INTERCEPT

WHITEPAPER

Infrastructure as Code met Azure:

Jouw praktische gids voor automatisering en CI/CD

Voorwoord

Welkom!

Cloud computing blijft zich ontwikkelen en daarmee ook de manieren om Azure-resources te deployen en beheren. Voor elke behoefte en voorkeur zijn er inmiddels passende opties beschikbaar. Of je nu net begint met Azure of als ervaren gebruiker je workflows wilt optimaliseren: het loont om de verschillende deploymentmethoden goed te begrijpen.

Het Azure Portal is voor veel gebruikers het startpunt. Dankzij de intuïtieve, grafische interface kun je snel resources verkennen en beheren, ideaal voor beginners en bij het oplossen van problemen.

Maar naarmate je omgeving groeit, kent deze zogeheten ‘Click-Ops’-aanpak ook nadelen, zoals inconsistente configuraties en het ontbreken van een centrale bron van documentatie. In zulke gevallen biedt Infrastructure as Code (IaC) een schaalbaarder en betrouwbaarder alternatief.

In dit e-book verkennen we verschillende deploymenttechnologieën, met hun unieke eigenschappen, voordelen en toepassingen. Zo kies je gericht de tools die het beste passen bij jouw situatie – en haal je het maximale uit je Azure-omgeving.

Aan de slag!

Table of Content

ClickOps vs. Infrastructure as Code (IaC):

Wanneer is ClickOps nuttig en wanneer niet?

In dit hoofdstuk verkennen we ClickOps en Infrastructure as Code (IaC): Wanneer is handmatig klikken in het Azure Portal zinvol en wanneer kan je beter kiezen voor een geautomatiseerde aanpak? Je leert de beperkingen van ClickOps kennen, ontdekt de voordelen van IaC en weet na afloop wanneer een grafische interface nog wél zinvol is.

De waarde van ClickOps

Waarom gebruiken we het Azure Portal eigenlijk? Misschien ken je de uitspraak: *“Friends don’t let friends click in the Azure portal.”* Maar waarom eigenlijk niet? Is het omdat mensen grafische interfaces niet fijn vinden? Of geven ze simpelweg de voorkeur aan command-line tools?

De afkeer van grafische interfaces blijkt niet de werkelijke reden. In werkelijkheid is het Azure Portal vooral waardevol voor wie net begint met Azure: het helpt je de mogelijkheden van het platform te verkennen en vertrouwd te raken met de interface. Ook voor snelle checks of troubleshooting blijft het een handig hulpmiddel. De visuele feedback in real-time spreekt bovendien veel gebruikers aan.

De keerzijde van ClickOps

Toch kent ClickOps ook duidelijke nadelen, vooral zichtbaar in de risico’s die komen kijken bij het handmatig beheren van je infrastructuur via het Azure Portal. Onderstaande voorbeelden illustreren dat als geen ander:

→ Voorbeeld 1:

Stel, je dient 20 virtuele machines (VM’s) uit te rollen via het Azure Portal. Allemaal met exact dezelfde configuratie, om consistentie binnen je omgeving te behouden. Dit handmatig doen vergroot de kans op fouten: één gemiste instelling en je krijgt inconsistente resultaten. Dat maakt beheer en fout-opsporing lastig.

→ Voorbeeld 2:

Een ander belangrijk nadeel van ClickOps is het ontbreken van een centrale bron van waarheid. Handmatig resources aanmaken in het Azure Portal laat geen script of eenduidige documentatie achter van de configuratie. Daardoor is het lastig om omgevingen te reproduceren of de actuele staat van je infrastructuur te achterhalen. Dit probleem wordt groter zodra meerdere beheerders betrokken zijn, die ieder op hun eigen manier werken, met meer afwijkingen en inconsistentie als gevolg.

→ Voorbeeld 3:

Ook documentatie vormt een uitdaging bij ClickOps. Elke handmatige stap moet je zelf vastleggen, een tijdrovend en foutgevoelig proces. Zonder nauwkeurige documentatie wordt het moeilijk om wijzigingen te auditen, configuraties te volgen of nieuwe teamleden goed in te werken.

Friends don’t let friends click in the Azure portal.

Het alternatief: Infrastructure as Code (IaC)

De voorgaande voorbeelden laten zien dat ClickOps niet altijd volstaat. Maar wat is een geschikt alternatief? Infrastructure as Code (IaC) biedt een betrouwbaardere en efficiëntere manier om infrastructuur te beheren. Met tools zoals Azure Bicep tackle je de uitdagingen van ClickOps. IaC zorgt voor consistente configuraties, biedt één centrale bron van waarheid en maakt documentatie eenvoudiger en betrouwbaarder. Bovendien vereenvoudigt het het schalen en beheren van je infrastructuur.

Wat is Infrastructure as Code?

IaC is een methode waarbij je infrastructuur definieert via machineleesbare configuratie-bestanden, in plaats van via handmatige instellingen of interactieve tools. Hiermee automatiseer je provisioning, beheer en monitoring, waardoor handmatige acties overbodig worden. Nu je weet wat IaC inhoudt, zullen we eens kijken naar de belangrijkste voordelen ervan.

Voordeel 1: Schaalbaarheid

IaC maakt het mogelijk om infrastructuur snel en efficiënt op te schalen. Omdat alles in code staat, kun je meerdere omgevingen consistent uitrollen met een minimale kans op fouten. Dat is vooral waardevol bij grote applicaties of omgevingen die snel moeten opschalen. Ook het klonen van development-, test- en productieomgevingen verloopt eenvoudig. Het voorkomt het bekende probleem: *“het werkt op mijn machine, maar niet elders”*.

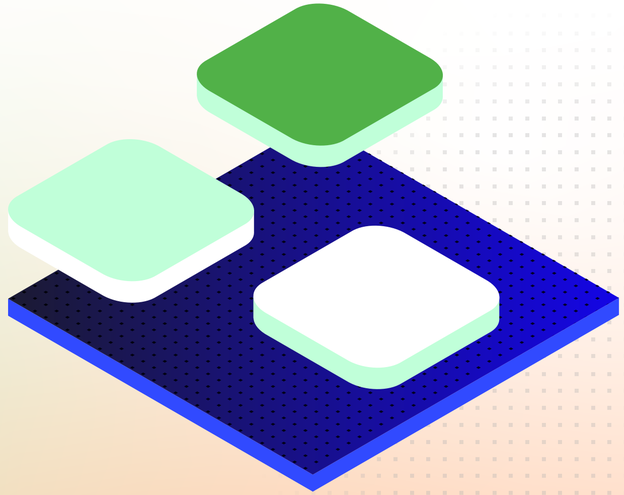
Voordeel 2: Consistentie

Een ander belangrijk voordeel van IaC is haar consistentie. Door infrastructuur vast te leggen in code, verloopt elke deployment op identieke wijze. Hierdoor voorkom je configuratieverschillen en verhoog je de betrouwbaarheid. Ook wijzigingen voer je gestructureerd en gecontroleerd door: aanpassingen in de code worden automatisch doorgevoerd in alle omgevingen, wat synchronisatie garandeert.

