DATA ANZ
AUSTRALIA

INTRODUCTION TO BICEP

Martin Cairney

Microsoft Data Platform MVP

Managing Consultant, Telstra Purple

SPONSORS

Telstra Purple	Sirius Technology	Databricks
https://purple.telstra.com	https://siriuspeople.com.au	https://databricks.com

CODE OF CONDUCT

Data ANZ seeks to provide a respectful, friendly professional experience for everyone, regardless of gender, sexual orientation, physical appearance, disability, age, race or religion.

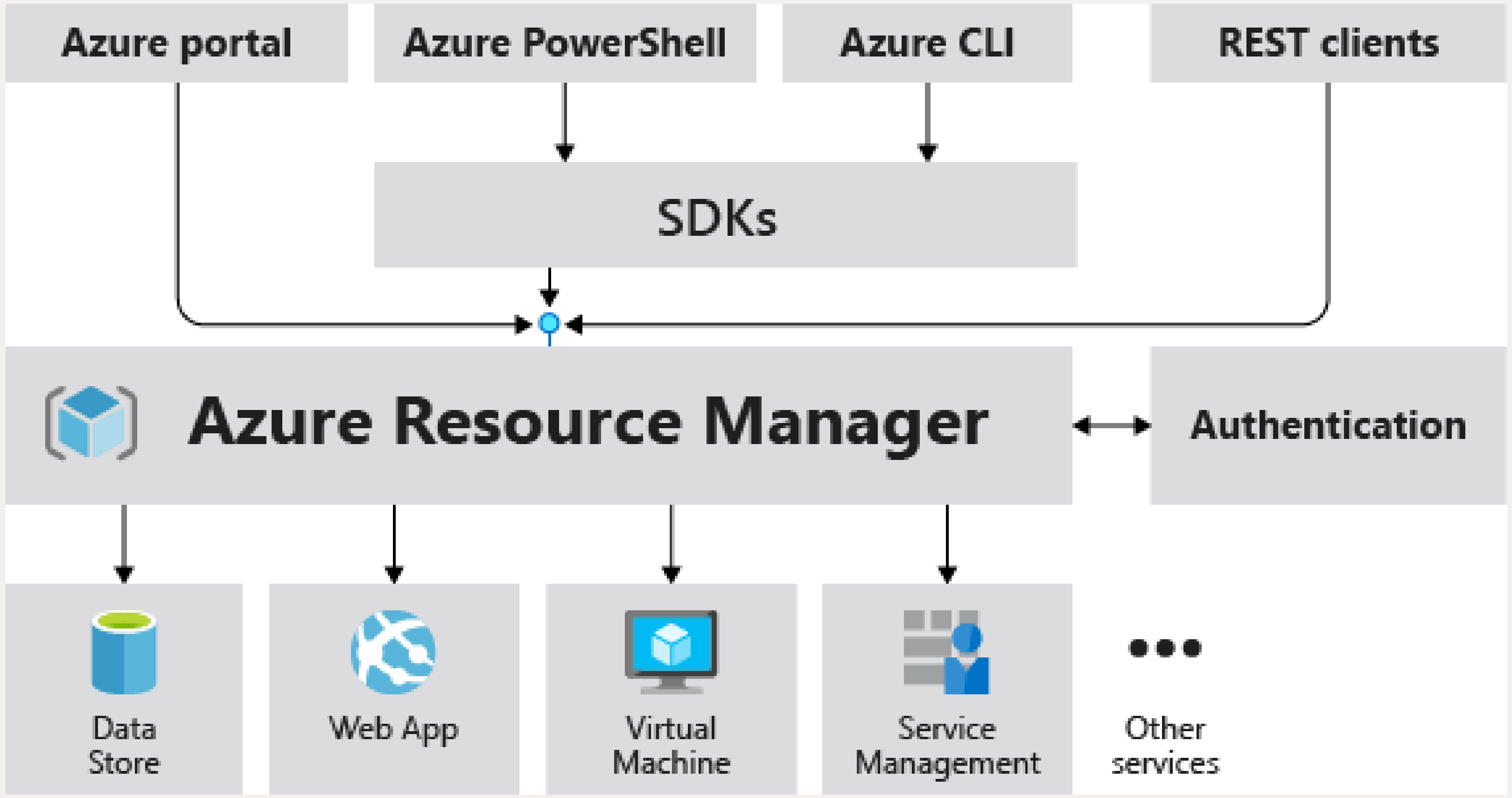
We do not tolerate any behaviour that is harassing or degrading to any individual, in any form. Individuals are responsible for knowing and abiding by these standards. We encourage everyone to assist in creating a welcoming and safe environment.

- Be aware of others
- Be friendly and patient
- Be welcoming and respectful
- Be open to all questions and viewpoints
- Be understanding of differences
- Be kind and considerate to others

Agenda

- *What is Azure Resource Manager?*
- *The complexity of ARM Templates*
- *Introduction to BICEP*
  - *File structure*
  - *Parameters/Variables*
  - *Resources*
  - *Modules*
  - *Functions*
  - *Scopes*

Azure Resource Manager



Benefits of Azure Resource Manager

- ➢ Deploy, manage, and monitor all the resources for your solution as a group, rather than handling these resources individually.
- ➢ Redeploy your solution throughout the development lifecycle and have confidence your resources are deployed in a consistent state.
- ➢ Apply tags to resources to logically organize all the resources in your subscription.
- ➢ Clarify your organization's billing by viewing costs for a group of resources sharing the same tag.
- ➢ Manage your infrastructure through declarative templates rather than scripts.
- ➢ Define the dependencies between resources so they're deployed in the correct order.
- ➢ Apply access control to all services because Azure role-based access control (Azure RBAC) is natively integrated into the management platform.
- ➢ Provides integrated monitoring and diagnostics.

