



WoF Bootcamp

Infrastructure as Code

(22WC21)

Agenda

- Einführung in Infrastructure as Code (IaC)
- Erste Schritte mit Bicep und ARM Templates
- Deployment erster Ressourcen in Azure mit Bicep/ARM
- Ausblick auf Terraform
- Umgang mit Modulen
- Aufbau einer Pipeline zum automatischen Bereitstellen von Azure Ressourcen
- Ausblick auf Enterprise Scale

Einführung in Infrastructure as Code (IaC)

Was ist Infrastruktur als (aus) Code?

- Infrastructure as Code (IaC) ist ein Ansatz zur Infrastrukturautomatisierung, der auf Denkmethoden aus der Softwareentwicklung basiert..
- Es betont **konsistente, wiederholbare** Routinen für die **Bereitstellung** und **Änderung** von Systemen und deren Konfiguration.
- **Änderungen werden am Code (der Definitionen) vorgenommen** und dann durch unbeaufsichtigte Prozesse, die eine gründliche Validierung umfassen, auf Systeme ausgerollt.

Tooling Kategorien

Ad Hoc Scripts

Configuration
Management (CM)
Tools

Server Templating
Tools

Server Provisioning
Tools

Ad-Hoc Scripts

- Der einfachste Ansatz zum Automatisieren von allem besteht darin, ein Ad-hoc-Skript zu schreiben.

Sie nehmen manuelle Arbeit
und zerlegen sie in Schritte



Wählen Sie Ihren Favoriten
Skriptsprache



Definieren Sie jede dieser
Schritte im Code



Ausführen dieses Skripts

```
# Update the apt-get cache
sudo apt-get update

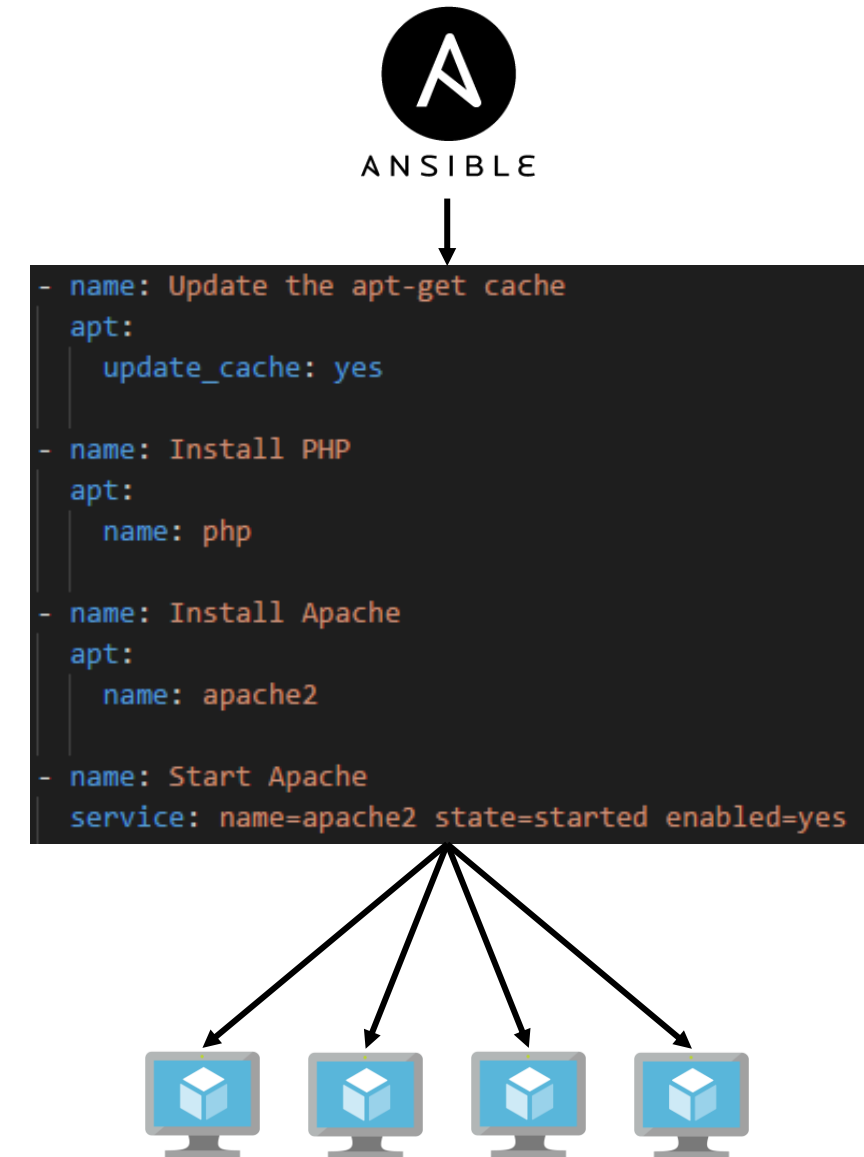
# Install PHP
sudo apt-get install -y php

# Install Apache
sudo apt-get install -y apache2

# Start Apache
sudo service apache2 start
```

Configuration Management Tools

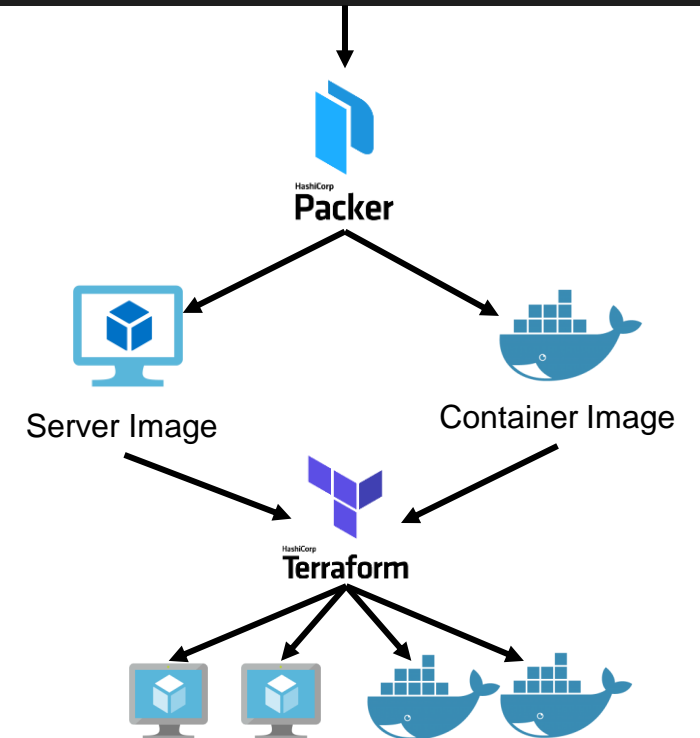
- Chef, Puppet, Ansible und SaltStack sind Konfigurationsverwaltungstools, die zum Installieren und Verwalten von Software auf vorhandenen Servern entwickelt wurden.
- **Coding conventions** – Konsistente & vorhersehbare Struktur, Dateilayout, klar benannte Parameter, Geheimverwaltung, etc..
- **Idempotent Code** – Führen Sie denselben Code wiederholt aus, während Sie dasselbe Ergebnis erzeugen.
- **Distribution** – Im Gegensatz zu Ad-hoc-Skripten sind CM-Tools speziell für die Verwaltung einer großen Anzahl von Remoteservern konzipiert..



Server Templating Tools

- Wachsen in Ihrer Popularität, sind Server-Vorlagen-Tools wie Docker, Packer und Vagrant.
- Erstellen Sie ein Abbild eines Servers, der einen vollständig eigenständigen "Schnappschuss" des Betriebssystems, der Software, der Dateien und aller anderen relevanten Details erfasst.
- Server-Templating ist eine Schlüsselkomponente der Umstellung auf unveränderliche Infrastrukturen.
- "Sysprep" – der einfache Start ins Server-Templating....

```
"builders": [{  
  "type": "azure-arm",  
  
  "client_id": "d4db5ab8-ca6b-451e-b0a8-2905523cf168",  
  "client_secret": "Passw0rd",  
  "tenant_id": "72f988bf-86f1-41af-91ab-2d7cd011db47",  
  "subscription_id": "e73c1dbe-2574-4f38-9e8f-c813757b1786",  
  
  "managed_image_resource_group_name": "PackerDemo-RG",  
  "managed_image_name": "myPackerImage",  
  
  "os_type": "Linux",  
  "image_publisher": "Canonical",  
  "image_offer": "UbuntuServer",  
  "image_sku": "16.04-LTS",  
}]
```



Servers (or “resources”) Provisioning Tools

- Bereitstellungstools wie Terraform, Azure ARM Templates, AWS CloudFormation und OpenStack Heat sind für die Erstellung der Server selbst verantwortlich.
- Sie können diese Tools nicht nur Server erstellen, sondern auch andere Ressourcen wie Datenbanken, Load Balancer, Firewall-Einstellungen, Speicher usw.

```
# Create an Azure resource group
resource "azurerm_resource_group" "terraform" {
  name     = "Terraform-RG"
  location = "${var.location}"
}

# Create a virtual network in the Terraform resource group
resource "azurerm_virtual_network" "terraform" {
  name            = "Terraform-VNet"
  address_space   = ["172.16.0.0/16"]
  resource_group_name = "${azurerm_resource_group.terraform.name}"
  location        = "${var.location}"
}

# Create a subnet in Terraform VNet
resource "azurerm_subnet" "terraform" {
  name                 = "Subnet-01"
  resource_group_name = "${azurerm_resource_group.terraform.name}"
  virtual_network_name = "${azurerm_virtual_network.terraform.name}"
  address_prefix       = "172.16.1.0/24"
}
```



Terraform



Und so viel mehr...