Wanneer werkt ClickOps wel en wanneer niet?

In dit hoofdstuk vergeleken we ClickOps met IaC. We zagen dat IaC zorgt voor schaalbaarheid, consistentie en betrouwbaarheid, doordat het infrastructuurbeheer automatiseert met code. Je rolt omgevingen uniform uit en voorkomt fouten.
Maar ClickOps is daarmee niet overbodig. Voor wie net begint met Azure, of zich bezighoudt met foutopsporing en snelle analyses, blijft de grafische interface waardevol. Alleen mist ClickOps de schaalbaarheid en herhaalbaarheid die IaC biedt.

De conclusie? ClickOps en IaC vullen elkaar uitstekend aan. Waar ClickOps een intuïtieve manier biedt om infrastructuur te verkennen, zorgt IaC voor efficiëntie, controle en beheersbaarheid op schaal. Benieuwd welke IaC-taal het beste past bij jouw situatie? Lees dan verder in het volgende hoofdstuk.



HOOFDSTUK 2

Aan de slag met de juiste Infrastructure as Code-taal

In dit hoofdstuk helpen we je de juiste Infrastructure as Code-taal te kiezen. We laten zien hoe je snel aan de slag gaat en delen best practices zoals versiebeheer en automatisering. Na afloop weet je precies welke IaC taal het beste past bij jouw omgeving, voorkeuren en projectdoelen. Let's go!

Hoe kies je de juiste IaC-taal?

Om deze vraag te beantwoorden, doorloop je een aantal stappen:

Stap 1: Check welke talen je omgeving ondersteunt

Voordat je een IaC-taal kiest, is het belangrijk om te achterhalen welke tools je omgeving ondersteunt. Je kunt dit als volgt aanpakken:

→ **Check de officiële documentatie:**
Cloudproviders zoals Azure publiceren regelmatig een overzicht van ondersteunde IaC-tools. Denk aan Bicep, Terraform, Pulumi en ARM-templates (Azure Resource Manager).

→ **Beoordeel beleids- en compliance-eisen:**
Sommige organisaties stellen eisen aan welke tools gebruikt mogen worden, bijvoorbeeld vanwege beveiliging of governance. Houd daar rekening mee.

→ **Analyseer je bestaande infrastructuur:**
Gebruikt jouw team al een specifieke tool zoals Terraform of ARM-templates? Dan is het vaak slim om hierbij te blijven voor consistentie en efficiëntie.

→ **Test verschillende opties:**
Probeer een eenvoudige deployment uit met meerdere IaC-talen in een testomgeving. Zo ontdek je welke taal het beste aansluit bij jouw workflow en voorkeuren.

Stap 2: Kies de juiste taal

De keuze voor een Infrastructure as Code-taal, zoals Bicep, Terraform of Pulumi, is bepalend voor hoe je werkt én hoe schaalbaar je oplossing zal zijn. Deze stap vormt dus de basis voor een succesvolle start met IaC. Houd daarbij rekening met de volgende punten:

→ **Onderzoek welke talen je omgeving ondersteunt.**
Zodra je weet welke opties binnen jouw infrastructuur mogelijk zijn, kun je gerichter kiezen en voorkom je technische obstakels verderop.

→ **Maak een bewuste keuze op basis van cloudprovider, organisatiebehoeften en persoonlijke voorkeuren.**

3. **Verdiep je in hoe je kunt starten met elke taal.**
Een soepele overstap naar Infrastructure as Code begint bij een taal die niet alleen bij je technische omgeving past, maar ook bij jouw manier van werken.

Zodra je deze aspecten in kaart hebt gebracht, kun je gericht de verschillende talen verder verkennen en bepalen welke het beste aansluit bij jouw project, team en toekomstvisie.

Populaire IaC-talen:

In deze sectie bespreken we de meest gebruikte IaC-talen: Bicep, Terraform, Pulumi en de declaratieve ARM-templates. Elk van deze talen heeft zijn eigen voordelen, afhankelijk van jouw context.

→ **Taal: Bicep**

Werk je met Azure, dan is Bicep een uitstekende keuze. Deze domeinspecifieke taal (DSL) is speciaal ontwikkeld om het werken met ARM-templates eenvoudiger te maken. Bicep is declaratief: je beschrijft de gewenste situatie, en Azure zorgt voor de uitvoering. Het is een stuk eenvoudiger te lezen en te schrijven dan de vaak complexe, JSON-gebaseerde ARM-templates. De taal sluit naadloos aan op Azure en integreert soepel met bestaande workflows binnen het Microsoft-ecosysteem. Hieronder vind je een voorbeeld van hoe je met Bicep eenvoudig een storage account uitrolt.

```
1. param location string = ‘East US’
2. param storageAccountName string = ‘mystorageaccount’
3.
4. resource storageAccount ‘Microsoft.Storage/storageAccounts@2022-09-01’ =
   {
   name: storageAccountName
5. location: location kind: ‘StorageV2’
6. sku: {      name: ‘Standard_LRS’      }    }
```

→ Taal: Terraform

Terraform is cloud-onafhankelijk. Dit houdt in dat je er infrastructuur mee kunt definiëren over meerdere cloudproviders, dus niet enkel binnen Azure.

Net als Bicep werkt Terraform declaratief, maar het is breder inzetbaar. Terraform maakt gebruik van de HashiCorp Configuration Language (HCL), een eenvoudige, toegankelijke taal om te leren. Werk je al in een multi-cloudomgeving of bereid je je daarop voor? Dan is Terraform een slimme keuze. Let er wel op dat bepaalde geavanceerde functionaliteiten alleen beschikbaar zijn via een licentie. Wil je liever volledig open-source werken? Dan is OpenTofu een goed alternatief. Een open-sourcevariant beheerd door de Linux Foundation. OpenToFu biedt vergelijkbare mogelijkheden en is vrij beschikbaar.

```
provider “azurerm” {
    features {}
  }

 resource “azurerm_storage_account” “storage” {
 name                     = “mystorageaccount”
 resource_group_name      = “myResourceGroup”
   location                 = “East US”
   account_tier             = “Standard”
   account_replication_type = “LRS”
 }
```

