Azure DevOps & CI/CD Pipelines

Ghid complet în limba română — **zero to hero** pentru junior cloud engineers

Versiune didactică orientată spre Azure Portal și Azure DevOps Services

WEBINAR CURS COMPLET

De ce contează acest curs

Problema reală

Mulți oameni știu să creeze o resursă în Azure, dar mult mai puțini înțeleg cum ajunge codul în producție în mod **repetabil, controlat și sigur**. Aici apare adevărata diferență dintre un administrator cloud și un inginer cloud modern.

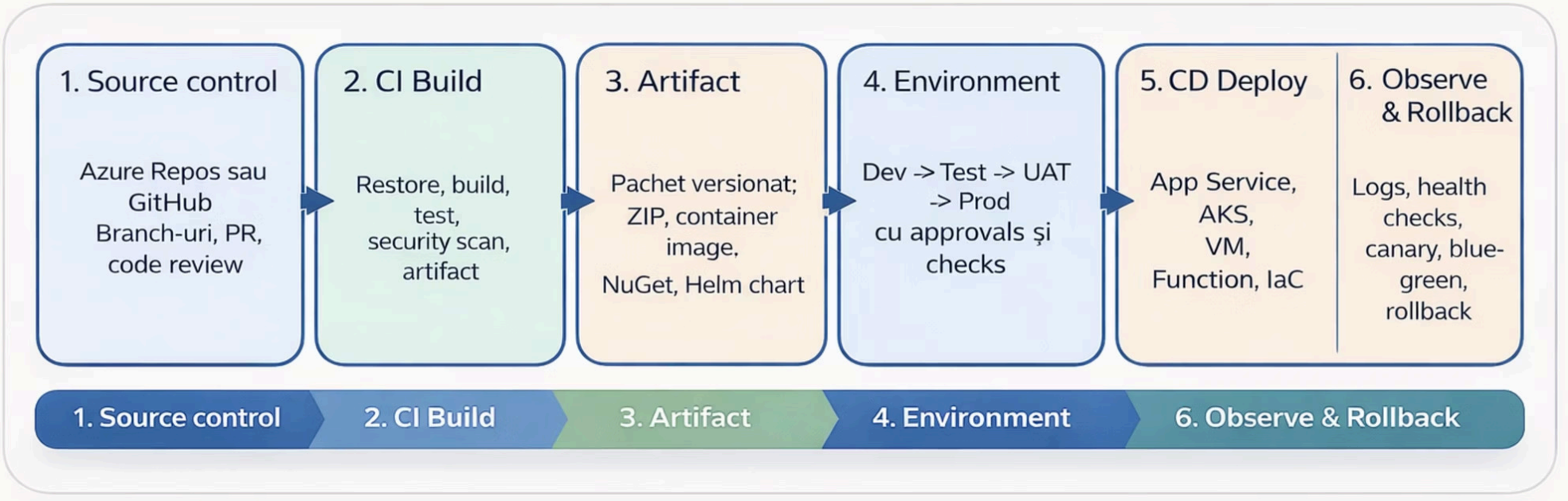
Ce schimbă CI/CD

CI/CD nu este doar automatizare. Este disciplina prin care:

- Reduci riscul de schimbare
- Crești viteza de livrare
- Obții trasabilitate completă
- Standardizezi între echipe
- Îmbunătățești compliance-ul

Stăpânirea acestui subiect schimbă complet profilul profesional: din om care «știe Azure» în om care poate **livra în Azure în mod responsabil**.

Fluxul complet CI/CD — de la cod la producție



1

1. Source Control

Azure Repos sau GitHub, Branch-uri, PR, code review

2

2. CI Build

Restore, build, test, security scan, artifact

3

3. Artifact

Pachet versionat: ZIP, container image, NuGet, Helm chart

4

4. Environment

Dev → Test → UAT → Prod cu aprobări și checks

5

5. CD Deploy

App Service, AKS, VM, Function, IaC

6

6. Observe & Rollback

Logs, health checks, canary, blue-green, rollback

Idee esențială: Separă clar CI de CD. CI dovedește că aplicația se poate construi și testa. CD dovedește că aplicația se poate livra repetabil, controlat și auditabil.

Analogii practice — înțelege CI/CD vizual

🏭 Fabrica auto

Azure DevOps este **banda de producție a fabricii**. Repozitoriul este depozitul de piese, build-ul este linia de asamblare, artifact-ul este produsul ambalat, iar release-ul este camionul care livrează în magazine.

