η πολιτική αποτύχει, αλλά επιτρέπει σε εξουσιοδοτημένο χρήστη να το συνεχίσει;

a. `advisory`
b. `hard-mandatory`
c. `soft-mandatory`
d. `mandatory`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το soft-mandatory σταματά το run και επιτρέπει override σε χρήστη με το σχετικό δικαίωμα. Το advisory μόνο προειδοποιεί και το run συνεχίζει. Το hard-mandatory σταματά το run και θέλει διόρθωση του κώδικα. Το `mandatory` είναι level του OPA, όχι του Sentinel.

Ερώτηση 26.4

Τρέχεις `terraform login` χωρίς όρισμα. Πού αποθηκεύεται το token;

a. Σε απλό κείμενο στο `~/.terraform.d/credentials.tfrc.json`
b. Στο block `cloud` της configuration
c. Κρυπτογραφημένο στο `terraform.tfstate`
d. Στη μεταβλητή περιβάλλοντος `TF_TOKEN_app_terraform_io`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το `terraform login` γράφει το token σε απλό κείμενο στο `credentials.tfrc.json` του φακέλου `.terraform.d` στο home σου και το λέει πριν ζητήσει επιβεβαίωση. Δεν αλλάζει ποτέ τα αρχεία της configuration, άρα το b είναι λάθος. Το state δεν κρατά credentials του CLI, οπότε το c είναι λάθος. Τη μεταβλητή `TF_TOKEN_app_terraform_io` την ορίζεις εσύ, συνήθως σε pipeline. Το login δεν μπορεί να αλλάξει το περιβάλλον του shell σου, άρα το d είναι λάθος.

Ερώτηση 26.5

Ποιοι δύο τρόποι δίνουν το όνομα του organization στο block `cloud`; Επίλεξε δύο.

a. Το argument `organization` μέσα στο block
b. Η μεταβλητή περιβάλλοντος `TF_CLOUD_ORGANIZATION`
c. Μια input variable, για παράδειγμα `organization = var.org`
d. Ένα local, για παράδειγμα `organization = local.org`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b

Το block `cloud` διαβάζεται από το init πριν υπολογιστεί οποιαδήποτε έκφραση. Γι' αυτό δέχεται μόνο σταθερές τιμές ή τις μεταβλητές περιβάλλοντος `TF_CLOUD_ORGANIZATION`, `TF_CLOUD_PROJECT`, `TF_CLOUD_HOSTNAME` και `TF_WORKSPACE`. Αν υπάρχουν και τα δύο, κερδίζει η τιμή του block. Οι input variables και τα locals είναι named values και το init απαντά `Variables not allowed`, οπότε τα c και d είναι λάθος.

Ερώτηση 26.6

Ένα workspace έχει τη Terraform variable `region` με τιμή `eu-central-1`. Ένα global variable set, χωρίς priority, ορίζει `region` με τιμή `us-east-1`. Κανείς δεν δίνει `-var`. Ποια τιμή χρησιμοποιεί το run;

a. `us-east-1`, γιατί τα variable sets νικούν τις μεταβλητές του workspace.
b. Αυτή που άλλαξε τελευταία.
c. Το run αποτυγχάνει λόγω σύγκρουσης.
d. `eu-central-1`, γιατί η μεταβλητή του workspace νικά τα variable sets.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Η μεταβλητή του ίδιου του workspace είναι πιο ειδική από κάθε variable set χωρίς priority και κερδίζει. Το global set είναι μάλιστα το ασθενέστερο από τα sets. Μόνο ένα priority variable set θα έδινε το a. Η σειρά δεν εξαρτάται από το πότε άλλαξε κάτι, άρα το b είναι λάθος. Η σύγκρουση λύνεται με τη σειρά προτεραιότητας και δεν προκαλεί σφάλμα, άρα το c είναι λάθος.

Ερώτηση 26.7

Ένα workspace έχει execution mode Local. Τι κάνει το HCP Terraform για αυτό;

a. Τρέχει plan και apply στα δικά του μηχανήματα και στέλνει την έξοδο στο CLI.
b. Αποθηκεύει και κλειδώνει το state. Το plan και το apply τρέχουν στο μηχάνημά σου.
c. Εφαρμόζει τις μεταβλητές του workspace στο τοπικό run.
d. Τρέχει τις Sentinel πολιτικές πριν από κάθε τοπικό apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Σε local execution mode το workspace λειτουργεί μόνο ως remote backend για το state. Το a περιγράφει το remote mode. Οι μεταβλητές του workspace και τα variable sets δεν χρησιμοποιούνται σε local mode, άρα το c είναι λάθος. Οι πολιτικές δεν τρέχουν σε local execution mode, οπότε το d είναι λάθος.

Ερώτηση 26.8

Σωστό ή λάθος; Σε workspace συνδεδεμένο με repository VCS μπορείς να τρέξεις `terraform plan` από το CLI, αλλά όχι `terraform apply`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το `terraform plan` από το CLI τρέχει speculative plan, που επιτρέπεται σε κάθε workspace και βοηθά κατά την ανάπτυξη. Το remote apply από το CLI απαγορεύεται σε workspace με VCS, γιατί η πηγή της αλήθειας είναι το repository. Η αλλαγή εφαρμόζεται με merge στο κλαδί του workspace.

Ερώτηση 26.9

Η εταιρεία θέλει το HCP Terraform να διαχειρίζεται servers σε ιδιωτικό datacenter. Το δίκτυο δεν επιτρέπει καμία εισερχόμενη σύνδεση από το internet. Ποια λύση ταιριάζει;

a. Execution mode Local σε όλα τα workspaces
b. Ένα run task σε φάση pre-plan
c. HCP Terraform agents μέσα στο datacenter
d. Dynamic provider credentials

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Οι agents τρέχουν μέσα στο ιδιωτικό δίκτυο, ρωτούν το HCP Terraform για δουλειά με εξερχόμενη σύνδεση και εκτελούν εκεί τα runs. Το Local mode θα έβγαζε την εκτέλεση στους σταθμούς εργασίας και θα έχανε τα remote runs, άρα το a δεν λύνει το πρόβλημα. Ένα run task στέλνει δεδομένα σε εξωτερική υπηρεσία, δεν εκτελεί το Terraform, οπότε το b είναι λάθος. Τα dynamic credentials αφορούν την ταυτοποίηση στον provider, όχι τη δικτυακή πρόσβαση, άρα το d είναι λάθος.

Ερώτηση 26.10

Ποιο είναι το κύριο όφελος των dynamic provider credentials έναντι ενός access key σε sensitive μεταβλητή του workspace;

a. Το access key δεν εμφανίζεται πια στα logs.
b. Δεν χρειάζεται IAM role στο AWS.
c. Το plan τρέχει πιο γρήγορα.
d. Κάθε run παίρνει προσωρινά credentials μέσω OIDC και δεν υπάρχει μακρόβιο κλειδί για ανανέωση ή διαρροή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Με dynamic credentials το HCP Terraform δίνει στο AWS ένα token ταυτότητας OIDC και παίρνει προσωρινά credentials που ισχύουν μόνο για το run. Μια sensitive μεταβλητή ήδη κρύβει την τιμή στο UI, άρα το a δεν είναι το όφελος. Ο μηχανισμός θέλει IAM role με σχέση εμπιστοσύνης, οπότε το b είναι λάθος. Η ταχύτητα του plan δεν επηρεάζεται, άρα το c είναι λάθος.

Ερώτηση 26.11

Τι κάνει η λειτουργία continuous validation των health assessments;

a. Τρέχει apply κάθε 24 ώρες για να διορθώσει το drift.
b. Ελέγχει περιοδικά τα check blocks, τα preconditions και τα postconditions της configuration χωρίς να αλλάξει την υποδομή.
c. Κάνει `terraform validate` σε κάθε commit.
d. Συγκρίνει το κόστος της υποδομής με τον προϋπολογισμό.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το continuous validation ξαναελέγχει τους όρους της configuration, κυρίως τα check blocks, ώστε να μάθεις όταν κάτι χάλασε μετά το apply. Κανένα health assessment δεν αλλάζει την υποδομή, άρα το a είναι λάθος. Το drift το βρίσκει το drift detection και το διορθώνεις εσύ. Το `terraform validate` ελέγχει μόνο τη σύνταξη και δεν είναι μέρος των health assessments, οπότε το c είναι λάθος. Το κόστος είναι δουλειά του cost estimation, άρα το d είναι λάθος.

Ερώτηση 26.12

Ποια είναι η σωστή τιμή του `source` για το module `vpc` του provider `aws`, δημοσιευμένο στο private registry του organization `acme` στο HCP Terraform;

a. `acme/vpc/aws`
b. `app.terraform.io/vpc/aws`
c. `app.terraform.io/acme/vpc/aws`
d. `registry.terraform.io/acme/vpc`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Ένα private module γράφεται ως hostname, organization, όνομα και provider. Το a δεν έχει hostname, οπότε το Terraform θα έψαχνε στο δημόσιο Registry. Το b δεν έχει organization. Το d δείχνει στο δημόσιο Registry και του λείπει ο provider.

ΚΕΦΑΛΑΙΟ 27

Τελικό project

Άσκηση 27.1 · Κανένα site χωρίς υπεύθυνο

Πρόσθεσε στη μεταβλητή `sites` δεύτερο block `validation` που απορρίπτει `owner` κενό ή μόνο με κενά. Δοκίμασέ το με ένα site `news` που έχει `owner = ""`. Το plan πρέπει να σταματήσει με το δικό σου μήνυμα.

ΛΥΣΗ · TERMINAL

```
rm backend.tf
```

ΛΥΣΗ · variables.tf

```
variable "sites" {
  description = "Τα στατικά sites. Το κλειδί είναι το σύντομο όνομα του site."
  type = map(object({
    owner = string
  }))

  validation {
    condition     = alltrue([for name in keys(var.sites) : can(regex("^[a-z0-9]{3,12}$", name))])
    error_message = "Κάθε κλειδί του sites θέλει 3 έως 12 πεζά λατινικά γράμματα ή ψηφία."
  }

  validation {
    condition     = alltrue([for site in values(var.sites) : trimspace(site.owner) != ""])
    error_message = "Κάθε site θέλει owner."
  }
}
```

ΛΥΣΗ · terraform.tfvars

```
sites = {
  docs = { owner = "maria" }
  news = { owner = "" }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
╷
│ Error: Invalid value for variable
…
│ Κάθε site θέλει owner.
╵
```

Η `values` δίνει τα objects της μεταβλητής χωρίς τα κλειδιά. Η `trimspace` σβήνει τα κενά, οπότε και ένα `owner = "  "` αποτυγχάνει. Τα δύο blocks `validation` ελέγχονται χωριστά, όπως στο κεφάλαιο 13. Το καθένα δίνει το δικό του μήνυμα. Το κλειδί `news` περνά τον πρώτο έλεγχο, οπότε το σφάλμα είναι μόνο ένα. Το λάθος σταματά στο plan, πριν φτιαχτεί bucket χωρίς υπεύθυνο.

Άσκηση 27.2 · Τέταρτο site

Πρόσθεσε το site `news` με υπεύθυνο τον `kostas`. Μην κάνεις apply. Τρέξε plan και βρες ποια resources φέρνει η νέα γραμμή του `terraform.tfvars`.