→ Taal: Pulumi

In tegenstelling tot Bicep en Terraform stelt Pulumi je in staat om infrastructuur te definiëren met behulp van algemene programmeertalen zoals Python, TypeScript en C#Dat maakt Pulumi vooral interessant voor developers die graag werken in een imperatieve programmeerstijl. Pulumi biedt veel vrijheid en mogelijkheden, maar vereist daardoor ook wat meer programmeerkennis. In het voorbeeld hieronder tonen we hoe je met Pulumi (in TypeScript) een storage account kunt uitrollen.

```
import * as azure from “@pulumi/azure”;

  const storageAccount = new azure.storage.Account(“storage”, {
   name: “mystorageaccount”,
   resourceGroupName: “myResourceGroup”,
   location: “East US”,
    accountTier: “Standard”,
    accountReplicationType: “LRS”,
 });
```

Declaratieve taal: ARM-templates

ARM-templates zijn op JSON gebaseerde bestanden waarmee je Azure-resources op een declaratieve manier kunt definiëren. Ze bieden een diepe integratie met Azure, maar staan erom bekend lastig te lezen, schrijven en onderhouden te zijn. Zeker bij complexe deployments. Om die reden introduceerde Microsoft Bicep als gebruiksvriendelijkere abstractielaag bovenop ARM-templates. Het In het voorbeeld hieronder zie je hoe je met een traditionele ARM-template een storage account kunt uitrollen.

```
{
  “$schema”: “https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#”,
  “contentVersion”: “1.0.0.0”,
  “resources”: [
    {
      “type”: “Microsoft.Storage/storageAccounts”,
      “apiVersion”: “2022-09-01”,
      “name”: “mystorageaccount”,
      “location”: “East US”,
      “sku”: {
        “name”: “Standard_LRS”
      },
      “kind”: “StorageV2”
    }
  ]
}
```

Elk van deze tools heeft zijn eigen sterke punten. Werk je uitsluitend in Azure? Dan is Bicep meestal de beste keuze. Voor meer flexibiliteit over meerdere cloudplatforms is Terraform zeer geschikt. En geef je de voorkeur aan traditionele programmeertalen? Dan is Pulumi mogelijk precies wat je zoekt.

Stap 3: Begin met het schrijven van Infrastructure as Code

De eerste stappen zetten met Infrastructure as Code (IaC) kan overweldigend lijken, maar door het proces op te splitsen wordt het al snel behapbaar. Hier zijn een paar tips om je op weg te helpen:

Tip 1: Richt je development omgeving in:t:

- → Installeer de benodigde CLI en tools voor de IaC-taal van jouw keuze.
- → Zorg dat je toegang hebt tot een Azure-abonnement.
- → Gebruik een developmentomgeving zoals **Visual Studio Code** dat uitgebreide ondersteuning biedt voor IaC. Installeer bijvoorbeeld de de Bicep Extensie voor VS Code , de Terraform Extension van HashiCorp of de Pulumi Extension voor syntaxis-hightlighting, autocompletion en een eenvoudige deployment.
- → Gebruik de PowerShell extensie in VS Code om je scripts uit te voeren en infrastructuur direct te deployen.

2. Tip 2: Leer via officiele bronnen:

- → Microsoft biedt uitstekend lesmateriaal voor Bicep. Zo zijn er bijvoorbeeld stap-voor-stap leermodules te volgen via Microsoft Learn.
- → Ook Terraform en Pulumi bieden uitgebreide documentatie en hands-on labs via hun eigen websites.

Tip 3: Begin klein

- → Start met een een eenvoudige resource, zoals het uitrollen van een storage account (zie eerdere voorbeelden).
- → Test je scripts in een veilige sandboxomgeving, vóór je ze in productie brengt.
- → Volg best practices zoals het gebruik van parameters en een modulaire structuur. Bekijk ook de Azure Verified Modules die ondersteuning bieden vooor zowel Bicep als Terraform.

Tip 4: Maak gebruik van version control en automatisering

- → Sla je IaC-code op in een Git-repository.
- → Automatiseer je deployment met CI/CD-pipelines via bijvoorbeeld Azure DevOps of GitHub Actions.
- → Valideer en test je infrastructuurcode grondig voor je wijzigingen toepast.

Terugkomend op de hoofdvraag: Hoe kies je nu de juiste taal?

We hebben in dit hoofdstuk gezien dat overstappen van handmatige processen naar code spannend kan zijn, maar ook een logische stap richting toekomstbestendige infrastructuur. Ook zagen we dat welke taal je kiest afhangt van je werkomgeving en je doelstellingen:

- Werk je in **Azure?** Dan is Bicep vaak de beste keuze.
- Heb je meer **flexibiliteit** nodig, dan passen Terraform of Pulumi mogelijk beter.
- Werk je al met **ARM-templates**? Dan helpt Bicep om je processen te vereenvoudigen.

Maar het allerbelangrijkste: beoordeel sowieso eerst welke talen door jouw omgeving worden ondersteund. Zo weet je zeker dat je keuze aansluit bij het beleid van je organisatie, de werkwijze van je team en de mogelijkheden van je cloudprovider.

We benadrukten ook het belang van klein beginnen, experimenteren en stapsgewijs uitbreiden. Naarmate je meer ervaring opdoet zal het werken met IaC net zo vanzelfsprekend worden als het in elkaar zetten van een server vroeger. De tijd van de schroevendraaier ligt misschien achter ons, maar precisie en structuur blijven onmisbaar.

Nu je weet hoe je de juiste IaC-taal kiest én hoe je ermee begint, ben je klaar voor hoofdstuk 3, waarin we dieper ingaan op modulariteit. De logische volgende stap richting een schaalbare, herbruikbare en goed beheersbare infrastructuur. Let’s go!

Naarmate je meer ervaring opdoet zal het werken met IaC net zo vanzelfsprekend worden als het in elkaar zetten van een server vroeger.

HOOFDSTUK 3

Modulariteit en herbruikbaarheid in Azure Bicep

In dit hoofdstuk laten we zien hoe je met herbruikbare Bicep-modules een gestandaardiseerde en schaalbare infrastructuur opbouwt. Modulariteit helpt om fouten te voorkomen, maakt samenwerking tussen teams makkelijker en houdt je omgeving overzichtelijk en flexibel.

Let’s dive in!

Hoe maak en beheer je herbruikbare Azure Bicep-modules voor een schaalbare en gestandaardiseerde infrastructuur?

Hoe dragen modules bij aan standaardisatie en het vereenvoudigen van complexe configuraties? Van de vele tools voor Infrastructure as Code (IaC) springt Azure Bicep eruit door zijn eenvoud en focus op modulariteit. Bicep is een domeinspecifieke taal (DSL) waarmee je Azure-resources declaratief kunt deployen. De syntaxis is compacter en gebruiksvriendelijker dan die van traditionele JSON-gebaseerde ARM-templates.

Een belangrijk kenmerk van Bicep is de ondersteuning voor modules: herbruikbare bouwstenen die je infrastructuur consistenter, schaalbaarder en beter beheersbaar maken.

