Microsoft

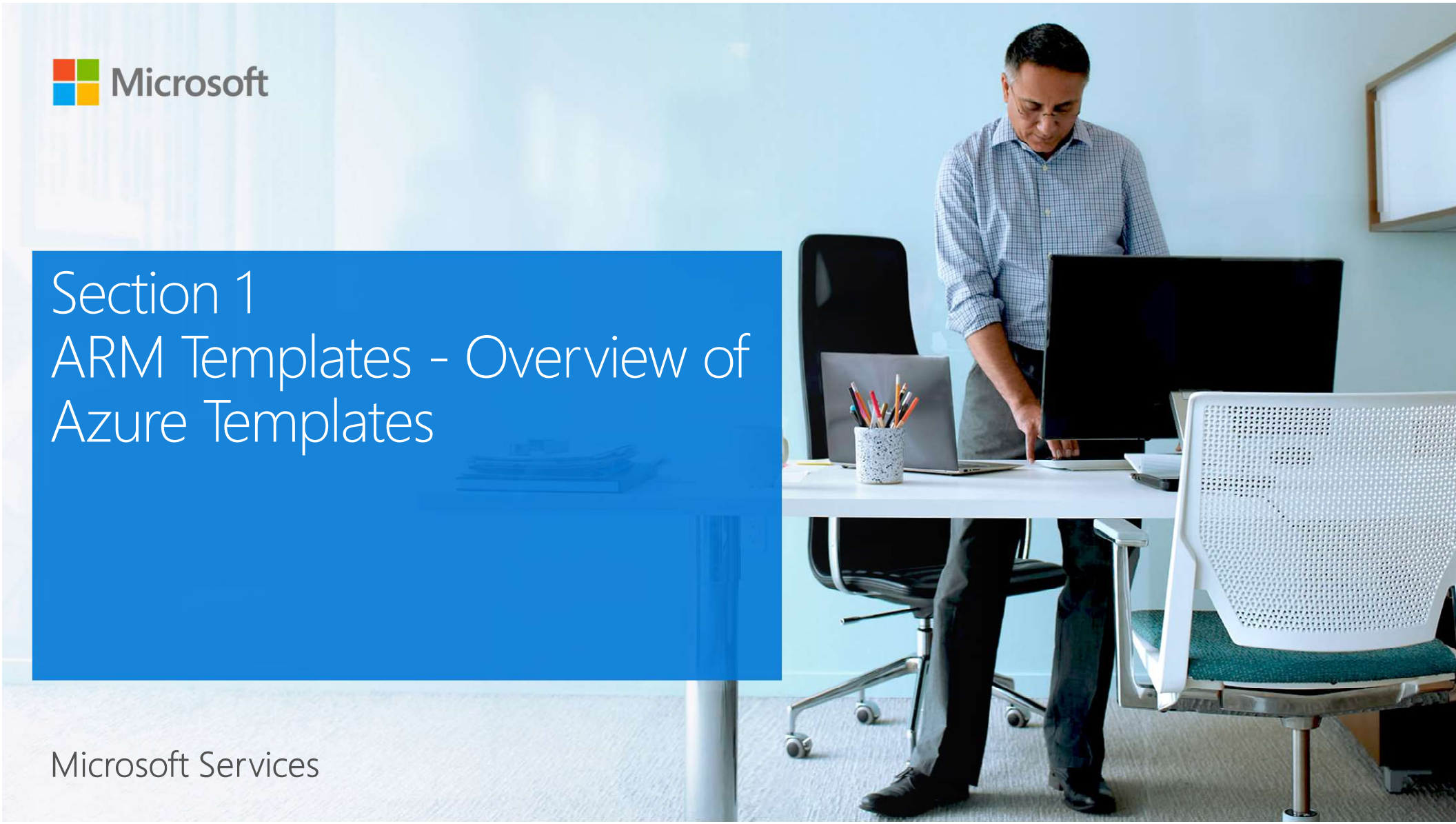
WorkshopPLUS – Azure: Infrastructure as Code

Module 4 – ARM Templates

Microsoft Services

Microsoft

Section 1
ARM Templates - Overview of Azure Templates



Microsoft Services

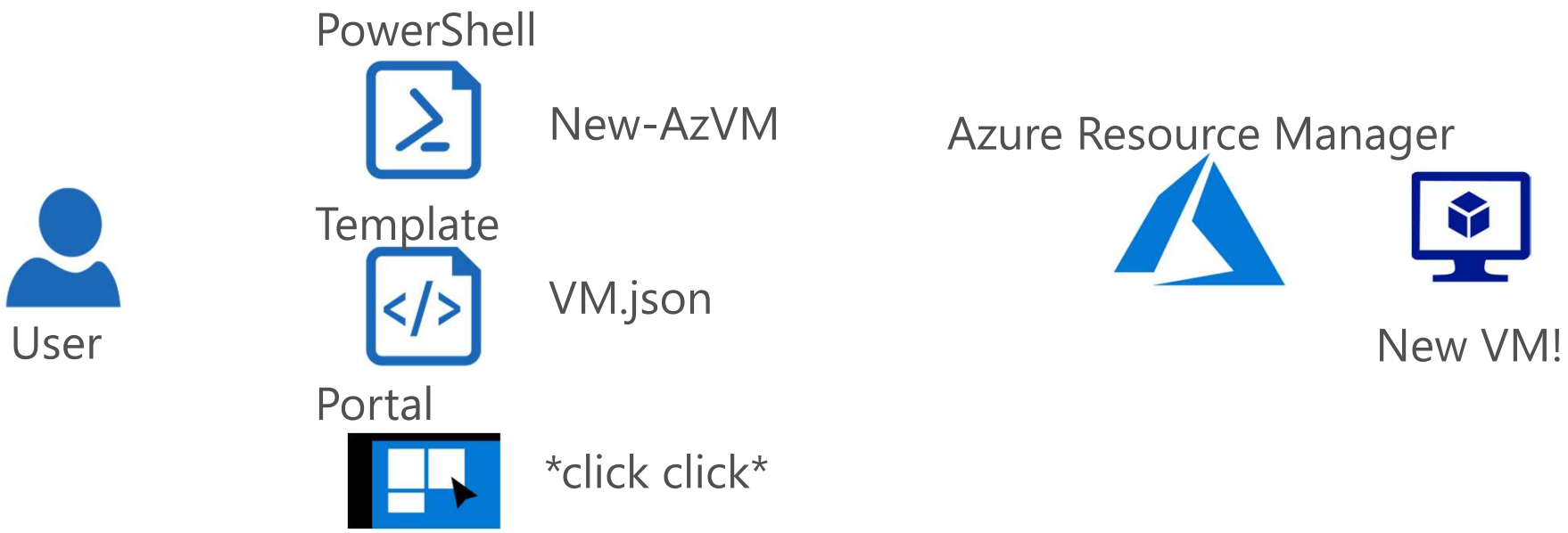
Objectives

After completing this learning unit, you will

- Be able to describe what a template is
- The sections of template architecture
- Differences in deployment mechanisms
- Template features

Instructing Azure to build your scenario

- Portal, PowerShell, & Templates achieve the same results
- How to achieve same results wildly varies by tool



Scenario considerations for deciding on the tool

- How many of the same resources are being deployed
- Frequency of updating those resources
- Remediation methods if deployment fails
- Familiarity with resource features
- Ease of destroy & re-deploying resources

Tool comparison for deployment/administration

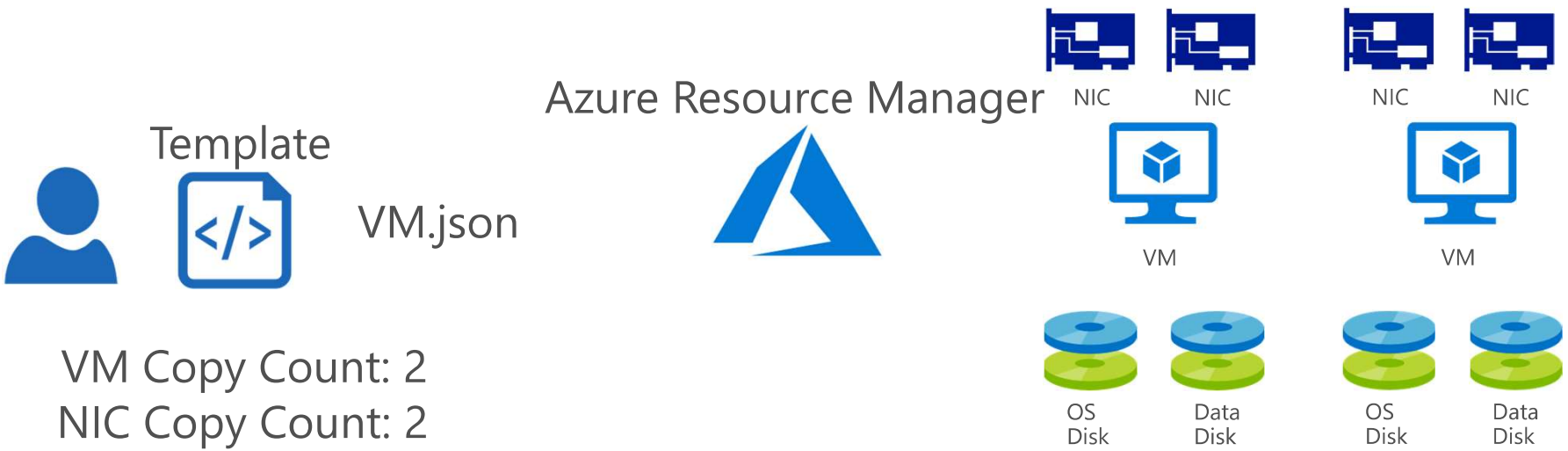
Task / Operation	Portal	PowerShell/CLI	Templates
Learning new resources	Easy, point and click	Moderate, minimal help	Hard, exact syntax
Built-in wizard	Yes, all resources	Minimal	No
Azure documentation	Frequent examples	Less frequent examples	Least frequent examples
Update resources	Adhoc only	Adhoc only	Yes, original code req.
Create multiples of resource	No	No, scripting required	Yes, native capability
Create multiple resource types	Yes, wizard only	No, scripting required	Yes, native capability
Set deployment order	No, wizard only	No, scripting required	Yes, native capability
Resume failed deployments	No, re-deploy only	No, scripting required	Yes, re-push code
Version-control	No	Yes, since it's code	Yes, since it's code
Ability to teardown & re-deploy	No	No	Yes

What are Azure Templates?

- Written in industry standard JavaScript Object Notation (JSON)
- Declarative syntax = Describes the end state
  I want 3 VMs configured a specific way; then I want 1 VM configured another way
- Integrates with Azure's existing governance structure
  The Identity pushing the code is fenced by role-based access control, Policy, etc.
- Works with common source code repository solutions
- Azure Resource Manager (ARM) orchestrates the instructions
  Example tools that push templates to ARM: Portal, PowerShell, REST API

Template advantages – Iterative/Updatable

- Template updates can be pushed to update environment



VM Copy Count: 2
NIC Copy Count: 2
Data Disk Count: 1

- Remember: The template describes the desired end state

Template advantages – Modular or Specific

- Templates can be flexible & modular or fixed & specific
  - Modular example: generic VM tier whose configuration is decided at deployment
    Great for re-using framework, but authoring the parameter input takes longer
  - Modular real life: X count of VMs with Y count of disks on Z virtual network
  - Specific example: Solution describes the full end to end server-side environment
    Great for deploying a replica of an end state with faster turnaround
  - Specific real life: Deploy a SharePoint solution in many environments
- All resources in a template are deployed into the same resource group or subscription

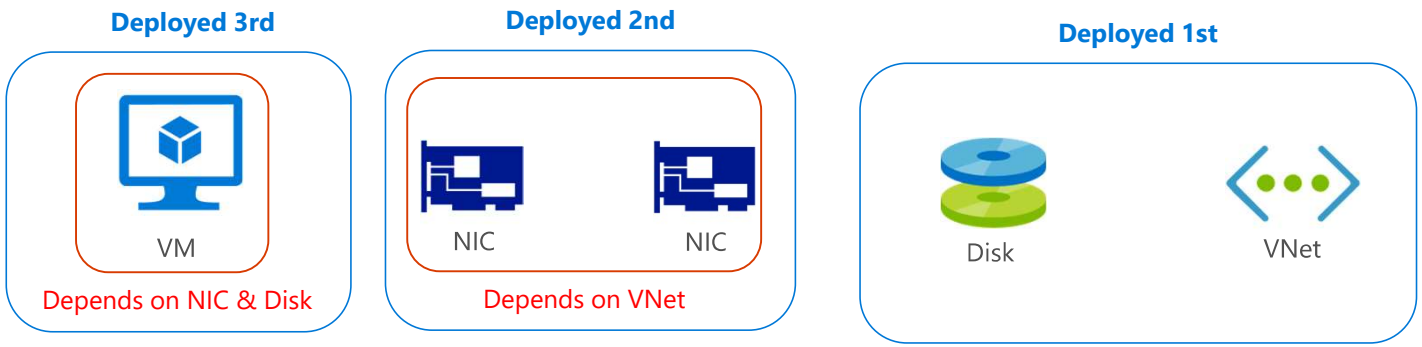
Templates – Deployment order / Dependencies

- No need to write deployment order as serial code

  PowerShell serial code: **T**op of code-first to deploy; **B**ottom of code-last to deploy
- Code dictates deployment order by stating dependencies
  - Template example: VM depends on the NIC which depends on the Vnet
  - Only resources from the same template should be defined as dependencies
- All Resources are deployed at the same time

  Dependencies are deployed first in the “at the same time” parallel manner

VM.json



Template advantages – Teardown

- Deployment options can state to remove anything existing that's not in the template at the scope it's deployed to

  Great for removing previous builds

- Remember: This is a destructive action, use carefully

Template Structure

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
```

States if you're performing a resource group or a subscription scope of deployment

```
    "contentVersion": "",
```

Your version of the template, it has no bearing on the actual deployment

```
    "parameters": {  },
```

The only values that are end user provided which customize the deployment on execution

```
    "variables": {  },
```

Constructed values that can reduce complex expressions

```
    "functions": [  ],
```

User-defined functions that are available as expressions within the template.

```
    "resources": [  ],
```

Resource types that are deployed or updated in a resource group or subscription

```
    "outputs": {  }
```

Values that are returned after deployment

```
}
```

Template Structure - Parameters

```
"parameters": {
    "<parameter-name>" : {
        "type" : <string, securestring, int, bool, object, secureObject, array>
        "defaultValue": "<default-value-of-parameter>",
        "allowedValues": [ "<array-of-allowed-values>" ],
        "minValue": <minimum-value-for-int>,
        "maxValue": <maximum-value-for-int>,
        "minLength": <minimum-length-for-string-or-array>,
        "maxLength": <maximum-length-for-string-or-array-parameters>,
        "metadata": {
            "description": "<description-of-the parameter>"
        }
    }
}
```

Template Structure – Parameters - Example

```
"parameters": {
        "AzureRegion": {
            "type": "string",
            "defaultValue": "EastUS",
            "allowedValues": ["EastUS", "WestUS", "CentralUS"],
            "metadata": {
                "description": "Azure region to deploy to"
            }
        },
        "VMCount": {
            "type": "int",
            "defaultValue": "1",
            "maxValue": 5,
            "metadata": {
                "description": "Number of VMs to deploy"
            }
        }
    },
```

Template Structure – Variables - Example

```
"variables": {
      "GeneralName": "[concat(parameters('Department'), parameters('Application'))]",
      "VMName": "[toLower(variables('GeneralName'))]"
   },
```

Template – Sample – Hard Coded

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
    },
    "variables": {
    },
    "resources": [{
        "apiVersion": "2018-10-01",
        "type": "Microsoft.Network/virtualNetworks",
        "name": "Engineering-vnet",
        "location": "EastUS",
        "properties": {
            "addressSpace": {
                "addressPrefixes": ["10.10.10.0/24"]
            }
        }
    }],
    "outputs": {}
}
```

