Infrastructure as Code cu Bicep & Terraform

Ghid didactic complet în limba română — **zero to hero** pentru Azure-first engineers

WEBINAR · CURS COMPLET

Ce vei învăța în acest curs

Scopul acestui material este să treacă studentul de la deploy manual în Azure Portal la gândire de platformă, modularitate, control al schimbărilor, review, versionare și automatizare repetabilă cu Bicep și Terraform.

01

Resurse Azure & deploy manual

Înțelegi ce construiești înainte de a automatiza.

02

Bicep — IaC nativ Azure

Înveți IaC cu fricțiune mică, sintaxă modernă și module.

03

Terraform — state, backend, multi-cloud

Înțelegi state, backend, module enterprise și multi-cloud thinking.

04

Pipelines, policy & day-2 operations

CI/CD, monitoring, observabilitate și producție sigură.

Infrastructure as Code în Azure — ce alegi și când?

Bicep, Terraform și ARM Templates explicate vizual pentru studenți și ingineri cloud.

Cele trei opțiuni principale

ARM Templates

JSON nativ pentru Azure + control total + compatibilitate maximă. Verbose, greu de citit. **Bun când moștenești template-uri vechi sau ai nevoie de JSON pur.**

Bicep

DSL modernă peste ARM + sintaxă curată + modules și reuse + excelent pentru Azure. Focus pe Azure. **Alegerea recomandată pentru proiecte noi Azure-first.**

Terraform

Tool multi-cloud bazat pe state + standard de industrie + Azure + AWS + GCP + ecosistem bogat. State de gestionat. **Foarte bun pentru platforme mari și echipe mixte.**

Regula simplă de decizie

- Vrei doar Azure și vrei experiența cea mai simplă?

  → **Bicep**

- Ai deja Terraform în companie sau lucrezi multi-cloud?

  → **Terraform**

- Ai template-uri JSON vechi sau exportate din Azure?

  → **ARM Templates**

- Vrei modularitate, standardizare și module verificate?

  → **Bicep sau Terraform cu AVM**

1. De ce Infrastructure as Code este o abilitate fundamentală

Dacă Azure Portal este volanul cu care înveți să conduci, **Infrastructure as Code este pilotul automat, cartea tehnică a mașinii și jurnalul de service în același timp.** Cu IaC descrii infrastructura în fișiere text, o versionezi în Git, o revizuiești prin pull requests, o validezi în pipeline și o refaci identic de câte ori ai nevoie.

Beneficiile cheie ale IaC

Replicabilitate

Același mediu poate fi recreat în dev, test și prod fără click-uri manuale diferite.

Audit și guvernanță

Orice schimbare este vizibilă în cod, nu doar în memoria celui care a făcut-o.

Viteză

Infrastructură complexă poate fi lansată în minute, nu în ore sau zile.

Standardizare

Naming, tags, securitate și politici pot fi aplicate consecvent.

Disaster recovery & onboarding

Un nou coleg sau un nou proiect pornește mult mai repede.

Analogii utile pentru studenți

> Portalul este ca și cum ai mobila o casă mutând fiecare obiect manual. **IaC este planul complet al casei plus lista exactă de materiale.**

> Un deploy manual poate funcționa o dată. **IaC funcționează de fiecare dată, pentru că procesul este documentat executabil.**

> Un screenshot îți arată cum arată infrastructura. **Codul IaC îți permite să o refaci.**

Mesajul-cheie: Portalul te ajută să înveți. IaC te ajută să scalezi, să repeți, să auditezi și să livrezi fără surprize.