Azure Bicep - Demo

Installation

Example –Resource Definition in BICEP

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = {  
  name: 'azstorageacct'  
  location: 'eastus'  
  sku: {  
    name: 'Standard_LRS'  
  }  
  kind: 'StorageV2'  
  properties: {  
    accessTier: 'Hot'  
  }  
}
```

Deploying resources with Dependencies

```
resource appServicePlan 'Microsoft.Web/serverFarms@2020-06-01' = {  
  name: 'toy-product-launch-plan-starter'  
  location: 'eastus'  
  sku: {  
    name: 'F1'  
    tier: 'Free'  
  }  
}
```

```
resource appServiceApp 'Microsoft.Web/sites@2020-06-01' = {  
  name: 'toy-product-launch-1'  
  location: 'eastus'  
  properties: {  
    serverFarmId: appServicePlan.id  
    httpsOnly: true  
  }  
}
```

BICEP – Parameter Example

```
param appServiceAppName string
param appServiceAppName string = 'appsvcdemo1'
param location string = resourceGroup().location
param storageAccountName string = uniqueString(resourceGroup().id)
```

Using Parameter Value in Template

```
resource appServiceApp 'Microsoft.Web/sites@2020-06-01' = {
  name: appServiceAppName
  location: 'eastus'
  properties: {
    serverFarmId: appServicePlan.id
    httpsOnly: true
  }
}
```

BICEP – Parameter in Deployment command

```
az deployment group create --template-file main.bicep \  
  --parameters environmentType=nonprod
```

BICEP – Objects

Object definition

```
param appServicePlanSku object = {  
  name: 'F1'  
  tier: 'Free'  
  capacity: 1  
}
```

Referencing object in code

```
resource appServicePlan 'Microsoft.Web/serverFarms@2020-06-01' = {  
  name: appServicePlanName  
  location: location  
  sku: {  
    name: appServicePlanSku.name  
    tier: appServicePlanSku.tier  
    capacity: appServicePlanSku.capacity  
  }  
}
```

BICEP – Parameter Decorator

```
@description('The name of the environment. This must be dev, test, or prod.')  
param environmentName string = 'dev'
```

```
@description('The unique name of the solution. This is used to ensure that  
resource names are unique.')  
param solutionName string = 'toyhr${uniqueString(resourceGroup().id)}'
```


BICEP – Parameter File

Parameter files make it easy to specify parameter values together as a set. Parameter files are created by using the JSON language. You can supply a parameter file when you deploy your Bicep template. Here's what a parameter file looks like:

```
{
  "$schema":
    "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "appServicePlanInstanceCount": {
      "value": 3
    },
    "appServicePlanSku": {
      "value": {
        "name": "P1v3",
        "tier": "PremiumV3"
      }
    },
    "cosmosDBAccountLocations": {
      "value": [
        {
          "locationName": "australiaeast"
        },
        {
          "locationName": "southcentralus"
        },
        {
          "locationName": "westeurope"
        }
      ]
    }
  }
}
```

Using parameter file in Deployment