Template – Sample – Modernized

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "Department": {
            "type": "string"
        },
        "VnetAddressPrefix": {
            "type": "string"
        }
    },
    "variables": {
        "VnetName": "[concat(parameters('Department'),'-vnet')]"
    },
    "resources": [{
        "apiVersion": "2018-10-01",
        "type": "Microsoft.Network/virtualNetworks",
        "name": "[variables('VnetName')]",
        "location": "[resourceGroup().location]",
        "properties": {
            "addressSpace": {
                "addressPrefixes": ["[parameters('VnetAddressPrefix')]"]
            }}}],
    "outputs": {}}
```

Lab: Deployments

Create a VM in Azure Portal then explore & deploy its template

Knowledge Check

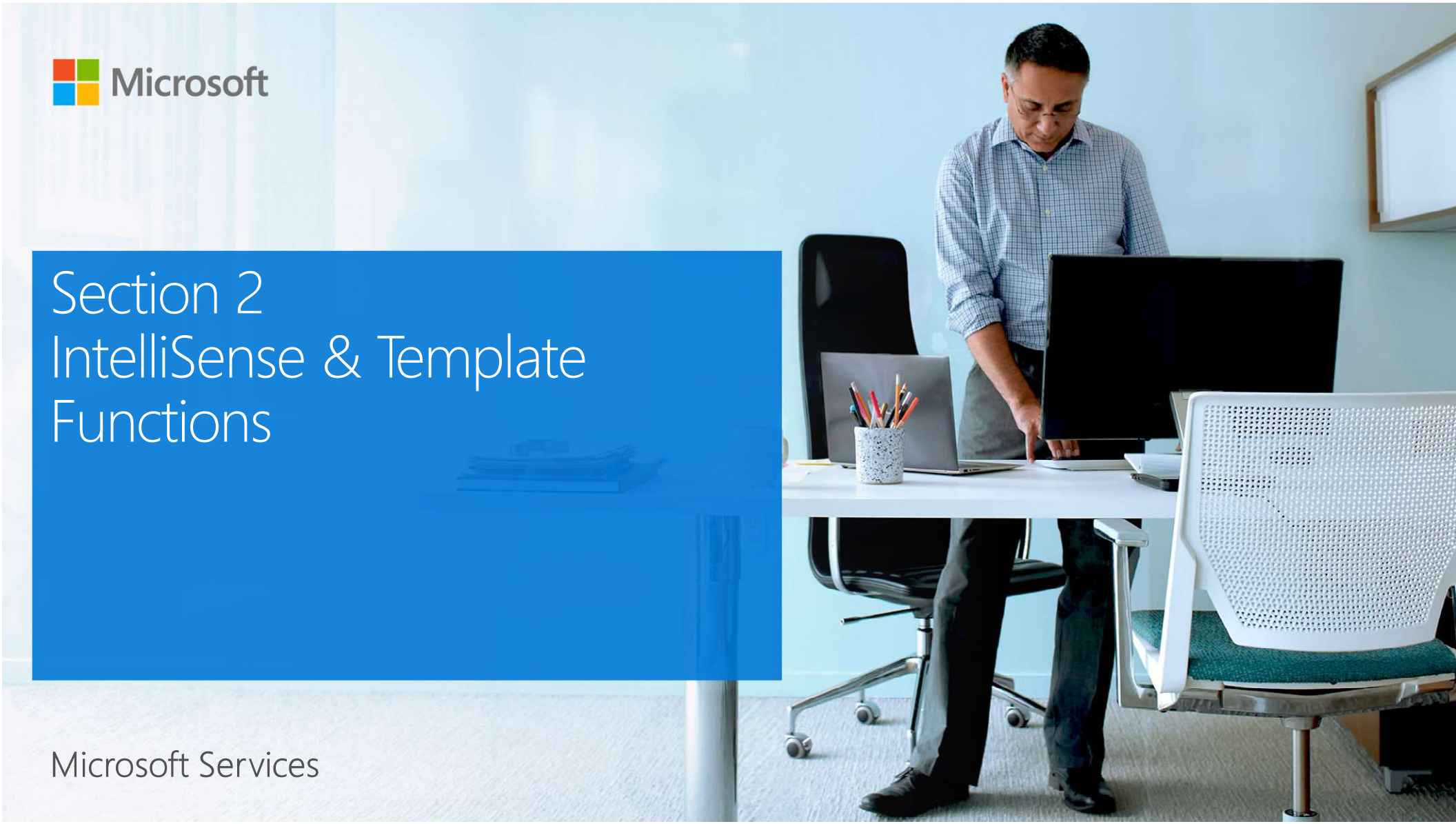
- What template section has values inputted at time of deployment?

  Parameters
- What are examples of template functions?
  - String: length, concat, toLower
  - Resource: resourceGroup, resourceId
  - Numeric: add, min, max
  - https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-template-functions
- Where can JSON code be stored?

  Any source code repository tool such as Azure DevOps, Subversion, & GitHub

Microsoft

Section 2
IntelliSense & Template Functions



Microsoft Services

Objectives

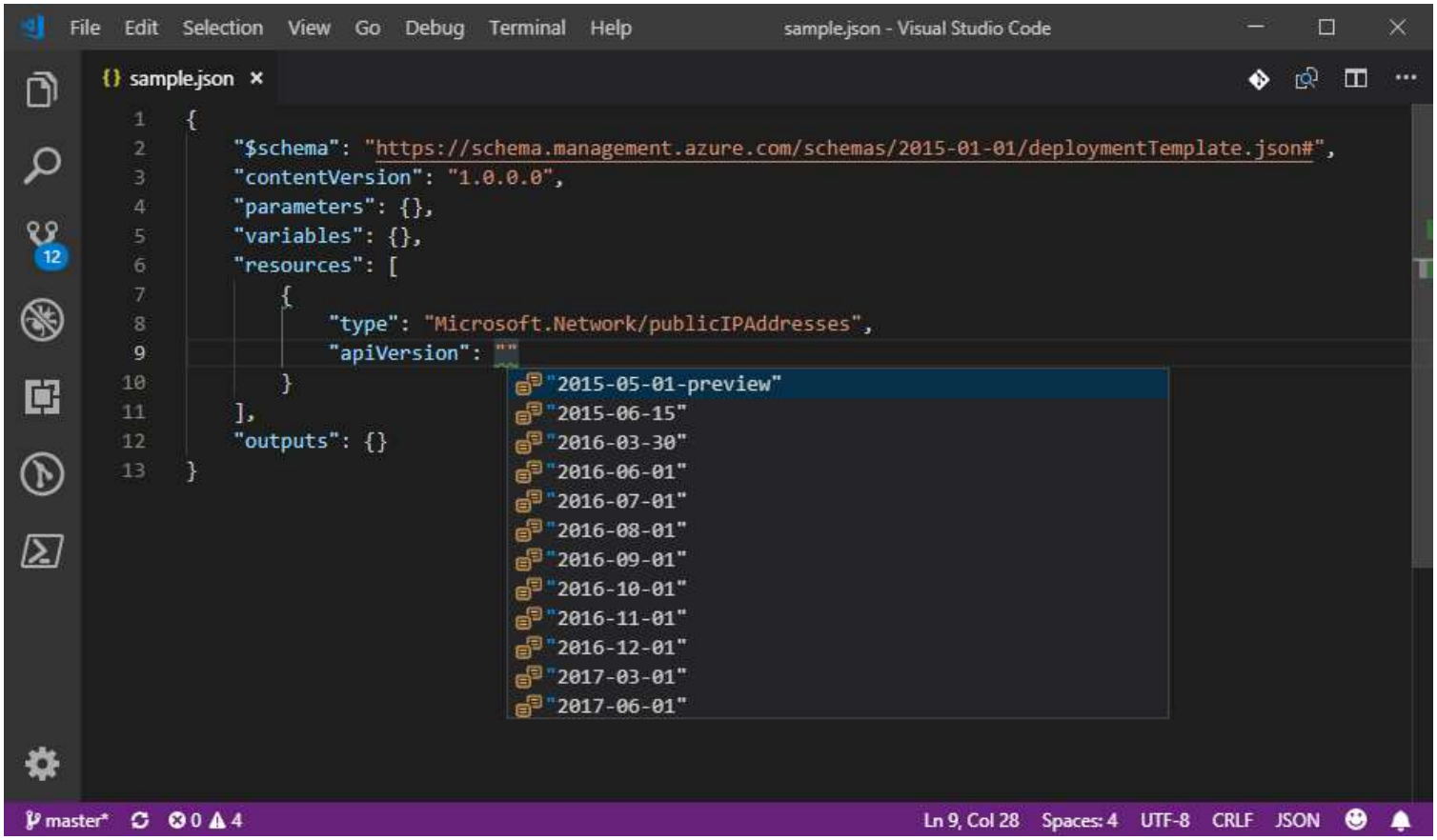
- Learn how to efficiently write ARM templates
- Leverage IntelliSense and snippets in VS Code
- Understand functions available in ARM templates

IntelliSense in VS Code

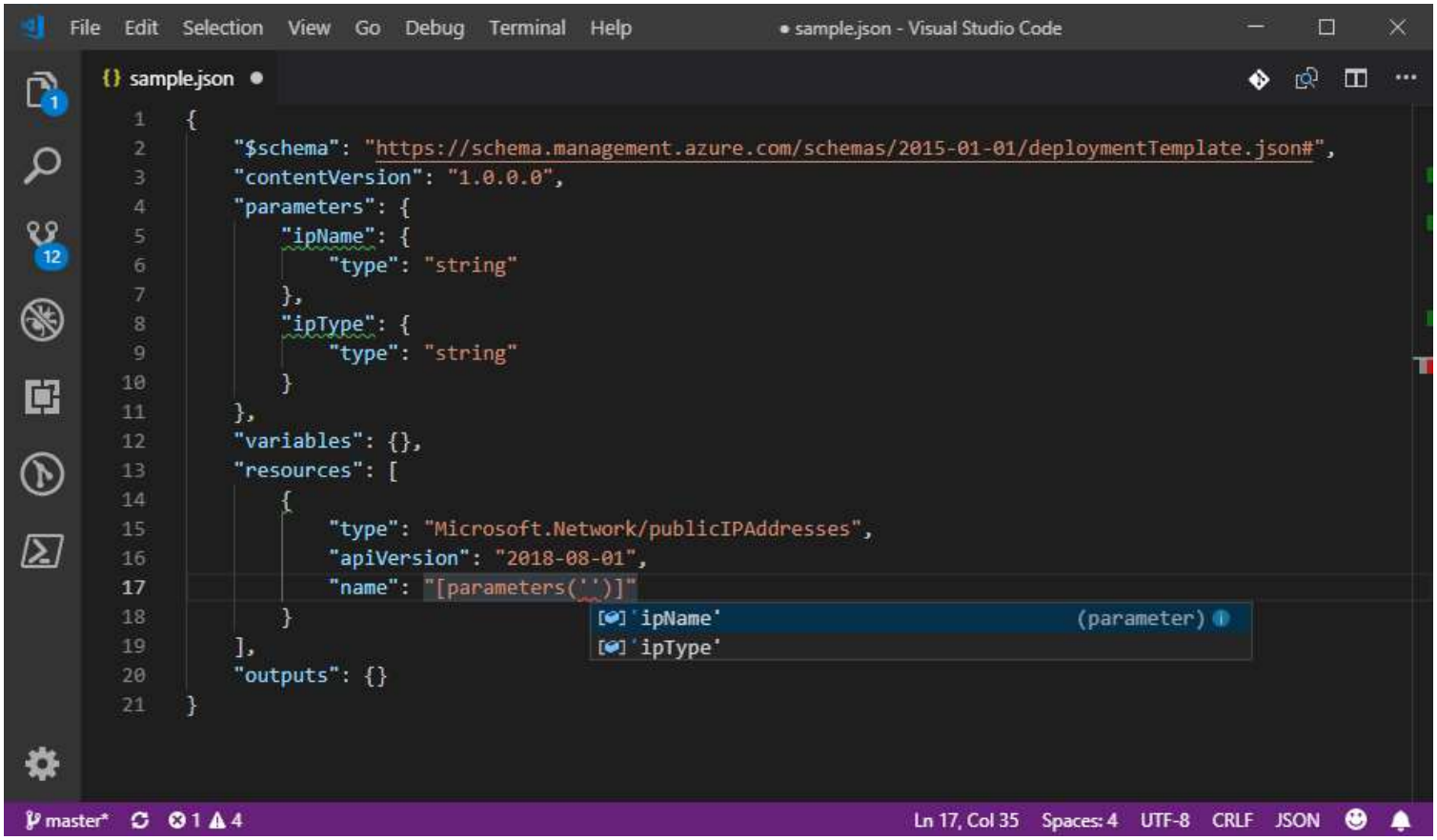
- IntelliSense is built into VS Code
- Offers suggestions while building JSON template
- Calls ARM API for core suggestions out of the box
- VS Code extensions enhance suggestions

  Azure Resource Manager Tools extension adds IntelliSense for:
  - Template Language Expression (TLE) function names
  - Parameter & Variable references
  - resourceGroup() & subscription() properties
  - Properties of references to variables that are objects

Native IntelliSense in VS Code



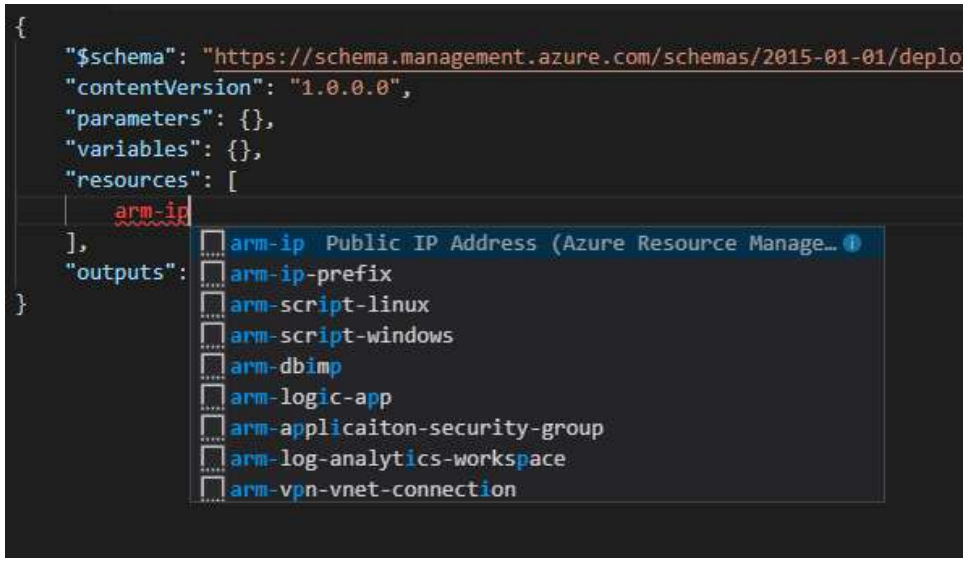
Enhanced IntelliSense in VS Code

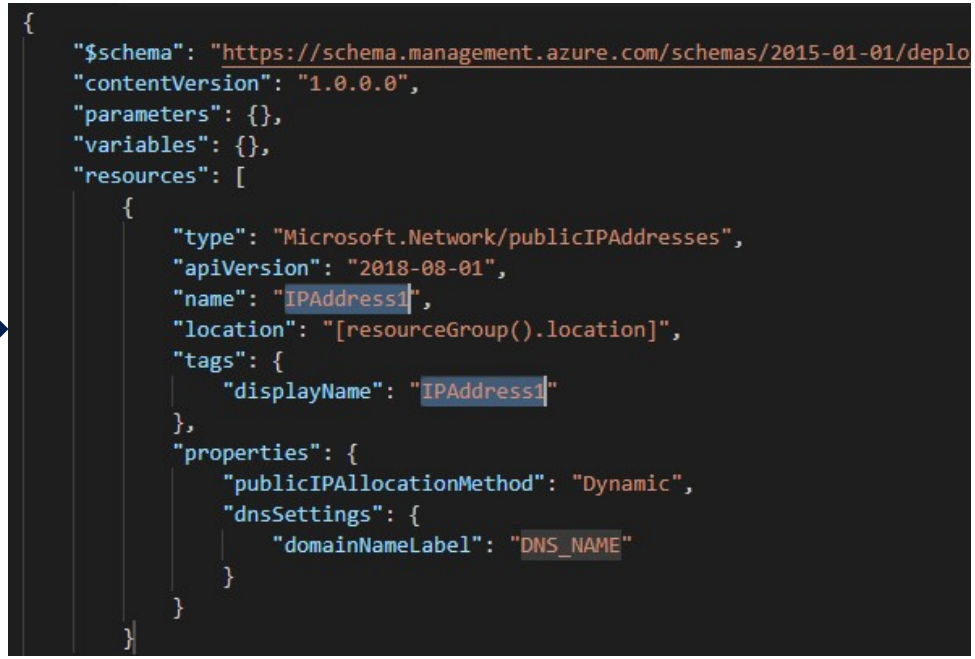


Snippets in VS Code

- Snippets quickly insert common code blocks
- Customizable in JSON format
- ARM template snippet samples available:
  - GitHub repository: Azure/azure-xplat-arm-tooling
  - VS Code Extension: Azure Resource Manager Snippets

Snippets in VS Code





ARM Template Functions

- Functions are used to dynamically generate values
- Expressions evaluated at time of deployment
- Manipulate string, integer, arrays and objects
- Return values from Azure resources
- Comparison and logical functions to evaluate values

ARM Template Functions - Concat

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "projectName": {
            "type": "string",
            "defaultValue": "ContosoWeb"
        }
    },
    "variables": {},
    "resources": [],
    "outputs": {
        "concatOutput": {
            "value": "[concat(parameters('projectName'), '-', resourceGroup().location)]",
            "type": "string"
        }
    }
}
```

```
Name           Type                       Value
=============  =========================  ==========
concatOutput   String                     ContosoWeb-eastus
```

ARM Template Functions - UniqueString

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [],
    "outputs": {
        "uniqueValue": {
            "value": "[concat('storage', '-', uniqueString(subscription().subscriptionId))]",
            "type": "string"
        }
    }
}
```

