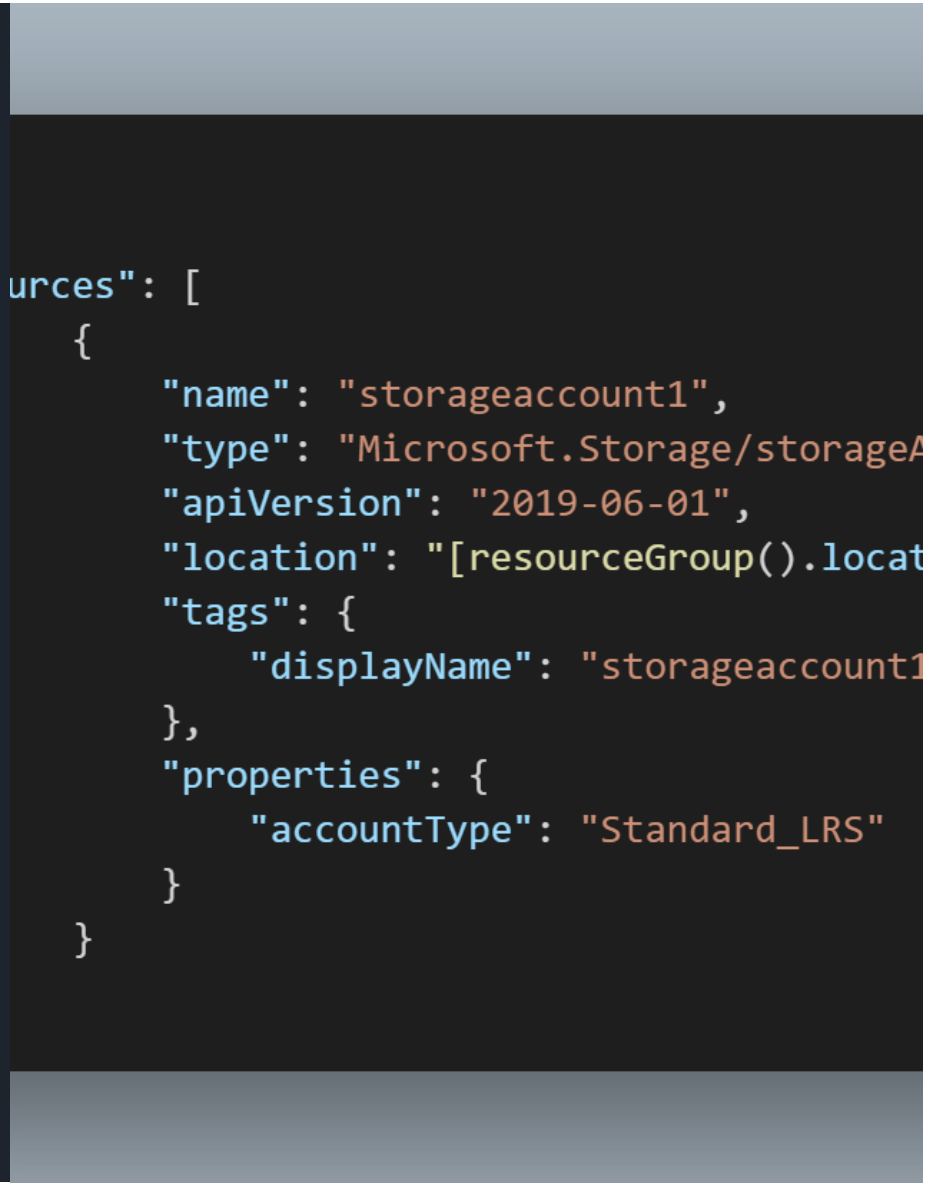
Deploying Resources to Azure using ARM Templates

