Microsoft

AZ-400.00 Learning Path 06: Manage infrastructure as code using Azure and DSC

Agenda

- Module 01: Explore infrastructure as code and configuration management.
- Module 02: Create Azure resources using Azure Resource Manager templates.
- Module 03: Implement Bicep.
- Module 04: Create Azure resources by using Azure CLI.
- Module 05: Explore Azure Automation with DevOps.
- Module 06: Implement Desired State Configuration (DSC).
- Labs & Learning Path review and takeaways.

Learning Path overview

Learning objectives

After completing this Learning Path, students will be able to:

1. Apply infrastructure and configuration as code principles
2. Deploy and manage infrastructure using Microsoft automation technologies such as ARM templates, Azure CLI, DSC and Azure Automation

Module 01: Explore infrastructure as code and configuration management

Explore environment deployment

Manual deployment:

- Snowflake servers
- Deployment steps vary by environment
- More verification steps and more elaborate manual processes
- Increased documentation to account for differences
- Deployment on weekends to allow time to recover from errors
- Slower release cadence to minimize pain and long weekends

Infrastructure as code:

- Consistent servers between environments
- Environments created or scaled easily
- Fully automate creation and updates of environments
- Transition to immutable infrastructure
- Use blue/green deployments
- Treat servers as commodities, not pets

Examine environment configuration

Manual configuration:

- Configuration bugs difficult to Identify
- Error prone
- More verification steps and more elaborate manual processes
- Increased documentation
- Deployment on weekends to allow time to recover from errors
- Slower release cadence to minimize requirement for long weekends

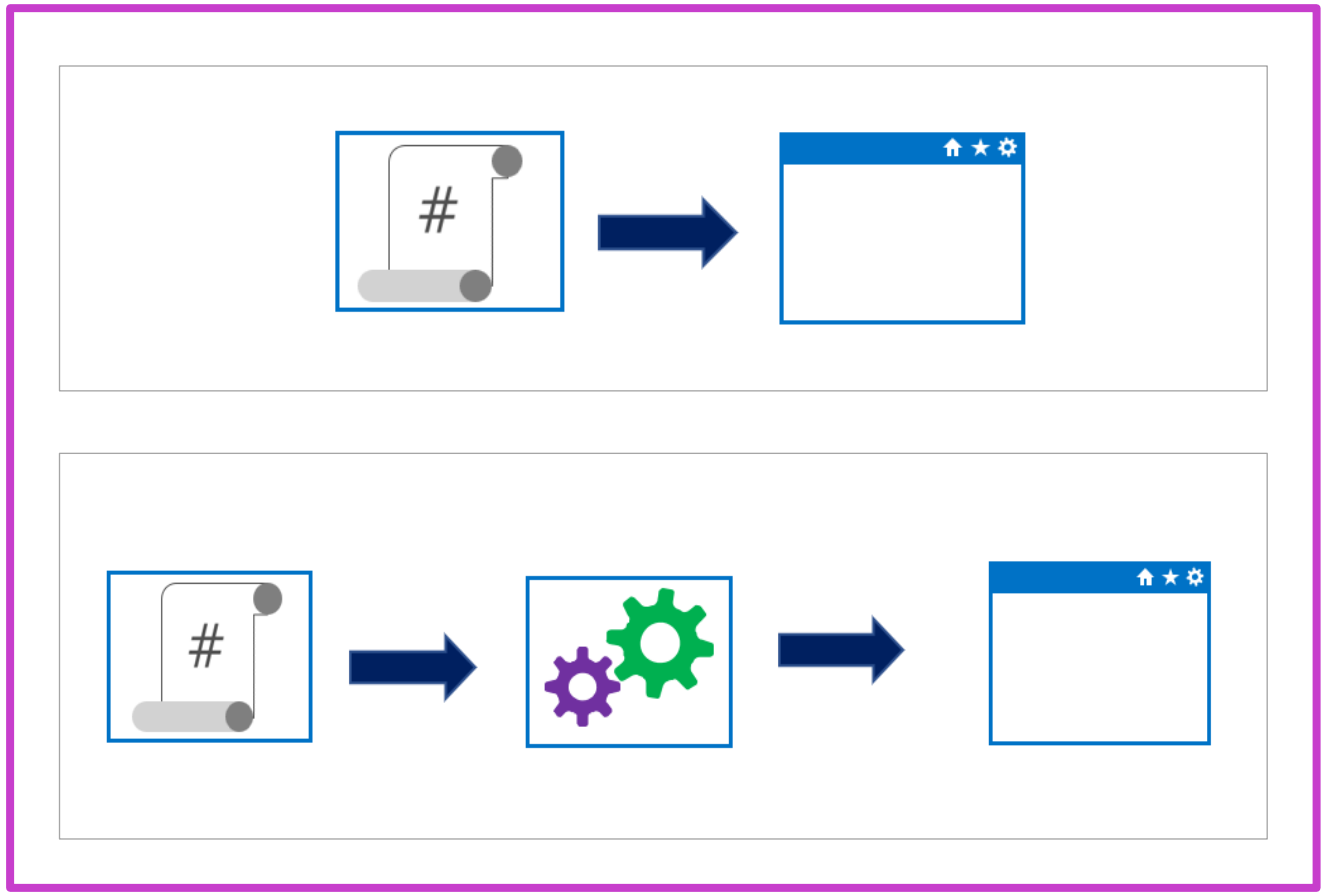
Configuration as code:

- Bugs easily reproducible
- Consistent configuration
- Increase deployment cadence to reduce amount of incremental change
- Treat environment and configuration as executable documentation

Understand imperative versus declarative configuration

Approaches to implementing infrastructure and configuration as code

- **Declarative:**
  Functional
  Defines **what** the final state should be
- **Imperative:**
  Procedural
  Defines **how** to achieve that final state



