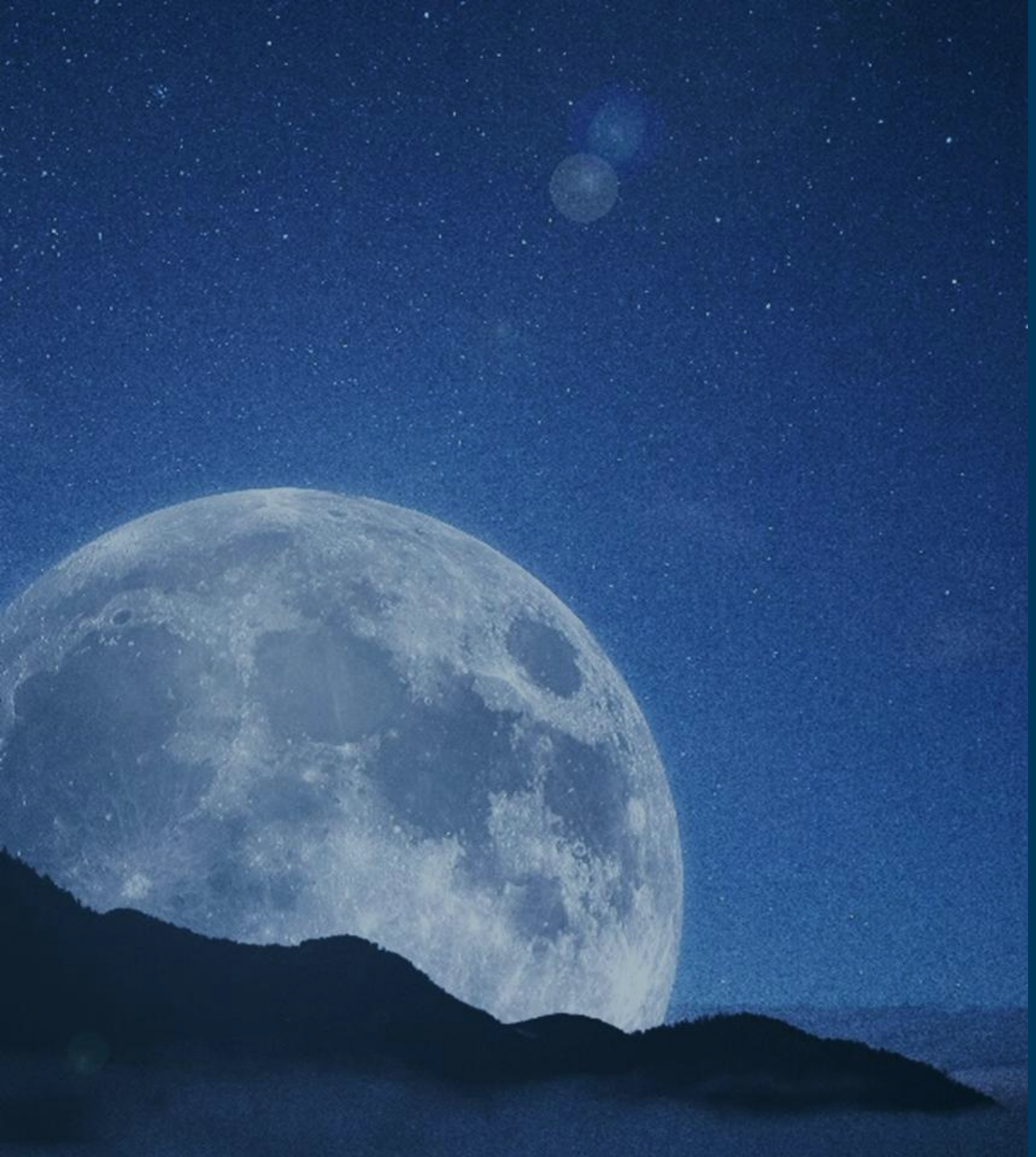
LUNARIS
Digital Solutions

BICEP BEST PRACTICES

Wiederverwendbare und wartbare Azure Deployments

Florian Bader



MIT WEM HABT IHR ES ZU TUN?