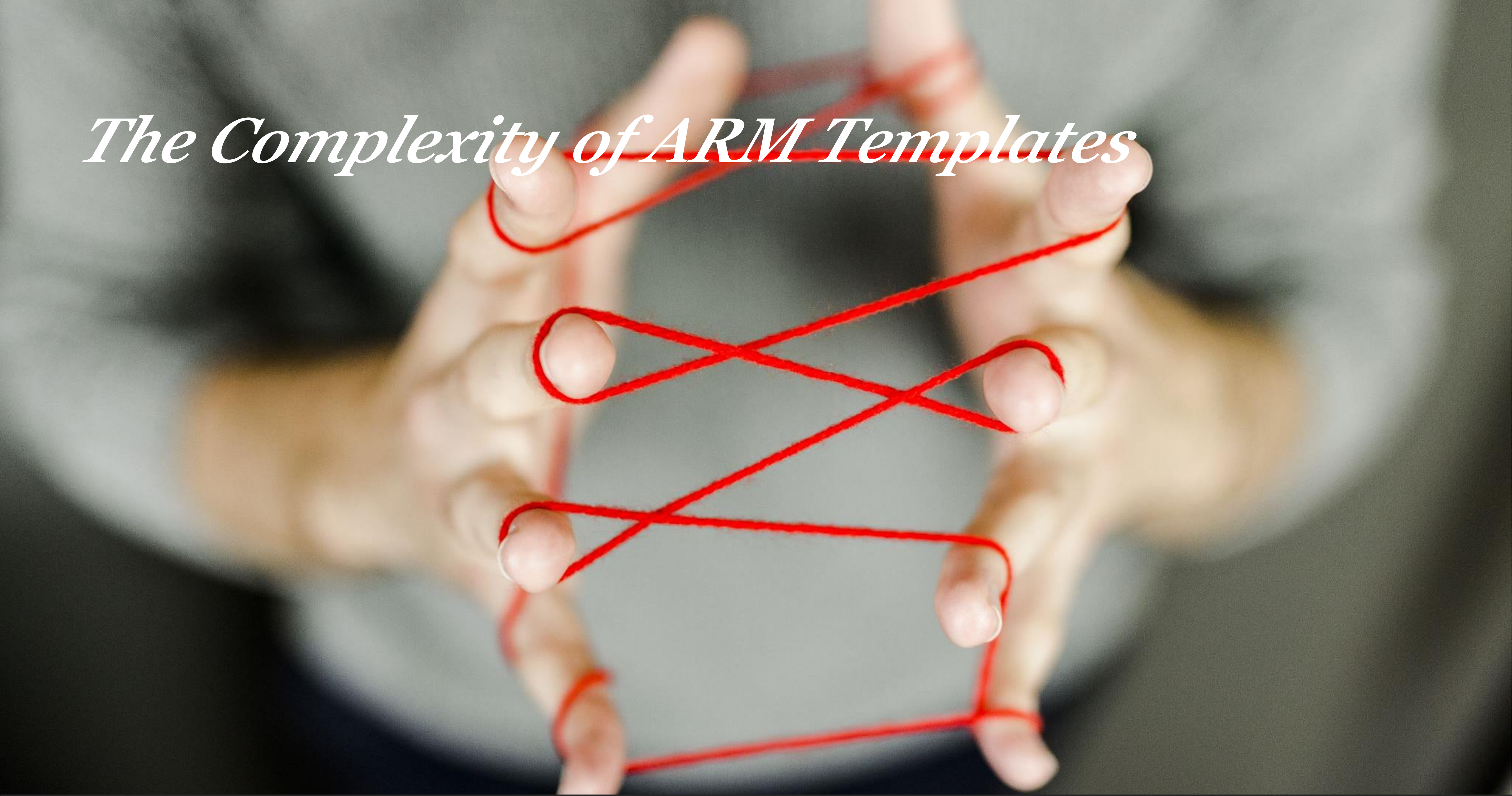
Control Plane vs Data Plane

Control Plane

- Requests always sent to the Azure Resource Manager URL
- Requests are handled by Azure Resource Manager
    - Sends individual requests to the resource providers to complete
- Knows when to create new resources and when to update existing ones

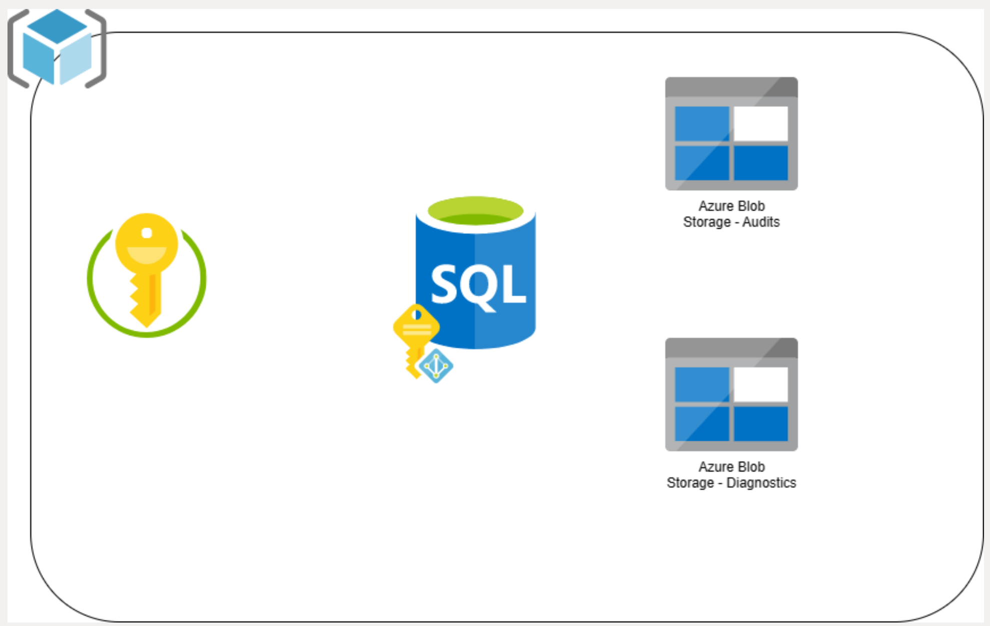
Data Plane

- Requests send to endpoints that are specific to the resource (e.g. https://myaccount.blob.core.windows.net/mycontainer/myblob)
- Not limited to REST API access
- Requests handled by the specific resource



The Complexity of ARM Templates

What's in our Sample



- Resources deployed to single Resource Group
- Azure SQL DB with User-Assigned Managed Identity
- Azure AD Administrator assigned to SQL
- Storage Account for SQL Audit Logs
- Storage Account for SQL Diagnostic Logs
- Key Vault with SAS Token for Storage Accounts
- RBAC Permissions applied to resources

What does that look like in ARM?