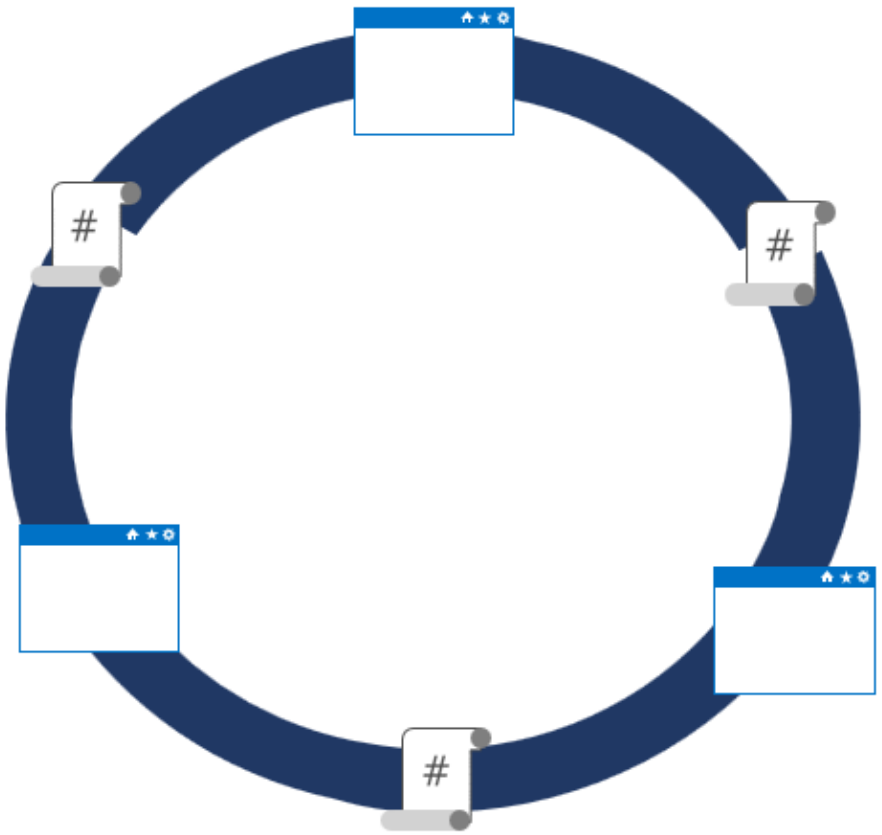
Understand idempotent configuration

Idempotence – Definition:

- Mathematical term used in the context of infrastructure and configuration as code
- Ability to apply one or more operations against a resource, resulting in the same outcome

To attain idempotence:

- Automatically configure and reconfigure an existing set of resources, or
- Discard existing resources and spin up a fresh environment



Module 02: Create Azure resources using Azure Resource Manager templates

Why use Azure Resource Manager templates?

- Make deployments faster and more repeatable
- Improve consistency by providing a common language
- Enable you to deploy multiple resources in the correct order by mapping out resource dependencies
- Reduce manual, error-prone tasks
- Templates can be linked together to provide a modular solution (might build on QuickStart Templates)

Explore template components

1. JSON data stored as an object in text Collection of key-value pairs

2. Templates can contain the following sections:
   - Parameters
   - Variables
   - Functions
   - Resources
   - Outputs

Manage dependencies

Some resources will depend on other resources before you can deploy them

Define this relationship by marking the dependency with the **dependsOn** element

```
129         "type": "Microsoft.Compute/virtualMachines",
130         "name": "[variables('vmName')]",
131         "location": "[parameters('location')]",
132         "apiVersion": "2018-10-01",
133         "dependsOn": [
134           "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
135           "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
136         ],
```

Modularize templates

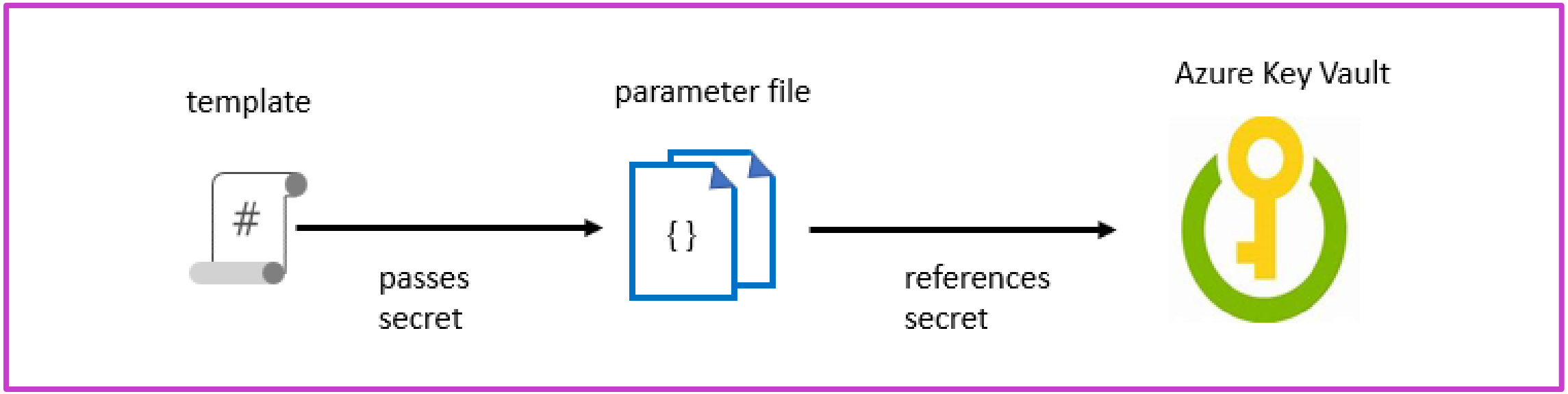
Best practice: Modularize templates into individual components:
Use linked templates to break the solution into individual pieces
Reuse those elements across different deployments

```
"resources": [
  {
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "properties": {
        "mode": "Incremental",
        "templateLink": {
            "uri":"https://linkedtemplateek1store.blob.core.windows.net/linkedtemplates/linkedStorageAccount.json?sv=2
            se=2018-12-31T14%3A32%3A29Z&sp=r"
        },
        "parameters": {
            "storageAccountName":{"value": "[variables('storageAccountName')]"},
            "location":{"value": "[parameters('location')]"}
        }
    }
  },
```

Manage secrets in templates

When passing a secure value (e.g., a password) as a parameter during deployment:

- Create a key vault and secret using Azure CLI or PowerShell
- Enable Azure Resource Manager access for template deployment
- Reference the key pair in the parameter file, **not** the template
- Enable access to the secret. **Owner** and **Contributor** roles grant access
- Deploy the template and pass in the parameter file



Module 03: Implement Bicep

What is Bicep?

