Introduction to Bicep for the Cloud DBA

Frank Geisler

Managing Director, Founder and Owner

Frank Geisler MSc. IT

Frank Geisler has been self-employed since 1997, Microsoft Data Platform MVP since 2014 and holds numerous Microsoft certifications

frank_geisler@geislers.net

Activities

- Microsoft Data Platform MVP
- Microsoft P-TSP
- Board Member of PASS Deutschland e.V.
- Founding Member of PASS Deutschland e.V.,
- Chapter Lead PASS RG Münsterland
- Author
- Speaker

Agenda

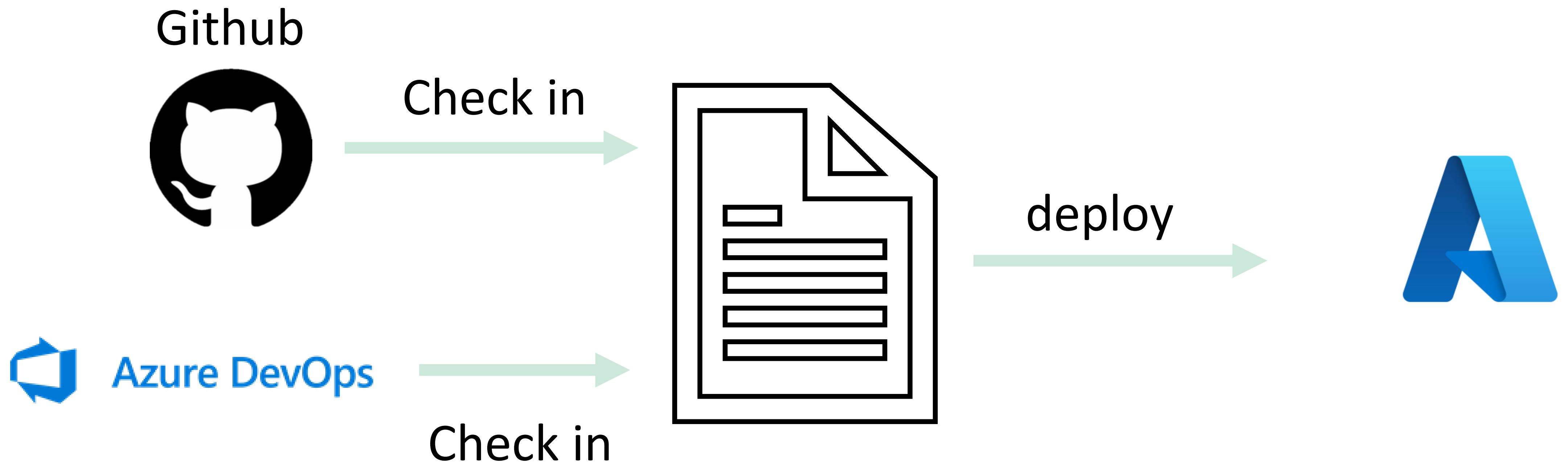
- Infrastucture as Code
- What is Bicep
- Elements of Bicep
- Advanced Concepts
- Demos

Infrastructure as Code

What is Infrastructure as Code

- Process of automating infrastucture provisioning
- Uses descriptive language not imperative
- Uses versioning system
- Generates always the same result
- Infrastructure as part of your deployment process

The IAC Process



Why Infrastructure as Code?

- Increase confidence in deployments
- Manage multiple environments
- Better understand your Cloud Environment

Imperative VS Declarative

Imperative

- Bash / Azure PowerShell
- Script has Create Commands
- Disadvantages: Scripts become complex to manage
- Commands may be deprecated/updated → review of scripts

declerative

- Json
- Bicep
- Ansible
- Terraform

Imperative VS Declarative

Imperative

```
#!/usr/bin/env bash
az group create \
        --name storage-resource-group \
        --location eastus

az storage account create \
        --name mystorageaccount \
        --resource-group storage-resource-group \
        --kind StorageV2 \
        --access-tier Hot \
        --https-only true
```