Azure Bicep en het belang van modulariteit

Bicep maakt het definiëren en uitrollen van Azure-resources intuïtiever en eenvoudiger dan met de complexe JSON-templates van ARM. Dit is zowel voor beginners als gevorderde gebruikers een voordeel. De echte kracht zit echter in de mogelijkheid om modulair te werken.

Modulariteit

Modules in Bicep bundelen specifieke resources en configuraties tot herbruikbare componenten. Daarmee kun je gestandaardiseerde bouwstenen ontwikkelen die eenvoudig her te gebruiken zijn en te delen zijn binnen verschillende projecten en teams. In een module leg je de benodigde logica vast en stel je alleen relevante parameters beschikbaar. Dat voorkomt duplicatie, verhoogt de consistentie en vereenvoudigt het beheer van complexe omgevingen.

Het bouwen van gestandaardiseerde bouwblokken

Met modules in Azure Bicep bouw je herbruikbare bouwstenen die je organisatiebreed kunt inzetten. Dat kan gaan om eenvoudige resources, een storage account bijvoorbeeld, maar ook meer complexe resources, zoals gelaagde applicaties met meerdere componenten. Alles is als module vast te leggen en het helpt consistentie te behouden.

Stel dat je bijvoorbeeld een module maakt voor het uitrollen van een virtueel netwerk (VNet). Deze module bevat dan automatisch de configuratie voor subnets, network security groups (NSG's) en route tables. Doordat dit alles in één module zit, kun je het moeiteloos inzetten in andere projecten, zonder telkens opnieuw alles te moeten definiëren.

```
module vnet 'vnet.bicep' = {
name: 'myVnetModule'
params: {
vnetName: 'myVnet'
addressPrefix: '10.0.0.0/16'
subnets: [
 {
   name: 'subnet1'
   addressPrefix: '10.0.1.0/24'
 },
 {
   name: 'subnet2'
   addressPrefix: '10.0.2.0/24'
 }
    ]
  }
}
```

In bovenstaand voorbeeld bevat de vnet.bicep module de logica om een virtueel netwerk (VNet) met twee subnets uit te rollen. Door deze module te gebruiken, kun je het VNet heel simpel consistent uitrollen in meerdere omgevingen.

Complexe configuraties vereenvoudigen

Een van de belangrijkste voordelen van modules in Azure Bicep is dat je complexe configuraties kunt vereenvoudigen. Infrastructure as Code (IaC) kan namelijk behoorlijk ingewikkeld worden, vooral binnen grootschalige omgevingen waarin veel resources onderling van elkaar afhankelijk zijn. Met modules verberg je die complexiteit als het ware achter een eenvoudige interface. Hierdoor wordt het veel makkelijker om infrastructuur te implementeren en te beheren.

Stel je wilt een high availability application deployen, bestaand uit meerdere onderdelen zoals VMs, load balancers en databases. Zonder modules kan het definieren en beheren van al die resources behoorlijk overweldigend zijn. Maar door elk onderdeel simpelweg in een module onder te brengen kun je het deploymentproces veel makkelijker maken.

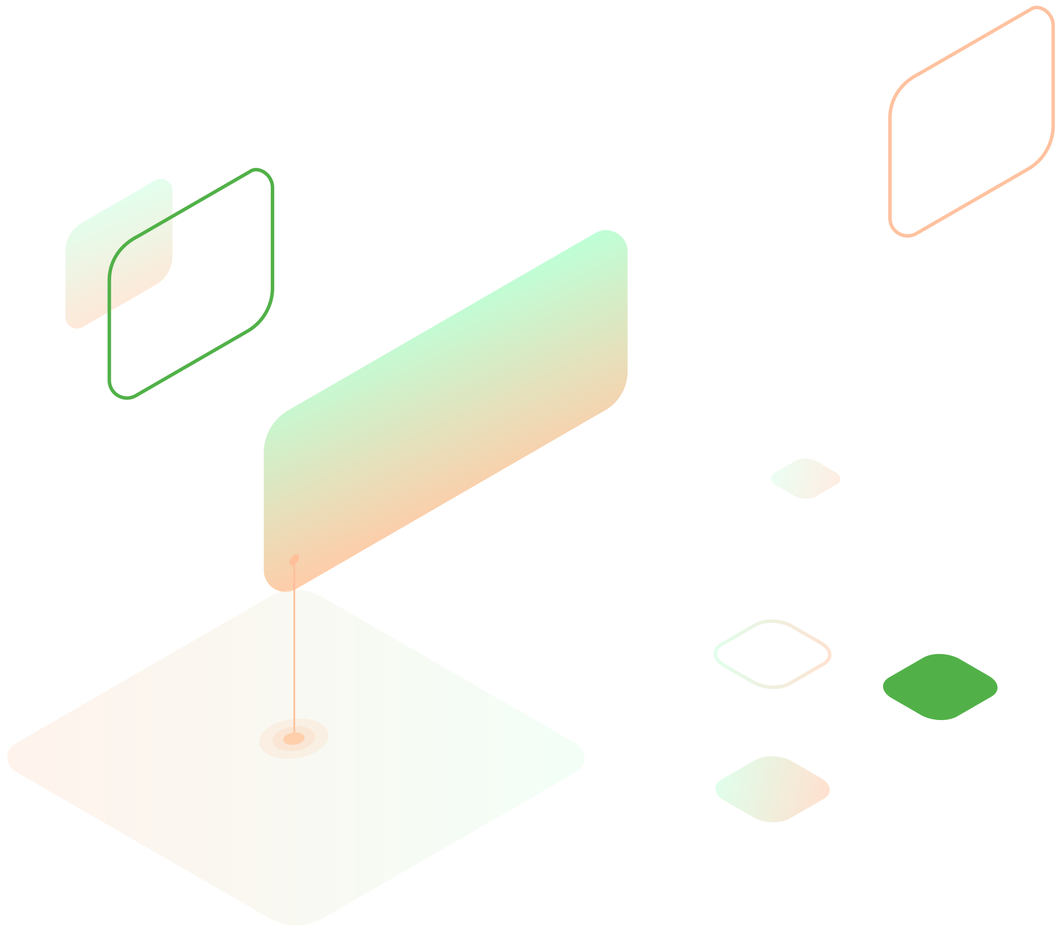
In het voorbeeld hiernaast is elk onderdeel van de webapplicatie ondergebracht in een eigen module. De vm.bicep-module beheert de configuratie van de virtuele machine, de lb.bicep-module regelt de load balancer en de db.bicep-module verzorgt de database. Door deze modules te gebruiken kan je de volledige webapplicatie uitrollen zonder dat je je hoeft te verdiepen in de afzonderlijke details van elk onderdeel apart. Dat scheelt enorm!

```
module vm ‘vm.bicep’ = {
 name: ‘myVmModule’
      params: {
           vmName: ‘myVm’
           vmSize: ‘Standard_DS1_v2’
           adminUsername: ‘adminUser’
           adminPassword: ‘P@ssw0rd!’
           }
}
module lb ‘lb.bicep’ = {
 name: ‘myLbModule’
      params: {
           lbName: ‘myLb’
           frontendIpConfiguration: {
           name: ‘myFrontendConfig’
           publicIpAddressId: ‘myPublicIp’
           }
 }
}
module db ‘db.bicep’ = {
 name: ‘myDbModule’
      params: {
           dbName: ‘myDb’
           skuName: ‘Standard_DTU2’
           maxSizeBytes: ‘1073741824’
 }
}
```

Werk efficiënter samen met herbruikbare modules

De modulariteit van Azure Bicep vereenvoudigt niet alleen het beheer van infrastructuur, maar stelt ook in staat om binnen teams efficiënter samen te werken. Door een team van inhoudelijke experts de modules te laten ontwikkelen en onderhouden, kun je ervoor zorgen dat best practices worden gevolgd en dat de modules up-to-date blijven met de laatste ontwikkelingen.