Azure Bicep is the next revision of **ARM templates** designed to solve some of the issues developers were facing when deploying their resources to Azure.

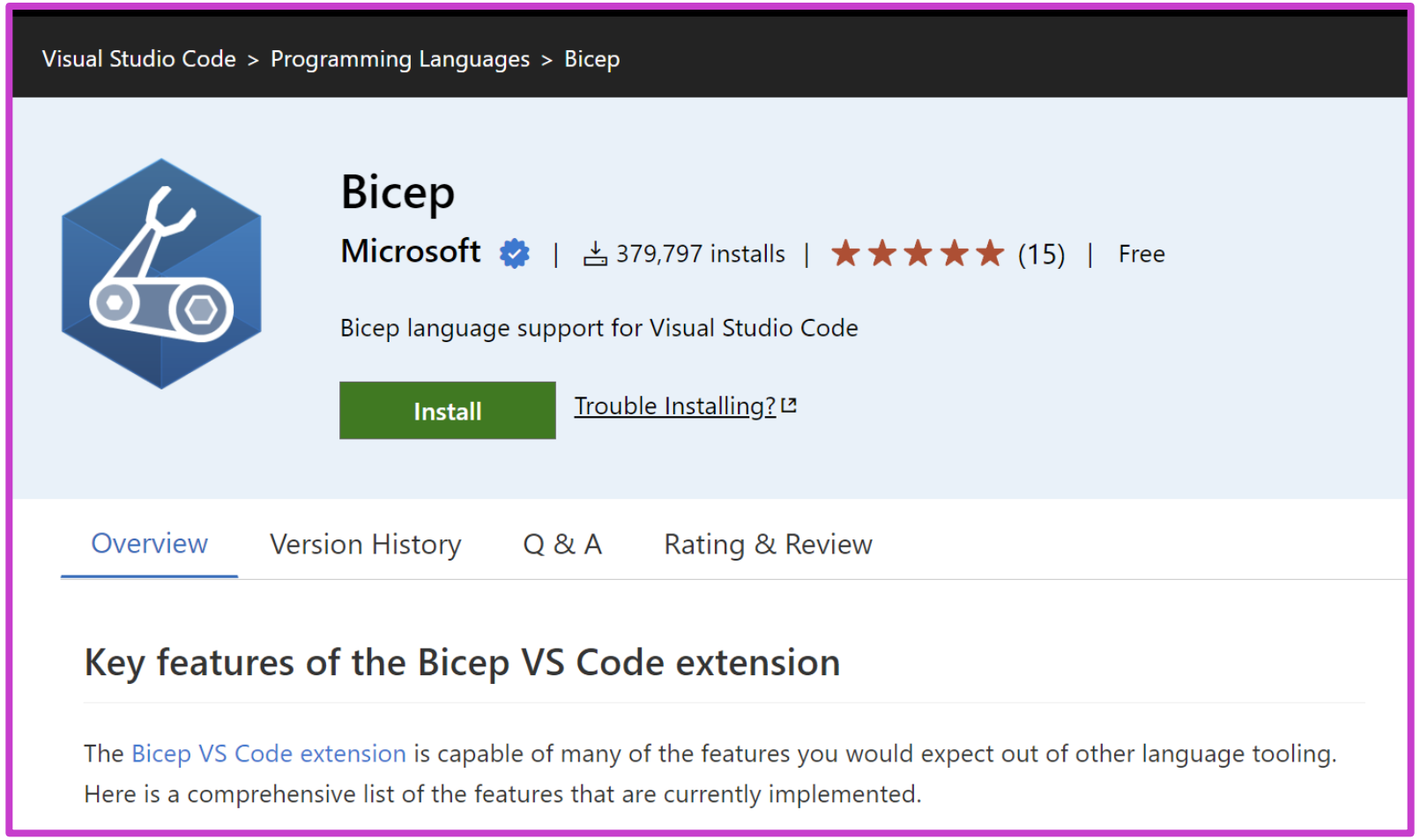
Note: Beware that when converting ARM templates to Bicep, there might be issues since it's still a work in progress.

Code

```
param storageName string =
'stg${uniqueString(resourceGroup().id)}'
param location string = resourceGroup().location

resource storageaccount
'Microsoft.Storage/storageAccounts@2021-02-01' = {
    name: 'name'
    location: location
    kind: 'StorageV2'
    sku: {
        name: 'Premium_LRS'
    }
}
```

Install Bicep

- **Install the Bicep CLI or the [Visual Studio Code Extension](#).**
- **The extension provides language support, IntelliSense, and linting support.**
- az bicep install
  choco install bicep
  winget install -e --id Microsoft.Bicep
  bicep --help