declerative

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = {
        name: 'mystorageaccount‘
        location: 'eastus‘
        sku: {
                name: 'Standard_LRS‘
        }
        kind: 'StorageV2‘
        properties: {
                accessTier: 'hot‘
                supportsHttpsTrafficOnly: true
        }
}
```

What is Bicep?

What is Bicep?

- Azure Resource Manager Template Language
- Declaratively deploy Azure Resources
- Domain specific language
- Intended to be easy to understand
- Simplified Json Notation

Benefit of Bicep

- Simpler Syntax than Json
- Modules: Break down large scripts in small chunks
- Automatic dependency management
- Type validation and Intellisense
  (Bicep Extension for Visual Studio Code)

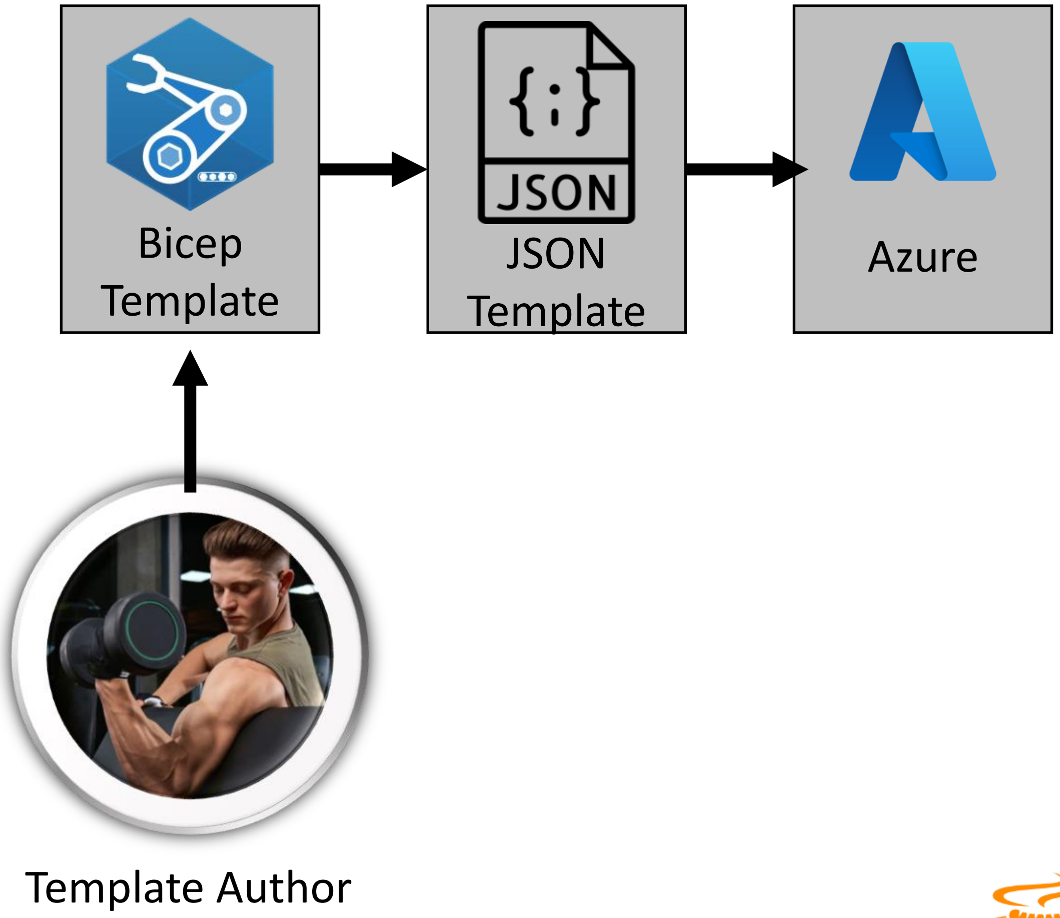
BICEP EXAMPLE