```
Name           Type                   Value
=============  =====================  ==========
uniqueValue    String                 storage-5wmrk44e5a4p4
```

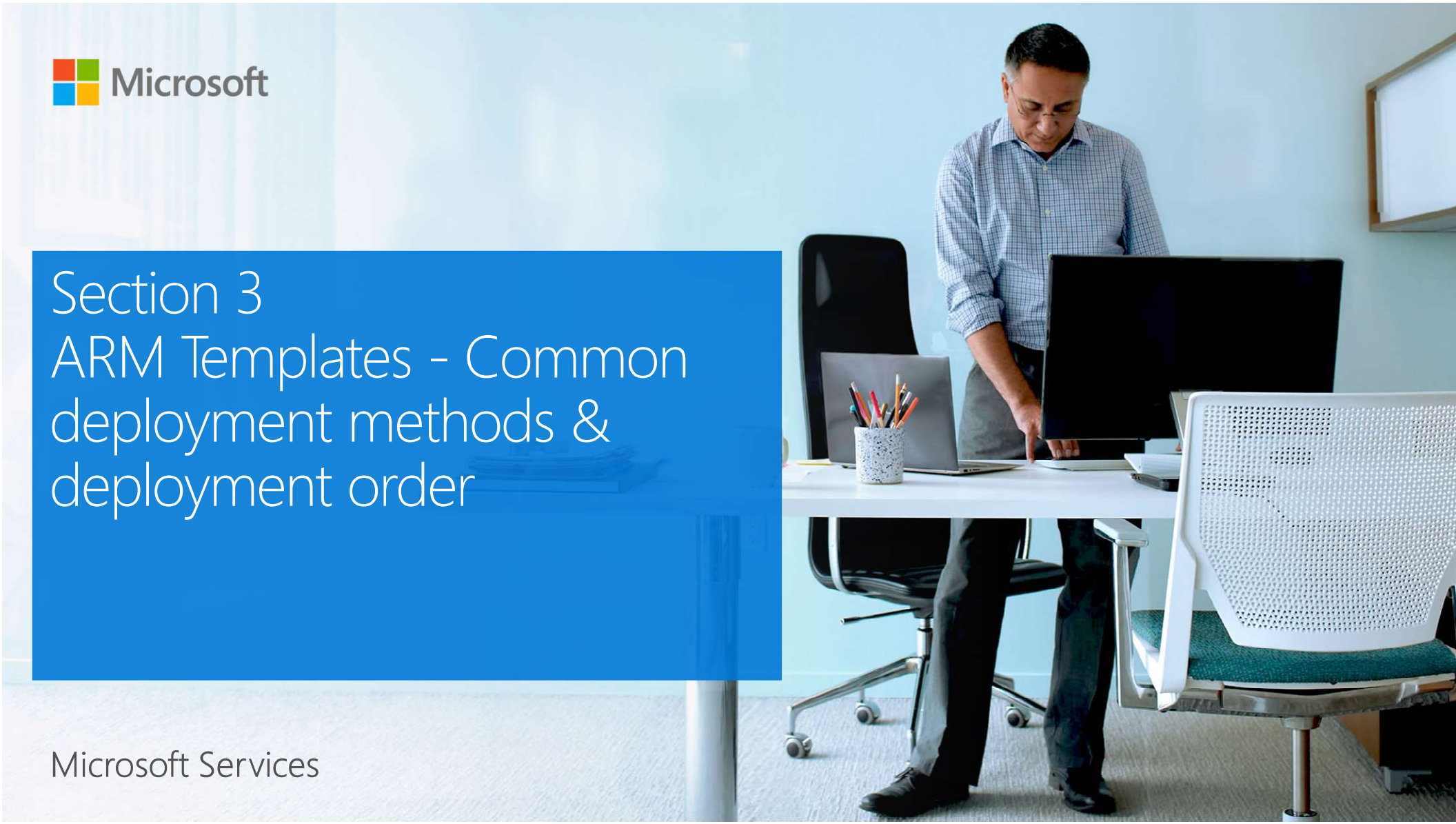
Lab: IntelliSense & Template Functions

Knowledge Check

- Does Visual Studio Code offer IntelliSense for ARM templates?
- How do snippets help improve development efficiency?
- What are some options to make ARM templates more dynamic?

Microsoft

Section 3
ARM Templates - Common deployment methods & deployment order



Microsoft Services

Objectives

After completing this learning unit, you will

- Describe mechanisms for providing Parameters
- Understand a parameter template structure
- Comprehend Linking and nesting templates
- Demonstrate deployment order

Templates & Parameters – Being pushed to ARM

- Both Template & Parameter in file, code, or URI links can be pushed to ARM. All achieve the same results.
- The method of push can be different for both Template and Parameter
- Examples:
  - Static template URI + parameters coded as PowerShell arguments
  - Static template URI + unique parameter URI
  - Static template file + unique parameter file

Pushing templates to Azure Resource Manager

- Methods: File, URI, Object/Hashtable
  - `New-AzResourceGroupDeployment -TemplateFile "C:\temp\azuredeploy.json" […]`
  - `New-AzResourceGroupDeployment -TemplateUri "https://contoso.com/azuredeploy.json" […]`
  - `New-AzResourceGroupDeployment -TemplateObject $TemplateHashTable`

Pushing parameters to Azure Resource Manager

- Methods: File, URI, Object/Hashtable, Code Arguments
  - `New-AzResourceGroupDeployment -TemplateParameterUri https://contoso.com/template.parameters.json […]`
  - `New-AzResourceGroupDeployment -TemplateParameterObject $params […]`
  - `New-AzResourceGroupDeployment -VMName "MyVM" -DataDiskCount "1" […]`
- Remember: Parameters are just answers for Templates

Template Parameter Structure – File Example

- Content of parameter file

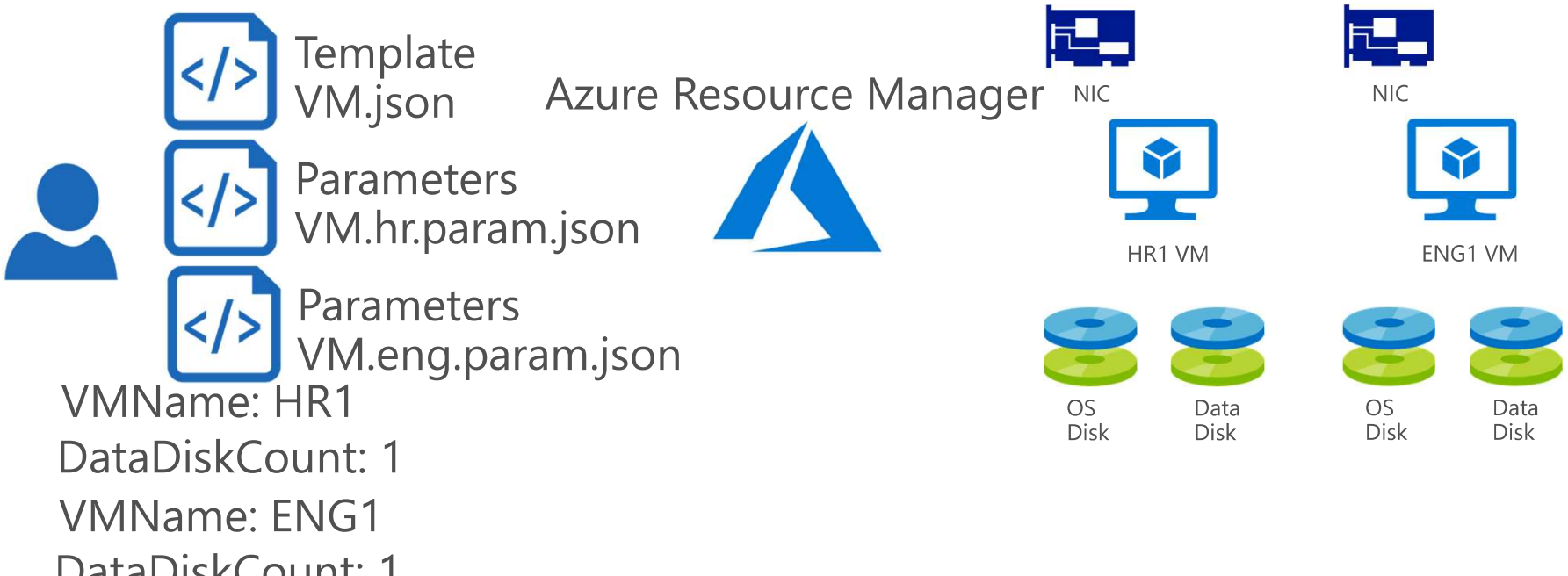
```
{
   "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
   "contentVersion": "1.0.0.0",
   "parameters": {
      "VMName": {
         "value": "HR1"
      },
      "DataDiskCount": {
         "value": "1"
      }
        }
}
```

- PowerShell command to deploy template file with parameter file

```
New-AzResourceGroupDeployment -TemplateFile "C:\temp\template.json“
-TemplateParameterFile "C:\temp\template.parameters.json“ […]
```

Template reuse – just change parameters!

- 1-time use templates are inefficient, code for reusability!



How many templates are needed?

- Single templates
  - Ideal for: Small solutions, testing with resource types, long term minimal changes
  - Example: SCCM instance
- Multiple templates
  - Ideal for: Large solutions, modern enterprises, & compartmentalized teams
  - Examples
    - Need to deploy to multiple resource groups
    - Setting up base environment, like virtual network, log analytics, etc
    - Team responsibility: Database, Cyber Security, System Administrators
    - Conditions – such as, only deploy if statement is true

Nested & Linked Templates

- Nested templates: Template's code is embedded in a parent template as a resource. (A template within a template)

  Parameters and variables can only come from the parent template
- Linked templates – A resource in a parent template that kicks off an independent deployment via URI

  Parameters and variables can be defined from the parent template OR a separate parameter file URI

Nested and Linked Templates – code example

```
{
"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "",

"parameters": { },

"variables": { },

"functions": [ ],

"resources": [
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2018-05-01",
            "name": “ChildTemplate",
            "properties": {
            "mode": "Incremental",
            [...]
"outputs": { }
```

Minor code difference to distinguish nested or linked

Nested Template – code example

```
"resources": [
{
 "type": "Microsoft.Resources/deployments",
 "apiVersion": "2018-05-01",
 "name": "nestedTemplate",
 "properties": {
  "mode": "Incremental",
  "template": {
   "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
   "contentVersion": "1.0.0.0",
   "resources": [
    {
     "type": "Microsoft.Storage/storageAccounts",
     "apiVersion": "2018-07-01",
     "name": "[variables('storageName')]",
     "location": "West US",
     "properties": {
     "accountType": "Standard_LRS"
     […]
```

Parent resource indicating a nested or linked template

Indicates a nested template

The same contents of a typical template but without parameters, variables, or outputs

Linked Template – code example

```
"resources": [
{
 "type": "Microsoft.Resources/deployments",
 "apiVersion": "2018-05-01",
 "name": “linkedTemplate",
 "properties": {
  "mode": "Incremental",
  "templateLink": {
    "uri":"https://storageacct.blob.core.windows.net/newStorageAccount.json",
    "contentVersion":"1.0.0.0"
    },
    "parametersLink": {
    "uri":"https://storageacct.blob.core.windows.net/Templates/newStorageAccount.parameters.json",
    "contentVersion":"1.0.0.0"
    }
     […]
```

Parent resource indicating a nested or linked template

Indicates a linked template

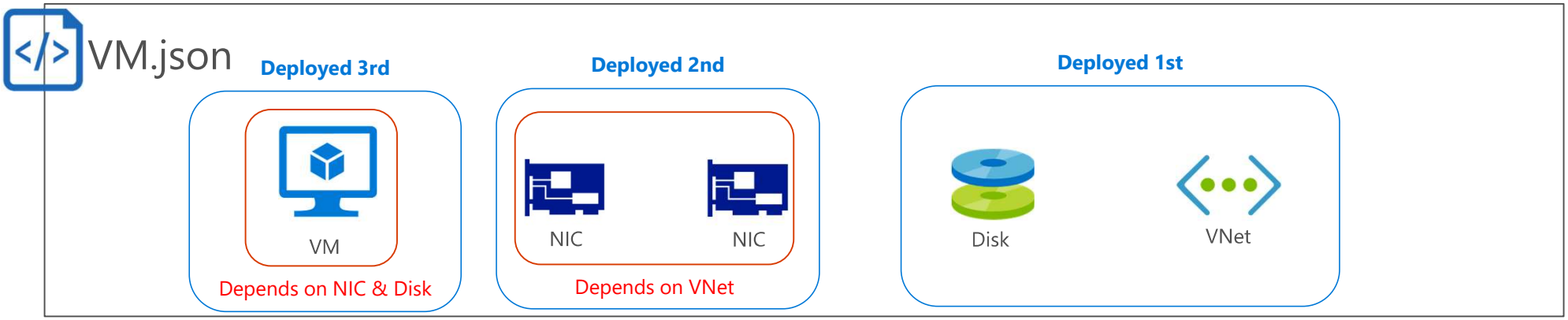
Links to the template and parameter URIs

Templates – Deployment order / Dependencies

- No need to write deployment order as serial code

  PowerShell serial code: **T**op of code-first to deploy; **B**ottom of code-last to deploy
- Code dictates deployment order by stating dependencies
  - Template example: VM depends on the NIC which depends on the Vnet
  - Only resources from the same template should be defined as dependencies
- All Resources are deployed at the same time

  Dependencies are deployed first in the "at the same time" parallel manner



Templates – Deployment Order - Code