🏙️ Orașul și logistica

Azure este **orașul în care rulează aplicația**. Azure DevOps este sistemul logistic care decide cum se construiește marfa, cine o aprobă, ce rută urmează și cum ajunge în magazin fără să se strice pe drum.

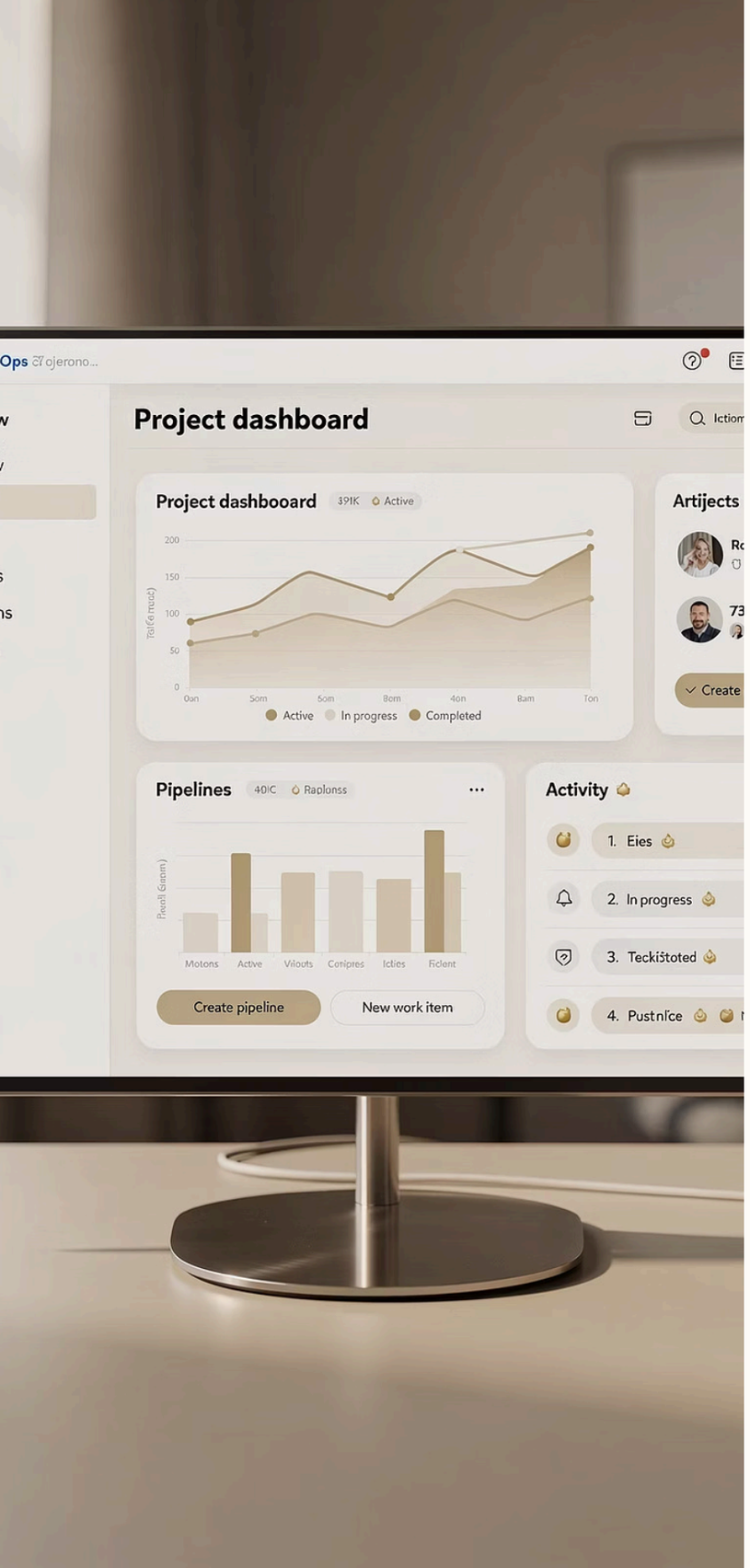
⚠️ Fără CI/CD

Fiecare deploy devine un mic proiect haotic:

- Un coleg compilează local
- Alt coleg copiază fișiere manual
- Altcineva modifică o setare în portal
- Dacă apare o problemă, nimeni nu știe ce s-a schimbat

✅ Cu CI/CD

Schimbarea devine o **rutină controlată**: construiești, testezi, ambalezi, aprobi, livrezi, observi și poți reveni rapid. Beneficii: mai puține erori umane, timp mai mic până la livrare, standardizare, compliance mai bun și colaborare clară între Dev, QA, Security și Ops.