Israel Orenuga

Cloudville IaC series

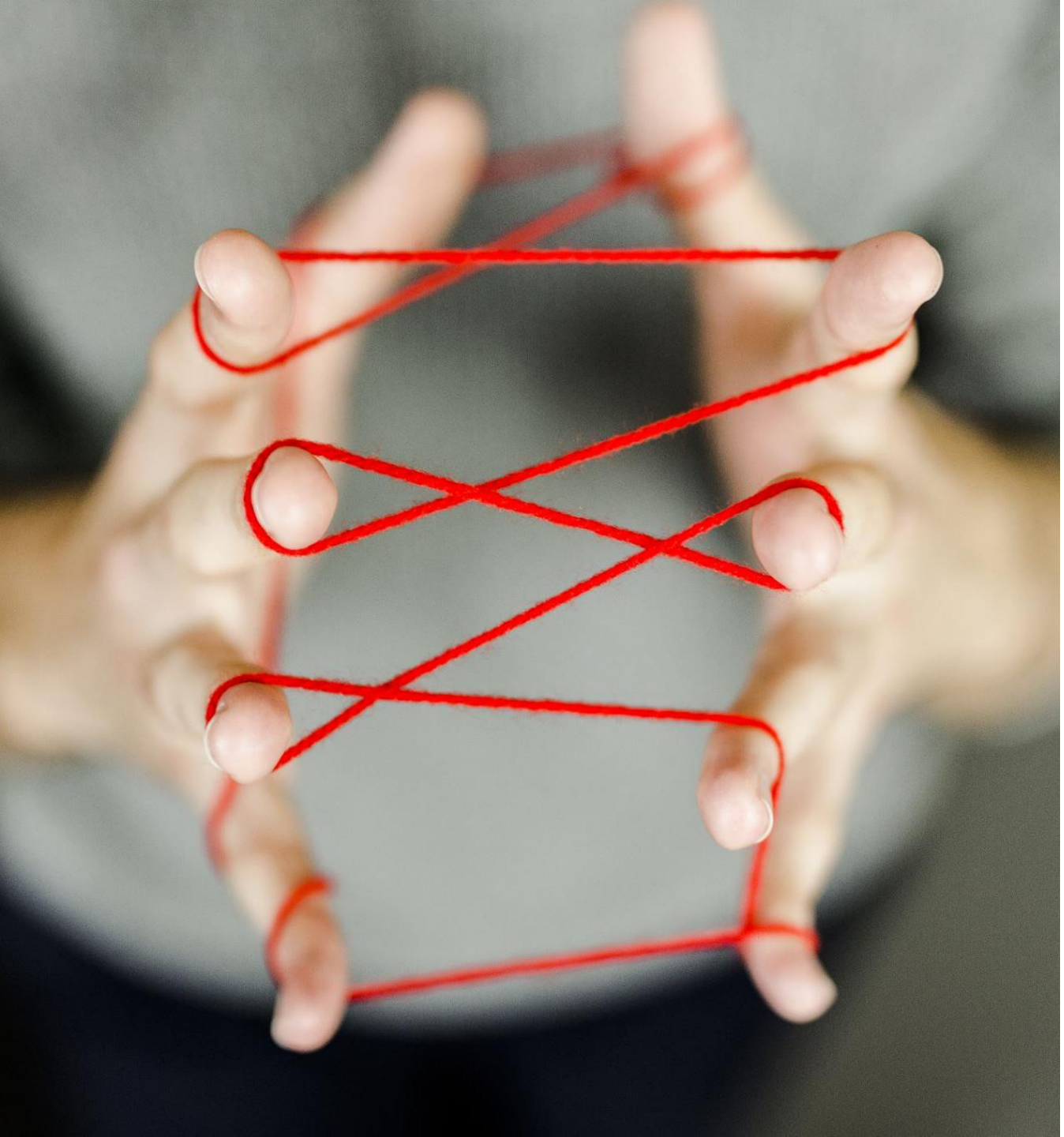


Objectives

- Simple Explanation of IaC
- Gentle Intro to IaC using ARM Templates
- Explain the structure of an ARM Template and its different sections
- Demos

Infrastructure as code?

- Representing resources, or a whole environment as code.
- Allows for:
  - Consistency: Define once, Deploy Many times but same result each time. **(Idempotence)**
  - Efficiency: Several resources can be deployed at once.
  - Auditability (Version Control)
- Common IaC tools/platforms include:
  - ARM Templates,
  - Terraform,
  - Pulumi