Understand Bicep file structure and syntax

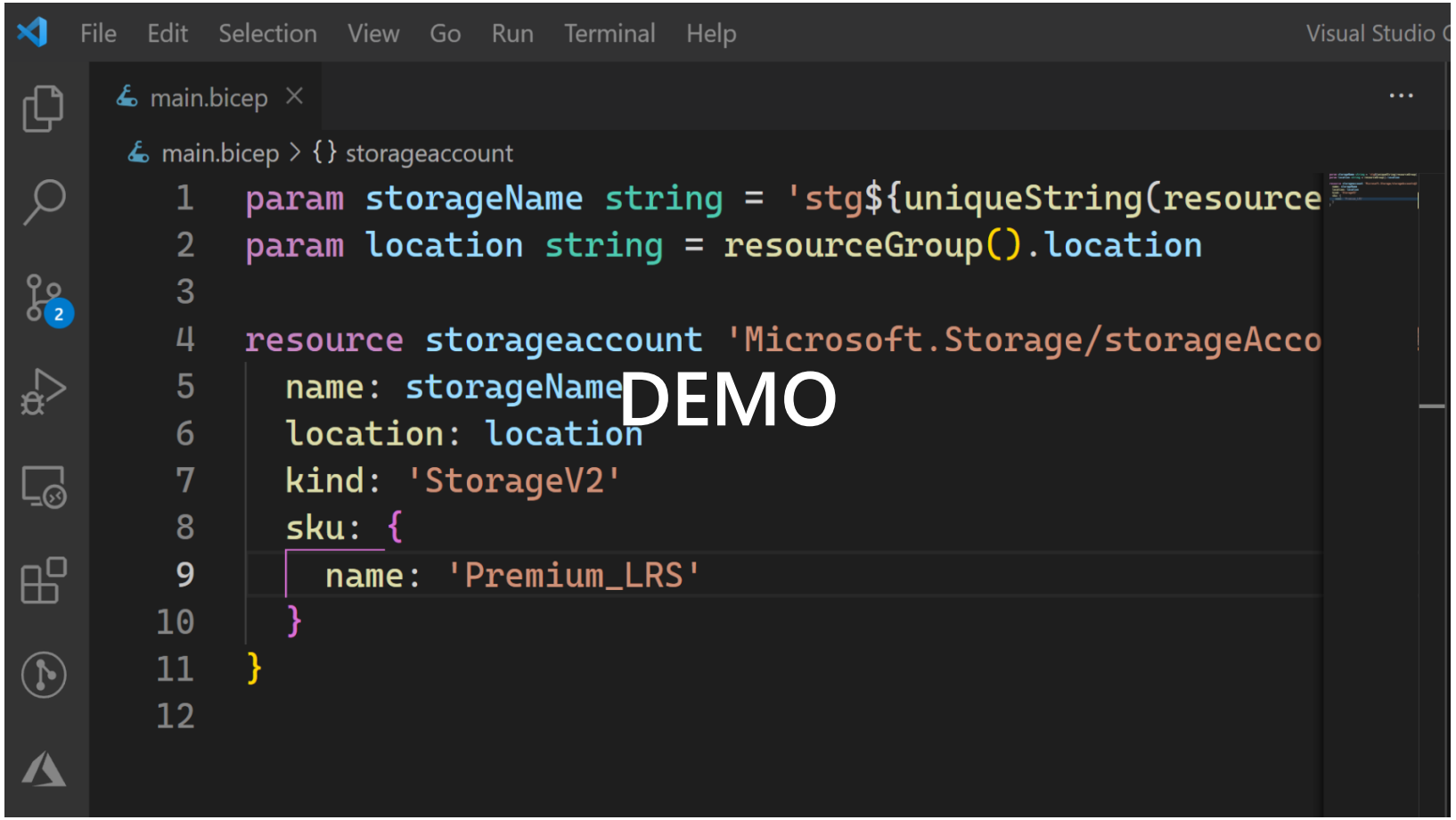
Azure Bicep comes with its own syntax, however, it's easy to understand and follow:

- Scope.
- Parameters.
- Variables.
- Resources.
- Modules.
- Outputs.

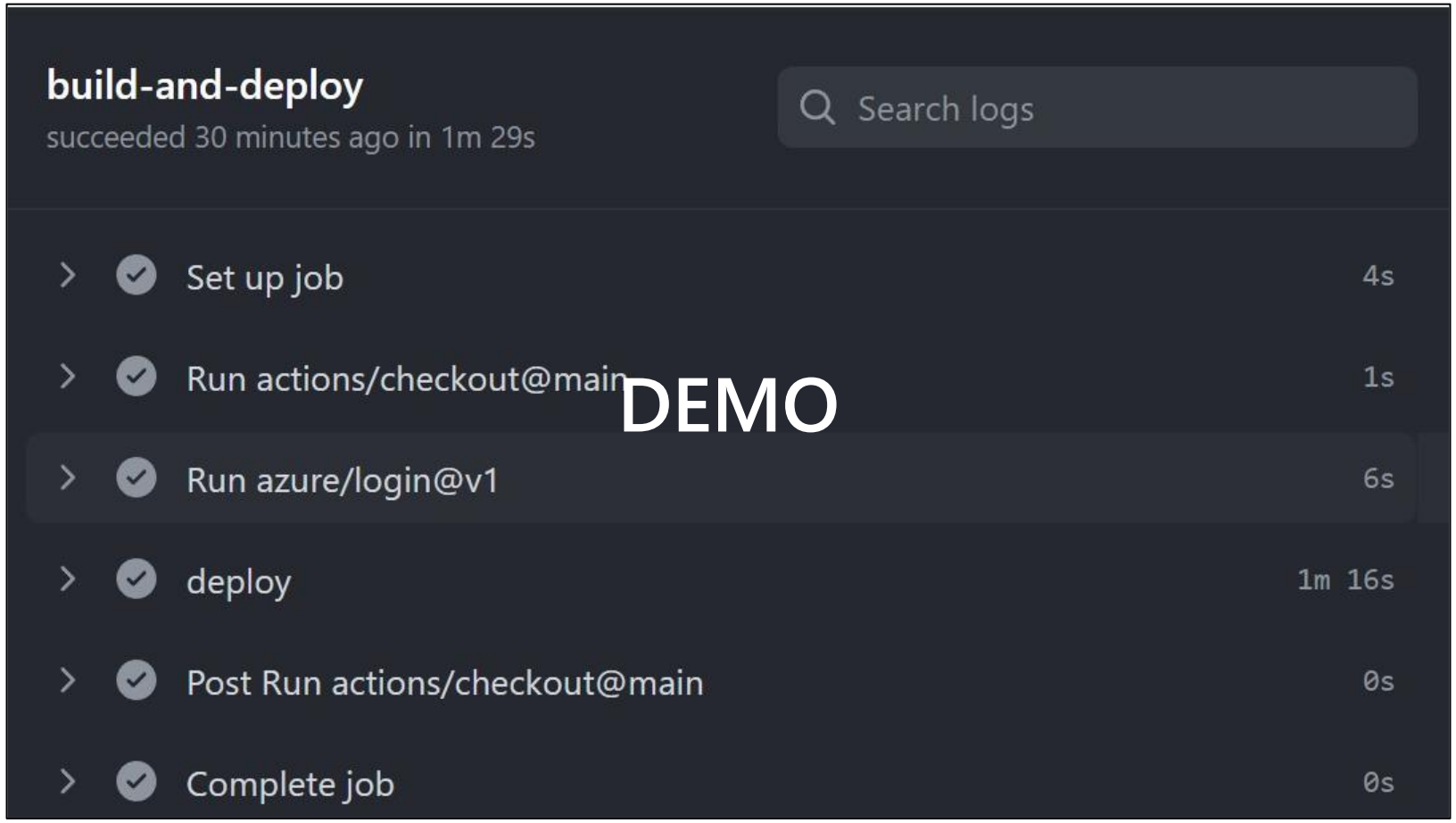
Other features: Loops, conditional deployment, multiline strings, referencing an existing cloud resource, and many more.


```
@minLength(3)
@maxLength(11)
param storagePrefix string
param storageSKU string = 'Standard_LRS'
param location string = resourceGroup().location

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2023-08-09' = {
    name: uniqueStorageName
    location: location
    sku: {
       name: storageSKU
    }
    kind: 'StorageV2'
    properties: {
        supportsHttpsTrafficOnly: true
    }

    resource service 'fileServices' = {
       name: 'default'

       resource share 'shares' = {
       name: 'exampleshare'
       }
    }
}

module webModule './webApp.bicep' = {
    name: 'webDeploy'
    params: {
        skuName: 'S1'
        location: location
    }
}

output storageEndpoint object = stg.properties.primaryEndpoints
```