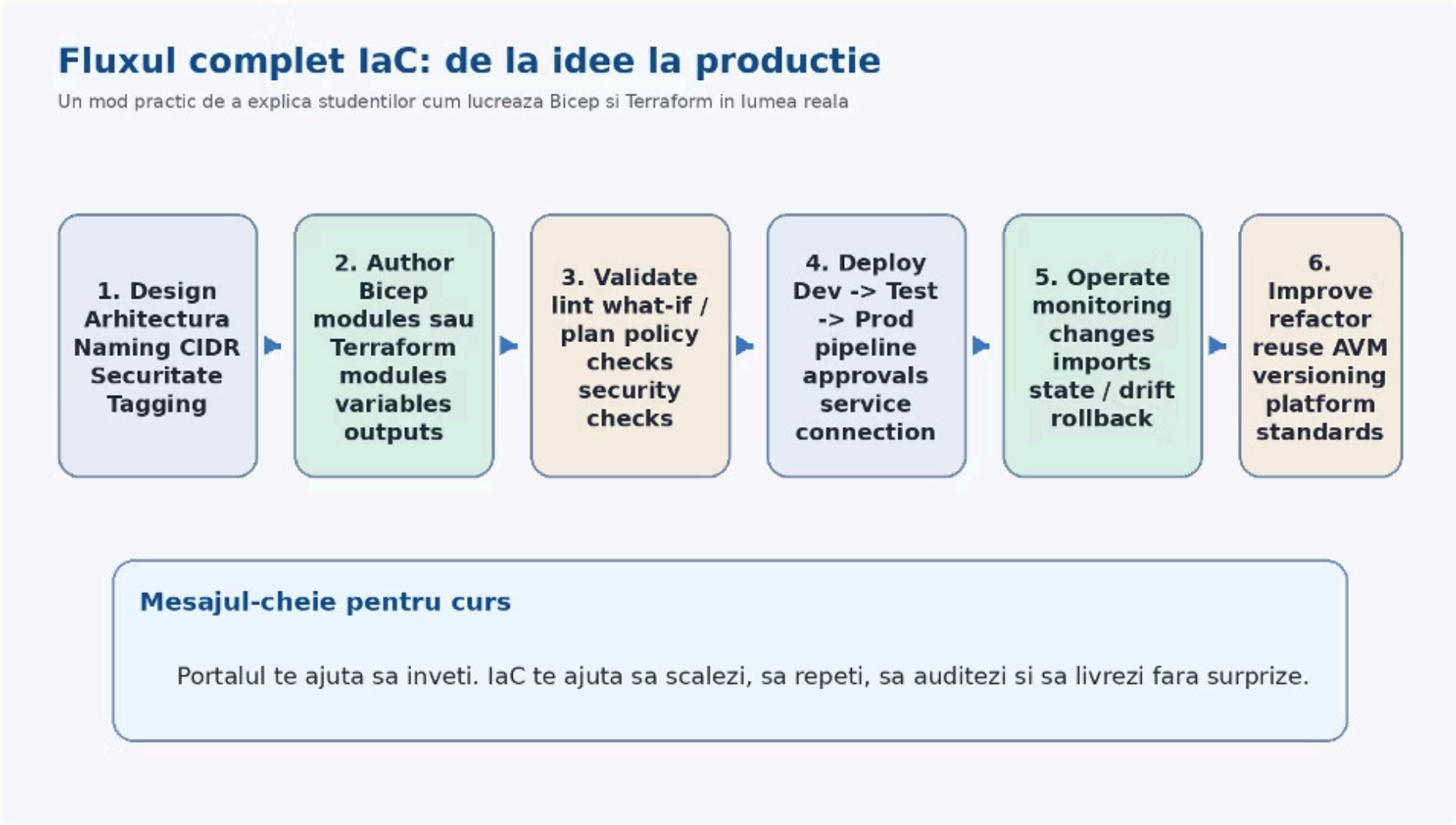
2. ARM Templates vs Bicep vs Terraform — comparație clară

Criteriu	ARM Templates	Bicep	Terraform
Sintaxă	JSON, verbos, greu de menținut la mână	DSL compactă, lizibilă, transpilează în ARM JSON	HCL, lizibil, orientat pe modules și providers
Scop principal	Azure nativ, model vechi dar încă valid	Azure nativ modern, recomandat pentru proiecte noi Azure-first	Multi-cloud și enterprise IaC standard
State	Nu folosește fișier de state separat; ARM calculează desired state la deploy	La fel ca ARM, fără state separat	Are state explicit; local sau remote backend
Modularitate	Posibilă dar greoaie prin nested templates	Foarte bună prin modules, registries și AVM	Foarte bună prin modules, registry și compositions
Cea mai bună alegere când...	Moștenești JSON existent sau ai output exportat	Vrei cea mai bună experiență în Azure și sintaxă modernă	Ai standard corporativ Terraform sau mediu multi-cloud

Concluzia practică: Dacă organizația este Azure-first și nu are un standard impus pe Terraform, Bicep este în general cea mai simplă și naturală alegere pentru proiecte noi. Microsoft recomandă Bicep peste ARM JSON pentru authoring nou. Terraform rămâne excelent când standardizarea cross-cloud sau ecosistemul de module deja existent contează mai mult.

Fluxul complet IaC: de la idee la producție

Un mod practic de a explica studenților cum lucrează Bicep și Terraform în lumea reală.



1

1. Design

Arhitectură, Naming, CIDR, Securitate, Tagging

2

2. Author

Bicep modules sau Terraform modules, variables, outputs

3

3. Validate

lint, what-if / plan, policy checks, security checks

4

4. Deploy

Dev → Test → Prod, pipeline, approvals, service connection

5

5. Operate

monitoring, changes, imports, state / drift, rollback

6

6. Improve

refactor, reuse AVM, versioning, platform standards

3. Concepte-cheie pe care studentul trebuie să le stăpânească

Declarativ vs Imperativ

Bicep, ARM și Terraform descriu starea dorită. Nu spui pas cu pas ce să execute operatorul, ci ce resurse vrei să existe și cu ce configurație.

Idempotență

Rulezi același cod de mai multe ori și ajungi la aceeași stare. Asta reduce surprizele.

