IaC Forged in Code
Bicep, Terraform & Pulumi

Mike Benkovich
Cloud Engineer
mike@benko.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

https://askdad.benkotips.com

No plan...

Building Without Blueprints

WE CLICK, WE PATCH, WE PRAY...

NO PLAN, NO GUARANTEES

OUR CURRENT 'BUILD BY HAND' APPROACH IS FRAGILE

Under Construction!

- Crews digging without a traffic plan
- Chaos, delays, and wasted resources
- Reflects manual cloud provisioning

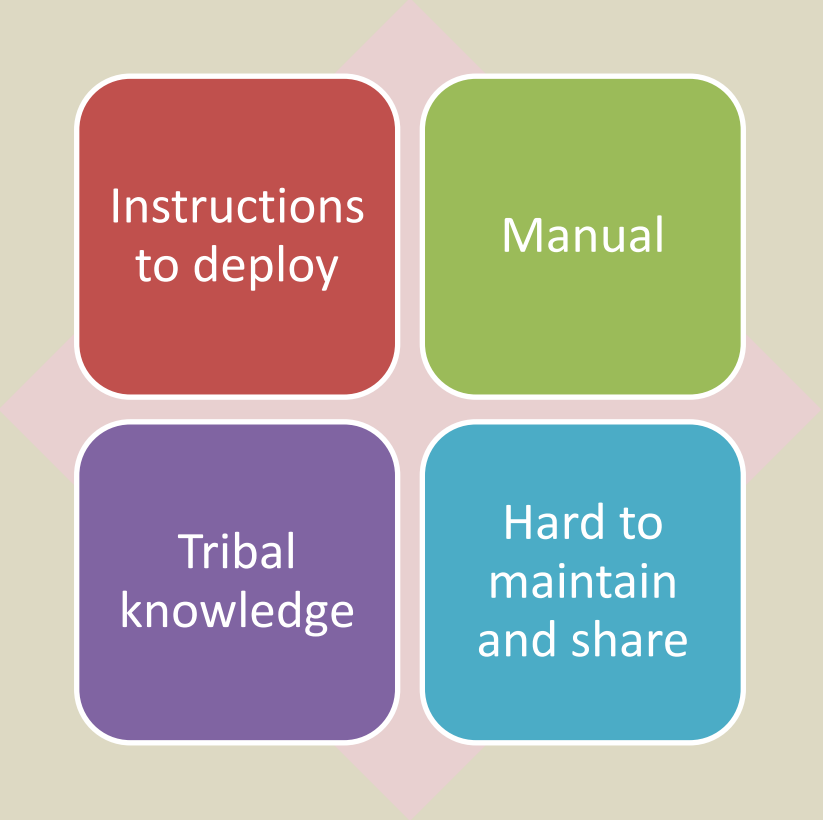
Living in Code, Not the Portal

Portal-only pitfalls

Snowflake servers

Runaway permissions

Infrastructure as docx



Use Azure Functions to create a function that conn Azure services

This topic shows you how to create a function in Azure Functions that listens to messages on an Azure Sto messages to rows in an Azure Storage table. A timer triggered function is used to load messages into the reads from the queue and writes messages to the table. Both the queue and the table are created for you b on the binding definitions.

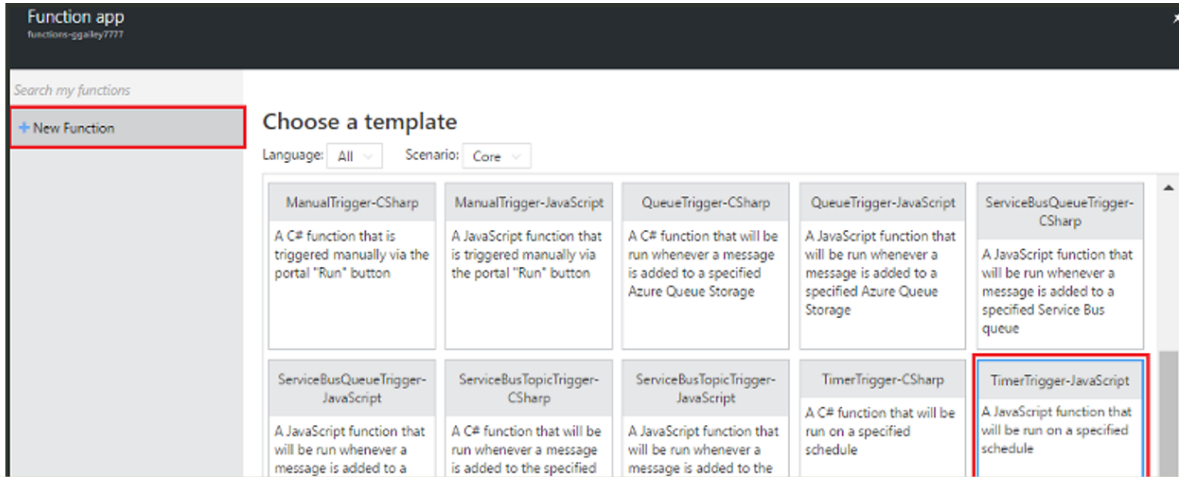
To make things more interesting, one function is written in JavaScript and the other is written in C# scrip a function app can have functions in various languages.