Demonstration: Create Bicep templates



Demonstration: Deploy a Bicep file from GitHub workflows



Module 04: Create Azure resources by using Azure CLI

What is Azure CLI?

- **Command-line program to connect to Azure (Azure Cloud Shell, PowerShell, or Bash)**
- **Execute administrative commands on Azure resources through a terminal, command-line prompt, or script, instead of a web browser:**

  For example, to restart a VM use the command:

  ```
  az vm restart -g MyResourceGroup -n MyVm
  ```

  Can be installed on Linux, macOS, or Windows computers, and added as a module to PowerShell
- **Can be used interactively or scripted:**

  Interactive: Issue commands directly at the shell prompt

  Scripted: Assemble the CLI commands into a shell script and then execute the script

Work with Azure CLI

Commands in the CLI are structured in groups and subgroups:

Use az find to find commands you need

```
az find blob
```

Use the help argument to get more detail about the command

```
az storage blob --help
```

Creating a new Azure resource typically involves the following process:

Demonstration: Run templates using Azure CLI

Process to deploy and verify a sample Azure deployment template with custom script extension using Azure CLI

1. Create a resource group to deploy your resources to
2. Run **curl** to download the GitHub template
3. Validate the template
4. Deploy the resource
5. Obtain the IP address
6. Run **curl** to access your web server and verify the successful deployment and running of the custom script extension

✓ **Always remember to delete any resources you deployed when no longer needed to avoid incurring additional costs on them**

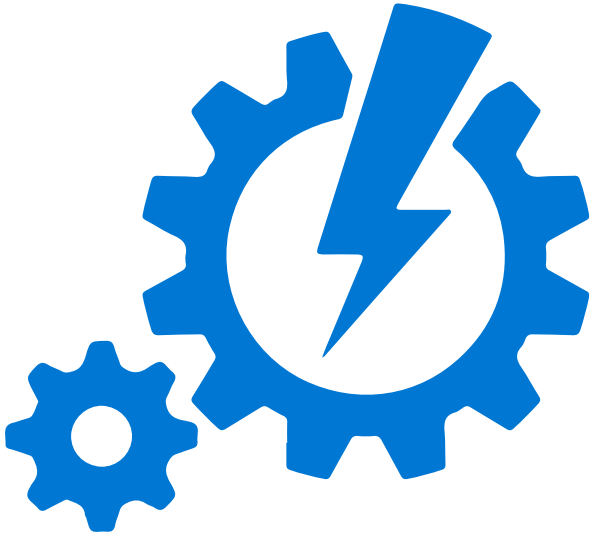
Module 05: Explore Azure Automation with DevOps

What is Azure Automation?

An Automation service integrated with Microsoft Azure for automating and simplifying the creation, deployment, monitoring and maintenance of Azure resources and resources external to Azure

Azure Automation Capabilities include:

- Manage Shared resources
- State configuration
- Integration with GitHub, Azure DevOps Git/TFVC
- Update management
- Can automate Windows or Linux environments
- Can apply to any system that exposes an API over internet protocols