Drift

Dacă cineva modifică manual o resursă în portal și codul nu reflectă schimbarea, apare drift. Terraform detectează drift prin state + refresh/plan; Bicep se bazează mai mult pe what-if și guvernanță.

Modules

Un modul este o bucată reutilizabilă de infrastructură: de exemplu un VNet standard, un storage account standard sau un set complet App Service + plan + insights.

Parameters / Variables

Separați ce se schimbă între medii de codul comun. De exemplu: nume, locație, SKU, CIDR, număr de noduri.

Outputs

Expui valori pentru consum ulterior: resource IDs, endpoint-uri, subnet IDs, connection strings non-sensitive etc.

Dependencies

Resursele trebuie create în ordinea corectă, dar limbajele declarative rezolvă mult din ordonare automat atunci când referențiezi resursele corect.

4. ARM Templates pe înțelesul tuturor

ARM template-ul este formatul JSON nativ al platformei Azure Resource Manager. Gândește-l ca pe **limba oficială a control plane-ului Azure**. Este puternic, dar destul de zgomotos pentru authoring manual.

Proiecte vechi cu JSON

Proiecte care deja folosesc ARM JSON și nu justifică migrarea imediată.

Tool-uri externe

Scenarii unde tool-uri externe exportă sau generează ARM JSON automat.

Inspecție payload

Cazuri în care vrei să inspectezi exact payload-ul final care merge spre Azure.

Punct important: Bicep nu este un competitor separat de ARM, ci un strat de authoring mai prietenos care se **transpilează în ARM JSON** la deploy. Asta înseamnă că investiția în Bicep rămâne nativă pentru Azure, nu ocolește platforma.

5. Bicep în profunzime

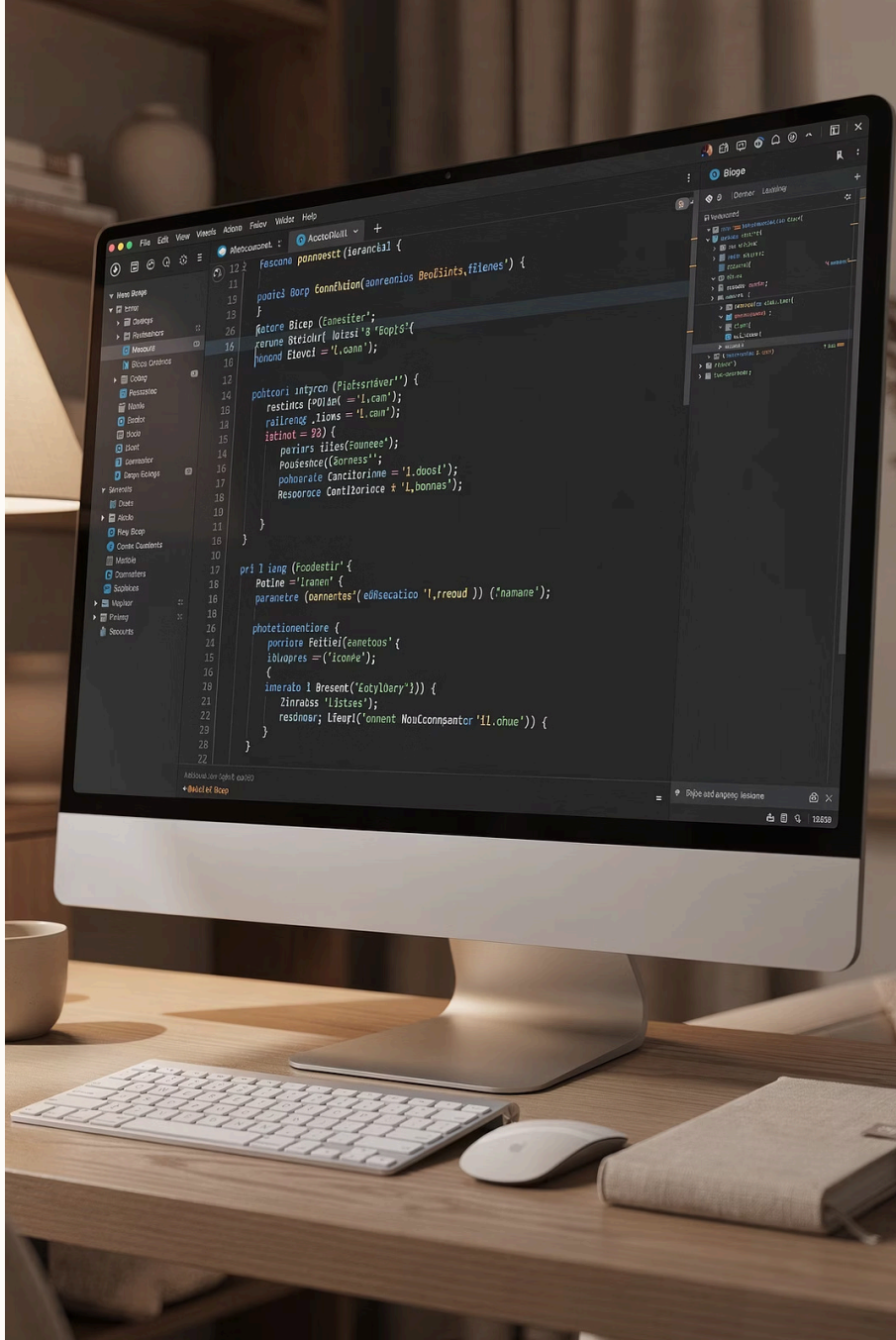
Bicep este limbajul IaC recomandat pentru Azure-first. El păstrează modelul declarativ ARM, dar reduce enorm complexitatea sintaxei. În loc să scrii JSON cu proprietăți repetitive, lucrezi într-o sintaxă mai apropiată de un limbaj modern: clară, compactă și ușor de modulizat.