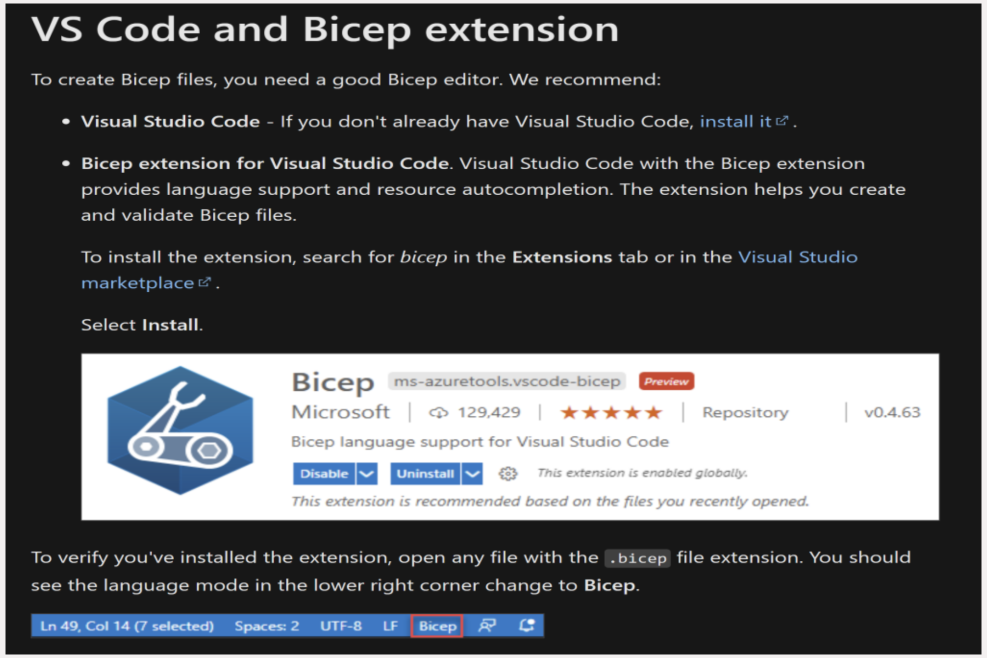
This Photo by Unknown Author is licensed under CC BY-SA

Introducing BICEP

What is BICEP?

- A Domain Specific Language for writing Infrastructure as Code
- Built on top of JSON for producing ARM Templates more concisely and intuitively
- Open Source
- Modular
- Type Safe
- Integrated with Azure CLI

BICEP Requirements



VS Code and Bicep extension

To create Bicep files, you need a good Bicep editor. We recommend:

- **Visual Studio Code** - If you don't already have Visual Studio Code, install it.
- **Bicep extension for Visual Studio Code**. Visual Studio Code with the Bicep extension provides language support and resource autocompletion. The extension helps you create and validate Bicep files.

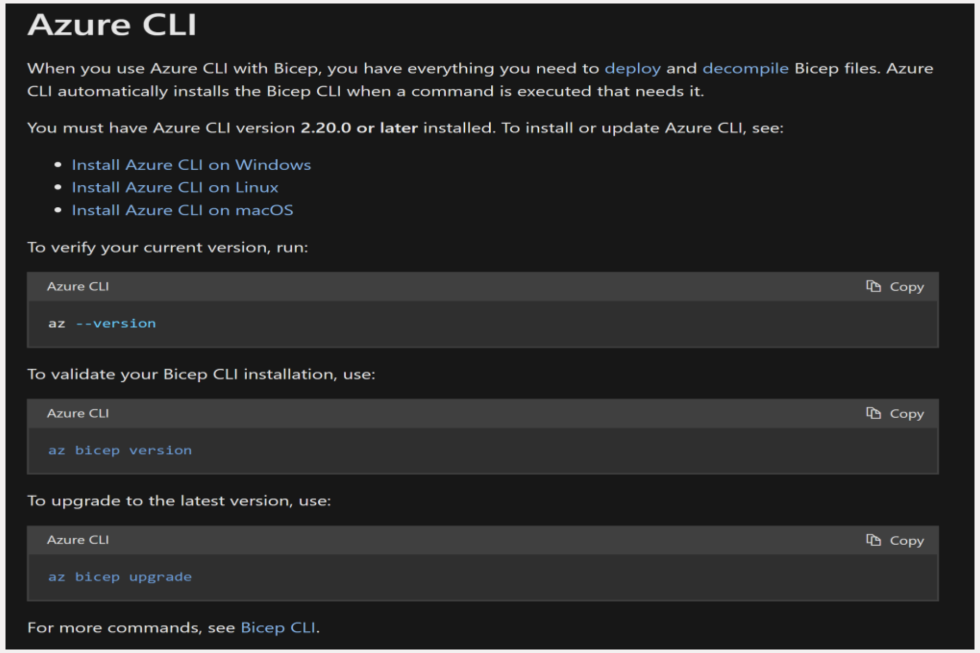
  To install the extension, search for *bicep* in the **Extensions** tab or in the Visual Studio marketplace.

  Select **Install**.

