Introduction

This module introduces you to features and tools for managing and deploying Azure resources. You learn about the Azure portal (a graphic interface for managing Azure resources), the command line, and scripting tools that help deploy or configure resources. You also learn about Azure services that help you manage your on-premises and multicloud environments from within Azure.

Learning objectives

After completing this module, you'll be able to:

- Describe the Azure portal.
- Describe Azure Cloud Shell, including Azure CLI and Azure PowerShell.
- Describe the purpose of Azure Arc.
- Describe Azure Resource Manager (ARM), ARM templates, and Bicep.

Describe tools for interacting with Azure

To get the most out of Azure, you need a way to interact with the Azure environment, the management groups, subscriptions, resource groups, resources, and so on. Azure provides multiple tools for managing your environment, including the:

- Azure portal
- Azure PowerShell
- Azure Command Line Interface (CLI)

What is the Azure portal?

The Azure portal is a web-based, unified console that provides an alternative to command-line tools. With the Azure portal, you can manage your Azure subscription by using a graphical user interface. You can:

- Build, manage, and monitor everything from simple web apps to complex cloud deployments
- Create custom dashboards for an organized view of resources
- Configure accessibility options for an optimal experience

The Azure portal is designed for resiliency and continuous availability. It maintains a presence in every Azure datacenter. This configuration makes the Azure portal resilient to individual datacenter failures and avoids network slowdowns by being close to users. The Azure portal updates continuously and requires no downtime for maintenance activities.

Azure Cloud Shell

Azure Cloud Shell is a browser-based shell tool that allows you to create, configure, and manage Azure resources using a shell. Azure Cloud Shell support both Azure PowerShell and the Azure Command Line Interface (CLI), which is a Bash shell.

You can access Azure Cloud Shell via the Azure portal by selecting the Cloud Shell icon:

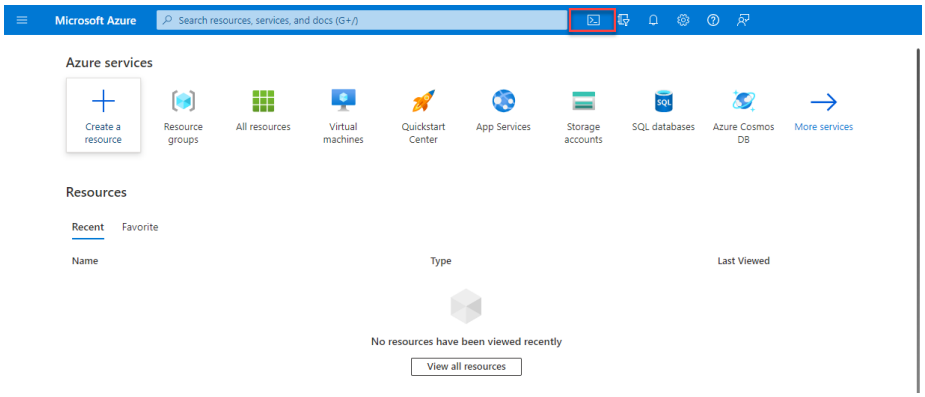

Figure 1: Screenshot of the Azure portal with the Cloud Shell icon emphasized.

Azure Cloud Shell has several features that make it a unique offering to support you in managing Azure. Some of those features are:

- It is a browser-based shell experience, with no local installation or configuration required.
- It is authenticated to your Azure credentials, so when you log in it inherently knows who you are and what permissions you have.
- You choose the shell you're most familiar with; Azure Cloud Shell supports both Azure PowerShell and the Azure CLI (which uses Bash).

What is Azure PowerShell?

Azure PowerShell is a shell with which developers, DevOps, and IT professionals can run commands called command-lets (cmdlets). These commands call the Azure REST API to perform management tasks in Azure. Cmdlets can be run independently to handle one-off changes, or they may be combined to help orchestrate complex actions such as:

- The routine setup, teardown, and maintenance of a single resource or multiple connected resources.
- The deployment of an entire infrastructure, which might contain dozens or hundreds of resources, from imperative code.

Capturing the commands in a script makes the process repeatable and automatable.

In addition to be available via Azure Cloud Shell, you can install and configure Azure PowerShell on Windows, Linux, and Mac platforms.

What is the Azure CLI?

The Azure CLI is functionally equivalent to Azure PowerShell, with the primary difference being the syntax of commands. While Azure PowerShell uses PowerShell commands, the Azure CLI uses Bash commands.

The Azure CLI provides the same benefits of handling discrete tasks or orchestrating complex operations through code. It's also installable on Windows, Linux, and Mac platforms, as well as through Azure Cloud Shell.

Due to the similarities in capabilities and access between Azure PowerShell and the Bash based Azure CLI, it mainly comes down to which language you're most familiar with.

Describe the purpose of Azure Arc

Managing hybrid and multi-cloud environments can rapidly get complicated. Azure provides a host of tools to provision, configure, and monitor Azure resources. What about the on-premises resources in a hybrid configuration or the cloud resources in a multi-cloud configuration?

In utilizing Azure Resource Manager (ARM), Arc lets you extend your Azure compliance and monitoring to your hybrid and multi-cloud configurations. Azure Arc simplifies governance and management by delivering a consistent multi-cloud and on-premises management platform.

Azure Arc provides a centralized, unified way to:

- Manage your entire environment together by projecting your existing non-Azure resources into ARM.
- Manage multi-cloud and hybrid virtual machines, Kubernetes clusters, and databases as if they are running in Azure.
- Use familiar Azure services and management capabilities, regardless of where they live.
- Continue using traditional ITOps while introducing DevOps practices to support new cloud and native patterns in your environment.
- Configure custom locations as an abstraction layer on top of Azure Arc-enabled Kubernetes clusters and cluster extensions.