Ce este Azure DevOps — pe scurt

Azure DevOps este o **suită de servicii** pentru planificare, versionare, build, test și livrare. Valoarea reală apare când conectezi codul, work items, artefactele și deploy-urile într-un singur fir logic.

1

Azure Boards

Backlog, work items, sprint-uri și trasabilitate completă a cerințelor.

2

Azure Repos

Git hosting nativ în Azure DevOps, cu suport pentru branch-uri, PR și code review.

3

Azure Pipelines

Automatizare CI/CD, atât YAML cât și Classic. Inima întregului flux de livrare.

4

Service Connections, Environments & Library

Elementele de control și integrare necesare pentru livrare sigură și auditabilă.

Conceptele fundamentale — dicționar esențial

Organization	Containerul principal din Azure DevOps. Conține proiecte, utilizatori, permisiuni și servicii.
Project	Spațiul de lucru pentru o aplicație, o platformă sau un produs.
Repository	Locul în care stă codul sursă. Poate fi Azure Repos sau GitHub.
Pipeline	Fluxul automat care execută build, test și deployment.
Stage	Grup logic mare: Build, Test, Deploy-Dev, Deploy-Prod.
Job	Unitate executată de un agent sau de server.
Step/Task	O comandă individuală: script, task Docker, AzureCLI, test, publish artifact.
Agent	Mașina care execută job-ul. Poate fi Microsoft-hosted sau self-hosted.
Artifact	Produsul rezultat din build: zip, fișier, pachet, imagine de container, chart.
Environment	Obiect logic ce reprezintă un mediu țintă și poate avea approvals și checks.
Service Connection	Identitatea prin care pipeline-ul se conectează la Azure, GitHub, ACR, Kubernetes sau alt serviciu extern.
Variable group / Secure files / Secrets	Metode de a injecta configurări și secrete fără a le hardcode în repo.

CI versus CD — nu le confunda

CI — Continuous Integration

Fiecare schimbare este integrată frecvent, compilată și verificată automat. Accentul este pe:

- Build
- Unit tests
- Code quality
- Artifact

Întrebarea CI: «Este codul bun și livrabil?»

CD — Continuous Delivery sau Deployment

Depinde de cultura organizației:

- **Continuous Delivery:** Pipeline-ul duce pachetul până la poarta producției și **așteaptă o aprobare** manuală.
- **Continuous Deployment:** Sistemul poate merge până la **livrare automată în producție**, fără intervenție umană.

Întrebarea CD: «Poate fi pus în mediu în mod controlat și repetabil?»

Separă clar cele două concepte. CI dovedește că aplicația se poate construi și testa. CD dovedește că aplicația se poate livra repetabil, controlat și auditabil.

YAML vs Classic — ce alegi și de ce

Microsoft recomandă pornirea proiectelor noi cu **YAML**. Motivul: definiția pipeline-ului stă în repo, lângă cod, este versionată, se poate valida prin pull request și se poate standardiza prin template-uri. Classic rămâne util în organizații cu multe release-uri istorice sau echipe orientate spre UI.

Criteriu	YAML	Classic	Recomandare didactică
Unde trăiește definiția	Fișier azure-pipelines.yml, versionat cu aplicația	Interfața web Azure DevOps	Predă YAML ca standard
Versionare	Foarte bună	Separată de cod	Predă YAML ca standard
Template-uri	Excelent	Limitat	Arată template-uri devreme
Onboarding inițial	Mediu	Foarte ușor	Demonstrează și UI
Audit și review	Natural prin Git (PR și istoric)	Mai greu de urmărit	Leagă de PR-uri
Legacy support	Bun	Foarte bun	Nu ignora Classic
Când are sens	Platform engineering, standardizare, scale, GitOps	Legacy, demo-uri rapide, migrare treptată	—

Notă importantă: Un pipeline Classic de build poate fi exportat în YAML în anumite cazuri, dar release-urile clasice nu se exportă direct în YAML — contează când planifici migrarea.

Azure Repos vs GitHub — comparație și relație

Ambele pot funcționa foarte bine cu Azure Pipelines. Nu trebuie să gândești alegerea ca pe un război religios. În multe organizații, GitHub este sursa principală de cod, iar Azure DevOps este folosit pentru Boards, enterprise governance și pipelines.