  Bicep ms-azuretools.vscode-bicep Preview
  Microsoft | 129,429 | ★★★★★ | Repository | v0.4.63
  Bicep language support for Visual Studio Code
  Disable | Uninstall | *This extension is enabled globally.*
  This extension is recommended based on the files you recently opened.

To verify you've installed the extension, open any file with the `.bicep` file extension. You should see the language mode in the lower right corner change to **Bicep**.

Ln 49, Col 14 (7 selected) Spaces: 2 UTF-8 LF Bicep



Azure CLI

When you use Azure CLI with Bicep, you have everything you need to deploy and decompile Bicep files. Azure CLI automatically installs the Bicep CLI when a command is executed that needs it.

You must have Azure CLI version **2.20.0 or later** installed. To install or update Azure CLI, see:

- Install Azure CLI on Windows
- Install Azure CLI on Linux
- Install Azure CLI on macOS

To verify your current version, run:

Azure CLI

```
az --version
```

To validate your Bicep CLI installation, use:

Azure CLI

```
az bicep version
```

To upgrade to the latest version, use:

Azure CLI

```
az bicep upgrade
```

For more commands, see Bicep CLI.

https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/install#vs-code-and-bicep-extension https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/install#azure-cli

BICEP File Structures

1
```
targetScope = '<scope>'
```

2
```
@<decorator>(<argument>)
param <parameter-name> <parameter-data-type> = <default-value>
```

3
```
var <variable-name> = <variable-value>
```

4
```
resource <resource-symbolic-name> '<resource-type>@<api-version>' = {
  <resource-properties>
}
```

5
```
module <module-symbolic-name> '<path-to-file>' = {
  name: '<linked-deployment-name>'
  params: {
    <parameter-names-and-values>
  }
}
```

6
```
output <output-name> <output-data-type> = <output-value>
```

1. Normally Resource Group, but can go up the levels.

2. Defined in the BICEP rather than needing a separate Parameter file like ARM Templates. Decorators allow constraints on the format/values.

3. Similar to ARM – define values that may be used repeatedly.

4. `resource` keyword defines a resource to deploy. Child resources can be defined either within a `resource` definition or at the same level. The symbolic name is how it is referenced throughout the remainder of the file.

5. `module` keyword defines a reference to another BICEP file, which contains further resources to deploy. Allows more simple reuse. The symbolic link allows the `module` to be referenced from anywhere within the file.

6. `output` return values from the deployment. Similarly to ARM you use these to expose the values to other resources or operations.

NOTE: BICEP is newline sensitive, so:
```
        resource sa 'Microsoft.Sql/server@2021-11-01' = if (p1 == 'value') {
          . . .
        }
```

Can't be written as:
```
        resource sa 'Microsoft.Sql/server@2021-11-01' =
               if (p1 == 'value') {
                 . . .
               }
```

Getting Started vs Mature Deployments

Local File

- Relative path from the calling module

```
module <symbolic-name> '../module.bicep' = {
```

Private Registry

- Hosted in an Azure Container Registry (ACR)

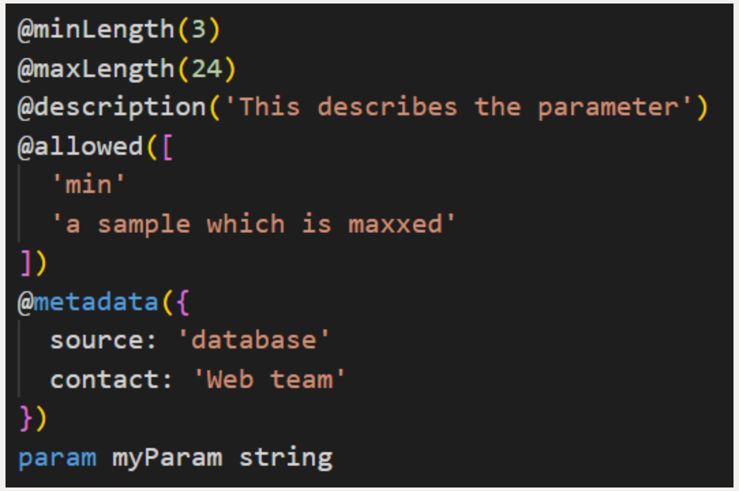
```
module <symbolic-name> 'br:<registry-name>.azurecr.io/<file-path>:<tag>' = {
```

Public Registry

- Hosted in an Microsoft Container Registry (ACR)

```
module <symbolic-name> 'br/public:<file-path>:<tag>' = {
```

BICEP Parameters & Variables



```
@minLength(3)
@maxLength(24)
@description('This describes the parameter')
@allowed([
  'min'
  'a sample which is maxxed'
])
@metadata({
  source: 'database'
  contact: 'Web team'
})
param myParam string
```

Minimum Requirements:

- Include name & data type
- optionally specify a default value by appending: " = `'value'` "

Decorators

- Add constraints or metadata
- `@secure()` decorator for strings/objects means parameter value is not saved to deployment history or logged.

Available Data Types

• string	s1 = 'a string'	• bool	b1 = false
• int	i1 = 42	• object	o1 = {p1: 'val', p2: 9}
• array	a1 = [v1, v2]		

BICEP Parameters & Variables

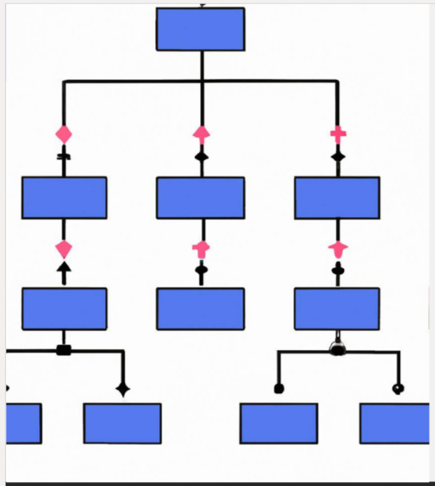
```
param inputValue string = 'initialValue'

var stringVar = '${toLower(inputValue)}${uniqueString(resourceGroup().id)}'
var concatToVar =  '${stringVar}AddToVar'
var concatToParam = '${inputValue}AddToParam'
```