```
az deployment group create \  
  --template-file main.bicep \  
  --parameters main.parameters.json
```

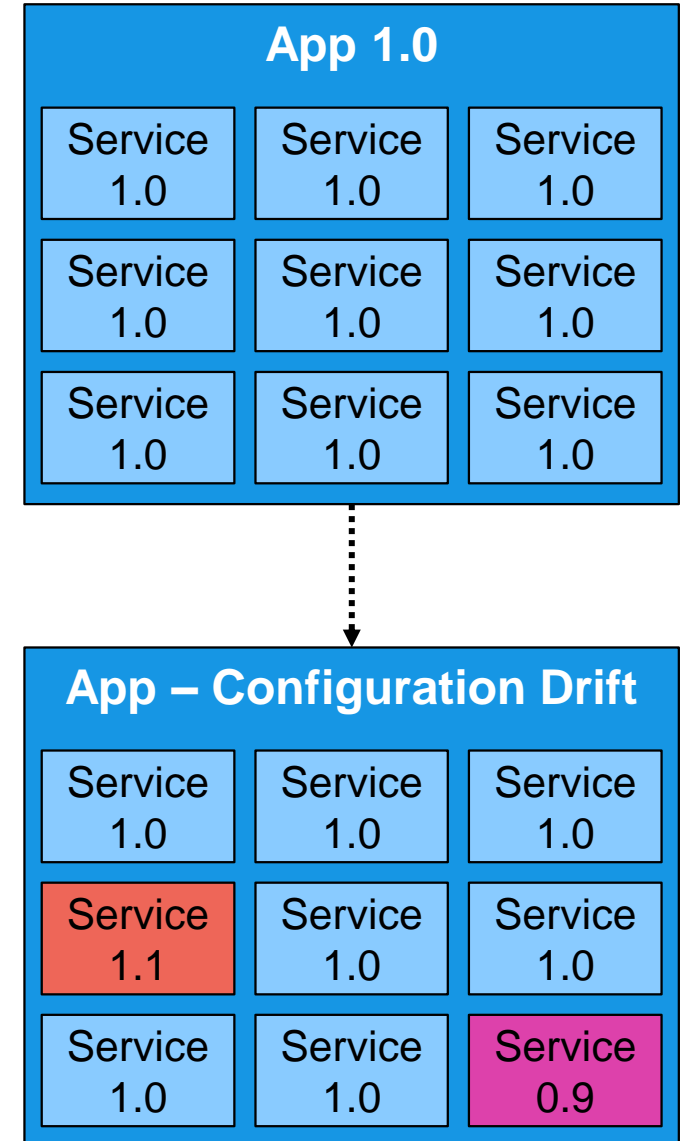
Challenge 1

- Installation der Tools auf eurem Rechner
- Nutzung von Bicep Code, um einen Storage Account in einer **vorhandenen Ressource Gruppe** bereitzustellen

“Mutable & Immutable” Infrastructure

Mutable Infrastructure

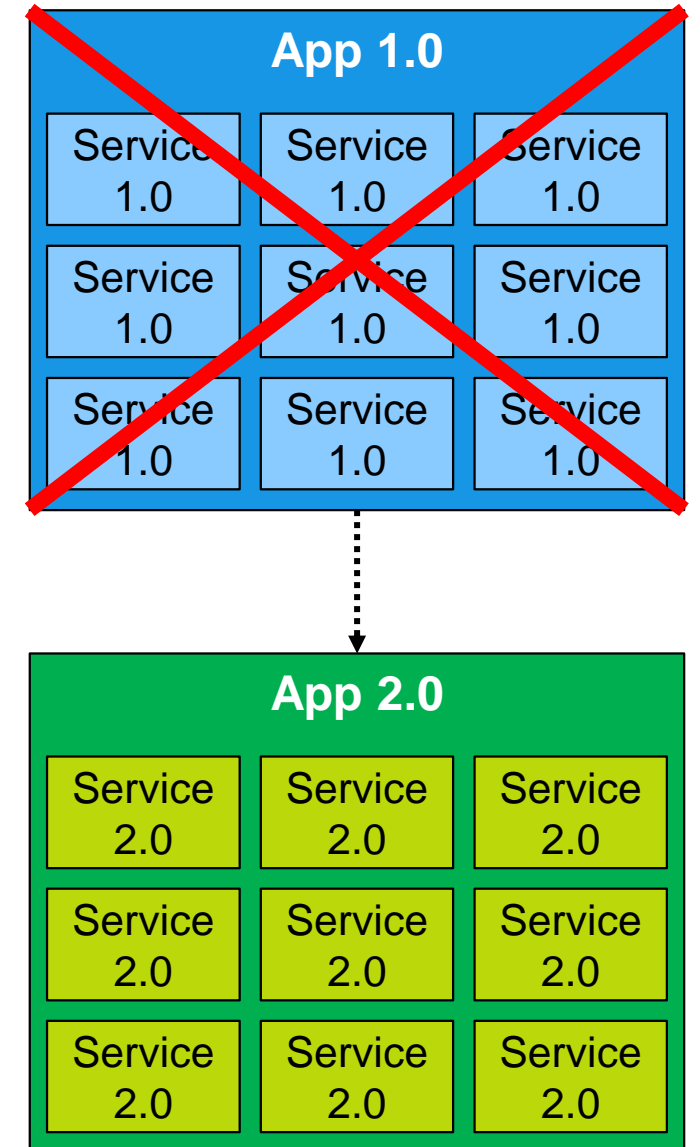
- Die Infrastruktur wird ständig aktualisiert, gepatcht und abgestimmt, um den laufenden Anforderungen des Zwecks gerecht zu werden, dem sie dient.
- CM-Tools wie Chef oder Puppet sind in der Regel standardmäßig ein veränderliches Infrastrukturparadigma.
- Im Laufe der Zeit erstellt jeder Server, wenn Sie immer mehr Updates anwenden, einen einzigartigen Änderungsverlauf.
- Infolgedessen wird jeder Dienst etwas anders als alle anderen, was zu Konfigurationsdrift führt und zu Fehlern führen kann, die schwierig zu diagnostizieren und reproduzieren können.



Mutable & Immutable Infrastructure

Immutable Infrastructure

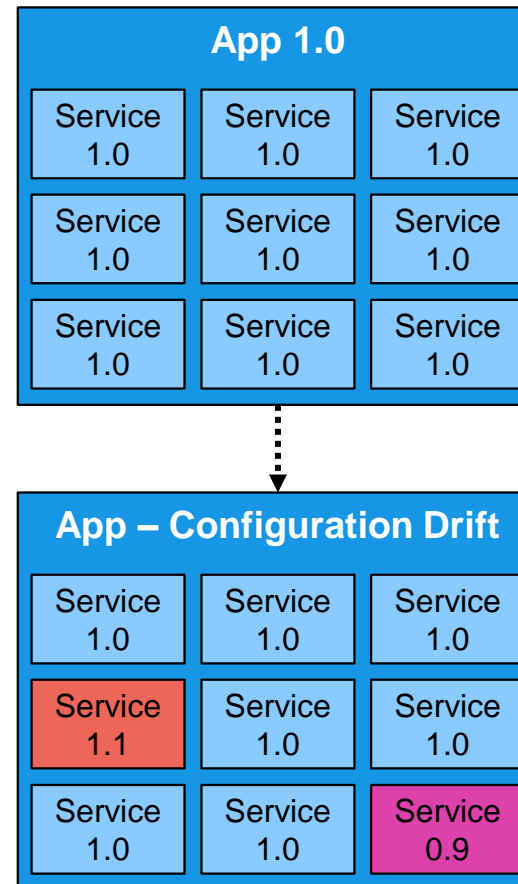
- Wenn Sie Terraform zum Bereitstellen von Computer-Images verwenden, die von Docker oder Packer erstellt wurden, werden "Änderungen" als Bereitstellungen einer völlig neuen App-Version vorgenommen.
- Reduziert die Wahrscheinlichkeit von Konfigurationsdrift-Fehlern, macht es einfacher zu wissen, welche Software auf jedem Server ausgeführt wird.
- Automatisierte Tests sind effektiver, da sich ein unveränderliches Bild, das Ihre Tests in der Testumgebung besteht, wahrscheinlich genau so verhält.
- Blau / Grün



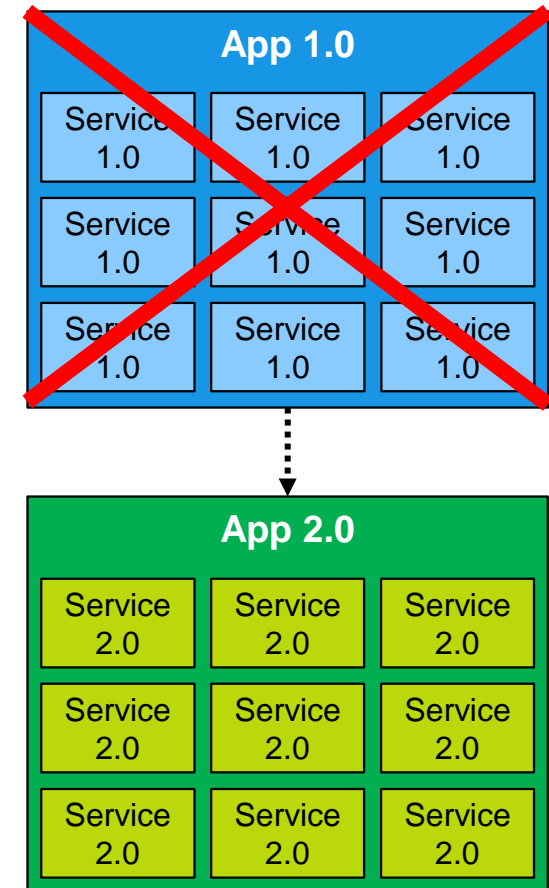
Mutable & Immutable Infrastructure

- Ein Ansatz ist nicht notwendig besser als der andere, es hängt von Ihrem Anwendungsfall.
- Mit dem veränderlichen Ansatz muss sich das Team der Infrastruktur "Geschichte" bewusst sein.
- Im Allgemeinen ist der unveränderliche Ansatz für zustandslose Anwendungen besser.
- Unveränderliche Laufwerke keine Abweichungen und keine Änderungen. Es ist, was es ist.

Mutable



Immutable



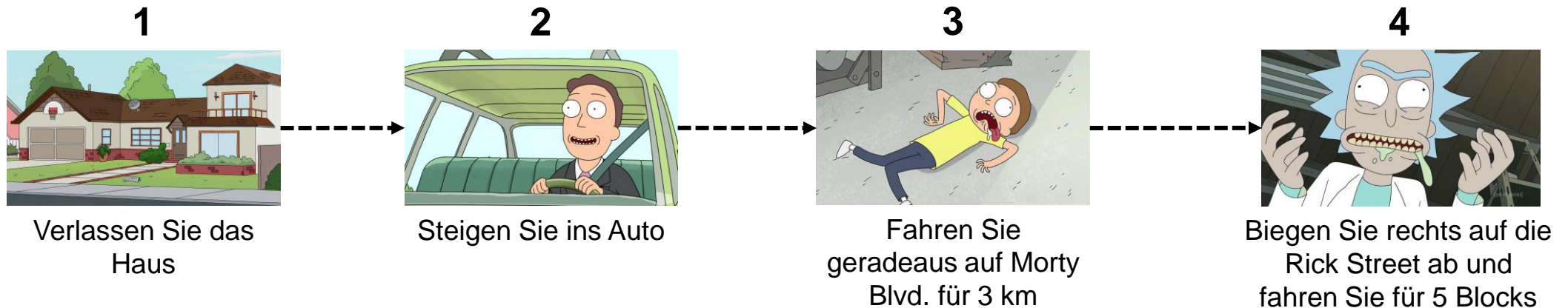
Deployment erster Ressourcen in Azure mit Bicep/ARM

Imperative Code vs. Declarative Code

Imperativ (Verfahren):

Definiert bestimmte Befehle, die in der entsprechenden Reihenfolge ausgeführt werden müssen, um mit der gewünschten Schlussfolgerung zu enden.

AKA “The How”



Imperative Code vs. Declarative Code

Deklarativ (funktional):

Definiert den **gewünschten Zustand**, und das System führt aus, was geschehen muss, um diesen gewünschten Zustand zu erreichen.

AKA “The What”



Meine Adresse ist 9 Rick Street,
Tel-Aviv Israel 4580800

Ich gehe davon aus,
dass Sie ein Navi
haben?!

Ein Wort zum „desired State“...

Example: Ansible Playbook

```
tasks:
- name: Create VM
  azure_rm_virtualmachine:
    name: "MyServer"
    count: "5"
    resource_group: "My_Resource_Group"
    vm_size: "Standard_DS2_v2"
```



```
tasks:
- name: Create VM
  azure_rm_virtualmachine:
    name: "MyServer"
    count: "10"
    resource_group: "My_Resource_Group"
    vm_size: "Standard_DS2_v2"
```



15 Servers



Example: Terraform Plan

```
resource "azurerm_virtual_machine" "terraform" {
  name = "MyServer"
  count = "5"
  resource_group_name = "My_Resource_Group"
  vm_size = "Standard_DS2_v2"
```



```
resource "azurerm_virtual_machine" "terraform" {
  name = "MyServer"
  count = "10"
  resource_group_name = "My_Resource_Group"
  vm_size = "Standard_DS2_v2"
```



10 Servers



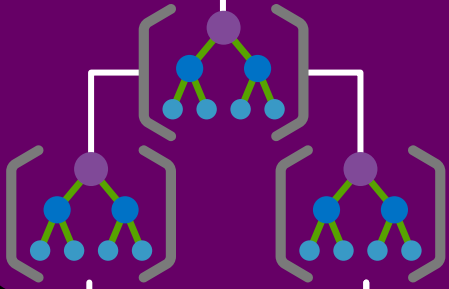
Talk about Scopes



targetScope = 'tenant'

Bicep

Setting schema to use



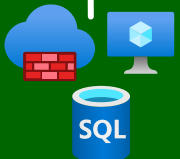
targetScope = 'managementGroup'



targetScope = 'subscription'



targetScope = 'resourceGroup'



Resources



Azure Scopes

ARM Schemas

Bicep

```
targetScope = 'subscription'
```



Bicep 'targetScope' Examples

Tenant – Deploy Management Group



mgmtGroups.bicep X

mgmtGroups.bicep