Aspect	Azure Repos	GitHub + Azure DevOps
Integrare nativă	Foarte strânsă, fără service connection externă pentru repo	Necesită conectare și autorizare GitHub
Experiență enterprise	Puternică în ecosistem Azure DevOps complet	Foarte bună, mai ales când echipa folosește deja GitHub
Pull requests	Da	Da, foarte matur
Boards linkage	Nativ	Suportat prin integrare Azure Boards - GitHub
Când are sens	Platforme interne, totul într-un loc	Echipă modernă de dezvoltare, open source, colaborare externă

Build Pipelines — ce face de fapt CI

Un build pipeline profesionist **nu doar compilează — el validează**. Build-ul este poarta tehnică. Dacă build-ul este instabil, tot lanțul de release devine instabil.

01

Checkout & Toolchain

Checkout al codului și setarea versiunii de toolchain (ex: .NET SDK, Node, Python).

02

Restore Dependencies

Descărcarea pachetelor și dependențelor necesare pentru build.

03

Build / Package

Compilarea aplicației sau construirea pachetului.

04

Teste & Quality

Unit tests, integration tests rapide, linting, code quality și security scan.

05

Publish Artifact

Publicarea artifact-ului rezultat cu naming/versioning clar pentru urmărit ce s-a livrat.

Regulă sănătoasă: Nu livra manual ce nu poți produce repetabil în build. Build pipeline-ul este secția de control calitate din fabrică — verifici, etichetezi și abia apoi pui pe raft ca artifact.

Release & Deployment Pipelines — partea de CD

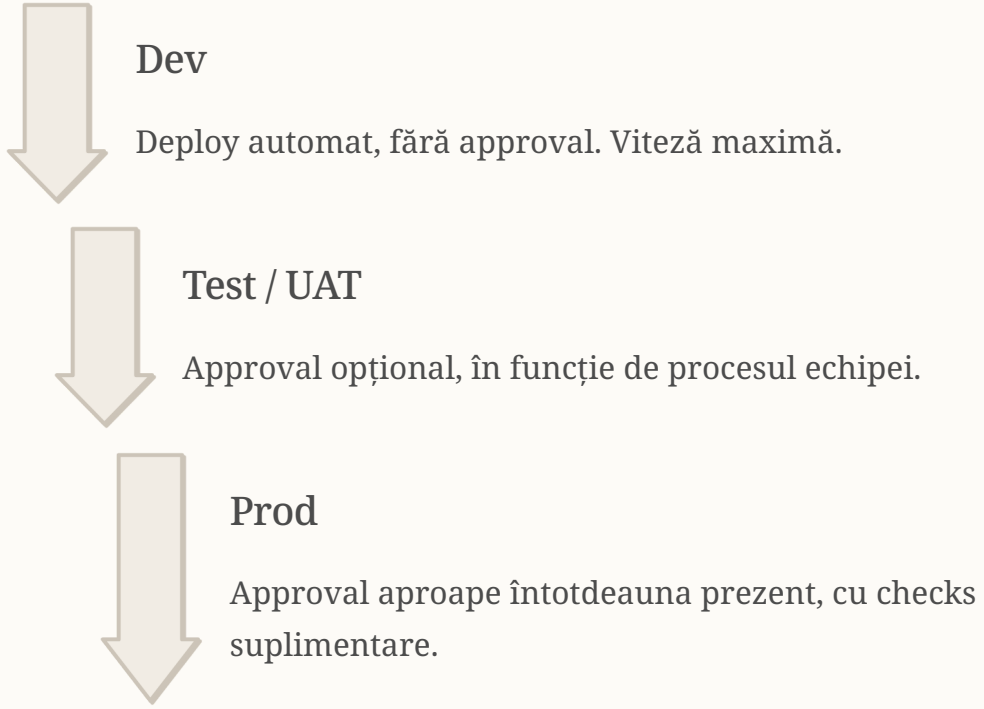
Ce înseamnă deploy-ul în realitate

Artifact-ul validat este promovat din mediu în mediu. În YAML faci asta cu **multi-stage pipelines și deployment jobs**. În modelul Classic faci asta cu **release pipelines și stage-uri vizuale**.

Deploy-ul nu înseamnă doar copiere de fișiere. Înseamnă și:

- Setări pe environment
- Injecție de secrete
- Ordine între componente
- Checks și approvals
- Health validation
- Opțiune de rollback
- Evidență clară a cine a aprobat și ce s-a livrat

Fluxul de promovare



Service Connections — identitatea pipeline-ului

Ce este un Service Connection

Service connection-ul este **podul de identitate** dintre Azure DevOps și un serviciu extern: Azure subscription, GitHub, ACR, Kubernetes, Docker registry și multe altele.