Variables

- Do not need a data type – type is inferred
- Can have a MAX of 256 variables in a BICEP file
- Can't have the same name as a Parameter, Module or Resource
- Can re-use the value from other Parameters or Variables

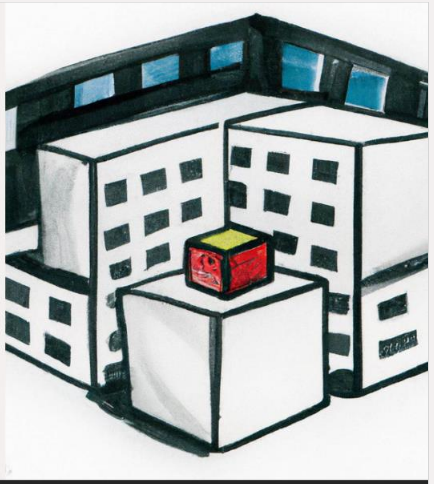
    Format for string interpolation is `${param_or_var_name}`

Creating Resources



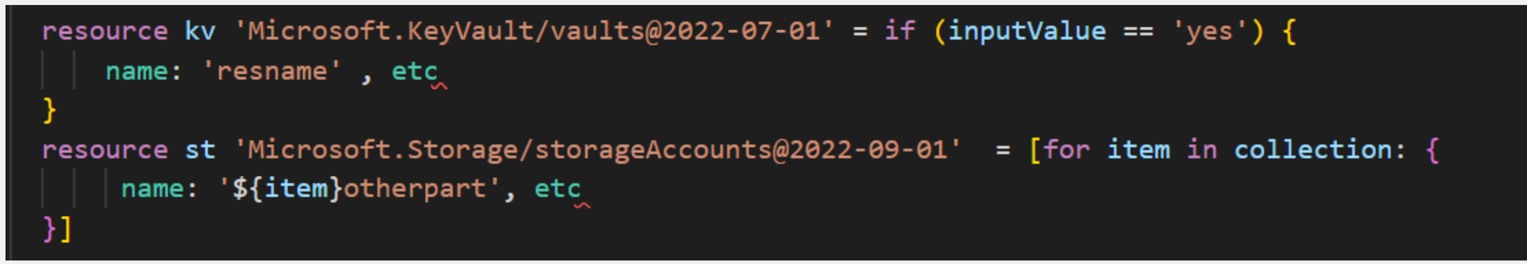
Child Resources

Extension Resources



Existing Resources

Creating Resources


```
resource kv 'Microsoft.KeyVault/vaults@2022-07-01' = if (inputValue == 'yes') {
    name: 'resname' , etc
}
resource st 'Microsoft.Storage/storageAccounts@2022-09-01'  = [for item in collection: {
     name: '${item}otherpart', etc
}]
```

BICEP Creating Resources

Declared with a ***symbolic name***

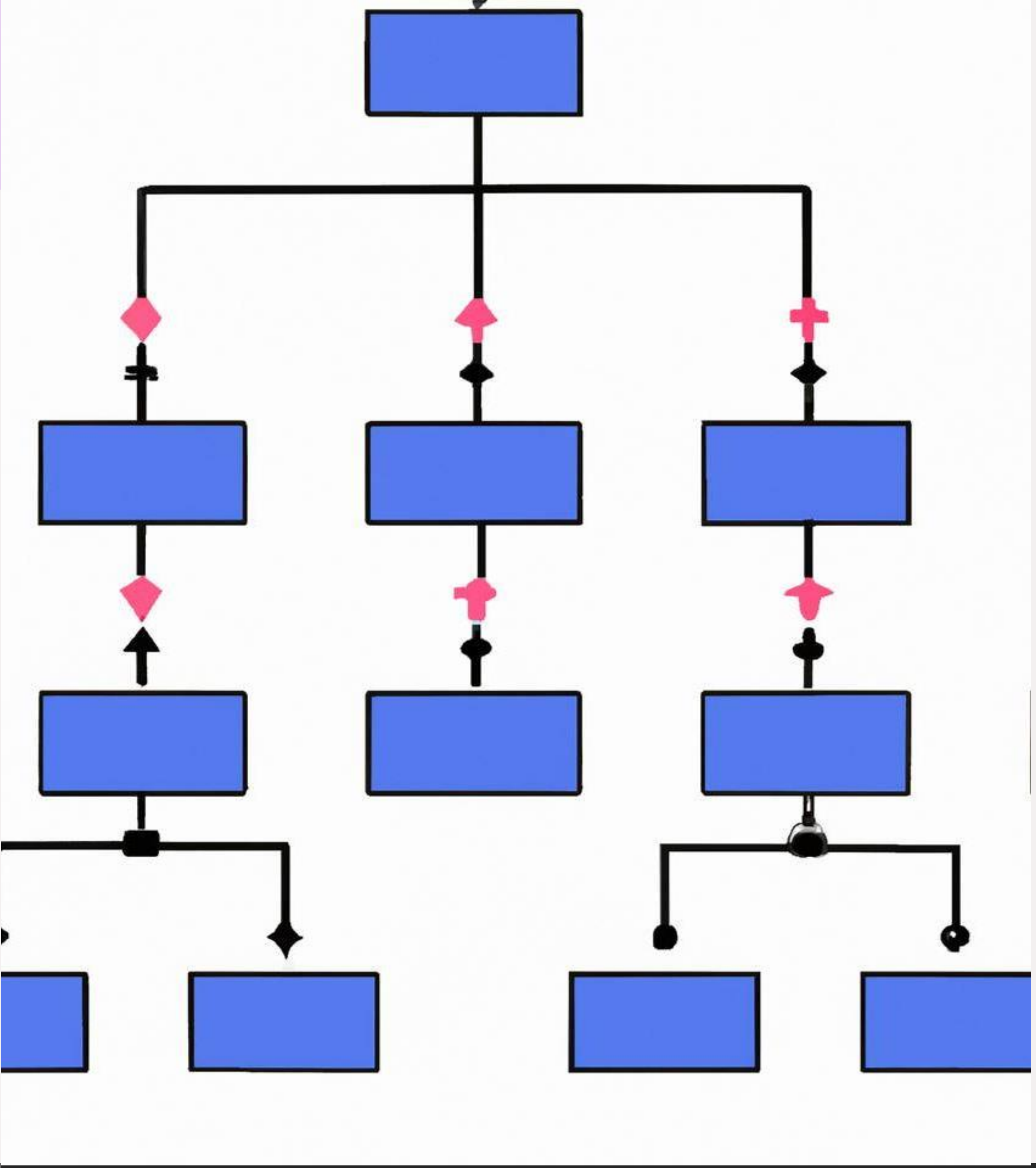
- Not the same as the resource name
- Used to reference the resource in other parts of the file
- Case sensitive
- **name** in the definition is the actual resource name (be wary of restrictions)