```
1 targetScope = 'tenant'
2
3 @description('Prefix for the management group hierarchy. This management group will be created as part of the deployment.')
4 @minLength(2)
5 @maxLength(10)
6 param parTopLevelManagementGroupPrefix string = 'alz'
7
8 @description('Display name for top level management group.')
9 @minLength(2)
10 param parTopLevelManagementGroupDisplayName string = 'Azure Landing Zones'
11
12 // Level 1
13 resource resTopLevelMG 'Microsoft.Management/managementGroups@2021-04-01' = {
14   name: parTopLevelManagementGroupPrefix
15   properties: {
16     displayName: parTopLevelManagementGroupDisplayName
17   }
18 }
19
20 output outTopLevelMGId string = resTopLevelMG.id
21
22
23
24
25
26
27
28
29
30
31
```

Bicep 'targetScope' Examples

Management Group – Deploy Management Group with 'scope' Tenant



mgmtGroups.bicep X

mgmtGroups.bicep

```
1 targetScope = 'managementGroup'
2
3 @description('Prefix for the management group hierarchy. This management group will be created as part of the deployment.')
4 @minLength(2)
5 @maxLength(10)
6 param parTopLevelManagementGroupPrefix string = 'alz'
7
8 @description('Display name for top level management group.')
9 @minLength(2)
10 param parTopLevelManagementGroupDisplayName string = 'Azure Landing Zones'
11
12 // Level 1
13 resource resTopLevelMG 'Microsoft.Management/managementGroups@2021-04-01' = {
14   scope: tenant()
15   name: parTopLevelManagementGroupPrefix
16   properties: {
17     displayName: parTopLevelManagementGroupDisplayName
18   }
19 }
20
21 output outTopLevelMGId string = resTopLevelMG.id
22
23
24
25
26
27
28
29
30
31
```

Bicep 'targetScope' Examples

Subscription – Create Resource Group



resourceGroup.bicep X



resourceGroup.bicep

```
1 targetScope = 'subscription'
2
3 @description('Azure Region where Resource Group will be created. No Default')
4 param parResourceGroupLocation string
5
6 @description('Name of Resource Group to be created. No Default')
7 param parResourceGroupName string
8
9 resource resResourceGroup 'Microsoft.Resources/resourceGroups@2021-04-01' = {
10   location: parResourceGroupLocation
11   name: parResourceGroupName
12 }
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```


Bicep 'targetScope' Examples

Resource Group – Create Public IP



publicIP.bicep ×

publicIP.bicep

```
1 @description('Name of Public IP to create in Azure. Default: None')
2 param parPublicIPName string
3
4 @description('Public IP Address SKU. Default: None')
5 param parPublicIPSKU object
6
7 @description('Properties of Public IP to be deployed. Default: None')
8 param parPublicIPProperties object
9
10 @description('Azure Region to deploy Public IP Address to. Default: Current Resource Group')
11 param location string = resourceGroup().location
12
13 @description('Tags to be applied to resource when deployed. Default: None')
14 param parTags object
15
16 resource resPublicIP 'Microsoft.Network/publicIPAddresses@2021-02-01' = {
17   name: parPublicIPName
18   tags: parTags
19   location: location
20   sku: parPublicIPSKU
21   properties: parPublicIPProperties
22 }
23
24 output outPublicIPID string = resPublicIP.id
25
26
27
28
29
30
31
```

Notice that 'targetScope' is not specified?

This is because it defaults to the Resource Group scope if not specified.

But what Tenant, Management Group, Subscription, Resource Group will it be deployed to?



- Great question! This will depend on where you target and start the deployment at of the Bicep file itself.
- **For example:**
 - **New-AzTenantDeployment** -Location 'northeurope' -Name 'demo' -TemplateFile .\main.bicep
 - Targets the tenant of the logged in user/principal
 - **New-AzManagementGroupDeployment** -Location 'northeurope' -Name 'demo' -TemplateFile .\main.bicep -ManagementGroupId 'alz'
 - Targets the 'alz' Management Group (using it's ID/Name – not Display Name)
 - **New-AzSubscriptionDeployment** -Location 'northeurope' -Name 'demo' -TemplateFile .\main.bicep
 - Targets the Subscription currently selected in the Az PowerShell module of the user/principal
 - **New-AzResourceGroupDeployment** -Name 'demo' -TemplateFile .\main.bicep -ResourceGroupName 'rg-demo'
 - Targets the 'rg-demo' Resource Group (using it's Name)



But what Tenant, Management Group, Subscription, Resource Group will it be deployed to?



- Great question! This will depend on where you target and start the deployment at of the Bicep file itself.
- **For example:**
 - **New-AzTenantDeployment** -Location 'northeurope' -Name 'demo' -TemplateFile .\main.bicep
 - Targets the tenant of the logged in user/principal
 - **New-AzManagementGroupDeployment** -Location 'northeurope' -Name 'demo' -TemplateFile .\main.bicep -ManagementGroupId 'alz'
 - Targets the 'alz' Management Group (using it's ID/Name – not Display Name)
 - **New-AzSubscriptionDeployment** -Location 'northeurope' -Name 'demo' -TemplateFile .\main.bicep
 - Targets the Subscription currently selected in the Az PowerShell module of the user/principal
 - **New-AzResourceGroupDeployment** -Name 'demo' -TemplateFile .\main.bicep -ResourceGroupName 'rg-demo'
 - Targets the 'rg-demo' Resource Group (using it's Name)



Bicep Resource Scope Functions



- Used when you need to deploy across multiple scopes within a single deployment
- 1 per Azure Scope: [Tenant](#), [Management Group](#), [Subscription](#), [Resource Group](#)

Azure Scope	Bicep Function	Example
Tenant	tenant()	tenant()
Management Group	managementGroup()	managementGroup(<Management Group Name/ID>)
Subscription	subscription()	subscription(<Subscription ID>)
Resource Group	resourceGroup()	resourceGroup(<RG Name>)
		resourceGroup(<Subscription ID>, <RG Name>)

- All can be used for setting the scope for [modules](#) or [extension resources](#)
- The **Subscription** & **Resource Group** scope functions can be used to also retrieve information about that scope. Like: [Name](#), [ID](#), etc.
- Leave the input parameter section blank on the function to do this e.g: [subscription\(\)](#)



Useful Link: <https://docs.microsoft.com/azure/azure-resource-manager/bicep/bicep-functions-scope>

Bicep 'scope' Keyword on Specific Resources



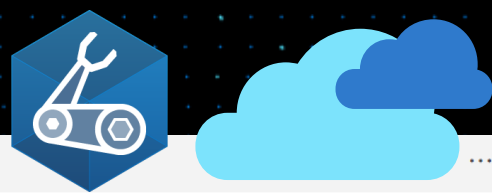
- Only supported on [extension resource types](#)
- Common examples of Extension Resources:

Name	Resource type
Role assignments	Microsoft.Authorization/roleAssignments
Policy assignments	Microsoft.Authorization/policyAssignments
Locks	Microsoft.Authorization/locks
Diagnostic settings	Microsoft.Insights/diagnosticSettings

- Tenant-scoped resources can use the 'scope' keyword so they can be deployed from any template
- An extension resource is a resource that modifies another resource.
- To specify a different scope for a resource type that isn't an extension type you **must** use a **module** and use its associated 'scope' property
- If the scope property on a module is left blank, it will use the scope that the calling bicep file is deployed to e.g. the parents target scope