Andere teams binnen je organisatie kunnen deze modules vervolgens eenvoudig hergebruiken. Het enige wat zij hoeven te doen, is een hoofd-deployment- of parameterbestand schrijven—zonder zich in alle technische details te verdiepen. Op die manier wordt het ook voor minder ervaren gebruikers mogelijk om een complexe infrastructuur te deployen.
Het verlaagt de drempel en helpt de productiviteit te behouden.



Modules maken en gebruiken

Om een module te maken in Azure Bicep, definieer je deze in een apart *.bicep*-bestand en voorzie je het van de benodigde parameters.
In dit modulebestand staat de logica voor het uitrollen van een specifieke resource of een groep van resources. Vervolgens kun je deze module aanroepen vanuit je hoofddeploymentbestand, waarbij je de benodigde parameters meegeeft.

Laten we als voorbeeld een module maken voor het uitrollen van een opslagaccount:

// storage.bicep

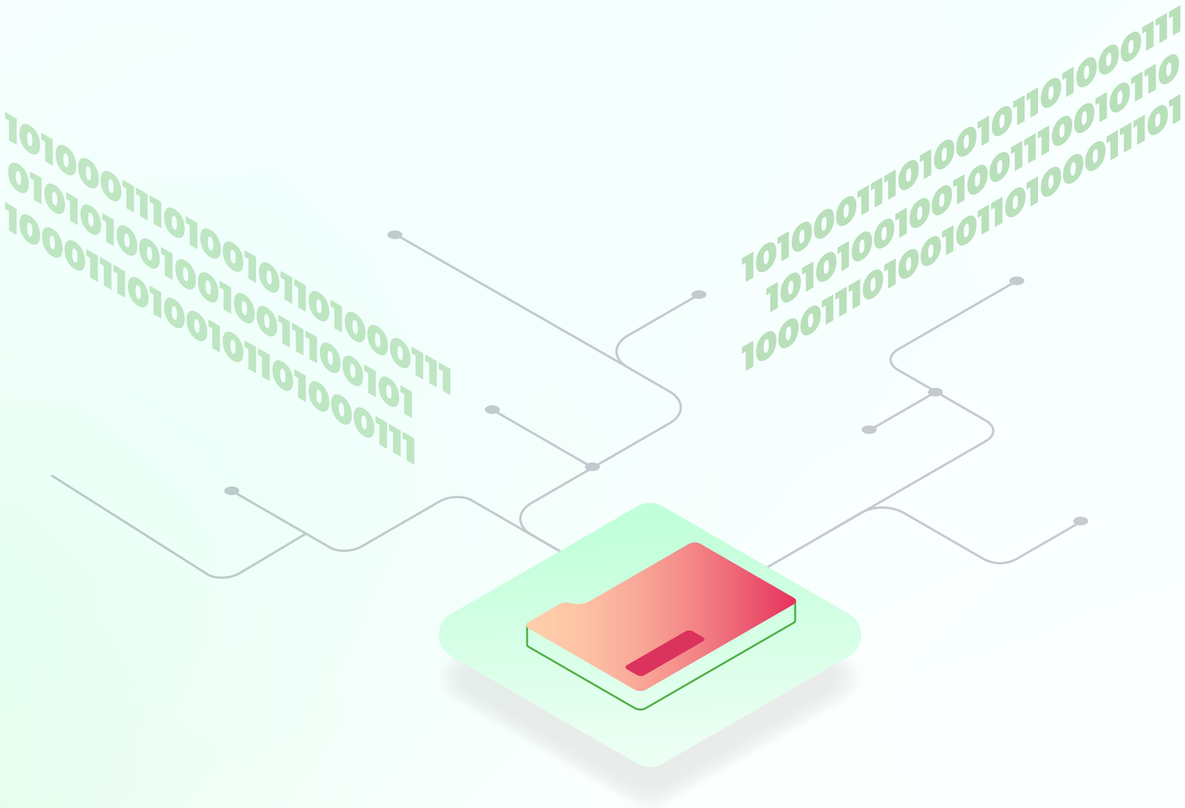
```
resource storageAccount ‘Microsoft.Storage/storageAccounts@2021-01-01’ = {
name: params.storageAccountName
location: resourceGroup().location
sku: {
name: ‘Standard_LRS’
}
kind: ‘StorageV2’
}
// main.bicep
module storage ‘storage.bicep’ = {
name: ‘myStorageModule’
params: {
storageAccountName: ‘mystorageaccount’
}
}
```

Door op deze manier te werken, kun je een bibliotheek opbouwen van herbruikbare modules die gedeeld kunnen worden tussen projecten en teams. Dit vergroot de consistentie en verkleint de kans op fouten of verkeerde configuraties.

Modules onderhouden en updaten

Wanneer je een gespecialiseerd team van inhoudelijke experts verantwoordelijk maakt voor het ontwikkelen en onderhouden van de modules, weet je dat de modules van hoge kwaliteit blijven en ze voldoen aan best practices. Het team kan de modules continu bijwerken zodra er nieuwe functionaliteiten of verbeteringen beschikbaar komen, zodat je infrastructuur altijd up-to-date blijft.

Door het onderhoud van modules centraal te organiseren, kun je snel inspelen op eventuele problemen of bugs. Dit ontlast de afzonderlijke teams, waardoor zij zich kunnen richten op het leveren van waarde aan de organisatie.



Terug naar de hoofdvraag: Hoe maak en beheer je herbruikbare Azure Bicep-modules voor een schaalbare en gestandaardiseerde infrastructuur?

In dit hoofdstuk heb je geleerd hoe je Azure Bicep-modules opzet, hergebruikt en beheert om je infrastructuur gestandaardiseerd, schaalbaar en beheersbaar te houden. We zagen hoe een modulaire aanpak complexe configuraties vereenvoudigt en bijdraagt aan de consistentie en de schaalbaarheid binnen je organisatie.