```
"resources": [
{
"type": "Microsoft.Network/networkInterfaces",
"apiVersion": "2018-11-01",
"name": "[variables('nicName')]",
"location": "[parameters('location')]",
"dependsOn": [
   "[resourceId('Microsoft.Network/publicIPAddresses/', variables('publicIPAddressName'))]",
   "[variables('virtualNetworkName')]"
], […]
```

Defines that this resource has dependencies that must be deployed first

The dependency resources must be uniquely identified such as by their name or resourceId. They must also be deployed in the same template

- Linked and nested template resources can also be defined as dependencies
  Example: Deploy Virtual Network linked template resource before deploying a VM & NIC

Lab: Deployment Methods

Deploy templates a variety of ways

Knowledge Check

- What is the difference between a nested and linked template?
  - Linked templates are resource deployments whose template and parameters are separate URI files.
  - Nested templates are resource deployments whose entire template contents are in-line as part of the parent template. The parameters must come from the parent template.

Microsoft

Section 4
Copy element & copyIndex( ) function

Microsoft Services

Objectives

- Learn how to efficiently create multiple instances of a resource, property or variable

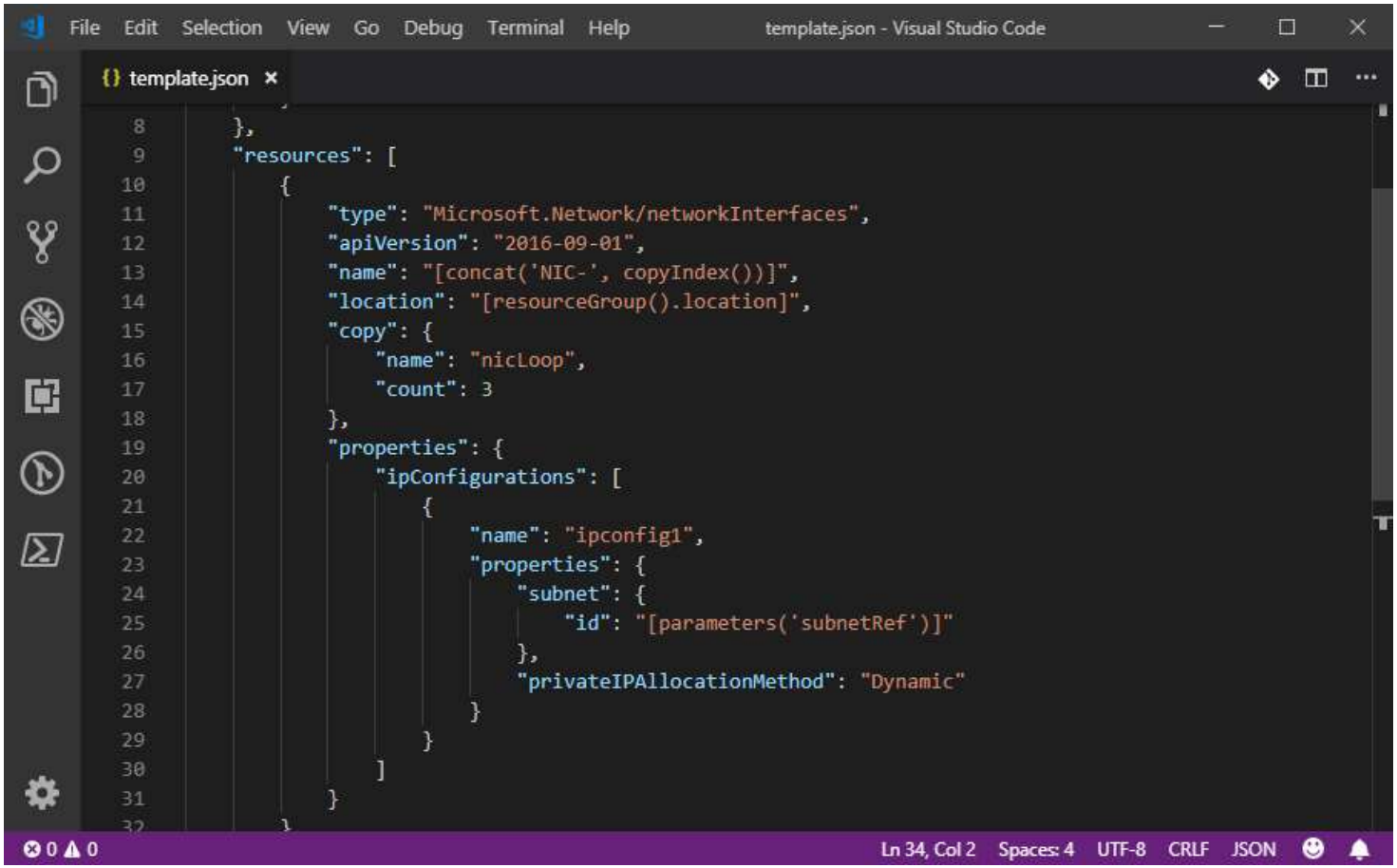
Overview

- Copy loops simplify multiple instance creation
- ARM templates can run a copy loop for:
  - Resources
  - Properties
  - Variables
- Up to 800 copies

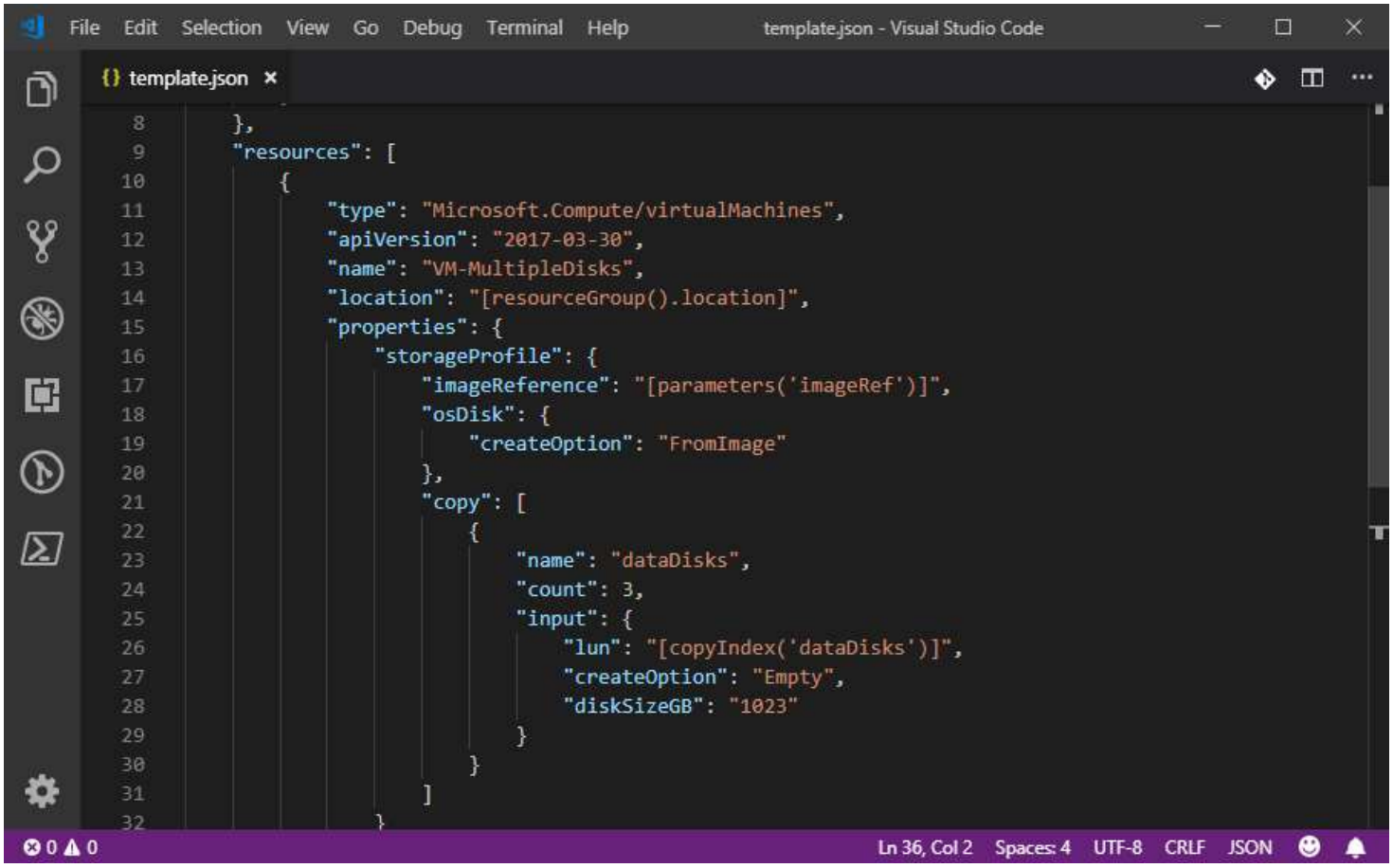
copyIndex( ) function

- Returns the index of an iteration loop
- Can be used to create dynamic name
- Can be used to return values from an array
- Has two optional parameters:
  - loopName - The name of the loop for getting the iteration.
  - offset - The number to add to the zero-based iteration value.

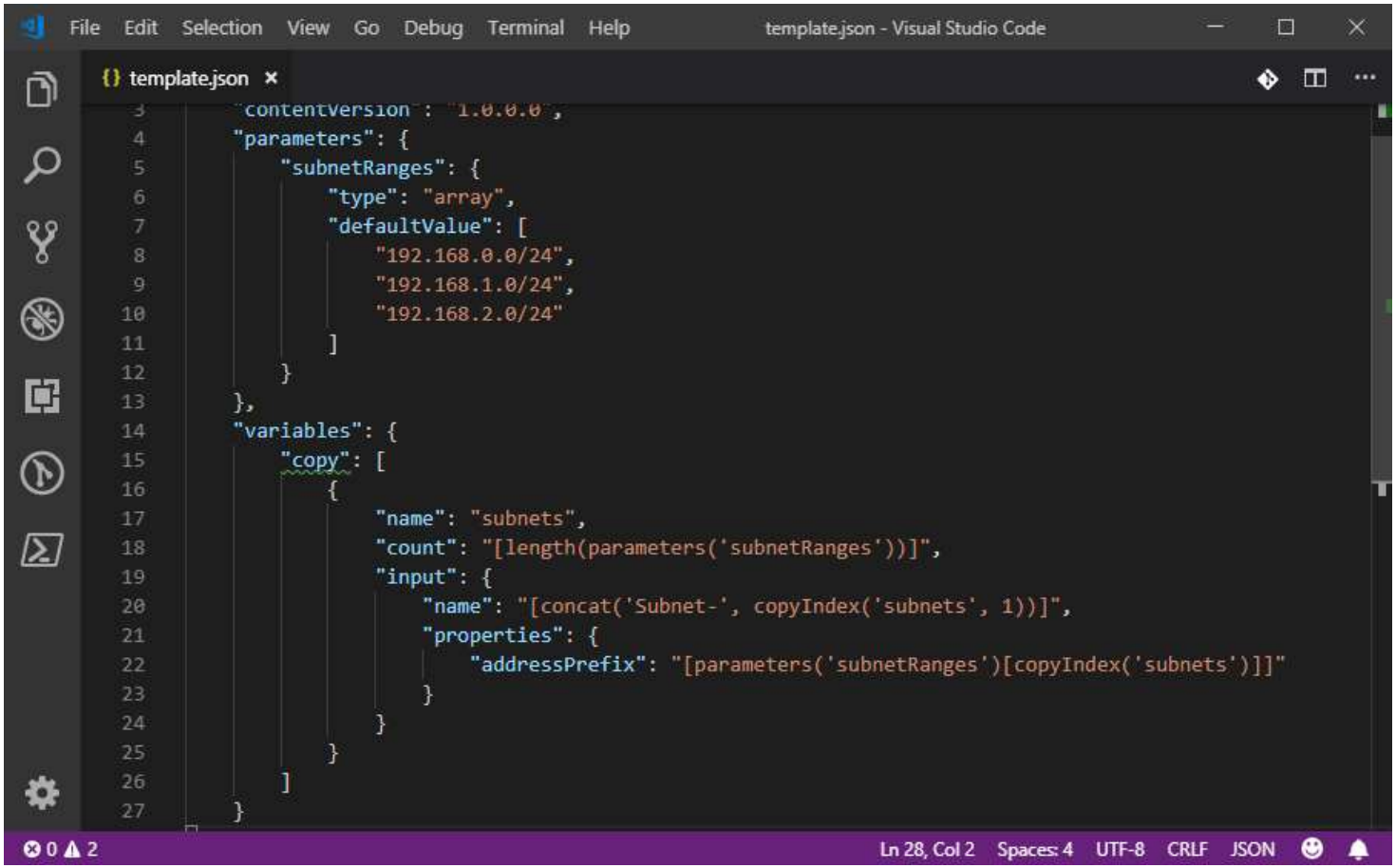
Resource Iteration



Property Iteration



Variable Iteration



Lab: Copy element & copyIndex( ) function

Microsoft

Section 5
ARM Templates – Advanced Template Architecting

Microsoft Services

Objectives

After completing this learning unit, you will

- Understand reusability capabilities with Variables
- Recognize methods for storing reusable values
- Constraining parameters to specific values
- How to use Conditions/IF statements for resource deployment eligibility

Governance… Governance… Governance…

- Governance strategies ideally should occur before any buildouts
- Resource names usually incorporate purpose, region/location, environment classification, etc.
  - Example:
    - ADFS-P-EUS-1-vm
      Identifies this resource as the 1st ADFS production VM in East US
    - Engineering-P-EUS-vnet
      Identifies this resource as the production vnet in East US for engineering
- Resource name is derived as opposed to a parameter
  - Example: By knowing a production VM will belong to 'Engineering', the name of the dependent Vnet can be generated by a governance formula

Governance… Governance… Governance…

- Always use derivable names, never randomization
- Randomization means the name must be provided as explicit parameter input

Governance on resource name - example

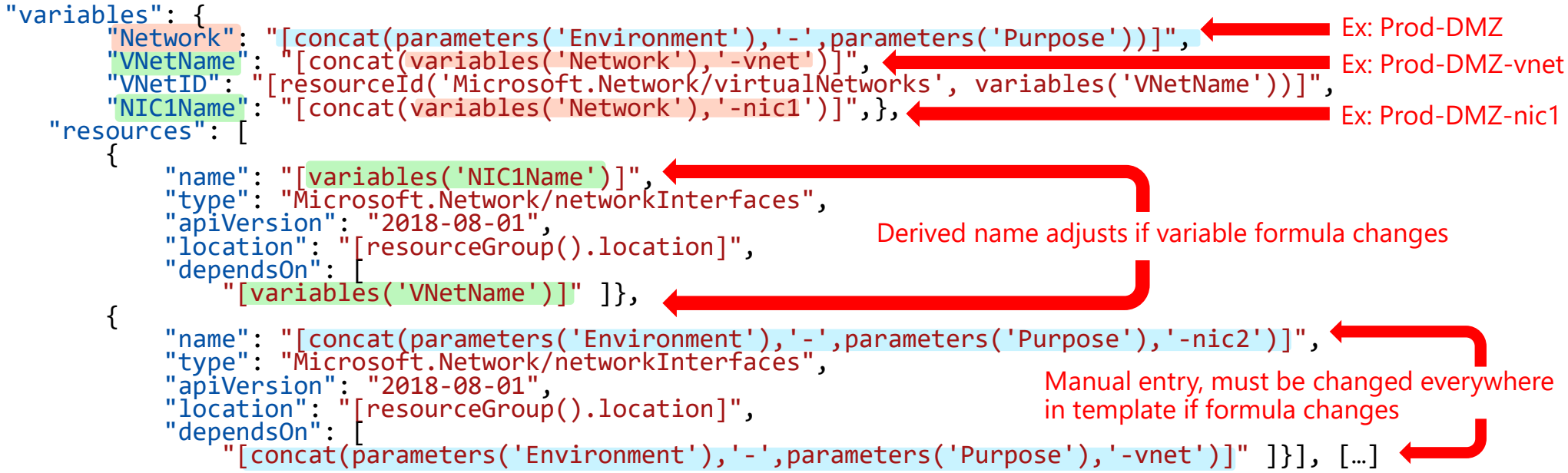
- Environment: Production; Type: Vnet; Purpose: Engineering
- The Variables section of a template can concatenate the values to form the proper name

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "Environment": {
           "type": "string"},
        "Purpose": {
           "type": "string"},
    "variables": {
        "VNetName": "[concat(parameters('Environment'),'-',parameters('Purpose'),'-vnet')]"},[…]
```

Vnet name: Production-Engineering-vnet

Reusability of a variable vs manual entry

- By referencing a Variable, its formula may change but your reference never needs to. A manual entry must be corrected everywhere



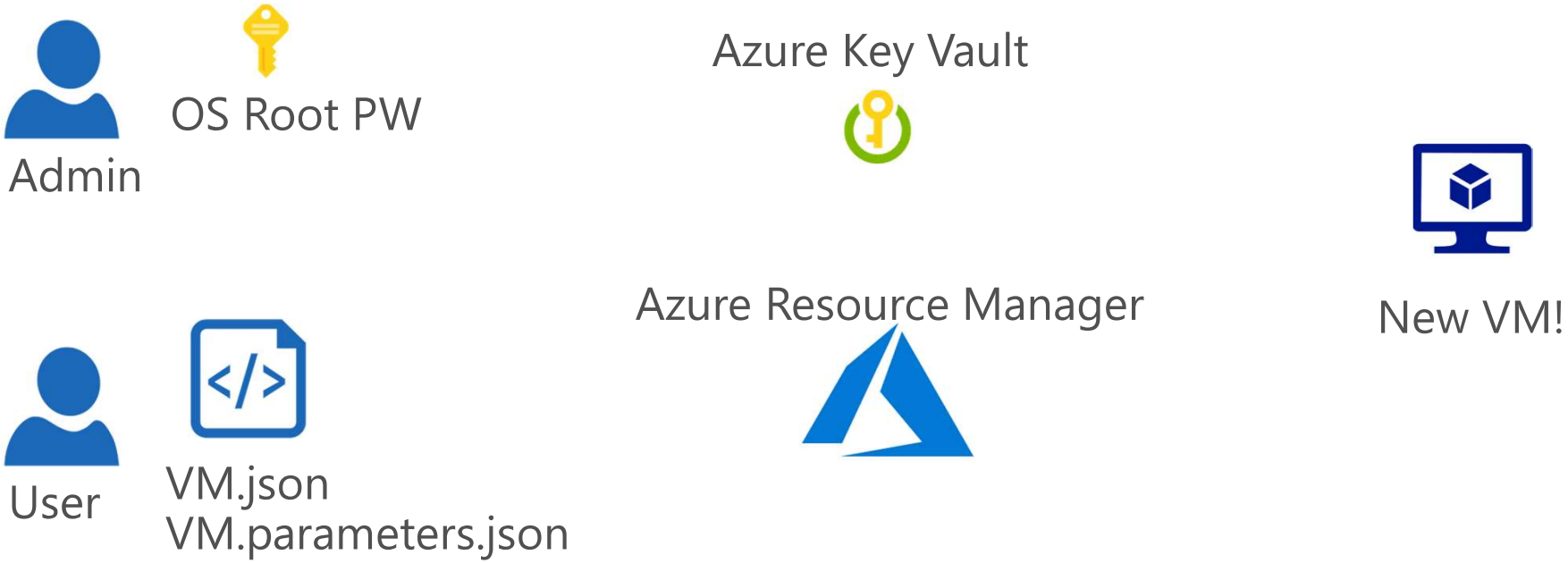
Passing secure values to ARM for Deployments

- Secrets, like Passwords, can be securely passed to ARM as SecureString or pulled from Azure Key Vault

- For templates, the parameter must be type: SecureString
  Values are not included in historical deployment logs or template downloads

Storing & retrieving Secrets in Azure Key Vault

- Azure Key Vault can store secret values that are injected by the ARM service at time of deployment. Ex: OS Admin PW
  - As a key value regularly changes (like all passwords should) the reference doesn't
  - The value can be versioned (latest and previous values can be retrieved)



Azure Key Vault Example

- Template parameter files designate location of keyvault / secret pair

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminPassword": {
            "reference": {
                "keyVault": {
                "id": "/subscriptions/<SubscriptionID>/resourceGroups/<RG>/providers/Microsoft.KeyVault/vaults/<KeyVaultName>"
                },
                "secretName": "vmAdminPassword"
            }
        },
    }
}
```

Resource ID of KeyVault resource

Name of Secret. Latest version is assumed

Passing SecureStrings without Key Vault

- Template parameter designates the value requirement by having type SecureString

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminPassword": {
            "type": "securestring"}
        }, […]
```

Whether using Key Vault or not, the parameter's type will be SecureString

```
$adminPassword = Read-Host -AsSecureString
New-AzResourceGroupDeployment -TemplateFile C:\vm.json `
                              -AdminPassword $adminPassword [...]
```

Template Structure - Parameters

```
"parameters": {
    "<parameter-name>" : {
        "type" : <string, securestring, int, bool, object, secureObject, array>
        "defaultValue": "<default-value-of-parameter>",
        "allowedValues": [ "<array-of-allowed-values>" ],
        "minValue": <minimum-value-for-int>,
        "maxValue": <maximum-value-for-int>,
        "minLength": <minimum-length-for-string-or-array>,
        "maxLength": <maximum-length-for-string-or-array-parameters>,
        "metadata": {
            "description": "<description-of-the parameter>"
        }
    }
}
```

Parameters – Restricting Input – Template Example

```
"parameters": {
        "AzureRegion": {
          "type": "string",
          "defaultValue": "EastUS",
          "allowedValues": ["EastUS", "WestUS", "CentralUS"],
          "metadata": {
               "description": "Azure region to deploy to"
           }
        },
        "VMCount": {
          "type": "int",
          "defaultValue": "1",
          "maxValue": 5,
          "metadata": {
               "description": "Number of VMs to deploy"
           }
        }
    },
```

Use Objects for Flexibility & Depth

- Objects are hierarchical properties with attribute values
- Coding templates to be driven by objects reduces template complexity by offloading the customizations to the parameter file
- Manipulate Parameter file objects with PowerShell

Use Objects for Flexibility & Depth – Parameter File

```
"VNet": {
    "value": {
        "EnableDDoSProtection": false,
        "AddressSpace": {
            "AddressPrefixes": [
                "10.1.0.0/16",
                "10.2.0.0/16"
            ]},
        "DhcpOptions": {
            "DnsServers": [
                "4.4.4.4", "5.5.5.5"
            ]},
        "Subnets": [{
                "AddressPrefix": "10.1.0.0/24",
                "Name": "default"
            },
            {
                "AddressPrefix": "10.1.255.224/27",
                "Name": "GatewaySubnet"
            },
            {
                "AddressPrefix": "10.1.250.0/23",
                "Name": "Management"
            }]}}}
```

Template: [parameters('vNet').addressSpace.addressPrefixes]
This results in all entries being inserted at once

Use Objects for Flexibility & Depth – Template File

```
{
"type": "Microsoft.Network/virtualNetworks",
"name": "[concat(variables('EnvironmentContext'),'-vnet')]",
"location": "[resourceGroup().location]",
"properties": {
    "dhcpOptions":{
        "dnsServers":
            "[parameters('vNet').dhcpoptions.dnsservers]"
    },
    "enableDdosProtection": "[parameters('vNet').enableDdosProtection]",
    "addressSpace": {
        "addressPrefixes":
            "[parameters('vNet').addressSpace.addressPrefixes]"
        },
    "copy": [
        {
            "name": "subnets",
            "count": "[length(parameters('vNet').subnets)]",
            "input": {
                "name": "[parameters('vNet').subnets[copyIndex('subnets')].name]",
                "properties": {
                    "addressPrefix": "[parameters('vNet').subnets[copyIndex('subnets')].addressPrefix]",
                }}}]}} […]
```

Nested property – array value

Nested property – boolean value

Loop the Property 'subnets', which is an Array in the parameter file, and output its contents for each loop

Count the entries of the array for looping

Each array element's output is input

For a particular array element (the iteration of the loop), use the object's nested values

Use Objects for Flexibility & Depth – Compare

```
"Subnets": [{
                "AddressPrefix": "10.1.0.0/24",
                "Name": "default"
            },
            {
                "AddressPrefix": "10.1.255.224/27",
                "Name": "GatewaySubnet"
            },
            {
                "AddressPrefix": "10.1.250.0/23",
                "Name": "Management"
            }]
```

Parameter file: 3 entries for subnets

```
 "copy": [
        {
            "name": "subnets",
            "count": "[length(parameters('vNet').subnets)]",
            "input": {
                "name": "[parameters('vNet').subnets[copyIndex('subnets')].name]",
                "properties": {
                    "addressPrefix": "[parameters('vNet').subnets[copyIndex('subnets')].addressPrefix]",
                 }}}]
```

Template file: Loop based on the number of entries

Use Objects for Flexibility & Depth – Compare

```
"VNet": {
    "value": {
        "EnableDDoSProtection": false,
        "AddressSpace": {
            "AddressPrefixes": [
                "10.1.0.0/16",
                "10.2.0.0/16"
            ]},
```

Parameter file: Array for AddressPrefixes & Boolean for EnableDDoSProtection

---

```
{
"type": "Microsoft.Network/virtualNetworks",
"name": "[concat(variables('EnvironmentContext'),'-vnet')]",
"location": "[resourceGroup().location]",
"properties": {
    "dhcpOptions":{
        "dnsServers":
            "[parameters('vNet').dhcpoptions.dnsservers]"
    },
    "enableDdosProtection": "[parameters('vNet').enableDdosProtection]",
    "addressSpace": {
        "addressPrefixes":
            "[parameters('vNet').addressSpace.addressPrefixes]"
        },
```

