Microsoft

Bicep Demo

Agenda

- Introduction to Azure Bicep and its benefits over ARM templates
- Setting up an Azure Bicep development environment
- Basic syntax and structure of an Azure Bicep file
- Parameters in Azure Bicep
- Using Bicep modules to simplify resource declaration
- Bicep CLI
- Deploying Bicep templates using DevOps

Introduction to Azure Bicep and its benefits over ARM templates

1

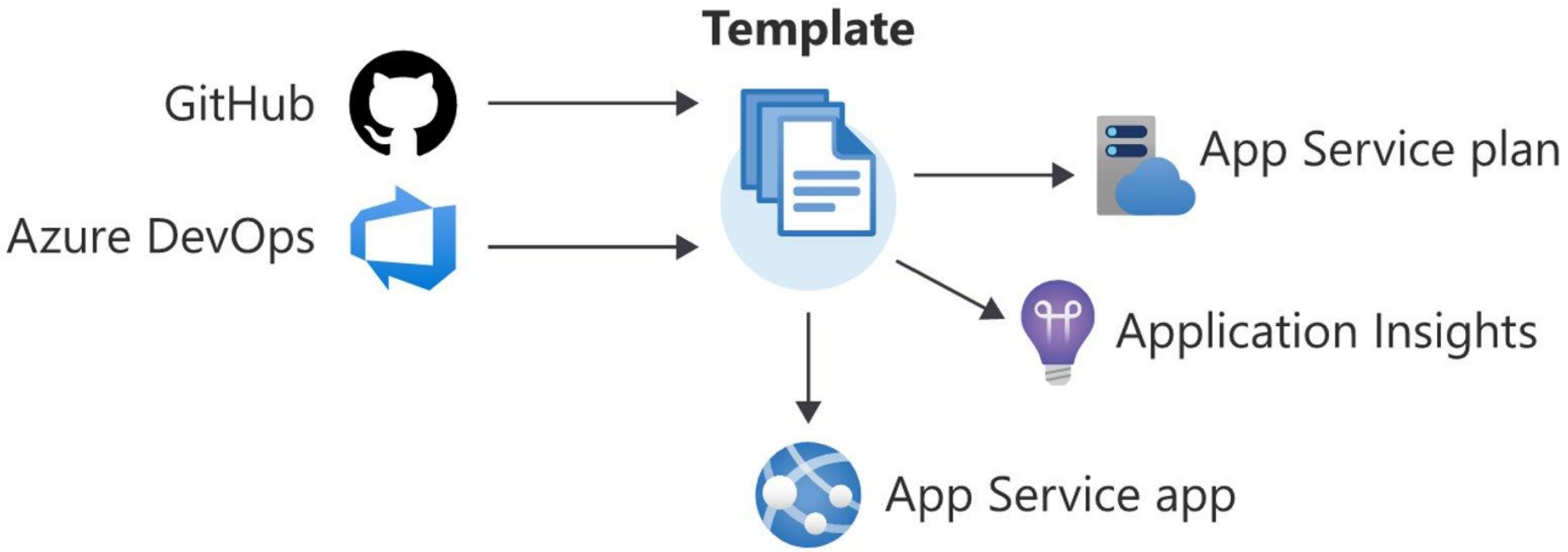
What is Infrastructure As Code

Infrastructure as code (IaC), is the process of provisioning infrastructure resources similarly to how software is deployed.

IaC deployments are automated, consistent, and repeatable.

IaC increase confidence in your deployments and increase efficiency.

What is Infrastructure As Code



ARM Template

ARM templates are files that define the infrastructure and configuration for your deployment.

These templates describe each resource in the deployment, but they don't describe how to deploy the resources.

When you submit a template to Resource Manager for deployment, the control plane deploy the defined resources in an organized and consistent manner.

Bicep Template

Bicep is a Domain Specific Language (DSL). It is a transparent abstraction layer on top of ARM templates. It improves the authoring experience of Infrastructure as code.

It has a much cleaner syntax, improves type safety, and has much better support for modularity to be able to re-use your code. Because it is a transparent abstraction layer on top of ARM and ARM templates, there is full feature parity.