Modules, params, vars, outputs & loops

Azure CLI, PowerShell, VS Code & pipelines

Scope-uri: resource group, subscription, management group, tenant

Compatibilitate directă cu Azure Verified Modules (AVM)



5.1 Anatomia unui fișier Bicep simplu

Ce învață studentul aici: target scope, parametri, variabile, nume determinist, definire de resursă și output. Este suficient pentru a înțelege scheletul Bicep.

```
targetScope = 'resourceGroup'

@description('Locatia deployment-ului')
param location string = resourceGroup().location

@description('Prefix de nume')
param prefix string

var storageName = toLower('st${uniqueString(resourceGroup().id, prefix)}')

resource sa 'Microsoft.Storage/storageAccounts@2025-01-01' = {
 name: storageName
 location: location
 sku: {
 name: 'Standard_LRS'
 }
 kind: 'StorageV2'
 properties: {
 minimumTlsVersion: 'TLS1_2'
 allowBlobPublicAccess: false
 }
}

output storageAccountName string = sa.name
```

5.2 Primul modul Bicep — structura de foldere

Un curs bun nu se oprește la un fișier monolitic. Studentul trebuie să învețe să spargă infrastructura în module. Mai jos este un exemplu realist: o arhitectură mică Azure cu Log Analytics, VNet, subnet și NSG. Părintele orchestrează, modulele livrează componente reusable.

Structura recomandată de foldere

```
iac-bicep/
  main.bicep
  main.dev.bicepparam
  modules/
    loganalytics.bicep
    network.bicep
    nsg.bicep
```

Cum rulezi

```
az group create \
 -n rg-iac-lab \
 -l swedencentral

az deployment group create \
 -g rg-iac-lab \
 -f main.bicep \
 -p prefix=lab01
```

5.2 Modul: loganalytics.bicep & network.bicep

modules/loganalytics.bicep

```
param workspaceName string
param location string

resource law 'Microsoft.OperationalInsights/workspaces@2023-09-01' = {
  name: workspaceName
  location: location
  properties: {
    sku: {
      name: 'PerGB2018'
    }
    retentionInDays: 30
  }
}

output workspaceId string = law.id
```

modules/network.bicep

```
param vnetName string
param location string
param addressSpace string
param subnetName string
param subnetPrefix string
param nsgId string

resource vnet 'Microsoft.Network/virtualNetworks@2024-05-01' = {
 name: vnetName
 location: location
 properties: {
 addressSpace: {
 addressPrefixes: [ addressSpace ]
 }
 subnets: [
 {
 name: subnetName
 properties: {
 addressPrefix: subnetPrefix
 networkSecurityGroup: { id: nsgId }
 }
 }
 ]
 }
}

output subnetId string = resourceId(
 'Microsoft.Network/virtualNetworks/subnets',
 vnet.name, subnetName)
```

5.2 Fișierul orchestrator: main.bicep

```
param location string = resourceGroup().location
param prefix string = 'demo'

module nsg './modules/nsg.bicep' = {
 name: 'nsgDeploy'
 params: {
 nsgName: '${prefix}-app-nsg'
 location: location
 }
}

module law './modules/loganalytics.bicep' = {
 name: 'lawDeploy'
 params: {
 workspaceName: '${prefix}-law'
 location: location
 }
}

module net './modules/network.bicep' = {
 name: 'networkDeploy'
 params: {
 vnetName: '${prefix}-vnet'
 location: location
 addressSpace: '10.40.0.0/16'
 subnetName: 'app'
 subnetPrefix: '10.40.1.0/24'
 nsgId: nsg.outputs.nsgId
 }
}

output subnetId string = net.outputs.subnetId
output workspaceId string = law.outputs.workspaceId
```

5.3 Bicep best practices care chiar contează

Folosește .bicepparam pentru medii diferite

În loc să dublezi codul pentru dev, test și prod, folosește fișiere `.bicepparam` separate.

Păstrează module mici și coerente

Un modul pentru rețea, unul pentru monitoring, unul pentru compute etc. Coerența facilitează reutilizarea.