Template file: Insertion of whole value from Parameter

Lab: Deployment Flexibility

Deploy Template using Key Vault, Variables, & Objects

Knowledge Check

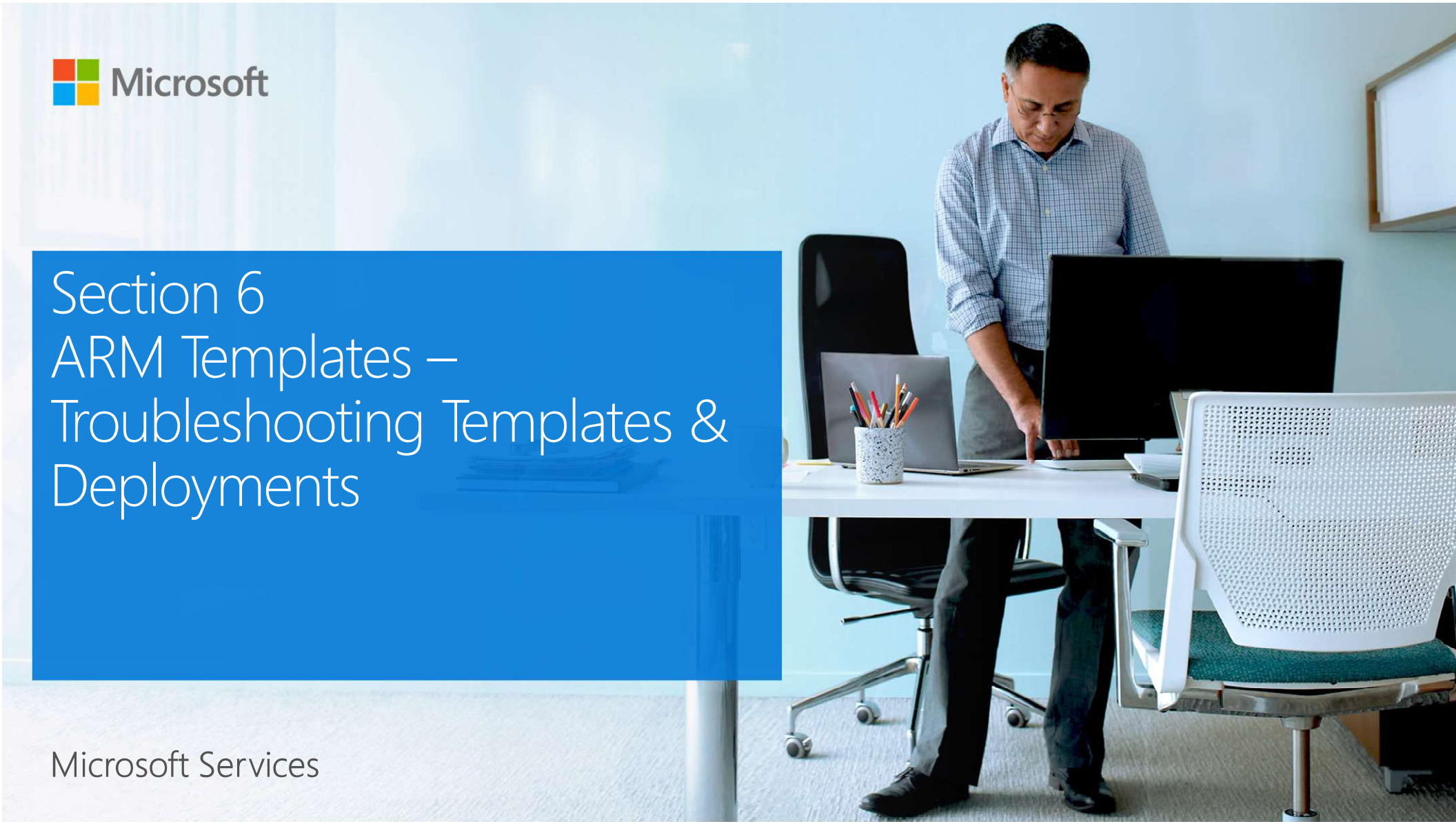
- What are the advantages of establishing governance?

  Reduce uncontrolled spend, understand a resource's purpose based on name, follow corporate policy especially security policy
- What Parameter properties can be used to restrict or fence the value input?

  Allowed Values, Max and Min, Input type (integer, string, etc.)

Microsoft

Section 6
ARM Templates – Troubleshooting Templates & Deployments



Microsoft Services

Objectives

After completing this learning unit, you will

- Understand how to read error messages
- Comprehend what transient & permanent errors are
- Use intellisense to identify issues prior to deployment

Template deployments may encounter issues

- Templates will fail to deploy for temporary and permanent reasons. These are called **Validation** & **Deployment** errors.
- Error messages should state the cause and possibly line # in code issue originates from
- Issues can be caught at time of template creation or may be found at deployment runtime/validation
- The error messages are user-friendly but must be read carefully to understand their meaning
- Words of wisdom: “Every new error message, is progress”
  - - Chase Dafnis

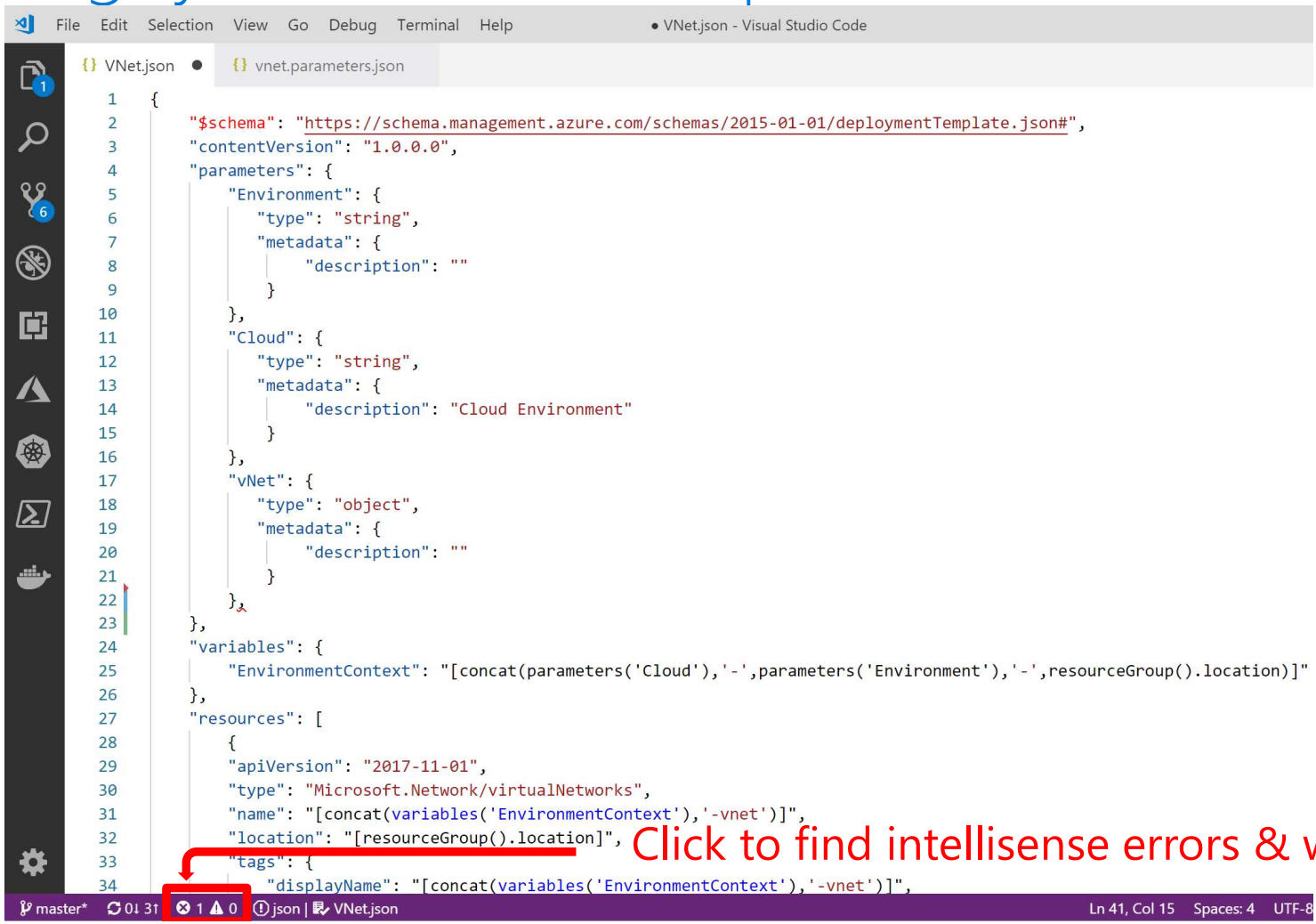
Client side / Validation Errors

- Syntax issues such as a misplaced brackets & quotes will be caught by intellisense
- Intellisense /may/ also catch invalid or missing attribute values based on the Azure schema for the resource type
- Template submissions for deployment go through a validation process prior to actual deployment
- Azure Portal based deployments will include additional validation checks such as exceeding subscription quota

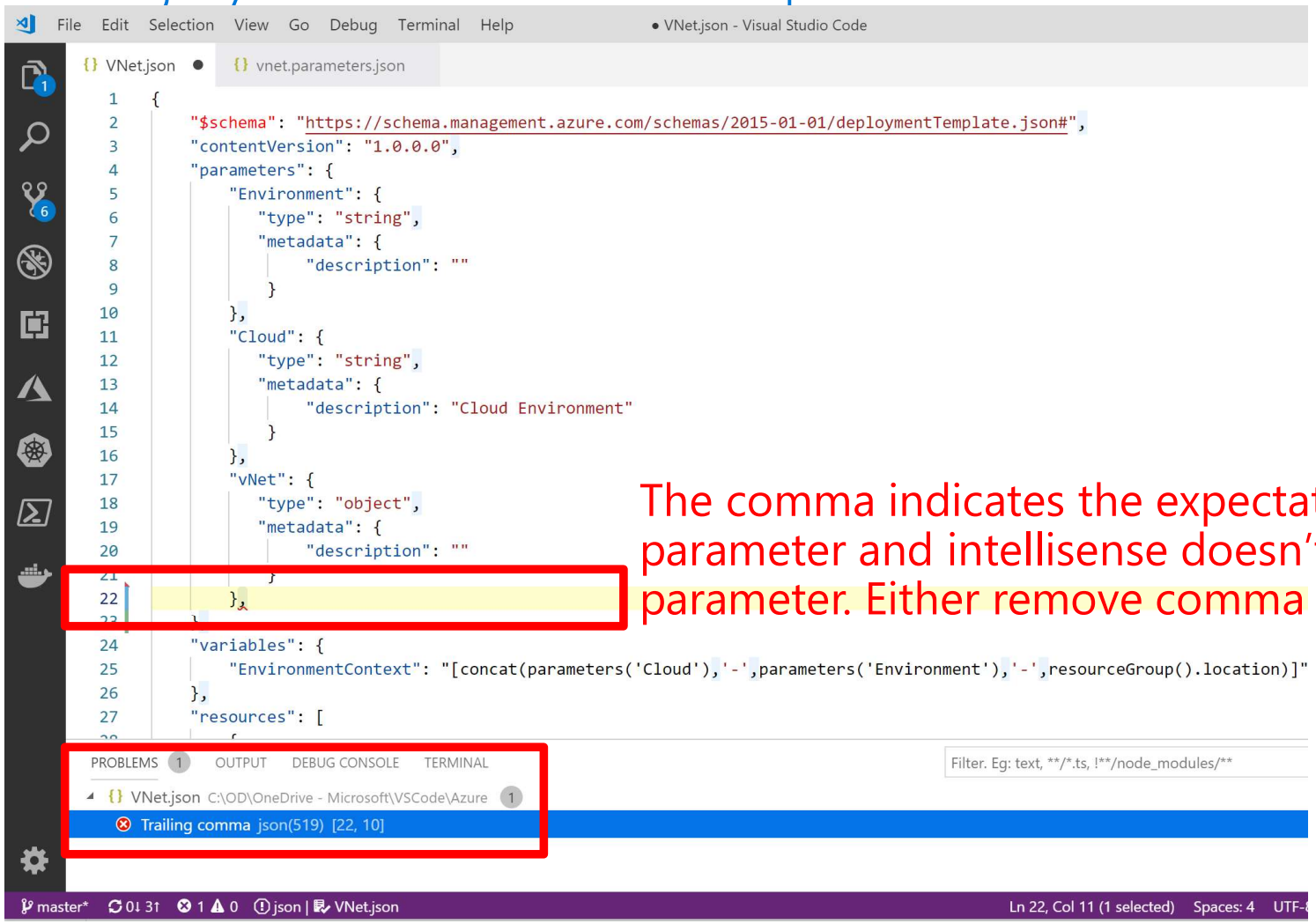
Deployment Errors

- ARM will catch anything not caught by client tooling
- Transient issues such as throttling or an Azure service being unavailable will self-resolve with time
- Resource value violations, such as conflicting Vnet IP space, will be clearly indicated in deployment feedback
- Poor template architecting, such as lack of inputting resource dependencies, may or may not error out
- Azure governance such as Policies & RBAC/Authorizations of the deploying user will always hold true

Decoding syntax errors in Template – What's wrong here?



Decoding syntax errors in Template – What's wrong here?



The comma indicates the expectation of another parameter and intellisense doesn't see a following parameter. Either remove comma or add parameter

Decoding an error message

Input to create new storage account:

```
New-AzResourceGroupDeployment -name storage -
ResourceGroupName test -TemplateFile
c:\temp\storage.json
```

- Multiple generic error messages will be displayed.
- Look for the specific one that details the name of the resource, line number, and/or error code

Decoding an error message - Example

Error #

Outer error message

1.
```
New-AzResourceGroupDeployment : 9:19:17 PM - Error: Code=InvalidTemplateDeployment; Message=The template deployment
'storage' is not valid according to the validation procedure. The tracking id is
'c777c21b-d16d-4f3c-afa7-d552cbd4bd1a'. See inner errors for details. Please see https://aka.ms/arm-deploy for usage
details.
At line:1 char:1
+ New-AzResourceGroupDeployment -name storage -ResourceGroupName test - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId :
Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet
```

2.
```
New-AzResourceGroupDeployment : 9:19:17 PM - Error: Code=PreflightValidationCheckFailed; Message=Preflight validation
failed. Please refer to the details for the specific errors.
At line:1 char:1
+ New-AzResourceGroupDeployment -name storage -ResourceGroupName test - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId :
Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet
```

3.
```
New-AzResourceGroupDeployment : 9:19:17 PM - Error: Code=StorageAccountAlreadyTaken; Message=The storage account named
test is already taken.
At line:1 char:1
+ New-AzResourceGroupDeployment -name storage -ResourceGroupName test - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId :
Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet
```

Inner error message

4.
```
New-AzResourceGroupDeployment : The deployment validation failed
At line:1 char:1
+ New-AzResourceGroupDeployment -name storage -ResourceGroupName test - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (:) [New-AzResourceGroupDeployment], InvalidOperationException
    + FullyQualifiedErrorId :
Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet
```

Decoding an error message – Example Template

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "variables": {},
    "resources": [
        {
            "name": "test",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2015-06-15",
            "location": "[resourceGroup().location]",
            "tags": {
                "displayName": "test"
            },
            "properties": {
                "accountType": "Standard_LRS"
            }
        }
    ],
    "outputs": {}
}
```

- Storage account names must be globally unique

Unique resource names

- Public facing resource types will share a Microsoft owned shared namespace
- All those resource types must be globally unique amongst all Azure customers
- Incorporate a unique prefix/suffix or UniqueString function
- Example resource types that have a unique requirement:
  - Storage: [Name].service.core.windows.net
  - Traffic Manager profile: [Name].trafficmanager.net
  - Public IP: [Name].region.cloudapp.azure.com
  - Event Hub: [Name].servicebus.windows.net

Decoding dependency errors in Template – 1st Deployment

```
VERBOSE: Performing the operation "Creating Deployment" on target "net".
VERBOSE: 4:18:44 PM - Template is valid.
VERBOSE: 4:18:48 PM - Create template deployment 'net'
VERBOSE: 4:18:48 PM - Checking deployment status in 5 seconds
VERBOSE: 4:18:53 PM - Resource Microsoft.Network/ddosProtectionPlans 'ddosprotectionplan' provisioning status is succeeded
New-AzResourceGroupDeployment : 4:18:53 PM - Resource Microsoft.Network/networkInterfaces 'nic1' failed with message '{
  "error": {
    "code": "InvalidResourceReference",
    "message": "Resource
/subscriptions/d35279cd-7ede-419c-b4f1-bc456ee6d803/resourceGroups/Net/providers/Microsoft.Network/virtualNetworks/vnet1/subnets/subnet1
referenced by resource /subscriptions/d35279cd-7ede-419c-b4f1-bc456ee6d803/resourceGroups/net/providers/Microsoft.Network/networkInterfaces/nic1
was not found. Please make sure that the referenced resource exists, and that both resources are in the same region.",
    "details": []
  }
}'
At line:1 char:1
+ New-AzResourceGroupDeployment -Name net -ResourceGroupName net `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet

VERBOSE: 4:18:53 PM - Checking deployment status in 5 seconds
VERBOSE: 4:18:58 PM - Resource Microsoft.Network/virtualNetworks 'vnet1' provisioning status is running
VERBOSE: 4:18:58 PM - Checking deployment status in 8 seconds
VERBOSE: 4:19:07 PM - Resource Microsoft.Network/virtualNetworks 'vnet1' provisioning status is succeeded
New-AzResourceGroupDeployment : 4:19:07 PM - Template output evaluation skipped: at least one resource deployment operation failed. Please list
deployment operations for details. Please see https://aka.ms/arm-debug for usage details.
At line:1 char:1
+ New-AzResourceGroupDeployment -Name net -ResourceGroupName net `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet
```

1. nic1 failed

2. Due to resource vnet1/subnets/subnet1 not existing yet

3. vnet1 was successful later on in deployment

Decoding dependency errors in Template - 2nd Deployment