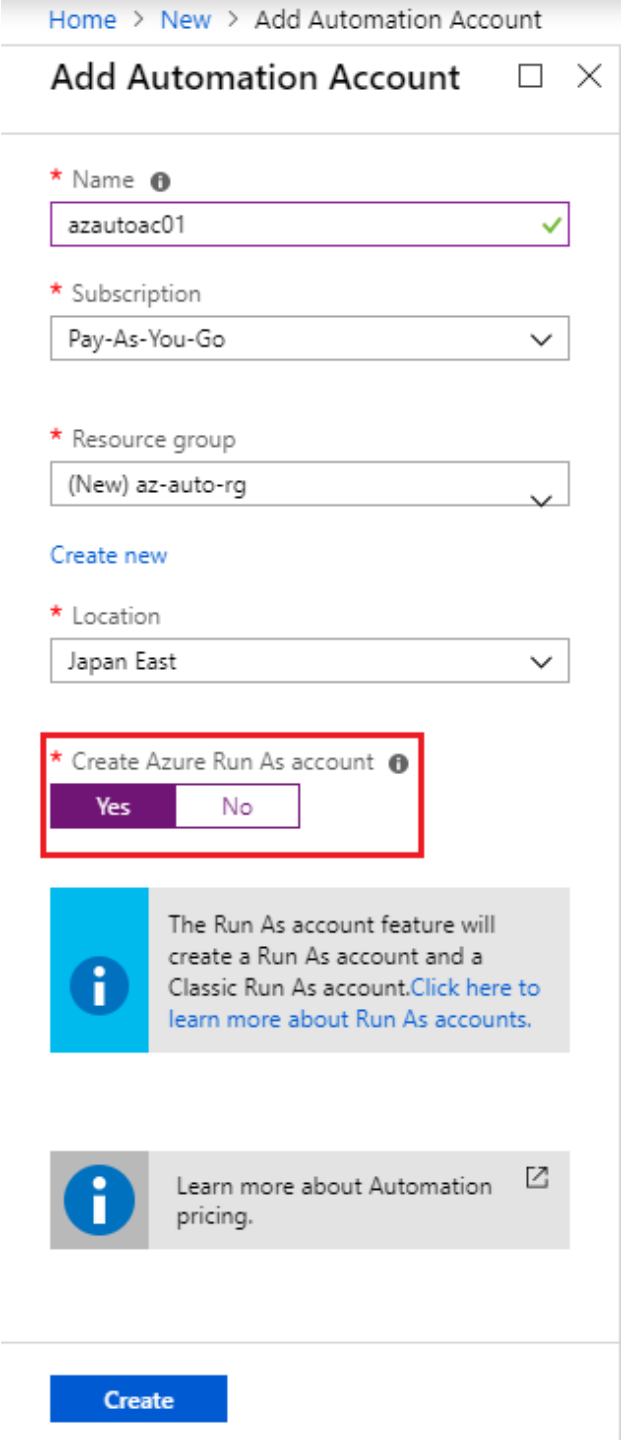


Create automation accounts

- To use Azure Automation, you must create an Automation account
- Automation account acts as a container in which you store, manage and use automation artifacts
- Provides a way to separate your environments or further organize your Automation workflows and resources
- Requires subscription-owner level access as provides access to all Azure resources via an API
- Need at least one automation account but should have multiple for access control

Run As account:

- Creates a Microsoft Entra service principal which allows access to Azure resources when running automation



What is a runbook?

- A *runbook* is a set of tasks that perform some automated process in Azure Automation
- Runbooks serve as repositories for your custom scripts and workflows
- Can create your own or import and modify from community via Runbook Gallery
- **Runbook Types available:**
  - Graphical runbook
  - PowerShell runbooks
  - PowerShell Workflow runbooks
  - Python runbooks

Understand automation shared resources

1. Azure Automation contains shared resources that are globally associated available to be used in, or with a runbook

2. Currently Eight Categories:
   - Schedules
   - Modules
   - Modules gallery
   - Python packages
   - Credentials
   - Connections
   - Certificates
   - Variables

Explore runbook gallery

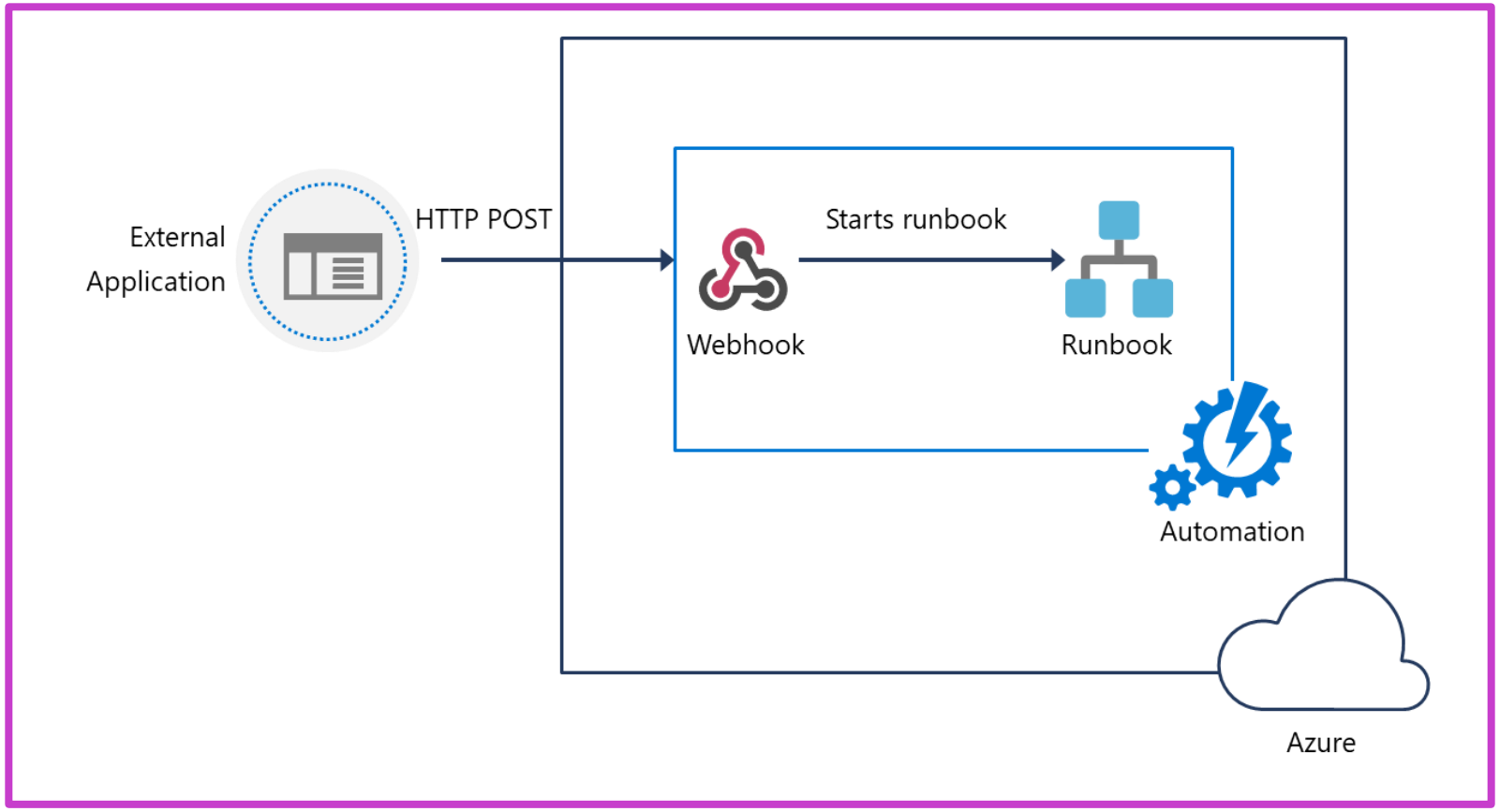
- Can import pre-existing runbooks from the runbook repository at the **Azure Automation GitHub**
- Runbooks provided to help eliminate the time it takes to build custom solutions
- Already been built by Microsoft and the Microsoft community
- Can be used with or without modification
- Can review the code or a visualization of the runbook code on the gallery as well as see source projects, rating, etc.
- **Considerations:**