You can see this scenario demonstrated in a video on Channel 9.

Create a function that writes to the queue

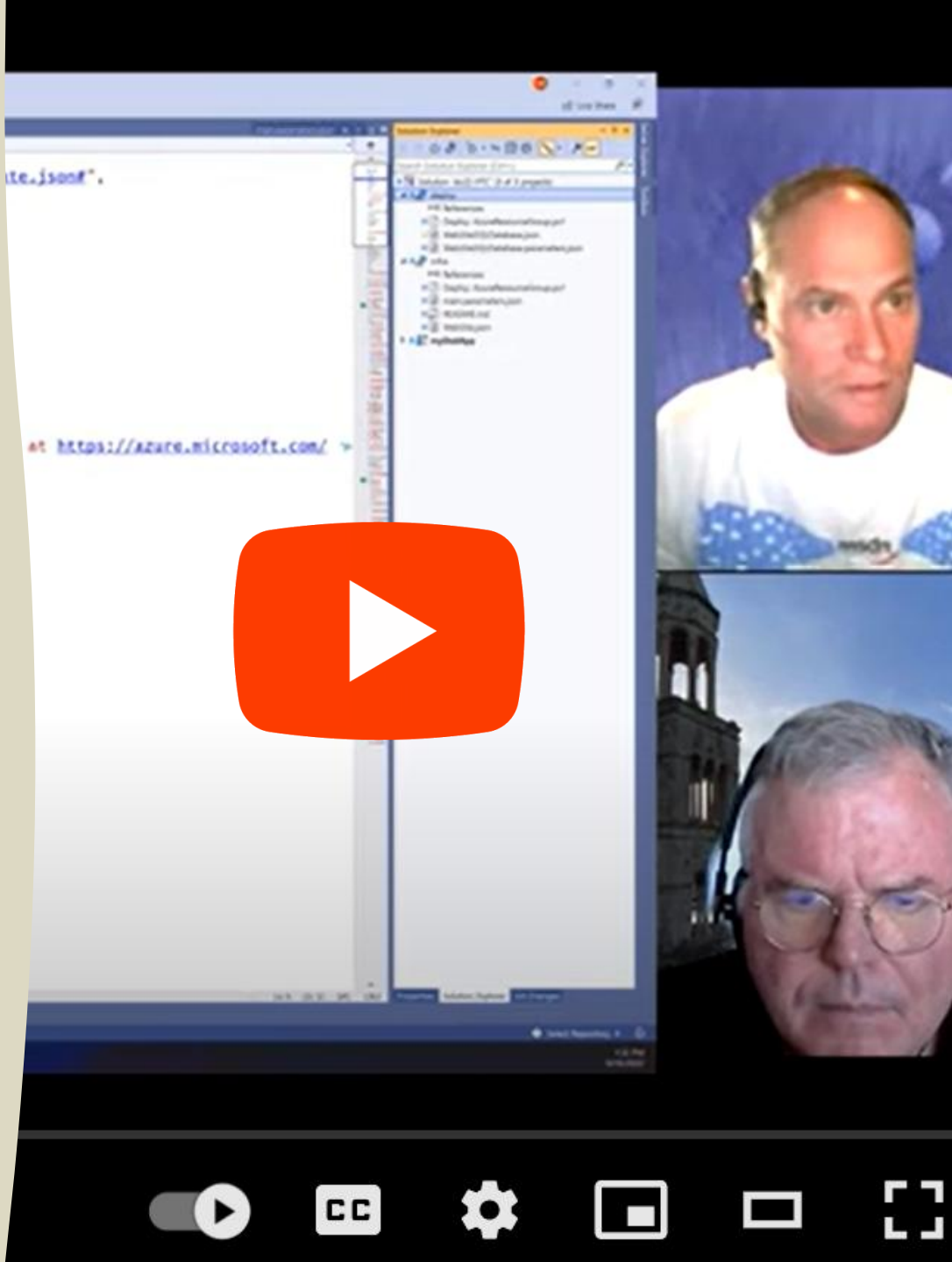
Before you can connect to a storage queue, you need to create a function that loads the message queue. uses a timer trigger that writes a message to the queue every 10 seconds. If you don't already have an Azu Try Azure Functions experience, or create your free Azure account.