Bicep Template

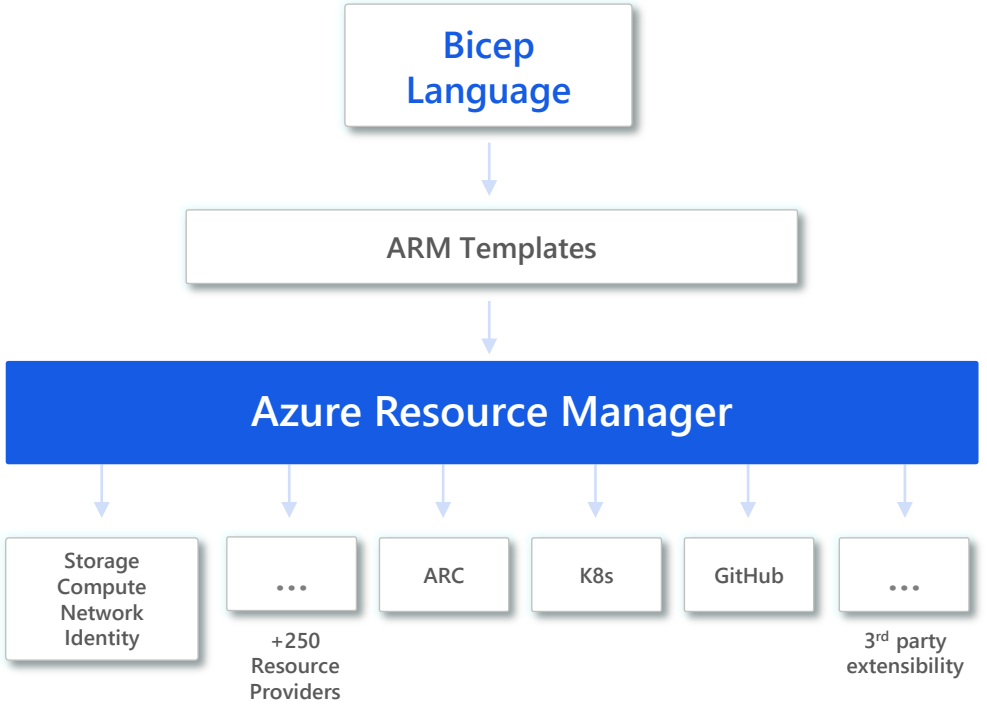
Anything you can do with ARM templates can also be done using Bicep. All resource types, API versions and properties valid inside of ARM templates are also valid in Bicep.

Bicep Template

When you deploy a resource or series of resources to Azure, you submit the Bicep template to Resource Manager, which still requires JSON templates.

The tooling that's built into Bicep converts your Bicep template into a JSON template. This process is known as transpilation, which essentially treats the ARM template as an intermediate language. The conversion happens automatically when you submit your deployment, or you can do it manually.

Bicep Template



Bicep Benefits

- There is day 0 support for new resource types and API versions and there is no state file to manage (Azure is the state).
- Modularity is a key aspect of Bicep. The modular approach allows you to extract pieces of your code into reusable modules, exposing parameters and outputs as the contract to other Bicep templates.
- When comparing the Bicep syntax with ARM templates there is a huge difference in terms of syntax. Bicep is much easier to read and to author.

Bicep vs ARM

Bicep provides a simpler syntax to use when you're writing templates. Look at the following examples of two templates.

```
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"
      }
    }
  ]
```

Comparison showing Bicep code on the left and the corresponding JSON code on the right.

Setting up an Azure Bicep development environment

Installing the Bicep CLI (Azure CLI)

- 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
- **Run the Bicep commands with the *az bicep* syntax for Azure CLI**