Bicep Resource Scope Function Examples



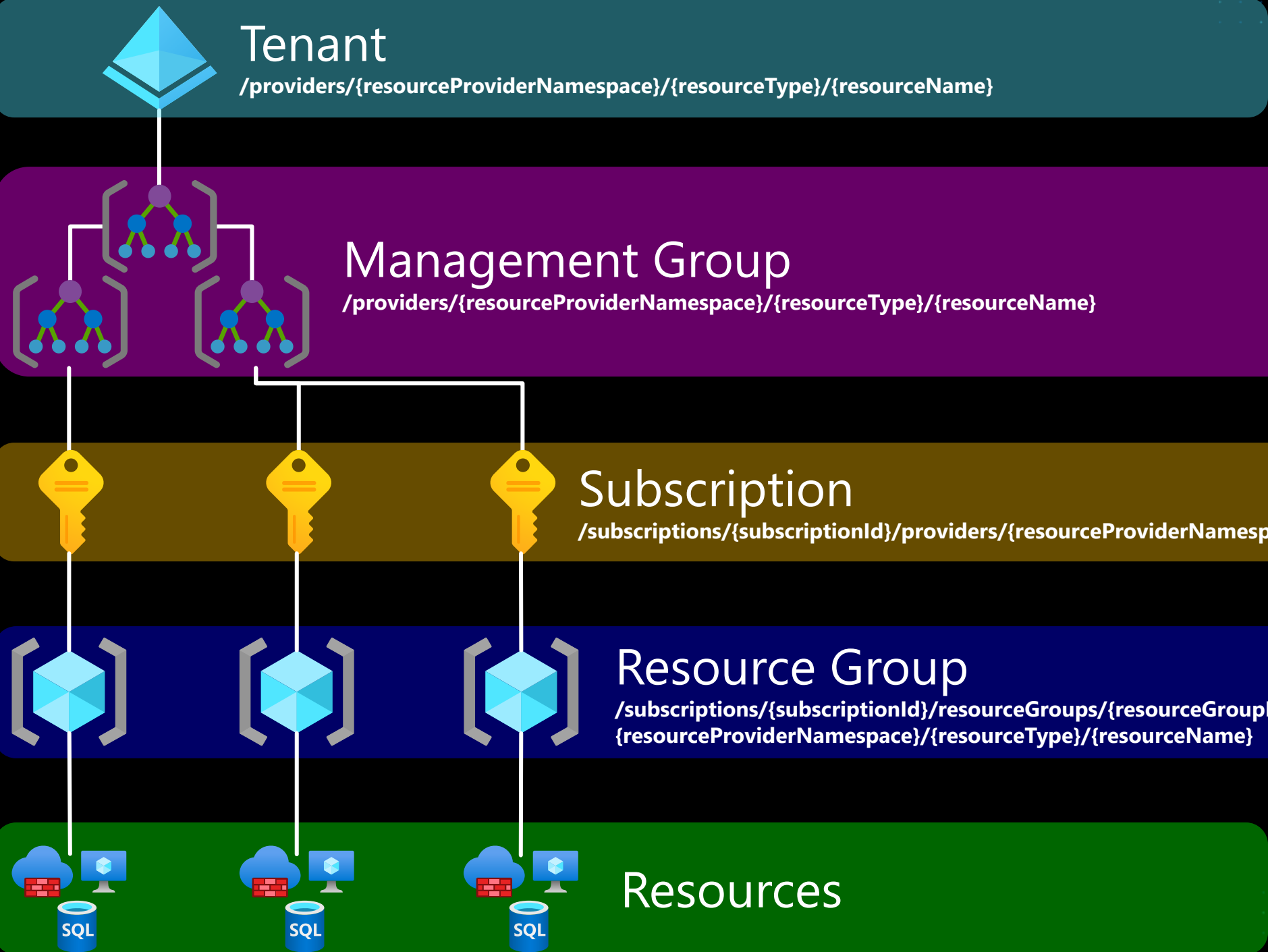
demos.bicep ×

demos.bicep

```
1 // Module Example Snippet - Management Group
2 param parManagementGroupName string
3
4 module mgDeployModule './path-to-module/module.bicep' = {
5   name: 'deployToMG'
6   scope: managementGroup(parManagementGroupName)
7 }
8
9 // Module Example Snippet - Subscription
10 targetScope = 'subscription'
11
12 module networkModule 'modules/hubNetwork.bicep' = {
13   name: 'hubNetworkModule'
14   scope: resourceGroup('rsg-hub-networking')
15 }
16
17 // Module Example Snippet - Resource Group Different Subscription
18 module networkModule 'modules/spokeNetwork.bicep' = {
19   name: 'spokeNetworkModule'
20   scope: resourceGroup('f0750bbe-ea75-4ae5-b24d-a92ca601da2c', 'rsg-spoke-network')
21 }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

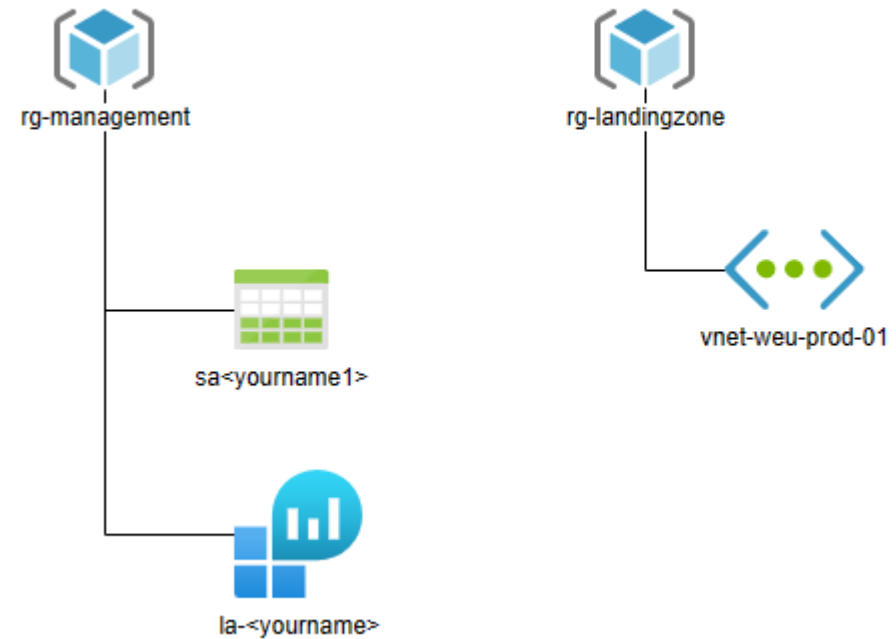
Azure Scopes

With Resource IDs



Challenge 2

- Bereitstellung folgender Struktur mit Hilfe von Bicep Code:



Ausblick auf Terraform

What it is (and what it is not)

- It is....
 - Abstraction layer over common public/private cloud provisioning/automation languages
 - Provides a consistent and unified workflow for provisioning any infrastructure, anywhere. i.e. **Infrastructure-as-code** using a declarative language
- It is NOT....
 - “Write once, deploy anywhere” Infrastructure as Code



How it works

- Hashicorp Configuration Language (or HCL)
 - Stanzas/blocks containing Key/value pairs
 - Interpolation Syntax `${}` is used to reference other resources, call functions, and calculate conditional values
 - Comments can be specified using `#`, `//` or `/* */`
- Leverages Providers
 - [Azure](#), AWS, Google, Kubernetes, Docker, VMware etc.
 - Resources – the things you can create
 - Data Sources – the things you can reference

```
resource "azurerm_resource_group" "testrg" {
  name = "storagergtest"
  location = "australiaeast"
}

resource "azurerm_storage_account" "storageaccount" {
  //storage account name must be globally unique!
  name = "storageacc123"
  resource_group_name = "${azurerm_resource_group.testrg.name}"
  location = "australiaeast"
  account_tier = "standard"
  account_replication_type = "LRS"

  tags = {
    environment = "production"
  }
}
```

How it works (Continued)

- Variables

- Values can be supplied as a .tfvars file containing simple key/value pairs, env variables, or command parameters.

- Functions

- String and math (all the usual)
- Count – simple method for deploying multiple resources
- Conditional "\${var.env == "production" ? var.prod_subnet : var.dev_subnet}"
- CIDR

- Provisioners

- local-exec, remote-exec, file

- Modules

- Reusable template code
- Container of resources that are used together

```
module "vaultstorage" {  
  source = "../modules/storage/account"  
  
  name          = "${var.vault_storage_account_name}"  
  resource_group_name = "${azurerm_resource_group.storage_rg.name}"  
  location      = "${azurerm_resource_group.storage_rg.location}"  
  
  tags {  
    Application = "Vault"  
    Environment = "SS-Prod"  
    Category    = "Storage Account"  
  }  
}
```

How it works (Continued)

- State

- Responsible for mapping *"azure_virtual_machine"* *"vm"* to *"/subscriptions/dcf628c7-fc9d-4e40-af2c-5c963345a237/resourceGroups/BDIterraformdemo/providers/Microsoft.Compute/virtualMachines/BDlvm-vm"*
- Tracks dependencies between resources
 - Knows that if the VM is deleted, to also delete the Disk(s)
- Provides the ability to pass in previous deployments as parameters

How to use it



- Terraform ships as a standalone executable
- Requires Azure CLI
- Code written in .tf files
- Perform deployments with the following commands:
 - `terraform init` – downloads referenced providers locally
 - `terraform plan` – displays a list of all resources that will be added or removed
 - `terraform apply` – deploys resources
 - `terraform destroy` – deletes resources

ARM Template/Terraform Template

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string"
    },
    "accountType": {
      "type": "string"
    },
    "kind": {
      "type": "string"
    }
  },
  "variables": {
    "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
  },
  "resources": [
    {
      "name": "[variables('storageAccountName')]",
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2018-07-01",
      "location": "[parameters('location')]",
      "properties": {
        "accessTier": "[parameters('accessTier')]",
        "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTrafficOnly')]"
      },
      "dependsOn": [],
      "sku": {
        "name": "[parameters('accountType')]"
      },
      "kind": "[parameters('kind')]"
    }
  ],
  "outputs": {}
}
```

Variables

Dependencies

Functions

Resources

```
variable "location" {
  type = "string"
}

variable "accountType" {
  type = "string"
}

variable "tier" {
  type = "string"
}
```

```
resource "random_string" "rnd"{
  length = 8
  special = false
  upper = false
  lower = true
}
```

```
resource "azurerm_resource_group" "testrg" {
  name = "resourceGroupName"
  location = "${var.location}"
}
```

```
resource "azurerm_storage_account" "testsa" {
  name = "${randomstring.rnd.result}storacc"
  resource_group_name = "${azurerm_resource_group.testrg.name}"
  location = "${var.location}"
  account_tier = "${var.tier}"
  account_replication_type = "${var.accountType}"
  enable_https_traffic_only = true
}
```

Why use Terraform?