Nu hardcode secrete

Leagă-te de Key Vault sau injectează valori în pipeline în mod securizat. Niciodată parole în fișierele IaC.

Rulează validation, linter și what-if înainte de producție

Validarea preventivă reduce riscul de surprize în mediile critice.

Versionează modulele interne

Evită schimbări breaking fără comunicare. Tratează modulele ca pe API-uri cu versiuni.

6. Terraform în profunzime

Terraform este foarte popular pentru că nu este limitat la Azure. Cu același model mental poți descrie Azure, AWS, GitHub, Datadog, Cloudflare și multe altele. Pentru companii mari, acesta este un avantaj major: **un singur limbaj de automatizare pentru întregul ecosistem.**

provider

Plugin-ul care știe să vorbească cu o platformă, de exemplu `azurerm`.

state

Fișierul care ține evidența resurselor administrate.

backend

Locul unde state-ul este stocat (local sau remote).

module

Componentă reutilizabilă, similară conceptual cu modulele Bicep.

plan

Previzualizarea schimbărilor înainte de apply.

6.1 Anatomia unui proiect Terraform simplu

```
terraform {
 required_version = ">= 1.7.0"

 required_providers {
 azurerm = {
 source = "hashicorp/azurerm"
 version = ">= 4.0"
 }
 }
}

provider "azurerm" {
 features {}
}

resource "azurerm_resource_group" "rg" {
 name = "rg-tf-demo"
 location = "swedencentral"
}

resource "azurerm_storage_account" "sa" {
 name = "stfdemo001234"
 resource_group_name = azurerm_resource_group.rg.name
 location = azurerm_resource_group.rg.location
 account_tier = "Standard"
 account_replication_type = "LRS"
 min_tls_version = "TLS1_2"
}
```

6.2 State management în Terraform — partea cea mai importantă

Dacă studentul reține un singur lucru despre Terraform, acesta trebuie să fie **state-ul**. Terraform nu doar trimite comenzi spre Azure; el menține o hartă a resurselor pe care le administrează. Fără el, Terraform nu știe corect ce există deja, ce trebuie schimbat și ce trebuie șters.

Model	Avantaje	Riscuri / observații
Local state	Simplu pentru laborator sau PoC individual	Nu este bun pentru echipe; risc de pierdere, conflicte și lipsă de locking centralizat
Remote state în Azure Storage	Potrivit pentru echipe; acces centralizat, locking, backup și control mai bun	Necesită bootstrap corect: storage account, container, RBAC, securizare acces

Exemplu backend azurerm

```
terraform {
 backend "azurerm" {
 resource_group_name = "rg-tf-state"
 storage_account_name = "sttfstateprod01"
 container_name = "tfstate"
 key = "platform/dev.tfstate"
 }
}
```

6.2 State locking și Workspaces

De ce contează locking-ul

Când două persoane rulează `terraform apply` în același timp, state locking previne coruperea state-ului. Backend-ul `azurerm` folosește capabilitățile native Azure Blob Storage pentru locking și consistency checking. Dacă locking-ul eșuează, Terraform nu ar trebui forțat decât după investigație; altfel riști inconsistent state.

Workspaces în Terraform

Workspaces separă state-ul logic pentru aceeași configurație. Pot fi utile pentru medii simple, dar nu sunt o scuză pentru a amesteca totul într-un singur repo fără structură. În producție, multe echipe preferă foldere separate, backends separate și pipelines separate pentru claritate.

În practică, storage account-ul pentru state trebuie tratat ca **infrastructură critică**: acces minim necesar, versioning, soft delete pentru blobs, RBAC clar și separare pe medii. Pentru producție, state-ul nu ar trebui ținut într-un laptop local.

6.3 Primul modul Terraform

```
module "network" {
  source              = "./modules/network"
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.rg.location
  vnet_name           = "vnet-platform-dev"
  address_space       = ["10.50.0.0/16"]
  subnet_name         = "app"
  subnet_prefixes     = ["10.50.1.0/24"]
}
```

Mesajul-cheie: Modulul nu trebuie să facă prea multe lucruri diferite. O componentă coerentă este mai ușor de testat, versionat și reutilizat.

7. Azure Verified Modules (AVM) — de ce sunt importante

Azure Verified Modules sunt module validate și susținute într-un model consistent pentru deploy și management de resurse Azure. Există variante atât pentru Bicep, cât și pentru Terraform. Pentru studenți, AVM sunt o punte excelentă între teoria IaC și standardizarea enterprise: înveți să nu reinventezi de fiecare dată roata pentru resurse comune.

Accelerează adoptia IaC

Componente preconstruite care reduc timpul de implementare pentru resurse standard.

Naming, outputs și pattern-uri consistente

Încurajează standardizarea la nivel de organizație.

Arhitecturi standardizate rapide

Utile când vrei să livrezi rapid arhitecturi standardizate fără boilerplate.