Tipic, pentru un deploy în Azure vei crea o conexiune **Azure Resource Manager**. Pentru imagini de container poți avea și **Docker Registry service connection**. Pentru repo-uri GitHub private poți avea o conexiune GitHub sau integrare de repo.

Bune practici reale

- Nume clare: `sc-az-prod-rg` sau `sc-acr-shared`
- Scop cât mai restrâns: Resource Group înainte de Subscription
- Nu oferi acces automat tuturor pipeline-urilor fără motiv clar
- Separă conexiunile pe medii când organizația are nevoie de control
- Revizuiește periodic cine folosește conexiunea și dacă permisiunile mai sunt necesare

Important pentru securitate: Principiul **least privilege** trebuie predat devreme. O conexiune prea largă poate transforma o greșeală YAML într-o problemă de securitate sau de cost.

Environments, Approvals și Checks

Ce este un Environment

Environment-ul este obiectul logic care îți permite să spui: *«Aici deployez și aici aplic reguli de control»*. În jurul environment-ului poți pune **approvals și checks**, de exemplu aprobare manuală înainte de producție.

Acest model este foarte valoros pentru **separarea responsabilităților**. Dezvoltatorul scrie YAML-ul, dar aprobarea de producție poate aparține owner-ului de produs, echipei de operațiuni sau change management.

Cum configurezi în portal

Pipelines → Environments → selectezi environment-ul → Approvals and checks → Add check. De aici poți seta approval manual, reguli sau alte controale.

🏢 Analogie practică

> Environment-ul este poarta unei clădiri. Pipeline-ul poate ajunge la poartă automat, dar accesul final poate necesita paznic, card de acces și validare.

Model de guvernare sănătos

Dev

Fără approval, viteză mare. Deploy automat la fiecare commit.

Test / UAT

Approval opțional, în funcție de procesul echipei și maturitatea testelor.

Prod

Approval aproape întotdeauna prezent, eventual și check-uri suplimentare de business.

Strategii de Deployment

În Azure Pipelines, deployment jobs YAML au suport explicit pentru **runOnce**, **rolling** și **canary**. Blue-green este mai degrabă un pattern de arhitectură și routing decât un keyword nativ de strategie.

Strategie	Cum funcționează	Avantaj	Risc/Limită	Unde are sens
runOnce	Livrezi o singură dată în environment.	Simplu și clar.	Nu oferă rollout gradual.	Dev, Test, internal apps.
rolling	Livrezi pe loturi de instanțe, nu toate simultan.	Mai puțin risc decât all-at-once.	Mai complex de modelat.	VM-uri, grupuri de noduri.
canary	Livrezi la subset mic, observi, apoi extinzi.	Validezi pe trafic real.	Necesită observabilitate bună.	AKS, trafic mare, prod.
blue-green	Două medii aproape identice; muți traficul între ele.	Rollback foarte rapid.	Cost mai mare și control de trafic necesar.	App Service slots, routing controlat.

Rezumat pentru juniori: runOnce = totul odată | rolling = pe loturi | canary = întâi puțin și observi | blue-green = ai două lumi paralele și schimbi traficul între ele.

Blue-Green și Canary — explicații simple

🎭 Blue-Green

> Este ca atunci când ai două scene identice într-un teatru. Publicul se uită la scena albastră, iar tu pregătești scena verde în spate. Când totul este gata, schimbi reflectorul și publicul vede scena nouă fără pauză.

În Azure: Pentru web apps, **deployment slots** sunt foarte intuitive. Livrezi în staging slot, faci smoke tests, apoi faci swap cu production slot.

- Rollback = swap înapoi, instant
- Zero downtime garantat
- Excelent pentru App Service

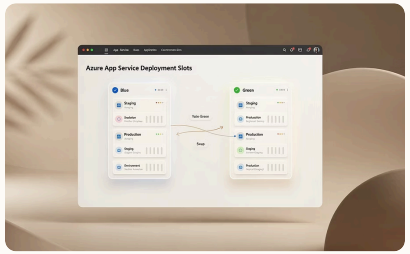
🐦 Canary

> Este ca atunci când lansezi un produs nou întâi într-un singur magazin sau într-un singur cartier. Observi reacțiile, măsori și abia apoi scalezi la tot orașul.

În Azure: Pe platforme containerizate, **AKS și controlul traficului** sunt terenul natural pentru canary.