```
param location string = resourceGroup().location
param namePrefix string = 'storage‘

var storageAccountName = '${namePrefix}${uniqueString(resourceGroup().id)}‘
var storageAccountSku = 'Standard_RAGRS‘

resource storageAccount 'Microsoft.Storage/storageAccounts@2019-06-01' = {
		name: storageAccountName
		location: location
		kind: 'StorageV2‘
		sku: {
				name: storageAccountSku
		}
		properties: {
				accessTier: 'Hot‘
				supportsHttpsTrafficOnly: true
		}
}

output storageAccountId string = storageAccount.id
```

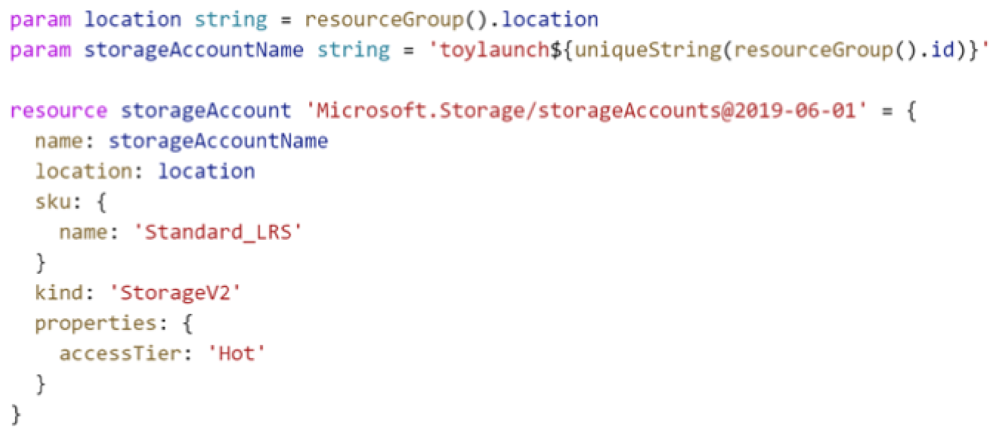
HOW BICEP WoRKS

- Tooling of bicep transforms bicep script to Json Script (transpiling)
- Azure Resource Manager only understands Json

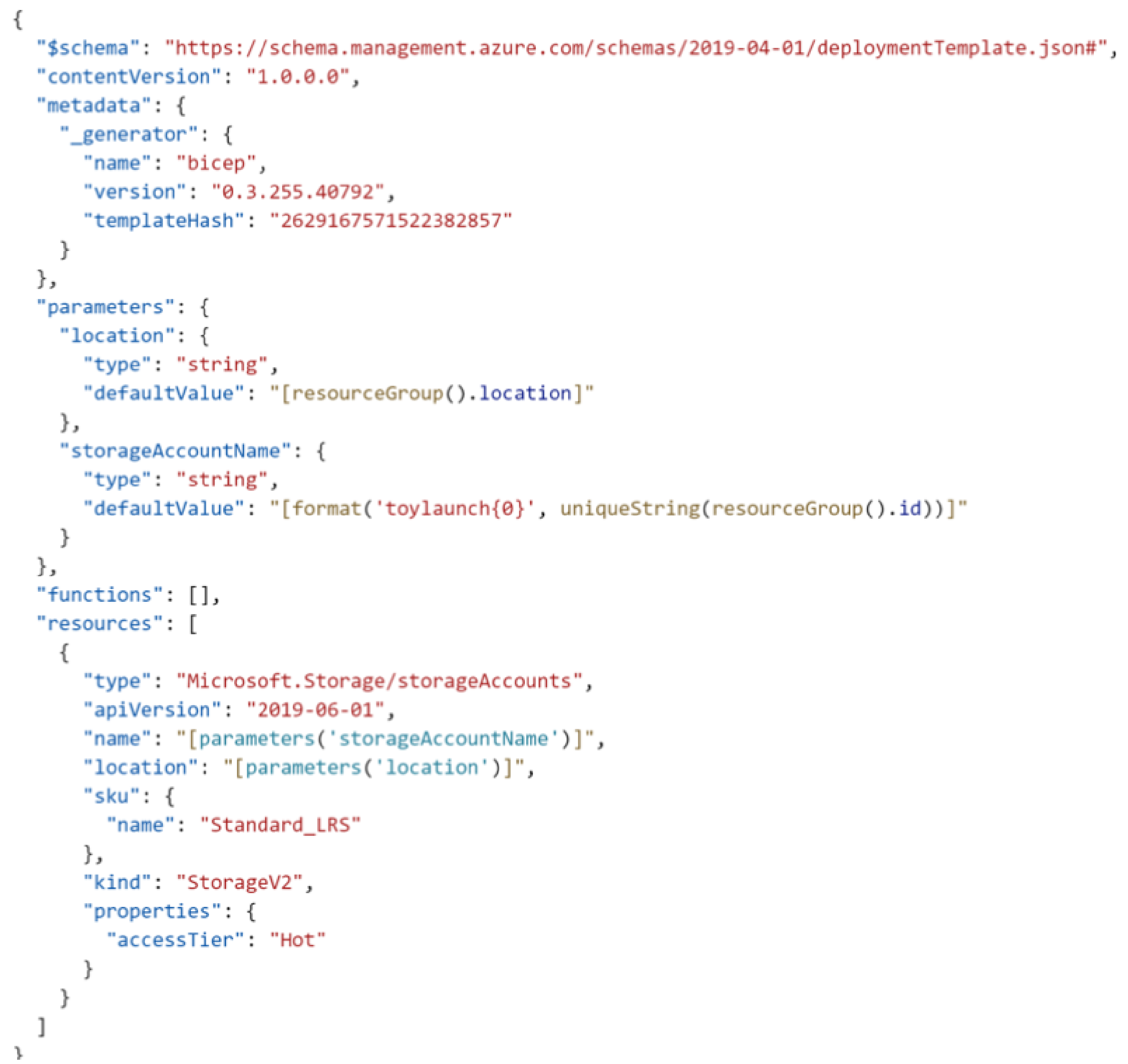
```
az deployment group create \
        --template-file ./main.bicep \
        --resource-group storage-resource-group
```



Comparing bicep and JSON