Door modules centraal op te slaan, kunnen teams met uiteenlopende ervaringsniveaus infrastructuur makkelijker deployen. Dit verhoogt de productiviteit en stimuleert samenwerking. Dit stelt je als organisatie in staat flexibeler op te treden en beter in te spelen op veranderende zakelijke behoeften.

Dit hoofdstuk bood je de handvatten om herbruikbare Azure Bicep-modules te bouwen en te beheren. In het volgende hoofdstuk gaan we een stap verder dan dat en richten we ons op het optimaliseren van deze modules, het efficiënter inrichten en het vereenvoudigen van het onderhoud ervan.

HOOFDSTUK 4

Infrastructuur uitrollen met betrouwbare modules en CI/CD

In dit hoofdstuk bouwen we voort op de vorige inzichten, met de focus op het automatisch en betrouwbaar uitrollen van infrastructuur via Azure Verified Modules (AVM) en CI/CD. Door gebruik te maken van AVM in CI/CD-pipelines (zoals Azure DevOps of GitHub Actions), wordt het managen van je infrastructuur meer schaalbaar en veiliger. In dit hoofdstuk laten we zien hoe je met AVM en CI/CD het deploymentproces kunt versimpelen, hoe je tijd bespaart en hoe je de risico's kunt beperken.

Hoe zet je AVM en CI/CD in voor betrouwbare en schaalbare infrastructuur?

Zodra je kiest voor Azure Bicep (want wij geloven sterk in Azure en Infrastructure as Code), is het doel om je infrastructuur schaalbaar, herhaalbaar, voorspelbaar, maar bovenal beheersbaar te maken. Het devies luidt: *“Automatiseer, automatiseer, automatiseer*” en dat is precies wat we gaan doen!

Het gebruik van CI/CD-pipelines en Azure Verified Modules helpt voor een professionele, efficiënte infrastructuur. Laten we eens kijken welke stappen benodigd zijn om je infrastructuur schaalbaar, herhaalbaar, voorspelbaar en beheersbaar te maken.

Uitdaging: Het schaalbaar, herhaalbaar en beheersbaar maken

Waarom zou je als organisatie kiezen voor een geautomatiseerde CI/CD-aanpak? Een logisch startpunt is het volgen van de richtlijnen uit het Well-Architected Framework (WAF) en het Cloud Adoption Framework (CAF): de principes die je als organisatie idealiter nastreeft.

Een **main.bicep-bestand** in combinatie met **parameterbestanden** ondersteunt deze richtlijnen. Het biedt standaardisatie, maar ook flexibiliteit: je kunt eenvoudig waarden aanpassen via parameters. Denk bijvoorbeeld aan het wijzigen van de SKU van een virtuele machine, het aanpassen van back-upbeleid of het in- of uitschakelen van bepaalde services tijdens de ontwikkelfase.

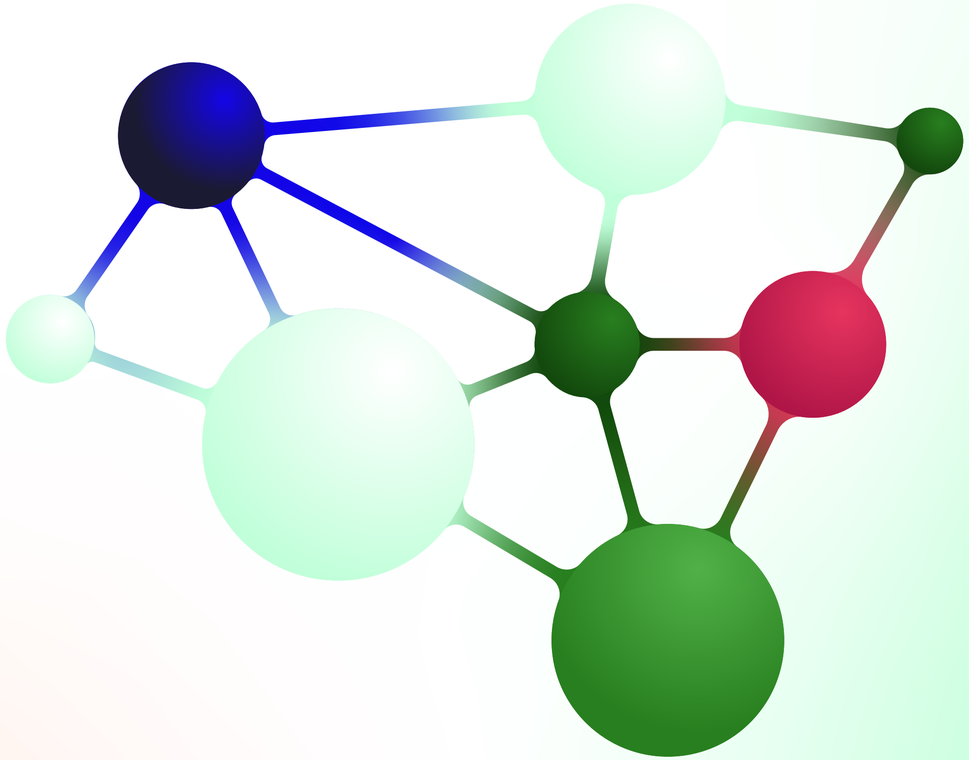
Het gebruik van een *main.bicep*-bestand in combinatie met parameterbestanden helpt om aan deze richtlijnen te voldoen. Het biedt standaardisatie, maar ook flexibiliteit: je kunt waarden simpel aanpassen via parameters. Zo kan je bijvoorbeeld de SKU van een virtuele machine wijzigen, back-upbeleid aanpassen of bepaalde services tijdens de ontwikkelfase in- of uitschakelen.

CI/CD maakt dit proces nog krachtiger. Met tools zoals PSRule.Rules.Azure kun je pull requests automatisch laten valideren voordat de infrastructuur daadwerkelijk wordt uitgerold. Zo wordt het bicepparam-bestand vooraf gecontroleerd en weet je zeker dat elke deployment voldoet aan de best practices.

Waarom het gebruik van Azure Verified Modules (AVM)?

Azure Verified Modules bieden:

- **Standaardisatie:** Microsoft-gevalideerde sjablonen zorgen voor naleving van best practices.
- **Modulariteit:** Infrastructuurcomponenten worden ingekapseld in herbruikbare modules.
- **Beveiliging en compliance:** AVM bevat ingebouwde beleidsregels en beschermingsmaatregelen.
- **Schaalbaarheid:** Modules zijn eenvoudig herbruikbaar binnen verschillende omgevingen en teams.