1. Go to the Azure portal and locate your function app.
2. Click **New Function** > **TimerTrigger-JavaScript**.
3. Name the function **FunctionsBindingsDemo1**, enter a cron expression value of `0/10 * * * *` click **Create**.



Infrastructure as Video

- Show and Tell
- Click to Deploy
- Works Today

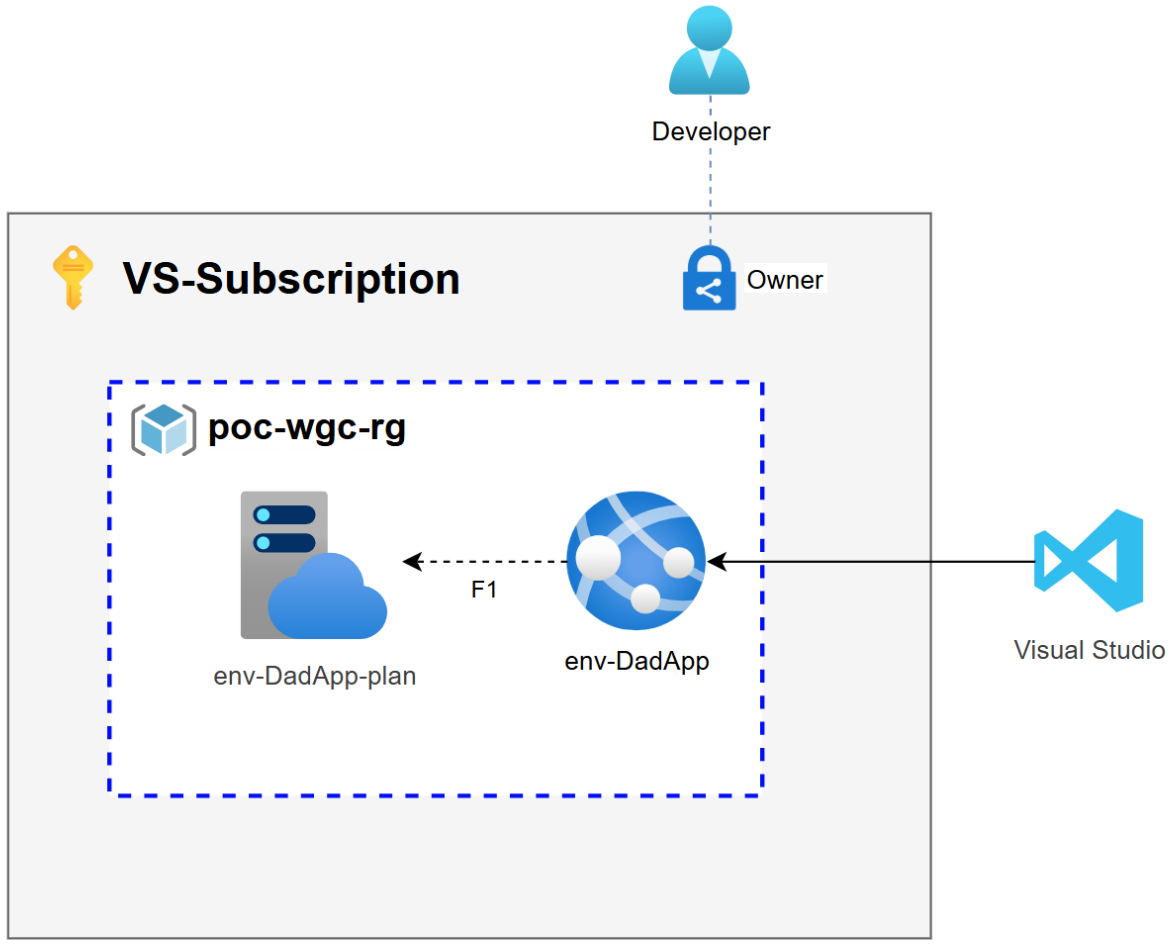


Proof of Concept (POC)

It Works!

Deploy from IDE

Manage in Portal



WARNING!