  Python runbooks are also available from the Azure Automation GitHub. To find them, filter by language and select **Python**

  You cannot use PowerShell to import directly from the Runbook Gallery

Examine webhooks

- Automate the process of starting a runbook either by scheduling it, or by using a *webhook*
- Uses a HTTP request to start a runbook
- Reduces complexity and allows external services such as Azure DevOps, GitHub, or custom applications to use webhooks



Webhook Syntax: **http://< Webhook Server >/token?=< Token Value >**

Explore source control integration

- Azure Automation supports source control integration
- Easier collaboration
- Increased auditing and traceability
- Roll back to earlier versions of your runbooks
- Can push code from Azure Automation to source control or pull your runbooks from source control to Azure Automation

Azure Automation supports the following source Control options:

GitHub

Azure DevOps (Git)

Azure DevOps (TFVC)

Explore PowerShell workflows

- Allows automation and orchestration of multi-environment tasks
- Built on PowerShell and based on Windows Workflow Foundation
- Characteristics:
  - Contain Activities – Which are a core component of a workflow, specific tasks in a workflow
  - Tasks can be run in parallel
  - Can be long-running and repeated over and over (idempotent)
  - Be interrupted—can be stopped and restarted, suspended and resumed
  - Continue after an unexpected interruption, such as a network outage or computer/server restart

Create a workflow

```
Workflow MyFirstRunbook-Workflow
{
Param(
        [string]$VMname,
        [string]$ResourceGroupName,
     )
     . . . .
     Start-AzureRmVM –Name $VMName –ResourceGroupName $ResourceGroupName
}
```

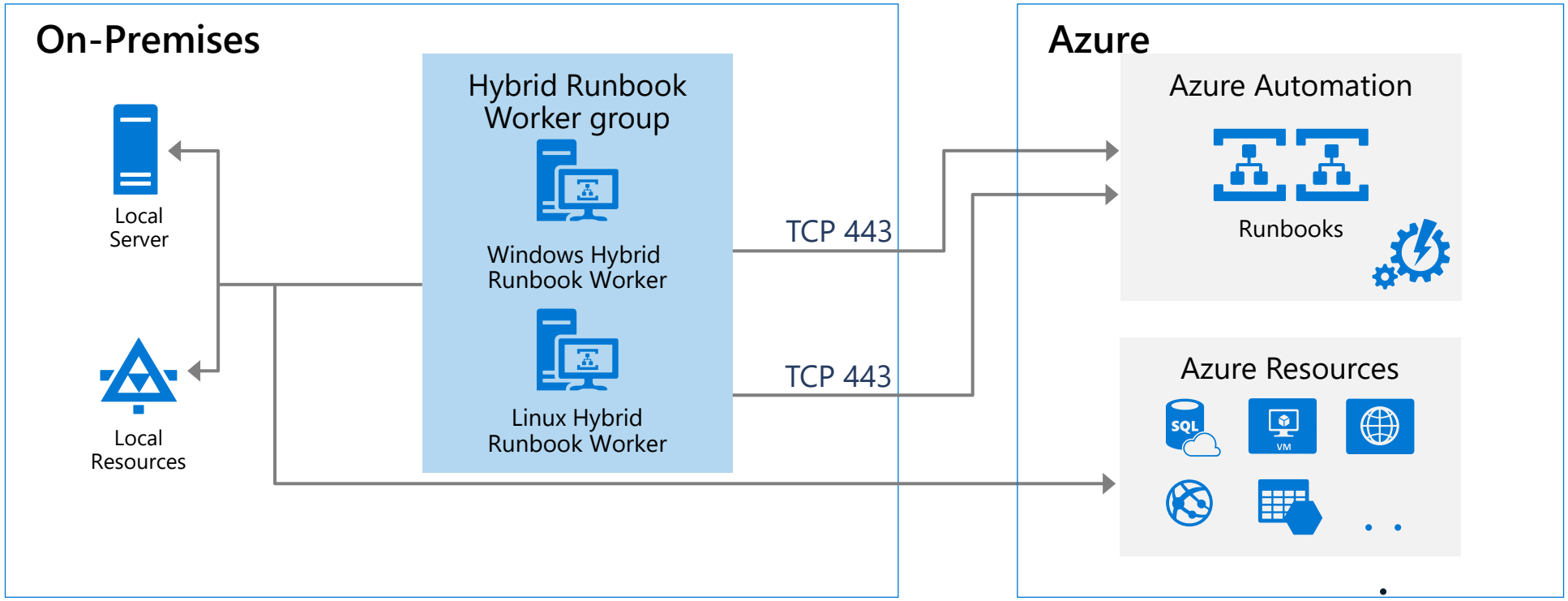
There are syntax differences between PowerShell scripts and Workflows

