Infrastructure as Code – Intro to Bicep

What is Infrastructure as Code (IaC)?

- **Idempotent!**
- Codify Infrastructure: VMs, Storage, Networks, etc.
- Declarative or Imperative Definitions (Terraform vs Ansible)
- Repeatable, version-controlled deployments
- Enables DevOps, CI/CD, Automation

Infrastructure as Code

Infrastructure as Code (IaC) is the practice of managing and provisioning cloud infrastructure using machine-readable configuration files instead of manual processes or GUIs.

Develop a passion for learning.

Why IaC Matters in Azure

Why Azure Engineers Use IaC!

- Consistency across environments
- Avoid drift from manual changes
- Supports automation pipelines
- Auditability and Collaboration

Pro Tip!

Always parameterize resource names and locations in your Bicep templates—this makes your deployments reusable across environments.

Develop a passion for learning.

Azure's First IaC Language: ARM JSON

JSON Arm Templates

- JSON-Based syntax for defining Azure Resources.
- Used by Azure Portal, CLI, and SDK
- Declarative and Expressive
- Not Human-Friendly

ARM JSON templates are too verbose, hard to read, and difficult to maintain—Bicep solves this with cleaner syntax and better tooling.

```
{
  "$schema": "https://schema.management.azure.com/schemas/...
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "type": "string"
    },
    "location": {
      "type": "string",
      "defaultValue": "eastus"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2022-09-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {}
    }
  ],
  "outputs": {
    "storageAccountName": {
      "type": "string",
      "value": "[parameters('storageAccountName')]"
    }
  }
}
```

Develop a passion for learning.

Enter Bicep – A Better Way to Deploy

Bicep: ARM JSON, Reimagined

- DSL (Domain-Specific Language) that compiles to ARM
- Concise, Readable, Maintainable
- Strong Tooling support (VS Code, Linting, Auto-Complete)
- Built by Microsoft, fully supported

Bicep is preferred today because it's easier to read, faster to write, and fully supported by Azure—all without sacrificing the power of ARM templates.

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

resource sa 'Microsoft.Storage/storageAccounts@...
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {}
}

output saName string = sa.name
```

Develop a passion for learning.

Declarative vs. Imperative

Where Bicep Fits: Declarative by Design

- **Imperative:** Step-by-Step (e.g., Bash, PowerShell, Ansible)
- **Declarative:** Describe Desired State (e.g., Bicep, Terraform)
- Bicep focuses on ***what*** to deploy, not ***how***.
- Azure Resource Manager ensures consistency

Declarative Order (like Bicep):

"I'd like a venti iced oat milk caramel macchiato, light ice, with two extra pumps of vanilla and an extra shot of espresso"

Why It's Declarative:

You describe what you want, not how they should make it. The barista figures out the steps. Just like Bicep describes desired state, not the procedure.

Develop a passion for learning.

Recap – Why Bicep?

Why Bicep is the Future of Azure IaC

- **Cleaner syntax, fewer errors**
- **Full support from Microsoft**
- **Replaces ARM without sacrificing compatibility**
- **Easy to learn, Easy to extend**

To Summarize

Bicep is a Domain-Specific Language (DSL) for deploying Azure resources declaratively, offering a cleaner alternative to ARM JSON while compiling to the same underlying engine.

Develop a passion for learning.