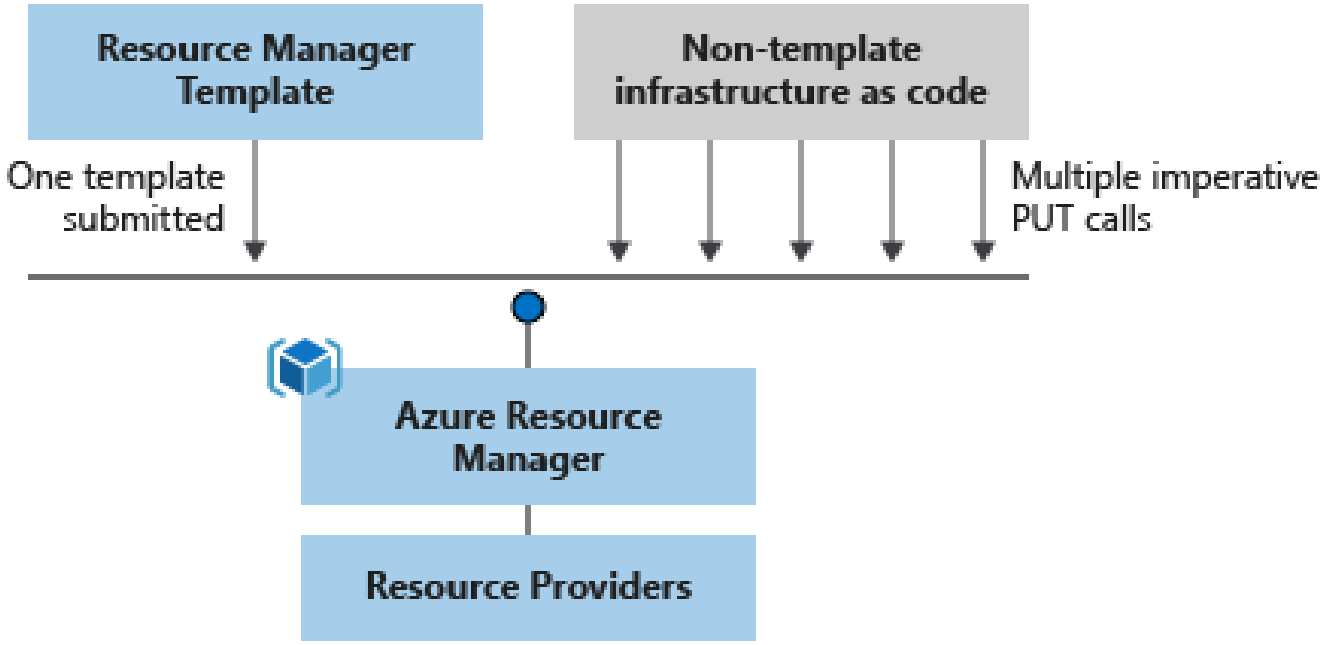


ARM Templates

- ARM templates are JavaScript Object Notation (JSON) files that define the infrastructure and configuration for your deployment
- Declarative syntax: I want x. I don't really care how you'll do it, just get me x. (versus imperative: do a before b, then do c,d,e,f up to w so that I can have x). **No need to define specific steps.**

ARM Templates vs "Scripts"

- Why don't I just create infrastructure using powershell scripts or something similar?



