InfernoRed
TECHNOLOGY

Unlock the full potential of Azure with Bicep!

Roberto Hernández

CONTENT

- Infrastructure as Code (IaC)
- Azure IaC Tools
- Installing Bicep
- Core Bicep Syntax and Features
- Code reusability in Bicep (Modules)
- Loops and Conditional expressions in Bicep
- Bicep Best Practices
- Q&A

Infrastructure as Code (IaC)

(What is?) Is the process of managing and provisioning computer data center resources through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

The definitions may be in a version control system, rather than maintaining the code through manual processes.

Principles: Automation, Repeatability, Version Control, Scalability, and Improved Security.

Benefits of IaC

Eliminate environment drift in release pipelines.

Eliminate manual configuration

Enforces consistency

InfernoRed TECHNOLOGY

IaC in Azure

Azure CLI	Imperative Bash/Powershell
Azure Resource Manager (ARM)	Declarative JSON/XML templates.
Azure Bicep	Language Declarative Easier to use syntax. Easy migration path from ARM

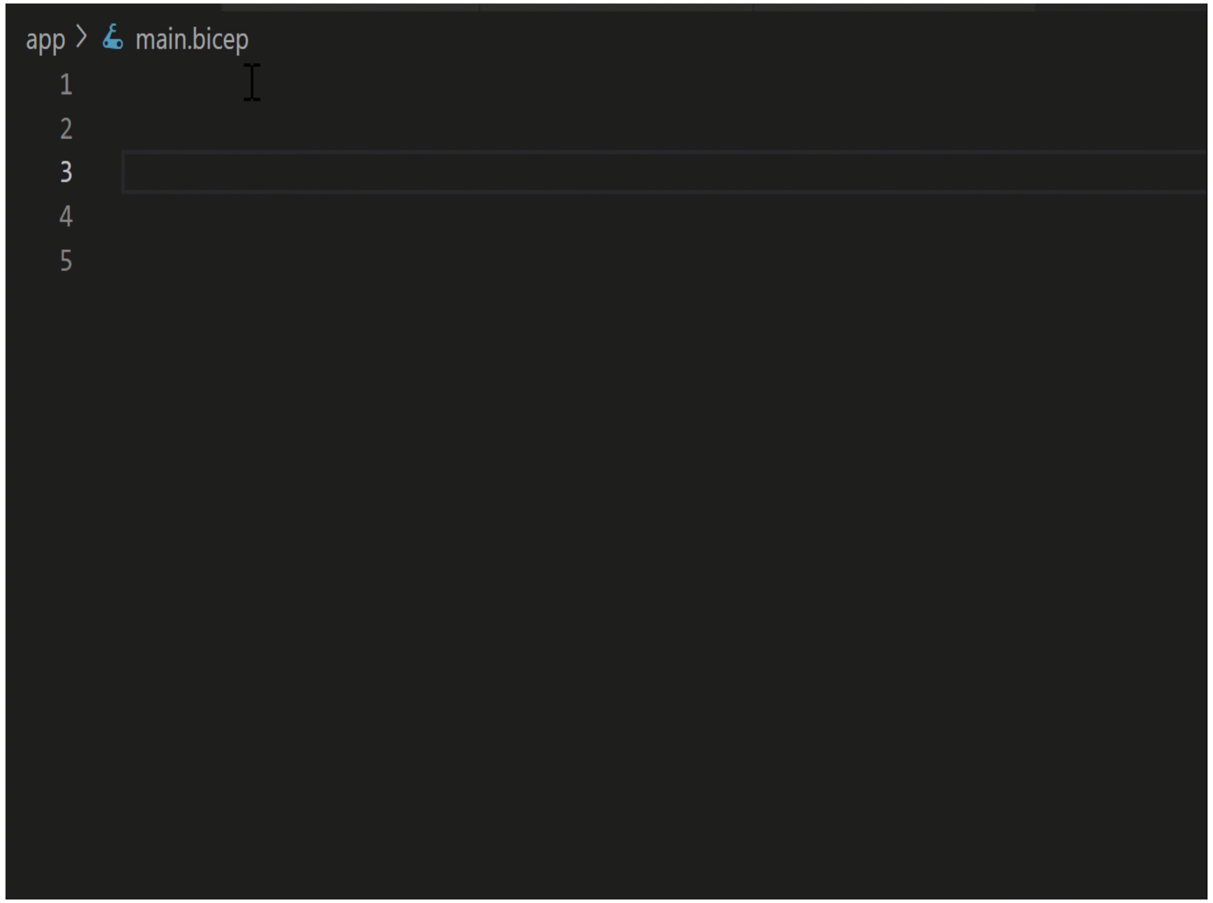
InfernoRed TECHNOLOGY

What is Azure Bicep?

Bicep is a transparent abstraction over ARM templates.

Key features and benefits (easier syntax, modularity, integration with ARM).

High-level comparison of Bicep vs. ARM templates vs. Terraform.



InfernoRed TECHNOLOGY

DEMO

Setting Up Your Environment for Bicep

Core Bicep Syntax and Features

Basic Structure: Declare and initialize resources with Bicep.

Variables and Parameters: Using variables and parameters for dynamic values.

Outputs: Defining and using outputs.

Expressions and Loops: Advanced syntax for handling complex requirements.

DEMO

Core Bicep Syntax and Features

Bicep Modules and Reusability

Organize code and make it reusable.

Use and share modules across different projects within your organization.

Use modules from the Azure Verified Modules (AVM) repository.

```
module stgModule '../storageAccount.json' = {
  name: 'storageDeploy’
  params: {
   storagePrefix: 'examplestg1’
  }
}
```

Azure Verified Modules

Initiative to consolidate and set the standards for what a good Infrastructure-as-Code module looks like.
Modules will then align to these standards, across languages (Bicep, Terraform etc.) and will then be classified as AVMs and available from their respective language specific registries.

```
@description(Storage accounts using an AVM.’)
module storage 'br/public:avm/res/storage/storage-
account:0.9.0' = {
  name: 'myStorage’
  params: {
    name: 'store${resourceGroup().name}’
  }
}
```

https://azure.github.io/Azure-Verified-Modules/

InfernoRed TECHNOLOGY

DEMO

Bicep Modules and Reusability

DEMO

Looping and Conditionals
Bicep Syntax and Features

Best Practices (Parameters)

Naming	Name should be readable.
Document	Use the description attributes to expand as appropriate.
@allowed	Use the allowed attribute sparingly as the constraints may become obsolete quickly.

Best Practices (Names)

Naming	Use camel case for variables and resources. For resources specifically follow the Azure Naming guidelines.
Unique	Use the built-in uniqueString function to prevent name clashes across your environments. Use the allowed attribute sparingly as the constraints may become obsolete quickly.

InfernoRed TECHNOLOGY

Best Practices (Output Parameters)

Sensitive Data

Make sure you don't create outputs for sensitive data. Output values can be accessed by anyone who has access to the deployment history. They're not appropriate for handling secrets.

Instead of passing property values around through outputs, use the existing keyword to look up properties of resources that already exist. It's a best practice to look up keys from other resources in this way instead of passing them around through outputs. You'll always get the most up-to-date data.

DEMO

Walkthrough some of the Best Practices in the Demo Code

THE END

Roberto Hernandez
roberto.hernandez@infernored.com

https://github.com/overridethis/bicep-walkthrough