ΛΥΣΗ · TERMINAL

```
rm backend.tf
```

ΛΥΣΗ · terraform.tfvars

```
sites = {
  docs = { owner = "maria" }
  blog = { owner = "nikos" }
  shop = { owner = "eleni" }
  news = { owner = "kostas" }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # module.site["news"].aws_s3_bucket.this will be created
…
  # module.site["news"].aws_s3_bucket_website_configuration.this will be created
…
  # module.site["news"].aws_s3_object.index will be created
…
  # module.site["news"].aws_ssm_parameter.bucket will be created
…
Plan: 16 to add, 0 to change, 0 to destroy.
…
```

Μία γραμμή στη μεταβλητή φέρνει ένα νέο instance του module με τα τέσσερα resources του. Το αντίγραφο δεν έχει state, οπότε το plan σχεδιάζει και τα τέσσερα sites, `16 to add`. Στο project, με τα τρία sites ήδη στο state, το ίδιο plan θα έλεγε `4 to add`. Τα άλλα instances δεν επηρεάζονται, γιατί το `for_each` ταυτοποιεί κάθε instance με το κλειδί του και όχι με τη θέση του.

Άσκηση 27.3 · Μια ξεχασμένη παράμετρος

Παλιότερα υπήρχε το site `old`. Το bucket του σβήστηκε, αλλά κάποιος ξέχασε το parameter του. Φτιάξε το με το `aws ssm put-parameter`, με όνομα `/sites/old/bucket`, τύπο `String` και όποια τιμή θέλεις. Τρέξε plan και εξήγησε τι βλέπεις. Στο τέλος σβήσε το parameter.

ΛΥΣΗ · TERMINAL

```
rm backend.tf
terraform init
aws ssm put-parameter --name /sites/old/bucket --type String --value tfbook-old
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
│ Warning: Check block assertion failed
…
│     │ data.aws_ssm_parameters_by_path.sites.names is list of string with 1 element
…
│ Υπάρχουν parameters στο /sites χωρίς site στη μεταβλητή sites.
…
```

ΛΥΣΗ · TERMINAL

```
aws ssm delete-parameter --name /sites/old/bucket
```

Το check `parameters` απέτυχε ως `Warning`. Η λίστα των ονομάτων έχει ένα στοιχείο, γιατί η άσκηση δεν έκανε apply και το μόνο parameter είναι το ορφανό. Το `old` δεν είναι κλειδί της μεταβλητής `sites` και το μήνυμα το λέει. Το plan ολοκληρώθηκε κανονικά και σχεδιάζει τα resources όπως πάντα. Ένα check

δεν μπλοκάρει ποτέ τη δουλειά. Αυτό ταιριάζει εδώ, γιατί το ορφανό parameter δεν είναι λόγος να σταματήσει μια αλλαγή σε άλλο site. Το Terraform δεν διαχειρίζεται αυτό το parameter και δεν θα το σβήσει. Το σβήνει όποιος το ξέχασε.

Άσκηση 27.4 · Το parameter στα outputs

Το pipeline θέλει το όνομα του parameter κάθε site χωρίς να το μαντεύει. Πρόσθεσε στο module output `parameter` και στο root module output `parameters`, ένα map από όνομα site σε όνομα parameter. Κάνε apply, δείξε το output σε JSON και κάνε destroy.

ΛΥΣΗ · TERMINAL

```
rm backend.tf
```

ΛΥΣΗ · modules/site/outputs.tf

```
output "parameter" {
  value = aws_ssm_parameter.bucket.name
}
```

ΛΥΣΗ · outputs.tf

```
output "parameters" {
  description = "Το όνομα του parameter κάθε site"
  value       = { for name, site in module.site : name => site.parameter }
}
```

ΛΥΣΗ · TERMINAL

```
terraform init
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
terraform output -json parameters
```

ΕΞΟΔΟΣ

```
{"blog":"/sites/blog/bucket","docs":"/sites/docs/bucket","shop":"/sites/shop/bucket"}
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Ένα module δείχνει προς τα έξω μόνο ό,τι δηλώνει ως output (κεφάλαιο 15). Χωρίς το `parameter` στο module, το `site.parameter` στο root θα έβγαζε σφάλμα `Unsupported attribute`. Το `for` του root φτιάχνει το map όπως τα `buckets` και `endpoints`. Το `-json` δίνει μια γραμμή που ένα script διαβάζει με `jq`.

Άσκηση 27.5 · Μετονομασία μέσα στο module

Το resource `aws_s3_object.index` του module θα λέγεται `aws_s3_object.home`. Κάνε apply, μετονόμασε το resource και γράψε όσα blocks `moved` χρειάζονται ώστε το plan να μη σβήνει τίποτα. Πόσα blocks θέλουν τα τρία sites; Με Moto τρέξε το πρώτο apply δύο φορές, για τα tags των νέων buckets. Στο τέλος κάνε destroy.

ΛΥΣΗ · TERMINAL

```
rm backend.tf
terraform init
terraform apply -auto-approve
terraform apply -auto-approve
```

ΛΥΣΗ · modules/site/main.tf

```
resource "aws_s3_bucket" "this" {
  bucket = var.bucket

  tags = {
    Site  = var.name
    Owner = var.owner
  }
}

resource "aws_s3_bucket_website_configuration" "this" {
  bucket = aws_s3_bucket.this.id

  index_document {
    suffix = "index.html"
  }
}

resource "aws_s3_object" "home" {
  bucket       = aws_s3_bucket.this.id
  key          = "index.html"
  content      = "<h1>${var.name}</h1>\n"
  content_type = "text/html"
}

moved {
  from = aws_s3_object.index
  to   = aws_s3_object.home
}
```

ΛΥΣΗ · TERMINAL

```
terraform plan
```

ΕΞΟΔΟΣ

```
…
  # module.site["blog"].aws_s3_object.index has moved to module.site["blog"].aws_s3_object.home
…
  # module.site["docs"].aws_s3_object.index has moved to module.site["docs"].aws_s3_object.home
…
  # module.site["shop"].aws_s3_object.index has moved to module.site["shop"].aws_s3_object.home
…
Plan: 0 to add, 0 to change, 0 to destroy.
…
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
```

Ένα block αρκεί. Μέσα στο module οι διευθύνσεις του `moved` είναι σχετικές με το module, όπως είπε το κεφάλαιο 23. Το block ισχύει για κάθε instance, οπότε το plan δείχνει τρεις μεταφορές. Στο κεφάλαιο χρειάστηκαν δύο blocks, γιατί η μεταφορά περνούσε από το root στο module και μόνο το root βλέπει τα κλειδιά των instances. Το `key` του object μένει `index.html`. Άλλαξε μόνο το όνομα του resource στον κώδικα.

Άσκηση 27.6 · Το project σε δεύτερο state

Αυτή τη φορά κράτα το `backend.tf`. Στήσε το project σε νέο bucket state με όνομα `tfbook-` και μετά τον αριθμό του λογαριασμού και το `-state-v2`. Φτιάξε το bucket με το CLI. Το state πρέπει να γραφτεί στο key `v2/terraform.tfstate`, χωρίς να αλλάξεις το `backend.tf`. Κάνε apply, δείξε ποια keys υπάρχουν στο bucket και καθάρισε τα πάντα, μαζί με το bucket.

ΛΥΣΗ · TERMINAL

```
ACCOUNT=$(aws sts get-caller-identity --query Account --output text)
aws s3 mb "s3://tfbook-$ACCOUNT-state-v2"
terraform init -backend-config="bucket=tfbook-$ACCOUNT-state-v2" \
  -backend-config="key=v2/terraform.tfstate"
terraform apply -auto-approve
```

ΛΥΣΗ · TERMINAL

```
aws s3api list-objects-v2 --bucket "tfbook-$ACCOUNT-state-v2" --query 'Contents[].Key'
```

ΕΞΟΔΟΣ