Element	Description
schema	A required section that defines the location of the JSON schema file that describes the structure of JSON data. The version number you use depends on the scope of the deployment and your JSON editor.
contentVersion	A required section that defines the version of your template (such as 1.0.0.0). You can use this value to document significant changes in your template to ensure you're deploying the right template.
apiProfile	An optional section that defines a collection of API versions for resource types. You can use this value to avoid having to specify API versions for each resource in the template.
parameters	An optional section where you define values that are provided during deployment. These values can be provided by a parameter file, by command-line parameters, or in the Azure portal.
variables	An optional section where you define values that are used to simplify template language expressions.
functions	An optional section where you can define user-defined functions that are available within the template. User-defined functions can simplify your template when complicated expressions are used repeatedly in your template.
resources	A required section that defines the actual items you want to deploy or update in a resource group or a subscription.
output	An optional section where you specify the values that will be returned at the end of the deployment.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    Select or create a parameter file...
    "parameters": {},
    "variables": {},
    "resources": [],
    "outputs": {}
}
```

STRUCTURE OF AN ARM TEMPLATE

https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/syntax

Resources… "The Koko"

- Be familiar with the resource(s) you want to deploy
- Know its **Resource Provider** and find the resource (type) through it.
  - Every resource falls under a provider. Usual format is "Microsoft.xxxxxx" e.g., Microsoft.Storage, Microsoft.Compute etc.
  - For example, a storage account is a resource type under the Microsoft.Storage resource provider

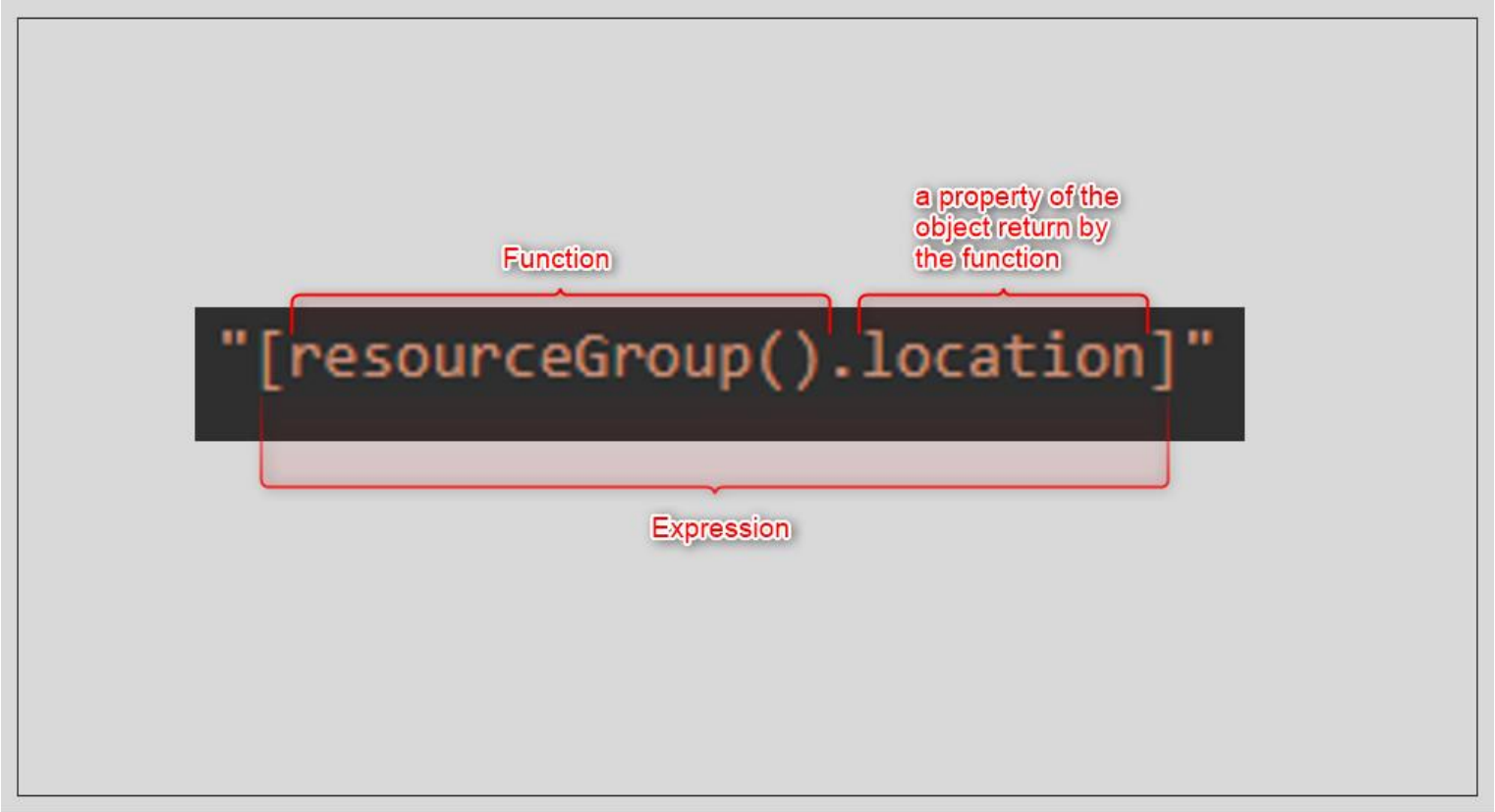
Importer, Exporter…

- Not every time starting templates from scratch…
  - Sometimes import from other sources or export a template from the portal…
  - No time, seriously…
  - Quickstart Templates: https://github.com/Azure/azure-quickstart-templates
- You can easily find the references for the Resource Provider and most Azure Resources using the following links:
  - Resource Providers: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/azure-services-resource-providers
  - Azure Resources: https://docs.microsoft.com/azure/templates
- **If you're not using VSCode you are on a looong thing!!**

Demo Time!!

Expressions

- Dynamically executed code
- Easily recognizable by the square brackets []
- Usually returns a string, integer, Boolean, array or object.
- Commonly contains functions



Template Functions

- Functions are used to compute and return a value for use within an ARM Template
- Most Functions are "built-in"
- Format: functionName(arg1,arg2,arg3) **Note: Arguments are optional.**
- Common Template Functions include:
  - Parameter()
  - Variable()
  - Concat()
  - UniqueString()
  - ResourceId()
  - ResourceGroup()
  - Reference()
  - SubscriptionId()
- Functions can also be user defined.
- Read through (at least once) the template functions available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions

More Demo!!

Bicep: Sneak Peek…

- Bicep was built on the weakness of JSON templates (i.e., poor readability & steep learning curve)
- Bicep is simpler, cleaner, intuitive.
- **JSON before BICEP**: An understanding of JSON ARM Templates significantly accelerates your learning/understanding of Bicep.
- Bicep > JSON ARM Template > Azure Resource Manager
  - In the background, Bicep templates get converted to JSON ARM templates before being sent to Azure Resource Manager for execution.
- Bicep Playground: Lets you convert existing JSON templates to Bicep templates. (Warning: Not 100% effective)

Bicep JSON

JSON Copy

```json
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]"
    },
    "storageAccountName": {
      "type": "string",
      "defaultValue": "[format('toylaunch{0}', uniqueString(resourceGroup().id))]"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2021-06-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {
        "accessTier": "Hot"
      }
    }
  ]
}
```

Bicep JSON

Bicep Copy

```
param location string = resourceGroup().location
param storageAccountName string = 'toylaunch${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

JSON vs BICEP

(NOTE: BOTH ARE FOR THE SAME RESOURCE)

Questions?

What is all this "Abracadabra"?