Florian Bader

Solution Architect | CTO

Florian.Bader@lunaris.digital

https://lunaris.digital
https://github.com/florianbader

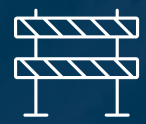
LUNARIS
Digital Solutions

KEY TAKE AWAYS

Was ist Bicep und
wieso braucht es das?

Wie strukturiere ich meinen
Infrastructure as Code in Bicep?

Wie standardisiere ich meinne
Infrastructure as Code in Bicep?



Wie stelle ich die Qualität meiner
Infrastructure as Code in Bicep
sicher?

```
targetScope = 'resourceGroup'

type AppSettings = {
  appName: string
  count: int
}

@description('Decorator example.')
@allowed([
  'dev'
  'prod'
])
param environment string = 'dev'

@description('Type example.')
param settings AppSettings = {
  appName: 'demo'
  count: 2
}

param deployStorage bool = true

var location = resourceGroup().location
var storageName = 'st${uniqueString(resourceGroup().id, settings.appName, environment)}'
var names = [for i in range(0, settings.count): '${settings.appName}-${i + 1}']


resource storage 'Microsoft.Storage/storageAccounts@2023-05-01' = if (deployStorage) {
  name: storageName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  tags: {
    env: environment
    app: settings.appName
  }
  properties: {
    supportsHttpsTrafficOnly: true
  }
}


output summary object = {
  environment: environment
  location: location
  names: names
  storage: deployStorage ? storage.name : 'skipped'
}
```

BICEP FÜR AZURE DEPLOYMENTS

Infrastructure as Code in Azure

Bicep ist eine deklarative Sprache für Infrastructure as Code

- Zielzustand wird beschrieben
- Azure Resource Manager (ARM) kümmert sich inkrementell um den Weg dahin

Bicep ist ein Transpiler nach ARM JSON

- (Fast) Alles was in ARM geht, geht auch in Bicep

Bicep und ARM brauchen keine State Files

- Zustand ist immer der aktuelle Stand in Azure

Bicep Konstrukte:

- Resources, Parameters, Variables, Outputs
- Functions, Loops, Conditions, Modules
- Types, Decorators

ALLER ANFANG IST SCHWER

Wenn aus pragmatischem Start, technische Schulden werden

Big Ball of Bicep

Parameter Wildwuchs

Environment Logik im Code

Copy-Paste statt Wiederverwendung

Kein Gedanke an Security, Betrieb und Compliance

DER ERSTE HEBEL: MODULARISIERUNG

Von der großen Datei zu fachlich geschnittenen Bausteinen

Module schneiden Verantwortung

- Netzwerk, App, Datenbank, Identity, Monitoring
- Jede Einheit hat einen klaren Zweck

Main-Bicep wird zur Orchestrierung

- Module aufrufen, Abhängigkeiten verbinden, Umgebungswerte übergeben

Module brauchen klare Schnittstellen

- Wenige, verständliche Parameter
- Output nur für wirklich benötigte Werte

Fachlich schneiden, nicht nur Dateien auslagern!


```
module storageAccountModule './modules/storageAccount.bicep' = {
  name: 'storageAccount'
  params: {
    location: location
    storageAccountName: storageAccountName
    storageSkuName: storageSkuName
  }
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2023-05-01' existing = {
  name: storageAccountName
}

module keyVaultModule './modules/keyVault.bicep' = {
  name: 'keyVault'
  params: {
    location: location
    keyVaultName: keyVaultName
    keyVaultSkuName: keyVaultSkuName
    workshopSecretValue: workshopSecretValue
  }
}

resource keyVault 'Microsoft.KeyVault/vaults@2023-07-01' existing = {
  name: keyVaultModule.outputs.keyVaultName
}

var storageConnectionString =
'DefaultEndpointsProtocol=https;AccountName=${storageAccount.name};AccountKey=${storageAc
count.listKeys().keys[0].value};EndpointSuffix=${environment().suffixes.storage}'

module appServiceModule './modules/appService.bicep' = {
  name: 'appService'
  params: {
    location: location
    appServicePlanName: appServicePlanName
    appServicePlanSkuName: appServicePlanSkuName
    appServicePlanSkuTier: appServicePlanSkuTier
    webAppName: webAppName
    storageConnectionString: storageConnectionString
    workshopSecretValue: keyVault.getSecret(keyVaultModule.outputs.workshopSecretName)
  }
  dependsOn: [
    storageAccountModule
  ]
}
```

```bicep
// src/Infrastructure/app1/modules/appService.bicep
param location string

param appServicePlanName string
param appServicePlanSkuName string
param appServicePlanSkuTier string

param webAppName string
param appSettings array

module sharedAppService '../../modules/appService.bicep' = {
  name: 'sharedAppService'
  params: {
    location: location
    appServicePlanName: appServicePlanName
    appServicePlanSkuName: appServicePlanSkuName
    appServicePlanSkuTier: appServicePlanSkuTier
    webAppName: webAppName
    appSettings: appSettings
  }
}

// src/Infrastructure/modules/appService.bicep
resource appServicePlan 'Microsoft.Web/serverfarms@2023-
12-01' = {
  name: appServicePlanName
  location: location
  sku: {
    name: appServicePlanSkuName
    tier: appServicePlanSkuTier
  }
  kind: 'app‘
}

resource webApp 'Microsoft.Web/sites@2023-12-01' = {
  name: webAppName
  location: location
  kind: 'app'
  properties: {
    serverFarmId: appServicePlan.id
    siteConfig: {
      appSettings: appSettings
    }
  }
}
```