Nu elimină nevoia de design

AVM reduc munca repetitivă, dar nu înlocuiesc înțelegerea resurselor de bază.

7.1 Când să folosești AVM și când nu

✅ Folosește AVM când...

- Vrei viteză și consistență pe resurse standard
- Echipa are mai multe proiecte similare
- Vrei să reduci boilerplate-ul

⚠️ Evită să te bazezi orbește pe AVM când...

- Nu înțelegi încă ce face modulul sub capotă
- Ai nevoie de comportamente foarte custom și nevalidate
- Studentul nu a învățat mai întâi bazele resursei pe care o creează

> **Regulă bună:** Învață mai întâi resursa, apoi accelerează cu module verified.

8. Deploying multi-resource architectures

Aici începe zona cu adevărat valoroasă pentru un curs. Un student știe că a înțeles IaC abia când poate codifica o arhitectură mai mare, nu doar o resursă izolată.

Rețea completă

VNet, subnets, NSG, route tables, private DNS links

Platformă de aplicație

App Service Plan, Web App, Key Vault, Application Insights, Log Analytics

AKS foundation

VNet, subnete dedicate, Log Analytics, managed identity, private DNS / private endpoints

Landing zone mică

Resource groups, tags, policy assignments, role assignments, monitoring, network baseline

8.1 Pattern de arhitectură recomandat pentru studenți

Împarte infrastructura pe layere. Gândește în module și responsabilități, nu într-un fișier uriaș. Această împărțire face codul mai ușor de citit și ajută și la pipeline-uri: unele layere se schimbă rar, altele mai des.

1

Foundation

Resource groups, locks, tags, role assignments de bază

2

Network

VNet, subnets, NSG, UDR, private DNS

3

Shared Services

Monitoring, Key Vault, registries, bastion, firewall dacă există

4

Workload

AKS, App Service, Function, SQL, Storage etc.

8.2 Ce se întâmplă în lumea reală

Mediu nou lansat rapid pentru un client nou

IaC permite clonarea controlată a arhitecturii. Un mediu complet poate fi recreat în minute, nu în zile de click-uri manuale.

Audit cere dovadă de conformitate

Cu IaC vezi imediat standardul și poți aplica remedieri la scară: de exemplu, că storage account-urile au TLS minim și public access dezactivat.

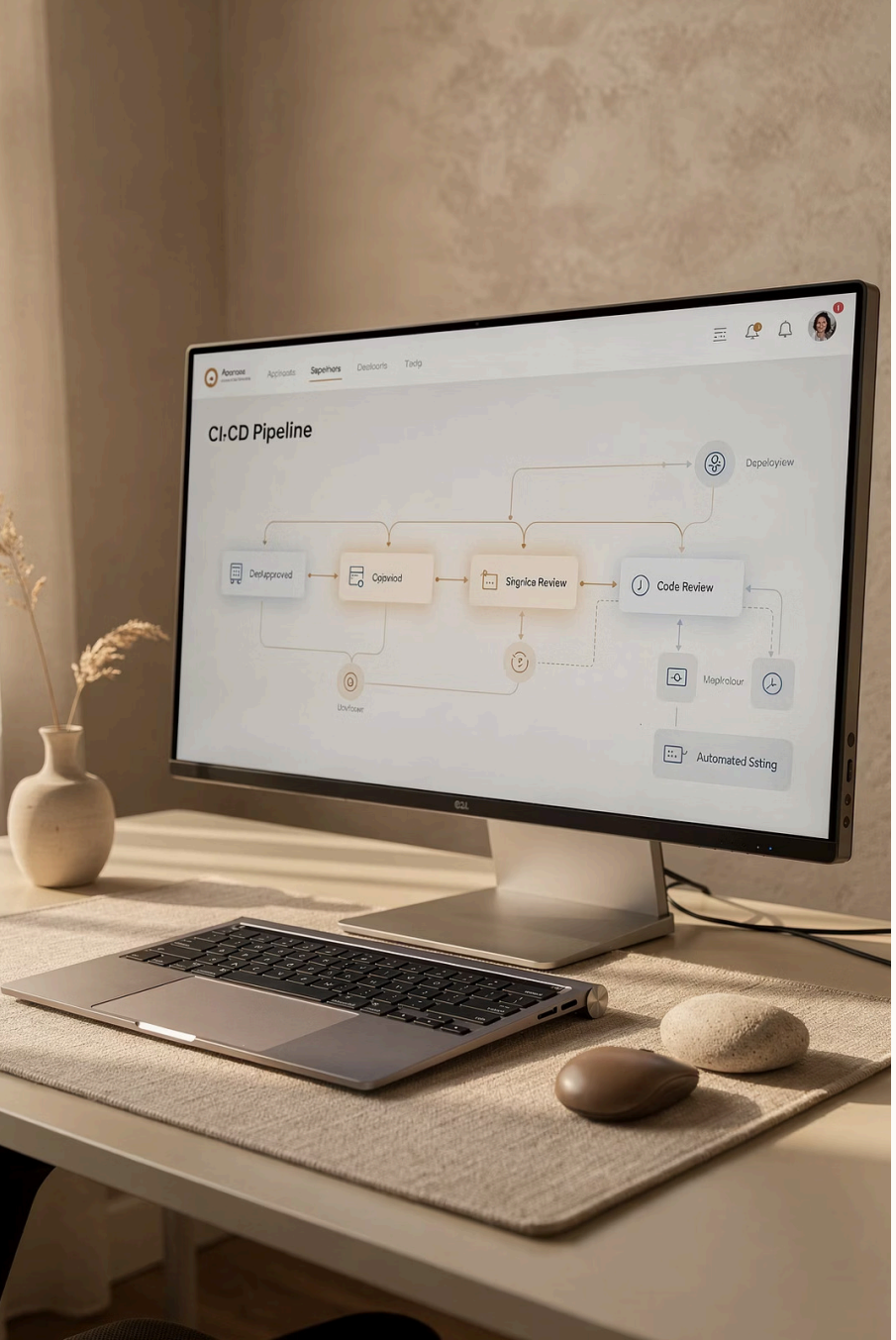
Un coleg a modificat ceva manual în portal