Requires keyword ***workflow*** to identify a workflow command

Add parameter values using keyword ***Param***

Explore hybrid management

Hybrid Runbook Worker **feature of Azure Automation allows you to run runbooks that manage local resources in your private datacenter, on machines located in your datacenter.**



Demonstration: Create and run a workflow runbook

- This demonstration will create a new PowerShell workflow runbook, test, publish and then run the runbook
- You can complete this walkthrough task by completing the steps outlined below, or you can simply read through them, depending on your available time

Examine checkpoint and parallel processing

Checkpoints:

If a workflow ends in an error or is suspended, it will start from its last checkpoint the next time it runs

You can set a checkpoint in a workflow with the ***Checkpoint-Workflow*** activity

A checkpoint is a snapshot of the current state of the workflow

Parallel Processing:

Use the ***Parallel*** keyword to create a script block with multiple commands that run concurrently

Tasks within this script block will be run ***concurrently*** or in ***parallel***

Can use *For Each* with *Parallel* keyword for more granular control on parallelism

Module 06: Implement Desired State Configuration (DSC)

Understand configuration drift

Configuration drift:

- Process whereby a set of resources change their state over time
- Can occur from changes made by people, processes, or programs

Potential security risks introduced by configuration drift:

- Open ports that should have been closed
- Inconsistent patching across environments
- Software that doesn’t meet compliance requirements

Solutions that can help:

- Windows PowerShell Desired State Configuration
- Azure Policy
- Many non-Microsoft solutions integrated with Azure

Explore Desired State Configuration (DSC)

- Ensures that an environment is maintained in a state that you specify (*defined state*), to eliminate configuration drift, and no deviation from that defined state
- **Components:**
  - **Configurations**: Idempotent declarative PowerShell scripts
  - **Resources**: PowerShell scripts compiled into .mof format files to ensure state
  - **Local Configuration Manager (LCM)**: Client engine to ensure configuration
- **Methods of Implementing DSC:**
  - **Push mode**: A user actively pushed put a configuration to environments
  - **Pull mode**: Clients get state from remote *pull service* automatically. Remote service is provided by a pull server, which acts as a central control and manager for the configurations

Explore Azure Automation State configuration (DSC)

- A cloud-based implementation of PowerShell DSC, available as part of Azure Automation
- **Characteristics:**
  - **Built-in pull server**: Built-in pull server in Azure Automation eliminates the need to set up and maintain your own pull server
  - **Management of all DSC artifacts**: Manage all DSC configurations, resources, and target nodes in single instance in the Azure portal or PowerShell
  - **Ability to Import Reporting Data directly into Azure Log Analytics**: Can send this data to your Log Analytics workspace

Examine DSC configuration file

DSC configurations **are PowerShell scripts that define a special type of function.**

Config File Elements:

- Configuration block: Name of the configuration
- Node block: Define nodes being configured e.g., VMs and servers
- Resource blocks: Defines the actual configuration state for the nodes

```
configuration LabConfig
 {
     Node WebServer
     {
         WindowsFeature IIS
         {
             Ensure = 'Present'
             Name = 'Web-Server'
             IncludeAllSubFeature = $true
         }
     }
}
```

Demonstration: Import and compile

- This walkthrough will create a configuration file, will then import that configuration to Azure Automation State configuration (DSC) and then compile the configuration file to create the MOF file
- You can complete this walkthrough task by completing the steps outlined below, or you can simply read through them, depending on your available time

Demonstration: Onboarding machines for management

- This walkthrough will follow on from the last walkthrough and will on-board virtual machines for management
- You can complete this walkthrough task by completing the steps outlined below, or you can simply read through them, depending on your available time

Implement DSC and Linux Automation on Azure

- Support for PowerShell DSC on Linux OS deprecated September 2023
- Replacement = **Azure Automanage Machine Configuration**
- Configuration Resources:

  ✓Operating System Settings
  ✓Application Configuration or presence
  ✓Environment Settings

Labs

Lab: Deployments using Azure Resource Manager templates

Lab overview:

In this lab, you will create an Azure Resource Manager template and modularize it by using a linked template. You will then modify the main deployment template to call the linked template and updated dependencies, and finally deploy the templates to Azure.

Objectives:

- Create Resource Manager template
- Create a Linked template for storage resources
- Upload Linked Template to Azure Blob Storage and generate SAS token
- Modify the main template to call Linked template
- Modify main template to update dependencies
- Deploy resources to Azure using linked templates

Duration:

Learning Path review and takeaways

What did you learn?

1. Apply infrastructure and configuration as code principles

2. Deploy and manage infrastructure using Microsoft automation technologies such as ARM templates, Azure CLI, DSC and Azure Automation

Learning Path review questions

1. What benefits can you achieve by modularizing your infrastructure and configuration resources?
2. Which method of approach for implementing Infrastructure as Code states what the final state of an environment should be without defining how it should be achieved?
3. Which term defines the ability to apply one or more operations against a resource, resulting in the same outcome every time?
4. Which term is the process whereby a set of resources change their state over time from their original state in which they were deployed?
5. Which Resource Manager deployment mode only deploys whatever is defined in the template, and does not remove or modify any other resources not defined in the template?

Learning Path review questions

6. How are the dependencies defined in a .bicep file?

7. What is the behavior of the webAppName parameter for a team that created a template that contains this line:
   param webAppName string = ‘mySite${uniqueString(resourceGroup().id)}’?

8. How can you reuse a Bicep template in other Bicep templates?

Microsoft