VON MODULEN ZU APP STANDARDS

Wiederverwendbare Deployment Patterns

Ein Modul ist noch kein Standard

- Wiederverwendung braucht klare Konventionen
- Gleicher App Type sollten ähnlich deployed werden

App Standards bündeln typische Bausteine

- App Service, Identity, Key Vault, Monitoring, Role Assignments als gemeinsames Deployment Pattern

Die App beschreibt nur noch ihre Abweichungen

- Größe, Region, Features, Umgebung
- Standards kommen aus zentraler Struktur

Standards versionieren!

- Code
- Repository
- Private Registry

SAUBERE SCHNITTSTELLEN

Module sollten Erwartungen ausdrücken

Viele lose Einzelparameter sind

- Schwer zu verstehen
- Schwer konsistent zu halten

Besser: Fachliche Konfigurationsobjekte und zusammengehörige Werte bündeln

Custom Typen machen Erwartungen explizit

- Pflichtfelder und optionale Felder
- Erlaubte Werte über Union Types

Decorators ergänzen die Erwartungen

- Validierung: @allowed, @minLength, @maxLength
- Dokumentation mit im Code: @description, @metadata

fail()-Funktion statt gefährlicher Defaults

```
@export()
type appSettingType = {
  @minLength(1)
  name: string
  value: string
}

@minLength(2)
@maxLength(40)
param appServicePlanName string

@allowed([
  'F1'
  'D1'
  'B1'
  'B2'
  'B3'
  'S1'
  'S2'
  'S3'
  'P1v3'
  'P2v3'
  'P3v3'
])
param skuName string

@allowed([
  'Free'
  'Shared'
  'Basic'
  'Standard'
  'PremiumV3'
])
param skuTier string

@minLength(2)
@maxLength(60)
param webAppName string

@description('Application settings passed directly to the web app configuration.')
param appSettings appSettingType[]
```

```bicep
@export()
type secretType = {
  @minLength(1)
  name: string
  value: string
}


@allowed([
  'standard'
  'premium'
])
param keyVaultSku string


param location string


@minLength(3)
@maxLength(24)
param keyVaultName string


@description('Secrets created in the vault to demonstrate looping over arrays of objects.')
param secrets secretType[] = []


module keyVault 'br/public:avm/res/key-vault/vault:0.13.3' = {
  name: 'keyVaultDeployment'
  params: {
    name: keyVaultName
    location: location
    sku: keyVaultSku
    enablePurgeProtection: false
    enableRbacAuthorization: false
    enableVaultForTemplateDeployment: true
    secrets: [for secret in secrets: {
      name; secret.name
      value: secret.value
    }]
  }
}
```

AZURE VERIFIED MODULES

Standards nutzen, statt das Rad neu zu erfinden

Azure Verified Modules (ehemals Public Module Registry) liefert standardisierte Module

- Für Bicep aber auch für Terraform
- Mit Microsoft definierten Qualitätsstandards und Support durch das Produktteam