```
param location string = resourceGroup().location
param storageAccountName string = 'toylaunch${uniqueString(resourceGroup().id)}'

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



```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.3.255.40792",
      "templateHash": "2629167571522382857"
    }
  },
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]"
    },
    "storageAccountName": {
      "type": "string",
      "defaultValue": "[format('toylaunch{0}', uniqueString(resourceGroup().id))]"
    }
  },
  "functions": [],
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2019-06-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {
        "accessTier": "Hot"
      }
    }
  ]
}
```

When is BICEP the Right Tool

- Azure native
- Azure integration
- Azure support
- No state management
- Easy transition from Json

When is BICEP **NOT** the Right Tool

- Multicloud
- Existing Toolset

Elements of Bicep

Elements of BICEP

- Resource
- Parameters
- Variables
- Modules
- Outputs

Example of Resource

Ressource Name in Bicep Template
(symbolic name)

Ressource Type

Properties

```
resource appServicePlan 'Microsoft.Web/serverFarms@2021-03-01' = {
    name: 'myAppServicePlan'
    location: 'westus3'
    sku: {
      name: 'F1'
    }
}
```

Resource using other Resource

Ressource Type

```
resource appServiceApp 'Microsoft.Web/sites@2021-03-01' =
{
    name: 'MyWebsite'
    location: 'westus3'
    properties: {
      serverFarmId: appServicePlan.id
      httpsOnly: true
    }
  }
```

Property of resource

Reference to resource name

Parameters

- Bring values from outside
- Good use for things that change between deployments
  - Names of Resources
  - Locations
  - Settings e.g. Price settings
- Think about naming convention!
- Parameters can come from file

Parameter Name

Default value

Datatype

```
param appServiceAppName string = 'sqlbitsapp'
```

Using Parameter in Template

Parameter

```
resource appServiceApp 'Microsoft.Web/sites@2021-03-01' = {
    name: appServiceAppName
    location: 'westus3‘
    properties: {
      serverFarmId: appServicePlan.id
      httpsOnly: true
    }
  }
```

Allowed Values

- You can specify which values are allowed for a parameter

```
@allowed([
    'westeurope'
    'uksouth'
    'moon'
])

param location string
```

Variables

- Don’t change value from outside but reuse the value within template
- Using variables to hold values from complex expressions
- A variable must have a value
- Variables are not strongly typed