```
VERBOSE: Performing the operation "Creating Deployment" on target "net".
VERBOSE: 4:06:23 PM - Template is valid.
VERBOSE: 4:06:26 PM - Create template deployment 'net'
VERBOSE: 4:06:26 PM - Checking deployment status in 5 seconds
VERBOSE: 4:06:32 PM - Resource Microsoft.Network/networkInterfaces 'nic1' provisioning status is running
VERBOSE: 4:06:32 PM - Checking deployment status in 5 seconds
VERBOSE: 4:06:37 PM - Resource Microsoft.Network/virtualNetworks 'vnet1' provisioning status is succeeded
VERBOSE: 4:06:37 PM - Checking deployment status in 5 seconds
VERBOSE: 4:06:42 PM - Checking deployment status in 5 seconds
VERBOSE: 4:06:48 PM - Checking deployment status in 5 seconds
VERBOSE: 4:06:53 PM - Checking deployment status in 5 seconds
VERBOSE: 4:06:58 PM - Checking deployment status in 5 seconds
VERBOSE: 4:07:04 PM - Checking deployment status in 5 seconds
VERBOSE: 4:07:09 PM - Checking deployment status in 5 seconds
VERBOSE: 4:07:15 PM - Checking deployment status in 5 seconds
VERBOSE: 4:07:20 PM - Checking deployment status in 5 seconds
VERBOSE: 4:07:25 PM - Checking deployment status in 5 seconds
VERBOSE: 4:07:31 PM - Resource Microsoft.Network/networkInterfaces 'nic1' provisioning status is succeeded

DeploymentName           : net
ResourceGroupName        : net
ProvisioningState        : Succeeded
Timestamp                : 5/1/2019 8:07:26 PM
Mode                     : Incremental
TemplateLink             :
Parameters               :
Outputs                  :
DeploymentDebugLogLevel  :
```

2. NICs depend on Vnets, which was already created successfully in previous deployment

1. Nic1 now deploys successfully on 2nd push with no code changes to template. Why?

Decoding dependency errors in Template - Solution

Add “dependsOn” to resource when the dependency is deployed in the same template

```
{
    "name": "nic1",
    "type": "Microsoft.Network/networkInterfaces",
    "apiVersion": "2018-08-01",
    "location": "[resourceGroup().location]",
    "dependsOn": [
        "vnet1"
    ],
    "properties": {
        "ipConfigurations": [
        {
        "name": "ipConfig1",
        "properties": {
          "privateIPAllocationMethod": "Dynamic",
          "subnet": {
           "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets', 'vnet1', 'subnet1')]"}}}]}},
{
    "name": "vnet1",
    "type": "Microsoft.Network/virtualNetworks",
    "apiVersion": "2018-08-01",
    "location": "[resourceGroup().location]",
    "dependsOn": [
        "ddosprotectionplan"
    ], […]
```

Decoding Circular Dependency errors in Template

```
VERBOSE: Performing the operation "Creating Deployment" on target "net".
New-AzResourceGroupDeployment : 5:57:15 PM - Error: Code=InvalidTemplate; Message=Deployment template validation failed:
'Circular dependency detected on resource: '/subscriptions/d35279cd-7ede-419c-b4f1-bc456ee6d803/resourceGroups/net/providers/Microsoft.Network/virtualNetworks/vnet1'. Please see https://aka.ms/arm-template/#resources for usage details.'.
At line:1 char:1
+ New-AzResourceGroupDeployment -Name net -ResourceGroupName net `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet

New-AzResourceGroupDeployment : The deployment validation failed
At line:1 char:1
+ New-AzResourceGroupDeployment -Name net -ResourceGroupName net `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (:) [New-AzResourceGroupDeployment], InvalidOperationException
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet
```

Decoding Circular dependency errors in Template (Cont.)

- 2 resources claim to depend on eachother which results in neither deploying

```
{
    "name": "nic1",
    "type": "Microsoft.Network/networkInterfaces",
    "apiVersion": "2018-08-01",
    "location": "[resourceGroup().location]",
    "dependsOn": [
        "vnet1"
    ],
    "properties": {
        "ipConfigurations": [
        {
        "name": "ipConfig1",
        "properties": {
            "privateIPAllocationMethod": "Dynamic",
            "subnet": {
           "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets', 'vnet1', 'subnet1')]"}}}]}},
{
    "name": "vnet1",
    "type": "Microsoft.Network/virtualNetworks",
    "apiVersion": "2018-08-01",
    "location": "[resourceGroup().location]",
    "dependsOn": [
        "ddosprotectionplan",
        "nic1"
    ], […]
```

Decoding invalid template errors – Parameter value input

```
New-AzResourceGroupDeployment -Name storage
-ResourceGroupName storage `
-TemplateFile '.\storage.json' `
  -StorageAcctSKU "Premium_LRS" -Verbose
```

```
VERBOSE: Performing the operation "Creating Deployment" on target "storage".
New-AzResourceGroupDeployment : 9:31:58 PM - Error: Code=InvalidTemplate; Message=Deployment template validation failed:
'The provided value 'Premium_LRS' for the template parameter 'StorageAcctSKU' at line '7' and column '27' is not valid.
The parameter value is not part of the allowed value(s): 'Standard_LRS,Standard_GRS,Standard_RAGRS'.'.
At line:1 char:1
+ New-AzResourceGroupDeployment -Name storage -ResourceGroupName storag ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet

New-AzResourceGroupDeployment : The deployment validation failed
At line:1 char:1
+ New-AzResourceGroupDeployment -Name storage -ResourceGroupName storag ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (:) [New-AzResourceGroupDeployment], InvalidOperationException
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet
```

Decoding invalid template errors – Parameter value input
(Cont.)

Parameter value not matching parameter definition

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "StorageAcctSKU": {
           "type": "string",
           "allowedValues": ["Standard_LRS", "Standard_GRS", "Standard_RAGRS"]
        }
    },
    "resources": [
        {
            "name": "[uniqueString(resourceGroup().id)]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-11-01",
            "location": "[resourceGroup().location]",
            "sku":{
                "name": "[parameters('StorageAcctSKU')]"}}]
}
```

Our deployment parameter value was “Premium_LRS”

Decoding authorization errors

```
New-AzResourceGroupDeployment -Name storage
-ResourceGroupName storage `
-TemplateFile '.\storage.json' `
  -StorageAcctSKU "Standard_LRS" -Verbose
```

```
VERBOSE: Performing the operation "Creating Deployment" on target "storage".
New-AzResourceGroupDeployment : 11:26:08 PM - Error: Code=InvalidTemplateDeployment; Message=The template deployment
failed with error: 'Authorization failed for template resource 'fbtxjxl3y6sqy' of type
'Microsoft.Storage/storageAccounts'. The client 'chdafni@microsoft.com' with object id
'd30cdb0d-140e-432d-82be-441dacf8d1b4' does not have permission to perform action
'Microsoft.Storage/storageAccounts/write' at scope '/subscriptions/90934b9d-9304-4cc2-b376-308dcab7914e/resourceGroups/stor
age/providers/Microsoft.Storage/storageAccounts/fbtxjxl3y6sqy'.'.
At line:1 char:1
+ New-AzResourceGroupDeployment -Name storage -ResourceGroupName storag ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet

New-AzResourceGroupDeployment : The deployment validation failed
At line:1 char:1
+ New-AzResourceGroupDeployment -Name storage -ResourceGroupName storag ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (:) [New-AzResourceGroupDeployment], InvalidOperationException
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet
```

Decoding invalid template error – Incorrect Segment Length

- Root resource name length = length of resource type (-) 1
  - A "/" is an indicator of a new segment
  - Ex: Microsoft.Web/sites = length of 2. Therefore Name should be: length 1
- Nested resources name & type must be same length

```
VERBOSE: Performing the operation "Creating Deployment" on target "webapp".
New-AzResourceGroupDeployment : 9:53:14 AM - Error: Code=InvalidTemplate; Message=Deployment template validation failed:
'The template resource 'web' for type 'Microsoft.Web/sites/config' at line '34' and column '13' has incorrect segment
lengths. A nested resource type must have identical number of segments as its resource name. A root resource type must
have segment length one greater than its resource name. Please see https://aka.ms/arm-template/#resources for usage
details.'.
At line:1 char:1
+ New-AzResourceGroupDeployment -Name webapp -ResourceGroupName webapp  ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet

New-AzResourceGroupDeployment : The deployment validation failed
At line:1 char:1
+ New-AzResourceGroupDeployment -Name webapp -ResourceGroupName webapp  ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (:) [New-AzResourceGroupDeployment], InvalidOperationException
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploym
   entCmdlet
```

Decoding invalid template error – Incorrect Segment Length

- Answer: Nested resource name: web | Nested resource type: config
  "Microsoft.web/sites/" is implied since it is a nested resource
- Overall, nested resource Name length must match resource type length

```
{
    "name": "[uniqueString(resourceGroup().id)]",
    "type": "Microsoft.Web/sites",
    "apiVersion": "2016-08-01",
    "location": "[resourceGroup().location]",
    "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms', 'AppSvcPlan')]"
    ],
    "properties": {
        "name": "[uniqueString(resourceGroup().id)]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', 'AppSvcPlan')]"
    },
    "resources": [
        {
            "apiVersion": "2016-08-01",
            "name": "web",
            "type": "Microsoft.Web/sites/config",
            "dependsOn": [
                "[concat('Microsoft.Web/sites/', uniqueString(resourceGroup().id))]"
            ],
            "properties": {
                "foo": "bar"}}]}
```

Root resource

Length: 1 (just a single word name)

Length: 2 (The "/" is a separator of length)

Nested resource

Length: 1 (just a single word name)

Length: 3

Decoding ResourceID error – one multi-segmented argument

- ResourceID function should include full resource type length & each namespace is in its own segment

  Goal of template: Attach 'nic' to the virtual network 'vnet' & the subnet 'subnet1'

  - Incorrect:

The value of "concat" function includes a segment '/' which is not allowed per the error message

```
"[resourceId('Microsoft.Network/virtualNetworks',concat('vnet','/subnets/subnet1'))]"
```

  - Correct:

```
"[resourceId('Microsoft.Network/virtualNetworks/subnets','vnet', 'subnet1')]"
```

```
VERBOSE: Performing the operation "Creating Deployment" on target "net".
VERBOSE: 11:03:53 PM - Template is valid.
VERBOSE: 11:03:54 PM - Create template deployment 'nic'
VERBOSE: 11:03:54 PM - Checking deployment status in 5 seconds
New-AzResourceGroupDeployment : 11:04:00 PM - Resource Microsoft.Network/networkInterfaces 'nic' failed with message '{
  "error": {
    "code": "InvalidTemplate",
    "message": "Unable to process template language expressions for resource
'/subscriptions/11504d44-374a-4e45-b714-13fc9ac28c60/resourceGroups/net/providers/Microsoft.Network/networkInterfaces/nic' at line '9' and column '9'. 'Unable
to evaluate template language function 'resourceId': function requires exactly one multi-segmented argument which must be resource type including resource
provider namespace. Current function arguments 'Microsoft.Network/virtualNetworks,vnet/subnets/subnet1'. Please see
https://aka.ms/arm-template-expressions/#resourceid for usage details.'"
  }
}'
At line:1 char:1
+ New-AzResourceGroupDeployment -Name nic -ResourceGroupName net `
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-AzResourceGroupDeployment], Exception
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.ResourceManager.Cmdlets.Implementation.NewAzureResourceGroupDeploymentCmdlet
```

Decoding invalid template error – Debug Logging

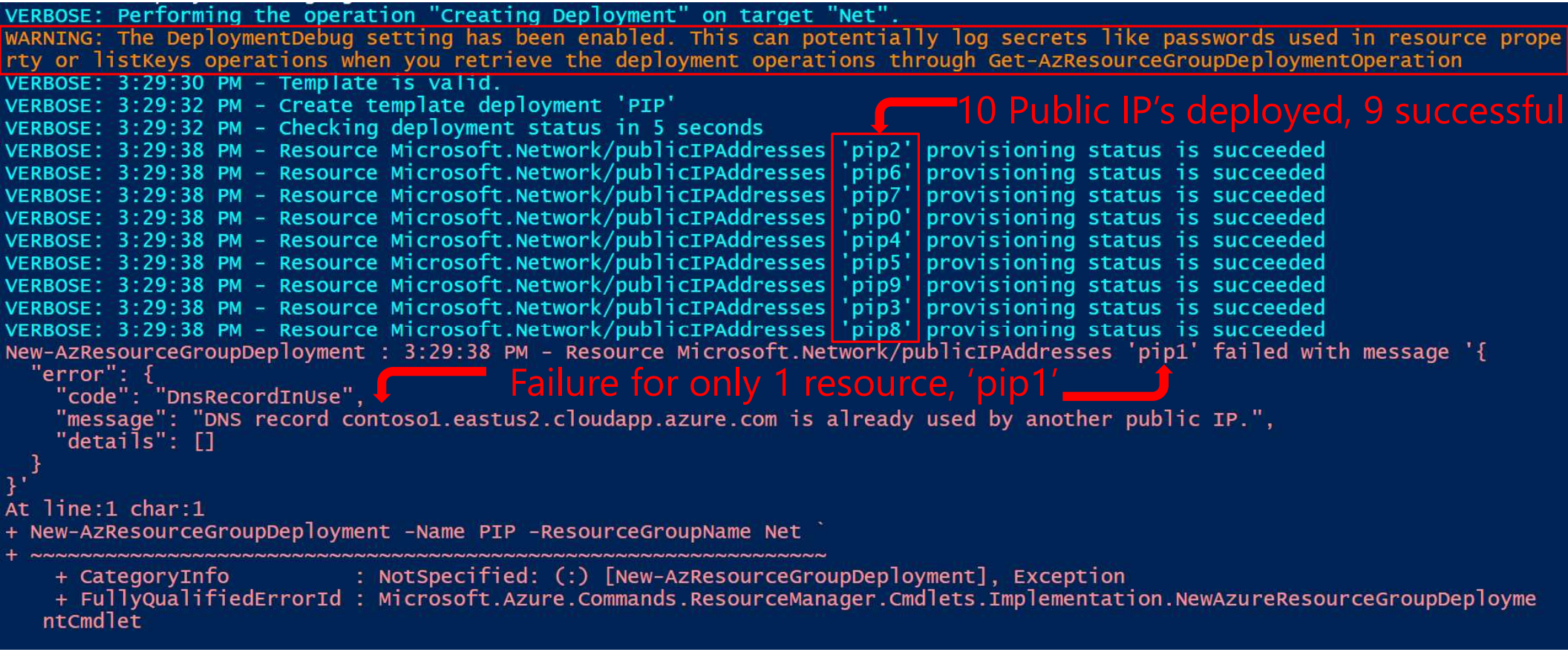
- Using "Copy" or large templates can make it difficult to decipher which resource(s) are erroring and why
- DeploymentDebugLogLevel captures resource provider (RP) request/response operations

  Should only be used for troubleshooting as it can log secrets

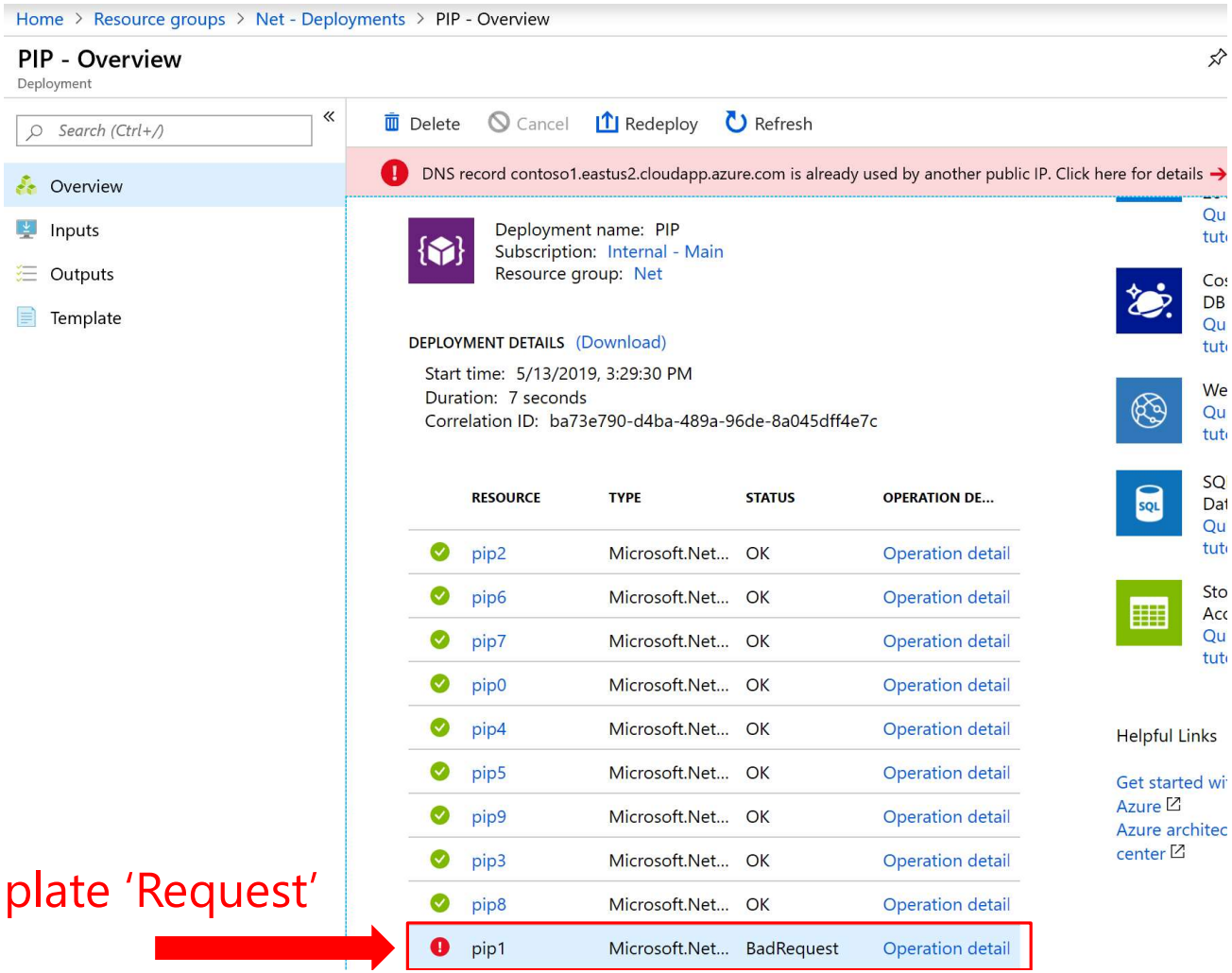
```
New-AzResourceGroupDeployment -Name PIP
-ResourceGroupName Net `
-TemplateFile ".\pip.json" -PIPCount 10 `
-Verbose -DeploymentDebugLogLevel All
```

```
VERBOSE: Performing the operation "Creating Deployment" on target "Net".
WARNING: The DeploymentDebug setting has been enabled. This can potentially log secrets like passwords used in resource prope
rty or listKeys operations when you retrieve the deployment operations through Get-AzResourceGroupDeploymentOperation
```

Decoding invalid template error – Debug Logging (Cont.)