- Cleaner Code - Easier to use !!
 - Not JSON - Less nesting, less ({{brackets}})
 - No API version specified
 - Supports inline comments!
- Implicit dependencies vs arm - Dependencies must be explicitly defined using nesting, or DependsOn property
- One language and Cross Platform – for customers using a multi cloud strategy.

Providers

Defines how
Terraform will interact
with:

Cloud

Azure

AWS

Google

AliCloud

OpenStack

Infrastructure

Kubernetes

Docker

Rancher

Network

DNS

Cloudflare

HTTP

Version Control

GitHub

GitLab

Bitbucket

Monitoring

DataDog

Grafana

PagerDuty

Database

InfluxDB

MySQL

PostgreSQL

Etc.

Azure Provider

Authentication

Azure CLI

Service Principal

MSI

Arguments

```
provider "azurerm" {  
  subscription_id = "{My Subscription ID}"  
  client_id       = "{My Service Principle ID}"  
  client_secret   = "{My Service Principle Password}"  
  tenant_id      = "{My Tenant ID}"  
}
```

Environment Variables

Azure Resources & Datasources

```
Configure the Azure Provider
provider "azurerm" { }

# Create a resource group
resource "azurerm_resource_group" "network" {
  name     = "production"
  location = "West US"
}

# Create a virtual network within the resource group
resource "azurerm_virtual_network" "network" {
  name                = "production-network"
  address_space       = ["10.0.0.0/16"]
  location             = "${azurerm_resource_group.network.location}"
  resource_group_name = "${azurerm_resource_group.network.name}"

  subnet {
    name           = "subnet1"
    address_prefix = "10.0.1.0/24"
  }

  subnet {
    name           = "subnet2"
    address_prefix = "10.0.2.0/24"
  }

  subnet {
    name           = "subnet3"
    address_prefix = "10.0.3.0/24"
  }
}
```

Provisioning for Azure IaaS

Compute (VMSS, Disk, Image, Snapshot, ...)

Networking (Vnet, LB, DNS, ...)

Azure Active Directory

Database (MySQL, PostgreSQL, SQL)

Monitoring

Storage (Storage Account, Blob, Share, ...)

...

Provisioning for Azure PaaS

Containers (AKS, ACI)

Web Apps

CosmosDB

Data Lake






Logic Apps

KeyVault

...

Catch-All
ARM Template

The Major (cross-platform) Players

Tool	Tool Type	Infrastructure	Architecture	Approach	Manifest Written Language
 puppet	Configuration Management	Mutable	Pull	Declarative	Domain Specific Language (DSL) & Embedded Ruby (ERB)
 CHEF	Configuration Management	Mutable	Pull	Declarative & Imperative	Ruby
 ANSIBLE	Configuration Management	Mutable	Push	Declarative & Imperative	YAML
 SALTSTACK	Configuration Management	Mutable	Push & Pull	Declarative & Imperative	YAML
 Terraform	Provisioning	Immutable	Push	Declarative	HashiCorp Configuration Language (HCL)



Terraform vs. All The REST

- Cloud-Agnostic (well, kinda...)
- The “*terraform plan*” command
- State
- Terraform Modules
- HashiCorp Configuration Language (HCL)
- There is more...

As part of the vBrownbag “API Zero to Hero” series, Byron Schaller & myself will run the “Exploring Terraform APIs” session next month.



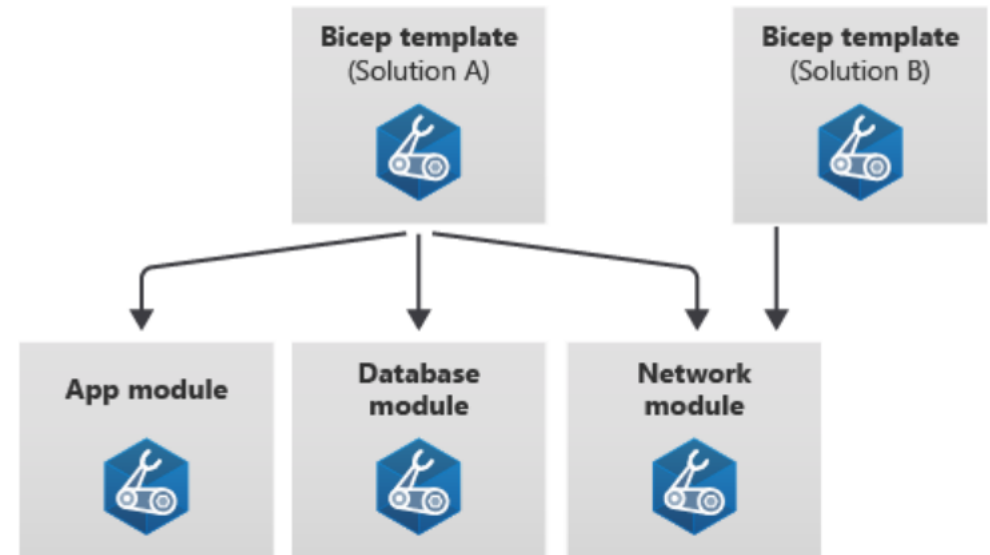
Umgang mit Modulen in Bicep

BICEP – Modules (reviewed)

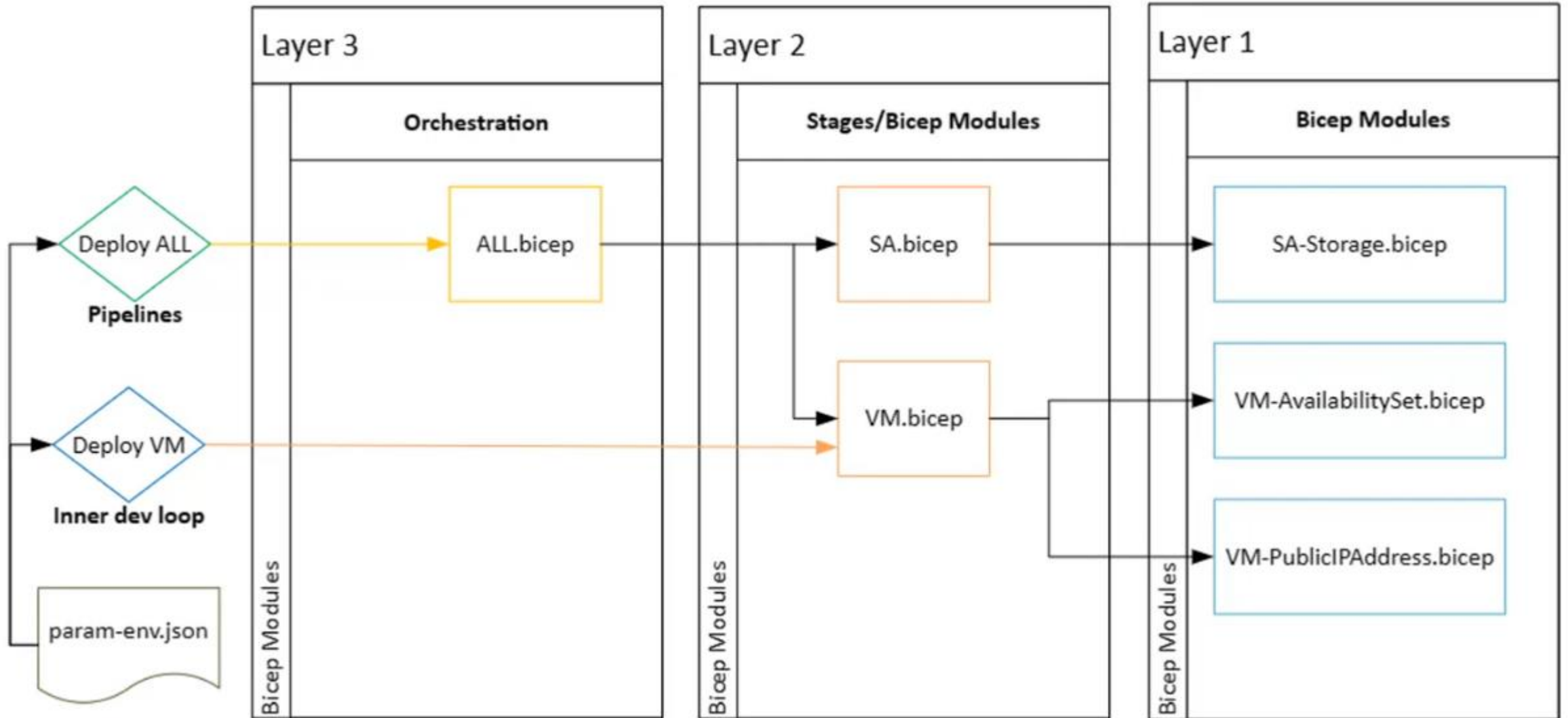
The IT manager can see that your Bicep code is becoming more complex and has an increasing number of resources defined, so they've asked if you can make the code more **modularized**.

You can create individual Bicep files, called modules, for different parts of your deployment. The main Bicep template can reference these modules. Behind the scenes, modules are transpiled into a single JSON template for deployment. Imagine you have a Bicep template that deploys application, database, and networking resources for **solution A**. You might split up this template into three modules, each of which is focused on its own set of resources.

As a bonus, you can now reuse the modules in other templates for other solutions too. So when you develop a template for **solution B**, which has similar networking requirements to **solution A**, you can reuse the networking module.