Variable Name

Value

```
var appServicePlanName = 'myAppServicePlan'
```

Conditional Values

- You can use if while assigning values for variables

```
var comment = (location == 'Moon') ? 'not a proper location‘ : 'Azure
location'
```

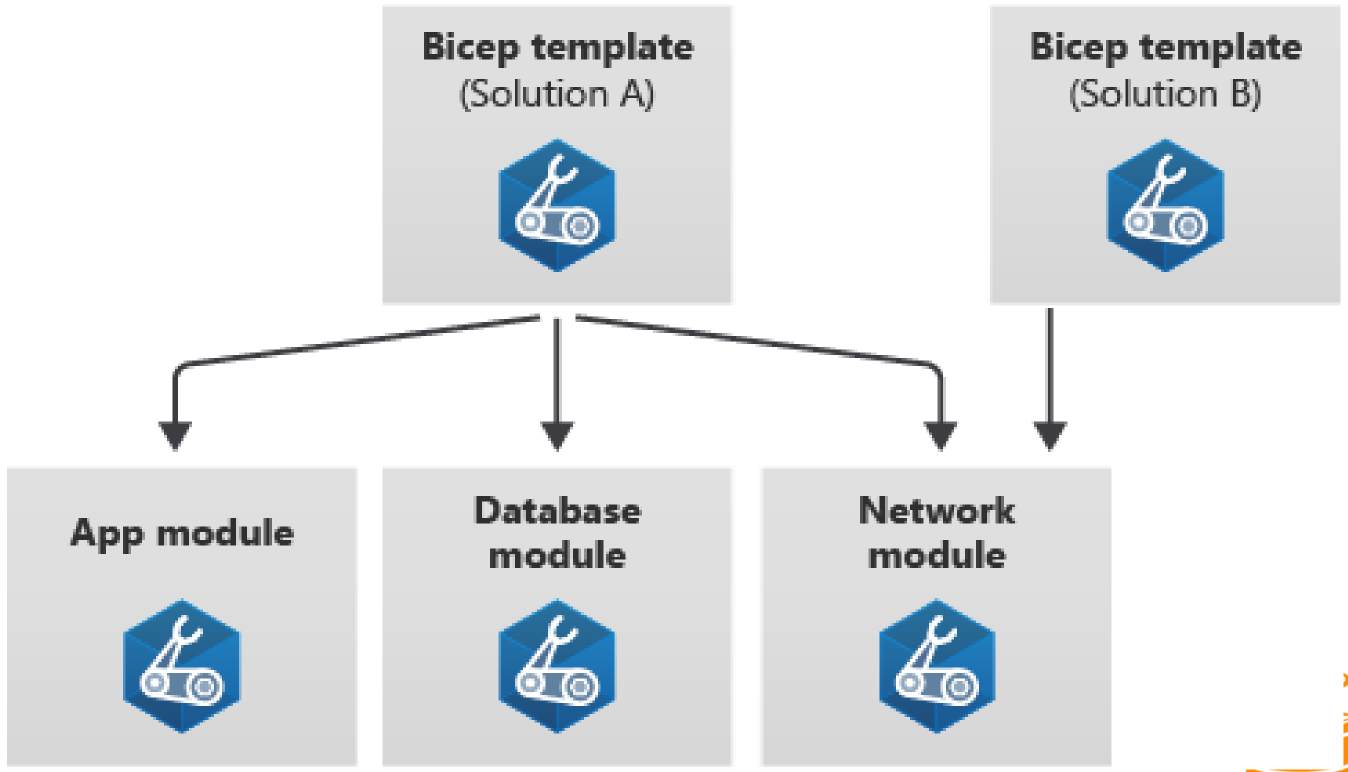
Comparison

„true“ part

„false“ part

Modules

- Use Modules to organize and reuse bicep
- Create smaller units



Using Modules

Use another bicep file as module in this file

Relative Filepathname

Parameters passed to module

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

Outputs

- create outputs for information the parent module might need

- WARNING:
  - Do not use outputs for secret values
  - Outputs are logged

- parent template can use module outputs
  - in variables
  - use properties for other resource definitions
  - expose variables and properties as outputs itself
- Exposing outputs can lead to reusable set of Bicep modules that can shared with team- Good practice: Meaningful description for output

```
@description('The fully qualified Azure resource ID of the blob container within the storage account.')
output blobContainerResourceId string = storageAccount::blobService::container.id
```