Prin code review și deploy-uri controlate reduci șansele ca acea schimbare să rămână nedocumentată și să creeze drift.

9. Cum aleg între Bicep și Terraform într-o organizație reală

Nu există un răspuns universal, dar există criterii foarte clare.

Întrebare	Indicator pentru Bicep	Indicator pentru Terraform
Platforma este aproape exclusiv Azure?	Da, Bicep este foarte natural	Poate, dar avantajul Terraform scade dacă nu folosești și alte providere
Aveți echipă platform engineering cu standard Terraform deja matur?	Poate fi folosit doar în proiecte punctuale	Da, continuitatea operațională contează mult
Vreți authoring foarte apropiat de Azure ARM?	Da	Nu este prioritatea lui principală
Vreți același limbaj pentru Azure + GitHub + Datadog + alte SaaS?	Mai puțin potrivit	Foarte potrivit
Juniorii trebuie să învețe repede Azure IaC?	Bicep este adesea mai simplu pentru start	Terraform merită imediat după ce înțeleg bazele



10. CI/CD pentru IaC

IaC fără pipeline este mai bun decât click-ops, dar adevărata valoare apare când codul trece printr-un flux controlat: lint, validate, what-if sau plan, approvals, deploy și verificare post-deploy.

Bicep

bicep build/lint → az deployment what-if → deployment create

Terraform

terraform fmt → validate → plan → aprobare plan → apply

Separare pe medii

Dev automatizat mai agresiv, prod cu approvals și change window.

Identități gestionate

Service connections și identități gestionate în mod minim necesar.

10.1 Exemplu de pipeline logic — 4 stage-uri

1	2
Stage 1 — Validate • lint / fmt • security checks • dependency checks	Stage 2 — Preview • Bicep what-if sau Terraform plan • publish plan artefact
3	**4**
Stage 3 — Deploy Dev • deploy automat • smoke tests	Stage 4 — Deploy Prod • approval manual • deploy controlat • post-deploy validation

11. Azure Portal în povestea IaC

Deși Bicep și Terraform se rulează de obicei prin CLI, DevOps sau Cloud Shell, Azure Portal rămâne util pentru învățare și bootstrap.

Bootstrap inițial

Creezi subscription-level resurse inițiale pentru laborator: un resource group sau un storage account pentru Terraform state.

Inspecție rapidă

Inspectezi rapid resursele create de cod și compari rezultatul deploy-ului cu designul intenționat.

Exerciții simple

Folosești *Deploy a custom template* pentru ARM/Bicep în anumite exerciții introductive.

Important pedagogic: Portalul nu dispare. El devine instrument de observare și verificare, nu instrument principal de producție.

12. Laborator complet — zero to hero

Obiectivul laboratorului

Studentul va construi aceeași arhitectură mică în două moduri: o dată cu Bicep, o dată cu Terraform. La final va înțelege diferența dintre transpile-to-ARM și state-based IaC.

Arhitectura laboratorului

1 Resource Group de laborator	1 Log Analytics Workspace	1 VNet 10.60.0.0/16	1 Subnet app 10.60.1.0/24
1 NSG asociat subnetului		Optional: 1 Storage Account securizat	

12.3 Pașii pentru Bicep

01

Creează resource group-ul de laborator

```
az group create -n rg-iac-lab -l swedencentral
```

02

Construiește modulele

Module pentru workspace, nsg și network conform structurii de foldere recomandate.

03

Leagă output-urile între module

Asigură-te că nsgId din modulul NSG este transmis corect modulului network.