Building Out Deployment Layers



BICEP – Referencing Module

```
module myModule 'modules/my-module.bicep' = {  
  name: 'MyModule'  
  params: {  
    location: location  
  }  
}
```

Existing Resource reference- BICEP

```
resource keyVault 'Microsoft.KeyVault/vaults@2021-04-01-preview' existing = {  
  name: keyVaultName  
}
```

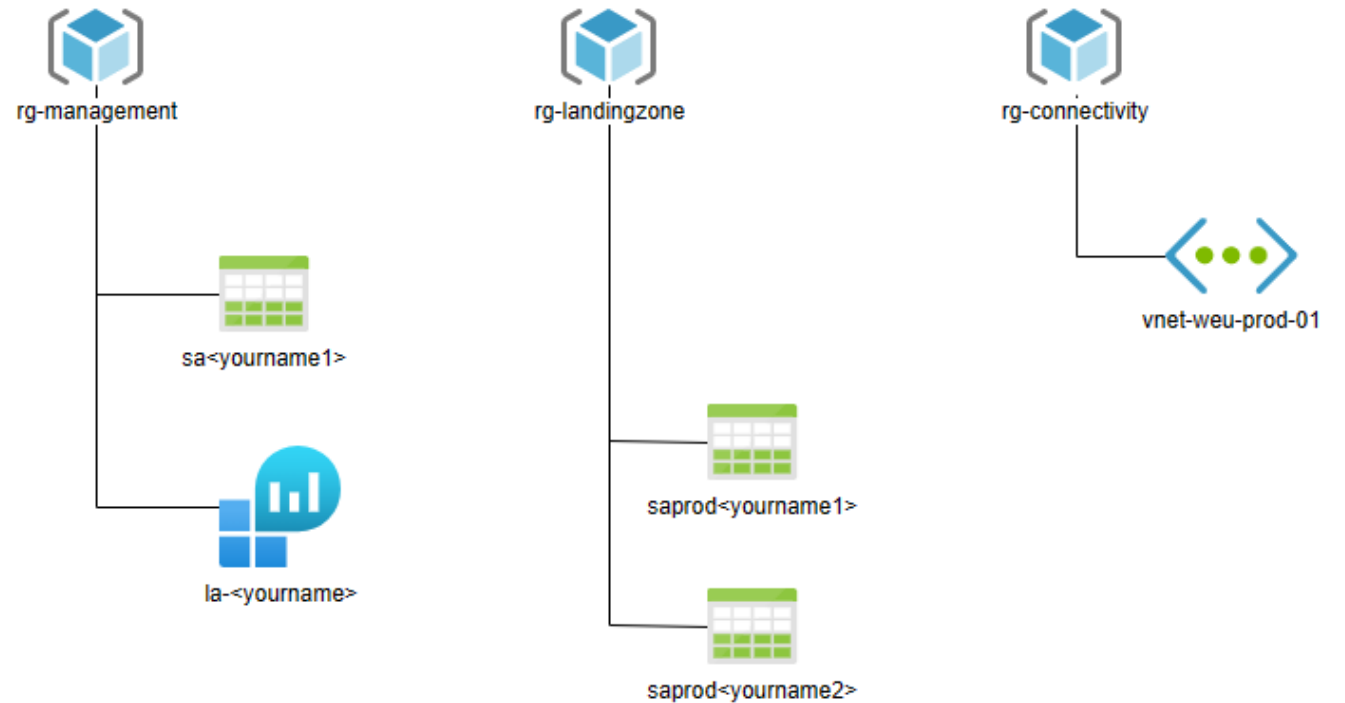
```
module applicationModule 'application.bicep' = {  
  name: 'application-module'  
  params: {  
    apiKey: keyVault.getSecret('ApiKey')  
  }  
}
```

Creating Multiple Resources- BICEP

```
param storageAccountNames array = [  
    'saauditus'  
    'saauditeurope'  
    'saauditapac'  
]  
  
resource storageAccountResources 'Microsoft.Storage/storageAccounts@2021-01-01' = [for  
storageAccountName in storageAccountNames: {  
    name: storageAccountName  
    location: resourceGroup().location  
    kind: 'StorageV2'  
    sku: {  
        name: 'Standard_LRS'  
    }  
}]
```

Challenge 3

- Bereitstellung folgender Struktur mit Hilfe von Modulen* im Bicep Code:



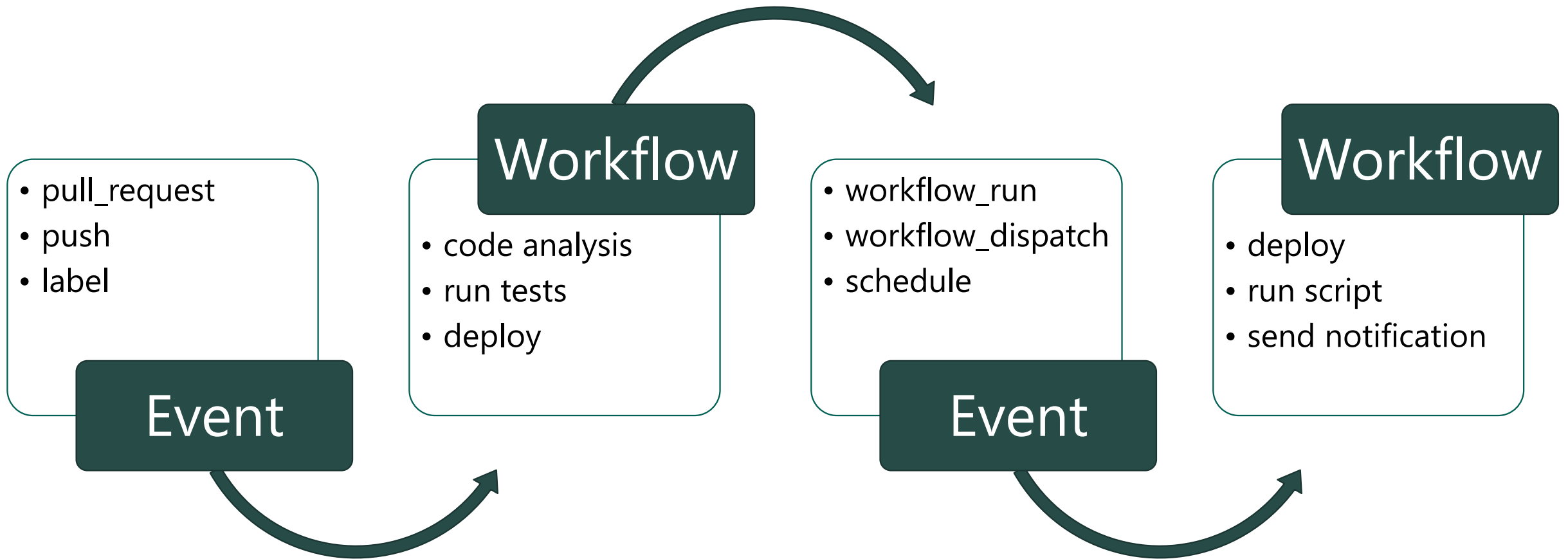
*die RGs und die SAs per Modul

Aufbau einer Pipeline zum automatischen Bereitstellen von Azure Ressourcen

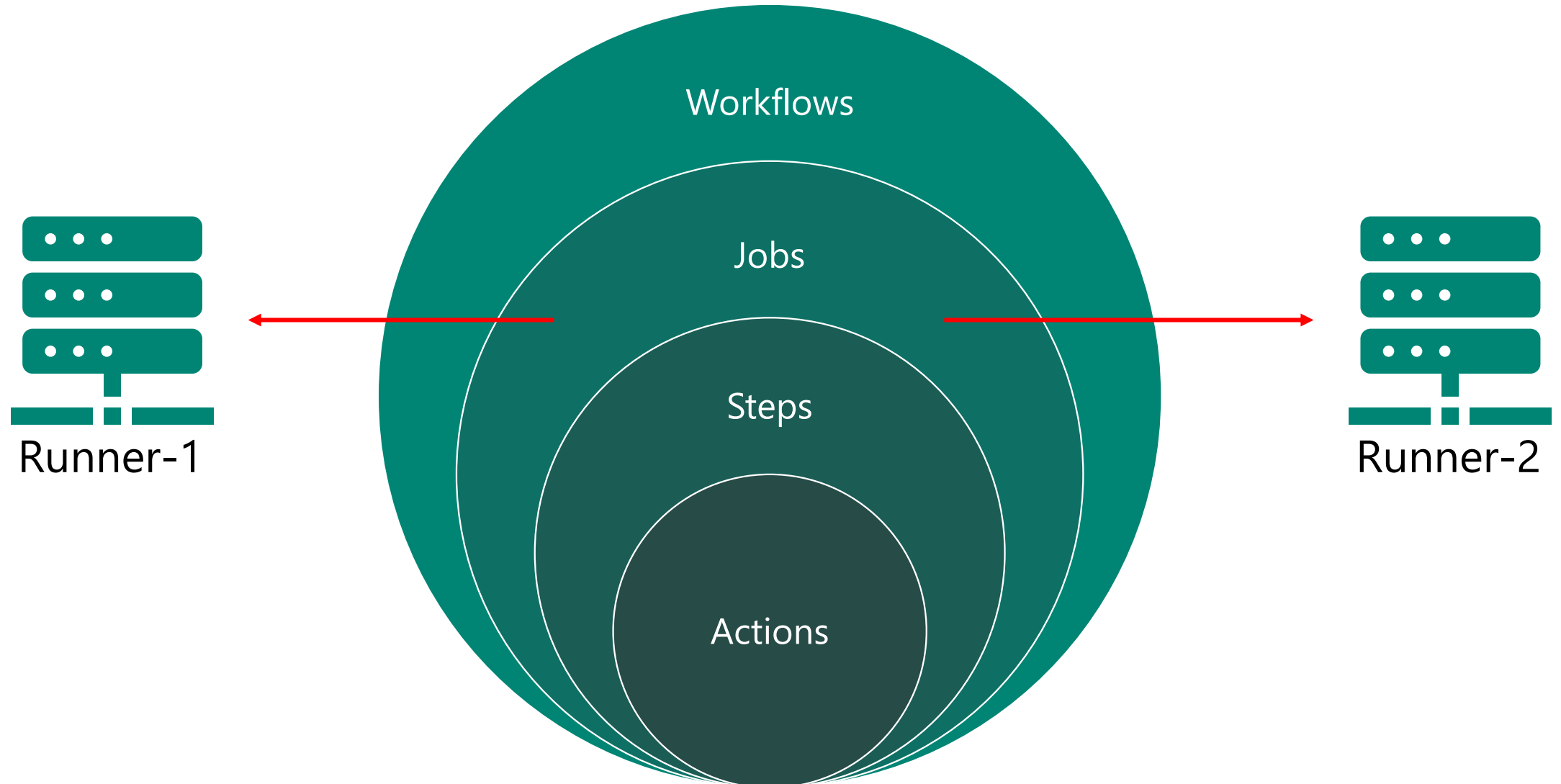
What are GitHub Actions

- Workflow Automation and CI/CD Engine
- YAML based
- Built on Azure Pipeline codebase
- First Release Nov. 2019

The Theory

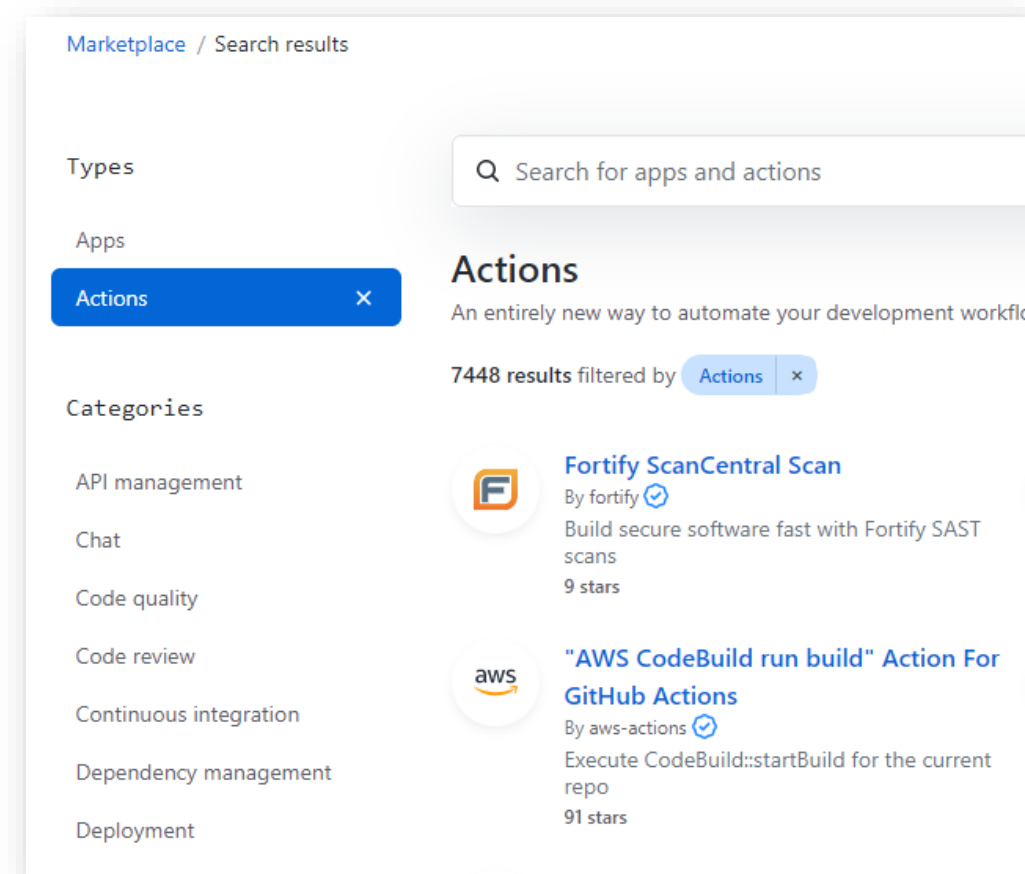


Workflow Structure



What is an “Action”

- Small piece of code to automate your DevOps activities, like:
 - Build and Test Projects
 - Deploy packages to different environments
 - **Execute your custom logic**
 - And more ...
- 7K+ actions available on [GitHub Marketplace](#)
- Create your own Actions



Anatomy of a GitHub Workflow

Your yaml file is called a *workflow*

```
ci.yml

1  on: push
2  jobs:
3    test:
4      strategy:
5        matrix:
6          platform: [ubuntu-latest, macos-latest, windows-latest]
7      runs-on: ${ matrix.platform }
8      steps:
9        - uses: actions/checkout@v1
10       - uses: actions/setup-node@v1
11         with:
12           version: 12
14       - run: npm install-ci-test
15       - uses:
```

Deploy to Azure

Automate your workflows using GitHub Actions for Azure

 Azure/actions

Amazon ECR Login

Logs in the local Docker client to Amazon ECR

 aws-actions/amazon-ecr-login

Deploy to Kubernetes

Deploy your app on any Kubernetes cluster

 Azure/k8s-actions

Code Climate Velocity

In-depth code metrics to speed up your engineering processes

 codeclimate/velocity-coverage

Label a pull request

Label pull requests based on changed files

 actions/labeler

Google Cloud Platform

A collection of GitHub Actions for Google Cloud Platform

 GoogleCloudPlatform/github-actions

Glo Boards

Integrate your Glo boards into your code workflow

 Axosoft/glo-actions

CircleCI

Integrate your CircleCI workflows into your code workflow



```
27  publish:
28    needs: [build]
```

Anatomy of a GitHub Workflow

The *event* triggering your workflow

The *different* jobs in your workflow

The individual *steps* of your jobs

```
ci.yml
1  on: push
2  jobs:
3    test:
4      strategy:
5        matrix:
6          platform: [ubuntu-latest, macos-latest, windows-latest]
7      runs-on: ${{ matrix.platform }}
8      steps:
9        - uses: actions/checkout@v1
10       - uses: actions/setup-node@v1
11         with:
12           version: 12
14       - run: npm install-ci-test
15       - uses:
```

Deploy to Azure

Automate your workflows using GitHub Actions for Azure



Azure/actions

Amazon ECR Login

Logs in the local Docker client to Amazon ECR



aws-actions/amazon-ecr-login

Deploy to Kubernetes

Deploy your app on any Kubernetes cluster



Azure/k8s-actions

Code Climate Velocity

In-depth code metrics to speed up your engineering processes

codeclimate/velocity-ci

Label a pull request

Label pull requests based on changed files



actions/labeler

Google Cloud Platform

A collection of GitHub Actions for Google Cloud Platform



GoogleCloudPlatform/github-actions

Glo Boards

Integrate your Glo boards into your code workflow



Axosoft/glo-actions

C

In
qu



```
27  publish:
28    needs: [build]
```

Challenge 4

- Der ganze bisherige Code sollte in eine GitHub Repo gebracht werden
- Ein Security Principal für das Deployment über GitHub Actions angelegt und hinterlegt werden
- Ein Github Workflow zum Deployment des Bicep Codes erstellen
- Per diesem Workflow alle Ressourcen mit einem Tag versehen

Deployment = Auto GitHub
Environment = Dev

Intro ESLZ

Feedback



Challenge 5 (optional)

Useful Link:

<https://aka.ms/LearnBicepScopes>

Deploy the following resources from a **single** Bicep deployment:

- 1 Management Group (intermediate root – aka beneath tenant root group)
 - With 1 child Management Group
- Assign the following built-in policies to the intermediate root Management Group: 'Allowed locations' & 'Allowed locations for resource groups'
- Assign the RBAC built-in Reader role to the child Management Group
- Move your Subscription into the child Management Group
- Deploy a VNET into the Subscription

Stretch Goals:

- Deploy 2 VNETs and peer them together, in either:
 - Seperate Subscriptions (if available) or seperate Resource Groups
- Make the child Management Group an array with a loop (so you can have multiple child MGs)
- Try using 'scope' for extension resource types