```
[
    "v2/terraform.tfstate"
]
```

ΛΥΣΗ · TERMINAL

```
terraform destroy -auto-approve
aws s3 rm "s3://tfbook-$ACCOUNT-state-v2/v2/terraform.tfstate"
aws s3 rb "s3://tfbook-$ACCOUNT-state-v2"
```

Το `-backend-config="key=..."` κερδίζει το `key` του block, όπως είπε το κεφάλαιο 19 για τιμές που δίνονται δύο φορές. Ο ίδιος κώδικας γράφει έτσι σε άλλο state χωρίς καμία αλλαγή στα αρχεία. Ο ίδιος μηχανισμός δίνει ένα state ανά περιβάλλον. Η σειρά στο καθάρισμα μετράει. Πρώτα το `destroy`, μετά το state και τελευταίο το bucket. Αν έσβηνες νωρίτερα το state, τα resources θα έμεναν στο AWS χωρίς κανέναν να τα θυμάται.

Ερωτήσεις πιστοποίησης

Ερώτηση 27.1

Η configuration `sites` κρατά το state σε bucket που το όνομά του περιέχει τον αριθμό του λογαριασμού. Ποιος τρόπος δίνει το όνομα στο S3 backend χωρίς να γραφτεί σταθερό στον κώδικα;

a. `bucket = "tfbook-${data.aws_caller_identity.current.account_id}-state"` μέσα στο block `backend`
b. `bucket = var.state_bucket` στο block `backend` και η τιμή στο `terraform.tfvars`
c. Partial configuration και η τιμή στο `terraform init -backend-config="bucket=..."`
d. Ένα local με το όνομα και `bucket = local.state_bucket` στο block `backend`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το backend ρυθμίζεται στο `init`, πριν το Terraform υπολογίσει οποιαδήποτε έκφραση, γιατί για να υπολογίσει χρειάζεται ήδη το state. Γι' αυτό το block `backend` δέχεται μόνο σταθερές τιμές. Μεταβλητές, locals και attributes από data sources απορρίπτονται, οπότε τα a, b και d είναι λάθος. Η partial configuration αφήνει την τιμή έξω από το block και τη δίνει στο `init`, όπως έκανε το project με το output του `bootstrap`.

Ερώτηση 27.2

Το `aws_ssm_parameter.site` έχει `for_each` με τα κλειδιά `docs` και `blog`. Το μεταφέρεις στο `module "site"`, που έχει `for_each` με τα ίδια κλειδιά, με όνομα `aws_ssm_parameter.bucket`. Ποιο ζευγάρι `from` και `to` σε block `moved` μεταφέρει σωστά το instance `docs`;

a. `from = aws_ssm_parameter.site` και `to = module.site.aws_ssm_parameter.bucket`
b. `from = aws_ssm_parameter.site["docs"]` και `to = module.site["docs"].aws_ssm_parameter.bucket`
c. `from = aws_ssm_parameter.site["docs"]` και `to = module.site.aws_ssm_parameter.bucket["docs"]`
d. Δεν χρειάζεται `moved`, γιατί τα κλειδιά είναι ίδια και το Terraform ταιριάζει μόνο του τα instances

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το κλειδί ανήκει στο instance του module, οπότε η σωστή διεύθυνση είναι η b. Το c γράφει το κλειδί στο resource, σε διεύθυνση που δεν υπάρχει στη configuration. Το plan τότε σβήνει το parameter. Το a καταλήγει ακριβώς στην ίδια λάθος διεύθυνση, όπως έδειξε το plan του κεφαλαίου. Το d είναι λάθος, γιατί το Terraform δεν μαντεύει ταυτότητες. Χωρίς `moved` το plan σβήνει τα παλιά και φτιάχνει νέα.

Ερώτηση 27.3

Κάποιος άλλαξε μια ρύθμιση από την κονσόλα. Τρέχεις `terraform apply -refresh-only` και απαντάς `yes`. Τι συμβαίνει;

a. Το state παίρνει τις πραγματικές τιμές και τίποτα δεν αλλάζει στο AWS.
b. Το Terraform επαναφέρει τη ρύθμιση στην τιμή της configuration.
c. Το Terraform γράφει τη νέα τιμή στα αρχεία `.tf`.
d. Το resource που άλλαξε αντικαθίσταται.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το refresh-only καταγράφει την πραγματικότητα στο state και δεν αγγίζει κανένα αντικείμενο, οπότε το b και το d είναι λάθος. Την επαναφορά την κάνει ένα κανονικό apply. Το Terraform δεν γράφει ποτέ στα αρχεία `.tf`. Αν δεχτείς το drift, πρέπει να αλλάξεις τον κώδικα εσύ, αλλιώς το επόμενο κανονικό apply θα το αναιρέσει.

Ερώτηση 27.4

Το site `blog` κλείνει. Θέλεις το Terraform να σταματήσει να διαχειρίζεται το `module.site["blog"]` χωρίς να σβηστεί τίποτα στο AWS. Τα άλλα instances του module μένουν όπως είναι. Τι κάνεις;

a. Block `removed` με `from = module.site["blog"]` και `destroy = false`
b. Σβήνεις το `blog` από το `terraform.tfvars` και κάνεις apply
c. `terraform state rm 'module.site["blog"]'` και μετά σβήνεις το `blog` από τη μεταβλητή
d. `terraform destroy -target='module.site["blog"]'`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το `state rm` βγάζει από το state τα τέσσερα resources του instance και αφήνει τα αντικείμενα στο AWS. Το κλειδί πρέπει να φύγει και από τη μεταβλητή, αλλιώς το επόμενο plan θα ήθελε να ξαναφτιάξει το site. Το a δεν δουλεύει. Το `from` ενός `removed` δέχεται μόνο ολόκληρο module, χωρίς κλειδί instance. Το Terraform απορρίπτει το `module.site["blog"]` με το σφάλμα `Module instance keys not allowed`. Το b και το d σβήνουν τα resources στο AWS.

Ερώτηση 27.5

Σωστό ή λάθος; Με S3 backend το `terraform destroy` σβήνει και το αντικείμενο του state από το bucket.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `destroy` σβήνει τα resources και γράφει ένα state χωρίς resources. Το αντικείμενο του state μένει στο bucket, όπως είδες στο καθάρισμα του project με το `sites/terraform.tfstate`. Το σβήνεις με το χέρι μόνο όταν το project δεν θα ξαναχρησιμοποιηθεί.

Ερώτηση 27.6

Το check `parameters` του project βρίσκει ένα ορφανό parameter. Ποια ισχύουν; Επίλεξε δύο.

a. Το plan βγάζει `Warning` και ολοκληρώνεται κανονικά.
b. Το apply σταματά πριν αλλάξει οποιοδήποτε resource.
c. Ο έλεγχος γίνεται σε κάθε plan και σε κάθε apply.
d. Το Terraform σβήνει το ορφανό parameter στο επόμενο apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Ένα check block βγάζει μόνο προειδοποίηση και δεν σταματά ποτέ plan ή apply, οπότε το b είναι λάθος. Ελέγχεται στο τέλος κάθε plan και κάθε apply, όπως έδειξε το κεφάλαιο 13. Το ορφανό parameter δεν είναι στο state. Το Terraform δεν διαχειρίζεται ό,τι δεν θυμάται, άρα δεν το σβήνει και το d είναι λάθος.

ΚΕΦΑΛΑΙΟ 28

Προσομοίωση εξέτασης

Ερωτήσεις πιστοποίησης

Ερώτηση 28.1

Μια ομάδα περιγράφει την υποδομή της με Terraform. Ποια πρόταση περιγράφει το δηλωτικό (declarative) μοντέλο που ακολουθεί;

a. Γράφεις με τη σειρά τις κλήσεις στο API που πρέπει να γίνουν.
b. Γράφεις ένα script που ελέγχει τι υπάρχει πριν φτιάξει κάτι νέο.
c. Γράφεις την κατάσταση που θέλεις και το Terraform υπολογίζει τις ενέργειες που τη φέρνουν.
d. Καταγράφεις τα κλικ στην κονσόλα και το Terraform τα επαναλαμβάνει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Στο δηλωτικό μοντέλο περιγράφεις το αποτέλεσμα. Το Terraform συγκρίνει την configuration με το state και την πραγματική υποδομή και κάνει μόνο όσα λείπουν. Το a και το b περιγράφουν το προστακτικό (imperative) μοντέλο, όπου γράφεις εσύ τα βήματα και τους ελέγχους. Το Terraform δεν καταγράφει ενέργειες της κονσόλας, οπότε το d είναι λάθος. Στόχος 1a. Ξαναδιάβασε το κεφάλαιο 1.

Ερώτηση 28.2

Μια ομάδα ακολουθεί το workflow write, plan, apply. Σε ποιο βήμα βλέπουν οι συνάδελφοι τι ακριβώς θα αλλάξει στην υποδομή, πριν αλλάξει οτιδήποτε;

a. Στο write, όταν το `terraform validate` δείχνει τις αλλαγές.
b. Στο plan, που δείχνει τις ενέργειες χωρίς να τις εκτελέσει.
c. Στο apply, αφού ολοκληρωθούν οι αλλαγές.
d. Στο `terraform init`, που συγκρίνει τη configuration με την υποδομή.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το plan συγκρίνει τη configuration με το state και την υποδομή και δείχνει κάθε create, update και destroy πριν γίνει. Γι' αυτό είναι το σημείο του review. Το `validate` ελέγχει μόνο την ορθότητα της configuration και δεν ξέρει τι υπάρχει, οπότε το a είναι λάθος. Μετά το apply οι αλλαγές έχουν ήδη γίνει, άρα το c αργεί. Το `init` κατεβάζει providers και modules και ρυθμίζει το backend. Δεν συγκρίνει τίποτα, οπότε το d είναι λάθος. Στόχος 3a. Ξαναδιάβασε τα κεφάλαια 1 και 3.

Ερώτηση 28.3

Η configuration σου χρειάζεται το ID ενός VPC που φτιάχνει και διαχειρίζεται άλλη ομάδα με το δικό της Terraform. Δεν θέλεις η δική σου configuration να μπορεί να το αλλάξει ή να το σβήσει. Τι γράφεις;

a. Ένα block `data "aws_vpc"` που βρίσκει το VPC με φίλτρο.
b. Ένα block `resource "aws_vpc"` με το ίδιο CIDR.
c. Ένα block `import` για το VPC και ένα block `resource` για αυτό.
d. Ένα `depends_on` προς το VPC της άλλης ομάδας.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Ένα data source μόνο διαβάζει ένα αντικείμενο που υπάρχει. Το `terraform destroy` της δικής σου configuration δεν το αγγίζει. Το b θα έφτιαχνε δεύτερο VPC. Το c θα έκανε το VPC managed resource και της δικής σου configuration, οπότε δύο states θα το διαχειρίζονταν. Το `depends_on` δέχεται μόνο αναφορές σε resources, data sources και modules του ίδιου module και δεν δίνει καμία τιμή, άρα το d είναι λάθος. Στόχος 4a. Ξαναδιάβασε το κεφάλαιο 12.

Ερώτηση 28.4

Ένας φάκελος περιέχει μόνο το αρχείο `main.tf` με τη γραμμή `resource "random_pet" "name" {}`. Δεν υπάρχει block `terraform`. Τι κάνει το `terraform init`;

a. Σταματά με σφάλμα, γιατί λείπει το `required_providers`.
b. Ψάχνει ένα εκτελέσιμο με όνομα `random` στο `PATH`.
c. Εγκαθιστά όλους τους providers του namespace `hashicorp`.
d. Βγάζει από το πρόθεμα `random` του τύπου ότι χρειάζεται τον `hashicorp/random` και εγκαθιστά την πιο πρόσφατη έκδοση.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Όταν ένας provider λείπει από το `required_providers`, το Terraform παίρνει το πρόθεμα του τύπου του resource ως τοπικό όνομα και υποθέτει το namespace `hashicorp`. Το init γράφει `Finding latest version of hashicorp/random`. Η δήλωση στο `required_providers` είναι καλή πρακτική και όχι υποχρέωση, οπότε το a είναι λάθος. Οι providers κατεβαίνουν από registry και όχι από το `PATH`, άρα το b είναι λάθος. Το init κατεβάζει μόνο όσους χρειάζεται η configuration, οπότε και το c είναι λάθος. Στόχος 2b. Ξαναδιάβασε το κεφάλαιο 5.

Ερώτηση 28.5

Ποια από τα παρακάτω κάνει το `terraform init`; Επίλεξε τρία.

a. Κατεβάζει τους providers και γράφει τις εκδόσεις τους στο `.terraform.lock.hcl`.
b. Κατεβάζει τα modules που καλεί η configuration.
c. Δημιουργεί τα resources που λείπουν.
d. Ρυθμίζει το backend όπου θα ζει το state.
e. Μορφοποιεί τα αρχεία `.tf` του φακέλου.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b και d

Το init ετοιμάζει τον φάκελο εργασίας. Εγκαθιστά providers και modules μέσα στο `.terraform`, καταγράφει τις εκδόσεις των providers στο lock file και ρυθμίζει το backend. Δεν αγγίζει την υποδομή, οπότε το c είναι λάθος. Αυτό είναι δουλειά του apply. Τη μορφή των αρχείων την αλλάζει μόνο το `terraform fmt`, άρα και το e είναι λάθος. Στόχος 3b. Ξαναδιάβασε τα κεφάλαια 1 και 5.

Ερώτηση 28.6

Η configuration έχει `resource "random_pet" "server" {}`. Θέλεις το `filename` ενός `local_file` να είναι το όνομα που παρήγαγε ο `random_pet`, με κατάληξη `.txt`. Ποια τιμή γράφεις;

a. `"random_pet.server.id.txt"`
b. `random_pet.server.id + ".txt"`
c. `"${random_pet.server.id}.txt"`
d. `"${random_pet.server}.txt"`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Μέσα σε string μια αναφορά μπαίνει σε `${...}` και η τιμή της γράφεται στη θέση της. Το a είναι απλό κείμενο και το plan δείχνει το `filename` ακριβώς ως `"random_pet.server.id.txt"`. Το b αποτυγχάνει με `Invalid operand`, γιατί ο τελεστής `+` δουλεύει μόνο με αριθμούς. Το d αποτυγχάνει με `Invalid template interpolation value`, γιατί το `random_pet.server` είναι ολόκληρο object και όχι string. Στόχος 4b. Ξαναδιάβασε τα κεφάλαια 6 και 10.

Ερώτηση 28.7

Ένα repository Git έχει πολλά modules σε υποφακέλους. Θέλεις το module του φακέλου `modules/vpc` στο tag `v2.0.0`. Ποιο `source` είναι σωστό;

a. `"git::https://example.com/infra.git/modules/vpc?ref=v2.0.0"`
b. `"git::https://example.com/infra.git//modules/vpc?ref=v2.0.0"`
c. `"git::https://example.com/infra.git?ref=v2.0.0"` μαζί με `path = "modules/vpc"`
d. `"./infra.git/modules/vpc"` μαζί με `version = "2.0.0"`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το διπλό `//` χωρίζει τη διεύθυνση του repository από τον υποφάκελο μέσα σε αυτό και το `?ref=` μπαίνει στο τέλος. Το Terraform κλωνοποιεί όλο το repository και χρησιμοποιεί τον υποφάκελο. Στο a η απλή κάθετος κάνει τον υποφάκελο μέρος της διεύθυνσης του repository και το init αποτυγχάνει με `Failed to download module`. Το block `module` δεν έχει argument `path` και θα το θεωρούσε input του module, οπότε το c είναι λάθος. Το d είναι τοπικό path και το `version` ισχύει μόνο για registry. Στόχος 5a. Ξαναδιάβασε το κεφάλαιο 16.

Ερώτηση 28.8

Δύο μηχανικοί δουλεύουν στην ίδια configuration, ο καθένας από το laptop του. Η configuration δεν έχει block `backend` και ο καθένας κάνει apply από το δικό του αντίγραφο του repository. Ποιο πρόβλημα θα εμφανιστεί;

a. Κάθε laptop έχει δικό του `terraform.tfstate`, οπότε ο ένας δεν βλέπει τι έφτιαξε ο άλλος και τα applies τους συγκρούονται ή φτιάχνουν διπλά resources.
b. Κανένα, γιατί το local backend συγχρονίζει τα states μέσω του provider.
c. Κανένα, γιατί το lock του local backend δουλεύει ανάμεσα σε διαφορετικά μηχανήματα.
d. Το Terraform αρνείται να τρέξει χωρίς block `backend`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το local backend γράφει το state στον δίσκο του μηχανήματος που τρέχει την εντολή. Δύο laptops σημαίνουν δύο ανεξάρτητα states για την ίδια υποδομή. Ο provider μιλά με το API και δεν ξέρει τίποτα για το state, οπότε το b είναι λάθος. Το lock του local backend προστατεύει μόνο από δύο εντολές στο ίδιο μηχάνημα, άρα και το c είναι λάθος. Χωρίς block `backend` το Terraform χρησιμοποιεί το local και δεν σταματά, οπότε το d είναι λάθος. Η λύση είναι ένα κοινό remote backend με locking. Στόχος 6a. Ξαναδιάβασε τα κεφάλαια 4 και 19.

Ερώτηση 28.9

Ποια arguments δέχεται ένα block `variable`; Επίλεξε τρία.

a. `type`
b. `value`
c. `default`
d. `required`
e. `description`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c και e

Το block `variable` δέχεται μεταξύ άλλων τα `type`, `default`, `description`, `sensitive`, `nullable`, `ephemeral` και blocks `validation`. Η τιμή δεν γράφεται μέσα στο block. Έρχεται από `-var`, από αρχεία `.tfvars`, από `TF_VAR_` ή από το `default`, οπότε argument `value` δεν υπάρχει. Μια μεταβλητή είναι υποχρεωτική όταν δεν έχει `default`. Argument `required` δεν υπάρχει. Το `validate` απορρίπτει και τα δύο με `Unsupported argument`. Στόχος 4c. Ξαναδιάβασε το κεφάλαιο 7.

Ερώτηση 28.10

Θέλεις να δεις ποια resources θα σβήσει το `terraform destroy`, χωρίς να σβηστεί τίποτα. Ποια εντολή τρέχεις;

a. `terraform destroy -dry-run`
b. `terraform show -destroy`
c. `terraform state list -destroy`
d. `terraform plan -destroy`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `plan -destroy` φτιάχνει το plan διαγραφής και σταματά εκεί, όπως κάθε plan. Η σημαία `-dry-run` δεν υπάρχει στο Terraform. Το `terraform show` δείχνει ένα state ή ένα αποθηκευμένο plan και δεν φτιάχνει νέο plan. Το `terraform state list` τυπώνει διευθύνσεις από το state και δεν έχει σημαία `-destroy`. Και οι τρεις λάθος επιλογές σταματούν με σφάλμα `flag provided but not defined`. Στόχος 3d. Ξαναδιάβασε το κεφάλαιο 3.

Ερώτηση 28.11

Η configuration έχει δύο ρυθμίσεις του provider `aws`.