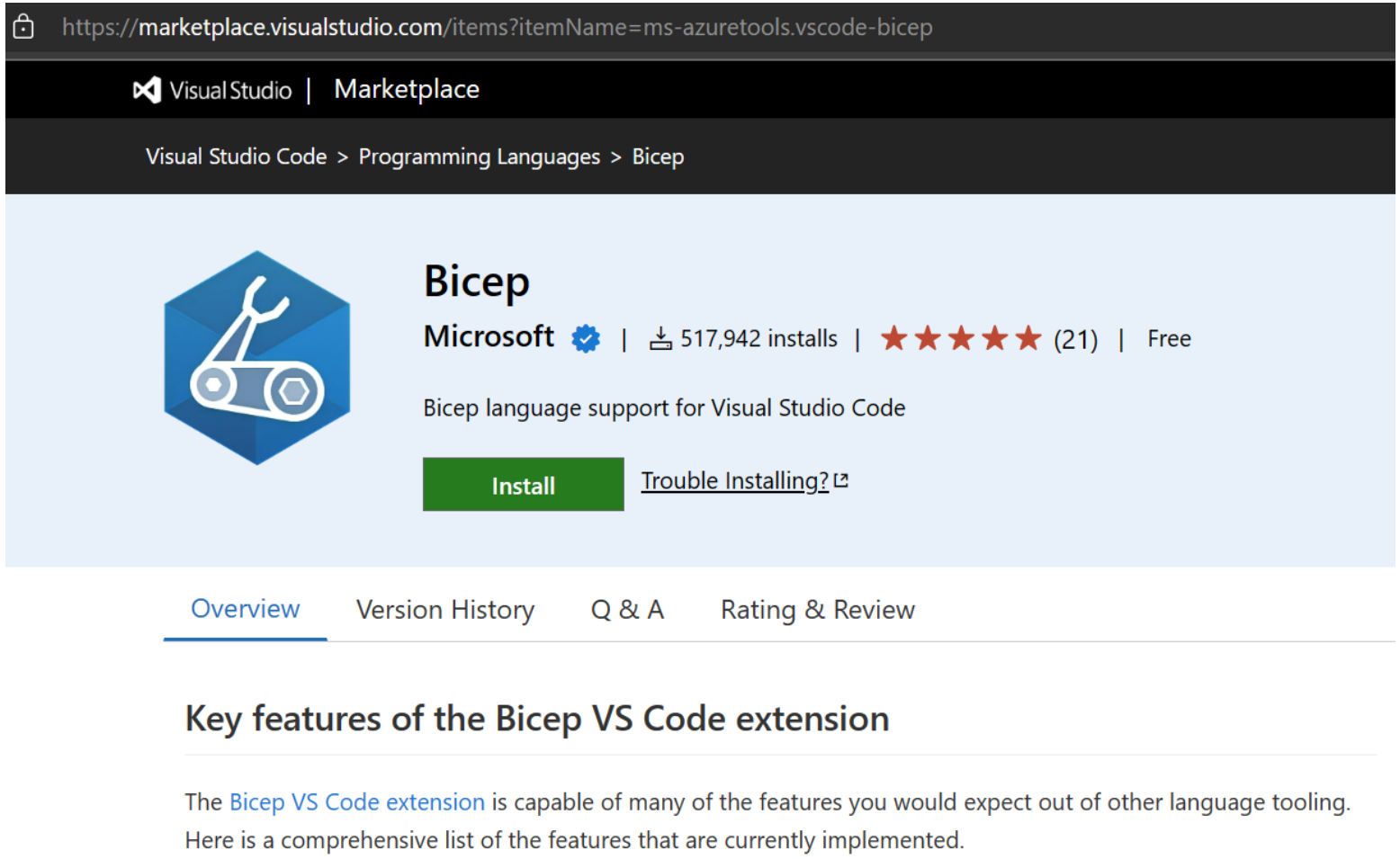
- az bicep install

Installing the Bicep CLI (Air-Gapped using Powershell)

```
# Create the install folder
$installPath = "$env:USERPROFILE\.bicep"
$installDir = New-Item -ItemType Directory -Path $installPath -Force
$installDir.Attributes += 'Hidden'
# Fetch the latest Bicep CLI binary
(New-Object Net.WebClient).DownloadFile("https://github.com/Azure/bicep/releases/latest/download/bicep-win-x64.exe", "$installPath\bicep.exe")
# Add bicep to your PATH
$currentPath = (Get-Item -path "HKCU:\Environment" ).GetValue('Path', '', 'DoNotExpandEnvironmentNames')
if (-not $currentPath.Contains("%USERPROFILE%\.bicep")) { setx PATH ($currentPath + ";%USERPROFILE%\.bicep") }
if (-not $env:path.Contains($installPath)) { $env:path += ";$installPath" }
```

- **When you manually install the Bicep CLI, run the Bicep commands with the ***bicep*** syntax, instead of the ***az bicep*** syntax for Azure CLI.**

Installing the Bicep VS Code Extension



Basic syntax and structure of an Azure Bicep file

Bicep Resource Definition

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

Bicep Resource with Dependencies

```
resource appServicePlan 'Microsoft.Web/serverFarms@2020-06-01' = {
  name: ‘app1-p-asp'
  location: 'eastus'
  sku: {
    name: 'F1'
    tier: 'Free'
  }
}

resource appServiceApp 'Microsoft.Web/sites@2020-06-01' = {
  name: ‘app1appsvc'
  location: 'eastus'
  properties: {
    serverFarmId: appServicePlan.id
    httpsOnly: true
  }
}
```

Bicep Existing Resource

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

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

Parameters in Azure Bicep

Bicep Parameters

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 Parameters in Command Line

az deployment group create --template-file main.bicep

Bicep Parameter File

```
{
  "$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"
        }
      ]
    }
  }
}
```