CLICKED TO DEPLOY

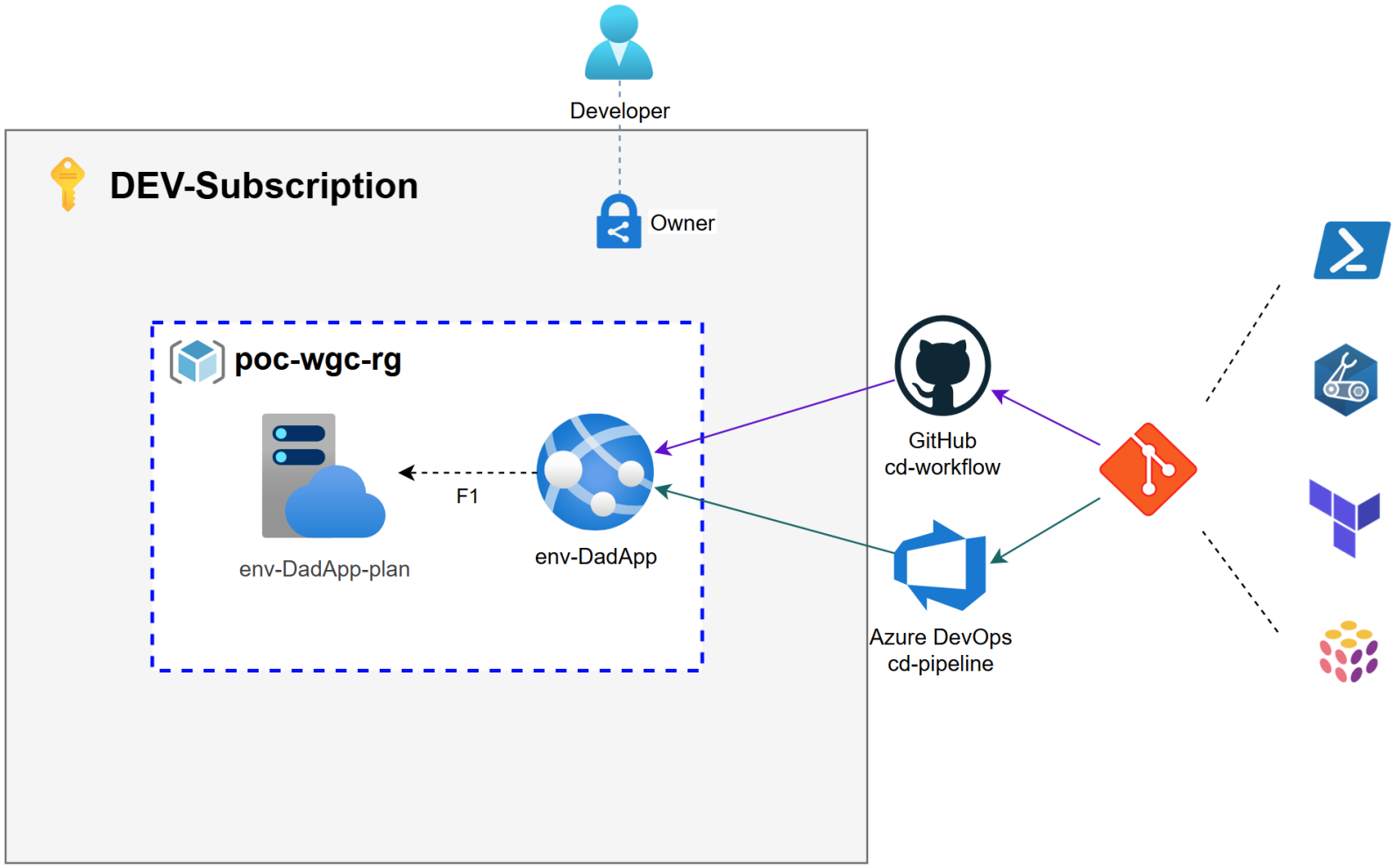
Permit to Build

Dev Pattern

Git Repo

IaC

DevOps CI/CD



The Moment You Go Code → Trust

“The moment you treat infrastructure as code, you swap guesswork for guarantees.”

Infrastructure as Code (IaC)

- Declarative
- Versioned & managed
- Repeatable
- Descriptive vs Procedural
- Environment drift
- Idempotent
- Current

main.bicep M

deploy > Bicep > main.bicep > ...

```
@description('Specifies the location for resources.')
param location string = 'centralus'

param appName string
param envName string
param color string
@secure()
param secretValue string

targetScope = 'subscription'

resource rg 'Microsoft.Resources/resourceGroups@2021-04-01' = {
  name: 'bnk-${appName}-${envName}-rg'
  location: location
}

module site 'mywebsite.bicep' = {
  scope: resourceGroup(rg.name)
  name:  deployment().name
  params: {
    appName: appName
    envName: envName
    secretValue: secretValue
    color: color
  }
}

output rgName string = rg.name
```

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 CLI

Azure CLI

Azure Command Line Interface

Extensible & updateable

Cross Platform, Linux, Windows, MacOS, Cloud Shell

Always up to date

Procedural

Azure CLI

```
# Deploy via Azure CLI
$appName = "IaC25"
$envName = "cli"
$location = "westeurope"

$rgName = "$appName-$envName-rg"
$planName = "$appName-$envName-plan"
$siteName = "$appName-$envName-site"

az group create --name $rgName --location $location

az appservice plan create --name $planName --resource-group $rgName `
  --location $location --sku F1

az webapp create --name $siteName --resource-group $rgName `
  --plan $planName --location $location
```

Azure CLI

Great to interrogate status of things

Good for basic scenarios

Proof of concept

Visibility of things

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"
    }  ... ]
```

ARM Example: App Service Plan

```
"resources": [