```hcl
provider "aws" {
  region = "eu-central-1"
}

provider "aws" {
  alias  = "us"
  region = "us-east-1"
}
```

Πώς φτιάχνεις ένα `aws_s3_bucket` με τη δεύτερη ρύθμιση;

a. Γράφεις `alias = "us"` μέσα στο resource.
b. Γράφεις `provider = aws.us` μέσα στο resource.
c. Γράφεις `provider = us` μέσα στο resource.
d. Γράφεις `providers = { aws = aws.us }` μέσα στο resource.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το meta-argument `provider` δέχεται αναφορά της μορφής `ΟΝΟΜΑ.ALIAS`, χωρίς εισαγωγικά. Το `alias` ορίζεται μόνο στο block `provider` και μέσα σε resource δίνει `Unsupported argument`. Στο c το Terraform διαβάζει το `us` ως τοπικό όνομα άλλου provider και ζητά τον ανύπαρκτο `hashicorp/us`. Το `providers` με map είναι argument του block `module` και όχι του resource. Στόχος 2c. Ξαναδιάβασε τα κεφάλαια 5 και 17.

Ερώτηση 28.12

Θέλεις να φέρεις στο Terraform δύο buckets που φτιάχτηκαν από την κονσόλα.

```hcl
locals {
  buckets = {
    logs   = "acme-logs"
    assets = "acme-assets"
  }
}

import {
  for_each = local.buckets
  to       = aws_s3_bucket.this[each.key]
  id       = each.value
}

resource "aws_s3_bucket" "this" {
  for_each = local.buckets
  bucket   = each.value
}
```

Τι δείχνει το `terraform plan`;

a. Ότι θα γίνει import στα `aws_s3_bucket.this["logs"]` και `aws_s3_bucket.this["assets"]`.
b. Σφάλμα, γιατί ένα block `import` δεν δέχεται `for_each`.
c. Ότι θα δημιουργηθούν δύο νέα buckets.
d. Ότι θα γίνει import σε όλα τα buckets του λογαριασμού.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Από την έκδοση 1.7 ένα block `import` δέχεται `for_each`. Κάθε στοιχείο του map δίνει ένα import με δική του διεύθυνση και δικό του id. Το plan γράφει `will be imported` για κάθε instance και η περίληψη ξεκινά με `Plan: 2 to import`. Τα buckets υπάρχουν ήδη, οπότε δεν δημιουργείται τίποτα και το c είναι λάθος. Το Terraform φέρνει μόνο όσα του ζητάς ρητά, άρα το d είναι λάθος. Στόχος 7a. Ξαναδιάβασε το κεφάλαιο 22.

Ερώτηση 28.13

Δίνεται η μεταβλητή `servers`.

```hcl
variable "servers" {
  type = map(object({
    size = string
    port = number
  }))
  default = {
    web = { size = "small", port = 80 }
    api = { size = "large", port = 8080 }
  }
}
```

Ποια έκφραση δίνει `8080`;

a. `var.servers[1].port`
b. `var.servers["api"].port`
c. `var.servers.port["api"]`
d. `var.servers["api"]["size"]`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `var.servers` είναι map, οπότε ένα στοιχείο βρίσκεται με το κλειδί του, εδώ `["api"]`. Το αποτέλεσμα είναι object και το attribute του διαβάζεται με τελεία. Ένα map δεν έχει αριθμητικές θέσεις, οπότε το a αποτυγχάνει με `Invalid index`. Το c ζητά από το map το κλειδί `port`, που δεν υπάρχει. Αποτυγχάνει με `Missing map element`. Το d είναι έγκυρη έκφραση, αλλά δίνει `"large"`. Στόχος 4d. Ξαναδιάβασε το κεφάλαιο 9.

Ερώτηση 28.14

Με ποιους τρόπους ξεκινά ένα run σε ένα workspace του HCP Terraform; Επίλεξε τρία.

a. Με commit στο κλαδί ενός repository που είναι συνδεδεμένο με το workspace.
b. Με email προς τη διεύθυνση του workspace.
c. Με `terraform plan` ή `terraform apply` από το CLI, με block `cloud` στη configuration.
d. Με αλλαγή ενός αρχείου στον φάκελο `.terraform` του laptop σου.
e. Με κλήση στο API του HCP Terraform, που ανεβάζει νέο configuration version.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c και e

Τα τρία workflows είναι το VCS-driven, το CLI-driven και το API-driven. Στο πρώτο ένα commit ή ένα merge ξεκινά run. Στο δεύτερο το CLI ανεβάζει τα αρχεία σου. Στο τρίτο ένα εργαλείο ανεβάζει την configuration μέσω API. Workflow με email δεν υπάρχει. Ο φάκελος `.terraform` είναι τοπικός και το HCP Terraform δεν τον παρακολουθεί. Στόχος 8a. Ξαναδιάβασε το κεφάλαιο 26.

Ερώτηση 28.15

Μια ομάδα κρατά τη configuration του Terraform σε Git και κάθε αλλαγή περνά από pull request. Ποια δύο οφέλη κερδίζει; Επίλεξε δύο.

a. Το Git εφαρμόζει μόνο του κάθε merge στην υποδομή.
b. Κάθε αλλαγή στην υποδομή ελέγχεται από συνάδελφο πριν εφαρμοστεί.
c. Το state δεν χρειάζεται πια, γιατί το Git κρατά το ιστορικό.
d. Το ιστορικό δείχνει ποιος άλλαξε τι στην υποδομή, πότε και γιατί.
e. Τα secrets της configuration προστατεύονται αυτόματα από το Git.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Η υποδομή ως κώδικας κερδίζει ό,τι κερδίζει κάθε κώδικας. Review πριν από την αλλαγή και ιστορικό με συγγραφέα, ημερομηνία και μήνυμα. Το Git αποθηκεύει αρχεία και δεν τρέχει τίποτα. Για αυτόματο apply χρειάζεται pipeline ή το VCS workflow του HCP Terraform, οπότε το a είναι λάθος. Το state κρατά την αντιστοίχιση με τα πραγματικά αντικείμενα και το Git δεν το αντικαθιστά, άρα και το c είναι λάθος. Ό,τι μπαίνει σε Git το διαβάζει όποιος έχει πρόσβαση στο repository, οπότε το e είναι λάθος. Στόχος 1b. Ξαναδιάβασε το κεφάλαιο 1.

Ερώτηση 28.16

Ένα `terraform apply` πρέπει να φτιάξει τα `local_file.a`, `local_file.b` και `local_file.bad`. Τα δύο πρώτα φτιάχνονται. Το τρίτο αποτυγχάνει με σφάλμα δικαιωμάτων στον δίσκο. Τι ισχύει μετά;

a. Το Terraform σβήνει αυτόματα τα `a` και `b`, ώστε η υποδομή να γυρίσει όπως ήταν.
b. Το state δεν γράφεται, οπότε το επόμενο plan προτείνει ξανά και τα τρία.
c. Τα `a` και `b` υπάρχουν και είναι στο state. Το επόμενο plan προτείνει μόνο το `bad`.
d. Τα `a` και `b` υπάρχουν, αλλά λείπουν από το state μέχρι να τρέξεις `terraform apply -refresh-only`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το Terraform δεν έχει rollback. Κάθε resource που ολοκληρώθηκε γράφεται στο state και μένει. Αφού διορθώσεις την αιτία, ένα νέο apply συνεχίζει από εκεί που σταμάτησε και το plan του γράφει `Plan: 1 to add`. Το a περιγράφει συναλλαγή βάσης δεδομένων, που το Terraform δεν κάνει. Το b και το d είναι λάθος, γιατί κάθε resource γράφεται στο state μόλις φτιαχτεί. Το refresh-only ενημερώνει άλλωστε μόνο όσα είναι ήδη στο state. Στόχος 3e. Ξαναδιάβασε τα κεφάλαια 3 και 25.

Ερώτηση 28.17

Τι επιστρέφει η έκφραση `merge({ a = 1, b = 2 }, { b = 3, c = 4 })`;

a. `{ a = 1, b = 3, c = 4 }`
b. `{ a = 1, b = 2, c = 4 }`
c. Σφάλμα, γιατί το κλειδί `b` υπάρχει και στα δύο.
d. `{ a = 1, b = [2, 3], c = 4 }`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Η `merge` ενώνει maps ή objects από αριστερά προς τα δεξιά. Όταν ένα κλειδί υπάρχει σε περισσότερα από ένα, κερδίζει το τελευταίο. Γι' αυτό το κλειδί `b` παίρνει `3` και η επιλογή b είναι λάθος. Ένα κοινό κλειδί δεν είναι σφάλμα, οπότε το c είναι λάθος. Η `merge` δεν συνδυάζει τιμές σε λίστα, άρα και το d είναι λάθος. Ο κανόνας του τελευταίου κάνει τη `merge` χρήσιμη για προεπιλογές που αντικαθίστανται, όπως `merge(local.default_tags, var.tags)`. Στόχος 4e. Ξαναδιάβασε το κεφάλαιο 10.

Ερώτηση 28.18

Το root module καλεί δύο modules.