Resource Modules für einzelne Azure Ressourcen

- Beispiel Key Vault, Storage, App Service, Network
- Inklusive typischer Child und Extension Resources

Pattern Module für wiederkehrende Architekturbausteine

- Beispiel: Hub Network, Landing Zones, AI Baseline
- Mehrere Ressourcen als gemeinsames Deployment

Bessere Defaults, einfaches Erstellen von Child Resources (z.B. Role Assignments)

ROLE ASSIGNMENTS

Rollen sauber Ressourcen zuordnen

Role Assignments gehören in die Infrastruktur

- Principal, Rolle, Scope explizit beschreiben

Keine Role IDs im Template

- GUIDs sind schwer lesbar

Besser: Rollen über Namen

- Modul nimmt Rollenname entgegen (Validierung über Union Type oder Allowed Values)
- Role Definition IDs wird intern aufgelöst

Scope bewusst klein wählen

- Ressource statt Resource Group
- Resource Group statt Subscription

Deterministische Namen verwenden: guid(scope, principalId, roleDefinitionId)


```
@description('The name of the Storage Account')
param storageAccountName string

@description('The ID of the principal to assign the role to')
param principalId string = ''

@description('The type of principal to assign the role to')
@allowed([
  'Device'
  'ForeignGroup'
  'Group'
  'ServicePrincipal'
  'User'
  ''
])
param principalType string = ''

@allowed([
  'Storage Blob Data Contributor'
  'Storage Blob Data Reader'
  'Storage Blob Data Owner'
  'Storage Queue Data Contributor'
  'Storage Table Data Contributor'
  'Storage Account Contributor'
])
param roleDefinitions string[]

var roles = {
  'Storage Account Contributor': '17d1049b-9a84-46fb-8f53-869881c3d3ab'
  'Storage Blob Data Contributor': 'ba92f5b4-2d11-453d-a403-e96b0029c9fe'
  'Storage Blob Data Reader': '2a2b9908-6ea1-4ae2-8e65-a410df84e7d1'
  'Storage Blob Data Owner': 'b7e6dc6d-f1e8-4753-8033-0f276bb0955b'
  'Storage Queue Data Contributor': '974c5e8b-45b9-4653-ba55-5f855dd0fb88'x
  'Storage Table Data Contributor': '0a9a7e1f-b9d0-4cc4-a60d-0319b160aaa3'
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' existing = {
  name: storageAccountName
}

resource roleAuthorization 'Microsoft.Authorization/roleAssignments@2022-04-01' = [
  for roleDefinition in roleDefinitions: {
    name: guid('storage-rbac', storageAccount.id, resourceGroup().id, principalId,
roles[roleDefinition])
    scope: storageAccount
    properties: {
      principalId: principalId
      roleDefinitionId: subscriptionResourceId('Microsoft.Authorization/roleDefinitions',
roles[roleDefinition])
      principalType: empty(principalType) ? null : principalType
    }
  }
]
```

```bicep
// functions.bicep
var resources = loadJsonContent('resources.json')
var locations = loadJsonContent('locations.json')

@export()
type resourceType =
  | 'storage'
  | 'keyVault'
  | 'appServicePlan'
  | 'appService'

@export()
type location = 'westeurope'

@export()
type environment = 'dev' | 'prd'

@export()
func getLocationShortName(location location) string => locations[location]

@export()
func constructResourceName(resourceType resourceType, appName string, environment
environment, location location) string =>
  '${toLower(appName)}-${environment}-${resources.prefixes[resourceType]}-
${locations[location]}-01'
```

```bicep
// appService.bicep
param tags object = {}

var appServicePlanName = constructResourceName('appServicePlan', appName,
environment, location)
var webAppName = constructResourceName('appService', appName, environment, location)

var storageAccountName = constructResourceName('storage', appName, environment,
location)

var keyVaultName = constructResourceName('keyVault', appName, environment, location)
var keyVaultUri =
'https://${keyVaultName}.${az.environment().suffixes.keyvaultDns}/‘

var resourceTags = union(commonTags, {
  application: appName
  environment: environment
  location: getLocationShortName(location)
}, tags)
```

GOVERNANCE IN INFRASTRUCTURE

Naming, Konfiguration und Tags standardisieren

Naming zentral definieren

- JSON-Datei für Resource Prefixes, Environment und Location Abkürzungen
- Custom Function für Resource Naming

Tags zentral definieren

- Tags über Funktion zusammenführen
- Abhängig von Parametern unterschiedliche Tags setzen

Allgemeine Variable zentral definieren

- Global Resourcen und ihre Namen für Monitoring, Networking und Co.