Decoding invalid template error – Debug Logging (Cont.)



We want the template 'Request' of this operation

Decoding invalid template error – Debug Logging (Cont.)

```
$operations = Get-AzResourceGroupDeploymentOperation
-DeploymentName PIP -ResourceGroupName Net

foreach($operation in $operations)

{
    Write-Host $operation.id
    Write-Host "Request:"
 $operation.Properties.Request | ConvertTo-Json -Depth 10
    Write-Host "Response:"
 $operation.Properties.Response | ConvertTo-Json -Depth 10
}
```

Pulls all logged content from deployment

'foreach' function isn't required but simplifies everything by outputting each operation performed by ARM

Template code submitted

Error or Success response

Decoding invalid template error – Debug Logging (Cont.)

Each operation will list the request and response, below is the entry that threw an error

```
Request:
{
    "content":  {
                    "location":  "eastus2",
                    "properties":  {
                                       "publicIPAllocationMethod":  "Dynamic",
                                       "dnsSettings":  {
                                                           "domainNameLabel":  "contoso1"
                                                       }
                                   }
                }
}
Response:
{
    "content":  {
                    "error":  {
                                  "code":  "DnsRecordInUse",
                                  "message":  "DNS record contoso1.eastus2.cloudapp.azure.com is already used by another public IP.",
                                  "details":  [

                                              ]
                              }
                }
}
```

The 'requested' properties of the public IP resource. This is our "hint" for where in the template the failure occurred

Decoding invalid template error – Debug Logging - Compare

```
{
    "name": "[concat('pip',copyIndex())]",
    "type": "Microsoft.Network/publicIPAddresses",
    "apiVersion": "2018-08-01",
    "location": "[resourceGroup().location]",
    "copy": {
        "name": "pip",
        "count": "[parameters('PIPCount')]"
    },
    "properties": {
        "publicIPAllocationMethod": "Dynamic",
        "dnsSettings": {
            "domainNameLabel": "[concat('contoso',copyIndex())]"}}
}
```

Template code

Ultimately it means the template needs to be re-architected to ensure global uniqueness for this property value

```
Request:
{
    "content":  {
                    "location":  "eastus2",
                    "properties":  {
                                       "publicIPAllocationMethod":  "Dynamic",
                                       "dnsSettings":  {
                                                           "domainNameLabel":  "contoso1"
                                                       }
```

```
Response:
{
    "content":  {
                    "error":  {
                                  "code":  "DnsRecordInUse",
                                  "message":  "DNS record contoso1.eastus2.cloudapp.azure.com is already used by another public IP.",
                                  "details":  [
                                                  ]
```

The deployment operation request / response

Lab: Template & Deployment troubleshooting

Investigate a variety of template failures to bring them into working order

Knowledge Check

- How can you accurately determine where in your template a deployment failure occurred?

  With DeploymentDebugLogLevel, the request and response can be found which shows what code section failed

Microsoft

Section 7
ARM Templates – Creating templates from scratch - architecture goals

Microsoft Services

Objectives

After completing this learning unit, you will

- Understand different template architecture strategies
- How to find resource type reference frameworks
- How to declare iterative updates

Choosing a deployment strategy

Different styles of architecting templates

- Modular – purpose specific, slim template, could be linked multiple times
  Ex: Each resource type is its own template | Ex: Base environment of resources types
- Generic – non hardcoded resource types, parameters determine most settings
  Ex: Deploy a VM tier & common dependencies like NIC, Availability set, Backup/DR integration
- Full Solution – All components needed to support an application, minimal input
  Ex: SharePoint farm with all tiers
- Business strategy
  - Skill specific IT departments manage their templates for resource types they manage
    Ex: Teams – Database, Networking, Security, Monitoring, Analytics
  - A department manages their entire Azure offering
    Ex: Engineering, Human Resources, Customer Service, Financing
- Hybrid of one of more styles

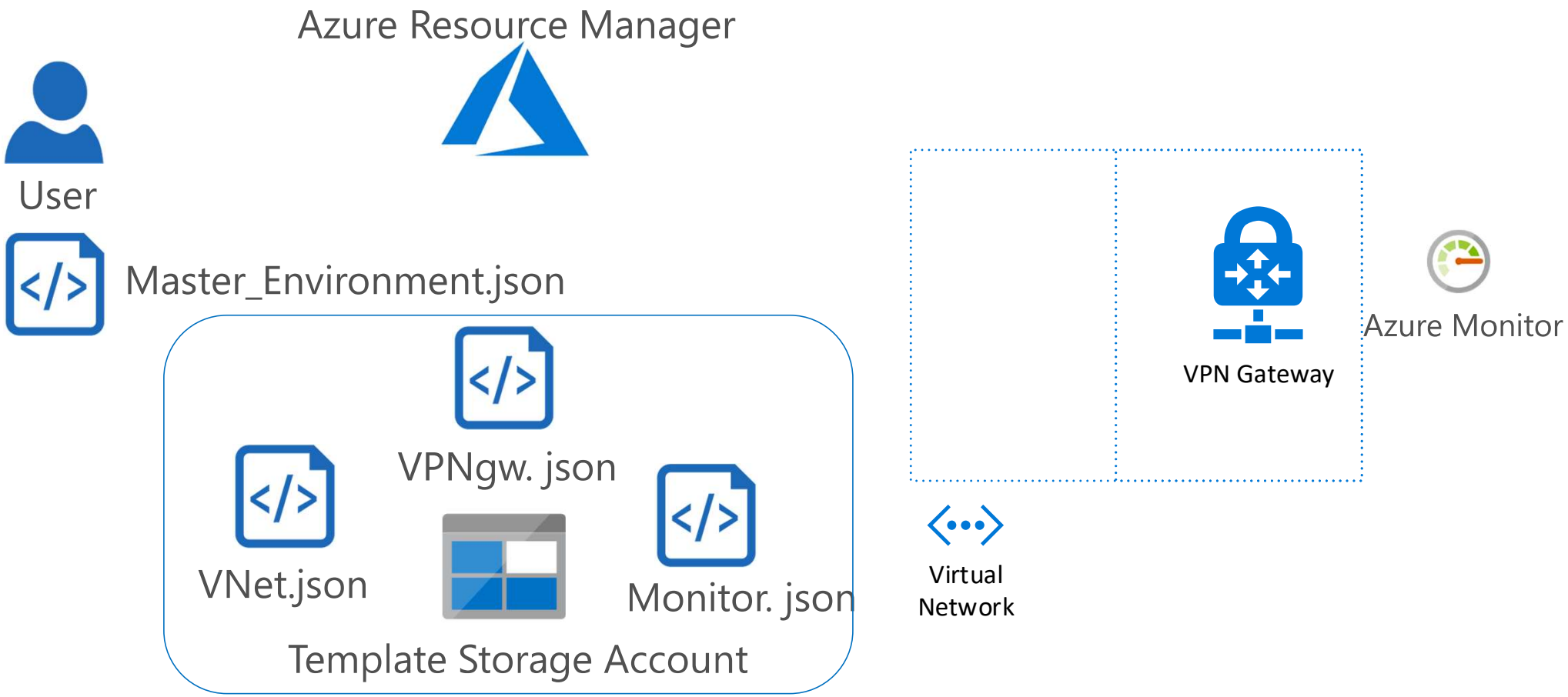
There's no right answer nor are you locked into that choice

Template Strategy - Module

- "Plug and play" templates - Select what "features" (templates) you want
- Linked templates can provide reference output for use in parent template

  Ex: Output the dynamically assigned IP as input for other templates
- Can be used as a baseline for initiatives

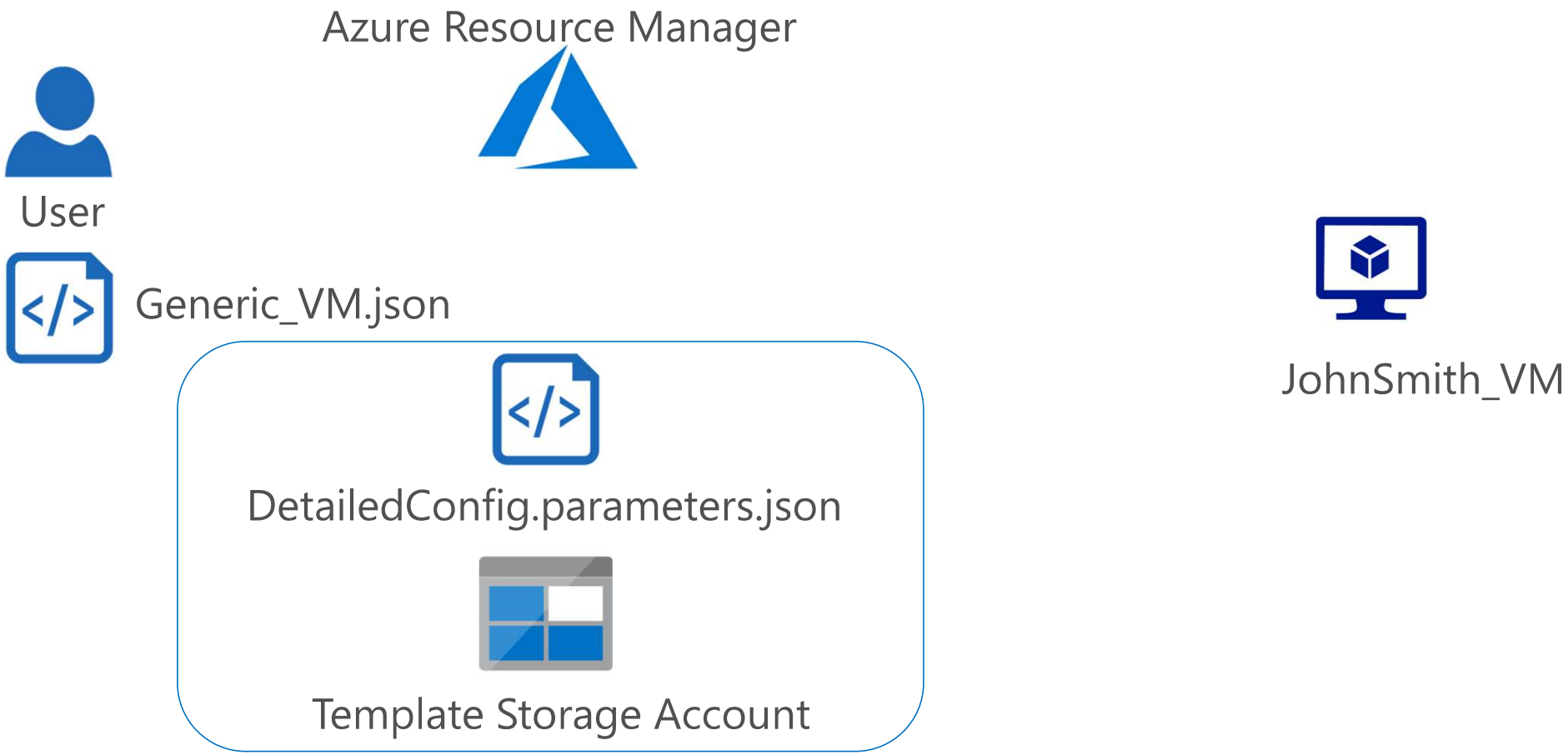
Choosing a deployment strategy - Modular



Template Strategy - Generic

- Minimal template code, requires more parameter code
- Very flexible since parameters are describing the config
  - Ex: # of disks, dynamic or static public IP, Windows unattended initial start-up
- Minimal template file = minimal upkeep
- Works well with programmatically generating input
  - Ex: PowerShell pulls input from external systems and converts it to the JSON parameter file
- Ensures key components are always included
  - Ex: Company wide VM template used for all VM deployments
- Takes the longest for creating parameter files

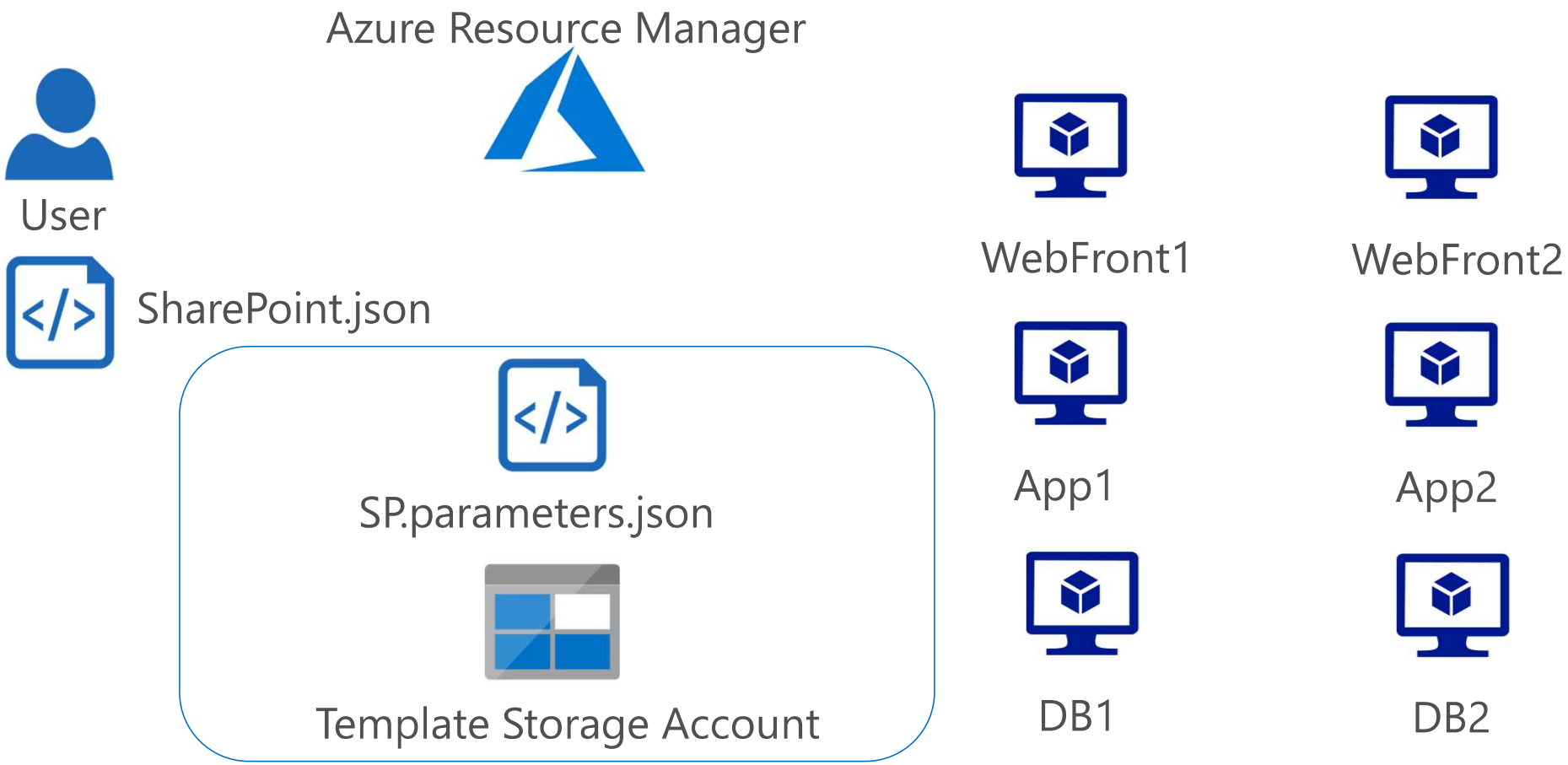
Choosing a deployment strategy - Generic



Template Strategy – Full Solution

- Minimal input from parameters
- Easier to troubleshoot since the template includes all needed resources
- Difficult to maintain enterprise standards as each full solution template will require changes
- Takes the longest to create initial template file

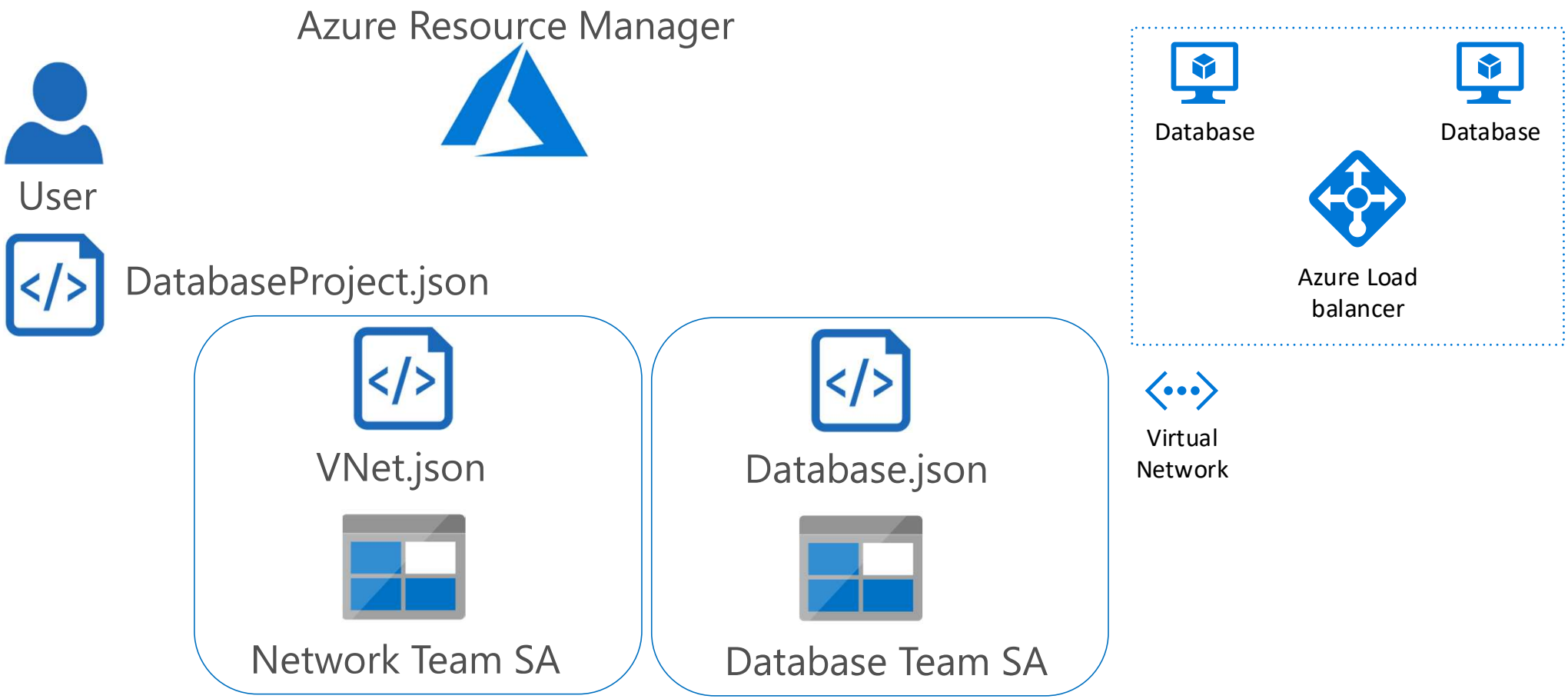
Choosing a deployment strategy – Full Solution



Template Strategy – Business Strategy

- Allows responsibility of Azure resource types & features to be maintained by the right audience
- Departments and teams can focus their skillsets on what matters to their business
- Better governance as each template originated from responsible party
- Users are the consumers of the templates
- Troubleshooting can be harder since others maintain it

Choosing a deployment strategy – Business Specific



Resource Type – Frameworks – How do I?