```hcl
module "network" {
  source = "./modules/network"
}

module "app" {
  source    = "./modules/app"
  subnet_id = module.network.subnet_id
}
```

Τι ισχύει για τη σειρά δημιουργίας;

a. Τα δύο modules φτιάχνονται παράλληλα, γιατί δεν υπάρχει `depends_on`.
b. Τα resources του `app` που χρησιμοποιούν το `subnet_id` περιμένουν όσα χρειάζεται το output `subnet_id` του `network`.
c. Το plan αποτυγχάνει, γιατί ένα module δεν μπορεί να διαβάσει output άλλου module.
d. Πρώτα φτιάχνεται ολόκληρο το `app`, γιατί είναι γραμμένο τελευταίο.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η αναφορά `module.network.subnet_id` είναι implicit dependency, όπως κάθε αναφορά. Το Terraform ακολουθεί την τιμή από το output του `network` μέχρι το resource που την παράγει. Το `depends_on` χρειάζεται μόνο για εξαρτήσεις χωρίς αναφορά, οπότε το a είναι λάθος. Η μεταφορά τιμών από ένα module σε άλλο μέσω του root είναι ο κανονικός τρόπος σύνθεσης, άρα το c είναι λάθος. Η σειρά των blocks δεν παίζει ρόλο, οπότε και το d είναι λάθος. Στόχος 5c. Ξαναδιάβασε τα κεφάλαια 15 και 18.

Ερώτηση 28.19

Ποιες εντολές παίρνουν lock στο state πριν δουλέψουν, όταν το backend υποστηρίζει locking; Επίλεξε τρία.

a. `terraform plan`
b. `terraform apply`
c. `terraform fmt`
d. `terraform destroy`
e. `terraform validate`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b και d

Οι εντολές που διαβάζουν το state για να υπολογίσουν αλλαγές ή που το γράφουν παίρνουν lock, ώστε κανείς άλλος να μην το αλλάξει στο μεταξύ. Και το plan, που δεν γράφει state, σταματά με `Error acquiring the state lock` αν ένα apply κρατά το lock. Το `fmt` και το `validate` δουλεύουν μόνο με τα αρχεία της configuration και δεν ανοίγουν καθόλου το state. Στόχος 6b. Ξαναδιάβασε το κεφάλαιο 19.

Ερώτηση 28.20

Δίνεται η παρακάτω configuration.