 {
   "name": "[variables('planName')]",
   "type": "Microsoft.Web/serverfarms",
   "apiVersion": "2023-01-01",
   "location": "[resourceGroup().location]",
   "sku": {
       "name": "F1",
       "capacity": 1
    },
  },
... ]
```

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'
```

Bicep Example: App Service Plan

```
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, provider based

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 locals

```
locals {
  rg_name = "${var.env}-${var.appName}-rg"
  prefix  = "${var.env}-${var.loc}-${var.app_name}"

  host_name = "${local.prefix}-web-${var.index}"
  site_name = "${local.prefix}-plan-${var.index}"
  keyvault_name = "${local.prefix}-kv-${var.index}"
}
```

Terraform Example: App Service Plan

```
resource "azurerm_app_service_plan" "plan" {
  name                = local.host_name
  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

Terraform AzAPI

AzAPI Terraform

Run via REST endpoints in Azure

Apply Terraform via setting body content

AzApi Resource Provider

```
# App Service Plan - Using AzAPI provider
resource "azapi_resource" "app_service_plan" {

 type      = "Microsoft.Web/serverfarms@2023-01-01"
 name      = local.plan_name
 parent_id = azurerm_resource_group.rg.id
 location  = var.location

 body = jsonencode({
   sku = {
     name     = "F1"
     capacity = 1
   }
   properties = {}
 })

 response_export_values = ["id"]
}
```

Deploy using **az rest** command

```
# App Service Plan - Using AzAPI provider

$planId = az rest --method put `
    --uri "https://management.azure.com/subscriptions/`
           $(az account show --query id -o tsv)`
           /resourceGroups/$resourceGroupName/providers`
           /Microsoft.Web/serverfarms/$planName`
           ?api-version=2023-01-01" `
    --body $planBody `
    --query id `
    --output tsv
```

Pulumi

Pulumi

Multi-language support
(Python, JavaScript, Go, C#, etc)