- Insert a resource type I've never worked with before or insert a new resource feature?
- Azure publishes a 'reference book' of each resource type
- Each resource type's API version contains the schema of all possible properties
- See properties not mentioned in Portal or PowerShell
- More up-to-date than 'user snippets' which are static
- Intellisense usually reads the schema to provide feedback
- Bookmark it: https://docs.microsoft.com/en-us/azure/templates/

Filter by title

- Data Lake Analytics
- Data Lake Store
- Deployment Manager
- DevTest Labs
- Domain Registration
- DNS
  - All resources
  - 2018-05-01
    - Dns Zones
      - **Dns Zones**
      - A
      - AAAA
      - CAA
      - CNAME
      - MX
      - NS
      - PTR
      - SOA
      - SRV
      - TXT
  - 2017-10-01
  - 2017-09-01
  - 2016-04-01

Resource Provider

API Version

Resource Type

Microsoft.Network dnsZones template reference

06/26/2019 • 2 minutes to read •

API Versions: 2018-05-01

Template format

To create a Microsoft.Network/dnsZones resource, add the following JSON to the resources section of your template.

JSON

Copy

```
{
  "name": "string",
  "type": "Microsoft.Network/dnsZones",
  "apiVersion": "2018-05-01",
  "location": "string",
  "tags": {},
  "properties": {
    "zoneType": "string",
    "registrationVirtualNetworks": [
      {
        "id": "string"
      }
    ],
    "resolutionVirtualNetworks": [
      {
        "id": "string"
      }
    ]
  },
  "resources": []
}
```

Copy / paste into your JSON code editor

Each property in schema is mentioned in detail on the page

JSON for all properties of this resource type's schema version

Property values

The following tables describe the values you need to set in the schema.

Microsoft.Network/dnsZones object

Name	Type	Required	Value
name	string	Yes	
type	enum	Yes	Microsoft.Network/dnsZones
apiVersion	enum	Yes	2018-05-01
location	string	Yes	Resource location.
tags	object	No	Resource tags.
properties	object	Yes	The properties of the zone. - ZoneProperties object
resources	array	No	TXT SRV SOA PTR NS MX CNAME CAA AAAA A

ZoneProperties object

Name	Type	Required	Value
zoneType	enum	No	The type of this DNS zone (Public or Private). - Public or Private
registrationVirtualNetworks	array	No	A list of references to virtual networks that register hostnames in this DNS zone. This is a only when ZoneType is Private. - SubResource object
resolutionVirtualNetworks	array	No	A list of references to virtual networks that resolve records in this DNS zone. This is a only when ZoneType is Private. - SubResource object

Explanation of each property

Resource Type – Frameworks – VS Code – copy / paste

Copied from reference page

DNSZone.json

C: ▸ temp ▸ {} DNSZone.json ▸ [ ] resources ▸ {} 0 ▸ {} properties

```
1  {
2      "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3      "contentVersion": "1.0.0.0",
4      "parameters": {},
5      "variables": {},
6      "resources": [
7          {
8              "name": "string",
9              "type": "Microsoft.Network/dnsZones",
10             "apiVersion": "2018-05-01",
11             "location": "string",
12             "tags": {},
13             "properties": {
14               "zoneType": "string",
15               "registrationVirtualNetworks": [
16                 {
17                   "id": "string"
18                 }
19               ],
20               "resolutionVirtualNetworks": [
21                 {
22                   "id": "string"
23                 }
24               ]
25             },
26             "resources": []
27         }
28     ],
29     "outputs": {}
30 }
```

Let Intellisense do the rest!

Delete what you don't need

Resource Type – Frameworks – VS Code – trimmed code

DNSZone.json

C: ▸ temp ▸ DNSZone.json ▸ ...

```
1  {
2      "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3      "contentVersion": "1.0.0.0",
4      "parameters": {},
5      "variables": {},
6      "resources": [
7          {
8              "name": "contoso.com",
9              "type": "Microsoft.Network/dnsZones",
10             "apiVersion": "2018-05-01",
11             "location": "global",
12             "tags": {},
13             "properties": {
14               "zoneType": "Public"
15             },
16             "resources": []
17         }
18     ],
19     "outputs": {}
20 }
```

Resource Type - Frameworks

- Use the reference site to ensure you're using the latest possible Resource features
- Repeat this process when building from scratch for each resource type
- Create new User Snippets for your common Resources
  - Alternatively, see if the GitHub repo for Azure VSCode snippets has been updated
- Deploying the template is the best way to receive quality feedback as it'll be evaluated by ARM

Iterative Updates

Updating your environment can take many forms:

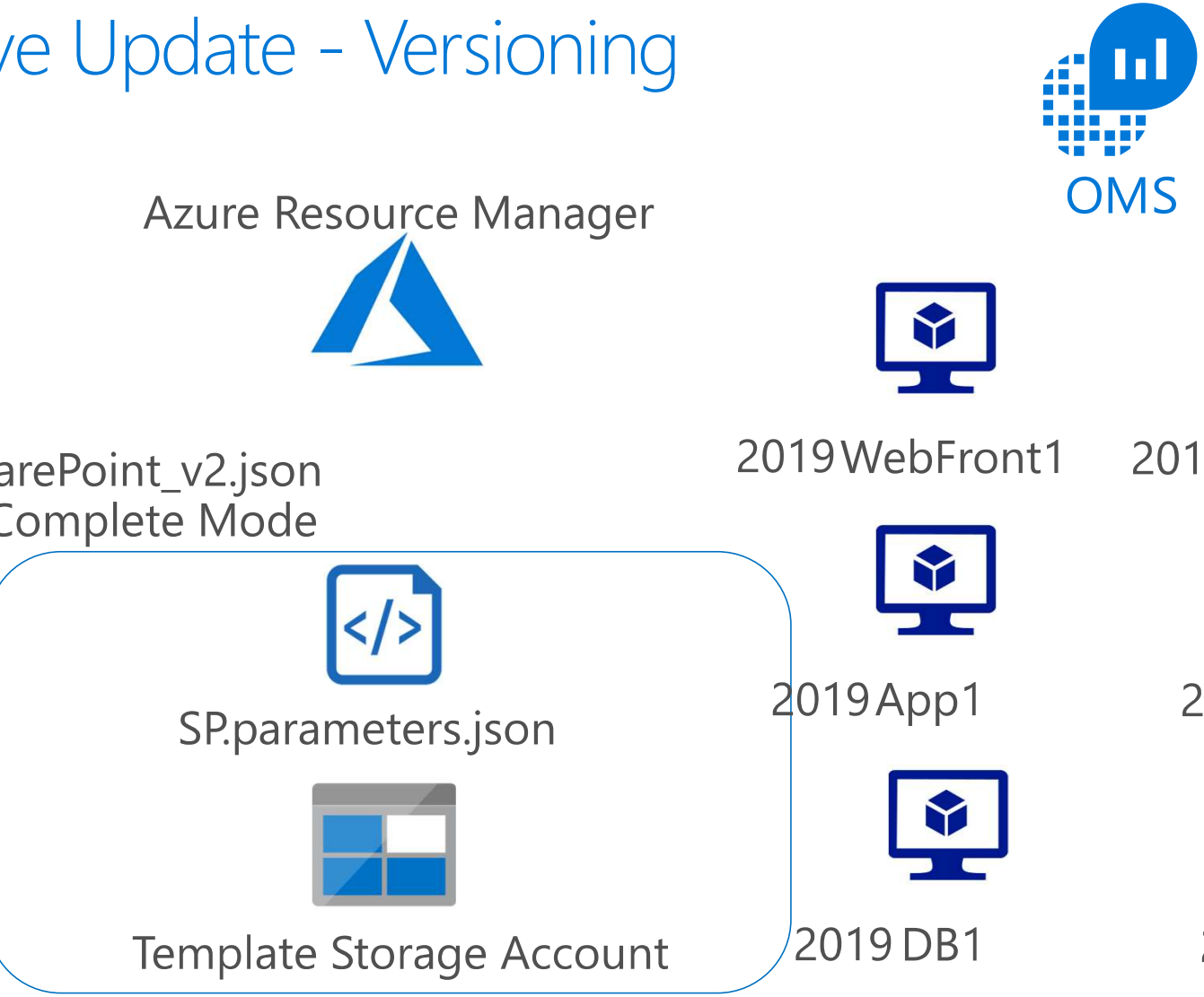
- Destroy/re-deploy with every new version of code
  Ensures a clean slate each time while decommissioning previous iterations
- Update a resource in-flight during initial deployment
  - Needed property values aren't known until after initial deployment
  - Ex: Setting a NIC's dynamic IP to static requires acquiring the existing IP first
- Re-configuring already deployed resources
  Template or Parameter changes that need to roll out to the original resources

Iterative Updates – Declarative – Warning!

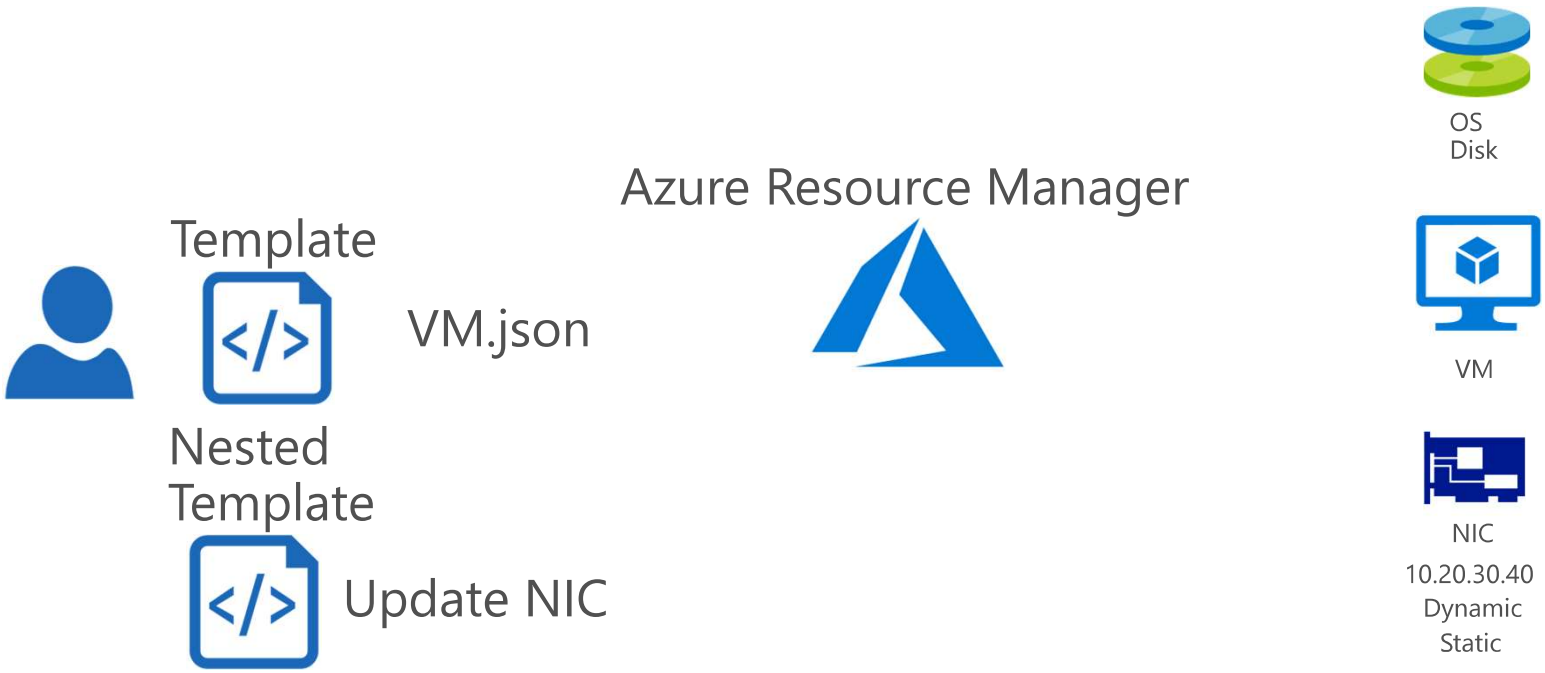
The template always defines the declared end state.

Most resource properties are mandatory and therefore their values will be stripped off if not included in each deployment

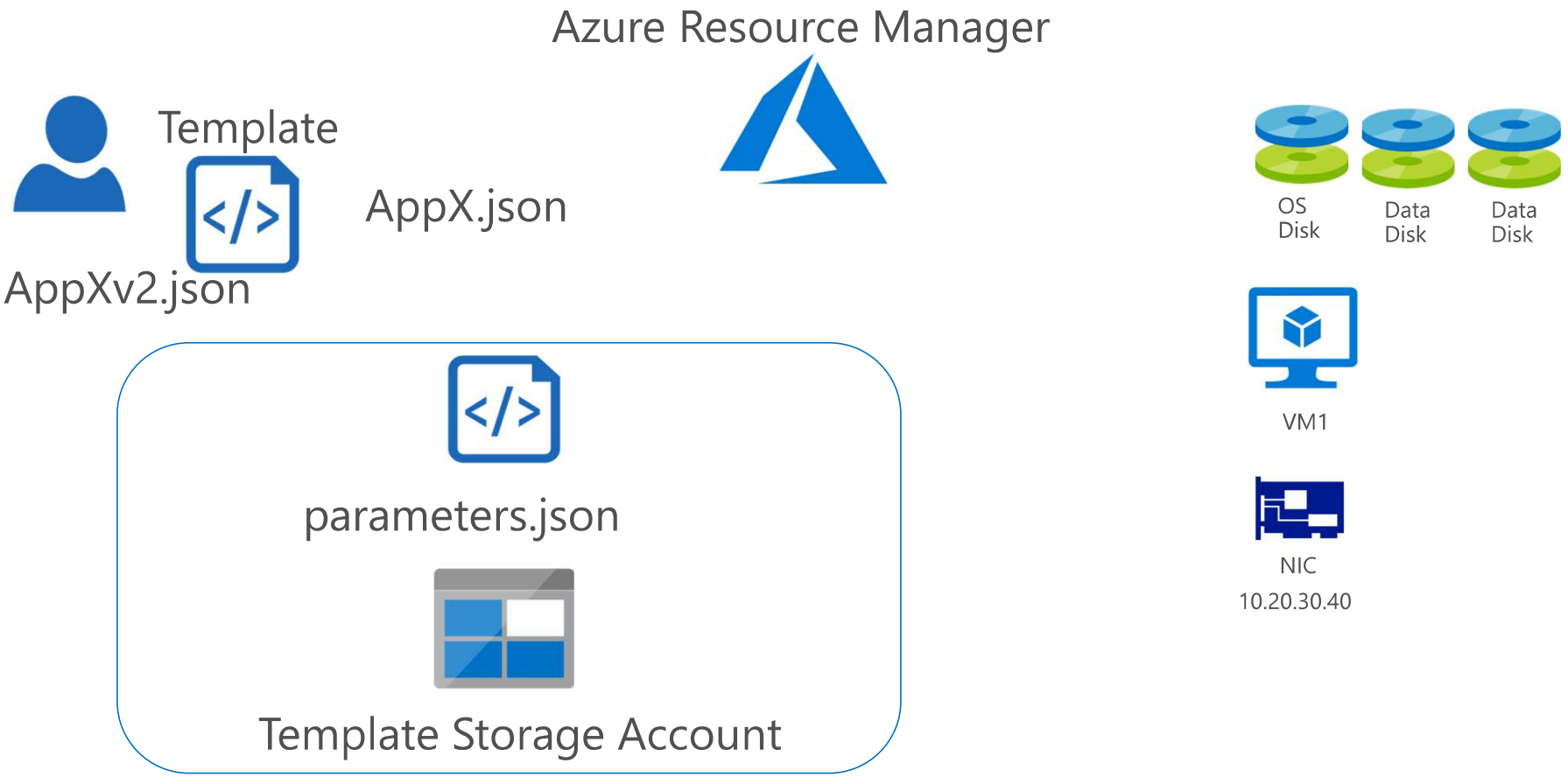
Iterative Update - Versioning



Iterative Update – in-flight re-deployments



Iterative Update – Re-configuration



Lab: Creating templates from scratch

Implement the various strategies to create templates from scratch to see the differences in their approaches

Knowledge Check

- Why are User Snippets not reliable in the long term?

  They are static and therefore not updated with the latest capabilities of a resource
- What is the correct way to write a template from scratch?

  There is no correct way, however it should be manageable and scalable to the extent that overhead allows. Picking a strategy is not one size fits all in the Enterprise. Previously established Governance will ensure the template strategies are successful