```hcl
resource "random_pet" "bucket" {}

locals {
  bucket_name = "logs-${random_pet.bucket.id}"
}

resource "local_file" "config" {
  filename = "${path.module}/config.txt"
  content  = local.bucket_name
}
```

Το `local_file.config` δεν αναφέρει απευθείας το `random_pet.bucket`. Τι ισχύει;

a. Τα δύο resources φτιάχνονται παράλληλα, γιατί το local κόβει την εξάρτηση.
b. Χρειάζεται `depends_on = [random_pet.bucket]` στο `local_file.config`.
c. Το plan αποτυγχάνει, γιατί ένα local δεν μπορεί να αναφέρει resource.
d. Το Terraform βλέπει την εξάρτηση μέσα από το local και φτιάχνει πρώτα το `random_pet.bucket`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Οι εξαρτήσεις ακολουθούν τις αναφορές, όσα ενδιάμεσα βήματα κι αν έχουν. Το `terraform graph` δείχνει το βέλος `"local_file.config" -> "random_pet.bucket"` και το apply φτιάχνει πρώτα το `random_pet`. Γι' αυτό το a είναι λάθος και το `depends_on` του b είναι περιττό. Ένα local μπορεί να αναφέρει resources, μεταβλητές και άλλα locals, οπότε το c είναι λάθος. Στόχος 4f. Ξαναδιάβασε τα κεφάλαια 6 και 8.

Ερώτηση 28.21

Ποια από τα παρακάτω πρέπει να μπουν σε version control μαζί με τη configuration; Επίλεξε δύο.

a. Τα αρχεία `.tf`
b. Ο φάκελος `.terraform`
c. Το `.terraform.lock.hcl`
d. Το `terraform.tfstate`
e. Το `terraform.tfstate.backup`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Τα αρχεία `.tf` είναι η ίδια η configuration. Το lock file κρατά τις εκδόσεις και τα hashes των providers, ώστε κάθε μέλος της ομάδας και κάθε pipeline να εγκαθιστά τα ίδια. Ο φάκελος `.terraform` περιέχει τα κατεβασμένα plugins και modules και ξαναφτιάχνεται με `terraform init`. Το state και το backup του μπορεί να περιέχουν secrets σε απλό κείμενο. Ανήκουν σε backend και όχι σε Git. Στόχος 2a. Ξαναδιάβασε τα κεφάλαια 4 και 5.

Ερώτηση 28.22

Ένα pipeline CI ελέγχει κάθε pull request. Θέλει να διαβάζει με πρόγραμμα τα σφάλματα της configuration, με αρχείο και γραμμή για το καθένα, ώστε να τα σημειώνει πάνω στον κώδικα. Ποια εντολή τρέχει;

a. `terraform validate -no-color`
b. `terraform fmt -check -diff`
c. `terraform show -json`
d. `terraform validate -json`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `validate -json` δίνει τα ίδια diagnostics σε JSON. Έχει τα πεδία `valid` και `error_count` και για κάθε σφάλμα `severity`, `summary` και `range` με αρχείο, γραμμή και στήλη. Το exit code μένει 1 όταν υπάρχουν σφάλματα. Το `-no-color` αφαιρεί μόνο τα χρώματα και αφήνει κείμενο για ανθρώπους. Το `fmt -check -diff` ελέγχει μόνο τη μορφή. Το `show -json` δίνει ένα state ή ένα αποθηκευμένο plan, όχι τα σφάλματα της configuration. Στόχος 3c. Ξαναδιάβασε τα κεφάλαια 2 και 25.

Ερώτηση 28.23

Ένα workspace του HCP Terraform έχει execution mode Remote. Στο laptop σου έχεις ορίσει `AWS_ACCESS_KEY_ID` και `AWS_SECRET_ACCESS_KEY` και τρέχεις `terraform plan` στον φάκελο με το block `cloud`. Ο AWS provider αποτυγχάνει, γιατί δεν βρίσκει credentials. Ποια είναι η αιτία και η λύση;

a. Το block `cloud` θέλει τα credentials του AWS στο argument `token`.
b. Το HCP Terraform διαβάζει μόνο το `~/.aws/credentials` του laptop.
c. Το run τρέχει σε μηχάνημα του HCP Terraform, που δεν βλέπει το shell σου. Τα credentials μπαίνουν ως sensitive environment variables στο workspace ή σε variable set.
d. Το plan χρειάζεται πρώτα `terraform login` με τα κλειδιά του AWS.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Με remote execution το CLI ανεβάζει την configuration και το plan τρέχει σε προσωρινό μηχάνημα του HCP Terraform. Οι μεταβλητές περιβάλλοντος του laptop δεν ταξιδεύουν μαζί. Ο provider βρίσκει credentials από τις μεταβλητές του workspace, από variable sets ή από dynamic credentials. Το `token` του block `cloud` είναι token του HCP Terraform και όχι του AWS, οπότε το a είναι λάθος. Το run δεν έχει πρόσβαση στα αρχεία του laptop σου, άρα το b είναι λάθος. Το `terraform login` παίρνει token για το HCP Terraform και δεν αφορά τον AWS provider, οπότε και το d είναι λάθος. Στόχος 8a. Ξαναδιάβασε το κεφάλαιο 26.

Ερώτηση 28.24

Θέλεις, αφού δημιουργηθεί ένα `aws_instance` με όνομα `web`, να βεβαιωθείς ότι πήρε δημόσιο DNS όνομα. Αν δεν πήρε, το apply πρέπει να βγάλει error. Τι γράφεις στο `lifecycle` του resource;

a. Ένα `postcondition` με `condition = self.public_dns != ""`
b. Ένα `precondition` με `condition = self.public_dns != ""`
c. Ένα `postcondition` με `condition = aws_instance.web.public_dns != ""`
d. Ένα `validation` με `condition = self.public_dns != ""`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το postcondition ελέγχει το αντικείμενο μετά τη δημιουργία και διαβάζει τα attributes του με `self`. Το `self` δεν υπάρχει στο precondition, που τρέχει πριν από την αλλαγή. Εκεί το validate απαντά `Invalid "self" reference`. Ένα resource δεν μπορεί να αναφέρει τον εαυτό του με τη διεύθυνσή του, οπότε το c σταματά με `Configuration for aws_instance.web may not refer to itself`. Το `validation` ανήκει μόνο στο block `variable`, άρα και το d είναι λάθος. Στόχος 4g. Ξαναδιάβασε το κεφάλαιο 13.

Ερώτηση 28.25

Θέλεις να δεις όλα τα resources που ανήκουν στο module `network`, όπως τα καταγράφει το state. Ποια εντολή τρέχεις;

a. `terraform state show module.network`
b. `terraform state list module.network`
c. `terraform output module.network`
d. `terraform show module.network`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `terraform state list` δέχεται διεύθυνση ως φίλτρο και τυπώνει όσα resources βρίσκονται κάτω από αυτήν, για παράδειγμα `module.network.aws_vpc.main`. Το `state show` θέλει τη διεύθυνση ενός συγκεκριμένου instance και με διεύθυνση module απαντά `Error parsing instance address`. Το `terraform output` δείχνει outputs του root module και όχι resources. Το `terraform show` δέχεται αρχείο state ή plan και όχι διεύθυνση. Στόχος 7b. Ξαναδιάβασε το κεφάλαιο 21.

Ερώτηση 28.26

Ένα child module δηλώνει `variable "name"` χωρίς `default`. Το root module το καλεί με ένα block `module` που γράφει μόνο το `source`. Τι γίνεται;

a. Το Terraform ρωτά στο terminal για την τιμή του `name`.
b. Το child παίρνει αυτόματα την τιμή της `var.name` του root, αν υπάρχει.
c. Το `terraform validate` σταματά με `Missing required argument` για το `name`.
d. Η μεταβλητή παίρνει `null` και το plan συνεχίζει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Οι μεταβλητές ενός child module είναι τα arguments του block `module`. Μια μεταβλητή χωρίς `default` είναι υποχρεωτικό argument και μια κλήση χωρίς αυτήν αποτυγχάνει ήδη στο `validate`. Το Terraform ρωτά στο terminal μόνο για μεταβλητές του root module, οπότε το a είναι λάθος. Ένα child module δεν βλέπει τις μεταβλητές του root, άρα το b είναι λάθος. Το `null` θα ίσχυε μόνο με `default = null`, οπότε και το d είναι λάθος. Στόχος 5b. Ξαναδιάβασε το κεφάλαιο 15.

Ερώτηση 28.27

Στο plan μιας νέας configuration βλέπεις τη γραμμή `+ id = (known after apply)`. Τι σημαίνει;

a. Η τιμή θα γίνει γνωστή μόνο όταν ο provider δημιουργήσει το αντικείμενο στο apply.
b. Η τιμή είναι sensitive και το Terraform την κρύβει.
c. Το resource δεν έχει `id` και το apply θα αποτύχει.
d. Το refresh απέτυχε και το Terraform θα ξαναδοκιμάσει στο apply.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Πολλά attributes, όπως τα IDs, τα δίνει το API τη στιγμή που φτιάχνεται το αντικείμενο. Στο plan το Terraform τα σημειώνει ως άγνωστα και τα συμπληρώνει στο apply. Μια sensitive τιμή εμφανίζεται ως `(sensitive value)`, οπότε το b είναι λάθος. Το `(known after apply)` είναι κανονική κατάσταση και όχι σφάλμα, άρα το c και το d είναι λάθος. Στόχος 3d. Ξαναδιάβασε το κεφάλαιο 3.

Ερώτηση 28.28

Η μεταβλητή `token` έχει `ephemeral = true`. Τρέχεις `terraform plan -out=tfplan -var token=abc` και μετά `terraform apply tfplan` χωρίς `-var`. Τι γίνεται;

a. Το apply παίρνει την τιμή από το αρχείο του plan.
b. Το apply χρησιμοποιεί κενό string για τη μεταβλητή.
c. Το apply σταματά με `No value for required variable`, γιατί η ephemeral τιμή δεν γράφτηκε στο αρχείο του plan.
d. Το apply ρωτά στο terminal για την τιμή και συνεχίζει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Μια ephemeral τιμή δεν αποθηκεύεται ούτε στο state ούτε στο αρχείο του plan. Γι' αυτό το μήνυμα εξηγεί ότι η μεταβλητή ορίστηκε στο plan και πρέπει να οριστεί ξανά στο apply. Με `terraform apply -var token=abc tfplan` το apply ολοκληρώνεται. Το a θα ίσχυε για μια κανονική μεταβλητή, που γράφεται στο αρχείο του plan. Το Terraform δεν εφευρίσκει τιμές, οπότε το b είναι λάθος. Το apply ενός αποθηκευμένου plan δεν ρωτά τίποτα, άρα και το d είναι λάθος. Στόχος 4h. Ξαναδιάβασε το κεφάλαιο 24.

Ερώτηση 28.29

Ποιες δύο προτάσεις για το block `backend` ισχύουν; Επίλεξε δύο.

a. Οι τιμές του μπορούν να έρθουν από input variables.
b. Μια configuration έχει το πολύ ένα block `backend`.
c. Γράφεται μέσα σε block `provider`.
d. Μετά από αλλαγή στο block χρειάζεται `terraform init` πριν από το επόμενο plan.
e. Κάθε resource μπορεί να έχει δικό του backend.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και d

Ένα δεύτερο block `backend` το init το απορρίπτει ως διπλή ρύθμιση. Μια αλλαγή στο block κάνει το plan να σταματά με `Backend initialization required` και να ζητά `terraform init`. Το backend ρυθμίζεται στο init, πριν υπολογιστούν μεταβλητές, οπότε το a δίνει `Variables not allowed`. Οι τιμές που αλλάζουν δίνονται με `-backend-config`. Το block `backend` ζει μέσα στο block `terraform`, άρα το c είναι λάθος. Όλο το state μιας configuration ζει σε ένα backend, οπότε και το e είναι λάθος. Στόχος 6c. Ξαναδιάβασε το κεφάλαιο 19.

Ερώτηση 28.30

Μια ομάδα διαχειρίζεται ήδη το AWS με Terraform. Θέλει να διαχειρίζεται με τον ίδιο τρόπο και τα repositories και τις ομάδες της στο GitHub. Τι χρειάζεται;

a. Ένα άλλο εργαλείο, γιατί το Terraform διαχειρίζεται μόνο υποδομή cloud.
b. Το HCP Terraform, γιατί το CLI δουλεύει μόνο με providers της HashiCorp.