- Expunere graduală la trafic real
- Necesită observabilitate bună
- Ideal pentru AKS și rollouts cu risc redus

Exemple reale de strategii pe servicii Azure



App Service — Blue-Green

Scenariul clasic: **production slot** + **staging slot**. Livrezi în staging, faci smoke tests, apoi swap. Rollback = swap înapoi, instant.

AKS — Canary

Canary are sens când poți controla routing-ul și observabilitatea. **Nu vrei să expui imediat 100% din trafic** unei versiuni noi. Controlezi procentul de trafic și observi metricile.

VM-uri tradiționale — Rolling

Rolling deployment **reduce riscul pentru servere stateful** sau ferme mari de aplicații. Livrezi pe loturi, nu atingi toate instanțele simultan.

Setup Zero to Hero — Pașii complet în portal

Construim un traseu didactic complet pentru livrarea unei aplicații web în Azure App Service. Presupunem că vrei să livrezi o aplicație web și, opțional, mai târziu, într-un cluster AKS.

1 Pasul 1 — Creează organizația și proiectul Azure DevOps

Intră în Azure DevOps, creează o Organization dacă nu există deja, apoi creează un Project nou. Alege un nume clar, de exemplu `devops-lab-webapp`. Public sau private — în training aproape mereu private.

2 Pasul 2 — Alege sursa codului

Decide dacă vei folosi Azure Repos sau GitHub. Pentru cursuri de bază, Azure Repos este mai simplu fiindcă totul stă în aceeași platformă. Dacă studenții folosesc deja GitHub, conectează repo-ul GitHub la pipeline.

3 Pasul 3 — Pregătește aplicația

Repo-ul trebuie să conțină codul aplicației și, ideal, fișierul `azure-pipelines.yml`. Dacă începi din UI, Azure DevOps poate genera un starter pipeline.

4 Pasul 4 — Creează resursele Azure țintă

În `portal.azure.com` creează Resource Group-ul și serviciul țintă: Azure App Service, Function App sau AKS. Pentru App Service, creează și deployment slot dacă vrei să demonstrezi blue-green.

5 Pasul 5 — Creează Service Connection către Azure

Project settings → Service connections → New service connection → Azure Resource Manager. Alege scopul potrivit (Subscription sau Resource Group) și salvează cu un nume clar, de exemplu `sc-az-demo-rg`.

Setup Zero to Hero — Pașii 6-10

1

Pasul 6 — Creează Environment-urile

Pipelines → Environments → creezi `dev`, `test`, `prod`. Pentru prod adaugă approval. Studenții văd diferența dintre deploy automat și deploy controlat.

2

Pasul 7 — Creează pipeline-ul

Pipelines → New pipeline. Alege repo-ul. Dacă folosești YAML, selectează Starter pipeline și înlocuiește conținutul cu unul relevant. Dacă demonstrezi Classic, creează un pipeline vizual de build și apoi un release pipeline.

3

Pasul 8 — Rulează primul build

Salvezi și rulezi pipeline-ul. Verifici checkout, restore, build, tests și publish artifact. **Nu treci la deploy până build-ul nu este curat.**

4

Pasul 9 — Adaugă stage-ul de deploy

În YAML adaugi un stage Deploy cu deployment job și environment. În Classic adaugi un Release stage cu artifact-ul rezultat și task-ul de deploy. Leagă service connection-ul creat mai devreme.

5

Pasul 10 — Validează și demonstrează trasabilitatea

Arată studenților unde se văd logurile, artifact-ul, approval-ul, istoricul environment-ului și diferența dintre un deploy reușit și unul eșuat.

Exemplu YAML complet — Build Stage

Observă elementele importante: trigger pe branch, pool Microsoft-hosted, variabile centralizate, artifact publicat în Build și deployment job cu environment dev. Aceasta este o structură sănătoasă de bază.

```yaml
trigger:
- main

pool:
 vmImage: ubuntu-latest

variables:
 buildConfiguration: Release
 azureServiceConnection: sc-az-demo-rg
 webAppName: my-demo-webapp
 artifactName: webapp-drop

stages:
- stage: Build
 jobs:
 - job: Build
 steps:
 - checkout: self
 - task: UseDotNet@2
 inputs:
 packageType: sdk
 version: 8.x
 - script: dotnet restore
 displayName: Restore
 - script: dotnet build --configuration $(buildConfiguration) --no-restore
 displayName: Build
 - script: dotnet test --configuration $(buildConfiguration) --no-build
 displayName: Test
 - script: dotnet publish -c $(buildConfiguration) -o $(Build.ArtifactStagingDirectory)/publish
 displayName: Publish app
 - publish: $(Build.ArtifactStagingDirectory)/publish
 artifact: $(artifactName)
```