Supports Multiple Cloud Providers

Modern structured coding practices

Pulumi

Supports multiple languages and APIs

Compiles into native runtime

Developers don’t have to learn yet another language

State and secret managed for you

Pulumi C# Example: App Service Plan

```
var planName = $"{appName}-{envName}-plan";

var appServicePlan = new AppServicePlan(
            planName, new AppServicePlanArgs
 {
     ResourceGroupName = resourceGroup.Name,
     Kind = "App",
     Sku = new SkuDescriptionArgs
     {
         Tier = "Basic",
         Name = "B1",
     },
  });
```

Pulumi C# Example: App Service Plan

```
var planName = $"{appName}-{envName}-plan";

var appServicePlan = new AppServicePlan(
            planName, new AppServicePlanArgs
 {
     ResourceGroupName = resourceGroup.Name,
     Kind = "App",
     Sku = new SkuDescriptionArgs
     {
         Tier = "Basic",
         Name = "B1",
     },
  });
```

Pulumi Go Example: Resource Group

```
func main() {
   pulumi.Run(func(ctx *pulumi.Context) error {
       appName := "azIaC"
       envName := "pulumi"
       rgName := appName + "-" + envName + "-rg"
       planName := appName + "-" + envName + "-plan"
       siteName := appName + "-" + envName + "-site"

       // Create an Azure Resource Group
        resourceGroup, err :=
           resources.NewResourceGroup(ctx, rgName, nil)

        // Create an App Service Plan
        appServicePlan, err := web.NewAppServicePlan(ctx, planName, &web.AppServicePlanArgs{
            ResourceGroupName: resourceGroup.Name,
            Kind:              pulumi.String("App"),
            Sku: &web.SkuDescriptionArgs{
                Tier: pulumi.String("Basic"),
                Name: pulumi.String("B1"),
            },
         })
       ...
```

Pulumi

- Learning Curve
- Complexity
- State Management by 3rd party
- Cost

Get Started

Ceate an account on https://pulumi.com

Download and install

Use Chocolaty installation

```
> choco install pulumi
```

Modules

Modules

Terraform

```
module "naming" {
  source  = "../tf-naming-module"
  appName = var.appName
  envName = var.envName
}
locals {
  # Add deployment-specific tag
  resource_tags = merge(
    module.naming.common_tags,
    {
      Deployment = basename(path.cwd)
    }
  )
}
```

Bicep

```
module naming 'br:imacoreacr.azurecr.io/bicep
   /modules/naming:latest' = {
  name: 'naming'
  params: {
    envName: envName
    appName: appName
  }
}
```

Azure Verified Modules

Opinionated, governance derived reusable code

Standardized Patterns

Cross language support

Maintained and Tested

https://github.com/Azure/Azure-Verified-Modules

```
module mod 'br:mcr.microsoft.com/bicep/…'
```

https://github.com/azure/azure-verified-modules

Azure / Azure-Verified-Modules Public

Notifications | Fork 107 | Star 440

Code | Issues 309 | Pull requests 8 | Discussions | Actions | Projects 3 | Security | Insights

Files

main

Go to file

- .devcontainer
- .github
- .vscode
- docs
- utilities
- .editorconfig
- .gitattributes
- .gitignore
- CODE_OF_CONDUCT.md
- LICENSE
- Makefile
- README.md
- SECURITY.md

Azure-Verified-Modules / README.md

NickAzureDevops fix broken image on Readme (#2006) 9bdc709 · 2 weeks ago History

Preview | Code | Blame 42 lines (28 loc) · 3.59 KB · Raw

> **Please note** that our documentation is published over at aka.ms/AVM. Please visit this site for more information and guidance!
>
> This repository is used for proposing and tracking the state of modules, tracking issues and feature requests as well as hosting documentation for the Azure Verified Modules (AVM) project. If you are looking for the AVM code repositories, please visit the Bicep and Terraform module indexes on the AVM portal for references.

Azure Verified Modules (AVM)

Welcome to the Azure Verified Modules (AVM) repository!