c. Έναν provider για το GitHub. Το workflow `init`, `plan`, `apply` μένει το ίδιο.
d. Ένα module που μεταφράζει τα resources του AWS σε resources του GitHub.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το Terraform είναι ανεξάρτητο από την υπηρεσία (service agnostic). Ό,τι έχει API και provider το διαχειρίζεσαι με την ίδια γλώσσα και τις ίδιες εντολές. Στο βιβλίο το είδες με αρχεία στον δίσκο, τυχαία ονόματα και secrets στο Vault, οπότε το a είναι λάθος. Το CLI δουλεύει με providers από κάθε namespace του Registry, άρα το b είναι λάθος. Κάθε provider έχει δικούς του τύπους resources και κανένα module δεν τους μεταφράζει, οπότε το d είναι λάθος. Στόχος 1c. Ξαναδιάβασε το κεφάλαιο 5.

Ερώτηση 28.31

Τι επιστρέφει η έκφραση `[for n in [1, 2, 3, 4] : n * 10 if n % 2 == 0]`;

a. `[10, 30]`
b. `[0, 20, 0, 40]`
c. `{ 2 = 20, 4 = 40 }`
d. `[20, 40]`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `if` στο τέλος ενός `for` κρατά μόνο τα στοιχεία που ικανοποιούν τη συνθήκη. Το `n % 2 == 0` ισχύει για το 2 και το 4, που πολλαπλασιάζονται επί 10. Το a κρατά τους μονούς. Το b υποθέτει ότι τα στοιχεία που απορρίπτονται γίνονται `0`, ενώ απλώς παραλείπονται. Οι τετράγωνες αγκύλες φτιάχνουν tuple και όχι object, οπότε το c είναι λάθος. Στόχος 4e. Ξαναδιάβασε το κεφάλαιο 10.

Ερώτηση 28.32

Ένα `local_file.notes` υπάρχει και είναι στο state. Σβήνεις το block του από τη configuration και τρέχεις `terraform plan`. Τι δείχνει;

a. Ότι θα σβηστεί, με την εξήγηση `because local_file.notes is not in configuration`.
b. Καμία αλλαγή, γιατί το Terraform αγνοεί resources που δεν βρίσκει στη configuration.
c. Ότι θα αφαιρεθεί από το state, χωρίς να σβηστεί το αρχείο.
d. Σφάλμα, γιατί το αρχείο υπάρχει ακόμη στον δίσκο.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το state θυμάται ότι το `local_file.notes` το διαχειρίζεται αυτή η configuration. Αφού δεν το βρίσκει πια στον κώδικα, το plan προτείνει destroy και εξηγεί γιατί. Αυτός είναι ένας από τους λόγους ύπαρξης του state. Χωρίς αυτό το Terraform δεν θα ήξερε ότι το αρχείο του ανήκει, οπότε το b και το d είναι λάθος. Για αφαίρεση από το state χωρίς διαγραφή χρειάζεται block `removed` ή `terraform state rm`. Δεν γίνεται μόνη της, άρα και το c είναι λάθος. Στόχος 2d. Ξαναδιάβασε το κεφάλαιο 4.

Ερώτηση 28.33

Ένα pipeline CI πρέπει να κάνει apply χωρίς να γράψει κάποιος `yes`. Ποιοι δύο τρόποι το πετυχαίνουν; Επίλεξε δύο.

a. `terraform apply -auto-approve`
b. `terraform apply -input=false`
c. `terraform apply -force`
d. `terraform plan -out=tfplan` και μετά `terraform apply tfplan`
e. `terraform validate` και μετά `terraform apply`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και d

Το `-auto-approve` παραλείπει την ερώτηση. Ένα αποθηκευμένο plan θεωρείται ήδη εγκεκριμένο και εκτελείται χωρίς ερώτηση. Είναι και ο ασφαλέστερος τρόπος, γιατί εφαρμόζεται ακριβώς ό,τι ελέγχθηκε. Το `-input=false` αφορά τις ερωτήσεις για τιμές μεταβλητών. Το apply ζητά ακόμη έγκριση και χωρίς είσοδο σταματά με σφάλμα. Η σημαία `-force` δεν υπάρχει στο `apply`. Το validate δεν αλλάζει τίποτα στην έγκριση, οπότε και το e είναι λάθος. Στόχος 3e. Ξαναδιάβασε τα κεφάλαια 3 και 7.

Ερώτηση 28.34

Ποιες δύο προτάσεις για τις policies του HCP Terraform ισχύουν; Επίλεξε δύο.

a. Τρέχουν στο laptop του χρήστη μαζί με το `terraform validate`.
b. Γράφονται σε Sentinel ή σε OPA.
c. Ελέγχουν το plan πριν από το apply και μπορούν να σταματήσουν το run.
d. Διορθώνουν αυτόματα τη configuration ώστε να περάσει τον έλεγχο.
e. Ελέγχονται μόνο μετά το apply, πάνω στο νέο state.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και c

Οι policies είναι policy as code. Γράφονται σε Sentinel, τη γλώσσα της HashiCorp, ή σε OPA με Rego. Τρέχουν στο HCP Terraform μετά το plan και πριν από το apply. Ανάλογα με το enforcement level μια αποτυχία προειδοποιεί ή σταματά το run. Δεν τρέχουν τοπικά, οπότε το a είναι λάθος. Ελέγχουν και αναφέρουν, δεν γράφουν κώδικα, άρα το d είναι λάθος. Ένας έλεγχος μετά το apply θα ερχόταν αφού η αλλαγή έχει γίνει, οπότε και το e είναι λάθος. Στόχος 8b. Ξαναδιάβασε το κεφάλαιο 26.

Ερώτηση 28.35

Ποια από τα παρακάτω είναι arguments του ίδιου του block `module` και όχι inputs του module που καλείς; Επίλεξε τρία.

a. `source`
b. `version`
c. `backend`
d. `for_each`
e. `provider_version`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b και d

Τα arguments του block `module` που δεν είναι inputs είναι τα `source`, `version`, `count`, `for_each`, `providers` και `depends_on`. Ένα module δεν έχει δικό του backend. Το state όλων των modules ζει στο backend του root. Argument `provider_version` δεν υπάρχει. Οι εκδόσεις των providers δηλώνονται στο `required_providers`. Αν γράψεις ένα από αυτά τα δύο, το Terraform το θεωρεί input και απαντά `Unsupported argument`, εκτός αν το module έχει μεταβλητή με αυτό το όνομα. Στόχος 5c. Ξαναδιάβασε τα κεφάλαια 15 και 16.

Ερώτηση 28.36

Μετά από κάθε apply θέλεις να ελέγχεις ότι η αρχική σελίδα ενός site απαντά. Το DNS καθυστερεί μερικά λεπτά, οπότε ο έλεγχος αποτυγχάνει συχνά αμέσως μετά το πρώτο apply. Δεν θέλεις αυτό να μπλοκάρει το apply. Τι χρησιμοποιείς;

a. Ένα `postcondition` στο resource του server.
b. Ένα `precondition` στο resource της εγγραφής DNS.
c. Ένα block `check` με scoped data source και `assert`.
d. Ένα block `validation` στη μεταβλητή με το όνομα του site.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το check block είναι ο μόνος μηχανισμός που βγάζει warning αντί για error. Ένα scoped data source μέσα σε αυτό διαβάζει τη σελίδα και το `assert` ελέγχει την απάντηση. Το apply ολοκληρώνεται και η προειδοποίηση εμφανίζεται σε κάθε plan και apply μέχρι να περάσει ο έλεγχος. Το postcondition και το precondition βγάζουν error, οπότε θα σταματούσαν τη δουλειά. Η validation ελέγχει μόνο την τιμή μιας μεταβλητής και όχι αν απαντά ένα site. Στόχος 4g. Ξαναδιάβασε το κεφάλαιο 13.

Ερώτηση 28.37

Το plan δείχνει ότι θα αφαιρέσει από ένα bucket το tag `Owner`, που πρόσθεσε μια συνάδελφος από την κονσόλα. Η ομάδα αποφάσισε ότι το tag είναι σωστό και πρέπει να μείνει. Τι κάνεις ώστε το Terraform να σταματήσει να το αφαιρεί;

a. Τρέχεις `terraform apply -refresh-only`.
b. Προσθέτεις το tag `Owner` στη configuration του bucket.
c. Τρέχεις τα plans με `-refresh=false`.
d. Τρέχεις `terraform state rm` για το bucket και μετά import.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Η configuration είναι η περιγραφή που επιβάλλει το Terraform. Για να μείνει μια αλλαγή που έγινε έξω από αυτό, τη γράφεις στον κώδικα και το επόμενο plan δεν βρίσκει διαφορά. Το refresh-only γράφει το tag στο state, αλλά η configuration εξακολουθεί να μην το έχει, οπότε το επόμενο plan θα το αφαιρέσει ξανά. Το `-refresh=false` απλώς κρύβει το drift, γιατί δεν διαβάζει την υποδομή. Το πρώτο κανονικό plan θα το ξαναδείξει. Το import θα έφερνε πίσω το ίδιο bucket με το ίδιο tag και η configuration θα διέφερε ακόμη. Στόχος 6d. Ξαναδιάβασε το κεφάλαιο 21.

Ερώτηση 28.38

Σωστό ή λάθος; Μετά το apply που έκανε το import, ένα block `import` μπορεί να μείνει στη configuration. Το επόμενο plan δεν προσπαθεί να κάνει ξανά import.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Όταν η διεύθυνση του `to` υπάρχει ήδη στο state με το ίδιο αντικείμενο, το block `import` δεν έχει τι να κάνει και το επόμενο plan λέει `No changes`. Μπορείς να το σβήσεις ή να το κρατήσεις ως καταγραφή του από πού ήρθε το resource. Στόχος 7a. Ξαναδιάβασε το κεφάλαιο 22.

Ερώτηση 28.39

Η μεταβλητή `db_password` έχει `sensitive = true`. Ένα output του root module φτιάχνει από αυτήν ένα connection string και δεν έχει `sensitive`. Τι γίνεται;

a. Το output τυπώνει το connection string κανονικά μετά το apply.
b. Το Terraform αφαιρεί αυτόματα τον κωδικό από το string.
c. Το output παίρνει την τιμή `null`.
d. Το `terraform validate` και το plan σταματούν με `Output refers to sensitive values`, μέχρι να βάλεις `sensitive = true` στο output.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Η ιδιότητα sensitive περνά σε κάθε έκφραση που χρησιμοποιεί την τιμή. Ένα output του root module που την περιέχει πρέπει να δηλωθεί ρητά sensitive, για να επιβεβαιώσεις ότι το θέλεις. Τότε το apply δείχνει `<sensitive>` στη θέση της τιμής. Το Terraform δεν αλλάζει ποτέ σιωπηλά μια τιμή, οπότε το b και το c είναι λάθος. Η τιμή γράφεται πάντως στο state και το `terraform output` την τυπώνει αν τη ζητήσεις με το όνομά της. Στόχος 4c. Ξαναδιάβασε τα κεφάλαια 8 και 24.

Ερώτηση 28.40

Σωστό ή λάθος; Αφού προσθέσεις ένα resource ενός νέου provider σε configuration που έχει ήδη περάσει από `init`, το `terraform plan` δουλεύει χωρίς νέο `terraform init`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ο νέος provider δεν είναι εγκατεστημένος και λείπει από το lock file. Το plan σταματά με `Inconsistent dependency lock file`. Ένα σκέτο `terraform init` εγκαθιστά τον νέο provider και κρατά τις εκδόσεις των υπόλοιπων όπως είναι κλειδωμένες. Το init ξανατρέχει κάθε φορά που αλλάζουν providers, modules ή backend. Στόχος 3b. Ξαναδιάβασε τα κεφάλαια 5 και 25.

Ερώτηση 28.41

Κάποιος αλλάζει από την κονσόλα του AWS ένα security group που διαχειρίζεται ένα workspace του HCP Terraform. Ποια λειτουργία το εντοπίζει χωρίς να περιμένει το επόμενο run της ομάδας;

a. Τα run triggers
b. Το drift detection των health assessments
c. Τα variable sets
d. Το private registry

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το drift detection τρέχει περιοδικά ένα speculative plan και δείχνει στο workspace ποια resources απέκλιναν από την configuration. Δεν αλλάζει τίποτα. Αποφασίζεις εσύ αν θα κάνεις apply ή αν θα αλλάξεις τον κώδικα. Τα run triggers ξεκινούν run όταν κάνει apply ένα άλλο workspace. Τα variable sets μοιράζουν μεταβλητές και το private registry μοιράζει modules και providers. Κανένα από αυτά δεν ελέγχει την υποδομή. Στόχος 8b. Ξαναδιάβασε το κεφάλαιο 26.

Ερώτηση 28.42

Δίνεται το block `terraform` μιας configuration.