Exemplu YAML complet — Deploy Stage

```yaml
- stage: Deploy_Dev
  dependsOn: Build
  jobs:
  - deployment: DeployWeb
    environment: dev
    strategy:
      runOnce:
        deploy:
          steps:
          - download: current
            artifact: $(artifactName)
          - task: AzureWebApp@1
            inputs:
              azureSubscription: $(azureServiceConnection)
              appType: webApp
              appName: $(webAppName)
              package: $(Pipeline.Workspace)/$(artifactName)
```

trigger

Definește când pornește pipeline-ul (la push pe main).

pool

Alege agentul Microsoft-hosted care execută job-ul.

variables

Evită hardcodarea repetitivă și fac pipeline-ul mai curat.

stages

Separă clar build-ul de deploy.

publish

Creează artifact-ul consumat mai târziu de deploy.

deployment job

Spune explicit că este o livrare către un environment, nu un job generic.

Cum explici YAML-ul studenților — linie cu linie

→ `trigger`

Definește când pornește pipeline-ul. De exemplu, la orice push pe branch-ul `main`. Poți configura și pentru PR-uri sau alte branch-uri.

→ `pool`

Alege agentul care execută job-ul. `ubuntu-latest` este un agent Microsoft-hosted. Poți folosi și agenți self-hosted pentru nevoi speciale.

→ `variables`

Evită hardcodarea repetitivă și fac pipeline-ul mai curat și mai ușor de întreținut. Valorile pot fi suprascrise la rulare.

→ `stages`

Separă clar build-ul de deploy. Fiecare stage poate depinde de altul (`dependsOn`) și poate rula condiționat.

→ `publish`

Creează artifact-ul consumat mai târziu de deploy. Artifact-ul este puntea dintre build și deploy — fără el, nu există trasabilitate.

→ `deployment job` + `environment`

Spune explicit că nu este un job generic, ci o livrare către un environment. Permite approvals, checks și istoric de deployment.

Cum faci același lucru în Classic

Classic are două lumi istorice: **build pipeline** și **release pipeline**. Merită să le arăți, pentru că multe organizații încă le folosesc. Avantajul didactic: studenții văd foarte clar conceptul de artifact și traseul vizual dintre stage-uri.

Build Pipeline Classic

1. Pipelines → Builds → New pipeline și alegi repo-ul
2. Adaugi task-uri vizuale de restore, build, test și publish artifact
3. Salvezi build-ul și îl rulezi

Release Pipeline Classic

1. Pipelines → Releases → New release pipeline
2. Adaugi artifact-ul provenit din build
3. Adaugi stage-uri: Dev și Prod
4. În fiecare stage configurezi task-urile de deployment (ex: Azure App Service deploy)
5. Activezi approvals înainte de Prod dacă vrei control manual

> **Dezavantaj important:** Definiția release-ului Classic nu stă elegant în Git împreună cu aplicația și nu se poate exporta direct în YAML — contează când planifici migrarea spre YAML.

Cum conectezi GitHub la Azure DevOps

01

Verifică accesul

Asigură-te că ai acces la repo-ul GitHub și permisiuni de autorizare pentru a conecta aplicații terțe.

02

Alege GitHub ca sursă

În Azure DevOps, când creezi pipeline-ul, alege GitHub ca sursă în loc de Azure Repos.

03

Acordă autorizațiile

Acordă autorizațiile cerute și selectează repo-ul dorit. Azure DevOps va instala un GitHub App sau va folosi OAuth.

04

Service Connection (dacă e necesar)

Dacă ai nevoie de resurse suplimentare, creează service connection GitHub acolo unde este cazul.

05

Verifică webhook-urile și trigger-ele

Pipeline-ul trebuie să pornească la push sau PR conform designului tău. Verifică că trigger-ele sunt configurate corect.

Greșeală frecventă la juniori: Presupun că alegerea GitHub schimbă logica CI/CD. Nu o schimbă. Sursa codului se schimbă, dar disciplina pipeline-ului rămâne aceeași.

Ce greșesc cel mai des juniorii

🔐 Secrete în YAML

Pun secrete direct în YAML sau în repo. Folosiți Variable Groups, Key Vault sau Secure Files.

📦 Fără artifact clar

Amestecă build-ul și deploy-ul fără artifact clar. Artifact-ul este puntea — fără el nu există trasabilitate.

🌍 Fără environments

Nu separă mediile și nu folosesc environments. Toate deploy-urile merg direct, fără control sau aprobare.

