Infrastructure as Code Bakeoff:
ARM/Bicep vs Terraform vs Pulumi

Mike Benkovich
Explore the Possible
www.benkoTIPS.com

Mike Benkovich

- Enterprise Cloud Architect & **Consultant**
- Live in **Minneapolis**
- Founder of **Imagine Technologies**, Inc.
- Developing Courses for **LinkedIn** Learning
- Blog www.benkoTIPS.com
- Follow @mbenko on **Twitter**
- Send me **Feedback**! mike@benko.com
- Azure **Office Hours** on Fridays! https://bit.ly/BnkAzHrs

Azure Office Hour Fridays

 @MikeBenkovich 05/04/2021 BenkoTIPS

Thinking about going to Cloud? I've been consulting around Azure for the last 8 years since I left Microsoft where I helped launch it in 2009. I want to offer my support, so I'm starting a thing called Azure Office Hours on Fridays, where anyone can block out 15 minutes to chat about anything Azure.

1. Find a time that works - https://bit.ly/BnkAzHrs
2. Let me know what you want to talk about
3. Let's chat!

I speak at conferences and have LinkedIn learning courses on Azure and DevOps including templating, compute, storage, messaging, networking and governance topics.

My calendar is open. Let's connect!

0 Comments

Today

Infrastructure as Code and Azure

Hello ARM and Bicep

Hello Terraform

Considerations

Why use Infrastructure as Code (IaC)?

Efficiency

Scalability

Security

Management

Maintenance

An Application is an Idea...

Data

Code

+ Infrastructure

= Application

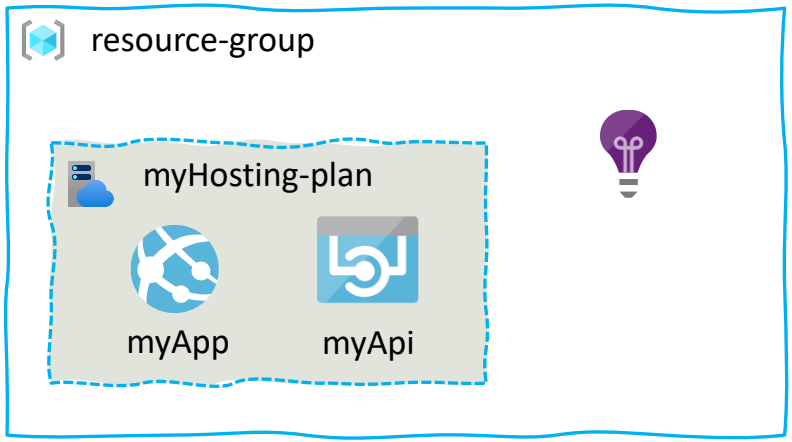
Cloud Application

- Runs in a cloud datacenter
- Virtualized hardware
- Monitored
- Configurable
- Scalable

Basic Infrastructure

Simple web app

- Resource Group
- Hosting Plan
- App Svc
- Api
- Insights
- Autoscale rule



Questions?

What should I use?

What tools do I need?

Cross Cloud support or on-prem?

Who manages the state?

Learning curve?

Readability of the language?

Immediate support for new features?

Does it support modules?

How does it do DevOps processes?

Azure Deployment Options

Azure Native

- Azure Portal
- Azure PowerShell
- Azure CLI
- ARM Templates
- Bicep

Cross Cloud (3rd Party)

- Terraform
- Pulumi
- Ansible
- Chef
- and more…

Infrastructure as Code (IaC)