Policy as Code

- Azure Policies über Bicep anlegen

QUALITÄTSSICHERUNG VOR DEM DEPLOYMENT

Fehler und riskante Änderungen finden, bevor Azure sie ausführt

Linting prüft das Template

- Regeln für Sicherheit, Struktur und Wartbarkeit
- Konfigurierbar über bicepconfig.json

Linter-Regeln bewusst schärfen

- Security-Regeln als Fehler behandeln
- Ausnahmen explizit mit Begründung setzen

What-If zeigt geplante Änderungen

- Create, Modify, Delete und NoChange vor dem Deployment
- Oft leider umfangreicher als gewollt

Qualität gehört in Pull Request Validierung

- Validate, Lint, What-If automatisieren
- Azure Cost Estimator
- Security Checks (checkov, Azure


```yaml
steps:
  - name: Install latest Bicep CLI
    shell: bash
    run: az bicep install --upgrade


  - name: Lint template
    shell: bash
    run: az bicep lint --file "${{ inputs.template-file }}"


  - name: Validate deployment
    uses: azure/bicep-deploy@v2
    with:
      type: deployment
      operation: validate
      scope: resourceGroup
      subscription-id: ${{ inputs.subscription-id }}
      resource-group-name: ${{ inputs.resource-group }}
      name: ${{ inputs.deployment-name }}
      template-file: ${{ inputs.template-file }}
      parameters-file: ${{ inputs.parameters-file }}

  - name: Preview deployment
    uses: azure/bicep-deploy@v2
    with:
      type: deployment
      operation: whatIf
      scope: resourceGroup
      subscription-id: ${{ inputs.subscription-id }}
      resource-group-name: ${{ inputs.resource-group }}
      name: ${{ inputs.deployment-name }}
      template-file: ${{ inputs.template-file }}
      parameters-file: ${{ inputs.parameters-file }}
```

```yaml
steps:
  - name: Deploy template as stack
    uses: azure/bicep-deploy@v2
    with:
      type: deploymentStack
      operation: create
      scope: resourceGroup
      subscription-id: ${{ inputs.subscription-id }}
      resource-group-name: ${{ inputs.resource-group }}
      name: ${{ inputs.stack-name || inputs.deployment-name }}
      template-file: ${{ inputs.template-file }}
      parameters-file: ${{ inputs.parameters-file }}
      location: ${{ inputs.location }}
      action-on-unmanage-resources: delete
```

```
az stack sub create \
--name 'az-stack-demo' \
--location 'westeurope' \
--template-file 'main.bicep' \
--parameters 'main.bicepparam' \
--action-on-unmanage 'detachAll' \
--deny-settings-mode 'denyWriteAndDelete' \
--deny-settings-apply-to-child-scopes
```

DEPLOYMENT STACKS

Ressourcen als zusammengehörige Einheit verwalten

Deployment Stacks verwalten mehrere Ressourcen als Einheit

- Stack definiert, welche Ressourcen zusammengehören
- Möglich auf Resource Group, Subscription und Management Group Scope

Problem: Incremental Deployment räumt nicht auf

- Eine gelöschte Ressource im Bicep wird nicht in Azure gelöscht
- Stacks machen den Lifecycle explizit steuerbar
  - Detach oder Delete

Deny Settings schützen Ressourcen

- Änderungen oder Löschungen außerhalb des Stacks und Bicep verhindern
- Kein Resource Lock sondern Stack Lock

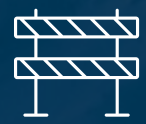
Leider kein What-If Support für den Stack, aber für die Ressourcen.

RECAP

Wartbarkeit entsteht durch Struktur

Standards helfen, konsistent zu bleiben

AVM hilft, das Rad nicht neu erfinden zu müssen



Saubere Pipeline schützt das Deployment


LUNARIS
Digital Solutions