🔑 Service connections prea largi

Leagă service connection-uri cu acces prea mare (Subscription în loc de Resource Group).

📛 Naming inconsistent

Nu păstrează naming consecvent pentru pipeline-uri, artifacts și environments. Greu de urmărit și de auditat.

❌ Build vs Deploy confundate

Nu înțeleg diferența dintre un eșec de build și un eșec de deployment — diagnosticul devine haotic.

🔄 Fără strategie de rollback

Nu au o strategie clară de rollback. Rollback-ul trebuie gândit înainte de primul incident, nu după.

Model de laborator practic pentru curs

Lab	Obiectiv	Servicii	Rezultat
1	Repo + pipeline YAML minimal	Azure Repos sau GitHub, Azure Pipelines	Build reușit și artifact publicat
2	Deploy automat în Dev	App Service, Environment dev, service connection	Prima livrare completă end-to-end
3	Approval înainte de Prod	Environment prod cu approval configurat	Flux controlat de aprobare demonstrat
4	Blue-green demo	App Service deployment slots	Swap și rollback demonstrat live
5	Canary conceptual / AKS demo	AKS, ACR, Kubernetes manifest	Înțelegerea rollout-ului gradual

Fiecare laborator construiește pe cel anterior, creând un traseu progresiv de la zero la un pipeline complet, cu control de business și strategii avansate de deployment.

Troubleshooting — cum gândește un inginer bun

Când pipeline-ul eșuează, cea mai mare parte a timpului pierdut nu vine dintr-o eroare sofisticată, ci din **lipsa unei metodologii simple de diagnostic**.

1 Identifică etapa exactă

Source, restore, build, test, artifact, deploy, approval sau post-deploy validation. Nu sări direct la soluție fără să știi unde s-a oprit.

2 Citește logul complet

Nu doar ultima eroare. Adesea cauza reală este câteva linii mai sus, mascată de erori secundare.

3 Clasifică problema

Este de cod, de agent, de permisiuni sau de target environment? Fiecare categorie are un set diferit de soluții.

4 Verifică service connection și scope

Verifică dacă service connection-ul există, are permisiunile corecte și este autorizat pentru pipeline-ul respectiv.

5 Verifică artifact-ul

Artifact-ul chiar există și este consumat din calea corectă? Verifică branch, trigger și condiții de rulare.

6 Verifică targetul din Azure

Dacă e vorba de deploy, verifică și targetul: App Service, AKS, slot, secrets, config. Problema poate fi în infrastructură, nu în pipeline.

Bune practici de design pe termen lung

Principii de arhitectură pipeline

- Folosește YAML pentru proiectele noi
- Separă build de deploy și păstrează artifact-ul ca obiect clar
- Standardizează template-uri între echipe
- Folosește environments și approvals pentru control de business
- Păstrează service connections restrânse și bine numite

Principii de operare și siguranță

- Gândește rollback-ul înainte de primul incident, nu după
- Leagă pipeline-ul de observabilitate și health checks
- Documentează naming, branching și regula de promovare între medii
- Nu hardcoda secrete — folosește Variable Groups sau Key Vault
- Revizuiește periodic permisiunile service connections

Cu cât pipeline-ul este mai predictibil, cu atât scade riscul de schimbare. În producție, disciplina pipeline-ului contează la fel de mult ca aplicația.

Concluzie

Azure DevOps și CI/CD nu sunt doar un capitol tehnic. Sunt **mecanismul prin care organizația transformă codul în livrare predictibilă**. Pentru un junior cloud engineer, stăpânirea acestui subiect schimbă complet profilul profesional: din om care «știe Azure» în om care poate livra în Azure în mod responsabil.

> Infrastructura și aplicația nu sunt complete fără un traseu de livrare repetabil. **Cloud-ul fără CI/CD este doar jumătate din poveste.**

Cheat Sheet Final — Referință rapidă

Proiect nou?

Începe cu **YAML**. Microsoft recomandă YAML pentru toate proiectele noi.

Ai medii și control?

Folosește **Environments** cu approvals și checks pentru fiecare mediu critic.

Identitate spre Azure?

Creează **Service Connection** cu scope restrâns și nume clar.

Artifact = puntea

Artifact-ul este puntea dintre build și deploy. Fără el, nu există trasabilitate.

Schimbare rapidă de trafic?

Blue-green pentru schimbare rapidă. **Canary** pentru expunere graduală.

Classic încă există

Classic rămâne valid, dar direcția modernă este **YAML + template-uri**.