Azure Verified Modules (AVM), as "One Microsoft", we want to provide and define the single definition of what a good IaC module is;

- How they should be constructed and built
  - Enforcing consistency and testing where possible

Considerations

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

Cattle, Not Pets

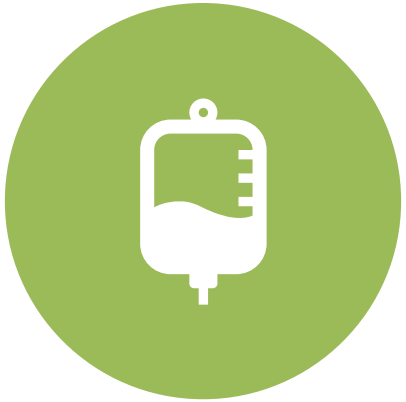
Immutable

Replace, don't repair

Consistent behavior

Environments on Demand

DEV, TEST, PROD PARITY



EPHEMERAL SANDBOXES

COST SAVINGS

Discovery & Naming

Naming standards

Resource Graph queries

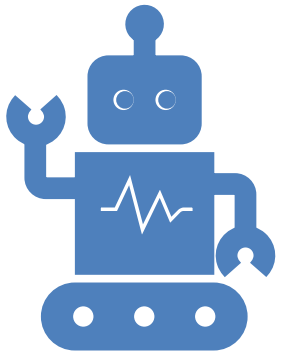
Organization & Tagging

Policy-enforced tags

Subscription models

Automation & Policy

Azure Policy as code



Automated remediations

Permit to Cloud: Runway Checklist

- Blueprint factory and module registry
- CI/CD pipelines for infra + apps
- Guardrails: policies & tests
- Monitoring and self-healing

Comparing

- Az CLI
- ARM & Bicep
- Terraform
- AzApi
- Pulumi
- AVM

Azure CLI

Easy way to get started with Azure. Used to work with Azure accounts, services and resources. Replaced PowerShell CMDlets with

PROS

- Low barrier of entry
- Scriptable
- Procedural
- Imperative
- Repeatable

CONS

- Complexity of services
- Looping logic
- Imperative
- No what-if
- Azure specific

ARM

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

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

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

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

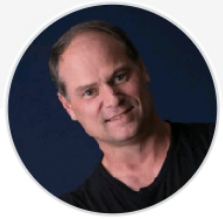
Terraform is a powerful and popular tool for IaC that supports multi-cloud deployments, but you have to be careful with your state and secrets

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.

Choose Stone Over Sand

- • Build fortresses in code
- • Trust, repeatability, resilience
- • Start your IaC journey today