04

Rulează what-if

```
az deployment group what-if -g rg-iac-lab -f main.bicep -p prefix=lab01
```

05

Rulează deployment create

```
az deployment group create -g rg-iac-lab -f main.bicep -p prefix=lab01
```

06

Verifică în portal

Confirmă că toate resursele există și sunt configurate corect.

12.4 Pașii pentru Terraform

01

Creează storage account-ul pentru remote state

Tratează-l ca infrastructură critică: RBAC, versioning, soft delete.

02

Configurează backend azurerm

Definește `resource_group_name`, `storage_account_name`, `container_name` și `key`.

03

Scrie resursele sau modulele echivalente

Aceeași arhitectură ca la Bicep: Log Analytics, VNet, subnet, NSG.

04

Rulează init, fmt, validate, plan și apply

`terraform init` → `terraform fmt -recursive` → `terraform validate` → `terraform plan -out=tfplan` → `terraform apply tfplan`

05

Compară experiența

Observă diferențele: plan-ul, state-ul și drift awareness față de experiența Bicep.

12.5 Ce trebuie să observe studentul

Bicep

Se simte mai apropiat de modelul nativ Azure. Transpilează în ARM JSON, fără state separat. Experiența este mai directă pentru Azure-only.

Terraform

Se simte mai apropiat de un orchestrator universal cu state explicit. Plan-ul oferă o previzualizare clară, iar drift awareness este mai pronunțat.

Ambele pot produce aceeași infrastructură, dar **experiența operațională este diferită**. Înțelegerea acestei diferențe este esența laboratorului.

13. Troubleshooting și greșeli frecvente

Naming invalid

Unele resurse Azure au reguli stricte pentru nume. Folosește funcții de normalizare și convention-over-configuration.

Order / dependency issues

Nu folosi `dependsOn` inutil. În Bicep referințele directe rezolvă de multe ori ordinea. În Terraform folosește referințe între resurse în loc de hardcoded strings.

State blocat în Terraform

Investighează de ce există lock. Nu sări direct la `force-unlock` fără să înțelegi contextul.

Drift din portal

Evită modificările manuale pe resurse administrate prin cod; dacă sunt necesare, readuce-le apoi în cod.

Secrete în repo

Nu pune parole, chei sau connection strings sensibile direct în fișierele IaC sau tfvars neprotejate.

Module prea mari

Când un modul face prea multe lucruri, devine greu de reutilizat și testat. Păstrează coerența și responsabilitatea unică.

14. Bune practici de network engineer & platform engineer aplicate în IaC

Planifică CIDR și subnetting înainte de primul deploy

Nu după. Schimbările de adresare ulterior sunt costisitoare și riscante.

Tagging standard

Owner, environment, cost center, criticality — aplicate consecvent prin cod, nu manual.

Separare foundation / workload

Layere cu cicluri de viață diferite nu ar trebui să fie în același modul sau pipeline.

Aprobări pentru producție

Ferestre de schimbare pentru medii critice și approvals manuale înainte de apply în prod.

Versionează modulele

Documentează breaking changes. Tratează modulele ca pe API-uri cu versiuni clare.

Observabilitate ca parte din cod

Diagnostics, logs, alerts, workspace links — nu adăugate după, ci incluse în modulele de resurse.

15. Cheat sheet rapid pentru studenți

Întrebare	Răspuns practic
Vreau Azure-only și simplu	Începe cu Bicep
Vreau multi-cloud sau standard enterprise existent	Învață Terraform
Vreau să reutilizez infrastructura	Folosește module
Vreau previzualizare schimbări	Bicep what-if sau Terraform plan
Vreau consistență pe echipă	Remote state, pipelines, approvals, code review
Vreau accelerare	Uită-te la Azure Verified Modules
Vreau producție sigură	Separare pe medii, RBAC, Key Vault, approvals, observabilitate

17. Comenzi utile — referință rapidă

Bicep

```
az bicep upgrade

az deployment group what-if \
  -g rg-demo \
  -f main.bicep \
  -p @main.dev.bicepparam

az deployment group create \
  -g rg-demo \
  -f main.bicep \
  -p @main.dev.bicepparam
```

Terraform

```
terraform init
terraform fmt -recursive
terraform validate
terraform plan -out=tfplan
terraform apply tfplan
terraform output
```

16. Concluzie — de la operator la cloud engineer

Studentul care înțelege Bicep și Terraform nu mai este doar un operator de portal. Devine un inginer capabil să descrie infrastructura ca produs: **repetabil, revizuibil, auditabil și pregătit pentru scală.** Exact această tranziție face diferența între un administrator care apasă butoane și un cloud engineer care construiește platforme.

01

Resursele Azure & deploy manual

Ca să înțelegi ce construiești.

02

Bicep

Pentru a învăța IaC nativ Azure cu fricțiune mică.

03

Terraform

Pentru a înțelege state, backend, module enterprise și multi-cloud thinking.

04

Pipelines, policy, monitoring & day-2 operations

Pentru producție sigură și scalabilă.