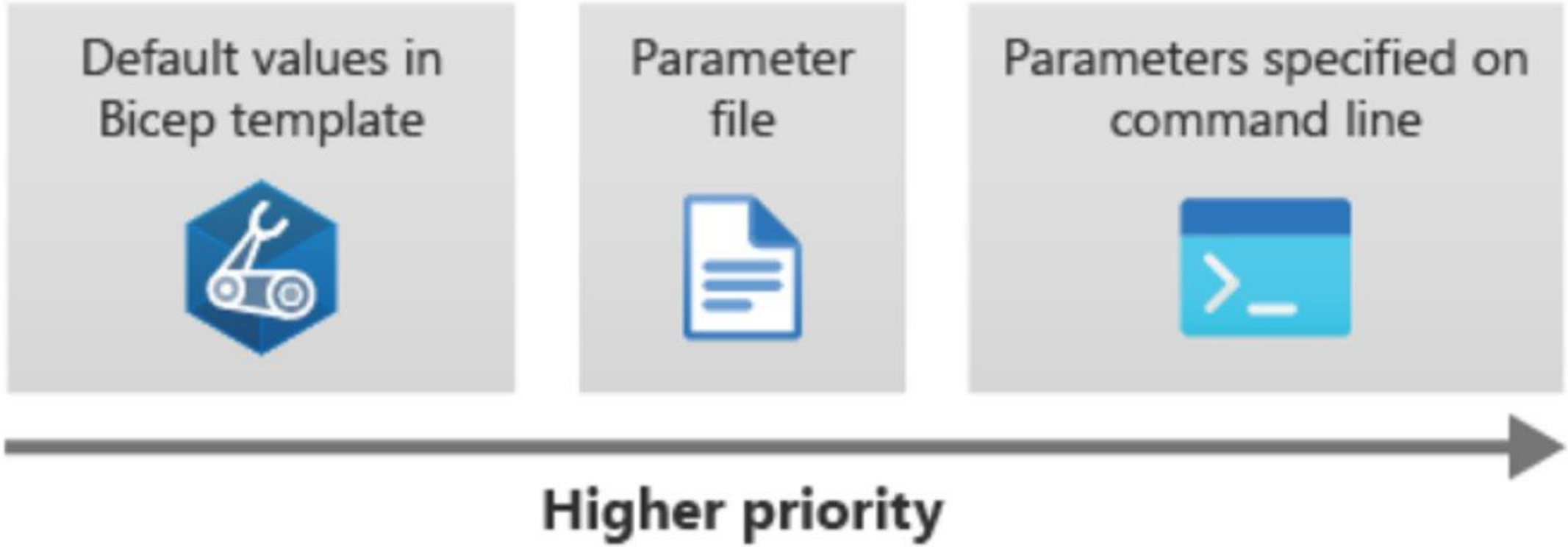
Bicep Parameters in Command Line

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

Bicep Parameter File with Command Line

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

Bicep Parameter Precedence



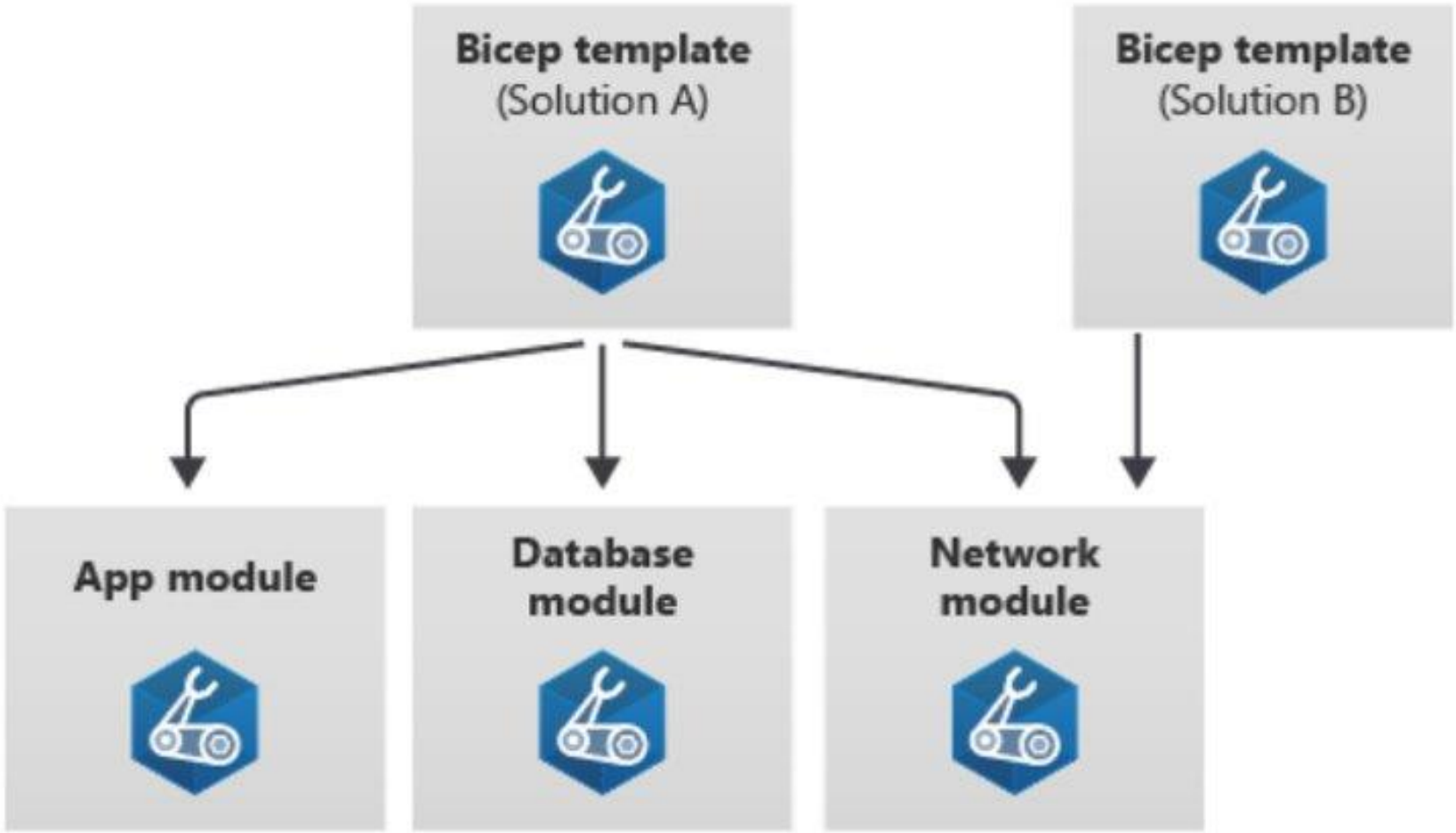
Using Bicep modules to simplify resource declaration

Bicep Modules

When your Bicep code is becoming more complex and has an increasing number of resources defined, 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.

Bicep Modules



Bicep Referencing Module

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

Bicep CLI

Bicep Decompile

The decompile command converts ARM template JSON to a Bicep file.

```
az bicep decompile --file main.json
```

Bicep Build (Compile to json)

The build command converts a Bicep file to an Azure Resource Manager template (ARM template).

Typically, you don't need to run this command because it runs automatically when you deploy a Bicep file.

```
az bicep build --file main.bicep
```

Bicep Publish

The publish command adds a module to a registry.

The Azure container registry must exist and the account publishing to the registry must have the correct permissions.

```
az bicep publish --file <bicep-file> --target br:<registry-name>.azurecr.io/<module-path>:<tag> --documentationUri <documentation-uri>
```

Bicep Deploy

You can target your deployment to a ***resource group***, ***subscription***, ***management group***, or ***tenant***. Depending on the scope of the deployment, you use different commands.

Default scope is "***Resource Group***"

Bicep Deploy Resource Group

To deploy to a resource group, use az deployment group create:

```
az deployment group create --resource-group <resource-group-name> --template-file <path-to-bicep>
```

Bicep Deploy Subscription

To deploy to a subscription, use az deployment sub create:

```
az deployment sub create --location <location> --template-file <path-to-bicep>
```

Bicep Deploy Management Group

To deploy to a management group, use az deployment mg create:

```
az deployment mg create --location <location> --template-file <path-to-bicep>
```

Bicep Deploy Tenant

To deploy to a tenant, use az deployment tenant create:

```
az deployment tenant create --location <location> --template-file <path-to-bicep>
```

Deploying Azure Bicep templates using DevOps

Bicep Resources

Learn

- Microsoft Docs
  - This is the recommended way to skill up on BICEP
- Bicep language for deploying Azure resources

Community

- https://github.com/Azure/bicep
- https://twitter.com/Biceplang
- Getting Started with Azure Bicep

Bicep Tools

- Bicep Playground
- VS Code Extension
- Best Practices