Mike Benkovich

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

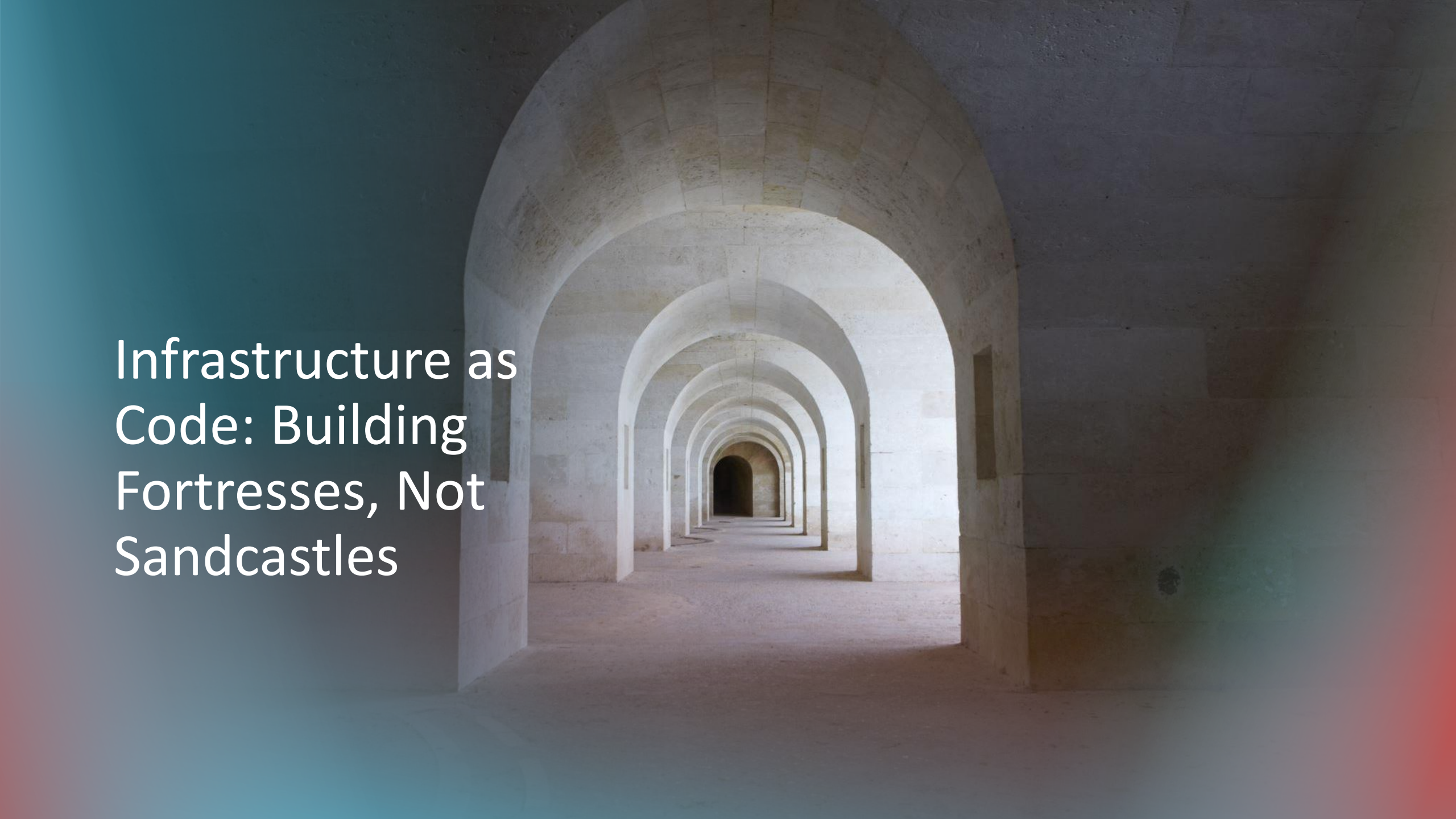
Call to Action...
Where can I get more info?

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**



Infrastructure as Code: Building Fortresses, Not Sandcastles

Repeatability…
Recovery ⇒ Resilience

State Management

- Working with same code in different environments
- Securing it is your responsibility
- Updating/fixing if something goes wrong
- How to import external changes

Environments

In Bicep/ARM it’s parameter files

Terraform workspaces…

Importance of Parameters

- Regional and subscription settings
- Naming consistency
- Reusability

Source of Truth: Git & GitOps

- Single pane of glass
- Pull requests for infra
- Audit and traceability

State Management

- • Remote backends
- • Locking and drift detection
- • Ensures consistent deployments

Az CLI

- Azure CLI / az

- Quick but per-machine drift

- Good for ad-hoc tasks

Terraform

- Multi-cloud consistency

- Modules and versioning

- Community ecosystem

ARM / Bicep

- Azure-native features
- First-class support
- Declarative JSON/Terraform alternative

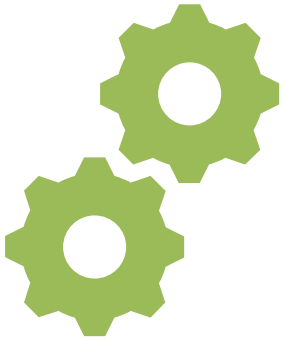
AzApi & Pulumi

AzApi for bleeding-edge resources

Pulumi: infrastructure in C#/TS/Python

IaC Tooling: Others

Ansible, Chef, Crossplane



Hybrid and config management

Governance Imperative

Cost of Ownership

SURPRISE BILLS

ORPHANED RESOURCES

TAGGING FOR CHARGEBACK

Case Study: American Airlines

- 80 executables, 125+ servers, 27,000 tests

- Manual cycles: months → minutes

- IaC + DevOps = continuous releases

Case Study: Thomson Reuters

- Monolithic Hadoop clusters

- EMR + CloudFormation pipelines

- 24h data-refresh → 1h automated workflows

Case Study: Flavorus Ticketing

- 150,000 tickets in 10s

- 550 Azure SQL DBs spun up via scripts

- On-demand IaC automation