Conditional Deployment

- Use an `if` statement at the start of the definition

Multiple Copies of Resource

- Use `for` loops to iterate through multiple deployments



Child Resources

BICEP Child Resources

Nested Definition

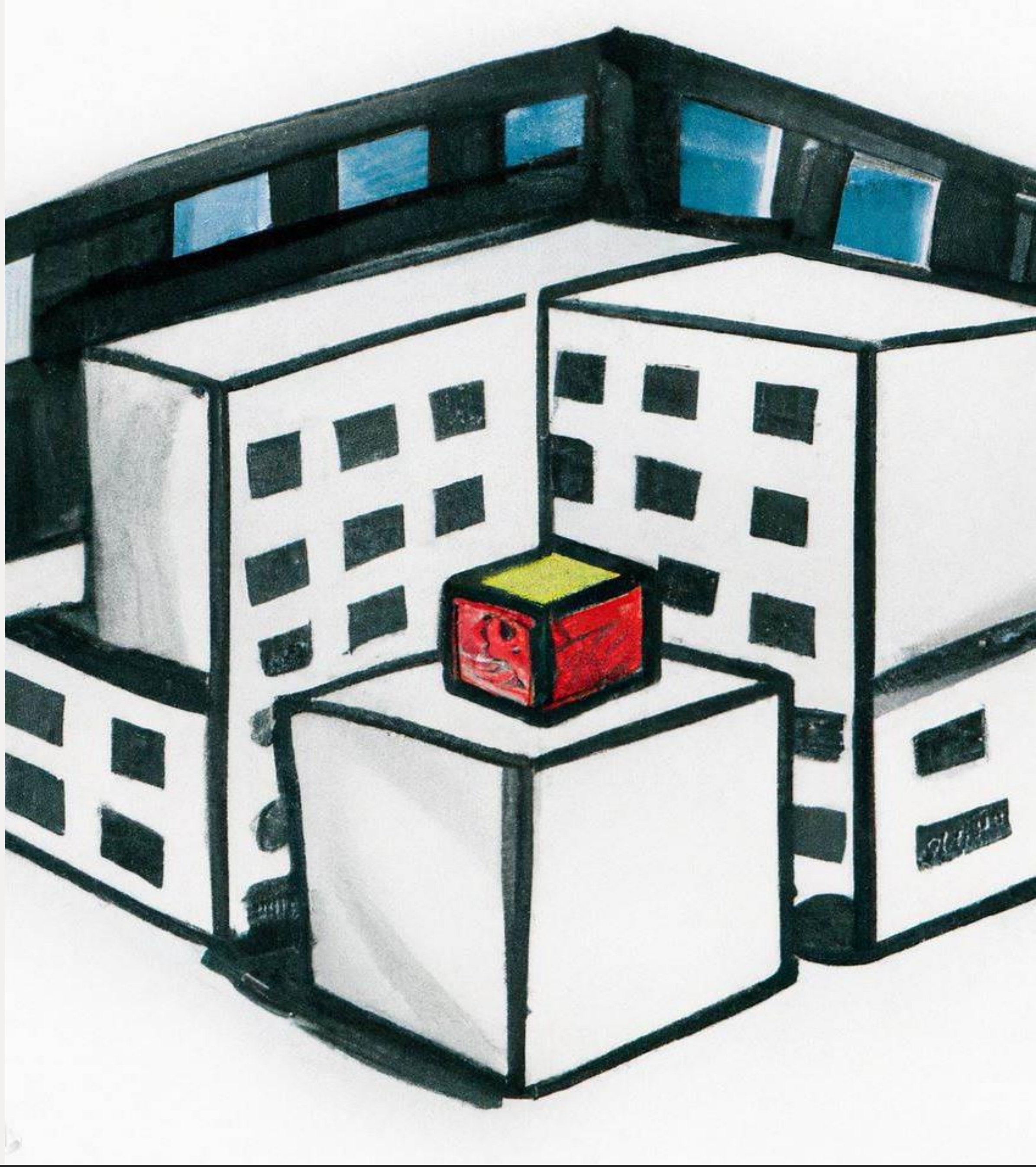
```
resource vm 'Microsoft.Compute/virtualMachines@2020-06-01' = {
    name: vmName
    location: location
    properties: {
      // ...
    }

    resource installCustomScriptExtension 'extensions' = {
      name: 'InstallCustomScript'
      location: location
      properties: {
        // ...
      }
    }
}
```

Use Parent Property

```
resource vm 'Microsoft.Compute/virtualMachines@2020-06-01' = {
    name: vmName
    location: location
    properties: {
      // ...
    }
}

resource installCustomScriptExtension 'Microsoft.Compute/virtualMachines/extensions@2020-06-01' = {
    parent: vm
    name: 'InstallCustomScript’
    location: location
    properties: {
      // ...
    }
}
```

References parent’s Symbolic Name

Extension Resources

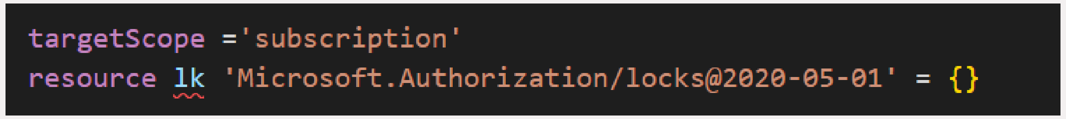


BICEP Extension Resources

An extension resource is a resource that modifies another resource.