Wanneer je werkt met meerdere teams, is goed versiebeheer onmisbaar. Door gebruik te maken van een deployment-pijplijn voorkom je fouten als git-sync-issues of het per ongeluk pushen van lokale testcode naar productie. CI/CD-validatie, met tools als bijvoorbeeld PSRule.Rules.Azure, zorgt ervoor dat pull requests automatisch gecontroleerd worden vóór deployment plaatsvindt. Engineers kunnen pas code pushen als die door de pipeline validatie komt. Dit vermindert de kansen op fouten aanzienlijk.

Het structureren van een AVM-based Bicep deployment

Hieronder delen we een voorbeeld van een mappenstructuur die je kan helpen om je CI/CD georganiseerd te houden.

Mappenstructuur

Een goed georganiseerde AVM-gebaseerde Bicep-projectstructuur kan er als volgt uitzien:

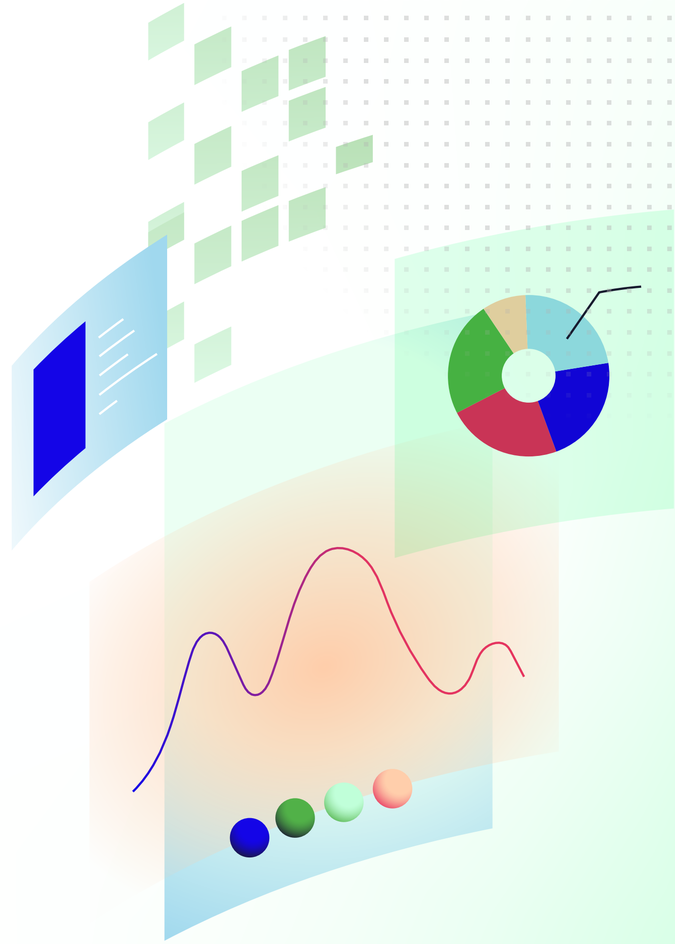
```
/infra
├── bicepparam
│   ├── dev.bicepparam
│   ├── acc.bicepparam
│   └── prod.bicepparam
├── main.bicep
├── azure-pipelines.yml
└── .github/workflows/deploy.yml
```

Een Azure Verified Module gebruiken in main.bicep

```bicep
targetScope = 'subscription'
@description('Customer Name (max 15 characters to prevent long resource names)')
@minLength(3)
@maxLength(15)
param customerName string

@description('Deployment Location')
param location string

@description('Location Short Code (2-4 characters, e.g., "weu" for West Europe)')
@minLength(2)
@maxLength(4)
param locationShortCode string

@description('Environment Type')
@allowed([
  'dev'
  'acc'
  'prod'
])
param environmentType string

// Resource Group Naming Convention
var resourceGroupName = 'rg-${customerName}-${environmentType}-${locationShortCode}'

// Storage Account Name
@description('Storage Account Name (must be unique, 3-24 characters, lowercase)')
@minLength(3)
@maxLength(24)
param storageAccountName string

// Azure Verified Modules - Start Here

module createResourceGroup 'br/public:avm/res/resources/resource-group:0.4.1' = {
  name: 'create-resource-group'
  params: {
    name: resourceGroupName
    location: location
```

CODE GAAT VERDER
OP VOLGENDE PAGINA

```
  }
}

module storageAccount 'br/public:avm/res/storage/storage-account:0.17.0' = {
  name: 'create-storage-account'
  scope: resourceGroup(resourceGroupName)
  params: {
    name: storageAccountName
    location: location
    kind: 'StorageV2'
    skuName: 'Standard_LRS'
  }
}

Environment-Specific Parameters (bicepparam/dev.bicepparam)
using './main.bicep'

param customerName = 'contoso'
param location = 'westeurope'
param locationShortCode = 'weu'
param environmentType = 'dev'

// Derived storage account name (matches Azure's naming rules)
param storageAccountName = 'st${customerName}${environmentType}${locationShortCode}'
```



Omgevingsspecifieke parameters (bicepparam/dev.bicepparam)

```yaml
name: ‘Infrastucure Deployment’

trigger:
- main

pool:
  vmImage: ubuntu-latest

parameters:
  - name: subscriptionId
    displayName: ‘Subscription Id’
    type: string
    default: ‘’

  - name: environmentType
    displayName: ‘Deployment Environment’
    type: string
    default: ‘dev’
    values:
      - dev
      - acc
      - prd

steps:
- task: AzureResourceManagerTemplateDeployment@3
  inputs:
    azureResourceManagerConnection: <azureResourceConnectionId>)’
    deploymentScope: ‘Subscription’
    subscriptionId: ${{ parameters.subscriptionId }}
    location: ‘westeurope’
    templateLocation: ‘Linked artifact’
    csmFile: ‘./infra/main.bicep’
    csmParametersFile: infra/bicepparam/${{ parameters.environmentType }}.bicepparam
    deploymentMode: ‘Incremental’
```

GitHub Actions

Als jouw organisatie de voorkeur geeft aan werken met GitHub Actions, dan ben je bij ons aan het juiste adres.
Hieronder vind je een voorbeeld van een workflow.