What can Azure Arc do outside of Azure?

Currently, Azure Arc allows you to manage the following resource types hosted outside of Azure:

- Servers
- Kubernetes clusters
- Azure data services
- SQL Server
- Virtual machines (preview)

Describe Azure Resource Manager and Azure ARM templates

Azure Resource Manager (ARM) is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your Azure account. Anytime you do anything with your Azure resources, ARM is involved.

When a user sends a request from any of the Azure tools, APIs, or SDKs, ARM receives the request. ARM authenticates and authorizes the request. Then, ARM sends the request to the Azure service, which takes the requested action. You see consistent results and capabilities in all the different tools because all requests are handled through the same API.

Azure Resource Manager benefits

With Azure Resource Manager, you can:

- Manage your infrastructure through declarative templates rather than scripts. A Resource Manager template is a JSON file that defines what you want to deploy to Azure.
- Deploy, manage, and monitor all the resources for your solution as a group, rather than handling these resources individually.
- Re-deploy your solution throughout the development life-cycle and have confidence your resources are deployed in a consistent state.
- Define the dependencies between resources, so they're deployed in the correct order.
- Apply access control to all services because RBAC is natively integrated into the management platform.
- Apply tags to resources to logically organize all the resources in your subscription.
- Clarify your organization's billing by viewing costs for a group of resources that share the same tag.

Infrastructure as code

Infrastructure as code is a concept where you manage your infrastructure as lines of code. At an introductory level, it's things like using Azure Cloud Shell, Azure PowerShell, or the Azure CLI to manage and configure your resources. As you get more comfortable in the cloud, you can use the infrastructure as code concept to manage entire deployments using repeatable templates and configurations. ARM templates and Bicep are two examples of using infrastructure as code with the Azure Resource Manager to maintain your environment.

ARM templates

By using ARM templates, you can describe the resources you want to use in a declarative JSON format. With an ARM template, the deployment code is verified before any code is run. This ensures that the resources will be created and connected correctly. The template then orchestrates the creation of those resources in parallel. That is, if you need 50 instances of the same resource, all 50 instances are created at the same time.

Ultimately, the developer, DevOps professional, or IT professional needs only to define the desired state and configuration of each resource in the ARM template, and the template does the rest. Templates can even execute PowerShell and Bash scripts before or after the resource has been set up.

Benefits of using ARM templates

ARM templates provide many benefits when planning for deploying Azure resources. Some of those benefits include:

- **Declarative syntax**: ARM templates allow you to create and deploy an entire Azure infrastructure declaratively. Declarative syntax means you declare what you want to deploy but don't need to write the actual programming commands and sequence to deploy the resources.
- **Repeatable results**: Repeatedly deploy your infrastructure throughout the development lifecycle and have confidence your resources are deployed in a consistent manner. You can use the same ARM template to deploy multiple dev/test environments, knowing that all the environments are the same.
- **Orchestration**: You don't have to worry about the complexities of ordering operations. Azure Resource Manager orchestrates the deployment of interdependent resources, so they're created in the correct order. When possible, Azure Resource Manager deploys resources in parallel, so your deployments finish faster than serial deployments. You deploy the template through one command, rather than through multiple imperative commands.
- **Modular files**: You can break your templates into smaller, reusable components and link them together at deployment time. You can also nest one template inside another template. For example, you could create a

template for a VM stack, and then nest that template inside of templates that deploy entire environments, and that VM stack will consistently be deployed in each of the environment templates.
- **Extensibility**: With deployment scripts, you can add PowerShell or Bash scripts to your templates. The deployment scripts extend your ability to set up resources during deployment. A script can be included in the template or stored in an external source and referenced in the template. Deployment scripts give you the ability to complete your end-to-end environment setup in a single ARM template.

Bicep

Bicep is a language that uses declarative syntax to deploy Azure resources. A Bicep file defines the infrastructure and configuration. Then, ARM deploys that environment based on your Bicep file. While similar to an ARM template, which is written in JSON, Bicep files tend to use a simpler, more concise style.

Some benefits of Bicep are:

- **Support for all resource types and API versions**: Bicep immediately supports all preview and GA versions for Azure services. As soon as a resource provider introduces new resource types and API versions, you can use them in your Bicep file. You don't have to wait for tools to be updated before using the new services.
- **Simple syntax**: When compared to the equivalent JSON template, Bicep files are more concise and easier to read. Bicep requires no previous knowledge of programming languages. Bicep syntax is declarative and specifies which resources and resource properties you want to deploy.
- **Repeatable results**: Repeatedly deploy your infrastructure throughout the development lifecycle and have confidence your resources are deployed in a consistent manner. Bicep files are idempotent, which means you can deploy the same file many times and get the same resource types in the same state. You can develop one file that represents the desired state, rather than developing lots of separate files to represent updates.
- **Orchestration**: You don't have to worry about the complexities of ordering operations. Resource Manager orchestrates the deployment of interdependent resources so they're created in the correct order. When possible, Resource Manager deploys resources in parallel so your deployments finish faster than serial deployments. You deploy the file through one command, rather than through multiple imperative commands.
- **Modularity**: You can break your Bicep code into manageable parts by using modules. The module deploys a set of related resources. Modules enable you to reuse code and simplify development. Add the module to a Bicep file anytime you need to deploy those resources.