- e.g. assign a role to a resource

targetScope

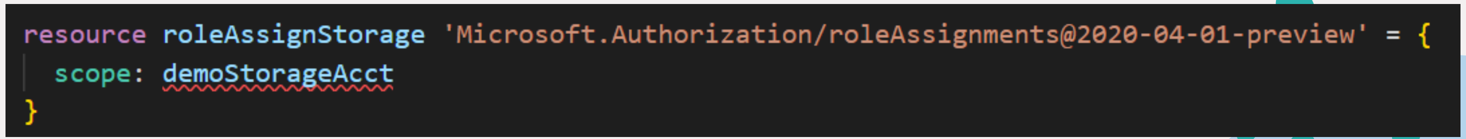


```
targetScope ='subscription'
resource lk 'Microsoft.Authorization/locks@2020-05-01' = {}
```

- Allows the resource to be deployed to a defined scope (e.g. at Subscription)

Scope Property

- Allows the resource to be applied to another resource
- Reference with Symbolic Name

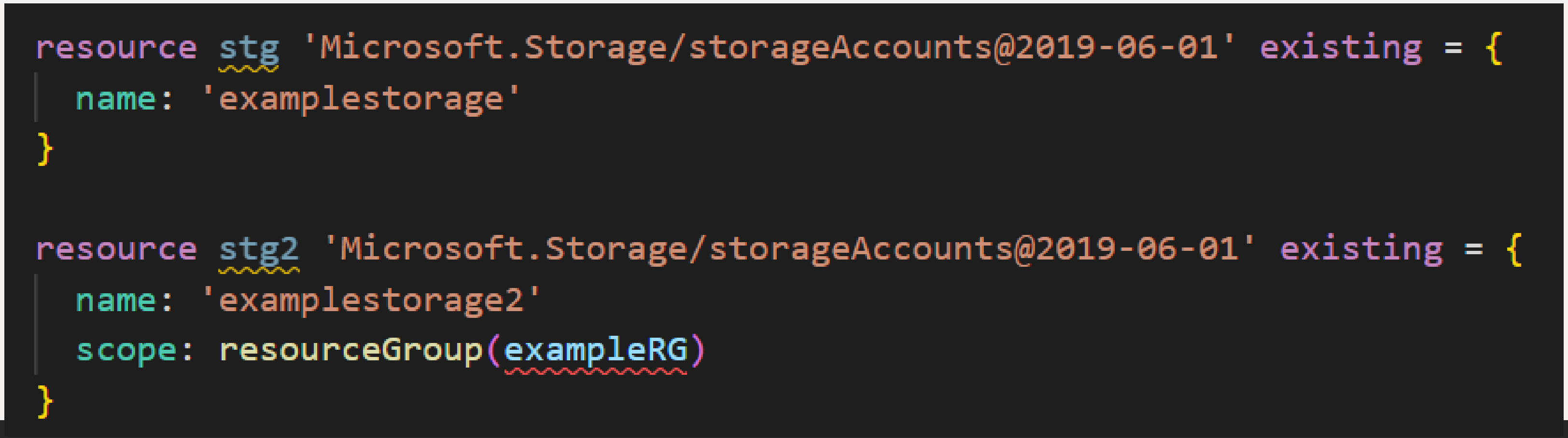


```
resource roleAssignStorage 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  scope: demoStorageAcct
}
```

Existing Resources

BICEP Existing Resources

- Declared with a ***existing*** keyword
- By default, references resources in the same Resource Group as the current deployment.
- Can reference a resource in a different Resource Group using the ***scope*** property.



```
resource stg 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
  name: 'examplestorage'
}

resource stg2 'Microsoft.Storage/storageAccounts@2019-06-01' existing = {
  name: 'examplestorage2'
  scope: resourceGroup(exampleRG)
}
```

BICEP Modules

A BICEP file deployed from another BICEP file.

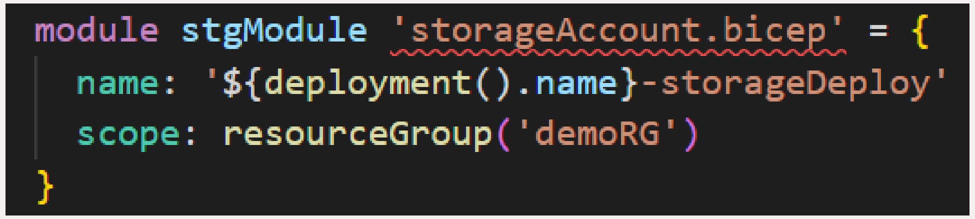
Encapsulate complex deployment details in a single file

- Abstracts complexity from calling file.

Like resources, can use conditions and loops

Parameters provided must match those in BICEP file (unless default values exist).