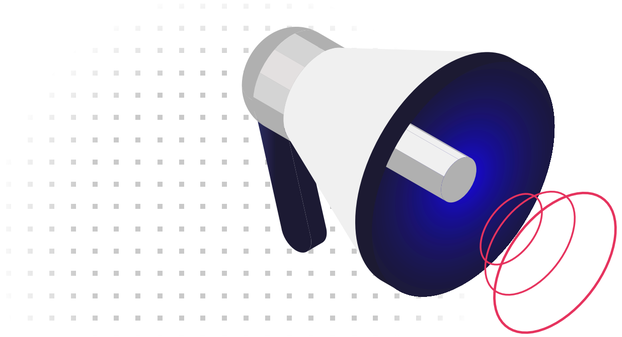
AVM-modules uitrollen met GitHub Actions

Workflow YAML (.github/workflows/deploy.yml)

```yaml
name: 'Infrastructure Deployment'

on:
  workflow_dispatch:
    inputs:
      subscriptionId:
        description: 'Azure Subscription ID'
        required: true

      environment:
        description: 'Deployment Environment'
        required: true
        type: choice
        options:
          - dev
          - acc
          - prd

permissions:
  id-token: write

jobs:
  deploy:
    runs-on: ubuntu-latest
    env:
      AZURE_SUBSCRIPTION_ID: ${{ inputs.subscriptionId }}
      AZURE_ENVIRONMENT: ${{ inputs.environment }}

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Azure Login
        uses: azure/login@v1
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ env.AZURE_SUBSCRIPTION_ID }}

      - name: Deploy Bicep Template
        uses: azure/arm-deploy@v1
        with:
          subscriptionId: ${{ env.AZURE_SUBSCRIPTION_ID }}
          scope: 'subscription'
          region: 'westeurope'
          template: ./infra/main.bicep
          parameters: "./infra/bicepparam/${{ env.AZURE_ENVIRONMENT }}.bicepparam"
          deploymentMode: Incremental
```

Best practices voor een succesvolle AVM-deployment

Azure Verified Modules (AVM) bieden veel voordelen, maar een goede implementatie staat of valt met het volgen van de belangrijkste best practices. Hieronder delen we die met je, rondom authenticatie en validatie en herbruikbaarheid:



Best practice 1: Best practices op het gebied van authenticatie

- Gebruik **OIDC** (OpenID Connect) voor GitHub Actions in plaats van Service Principal-wachtwoorden. [GitHub Docs - OIDC]
- Gebruik **Managed Identity** voor Azure DevOps waar mogelijk. [Microsoft Learn – Workload Identity Service]

Best practice 2: Validatie vóór de deployment

- Voer az bicep build uit om de **module-syntaxis te valideren** vóór de deployment.
- Gebruik az deployment sub what-if om de **veranderingen vooraf te bekijken** voordat je ze toepast.

> **Opmerking:**
> Het gebruik van de --what-if vlag werkt niet binnen een pipeline, omdat het vereist dat je een sleutel invoert om de “what-if” te valideren met [Y]. [Microsoft Learn].

Best practice 3: Herbruikbaarheid en schaalbaarheid

- Sla modules op in een **Azure Bicep Registry** voor gecentraliseerd beheer.
- Gebruik AVM om gebruik te maken van vooraf geteste, best-practice modules in plaats van aangepaste modules.

Terug naar de hoofdvraag: Hoe automatiseer en schaal je infrastructuur betrouwbaar met AVM en CI/CD?

Met Azure Verified Modules (AVM) maak je de uitrol van Azure-infrastructuur modulair, herbruikbaar én compliant. Door deze modules te integreren in een CI/CD-omgeving via **Azure DevOps Pipelines** of **GitHub Actions**, kunnen teams hun deployments automatiseren én tegelijkertijd best practices waarborgen. Beide tools bieden volledige ondersteuning voor CI/CD met AVM-modules. Welke je kiest, hangt af van je werkomgeving:

- Gebruik **Azure Pipelines** als je organisatie werkt met Azure DevOps.
- Gebruik **GitHub Actions** wanneer je repositories op GitHub staan.

Nu je weet hoe AVM en CI/CD samenwerken om infrastructuur efficiënt en betrouwbaar te deployen, zijn dit logische vervolgstappen om je setup verder te verbeteren:

- Verken extra AVM-modules voor netwerken, compute en security.
- Implementeer CI-validatie met behulp van de Bicep-linter en what-if-analyse om fouten vroegtijdig op te sporten.
- Sla AVM-modules op in een Azure Bicep Registry om bredere adoptie binnen teams te stimuleren.

Wat hebben we geleerd?

Alles samenvattend: Jouw pad naar een schaalbare en efficiënte infrastructuur

In dit e-book heb je ontdekt dat ClickOps weliswaar handig is voor het leren en oplossen van problemen, maar dat Infrastructure as Code (IaC) de echte sleutel is wanneer het aankomt op schaalbare deployment. Door gebruik te maken van pipelines en CI/CD kun je deployments standaardiseren en daarmee je consistentie en fouttolerantie aanzienlijk verbeteren.

We hebben ons gaandeweg verdiept in verschillende technologieën, Terraform, Pulumi, ARM-Templates en Azure Bicep: Elk met hun eigen kracht in het opzetten van schaalbare en compliant cloudinfrastructuren.

Azure Verified Modules (AVM) tillen dit naar een hoger niveau. Ze maken je infrastructuur modulair, herbruikbaar en compliant, waardoor deze makkelijker te beheren en onderhouden is.

Door AVM te combineren met Azure DevOps Pipelines of GitHub Actions automatiseer je het volledige deploymentproces, waarbij je teams toch aan best practices kunnen blijven voldoen.

Staat je code op GitHub, dan is GitHub Actions ideaal, anders biedt Azure DevOps Pipelines een krachtig alternatief.

Het opnemen van AVM Bicep-modules in je CI/CD-workflows stelt je in staat om een robuuste en wendbare cloudinfrastructuur op te bouwen één die innovatie ondersteunt en snel inspeelt op verandering.

Met de inzichten uit dit e-book ben je nu in staat met IaC je infrastructuur nu efficiënt te beheren. Door AVM Bicep-modules onderdeel te maken van je CI/CD-werkwijze, bouw je aan een cloudinfrastructuur die niet alleen robuust en veilig is, maar ook wendbaar en toekomstbestendig.

We hopen dat je dit e-book met plezier hebt gelezen en dat het je helpt om je infrastructuurbeheer te optimaliseren maar vooral klaar te maken voor de toekomst!

Team Intercept

INTERCEPT

Gratis Infrastructure Scan

Wist je dat je in aanmerking komt voor een gratis Infrastructure Scan?

Met een **Infrastructure Scan** krijg je snel inzicht in de prestaties van je Azure-omgeving. Je ontvangt waardevolle aanbevelingen om de betrouwbaarheid, beveiliging en kosten van je infrastructuur te verbeteren. Zo voorkom je onnodige kosten, beveiligingsrisico's en prestatieproblemen.

Waarom dit ertoe doet

Door de scan ontdek je direct verbeterpunten op het gebied van betrouwbaarheid, compliance, kostenbesparing, en beveiliging. Hiermee voorkom je onnodige uitgaven, downtime en risico's, zodat je infrastructuur optimaal blijft presteren.

→ Vraag nu je gratis Infrastructure Scan aan!

Security Workshop

Twijfel je over de security van je data? Dan loop je het risico alles te verliezen. Doe mee aan onze **gratis Azure Security Workshop** en ontdek in slechts één uur hoe je jouw security optimaliseert. Praktisch, helder en direct toepasbaar.

→ Meld je nu aan voor de gratis workshop