Code

Versioned & managed

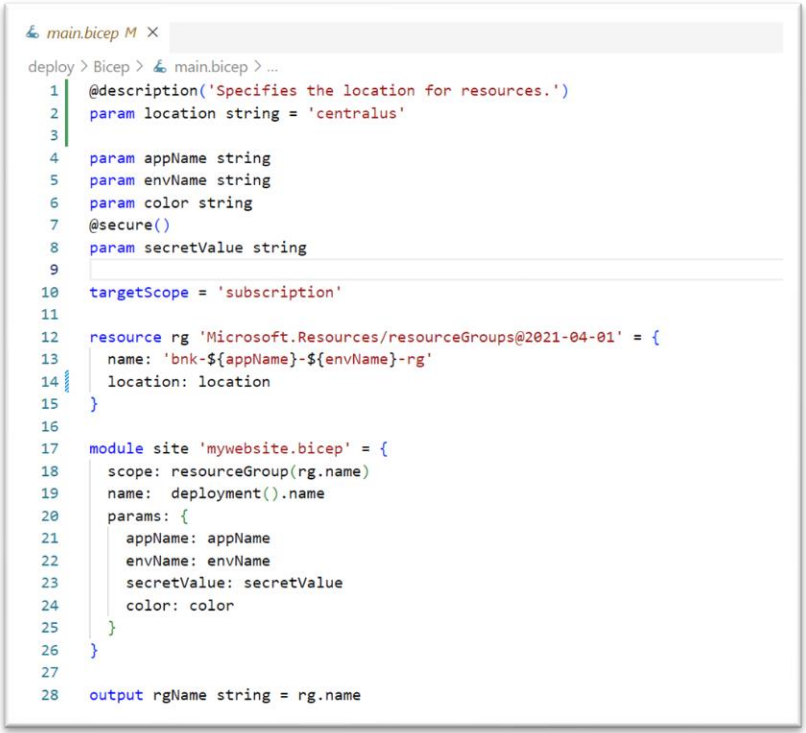
Repeatable

Descriptive vs Procedural

Environment drift

Idempotent

Current



ARM

ARM Template

- A declarative way to work with a resource provider
- Includes one or more resources
- Provides configuration information
- Each resource is translated into the REST call

Template Structure

```
{
  "$schema":"https://schema.management.azure.com/schemas/20...
  "contentVersion": "1.0.0.0",
  "parameters": {  },
  "variables": {  },
  "resources": [  ],
  "outputs": {  }
}
```

Resource section

```
"resources": [
    {
      "name": "[parameters('storageName')]",
      "type": "Microsoft.Storage/storageAccounts",
      "location": "[resourceGroup().location]",
      "apiVersion": "2016-01-01",
      "sku": {...},
      "dependsOn": [...],
      "tags": {...},
      "kind": "Storage"
    }  ... ]
```

Tools for ARM

Visual Studio – Resource Group Project

VS Code – ARM Extension

Azure Portal

GitHub

ARM Summary

Native to Azure

View deployments in Azure Portal

Verbose

Variables enable naming standards

Parameters ease testing across environments

Bicep

Project Bicep

- ARM Transpiler, generates ARM as output
- Simpler syntax reduces complexity of ARM
- Modularity
- Support for all resource types and API versions
- A domain-specific-language for Azure
- No state or state files to manage
- No cost, open source

Bicep File

```
targetScope = '<scope>'

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

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

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

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

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

Example Bicep Parameters

```
@minlength(3)
@description('Application Name')
param appName string

@allowed([
  'eus'
  'wus'
  'cus'
])
@description('Location of Data Center')
param loc string
```

Example Bicep Variables

```
var prefix = '${loc}-poc-'
var hostName_var = '${prefix}${appName}-plan'
var siteName_var = '${prefix}${appName}-site'
```

Example Bicep Resources

```
resource host 'Microsoft.Web/serverfarms@2021-01-15' = {
  name: hostName
  location: resourceGroup().location
  sku: {
    name: 'F1'
  }
}
```

Get started with Bicep

- Decompile existing ARM templates
- Code Extension enables snippets to simplify dev
- Deployment via same calls as for ARM

Terraform

Terraform

Created by Hashicorp

HCL Language

Multi-cloud

Tool for versioning infrastructure

Uses state information for execution plan

Install is simple download of the executable and run

Install/setup Terraform

- Built in to the Azure Cloud Shell in the portal
- Download the executable from Terraform's site, copy exe to path
- Use Chocolaty installation

```
> choco install terraform
```

Workspace and Files

Create folder for workspace

Initialize terraform in folder

- Associates workspace with backend
- Loads necessary modules

Add template files *.tf and *.tfvar files

- main.tf, vars.tf, output.tf, etc.

Terraform Providers

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 2.0"
    }
  ...
}
provider "azurerm" { ...
}
```

Terraform Variables

```
variable "prefix" {
  type = string
  default = "dadapp"
}

variable "src" {
  type = list
  default =  ["azARM", "code", "bicep", "terraform"]
}
```

Terraform Resources

```
resource "azurerm_app_service_plan" "plan" {
  name                = "tf-${var.prefix}-plan"
  location            = "${azurerm_resource_group.main.location}"
  resource_group_name = "${azurerm_resource_group.main.name}"

 sku {
      tier = "Free"
      size = "F1"
  }
}
```

Terraform Commands

init	Initializes the environment
plan	Compares the template to the saved state and shows what will change if applied
apply	Runs the template
destroy	Removes what was created

Terraform Summary

HCL Language less noisy

Cross cloud support

Environmental testing code takes some thought

State management

Secrets

Consider how you will secure your state

Pulumi

Pulumi

A developer focused collection of packages and libraries that can be run from within a custom application to operate cloud APIs to create and manage infrastructure.

PROS

- Multiple languages and APIs
- Compiled into native runtime
- Developers don’t have to another language
- State and secret management
-

CONS

- Vendor managed running of api’s
- Need paid plan for CI/CD integration
- State is 3rd party to Azure

Ansible

Ansible module and version matrix for Azure Next steps Ansible is an **open-source product that automates cloud provisioning, configuration management, and application deployments**. Using Ansible you can provision virtual machines, containers, and network and complete cloud infrastructures.