Some Tips

- Location from resource can be retrieved (to e.g. put other resources in the same location)

```
param location string = resourceGroup().location
```

- Uniqueness of resource names – you can generate an unique resource name with uniqueString()

```
param storageAccountName string = uniqueString(resourceGroup().id)
```

STRING Interpolation

- You can use String Interpolation within Bicep

```
param storageAccountName string = 'sqlbits${uniqueString(resourceGroup().id)}'
```

Advanced Concepts

Conditional Deployment

- Deploy Resources only in certain scenarios

Only deploy if deployStorageAccount is true

```
param deployStorageAccount bool

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = if (deployStorageAccount) {
name: 'teddybearstorage‘
location: resourceGroup().location
kind: 'StorageV2' // ...
}
```

Use expressions as conditions

```
@allowed([
    'Development‘
    'Production‘
])

param environmentName string

resource auditingSettings 'Microsoft.Sql/servers/auditingSettings@2021-11-01-
preview' = if (environmentName == 'Production‘)
{ parent: server
  name: 'default‘
  properties: { } }
```

Loops

- Use for Keyword to create loop
- for Keyword must be placed in resource declaration
- specify how to identify each item
- loop is over array of objects --> create multiple instances of resource

Loop on an Array

```
param storageAccountNames array = [
        'saauditus‘
        'saauditeurope‘
        'saauditapac‘
]
resource storageAccountResources 'Microsoft.Storage/storageAccounts@2021-09-01' = [for
storageAccountName in storageAccountNames: {
            name: storageAccountName
            location: resourceGroup().location
            kind: 'StorageV2‘
            sku: {
                    name: 'Standard_LRS‘
                }
}]
```

Loop on a range

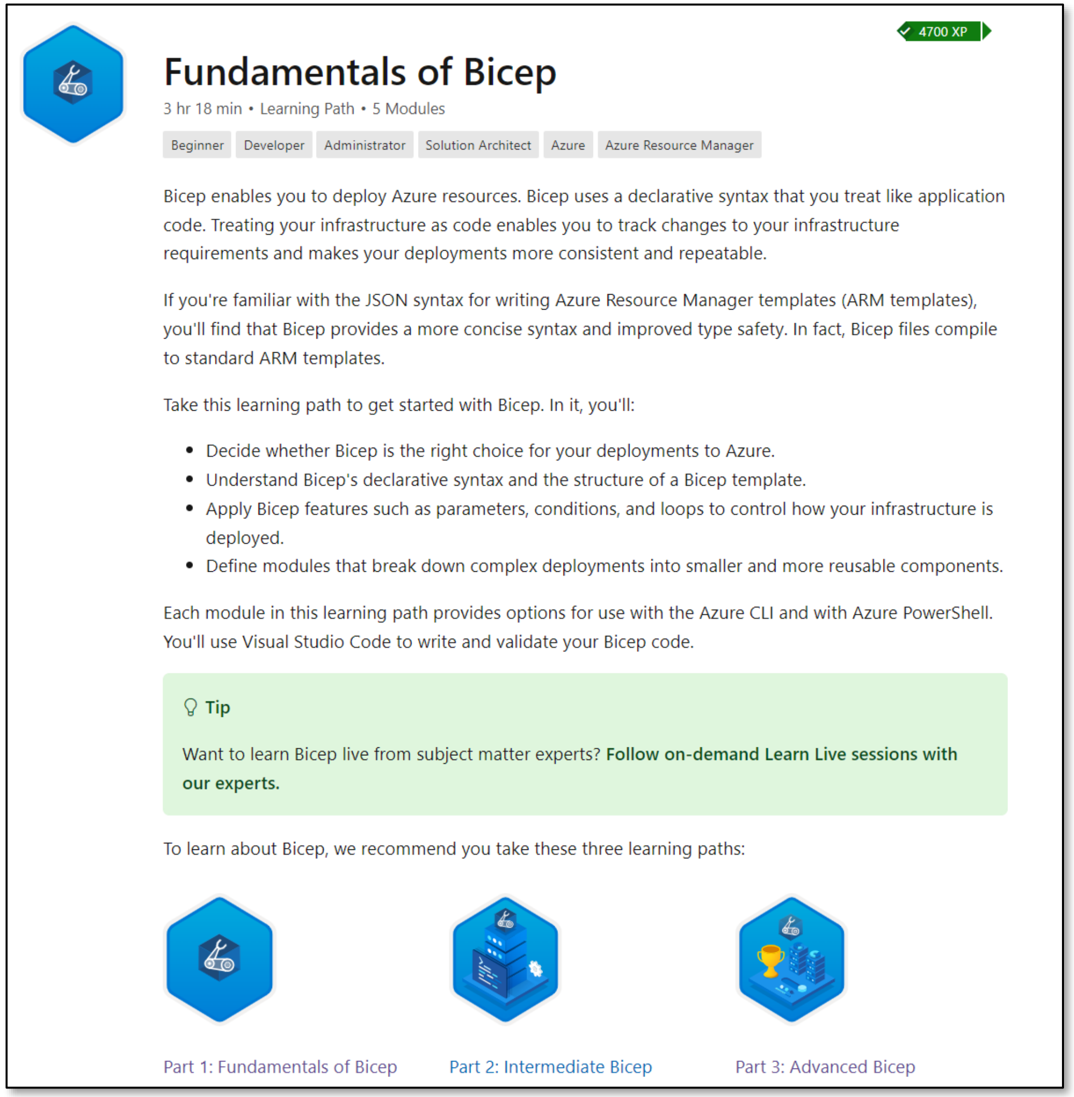
```
resource storageAccountResources
 'Microsoft.Storage/storageAccounts@2021-09-01' = [for i in range(1,4): {
          name: 'sa${i}‘
          location: resourceGroup().location
        kind: 'StorageV2‘
        sku: {
                    name: 'Standard_LRS‘
                }
}]
```