```hcl
terraform {
  required_version = ">= 1.12.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 6.0"
    }
  }
}
```

Ποια πρόταση ισχύει;

a. Το `required_version` ορίζει την έκδοση του AWS provider.
b. Το `~> 6.0` δέχεται μόνο εκδόσεις της σειράς 6.0, όπως η 6.0.3.
c. Το `~> 6.0` δέχεται και την 7.0.0, γιατί το άνω όριο περιλαμβάνεται.
d. Το Terraform CLI πρέπει να είναι 1.12.0 ή νεότερο και ο provider από 6.0 έως πριν από 7.0.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `required_version` αφορά μόνο το Terraform CLI. Οι providers παίρνουν εκδόσεις από το `version` μέσα στο `required_providers`, οπότε το a είναι λάθος. Ο `~>` αφήνει να μεγαλώνει μόνο το τελευταίο ψηφίο που γράφεις. Με δύο ψηφία μεγαλώνει το minor, οπότε δεκτές είναι όλες οι 6.x από 6.0 και πάνω. Το b περιγράφει το `~> 6.0.0`. Το άνω όριο εξαιρείται, άρα το c είναι λάθος. Στόχος 2a. Ξαναδιάβασε το κεφάλαιο 5.

Ερώτηση 28.43

Θέλεις να δεις τις εξαρτήσεις ανάμεσα στα resources μιας configuration σε μορφή που γίνεται διάγραμμα με το Graphviz. Ποια εντολή τρέχεις;

a. `terraform providers`
b. `terraform graph`
c. `terraform show -json`
d. `terraform state list`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το `terraform graph` τυπώνει το graph στη γλώσσα DOT, που ξεκινά με `digraph`. Κάθε βέλος δείχνει από ένα resource προς ένα άλλο που χρειάζεται, για παράδειγμα `"local_file.config" -> "random_pet.bucket"`. Το `terraform providers` δείχνει μόνο ποιους providers χρειάζεται κάθε module. Το `show -json` δίνει το state ή ένα plan σε JSON. Το `state list` τυπώνει διευθύνσεις χωρίς εξαρτήσεις. Στόχος 4f. Ξαναδιάβασε το κεφάλαιο 6.

Ερώτηση 28.44

Η configuration καλεί ένα module από το Terraform Registry. Πού το αποθηκεύει το `terraform init`;

a. Στο `~/.terraform.d/modules` του χρήστη, κοινό για όλα τα projects.
b. Στον φάκελο `modules` του project, δίπλα στα αρχεία `.tf`.
c. Μέσα στο `terraform.tfstate`.
d. Στο `.terraform/modules` του φακέλου εργασίας.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το init κατεβάζει κάθε remote module σε υποφάκελο του `.terraform/modules` με το όνομα της κλήσης και γράφει πηγή και έκδοση στο `.terraform/modules/modules.json`. Γι' αυτό ο φάκελος `.terraform` μένει έξω από το Git και ξαναφτιάχνεται με `init`. Κοινός φάκελος για modules στο home δεν υπάρχει, οπότε το a είναι λάθος. Ο φάκελος `modules` του project είναι σύμβαση για τοπικά modules που γράφεις εσύ, άρα το b είναι λάθος. Το state κρατά resources και outputs, όχι κώδικα, οπότε και το c είναι λάθος. Στόχος 5a. Ξαναδιάβασε τα κεφάλαια 15 και 16.

Ερώτηση 28.45

Σωστό ή λάθος; Το local backend κλειδώνει το state, οπότε δύο εντολές στον ίδιο φάκελο του ίδιου μηχανήματος δεν το αλλάζουν ταυτόχρονα.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το local backend κλειδώνει το `terraform.tfstate` με ένα lock του λειτουργικού συστήματος. Ένα δεύτερο plan όσο τρέχει ένα apply σταματά με `Error acquiring the state lock`. Το lock δεν προστατεύει όμως από ένα άλλο μηχάνημα με δικό του αντίγραφο του state. Γι' αυτό μια ομάδα χρειάζεται remote backend. Στόχος 6b. Ξαναδιάβασε τα κεφάλαια 19 και 25.

Ερώτηση 28.46

Ποιες δύο εντολές σβήνουν όλα τα resources που διαχειρίζεται μια configuration; Επίλεξε δύο.

a. `terraform plan -destroy`
b. `terraform destroy`
c. `terraform state rm` για κάθε resource
d. `terraform apply -refresh-only`
e. `terraform apply -destroy`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b και e

Το `terraform destroy` είναι συντόμευση του `terraform apply -destroy`. Και τα δύο δείχνουν plan διαγραφής και ζητούν `yes`. Το `plan -destroy` μόνο δείχνει τι θα σβηστεί. Το `state rm` κάνει το Terraform να ξεχάσει τα resources, αλλά τα αντικείμενα μένουν. Το refresh-only ενημερώνει μόνο το state. Στόχος 3f. Ξαναδιάβασε τα κεφάλαια 1 και 3.

Ερώτηση 28.47

Ένα `aws_secretsmanager_secret_version` παίρνει τον κωδικό στο write-only argument `secret_string_wo` και έχει `secret_string_wo_version = 1`. Έγινε apply. Ποιες δύο προτάσεις ισχύουν; Επίλεξε δύο.

a. Ο κωδικός δεν βρίσκεται ούτε στο αρχείο του plan ούτε στο state.
b. Αν αλλάξεις μόνο τον κωδικό, το επόμενο plan δείχνει update.
c. Για να σταλεί νέος κωδικός, ανεβάζεις το `secret_string_wo_version` σε `2`.
d. Το plan δείχνει τον κωδικό σε απλό κείμενο, αφού δεν θα αποθηκευτεί.
e. Κάνεις write-only οποιοδήποτε argument αν προσθέσεις `_wo` στο όνομά του.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και c

Ένα write-only argument στέλνεται στον provider και δεν αποθηκεύεται. Το plan το δείχνει ως `(write-only attribute)` και στο state η τιμή του είναι `null`, οπότε το d είναι λάθος. Αφού το Terraform δεν κρατά την τιμή, δεν έχει με τι να συγκρίνει τη νέα. Αν αλλάξεις μόνο τον κωδικό, το plan λέει `No changes` και το b είναι λάθος. Το version είναι κανονικό argument και η αλλαγή του φαίνεται στο plan. Τα write-only arguments τα δηλώνει ο provider στο schema του, άρα το e είναι λάθος. Στόχος 4h. Ξαναδιάβασε το κεφάλαιο 24.

Ερώτηση 28.48

Ποιες δύο προτάσεις για τα projects του HCP Terraform ισχύουν; Επίλεξε δύο.

a. Κάθε workspace ανήκει σε ακριβώς ένα project.
b. Ένα workspace μπορεί να ανήκει σε πολλά projects ταυτόχρονα.
c. Όλα τα workspaces ενός project μοιράζονται ένα κοινό state.
d. Το δικαίωμα μιας team σε ένα project ισχύει για όλα τα workspaces του, ακόμη και για όσα προστεθούν αργότερα.
e. Το `Default Project` σβήνεται όταν δεν έχει workspaces.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και d

Τα projects οργανώνουν τα workspaces και δίνουν δικαιώματα σε ομάδες από αυτά. Κάθε workspace ανήκει σε ένα μόνο project, οπότε το b είναι λάθος. Κάθε workspace κρατά το δικό του state, άρα το c είναι λάθος. Το `Default Project` μετονομάζεται αλλά δεν σβήνεται, οπότε και το e είναι λάθος. Στόχος 8c. Ξαναδιάβασε το κεφάλαιο 26.

Ερώτηση 28.49

Σωστό ή λάθος; Αφού μια ομάδα περάσει όλη την υποδομή της σε Terraform, οι αλλαγές με το χέρι από την κονσόλα του cloud δεν μπορούν πια να γίνουν.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Το Terraform δεν κλειδώνει τα αντικείμενα στο cloud. Όποιος έχει δικαιώματα μπορεί ακόμη να αλλάξει κάτι από την κονσόλα. Η διαφορά ανάμεσα στην υποδομή και στη configuration λέγεται drift. Το Terraform τη βρίσκει στο refresh του επόμενου plan και προτείνει να γυρίσει η υποδομή σε ό,τι λέει ο κώδικας. Το να περνά κάθε αλλαγή από τον κώδικα είναι κανόνας της ομάδας, που τον στηρίζουν τα δικαιώματα. Στόχος 1b. Ξαναδιάβασε τα κεφάλαια 1 και 21.

Ερώτηση 28.50

Ορίζεις `TF_LOG=DEBUG` και τρέχεις `terraform plan > plan.txt`. Δεν ορίζεις `TF_LOG_PATH`. Πού βρίσκεις τα logs;

a. Στο `plan.txt`, ανάμεσα στις γραμμές του plan.
b. Στο αρχείο `terraform.log` του φακέλου εργασίας.
c. Στο terminal, γιατί τα logs γράφονται στο stderr.
d. Στο `.terraform/logs`, όπου τα κρατά το init.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το Terraform γράφει τα logs στο stderr και την κανονική έξοδο στο stdout. Το `>` στέλνει μόνο το stdout στο αρχείο, οπότε το `plan.txt` έχει μόνο το plan και τα logs μένουν στο terminal. Αρχείο για τα logs γράφεται μόνο αν ορίσεις `TF_LOG_PATH`. Προεπιλεγμένο `terraform.log` και φάκελος `.terraform/logs` δεν υπάρχουν. Στόχος 7c. Ξαναδιάβασε το κεφάλαιο 25.

Ερώτηση 28.51

Η configuration χρησιμοποιεί τον provider `vault`, που χρειάζεται τη διεύθυνση του server και ένα token. Ποιος τρόπος κρατά το token έξω από τα αρχεία του repository;

a. `token = "hvs.example"` μέσα στο block `provider "vault"`.
b. Άδειο block `provider "vault"` και οι μεταβλητές περιβάλλοντος `VAULT_ADDR` και `VAULT_TOKEN` στο shell ή στο pipeline.
c. Μια γραμμή `vault_token` στο `terraform.tfvars`, που μπαίνει σε Git.
d. Μια μεταβλητή `vault_token` με το token στο `default`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · b

Ο provider διαβάζει τη διεύθυνση και το token από το `VAULT_ADDR` και το `VAULT_TOKEN`, οπότε τίποτα από τα δύο δεν γράφεται σε κώδικα. Σε pipeline οι τιμές έρχονται από το σύστημα secrets του CI. Το a, το c και το d βάζουν το token σε αρχεία που διαβάζει όποιος έχει πρόσβαση στο repository. Μένει επιπλέον για πάντα στο ιστορικό του Git. Στόχος 4h. Ξαναδιάβασε το κεφάλαιο 24.

Ερώτηση 28.52

Το module `hashicorp/dir/template` έχει `version = ">= 1.0.0"`. Το init εγκατέστησε πριν από μήνες την 1.0.1. Σήμερα υπάρχει η 1.0.2. Τι κάνει ένα νέο `terraform init` στον ίδιο φάκελο;

a. Εγκαθιστά την 1.0.2, γιατί το init παίρνει πάντα τη νεότερη επιτρεπτή έκδοση.
b. Αποτυγχάνει, γιατί το lock file κλειδώνει την 1.0.1.
c. Κρατά την 1.0.1. Για την 1.0.2 χρειάζεται `terraform init -upgrade`.
d. Κατεβάζει και τις δύο εκδόσεις και αφήνει το plan να διαλέξει.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Το init δεν ξανακατεβάζει ένα module που είναι ήδη εγκατεστημένο και ικανοποιεί το `version`. Το `-upgrade` του λέει να ψάξει τη νεότερη επιτρεπτή έκδοση και γράφει `Downloading registry.terraform.io/hashicorp/dir/template 1.0.2`. Το a περιγράφει τι γίνεται σε καθαρό φάκελο, όπως ένα νέο checkout στο CI. Γι' αυτό σε κοινή υποδομή γράφεις ακριβή έκδοση. Το lock file δεν καταγράφει modules, οπότε το b είναι λάθος. Στόχος 5d. Ξαναδιάβασε το κεφάλαιο 16.

Ερώτηση 28.53

Μεταφέρεις το `aws_s3_bucket.logs` του root module μέσα σε ένα νέο module που καλείς με `module "storage"`. Μέσα στο module το resource λέγεται `aws_s3_bucket.this`. Ποιο block, γραμμένο στο root module, αποτρέπει την αντικατάσταση του bucket;

a. `moved` με `from = aws_s3_bucket.logs` και `to = aws_s3_bucket.this`
b. `moved` με `from = module.storage.aws_s3_bucket.this` και `to = aws_s3_bucket.logs`
c. `removed` με `from = aws_s3_bucket.logs`
d. `moved` με `from = aws_s3_bucket.logs` και `to = module.storage.aws_s3_bucket.this`

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · d

Το `from` είναι η παλιά διεύθυνση και το `to` η νέα, όπως τις βλέπει το module όπου είναι γραμμένο το block. Από το root η νέα διεύθυνση περιέχει το `module.storage`. Το plan γράφει `has moved to` και δεν αλλάζει τίποτα. Το a δείχνει σε resource του root που δεν υπάρχει. Το b έχει τις διευθύνσεις ανάποδα. Το `removed` θα έβγαζε το bucket από τη διαχείριση του Terraform ή θα το έσβηνε. Στόχος 6d. Ξαναδιάβασε το κεφάλαιο 23.

Ερώτηση 28.54

Σωστό ή λάθος; Το `terraform fmt` μορφοποιεί και τα αρχεία `.tfvars` του φακέλου, όχι μόνο τα `.tf`.

a. Σωστό
b. Λάθος

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το fmt επεξεργάζεται τα αρχεία `.tf` και `.tfvars` του φακέλου. Σε ένα `.tfvars` στοιχίζει τα `=` διαδοχικών γραμμών όπως σε ένα block. Χωρίς `-recursive` δεν μπαίνει σε υποφακέλους. Στόχος 3g. Ξαναδιάβασε το κεφάλαιο 2.

Ερώτηση 28.55

Το workspace `app-prod` διαβάζει outputs του `network-prod`. Κάθε φορά που αλλάζει το δίκτυο, το `app-prod` πρέπει να ξανατρέχει για να πάρει τις νέες τιμές. Τι ρυθμίζεις;

a. Ένα variable set με τα outputs του `network-prod`.
b. Ένα policy set που ελέγχει το `network-prod`.
c. Ένα run trigger στο `app-prod` με source workspace το `network-prod`.
d. Health assessments στο `app-prod`.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · c

Ένα run trigger βάζει run στην ουρά του `app-prod` μετά από κάθε επιτυχημένο apply του source workspace. Ένα variable set κρατά τιμές που γράφεις εσύ και δεν ενημερώνεται από outputs. Ένα policy set ελέγχει plans και δεν ξεκινά runs. Τα health assessments ελέγχουν το ίδιο το workspace και δεν αντιδρούν σε αλλαγές άλλου. Στόχος 8c. Ξαναδιάβασε το κεφάλαιο 26.

Ερώτηση 28.56

Μια ομάδα σκέφτεται να κρατά το `terraform.tfstate` στο ίδιο repository Git με τον κώδικα. Ποιοι δύο λόγοι το κάνουν κακή ιδέα; Επίλεξε δύο.

a. Το state μπορεί να περιέχει secrets σε απλό κείμενο.
b. Το Git δεν κλειδώνει το state, οπότε δύο συνάδελφοι μπορούν να κάνουν apply από παλιά αντίγραφα.
c. Το Terraform αρνείται να διαβάσει state από φάκελο που είναι repository Git.
d. Το state είναι δυαδικό αρχείο που το Git δεν αποθηκεύει.
e. Το Git σβήνει αυτόματα αρχεία που αλλάζουν σε κάθε commit.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a και b

Το state κρατά κάθε attribute των resources, μαζί με κωδικούς και κλειδιά, χωρίς κρυπτογράφηση. Ένα repository το διαβάζουν πολλοί και το ιστορικό του κρατά κάθε παλιά έκδοση. Το Git επίσης δεν ξέρει τίποτα για locking. Αν κάποιος ξεχάσει `git pull`, το apply του δουλεύει με παλιό state. Το Terraform δεν ελέγχει αν ο φάκελος είναι repository, οπότε το c είναι λάθος. Το state είναι κείμενο JSON, άρα το d είναι λάθος. Το e δεν ισχύει για κανένα αρχείο. Η λύση είναι remote backend με locking. Στόχος 2d. Ξαναδιάβασε τα κεφάλαια 4 και 19.

Ερώτηση 28.57

Ένας φάκελος έχει τοπικό `terraform.tfstate` με resources. Προσθέτεις block `cloud` με organization και workspace και θέλεις το state να μεταφερθεί στο workspace. Τι κάνεις;

a. Τρέχεις `terraform init` και απαντάς `yes` όταν ρωτήσει αν θα μεταφέρει το υπάρχον state.
b. Τρέχεις `terraform init -migrate-state`.
c. Τρέχεις `terraform state push terraform.tfstate` πριν από το init.
d. Ανεβάζεις το `terraform.tfstate` στο repository του workspace.

ΣΩΣΤΗ ΑΠΑΝΤΗΣΗ · a

Το init βλέπει ότι υπάρχει state και ρωτά αν θα το μεταφέρει. Με `yes` το state γίνεται η πρώτη έκδοση στο workspace. Η σημαία `-migrate-state` απορρίπτεται για το HCP Terraform με `Invalid command-line option`, γιατί η μεταφορά εκεί γίνεται με ερωτήσεις. Πριν από το init κάθε εντολή state σταματά και ζητά `terraform init`, οπότε το c είναι λάθος. Το state δεν μπαίνει ποτέ σε repository, άρα και το d είναι λάθος. Στόχος 8d. Ξαναδιάβασε το κεφάλαιο 26.

TERRAFORM · ΛΥΣΕΙΣ ΚΑΙ ΑΠΑΝΤΗΣΕΙΣ

Πρώτα η δική σου προσπάθεια. Μετά η σύγκριση.

Ο συνοδευτικός τόμος του βιβλίου Terraform. Κάθε άσκηση λύνεται με τον κώδικα, την πραγματική έξοδο και την εξήγηση του γιατί η λύση είναι σωστή. Κάθε ερώτηση έχει τη σωστή απάντηση και εξηγεί τι δεν στέκει σε κάθε άλλη επιλογή. Στην εξέταση οι λάθος επιλογές είναι συνήθως σχεδόν σωστές.

- 172 λύσεις, καθεμία εκτελεσμένη σε δικό της φάκελο
- 223 απαντήσεις με εξήγηση για κάθε επιλογή
- η βαθμολόγηση της προσομοίωσης εξέτασης του κεφαλαίου 28



ΕΚΔΟΣΗ 1.0 · ΟΚΤΩΒΡΙΟΣ 2026