Name property becomes the name of the nested deployment in the generated ARM template – best practice to make it unique



```
module stgModule 'storageAccount.bicep' = {
  name: '${deployment().name}-storageDeploy'
  scope: resourceGroup('demoRG')
}
```

BICEP Functions

BICEP Functions - Groupings

Any Function

Array Functions

Date Functions

Deployment Functions

File Functions

Lambda Functions

Logical Functions

Numeric Functions

Object Functions

Resource Functions

Scope Functions

String Functions

BICEP Functions - Examples

Date Functions

- utcNow()
- dateTimeAdd()

File Functions

- loadJsonContent()
- loadTextContent()

Deployment Functions

- deployment()
- environment()

Lambda Functions

- map()
- reduce()
- toObject()

BICEP Functions - Examples

Scope Functions

- resourceGroup()
- subscription()
- Tenant()

Object Functions

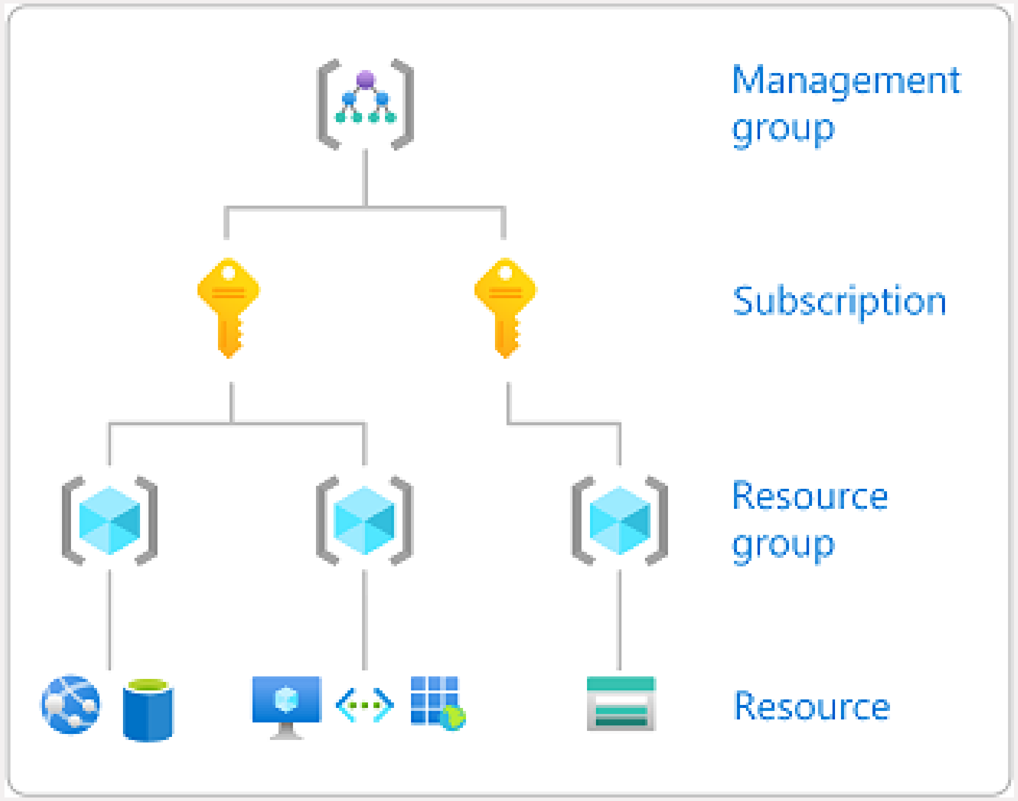
- items()
- json()
- length()

Resource Functions

- getSecret()
- subscriptionResourceID()

String Functions

- guid()
- format()
- uniqueString()
- substring()

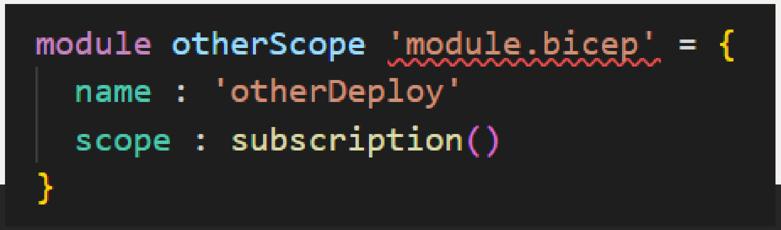


BICEP Scopes

Deployment Scopes

Resource Group

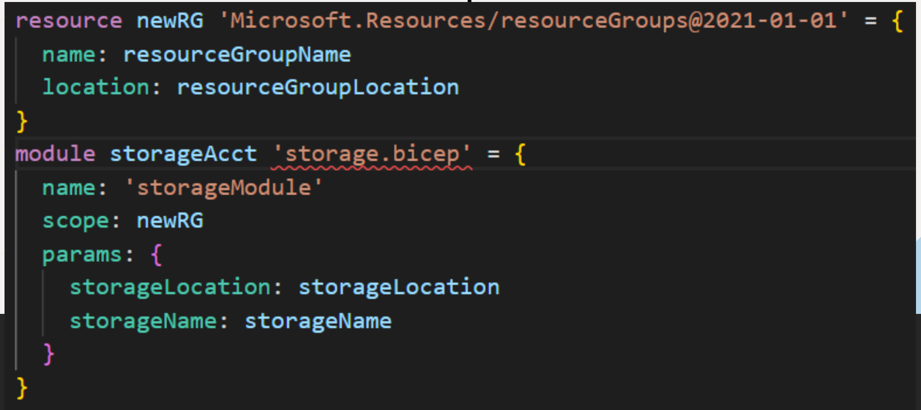
- The default
- Works for most resources
- Can deploy some resources to other Resource Groups
- Can deploy some resources to Subscription/Tenant



```
module otherScope 'module.bicep' = {
  name : 'otherDeploy'
  scope : subscription()
}
```

Subscription

- Must provide Location explicitly
- Deployment name is fixed to that location
- Can deploy to any Subscription or to Resource Groups



```
resource newRG 'Microsoft.Resources/resourceGroups@2021-01-01' = {
  name: resourceGroupName
  location: resourceGroupLocation
}
module storageAcct 'storage.bicep' = {
  name: 'storageModule'
  scope: newRG
  params: {
    storageLocation: storageLocation
    storageName: storageName
  }
}
```

What does that look like in BICEP?

This Photo by Unknown Author is licensed under CC BY-SA

DATA ANZ
AUSTRALIA

THANK YOU

First Last

email@contoso.com

www.contoso.com

Feedback Form

databricks

Technology Sirius

Telstra Purple