Structure of a bicep file (best practice)

```
// Target Scope for bicep file
targetScope = 'resourceGroup'

//============================================================================
// Parameters
//============================================================================
@description('Azure Region for the resources')
param location string = resourceGroup().location

@description('Environment Type - is set as tag on all resources')
@allowed([
'Development'
'Production'
'Test'
])
param environmentType string = 'Development'

//============================================================================
// Variables
//============================================================================

@description('This variable is used to construct the abbreviation for the environment.')
var environment = environmentType == 'Development' ? 'dev' : environmentType == 'Production' ? 'prd' : 'tst'

//============================================================================
// Resources
//============================================================================

//============================================================================
// Outputs
//============================================================================
```

Demos

Where can I lean more?


4700 XP

Fundamentals of Bicep

3 hr 18 min • Learning Path • 5 Modules

Beginner Developer Administrator Solution Architect Azure Azure Resource Manager

Bicep enables you to deploy Azure resources. Bicep uses a declarative syntax that you treat like application code. Treating your infrastructure as code enables you to track changes to your infrastructure requirements and makes your deployments more consistent and repeatable.

If you're familiar with the JSON syntax for writing Azure Resource Manager templates (ARM templates), you'll find that Bicep provides a more concise syntax and improved type safety. In fact, Bicep files compile to standard ARM templates.

Take this learning path to get started with Bicep. In it, you'll:

- Decide whether Bicep is the right choice for your deployments to Azure.
- Understand Bicep's declarative syntax and the structure of a Bicep template.
- Apply Bicep features such as parameters, conditions, and loops to control how your infrastructure is deployed.
- Define modules that break down complex deployments into smaller and more reusable components.

Each module in this learning path provides options for use with the Azure CLI and with Azure PowerShell. You'll use Visual Studio Code to write and validate your Bicep code.

Tip

Want to learn Bicep live from subject matter experts? **Follow on-demand Learn Live sessions with our experts.**

To learn about Bicep, we recommend you take these three learning paths:

Part 1: Fundamentals of Bicep Part 2: Intermediate Bicep Part 3: Advanced Bicep

[Fundamentals of Bicep - Training | Microsoft Learn]

Frank Geisler
frank_geisler@geislers.net
@FrankGeisler