PROS	CONS
Hybrid/cross cloud and on-premises	Lack of UI
Automation tool	No state
Python based	Linux/Python
YML	YML

IaC Considerations

- Learning Curve & Tools
- Variables and Parameters
- Preview
- History
- Modularity
- Looping
- Security
- Current

Comparison

Feature	ARM/Bicep	Terraform	Pulumi
Language	JSON + Bicep	HCL/DSL	Code Native, e.g. JavaScript, Python, C#...
Clouds	Azure only	Agnostic + on-prem	Agnostic + on-prem
State Files	Uses Azure Resource Manager natively	Plain-text	Encrypted
Naming standards	Variables & Parameters	Locals	Language native
Environments	Parameter files	Folder structure	Stacks
Preview Changes	az deployment … what-if	terraform plan	pulumi preview
Infrastructure Cleanup	No	terraform destroy	pulumi destroy
Deployment History	Yes – View in Portal	SCM, TF Cloud*	SCM, Pulumi Enterprise*
Code Reuse	Hosted JSON URIs	Modules + Registry	Code native packages, NPM

* refers to a premium feature from vendor, i.e. Terraform Cloud or Pulumi Enterprise
Source: https://julie.io/writing/arm-terraform-pulumi-infra-as-code

ARM

Overview

Azure native solution for infrastructure as code, provides idempotent declarative way to describe infrastructure shape and the ARM engine in Azure makes it so

PROS

- Native to Azure
- Works with the Portal
- Tooling is ok in VS, better in Code

CONS

- Verbose, hard to read
- Complex
- Cloud specific

Bicep

A domain specific transpiler for creating Azure ARM templates from a language that provides constructs for variables, looping, modules and scoped deployments

PROS

- Day 1 Current
- Easier to read and write thanks to tooling
- Output is ARM
- History is Azure Resource Manager native
-

CONS

- Newer to the field
- Cloud specific
-

Terraform

A popular cross-cloud tool for managing infrastructure by processing templates written in HCL into calls to management APIs, keeping track of state information describing cloud resources and services, available in Open Source and paid versions

PROS

- Declarative description of infrastructure
- HCL is easier to read, less clutter
- Works in multiple Cloud providers & on prem
- Broad adoption

CONS

- State management separate from cloud
- Changing between environments
- History/visibility of deployments
- Secrets stored in clear text in state

Pulumi

A developer focused collection of packages and libraries that can be run from within a custom application to operate cloud APIs to create and manage infrastructure.

PROS

- Multiple languages and APIs
- Compiled into native runtime
- Developers don’t have to another language
- State and secret management
-

CONS

- Vendor managed running of api’s
- Need paid plan for CI/CD integration
- State is 3rd party to Azure

Which is better? ChatGPT says…

Both Terraform and Bicep are popular Infrastructure as Code (IaC) solutions for Azure, but they have different strengths and use cases.

Terraform is a mature and widely adopted open-source tool that can be used to manage infrastructure across multiple cloud providers, including Azure. It supports Azure Resource Manager (ARM) templates and provides a way to create and manage resources using HashiCorp Configuration Language (HCL). It is known for its flexibility and ability to handle complex infrastructure setups.

Bicep, on the other hand, is a new open-source language from Microsoft, specifically designed to simplify the authoring of ARM templates. Bicep is built on top of ARM templates and provides a simpler and more readable syntax, making it easier to create and manage resources. It also provides a way to validate, test and debug the templates before deploying.

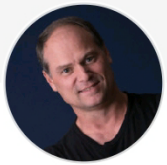
Both Terraform and Bicep have their own strengths and weaknesses, and the best one will depend on your specific requirements and use case. Terraform is a more mature and widely adopted tool, but Bicep is better for those who prefer a simpler and more readable language. If you are already familiar with ARM templates, then Bicep could be a great choice as it can make your templates more readable, and easier to maintain. But if you are looking for more flexibility and more powerful features, then Terraform might be a better choice.

Conclusion

Terraform is a powerful and popular tool for IaC that supports multi-cloud deployments, but you have to be careful with your state and secrets

Bicep is always current, uses Azure Resource Manager for State, secures secrets by default, and is a powerful domain specific language for IaC with Azure with great tooling

Understanding/architecting an effective **cloud infrastructure** still requires thought, vision and execution, but you have to decide on your priorities.



Mike Benkovich
Enterprise Cloud Architect,
Consultant, Developer Tools Ev...

Call to Action...
Where can I get more info?

Give me feedback on LinkedIn
(Scan the QR Code to the left)

Visit my blog **www.benkotips.com**

Azure Office Hour Fridays!
https://bit.ly/BnkAzHrs

Try it out with **low hanging fruit**